FUJITSU

FUJITSU Software
Interstage Application Server V12.3.0

J2EE ユーザーズガイド(旧版互換)

Windows/Solaris(64)/Linux

B1WS-1323-05Z0(00)
2020年8月

まえがき

本書の目的

本書は、InterstageのJ2EEコンポーネントを利用してアプリケーションの開発や運用を行うために、J2EEの概要、環境構築やアプリケーションの運用について紹介しています。

本書は、以下の方を対象にしています。

- J2EEコンポーネントを利用してアプリケーションを開発する方
- J2EEコンポーネントを利用したアプリケーションを運用する方

プラットフォームにより提供される機能が異なります。以下にその一覧を示します。

機能	Windows Windows32/64	Solaris Solaris64	Linux Linux32/64
Java Transaction Service(JTS)	○	×	○
Java Message Service(JMS)	○	○	○
J2EE Connector Architecture (connector)	○	○	○
Enterprise JavaBeans(EJB)	○	○	○
HTTPトンネリング	○	○	○
クラスタサービス	○	○	○

前提知識

本書を読む場合、以下の知識が必要です。

- 使用するOSに関する基本的な知識
- Javaに関する基本的な知識
- J2EEに関する基本的な知識
- Webサービスに関する基本的な知識
- XMLに関する基本的な知識
- インターネットに関する基本的な知識
- リレーショナルデータベースに関する基本的な知識
- トランザクションモデル(クライアント・サーバモデル)に関する知識

本書の関連情報として、次の文書を参考にしてください。

- JavaTM Platform, Enterprise Edition アプリケーション設計ガイド - J2EETM Blueprints -
  (この文書は、オラクル社のホームページからダウンロードしてください。)

本書の構成

本書は以下の構成になっています。

第1部 J2EE共通編

第1章 概要

J2EEおよびInterstageのJ2EEコンポーネントについて概要を説明します。

第2章 J2EEアプリケーションの設計

J2EEアプリケーションの開発に関する流れについて説明します。

第3章 J2EEアプリケーションの運用

J2EEアプリケーションを利用する場合の導入から運用について説明します。

第4章 JNDI

JNDIの概要について説明します。

第5章 J2EEアプリケーションのセキュリティ

セキュリティ機能の概要および設定方法について説明します。

第2部 Servlet/JSP編

第6章 Servletサービスの機能

Servletサービスの機能について説明します。

第7章 Webアプリケーションの開発

Webアプリケーションの開発方法について説明しています。

第8章 Webアプリケーションの呼び出し方法

Webアプリケーションの呼出し方法について説明します。

第9章 セションリカバリ機能

Servletサービスのセションリカバリ機能について説明します。

第3部 EJB編

第10章 EJBサービスの機能

EJBサービスを使用する上で必要な基本機能を説明します。

第11章 EJBアプリケーションの開発

EJBアプリケーションとクライアントアプリケーションの開発およびテスト方法について説明します。

第12章 Session Beanの実装

Session Beanの作成方法を説明します。

第13章 Entity Beanの実装

Entity Beanの作成方法を説明します。

第14章 Message-driven Beanの実装

Message-driven Beanの作成方法を説明します。

第15章 EJBアプリケーションの呼出し方法

Session BeanとEntity Beanを使用する場合プログラミング方法について説明します。

第16章 運用コマンドを使用してカスタマイズする方法

EJBアプリケーションの実行環境を、カストマイズツールの運用コマンドを使用してカスタマイズする方法について説明します。

製品の表記について

本書での以下の表記については、それぞれの基本ソフトウェアに対応した製品を示しています。

表記	説明
RHEL6(x86)	Red Hat Enterprise Linux 6 (for x86)を前提基本ソフトウェアとした本製品
RHEL6(Intel64)	Red Hat Enterprise Linux 6 (for Intel64)を前提基本ソフトウェアとした本製品
RHEL7(Intel64)	Red Hat Enterprise Linux 7 (for Intel64)を前提基本ソフトウェアとした本製品
RHEL8(Intel64)	Red Hat Enterprise Linux 8 (for Intel64)を前提基本ソフトウェアとした本製品

用語について

本書では、Windows(R)の場合はコンピュータのプロパティの設定を、Solaris/Linuxの場合はInterstage起動時の環境変数を、“システム環境変数”と記述している場合があります。

輸出許可

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

登録商標について

記載されている会社名、製品名などの固有名詞は、各社の商標または登録商標です。

本製品のマニュアルに記載されている他社製品の商標表示については、「マニュアル体系と読み方」の「マニュアルの読み方」－「登録商標について」を参照してください。

著作権

2020年8月 第5版

2020年2月 第4版

2019年6月 第3版

2018年7月 第2版

2017年7月 初版

目　次

第1部 J2EE共通編

第1章 概要

サポートするJ2EEの仕様範囲

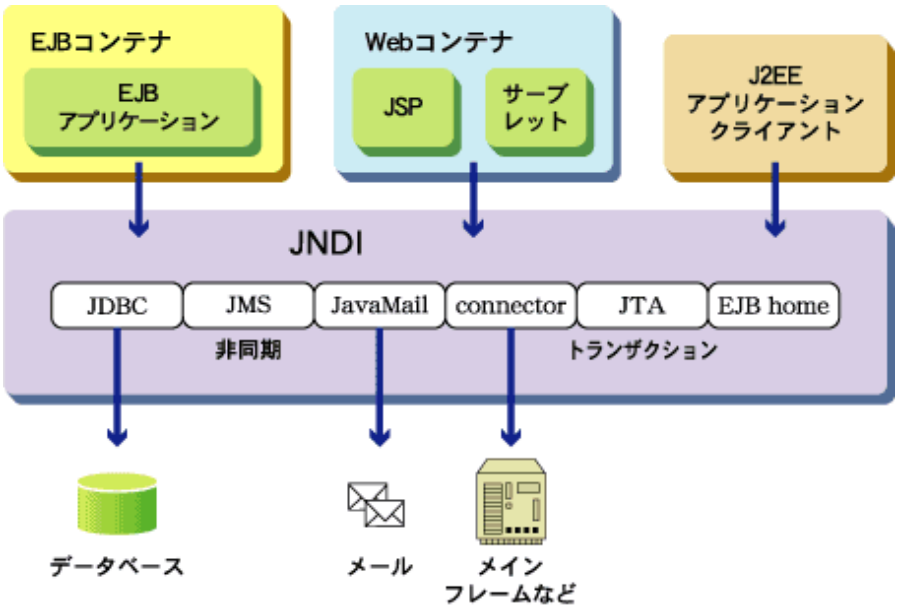
J2EEプラットフォームは、J2EEアプリケーションを実行するための標準環境です。

Interstageが提供するJ2EEプラットフォームは、企業規模の多階層サービスの実装に必要とされるさまざまな機能を提供します。

J2EEコンポーネントとして提供する機能

InterstageのJ2EEコンポーネントは、以下の規約に準じて作成されたJ2EEアプリケーションに対するサービスを提供します。

規約	バージョン
JSP	2.0
Servlet	2.4
EJB	2.1
JMS	1.1
JTA	1.0
JavaMail	1.4
JAF	1.1
JAXP	1.2
Connector	1.5
Web Services	1.1
JAX-RPC	1.1
SAAJ	1.2



Interstageでは以下のサービスを提供しています。

Servletサービス

Servletサービスは、サーバ上でWebアプリケーションの実行を制御するコンポーネントです。

EJBサービス

EJBサービスは、EJB規約に準拠した、サーバアプリケーションを実行するためのコンポーネントです。

Interstage Webサービス

Interstage Webサービスは、Webサービスの実行を制御するコンポーネントです。

JNDI

JNDIは、各アプリケーションで使用するリソース情報をアプリケーション動作環境ごとに定義するのではなく、Interstage環境で動作するすべてのアプリケーションで利用可能なJNDIサービスプロバイダ機能を提供するコンポーネントです。

JDBC

JDBCは、Javaアプリケーションからデータベースにアクセスするためのデータベース非依存のAPIを提供します。Interstageでは、各データベースが提供するJDBCドライバと連携するための機能を提供しています。詳細は“4.3 JDBC(データベース)を参照する場合の環境設定”を参照してください。

Java Transaction Service (JTS) Windows32/64 Linux32/64

JTSは、アプリケーションがトランザクションにアクセスする場合に、特定の実装を意識しないためのサービスを提供するコンポーネントです。

Java Message Service (JMS)

JMSは、分散環境において信頼できる非同期通信を提供するコンポーネントです。

J2EE Connector Architecture (connector)

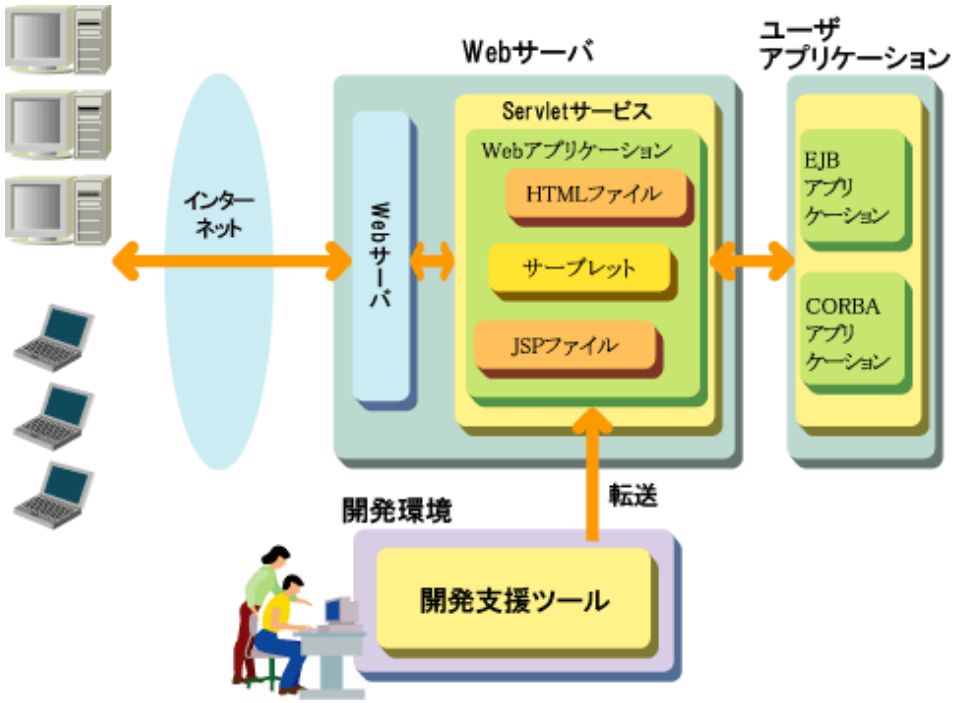
connectorは、EIS層に位置付けられたERPシステムやメインフレーム、データベースといった企業情報システムなどに共通に接続するためのコンポーネントです。

JavaMail

JavaMailは、Javaを用いて、環境やプロトコルに依存しない、メール送受信機能の実現とアプリケーションの作成を可能にするAPIを提供するコンポーネントです。またプロバイダとしては、SMTP、IMAPおよびPOP3を同梱しています。

1.1 Servletサービス

Servletサービスは、サーバ上でWebアプリケーションの実行を制御するコンポーネントです。
Webアプリケーションは、HTMLファイル、イメージファイル、サーブレット、JSPファイルなどのWebリソースから構成されているアプリケーションです。
WebサーバではHTMLファイルやイメージファイルの制御が可能ですが、Servletサービスは、それに加え、サーブレットやJSPファイルの制御を行っています。

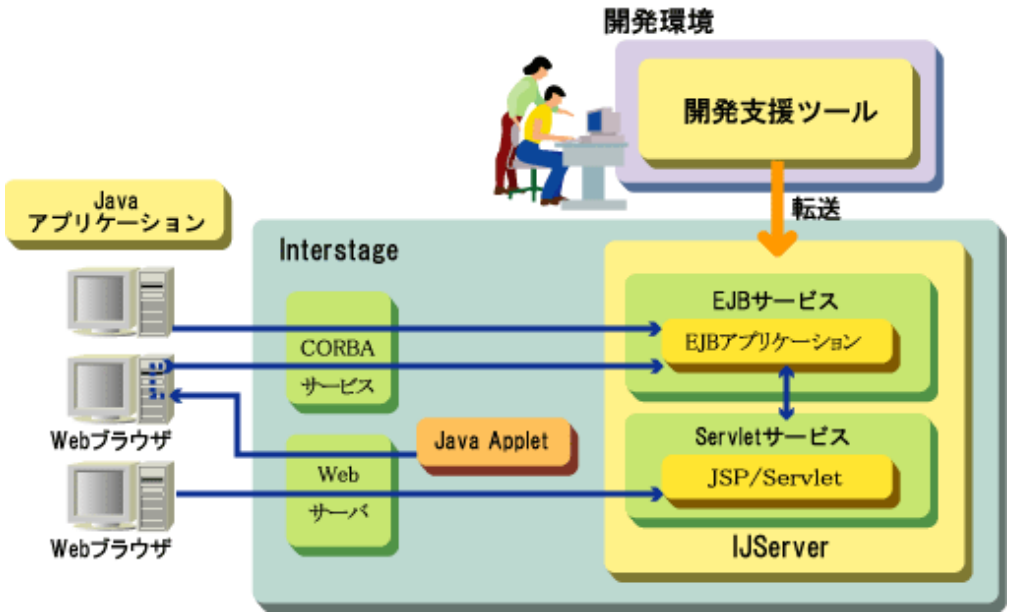


1.2 EJBサービス

EJBサービスは、EJB規約に準拠した、サーバアプリケーションを実行するためのコンポーネントです。クライアント/サーバ間の通信形態として、RMI over IIOPを採用しているため、RMIの規約でアプリケーションを開発することができます。

クライアントアプリケーションの形態としては、Javaアプレット、Javaアプリケーション、サーブレット、JavaServer Pages(JSP)ベースのアプリケーションなどが利用できます。
EJBアプリケーションは、EJB規約をサポートする開発ツールを使用して開発します。

以下にInterstageとEJBサービスの関係図を示します。



1.3 Interstage Webサービス

Interstage Webサービスは、Webサービスの実行を制御するコンポーネントです。

Webサービスとは、任意の方法で実現されたサーバプログラム(Webサービスアプリケーション)を、インターネット標準の技術を利用してネットワーク経由でアクセスできるように構築したものです。通信プロトコルにはSOAPを用い、利用(アクセス)するためのインタフェース情報の記述にはWSDLを用います。

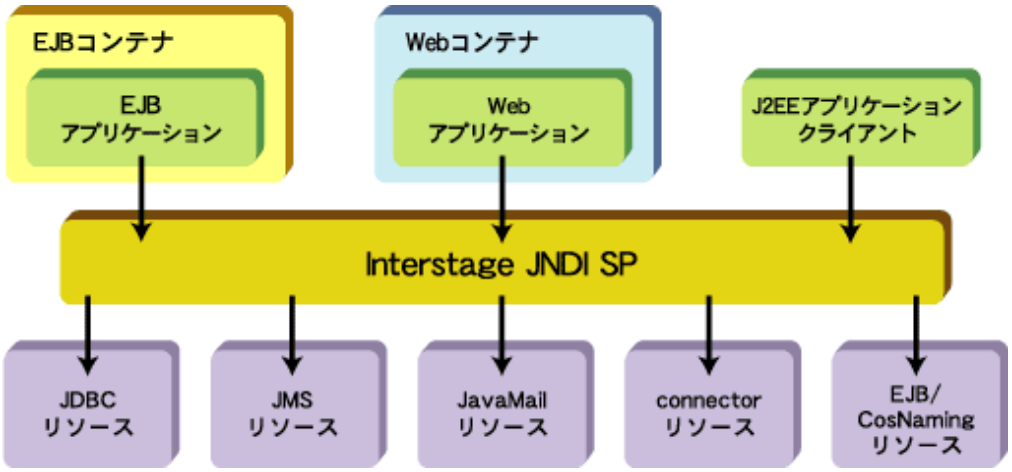
Webサービスを利用する(呼び出す)場合は、利用先(呼出し先)が提供するWSDLに記述されたインタフェース情報に基づき、SOAPでリクエストを行うクライアントプログラム(Webサービスクライアント)を任意の方法で作成します。

WebサービスはWebコンテナおよびEJBコンテナ上で動作します。

1.4 JNDI

Interstageでは、JNDI(Java Naming and Directory Interface)の考え方をベースとした、サーバアプリケーションを実行するための実行環境として、JNDIサービスプロバイダ(以降JNDI SPと記述する)機能を提供しています。
JNDI SPでは、EJBアプリケーション、Webアプリケーション、およびJ2EEアプリケーションクライアントで使用するJNDI名と、Interstage(運用)環境で使用するリソース名の対応付けが管理されます。
このJNDI SPを利用することで、各アプリケーションで使用するリソース情報を、アプリケーションの動作環境ごとに定義する必要がなくなり、Interstage環境で統合された管理を行うことが可能となります。

以下にJNDI SPの位置付けを示します。



1.5 Java Transaction Service(JTS) Windows32/64 Linux32/64

Java Transaction Service(JTS)は、Java Transaction API(JTA)を提供するトランザクションサービスです。
JTAを利用することにより、J2EEアプリケーション内で容易に分散トランザクションを制御できます。

ポイント

JTS/JTAを利用したアプリケーションを運用するには、データベース連携サービスが提供しているデータベース連携サービスが必要となります。
データベース連携サービスは、CORBAのOTS(Object Transaction Service)およびJ2EEのJTS(Java Transaction Service)を含めたトランザクションサービスであり、本製品では、J2EEアプリケーションから利用されるデータベース連携サービスを「JTS」と分類しています。

データベース連携サービスのJTSは、分散トランザクション機能を提供します。
分散トランザクションには、以下の2つのトランザクションが定義されています。

グローバルトランザクション

複数のJ2EEアプリケーション(EJBアプリケーションなど)を連携して、一つのトランザクションとして扱うことが可能となります。
複数のリソース(データベース、JMS、およびConnectorで接続された企業情報システムなど)を連携する処理を一つのトランザクションとして扱うことが可能となります。リソース間の連携は、2フェーズコミットプロトコルによって行われ、整合性が保障されます。JTSでは、トランザクション操作APIとしてJTA(Java Transaction API)を提供します。

ローカルトランザクション

ローカルトランザクションは、一般的にリソースマネージャによって管理されます。
ローカルトランザクションの操作は、リソースマネージャから提供されるAPIを使用することによりアプリケーションから直接操作が可能です。

JTSは、以下のシステム構成でトランザクションサービスとして機能します。

J2EEアプリケーション

J2EEアプリケーションは、アプリケーション開発者が実装する必要があります。

OTSシステム

データベース連携サービスのトランザクション管理基盤です。
JTSは、OTSシステムを利用してトランザクションの管理を行います。
そのため、JTSを利用するには、データベース連携サービスが必要となります。

リソース管理プログラム

リソースマネージャとトランザクション連携を行うためのシステムプログラムです。
J2EEで規定されるJDBCやConnectorなどのリソースインタフェースをサポートしています。

CORBAサービス

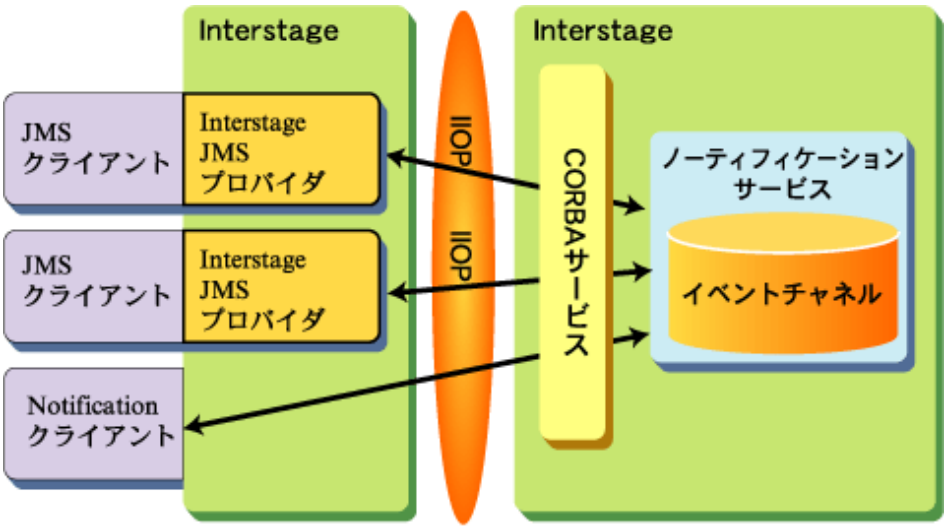
Interstageが提供する分散アプリケーション基盤です。
J2EEアプリケーションの運用やJTSを利用するためには、CORBAサービスが必要となります。

1.6 Java Message Service(JMS)

Interstage JMSは、イベントサービスのノーティフィケーションサービス上に構築され、通信層などメッセージング処理の基盤部分を利用します。

Interstage JMSプロバイダは、JMSクライアントからのメッセージ送受信要求を受けると、ノーティフィケーションサービスのイベントチャネルに対してメッセージの送受信を行います。

InterstageとInterstage JMSの関係を以下の図に示します。

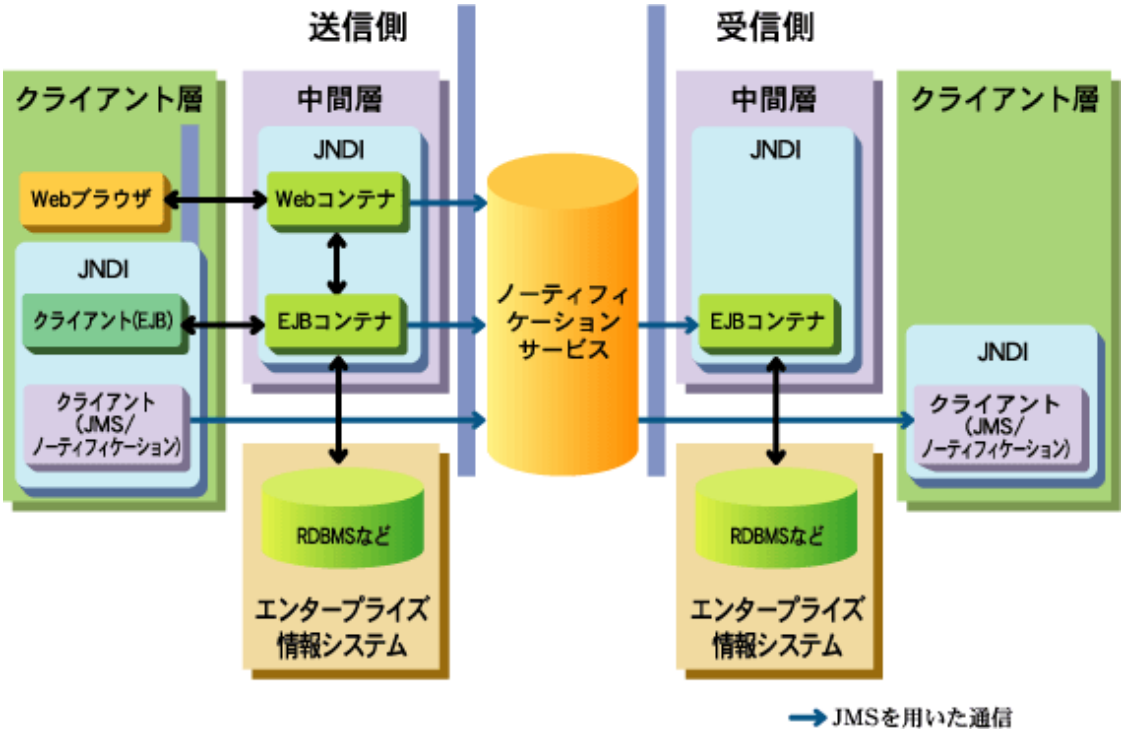


Interstage JMS構成

Interstage JMSの主な機能を以下に示します。

- Publish/Subscribeメッセージングモデル
- Point-To-Pointメッセージングモデル
- メッセージ保証
- メッセージセレクタ機能

Interstage JMSが連携可能なコンポーネントには、EJBサービス/Servletサービスなどがあります。これらの関係を以下の図に示します。



J2EEで規定されたJMS APIを使用して、非同期通信を行うアプリケーションの作成/運用が可能となります。

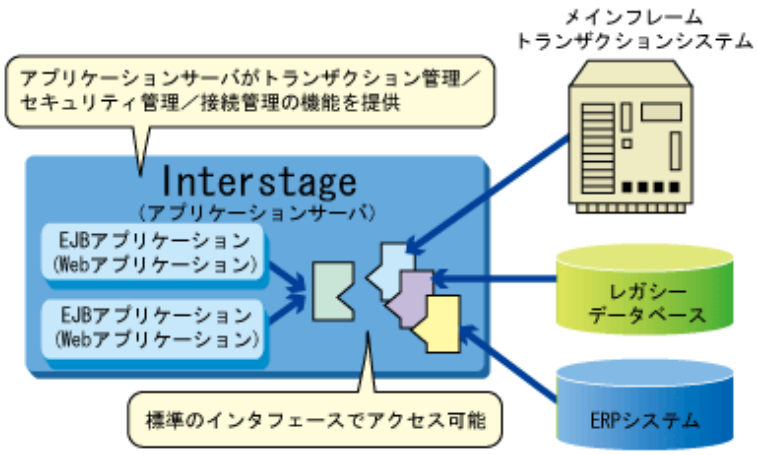
- 受信アプリケーションの所在、運用状態を意識せずにメッセージを送信
- 任意契機に任意条件でメッセージを受信

メッセージング基盤にノーティフィケーションサービスを使用しているため、ノーティフィケーションサービスを介したCORBAアプリケーションとの連携が可能となります。

1.7 J2EE Connector Architecture(connector)

既存のエンタープライズ情報システム(EIS)には、リソースを管理する仕組みとして多種多様なシステムが存在します。
しかし、それらのシステムは独自のインタフェースを持っているため、アプリケーションサーバと連携するにはアプリケーションサーバ固有の接続ドライバを提供する必要がありました。
また、アプリケーションサーバ側も、各EISへ接続するためにそれぞれ独自の実装を用意しなければなりませんでした。
そこで、EIS共通のインタフェースを定めたconnectorをサポートすることでこの問題を解決します。

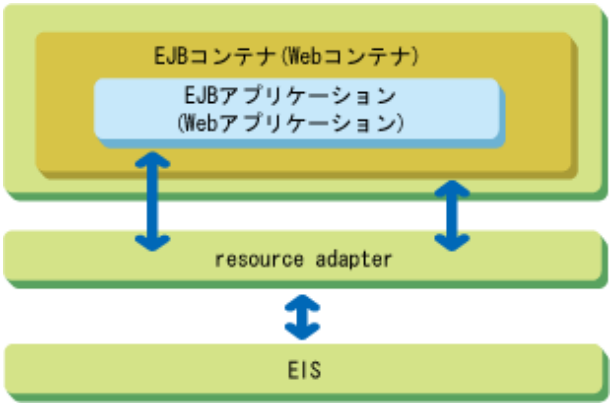
また、ユーザアプリケーションは、アプリケーションサーバが提供するトランザクション管理、接続管理、セキュリティ機能を利用できるようになります。
さらに、他社製品のアプリケーションサーバで動作していたアプリケーションを、そのままInterstage上に持ってきても動作させることが可能となります。



connectorを使用することにより、以下の理由からEISと統合したシステムをスピーディに構築できます。

- 共通のインタフェースでEISに接続できるため、EISごとに固有のインタフェースを理解する必要がない。
- トランザクション管理、接続管理、セキュリティ管理はアプリケーションサーバが提供するものを使用できる。

connectorは以下の要素から構成されています。



EJBアプリケーション(Webアプリケーション)

ビジネスロジックを実装したユーザアプリケーションです。resource adapterを介してリソースにアクセスします。Webアプリケーションからアクセスすることも可能です。

EJBコンテナ(Webコンテナ)

resource adapterと連携して、トランザクション管理、接続管理、セキュリティ管理の機能を提供します。

resource adapter

EISへの接続ドライバです。コンテナとJ2EEアプリケーションと連携するためのインタフェースを実装しています。通常、接続するEISによって異なる実装を持ちます。javax.resource.cciインタフェースが実装されていることが推奨されていますが、resource adapter固有のインタフェースが実装されている場合があります。resource adapterのインタフェースについてはresource adapterの仕様書を参照してください。
resource adapterは.rarという拡張子を持つファイルにアーカイブされて、Javaのライブラリおよび必要に応じてEISと連携するのに必要なネイティブライブラリが含まれます。
EJBアプリケーション(Webアプリケーション)は各リソース提供者もしくはサードベンダが提供しているresource adapterを介して、各リソースにアクセスします。

EIS

既存のエンタープライズ情報システム(Enterprise Information System)です。EISの例としてERPシステム、メインフレームトランザクション処理システム、レガシーデータベースシステムなどがあります。

第2章 J2EEアプリケーションの設計

本章では、J2EEアプリケーションの設計と開発について説明します。
ここでは、以下について説明します。

- J2EEアプリケーションのモデル
- J2EEアプリケーションが運用される環境(IJServer)
- クラスローダ
- トランザクション制御

2.1 J2EEアプリケーションのモデル

ここではJ2EEアプリケーションの標準的なモデルについて説明します。J2EEプラットフォームでシステムを構築する場合は、次のようなモデルを意識して設計します。

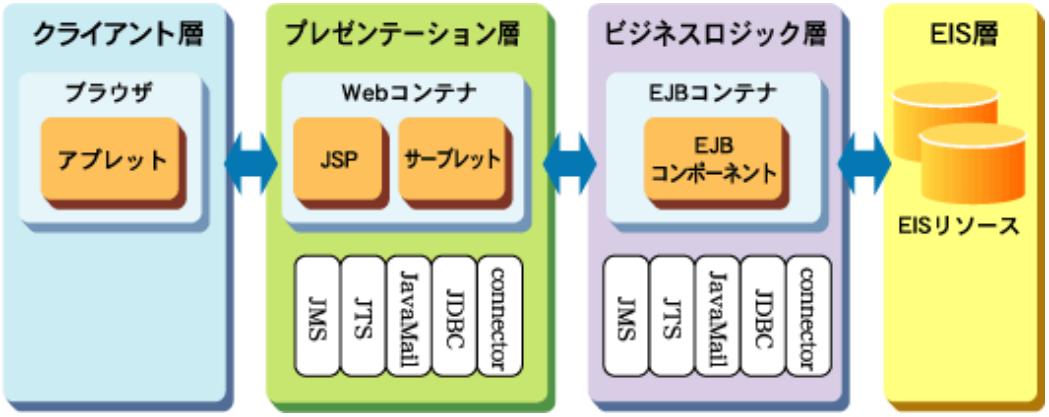
J2EEのモデル

J2EEでは、以下のようなリファレンスモデルを推奨しています。

- クライアント層
  システムに接続するユーザインタフェースを提供します。Webブラウザを基本としていますが、GUIのJavaアプリケーションもクライアントとして想定されています。
- プレゼンテーション層
  プレゼンテーションのロジックをカプセル化したもので、システムを利用するクライアントからのリクエストを受け付け、ビジネスロジック層のサービスへの橋渡しなどを行ってレスポンスをクライアントに返送するサービスを提供します。この層では主として、ServletやJSPなどのコンポーネントが使用されます。
- ビジネスロジック層
  プレゼンテーション層などからの要求に応じて業務処理やデータ提供などのビジネスサービスを供給します。一般的にこの層で業務に関する処理が行われますが、既存システムなどの資産がある場合はEIS層のリソースを利用する場合もあります。この層では主として、EJBコンポーネントを使用してビジネスロジックが実装されます。
  EIS層にあるサービス(処理やデータ)を利用する場合には、EIS層にある外部リソースや他システムなどと連携するための通信機能を提供するためにインテグレーション層を想定する場合があります。この層では、JDBCやconnectorなどのコンポーネントが使用されます。
- EIS(Enterprise Information System)層
  DB(データベース)、メインフレーム上で動作するレガシーシステムやパッケージソフトなどのリソースを提供します。

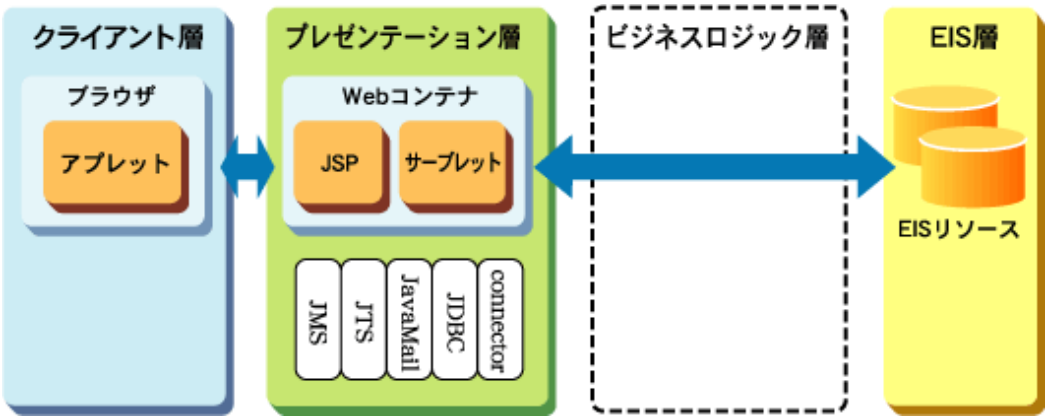
※J2EEシステムを構築する場合、システム設計者はInterstageが提供するJ2EEのコンポーネントを組み合わせて自由にシステムを構築する事ができます。以降でJ2EEのコンポーネントを用いた代表的なモデルを紹介します。

4層モデル

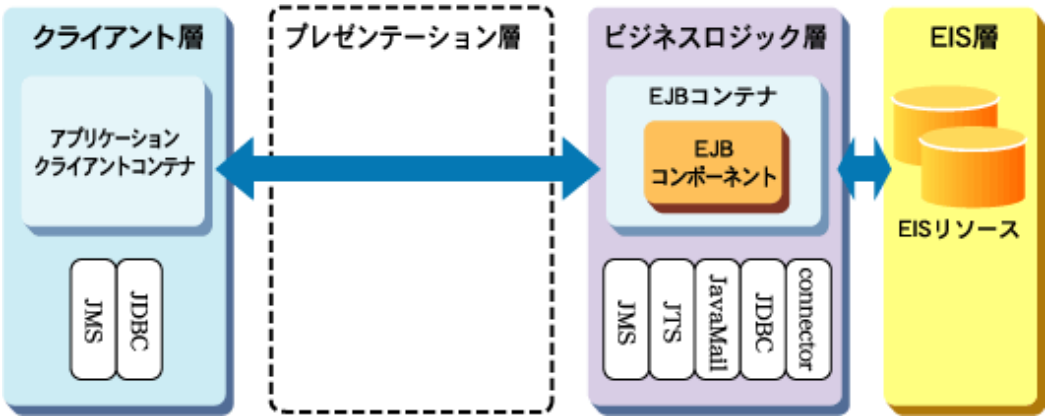


上図は4層モデルを示しています。これは大/中規模システムを構築するユーザ向けのモデルです。
このモデルでは、各層における役割分担が明確になります。通常クライアント層はWebのブラウザになります。プレゼンテーション層では、クライアントに対するビューの提供とクライアントからの要求をビジネスロジック層に引き渡す役割を担います。ビジネスロジック層では、プレゼンテーション層からの要求に応じてEIS層にあるリソースとのやり取りなどの固有ビジネスロジックを実行し、その結果をプレゼンテーション層に戻します。EIS層にはユーザの資産であるDBや既存システムが存在し、ビジネスロジック層からの利用を考慮して既存資産との接続を可能にするためのドライバプログラムなどの部品が提供されます。
4層モデルでは、各層で個別の機能に特化した開発が行いやすく、開発効率/保守性が高くなり再利用性の高い部品開発も容易になります。

3層モデル

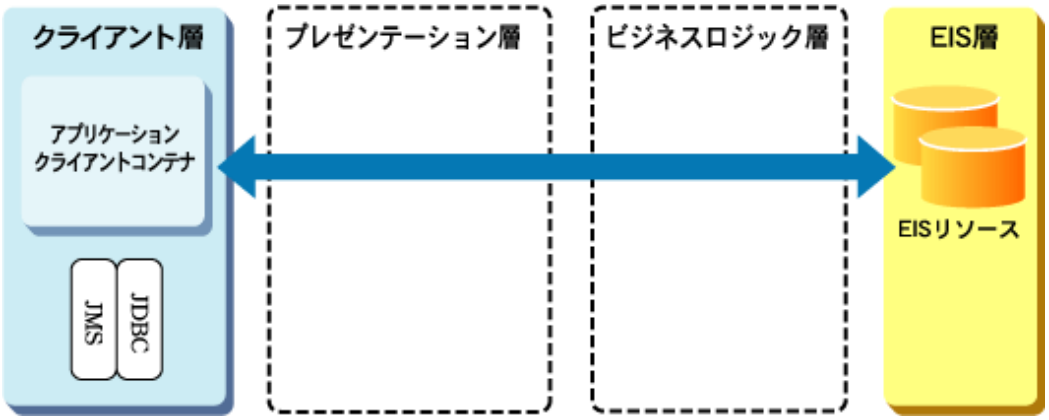


上図は3層モデルで、小/中規模システムを構築するユーザ向けのモデルです。
このモデルは、4層モデルからビジネスロジック層を除いたような構成になります。このため中間層であるプレゼンテーション層でビジネスロジックを実装する必要があり、DBへのアクセスなど比較的単純な要求を処理するシステムに適用される事が多くなります。クライアント層は通常Webのブラウザになります。



上図は3層モデルで、小/中規模システムを構築するユーザ向けのモデルです。
このモデルは、4層モデルからプレゼンテーション層を除いたような構成になります。クライアント層のアプリケーションは通常Javaアプリケーションとなり、必要に応じて利用者へのビューを実装します。クライアント層のJavaアプリケーションでは、複雑なビューを提供する事がきますが各クライアント端末に対してJavaアプリケーション配布が必要となります。

2層モデル



上図は2層モデルで、小規模システムを構築するユーザ向けのモデルです。
このモデルは、3層/4層モデルにおけるプレゼンテーション層やビジネスロジック層の機能をクライアント層で提供する必要があります。このため、2層モデルでは、単一の機能のみを処理するようなごく小規模なシステムで使用されます。

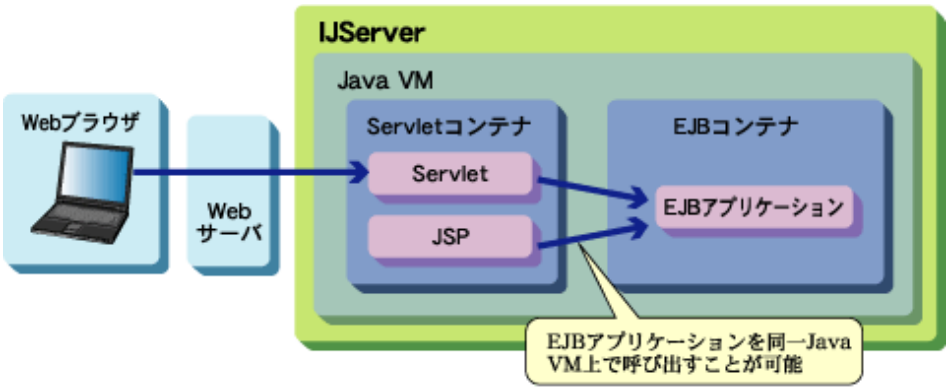
2.2 J2EEアプリケーションが運用される環境(IJServer)

Interstage Application Serverでは、運用性の向上を目的として、J2EEアプリケーションを運用する環境である**Interstage Java Server**(以降、**IJServer**と略します)という概念を導入しています。

IJServerとは

IJServerはJ2EEアプリケーションの実行環境であるEJBコンテナとServletコンテナを内包し、これらのコンテナの上位に位置づけられる論理的な概念です。
IJServerはInterstage Application Serverの特徴であるワークユニットと呼ばれるアプリケーション運用機能上で動作します。IJServerワークユニットとして動作することでワークユニットが提供している高度なアプリケーション運用操作/監視機能を利用できます。IJServerは、Interstage管理コンソールまたはisj2eeadminコマンドを使用して作成します。

ワークユニットの機能およびInterstage管理コンソールについては“運用ガイド(基本編)”を参照してください。
isj2eeadminコマンドについては、“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。



2.2.1 IJServerのタイプ

IJServerには4つのタイプがあり、用途に合わせてタイプを選択することができます。
通常はデフォルトのタイプである“WebアプリケーションとEJBアプリケーションを同一JavaVMで運用”のタイプを使用してください。

WebアプリケーションとEJBアプリケーションを同一JavaVMで運用

WebアプリケーションとEJBアプリケーションが動作するJava VMを同一で運用することができます。Servlet/JSPからEJBを高速に呼び出すことができ、アプリケーションが動作するJava VMが同一になるためメモリ資源を節約できます。
Webアプリケーションだけの場合でも運用可能です。

WebアプリケーションとEJBアプリケーションを別JavaVMで運用

WebアプリケーションとEJBアプリケーションが動作するJava VMを別々で運用することができます。アプリケーションが動作するJava VMを別々にすることでメモリ資源を消費しますが、Java VM毎にプロセス多重度を設定できたり、プロセスダウンのリスクが分散できたりするというメリットがあります。

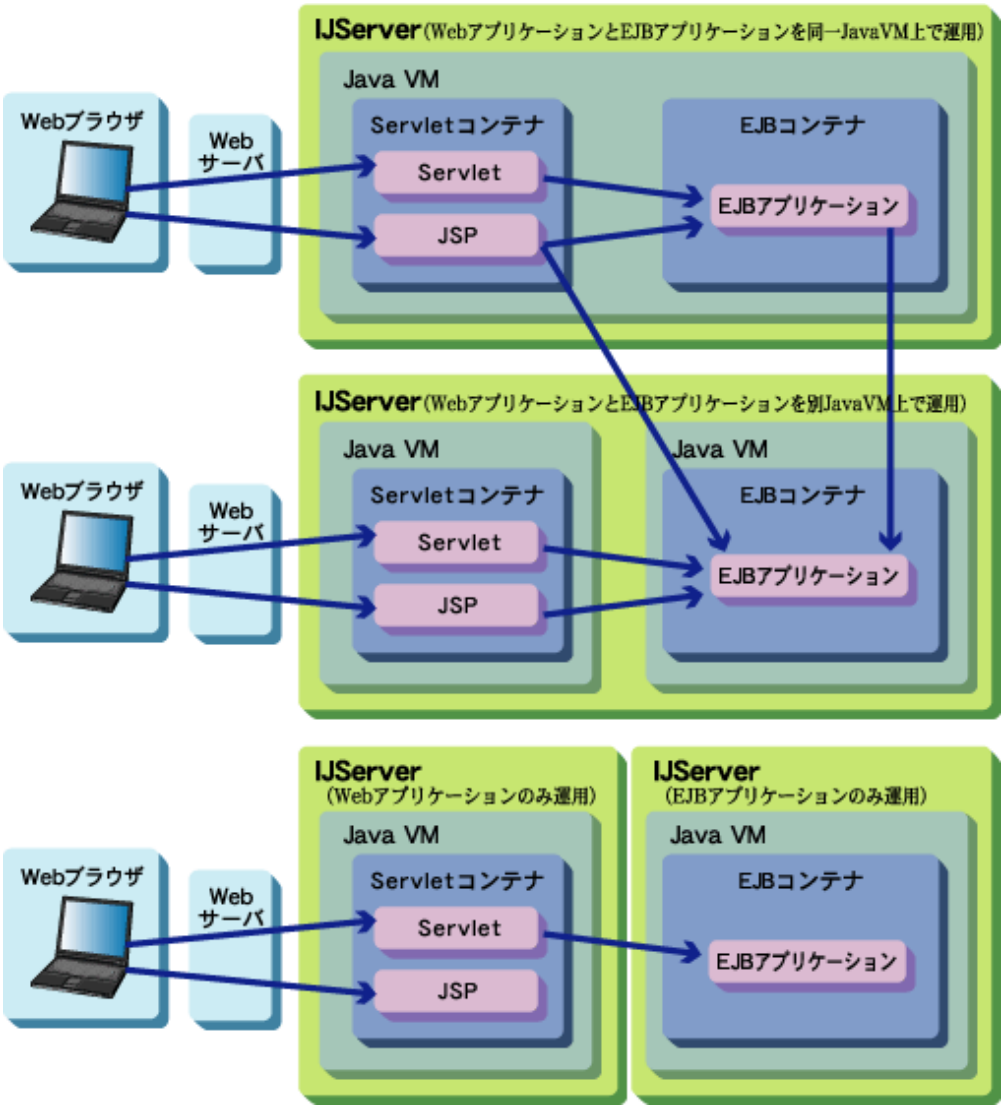
Webアプリケーションのみ運用

Webアプリケーションだけで運用することができます。

EJBアプリケーションのみ運用

EJBアプリケーションだけで運用することができます。

各種別の概要図を以下に示します。



注意

- WebアプリケーションとEJBアプリケーションを同一JavaVM上で運用する場合、EJBアプリケーションは別のIJServerのアプリケーション(またはEJBのクライアントアプリケーション)から呼び出すことはできません。
- WebアプリケーションとEJBアプリケーションを同一JavaVM上で運用する場合、EJBアプリケーションのみ配備して運用することはできません。
- WebアプリケーションとEJBアプリケーションを別JavaVM上で運用するIJServerに、EJBアプリケーションのみ配備されていた場合には、EJBのJava VMのみ起動されます。
- WebアプリケーションとEJBアプリケーションを別JavaVM上で運用するIJServerに、Webアプリケーションのみ配備されていた場合には、WebのJava VMのみ起動されます。
- Webアプリケーションのみ運用するIJServerに、EJBアプリケーションを配備しようとした場合、EJBアプリケーションの配備処理はスキップされます。また、EJBアプリケーションのみ運用するIJServerに、Webアプリケーションを配備しようとした場合、Webアプリケーションの配備処理はスキップされます。このため、EARファイルにWebアプリケーションとEJBアプリケーションが含まれている場合は、別Java VM上でWebとEJBを運用したい場合でも、Webアプリケーションのみ運用するIJServerとEJBアプリケーションのみ運用するIJServerそれぞれにEARファイルを配備することで、環境を構築することが可能です。

2.2.2 V8.0互換モードのIJServer

V9では、互換であるTomcat4.1ベースのServletサービスやJDK/JRE1.4を提供しており、その互換環境で動作する「V8.0互換モードのIJServer」を提供していました。
V10以降ではV8.0互換モードのIJServerを作成できません。本バージョン・レベルで提供するIJServerに移行してください。

2.2.3 IJServerのファイル構成

Interstage管理コンソールやisj2eeadminコマンドで作成したIJServerや配備したアプリケーションは以下の構成で管理されています。

Windows32/64

[J2EE共通ディレクトリ]¥ijserver¥
(J2EE共通ディレクトリのデフォルトは、C:¥Interstage¥J2EE¥var¥deployment です。)

Solaris64 Linux32/64

[J2EE共通ディレクトリ]/ijserver/
(J2EE共通ディレクトリのデフォルトは、/opt/FJSVj2ee/var/deployment です。)

←上位ディレクトリ 下位ディレクトリ→	説明
IJServer名	
apps	配備されたアプリケーションのファイルがモジュール名のディレクトリに展開されます。
モジュール名	EARファイルを配備した場合、その中のWAR/ejb-jar/RARファイルが展開されます。
Shared/lib Shared/classes	EARファイルに含まれていた場合に展開されます。 Shared/libディレクトリ配下のjarファイル、および、Shared/classesディレクトリ配下のクラスファイルは、EAR内のアプリケーションで共通に使用されます。 配備後に作成してもロード対象となります。
current	デフォルト設定時、配下にIJServerのカレントディレクトリが生成されます。 詳細は、“IJServerのカレントディレクトリ”を参照してください。
distribute	
モジュール名	EJBのクライアント配布物を格納します。 EARファイルを配備した場合は、配下にejb-jarファイル名のディレクトリが作成され、その配下に格納します。
conf	IJServerの内部資源です。
log	IJServer のログディレクトリです。IJServerのプロセス多重度分のプロセス通番ディレクトリがあり、このディレクトリにログファイルが出力されます。 ログディレクトリの場所は以下で変更可能です。 ・ Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [ログ出力ディレクトリ] ・ isj2eeadminコマンド 詳細は、Interstage管理コンソールヘルプ、および、“リファレンスマニュアル(コマンド編)”を参照してください。 なお、ログファイルは[ワークユニット] > “ワークユニット名” > [ログ参照]タブから参照できます。
プロセス通番	

←上位ディレクトリ 下位ディレクトリ→		説明
Shared		“2.3 クラスローダ”の“2.3.1 クラスローダの構成”および“2.3.4 IJServerで使用するクラスの設定について”を参照してください。 Shared/lib配下のjarファイルはIJServerが起動している場合は上書きおよび削除することはできません。
	classes	
	lib	
work		以下のサブディレクトリが生成され、コンテナのテンポラリファイル(JSPから生成されたjavaソースやコンパイル結果など)が格納されます。 Windows32/64 [プロセス通番]¥Catalina¥localhost¥Webアプリケーション名 Solaris64 Linux32/64 [プロセス通番]/Catalina/localhost/Webアプリケーション名
ext		“2.3 クラスローダ”の“2.3.1 クラスローダの構成”および“2.3.4 IJServerで使用するクラスの設定について”を参照してください。

注意

- アプリケーション資源の権限については、“3.5 J2EEアプリケーションの配備と設定”の“アプリケーションファイル保護レベル”を参照してください。
- Solaris64 Linux32/64
  “IJServer名”ディレクトリ配下にシンボリックリンクを作成しないでください。
  “IJServer名”配下のディレクトリにシンボリックリンクがある場合、IJServerの削除時に、シンボリックリンク参照先も削除します。

IJServerのカレントディレクトリ

デフォルトでは、IJServerのカレントディレクトリは以下のディレクトリです。

Windows32/64

[J2EE共通ディレクトリ]¥ijserver¥[IJServer名]¥current¥[IJServer名 *]¥[プロセスID]
(J2EE共通ディレクトリのデフォルトは、C:¥Interstage¥J2EE¥var¥deploymentです。)

Solaris64 Linux32/64

[J2EE共通ディレクトリ]/ijserver/[IJServer名]/current/[IJServer名 *]/[プロセスID]
(J2EE共通ディレクトリのデフォルトは、/opt/FJSVj2ee/var/deploymentです。)

- 上記の下線部分は、Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [カレントディレクトリ]で指定可能です。また、isj2eeadminコマンドで変更することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。
- 上記の[IJServer名 *]のディレクトリ名部分は、ワークユニットの起動時にIJServer名.old1、IJServer名.old2、・・・、IJServer名.old5のようにバックアップを残し、最新のディレクトリ名はIJServer名になります。

指定の際に、“IJServerで一意のカレントディレクトリとする”をチェックすると、指定ディレクトリそのものがカレントディレクトリとなり、実際のカレントディレクトリをIJServerとして一意にすることができます。

以下に、カレントディレクトリに関する設定とカレントディレクトリの関係をまとめます。

“カレントディレクトリ”の選択肢	ディレクトリ名	“IJServerで一意のカレントディレクトリとする”	カレントディレクトリ
デフォルトのディレクトリ構成	(指定不可) Windows32/64	ー (無効)	Windows32/64 J2EE共通ディレクトリ¥ijserver¥[IJServer名]¥current¥[IJServer名 *]¥[プロセスID]

“カレントディレクトリ”の選択肢	ディレクトリ名	“IJServerで一意のカレントディレクトリとする”	カレントディレクトリ
	J2EE共通ディレクトリ¥ijserver Solaris64 Linux32/64 J2EE共通ディレクトリ/ijserver		Solaris64 Linux32/64 J2EE共通ディレクトリ/ijserver/[IJServer名]/current/[IJServer名 *]/[プロセスID]
ユーザ指定	指定例: Windows32/64 C:¥tmp¥current Solaris64 Linux32/64 /tmp/current	チェック時	Windows32/64 C:¥tmp¥current Solaris64 Linux32/64 /tmp/current
		非チェック時	Windows32/64 C:¥tmp¥current¥[IJServer名]¥current¥[IJServer名 *]¥[プロセスID] Solaris64 Linux32/64 /tmp/current/[IJServer名]/current/[IJServer名 *]/[プロセスID]

注意

“IJServerで一意のカレントディレクトリとする”をチェックした場合は、以下の点に注意してください。

- カレントディレクトリは世代管理されません。
- 指定ディレクトリが存在しない場合、新規作成されます。
- IJServerの削除時やカレントディレクトリの変更時にも、指定ディレクトリは削除されません。
- Solaris64 Linux32/64
  以下の場合にcoreファイルが上書きされてしまう可能性があり、トラブル発生時に調査情報が入手できません。そのため、“IJServerで一意のカレントディレクトリとする”をチェックしないことを推奨します。
  - ワークユニット設定のプロセス多重度に2以上を指定している場合
  - ワークユニットを再起動した場合

2.2.4 起動/停止の実行クラス

実行クラスとは

IJServerの起動時や停止時に任意のJavaアプリケーションを呼び出すことができます。
IJServerの起動時に呼び出すJavaアプリケーションを起動時実行クラスと呼び、IJServerの停止時に呼び出すJavaアプリケーションを停止時実行クラスと呼びます。
実行クラスでは、IJServerやJava VMプロセス単位で一度だけ行う初期化や終了処理を実装することにより、以下のような処理に利用することができます。

- データベースの初期化、終了処理
- JNDIによるObjectのlookup処理
- EJBアプリケーションメソッドの呼び出し

実行クラスからは、以下がJNDIで参照可能です。

・ EJBHomeオブジェクト

　－ EJB Homeオブジェクト

　－ EJB LocalHomeオブジェクト

・ リソース参照

　－ JDBCデータソース

　－ JDBCデータソースを使用してコネクションの獲得・データベースアクセス・コネクション解放が可能です。

また、実行クラスからは、以下のEJBアプリケーションが呼び出しが可能です。

・ STATEFUL Session Bean

・ STATELESS Session Bean

・ Entity Bean BMP

・ Entity Bean CMP1.1

・ Entity Bean CMP2.0

実行クラスを呼び出す契機

実行クラスを呼び出す契機は以下のとおりです。

・ 起動時実行クラスは、IJServerの起動後、アプリケーションへのリクエストの受付を開始する前に呼び出されます。

・ 停止時実行クラスは、IJServerの停止時、アプリケーションへのリクエストの受付を停止した直後に呼び出されます。IJServerを強制停止した場合には停止時実行クラスは呼び出されません。

・ 実行クラスは複数指定することができ、呼び出す順番を決めることができます。

・ 実行クラスの用途に合わせてすべてのプロセス(Java VM)で呼び出すか、IJServer単位で一度のみ呼び出すかを選択することができます。

呼び出し方法	利用例	例
すべてのプロセスで呼び出し	すべてのプロセスで行わなければならない処理	JNDIによるObjectのlookup処理
一度のみ呼び出し	プロセスごとの実行が必要でない処理や、プロセス間で共有することのできる処理	データベースの初期化、終了処理

実行クラスの作成方法

実行クラスを作成するために、特別なクラスやインタフェースを用意する必要はありません。
以下の2つの条件を満たすJavaアプリケーションを作成してください。

・ mainメソッドが実装され、コマンドラインから実行が可能である。

・ publicクラスとして宣言されている。

以下に、単純な実行クラスの例を示します。

```
package test;
public class UserStartup{
  public static void main(String args[]){
    if(args.length == 1){
      System.out.println(args[0]);
    }
    //ユーザ実装をここに記述
  }
}
```

実行クラスの登録方法

実行クラスの登録は、以下に示す2つの設定をInterstage管理コンソールで行います。詳細はInterstage管理コンソールのヘルプを参照してください。また、isj2eeadminコマンドで登録することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

1. 実行クラスの設定
   実行クラスの設定情報を登録します。
2. クラスパスの設定
   実行クラスをワークユニットのクラスパスに設定します。

注意

- IJServerのタイプが「WebアプリケーションとEJBアプリケーションを別JavaVMで運用」の場合、Servletコンテナ用とEJBコンテナ用の2つのクラスパスが存在します。実行クラスを呼び出すコンテナのクラスパスに設定してください。
- IJServerのタイプが「WebアプリケーションとEJBアプリケーションを別JavaVMで運用」の場合、コンテナ依存の処理に対応するために、実行クラスを起動するコンテナを次の3つの中から選択してください。
  - Webコンテナ
  - EJBコンテナ
  - Web・EJB両方
- 起動時実行クラスで発生した例外がIJServerワークユニットにthrowされた場合に、IJServerの起動を続行するか中止するかを選択することができます。
- クラスローダの分離が「EAR間で分離」または「すべて分離」の場合、起動時実行クラスまたは停止時実行クラスからEJBアプリケーションを呼び出すことはできません。
- 実行クラスから別のJava VM上のEJBアプリケーションに対して処理を行う場合、実行クラスを起動する前にあらかじめEJBアプリケーションを配備したIJServerを起動しておく必要があります。
  IJServerのタイプが「WebアプリケーションとEJBアプリケーションを別JavaVMで運用」の場合、Webコンテナで設定した実行クラスから同一IJServer内のEJBアプリケーションに対する処理呼び出しは保証されません。
  以下に、処理が実行できない場合の例と回避方法を示します。

処理できない例	回避方法
IJServerのタイプが「WebアプリケーションとEJBアプリケーションを別JavaVMで運用」の場合、Webコンテナで設定した実行クラスから同一IJServer内のEJBアプリケーションのlookup処理。	以下のどちらかを選択する。 ・ 実行クラスを起動するコンテナをEJBコンテナとする。 ・ 実行クラスが設定されているIJServerとは別のIJServerにEJBアプリケーションを配備し、起動しておく。

- 起動時実行クラスまたは停止時実行クラスから呼び出したEJBアプリケーション内ではEJBタイマーサービスの機能は使用できません。

2.2.5 ネットワーク上の共有資源へアクセスする場合の環境設定 Windows32/64

IJServer上に配備されたアプリケーションから、ネットワーク上の共有資源の参照および更新を行う場合の設定手順を以下に示します。

共有資源へアクセスするIJServerが動作するマシン(マシンA)側の操作

1. Interstageを停止
2. Administrators権限を持ったユーザを作成

3. TransactionDirectorのログオンアカウントに2.のユーザを指定
   TransactionDirectorのログオンアカウントの指定は、Administrator権限でログインし、「コントロールパネル」の「サービス」または「コントロールパネル」－「管理ツール」－「サービス」を起動後、“TransactionDirector”を選択し、[ログオン]により行います。

   － Windows Server(R) 2012の場合は、INTERSTAGEおよびInterstage Operation Toolのログオンアカウントも2.のユーザを指定
   INTERSTAGEおよびInterstage Operation Toolのログオンアカウントの指定は、Administrator権限でログインし、「コントロールパネル」の「サービス」または「コントロールパネル」－「管理ツール」－「サービス」を起動後、“INTERSTAGE”および“Interstage Operation Tool”を選択し、[ログオン]により行います。

4. Interstageを起動

共有したい資源が存在するマシン(マシンB)側の操作

1. マシンAの2.で作成したユーザと同一のユーザ名/パスワードでユーザを作成
   このユーザは共有資源にアクセス可能な権限を保持していれば、Administrators権限を持っている必要はありません。
2. 共有したい資源に対して共有を設定

アプリケーションの実装方法

共有資源へUNC接続でアクセスします。

例

```
¥¥[ipアドレス]¥[共有名]¥a.txt または ¥¥[host名]¥[共有名]¥a.txt
```

実際のコーディングではエスケープ文字を追加してください。

注意

- 手順上、各マシンにユーザの作成、共用の設定などWindows(R) OS上の操作が必要です。これらの操作に伴うセキュリティ上の意味について、Windows(R) OS上の観点からも考慮してください。
  例: 作成したユーザのIDおよびパスワードの管理、意図しないユーザからのアクセス制御
- クラスタサービスを使用している場合は、本設定を使用できません。
- 共有資源へアクセスするIJServerが動作するマシン(マシンA)側で作成したユーザは、Guestsグループに所属しないでください。Guestsグループに所属したユーザをTransactionDirectorのログオンアカウントに指定した場合、Interstageが正常に動作しないことがあります。

2.2.6 Javaヒープ領域/メタスペース不足時の制御

Javaヒープ領域、またはメタスペースが不足した場合のIJServerの制御を選択できます。IJServerのタイプが“WebアプリケーションとEJBアプリケーションを別JavaVMで運用”のIJServerの場合は、Servletコンテナ、EJBコンテナごとに設定することができます。
以下に、指定可能なオプションについて説明します。

プロセスを再起動する

Javaヒープ領域、メタスペース、Cヒープ領域が不足した場合に、領域不足が発生したプロセスを再起動(注)します。再起動することで、運用を再開できます。また、IJServerを複数プロセスで運用することにより、プロセスの再起動中は別プロセスで処理が継続されるため、運用を停止することなく連続運用が可能となります。
プロセスの終了時にEXTP4435のメッセージをシスログに出力します。頻繁に発生する場合は、“メッセージ集” － “メッセージ番号がEXTPで始まるメッセージ” － “EXTP4435”のユーザの対処を参照し、JavaVMオプションを指定してください。

アプリケーションにjava.lang.OutOfMemoryErrorを返却する

通常のJavaアプリケーション同様、アプリケーション、およびServlet/EJBコンテナにjava.lang.OutOfMemoryErrorを返却します。
OutOfMemoryError がServlet/EJBコンテナに返却された場合の制御は、コンテナにより異なります。

- Servletコンテナの場合
  可能な限り処理を継続します。ただし、不足した資源や不足量、および発生のタイミングによっては、正常に運用できなくなる(リクエスト処理失敗でステータス500、セションの情報消失、内部異常など)可能性があります。

- EJBコンテナの場合
  プロセスを再起動(注)しますが、この場合はEXTP4435のメッセージは出力されません。
  ただし、「プロセスを再起動する」を指定した場合とは異なり、プロセスが終了しない場合があります。
  注) 再起動の回数は、リトライカウントで指定します。

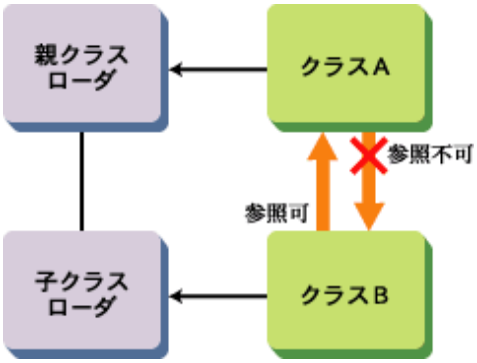
いずれの場合でもJavaプロセスで使用する資源については、“27.2 IJServerのチューニング”を参照し、チューニングと検証を行ってから運用してください。

2.3 クラスローダ

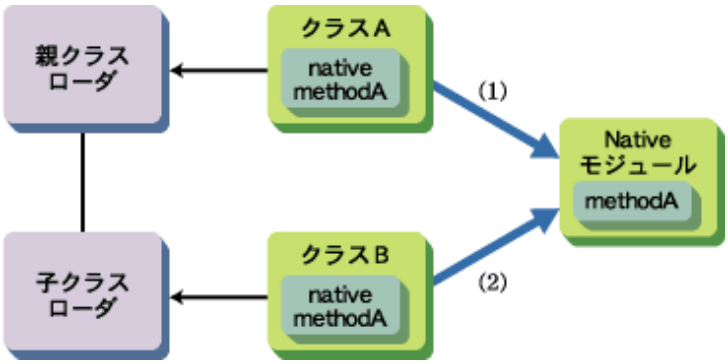
Javaのクラスローダはクラスファイルを検索し、クラスをメモリにロードする機能を提供します。
J2EEアプリケーションを作成する際は、クラスがどのクラスローダでロードされるかを理解してJ2EEアプリケーションを構築してください。

クラスローダの階層

Javaのクラスローダは親クラスローダと子クラスローダからなる階層構造をもっています。親クラスローダと子クラスローダの関係はオブジェクトのスーパークラスとサブクラスの関係に似ています。
子クラスローダでロードされたクラスから親クラスローダでロードされたクラスを参照することはできますが、親クラスローダでロードされたクラスから子クラスローダでロードされたクラスを参照することはできません。



JNI(Java Native Interface)を利用している場合はさらに注意が必要です。JNIを利用する場合はNativeモジュールをクラスローダがロードしますが、Javaのクラスローダでは同じNativeモジュールは同じクラスローダ上からのみ利用可能となります。



(1)と(2)で同じNativeモジュールを使用していますが、先にNativeモジュールを読み込んだ側からのみ利用可能となります。
クラスBが先に動作した場合はクラスAには“java.lang.UnsatisfiedLinkError”がスローされます。
上記は、Interstageの場合にも当てはまります。

クラスのロード

一般的には、クラスローダは委譲モデルを使ってクラスをロードします。クラスローダは関連する親クラスローダを持ち、クラスをロードするために呼び出されると、クラスローダはそれ自体でクラスのロードを試みる前に、該当クラスのロードを親クラスローダに委譲します。
Interstageが利用するクラスとアプリケーションが利用するクラスの独立性を高めるために、Interstageのクラスローダはこの委譲モデルとは少し違う動作をします。Interstageのクラスローダの検索順番については以降で説明します。

2.3.1 クラスローダの構成

ここではIJServerのクラスローダの階層構造について説明します。
IJServerで使用するクラスローダは、親クラスローダおよび子クラスローダからなる階層構造をもっています。このようにクラスローダが階層構造をもつことによりシステムとアプリケーションおよびアプリケーション間の独立性が向上します。

クラスローダの階層はカスタマイズ可能です。カスタマイズ方法については“2.3.2 クラスローダの分離”を参照してください。

各クラスローダは下表の資源をロードします。Interstageのデフォルトの設定では、クラスローダは表の上から順にクラスをロードします。クラスをロードする順序はカスタマイズ可能です。カスタマイズ方法については“2.3.3 クラスローダの検索順番の変更”を参照してください。

クラスローダの分離が“EAR間で分離”の場合

クラスローダ	ロードする資源	設定方法
システムクラスローダ	XMLパーサのクラス	ワークユニットのXMLパーサの環境設定
Applicationクラスローダ	IJServer内で共通のクラス(classファイル)	アプリケーション固有ライブラリパス
	Connectorが使用するConnector以外のクラス(classファイル) 注) ConnectorをIJServerに配備した場合	RARファイル内のマニフェストクラスパス
	EJBアプリケーションのクラス(classファイル)	ejb-jarファイル
	EJBアプリケーションが使用するEJBアプリケーション以外のクラス(classファイル)	ejb-jarファイル内のマニフェストクラスパス

クラスローダ	ロードする資源	設定方法
	アプリケーション内で共通に使用するクラス(classファイル)	EARファイル内のShared/classesディレクトリ
	IJServer内で共通のクラス(Jarファイル)	アプリケーション固有ライブラリパス
	Connectorのクラス(Jarファイル) 注) ConnectorをIJServerに配備した場合	RARファイル
	Connectorが使用するConnector以外のクラス(Jarファイル) 注) ConnectorをIJServerに配備した場合	RARファイル内のマニフェストクラスパス
	EJBアプリケーションが使用するEJBアプリケーション以外のクラス(Jarファイル)	ejb-jarファイル内のマニフェストクラスパス
	アプリケーション内で共通に使用するクラス(Jarファイル)	EARファイル内のShared/libディレクトリ
Webappクラスローダ	Webアプリケーションのクラス(classファイル)	WARファイル内のWEB-INF/classesディレクトリ
	Webアプリケーションが使用するWebアプリケーション以外のクラス(classファイル) 注) EARに含まれる場合	WARファイル内のマニフェストクラスパス
	Webアプリケーションのクラス(Jarファイル)	WARファイル内のWEB-INF/libディレクトリ
	Webアプリケーションが使用するWebアプリケーション以外のクラス(Jarファイル)	WARファイル内のマニフェストクラスパス
Interstageクラスローダ	Interstageのクラス	なし(設定できません)
	IJServer内で共通のクラス	ワークユニットのクラスパス
		IJServerのSharedディレクトリ内に保管
	複数のIJServer間で共通のクラス	J2EEプロパティのクラスパス

クラスローダの分離が“すべて分離”の場合

クラスローダ	ロードする資源	設定方法
システムクラスローダ	XMLパーサのクラス	ワークユニットのXMLパーサの環境設定
Applicationクラスローダ	IJServer内で共通のクラス(classファイル) 注) EAR、ejb-jarをIJServerに配備した場合	アプリケーション固有ライブラリパス
	Connectorが使用するConnector以外のクラス(classファイル) 注) ConnectorをIJServerに配備した場合	RARファイル内のマニフェストクラスパス
	EJBアプリケーションのクラス(classファイル)	ejb-jarファイル
	EJBアプリケーションが使用するEJBアプリケーション以外のクラス(classファイル)	ejb-jarファイル内のマニフェストクラスパス

クラスローダ	ロードする資源	設定方法
	アプリケーション内で共通に使用するクラス(classファイル)	EARファイル内のShared/classesディレクトリ
	IJServer内で共通のクラス(Jarファイル) 注) EAR、ejb-jarをIJServerに配備した場合	アプリケーション固有ライブラリパス
	Connectorのクラス(Jarファイル) 注) ConnectorをIJServerに配備した場合	RARファイル
	Connectorが使用するConnector以外のクラス(Jarファイル) 注) ConnectorをIJServerに配備した場合	RARファイル内のマニフェストクラスパス
	EJBアプリケーションが使用するEJBアプリケーション以外のクラス(Jarファイル)	ejb-jarファイル内のマニフェストクラスパス
	アプリケーション内で共通に使用するクラス(Jarファイル)	EARファイル内のShared/libディレクトリ
Webappクラスローダ	IJServer内で共通のクラス(classファイル) 注) WARをIJServerに配備した場合	アプリケーション固有ライブラリパス
	Webアプリケーションのクラス(classファイル)	WARファイル内のWEB-INF/classesディレクトリ
	Webアプリケーションが使用するWebアプリケーション以外のクラス(classファイル) 注) EARに含まれる場合	WARファイル内のマニフェストクラスパス
	IJServer内で共通のクラス(Jarファイル) 注) WARをIJServerに配備した場合	アプリケーション固有ライブラリパス
	Webアプリケーションのクラス(Jarファイル)	WARファイル内のWEB-INF/libディレクトリ
	Webアプリケーションが使用するWebアプリケーション以外のクラス(Jarファイル)	WARファイル内のマニフェストクラスパス
Interstageクラスローダ	Interstageのクラス	なし(設定できません)
	IJServer内で共通のクラス	ワークユニットのクラスパス
		IJServerのSharedディレクトリ内に保管
	複数のIJServer間で共通のクラス	J2EEプロパティのクラスパス

クラスローダの分離が“分離しない”の場合

クラスローダ	ロードする資源	設定方法
システムクラスローダ	XMLパーサのクラス	ワークユニットのXMLパーサの環境設定
	Interstageのクラス	なし(設定できません)
	IJServer内で共通のクラス	ワークユニットのクラスパス
		IJServerのSharedディレクトリ内に保管
	複数のIJServer間で共通のクラス	J2EEプロパティのクラスパス

クラスローダ	ロードする資源	設定方法
	IJServer内で共通のクラス	アプリケーション固有ライブラリパス
	複数のIJServer間で共通のクラス	環境変数:CLASSPATH
	Connectorのクラス	RARファイル
	EJBアプリケーションのクラス	ejb-jarファイル
	EJBアプリケーションが使用するEJBアプリケーション以外のクラス	ejb-jarファイル内のマニフェストクラスパス
	アプリケーション内で共通に使用するクラス	EARファイルのSharedディレクトリ内に保管
Webappクラスローダ	Webアプリケーションのクラス(classファイル)	WARファイル内のWEB-INF/classesディレクトリ
	Webアプリケーションが使用するWebアプリケーション以外のクラス(classファイル) 注) EARに含まれる場合	WARファイル内のマニフェストクラスパス
	Webアプリケーションのクラス(Jarファイル)	WARファイル内のWEB-INF/libディレクトリ
	Webアプリケーションが使用するWebアプリケーション以外のクラス(Jarファイル)	WARファイル内のマニフェストクラスパス

注意

- IJServerのextディレクトリ内に格納されたクラスはシステムクラスローダに設定され、上記のクラスに優先して使用されます。詳細は、“2.3.4 IJServerで使用するクラスの設定について”を参照してください。
- 上記のほか、IJServerの動作に必要な資源は自動的にロードされます。

2.3.2 クラスローダの分離

Interstageのデフォルトの設定では、EAR間でクラスローダが分離されます。
クラスローダの分離はアプリケーション間の参照関係やアプリケーションの活性変更に影響があります。
ここでは、クラスローダの分離をカスタマイズする機能について説明します。

設定方法

クラスローダの分離方法は、Interstage管理コンソールを使用して、以下のどちらかの方法で設定します。

- [ワークユニット] > [新規作成] > [詳細設定] > [共通定義] > [クラスローダの分離]
- [ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [共通定義] > [クラスローダの分離]

また、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

想定する設定パターンは下表のようになります。

設定値	設定パターン
EAR間で分離(デフォルト値)	EARを作成しないでejb-jarを個々に配備している場合に使用。
すべて分離	EARを作成してアプリケーションを配備している場合に使用。
分離しない	EAR間でアプリケーションを参照している場合や、システムクラスローダでEJBアプリケーションやconnectorをロードする場合に使用。

設定値	設定パターン
	V6とクラスローダの構成が同じV6互換モード。 HotDeployは使用しない、かつInterstageV6で作成したアプリケーションを移行する場合に使用。(または、EAR間で分離、すべて分離では動作しないアプリケーションの場合)

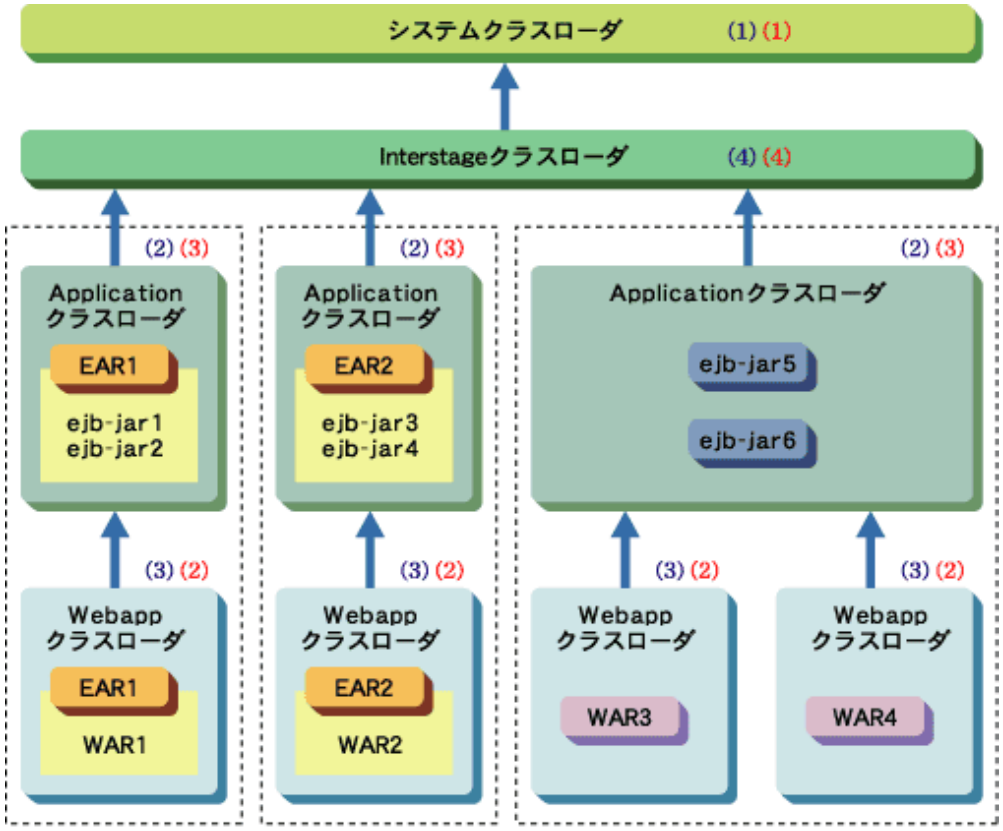
注意

- WebアプリケーションはWebアプリケーション間でクラスを相互に参照することはないため、設定値(“EAR間で分離”、“すべて分離”、“分離しない”)にかかわらず、すべての場合でWebアプリケーション間のクラスローダは分離されます。
- IJServerのタイプが“WebアプリケーションとEJBアプリケーションを同一JavaVMで運用”の場合は、WebアプリケーションやEJBアプリケーションから他のIJServerに配備されたEJBアプリケーションを呼び出すことはできません。

クラスローダの分離パターン

EAR間で分離

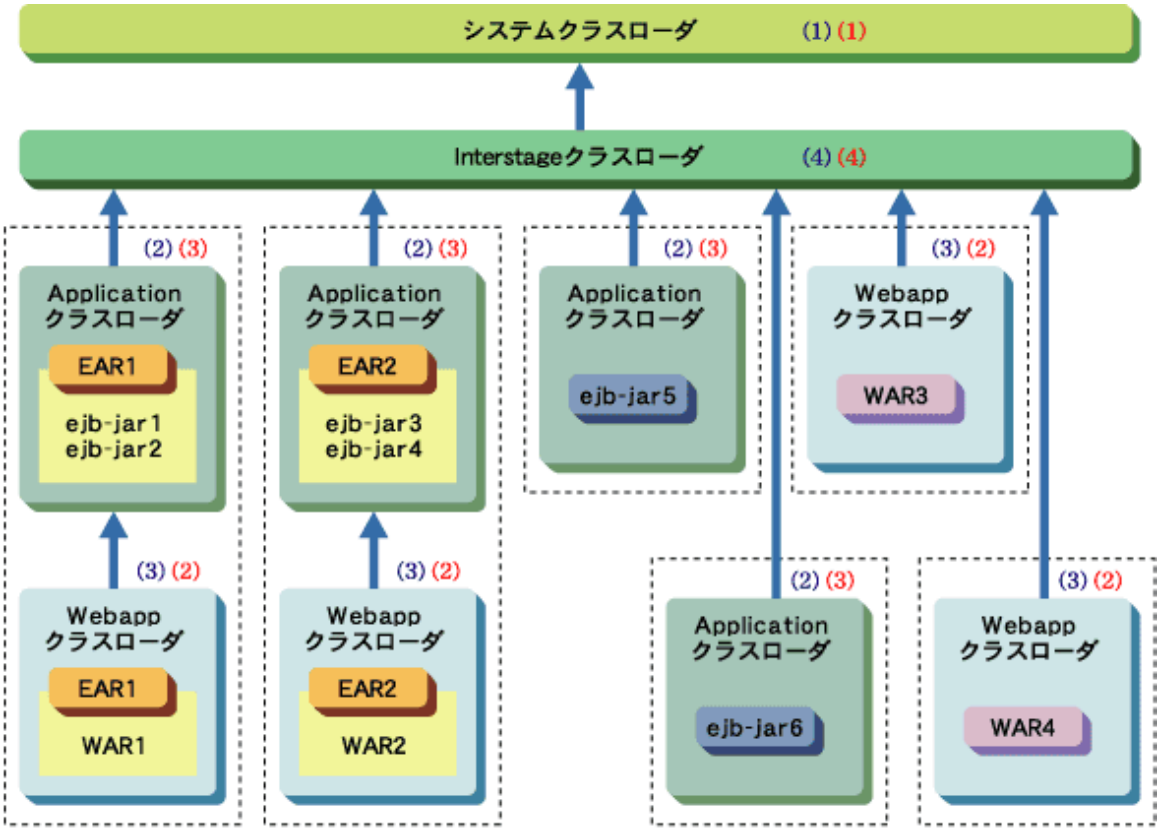
“EAR間で分離”を選択した場合、EARの配備では、下図のようにEAR間でクラスローダが分離されます。
ejb-jar、WARの配備では、クラスローダは配備単位で分離されません。
この場合、J2EEアプリケーションの活性変更は点線で囲まれた単位で可能となります。



注）図中の↑はクラスローダの親子関係を表しています。（終点が親）
(1), (2), (3), (4) は、“親が先”設定時のクラスローダの検索順序を表しています。
(1), (2), (3), (4) は、“親が後”設定時のクラスローダの検索順序を表しています。

すべて分離

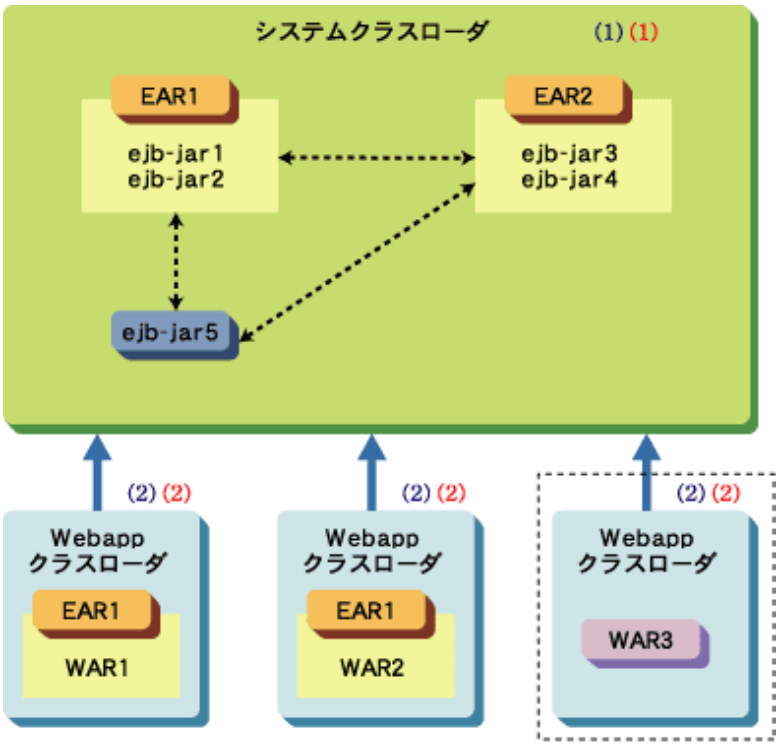
“すべて分離”を選択した場合、下図のように配備単位でクラスローダが分離されます。
この場合、J2EEアプリケーションの活性変更は点線で囲まれた単位で可能となりますが、EJBアプリケーションEJB-JAR5、EJB-JAR6の相互参照およびWebアプリケーションWAR3、WAR4からEJBアプリケーションの参照はできません。そのため、異なるアプリケーションで同じパッケージ名、クラス名のクラスを使用することができるようになります。



注）図中の↑はクラスローダの親子関係を表しています。（終点が親）
(1), (2), (3), (4) は、“親が先”設定時のクラスローダの検索順序を表しています。
(1), (2), (3), (4) は、“親が後”設定時のクラスローダの検索順序を表しています。

分離しない

“分離しない”を選択した場合、下図のように、Webアプリケーション以外のクラスはすべて“システムクラスローダ”でロードされます。
この場合、J2EEアプリケーションの活性変更はWARファイルを配備したWebアプリケーションのみが対象となります(下図の点線で囲まれた個所)。
アプリケーション間の参照はEJBアプリケーションがお互いのクラスを参照することができます。また、Webアプリケーションからすべてのアプリケーションのクラスを参照することができます。



注）図中の↑はクラスローダの親子関係を表しています。（終点が親）
　　また、↕はクラスの参照関係を表しています。
(1), (2) は、“親が先”設定時のクラスローダの検索順序を表しています。
(1), (2) は、“親が後”設定時のクラスローダの検索順序を表しています。

2.3.3 クラスローダの検索順番の変更

Interstageのデフォルトの設定では、Applicationクラスローダ、Webappクラスローダの順番でクラスが検索されます。
クラスローダの検索順番を“親が後”に設定することでWebappクラスローダ、Applicationクラスローダの順番でクラスを検索することもできます。クラスローダの検索順番を“親が後”に設定する利点としては、たとえば同一アプリケーション内に同一クラス名を持つクラスがあった場合、クラスローダの検索順番を“親が後”に設定することにより、クラスローダは親のクラスローダにロードされるクラスをオーバーライドし、独自バージョンのクラスを持つことができます。
クラスローダの分離を“分離しない”とした場合は、クラスローダはWebappクラスローダ、システムクラスローダの2つだけが存在し、システムクラスローダは常に最初に検索されるため、クラスローダの検索順番は設定不要です。

設定方法

クラスローダの検索順番は、Interstage管理コンソールを使用して、以下のどちらかの方法で設定します。

- [ワークユニット] > [新規作成] > [詳細設定] > [共通定義] > [クラスローダの検索順序]
- [ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [共通定義] > [クラスローダの検索順序]

また、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

設定値によりクラスローダの検索順番は下表のようになります。

設定値	検索順番
親が先 (デフォルト値)	システムクラスローダ Applicationクラスローダ Webappクラスローダ Interstageクラスローダ

設定値	検索順番
親が後	システムクラスローダ Webappクラスローダ Applicationクラスローダ Interstageクラスローダ

優先度の例外

クラスローダの優先度については以下の2つの例外があります。

- JDKのクラス
  JDKのクラスは常に最初にロードされますので、このクラスはユーザが置き換えることはできません。
- 特定パッケージ名で始まるクラス
  以下のパッケージ名で始まるクラスは常に最初に親クラスローダにロードが委譲されます。
  親クラスローダに以下のパッケージ名で始まるクラスが存在する場合は、子クラスローダでクラスを置き換えることはできません。

パッケージ名	種類
javax	Java extensions
org.xml.sax	SAX 1 & 2
org.w3c.dom	DOM 1 & 2
com.fujitsu.interstage	Interstageのクラス
com.fujitsu.ObjectDirector	Interstageのクラス(ObjectDirector)

2.3.4 IJServerで使用するクラスの設定について

ここではIJServerが使用するクラスやアプリケーションが使用するクラスの設定方法について説明します。

設定したクラスの参照順については、“2.3.1 クラスローダの構成”および“2.3.3 クラスローダの検索順番の変更”を参照してください。
また、設定したクラスのHotDeploy機能、およびクラスのオートリロード機能での扱いについては、それぞれ“3.5.3 J2EEのHotDeploy機能”、“3.5.4 クラスのオートリロード機能”を参照してください。

設定の確認

IJServerを起動すると起動時のクラスパスの情報が以下の起動情報(info.log)およびコンテナログ(container.log)に出力されます。起動情報およびコンテナログを参照することでクラスパスの設定を確認することができます。起動情報およびコンテナログは、Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [ログ参照]タブから参照できます。

Windows32/64

起動情報　: IJServer名¥log¥[プロセス通番]¥info.log
コンテナログ : IJServer名¥log¥[プロセス通番]¥container.log

Solaris64 Linux32/64

起動情報　: IJServer名/log/[プロセス通番]/info.log
コンテナログ : IJServer名/log/[プロセス通番]/container.log

設定方法

ここでは、以下の設定方法について説明します。

- XMLパーサ
- 複数のIJServer間で共通のクラス

- IJServer内で共通のクラス
- 環境変数:CLASSPATH
- アプリケーションのクラス

XMLパーサ

“2.3.5 XMLパーサの設定”を参照してください。

複数のIJServer間で共通のクラス

複数のIJServer間で共通に使用するクラスは、J2EEプロパティのクラスパスに設定します。
複数のIJServer間で共通のクラスにはJDBCドライバなどユーザがアプリケーションで使用するライブラリを自由に設定することができます。

J2EEプロパティのクラスパスは、Interstage管理コンソールを使用して、以下の方法で設定します。

- [システム] > [環境設定]タブ > [J2EEプロパティ] > [クラスパス]

また、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

IJServer内で共通のクラス

ここでは、IJServer内で共通に使用するクラスの設定方法について説明します。
IJServer内で共通に使用するクラスの設定方法はワークユニットのクラスパスに設定する方法、IJServerのSharedディレクトリに設定する方法、アプリケーション固有ライブラリパスに設定する方法、IJServerのextディレクトリに設定する方法があります。

ワークユニットのクラスパスに設定する方法

ワークユニットのクラスパスに、jarファイルまたはクラスファイルが保管されたディレクトリを絶対パスで指定します。
ワークユニットのクラスパスは、Interstage管理コンソールを使用して、以下のどちらかの方法で設定します。

- [ワークユニット] > [新規作成] > [詳細設定] > [ワークユニット設定] > [クラスパス]
- [ワークユニット] > “ワークユニット名” > [環境設定]タブ > [ワークユニット設定] > [クラスパス]

また、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

IJServerディレクトリ配下のSharedディレクトリに設定する方法

以下のどちらかの方法で設定します。

- IJServerのShared/libディレクトリにjarファイルを保管
- IJServerのShared/classesディレクトリにクラスファイルを保管

Shared/libとShared/classes内に同一のパッケージ名、クラス名のクラスが存在した場合は、Shared/classesディレクトリ内のクラスがロードされます。

注意

Sharedディレクトリに資源を保管する際、保管できる権限が一般ユーザに付与されていない場合は、必要に応じて管理者が権限を変更してください。

アプリケーション固有ライブラリパスに設定する方法

ワークユニットのアプリケーション固有ライブラリパスに、jarファイルまたはアプリケーション固有ライブラリのクラスファイルが保管されたディレクトリを絶対パスで指定します。
ワークユニットのアプリケーション固有ライブラリパスは、Interstage管理コンソールを使用して、以下のどちらかの方法で設定します。

- [ワークユニット] > [新規作成] > [詳細設定] > [ワークユニット設定] > [アプリケーション固有ライブラリパス]
- [ワークユニット] > “ワークユニット名” > [環境設定]タブ > [ワークユニット設定] > [アプリケーション固有ライブラリパス]

また、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

IJServerのextディレクトリに設定する方法

IJServerのextディレクトリにjarファイルを格納することで、コンテナが動作するために必要なクラスパスより優先して使用することが可能です。
ただし、以下の点に注意してください。

- extディレクトリに複数のjarファイルが存在した場合、任意の順番でクラスパスに設定されます。
- extディレクトリのjarファイルをロードするには、IJServerを再起動してください。
- extディレクトリのjarファイルにコンテナが使用するクラスと同名のクラスが存在する場合、コンテナもそのクラスを使用して動作します。したがってコンテナが誤動作する可能性があるため、十分に動作検証を実施した上でこの機能を使用してください。

IJServerの実行に必要なクラスと、アプリケーションの実行に必要なクラスの干渉を避けるためにも、前記の“アプリケーション固有ライブラリパスに設定する方法”または、J2EEアプリケーション内にクラスを含めることを推奨します。後者については“アプリケーションのクラス”を参照してください。

環境変数:CLASSPATH

環境変数:CLASSPATH(自動起動時はシステム環境変数)は、クラスローダの分離に“分離しない”を設定している場合に、“システムクラスローダ”でロードされます。
クラスローダの分離が“EAR間で分離”または“すべて分離”の場合は環境変数:CLASSPATHに設定されているクラスはロードされません。

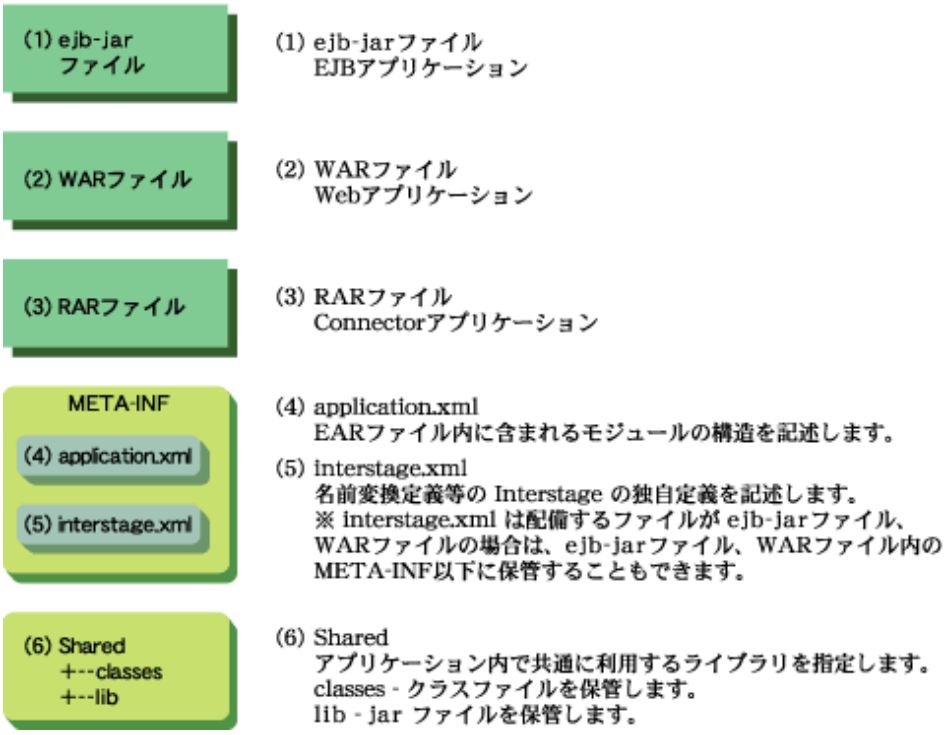
アプリケーションのクラス

アプリケーションのクラスは、EARファイル、ejb-jarファイル、WARファイル、RARファイルをInterstage管理コンソールを使用して、以下の方法で配備することで設定します。

- [ワークユニット] > “ワークユニット名” > [配備] > [配備ファイル]

EARファイルの構成

EARファイルは以下のようなファイル構成で構築します。



EJBアプリケーションのクラス

EJBアプリケーションのクラスは、以下のどちらかの方法で設定します。

- EARファイル内にejb-jarファイルを格納してIJServerに配備する
- ejb-jarファイルをIJServerに配備する

Webアプリケーションのクラス

Webアプリケーションのクラスは、以下のどちらかの方法で設定します。

- EARファイル内にWARファイルを格納してIJServerに配備する
- WARファイルをIJServerに配備する

Connectorのクラス

Connectorのクラスは、以下のいずれかの方法で設定します。

- EARファイル内にRARファイルを格納してIJServerに配備する
- RARファイルをIJServerに配備する
- RARファイルをConnectorサービスに配備する

Connectorサービスに配備されたConnectorは配備先をワークユニットのクラスパスに設定する必要があります。

アプリケーション内で共通のクラス

EJBアプリケーション、Webアプリケーションからユーティリティクラスのような共通に使用されるライブラリをIJServerに配備する方法を説明します。
アプリケーション内で共通のクラスを使用するには以下の2つの方法があり、クラスを参照できる範囲が異なります。

方法	クラスが参照可能な範囲
マニフェストクラスパスを設定する方法	マニフェストクラスパスが指定されたEJBアプリケーション(ejb-jar)、Webアプリケーション(WAR)内でのみ参照可能
EARファイル内のSharedディレクトリを使用する方法	アプリケーション内のすべてのクラスから参照可能

どちらの方法もユーティリティクラスのように共通に使用されるクラスをejb-jarファイル、WARファイルに含めることなくEJBアプリケーション、Webアプリケーションから使用することができるようになります。

マニフェストクラスパスを設定する方法

以下のようにEARファイル内にアプリケーション内で共通のクラスとマニフェストファイルを保管します。

1. ユーティリティクラスをEARファイルのトップまたは、EARファイル内の任意のディレクトリ内に保管する。
2. ユーティリティクラスを使用するejb-jarファイルまたは、WARファイル内のMETA-INF/MANIFEST.MF内に以下のエントリを記述する。*path*には、ユーティリティクラスが含まれるjarファイルまたは、クラスファイルが保管されたディレクトリをEARファイル内の相対パスで指定する。

```
Manifest-Version: 1.0
Class-Path: path
```

例

たとえば、web1.warからutility1.jarとutility2.jarおよび、com.fujitsu.Utility.classを利用する場合は以下のように定義したマニフェストファイルをweb1.warに含めます。

EARファイルの構成

- web1.war
- utility1.jar
- util/utility2.jar
- util/com/fujitsu/Utility.class

WARに含まれるマニフェストファイル

```
Manifest-Version: 1.0
Class-Path: utility1.jar util/utility2.jar util/
```

注意

マニフェストファイルを作成する際には下記の点に注意してください。

- ヘッダの':'の後には1つの半角空白が必要です。
- UTF-8エンコード形式ではすべての行を72バイト以内で記述する必要があります。72バイトを超える場合は次の行の先頭に1つの半角空白を記述して、その後に記述を継続してください。
- Class-Pathへの記載について、ディレクトリの場合は末尾に"/"(スラッシュ)を付加してください。

EARファイル内のSharedディレクトリを使用する方法

アプリケーション内で共通のクラスをEARファイル内のSharedディレクトリに保管してEARファイルをIJServerに配備することで設定します。
jarファイルの場合はShared/libディレクトリに、クラスの場合はShared/classesディレクトリに保管します。
EARファイル内のSharedディレクトリはInterstageの独自機能であり、他のアプリケーションサーバでは有効になりません。
また、SharedディレクトリはEARファイルに対してのみ有効であり、ejb-jarファイル、RARファイル、WARファイルに対しては有効になりません。
Shared/libとShared/classes内に同一のパッケージ名、クラス名のクラスが存在した場合は、Shared/classesディレクトリ内のクラスがロードされます。

2.3.5 XMLパーサの設定

ここでは、IJServerで使用するXMLパーサの設定方法について説明します。
Interstageでは以下のXMLパーサが使用可能です。

- Xerces2
- その他

IJServerごとに使用するXMLパーサを設定する

使用するXMLパーサは、Interstage管理コンソールを使用して、以下のどちらかの方法で設定します。

- [ワークユニット] > [新規作成] > [詳細設定] > [共通定義] > [使用するXMLパーサの種別]
- [ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [共通定義] > [使用するXMLパーサの種別]

また、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

設定する値は以下です。

- Xerces2 (デフォルト)
- その他
  この場合、XMLパーサのjarファイルが保管されているディレクトリをフルパスで指定します。指定したディレクトリ内にはorg.w3c.dom、org.xml.sax、javax.xml.parsers等のインタフェースとXMLパーサの実装クラスの両方を保管してください。どちらか一方のみの場合は、IJServerは起動できません。
  設定後にXMLパーサのjarファイルが削除されるなど設定内容に不備があった場合にはデフォルトのXMLパーサが使用されます。
  Solaris64 Linux32/64
  注) 指定するディレクトリはワークユニットの起動ユーザに対するRead権限が必要です。

注意

- XMLパーサは、指定した種別により以下のようにIJServer起動時のパラメータに設定されます。

XMLパーサの種別	パラメータに設定されるXMLパーサ
Xerces2	なし ※JDKに含まれるXMLパーサが使用される
その他	IJServer起動時にJava VMオプション:“-Djava.endorsed.dirs”に指定したディレクトリが設定される

- 以下にはXMLパーサのクラスは設定できません。設定した場合は無視されます。
  - J2EEプロパティ
  - ワークユニットのクラスパス
  - IJServerのSharedディレクトリ

アプリケーションごとに使用するXMLパーサを指定する

他のクラスと同じように以下の設定を行うとアプリケーションが個々に使用するXMLパーサを設定することができます。

- EARファイル内に含める
- ejb-jarファイル内に含める
- WARファイル内に含める

注意

- 使用するXMLパーサは、IJServerのXMLパーサの環境設定で指定したXMLパーサと以下のインタフェースのレベルを合わせる必要があります。インタフェースのレベルが合っていない場合は、XMLパーサが使用できない可能性があります。XMLパーサが使用できない場合は、ワークユニットの環境定義で選択したXMLパーサとアプリケーションが使用するXMLパーサのインタフェースのレベルが合っているか確認してください。インタフェースのレベルが合っていない場合は、アプリケーションが使用するXMLパーサのインタフェースのレベルをワークユニットの環境定義で選択したXMLパーサに合わせてください。

  ー JAXP(javax.xml.parsersパッケージ)

  ー SAX(org.xml.saxパッケージ)

  ー DOM(org.w3c.domパッケージ)

- IJServerのXMLパーサの環境設定で指定したXMLパーサと同一のXMLパーサは、指定しても無視され、XMLパーサの環境設定で指定したXMLパーサが使用されます。

- 以下のようにXMLパーサの種類を指定する場合は、IJServerのXMLパーサの環境設定で指定したXMLパーサを指定する必要があります。
  IJServerのXMLパーサの環境設定で指定したXMLパーサと違うXMLパーサを指定した場合は、IJServerの起動に失敗します。

  ー システムプロパティでの指定
  DOM：javax.xml.parsers.DocumentBuilderFactory
  SAX：javax.xml.parsers.SAXParserFactory

  ー 以下のjaxp.propertiesでの指定
  Windows32/64
  JDK使用時:C:¥Interstage¥JDK8¥jre¥lib¥jaxp.properties
  JRE使用時:C:¥Interstage¥JRE8¥lib¥jaxp.properties
  Solaris64 Linux32/64
  JDK使用時:/opt/FJSVawjbk/jdk8/jre/lib/jaxp.properties
  JRE使用時:/opt/FJSVawjbk/jre8/lib/jaxp.properties

2.3.6 トレース機能によるトラブル調査

J2EEアプリケーションを作成する際は、クラスがどのクラスローダでロードされるかを考慮する必要があります。
ロードするクラスローダを誤った場合、J2EEアプリケーションが正常に動作しない場合があります。このようなトラブル発生時の調査に役立てるために、トレース機能を提供しています。
トレース機能は、クラスがロードされた時にどのクラスローダ上でロードされたかを“コンテナログ”に出力します。“コンテナログ”では以下が確認できます。

- クラスがロードされた順番
- クラスがロードされたクラスローダ

出力形式

トレース情報の形式を以下に示します。

- クラスローダが“Webappクラスローダ”または“Applicationクラスローダ”の場合

  [タイムスタンプ] [Loaded クラス名 from リポジトリ by クラスローダの種別]

- クラスローダが“システムクラスローダ”、“Interstageクラスローダ”の場合

  [タイムスタンプ] [Loaded クラス名 by クラスローダの種別]

出力項目

<table>
<tr><th>項目</th><th colspan="2">内容</th></tr>
<tr><td>タイムスタンプ</td><td colspan="2">クラスをロードした日時</td></tr>
<tr><td>クラス名</td><td colspan="2">ロードしたクラスのクラス名(パッケージ名を含む)</td></tr>
<tr><td>リポジトリ</td><td colspan="2">ロードしたクラスの保管先ディレクトリ名またはjarファイル名</td></tr>
<tr><td rowspan="6">クラスローダの種別</td><td colspan="2">クラスをロードしたクラスローダの名称</td></tr>
<tr><td>クラスローダ</td><td>表示名</td></tr>
<tr><td>システムクラスローダ</td><td>System</td></tr>
<tr><td>Interstageクラスローダ</td><td>Interstage</td></tr>
<tr><td>Catalinaクラスローダ
注) Interstageのシステムで使用します</td><td>Catalina</td></tr>
<tr><td>Applicationクラスローダ</td><td>Application</td></tr>
<tr><td>Webappクラスローダ</td><td>Webapp</td></tr>
</table>

注意

システムクラスローダでロードされたクラスから使用されるクラスのトレースは出力されません。システムクラスローダでロードされるクラスは“2.3.1 クラスローダの構成”を参照してください。
システムクラスローダでロードされたクラスから使用されるクラスのトレースを出力したい場合はワークユニットのJavaVMオプションに-verbose:classを指定してJavaVMのトレース情報を取得してください。

ログの出力イメージ

```
[24/02/2004 16:17:22:093 +0900] [Loaded com.xxx.ClassA by Interstage]
[24/02/2004 16:17:22:101 +0900] [Loaded com.xxx.ClassB from c:¥Interstage¥J2EE¥lib¥xxx.jar by Webapp]
```

設定方法

トレースを出力するには、ワークユニットのJavaVMオプションに以下のオプションを設定します。

```
-Dcom.fujitsu.interstage.j2ee.ijserver.loader.trace=true
```

注意

トレース機能は、開発時のデバッグのための機能です。運用環境では使用しないことを推奨します。

使用例

ここではクラスローダのトレース機能の使用例について説明します。

契機

以下の問題が発生した場合にクラスローダのトレース情報を採取してください。クラスローダのトレース情報は問題解決のための手助けとなります。

- ServletからEJBアプリケーションの呼び出しに失敗。コンテナログの解析の結果、ClassAでClassCastExceptionが発生していることを検出。

解析手順

以下の手順で解析を行います。

1. ワークユニットのJavaVMオプションに以下のオプションを設定。
   -Dcom.fujitsu.interstage.j2ee.ijserver.loader.trace=true
2. 再現テストを実施。
3. コンテナログに出力されたトレース情報をClassAで検索し、以下の2行を検出。

```
Loaded ClassA C:¥Interstage¥J2EE¥var¥deployment¥ijserver¥kaz001¥apps
¥j2eesample.ear¥CartBean.jar by Application
Loaded ClassA C:¥Interstage¥J2EE¥var¥deployment¥ijserver¥kaz001¥apps
¥j2eesample.ear¥j2eesample.war¥WEB-INF¥classes by Webapp
```

上記からClassAがEJBアプリケーションとWebアプリケーションの2箇所に保管されていることが判明。

検討

クラスのバッティングが原因でServletからEJBアプリケーションの呼び出しに失敗しているため、以下の対応を検討してください。

1. EJBアプリケーションとWebアプリケーションの両方に同じクラスが存在しないようにアプリケーションの構成を変更することができないか。
2. クラスローダの検索順番を“親が先”に設定することで回避可能か。
3. クラスローダの分離を“分離しない”に設定することで回避可能か。

2.3.7 クラスローダ使用時の注意事項

ここではクラスローダを使用する際の注意事項について説明します。

JDBCドライバを使用する際の注意事項

JDBCドライバは“Interstageクラスローダ”でロードする必要があります。したがってJDBCドライバは以下の場所に定義してください。

- J2EEプロパティのクラスパス
- ワークユニットのクラスパス
- IJServerのSharedディレクトリ

定義に誤りがある場合、なんらかのエラーや例外(Exception)が発生し、JDBCドライバを使用できません。

Webアプリケーションのディレクトリ構成である“WEB-INF/lib”に、クラスパスに設定するJDBCドライバを格納しないでください。
格納した場合、以下の機能が使用できない場合があります。

- 事前コネクト
- 異常時の再接続

Connectorを使用する際の注意事項

Connectorを使用する際はRARファイル内に共有ライブラリが含まれている場合があります。
その場合は、RARファイルの配備先ディレクトリをワークユニット環境設定に設定する必要があります。
Interstage管理コンソールで、[ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [ワークユニット設定]の以下の項目にRARファイルの配備先ディレクトリを設定してください。

Windows32/64

パス

Solaris64 Linux32/64

ライブラリパス

J2EEアプリケーションでJNIを使用する場合の注意事項

同じNativeモジュールは別々のクラスローダでロードすることはできません。JNIを使用するクラスがアプリケーション(EAR、ejb-jar、WAR、RAR)に含まれている場合、同じNativeモジュールを別々のクラスローダでロードした場合、java.lang.UnsatisfiedLinkErrorがスローされます。
java.lang.UnsatisfiedLinkErrorがスローされた場合は以下の対処を行ってください。

- JNIを使用するクラス(nativeメソッドを実装しているクラス)をアプリケーションに含めずに、JNIを使用するクラスをワークユニットのクラスパスに設定、または、IJServerのSharedディレクトリに保管してください。
- JNIを使用するクラスがEJBアプリケーションのクラスの場合は、クラスローダの分離を“分離しない”にすることで回避可能です。

また、JNIを使用するクラスがアプリケーション(EAR、ejb-jar、WAR、RAR)に含まれている場合、そのアプリケーションはHotDeploy／クラスのオートリロードを使用することはできません。
使用した場合は、java.lang.UnsatisfiedLinkErrorがスローされ、HotDeploy／クラスのオートリロードに失敗する可能性があります。java.lang.UnsatisfiedLinkErrorがスローされ、HotDeploy／クラスのオートリロードに失敗した場合は、IJServerを再起動してください。
JNIを使用するアプリケーションでHotDeploy／クラスのオートリロードを使用する場合は、以下の対処を行う必要があります。

- JNIを使用するクラス(nativeメソッドを実装しているクラス)をアプリケーションに含めずに、JNIを使用するクラスをワークユニットのクラスパスに設定、または、IJServerのSharedディレクトリに保管してください。

なお、JNIを使用するクラスを入れ替える場合はIJServerを再起動する必要があります。

2.4 トランザクション制御

ここでは、IJServerに配備されたJ2EEアプリケーションのトランザクション制御について説明します。

トランザクションの制御方法

Webアプリケーションでトランザクションを制御する場合は、JNDIからUserTransactionを取得して制御します。

EJBアプリケーションでトランザクション制御する場合は、トランザクション種別にBean(Bean Managed Transaction)を指定してUserTransactionで制御するか、またはトランザクション種別にContainer(Container Managed Transaction)を指定して、コンテナがトランザクションを制御するかを選択できます。

UserTransactionの使用方法については、“4.12 UserTransactionインタフェースを使用したトランザクション制御”を参照してください。
EJBのトランザクション制御については、“10.4 EJBサービスのトランザクション制御”を参照してください。

トランザクションは、デフォルトのトランザクションと分散トランザクションを選択できます。

デフォルトのトランザクション

同一Java VM上のJ2EEアプリケーションは、デフォルトのトランザクションを使用したトランザクション連携によってデータベースにアクセスできます。
トランザクションを開始してから、そのトランザクション内で同一のデータソースに対して、同一のユーザID/パスワードで接続した場合、1つのコネクションを共有して使用できるため、トランザクション連携が可能になります。

また、J2EEアプリケーションでトランザクションを開始し、同一Java VM上のJ2EEアプリケーションにアクセスする場合、同一のトランザクションで動作させることができます。
EJBアプリケーションをトランザクション連携させる場合には、トランザクション種別に“Container”を指定し、トランザクションが連携可能なトランザクション属性(“Required”など)を指定してください。

分散トランザクション Windows32/64 Linux32/64

以下のトランザクション連携を行う場合には、分散トランザクションを使用してください。

- 異なるリソースへのアクセスでトランザクション連携する場合
- 異なるIJServer間でトランザクション連携する場合
- J2EEアプリケーションクライアントからトランザクションの開始と終了を制御する場合

注意

- 分散トランザクションはEJBアプリケーションだけで使用できるため、IJServerの運用タイプは以下のいずれかにしてください。
  - EJBアプリケーションのみ運用でのEJBアプリケーションを運用するプロセス
  - WebアプリケーションとEJBアプリケーションを運用(別Java VM)

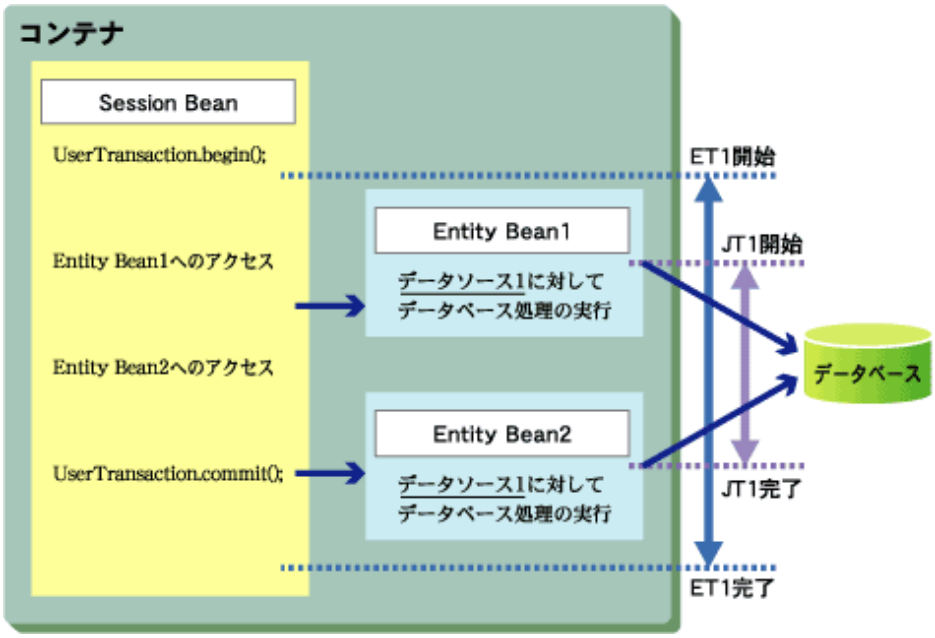
  分散トランザクションの指定はIJServerの定義時に指定します。指定方法についてはInterstage管理コンソールのヘルプを参照してください。
  分散トランザクションの詳細については“第5部 JTS/JTA編 ”を参照してください。
- 異なるサーバマシンで連携しているEJBアプリケーション間では、分散トランザクションを使用できません。
- 分散トランザクション機能を使用し、Entity Beanメソッドのいずれかにトランザクション属性 NotSupported、Supports、Never が指定されている場合は、対象EJBアプリケーション起動時にエラーが発生して起動に失敗します。
- IJServerのプロセスを多重で起動できません。
- 使用できるリソースマネージャは、最大32個までです。

EJBアプリケーションのトランザクション

EJBアプリケーションのトランザクション機能では、トランザクション内でJDBCコネクションをキャッシュしています。このため、分散したアプリケーションを1つのトランザクションで制御できます。

トランザクション内でのJDBCコネクションのキャッシュ

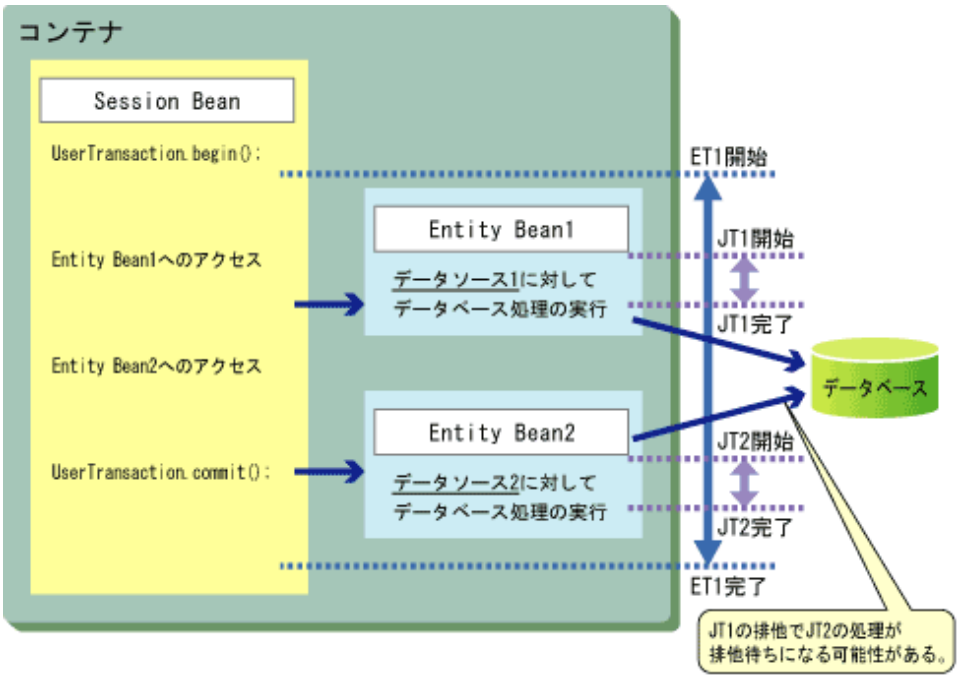
通常のJDBCアプリケーションでは、データソースのgetConnectionメソッドで取得したコネクションごとにトランザクションが管理されます。
EJBアプリケーションの場合には、getConnectionメソッドで取得したコネクションを、コンテナがトランザクション内でキャッシュします。同一のトランザクション内で同一のデータソースに対して、再度getConnectionメソッドが実行した場合、コンテナがキャッシュされたコネクションを返却するため、以下の図のように分散したアプリケーションの処理も1つのトランザクションで制御できます。



EJBアプリケーションのトランザクション(ET1)を開始してから、getConnectionメソッドを複数回実行していますが、同じデータソースを使用しているため、Entity Bean1とEntity Bean2は同一コネクションで処理します。
このため、Entity Bean1とEntity Bean2のそれぞれのデータベース処理を、同じトランザクション(JT1)で処理できます。

注意

異なるデータソースから取得したコネクションに対する処理は、別々のトランザクションで処理します。それぞれのデータソースが同じデータベースを対象とする場合も、別々のトランザクションで処理します。このため、以下の図のようにアプリケーションを構築すると処理が停止する場合があります。



トランザクション連携可能なリソース

デフォルトのトランザクションと分散トランザクションで制御できるリソースは以下です。

デフォルトのトランザクションの場合

以下のリソースをトランザクション制御可能です。

- JDBCデータソース

また、Message-driven Beanがメッセージを受信するJMSコネクションファクトリについては、Message-driven Beanのトランザクション種別にContainerを指定して、トランザクション種別にRequiredを指定することで、コンテナがトランザクションを制御します。

分散トランザクションの場合 Windows32/64 Linux32/64

以下のリソースをトランザクション制御可能です。

- JDBCデータソース
- JMSコネクションファクトリ
- connectorコネクションファクトリ

分散トランザクションを使用する場合には、あらかじめ分散トランザクション連携可能なリソースを定義しておく必要があります。

第3章 J2EEアプリケーションの運用

Interstageでは、InterstageやIJServerの起動/停止(運用操作)、環境設定を行うためのInterstage管理コンソールを提供しています。
8.0以降では、複数のサーバに同一のIJServerを作成するような、大規模システムの構築をサポートするisj2eeadminコマンドも提供しています。

Interstage管理コンソールについて詳細は、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドについて詳細は、“リファレンスマニュアル(コマンド編)”の“J2EE運用コマンド”を参照してください。

ここでは、J2EEアプリケーション運用のために必要な以下の作業について説明します。

- J2EEアプリケーションの準備
- IJServerの作成
- J2EEアプリケーションの配備と設定
- Servletサービスの運用準備
- Webサーバコネクタにおけるリクエストの振り分け制御
- JTSを利用する場合の手順
- JMSを利用する場合の手順
- JavaMailを利用する場合の手順
- 動作環境のカスタマイズと確認
- アプリケーションのデバッグ

以下の機能を利用する場合については、別章で説明していますので、そちらを参照してください。

- JNDIを利用したリソース定義・参照機能については、“JNDI”
- セキュリティ機能については、“J2EEアプリケーションのセキュリティ”

ポイント

- JVMのバージョンの違いによる非互換等の問題を防ぐため、開発/配備/運用で使用するJDK/JREのバージョンは一致させることを推奨します。
- Windows32/64
  IJServer上に配備されたアプリケーションから、ネットワーク上の共有資源にアクセスする場合は、“2.2.5 ネットワーク上の共有資源へアクセスする場合の環境設定 ”を参照してください。

3.1 J2EEアプリケーションの準備

ここでは、J2EEアプリケーションの準備について説明します。

J2EEアプリケーションの開発

J2EEアプリケーションを開発します。
また、開発の詳細については、以下を参照してください。

- リソースおよびEJBの参照については、“4.10 オブジェクトの参照方法”
- Webアプリケーションについては、“第7章 Webアプリケーションの開発”

- EJBアプリケーションについては、“第11章 EJBアプリケーションの開発”
  また各実行環境における開発の詳細については、“第12章 Session Beanの実装”、“第13章 Entity Beanの実装”、“第14章 Message-driven Beanの実装”、“第15章 EJBアプリケーションの呼出し方法”
- Webサービスアプリケーションについては、“第18章 Webサービスの開発”

deployment descriptorの設定

J2EEアプリケーションクライアント、Webアプリケーション、Webサービスアプリケーションの場合、deployment descriptorを設定します。deployment descriptorでは、J2EEアプリケーション自身の情報やJ2EEアプリケーションが参照するリソース名等の設定を行います。詳細は、以下を参照してください。

- J2EEアプリケーションクライアントについては、“4.13 J2EEアプリケーションクライアントのdeployment descriptorファイルの詳細設定”
- Webアプリケーションについては、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”
- EJBアプリケーションは、Interstage StudioのXMLエディタを使用します。詳細は、“Interstage Studio ユーザーズガイド”を参照してください。
- Webサービスアプリケーションについては、“18.1.3 deployment descriptorを編集する”

クラスファイルのパッケージ化

プログラムとして作成したクラスファイルをパッケージ化します。

- J2EEアプリケーションクライアントは、jarファイルに格納しパッケージ化します。
- Webアプリケーションは、WARファイルに格納しパッケージ化します。
- EJBアプリケーションは、jarファイルに格納しパッケージ化します。

アプリケーションごとに作成された上記のパッケージを、さらにEnterprise ARchive (EAR)ファイルとしてパッケージ化することができます。それによって運用で使用するアプリケーションすべてを1つのパッケージとして流通できるようになります。

3.2 IJServerの作成

IJServerの作成は、以下のどちらかの方法で行います。

- Interstage管理コンソールを使用する
  [ワークユニット] > [新規作成]で作成します。
- コマンドを使用する
  J2EE運用コマンド“isj2eeadmin”を使用して、作成します。“isj2eeadmin”について詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

3.3 ワークユニットの設計

ワークユニットはアプリケーションの運用の単位となります。そのため、1つのワークユニットには、同時に起動、停止を行うアプリケーションを設定します。
ワークユニットは運用の単位となるため、ワークユニット内の1つのアプリケーションが起因してワークユニット異常終了となった場合でも、ワークユニット内のすべてのアプリケーションも停止となります。任意のアプリケーションが原因で、他のアプリケーションに影響を及ぼしたくない場合は、ワークユニットを分けることも、考慮が必要です。

1つのワークユニットには、1つのIJServerが対応しています。
IJServerの詳細については、“2.2 J2EEアプリケーションが運用される環境(IJServer)”を参照してください。

3.3.1 アプリケーションプロセス多重度

複数のクライアントから、1つのアプリケーションに同時に要求が出された場合に、実行単位(プロセス)を複数にして、同時に処理できるプロセス多重度を設定することができます。

IJServerワークユニットの場合

プロセス多重度は、以下により決定します。

- 1処理あたりの処理時間
- クライアントへのレスポンス時間
- 時間あたりの要求数
- 1プロセスあたりのスレッド数

なお、必要以上にプロセス多重度を大きくすると、メモリなどシステム資源に影響があるため、妥当な多重度を設定する必要があります。
プロセス多重度は、Interstage管理コンソールのワークユニット設定で定義します。

3.3.2 アプリケーション自動再起動

クライアントからの入力データの誤りなどにより、アプリケーションが異常終了した場合に、アプリケーションを自動的に再起動させることができます。自動再起動を設定すると、クライアントからの新たな要求を処理することができます。

自動再起動の設定では、連続異常終了回数を設定します。連続異常終了回数とは、アプリケーションの障害などにより、該当アプリケーションが1度も正常に処理されず、異常終了と再起動が繰り返される回数です。連続して異常終了した回数に達した場合、ワークユニットは異常終了します。ワークユニットが異常終了した場合、ワークユニット内のアプリケーションのプロセスはすべて停止され、該当ワークユニットで処理中の要求はクライアントへエラーで復帰します。
連続終了回数まで1度でも該当アプリケーションの処理が正しく動作した場合(アプリケーションが復帰)、終了回数はリセットされます。

なお、異常終了回数に1が設定された場合は、再起動は行われません。0を設定した場合は、無限に再起動を実施します。
また、プロセスの再起動が失敗した場合は、ワークユニットは異常終了となり、同一ワークユニット内で動作しているすべてのアプリケーションが停止します。

プロセス再起動失敗時の縮退運用

プロセスの再起動が失敗した場合は、ワークユニットは異常終了となりますが、アプリケーションが複数多重で起動されているワークユニットでは、これを回避し、プロセスの再起動に失敗した場合でも、プロセス再起動失敗のメッセージを出力し、残りのプロセスでのワークユニット運用を継続する機能を提供します。詳細は、「3.3.12 ワークユニットのアプリケーション自動再起動失敗時の縮退運用」を参照してください。

IJServerワークユニットの場合

アプリケーション自動再起動は、Interstage管理コンソールのワークユニット設定に定義します。

自動再起動の設定では、Webアプリケーションの場合は、一定時間内での異常終了回数を設定します。アプリケーションの異常終了が異常終了回数に達した場合、ワークユニットは異常終了します。ワークユニットが異常終了した場合、ワークユニット内のアプリケーションのプロセスはすべて停止され、該当ワークユニットで処理中の要求はクライアントへエラーで復帰します。
一定時間内に異常終了回数まで達しなかった場合、異常終了回数はリセットされます。

3.3.3 サーバアプリケーションタイマ機能

アプリケーションの最大処理時間(タイムアウト時間)を指定し、ワークユニット配下で動作するサーバアプリケーションの呼び出しから復帰までの、応答時間の監視を行うことができます。アプリケーション障害によるハングアップ、ループなどが原因となる

処理遅延によるクライアントへのレスポンス遅延を防止する場合に使用します。
なお、タイマ機能はサーバアプリケーションの障害などにより、アプリケーションの復帰が遅延する場合の対処として使用してください。一般的な運用で頻繁にタイムアウトが発生するような使用は避けてください。

IJServerワークユニットの場合

タイムアウト時の振る舞いとして以下の形態を選ぶことができます。どちらの形態でも、タイムアウト発生時にスレッドダンプを自動的に採取します。

- タイムアウト時間超過メッセージを出力し、タイムアウトとなった処理が属するサーバアプリケーションプロセスを強制的に停止します。プロセス強制停止後、クライアントにエラー/例外が復帰します。
  該当プロセス上の複数スレッドで同時に処理が実行されていることを考慮し、プロセスを強制停止させても問題がない場合に、本形態を選択してください。
- タイムアウト時間超過メッセージのみ出力し、サーバアプリケーションプロセスは停止しません。ただし、本形態は、メッセージ出力後、正常にアプリケーションが復帰することがありますので、使用時には注意が必要となります。なお、最初のタイムアウト発生から10分間は、同一プロセスからのタイムアウト時間超過メッセージの出力は抑止されます。

スレッドダンプはコンテナ情報ログ(info.log)に採取されます。また、タイムアウト発生直後と10秒後の2回出力されます。これにより、2回のスレッドダンプで変化がなかったスレッド上の動作アプリケーションで問題があることが検出できます。
アプリケーションの最大処理時間は、Interstage管理コンソールのワークユニット設定に定義します。

注意

プロセス強制停止処理は、スレッドダンプの出力前にプロセスが強制停止されることを防止するために、2回目のスレッドダンプ出力処理の10秒後に実施します。それにより、タイムアウト時間超過メッセージが出力されてから、10秒後に2回目のスレッドダンプが出力され、さらに10秒後にプロセスが強制停止されるため、少なくとも20秒間プロセスは停止しません。そのため、メッセージが出力された場合でも、プロセス停止までの間に正常にアプリケーションが復帰し、その後にプロセスが強制停止される場合があります。

3.3.4 カレントディレクトリ

ワークユニットで起動したアプリケーションが動作する作業ディレクトリ(カレントディレクトリ)を指定することができます。
カレントディレクトリにより、ワークユニット配下で動作するアプリケーションはそれぞれ異なった作業ディレクトリで動作することが可能となります。

IJServerワークユニットの場合

アプリケーションプロセスのカレントディレクトリは、Interstage管理コンソールの[システム] > [ワークユニット] > [ワークユニット名] > [環境設定]画面の[ワークユニット設定]の[カレントディレクトリ]で指定することができます。
なお、デフォルトでは指定されたディレクトリ配下に以下の形式でディレクトリが作成され、プロセスごとに異なるディレクトリ上で動作します。

Windows32/64

[指定ディレクトリ]¥[ワークユニット名]¥current¥[ワークユニット名]¥[プロセスID]

Solaris64 Linux32/64

[指定ディレクトリ]/[ワークユニット名]/current/[ワークユニット名]/[プロセスID]

[指定ディレクトリ] : Interstage管理コンソールの[システム] > [ワークユニット] > [ワークユニット名] > [環境設定]画面の[ワークユニット設定]の[カレントディレクトリ]で指定されているディレクトリ
[ワークユニット名] :対象のワークユニット名
[プロセスID] :アプリケーションプロセスのプロセスID

また、上記のフォルダを作成せずに、ワークユニット単位に全てのプロセスが指定されたフォルダをカレントディレクトリとすることもできます。その場合は、Interstage管理コンソールの[システム] > [ワークユニット] > [ワークユニット名] > [環境設定]画面の[ワークユニット設定]の[カレントディレクトリ]で、[IJServerで一意のカレントディレクトリとする]を指定してください。この場合、カレントディレクトリの世代管理も実施しません。

Solaris64 Linux32/64

Solaris/Linuxの場合、アプリケーションが異常終了しコアファイルを出力した場合、それぞれのカレントディレクトリ配下に出力されます。

カレントディレクトリの世代管理機能

ワークユニットのカレントディレクトリのバックアップを0～5世代まで任意の世代数残すことができます。
バックアップはワークユニットの起動時に作成され、管理コンソールの[システム] > [ワークユニット] > [ワークユニット名] > [環境設定] 画面の[ワークユニット設定]の[退避するカレントディレクトリの世代数]で指定された世代数まで残ります。前回起動時に作成されたワークユニット名のディレクトリが、「ワークユニット名.old1」としてバックアップされます。「ワークユニット名.old1」は「ワークユニット名.old2」としてバックアップされ、指定された世代数「n」の「ワークユニット名.oldn」までバックアップが作成されます。「ワークユニット名.oldn」が既に存在する場合は、「ワークユニット名.oldn」を削除し、一つ前の数字のディレクトリが「ワークユニット名.oldn」として残されます。

注意

- バックアップするカレントディレクトリの世代数の省略値は1です。そのため、ワークユニットのカレントディレクトリで指定したディレクトリ配下には、必ず前回起動時に作成されたディレクトリとファイルが残ります。
- IJServerワークユニットのカレントディレクトリの詳細については、“IJServerのカレントディレクトリ”を参照してください。
- カレントディレクトリに指定するディスクの容量は、指定された世代数に合わせて十分な容量を見積もってください。

3.3.5 環境変数

IJServerワークユニットの場合

ワークユニット配下で動作するアプリケーションが使用する環境変数を設定することが可能です。アプリケーションプロセス上に反映したい環境変数は、Interstage管理コンソールのワークユニット設定で定義可能です。アプリケーションで使用しているデータベース処理などが環境変数を使用する場合などに使用してください。

- ワークユニット配下で動作するアプリケーションプロセスの環境変数は、Interstage管理コンソールのワークユニット設定に記載された環境変数およびシステム環境変数の両方が有効となります。
  なお、Interstage実行環境で設定されている環境変数は、Windowsでは、システム環境変数が引き継がれ、SolarisおよびLinuxでは、Interstageの起動方法により以下のとおり異なります。

  ー isstartコマンドを使用してInterstageを起動する場合
  isstartコマンド実行環境で設定されている環境変数が引き継がれます。なお、システム初期化スクリプト(S99startis)またはunitファイル(FJSVtd_start.service)を使用して、サーバ起動時にInterstageを自動起動する運用となっている場合は、システム初期化スクリプトまたはunitファイルの実行環境の環境変数が引き継がれます。

  ー Interstage管理コンソールよりInterstageを起動する場合
  Interstage JMXサービスの実行環境で設定されている環境変数が、Interstageに引き継がれます。Interstage JMXサービスはisjmxstartコマンドを使用して起動します。そのため、isjmxstartコマンド実行環境で設定されている環境変数が引き継がれます。なお、システム初期化スクリプト(S95isjmxstart)またはunitファイル(FJSVisjmx_start.service)を使用して、サーバ起動時に自動起動する運用となっている場合は、システム初期化スクリプトまたはunitファイルの実行環境の環境変数が引き継がれます。

- Interstage管理コンソールのワークユニット設定の環境変数とシステムの環境変数が重複している場合には、Interstage管理コンソールのワークユニット設定の環境変数が優先されます。

有効となる設定値

以下に環境変数ごとに有効となる設定値を説明します。

PATH、LD_LIBRARY_PATHおよびCLASSPATH以外の環境変数

Interstage管理コンソールのワークユニット設定の環境変数に記載された環境変数と、Interstage実行環境に設定されている環境変数のどちらも有効です。
ただし、環境変数名が重複している場合はInterstage管理コンソールのワークユニット設定で設定した環境変数に置き換えられます。

LD_LIBRARY_PATH Solaris64 Linux32/64

Interstage管理コンソールのワークユニット設定のライブラリパスの値が、Interstage実行環境で設定されているLD_LIBRARY_PATHの値の前に設定されます。
そのため、どちらも有効となりますが、Interstage管理コンソールのワークユニット設定値が優先されます。

PATH

Interstage管理コンソールのワークユニット設定のパスの値が、Interstage実行環境で設定したPATHの値の前に設定されます。
そのため、どちらも有効となりますが、Interstage管理コンソールのワークユニット設定値が優先されます。

CLASSPATH

Interstage管理コンソールのワークユニット設定のクラスパスの値が、Interstage実行環境で設定されているCLASSPATHの値の前に設定されます。
そのため、どちらも有効となりますが、Interstage管理コンソールのワークユニット設定値が優先されます。

3.3.6 キュー制御

複数クライアントから同一アプリケーション(オブジェクト)に対して要求が出された場合に、すべての要求分のアプリケーションを起動してしまうと、サーバの負荷が大きくなってしまいます。
このような場合に、キュー制御で負荷を平準化させることができます。
クライアントからの要求は、該当するアプリケーションの待ち行列であるキューに接続(キューイング)され、キューイングされた要求は、アプリケーションにより順に処理されます。

キューの単位はアプリケーション種別により異なります。
キュー制御の詳細は、「OLTPサーバ運用ガイド」の「ワークユニットの機能」を参照してください。

3.3.7 キュー閉塞/閉塞解除

キューの閉塞、閉塞解除を行い、クライアントからの要求を一時的に受け付けないような状態にしたり、要求の受け付けを再開したりすることができます。キューの閉塞、閉塞解除は以下のような場合に有効です。

- 時間帯により使用を制限したい業務がある場合
- 負荷が高く、一時的に要求を受け付けない状態にしたい場合
- 業務停止前に要求を抑止し、すべての処理が終わったのち停止を行う場合

なお、キューの閉塞中にクライアントからの要求を行った場合、クライアントへ異常が復帰します。

キュー閉塞/閉塞解除の詳細については、「OLTPサーバ運用ガイド」の「ワークユニットの機能」を参照してください。

3.3.8 最大キューイング機能

キューイングされる要求の最大数が指定できます。これにより、サーバアプリケーションに一定以上の負荷が掛かった場合に、クライアントからの要求を制限することができます。
クライアントからの要求が指定された最大キューイング数を超えた場合、クライアントへ以下の例外が通知されます。

```
システム例外:NO_RESOURCES
マイナーコード(16進表記(10進表記)):0x464a0894(1179256980)
```

なお、サーバアプリケーションが現在処理中の要求は、キューイングされている数には含まれません。サーバアプリケーション上で処理中の要求を除いた、処理待ちの要求の最大数が指定できます。

IJServerワークユニットの場合

最大キューイング機能については、Interstage管理コンソールのワークユニット設定で設定します。
クライアントへ通知されるエラーの詳細については、「メッセージ集」の「CORBAサービスから通知される例外情報/マイナーコード」を参照してください。

3.3.9 滞留キュー数のアラーム通知機能

処理状況に応じて、処理が終わらず滞留しているキュー数が、設定した監視キューイング数を超えるとアラーム通知を行います。また、設定した監視再開キューイング数まで滞留キュー数が減少すると、監視を再開します。
これにより、システムの負荷状態をリアルタイムに監視することができます。
滞留キュー数を監視する場合、以下の3つのポイントを監視できます。

- 滞留キュー数が最大キューイング数を超過した場合
- 滞留キュー数が監視キューイング数(任意)と同じになった場合
- 滞留キュー数が監視再開キューイング数(任意)と同じになった場合

IJServerワークユニットの場合

上記ポイントは、Interstage管理コンソールのワークユニット設定で定義します。
滞留キュー数の処理監視については、Systemwalker CentricMGRと連携することで、集中監視を行うことが可能となります。
監視対象としたキューの滞留キュー数が、上記のポイントに達した時、Systemwalker CentricMGR 運用管理クライアントの画面上に、状態の遷移が表示されます。
これにより、滞留キュー数の増減に対してリアルタイムな監視が可能となります。
Systemwalker CentricMGRとの連携については、「Systemwalkerとの連携」を参照してください。

また、Systemwalker CentricMGRとの連携を行わない場合でも、それぞれの事象になった場合、メッセージを出力します。それぞれのメッセージを監視することで、滞留キュー数の状態を把握することでできます。

以下にそれぞれの事象ごとに出力されるメッセージを示します。メッセージの詳細については、「メッセージ集」を参照してください。

事象	出力メッセージ
滞留キュー数が最大キューイング数を超過した場合	od11108: 待ちメッセージ数が最大キューイング数を超えました
滞留キュー数が監視キューイング数(任意)と同じになった場合	od11107: 待ちメッセージ数が監視キューイング数となりました
滞留キュー数が監視再開キューイング数(任意)と同じになった場合	od11109: 待ちメッセージ数が監視再開キューイング数となりました

3.3.10 バッファ制御

クライアントからの要求データは、キューイングされているあいだは、一時的に共用メモリ上に保持しつづけます。

この共用メモリ上のデータ域を通信バッファといい、バッファ域を管理する制御をバッファ制御といいます。通信バッファは1つの要求データを埋め込む領域であるバッファ域が複数個で構成されます。なお、バッファ域の大きさをバッファ長、バッファ域の数をバッファ数といいます。

IJServerワークユニットの場合

このバッファ長、バッファ数については、Interstage管理コンソールのワークユニット設定で設定します。
1つの要求データを複写するバッファ域の大きさは、一定の長さ(4096バイト)用意しており、要求データを複写してキューイングします。このとき要求データ長が一定の長さ以内であれば、データがバッファ域に収まりきるため、そのままバッファ域に複写し、キューイングします(以下の注意参照)。
Interstage管理コンソールのワークユニット設定でバッファのチューニングを行った場合、バッファ域は、IJServer単位で作成され、バッファのチューニングを行っていない場合は、デフォルトバッファを共用します。

要求データは、バッファ域内に収まりきり、キューイングすることが性能的にも良い処理といえます。そのため、要求データ長が一定の長さ(4096バイト)を超える要求が多く存在する業務においては、バッファ長とバッファ数を使用者がチューニングできます。
ただし、バッファ長とバッファ数を大きくしすぎると多くの共用メモリを使用する必要があり、非効率です。そのため、業務ごとに最適なバッファ長、バッファ数を設定する必要があります。

Interstage管理コンソールのワークユニット設定でバッファ数をチューニングしない場合、デフォルトバッファ域を使用します。デフォルトバッファ域は、Interstage管理コンソールのワークユニット設定でバッファ数、バッファ長をチューニングしているIJServerのEJBアプリケーション、EJBアプリケーション、CORBAアプリケーションを除く、CORBA通信で共用されます。デフォルトバッファは、通常チューニングする必要ありません。しかし、od11110またはod11111が出力された場合は、デフォルトバッファのバッファ数が不足していることが考えられるため、CORBAサービスの動作環境ファイル(config)のnumber_of_common_bufferパラメタを修正することで、チューニングが可能です。

注意

要求データ長が固定長を超える場合は、サーバアプリケーション側で処理実行時に、残りデータが受信されます。クライアントがマルチスレッド構成で、同時に複数の要求を送信した場合、1つ目の要求データが読み込まれるまで、次の要求が処理されません。また、次以降の要求データはネットワーク上に滞留するため、要求データの量が多い場合は、クライアント側でソケットへの書き込みでタイムアウトが発生しod10965メッセージが出力される場合があります。

バッファ長

一つの要求データには、実際に使用者が設定したデータ部と、Interstageが使用する制御データ部が含まれます。
性能監視ツールで採取されるオペレーションごとのデータ長を参考にバッファ長の妥当性の確認を行うことが可能です。

バッファ数

バッファ数はクライアントからの同時要求数により求める必要があります。
クライアントからの要求が同時にキューにキューイングされる数分必要です。同時にキューイングされる数については、アプリケーションの多重度とアプリケーションの処理時間と同時にクライアントからくる要求数より求めてください。
バッファ数に関しては、isinfobjコマンドの滞留キュー数の結果や、性能監視ツールで採取される処理待ち要求数を参考に、バッファ数の妥当性の確認を行うことが可能です。性能監視ツールについては、「性能監視」を参照してください。

3.3.11 予兆監視

予兆監視として、以下の機能を提供します。

予兆監視警告メッセージ (Javaヒープ領域)

JavaVMのメモリ割り当てプールおよびメタスペースを監視して、Javaヒープ領域不足の危険性を警告メッセージ(EXTP4368/IJServer30004)で通知する機能を提供します。

参考

Javaヒープ領域は、New世代領域とOld世代領域に大別されます。New世代領域とOld世代領域をメモリ割り当てプールと呼びます。JDK/JRE 7以前でJavaヒープに存在していたPermanent世代領域は、JDK/JRE 8では廃止され、Permanent世代領域の代替として、メタスペースが導入されました。以下の説明で、単に「ヒープ領域」と記載している場合は、メモリ割り当てプールを指します。Javaヒープの構造については、「チューニングガイド」の「JDK/JRE 8のチューニング」の「基礎知識」を参照してください。

予兆監視警告メッセージ (ガーベジコレクション)

JavaVMのガーベジコレクション処理の影響で業務レスポンス低下が発生する可能性を検出し、警告メッセージ(EXTP4368/IJServer30004)で通知する機能を提供します。

管理コンソールモニタ機能

JavaVMの性能情報をInterstage管理コンソールのモニタ機能で参照できます。
警告メッセージはイベントログに出力されます。

予兆監視のチューニング

デフォルトでは予兆監視警告メッセージ(EXTP4368)はシステムログ/イベントログに出力されません。予兆監視警告メッセージ(EXTP4368)を出力させるには、ワークユニットのJavaVMオプションの設定が必要です。

注意

- JavaVMのヒープ領域使用量、および、メタスペース使用量が、短時間、かつ、急激に増加した場合、予兆監視の警告メッセージが出力されない場合があります。この場合、イベントログに、メモリ不足を示すメッセージが出力されていますので、そのメッセージに従い対処を行ってください。
- Interstage Application Serverが提供するJavaVM以外を使用した場合、予兆監視、および、Interstage管理コンソールのモニタ画面におけるJavaVMのメタスペース情報やガーベジコレクション情報は利用できません。また、JavaVMのヒープ情報(Kbyte)の最小、最大、上限は、状態の再取得時における最小、最大、上限となります。
- Interstage Application Serverが提供するJavaVM以外を使用した場合、Interstage管理コンソールのモニタ画面で、以下の情報が常に0と表示されます。
  - JavaVMのメタスペース情報(byte)の現在、最小、最大
  - ガーベジコレクション情報の発生回数、処理トータル時間(msec)、平均ガーベジコレクション発生間隔(msec)
- ガーベジコレクション情報は、Full GCの情報を使用しています。

予兆監視警告メッセージ (Javaヒープ領域)

JavaVMのメモリ割り当てプールおよびメタスペースを監視して、Javaヒープ領域不足の危険性を警告メッセージ(EXTP4368/IJServer30004)で通知します。

参考

Javaヒープ領域は、New世代領域とOld世代領域に大別されます。New世代領域とOld世代領域をメモリ割り当てプールと呼びます。JDK/JRE 7以前でJavaヒープに存在していたPermanent世代領域は、JDK/JRE 8では廃止され、Permanent世代領域の代替として、メタスペースが導入されました。以下の説明で、単に「ヒープ領域」と記載している場合は、メモリ割り当てプールを指します。Javaヒープの構造については、「チューニングガイド」の「JDK/JRE 8のチューニング」の「基礎知識」を参照してください。

ヒープ領域およびメタスペースの問題を通知する警告メッセージ(EXTP4368/IJServer30004)には3種類のメッセージがあり、EXTP4368/IJServer30004の詳細メッセージで示されます。この他の詳細メッセージは、「予兆監視警告メッセージ (ガーベジコレクション)」を参照してください。

メッセージは、以下の基準で出力されます。

詳細メッセージ	警告の意味	発生条件 (注1)
There are possibilities of OutOfMemoryError because of the lack of the memory: TIME={0}	ヒープ領域不足のため、OutOfMemoryErr	ガーベジコレクション直後のヒープ使用率が95%より大きい場合。または、ガーベジコレクション直後のヒープ使

詳細メッセージ	警告の意味	発生条件 (注1)
SIZE={1} TIME: 発生時刻 SIZE: ヒープ領域への追加量の目安(バイト単位) (注2)	or発生する危険性があります。	用率が90%より大きく、かつ、前回のガーベジコレクション発生時よりヒープ使用サイズが増加している状態が3回以上続いた場合。
There are possibilities of OutOfMemoryError because of the lack of the Metaspace: TIME={0} SIZE={1} TIME: 発生時刻 SIZE: メタスペースへの追加量の目安(バイト単位)(注2)	メタスペース不足のため、OutOfMemoryErrorが発生する危険性があります。	メタスペースの使用量が90%より大きい場合。(注3)

注1) ガーベジコレクションは、Full GCの情報を使用しています。
注2) 警告を回避するためには、警告発生時のヒープ領域またはメタスペースのサイズを大きくする必要があります。「SIZE」として通知している値は、そのための目安となる追加量です。
なお「SIZE」として通知している値は、あくまで目安です。実際に必要な追加量とは異なる場合がありますので、不足リソースの情報を元に、きちんとチューニングを実施してください。

注3) メタスペースは、クラスローダごとに管理するため、あるクラスローダでロードされたクラス情報を別のクラスローダが管理するメタスペースに格納できません。また、各クラスローダーのメタスペースは、クラス情報ごとに取得するのではなく、ある程度のまとまったサイズごとに取得して、そこにクラス情報を設定していきます。このため、取得済みのメタスペースに空きがある場合でも、別のクラスローダが管理するものであれば、その空きを使用できません(使用済みと扱います)。メタスペースの構造は、チューニングガイドの「JDK/JRE 8のチューニング」-「基礎知識」-「Javaヒープ、メタスペースとガーベジコレクション」を参照してください。

上記の理由で、メタスペース全体として空きがある場合でも、本警告がでる場合があります。

警告メッセージ(Javaヒープ領域不足)の原因と対処

EXTP4368/IJServer30004(Javaヒープ領域不足の警告メッセージ)が出力される原因は、JavaVMによってあらかじめ予約されたヒープ領域またはメタスペースの不足です。予約するヒープ領域、メタスペースのサイズが不適切な場合や、Javaアプリケーションがメモリリークを起こしている可能性があります。

警告メッセージが出力された場合、そのまま業務を継続するとメモリ不足やレスポンス低下などの問題が発生する可能性があります。これらの問題を解決するために警告メッセージに記載されている不足リソースの情報を元にチューニングを実施してください。
警告を回避するためには、現在の上限値を20%増加させて運用を再開します。それでも警告が出力された場合にはさらに20%増加させ、警告が出力されなくなるまで繰り返しチューニング実施します。チューニングを繰り返して警告メッセージが出力されない状態とすることにより、安定稼動するシステムを構築することができます。ヒープ領域サイズとメタスペースサイズの上限値を増加しても回避できない場合は、Javaアプリケーションが大量にメモリを消費したり、メモリリークを起こすことがないか見直しを行ってください。

JavaVMのヒープ領域およびメタスペースをチューニングする場合、Interstage管理コンソールのワークユニット設定でJavaVMオプションに、ヒープ領域およびメタスペースの上限値を設定するオプションを記載します。
JavaVMのヒープ領域の上限値を設定するためには、-Xmxオプションを使用します。

例

JavaVMのヒープ領域の上限値を256MByteに設定する場合
```
-Xmx256m
```

また、JavaVMのメタスペースの上限値を設定するためには、-XX:MaxMetaspaceSizeオプションを使用します。

例

JavaVMのメタスペースの上限値を256MByteに設定する場合
-XX:MaxMetaspaceSize=256m (本設定を省略した場合は、無制限です。(デフォルト値))

なお、チューニングは開発フェーズ(システムテスト)で実施し、問題を解決してください。
また、チューニング方法には、上記に挙げたヒープ領域またはメタスペースを増加する方法の他に、IJServerのプロセス多重度を増加することにより問題を解決することもできます。

予兆監視警告メッセージ (ガーベジコレクション)

JavaVMのガーベジコレクション処理の影響で業務レスポンス低下が発生する可能性を検出し、警告メッセージ(EXTP4368/IJServer30004)で通知します。
JavaVMのガーベジコレクションの影響を通知する警告メッセージ(EXTP4368/IJServer30004)には以下の5種類があり、EXTP4368/IJServer30004の詳細メッセージで示されます。この他の詳細メッセージは、「予兆監視警告メッセージ (Javaヒープ領域)」を参照してください。

- ガーベジコレクション処理時間に関する警告メッセージ(1種類)
- ガーベジコレクション間隔に関する警告メッセージ(4種類)

警告メッセージは、ガーベジコレクション処理による業務レスポンス低下の危険性を通知するもので、JavaVMの動作異常を示すものではありません。また、警告が出力された状態で業務を継続したときに、Javaアプリケーションの実行に支障が発生すると一概に言えませんが、警告メッセージが出力されない状態にすることにより、安定稼動するシステムを構築することができます。

ガーベジコレクション処理時間に関する警告メッセージ

メッセージは、EXTP4368/IJServer30004の詳細メッセージで示されます。また、以下の基準で出力されます。

詳細メッセージ	警告の意味 (注)	発生条件
It takes long time to do the garbage collections many times: TIME={0} AVERAGE={1} TIME: 発生時刻 AVERAGE: 過去3回の平均ガーベジコレクション時間(ミリ秒単位)	ガーベジコレクションに長い時間かかる状態が続いています。	過去3回のガーベジコレクション時間の平均値が5秒より長い場合。

注) ガーベジコレクションは、Full GCの情報を使用しています。

ガーベジコレクション処理中は、メッセージに記載された平均ガーベジコレクション時間のあいだ、アプリケーションが停止する場合があります。メッセージが出力された場合、レスポンス低下などの問題が発生する可能性があります。

警告メッセージ(ガーベジコレクション処理時間)の原因と対処

EXTP4368/IJServer30004(ガーベジコレクション処理時間に関する警告メッセージ)が出力される原因として以下が考えられます。

- 物理メモリの枯渇など、システムのリソースが不足している。
- CPU負荷の高い別のアプリケーションが同時に動作しているなど、メッセージの対象となったJavaプロセスの実行が阻害されている。
- ヒープ領域のサイズ指定が大き過ぎて、ガーベジコレクション処理に時間を要している。

アプリケーションのレスポンスに問題がある場合は、これらの要因を確認してください。

ヒープ領域のサイズ指定を小さくするチューニングは、十分な検証が必要です。Javaアプリケーションがヒープ領域を多く消費するプログラムの場合、ヒープ領域のサイズ指定を小さくするとOufOfMemoryErrorが発生することも考えられます。

ガーベジコレクション間隔に関する警告メッセージ

ガーベジコレクション間隔に関する4種類のメッセージは、EXTP4368/IJServer30004の詳細メッセージで示されます。いずれも、ガーベジコレクションが短い間隔で連続して発生した場合に、以下の基準で出力されます。

詳細メッセージ	警告の意味 (注1)	発生条件 (注2)
Inefficent garbage collections are run with the short intervals: TIME={0} WEIGHT={1} TIME: 発生時刻 WEIGHT: ガーベジコレクション直前の旧世代の使用率(%単位)	非効率なガーベジコレクションが短い間隔で発生しています。(注3)	過去3回のガーベジコレクション間隔時間が20秒より短い場合、かつ、ガーベジコレクション直前のOld世代領域の使用率が65%よりも小さい場合。
The garbage collections are run with the short intervals because of the lack of the memory: TIME={0} SIZE={1} TIME: 発生時刻 SIZE: ヒープ領域への追加量の目安(バイト単位)(注4)	ヒープ領域不足のため、ガーベジコレクションが短い間隔で発生しています。	過去3回のガーベジコレクション間隔時間が20秒より短い場合、かつ、ガーベジコレクション直後のヒープ使用率が65%より大きい場合。
System.gc() are run with the short intervals: TIME={0} INTERVAL={1} TIME: 発生時刻 INTERVAL: System.gcのインターバル時間(ミリ秒単位)	System.gcが短い間隔で発生しています。	過去3回のガーベジコレクション間隔時間が20秒より短い場合、かつ、System.gcによって発生したGCの頻度が高い場合。
The garbage collections are run with the short intervals: TIME={0} INTERVAL={1} TIME: 発生時刻 INTERVAL:ガーベジコレクションのインタバール時間(ミリ秒単位)	ガーベジコレクションが短い間隔で発生しています。	過去3回のガーベジコレクション間隔時間が20秒より短い場合。

注1) ガーベジコレクションは、Full GCの情報を使用しています。
注2) 発生条件を満たした場合でも、ガーベジコレクションの処理時間が短い場合は、メッセージの出力を行わない場合があります。
注3) IJServerワークユニットでは、コネクタとServletコンテナ間の通信にSSLを使用する設定になっている場合、ワークユニット起動時に、このメッセージを含んだメッセージが出力される場合があります。この場合、起動後にInterstage管理コンソールのモニタ画面でJavaVMのヒープ情報およびJavaVMのメタスペース情報に問題がなければ、メッセージを無視してください。
注4) 警告を回避するためには、警告発生時のヒープ領域のサイズを大きくする必要があります。「SIZE」として通知している値は、そのための目安となる追加量です。
なお「SIZE」として通知している値は、あくまで目安です。実際に必要な追加量とは異なる場合がありますので、不足リソースの情報を元に、きちんとチューニングを実施してください。

頻繁なガーベジコレクション処理の実行は、アプリケーションのレスポンス低下などの問題が発生する可能性があります。

警告メッセージ(ガーベジコレクション間隔)の原因と対処

「Inefficent garbage collections are run with the short intervals」が出力された場合、ヒープ領域の空きがあるにもかかわらず、ガーベジコレクション処理(非効率なガーベジコレクション)が発生している可能性があります。原因として以下が考えられます。

- アプリケーションがSystem.gc()、Runtime.gc()を短い間隔で呼び出している(製品のバージョンによって、「System.gc() are run with the short intervals」が出力されることがあります)。

・ ヒープ領域の初期サイズ(-Xms)が小さく、ヒープ拡張を繰り返している。

・ ヒープ領域のNew世代領域の割合が大きくなるように指定している(New世代領域については、「チューニングガイド」の「JDK/JRE 8のチューニング」の「基礎知識」を参照してください)。

アプリケーションのレスポンスに問題がある場合は、これらの要因(Javaヒープ領域に関するチューニングオプションや、アプリケーションからのSystem.gc()、Runtime.gc()の呼び出し)に問題がないか確認してください。

「The garbage collections are run with the short intervals because of the lack of the memory」が出力された場合、ヒープ領域の不足が考えられます。ヒープ領域の不足を回避するには、現在の上限値を20%増加させて運用を再開します。それでも警告が出力された場合にはさらに20%増加させ、警告が出力されなくなるまで繰り返しチューニング実施します。ヒープ領域サイズの上限値を増加しても警告が出力される場合、Javaアプリケーションが短時間に大量のメモリを消費している可能性があります。

「System.gc() are run with the short intervals」が出力された場合、アプリケーションがSystem.gc()、Runtime.gc()を短い間隔で呼び出しています。頻繁なガーベジコレクション実行が、アプリケーションのレスポンスに影響している場合は、System.gc()、Runtime.gc()が必要かアプリケーションの見直しを行ってください。

上記の3つのメッセージの発生条件以外で、ガーベジコレクション処理が短い間隔で発生した場合は、「The garbage collections are run with the short intervals」のメッセージが出力されます。

管理コンソールモニタ機能

Interstage管理コンソールのモニタ機能で、JavaVMの性能情報を参照することができます。予兆監視の警告メッセージで指摘されている問題を解決するための参考情報として、JavaVMの状態を知ることができます。
なお、ワークユニット起動時や一時的に負荷の高い状態の場合、予兆監視のメッセージが出力される場合があります。この場合、Interstage管理コンソールのモニタ画面でJavaVMのヒープ情報およびJavaVMのメタスペース情報に問題がなければ、メッセージを無視してください。

項目	説明
プロセス通番	ワークユニットが起動したプロセスの通番を表示します。
プロセスID	JavaVMのプロセスIDを表示します。
コンテナタイプ	JavaVMのコンテナタイプを「1VM」「Web」または「EJB」で表示します。
JavaVMの運用時間(msec)	JavaVMを起動してからの稼働時間を表示します。
JavaVMのヒープ情報(Kbyte)	JavaVMのヒープ情報を表示します。 現在:ヒープ情報の現在値を表示します。 最小:JavaVM起動時からのヒープ情報の最小値を表示します。 最大:JavaVM起動時からのヒープ情報の最大値を表示します。 上限:JavaVMのヒープサイズの上限値を表示します。上限値は-Xmxオプションで指定したヒープサイズとほぼ同等です。
JavaVMのメタスペース情報(Kbyte)	JavaVMのメタスペースの情報を表示します。メタスペースとは、Javaクラスファイル内の情報等が保存される領域です。 現在:メタスペース情報の現在値を表示します。 最小:JavaVM起動時からのメタスペース情報の最小値を表示します。 最大:JavaVM起動時からのメタスペース情報の最大値を表示します。 上限:JavaVMのメタスペースの上限値を表示します。上限値は-XX:MaxMetaspaceSizeオプションで指定したメタスペースサイズとほぼ同等です。
ガーベジコレクション情報	JavaVMのガーベジコレクション(Full GC)の情報を表示します。 発生回数:JavaVM起動時からのガーベジコレクションの発生回数を表示します。 処理トータル時間(msec):JavaVM起動時からのガーベジコレクショ

項目	説明
	ンの処理時間の合計値を表示します。 平均ガーベジコレクション発生間隔(msec):JavaVM起動時から前回までに発生したガーベジコレクションの、発生間隔の平均時間を表示します。

予兆監視のチューニング

予兆監視機能の警告メッセージは下記に出力されます。

- コンテナログ(IJServer30004)
- システムログ/イベントログ(EXTP4368)

システムログ/イベントログにEXTP4368を出力させるには、ワークユニットのJavaVMオプションに以下のオプションを設定します。

```
-Dcom.fujitsu.interstage.j2ee.ijserver.alert-monitoring.syslog=true
```

3.3.12 ワークユニットのアプリケーション自動再起動失敗時の縮退運用

アプリケーションの異常終了やアプリケーション最大処理時間超過により、プロセスが強制停止された場合、アプリケーション自動再起動機能により、プロセスが再起動されます。このとき、アプリケーションの起動処理で異常が発生した場合や、アプリケーションの起動時間が、「ワークユニット起動待ち時間」を超過して再起動に失敗すると、ワークユニットは異常終了します。
これを回避し、プロセスの再起動に失敗した場合でも、アプリケーションが複数多重で起動されているワークユニットでは、プロセス再起動失敗のメッセージを出力し、残りのプロセスでのワークユニット運用を継続する機能を提供します。

本機能は、以下のワークユニットで有効です。

- IJServerワークユニット

本機能では、ワークユニット内のプロセス多重度が2以上で動作している場合は、アプリケーションプロセスの自動再起動に失敗した場合でも、プロセス多重度が1つ減少した状態でワークユニットの運用を継続します。また縮退したアプリケーションプロセスを復元する機能を提供します。

IJServerワークユニットの場合は、Interstage管理コンソールの[システム]＞[ワークユニット]＞[ワークユニット名]＞[環境設定]画面で、[ワークユニット設定]の「アプリケーション自動再起動失敗時の制御」の項目で「ワークユニットの運用を継続する」を選択します。

縮退運用中のワークユニットの復元

縮退運用しているワークユニットを復元する機能を提供します。本機能はアプリケーションの自動再起動失敗により、減少してしまったプロセス多重度を、プロセスの再起動を実行し、本来のプロセス多重度に復元する機能です。プロセス多重度を動的に変更している場合は、変更後のプロセス多重度に復元します。
縮退運用しているワークユニットは以下の契機で復元されます。

- コマンド(isrecoverwu)の操作による復元
- Interstage管理コンソールの操作による復元

3.4 ワークユニットの起動・停止

Interstage管理コンソールの左フレームで[ワークユニット]を指定してください。次に右フレームでワークユニット名の一覧より、起動するワークユニット名を選択し、起動または停止ボタンをクリックしてください。
詳細については、Interstage管理コンソールのヘルプを参照してください。

3.4.1 起動時間監視

ワークユニットの起動完了までの待ち時間を指定することができます。ワークユニット起動待ち時間を指定すると、起動時実行クラスやinit処理において問題が発生して、起動処理がハングアップした場合や遅延したときに、プロセスを強制停止し、起動処理を中止させることができます。

なお、ワークユニット起動待ち時間のデフォルトは、以下です。

- IJServerワークユニットの場合、600秒

変更する場合は、Interstage管理コンソールよりワークユニットの環境設定を変更してください。なお、起動完了まで処理を中断したくない場合は、ワークユニット起動待ち時間に0を指定します。
また、IJServerワークユニットの場合、ワークユニット起動待ち時間を超過し、起動に失敗した場合、自動的にスレッドダンプが採取されます。
スレッドダンプはコンテナ情報ログ(info.log)に採取されます。また、タイムアウト発生直後と10秒後の2回出力されます。これにより、2回のスレッドダンプの変化により処理に時間の掛かっている原因を調査することができます。

3.4.2 停止時間監視

ワークユニットの停止が実行された場合、プロセスの停止が完了するまでの待ち時間を指定することができます。ワークユニットのプロセス強制停止時間が指定されている場合、停止処理がハングアップした場合に、プロセスを完全停止させることができます。

なお、ワークユニットのプロセス強制停止時間はデフォルトが180秒です。変更する場合は、Interstage管理コンソールよりワークユニットの環境設定を変更してください。なお、停止完了まで処理を中断したくない場合は、ワークユニットのプロセス強制停止時間に0を指定します。
また、IJServerワークユニットの場合、ワークユニットのプロセス強制停止時間を超過した場合、自動的にスレッドダンプが採取されます。
スレッドダンプはコンテナ情報ログ(info.log)に採取されます。また、タイムアウト発生直後と10秒後の2回出力されます。これにより、2回のスレッドダンプの変化により処理に時間の掛かっている原因を調査することができます。

注意

Interstage管理コンソールよりワークユニットを通常停止した場合、停止が60秒を超えても終了しない場合、停止処理中のままInterstage管理コンソール上ではエラー復帰します。その場合、しばらくしてから状態の再取得を実施し停止が完了したかどうかを確認してください。停止処理中のから変化しない場合は、強制停止を実施することができます。

3.5 J2EEアプリケーションの配備と設定

J2EEアプリケーションの配備

J2EEアプリケーションを実行環境に配備します。

Webアプリケーション、EJBアプリケーションの場合

Interstage管理コンソール、または、ijsdeploymentコマンドを使用します。
必要に応じて、IJServerを作成し、パッケージ化したアプリケーションを配備してください。
Webアプリケーションの場合は、IJServerディレクトリ配下にコピーしないで動作させることもできます。詳細は、“3.5.5 サーバ上の任意の位置で実行するWebアプリケーションの配備”を参照してください。
ijsdeploymentコマンドについては、“リファレンスマニュアル(コマンド編)”の“J2EE運用コマンド”を参照してください。

J2EEアプリケーションクライアントの場合

クライアントの実行環境へclient-jarファイルを複写して、jarコマンド等を使用し、client-jarファイルの中のdeployment descriptorファイルを任意のディレクトリに解凍します。

なお、EARファイルにJ2EEアプリケーションクライアントが含まれている場合は、Interstage管理コンソールの配備機能を使用してEARファイルから展開してください。そして、解凍したファイルの中からclient-jarファイルを取り出します。
J2EEアプリケーションクライアントの展開先については、“2.2.3 IJServerのファイル構成”を参照してください。

注意

- IJServerの環境設定で指定しているXMLパーサが配備時にも使用されます。配備時に必要となるXMLパーサが準拠しているべきJAXPのバージョンは、配備するモジュールの種別、およびバージョンによって異なります。詳細は、“3.5.1 配備に必要なXMLパーサの設定”を参照してください。
- EJBアプリケーションの配備ではJavacを実行するため、JREの環境では配備に失敗します。JDKの環境をインストールするようにしてください。

参照

- HotDeploy機能については、“3.5.3 J2EEのHotDeploy機能”を参照してください。
- クラスのオートリロード機能については、“3.5.4 クラスのオートリロード機能”を参照してください。
- サーバ上の任意の位置で実行するWebアプリケーションを配備する方法については、“3.5.5 サーバ上の任意の位置で実行するWebアプリケーションの配備”を参照してください。

J2EEアプリケーションの設定

WebアプリケーションとEJBアプリケーションのカスタマイズ

Interstage管理コンソールを使用して、WebアプリケーションとEJBアプリケーションをカスタマイズします。
Interstage管理コンソールの[ワークユニット] > “ワークユニット名”のカスタマイズしたいアプリケーションをクリックしてください。
各定義の詳細については、Interstage管理コンソールのヘルプを参照してください。

JSPの事前コンパイル

JSPの初回アクセス時には、JSPのコンパイルが実行されるためレスポンスが劣化します。このレスポンス劣化を回避するため、IJServerを起動する前にJSPをコンパイルしておくことができます。
また、JSPのオートリロード機能と併用して使用することで、IJServerの起動中でもJSPを入れ替えることができます。ただし、JSPの事前コンパイルを行った直後に変更が反映されるのではありません。いつ変更が反映されるのかは、JSPのリロード機能の設定に依存します。
コマンドの詳細は、“リファレンスマニュアル(コマンド編)”の“ijscompilejsp”を参照してください。

クライアントの設定

J2EEアプリケーションクライアントやアプレットからEJBを参照する場合には、別途EJBクライアントの環境設定が必要となります。詳細については“4.2 EJBを参照する場合の環境設定”を参照してください。

HTTPトンネリング

J2EEでHTTPトンネリングを使用する場合は、“セキュリティシステム運用ガイド”の“J2EEのHTTPトンネリング”を参照してください。

アプリケーションファイル保護レベル Solaris64 Linux32/64

アプリケーションファイルのアクセス権は、IJServer定義のアプリケーションファイル保護レベルの設定に応じて、下記のとおりに設定されます。

アプリケーションファイル保護レベル	アプリケーションファイルのアクセス権	
	ディレクトリ	ファイル
高(管理者のみ)	755	644
低(一般ユーザ)	777	666

アクセス権の変更対象となる資源は、下記の通りです。

- IJServerディレクトリ/appsディレクトリ、および配下の資源
- IJServerディレクトリ/Shared/libディレクトリ
- IJServerディレクトリ/Shared/classesディレクトリ
- IJServerディレクトリ/extディレクトリ

デフォルトの設定では、[高(管理者のみ)]が設定されています。必要に応じてアプリケーションファイル保護レベルの設定を変更してください。
なお、この項目の選択状態では、ファイル、およびディレクトリのオーナは変更されません。例えば、一般ユーザでファイルを格納している状態で、[高(管理者のみ)]を選択した場合には、そのファイルのオーナ、および管理者のみがそのファイルにアクセスすることができます。

注意

IJServerは起動時にumaskが022に設定されます。IJServerからファイル生成を行う場合、このumaskの設定に従います。

3.5.1 配備に必要なXMLパーサの設定

モジュールを配備する場合に必要となる「XMLパーサが準拠しているべきJAXPのバージョン」は、「モジュールの種別、およびバージョン」によって異なっており、その関係は下表のとおりとなります。

モジュールの種別、およびバージョン	必要なJAXPのバージョン
J2EE1.4(EAR)	1.2以上
Servlet2.4(WAR)	1.2以上
EJB2.1(ejb-jar)	1.2以上
Connector1.5(RAR)	1.2以上
J2EE1.3(EAR)	1.1以上
Servlet2.3(WAR)	1.1以上
EJB2.0(ejb-jar)	1.1以上
Connector1.0(RAR)	1.1以上
J2EE1.2(EAR)	1.1以上
Servlet2.2(WAR)	1.1以上
EJB1.1(ejb-jar)	1.1以上

上記の表を参考にして、配備するモジュールに応じてIJServerの環境設定で適切なXMLパーサを設定してください。
XMLパーサの設定方法については、“2.3.5 XMLパーサの設定”を参照してください。

なお、IJServerの環境設定で設定可能なXMLパーサとJAXPのバージョンの関係は下表のとおりです。

XMLパーサ	JAXPバージョン
Xerces2	1.3
その他	指定されたパーサに依存

IJServerの環境設定については、Interstage管理コンソールのヘルプを参照してください。

注意

- deployment descriptorに以下のように定義するエンコードは、各XMLパーサによってサポートするエンコードが異なります。他の環境で配備できていたモジュールを配備した場合に、deployment descriptorの読み込みに失敗した場合には、使用しているXMLパーサが指定しているエンコードをサポートしているか確認してください。ほぼすべてのXMLパーサがサポートする“UTF-8”を使用することを推奨します。

```
<?xml version="1.0" encoding="UTF-8"?>
. . .
```

3.5.2 J2EEアプリケーション(EARファイル)のdeployment descriptor

J2EEアプリケーション(EARファイル)のdeployment descriptor(application.xml)は、J2EEアプリケーションの動作環境を設定します。
ここでは、application.xmlの記述方法について説明します。

application.xmlの記述形式

application.xmlは、J2EEアプリケーション(EARファイル)についてのdeployment descriptorです。
J2EEアプリケーションの構成物のパスや、セキュリティロールを記述します。パスは、EARファイルにパッケージングした時のモジュール内のパスを記述します。
deployment descriptorの記述形式はXML形式です。deployment descriptorの記述形式を以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
  version="1.4">
  <display-name>display_name</display-name>
  <module>
    <connector>uri</connector>
    <alt-dd>uri</alt-dd>
  </module>
  <module>
    <ejb>uri</ejb>
    <alt-dd>uri</alt-dd>
  </module>
  <module>
    <java>uri</java>
    <alt-dd>uri</alt-dd>
  </module>
  <module>
    <web>
      <web-uri>uri</web-uri>
      <context-root>context-root</context-root>
    </web>
    <alt-dd>uri</alt-dd>
  </module>
  <security-role>
    <role-name>role-name</role-name>
  </security-role>
</application>
```

注意

記述にあたっての注意事項

- 先頭の<?xml...>は、XML宣言を記述しているため、deployment descriptorファイルの先頭で必ず記述してください。
- applicationタグ、およびapplicationタグの各属性は、J2EE名前空間のアプリケーションスキーマのバージョンを記述しているため、XML宣言の後に必ず記述してください。
- <application>、</application>は、XMLファイルの開始と終了を示すルートタグです。必ず指定してください。
- 各タグの記載順序は、上記の記載順序に従ってください。
- 大文字・小文字は区別します。
- J2EE1.2または1.3のdeployment descriptorの記述形式も使用できます。

application.xmlのタグ

J2EEアプリケーション(EARファイル)のdeployment descriptor(application.xml)には、以下のタグを指定できます。

タグ名	説明	タグの省略	複数の指定
application	J2EEアプリケーション(EARファイル)のdeployment descriptorの開始と終了を定義します。	×	×
display-name	J2EEアプリケーション(EARファイル)の名前を定義します。	○	×
module	一つのモジュールに関する定義を記述します。 注) connector、ejb、java、webのいずれかを必ず定義する必要があります。	×	○
connector	connector(RARファイル)のパスを定義します。	×	×
ejb	EJBモジュール(ejb-jarファイル)のパスを定義します。	×	×
java	J2EEアプリケーションクライアント(client-jarファイル)のパスを定義します。	×	×
web	Webモジュールに関する定義を記述します。	×	×
web-uri	Webモジュール(WARファイル)のパスを定義します。	×	×
context-root	Webアプリケーション名を定義します。 注) EARファイル内で一意な値でなければなりません	×	×
alt-dd	モジュールのdeployment descriptorのパスを定義します。省略した場合は、モジュールに含まれるdeployment descriptorが使用されます。	○	×
security-role	アクセス制限で使用するセキュリティロールを定義します。	○	○
role-name	セキュリティロール名を定義します。ロール名には、セキュリティ機能の運用設定(Interstage ディレクトリサービス)で定義したセキュリティロール名を指定してください。 注) EARファイル内で一意な値でなければなりません	×	×

注意

- 省略が“×”であるタグは、省略すると配備できません。
- 複数の指定が“×”であるタグを重複して指定すると配備できません。

例

モジュール構成が、以下の場合のdeployment descriptorの例です。

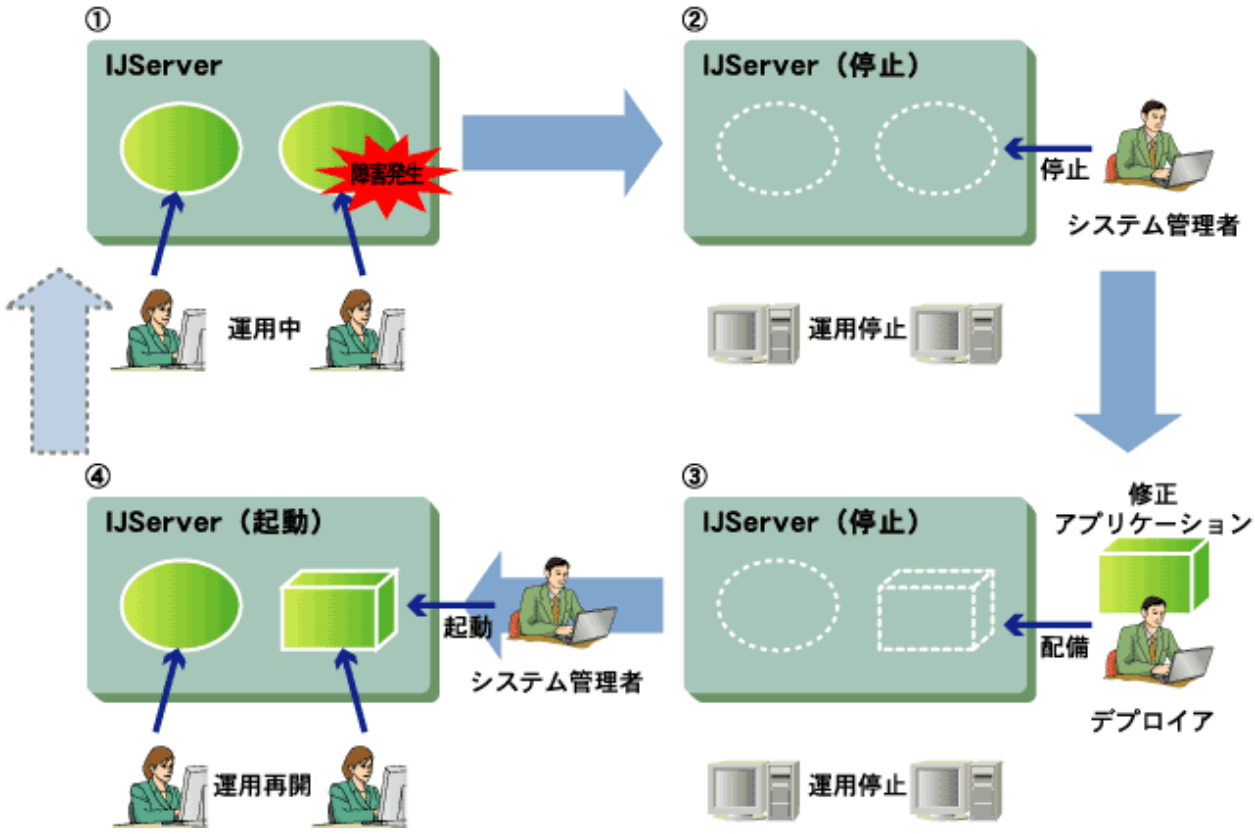
- EARファイルが以下を含む
  - connector.rar
  - ejb.jar
  - client.jar
  - web.war
  - connector.xml
  - ejb.xml
  - client.xml
  - web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/application_1_4.xsd"
  version="1.4">
  <display-name>J2EEApplication</display-name>
  <module>
    <connector>connector.rar</connector>
    <alt-dd>connector.xml</alt-dd>
  </module>
  <module>
    <ejb>ejb.jar</ejb>
    <alt-dd>ejb.xml</alt-dd>
  </module>
  <module>
    <java>client.jar</java>
    <alt-dd>client.xml</alt-dd>
  </module>
  <module>
    <web>
      <web-uri>web.war</web-uri >
      <context-root>WebApplication</context-root>
    </web>
    <alt-dd>web.xml</alt-dd>
  </module>
  <security-role>
    <role-name>Administrator</role-name>
  </security-role>
  <security-role>
    <role-name>Operator</role-name>
  </security-role></application>
```

3.5.3 J2EEのHotDeploy機能

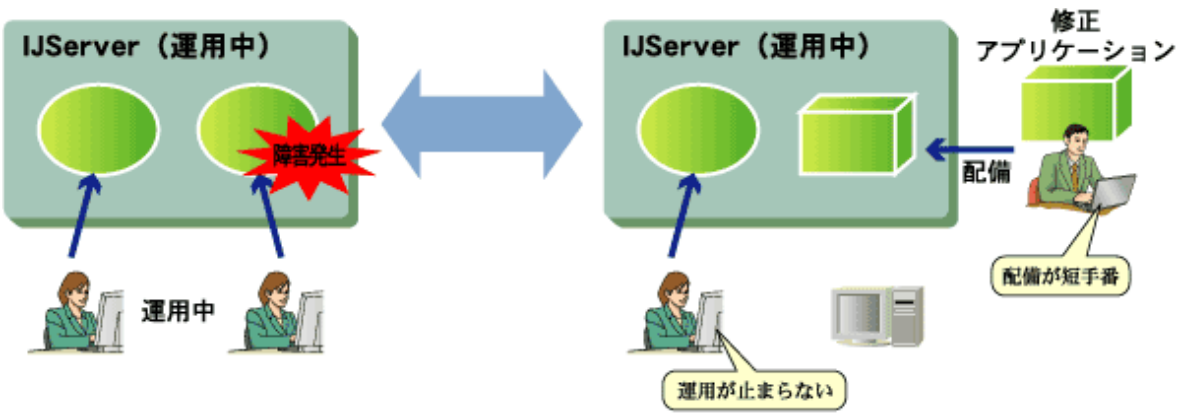
J2EEのHotDeploy機能を使用すると、IJServerを停止せずにモジュールの配備または再配備、配備解除が実行でき、運用中のIJServerに対してWebアプリケーションとEJBアプリケーションを追加または更新、削除できます。
配備中、または配備解除中ではないモジュールに対してリクエスト処理を行うことができるため、アプリケーションの開発が効率的に実施でき、IJServerの連続運用も可能です。

HotDeploy機能を使用しない場合のアプリケーション入れ替え方法



J2EEのHotDeploy機能を使用すると、以下のように配備のみでアプリケーションを入れ替えることが可能です。

HotDeploy機能を使用した場合のアプリケーション入れ替え方法



HotDeploy機能について、以下の順で説明します。

- 設計方法
- 運用方法
- 新規配備/再配備/配備解除/再活性
- 配備モジュールの状態
- 新規配備/再配備/配備解除/再活性時に非活性化/活性化されるモジュール
- Sharedディレクトリ

設計方法

開発効率の向上、および運用中の保守性の向上を目的とした機能として“HotDeploy機能”と“クラスのオートリロード機能”を提供しています。HotDeploy機能だけを使用しても十分な効果がありますが、クラスのオートリロード機能を使用することで、さらに開発効率が向上する場合があります。クラスのオートリロード機能については、“3.5.4 クラスのオートリロード機能”を参照してください。

運用方法

HotDeploy機能を使用する場合は、Interstage管理コンソールの[ワークユニット] > [新規作成]タブを選択し、[詳細設定]表示後の[共通定義]より設定してください。また、ワークユニット作成後は、Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブを選択し[共通定義]から変更できます。

HotDeploy機能を使用する場合は、後述の新規モジュールの配備または既存配備モジュールの再配備を効率的に行うため、HotDeploy機能を使用しない場合とIJServerの起動状態が異なる場合があります。

以下に、配備モジュールとIJServerの起動状態の関係について説明します。

配備モジュールが存在しない場合のIJServerの起動状態

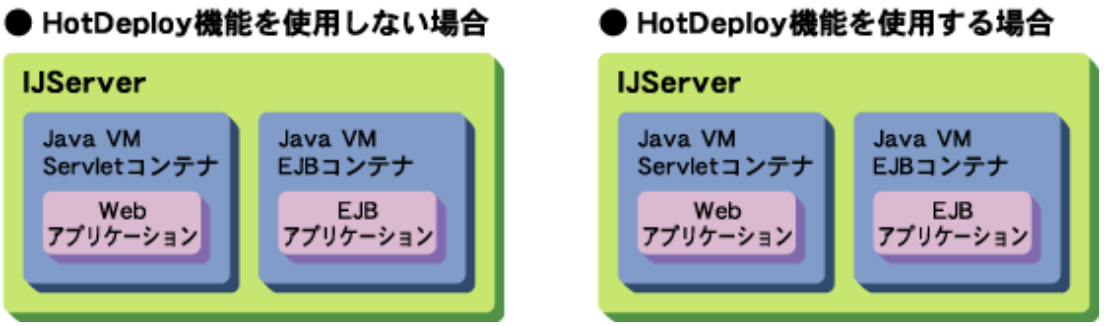
HotDeploy機能を使用する/しないに関係なく、配備モジュールが存在しない場合はIJServerの起動に失敗します。

配備モジュールが存在する場合のIJServerの起動状態

IJServerタイプが“Webアプリケーションと EJBアプリケーションを別JavaVMで運用”の場合は、以下のとおりです。
それ以外のIJServerタイプは、下図に示すHotDeploy機能を使用しない場合と同様です。

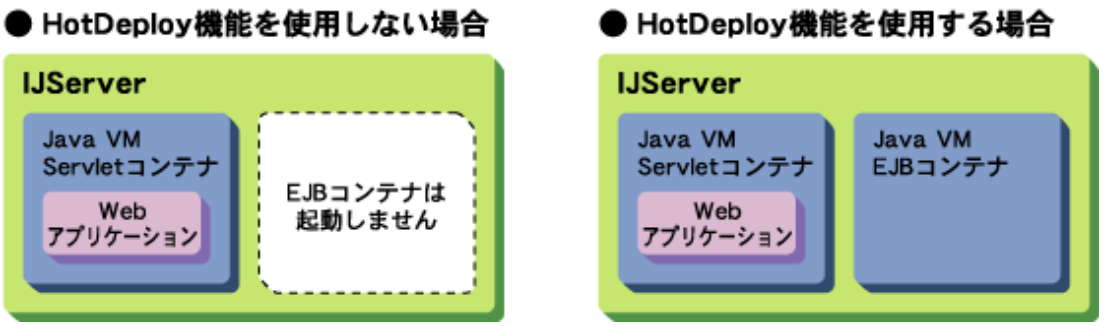
Webアプリケーションおよび EJBアプリケーションを配備した場合

HotDeploy機能の使用する/しないに関係なく、すべてのJava VM(Servletコンテナおよび EJBコンテナ)を起動します。



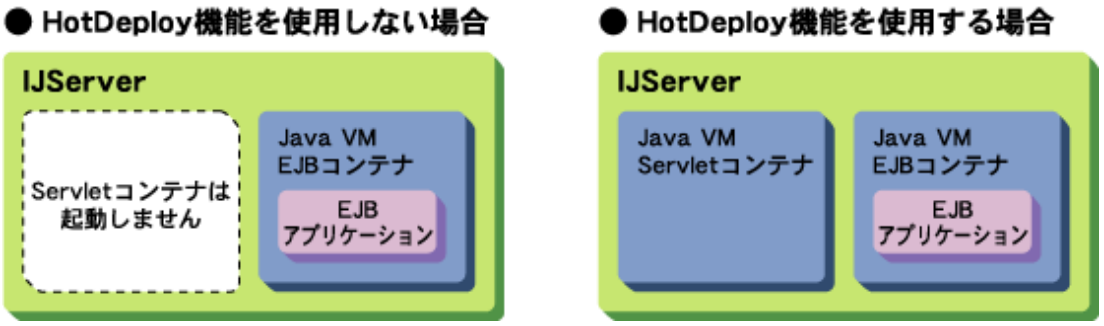
Webアプリケーションのみを配備した場合

アプリケーションが配備されているJava VMを起動します。なお、HotDeploy機能を使用する場合だけ、EJBコンテナに EJBアプリケーションが配備されている/いないに関係なくJava VM(EJBコンテナ)を起動します。



EJBアプリケーションのみを配備した場合

アプリケーションが配備されているJava VMを起動します。HotDeploy機能を使用する場合だけ、Servletコンテナに Webアプリケーションが配備されている/いないに関係なくJava VM(Servletコンテナ)を起動します。



配備モジュールが存在するが活性化に失敗する場合の起動状態

EJBアプリケーションの場合は、以下のとおりです。なお、Webアプリケーションの場合は、HotDeployを使用する/しないに関係なく、活性化が成功するアプリケーションだけを起動します。

HotDeploy機能を使用する場合

活性化が成功するEJBアプリケーションだけを起動します。

HotDeploy機能を使用しない場合

すべてのEJBアプリケーションは起動しません。

HotDeploy機能を使用する場合と使用しない場合について、以下の表に示します。

EAR間で分離する/すべて分離する
配備モジュールが存在しない場合は、IJServerの起動は行いません。
配備モジュールが存在する場合は、以下の表に従いIJServerの起動を行います。

		HotDeploy機能を使用しない場合		HotDeploy機能を使用する場合	
配備されているモジュールの状態/配備されているモジュールの種別		一部またはすべてのモジュールが活性不可能	すべてのモジュールが活性可能	一部またはすべてのモジュールが活性不可能	すべてのモジュールが活性可能
war		起動します		起動します	
ejb-jar		起動しません	起動します		
ear	warのみ	起動します			
	ejb-jarのみ	起動しません	起動します		
	warおよびejb-jar	含まれるejb-jarがすべて活性可能なモジュールの場合は起動します	起動します		

注意

HotDeploy機能を使用しない場合、EJBアプリケーションの活性化に失敗するとIJServerは起動しません。

新規配備/再配備/配備解除/再活性

新規配備/再配備/配備解除/再活性は、Interstage管理コンソールを使用して実行してください。新規配備/再配備はijsdeploymentコマンドでも実施できます。配備解除はijsundeloymentコマンドでも実施できます。

新規配備(新規モジュールの配備)

配備を実行すると、運用環境へのモジュールの配備と、配備モジュールの活性化を実行します。
Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [配備]で、配備対象のモジュールを「参照」ボタンから

選択してください。
ijsdeploymentコマンドを使用する場合、以下を実行してください。

```
ijsdeployment -n IJServerのワークユニット名 -f 配備対象のモジュール
```

再配備(既存配備モジュールの再配備)

配備を実行すると、配備モジュールの非活性化(注1)、モジュールの配備、配備モジュールの活性化を実行します。
Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [配備]で、再配備するモジュールを「参照」ボタンから選択してください。
ijsdeploymentコマンドを使用する場合、以下を実行してください。

```
ijsdeployment -n IJServerのワークユニット名 -f 配備対象のモジュール -r
```

配備解除

配備解除を実行すると、配備モジュールの非活性化(注1)、モジュールの配備解除を実行します。
Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [アプリケーション状態/配備解除]タブから[チェックボックス]で選択されている配備モジュールを配備解除します。
ijsundeploymentコマンドを使用する場合、以下を実行してください。

```
ijsundeployment -n IJServerのワークユニット名 -t 配備モジュール
```

再活性(注2)

Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [アプリケーション状態/配備解除]タブから配備モジュールを選択して再活性ボタンを押すと、配備モジュールの非活性化、配備モジュールの活性化を実行して、定義ファイルの再読み込みとすでに読み込まれたクラスファイルの破棄を行います。
なお、再活性を行う場合、以下の設定が反映されます。

- Webアプリケーションの場合
  - モジュールの環境設定画面の設定
  - モジュールの名前変換画面の設定
- EJBアプリケーションの場合
  - モジュールの環境設定画面の設定
  - モジュールの名前変換画面の設定
  - アプリケーションのアプリケーション環境定義画面の設定

注1)

配備モジュールの非活性化では、以下を実行します。

1. 新しいリクエストの受付けを停止
2. 非活性化処理が開始される前に受け付けたリクエストの処理が終了するまで待機
3. モジュールの非活性化

配備モジュールの非活性化では、新しいリクエストの受付けを停止して、処理中のリクエストの処理が終了するまで待機します。(※)1分間待機してもリクエストの処理が終了しない場合、配備モジュールが“非活性化処理中”のままエラーが発生します。この場合、リクエストの処理を終了するまで(状態が“非活性”となるまで)待ってから再度配備を実行するか、または、IJServerを再起動してください。
“非活性”状態となったモジュールについては、再活性することにより活性化することができます。異常状態となったモジュールについては、異常が発生した原因を取り除きIJServerを再起動することにより運用可能な状態となります。
配備モジュールの状態については“配備モジュールの状態”を、また再活性の方法については“再活性”を参照してください。

※) Session Beanの非活性化を行う場合、クライアントまたは別モジュールからの以下のリクエストは、Session Beanにとってそれぞれ別のリクエストになります。

- Session Beanのcreate

ー ビジネスメソッド呼び出し

ー Session Beanのremove

そのためビジネスメソッド処理中に非活性化された場合、ビジネスメソッドが復帰すると非活性化が行われてアクセスはできなくなります。ビジネスメソッド呼出し後にSession Beanのremoveを行っている場合、ビジネスメソッドは正常に復帰してもremoveでエラーになります。

注2)

再活性は、以下の場合に使用してください。

ー 異常状態のモジュールに対して異常原因を取り除いた後に配備モジュールを活性化する場合

ー IJServerを停止せずに特定のモジュールのチューニングパラメタやその他の動作モードを変更する場合

ー IJServerを停止せずに特定のモジュールのキャッシュをクリアしたい場合
例えばインスタンス管理モードがReadOnlyのEntity Beanインスタンスのキャッシュをクリアする場合（非活性化されるとアプリケーションが一度初期化されるため、アプリケーションやコンテナが保持している情報はクリアされます。）

参照

新規配備/再配備/配備解除/再活性の実行に失敗した場合については、“29.5 HotDeploy機能使用時の異常”を参照してください。

配備モジュールの状態

各モジュールの状態はInterstage管理コンソールから確認できます。Interstage管理コンソールには以下のように表示されます。

状態	説明	対処
活性	配備モジュールがリクエストを受付けることができます。	ー
非活性	配備モジュールがリクエストを受付けることができません。	コンテナログを参照し、非活性状態となっている原因を取り除き、再活性処理を行ってください。
活性（一部）	配備モジュールがリクエストを受付けることができますが、IJServerの一部のプロセスでモジュールが非活性状態となっています。	コンテナログを参照し、非活性状態となっている原因を取り除き、再活性処理を行ってください。
活性（不整合）	配備モジュールがリクエストを受付けることができますが、各プロセスで活性化されているモジュールの整合性が合っていません。 （配備中にIJServerの一部プロセスを再起動したため、あるプロセスが再配備前のモジュールをロードしています。） 活性（一部）と活性（不整合）が混在している状態もこの状態となります。	ー
活性化処理中	配備モジュールの活性化およびリクエスト受付開始の処理中です。	ー
非活性化処理中	配備モジュールのリクエスト受付停止および非活性化処理中です。	状態が変わらないときは、メモリ不足が発生して処理が中断された場合があります。IJServerを停止し、再度起動を実行してください。
異常	配備、配備解除、再活性中にIJServerの一部プロセスが再起動されたため、以下の異常が発生しています。	コンテナログを参照し、異常状態となっている原因を取り除いてください。

状態	説明	対処
	・ 活性化処理中のプロセスと非活性化処理中のプロセスが混在しています。 配備モジュールの活性化/非活性化/オートリロード処理中に予期せぬ異常が発生しています。	IJServerを停止し、再度起動を実行してください。

新規配備/再配備/配備解除/再活性時に非活性化/活性化されるモジュール

新規配備/再配備/配備解除/再活性を実行した場合、配備されたモジュールが非活性化、および活性化されますが、この場合配備するモジュールのクラスを参照するすべてのモジュールも非活性化および、活性化されます。
配備モジュールの参照関係はクラスローダの分離方法(EAR間で分離、すべて分離、分離しない)により異なります。

クラスローダの分離方法ごとの非活性化および、活性化する配備モジュールは、以下のように異なります。クラスローダの分離については、“2.3.2 クラスローダの分離”を参照してください。

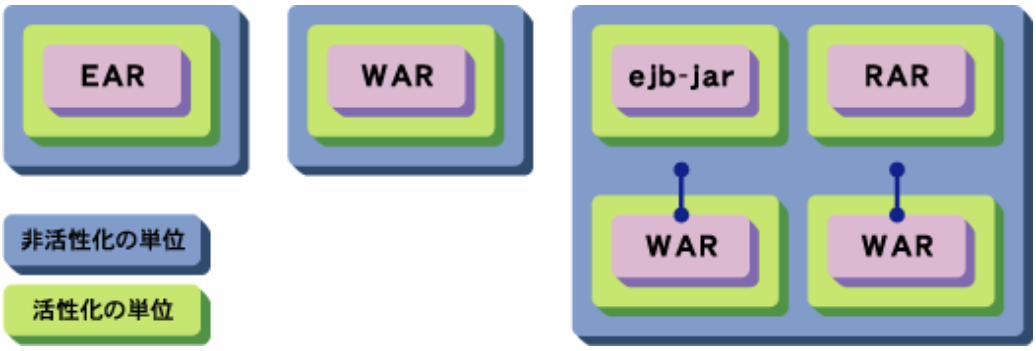
クラスローダの設定	非活性化/活性化される配備モジュール			
	EAR	WAR	ejb-jar	RAR
EAR間で分離	対象のEARのみ	対象のWARのみ	配備されているすべてのejb-jarまたはWAR、RAR	配備されているすべてのejb-jarまたはWAR、RAR
すべて分離	対象のEARのみ	対象のWARのみ	対象のejb-jarのみ	対象のRARのみ
分離しない	－※	－※	－※	－※

※)HotDeploy機能は使用できません。

以下にクラスローダの分離方法による非活性化、および活性化について説明します。
図中の非活性化の単位で示す範囲で非活性化し、活性化の単位で示す範囲で活性化します。なお、モジュールの活性化に失敗した場合、失敗したモジュールを除いて他のモジュールについては活性化を継続します。

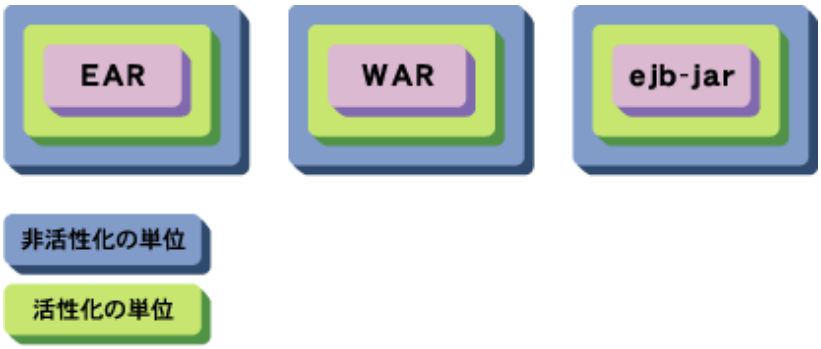
EAR間で分離の場合

EAR間で分離の場合、ejb-jarやRARのクラスは別のejb-jarまたはRAR、WARから参照することができるためejb-jarやRARを配備すると、配備されているすべてのejb-jarおよびRAR、WARが非活性化および活性化されます。なお、EARやWARを配備すると配備モジュールは個々に非活性化および活性化されます。



すべて分離の場合

すべて分離の場合、モジュールは個別に非活性化と活性化されます。



Sharedディレクトリ

Sharedディレクトリには以下の2種類があり、再活性の動作が異なります。

IJServerディレクトリ配下のSharedディレクトリ

IJServer内で共通に使用するクラスを設定するディレクトリです。本ディレクトリ内のクラスまたはjarファイルは再活性の対象外のため、置き換えた場合にはIJServerを再起動するまで反映されません。

EARに含まれるSharedディレクトリ

EAR内のアプリケーション間で共通に使用するクラスを設定するディレクトリです。本ディレクトリ内のクラスまたはjarファイルは再活性の対象です。

注意

J2EEのHotDeploy機能の注意事項

- Interstage管理コンソールのシステム、リソースおよびIJServerで指定した定義の変更は反映の対象となりません。
- 再活性した場合、ServletのセッションとSTATEFUL Session Beanのインスタンスは破棄されますので、再作成してください。ただし、セッションリカバリ機能使用時のServletのセッションについては、自動的にバックアップ・リカバリされ、継続可能です。
- JNIを使用するクラスを含むアプリケーションを再活性または、再配備した場合、nativeモジュールのロードに失敗し、java.lang.UnsatisfiedLinkErrorがスローされる可能性があります。この場合はIJServerを再起動する必要があります。JNIを使用するアプリケーションに対してHotDeployを行う場合は、“クラスローダ使用時の注意事項”の“J2EEアプリケーションでJNIを使用する場合の注意事項”を参照して対処を行ってください。
- IJServer上でWebサービスクライアントを運用する場合、該当アプリケーションにHotDeploy機能は使用しないでください。
- EJBアプリケーションのクライアント配布物を、他のIJServerなどのJava VMが直接参照している場合、上書き配備に失敗することがあります。
  クライアント配布物はコピーしてから参照するように変更してください。

3.5.4 クラスのオートリロード機能

クラスのオートリロード機能とは、IJServerを停止せずに配備済みのアプリケーションのクラスを入れ替えることができる機能です。
クラスのオートリロードは以下の変更があった場合に行われます。

- アプリケーション内に用意したjarファイルを置き換えた場合
- アプリケーションを実行する際にロードされたclassファイルを置き換えた場合
- WEB-INF/lib配下または、ear内のShared/lib配下に新規jarファイルを追加した場合

クラスのオートリロードを行う場合の設定は、Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブで、[オートリロード機能の使用]を“する”に設定します。

クラスのオートリロード機能を使用した場合、修正したアプリケーションのクラスやJarファイルを置き換えるだけで自動的にロードされるため、アプリケーションを配備し直す必要がなく、またIJServerの停止と起動の必要もないため、効率よく開発で

きます。
アプリケーションの開発を行う場合は、クラスのオートリロード機能を行うことを推奨します。

設計方法

配備モジュールのクラスファイルを頻繁に変更して動作確認する開発作業中は、クラスのオートリロード機能を使用することで開発効率を向上させることが可能です。ただし、クラスのオートリロード機能はコンテナがクラスファイルの変更を絶えず監視するため、処理性能が劣化します。このため、アプリケーションの開発時のみ使用してください。

また、以下のクラスについては入れ替えることができません。以下のクラスを入れ替える場合にはHotDeploy機能を使用してください。HotDeploy機能については“3.5.3 J2EEのHotDeploy機能”を参照してください。

- クラスのオートリロード機能で入れ替えることができないクラス
  EJBのインタフェース(Remoteインタフェース／Homeインタフェース／Localインタフェース／LocalHomeインタフェース)

運用方法

クラスのオートリロード機能ではクラスを定期的に監視するため、Interstage管理コンソールから監視時間間隔を定義する必要があります。Interstage管理コンソールの[ワークユニット]>“ワークユニット名”>[環境設定]タブで設定を行ってください。詳細はInterstage管理コンソールのヘルプを参照してください。

実際に入れ替えを行いたいクラスファイルは、配備ディレクトリに直接コピーしてください。
クラスのオートリロード機能の対象は以下のディレクトリに含まれるクラスです。コピー先(配備ディレクトリ)の詳細については、“2.2.3 IJServerのファイル構成”を参照してください。

Windows32/64

[J2EE共通ディレクトリ]¥ijserver¥[IJServer名]¥apps配下の以下のファイル

- WARファイルを配備した場合
  - [Webモジュール名]¥WEB-INF¥lib配下の拡張子が“.jar”のファイル
  - [Webモジュール名]¥WEB-INF¥classes配下の拡張子が“.class”のファイル
- ejb-jarファイルを配備した場合
  - [EJBモジュール名]配下の拡張子が“.class”のファイル
- rarファイルを配備した場合
  - [RARのファイル名]配下の拡張子が“.jar”のファイル
- EARファイルを配備した場合
  - [EARモジュール名]¥[WARのファイル名]¥WEB-INF¥lib配下の拡張子が“.jar”のファイル
  - [EARモジュール名]¥[WARのファイル名]¥WEB-INF¥classes配下の拡張子が“.class”のファイル
  - [EARモジュール名]¥[ejb-jarのファイル名]配下の拡張子が“.class”のファイル
  - [EARモジュール名]¥[RARのファイル名]配下の拡張子が“.jar”のファイル
  - [EARモジュール名]¥Shared¥lib配下の拡張子が“.jar”のファイル
  - [EARモジュール名]¥Shared¥classes配下の拡張子が“.class”のファイル

Solaris64 Linux32/64

[J2EE共通ディレクトリ]/ijserver/[IJServer名]/apps配下の以下のファイル

- WARファイルを配備した場合
  - [Webモジュール名]/WEB-INF/lib配下の拡張子が“.jar”のファイル

- [Webモジュール名]/WEB-INF/classes配下の拡張子が“.class”のファイル

・ ejb-jarファイルを配備した場合

- [EJBモジュール名]配下の拡張子が“.class”のファイル

・ rarファイルを配備した場合

- [RARのファイル名]配下の拡張子が“.jar”のファイル

・ EARファイルを配備した場合

- [EARモジュール名]/[WARのファイル名]/WEB-INF/lib配下の拡張子が“.jar”のファイル
- [EARモジュール名]/[WARのファイル名]/WEB-INF/classes配下の拡張子が“.class”のファイル
- [EARモジュール名]/[ejb-jarのファイル名]配下の拡張子が“.class”のファイル
- [EARモジュール名]/[RARのファイル名]配下の拡張子が“.jar”のファイル
- [EARモジュール名]/Shared/lib配下の拡張子が“.jar”のファイル
- [EARモジュール名]/Shared/classes配下の拡張子が“.class”のファイル

クラスファイルを入れ替えた場合、そのクラスを参照できるモジュールのクラスがすべてオートリロードされます。クラスファイルを参照できるモジュールは、クラスローダの設定により異なります。以下にオートリロードされるモジュールのクラスを示します。

オートリロードされるクラス

<table>
<tr><th rowspan="2">入れ替えるクラス</th><th colspan="3">クラスローダの設定</th></tr>
<tr><th>EAR間で分離</th><th>すべて分離</th><th>分離しない</th></tr>
<tr><td>個別に配備されたWARのクラス</td><td colspan="2">入れ替えたWARのクラスのみ</td><td rowspan="8">なし(クラスのオートリロード機能は使用できません。)</td></tr>
<tr><td>EARに含まれるWARのクラス</td><td colspan="2">入れ替えたWARのクラスのみ</td></tr>
<tr><td>個別に配備されたejb-jarのクラス</td><td>個別に配備されたejb-jarまたはRAR、WARのクラスすべて</td><td>入れ替えたejb-jarのクラスのみ</td></tr>
<tr><td>EARに含まれるejb-jarのクラス</td><td colspan="2">EARに含まれるモジュールのクラスすべて</td></tr>
<tr><td>EARに含まれるSharedディレクトリ配下のクラス</td><td colspan="2">EARに含まれるモジュールのクラスすべて</td></tr>
<tr><td>個別に配備されたRARのクラス</td><td>個別に配備されたejb-jarまたはRAR、WARのクラスすべて</td><td>なし(配備を行うことができないため、オートリロードの対象になりません。)</td></tr>
<tr><td>EARに含まれるRARのクラス</td><td colspan="2">EARに含まれるモジュールのクラスすべて</td></tr>
</table>

Sharedディレクトリ

Sharedディレクトリには以下の2種類があり、クラスのオートリロードの動作が異なります。

IJServerディレクトリ配下のSharedディレクトリ

IJServer内で共通に使用するクラスを設定するディレクトリです。
本ディレクトリ内のクラスまたはjarファイルはオートリロードの対象外のため、置き換えた場合にはIJServerを再起動するまで反映されません。

EARに含まれるSharedディレクトリ

EAR内のアプリケーション間で共通に使用するクラスを設定するディレクトリです。
本ディレクトリ内のクラスまたはjarファイルはオートリロードの対象です。

注意

クラスのオートリロードの注意事項

- アプリケーションの配備先資源を直接変更する際、変更できる権限が一般ユーザに付与されていない場合は、必要に応じて管理者が権限を変更してください。
- EJBのインタフェース(LocalHome／Local／Home／Remoteインタフェース)、およびresource adapterのインタフェースは変更できません。インタフェースを変更した場合、以下のエラーが発生する場合があるため、HotDeploy機能で再度配備してください。
  - NoClassDefFoundException
  - NoClassDefFoundError
  - NoSuchMethodError
  - CMP2.0のEntity Beanでは、[CMP2.x-XXXX]のメッセージなどが出力される場合があります。
- モジュールが非活性状態の場合は、オートリロードされません。
- アプリケーションが非活性状態となっても、該当アプリケーションに含まれる、あるいはアプリケーションが参照しているクラスファイルやjarファイルがクラスローダでロードされ、リロードの監視対象となる場合があります。
  以下のようなクラスファイル、jarファイルが該当します。
  - アプリケーションの起動(活性化)処理の延長でロード済みのクラスファイル
  - RAR内のjarファイル
  - ejb-jar、RARのマニフェストファイルのClass-Pathに指定されたjarファイル

  そのため、更新された場合にはリロードの監視でチェックされ、同じクラスローダ上で動作しているアプリケーションのリロードが行われます。
  このとき、非活性のアプリケーションは活性化されません。
  活性化したい場合は、HotDeploy機能を使用した再配備・再活性または、ワークユニットの再起動を行ってください。
- クラスのオートリロード機能はコンテナがクラスファイルの変更を絶えず監視するため、処理性能が劣化します。このため、アプリケーション開発時のみ使用してください。
- クラスのオートリロード機能は、クラスの入れ替えだけを行います。このため、Interstage管理コンソールのシステム、リソースおよびIJServerで変更した定義、deployment descriptor(Webアプリケーションのweb.xmlを除く)などの定義を変更した結果は有効となりません。
- Webアプリケーションのdeployment descriptor(web.xml)は監視対象ではありませんが、クラスやjarファイルの更新があった場合に再読み込みされます。
  deployment descriptor(web.xml)の更新を反映させるためには、まずdeployment descriptor(web.xml)の更新を行った後に他のロード済みのクラスファイルまたはjarファイルの更新を行い、オートリロードを動作させてください。
  deployment descriptor(web.xml)のみ更新の場合は、モジュールの再活性またはワークユニットの再起動を行ってください。
  また、deployment descriptor(web.xml)の更新によりサーブレット定義がなくなった場合でも、Interstage管理コンソールのWebアプリケーションのモニタには、それまでの情報が累積で表示されます。情報をリセットする場合は、モジュールの再活性またはワークユニットの再起動を行ってください。
- アプリケーションの配備先に保管されたinterstage.xmlは、IJServerが動作するために使用するファイルのため、削除は行わないでください。また、interstage.xmlをテキストエディタなどで編集する場合は<web>タグ、および<ejb>タグ以外は編集しないように注意してください。
  ファイルの削除または、<web>タグおよび<ejb>タグ以外を編集した場合、IJServerが正常に動作しません。この場合は、該当するIJServerを削除する必要があります。
  なお、アプリケーションの配備先については“2.2.3 IJServerのファイル構成”を参照してください。
  interstage.xmlの編集については“4.11.2 interstage.xmlファイル”を参照してください。

- クラスのオートリロードを使用する場合、クラスファイルおよびjarファイルの入れ替えのタイミングによっては、以下のようにアプリケーションの非活性化／活性化が動作します。

	アプリケーションの非活性化、活性化
クラスファイルを追加した場合	動作しない
jarファイルを追加した場合	動作する
ロード済みのクラスファイルを置き換えた場合	動作する
ロードされていないクラスファイルを置き換えた場合	動作しない
jarファイルを置き換えた場合	動作する

  アプリケーションの非活性化／活性化が動作した場合、Servletのセッションと STATEFUL Session Beanのインスタンスは破棄されますので、再作成する必要があります。

- JNIを使用するクラスを含むアプリケーションに対してクラスのオートリロードを行った場合、nativeモジュールのロードに失敗し、java.lang.UnsatisfiedLinkErrorがスローされる可能性があります。この場合はIJServerを再起動する必要があります。JNIを使用するアプリケーションに対してクラスのオートリロードを行う場合は、“クラスローダ使用時の注意事項”の“J2EEアプリケーションでJNIを使用する場合の注意事項”を参照して対処を行ってください。

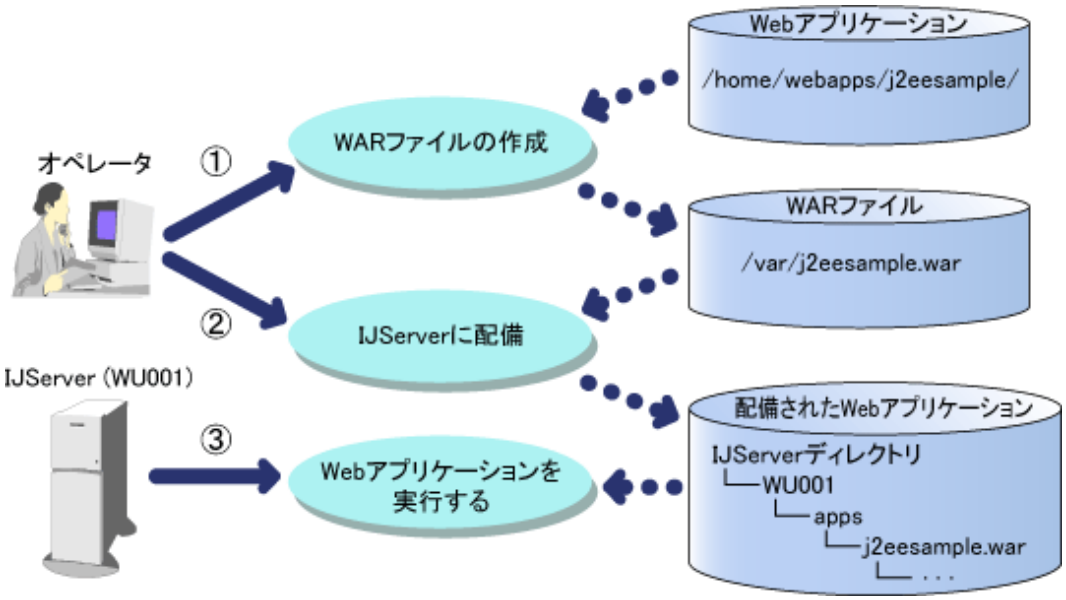
- IJServer上でWebサービスクライアントを運用する場合、該当アプリケーションにオートリロード機能は使用しないでください。

- Windows32/64
  IJServerが起動している場合、jarファイルはプロセスで使用中となり、上書きおよび削除できない場合があります。その場合は、IJServerをいったん停止しjarファイルの反映後に再起動、または、HotDeploy機能による配備解除/再配備を行ってください。

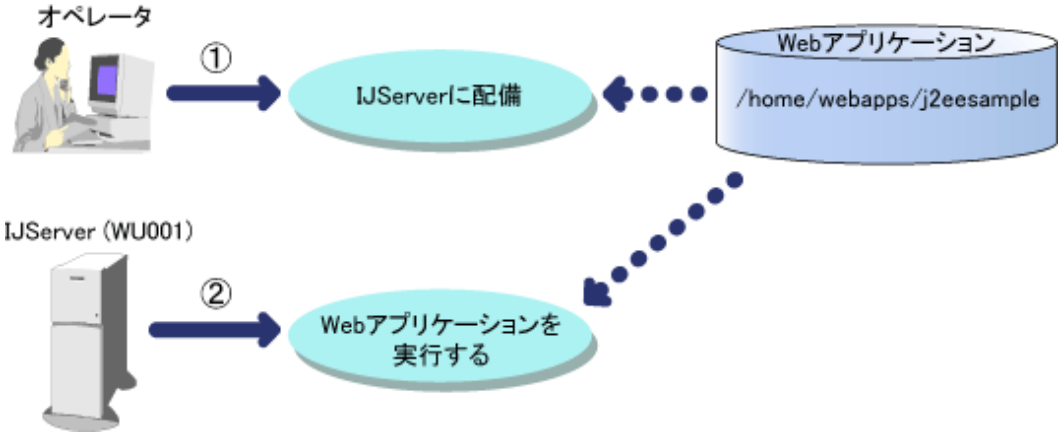
- Solaris64 Linux32/64
  ファイルをコピーする場合、ファイルのアクセス権はコピー前と同様に設定してください。アクセス権に不当な設定をした場合、アプリケーションの実行に失敗または配備失敗、配備解除失敗などが発生する可能性があります。

3.5.5 サーバ上の任意の位置で実行するWebアプリケーションの配備

Interstage Application Server V7以前では、WebアプリケーションをIJServerに配備するためには、まず、以下の図のようにマシン上に展開されているWebアプリケーションから「WARファイルを作成」します。次に、作成したWARファイルを「IJServerに配備」します。IJServerに配備したWebアプリケーションはIJServerディレクトリ配下に展開されます。そして、IJServerはIJServerディレクトリ配下に展開された「Webアプリケーションを実行」します。



本製品の8.0以降では、サーバ上の特定ディレクトリ配下に展開されたWebアプリケーションを、IJServerディレクトリ配下にコピーすることなく、IJServerで動作させることができます。
Interstage管理コンソールで配備する際に、“サーバ上の任意の位置で実行するWebアプリケーションを配備する”を選択してください。なお、ijsdeploymentコマンドを使用して配備する場合については、“リファレンスマニュアル(コマンド編)”を参照してください。

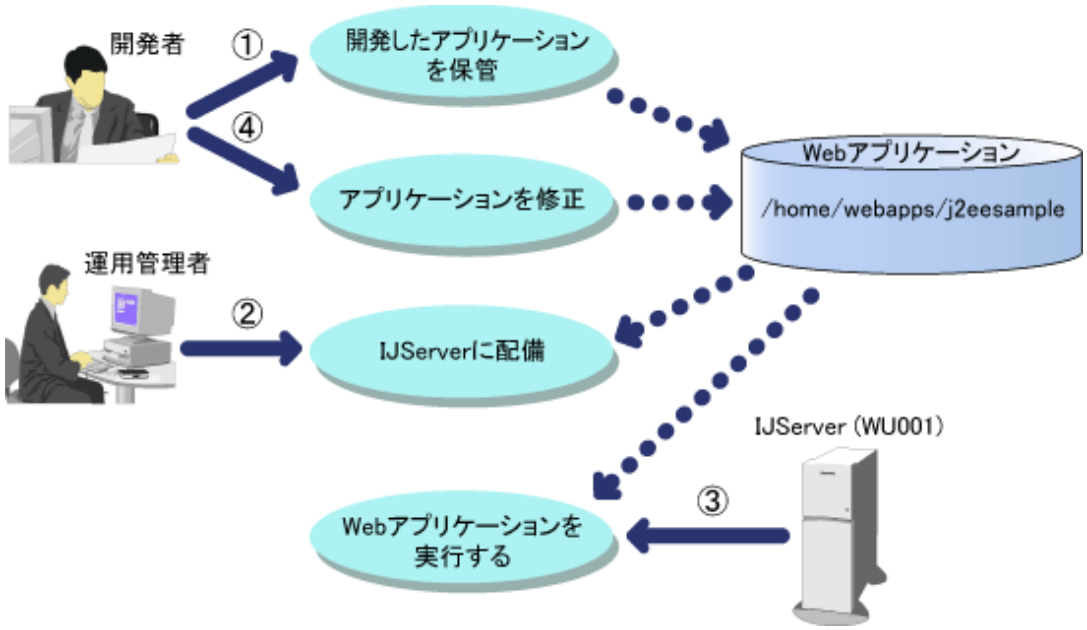


注意

- Webアプリケーションを展開したディレクトリ配下の資源にはIJServerの起動ユーザに対して必要な権限を与えてください。
  クラスファイル、JARファイル、JSP、静的リソース(htmlなど)にIJServerの起動ユーザに対する読み込み権限がない場合は、リソースの読み込みに失敗します。
- Webサービスを含むWebアプリケーションの場合は、本配備方法は使用できません。

1つのアプリケーションを1つのIJServerで実行する場合の運用方法

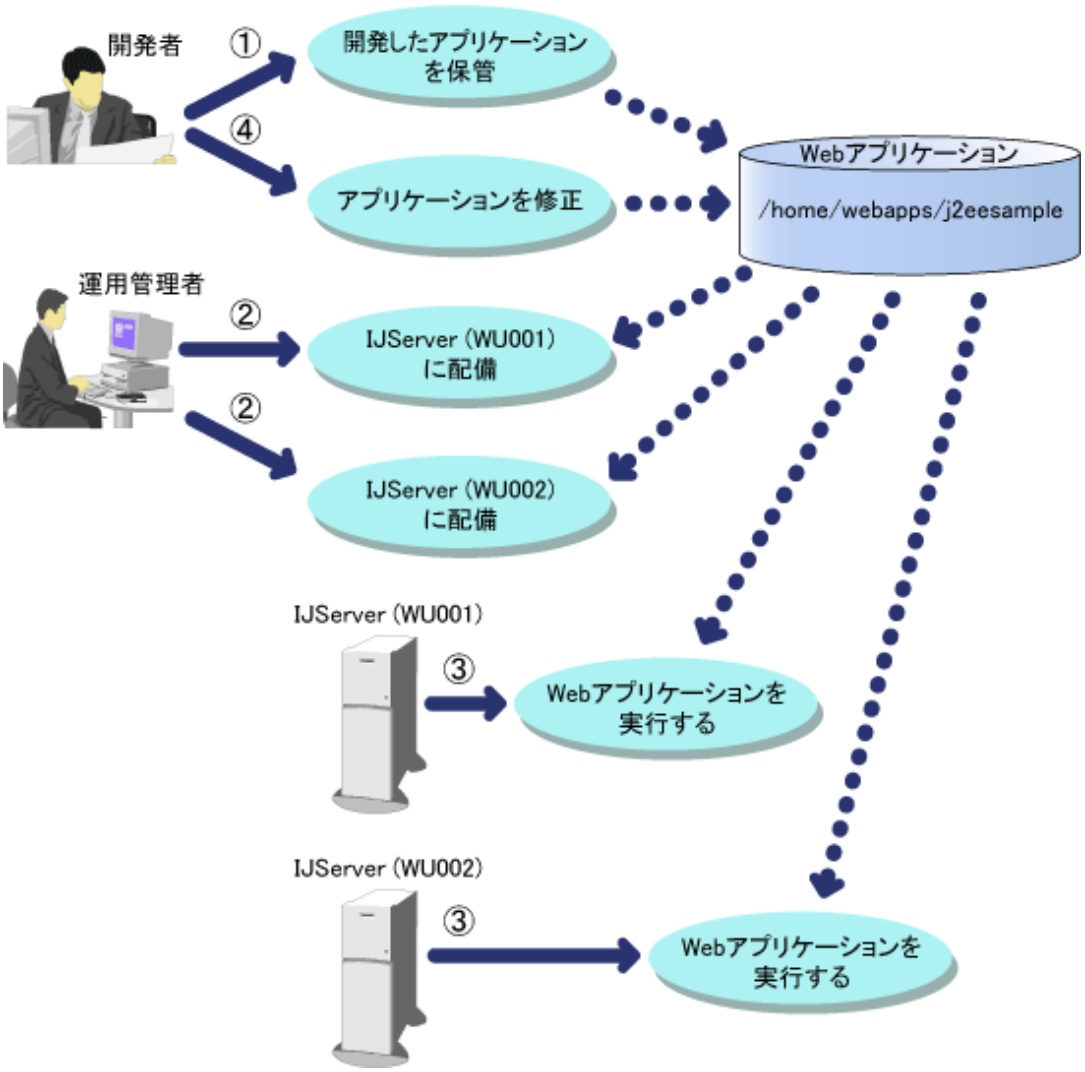
サーバ上の任意のディレクトリ配下に展開されたWebアプリケーションをIJServerディレクトリ配下にコピーすることなくIJServerで動作させることができます。



1. 開発したアプリケーションを保管
   アプリケーションの開発者は、開発したアプリケーションをサーバマシン上の任意のディレクトリに保管します。
2. IJServerに配備
   運用管理者はサーバマシン上の任意のディレクトリに保管されたWebアプリケーションをIJServerに配備します。
3. Webアプリケーションを実行する
   IJServerはサーバマシン上の任意のディレクトリに保管されたWebアプリケーションを実行します。
4. アプリケーションを修正
   アプリケーションの修正が発生した場合、アプリケーションの開発者はサーバマシン上の任意のディレクトリに保管されたWebアプリケーションの資源を置き換えることができます。
   クラスファイル、JARファイルを置き換えた場合はクラスのオートリロード、再活性、およびIJServerの再起動後に有効となります。
   JSPファイルを置き換えた場合は、JSPファイルのリロード、およびIJServerの再起動後に有効となります。
   静的ファイル(htmlなど)を置き換えた場合は、置き換えた時点で有効となります。

1つのアプリケーションを複数のIJServerで実行する場合の運用方法

以下のように1つのWebアプリケーションを複数のIJServerで実行することができます。
この場合は、1箇所を修正することで複数のIJServerに修正を反映することができます。



1. 開発したアプリケーションを保管
   アプリケーションの開発者は、開発したアプリケーションをサーバマシン上の任意のディレクトリに保管します。

2. IJServerに配備
   運用管理者はサーバマシン上の任意のディレクトリに保管されたWebアプリケーションを2つのIJServer(WU001とWU002)に配備します。

3. Webアプリケーションを実行する
   2つのIJServer(WU001とWU002)は、同じディレクトリに保管されたWebアプリケーションを実行します。
   注) この場合、複数のIJServerが同じWebアプリケーションを同時に実行するため、Webアプリケーション内の資源を書き換えるWebアプリケーションの場合は、資源を書き換える際には書き換える資源に対して排他制御を考慮する必要があります。

4. アプリケーションを修正
   アプリケーションの修正が発生した場合、アプリケーションの開発者はサーバマシン上の任意のディレクトリに保管されたWebアプリケーションの資源を置き換えることができます。
   この場合、置き換えた資源は2つのIJServer(WU001とWU002)に反映されます。
   クラスファイル、JARファイルを置き換えた場合はクラスのオートリロード、再活性、およびIJServerの再起動後に有効となります。
   JSPファイルを置き換えた場合は、JSPファイルのリロード、およびIJServerの再起動後に有効となります。
   静的ファイル(htmlなど)を置き換えた場合は、置き換えた時点で有効となります。

3.5.6 配備の事前設定

配備時に指定するWebアプリケーションの設定をWebモジュール内に定義する方法について説明します。

Webモジュール定義ファイル(interstage-web.xml)を配備ファイルのMETA-INFディレクトリに格納し配備を行うことで、Interstage管理コンソールの[システム] > [ワークユニット] > [IJServer] > [配備] > [詳細設定] > [Webアプリケーション設定]で行う配備時の設定項目を事前に設定しておくことができます。
また、“サーバ上の任意の位置で実行するWebアプリケーション”を配備する場合は、Webモジュール定義ファイルをWebアプリケーション保管先ディレクトリ配下のMETA-INFディレクトリに格納し配備を行うことで、配備時に指定するWebアプリケーションの設定項目を事前に設定しておくことができます。

注意

配備後にモジュールの配備先に格納されているWebモジュール定義ファイルの設定値を変更した場合、設定値は有効になりません。
設定値を変更する場合は、下記のいずれかの操作を行ってください。

- Interstage管理コンソールを使用して、当該モジュールの環境設定画面で設定値を入力し、[適用]操作を行ってください。
- 配備ファイルに格納されているWebモジュール定義ファイルの設定値を変更し、再配備を行ってください。
  なお、“サーバ上の任意の位置で実行するWebアプリケーションを配備する”を選択して配備している場合は、ディレクトリに格納されているWebモジュール定義ファイルの設定値を変更し、再配備を行ってください。

Webモジュール定義ファイル(interstage-web.xml)

Webモジュール定義ファイル(interstage-web.xml)について説明します。

- Webモジュール定義ファイルの記述形式
- Webモジュール定義ファイルのタグ一覧
- Webモジュール定義ファイルのタグ詳細
- Webモジュール定義ファイルの例

Webモジュール定義ファイルの記述形式

Webモジュール定義ファイルの記述形式はXML形式です。Webモジュール定義ファイルの記述形式を以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<interstage-web-app>
  <war>
    <context-root>context-root</context-root>
    <shared-context>shared-context</shared-context>
    <session>
      <cookie>cookie</cookie>
      <web-browser>web-browser</web-browser>
      <cookie-security>cookie-security</cookie-security>
    </session>
    <encoding>encoding</encoding>
    <authentication>
      <web-server>web-server</web-server>
    </authentication>
  </war>
</interstage-web-app>
```

注意

- 各タグの記載順序は、上記の記載順序に従ってください。
- 先頭の<?xml. . . >は、XML宣言を記述しているため、ファイルの先頭で必ず記述してください。また、マルチバイト文字を使用(コメントも含む)する場合は、エンコード形式(「encoding=」部分)に、“UTF-8”などの適切な値を指定してください。

- <interstage-web-app>、</interstage-web-app>は、XMLファイルの開始と終了を示すルートタグです。必ず指定してください。
- 設定する値の先頭、および末尾に指定した空白やタブ文字、改行文字などの制御文字は無視されます。
- 各項目のデフォルト値を使用する場合は、タグに空値を設定するのではなく、タグ自体を省略してください。

Webモジュール定義ファイルのタグ一覧

Webモジュール定義ファイルのタグ一覧を以下に示します。
また、“Interstage管理コンソール画面との対応”欄に、Interstage管理コンソールの[システム] > [ワークユニット] > [IJServer] > [配備] > [詳細設定]画面との対応を示します。

タグ名	タグの省略	複数の指定	Interstage管理コンソール画面との対応
war	○	×	Webアプリケーション設定
context-root	○	×	Webアプリケーション名
shared-context	○	×	コンテキストの共有
session	○	×	セッション
cookie	○	×	クッキーに設定する/クッキーに設定しない
web-browser	○	×	Webブラウザでセッションを保存する
cookie-security	○	×	クッキーにSecure属性を常に付加する
encoding	○	×	エンコーディング
authentication	○	×	認証
web-server	○	×	Webサーバの認証情報を使用する/しない

Webモジュール定義ファイルのタグ詳細

各タグに指定する値についての説明を行います。各項目の詳細については、Interstage管理コンソールのヘルプの“IJServer:配備”を参照してください。

context-root

Webアプリケーション名を指定します。64文字以内の値を指定してください。
デフォルト値については、Interstage管理コンソールのヘルプの[配備]を参照してください。
なお、ijsdeploymentコマンドの-cオプションでWebアプリケーション名を指定して配備を行った場合、本タグで指定した値ではなく、-cオプションで指定した値が有効になります。
ijsdeploymentコマンドについては、“リファレンスマニュアル(コマンド編)”の“J2EE運用コマンド”を参照してください。

shared-context

他のWebアプリケーションへのディスパッチを許可するかどうかを以下から選択します。大文字・小文字を区別しません。

- True(許可する)
- **False**(許可しない)・・・デフォルト値

cookie

クッキーを使用してセションを管理するかどうかを以下から選択します。大文字・小文字を区別しません。

- **True**(クッキーに設定する)・・・デフォルト値
- False(クッキーに設定しない)

web-browser

Webブラウザをいったん終了して再起動した時に、セションの有効時間内であれば再度接続可能とするかどうかを以下から選択します。大文字・小文字を区別しません。

ー True(Webブラウザでセションを保存する)

ー **False**(Webブラウザでセションを保存しない)・・・デフォルト値

※cookieタグにFalseを指定した場合は、本設定値は無視されます。

cookie-security

セション管理用クッキーにSecure属性を常に付加するかどうかを以下から選択します。大文字・小文字を区別しません。

ー Always(常に付与する)

ー **Auto**(接続時のメソッドで判断する)・・・デフォルト値

※cookieタグにFalseを指定した場合は、本設定値は無視されます。

encoding

Webクライアントからのリクエストボディの処理に使用するエンコーディングを指定します。
指定できる値は、“SJIS”、“EUC_JP”など、Javaでサポートされているエンコーディングです。

web-server

Webサーバの認証情報を使用するかどうかを以下から選択します。大文字・小文字を区別しません。

ー True(Webサーバの認証情報を使用する)

ー **False**(Webサーバの認証情報を使用しない)・・・デフォルト値

Webモジュール定義ファイルの例

Webアプリケーション名のみを指定する場合

```
<?xml version="1.0" encoding="UTF-8"?>
<interstage-web-app>
  <war>
    <context-root>sampleWarModule</context-root>
  </war>
</interstage-web-app>
```

すべて指定する場合

```
<?xml version="1.0" encoding="UTF-8"?>
<interstage-web-app>
  <war>
    <context-root>sampleWarModule</context-root>
    <shared-context>True</shared-context>
    <session>
      <cookie>True</cookie>
      <web-browser>True</web-browser>
      <cookie-security>Always</cookie-security>
    </session>
    <encoding>EUC_JP</encoding>
    <authentication>
      <web-server>True</web-server>
    </authentication>
  </war>
</interstage-web-app>
```

3.6 Servletサービスの運用準備

Servletサービスの運用準備について説明します。

Webサーバの環境設定

Servletサービスとの接続をサポートするWebサーバは以下のとおりです。

- Interstage HTTP Server
  環境設定については、“3.6.1 Interstage HTTP Serverの環境設定”を参照してください。
- Windows32/64
  Microsoft(R) Internet Information Services
  環境設定については、“3.6.2 Microsoft(R) Internet Information Services 8.0/8.5/10.0の環境設定 ”を参照してください。

IJServerとWebサーバを分離した運用

詳細は、“3.6.3 IJServerとWebサーバを分離して運用する場合の手順”を参照してください。

3.6.1 Interstage HTTP Serverの環境設定

ここでは、IJServerとInterstage HTTP Serverを連携させるための環境設定方法について説明します。

WebサーバからServletサービスへの通信はWebサーバコネクタが行います。
WebサーバコネクタはApache APIを使用したDSO(Dynamic Shared Objects)モジュールとしてWebサーバ上で動作するため、Webサーバの起動と停止に連動します。
IJServerとWebサーバをそれぞれ別のマシンに分離して運用する場合には、Webサーバコネクタの環境設定を行う必要があります。詳細は、“3.6.3 IJServerとWebサーバを分離して運用する場合の手順”を参照してください。

ポイント

Interstage HTTP Serverの環境定義ファイル(httpd.conf)に以下に示すWebサーバコネクタの定義情報が設定されています。Interstage管理コンソールを使用せずにInterstage HTTP Serverの環境定義ファイル(httpd.conf)を直接編集する場合には、本定義情報の削除、または編集をしないでください。

Windows32/64

```
LoadModule ihs2_redirector2_module "C:/Interstage/F3FMjs5/gateway/ihs2/mod_ihs2_redirector2.so"
```

Solaris64 Linux32/64

```
LoadModule ihs2_redirector2_module "/opt/FJSVjs5/gateway/ihs2/mod_ihs2_redirector2.so"
```

3.6.2 Microsoft(R) Internet Information Services 8.0/8.5/10.0の環境設定 Windows32/64

ここでは、IJServerとMicrosoft(R) Internet Information Services 8.0/8.5/10.0を連携させるための環境設定方法について説明します。

Microsoft(R) Internet Information ServicesのWebサーバコネクタはISAPI APIを使用した組み込みISAPIフィルターとISAPIエクステンションとしてWebサーバ上で動作するため、Webサーバの起動と停止に連動します。
IJServerとWebサーバをそれぞれ別のマシンに分離して運用する場合には、Webサーバコネクタの環境設定を行う必要があります。詳細は、“3.6.3 IJServerとWebサーバを分離して運用する場合の手順”を参照してください。

ポイント

Microsoft(R) Internet Information ServicesのWebサーバコネクタでは、Webサーバ名が“FJapache”のInterstage HTTP ServerのWebサーバコネクタの設定を参照することで同様の動作を実施しているため、Microsoft(R) Internet Information ServicesのWebサーバコネクタを使用する場合でもWebサーバ名が“FJapache”のInterstage HTTP Serverを使用する際と同様に、Interstage HTTP Server とWebサーバコネクタの設定をInterstage管理コンソールから行う必要があります。

注意

同一マシン上において、Interstage HTTP ServerとMicrosoft(R) Internet Information Servicesは、それぞれのWebサーバに異なるポート番号を設定することにより、共存することは可能ですが、Webサーバ名が“FJapache”のInterstage HTTP ServerのWebサーバコネクタを同時に利用することはできません。

また、同一マシン上において、Microsoft(R) Internet Information ServicesでWebサーバコネクタとJK2コネクタを同時に利用することはできません。

InterstageとMicrosoft(R) Internet Information Servicesの連携は、以下の手順で行います。

1. Microsoft(R) Internet Information ServicesとInterstageのインストール
2. Interstage HTTP Serverの自動起動の抑止
3. Microsoft(R) Internet Information Servicesの環境設定
4. Interstageの環境設定

1.Microsoft(R) Internet Information ServicesとInterstageのインストール

サーバマシンに、Microsoft(R) Internet Information ServicesとInterstageをインストールします。

ポイント

Microsoft(R) Internet Information Servicesを使用する場合は、Interstage HTTP Serverを必ずインストールしてください。Interstage HTTP Serverをインストールしないと、Microsoft(R) Internet Information ServicesとInterstageを連携させることができません。
また、Interstage HTTP Serverをインストール後、Webサーバ名が“FJapache”のInterstage HTTP Serverを削除した場合、Microsoft(R) Internet Information Servicesを連携させることができません。この場合、“FJapache”の名前でInterstage HTTP Serverを作成し直す必要があります。

2.Interstage HTTP Serverの自動起動の抑止

Interstage HTTP Serverの自動起動を抑止するための設定を行います。

1. Interstage HTTP Serverの停止
   Interstage管理コンソールの[サービス]>[Webサーバ]>[FJapache]>[状態]タブで、[停止]ボタンを押下してInterstage HTTP Serverを停止します。
2. Interstage HTTP Serverの自動起動の抑止
   Interstage HTTP Serverの自動起動を抑止します。詳細は、“Interstage HTTP Server運用ガイド”の“運用・保守”の“自動起動の設定”を参照してください。

注意

Interstage HTTP ServerとMicrosoft(R) Internet Information Servicesを共存させる場合、Interstageの再起動を行った際には、Webサーバ名が“FJapache”のInterstage HTTP Server以外のWebサーバを個別に起動してください。

3.Microsoft(R) Internet Information Servicesの環境設定

Microsoft(R) Internet Information Services上でWebサーバコネクタを動作させるための設定を行います。以降の操作は、Administrator権限でログインした状態で操作を行ってください。

注意

Microsoft(R) Internet Information Servicesの環境設定は、バックアップリストアおよび移出移入の対象となりません。

ISAPI 拡張とISAPI フィルターのインストール

サーバーマネージャーの[役割と機能の追加]の[サーバーの役割]で[Webサーバー (IIS)]を選択します。
次に[Webサーバーの役割(IIS)]の[役割サービス]で[アプリケーション開発]配下にある“ISAPI フィルター”と“ISAPI拡張”を選択してインストールします。

サービスの停止

- Microsoft(R) Internet Information Services 8.0/8.5の場合

  サーバーマネージャーの[IIS] > サービスで“World Wide Web Publishing Service”および“Windows Process Activation Service”を停止します。

- Microsoft(R) Internet Information Services 10.0の場合

  サーバーマネージャーの[IIS] > サービスで“World Wide Web 発行サービス”および“Windows プロセス アクティブ化サービス”を停止します。

ISAPIフィルターの追加

以下の手順でISAPIフィルターを追加します。

1. インターネットインフォメーションサービス(IIS)マネージャーの[サイト] > “サイト名” > “ISAPIフィルター”をダブルクリックしてISAPIフィルター画面を開きます。
2. 操作ペインの“追加”をクリックして、表示された“ISAPI フィルターの追加”ダイアログボックスで以下の値を入力して[OK]ボタンをクリックします。

フィルタ名	F3FMjs5 (任意の値)
実行可能ファイル	C:¥Interstage¥F3FMjs5¥gateway¥isapi¥isapi_redirector2.dll

アプリケーションの追加

以下の手順でアプリケーションを追加します。

1. インターネットインフォメーションサービス(IIS)マネージャーの[サイト] > “サイト名”を右クリックして“アプリケーションの追加”をクリックします。
2. 表示された“アプリケーションの追加”ダイアログボックスで以下の値を入力して[OK]ボタンをクリックします。

エイリアス	F3FMjs5
アプリケーション プール	DefaultAppPool または任意のアプリケーション プール
物理パス	C:¥Interstage¥F3FMjs5¥gateway¥isapi

注意

Webサーバコネクタは、ISAPIフィルタでURLの書き換え処理を行っています。
また、Microsoft(R) Internet Information Servicesでは、元URLと書き換え後のURLの両方が同じアプリケーションプール内で実行されるように設定する必要があります。
そのため、以下のいずれかの設定を行ってください。

- デフォルトのサイト(Default Web Site)でデフォルトのアプリケーション プール(DefaultAppPool)を使用する場合
  デフォルトのサイト(Default Web Site)と"F3FMjs5"に、デフォルトのアプリケーション プール(DefaultAppPool)を設定してください。

  設定例:デフォルトのサイト(Default Web Site)でデフォルトのアプリケーション プール(DefaultAppPool)を使用する場合

```
インターネットインフォメーションサービス（IIS）マネージャー
 ├[アプリケーション プール]
 |   └[DefaultAppPool]
 └[サイト]
     └[Default Web Site]：DefaultAppPoolを設定
         └[F3FMjs5]：DefaultAppPoolを設定
```

- デフォルトのサイト(Default Web Site)で任意のアプリケーション プールを使用する場合
  デフォルトのサイト(Default Web Site)と"F3FMjs5"に、任意のアプリケーション プールを設定してください。

  設定例:デフォルトのサイト(Default Web Site)で任意のアプリケーション プール(IASPool)を使用する場合

```
インターネットインフォメーションサービス（IIS）マネージャー
 ├[アプリケーション プール]
 |   ├[DefaultAppPool]
 |   └[IASPool]
 └[サイト]
     └[Default Web Site]：IASPoolを設定
         └[F3FMjs5]：IASPoolを設定
```

- 任意のサイト(IAS Web Site)で任意のアプリケーション プールを使用する場合
  任意のサイト(IAS Web Site)と"F3FMjs5"に、任意のアプリケーション プールを設定してください。

  設定例:任意のサイト(IAS Web Site)で任意のアプリケーション プール(IASPool)を使用する場合

```
インターネットインフォメーションサービス（IIS）マネージャー
 ├[アプリケーション プール]
 |   ├[DefaultAppPool]
 |   └[IASPool]
 └[サイト]
     └[IAS Web Site]：IASPoolを設定
         └[F3FMjs5]：IASPoolを設定
```

3. インターネットインフォメーションサービス(IIS)マネージャーの[サイト]＞“サイト名”＞[F3FMjs5]＞“ハンドラー マッピング”をダブルクリックしてハンドラー マッピング画面を開きます。
4. “ISAPI-dll”を選択して、操作ペインの“編集”をクリックします。
5. 表示された“スクリプト マップの編集”ダイアログボックスで以下の値を入力して[OK]ボタンをクリックします。

要求パス	*.dll
モジュール	IsapiModule
実行可能ファイル	C:¥Interstage¥F3FMjs5¥gateway¥isapi¥isapi_redirector2.dll

6. 続いて表示されるダイアログボックスで[はい]をクリックします。
7. “ISAPI-dll”を再度選択して、操作ペインの“機能のアクセス許可の編集”をクリックします。
8. 表示された“機能のアクセス許可の編集”ダイアログボックスですべてのチェックボックスを選択状態にして[OK]ボタンをクリックします。

ワーカー プロセスの設定

以下の手順でワーカー プロセスの設定をします。

1. インターネットインフォメーションサービス(IIS)マネージャーの[アプリケーション プール]＞“上記のアプリケーションの追加で選択したアプリケーション プール名”を選択して操作ペインの“リサイクルの設定”をクリックします。
2. 表示された“アプリケーション プールのリサイクル設定の編集”ダイアログボックスですべてのチェックボックスの選択を解除して[次へ]をクリックします。次の画面では何も選択せずに[終了]ボタンをクリックします。
3. 上記の1.で選択したアプリケーション プールを再度選択して、操作ペインの“詳細設定”をクリックします。
4. 表示された“詳細設定”ダイアログボックスで以下の値を入力して[OK]ボタンをクリックします。

ワーカー プロセスの最大数	1
32 ビット アプリケーションの有効化 (注)	True

注) Windows Server(R) x64 Editions(32ビット互換)でInterstage Application Serverを運用する場合にのみ設定します。

サービスの開始

- Microsoft(R) Internet Information Services 8.0/8.5の場合

  サーバーマネージャーの[IIS] > サービスで“World Wide Web Publishing Service”および“Windows Process Activation Service”を開始します。

- Microsoft(R) Internet Information Services 10.0の場合

  サーバーマネージャーの[IIS]＞サービスで“World Wide Web 発行サービス”および“Windows プロセスアクティブ化サービス”を開始します。

4.Interstageの環境設定

Interstageの環境設定(ワークユニットの作成、Webアプリケーションの配備など)は、Interstage管理コンソールで、Interstage HTTP Serverを使用する場合と同様の操作で行います。
ただし、Interstage HTTP Serverを使用する場合と、Microsoft(R) Internet Information Servicesを使用する場合とでは、以下の差異があります。

- Microsoft(R) Internet Information Servicesを使用する場合は、同一マシン上では、複数のWebサイトに対して、Webサーバコネクタを設定して使用することはできません。また、Interstage管理コンソールで、Interstage HTTP Serverの設定でWebサーバのバーチャルホストを設定して使用することはできません。
- Webサーバの設定は、Microsoft(R) Internet Information Servicesが提供するインターネットインフォメーションサービス（IIS）マネージャーで行います。

注意

- WebサーバコネクタとServletコンテナ間でSSL通信を行う場合には、Microsoft(R) Internet Information Servicesを実行するユーザがInterstage証明書環境へのアクセスを許可する、Administrators権限を所有している必要があります。一般権限のユーザでSSL機能を使用する場合、エクスプローラでInterstage証明書環境のディレクトリを選択し、「プロパティ」メニューの「セキュリティ」タブの画面で、ユーザまたはグループを追加することでアクセス権限を追加できます。追加したユーザまたはグループについては「フルコントロール」を設定するようにしてください。
  Interstage証明書環境へのアクセス権限の詳細については、“セキュリティシステム運用ガイド”の“Interstage証明書環境の構築と利用”の“Interstage証明書環境のアクセス権限の設定”を参照してください。
- Webサーバコネクタのログ出力ディレクトリ、およびWebサーバコネクタのログファイルに対して、一般ユーザ権限でフルコントロール可能なアクセス許可を付与してください。または、Microsoft(R) Internet Information Servicesのワーカープロセスグループおよび匿名ユーザーアカウントに、フルコントロール可能なアクセス許可を付与してください。
  ワーカープロセスグループは"IIS_IUSRS"、匿名ユーザーアカウントはデフォルトでは"IUSR"となります。
  ワーカープロセスグループ、および匿名ユーザーアカウントについては、Microsoft(R) Internet Information Servicesのマニュアルを参照してください。
- ワークユニットの作成時、Webサーバは必ず“FJapache”を選択してください。Webサーバに“FJapache”を選択しない、またはワークユニットの環境設定で、Webサーバを“FJapache”から変更した場合は、Microsoft(R) Internet Information ServicesのWebサーバコネクタを使用することはできません。

3.6.3 IJServerとWebサーバを分離して運用する場合の手順

IJServerとWebサーバをそれぞれ別のサーバマシンに分離して運用することができます。
この機能により、以下のシステム構築が可能です。

- セキュリティの向上
  IJServerをDMZ(DeMilitarized Zone:非武装地帯)で運用させたくない場合、WebサーバをDMZのマシンで運用して、IJServerはFirewallを越えたイントラネット上のマシンで運用します。
- 負荷分散
  マシンのCPU負荷が大きい場合、IJServerとWebサーバを分離したシステムを構築し、負荷を分散します。
  なお、IPCOMを使用することでIJServerが稼動するマシンの稼動状態やCPU負荷状態を監視した負荷分散が行える

ようになります。IPCOMを使用する場合には、IPCOMの分散ポートの設定で、常設コネクション数を確保(Webアクセラレーション機能)してください。常設コネクション数は、Servletコンテナの同時処理数と同じにしてください。
IPCOMの設定の詳細については、IPCOMのマニュアルを参照してください。

IJServerとWebサーバ間はhttpまたはhttpsで接続します。
IJServerとWebサーバを別のサーバマシンに分離する場合の運用手順について、同一のサーバマシン上に運用する場合との違いを説明します。

- 設定方法
- ServletおよびJSPでセションを使用する場合のリクエストの振り分けについて
- 運用準備の具体例

ポイント

- IJServerとWebサーバでは、同一バージョン・レベルのInterstageを使用してください。
- IJServerとWebサーバを別のサーバマシンに分離して運用する場合には、最初に以下を設定する必要があります。この設定をしないと、Webサーバコネクタを操作することができません。
  - IJServer用マシンとWebサーバ用マシンそれぞれのInterstage管理コンソールで、[システム] > [環境設定]タブ > [Servletサービス詳細設定] > [Webサーバとワークユニットを同一のマシンで運用する]で[運用しない]を選択します。
    なお、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。
- Servletコンテナに接続するWebサーバのIPアドレスを制限する場合、かつ、Webサーバに複数のIPアドレスが設定されている場合、Interstage管理コンソールで[ワークユニット] > “ワークユニット名” > [環境定義]タブ > [詳細設定]で、[Webサーバコネクタ(コネクタ)設定]の[要求を受け付けるWebサーバのIPアドレス]にWebサーバに設定されているすべてのIPアドレスを指定してください。
  なお、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。
- IJServerとWebサーバを同一のサーバマシン上で運用している場合でも、別のサーバマシン上のIJServerやWebサーバとも連携を行う場合にはIJServerとWebサーバを別のサーバマシンに分離する場合の運用手順に従ってIJServer用とWebサーバ用の設定をそれぞれ行う必要があります。

設定方法

IJServer用のマシンとWebサーバ用のマシンで以下の設定を行うことで、IJServerとWebサーバを分離して運用することができます。

1. IJServerの作成
   IJServer用マシンのInterstage管理コンソール、またはisj2eeadminコマンドを使用して、IJServerを作成します。
2. 接続先IJServerの設定
   Webサーバ用マシンのInterstage管理コンソール、またはisj2eeadminコマンドを使用して、「IJServerの作成」で指定した情報をもとにWebサーバコネクタの接続先情報を設定します。
3. Webアプリケーションの配備
   IJServer用マシンのInterstage管理コンソールから、「IJServerの作成」で作成したIJServerにWebアプリケーションを配備します。
4. 接続先Webアプリケーションの設定
   Webサーバ用マシンのInterstage管理コンソール、またはisj2eeadminコマンドを使用して、「接続先IJServerの設定」で作成した接続先IJServerの情報に「Webアプリケーションの配備」で配備したWebアプリケーション名を設定します。

IJServer用マシンで以下の操作を行った場合には、上記操作と同じようにWebサーバ用マシンにIJServer用マシンで行った操作内容を反映する必要があります。
以下では、Interstage管理コンソールを使用する場合について説明していますが、isj2eeadminコマンドを使用する場合も同様の操作が必要です。

- [ワークユニット] > “ワークユニット名” > [環境定義]タブ > [詳細設定]で、[Webサーバコネクタ(コネクタ)設定]または[Servletコンテナ設定]の以下の項目を変更した場合

  ー [Webサーバコネクタ(コネクタ)設定]

    - コネクタとServletコンテナ間のSSLの使用
    - コネクタとServletコンテナ間のSSL定義

  ー [Servletコンテナ設定]

    - ServletコンテナのIPアドレス
    - ポート番号

  Webサーバ用のマシンのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ]を指定します。次に右フレームからIJServer用マシンで変更を行ったワークユニット名をクリックして、IJServer用マシンで変更を行った値を反映します。

- IJServerの削除をした場合
  Webサーバ用のマシンのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [一覧タブ]から削除したワークユニット名をチェックして削除します。

- Webアプリケーションを配備解除した場合
  Webサーバ用のマシンのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ]を指定します。次に右フレームからIJServer用マシンで配備解除を行ったワークユニット名をクリックして配備解除したWebアプリケーション名を削除します。

ServletおよびJSPでセションを使用する場合のリクエストの振り分けについて

ServletおよびJSPでセションを使用する場合、クライアントからのリクエストはWebサーバコネクタの振り分け制御によってセションを作成したServletコンテナにリクエストが振り分けられます。
WebサーバコネクタはServletコンテナ識別子によりセションを作成したServletコンテナを識別し、セションを作成したServletコンテナへリクエストを振り分けます。
Servletコンテナ識別子はWebサーバとワークユニットを同一マシンで運用しない場合には、Interstage管理コンソールから定義を行うことができます。

- [サービス] > [Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]
- [サービス] > [Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > “ワークユニット名” > [環境設定]

定義方法の詳細についてはInterstage管理コンソールのヘルプを参照してください。

なお、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

Webサーバとワークユニットを同一マシンで運用する場合には、Servletコンテナ識別子は自動的に採番され内部的に管理されますので、Servletコンテナ識別子を意識する必要はありません。

運用準備の具体例

以下のマシン構成の場合について、具体的な運用手順例を説明します。

- 1台のIJServer用マシンと1台のWebサーバ用マシン(サーバマシン2台構成)
- 1台のIJServer用マシンと2台のWebサーバ用マシン(サーバマシン3台構成)
- 2台のIJServer用マシンと1台のWebサーバ用マシン(サーバマシン3台構成)
- 2台のIJServer用マシン、2台のWebサーバ用マシン、1台の負荷分散マシン(サーバマシン5台構成)

IJServer用マシンを3台以上で構成する場合は、“2台のIJServer用マシンと1台のWebサーバ用マシン(サーバマシン3台構成)”の手順1.2.3.を参考にして、IJServer用マシンにワークユニットを作成しWebサーバ用マシンに接続先情報を設定してください。

Webサーバ用マシンを3台以上で構成する場合は、“1台のIJServer用マシンと2台のWebサーバ用マシン(サーバマシン3台構成)”の手順2.3.を参考にして、Webサーバ用マシンに接続先情報を設定してください。

以下では、Interstage管理コンソールを使用する場合の手順を説明していますが、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

1台のIJServer用マシンと1台のWebサーバ用マシン(サーバマシン2台構成)

条件

サーバマシンA(IPアドレス:192.0.2.1)	Webサーバを運用
サーバマシンB(IPアドレス:192.0.2.2)	IJServerを運用

設定手順

1. IJServer用のサーバマシンBの設定
   サーバマシンBのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容のワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.2
ポート番号	9000

2. Webサーバ用のサーバマシンAの設定
   サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]タブを選択し、以下の設定内容の接続先情報を作成します。

項目名	設定値例
ワークユニット名	MyIJServer
ServletコンテナのIPアドレス:ポート番号	192.0.2.2:9000

3. IJServer用のサーバマシンBへWebアプリケーションを配備
   サーバマシンBのInterstage管理コンソールで、[ワークユニット] > “ワークユニット名(例:MyIJServer)” > [配備]タブを選択し、Webアプリケーションを配備します。

4. Webサーバ用のサーバマシンAへWebアプリケーション名を追加
   サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ]を指定します。
   次に右フレームで3.の操作で配備を行ったワークユニット名をクリックして配備したWebアプリケーション名を追加します。

1台のIJServer用マシンと2台のWebサーバ用マシン(サーバマシン3台構成)

条件

サーバマシンA(IPアドレス:192.0.2.1)	Webサーバを運用
サーバマシンB(IPアドレス:192.0.2.2)	Webサーバを運用
サーバマシンC(IPアドレス:192.0.2.3)	IJServerを運用

設定手順

1. IJServer用のサーバマシンCの設定
   サーバマシンCのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容のワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.1 192.0.2.2
ServletコンテナのIPアドレス	192.0.2.3
ポート番号	9000

2. Webサーバ用のサーバマシンAの設定
   サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]タブを選択し、以下の設定内容の接続先情報を作成します。

項目名	設定値例
ワークユニット名	MyIJServer
ServletコンテナのIPアドレス:ポート番号	192.0.2.3:9000

3. Webサーバ用のサーバマシンBの設定
   サーバマシンBのInterstage管理コンソールで2.と同じ操作を行います。
4. IJServer用のサーバマシンCへWebアプリケーションを配備
   サーバマシンCのInterstage管理コンソールで、[ワークユニット] > “ワークユニット名(例:MyIJServer)” > [配備]タブを選択し、Webアプリケーションを配備します。
5. Webサーバ用のサーバマシンAへWebアプリケーション名を追加
   サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ]を指定します。
   次に右フレームで4.の操作で配備を行ったワークユニット名をクリックして配備したWebアプリケーション名を追加します。
6. Webサーバ用のサーバマシンBへWebアプリケーション名を追加
   サーバマシンBのInterstage管理コンソールで5.と同じ操作を行います。

2台のIJServer用マシンと1台のWebサーバ用マシン(サーバマシン3台構成)

条件

サーバマシンA(IPアドレス:192.0.2.1)	Webサーバを運用
サーバマシンB(IPアドレス:192.0.2.2)	IJServerを運用
サーバマシンC(IPアドレス:192.0.2.3)	IJServerを運用

設定手順

1. IJServer用のサーバマシンBの設定
   サーバマシンBのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容のワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.2
ポート番号	9000

2. IJServer用のサーバマシンCの設定
   サーバマシンCのInterstage管理コンソールで1.と同じ操作を行います。ただし、Servletコンテナの IPアドレスには192.0.2.3を指定します。

   ー サーバマシンBとCに同じWebアプリケーションを配備する場合
   ワークユニット名は、サーバマシンBと同じ名前を設定してください。ワークユニットの環境設定も同じにする必要があります。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.3
ポート番号	9000

   ー サーバマシンBとCに異なるWebアプリケーションを配備する場合
   ワークユニット名は、サーバマシンBとは異なる名前を設定してください。

項目名	設定値例
ワークユニット名	MyIJServer_2
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.3
ポート番号	9000

3. Webサーバ用のサーバマシンAの設定
   サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]タブを選択し、接続先情報を作成します。

   ー サーバマシンBとCに同じWebアプリケーションを配備する場合
   サーバマシンBとCに設定したワークユニット名の接続先情報を設定します。

項目名	設定値例
ワークユニット名	MyIJServer
ServletコンテナのIPアドレス:ポート番号	192.0.2.2:9000 192.0.2.3:9000

   ー サーバマシンBとCに異なるWebアプリケーションを配備する場合
   サーバマシンBとCの接続先情報を個々に作成します。ワークユニット名は、接続先のサーバマシンと同じ名前を設定してください。

   <サーバマシンBの接続先情報>

項目名	設定値例
ワークユニット名	MyIJServer
ServletコンテナのIPアドレス:ポート番号	192.0.2.2:9000

   <サーバマシンCの接続先情報>

項目名	設定値例
ワークユニット名	MyIJServer_2
ServletコンテナのIPアドレス:ポート番号	192.0.2.3:9000

4. IJServer用のサーバマシンBにWebアプリケーションを配備
   サーバマシンBのInterstage管理コンソールで、[ワークユニット] > “ワークユニット名(例:MyIJServer)” > [配備]タブを選択し、Webアプリケーションを配備します。

5. IJServer用のサーバマシンCにWebアプリケーションを配備
   サーバマシンCのInterstage管理コンソールで4.と同じ操作を行います。

6. Webサーバ用のサーバマシンAにWebアプリケーション名を追加
   サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ]を指定します。
   次に右フレームで4.の操作で配備を行ったワークユニット名をクリックして配備したWebアプリケーション名を追加します。

2台のIJServer用マシン、2台のWebサーバ用マシン、1台の負荷分散マシン(サーバマシン5台構成)

条件

サーバマシンA(IPアドレス:192.0.2.1)	負荷分散装置を運用
サーバマシンB(IPアドレス:192.0.2.2)	Webサーバを運用
サーバマシンC(IPアドレス:192.0.2.3)	Webサーバを運用
サーバマシンD(IPアドレス:192.0.2.4)	IJServerを運用
サーバマシンE(IPアドレス:192.0.2.5)	IJServerを運用

設定手順

1. IJServer用のサーバマシンDの設定
   サーバマシンDのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容のワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.2 192.0.2.3
ServletコンテナのIPアドレス	192.0.2.4
ポート番号	9000

2. IJServer用のサーバマシンEの設定
   サーバマシンEのInterstage管理コンソールで1.と同じ操作を行います。ただし、ServletコンテナのIPアドレスには192.0.2.5を指定します。

3. Webサーバ用のサーバマシンBの設定
   サーバマシンBのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]タブを選択し、以下の設定内容の接続先情報を作成します。

項目名	設定値例
ワークユニット名	MyIJServer
ServletコンテナのIPアドレス:ポート番号	192.0.2.4:9000 192.0.2.5:9000

4. Webサーバ用のサーバマシンCの設定
   サーバマシンCのInterstage管理コンソールで3.と同じ操作を行います。
   ServletコンテナのIPアドレス:ポート番号はサーバマシンBで設定した値と同じ順番で定義してください。順番を変更した場合はセションが正常に引き継がれない可能性があります。

5. 負荷分散装置用のサーバマシンAの設定
   サーバマシンAの負荷分散装置の設定を行います。
   負荷分散装置の設定方法については負荷分散装置のマニュアルを参照してください。

6. IJServer用のサーバマシンDにWebアプリケーションを配備
   サーバマシンDのInterstage管理コンソールで、[ワークユニット] > “ワークユニット名(例:MyIJServer)” > [配備]タブを選択し、Webアプリケーションを配備します。

7. IJServer用のサーバマシンEにWebアプリケーションを配備
   サーバマシンEのInterstage管理コンソールで6.と同じ操作を行います。

8. Webサーバ用のサーバマシンBにWebアプリケーション名を追加
   サーバマシンBのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ]を指定します。
   次に右フレームで6.の操作で配備を行ったワークユニット名をクリックして配備したWebアプリケーション名を追加します。

9. Webサーバ用のサーバマシンCにWebアプリケーション名を追加
   サーバマシンCのInterstage管理コンソールで8.と同じ操作を行います。

3.7 Webサーバコネクタにおけるリクエストの振り分け制御

Webサーバコネクタにおけるリクエストの振り分け制御について、コマンドによる制御と故障監視による制御について説明します。

3.7.1 コマンドによる振り分け操作と状態表示

Webサーバコネクタでは、各IJServerワークユニットに対しリクエスト振り分け制御を行っています。
以下のコマンドを使用することにより、WebサーバコネクタにおけるIJServerワークユニットのリクエスト振り分け制御状態(振り分け対象にする／しない)の変更や、現在のリクエスト振り分け制御状態の表示ができます。
例えばWebアプリケーションの定期保守、ネットワーク障害やマシンのハード障害などの場合に当該IJServerワークユニット(または当該マシン)を一時的にリクエスト振り分け対象から外し、保守完了後や障害対処後に振り分け対象に戻します。この操作により、安定かつ継続的なシステムの運用が可能になります。

- 振り分け操作コマンド(ijsdispatchcont)
  IJServerワークユニットをリクエストの振り分け対象にする／しないの操作を、IPアドレスまたはIPアドレス:ポート番号で指定して行います。
  振り分け操作コマンドで操作した状態は、Webサーバの再起動後や異常停止後でも保持されます。振り分けの状態を変更したい場合は、再度振り分け操作コマンドを実行します。

- 振り分け状態表示コマンド(ijsprintdispatchcont)
  IJServerワークユニットへの振り分け状態を“IPアドレス:ポート番号 ワークユニット名”の一覧で表示します。

振り分け操作コマンドで指定するIJServerワークユニットのIPアドレスまたはIPアドレス:ポート番号は、事前に振り分け状態表示コマンドを使用して確認します。振り分け状態表示コマンドで表示されたIPアドレスまたはIPアドレス:ポート番号のみで操作可能です。
各コマンドの詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

ポイント

Webサーバコネクタから各IJServerワークユニットへのリクエスト振り分けは処理中リクエストが最も少ないところに振り分けられます。ただし、Webアプリケーションでセション管理を使用している場合には、セションを生成したIJServerワークユニットと同じIJServerワークユニットに以降のリクエストも振り分けられます。

以下に運用パターンの例を示します。サーバマシンの構成とコマンドパラメータの指定方法によってさまざまな運用パターンがあります。

- パターン1:マシン単位の振り分け操作
- パターン2:IJServerワークユニット単位の振り分け操作(1)
- パターン3:IJServerワークユニット単位の振り分け操作(2)
- パターン4:IJServerワークユニットの接続抑止

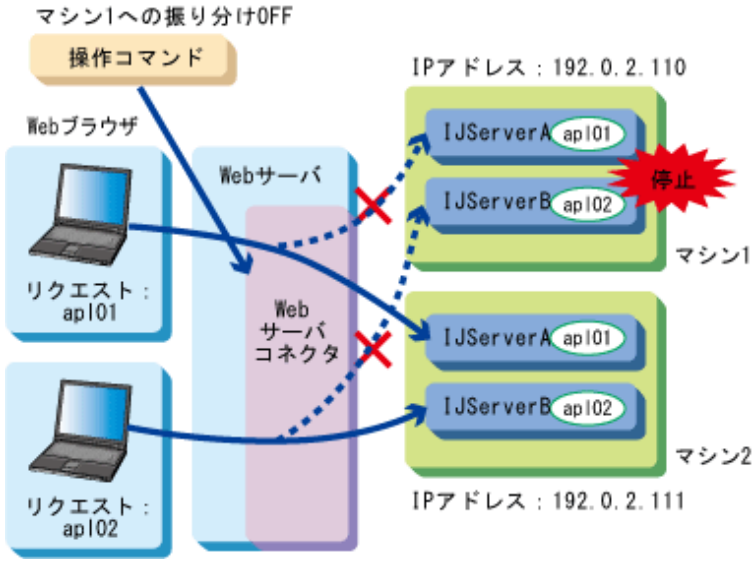
パターン1:マシン単位の振り分け操作

マシンの保守を行う場合やマシンのハード障害が発生した場合などにマシンへの振り分けを制御するには、以下のように運用します。

IPアドレス192.0.2.110のマシン(マシン1)とIPアドレス192.0.2.111のマシン(マシン2)をそれぞれ2つのIJServerワークユニット(IJServerA、IJServerB)で構成しています。
IJServerAにはapl01、IJServerBにはapl02のWebアプリケーションを配備しています。
マシン1を停止するため、IPアドレス192.0.2.110を振り分け対象からはずして運用します。

```
ijsdispatchcont OFF 192.0.2.110
```

負荷分散のためマシン1とマシン2に振り分けられていたapl01、apl02は共にマシン2のみでの運用になります。



マシン1復旧時には、以下のコマンドで振り分け対象に戻します。

```
ijsdispatchcont ON 192.0.2.110
```

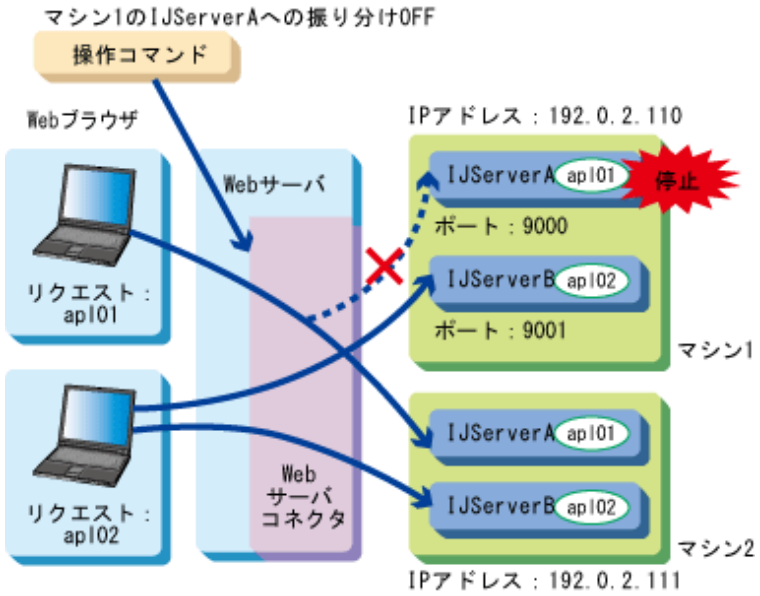
パターン2:IJServerワークユニット単位の振り分け操作(1)

Webアプリケーションの保守を行う場合などにIJServerワークユニットへの振り分けを制御するには、以下のように運用します。

パターン1同様、IPアドレス192.0.2.110のマシン(マシン1)とIPアドレス192.0.2.111のマシン(マシン2)をそれぞれ2つのIJServerワークユニット(IJServerA、IJServerB)で構成しています。
IJServerAにはapl01、IJServerBにはapl02のWebアプリケーションを配備しています。
マシン1のIJServerAを停止するため、振り分け対象からはずして運用します。

```
ijsdispatchcont OFF 192.0.2.110:9000
```

負荷分散のためマシン1とマシン2に振り分けられていたapl01はマシン2のみでの運用となります。apl02は変わらずマシン1とマシン2に振り分けられます。



マシン1のIJServerA復旧時には、以下のコマンドで振り分け対象に戻します。

```
ijsdispatchcont ON 192.0.2.110:9000
```

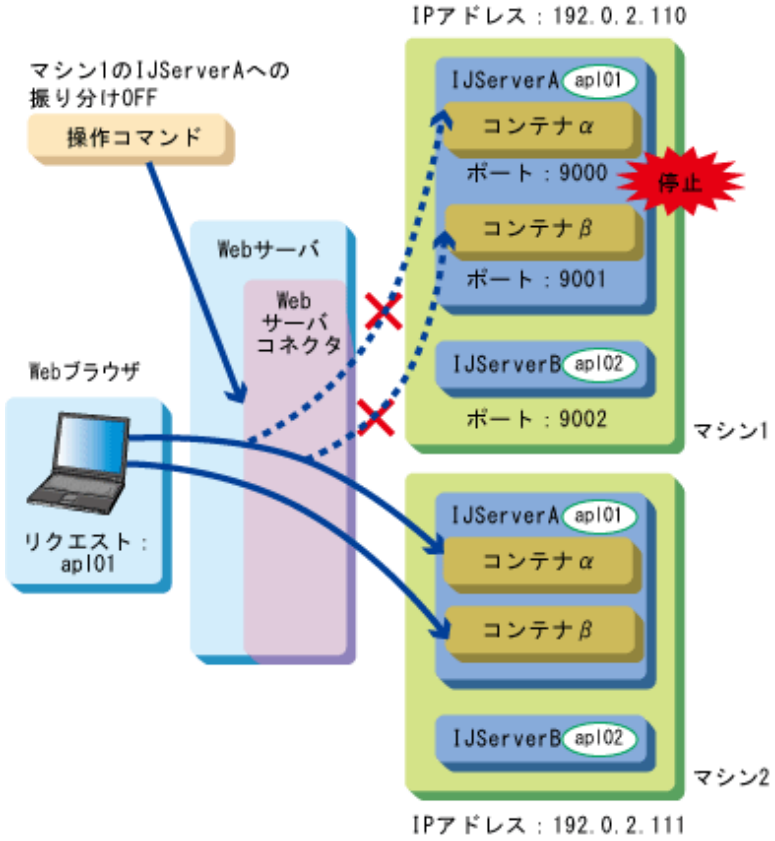
パターン3：IJServerワークユニット単位の振り分け操作(2)

IJServerワークユニットのプロセス多重度が2以上の場合で、Webアプリケーションの保守を行うためIJServerワークユニットへの振り分けを制御するには、以下のように運用します。

IPアドレス192.0.2.110のマシン(マシン1)とIPアドレス192.0.2.111のマシン(マシン2)をそれぞれ2つのIJServerワークユニット(IJServerA、IJServerB)で構成しています。
IJServerAのプロセス多重度を2とし、2つのServletコンテナ(コンテナ α 、コンテナ β)を起動しています。IJServerAにはapl01、IJServerBにはapl02のWebアプリケーションを配備しています。
マシン1のIJServerAを停止するため、振り分け対象からはずして運用します。Servletコンテナ数分の振り分け操作コマンドを実行します。

```
ijsdispatchcont OFF 192.0.2.110:9000
ijsdispatchcont OFF 192.0.2.110:9001
```

負荷分散のため、マシン1のコンテナ α 、コンテナ β とマシン2のコンテナ α 、コンテナ β に振り分けられていたapl01は、マシン2のコンテナ α 、コンテナ β での運用となります。apl02は変わらず、マシン1のIJServerBとマシン2のIJServerBに振り分けられます。



マシン1のIJServerA復旧時には、以下のコマンドで振り分け対象に戻します。

```
ijsdispatchcont ON 192.0.2.110:9000
ijsdispatchcont ON 192.0.2.110:9001
```

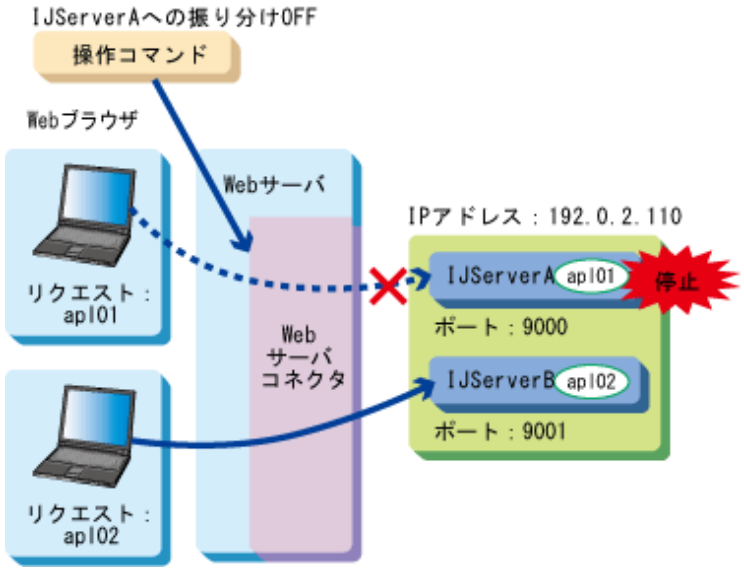
パターン4：IJServerワークユニットの接続抑止

IJServerワークユニット側のマシンが1台の構成の場合でWebアプリケーションの保守を行うためIJServerワークユニットへの接続を抑止するには、以下のように運用します。

IPアドレス192.0.2.110のマシンを2つのIJServerワークユニット(IJServerA、IJServerB)で構成しています。
IJServerAにはapl01、IJServerBにはapl02のWebアプリケーションを配備しています。
IJServerAを停止するため、振り分け対象からはずして運用します。

```
ijsdispatchcont OFF 192.0.2.110:9000
```

振り分け対象がIJServerAのみだったapl01は処理不可となります。apl02は変わらずIJServerBでの処理が可能です。
※)apl01のように振り分け先が1つも存在しなくなった場合は、リクエスト時にHTTPステータスコード503(Service Temporarily Unavailable)をWebブラウザへ返却します。



IJServerA復旧時には、以下のコマンドで振り分け対象に戻します。

```
ijsdispatchcont ON 192.0.2.110:9000
```

3.7.2 Webサーバコネクタの故障監視

注意

- 以下の場合、故障監視を行うWebサーバの名前には、同じ名前を使用しないでください。
  - Webサーバコネクタ(Interstage HTTP Server 2.2用)を併用する場合、かつ
  - Webサーバコネクタ(Interstage HTTP Server 2.2用)のいずれか一方、または両方で故障監視機能を使用する場合

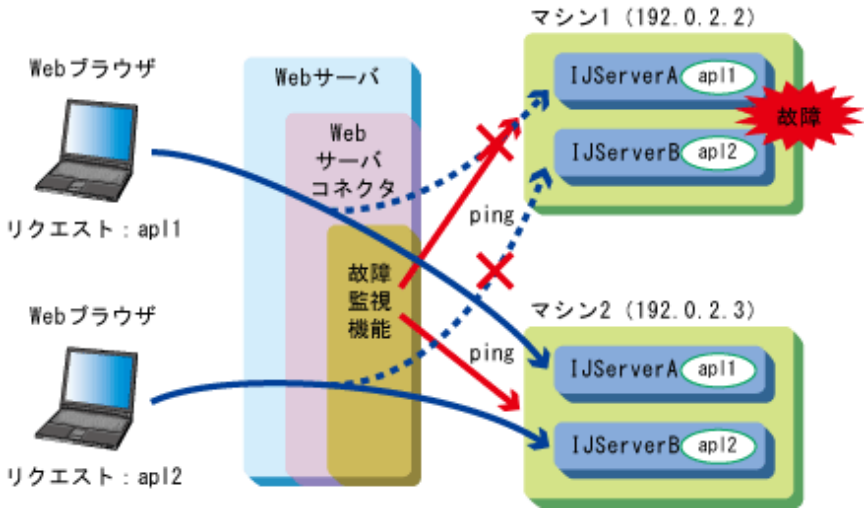
IJServerとWebサーバをそれぞれ別のサーバマシンに分離して負荷分散運用を行う場合に、分散先のIJServerマシンやServletコンテナの稼動状況を監視し、故障したIJServerマシンやServletコンテナを自動的に振り分けの対象から除外することや、故障から復旧したIJServerマシンやServletコンテナを自動的に振り分けの対象に戻すことができます。
Webサーバコネクタの故障監視機能によって振り分けの対象から除外されたIJServerマシンやServletコンテナに対しては、Webサーバコネクタがリクエストの振り分けを抑止します。

故障監視方式

故障監視機能には、以下の2種類の故障監視方式があります。

ping監視

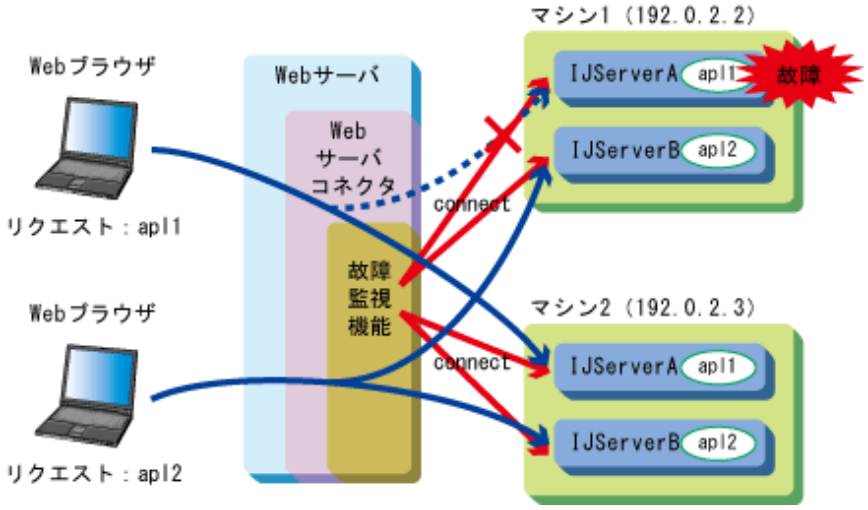
IJServerマシンのIPアドレスに対してping(ICMP ECHO)を発行し、応答(ICMP ECHO REPLY)の有無によって、稼動状況を監視します。



例えば、上図のように、故障監視機能はマシン1とマシン2のIPアドレスに対して定期的(例えば60秒ごと)にpingを発行し、応答があるかどうかを監視します。マシン1のハード故障などによってpingの応答がなくなると、自動的に振り分けの対象から除外され、Webサーバコネクタはマシン1へのリクエストの振り分けを停止します。
故障したマシンを修理して、再びpingの応答が返るようになると、自動的に振り分けの対象に戻され、Webサーバコネクタは再びマシン1へリクエストの振り分けを開始します。

ポート監視

Servletコンテナがリクエストを受け付けるTCPポートに対して、socketのconnectによるTCPコネクションの接続可否によって、Servletコンテナの稼動状況を監視します。



例えば、上図のように、故障監視機能はマシン1とマシン2上でオープンされているServletコンテナがリクエストを待ち受けているTCPポートに対して定期的(例えば60秒ごと)にconnectを行い、接続できるかどうかを監視します。マシン1のIJServerAの停止などによりServletコンテナがリクエストを待ち受けるTCPポートがクローズし、connectによるTCPコネクション接続ができなくなると、自動的に振り分け対象から除外され、Webサーバコネクタは、マシン1のIJServerAのServletコンテナへのリクエストの振り分けを停止します。
IJServerAを起動して、再びconnectによるTCPコネクション接続ができるようになると、自動的に振り分けの対象に戻され、Webサーバコネクタは再びマシン1のIJServerAのServletコンテナへのリクエストの振り分けを開始します。

事前準備

故障監視機能を使用する場合には、事前に以下の設定が必要です。

IJServerとWebサーバを分離して運用する場合の手順の実施

故障監視機能は、IJServerとWebサーバを分離して運用する構成の場合のみ使用することができます。IJServer用マシンとWebサーバ用マシンそれぞれのInterstage管理コンソールで、[システム] > [環境設定]タブ > [Servletサービス詳

細設定] > [Webサーバとワークユニットを同一のマシンで運用する]で[運用しない]を選択します。また、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

設定項目

故障監視機能の設定は、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [故障監視設定]タブで行います。また、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。
以下の設定項目があります。

項目名	意味
故障監視方式	以下のいずれかの監視方法を選択します。 ・ 使用しない 故障監視機能を使用しません。 ・ ping監視 IJServerマシンの稼動状況をpingで行います。 ・ ポート監視 IJServerマシンの稼動状況をpingで行うと同時に、Servletコンテナの稼動状況をconnectで行います。
故障監視間隔	ping監視またはポート監視による稼動状況の監視を実施する間隔を設定します。
応答待ち時間	IJServerマシンにpingまたはconnectを発行してから、応答が返るまでの待ち時間を設定します。 待ち時間を超えても応答がなかった場合、再度、故障時のリトライ回数に設定した回数のpingまたはconnectを発行し、そのすべてに応答がなかった場合に故障と判断されます。
故障時のリトライ回数	IJServerマシンにpingまたはconnectを発行してから、応答待ち時間を超えても応答がなかった場合のリトライ回数を設定します。
起動待ち時間	Webサーバコネクタ起動時に、故障監視機能が未起動の場合、故障監視機能の起動待ちをする時間を設定します。 この時間を超えても故障監視機能が起動しなかった場合、Webサーバコネクタは故障監視機能を使用せずに動作します。 故障監視機能を使用せずに動作する場合、Webサーバコネクタは、クライアントからのリクエストを、故障中のIJServerマシンやServletコンテナに振り分けてしまうことがあります。

運用準備の具体例

以下のような運用を行う場合について、具体的な運用手順例を説明します。

- 2台のIJServerマシンに負荷分散を行う構成で、IJServerマシンの稼動状況を監視する
- 2台のIJServerマシンに負荷分散を行う構成で、IJServerマシンの稼動状況およびServletコンテナの稼動状況を監視する

以下では、Interstage管理コンソールを使用する場合の手順を説明していますが、isj2eeadminコマンドを使用して設定することもできます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

運用準備の具体例1

2台のIJServerマシンに負荷分散を行う構成で、IJServerマシンの稼動状況を監視する例を示します。

条件

サーバマシンA(IPアドレス:192.0.2.1)	Webサーバを運用

サーバマシンB(IPアドレス:192.0.2.2)	IJServerを運用
サーバマシンC(IPアドレス:192.0.2.3)	IJServerを運用

設定手順

1. サーバマシンBのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容のワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.2
ポート番号	9000

2. サーバマシンCのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容のワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.3
ポート番号	9000

3. サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]タブを選択し、以下の設定内容の接続先情報を作成します。

項目名	設定値例
ワークユニット名	MyIJServer
ServletコンテナのIPアドレス:ポート番号	192.0.2.2:9000 192.0.2.3:9000

4. サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [故障監視設定]タブを選択し、以下の設定内容で故障監視機能を設定します。

項目名	設定値例
故障監視方式	ping監視
故障監視間隔	60
応答待ち時間	10
故障時のリトライ回数	3

5. サーバマシンAのInterstage管理コンソールで、[サービス] > [Webサーバ] > “Webサーバ名” > [状態]タブを選択し、Webサーバを再起動してください。

運用準備の具体例2

2台のIJServerマシンに負荷分散を行う構成で、IJServerマシンの稼動状況およびServletコンテナの稼動状況を監視する例を示します。

条件

サーバマシンA(IPアドレス:192.0.2.1)	Webサーバを運用

サーバマシンB(IPアドレス:192.0.2.2)	IJServerを運用
サーバマシンC(IPアドレス:192.0.2.3)	IJServerを運用

設定手順

1. サーバマシンBのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容の、2つのワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer1
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.2
ポート番号	9000

項目名	設定値例
ワークユニット名	MyIJServer2
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.2
ポート番号	9001

2. サーバマシンCのInterstage管理コンソールで、[ワークユニット] > [新規作成]タブを選択し、以下の設定内容の、2つのワークユニットを作成します。

項目名	設定値例
ワークユニット名	MyIJServer1
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.3
ポート番号	9000

項目名	設定値例
ワークユニット名	MyIJServer2
要求を受け付けるWebサーバのIPアドレス	192.0.2.1
ServletコンテナのIPアドレス	192.0.2.3
ポート番号	9001

3. サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [新規作成]タブを選択し、以下の設定内容の、2つの接続先情報を作成します。

項目名	設定値例
ワークユニット名	MyIJServer1
ServletコンテナのIPアドレス:ポート番号	192.0.2.2:9000 192.0.2.3:9000

項目名	設定値例
ワークユニット名	MyIJServer2

項目名	設定値例
ServletコンテナのIPアドレス:ポート番号	192.0.2.2:9001 192.0.2.3:9001

4. サーバマシンAのInterstage管理コンソールで、[Webサーバ] > “Webサーバ名” > [Webサーバコネクタ] > [故障監視設定]タブを選択し、以下の設定内容で故障監視機能を設定します。なお、ポート監視を選択した場合の監視先は、3の手順の“ServletコンテナのIPアドレス:ポート番号”で設定したすべてのIPアドレス:ポート番号となります。

項目名	設定値例
故障監視方式	ポート監視
故障監視間隔	60
応答待ち時間	10
故障時のリトライ回数	3

5. サーバマシンAのInterstage管理コンソールで、[サービス] > [Webサーバ] > “Webサーバ名” > [状態]タブを選択し、Webサーバを再起動してください。

稼動状況の表示

Webサーバマシンで、以下のコマンドを使用することにより、分散先のIJServerの稼動状況を表示することができます。コマンドの詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

```
svmondspstat
```

すべてのIJServerの状態を表示する

オプション指定なしでsvmondspstatコマンドを実行することにより、すべての分散先についてのIJServerの稼動状況が表示されます。

条件

- IJServer名がMyIJServer1とMyIJserver2の2つのIJServerへの振り分け
- 振り分け先のServletコンテナのIPアドレス:ポート番号が以下の4つ
  - 192.0.2.2:9000
  - 192.0.2.3:9000
  - 192.0.2.2:9001
  - 192.0.2.3:9001
- 故障監視方式がポート監視
- MyIJServer2の192.0.2.2:9001のみ故障中

表示例

上記の条件の場合、以下のように表示されます。

```
Status  IP Address  : Port Number : WorkUnit Name  :  WebServer Name
-----------------------------------------------------------------
ACTIVE  192.0.2.2 : 9000        : MyIJServer1    : FJapache
ACTIVE  192.0.2.3 : 9000        : MyIJServer1    : WEB001
DOWN    192.0.2.2 : 9001        : MyIJServer2    : FJapache
ACTIVE  192.0.2.3 : 9001        : MyIJServer2    : WEB001
```

特定のIJServerの状態のみを表示する

以下のように、Webサーバコネクタの“ワークユニット名”で設定したワークユニット名を指定してコマンドを実行することにより、特定のIJServerの稼動状況のみを表示することができます。

実行例

```
svmondspstat -i MyIJServer1
```

条件

- IJServer名がMyIJServer1とMyIJserver2の2つのIJServerへの振り分け
- 振り分け先のServletコンテナのIPアドレス:ポート番号が以下の4つ
  - 192.0.2.2:9000
  - 192.0.2.3:9000
  - 192.0.2.2:9001
  - 192.0.2.3:9001
- 故障監視方式がポート監視
- MyIJServer1のServletコンテナがすべて稼働中

表示例

上記の条件の場合、以下のように表示されます。

```
Status  IP Address  : Port Number : WorkUnit Name  :  WebServer Name
-------------------------------------------------------------------
ACTIVE  192.0.2.2 : 9000        : MyIJServer1    : FJapache
ACTIVE  192.0.2.3 : 9000        : MyIJServer1    : WEB001
```

留意点

故障監視機能を使用する場合、下記の留意点があります。

- 以下のすべてのIJServerクラスタを、故障監視設定で設定した条件で監視します。IJServerクラスタごとに別条件で監視を行うことはできません。
  - Webサーバコネクタ(Interstage http Server用)の分散先
    Interstage管理コンソールの「ServletコンテナのIPアドレス:ポート番号」の項目で設定
  - Webサーバコネクタ(Interstage http Server2.2用)の分散先
    wscadmin add-instance-refサブコマンドで設定
    add-instance-refサブコマンドについては、お使いのJava EE 6または、Java EE 7のマニュアルのwscadmin add-instance-refサブコマンドを参照してください。(注)
- WebサーバマシンとIJServerマシンの間にファイアウォールを設置する場合、ファイアウォールで、WebサーバマシンのIPアドレスから分散先のIJServerマシンのIPアドレス宛てにPINGの通過を許可する設定を行う必要があります。
- WebサーバマシンとIJServerマシンの間の経路に設置されたネットワーク機器の高負荷などによってpingまたはconnectの応答の遅延やロストが発生すると、IJServerが実際には稼動状態であっても故障と判定されてしまう場合があります。Webサーバと IJServerマシンの経路の状態に沿って、“応答待ち時間”や“故障時のリトライ回数”に適切な値を設定してください。
- WebサーバマシンとIJServerマシンの間にIPCOMなどのロードバランサを設置して負荷分散を行う場合、本機能を使用することはできません。
- 故障監視機能を有効にした場合や故障監視機能の設定変更を行った場合、Webサーバを起動または再起動した時点で設定変更された内容で振り分け先の故障監視を開始します。
  また、設定変更後に、Webサーバの再起動を行わない場合、クライアントからの最初のアクセスが発生した時点で、設定変更された内容で振り分け先の故障監視を開始します。
- Webサーバの起動直後から、故障監視機能によって故障と判定されるまでの間、故障中の振り分け先にリクエストが振り分けられてしまうことがあります。
  振り分け先の状態の取得開始から、故障と判定されるまでの時間は、「応答待ち時間×故障時のリトライ回数」です。

- 故障監視機能を有効にした後や故障監視機能の設定変更を行った後、Webサーバの再起動を行わなかった場合、クライアントからの最初のアクセスが発生した時点で振り分け先が故障していても、故障監視機能によって故障と判定されるまでの間、故障中の振り分け先にリクエストが振り分けられてしまうことがあります。
  振り分け先の状態の取得開始から、故障と判定されるまでの時間は、「応答待ち時間×故障時のリトライ回数」です。
- 実際に振り分け先が故障してから、故障監視機能によって故障と判定されるまでの間、故障中の振り分け先にリクエストが振り分けられてしまうことがあります。
  実際に振り分け先が故障してから、故障と判定されるまでの時間は、「故障監視間隔＋応答待ち時間×故障監視のリトライ回数」以内です。
- 実際に振り分け先が故障してから、故障監視機能によって故障と判定されるまでの間に、故障中の振り分け先にリクエストが振り分けられた場合、そのリクエストについては、応答が1分程度以上遅延するかまたは、Webブラウザから“500 Internal Server Error”のステータスコードやエラーメッセージが通知されます。
  この場合、故障検出が行われた後に再度アクセスすることにより、稼動中の振り分け先にリクエストが振り分けられます。
- 故障中の振り分け先が復旧してから、故障監視機能によって復旧と判断されるまでの間は、復旧した振り分け先にリクエストは振り分けられません。
  実際に振り分け先が復旧してから、復旧と判定されるまでの時間は、「故障監視間隔」以内です。
- 故障しているサーバが存在する状態で、故障監視する設定を行うか、または故障監視条件を変更するか、または監視対象のサーバを動的に追加した場合、故障していたサーバの復旧時に、いったん故障検出のメッセージが出力され、すぐに復旧検出のメッセージが表示される場合があります。

  注）

EE
  - Java EE 6を使用している場合
    「Java EE運用ガイド(Java EE 6編)」の「Java EE 6運用コマンド」
  - Java EE 7を使用している場合
    「Java EE 7 設計・構築・運用ガイド」の「Java EE 7運用コマンド」

3.8 JTSを利用する場合の手順 Windows32/64 Linux32/64

JTSを利用する場合の手順を説明します。

3.8.1 運用開始までの手順

分散トランザクション機能(JTS)を利用したアプリケーションを運用するには、トランザクションサービス(OTS)が必要です。
また、JTSはJ2EEアプリケーションクライアント、Webアプリケーション、およびEJBアプリケーションから利用することが可能です。
それぞれのサービスのインストール手順、および環境設定方法については、“インストールガイド”および“運用ガイド（基本編）”を参照してください。
分散トランザクション機能、およびJTSの提供するJTAインタフェースについては、“第5部 JTS/JTA編 ”を参照してください。

JTSアプリケーションの運用開始までの手順を以下に示します。

1. リソースマネージャの環境設定
2. トランザクションサービスの環境設定
3. リソース定義情報の登録
4. データベースの起動
5. トランザクションサービスの起動
6. アプリケーションの起動

注意

Interstage管理コンソールではJTS用リソース管理プログラムは、“トランザクションサービス(JTS RMP)”と表示されます。

1.リソースマネージャの環境設定

リソースマネージャの環境設定はリソースマネージャのマニュアルを参照してください。
JTSで各種リソースマネージャを利用するためには、必要なクラスパスを設定する必要があります。必要なクラスライブラリは各リソースマネージャのマニュアルを参照してください。

リソースマネージャのクラスパスを設定するには、Interstage管理コンソールのシステムの環境設定を開いて、[J2EEプロパティ]にクラスパスを設定してください。

データベースのJDBCドライバ

データベースを利用する場合に設定する必要があります。

例

Oracleの場合

－ ojdbc6.jar

リソースアダプタのクラスライブラリ

Connectorを利用してリソースアダプタと連携する場合に設定する必要があります。

注意

リソースマネージャの環境設定に関する注意事項

Interstage管理コンソールを使用しない場合、環境変数classpathはシステム環境変数に設定してください。
環境変数を有効にするためには、Interstageの再起動が必要になります。

注意

リソースマネージャにOracleを使用する場合の注意事項

- データベースの構築
  JTSとEJBを利用した分散トランザクション機能配下でOracleを利用する場合は、Oracle JVMが有効になるようにデータベースを構築する必要があります。
- データベースの設定
  Oracleデータベースで分散トランザクションを利用可能とするために、以下の設定が必要になります。

  1. SYSユーザとして sqlplus にログオンします。

     sqlplus "sys/*password*@*ORACLE_SID* AS SYSDBA"

  2. 次の sql を実行します。

     grant select on DBA_PENDING_TRANSACTIONS to *username*
     ※*username*はデータソース定義に設定するユーザ名を指定してください。

- Oracle Real Application Clustersとの連携
  JTSは、Oracleのオプションであるオプション Real Application Clusterとの連携をサポートしていません。

2.トランザクションサービスの環境設定

Interstage管理コンソールで、トランザクションサービス(OTS)の環境設定を行います。

注意

トランザクションサービスの環境設定に関する注意事項

セットアップの詳細設定を変更した場合、トランザクションサービスの環境が再構築されます。
そのため、今まで登録されていたJTS用リソース定義情報は、すべて削除されます。
再度、対象のリソースをグローバルトランザクションで使用するには、リソースの環境設定でJDBCデータソースを使用する場合は、“分散トランザクションを使用する”を選択し、connectorを使用する場合は、“グローバルトランザクションの利用”を“する”に選択し、適用してください。

3.リソース定義情報の登録

Interstage管理コンソールを使用して、JTS用リソース定義の登録を行います。

- JDBCデータソースをグローバルトランザクションで使用する場合
  サービスリストからJDBCデータソースの新規作成を選択して、“分散トランザクションを使用する”をチェックし、適用してください。
- connectorをグローバルトランザクションで使用する場合
  サービスリストからリソースアダプタの配備を選択して、“グローバルトランザクションの利用”を“する”にチェックし、適用してください。
- 登録済みのリソース定義をグローバルトランザクションで使用する場合
  サービスリストから対象リソースの環境設定を選択して、以下をチェックし、適用してください。
  - JDBCリソースの場合、“分散トランザクションを使用する”
  - connectorの場合、“グローバルトランザクションの利用”の“する”

注意

リソース定義情報の登録に関する注意事項

- JDBCデータソースでサポートされるデータベースは、Oracleのみになります。
  SymfowareとSQL Serverでは、“分散トランザクションを使用する”が選択できません。
- JTSを使用した分散トランザクション処理で、リソースマネージャにOracleを利用する場合、かつ
  同一ホスト上で、かつ同一データベースインスタンスを参照するデータソースを複数登録する場合、
  以下の対処を行ってください。

  1. データベースのあるホストのIPアドレスを異なるホスト名になるように以下のファイルに定義する。

     Windows32/64

     %SystemRoot%¥system32¥drivers¥etc¥hosts

     Linux32/64

     /etc/hosts

     #### 例

     記述例

     ```
     192.0.2.110  oracle_db1
     192.0.2.110  oracle_db2
     ```

  2. JDBCデータソース定義の接続ホスト名に上記のホスト名をデータソース単位に割り振って記述する。
     その結果、同一ホスト、同一インスタンスのデータベースURLが異なるように設定する。

例

サーバURLの例

```
データソース1のサーバURL:jdbc:oracle:thin:@oracle_db1:1521:ORCL
データソース2のサーバURL:jdbc:oracle:thin:@oracle_db2:1521:ORCL
```

4.データベースの起動

データベースを起動します。
データベースの起動方法については、データベースのマニュアルを参照してください。

5.トランザクションサービスの起動

Interstage管理コンソールを使用して、Interstageを起動します。同時にトランザクションサービスも起動します。
システムの環境設定でトランザクションサービス(OTS)の環境設定が行われている場合は、Interstageの起動と一緒にトランザクションサービスも起動します。

注意

トランザクションサービスの起動に関する注意事項

“トランザクションサービス(JTS RMP)”は連携するデータベースの起動に依存してダウンリカバリ処理を実施するため、サーバマシンの再起動時には自動起動されません。そのため、データベースの起動後に以下のいずれかの作業が必要になります。

- Interstage管理コンソールを使用して、Interstageを再起動する
- otsstartrscコマンドを利用して、JTS用リソース管理プログラムを起動する。

6.アプリケーションの起動

Interstage管理コンソールで、ワークユニットを起動します。

3.8.2 運用終了までの手順

JTSアプリケーションの運用終了までの手順を以下に示します。

1. アプリケーションの停止
   Interstage管理コンソールを使用して、該当するワークユニットを停止します。
2. トランザクションサービスの停止
   Interstage管理コンソールを使用して、Interstageを停止します。同時にトランザクションサービスも停止します。
3. データベースの停止
   データベースを停止します。データベースの停止方法については、データベースのマニュアルを参照してください。
4. リソース定義情報の解除
   Interstage管理コンソールを使用して、グローバルトランザクションで利用するリソース定義を解除します。
5. トランザクションサービスの環境設定解除
   Interstage管理コンソールを使用して、トランザクションサービス(OTS)の環境設定解除を行います。

3.9 JMSを利用する場合の手順

JMSを利用する場合の手順を説明します。

3.9.1 運用開始までの手順

JMSを利用するには、次のコンポーネントが必要です。カスタムインストールを行う場合は、ObjectDirector EventServiceおよびJMSをインストールしてください。ObjectDirectorおよびJ2EE共通は、標準でインストールされます。
カスタムインストールの方法については、“インストールガイド”の“インストール”を参照してください。

- ObjectDirector
- ObjectDirector EventService
- J2EE共通
- JMS

JMSアプリケーションの運用開始までの手順を以下に示します。
Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”を参照してください。

1. イベントチャネル運用マシンの運用前の環境設定
   イベントチャネル運用マシンの運用前の環境設定を行います。
   詳細は、“23.1 イベントチャネル運用マシンの運用前の環境設定”を参照してください。
2. JMSアプリケーション運用マシンの運用前の環境設定
   JMSアプリケーション運用マシンの運用前の環境設定を行います。
   詳細は、“23.3 JMSアプリケーション運用マシンの運用前の環境設定”を参照してください。
3. イベントチャネル運用マシンの運用開始
   以下の手順で、イベントチャネル運用マシンの運用を開始します。
   1. Interstageの起動
      Interstage管理コンソールを使用して、Interstageの起動によりイベントサービスを起動します。
   2. 静的イベントチャネルの起動
      Interstage管理コンソールを使用して、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルを起動します。

      注意

      イベントチャネルの自動起動が設定されている場合、イベントチャネルはInterstageの起動(イベントサービスの起動)時に自動的に起動されます。イベントチャネルの自動起動は、Interstage管理コンソールを使用して変更することができます。初期値は“自動起動する”です。
4. JMSアプリケーション運用マシンの運用開始
   javaコマンドを直接使用して、JMSアプリケーションを起動します。

3.9.2 運用終了までの手順

JMSアプリケーションの運用終了までの手順を以下に示します。

1. JMSアプリケーション運用マシンの運用終了
   JMSアプリケーションを停止します。
2. イベントチャネル運用マシンの運用終了
   以下の手順で、イベントチャネル運用マシンの運用を終了します。
   1. 静的イベントチャネルの停止
      Interstage管理コンソールを使用して、JMSアプリケーションがメッセージの送受信に使用しているイベントチャネルを強制停止します。
   2. Interstageの停止
      Interstage管理コンソールを使用して、Interstageの停止によりイベントサービスを強制停止します。

3. JMSアプリケーション運用マシンの環境削除
   JMSアプリケーション運用マシンの運用後の環境削除を行います。
   詳細は、“23.4 JMSアプリケーション運用マシンの運用後の環境削除”を参照してください。

4. イベントチャネル運用マシンの環境削除
   イベントチャネル運用マシンの運用後の環境削除を行います。
   詳細は、“23.2 イベントチャネル運用マシンの運用後の環境削除”を参照してください。

3.9.3 イベントチャネル動作状況の参照

Interstage管理コンソールを使用して、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルの動作状況を参照することができます。
Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”を参照してください。

表示するイベントチャネルの情報を以下に示します。

	表示項目	内容
1	グループ名	イベントチャネルが含まれるグループ名です。 ・ 動的生成:EventFactory(テンポラリトピックまたはテンポラリキュー) ・ 静的生成:利用者が作成したイベントチャネルグループ名
2	チャネル名	表示対象となるイベントチャネル名です。ただし、テンポラリトピックまたはテンポラリキュー(動的生成チャネル)の場合は識別IDです。
3	タイプ	イベントチャネルのタイプです。 ・ Topic:Publish/Subscribeメッセージングモデルで使用 ・ Queue:Point-To-Pointメッセージングモデルで使用 ・ TemporaryTopic:Publish/Subscribeメッセージングモデルで使用 ・ TemporaryQueue:Point-To-Pointメッセージングモデルで使用
4	Destination	イベントチャネルに関連付けられているDestination定義のJNDI名です。
5	ユニット	不揮発チャネル運用時の保存先のユニットIDです。
6	起動状態	イベントチャネルの状態を表示します。 ・ 起動　:イベントチャネルは起動している ・ 起動処理中:イベントチャネルは起動処理中である ・ 停止　:イベントチャネルは停止している ・ 停止処理中:イベントチャネルは停止処理中、または閉塞終了モードで停止中
7	現蓄積メッセージ数	イベントチャネルに蓄積しているメッセージの現数です。
8	接続コンシューマ数	イベントチャネルに接続されているサブスクライバ数またはレシーバ数です。(注)
9	接続プロデューサ数	イベントチャネルに接続されているパブリッシャ数またはセンダ数です。

注) EJBのMessage-driven Beanを使用している場合は、接続コンシューマ数に以下の値が加算されます。

・ Point-To-Pointメッセージングモデル
接続コンシューマ数の加算値は、通信状態により初期値から最大値まで増加します。

process × instance(初期値) ≦ 接続コンシューマ数の加算値 ≦ process × instance × 2 (最大値)

process:IJServerの[プロセス多重度]
instance:Message-driven Beanの[初期起動インスタンス数(同時実行スレッド数)]

・ Publish/Subscribeメッセージングモデル

接続コンシューマ数の加算値 ＝ 1

注意

イベントチャネルに蓄積されたメッセージは、以下の契機で削除されます。

・ 揮発運用時のTopicタイプのイベントチャネル
  ― コンシューマ(サブスクライバ)がメッセージを取り出した
  ― メッセージの生存時間に達した
  ― コンシューマ(サブスクライバ)の接続情報を回収した
  ― イベントチャネルを強制停止した
・ 揮発運用時のQueueタイプのイベントチャネル
  ― コンシューマ(レシーバ)がメッセージを取り出した
  ― メッセージの生存時間に達した
  ― イベントチャネルを強制停止した
・ 不揮発化運用時のTopicタイプのイベントチャネル
  ― コンシューマ(サブスクライバ)がメッセージを取り出した
  ― メッセージの生存時間に達した
  ― コンシューマ(サブスクライバ)の接続情報を回収した
・ 不揮発化運用時のQueueタイプのイベントチャネル
  ― コンシューマ(レシーバ)がメッセージを取り出した
  ― メッセージの生存時間に達した

  ※不揮発化運用時のQueueタイプのイベントチャネルの場合、イベントチャネルの強制停止では、メッセージを削除することはできません。イベントチャネルに蓄積されたメッセージを削除するには、コンシューマ(レシーバ)でメッセージを取り出してください。

イベントチャネルが閉塞終了中の場合、イベントチャネルにメッセージが蓄積されている間は、イベントチャネルは停止しません。イベントチャネルに蓄積されているメッセージがコンシューマに配信されて削除されるか、生存時間に達して削除された段階で、イベントチャネルが停止します。

Interstage管理コンソールを使用して、メッセージの保存先(ユニット)の状況を参照することができます。

表示する保存先(ユニット)の情報を以下に示します。

	表示項目	内容
1	ユニットID	ユニット名

	表示項目	内容
2	ユニットモード	不揮発チャネル運用時のユニット種別です。 ・ 標準:標準ユニット ・ 拡張:拡張ユニット
3	システム使用率(%)	格納ディレクトリのシステム用(ユニット制御用)ファイルの使用率を表示します。
4	イベントデータ使用率(%)	格納ディレクトリのイベントデータ用ファイルの使用率を表示します。
5	システム領域数	不揮発化チャネル運用時のシステム用(ユニット制御用)データ格納域の数を表示します。
6	イベントデータ領域数	不揮発化チャネル運用時のイベントデータ用データ格納域の数を表示します。

3.10 JavaMailを利用する場合の手順

JavaMailを利用する場合の手順を説明します。

運用前

1. アプリケーションの作成
   メールを送信/受信するアプリケーションを作成します。詳細は以下を参照してください。
   - メール送信を行うアプリケーション
   - メール受信を行うアプリケーション
2. メールサーバの環境設定
   JavaMailアプリケーションを使用してメールの送受信を行うには、メール送信用のSMTPサーバ、およびメール受信用のPOP3サーバ、またはIMAPサーバが必要となります。
   メールの送信、および受信用のサーバの環境設定を行ってください。詳細は、メールサーバのマニュアルを参照してください。
3. メールサーバの起動
   メール送信用のSMTPサーバ、およびメール受信用のPOP3サーバ、またはIMAPサーバを起動してください。詳細は、メールサーバのマニュアルを参照してください。
4. リソースの定義
   JavaMailリソースの設定を行ってください。詳細は、“第4章 JNDI”を参照してください。

運用後

1. メールサーバの停止
   メールサーバを停止します。詳細は、メールサーバのマニュアルを参照してください。

3.10.1 メール送信を行うアプリケーション

Mailの送信手順を次に示します。

1. JavaMailリソースのlookup処理
2. メッセージの作成
3. SMTPサーバとの接続
4. メッセージの送信

1.JavaMailリソースのlookup処理

JavaMailリソースのlookup処理を行います。

```
// Mailリソースのlookup処理
    InitialContext nctx = new InitialContext();
    session = (Session) nctx.lookup("java:comp/env/mail/MailSession");
}
catch(NamingException ex) { }
```

2.メッセージの作成

送信するメッセージを作成します。
メッセージには次の内容を設定します。

- 送信者(From)
- 宛先(To)
- 宛先(Cc)
- 宛先(Bcc)
- 題名(Subject)
- 本文

```
// メッセージの作成
MimeMessage msg = null;
try {
    // メッセージの生成
    msg = new MimeMessage(session);
    // 送信者(From)の設定
    msg.setFrom(new InternetAddress("<from-address>"));
    // 宛先(To)の設定
    Address[] toAddress = {new InternetAddress("<to-address>")};
    msg.setRecipients(Message.RecipientType.TO, toAddress);
    // 宛先(Cc)の設定
    Address[] ccAddress = {new InternetAddress("<cc-address>")};
    msg.setRecipients(Message.RecipientType.CC, ccAddress);
    // 宛先(Bcc)の設定
    Address[] bccAddress = {new InternetAddress("<bcc-address>")};
    msg.setRecipients(Message.RecipientType.BCC, bccAddress);
    // 題名(Subject)の設定
    String subject = new String("<Subject>");
    msg.setSubject(subject);
    // 本文の設定
    String msgTxt = new String("<Message Text>");
    msg.setText(msgTxt);
}
catch(AddressException ex) { }
catch(MessagingException ex) { }
```

3.SMTPサーバとの接続

SMTPサーバに接続します。

```
// SMTPサーバとの接続
Transport transport = null;
try {
    transport = session.getTransport("smtp");
    transport.connect();
}
catch(NoSuchProviderException ex) { }
catch(MessagingException ex) { }
```

4.メッセージの送信

作成したメッセージを送信します。

```
// メッセージの送信
try {
    transport.sendMessage(msg, msg.getAllRecipients());
}
catch(MessagingException ex) { }
```

3.10.2 メール受信を行うアプリケーション

Mailの受信手順を次に示します。

1. JavaMailリソースのlookup処理
2. サーバへの接続
3. 受信ディレクトリのオープン
4. メッセージの取出し

1.JavaMailリソースのlookup処理

JavaMailリソースのlookup処理を行います。

```
// Mailリソースのlookup
Session session = null;
try {
    InitialContext nctx = new InitialContext();
    session = (Session) nctx.lookup("java:comp/env/mail/MailSession");
}
catch(NamingException ex) { }
```

2.サーバへの接続

POP3サーバへ接続する場合

```
// POP3サーバへの接続
Store store = null;
try {
    store = session.getStore("POP3");        /* POP3サーバへ接続*/
    store.connect("<hostname>", "<user>", "<password>");
}
catch(NoSuchProviderException ex) { }
catch(MessagingException ex) { }
```

IMAPサーバへ接続する場合

```
// IMAPサーバへの接続
Store store = null;
try {
    store = session.getStore("imap");        /* IMAPサーバへ接続 */
    store.connect("<hostname>", "<user>", "<password>");
}
catch(NoSuchProviderException ex) { }
catch(MessagingException ex) { }
```

3.受信ディレクトリのオープン

受信ディレクトリをオープンします。

```
// 受信ディレクトリのオープン
Folder inbox = null;
try {
```

```
    Folder rootFolder = store.getDefaultFolder();
    inbox = rootFolder.getFolder("INBOX");
    inbox.open(Folder.READ_WRITE);
}
catch(MessagingException ex) { }
```

4.メッセージの取出し

受信したメッセージを取出します。
メッセージから次の内容を取出します。

- 送信者(From)
- 宛先(To)
- 宛先(Cc)
- 宛先(Bcc)
- 題名(Subject)
- 本文

```
// メッセージの取出し
try {
    Message msg = inbox.getMessage(1);
    // 送信者(From)の取出し
    Address[] fromAddress = msg.getFrom();
    // 宛先(To)の取出し
    Address[] toAddress = msg.getRecipients(Message.RecipientType.TO);
    // 宛先(Cc)の取出し
    Address[] ccAddress = msg.getRecipients(Message.RecipientType.CC);
    // 宛先(Bcc)の取出し
    Address[] bccAddress = msg.getRecipients(Message.RecipientType.BCC);
    // 題名(Subject)の取出し
    String subject = msg.getSubject();
    // 本文の取出し
    Object content = msg.getContent();
    String text = content.toString();
}
catch(MessagingException ex) { }
catch(IOException ex) { }
```

3.11 動作環境のカスタマイズと確認

ここでは、動作環境のカスタマイズの方法と、何らかの理由により、J2EEアプリケーションが動作しなくなった場合に確認すべき内容について説明します。

ポイント

Windows32/64

Interstage Application Serverのインストール後には、J2EEアプリケーションが登録されていない状態でデフォルトのIJServerが登録されます。デフォルトのIJServerは“IJServer”という名前で登録され、実際の運用に利用できます。
すぐにJ2EEアプリケーションを実行させたい場合には、お手持ちのJ2EEアプリケーションを配備してください。

ここでは、以下について説明します。

- 環境変数の設定
- Javaの環境設定
- IJServerを利用する場合の設定

- EJBサービス運用コマンドを利用する場合の設定

環境変数の設定

環境変数CLASSPATHに、次の値が設定されていない場合は設定してください。

Windows32/64

C:¥Interstage¥J2EE¥lib¥isj2ee.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isj2ee.jar

Javaの環境設定

Javaのインストール

Interstageのサーバパッケージをインストールする場合

- J2EEをインストールするとJDK8がインストールされます。

Javaの環境変数

Interstage上で、J2EEアプリケーションを動作させる場合は、Javaの環境設定を行う必要があります。環境変数PATHに、以下の値が設定されていない場合は設定してください。

Windows32/64

JDK使用時:C:¥Interstage¥JDK8¥jre¥bin
JRE使用時:C:¥Interstage¥JRE8¥bin

Solaris64 Linux32/64

JDK使用時:/opt/FJSVawjbk/jdk8/jre/bin
JRE使用時:/opt/FJSVawjbk/jre8/bin

注意

Javaの環境変数に関する注意事項

環境変数の“PATH”は、使用するシェルによっては“path”となる場合がありますので、環境に合わせて設定してください。
ディレクトリ中にバージョンが含まれている場合は適宜読み替えて設定してください。
Java VMのバージョンの違いによる非互換等の問題を防ぐため、開発/配備/運用で使用するJDK/JREのバージョンは一致させる事を推奨します。また、J2EEアプリケーションクライアント、Webアプリケーション、EJBアプリケーションが連携して動作する場合なども、おのおのが使用するJava VMのバージョンの違いによる非互換等の問題を防ぐため、使用するJDK/JREのバージョンを一致させることを推奨します。

IJServerを利用する場合の設定

Java環境設定ファイル

IJServerを利用する場合はJava環境設定ファイルへの設定が必要です。
Interstageのインストール時にJavaの環境設定が設定されていない場合、または、SolarisまたはLinuxでカスタムインストールでJavaを後からインストールした場合または、Javaを追加インストールした場合は、Java環境設定ファイルにJavaを追加で設定してください。

Java環境設定ファイルは、以下のディレクトリに作成されます。

Windows32/64

C:¥Interstage¥J2EE¥etc¥java_config.txt

Solaris64 Linux32/64

/opt/FJSVj2ee/etc/java_config.txt

以下に、設定形式と設定する場合の注意事項について説明します。

設定形式

以下の形式で設定してください。

```
使用するJavaの種別 = Javaのインストールディレクトリ
```

ー 使用するJavaの種別
使用するJavaは、以下のように記述してください。

- JDKを使用する場合・・・JDK60DIR
- JREを使用する場合・・・JRE60DIR

ー Javaのインストールディレクトリ
Javaのインストールディレクトリは、絶対パス形式で設定してください。

例

Windows32/64

C:¥Interstage¥jdk8にインストールしたJDK8を使用する場合の記述例
JDK60DIR = C:¥Interstage¥jdk8

Solaris64 Linux32/64

/opt/FJSVawjbk/jdk8にインストールしたJDK8を使用する場合の記述例
JDK60DIR = /opt/FJSVawjbk/jdk8

注意

Java環境設定ファイルに関する注意事項

- Java環境設定ファイル(java_config)には、コメント行を記述することはできません。先頭が#、!などの記号の場合は、不当な情報として扱います。

Windows32/64

- 業務運用中に、Java環境設定ファイルの削除や内容の変更をしないでください。
- Interstageのインストール時にJDKを選択した場合、Java環境としてJDKの中に含まれるJREを使用することはできません。この場合、「使用するJavaの種別」に“JDK60DIR”を指定し、Java環境としてJDKを使用するようにしてください。

Solaris64 Linux32/64

- Java環境設定ファイルに情報を設定する場合は、管理者権限で実施してください。
- 業務運用中に、Java環境設定ファイルの削除や内容の変更をしないでください。
- Java環境設定ファイルの「使用するJavaの種別」にJRE60DIRを指定した場合は、「Javaのインストールディレクトリ」に、JDK配下のJREを設定しないでください。JREをインストールしたディレクトリを設定してください。

EJBサービス運用コマンドを利用する場合の設定

EJBのJava環境設定ファイル

EJBサービス運用コマンドを利用する場合は、EJBのJava環境設定ファイルの設定が必要です。
以下のいずれかに該当している場合は、Java環境設定ファイルにJavaを追加で設定してください。

- Interstageのインストール時にJavaの環境設定が設定されていない
- SolarisまたはLinuxでカスタムインストールでJavaを後からインストールした

・ Javaを追加インストールした

Java環境設定ファイルは、以下のディレクトリに作成されます。

Windows32/64

C:¥Interstage¥EJB¥etc¥java_config.txt

Solaris64 Linux32/64

/opt/FJSVejb/etc/java_config.txt

以下に設定形式と設定する場合の注意事項について説明します。

設定形式

以下の形式で設定してください。

使用するJavaの種別 = Javaのインストールディレクトリ

ー 使用するJavaの種別
使用するJavaの種別は、以下のように記述してください。

- JDKを使用する場合・・・JDK60DIR

- JREを使用する場合・・・JRE60DIR

ー Javaのインストールディレクトリ
Javaのインストールディレクトリは、絶対パス形式で設定してください。

例

Windows32/64

C:¥Interstage¥jdk8にインストールしたJDK8を使用する場合の記述例
JDK60DIR = C:¥Interstage¥jdk8

Solaris64 Linux32/64

/opt/FJSVawjbk/jdk8にインストールしたJDK8を使用する場合の記述例
JDK60DIR = /opt/FJSVawjbk/jdk8

注意

Java環境設定ファイルに関する注意事項

・ Java環境設定ファイル(java_config)には、コメント行を記述することはできません。先頭が#、!などの記号の場合は、不当な情報として扱います。

Windows32/64

・ 業務運用中に、Java環境設定ファイルの削除や内容の変更をしないでください。

・ Interstageのインストール時にJDKを選択した場合、Java環境としてJDKの中に含まれるJREを使用することはできません。この場合、「使用するJavaの種別」に“JDK60DIR”を指定し、Java環境としてJDKを使用するようにしてください。

Solaris64 Linux32/64

・ Java環境設定ファイルに情報を設定する場合は、管理者権限で実施してください。

・ 業務運用中に、Java環境設定ファイルの削除や内容の変更をしないでください。

・ Java環境設定ファイルの「使用するJavaの種別」にJRE60DIRを指定した場合は、「Javaのインストールディレクトリ」に、JDK配下のJREを設定しないでください。
JREをインストールしたディレクトリを設定してください。

3.12 アプリケーションのデバッグ

アプリケーションのデバッグ情報は、IJServerのログファイルに出力されます。

IJServerのログ

IJServerのログファイルには以下の情報が出力されます。アプリケーションの問題を特定したい場合に、このログファイルを利用できます。ログの出力場所は“2.2.3 IJServerのファイル構成”を参照してください。

- コンテナログ(container.log)
  - アプリケーションの標準出力、標準エラー出力
  - GenericServletクラスのlogメソッドの出力
  - ServletContextクラスのlogメソッドの出力
  - EJBコンテナログ/Servletコンテナログ
  - EJBコンテナ/Servletコンテナのエラーメッセージ
  - EJBのスナップ出力
  - Webサービスのログ
- コンテナ情報ログ(info.log)
  - Java VMプロセスの起動情報(ARGV、ENV)
  - Java VMプロセスの起動エラーメッセージ
  - スレッドダンプ
  - コンテナログに出力できないメッセージ

デバッグ方法

アプリケーションのデバッグ方法には、以下の方法があります。

- スナップを利用したデバッグ
  スナップを利用して、ロギングした各種情報を確認する方法です。
  詳細は、“3.13 スナップを利用したデバッグ”を参照してください。
- アプリケーションのデバッグ情報を利用したデバッグ
  アプリケーション実行時に、標準出力や標準エラー出力に出力されるデバッグ情報を確認する方法です。
  詳細は、“3.12.1 アプリケーションのデバッグ情報を利用したデバッグ”を参照してください。
- デバッガを利用したデバッグ
  Interstage Studioのデバッガを利用して、プログラム中の変数を参照または変更しながらアプリケーションの動作を確認する方法です。
  詳細は、“3.12.2 デバッガを利用したデバッグ”を参照してください。
- スレッドダンプ自動採取
  アプリケーションがタイムアウトまたは無応答になった場合にスレッドダンプを自動採取する方法です。
  詳細は、“3.12.3 スレッドダンプ自動採取”を参照してください。
- Javaメソッドトレースを利用したデバッグ
  Javaメソッドトレース機能を利用して、各メソッドの引数、復帰値を確認する方法です。
  詳細は、“3.12.4 Javaメソッドトレースを利用したデバッグ”を参照してください。

3.12.1 アプリケーションのデバッグ情報を利用したデバッグ

EJBアプリケーションの開発時に、あらかじめデバッグ情報を出力する処理を記述しておき、その情報をもとにデバッグする方法です。

デバッグ情報について

アプリケーションのデバッグ情報は、標準出力または、標準エラー出力を使用します。
デバッグ情報はIJServerのログファイルに出力されます。
IJServerのログについては“IJServerのログ”参照してください。

標準エラー出力の例外情報の出力について

例外クラスのprintStackTrace()メソッドを使用することで、例外がどこで発生したかをある程度、判断することができます。

例

```
catch(Exception e) {
  String emsg = e.getMessage();
  System.err.println("SampleBean.ejbCreate:Exception occured;");
  e.printStackTrace();
  throw new javax.ejb.EJBException(emsg);
}
```

注意

- 出力先のディスクに空き容量がない場合には、標準出力および標準エラー出力は出力されないので、ディスクの空きを確保してください。また、不要となった標準出力および標準エラー出力ファイルは、OSの機能を使って削除してください。
- SQL Serverを使用している場合は、“JDBCドライバロギング機能”を参照してください。

3.12.2 デバッガを利用したデバッグ

Interstage Studioが提供するデバッガを利用してデバッグする方法です。
デバッガを利用すると、開発したアプリケーションを実行させながら、処理の論理的な誤りを検出することができます。

通常、プログラムソース上にブレークポイントを設定し、ブレークポイントで停止した状態でプログラム中の変数を参照あるいは、変更しながらデバッグを行います。

デバッガを利用したデバッグの詳細は、“Interstage Studio ユーザーズガイド”を参照してください。

注意

- IJServerをデバッグする場合は、プロセス多重度を2以上にして起動することができません。ワークユニットの「プロセス多重度」は必ず1に設定してください。
- デバッグ起動モードに設定された複数のIJServerを同時に起動することはできません。

3.12.3 スレッドダンプ自動採取

アプリケーションがタイムアウトまたは無応答になった場合にスレッドダンプを自動採取する機能です。採取されたスレッドダンプを調査することでアプリケーションの無応答や処理のボトルネックを検出することができます。
スレッドダンプはコンテナ情報ログ(info.log)に採取されます。
スレッドダンプは下記の採取契機に10秒間隔で2回採取されます。2回のスレッドダンプで変化がなかったスレッド上の動作アプリケーションで問題があることが検出できます。
なお、1度スレッドダンプが採取されてから10分間はスレッドダンプは採取されません。

スレッドダンプ採取契機

- IJServerの起動時のタイムアウト
  起動時実行クラス、サーブレットのinitメソッド等IJServerの起動時に動作する処理に問題があるか、処理に時間がかかっている可能性があります。タイムアウト値は、“ワークユニット起動待ち時間”で設定します。
- アプリケーションのタイムアウト
  アプリケーションに問題があるか、処理に時間がかかっている可能性があります。タイムアウト値は、“アプリケーション最大処理時間”で設定します。
- IJServerの停止時のタイムアウト
  IJServerワークユニットの停止で、“プロセス強制停止時間”を超過した場合、自動的にスレッドダンプが採取されます。タイムアウト値は、“プロセス強制停止時間”で設定します。
- JavaVMへの生存監視で30秒間応答がない場合
  JavaVMへの生存監視で30秒間応答がない場合、EXTP4367メッセージを出力し、自動的にスレッドダンプが採取されます。システムの負荷が高い、または、メモリ不足などが発生している可能性があります。

3.12.4 Javaメソッドトレースを利用したデバッグ

Javaメソッドトレース機能を利用してデバッグする方法です。
J2EEアプリケーションのメソッドレベルでのトレースを採取することにより、アプリケーションの処理がどこまで正常に行われているか、どの処理で停止または異常が起こっているかを診断する上で有用な情報を得ることができます。
メソッドトレース機能については、“トラブルシューティング集”の“Javaツール機能”－“メソッドトレース機能”を参照してください。

3.13 スナップを利用したデバッグ

スナップは、以下の情報をロギングする機能です。
J2EEアプリケーションを開発するときに、デバッグ情報として使用できます。

- J2EEアプリケーション実行中の各種入出力情報
- J2EEアプリケーションのユーザデバッグ情報

注意

スナップを使用するには、Interstage管理コンソールでIJServerを起動してください。

スナップが出力する情報

スナップには以下の情報が出力されます。

クライアントから呼び出されたEJBアプリケーションのメソッド情報

クライアントアプリケーションから呼び出されたEJBアプリケーションの以下のメソッド情報を出力します。

- メソッド呼出し時の情報
- メソッド復帰時の情報
- メソッドで例外が発生したときの情報

EJBアプリケーションが、以下のすべての条件を満たしている場合にだけ、出力できます。

- インストールガイドに記載されているV6以降のInterstage管理コンソールを使用して配備したEJBアプリケーションであること

本情報が出力されるのは下記メソッド呼出し時だけです。

	Session Bean	Entity Bean
Homeインタフェースメソッド	create remove(handle) remove(primarykey)	create remove(handle) remove(primarykey) findByPrimaryKey find<Enumeration型> find<Collection型> find<Object> ejbHomeメソッド
Remoteインタフェースメソッド	ビジネスメソッド remove	ビジネスメソッド remove
LocalHomeインタフェースメソッド	ビジネスメソッド remove	create remove(primarykey) findByPrimaryKey find<Enumeration型>
Localインタフェースメソッド	ビジネスメソッド remove	ビジネスメソッド remove

詳細は、“3.13.1 クライアントから呼び出されたEJBアプリケーションのメソッド情報”を参照してください。

EJBアプリケーションのメソッド情報

EJBアプリケーションのメソッド呼出し時の以下の情報を出力します。

- メソッド呼出し時の情報
- メソッド復帰時の情報
- メソッドで例外が発生したときの情報

詳細は、“3.13.2 EJBアプリケーションのメソッド情報”を参照してください。

javax.transaction.UserTransactionのAPI情報

javax.transaction.UserTransaction API呼出し時の以下の情報を出力します。

- メソッド呼出し時の情報
- メソッド復帰時の情報
- メソッドで例外が発生したときの情報

詳細は、“3.13.3 javax.transaction.UserTransaction API情報”を参照してください。

データベース操作文情報(Entity Bean形態がCMPの場合のみ)

Entity Bean形態がCMPの場合、コンテナが行っているデータベース操作の以下の情報を出力します。

- データベース操作呼出し時の情報
- データベース操作復帰時の情報
- データベース操作で例外が発生したときの情報

詳細は、“3.13.4 データベース操作文情報”を参照してください。

EJBコンテナのトランザクション制御情報

トランザクション種別がContainerでトランザクション属性がRequiredまたはRequiresNewの場合、コンテナがjavax.transaction.TransactionManagerのAPIを呼び出すときの以下の情報を出力します。

- トランザクションの開始(begin)
- トランザクションの完了(commit/rollback)
- トランザクションにロールバックを指定(setRollbackOnly)
- トランザクションの中断/再開(suspend/resume)

詳細は、“3.13.5 EJBコンテナのトランザクション制御情報”を参照してください。

J2EEアプリケーションのユーザデバッグ情報

J2EEアプリケーションより出力されたデバッグ情報を出力します。詳細は、“3.13.6 J2EEアプリケーションのユーザデバッグ情報”を参照してください。

スナップの出力レベル

出力される情報は、IJServerの起動時に指定する出力レベルの設定により異なります。

出力レベル	出力される内容
1	J2EEアプリケーションメソッドの呼出しシーケンスが確認できます。
2	レベル1情報に加え、メソッド実行時のパラメタ情報、復帰情報が確認できます。Entity Beanの形態がCMPである場合は、データベース操作の情報を出力するため、データの流れやデータベースとの関係が確認できます。
10	J2EEアプリケーションのユーザデバッグ情報が確認できます。
11	レベル1の情報に加え、J2EEアプリケーションのユーザデバッグ情報が確認できます。
12	レベル2の情報に加え、J2EEアプリケーションのユーザデバッグ情報が確認できます。

スナップの環境設定

スナップを取得する場合は、スナップ取得を行うIJServerのワークユニット設定に、以下のように出力レベルを指定します。

パラメタ	値
Java VMオプション(Java Command Option)	-DFJSNAP=出力レベル

スナップは、ディスクの空き容量がない場合を除き、出力レベルに従ってすべての情報が無制限に出力されます。

スナップに関する注意事項

- IJServerに配備されたすべてのJ2EEアプリケーションから出力されるスナップは、コンテナログへ出力されます。
- J2EEアプリケーションをスレッド多重で動作させた場合、すべてのスナップは同一のコンテナログへ出力されます。この場合、スナップがスレッドごとに交錯して出力されますので、スレッド多重では使用しないでください。
- EJBアプリケーションのメソッドの復帰値とパラメタで大量データを使用する場合、メモリ不足エラーが発生する可能性があります。スナップを使用する際は少量データで使用してください。
- J2EEアプリケーション実行中にメモリ不足が発生した場合、スナップ情報は出力されません。
- 環境変数、および値に誤り(スペルミスなど)がある場合、IJServerは起動しますがスナップは出力されません。

3.13.1 クライアントから呼び出されたEJBアプリケーションのメソッド情報

クライアントから呼び出されたEJBアプリケーションのメソッド情報は、クライアントアプリケーションからの呼出し時、復帰時、および例外発生時に出力されます。

出力形式

以下に、出力形式を出力レベルごとに示します。

レベル1

・ メソッド呼出し時

```
日付 時間 :Client Call :Bean名 メソッド名
```

・ メソッド復帰時

```
日付 時間 :Client Return :Bean名 メソッド名
```

・ エラー復帰時

```
日付 時間 :Client Throw :Bean名 メソッド名 例外クラス名 :例外詳細文字列
```

レベル2

・ メソッド呼出し時

```
日付 時間 Client Call :Bean名 メソッド名
Param : パラメタ情報
```

・ メソッド復帰時

```
日付 時間 :Client Return :Bean名 メソッド名
ReturnValue :復帰値情報
ObjectField :フィールド情報
```

・ エラー復帰時

```
日付 時間 :ClientThrow :Bean名 メソッド名 例外クラス名 :例外詳細文字列
```

出力内容

以下に出力される項目と内容について示します。

出力項目	内容	出力レベル	
		1	2
日付	メソッド呼出しの開始/終了の日付を“日/月/年”の形式で示します。	○	○
時間	メソッド呼出しの開始/終了の時間を“時:分:秒.ミリ秒”の形式で出力します。	○	○
Call Return Throw	・ Call :メソッド呼出し時の情報であることを示します。 ・ Return:メソッド復帰時の情報であることを示します。 ・ Throw :メソッド例外時の情報であることを示します。	○	○
Bean名	メソッドを呼び出したEJBアプリケーション名を示します。	○	○
メソッド名	メソッド名を示します。	○	○
例外クラス名	メソッド呼出しで例外が発生した場合の例外クラス名を示します。また、発生した例外に詳細文字列が含まれている場合は、その詳細文字列も出力されます。	○	○

出力項目	内容	出力レベル	
		1	2
パラメタ情報 (Param)	メソッド呼出し時のパラメタ情報(パラメタの型、値)を以下のどちらかの形式で示します。 ・ (型)パラメタ ・ (型)<Object> パラメタがない場合は、項目名のみが出力されます。 配列クラス、java.utilパッケージのHashtableなどは格納されているすべての値を出力します。 また、publicフィールドを持つユーザオブジェクト(Stringを除く)をパラメタとして使用している場合は、“<Object>”を付加し、ObjectField項目の出力を行います。	×	○
復帰値情報 (ReturnValue)	メソッドの復帰値情報(復帰値の型、値)を以下のどちらかの形式で示します。 ・ (型)復帰値 ・ (型)<Object> voidの場合は、項目名だけが出力されます。 配列クラス、java.utilパッケージのHashtableなどは格納されているすべての値を出力します。 また、publicフィールドを持つユーザオブジェクト(Stringを除く)を復帰値として使用している場合は、“<Object>”を付加し、ObjectField項目の出力を行います。	×	○
フィールド情報 (ObjectField)	オブジェクトのpublicなフィールド情報を以下のどちらかの形式で示します。 ・ (型)フィールド名 = フィールド値 ・ (型)フィールド名 = <Object> プリミティブ型とString型の場合は型、変数名、値を出力します。 その他の場合は型、変数名、“<Object>”を出力します。	×	○

○:設定した出力レベルで出力される項目　×:出力されない項目

出力例

以下に、出力例を示します。

・ レベル1:正常終了時

```
23/10/2000 09:49:20.159 : Client Call :SampleBean  business
23/10/2000 09:49:21.229 : Client Return :SampleBean business
```

・ レベル1:異常終了時

```
23/10/2000 09:49:20.159 : Client Call :SampleBean business
23/10/2000 09:49:21.229 : Client Throw :SampleBean business
java.rmi.RemoteException:SampleBean Internal error
```

・ レベル2:正常終了時

```
23/10/2000 09:49:15.454 : Client Call :SampleBean  business
        Param        : (int)1,
        (java.lang.String)"Sample In",
        (java.util.Hashtable)["one", "two"]
23/10/2000 09:49:15.514 : Client Return :SampleBean  business
        ReturnValue : (pack.Sample)pack.Sample@abc123<Object>
```

```
        ObjectField: (int)i = 3,
        (java.lang.String)str = "hello"
```

・ レベル2:異常終了時

```
23/10/2000 09:49:20.159 : Client Call :SampleBean business
        Param : (int)1,
        (java.lang.String)"Sample In",
        (java.util.Hashtable)["one", "two"]
23/10/2000 09:49:21.229 : Client Throw :SampleBean business
java.rmi.RemoteException:SampleBean Internal error
```

3.13.2 EJBアプリケーションのメソッド情報

EJBアプリケーションのメソッド情報は、EJBアプリケーション内の全メソッドの呼出し時、復帰時、および例外発生時に出力されます。

出力形式

以下に、出力形式を出力レベルごとに示します。

レベル1

・ メソッド呼出し時

```
日付    時間    : Call   :Bean名 メソッド名
```

・ メソッド復帰時

```
日付    時間    : Return :Bean名 メソッド名
```

・ エラー復帰時

```
日付    時間    : Throw  :Bean名 メソッド名 例外クラス名: 例外詳細文字列
```

レベル2

・ メソッド呼出し時

```
日付    時間    : Call   :Bean名 メソッド名
   Param       : パラメタ情報
   TranStatus  : トランザクション状態
```

・ メソッド復帰時

```
日付    時間    : Return :Bean名 メソッド名
   ReturnValue : 復帰値情報
   ObjectField : フィールド情報
   TranStatus  : トランザクション状態
```

・ エラー復帰時

```
日付    時間    : Throw  :Bean名 メソッド名 例外クラス名: 例外詳細文字列
   TranStatus  : トランザクション状態
```

出力内容

下表に出力される項目と内容について示します。

出力項目	内容	出力レベル	
		1	2
日付	メソッド呼出しの開始/終了の日付を“日/月/年”の形式で示します。	○	○

出力項目	内容	出力レベル	
		1	2
時間	メソッド呼出しの開始/終了の時間を“時:分:秒.ミリ秒”の形式で出力します。	○	○
Call Return Throw	・ Call :メソッド呼出し時の情報であることを示します。 ・ Return:メソッド復帰時の情報であることを示します。 ・ Throw :メソッド例外時の情報であることを示します。	○	○
Bean名	メソッドを呼び出したEJBアプリケーション名を示します。	○	○
メソッド名	メソッド名を示します。	○	○
例外クラス名	メソッド呼出しで例外が発生した場合の例外クラス名を示します。 また、発生した例外に詳細文字列が含まれている場合は、その詳細文字列も出力されます。	○	○
パラメタ情報 (Param)	メソッド呼出し時のパラメタ情報(パラメタの型、値)を以下のどちらかの形式で示します。 ・ (型)パラメタ ・ (型)<Object> パラメタがない場合は、項目名のみが出力されます。 配列クラス、java.utilパッケージのHashtableなどは格納されているすべての値を出力します。 また、publicフィールドを持つユーザオブジェクト(Stringを除く)をパラメタとして使用している場合は、“<Object>”を付加し、ObjectField項目の出力を行います。	×	○
復帰値情報 (ReturnValue)	メソッドの復帰値情報(復帰値の型、値)を以下のどちらかの形式で示します。 ・ (型)復帰値 ・ (型)<Object> voidの場合は、項目名だけが出力されます。 配列クラス、java.utilパッケージのHashtableなどは格納されているすべての値を出力します。 また、publicフィールドを持つユーザオブジェクト(Stringを除く)を復帰値として使用している場合は、“<Object>”を付加し、ObjectField項目の出力を行います。	×	○
フィールド情報 (ObjectField)	オブジェクトのpublicなフィールド情報を以下のどちらかの形式で示します。 ・ (型)フィールド名 = フィールド値 ・ (型)フィールド名 = <Object> プリミティブ型とString型の場合は型、変数名、値を出力します。 その他の場合は型、変数名、“<Object>”を出力します。	×	○
トランザクション状態 (TranStatus)	以下の情報を出力します。 ・ [出力項目がCallの場合] メソッド呼出し前のトランザクション状態 ・ [出力項目がReturnまたはThrowの場合] メソッド終了後のトランザクション状態 当項目はトランザクションの使用に関係なく出力されます。	×	○

○:設定した出力レベルで出力される項目　×:出力されない項目

出力例

以下に、出力例を示します。

- レベル1:正常終了時

```
23/10/2000 09:49:15.454 : Call   :SampleBean  business
23/10/2000 09:49:15.514 : Return :SampleBean  business
```

- レベル1:異常終了時

```
23/10/2000 09:49:20.159 : Call   :SampleBean  business
23/10/2000 09:49:21.229 : Throw  :SampleBean  business
java.rmi.EJBException: SampleBean Internal error
```

- レベル2:正常終了時

```
23/10/2000 09:49:15.454 : Call   :SampleBean  business
       Param            : (int)1,
                          (java.lang.String)"Sample In",
                          (java.util.Hashtable)["one", "two"]
       TranStatus       : STATUS_ACTIVE
23/10/2000 09:49:15.514 : Return :SampleBean  business
       ReturnValue      : (pack.Sample)pack.Sample@abc123<Object>
       ObjectField      : (int)i = 3,
                          (java.lang.String)str = "hello"
       TranStatus       : STATUS_NO_TRANSACTION
```

- レベル2:異常終了

```
23/10/2000 09:49:20.159 : Call   :SampleBean  business
       Param            : (int) 1,
                          (java.lang.String)"Sample In"
                          (java.util.Hashtable)["one", "two"]
       TranStatus       : STATUS_ACTIVE
23/10/2000 09:49:21.229 : Throw  :SampleBean  business
java.rmi.EJBException: SampleBean Internal error
       TranStatus       : STATUS_MARKED_ROLLBACK
```

3.13.3 javax.transaction.UserTransaction API情報

javax.transaction.UserTransaction API情報は、J2EEアプリケーションからjavax.transaction.UserTransactionメソッドを使用した場合に出力されます。

出力形式

以下に、出力形式を出力レベルごとに示します。

レベル1

- メソッド呼出し時

```
日付    時間    : Call   :javax.transaction.UserTransaction メソッド名
```

- メソッド復帰時

```
日付    時間    : Return :javax.transaction.UserTransaction メソッド名
```

- エラー復帰時

```
日付    時間    : Throw  :javax.transaction.UserTransaction メソッド名
例外クラス名: 例外詳細文字列
```

レベル2

・ メソッド呼出し時

```
日付     時間     : Call   :javax.transaction.UserTransaction メソッド名
    Param        : パラメタ情報
    TranStatus   : トランザクション状態
```

・ メソッド復帰時

```
日付     時間     : Return :javax.transaction.UserTransaction メソッド名
    ReturnValue  : 復帰値情報
    TranStatus   : トランザクション状態
```

・ エラー復帰時

```
日付     時間     : Throw  :javax.transaction.UserTransaction メソッド名
例外クラス名: 例外詳細文字列
    TranStatus   : トランザクション状態
```

出力内容

以下に、出力される項目と内容について示します。

出力項目	内容	出力レベル	
		1	2
日付	メソッド呼出しの開始/終了の日付を“日/月/年”の形式で示します。	○	○
時間	メソッド呼出しの開始/終了の時間を“時:分:秒.ミリ秒”の形式で出力します。	○	○
Call Return Throw	・ Call :メソッド呼出し時の情報であることを示します。 ・ Return:メソッド復帰時の情報であることを示します。 ・ Throw :メソッド例外時の情報であることを示します。	○	○
メソッド名	メソッド名を示します。	○	○
例外クラス名	メソッド呼出しで例外が発生した場合の例外クラス名を示します。また、発生した例外に詳細文字列が含まれている場合は、その詳細文字列も出力されます。	○	○
パラメタ情報 (Param)	メソッド呼出し時のパラメタ情報(パラメタの型、値)を以下の形式で示します。 ・ (型)パラメタ	×	○
復帰値情報 (ReturnValue)	メソッドの復帰値情報(復帰値の型、値)を以下の形式で示します。 ・ (型)復帰値	×	○
トランザクション状態 (TranStatus)	以下の情報を出力します。 ・ [出力項目がCallの場合] メソッド呼出し前のトランザクション状態 ・ [出力項目がReturnまたはThrowの場合] メソッド終了後のトランザクション状態	×	○

○:設定した出力レベルで出力される項目　×:出力されない項目

出力例

以下に、出力例を示します。

・ レベル1:正常終了

```
18/10/2000 18:02:28.647 : Call   :javax.transaction.UserTransaction  getStatus
18/10/2000 18:02:28.647 : Return :javax.transaction.UserTransaction  getStatus
```

・ レベル1:異常終了

```
18/10/2000 18:02:28.577 : Call   :javax.transaction.UserTransaction  getStatus
18/10/2000 18:02:28.607 : Throw  :javax.transaction.UserTransaction  getStatus
javax.transaction.SystemException: Internal error
```

・ レベル2:正常終了

```
18/10/2000 18:02:28.647 : Call   :javax.transaction.UserTransaction  getStatus
       Param            :
       TranStatus       :STATUS_MARKED_ROLLBACK
18/10/2000 18:02:28.647 : Return :javax.transaction.UserTransaction  getStatus
       ReturnValue      :(int)1
       TranStatus       :STATUS_MARKED_ROLLBACK
```

・ レベル2:異常終了

```
18/10/2000 18:02:28.577 : Call   :javax.transaction.UserTransaction  getStatus
       Param            :
       TranStatus       :STATUS_MARKED_ROLLBACK
18/10/2000 18:02:28.607 : Throw  :javax.transaction.UserTransaction  getStatus
javax.transaction.SystemException: Internal error
       TranStatus       :STATUS_MARKED_ROLLBACK
```

3.13.4 データベース操作文情報

データベース操作文情報は以下のメソッドの実行時に出力されます。

・ java.sql.Connectionクラスの prepareStatementメソッド

・ java.sql.PreparedStatementクラスの executeQueryメソッド

・ java.sql.PreparedStatementクラスの executeUpdateメソッド

出力形式

以下に、出力形式を出力レベルごとに示します。

レベル1

出力される情報はありません。

レベル2

・ 呼出し時

```
日付     時間     : Call   :クラス名 メソッド名
    Param        : パラメタ情報
    TranStatus   : トランザクション状態
```

・ 復帰時

```
日付     時間     : Return :クラス名 メソッド名
    ReturnValue  : 復帰値情報
    TranStatus   : トランザクション状態
```

・ エラー復帰時

```
日付     時間     : Throw  :クラス名 メソッド名 例外クラス名: 例外詳細文字列
    TranStatus   : トランザクション状態
```

出力内容

以下に、出力される項目と内容について示します。

出力項目	内容	出力レベル	
		1	2
日付	データベース操作文の開始/終了の日付を“日/月/年”の形式で示します。	×	○
時間	データベース操作文の開始/終了の時間を“時:分:秒.ミリ秒”の形式で出力します。	×	○
Call Return Throw	・ Call :データベース操作文の呼出し時の情報であることを示します。 ・ Return:データベース操作文の復帰時の情報であることを示します。 ・ Throw :データベース操作文の例外時の情報であることを示します。	×	○
クラス名	データベース操作文実行時のクラス名を示します。	×	○
メソッド名	データベース操作文実行時のメソッド名を示します。	×	○
例外クラス名	データベース操作文で例外が発生した場合の例外クラス名を示します。 また、発生した例外に詳細文字列が含まれている場合は、その詳細文字列も出力されます。	×	○
パラメタ情報 (Param)	データベース操作文実行時のSQL文を以下の形式で出力します。 ・ (型)パラメタ	×	○
復帰値情報 (ReturnValue)	データベース操作文の復帰値(復帰値の型、値)を以下の形式で出力します。 ・ (型)フィールド名 = フィールド値	×	○
トランザクション状態 (TranStatus)	以下の情報を出力します。 ・ [出力項目がCallの場合] データベース操作文呼出し時のトランザクション状態 ・ [出力項目がReturnの場合] データベース操作文復帰時のトランザクション状態 ・ [出力項目がThrowの場合] データベース操作文の例外時のトランザクション状態	×	○

○:設定した出力レベルで出力される項目　×:出力されない項目

出力例

以下に、出力例を示します。

レベル1

出力される情報はありません。

レベル2

・ 正常終了

```
23/10/2000 09:49:15.514 : Call    :SampleCMP  java.sql.Connection prepareStatement
       Param             :(java.lang.String)"INSERT INTO CT2.CATEGORY (C_ID,C_NAME)
VALUES (?,?)"
       TranStatus        :STATUS_ACTIVE
```

```
23/10/2000 09:49:15.524 : Return :SampleCMP  java.sql.Connection prepareStatement
       ReturnValue      :(java.sql.PreparedStatement)java.sql.PreparedStatement@1cb0f4
       TranStatus       :STATUS_ACTIVE
```

・ 異常終了

```
23/10/2000 09:49:15.514 : Call   :SampleCMP  java.sql.Connection prepareStatement
       Param            :(java.lang.String)"INSERT INTO CT2.CATEGORY (C_ID,C_NAME)
VALUES (?,?)"
       TranStatus       :STATUS_ACTIVE
23/10/2000 09:49:15.524 : Throw  :SampleCMP  java.sql.Connection prepareStatement
java.sql.SQLException: SQLState received from backend server
       TranStatus       :STATUS_ACTIVE
```

3.13.5 EJBコンテナのトランザクション制御情報

EJBアプリケーションのトランザクション種別がContainerで、トランザクション属性がRequiredまたはRequiresNewの場合、EJBコンテナはjavax.transaction.TransactionManagerインタフェースのメソッドを使用して、トランザクションを制御します。
コンテナのトランザクション制御情報は、EJBコンテナが使用するjavax.transaction.TransactionManagerAPI呼出し時の以下の情報を出力します。

・ トランザクションの開始(begin)

・ トランザクションの完了(commit/rollback)

・ トランザクションにロールバックを指定(setRollbackOnly)

・ トランザクションの中断/再開(suspend/resume)

出力形式

以下に、出力形式を出力レベルごとに示します。

レベル1

・ トランザクション制御開始時

```
日付    時間    : Call   :javax.transaction.TransactionManager メソッド名
```

・ トランザクション制御完了時

```
日付    時間    : Return :javax.transaction.TransactionManager メソッド名
```

・ エラー発生時

```
日付    時間    : Throw  :javax.transaction.TransactionManager メソッド名
例外クラス名: 例外詳細文字列
```

レベル2

・ トランザクション制御開始時

```
日付    時間    : Call   :javax.transaction.TransactionManager メソッド名
   Param       : パラメタ情報
   TranStatus  : トランザクション状態
```

・ トランザクション制御完了時

```
日付    時間    : Return :javax.transaction.TransactionManager メソッド名
   ReturnValue : 復帰値情報
   TranStatus  : トランザクション状態
```

・ エラー発生時

```
日付     時間     : Throw   :javax.transaction.TransactionManager メソッド名
例外クラス名: 例外詳細文字列
    TranStatus   : トランザクション状態
```

出力内容

以下に、出力される項目と内容について示します。

出力項目	内容	出力レベル	
		1	2
日付	トランザクション制御の開始/終了の日付を“日/月/年”の形式で示します。	○	○
時間	トランザクション制御の開始/終了の時間を“時:分:秒.ミリ秒”の形式で出力します。	○	○
Call Return Throw	・ Call :メソッド呼出し時の情報であることを示します。 ・ Return:メソッド復帰時の情報であることを示します。 ・ Throw :メソッド例外時の情報であることを示します。	○	○
メソッド名	呼び出されたメソッド名を示します。	○	○
例外クラス名	メソッド呼出しで例外が発生した場合の例外クラス名を示します。また、発生した例外に詳細文字列が含まれている場合は、その詳細文字列も出力されます。	○	○
パラメタ情報 (Param)	メソッド呼出し時のパラメタ情報(パラメタの型、値)を以下の形式で示します。 ・ (型)パラメタ	×	○
復帰値情報 (ReturnValue)	メソッドの復帰値情報(復帰値の型、値)を以下の形式で示します。 ・ (型)復帰値	×	○
トランザクション状態 (TranStatus)	以下の情報を出力します。 ・ [出力項目がCallの場合] メソッド呼出し前のトランザクション状態 ・ [出力項目がReturnまたはThrowの場合] メソッド終了後のトランザクション状態	×	○

○:設定した出力レベルで出力される項目　×:出力されない項目

出力例

以下に、出力例を示します。

・ レベル1:正常終了

```
18/10/2000 18:02:28.647 : Call   : javax.transaction.TransactionManager  begin
18/10/2000 18:02:28.647 : Return : javax.transaction.TransactionManager  begin
```

・ レベル1:異常終了

```
18/10/2000 18:02:28.577 : Call   : javax.transaction.TransactionManager  begin
18/10/2000 18:02:28.607 : Throw  : javax.transaction.TransactionManager  begin
javax.transaction.SystemException: Internal error
```

・ レベル2:正常終了

```
18/10/2000 18:02:28.647 : Call   : javax.transaction.TransactionManager  begin
       Param            :
       TranStatus       : STATUS_NO_TRANSACTION
18/10/2000 18:02:28.647 : Return : javax.transaction.TransactionManager  begin
       ReturnValue      :
       TranStatus       : STATUS_ACTIVE
```

・ レベル2:異常終了

```
18/10/2000 18:02:28.647 : Call   : javax.transaction.TransactionManager  begin
       Param            :
       TranStatus       : STATUS_NO_TRANSACTION
18/10/2000 18:02:28.647 : Throw  : javax.transaction.TransactionManager  begin
javax.transaction.SystemException: Internal error
       TranStatus       : STATUS_NO_TRANSACTION
```

3.13.6 J2EEアプリケーションのユーザデバッグ情報

J2EEアプリケーションのユーザデバッグ情報は、J2EEアプリケーションからログ出力メソッドを呼び出す時に出力されます。ログ出力メソッドごとに出力されるユーザデバッグ情報の詳細は、“3.13.7 サポート対象ログ出力メソッド”を参照してください。

ここでは、ユーザデバッグ情報をコンテナログへ出力するための方法、出力形式および出力内容を説明します。

ユーザデバッグ情報の出力方法

J2EEアプリケーションのユーザデバッグ情報をコンテナログへ出力するためには、ログ出力メソッドをJ2EEアプリケーションクラス内に記述します。記述手順は以下のとおりです。

1. import文を記述します。
2. デバッグ情報を出力するためのクラスを獲得します。
3. 2.で獲得したクラスを使用して各種ログ出力メソッドの呼出しを行い、J2EEアプリケーション任意のデバッグ情報を出力します。

例

以下に例を示します。

```
package  sample.ejb.entity.bmp;

  import java.rmi.*;
  import javax.ejb.*;
  import java.sql.*;
  import javax.sql.*;
  import javax.naming.*;
  import java.util.logging.*; //ロギングクラスの追加 (1)

  public class EntityBMPUseLogger implements javax.ejb.EntityBean {
     // デバッグ情報を出力するためのクラス(Logger)を獲得
    private static Logger logger = Logger.getLogger("UseLoggerCMP");  //      (2)
    private  boolean isDebug = false;
    ・・・省略・・・
    public void setEntityContext(javax.ejb.EntityContext ctx)
    throws javax.ejb.EJBException, java.rmi.RemoteException {
      ・・・省略・・・
      public EntityBMPUseLoggerPrimaryKey ejbCreate(int No, String Name, int Stock)
      throws javax.ejb.DuplicateKeyException, javax.ejbCreateException,
      javax.ejb.RemoteException {
        if(isDebug) {  // Debug Mode ?
          // スナップにログを出力する
```

```
        logger.log(Level.FINE,"ejbCreate START");  //                    (3)
    }
・・・省略・・・
```

ユーザデバッグ情報以外のログ出力を抑止する方法

J2EEアプリケーションのユーザデバッグ情報を出力する場合(IJServerのJava VMオプションに-DFJSNAP=10もしくは11もしくは12を設定した場合)、コンテナログにIJServerプロセスで動作するモジュールや他社製品のモジュール(JDBCドライバ)でJDKが提供するjava.util.loggingパッケージを使用してロギングしていた情報も出力されます。
J2EEアプリケーションのユーザデバッグ情報以外のログを出力させたくない場合は、以下を定義してください。

定義ファイル格納ディレクトリ	Windows32/64 C:¥Interstage¥EJB¥etc Solaris64 Linux32/64 /opt/FJSVejb/etc
定義ファイル名	FJlogging.properties
変更する定義 (注1)	.level=OFF
追加する定義 (注2)	“Logger名”=ALL 定義例: (Logger名がUseLoggerCMPの場合) UseLoggerCMP.level=ALL

注1)

.levelの初期設定値はALLとなっています。OFFに変更してください。

注2)

J2EEアプリケーションがLogger作成時に使用しているLogger名を指定してください。
J2EEアプリケーションが利用しているオープンソースのフレームワーク、ユーティリティモジュール(例Commons-logging)、またはIJServerプロセスで動作する他社製品(JDBCドライバ等)が、java.util.loggingパッケージを使用して出力しているログの情報を取得したい場合は、公開されているドキュメントを参照してLogger名を確認してから定義を追加してください。

出力形式

J2EEアプリケーションから出力されたユーザデバッグ情報に日付、時間などの情報を加えて出力されます。
各種ログ出力メソッドを使用して出力される形式は、以下の形式です。
使用するAPIによって出力形式は異なります。詳細は、“3.13.7 サポート対象ログ出力メソッド”を参照してください。
それぞれの出力形式を説明します。

- メッセージ(指定文字列)だけを出力

```
日付 時間 : Log Message ログレベル メッセージ
```

- メッセージ(指定文字列) + パラメタ(任意のObject)

```
日付 時間 : Log Message ログレベル メッセージ
Log Param :パラメタ情報
```

- メッセージ(指定文字列) + 例外(任意の例外)

```
日付 時間 : Log Exception ログレベル メッセージ 例外情報
```

出力内容

以下に、出力項目と出力内容を説明します。

出力項目	出力内容
日付	デバッグ情報出力時の日付を“日/月/年”の形式で示します。

出力項目	出力内容
時間	デバッグ情報出力時の時間を“時:分:秒.ミリ秒”の形式で出力します。
Log Message	デバッグ情報であることを示します。
Log Exception	デバッグ情報(例外情報)であることを示します。
ログレベル	デバッグ情報のレベル(ログ出力メソッドへ指定したレベル)を下記文字列で出力します。 "[FINEST]","[FINER]","[FINE]","[CONFIG]","[INFO]","[WARNING]","[SEVERE]"
メッセージ	J2EEアプリケーションが指定したデバッグ情報(任意の文字列)を出力します。
パラメタ情報 (Log Param)	J2EEアプリケーションが指定したパラメタ情報(任意のObject)の型、値を以下のいずれかの形式で出力します。 ・ (型)パラメタ ・ (型)<Object> 配列クラス、java.utilパッケージのHashtableなどは格納されているすべての値を出力します。 publicフィールドを持つユーザオブジェクト(Stringを除く)を復帰値として使用している場合は、“<Object>”を付加し、ObjectField項目の出力を行います。
例外情報	J2EEアプリケーションが指定した例外情報(任意の例外)を出力します。 発生した例外に詳細文字列が含まれている場合は、その詳細文字列も出力されます。
フィールド情報 (ObjectField)	オブジェクトのpublicなフィールド情報を以下の形式で示します。 ・ (型)フィールド名 = フィールド値 ・ (型)フィールド名 = <Object> プリミティブ型とString型の場合は型、変数名、値を出力します。 その他の場合は型、変数名、“<Object>”を出力します。

出力例

以下に、出力例を示します。

- メッセージ(指定文字列)
  メソッドが、logger.log(Level.INFO,"DBAccess start!!");の場合

```
23/10/2000 09:49:15.454 : Log Message: [INFO] DBAccess start!!
```

- メッセージ(指定文字列) + パラメタ(任意のObject)
  メソッドが、logger.log(Level.INFO," prepareStatement ",sql);の場合("sql"は、java.lang.String型とする)

```
23/10/2000 09:49:15.454 : Log Message: [INFO] prepareStatement
Log Param : (java.lang.String)"SELECT * FROM EMP_EJB1 WHERE (ID=?)"
```

- メッセージ(指定文字列) + 例外(任意の例外)
  メソッドが、logger.log(Level.SEVERE"Error!!", ex)の場合("ex"は java.lang.Throwable型とする)

```
23/10/2000 09:49:15.454:LogException:[SEVERE] Error!! java.lang.NullPointerException
```

注意

- J2EEアプリケーションのユーザデバッグ情報を取得するとき以外に、スナップを出力するための記述をプログラム上にしないでください。ログ出力メソッド呼出し時のオーバヘッドにより、性能劣化する場合があります。
  スナップの環境を設定しない場合、または、IJServerの起動時に指定するスナップの出力レベルを1または2にしてス

ナップ出力をした場合、J2EEアプリケーションのデバッグ情報がコンテナログに出力されることはありませんが、プログラミングするときは、注意してください。

- J2EEアプリケーションからjava.util.logging.LogManagerクラスのメソッドを使用しないでください。スナップ機能利用時に正常に動作しない場合があります。
- J2EEアプリケーションのユーザデバッグ情報を取得する場合、Interstageが出力しているデバッグ情報も合わせて出力されますが、動作に問題はありません。

3.13.7 サポート対象ログ出力メソッド

以下に、サポート対象であるログ出力メソッドの一覧を示します。
ログ出力メソッドの詳細は、“Java 2 プラットフォーム API 仕様”を参照してください。

サポートメソッド	出力されるデバッグ情報
config(String msg)	ログレベル="[CONFIG]" メッセージ=msgで指定した文字列
fine(String msg)	ログレベル="[FINE]" メッセージ=msgで指定した文字列
finer(String msg)	ログレベル="[FINER]" メッセージ=msgで指定した文字列
finest(String msg)	ログレベル="[FINEST]" メッセージ=msgで指定した文字列
info(String msg)	ログレベル="[INFO]" メッセージ=msgで指定した文字列
severe(String msg)	ログレベル="[SEVERE]" メッセージ=msgで指定した文字列
warning(String msg)	ログレベル="[WARNING]" メッセージ=msgで指定した文字列
log(Level level, String msg)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列
log(Level level, String msg, Object param1)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 パラメタ情報=param1で指定した情報
log(Level level, String msg, Object[] params)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 パラメタ情報=paramsで指定した情報
log(Level level, String msg, Throwable thrown)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 例外情報=thrownで指定した情報
log(LogRecord record)	LogRecordクラスに設定した以下の情報を出力します。 ログレベル=setLevel()メソッドで設定したレベルに対応する文字列 メッセージ=setMessage()メソッドで設定した文字列 パラメタ情報=setParameters()メソッドで設定した情報 例外情報=setThrown()メソッドで設定した情報 注)パラメタ情報と例外情報が同時に設定されている場合、例外情報のみ出力され、パラメタ情報は出力されません。
logp(Level level, String sourceClass, String sourceMethod, String msg)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列

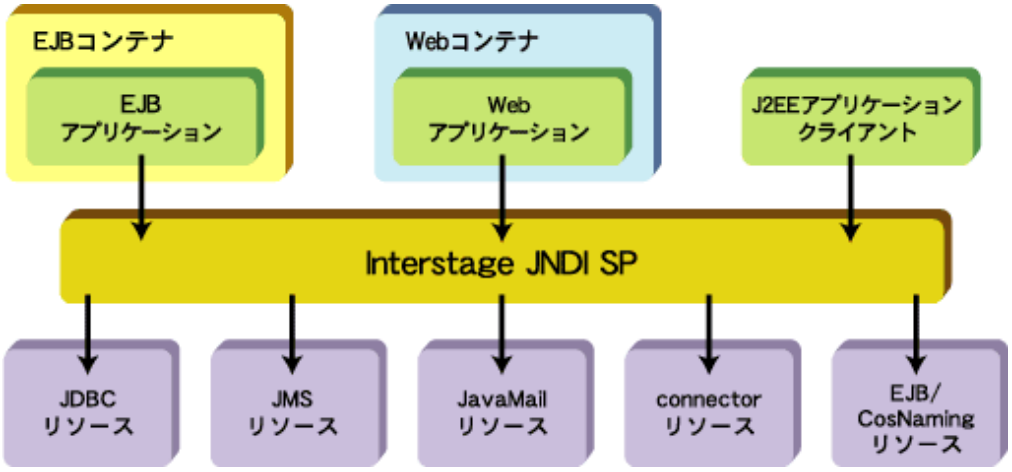
サポートメソッド	出力されるデバッグ情報
logp(Level level, String sourceClass, String sourceMethod, String msg, Object param1)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 パラメタ情報=param1で指定した情報
logp(Level level, String sourceClass, String sourceMethod, String msg, Object[] params)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 パラメタ情報=paramsで指定した情報
logp(Level level, String sourceClass, String sourceMethod, String msg, Throwable thrown)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 例外情報=thrownで指定した情報
logrb(Level level, String sourceClass, String sourceMethod, String bundleName, String msg)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列
logrb(Level level, String sourceClass, String sourceMethod, String bundleName, String msg, Object param1)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 パラメタ情報=param1で指定した情報
logrb(Level level, String sourceClass, String sourceMethod, String bundleName, String msg, Object[] params)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 パラメタ情報=paramsで指定した情報
logrb(Level level, String sourceClass, String sourceMethod, String bundleName, String msg, Throwable thrown)	ログレベル=levelに対応する文字列 メッセージ=msgで指定した文字列 例外情報=thrownで指定した情報
throwing(String sourceClass, String sourceMethod, Throwable thrown)	ログレベル="[FINER]" メッセージ="THROW" 例外情報=thrownで指定した情報

第4章 JNDI

本章では、JNDIの利用方法について説明します。

JNDIサービスプロバイダを使用する場合の環境設定

InterstageではJ2EEオブジェクトへのアクセスを可能にするJNDIサービスプロバイダを実装しています。Interstageが実装するJNDIサービスプロバイダを使用する場合の環境設定については"4.1 JNDIサービスプロバイダの環境設定"を参照してください。



各種オブジェクトを参照可能にするための環境設定

Interstageが実装するJNDIサービスプロバイダにより、J2EEアプリケーションはJNDIのインタフェースを使用して以下のようなオブジェクトを参照することができます。

- ネーミングサービスに登録されたEJB Homeオブジェクト
- 同一プロセス上で運用しているEJBアプリケーションのEJB Local Homeオブジェクト
- 自マシンに定義された、J2EEの各種リソース(JDBCデータソースなど)
- アプリケーションのdeployment descriptorに記述された環境エントリ
- 固定名で提供される特定のオブジェクト(UserTransactionおよびORB)

JNDIで参照可能なオブジェクトは以下です。

カテゴリ	オブジェクト名	Webアプリケーション	EJBアプリケーション	J2EEアプリケーションクライアント	アプレット
EJBオブジェクト	EJB Homeオブジェクト	○	○	○	○
	EJB Local Homeオブジェクト	○	○	×	×
リソース参照	JDBCデータソース	○	○	○	×
	JMSコネクションファクトリ	○	○	○	×

カテゴリ	オブジェクト名	Webアプリケーション	EJBアプリケーション	J2EEアプリケーションクライアント	アプレット
	JavaMailメールセッション	○	○	○	×
	URL(Uniform Resource Locator)	○	○	○	×
	connectorコネクションファクトリ	○	○	×	×
リソース環境参照	JMS Destination	○	○	○	×
その他	環境エントリ	○	○	○	×
	UserTransaction	○	○	○	×
	ORB	○	○	○	○

deployment descriptorの記述

参照するオブジェクトの情報を、各アプリケーションのdeployment descriptorファイルに記述してください。deployment descriptorへの記述方法については“4.9 deployment descriptorファイルへの記述”を参照してください。
ただし、アプレットの場合は、EJBのみ参照可能であるためdeployment descriptorの指定等は必要ありません。lookupの引数にEJBアプリケーション名を指定するだけで、EJB Homeオブジェクトを参照可能です。

オブジェクトの参照方法

deployment descriptorに記述されたオブジェクトについては、JNDIインタフェースのlookupメソッドを使用してJ2EEアプリケーションから参照することが可能です。詳細は“4.10 オブジェクトの参照方法”を参照してください。

名前変換機能

アプリケーションの記述と運用環境の実名が異なる場合には名前変換機能を使用して、deployment descriptorの参照名と運用環境の実名を対応づけることが可能です。このように、名前変換ファイルを使用して、deployment descriptorファイルの参照名と運用環境の実名の対応関係を定義すると、運用環境に依存しないアプリケーションを作成することができます。名前変換機能については“4.11 名前変換機能”を参照してください。

UserTransactionの使用方法

lookupしたUserTransactionオブジェクトを使用してトランザクションの開始と終了を制御することが可能です。UserTransactionの使用方法については“4.12 UserTransactionインタフェースを使用したトランザクション制御”を参照してください。

4.1 JNDIサービスプロバイダの環境設定

Interstageが提供するJNDIサービスプロバイダ(以降、JNDI SP)を使用する場合の環境設定について以下に記述します。

- Webアプリケーション
  デフォルトでInterstageのJNDI SPが動作するため、環境設定の必要はありません。
- EJBアプリケーション
  デフォルトでInterstageのJNDI SPが動作するため、環境設定の必要はありません。
- J2EEアプリケーションクライアント
  J2EEアプリケーションクライアントの場合に必要な設定については、“4.1.1 J2EEアプリケーションクライアント”を参照してください。
- アプレット
  アプレットの場合に必要な設定については、“4.1.2 アプレット”を参照してください。

4.1.1 J2EEアプリケーションクライアント

JNDI環境プロパティについて

J2EEアプリケーションクライアントの場合は、JNDI環境プロパティを設定します。
JNDI環境プロパティとは、アプリケーションからJNDI SPへアクセスするために(new javax.naming.InitialContext()時に)生成する、InitialContextの初期化処理に使用される環境プロパティです。環境プロパティは、以下に示すファイルまたは引数で指定します。なお、環境プロパティが重複して指定された場合は、以下の順で上書きされます(“3”で指定した環境プロパティが最も優先されます)。

1. FJndi.propertiesファイル
   環境プロパティ“java.naming.factory.initial”、“com.fujitsu.interstage.isas.SystemName”は指定できません。
   FJndi.propertiesファイルは、以下に配置します。

   Windows32/64

   C:¥Interstage¥J2EE¥etc

   Solaris64

   /etc/opt/FJSVj2ee/etc

   Linux32/64

   /etc/opt/FJSVj2ee/etc
2. javax.naming.InitialContext(Hashtable environment)引数
3. アプリケーション起動時のコマンドラインでの引数(-D)

ただし、環境プロパティの“java.naming.factory.initial”については、以下の順で上書きされます(“2”で指定された環境プロパティが最も優先されます)。

1. アプリケーション起動時のコマンドラインでの引数(-D)
2. javax.naming.InitialContext(Hashtable environment)の引数

JNDI環境プロパティに設定可能な値

JNDI環境プロパティに設定可能な値を説明します。

ポイント

Windows32/64

環境プロパティ、および“java.naming.factory.initial”の値の文字列は、大文字小文字が区別されます。

Solaris64 Linux32/64

環境プロパティ、および値の文字列は、大文字小文字が区別されます(“VerificationMethod”の値、および“com.fujitsu.ObjectDirector.CORBA.GlobalTransactionMode”の値を除きます)。

各環境プロパティについて、以下に説明します。

java.naming.factory.initial

JNDI SPへアクセスするためのInitialContextファクトリクラス名として以下の値を指定します。

－ com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient

注) 本環境プロパティは、new javax.naming.InitialContext(Hashtable environment)の引数environment、またはアプリケーション起動時のコマンドラインの引数(-D)のいずれかで、必ず指定する必要があります。

FJUserID

ディレクトリサービスのユーザ認証で使用するユーザ名を指定します。省略時はユーザ認証を行いません。
注) 必ず、FJPasswordと対で指定します。

FJPassword

ディレクトリサービスのユーザ認証で使用するパスワードを指定します。省略時はユーザ認証を行いません。
注) 必ず、FJUserIDと対で指定します。

com.fujitsu.interstage.j2ee.DeploymentDescriptorClient

J2EEアプリケーションクライアントのdeployment descriptorファイル名をフルパスで指定します。

注) J2EE 1.4以降のdeployment descriptorファイルを指定する場合、JDK/JRE 5.0以降を使用してください。

EBEproperties

名前変換ファイル名を指定します(パス指定は不可)。省略時は、名前変換を行いません。

HTTPGW

HTTPトンネリングを処理するゲートウェイを指定します。省略時は、HTTPトンネリングを使用しません。

書式

－ Interstage HTTP Serverの場合

```
https://ホスト名/URL名  (SSL通信を行う場合)
http://ホスト名/URL名  (SSL通信を行わない場合)
```

－ Internet Information Servicesの場合 Windows32/64

```
https://ホスト名/cgi識別名/ゲートウェイ名  (SSL通信を行う場合)
http://ホスト名/cgi識別名/ゲートウェイ名  (SSL通信を行わない場合)
```

ホスト名

HTTP-IIOPゲートウェイが動作するWebサーバを指定します。
指定可能なホスト名の形式を以下に示します。

- SSL通信を行う場合

```
ホスト名・IPv4アドレス
ホスト名・IPv4アドレス:ポート番号
```

注) IPv6環境ではSSL機能が使用できないため、IPv6形式のアドレスは指定できません。

- SSL通信を行わない場合

```
ホスト名・IPv4アドレス
ホスト名・IPv4アドレス:ポート番号
[IPv6アドレス]
[IPv6アドレス]:ポート番号
```

注) IPv6形式のアドレスは、大カッコ(“[” と “]”)で囲む必要があります。

URL名

Interstage HTTP Serverの場合、od-httpgwを指定します。URL名にはLocationディレクティブのURLを指定します(詳細は“Interstage HTTP Server 運用ガイド”参照)。

cgi識別名

Internet Information Servicesの場合、「仮想ディレクトリ」のエイリアス名を指定します。

ゲートウェイ名

Internet Information Servicesの場合、ODhttp.dll(HTTP-IIOPゲートウェイ)を指定します。

VerificationMethod

deployment descriptorファイルおよび名前変換ファイルのParserによる検証方法を、以下のどちらかで指定します。

- Well-formed:ウェルフォームドXML文書か否かの検証のみ行う。
- DTD　　:ウェルフォームドXML文書か否かの検証およびDTDによる検証を行う。(デフォルト)

com.fujitsu.ObjectDirector.CORBA.GlobalTransactionMode Windows32/64 Linux32/64

分散トランザクション制御を行うか否かを、以下のどちらかで指定します。

- True :行う
- False:行わない(デフォルト)

注意

Javaのシステムプロパティ“java.home”は、アプリケーション起動時のコマンドラインの引数で変更できます。ただし、orb.propertiesが参照できなくなり、動作異常を招く場合があるため、変更には、十分注意してください。

Windows32/64

例:java -Djava.home=x:¥aaaa¥bbbb myapplication

Solaris64 Linux32/64

例:java -Djava.home=/aaaa/bbbb myapplication

例

FJjndi.propertiesファイルの記述例

Windows32/64

deployment descriptorファイル名:C:¥env¥application-client.xml
名前変換ファイル名:ClientebeProperties.xml

```
com.fujitsu.interstage.j2ee.DeploymentDescriptorClient=C:¥env¥application-client.xml
EBEproperties=ClientebeProperties.xml
```

Solaris64

deployment descriptorファイル名:/export/home/j2eeapl/application-client.xml
名前変換ファイル名:ClientebeProperties.xml

```
com.fujitsu.interstage.j2ee.DeploymentDescriptorClient=/export/home/j2eeapl/application-
client.xml
EBEproperties=ClientebeProperties.xml
```

Linux32/64

deployment descriptorファイル名:/home/j2eeapl/application-client.xml
名前変換ファイル名:ClientebeProperties.xml

```
com.fujitsu.interstage.j2ee.DeploymentDescriptorClient=/home/j2eeapl/application-client.xml
EBEproperties=ClientebeProperties.xml
```

4.1.2 アプレット

Javaアプリケーションとの違い

EJBクライアントを使用してクライアントアプリケーションをJavaアプレットとして開発する場合は、Javaアプリケーションと以下が違います。

- EJBアプリケーションのオブジェクトの所在をネーミングサービスに問い合わせる lookup処理

Javaアプレットでのlookup処理の記述方法

以下に、Javaアプレットでのlookup処理の記述例を示します。

```
// InitialContext獲得
Hashtable env = new Hashtable();                                    ・・・・1
env.put("java.naming.factory.initial",
        "com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient");  ・・・・1
env.put("java.naming.applet", this);                                ・・・・1
javax.naming.Context ic = new javax.naming.InitialContext( env);    ・・・・2
// lookup
java.lang.Object Obj = (java.lang.Object)ic.lookup("SampleBean");   ・・・・3
// homeのnarrow()
h = (SampleHome)javax.rmi.PortableRemoteObject.narrow( Obj, SampleHome.class);  ・・4
```

1. Contextを作成するときの環境情報を設定します。このとおりに記述してください。
2. lookupするためのContextを作成します。このとおりに記述してください。
3. lookupを行います。引数にEJBアプリケーション名を指定してください。lookupに失敗した場合、javax.naming.NameNotFoundException例外が発生します。
   失敗の理由は当例外の詳細メッセージとして通知されます。
   lookupに失敗したときの対処方法については、“29.9 EJBサービス使用時の異常”を参照してください。
4. lookupしたオブジェクトをnarrowします。javax.rmi.PortableRemoteObject.narrowを発行してください。

注意

Portable-ORBを使用しないでアプレットを利用する場合は、使用するEJBアプリケーションのクライアント配布物をWebサーバからダウンロードして使用することはできません。
クライアント配布物はクライアント環境に複写して、複写先のjarまたはフォルダをCLASSPATHに設定して使用してください。

参照

Javaアプレットの開発の詳細については、“15.6 Javaアプレットを使用する場合(プレインストール型Javaライブラリ)”、“15.7 Javaアプレットを使用する場合(Portable-ORB)”を参照してください。

Portable-ORBを使用する場合は、“4.2.1 クライアント環境での環境設定”を参照してください。

ポリシーファイルの設定

Javaアプレットを実行する場合には、実行するマシンごとにJBKプラグインが使用するポリシーファイルへの設定が必要です。設定する情報は以下のとおりです。

- EJBサービスのクラス
- JDKのクラス

例

設定例を以下に記載します。JBKプラグインが使用するポリシーファイルについての詳細は、Interstage Studioの“J Business Kit オンラインマニュアル”を参照してください。

- EJBサービスのクラス

  (Interstage インストールフォルダ : C:¥Interstageとした場合)
  grant codeBase "file:/C:/Interstage/EJBCL/LIB/-" {
  permission java.security.AllPermission;
  };

- JDKのクラス

  (Interstageインストールフォルダ : C:¥ Interstageとした場合)

  ー JDK8を使用する場合
  grant codeBase "file:/C:/Interstage/JDK8/jre/lib/-" {
  permission java.security.AllPermission;
  };

  ー JRE8を使用する場合
  grant codeBase "file:/C:/Interstage/JRE8/lib/-" {
  permission java.security.AllPermission;
  };

注意

旧バージョンと同様にJavaアプリケーションを実行する場合でも、システムプロパティに以下を指定するとEJBを参照することが可能ですが、通常はJ2EEアプリケーションクライアントで説明したように、JNDI SPを指定してください。

- java.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient

4.2 EJBを参照する場合の環境設定

配備されているEJBアプリケーションのEJB Local Home、または、EJB Homeのオブジェクトリファレンスを取得することができます。
EJB Local Homeのオブジェクトリファレンスは、同一IJServerに配備されたEJBアプリケーションのオブジェクトを直接取得します。
EJB Homeのオブジェクトリファレンスは、ネーミングサービスから取得します。

詳細は、“3.5 J2EEアプリケーションの配備と設定”を参照してください。
IJServer(Web + EJB[1VM])に配備するEJBは、ネーミングサービスに登録されません。そのため、配備したIJServer以外からEJBを参照することはできません。

下記の製品では、サーバの場合、自マシンにおいてデフォルトで動作するネーミングサービスからオブジェクトリファレンスを取得します。参照するネーミングサービスを変更するときは、Interstage管理コンソールの[システム] > [環境設定]タブ > [詳細設定] > [ネーミングサービス詳細設定] を設定してください。詳細はInterstage管理コンソールのヘルプを参照してください。

- Interstage Application Server Enterprise Edition

クライアントの場合、参照するネーミングサービスの設定が必要です。詳細は“チューニングガイド”の“CORBAサービスの動作環境ファイル”の“inithost/initial_hosts”を参照してください。

Interstageのクライアント機能をインストールしている場合、環境設定が必要です。詳細は、“4.2.1 クライアント環境での環境設定”を参照してください。ただし、IJServerでは起動時に自動的に設定されるため、環境設定は必要ありません。

以下の場合、EJB Homeのオブジェクトリファレンスを取得するには、参照対象のEJBアプリケーションが配備された時に出力されるクライアント配布物にクラスパスを設定する必要があります。

- IJServerに配備されたJ2EEアプリケーションから別のIJServerに配備されたEJBアプリケーションをlookupする場合
- J2EEアプリケーションクライアントまたはアプレットからEJBアプリケーションをlookupする場合

クライアント配布物の設定方法

IJServerごとのクライアント配布物の出力先は以下のとおりです。
クライアント配布物を、EJBアプリケーション呼出し元が参照可能なディレクトリへコピーしてください。EJBアプリケーション呼出し元が別のマシンの場合は、呼出し元のマシンへクライアント配布物をコピーしてください。
EJBアプリケーション呼出し元のクラスパスに、コピーしたクライアント配布物のパスを設定してください。

Windows32/64

C:¥Interstage¥J2EE¥var¥deployment¥ijserver¥[IJServer名]¥distribute¥[配備したejb-jarファイル名]¥[クライアント配布物のjar名]

Solaris64 Linux32/64

/opt/FJSVj2ee/var/deployment/ijserver/[IJServer名]/distribute/[配備したejb-jarファイル名]/[クライアント配布物のjar名]

※ [クライアント配布物のjar名]は配備したejb-jarファイル名の“.”を“_”に置換し、“_client.jar”を付加した名前です。

例

配備したejb-jarファイル名が“Sample.jar”の場合、クライアント配布物のjar名は“Sample_jar_client.jar”になります。

注意

クライアント配布物をコピーせず、直接他のIJServerなどのJava VMが参照した場合、上書き配備に失敗することや配備解除でファイルが残存することがあります。

4.2.1 クライアント環境での環境設定

J2EEアプリケーションクライアントやアプレットを動作させる場合、以下の環境設定が必要です。環境変数の設定を確認し、必要に応じて設定してください。
IJServerでは起動時に自動的に設定されるため、環境設定は必要ありません。

- CORBAサービスの設定
- ORBの指定
- 環境変数の設定

CORBAサービスの設定

CORBAサービスをインストールした後に以下の情報設定を行ってください。詳細は、“運用ガイド(基本編)”の“Interstageの環境設定”を参照してください。

ホスト名の定義

クライアントパッケージを使用している場合、inithostファイル内に、ネーミングサービスを起動しているホスト名の定義を行ってください。
以下のファイルにネーミングサービスが存在するホスト名とポート番号を記述する必要があります。

C:¥Interstage¥ODWIN¥etc¥inithost

[定義方法]

形式 ： ホスト名 ポート番号
記述例: hostname 8002

configファイルの修正

サーバパッケージまたはInterstage Studioを使用している場合、configファイル内の以下のステートメントについて、初期値を超えて動作する場合は、以下の加算値を加算してください。
なお、設定数値の初期値などについては、“チューニングガイド”を参照してください。

ステートメント	加算値
max_processes	追加するクライアントアプリケーションのプロセス数

ORBの指定

アプリケーションを起動するための環境設定として、使用するORB(Object Request Broker)を指定する必要があります。使用するORBを、以下の方法で指定してください。

- アプリケーション起動時のORBの指定方法
- 環境設定ファイルでのORBの指定方法

上記の方法により以下の値を設定することで、ORBを指定します。

プロパティ名	設定値
org.omg.CORBA.ORBClass	com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass	com.fujitsu.ObjectDirector.CORBA.SingletonORB
javax.rmi.CORBA.StubClass	com.fujitsu.ObjectDirector.rmi.CORBA.StubDelegateImpl
javax.rmi.CORBA.UtilClass	com.fujitsu.ObjectDirector.rmi.CORBA.UtilDelegateImpl
javax.rmi.CORBA.PortableRemoteObjectClass	com.fujitsu.ObjectDirector.rmi.CORBA.PortableRemoteObjectDelegateImpl

アプリケーション起動時のORBの指定方法

Javaアプリケーション実行時に、javaコマンドのパラメタとして使用するORBを指定します。
以下のように、-Dオプションに続けて必要な情報を記述してください。

```
java -Dorg.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB -
Dorg.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB -
Djavax.rmi.CORBA.StubClass=com.fujitsu.ObjectDirector.rmi.CORBA.StubDelegateImpl -
Djavax.rmi.CORBA.UtilClass=com.fujitsu.ObjectDirector.rmi.CORBA.UtilDelegateImpl -
Djavax.rmi.CORBA.PortableRemoteObjectClass=com.fujitsu.ObjectDirector.rmi.CORBA.PortableRemoteObjectDelegateImpl <アプリケーションのクラス名>
```

環境設定ファイルでのORBの指定方法

使用するORBを記述したテキストファイル(ファイル名:orb.properties)を作成し、Javaのシステムプロパティ“java.home”に設定されているディレクトリ配下のlibに格納してください。
クライアントパッケージまたはInterstage Studioを使用して、JBKがインストールされている状態では、以下に格納します。

```
<JBKインストールディレクトリ>¥jre¥lib (注1)
注1)J2EEアプリケーションクライアントの場合のみ
```

例

orb.propertiesファイルの設定例

```
org.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
javax.rmi.CORBA.StubClass=com.fujitsu.ObjectDirector.rmi.CORBA.StubDelegateImpl
```

```
javax.rmi.CORBA.UtilClass=com.fujitsu.ObjectDirector.rmi.CORBA.UtilDelegateImpl
javax.rmi.CORBA.PortableRemoteObjectClass=com.fujitsu.ObjectDirector.rmi.CORBA.Portab
leRemoteObjectDelegateImpl
```

環境変数の設定

Windows32/64

環境変数	設定値
CLASSPATH	C:¥Interstage¥J2EE¥lib¥isj2ee.jar (注1) C:¥Interstage¥ODWin¥etc¥class¥ODjava4.jar C:¥Interstage¥EJB¥lib¥fjcontainer94.jar (注2)

Solaris64

環境変数	設定値
CLASSPATH	/opt/FJSVj2ee/lib/isj2ee.jar (注1) /opt/FSUNod/etc/class/ODjava4.jar /opt/FJSVejb/lib/fjcontainer94.jar
LD_LIBRARY_PATH	/opt/FSUNod/lib

Linux32/64

環境変数	設定値
CLASSPATH	/opt/FJSVj2ee/lib/isj2ee.jar (注1) /opt/FJSVod/etc/class/ODjava4.jar /opt/FJSVejb/lib/fjcontainer94.jar
LD_LIBRARY_PATH	/opt/FJSVod/lib

注1) J2EEアプリケーションクライアントのみ必要
注2) EJBサービスの場合です。EJBクライアントの場合は、以下を指定してください。

- C:¥Interstage¥EJBCL¥lib¥fjcontainer94.jar

4.3 JDBC(データベース)を参照する場合の環境設定

JDBCデータソースを使用するための環境設定を行います。

環境設定を行う前に、使用するデータベースのマニュアルに従って、JDBCドライバをインストールしてください。

- Symfowareを使用する場合の環境設定

  以下の“使用するデータソースの種類”によって環境設定方法が異なります。

  ー Interstageのコネクションプーリングを使用する

  ー Symfowareのコネクションプーリングを使用する

  ー Postgresを使用する

  本製品の8.0まではSymfowareのコネクションプーリングを使用するデータソースのみ使用できましたが、本製品では、Interstageのコネクションプーリングを使用するデータソースが使用できます。Interstageのコネクションプーリングを使用することでJDBCネーミングサービスの起動などの操作が不要となり、Interstageが提供するコネクションプーリング機能を使用できるため、Interstageのコネクションプーリングを使用することをお勧めします。

注意

- 誤ったデータ資源名を指定した場合、コネクションはプーリングされますが、アプリケーションの実行時にエラーとなります。
- Windows32/64 Solaris64
  旧バージョンのSymfowareを使用する場合の環境設定については、“Symfoware Client JDBCドライバ オンラインマニュアル”を参照してください。

- Enterprise Postgresを使用する場合の環境設定

  "Symfowareを使用する場合の環境設定(Postgresを使用する)"を参照してください。
- Oracleを使用する場合の環境設定
- SQL Serverを使用する場合の環境設定
- PostgreSQLを使用する場合の環境設定

また、上記以外で開発中にアプリケーションの動作確認をする場合に、一時的にサポートしないデータベースと接続確認を行う場合には汎用定義のデータソースを使用できます。

- 汎用定義のデータソースを使用する場合の環境設定

注意

IJServerワークユニットでJDBCドライバを使用する場合、J2EEプロパティ、またはワークユニットのクラスパスに、JDBCドライバのパスを指定します。

J2EEプロパティのクラスパスをInterstage管理コンソールの[システム] > [環境設定] > [J2EEプロパティ]の"クラスパス"で指定する場合、使用できる文字に制約があります。このため、J2EEプロパティのクラスパスにJDBCドライバのパスを指定する可能性がある場合、JDBCドライバのインストール先には、Interstage管理コンソールの[システム] > [環境設定] > [J2EEプロパティ]の"クラスパス"で使用できない文字を含めないようにしてください。

使用できない文字については、Interstage管理コンソールのヘルプを参照してください。

Interstage管理コンソールのデータベース接続テスト

Interstage管理コンソールを使用して、データベースへ接続するための各設定情報が正しいか接続テストができます。
操作は、[リソース] > [JDBC] > [新規作成]画面、または、[リソース] > [JDBC] > [環境設定]画面で行います。

データベース接続テストを行う場合、以下の設定が必要です。

クラスパス、パス、ライブラリパス

以降に記載の各データベースの環境設定を参照し、必要なパスをInterstage管理コンソールの[システム] > [環境設定] > [J2EEプロパティ]に設定してください。

環境変数 Solaris64 Linux32/64

データベースのJDBCドライバが環境変数の設定を必要とする場合、以降に記載の各データベースの環境設定を参照し、Interstage JMXサービス起動前に、必要な環境変数を設定してください。環境変数の設定は、Interstage JMXサービスの起動方法の違いにより異なります。それぞれの起動方法による環境変数の設定方法を以下に示します。

- isjmxstartコマンドによる起動
  環境変数設定後、isjmxstopコマンドでInterstage JMXサービスを停止し、isjmxstartコマンドで起動することにより、設定した環境変数が有効になります。
- システム初期化スクリプトS95isjmxstartによる起動
  OS起動時に自動的に環境変数の設定を有効にする場合、システム初期化スクリプトS95isjmxstartを修正し、スク

リプトに環境変数の設定を記述してください。次回OS起動時から有効になります。S95isjmxstartの格納場所等については、“リファレンスマニュアル(コマンド編)”の“S95isjmxstart”を参照してください。

ー 起動用unitファイルFJSVisjmx_start.serviceによる起動 (RHEL7/RHEL8の場合)
OS起動時に自動的に環境変数の設定を有効にする場合、起動用unitファイルFJSVisjmx_start.serviceに対して、環境変数の設定を追加してください。次回OS起動時から有効になります。設定方法については、“チューニングガイド”の“RHEL7/RHEL8のunitファイルでの環境定義”を参照してください。

注意

- 本接続テストは、JDBCデータソースの定義内容が正しいかを確認します。
すべてのIJServerの環境設定が正しいことを、テストしていません。IJServerの環境変数、アクセス権などは別途確認してください。
- データベースにOracleを使用している場合、DB接続テストでは接続プロパティおよび暗黙的接続キャッシュのプロパティの妥当性はテストされません。接続プロパティおよび暗黙的接続キャッシュのプロパティが正しく設定されているかどうかは、Oracleのマニュアルを参照してください。
- Oracle RAC機能を使用する場合、アドレスリストのDBサーバにアクセスしコネクションを取得します。定義されているDBサーバが1つでも正しく接続することができればエラーにはならないため、すべてのホスト名の確認にはなりません。アドレスリストの設定については、DB接続テスト機能と合わせて、指定した値が正しいか確認してください。ホスト名を指定してOSが提供するpingコマンドを実行し、正しいIPアドレスから応答があれば、ホスト名が正しく設定されていることを確認できます。
- SymfowareのDB接続テストを実施した場合、“データ資源名”が正しいかどうかはチェックできません。

4.3.1 Symfowareを使用する場合の環境設定(Interstageのコネクションプーリングを使用する)

以下の環境設定が必要です。

- 環境変数の設定
- Interstageへのデータソース登録

また、Interstageと別のサーバシステム上にあるSymfowareにアクセスする場合は、上記に加え、以下の作業を行ってください。

- Symfowareがインストールされているサーバシステムの環境設定

1.環境変数の設定

設定する環境変数の設定項目と動作環境ごとの設定方法は、以下のとおりです。

環境変数の設定項目

以下の項目を設定してください。<>は可変情報です。

詳細は、Symfowareのマニュアルを参照してください。

Windows32/64

設定項目	設定値
パス	<Symfoware Server クライアント機能インストール先ディレクトリ>¥JDBC¥fjjdbc¥bin <Windows(R)システムディレクトリ>¥ESQL¥BIN
クラスパス	<Symfoware Server クライアント機能インストール先ディレクトリ>¥JDBC¥fjjdbc¥lib¥<JDBCドライバモジュール名>(注1)

Solaris64

設定項目	設定値
ライブラリパス	<FSUNrdb2bインストール先ディレクトリ>/FSUNrdb2b/lib (注2) <Symfoware Server クライアント機能インストール先ディレクトリ>/FJSVsymjd/fjjdbc/bin (注2)
クラスパス	<Symfoware Server クライアント機能インストール先ディレクトリ>/FJSVsymjd/fjjdbc/lib/<JDBCドライバモジュール名> (注1)(注2)

Linux32/64

設定項目	設定値
ライブラリパス	<FJSVrdb2bインストール先ディレクトリ>/FJSVrdb2b/lib (注2) <Symfoware Server クライアント機能インストール先ディレクトリ>/FJSVsymjd/fjjdbc/bin (注2)
クラスパス	<Symfoware Server クライアント機能インストール先ディレクトリ>/FJSVsymjd/fjjdbc/lib/<JDBCドライバモジュール名> (注1)(注2)

注1) JDBCドライバモジュール名は、対応するJDBC規約のバージョンによって異なります。使用するJDBCドライバモジュールのファイル名を指定してください。

注2) 以下のディレクトリのデフォルトは、/optです。

- FSUNrdb2bインストール先ディレクトリ
- FJSVrdb2bインストール先ディレクトリ
- Symfoware Server クライアント機能インストール先ディレクトリ

IJServerを使用する場合の設定方法

“IJServerを使用する場合の環境変数の設定方法”を参照して設定してください。

クライアント環境の場合の設定方法

クライアント環境でJ2EEアプリケーションクライアントを動作させる場合は、環境変数に設定してください。以下に設定例を記載します。

Windows(R)上の環境設定でシステム環境変数に設定する場合の例 Windows32/64

[スタート]－[コントロールパネル]－[システムとセキュリティ]－[システム]－[システムの詳細設定]の環境変数ボタンをクリックしてください。

注意

Windows Server(R) 2012の場合の説明です。使用するOSにより操作方法は異なります。

コマンドでシステム環境変数にfjsymjdbc3.jarを設定する場合の例

Windows32/64

Symfoware Server クライアント機能インストール先ディレクトリがC:¥SFWCLNT、Windows(R)システムディレクトリがC:¥Windowsの場合

```
set PATH=C:¥Windows¥ESQL¥BIN;%PATH%
set PATH=C:¥SFWCLNT¥JDBC¥fjjdbc¥bin;%PATH%
set CLASSPATH=C:¥SFWCLNT¥JDBC¥fjjdbc¥lib¥fjsymjdbc3.jar;%CLASSPATH%
```

Solaris64

Symfoware Server クライアント機能インストール先ディレクトリ、およびFSUNrdb2bインストール先ディレクトリが/optの場合

setenv LD_LIBRARY_PATH /opt/FSUNrdb2b/lib:${LD_LIBRARY_PATH}
setenv LD_LIBRARY_PATH /opt/FJSVsymjd/fjjdbc/bin:${LD_LIBRARY_PATH}
setenv CLASSPATH /opt/FJSVsymjd/fjjdbc/lib/fjsymjdbc3.jar:${CLASSPATH}

Linux32/64

Symfoware Server クライアント機能インストール先ディレクトリ、およびFJSVrdb2bインストール先ディレクトリが/optの場合
setenv LD_LIBRARY_PATH /opt/FJSVrdb2b/lib:${LD_LIBRARY_PATH}
setenv LD_LIBRARY_PATH /opt/FJSVsymjd/fjjdbc/bin:${LD_LIBRARY_PATH}
setenv CLASSPATH /opt/FJSVsymjd/fjjdbc/lib/fjsymjdbc3.jar:${CLASSPATH}

2.Interstageへのデータソース登録

Interstage管理コンソールで、データソースを定義します。詳細は、Interstage管理コンソールのヘルプを参照してください。
また、isj2eeadminコマンドを使用して変更することもできます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

3.Symfowareがインストールされているサーバシステムの環境設定

Interstageと別のサーバシステム上にあるSymfowareにアクセスするために使用する接続形態を“**RDB2_TCP**”と呼びます。
RDB2_TCPでSymfowareに接続するためには、以下の作業が必要です。
以下の作業は、Interstageと同一のサーバシステム上にあるSymfowareにアクセスする場合は不要です。

- RDB2_TCP接続用のパラメタの設定
- RDB2_TCPのポート番号の設定

RDB2_TCP接続用のパラメタの設定

Symfowareのシステム用動作環境ファイル内に、以下のRDB2_TCP接続用のパラメタを追加してください。

－ MAX_CONNECT_TCP = (n)
n ： 最大接続数(省略値は0)

参考

システム用動作環境ファイルは、Symfowareインストール時に指定した場所に格納されています。格納場所を指定しないでインストールした場合は、以下の場所に格納されています。

Windows32/64

[Symfowareがインストールされているドライブ]:¥SFWETC¥RDB¥ETC¥UXPSQLENV

Solaris64

/opt/FSUNrdb2b/etc/fssqlenv

Linux32/64

/opt/FJSVrdb2b/etc/fssqlenv

注意

システム用動作環境ファイル内にMAX_CONNECT_TCPを設定していない、または最大接続数に0が指定されている場合、J2EEアプリケーション実行時に、Symfoware ODBCドライバのエラーが出力されます。出力されるエラーの詳細については、“29.9.13 データベースを使用したときの異常”、また出力された例外情報については“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

RDB2_TCPのポート番号の設定

以下のファイルに、RDB2_TCP用のポート番号を設定してください。

Windows32/64

Windows Server(R)の場合
Windowsインストールディレクトリ¥Windows¥system32¥drivers¥etc¥services

Solaris64 Linux32/64

/etc/services

例

ポート番号に2050を割り当てる場合
RDBII 2050/TCP

4.3.2 Symfowareを使用する場合の環境設定(Symfowareのコネクションプーリングを使用する)

使用するデータソースの種類がSymfowareのコネクションプーリングを使用する場合には、Interstageのコネクションプーリングを使用する場合に加えて、以下の作業を行ってください。

- JDBCネーミングサービスの起動
- JDBCネーミングサービスへのデータソースの登録

1.JDBCネーミングサービスの起動

この操作はデータソースの種類がSymfowareのコネクションプーリングを使用する場合のみ必要です。
JDBCを使用する場合は、JDBCデータソースへの登録/参照、およびJ2EEアプリケーションを実行するためにSymfowareが提供するネーミングサービスを起動する必要があります。
以下の手順でネーミングサービスを起動してください。

1. Java環境を確認する
   Java環境の設定が行われているかを確認してください。Java環境の設定については、“Javaの環境設定”を参照してください。
2. 環境変数を確認する
   “環境変数の設定”に記載されている環境変数を設定してください。
3. ネーミングサービスを起動する
   ネーミングサービスを起動してください。

   例

   以下は、Javaコマンドを使用してネーミングサービスを起動し、ポート番号を10327に変更する例です。指定できる各パラメタの詳細は、Symfowareのマニュアルを参照してください。

   java com.fujitsu.symfoware.jdbc2.naming.SYMNameService 10327

注意

名前変換を利用する場合、JDBCネーミングサービスが起動していないとエラーが発生し、以下のメッセージが出力されますので、JNDI名(%s)について確認してください。

javax.naming.InvalidNameException physical-name is invalid NAME = %s

詳細は、“メッセージ集”の“lookup処理で例外が発生した場合の対処”を参照してください。

2.JDBCネーミングサービスへのデータソースの登録

この操作はデータソースの種類がSymfowareのコネクションプーリングを使用する場合のみ必要です。
JDBCを使用する場合は、JDBCネーミングサービスにデータソースを登録してください。
以下の手順でJDBCデータソースを登録してください。

1. Java環境を確認します。
   Java環境の設定が行われているかを確認してください。Java環境の設定については、“Javaの環境設定”を参照してください。
2. 環境変数を確認します。
   “環境変数の設定”に記載されている環境変数を設定してください。
3. JDBCデータソース登録ツールを起動します。
   管理者権限をもつユーザIDで、Symfowareが提供するJDBCデータソース登録ツールを起動します。詳細はSymfowareのマニュアルを参照してください。

   例

   ネーミングサービスのポート番号10327を指定してJDBCデータソース登録ツールを起動する例です。指定できる各パラメタの詳細は、Symfowareのマニュアルを参照してください。

   ```
   java com.fujitsu.symfoware.jdbc2.tool.FJJdbcTool 10327
   ```

4. JDBCデータソースを登録します。
   データソース一覧の[追加]ボタンをクリックし、データソースの登録に必要な情報を設定して[OK]ボタンをクリックします。

   例

データソース名	DS1
プロトコル	ローカル
データ資源名	DB1
ユーザ名	j2ee
パスワード	j2ee

4.3.3 Symfowareを使用する場合の環境設定(Postgresを使用する)

Windows32/64 Solaris64 Linux32/64

Symfoware Server(Postgres)、またはEnterprise Postgresを使用する場合は、PostgreSQL用のデータソースを作成して使用します。PostgreSQLのデータソースの作成方法については、以下の点を読み替えて、“4.3.6 PostgreSQLを使用する場合の環境設定”に従ってください。

- Symfoware Server、またはEnterprise Postgresのクライアント機能が提供するJDBCドライバを使用してください。
- クラスパス環境変数に設定するパス名は、Symfoware Server、またはEnterprise Postgresの「アプリケーション開発ガイド」の説明に従ってください。
- PostgreSQL用データソースの登録時には、以下の点に注意してください。
  - "クライアントバージョン"は、"V5"(Interstage 管理コンソールの場合は、"PowerGres Plus V5")を指定してください。
  - PostgreSQL用データソースの"接続ポート番号"のデフォルト値は"5432"です。適宜変更してください。
- 「アプリケーションの接続先切り替え機能」を使用する場合は、以下の点に注意してください。
  - "接続ホスト名"はホスト名として使用できる文字列の範囲内で任意に指定してください(無視されます)。
  - "その他データソース・プロパティ情報"にプロパティを追加して、以下の設定を行ってください。
    ・データソース・プロパティ名:serverName (先頭のsは小文字です)

・データ型：java.lang.String
・データソース・プロパティ値: "host1[:port1],host2[:port2]"形式の値

― データベースを更新するアプリケーションからデータソースを使用する場合は、"その他データソース・プロパティ情報"にプロパティを追加して、以下の設定を行ってください。
・データソース・プロパティ名：targetServerType (先頭のtは小文字です)
・データ型：java.lang.String
・データソース・プロパティ値：master

― "接続ポート番号"は"その他データソース・プロパティ情報"の"データソース・プロパティ名：serverName "の"データソース・プロパティ値"でポート番号を省略した場合にのみ使用されます。PostgreSQL用データソースの"接続ポート番号"のデフォルト値は"5432"です。適宜変更してください。

4.3.4 Oracleを使用する場合の環境設定

Oracleを使用する場合は、以下の環境設定が必要です。

Oracle Real Application Clustersを使用する場合は、付録B Oracle Real Application Clustersとの連携を参照してください。

- 環境変数の設定
- Interstageへのデータソース登録

デフォルトではFile System Service Providerを使用しないデータソースが定義されます。File System Service Providerを使用する場合の設定については、“JDBC(データベース)を参照する場合の共通事項”の“File System Service Providerを使用する場合の環境設定”を参照してください。

1.環境変数の設定

設定する環境変数の設定項目と、動作環境ごとの設定方法は以下のとおりです。

環境変数の設定項目

Oracle JDBCドライバを動作させるために必要な、以下を設定してください。Oracleホーム・ディレクトリは、Oracle製品のソフトウェアをインストールするために選択したディレクトリです。使用するOracleがインストールされている、Oracleホーム・ディレクトリを指定してください。
詳細は、Oracleのマニュアルを参照してください。
また、各JDBCドライバがサポートするJavaバージョンについては各JDBCドライバのマニュアルを参照してください。

設定項目	Oracleのバージョン	設定値 (注1)
クラスパス	Oracle11g以降	<Oracleホーム・ディレクトリ>¥jdbc¥lib¥ojdbc<n>.jar(注2)
		<Oracleホーム・ディレクトリ>¥jlib¥orai18n.jar

注1) SolarisまたはLinuxの場合、“¥”を“/”に読み替えてください。
注2) <n>は6,7,8などの数値です。JDK8をサポートしているJDBCドライバを使用してください。

OCIドライバを使用する場合には、上記の設定に加えて以下の設定が必要です。

Windows32/64

設定項目	Oracleのバージョン	設定値
パス	共通	<Oracleホーム・ディレクトリ>¥bin

Solaris64 Linux32/64

設定項目	Oracleの バージョン	設定値
ライブラリパス	Oracle11gリリース1 (注1)(注2)	<Oracleのインストールディレクトリ>/lib32
	Oracle11g以降 (注1)	<Oracleホーム・ディレクトリ>/lib
ORACLE_HOME	共通	<Oracleホーム・ディレクトリ>

注1) ライブラリパスは上記表に記載している順序で設定してください。

注2) 64bitのOracle Databaseと同じサーバに32bit版の本製品をインストールした場合に設定してください。

IJServerを使用する場合の設定方法

“IJServerを使用する場合の環境変数の設定方法”を参照して設定してください。

クライアント環境の場合の設定方法

クライアント環境でJ2EEアプリケーションクライアントを動作させる場合は、環境変数に設定してください。以下に設定例を記載します。

Windows(R)上の環境設定でシステム環境変数に設定する場合の例 Windows32/64

[スタート]－[コントロールパネル]－[システムとセキュリティ]－[システム]－[システムの詳細設定]の環境変数ボタンをクリックしてください。

注意

Windows Server(R) 2012の場合の説明です。使用するOSにより操作方法は異なります。

コマンドでシステム環境変数に設定する場合の例

ORACLE_HOMEは、Oracleのインストールホーム・ディレクトリです。

Windows32/64

```
set CLASSPATH=%CLASSPATH%;%ORACLE_HOME%¥jdbc¥lib¥ojdbc7.jar
set CLASSPATH=%CLASSPATH%;%ORACLE_HOME%¥jdbc¥lib¥orai18n.jar
```

Solaris64 Linux32/64

```
[Cシェルの場合]
setenv CLASSPATH ${CLASSPATH}:${ORACLE_HOME}/jdbc/lib/ojdbc7.jar
setenv CLASSPATH ${CLASSPATH}:${ORACLE_HOME}/jdbc/lib/orai18n.jar
```

2.Interstageへのデータソース登録

Interstage管理コンソールでデータソースを定義します。詳細は、Interstage管理コンソールのヘルプを参照してください。また、isj2eeadminコマンドを使用して変更することもできます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

データソースを定義する際に、使用する機能によって以下からデータソースの種類を選択できます。

データソースの種類	用途
Interstageのコネクションプーリングを使用する	Interstageのコネクションプーリング機能を使用できるデータソースです。デフォルトで選択されます。

データソースの種類	用途
Oracleのコネクションプーリングを使用する(注)	Oracleの暗黙的接続キャッシュを使用するデータソースです。以下の場合に選択してください。 ・ Oracle Real Application Clusters(Oracle RAC)の高速接続フェイルオーバー機能を使用する場合 ・ Statementキャッシュ機能を使用する場合 ・ Oracleの接続キャッシュ・プロパティによるチューニングを行う場合
Windows32/64 Linux32/64 分散トランザクションを使用する	分散トランザクション(グローバルトランザクション)環境を使用する場合に選択してください。

Interstage Application Server 8.0以前において定義可能であった「グローバルトランザクションを使用する」のチェックボックスは、削除されました。グローバルトランザクションを使用する場合は、「データソースの種類」で「分散トランザクションを使用する」を選択してください。

注) Oracle JDBC Driver 12.1では、暗黙的接続キャッシュがサポートされなくなりました。" Oracleのコネクションプーリングを使用する"のデータソースを使用する場合は、Oracle JDBC Driver 11.1、または11.2を使用してください。

ポイント

Interstageのデータソースを使ってマルチテナント・コンテナ・データベース内のプラガブル・データベース(PDB)へ接続する場合は、以下のいずれかの方法で接続してください。

- ociドライバを使用する場合
  - ― ドライバタイプ/ネットワークプロトコル : oci/tcp
  - ― SID/ネットサービス名 : 接続先のPDBのサービス名

  を指定してください。
- thinドライバを使用する場合

  Oracle Net ServicesでSIDをサービス名として解釈する設定を行い、
  - ― ドライバタイプ/ネットワークプロトコル : thin/tcp
  - ― SID/ネットサービス名 : 接続先のPDBのサービス名

  を指定してください。

Oracle Net Servicesの設定方法の詳細については、データベースのマニュアルを参照してください。

4.3.5 SQL Serverを使用する場合の環境設定

本章ではMicrosoft(R) JDBCドライバを使用してSQL Serverに接続する場合の環境設定について説明しています。

SQL Serverを使用する場合は、以下の環境設定が必要です。

- Microsoft(R) JDBCドライバのダウンロードとインストール
- リモート接続の設定
- 環境変数の設定
- Interstageへのデータソース登録
- JDBCドライバロギング機能

デフォルトではFile System Service Providerを使用しないデータソースが定義されます。File System Service Providerを使用する場合の設定については、“JDBC(データベース)を参照する場合の共通事項”の“File System Service Providerを使用する場合の環境設定”を参照してください。

1.Microsoft(R) JDBCドライバのダウンロードとインストール

ダウンロード

Microsoft(R) SQL Server(TM)には同梱されていません。Microsoft Corporationのホームページより、ダウンロードしてください。JDBCドライバは使用するSQL Server(TM)のバージョンに合わせて適切なものを使用してください。

インストール

インストールについては、Microsoft Corporationのホームページに記載されているインストール方法を参照してください。

2.リモート接続の設定

無償版や開発・テストシステム向けの製品など、使用する製品によっては、デフォルトの設定ではリモート接続できない場合があります。
Microsoft(R) SQL Server(TM)のマニュアルを参照して、リモート接続が可能となるよう設定してください。

3.環境変数の設定

設定する環境変数の設定項目と、動作環境ごとの設定方法は以下のとおりです。

環境変数の設定項目

Microsoft(R) JDBCドライバを動作させるために必要な、以下を設定してください。

設定項目	パス名 (注1)
クラスパス	<JDBCドライバのインストールディレクトリ> ¥sqljdbc_<version>¥<location>¥<JDBCドライバのファイル名> (注2)

注1) SolarisまたはLinuxの場合、“¥”を“/”に読み替えてください。
注2) <version>: Microsoft(R) SQL Server JDBC Driver 6.2の場合、“6.2”
<location>: 日本語版の場合は“jpn”、英語版の場合は“enu”
<JDBCドライバのファイル名>:IJServerワークユニットが動作するJavaをサポートしているJDBCドライバのファイル名。

IJServerを使用する場合の設定方法

“IJServerを使用する場合の環境変数の設定方法”を参照して設定してください。

クライアント環境の場合の設定方法

クライアント環境でJ2EEアプリケーションクライアントを動作させる場合は、環境変数に設定してください。以下に、設定例を記載します。

Windows(R)上の環境設定でシステム環境変数に設定する場合の例 Windows32/64

[スタート]－[コントロールパネル]－[システムとセキュリティ]－[システム]－[システムの詳細設定]の環境変数ボタンをクリックしてください。

注意

Windows Server(R) 2012の場合の説明です。使用するOSにより操作方法は異なります。

コマンドでシステム環境変数に設定する場合の例

Windows32/64

set CLASSPATH=%CLASSPATH%;C:¥mssqlserver¥sqljdbc_6.0¥jpn¥sqljdbc42.jar

Solaris64 Linux32/64

[Cシェルの場合]
setenv CLASSPATH ${CLASSPATH}:/opt/mssqlserver/sqljdbc_6.0/jpn/sqljdbc42.jar

4.Interstageへのデータソース登録

Interstage管理コンソールでデータソースを定義します。詳細は、Interstage管理コンソールのヘルプを参照してください。また、isj2eeadminコマンドを使用して変更することもできます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

5.JDBCドライバロギング機能

Microsoft(R) JDBCドライバでは、JDKが提供するjava.util.loggingパッケージのロギング機能を利用してデバッグが可能です。JDBCドライバのログを出力したい場合は、JDBCドライバのドキュメントを参照してください。デフォルトの標準出力、標準エラー出力はコンテナログに出力されます。

注意

Interstageのユーザスナップ情報の出力でも、JDKが提供するjava.util.loggingパッケージのロギング機能を利用しています。そのため、ユーザスナップ情報を出力すると、デフォルトではJDBCドライバのログも出力されます。JDBCドライバのログを抑止したい場合は、以下の定義を行ってください。

定義ファイル格納ディレクトリ	Windows32/64 C:¥Interstage¥EJB¥etc Solaris64 Linux32/64 /opt/FJSVejb/etc
定義ファイル名	FJlogging.properties
追加定義	com.microsoft.sqlserver.jdbc.level = OFF

4.3.6 PostgreSQLを使用する場合の環境設定

PostgreSQLを使用する場合は、以下の環境設定が必要です。

- 環境変数の設定
- Interstageへのデータソース登録

デフォルトではFile System Service Providerを使用しないデータソースが定義されます。File System Service Providerを使用する場合の設定については、“JDBC(データベース)を参照する場合の共通事項”の“File System Service Providerを使用する場合の環境設定”を参照してください。

注意

PowerGres PlusのJDBCドライバは、Windows版またはLinux版の本製品でのみ、使用できます。

1.環境変数の設定

設定する環境変数の設定項目と、動作環境ごとの設定方法は以下のとおりです。

環境変数の設定項目

PostgreSQLを動作させるために必要な、以下を設定してください。

設定項目	パス名 (注1)
クラスパス	<JDBCドライバ格納ディレクトリ>¥<JDBCドライバのファイル名>

注1) Linuxの場合、“¥”を“/”に読み替えてください。

IJServerを使用する場合の設定方法

“IJServerを使用する場合の環境変数の設定方法”を参照して設定してください。

クライアント環境の場合の設定方法

クライアント環境でJ2EEアプリケーションクライアントを動作させる場合は、環境変数に設定してください。以下に、設定例を記載します。

Windows(R)上の環境設定でシステム環境変数に設定する場合の例 Windows32/64

[スタート]－[コントロールパネル]－[システムとセキュリティ]－[システム]－[システムの詳細設定]の環境変数ボタンをクリックしてください。

注意

Windows Server(R) 2012の場合の説明です。使用するOSにより操作方法は異なります。

コマンドでシステム環境変数に設定する場合の例

クライアント機能を使用している場合でPowerGres Plus V9を利用する場合の設定例です。

Windows32/64

```
set CLASSPATH=C:¥PowerGresPlus¥share¥java¥postgresql-9.4.1212.jar;%CLASSPATH%
```

Solaris64 Linux32/64

```
[Cシェルの場合]
setenv CLASSPATH /opt/powergres91/share/java/postgresql-9.4.1212.jar:${CLASSPATH}
```

2.Interstageへのデータソース登録

Interstage管理コンソールでデータソースを定義します。詳細は、Interstage管理コンソールのヘルプを参照してください。また、isj2eeadminコマンドを使用して変更することもできます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

4.3.7 汎用定義のデータソースを使用する場合の環境設定

汎用定義について

JDBCデータソースの作成時に任意のデータベースやJDBCドライバのデータソースを作成できます。これをデータソースの汎用定義と呼びます。開発中にアプリケーションの動作確認をする場合に、一時的にサポートしないデータベースと接続確認を行う場合などに使用してください。実運用時にはサポートするデータベース／JDBCドライバを使用することを推奨します。(未サポートのデータベースやJDBCドライバを使用した場合、動作保証されません。)汎用定義で定義したデータソースを使用する場合、以下に注意してください。

コネクションプーリング機能

汎用定義で定義したデータソースクラスがjava.sql.ConnectionPoolDataSourceインタフェースを実装している場合、そのデータソースはInterstage側でコネクションプーリングされます。ただし、自動再接続機能は使用できません。
java.sql.ConnectionPoolDataSourceインタフェースを実装しないデータソースが定義された場合、Interstage側でプーリングは行いません。コネクションをプーリングするかはJDBCドライバ側の実装に依存しますので、JDBCドライバ(もしくは使用しているデータベース)のマニュアルを参照してください。

EJBのCMP1.1 Entity Bean、CMP2.0 Entity Bean

EJBコンテナがデータベースアクセス処理を行うCMP1.1 Entity BeanもしくはCMP2.0 Entity Beanは、汎用定義で定義したデータソースは使用できません。CMPの定義で使用できないデータソースを指定した場合、IJServer起動時にエラーとなります。またJ2EEのHotDeploy機能を用いてCMPの定義で使用できないデータソースを設定したモジュールを配備した場合、活性化処理に失敗し非活性状態になります。
ただし、開発時に一時的に使用したい場合、オプションを指定することでCMP1.1 Entity BeanもしくはCMP2.0 Entity Beanでも利用できます。汎用定義で定義したデータソースをCMP1.1 Entity Bean／CMP2.0 Entity Beanで使用できません。一時的に使用する場合は以下のオプションを指定してください。

定義ファイル格納ディレクトリ	Windows32/64 C:¥Interstage¥EJB¥etc Solaris64 Linux32/64 /opt/FJSVejb/etc
定義ファイル名	FJEJBconfig.properties
指定するキー	“cmp_generic_datasource_use”
指定する値	“yes”を指定します。

本オプションを指定してコンテナで異常が発生した場合、すぐにオプションの指定を中止してください。サポートしていないデータベースを利用した場合には以下の制限があります。

- 制限事項
  サポートしないデータベースを利用した場合、CMP1.1 Entity Beanのcreateメソッドにおいてすでに存在するレコードデータを挿入しようとした場合、java.rmi.RemoteExceptionが返却されます。(サポートするデータベースの場合、javax.ejb.DuplicateKeyExceptionが返却されます。)

必須API

汎用定義で定義したデータソースをJ2EEのJNDI機能でlookupして利用する場合、利用するデータベースもしくはJDBCドライバが以下のAPIをサポートしている必要があります。

インタフェース名	メソッド名	備考
javax.sql.ConnectionPoolDataSource	getPooledConnection()	java.sql.ConnectionPoolDataSourceインタフェースを実装したデータソースを定義した場合のみ。
	getPooledConnection(String user,String password)	
javax.sql.DataSource	getConnection()	
	getConnection(String user, String password)	
javax.sql.PooledConnection	addConnectionEventListener(ConnectionEventListener listener)	java.sql.ConnectionPoolDataSourceインタフェースを実装したデータソースを定義した場合のみ。
	close()	
	getConnection()	

インタフェース名	メソッド名	備考
	removeConnectionEventListener(ConnectionEventListener listener)	
java.sql.Connection	close()	J2EEのトランザクション開始後コネクションを取得する時、以下の順番でAutoCommitなどの設定が行われます。 1. setAutoCommit(false) 2. setTransactionIsolation(設定値) 3. setReadOnly(false) 2. の設定はDBコネクション設定においてIsolation-Levelをdefault以外に設定した場合のみ実行されます。
	commit()	
	getAutoCommit()	
	getTransactionIsolation()	
	isClosed()	
	isReadOnly()	
	setAutoCommit(boolean autoCommit)	
	setReadOnly(boolean readOnly)	
	setTransactionIsolation(int level)	

環境設定

以下の環境設定が必要です。

- 環境変数の設定
- Interstageへのデータソース登録
- その他データソース・プロパティ情報の設定

1.環境変数の設定

必要な環境設定については、使用する各データベースもしくはJDBCドライバのマニュアルを参照してください。

IJServerを使用する場合の設定方法

“IJServerを使用する場合の環境変数の設定方法”を参照して設定してください。

クライアント環境の場合の設定方法

クライアント環境でJ2EEアプリケーションクライアントを動作させる場合は、環境変数に設定してください。

2.Interstageへのデータソース登録

Interstage管理コンソールで、データソースを定義します。詳細は、Interstage管理コンソールのヘルプを参照してください。また、isj2eeadminコマンドを使用して変更することもできます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

3.その他データソース・プロパティ情報の設定

その他データソース・プロパティ情報に設定された情報は、DB接続テスト実施およびデータソース使用時にデータソースに対して以下のように設定されます。

1. データソースに「"set" + データソース・プロパティ名」という名前のメソッドが存在するかを確認します。ただし、データソース・プロパティ名の先頭の文字がアルファベットの小文字の場合は、それを大文字に変換し"set"につなげます。
   例) データソース・プロパティ名がpropertyの場合、setPropertyというメソッドが実行されます。

2. データソースに1.のメソッド名が存在して引数が1つのメソッドであれば、データソース・プロパティ値をその引数の型データに変換してデータソースに対してメソッドを実行します。

以下の場合にはエラーとなります。DB接続テスト機能を使用してデータソース・プロパティが正しく設定できているか確認してください。

- プロパティ名が誤っている場合
- プロパティのデータ型が誤っている場合
- プロパティ値にプロパティのデータ型に変換できない値が指定された場合

4.3.8 JDBC(データベース)を参照する場合の共通事項

IJServerを使用する場合の環境変数の設定方法

IJServerでWebアプリケーション、またはEJBアプリケーションを運用する場合は、以下のように設定してください。
Interstage管理コンソールを使用する場合の詳細については、Interstage管理コンソールのヘルプを参照してください。また、isj2eeadminコマンドを使用する場合の詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

設定項目	設定方法
Windows32/64 パス	以下のいずれかにパスを設定してください。 ・ J2EEプロパティのパス ・ IJServer環境設定のパス 以下のいずれかを使用して設定してください。 ・ Interstage管理コンソール — [システム] > [環境設定] > [J2EEプロパティ] > [パス] — [システム] > [ワークユニット] > [IJServer名] > [環境設定] > [パス] ・ isj2eeadminコマンド — J2EEシステム定義ファイル — IJServer定義ファイル 設定方法の詳細はInterstage管理コンソールのヘルプまたは“リファレンスマニュアル(コマンド編)”を参照してください。
Solaris64 Linux32/64 ライブラリパス	以下にライブラリパスを設定してください。 ・ J2EEプロパティのライブラリパス ・ IJServer環境設定のライブラリパス 以下のいずれかを使用して設定してください。 ・ Interstage管理コンソール — [システム] > [環境設定] > [J2EEプロパティ] > [ライブラリパス] — [システム] > [ワークユニット] > [IJServer名] > [環境設定] > [ライブラリパス] ・ isj2eeadminコマンド — J2EEシステム定義ファイル — IJServer定義ファイル 設定方法の詳細はInterstage管理コンソールのヘルプまたは“リファレンスマニュアル(コマンド編)”を参照してください。
クラスパス	以下にクラスパスを設定してください。

設定項目	設定方法
	・ J2EEプロパティのクラスパス ・ IJServer環境設定のクラスパス 以下のいずれかを使用して設定してください。 ・ Interstage管理コンソール ― [システム] > [環境設定] > [J2EEプロパティ] > [クラスパス] ― [システム] > [ワークユニット] > [IJServer名] > [環境設定] > [クラスパス] ・ isj2eeadminコマンド ― J2EEシステム定義ファイル ― IJServer定義ファイル 設定方法の詳細はInterstage管理コンソールのヘルプまたは“リファレンスマニュアル(コマンド編)”を参照してください。 クラスローダの分離をしないIJServerの場合、システム環境変数に設定しても有効となります。IJServerで使用するクラスパスの設定方法の詳細は“2.3.4 IJServerで使用するクラスの設定について”を参照してください。 Webアプリケーションのディレクトリ構成である“WEB-INF/lib”に、クラスパスに設定するJDBCドライバを格納しないでください。格納した場合、以下の機能が使用できない場合があります。 ・ 事前コネクト ・ 異常時の再接続
Solaris64 Linux32/64 ORACLE_HOME	以下に、ORACLE_HOMEを設定してください。 ・ IJServer環境設定の環境変数 ORALE_HOME=/opt/oracle 以下のいずれかを使用して設定してください。 ・ Interstage管理コンソール ― [システム] > [ワークユニット] > [IJServer名] > [環境設定] > [環境変数] ・ isj2eeadminコマンド ― IJServer定義ファイル 設定方法の詳細はInterstage管理コンソールのヘルプまたは“リファレンスマニュアル(コマンド編)”を参照してください。

File System Service Providerを使用する場合の環境設定

File System Service Providerを使用する場合の設定方法を説明します。

環境変数の設定

上記のJDBCドライバの設定に追加して以下の設定が必要です。File System Service Providerは、Interstageから提供され、以下のクラスパスに設定済みですので、IJServerの場合本作業は不要です。

設定項目	設定値
クラスパス	Windows32/64 C:¥Interstage¥J2EE¥lib¥providerutil.jar C:¥Interstage¥J2EE¥lib¥fscontext.jar Solaris64 Linux32/64

設定項目	設定値
	/opt/FJSVj2ee/lib/providerutil.jar /opt/FJSVj2ee/lib/fscontext.jar

最新のjarファイルを使用したい場合は、オラクル社のホームページからFile System Service Providerをダウンロードして、環境変数クラスパスを設定してください。
また、クライアント環境の場合は、Interstageから提供されているjar(上記の表)、または、ダウンロードしたjarのクラスパスを設定してください。

File System Service Providerへのデータソース登録

File System Service Providerに、データソースを登録します。Interstage管理コンソールまたはisj2eeadminコマンドを使用して、Interstageにデータソースを登録する場合には、Interstageへのデータソース登録と同時に、File System Service Providerにデータソースを登録するように指定できます。
この場合、本作業は必要ありません。詳細は、Interstage管理コンソールのヘルプ、または“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

File System Service Providerにデータソースを登録する場合には、以下に格納されているサンプルソースを参考にしてデータソース登録アプリケーションを作成し、データソースを登録してください。
サンプルソースはOracle用になっているため、SQL Serverを使用する場合はMicrosoft(R) JDBCドライバ用に、PostgreSQLを使用する場合はPostgreSQL用に修正して使用してください。
データソース登録アプリケーションの詳細については、各データベースのマニュアルを参照してください。

Windows32/64

C:¥Interstage¥J2EE¥sample¥datasource¥FJDSJNDILOCAL.java

File System Service Providerにデータソースを登録した場合、PROVIDER_URLで指定した場所に“.bindings”というファイルが、作成されます。

4.4 JDBC(データベース)のコネクション

ここでは、JDBC(データベース)のコネクション時の機能である以下について、説明します。

- コネクションプーリング
- 自動再接続機能

コネクションプーリングと自動再接続機能について詳細は、“27.2 IJServerのチューニング”を参照してください。

4.4.1 コネクションプーリング

コネクションプーリングとは、データベースにコネクション(接続)する場合に、要求されるたびにコネクションを作成するのではなく、コネクションをプール(保持)して再利用するという機能です。
このため、コネクションプーリングを使用することにより、同一ユーザからのアプリケーションからデータベースへコネクションする回数が軽減し、データベースへのコネクションを確立するアプリケーションの負荷も軽減されます。

ここでは、コネクションプーリングの以下について説明します。

- Interstageで使用できるコネクションプーリングの方式
- コネクションプールの生成単位
- コネクションプールとデータソース

Interstageで使用できるコネクションプーリングの方式

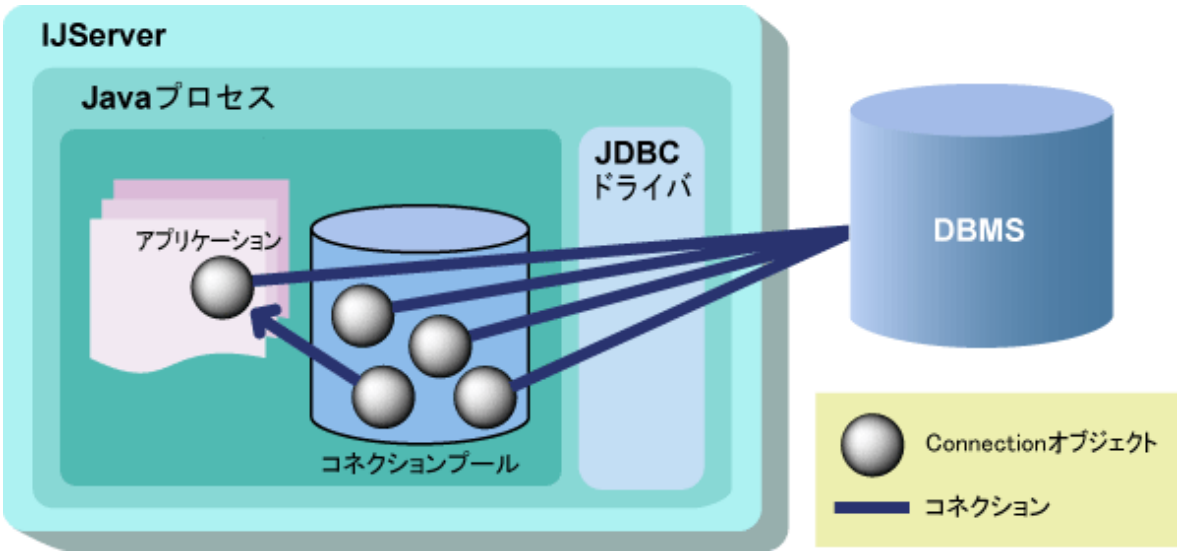
Interstage Application Serverでコネクションプーリングを使用する場合、以下の2種類の方式から選択できます。

- Interstageがコネクションをプールする
- JDBCがコネクションをプールする

Interstageがコネクションをプールする

コネクションプールはIJServer上の各Javaプロセス上に生成され、IJServerが管理します。

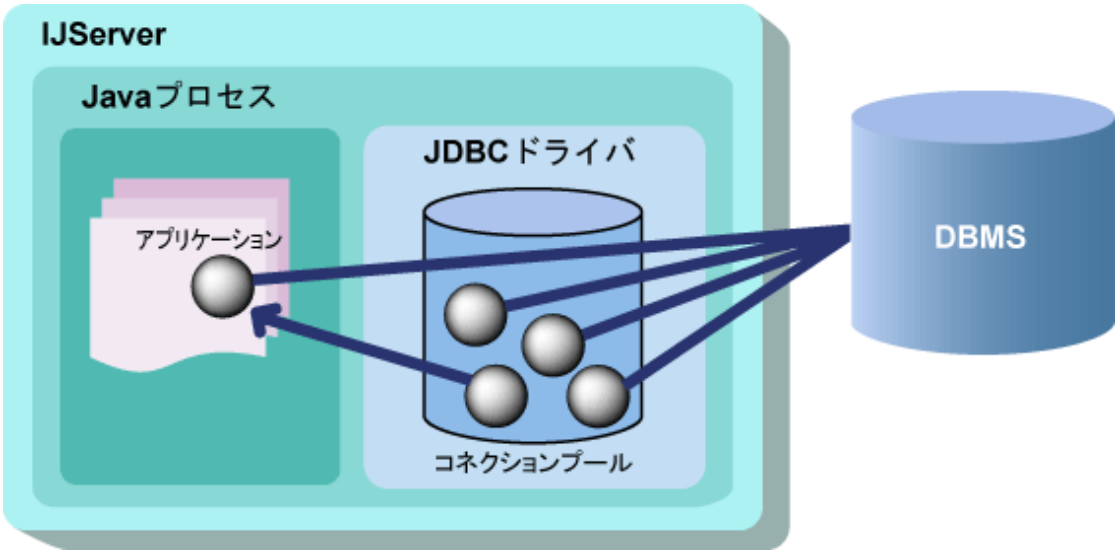
コネクションプールの管理はInterstageが行うため、コネクションプールのチューニング、および、JDBCコネクションの自動再接続機能などの機能が使用可能となります。また、コネクションの使用状況をモニタリングできます。



JDBCがコネクションをプールする

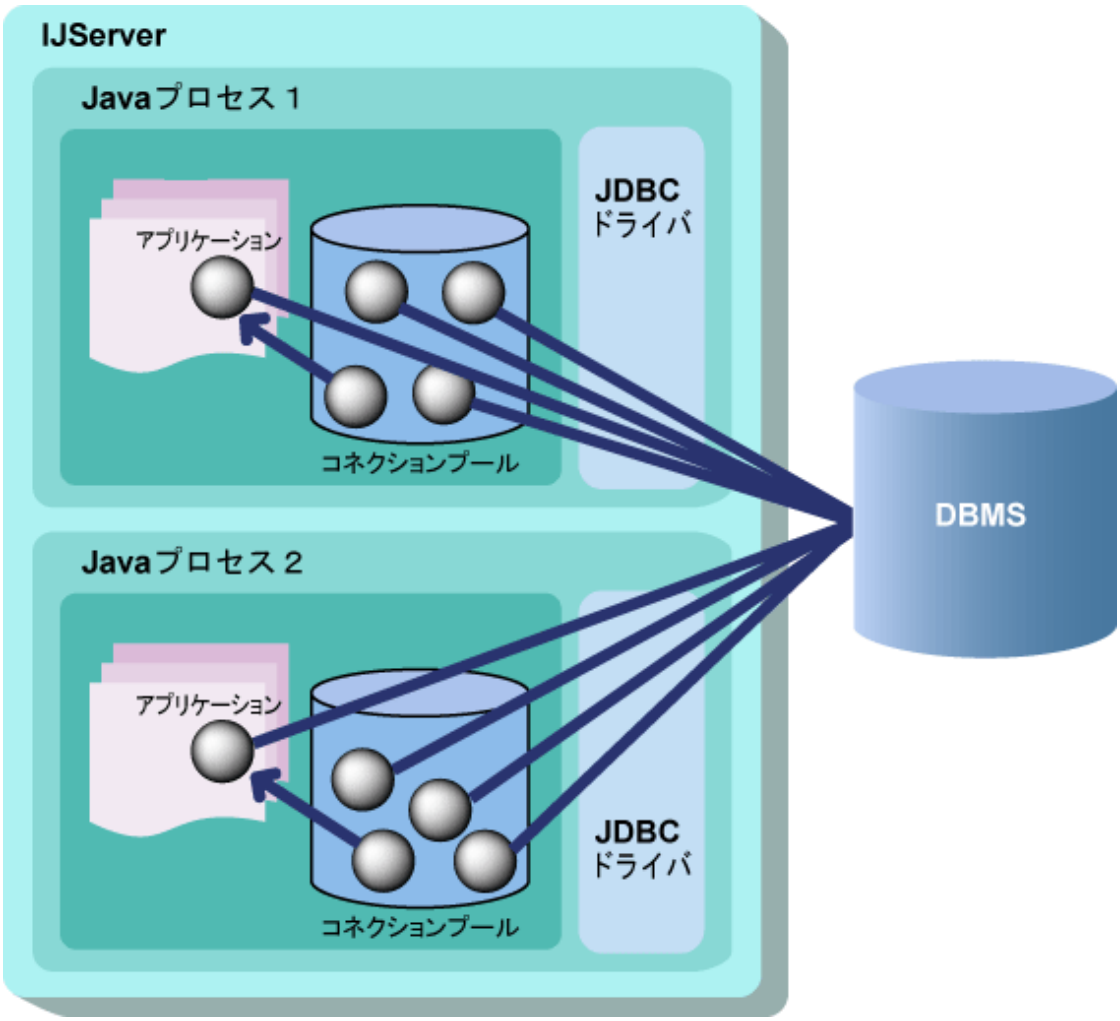
コネクションプールはJDBCドライバ上に生成され、JDBCドライバが管理します。

コネクションプールの管理はJDBCドライバが行うため、コネクションプールのチューニングはJDBCドライバの設定に依存します。このため、Interstageが提供しているJDBCコネクションの自動再接続機能、コネクションの使用状況のモニタリング機能は使用できません。



コネクションプールの生成単位

Interstageで生成されるコネクションプールは、各IJServerのプロセス多重度で、それぞれ生成されます。
また、Interstageでのコネクションプールの2種類の方式で違いはありません。IJServerのプロセス多重度ごとにそれぞれ生成されます。

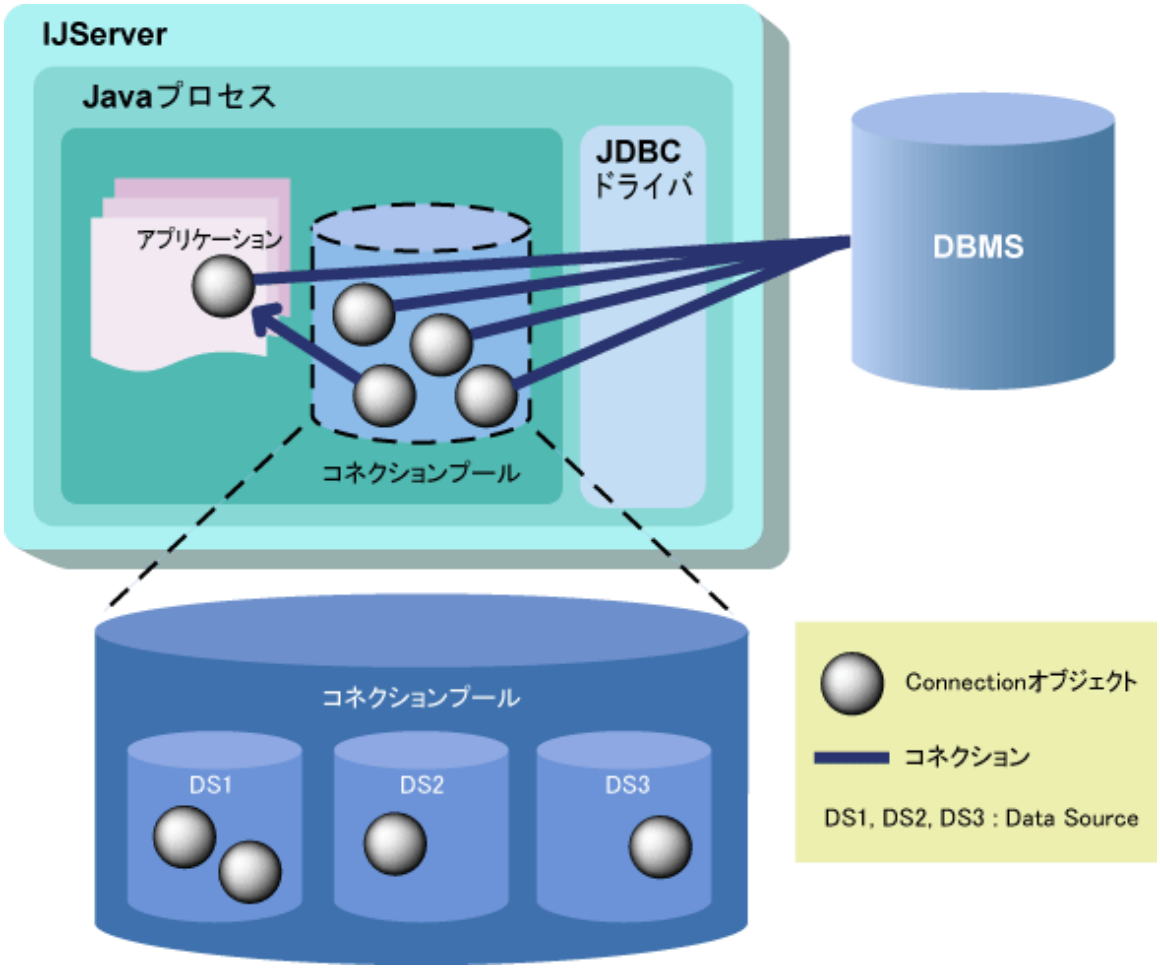


コネクションプールとデータソース

コネクションプールは各データソースで管理され、プーリングされています。
アプリケーションがConnectionオブジェクトを使用していない間、コネクションプールにプールされている状態になります。
プーリングされているコネクションは、以下のタイミングで破棄されます。

- アイドルタイムアウトが発生した時
- 自動再接続機能によりコネクションが無効であると判断した時
- IJServerを停止した時

- ijstuneコマンド実行時(実行中トランザクションが存在する場合、実行中トランザクション終了後)



また、同一のトランザクション内では、Connectionオブジェクトがキャッシュされて再利用されます。詳細は"トランザクションの制御方法"を参照してください。

データベースとコネクションプーリングの対応については、"27.2 IJServerのチューニング"を参照してください。

4.4.2 自動再接続機能

自動再接続機能とは、データベースサーバのダウンおよび通信回線の異常発生時に、無効となったプーリングされたコネクションを破棄したあとに、再度接続することで、データベースサーバ復旧後に、継続してデータベースアクセスを可能とする機能です。

自動再接続機能を使用することで、IJServerを再起動することなく、自動的にデータベースとの再接続を確立します。

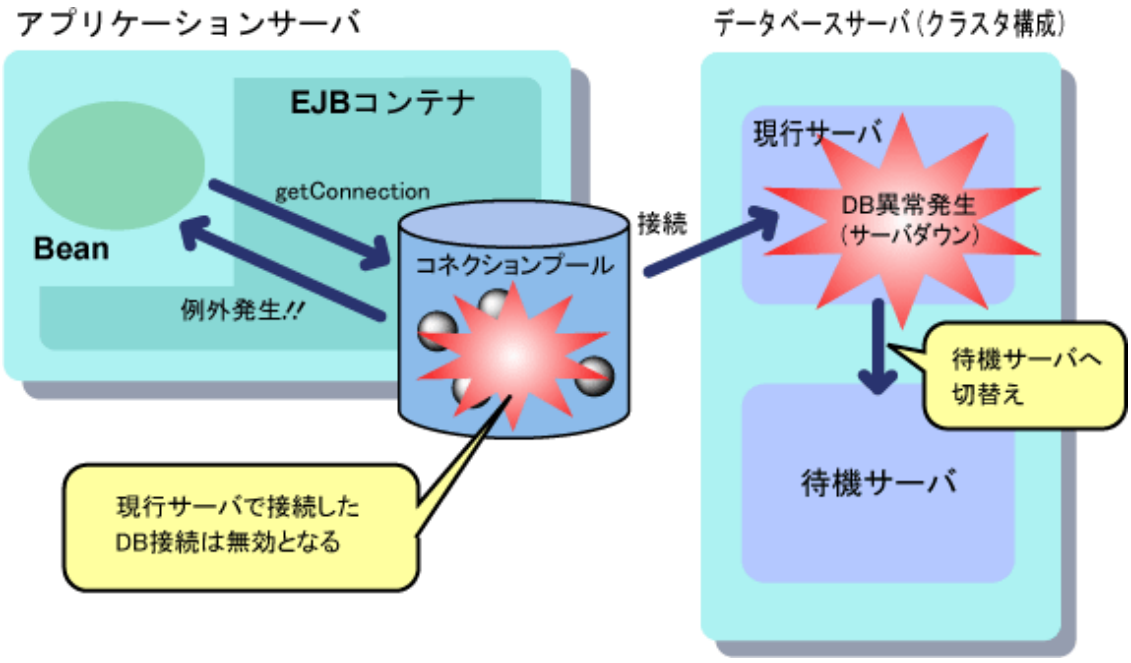
自動再接続機能を利用すると有効な例

以下のようなシステムで運用している場合を例にして、説明します。

- 使用しているデータベースは、Oracle Real Application Clusters(以降、Oracle RAC)である
- Oracle RACは、「現行サーバ」と「待機サーバ」に役割を分担させている
- 正常時は「現行サーバ」にのみ接続する運用方法である

アプリケーションサーバ(以降、IJServer)は、データベース接続したコネクションをプーリングし、以降のgetConnection呼び出し時にプーリングされたコネクションを返却しています。

正常時に接続する「現行サーバ」に異常が発生した場合、「待機サーバ」への切り替えが自動的に行われます。その際、IJServerでプーリングしていたコネクションは「現行サーバ」がダウンした時点で無効となるため、「待機サーバ」への切り替え後にアプリケーションからのデータベースアクセス時にエラーとなり、アプリケーションの運用ができなくなります。
そのため、IJServer側で自動的に無効となった、コネクションプール上のコネクションを開放するようなシステムに変更します。



自動再接続を使用する場合

IJServerは、以下の処理を行うことによりデータベースサーバがダウンした時のリカバリ(再接続)を行います。

SQL(注)文発行による接続確認

IJServerは、アプリケーションよりgetConnection呼び出しが来た場合、プールから取得したコネクションに対してSQL(注)文を発行し、接続確認を行います。
正常終了した場合は、取得したオブジェクトは有効なものであると判断して返却します。
例外が発生した場合は、プールから取得したコネクションを破棄してデータベースに直接getConnection呼び出しを行います。

データベース接続エラー時のリカバリ機構

SQL(注)文の発行で例外が発生した場合、またはアプリケーションからのgetConnection呼び出し時にプールがなく、直接データベースにgetConnection呼び出しをして例外が返却された場合、接続が正常に終了するまで指定された回数分、一定間隔でデータベース接続処理を繰り返します。
接続が正常に終了した時点で、アプリケーションへ接続したコネクションを返却します。
指定された回数実行しても接続に失敗する場合は、SQLExceptionを返却します。

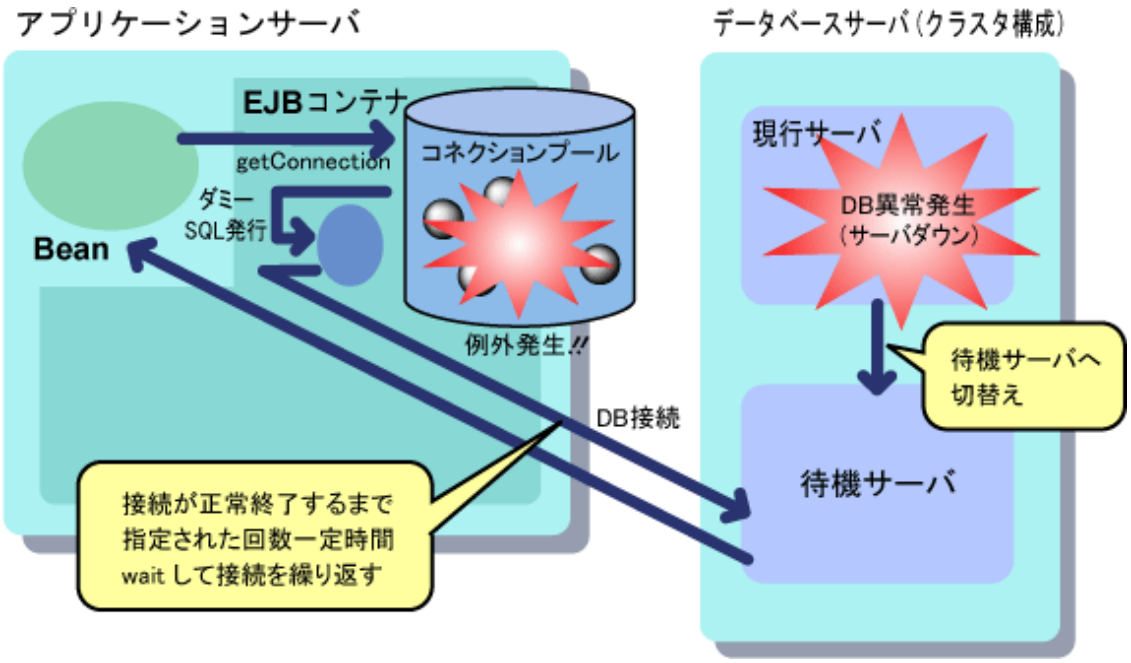
注)

SQL:Oracleが推奨している、以下のSQL文を発行します。

```
select sysdate from dual
```

本章の例では、データベースにOracleを使用しているため発行されるSQL文は“select sysdate from dual”になりますが、Symfoware、PostgreSQL、SQL Serverを使用する場合は、以下が発行されます。Symfowareの場合には、以下のSQL文は構文エラーとなりますが、エラー発生時のエラーコードでデータベースに正常に接続できたか確認しています。

```
select 1
```



再接続のタイミング

DBMS異常が発生した場合のDBMSへの再接続は、プールからコネクションを取得した時、またはプールにコネクションがない場合にDBMSから直接コネクションを取得した時に行います(DBMSへの接続を確認した時点で、トランザクションが開始し、SQL発行が可能な状態となります)。

トランザクション中、およびコネクションを取得した後にDBMSがダウンした場合、そのトランザクション内でDBアクセス、またはEntity Bean(CMP)にアクセス(SQL発行)を行うと、例外が発生します(この際、DBMSへの再接続は行われません)。この場合、JDBCコネクションをクローズしてトランザクションをロールバックしてから、再度処理を継続してください(トランザクション管理種別がContainerの場合は、コンテナがロールバックを行います)。

以下に、Entity Bean(CMP)で無効なコネクションを使用した場合に発生する例外を示します。

メソッド	返却される例外
create	java.rmi.RemoteException または javax.transaction.TransactionRolledbackException
findByPrimaryKey	
find<METHOD>	
remove	
find<METHOD>の返却値Enumerationに対するnextElementメソッド	java.lang.RuntimeException
find<METHOD>の返却値CollectionのIteratorに対するnextメソッド	

以下に、トランザクション完了時に発生する例外を示します。

メソッド	返却される例外
commit	javax.transaction.HeuristicMixedException または

メソッド	返却される例外
	javax.transaction.HeuristicRollbackException（トランザクションがロールバックにマークされている場合）
rollback	―

JDBCコネクションの自動再接続機能の設定について詳細は、“27.2 IJServerのチューニング”を参照してください。

Oracle RACと連携する場合は、付録B Oracle Real Application Clustersとの連携を参照してください。

4.4.3 サポートAPI

JDBCの標準APIのみサポートします。各JDBCドライバ固有のメソッド(OracleConnection固有のメソッドなど)は使用できないため、標準のAPIを使用してください。固有のメソッドを実行するためにオブジェクトをJDBCドライバ固有のクラスにキャストしようとした場合にはjava.lang.ClassCastExceptionが発生します。

4.5 JMSを参照する場合の環境設定

JMSを参照する場合、以下を行ってください。

- J2EEアプケーションの開発
- リソースアクセス定義

また、運用にあたっては、“3.9 JMSを利用する場合の手順”を参照してください。

J2EEアプリケーションの開発

利用目的に応じて以下のアプリケーションを開発します。

- J2EEアプリケーションクライアントの場合
  - ― Message Listenerアプリケーション
  - ― Durable Subscription機能を使用するアプリケーション
  - ― メッセージの不揮発化機能を使用するアプリケーション
  - ― ローカルトランザクション機能を使用するアプリケーション
  - ― グローバルトランザクション機能を使用するアプリケーション
- Webアプリケーションの場合
  - ― イベントチャネルの不揮発化機能を使用するアプリケーション
  - ― ローカルトランザクション機能を使用するアプリケーション
- EJBアプリケーションの場合

  メッセージを送信するといったEJBアプリケーションのみ作成することが可能ですが、利用目的に応じて次のようなEJBアプリケーションを開発することが可能です。
  - ― イベントチャネルの不揮発化機能を使用するアプリケーション
  - ― ローカルトランザクション機能を使用するアプリケーション
  - ― グローバルトランザクション機能を使用するアプリケーション

詳細は、“第24章 JMSアプリケーションの開発”を参照してください。

リソースアクセス定義

Interstage管理コンソール、またはisj2eeadminコマンドでリソースアクセス定義をします。詳細は、Interstage管理コンソールのヘルプを参照してください。isj2eeadminコマンドについて詳細は、“リファレンスマニュアル（コマンド編）”の“isj2eeadmin”を

参照してください。
Interstageのクライアント機能をインストールしている場合には、JMS運用コマンドを使用します。JMS運用コマンドについては、“リファレンスマニュアル(コマンド編)”の“JMS運用コマンド”を参照してください。

4.6 JavaMailを参照する場合の環境設定

JavaMailを参照する場合、Interstage管理コンソール、またはisj2eeadminコマンドでリソースアクセス定義をします。詳細は、Interstage管理コンソールのヘルプを参照してください。isj2eeadminコマンドについて詳細は、“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

4.7 URLを参照する場合の環境設定

URLを参照する場合、必ず名前変換機能を使用してください。
名前変換機能の詳細は、“4.11 名前変換機能”を参照してください。

例

記述例

以下に名前変換ファイルの記述例を示します。

```
<?xml version="1.0"?>
<!DOCTYPE fujitsu-ebe-definition SYSTEM 'fujitsu-ebe-definition.dtd'>
<fujitsu-ebe-definition>
  <ejb>
    <group-name>MyServer</group-name>
    <jndi-name>OperationBean</jndi-name>
    <res-entry>
      <res-ref-name>url/SVURL</res-ref-name>
      <datasource-name>
        http://192.0.2.150/cgi-bin/CAmngtop.cgi
      </datasource-name>
    </res-entry>
  </ejb>
</fujitsu-ebe-definition>
```

4.8 connectorを参照する場合の環境設定

connectorを参照する場合、resource adapterの配備後に環境設定を行ってください。
以下に、resource adapterの配備と環境設定について説明します。

resource adapterの配備

Interstage管理コンソール、またはisj2eeadminコマンドを使用して、resource adapterを運用環境に配備します。
配備時は、resource adapterファイル(.rarファイル)、またはresource adapterファイルを含むEARファイルを指定してください。
以下に配備時に設定するリソース定義情報の項目を示します。
config-property情報の内容については、resource adapterの仕様書を参照してください。

設定項目名	設定内容
resource adapterファイル名	指定されたresource adapterファイルが表示されます。この画面での変更はできません。
リソース名	リソースの名前を決めてください。リソース名がJNDIに登録する名前になります。
ユーザ名/パスワード	リソースに接続するために使用するユーザ名とパスワードを指定してください。省略できますが、パスワードだけの指定はできません。

設定項目名	設定内容
config-property情報	deployment descriptorに定義されたconfig-propertyの情報を変更したい場合に設定してください。プロパティ値だけ変更できます。

resource adapterファイルは以下のディレクトリ構成で展開されます。resource adapterのマニュアルを参照して、必要に応じてクラスパス、パス、ライブラリパスを設定してください。

IJServerに配備した場合

"2.2 J2EEアプリケーションが運用される環境(IJServer)"の"2.2.3 IJServerのファイル構成"を参照してください。
RAR内のjarについては、自動的にクラスパスに設定されます。

Interstage管理コンソールで[リソース] > [connector]に配備した場合

Windows32/64

[J2EE共通ディレクトリ]¥deployed¥jca¥ra¥[リソース名]
※J2EE共通ディレクトリのデフォルトは、C:¥Interstage¥J2EE¥var¥deployment です。

Solaris64 Linux32/64

[J2EE共通ディレクトリ]/deployed/jca/ra/[リソース名]
※J2EE共通ディレクトリのデフォルトは、/opt/FJSVj2ee/var/deployment です。

分散トランザクションを使用する場合 Windows32/64 Linux32/64

Interstage管理コンソールの[リソース] > [connector] > [配備] > [グローバルトランザクションの利用]を[する]に設定してください。
Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
また、isj2eeadminコマンドを使用して設定することもできます。詳細は、"リファレンスマニュアル(コマンド編)"の"isj2eeadmin"を参照してください。

注意

IJServerへ配備した場合、分散トランザクションは使用できません。Interstage管理コンソールの[リソース] > [connector]から配備してください。

リソース定義の参照・変更

配備終了後、Interstage管理コンソールを使用して、resource adapterの情報を参照できます。また、配備時に設定したユーザ名/パスワードとconfig-property情報のプロパティ値を変更できます。

環境設定

resource adapterを運用する場合は、配備実行後に以下の環境設定が必要です。
RARファイルは配備実行時に展開されるため、必要に応じて環境変数を設定してください。展開されるディレクトリの格納場所については、"resource adapterの配備"を参照してください。

以下の例は、リソース名が"RA01"で、RARファイル内にライブラリとRA01.jarが存在する場合のPATHとCLASSPATHの設定方法です。以下の例のように、Interstage管理コンソールを使用するか、isj2eeadminコマンドを使用して設定します。

- Interstage管理コンソールの[ワークユニット] > "ワークユニット名" > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [パス]に以下を設定してください。

Windows32/64

C:¥Interstage¥J2EE¥var¥deployment¥deployed¥jca¥ra¥RA01

Solaris64 Linux32/64

/opt/FJSVj2ee/var/deployment/deployed/jca/ra/RA01

- Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [クラスパス]に以下を設定してください。

  Windows32/64

  C:¥Interstage¥J2EE¥var¥deployment¥deployed¥jca¥ra¥RA01¥RA01.jar

  Solaris64 Linux32/64

  /opt/FJSVj2ee/var/deployment/deployed/jca/ra/RA01/RA01.jar

分散トランザクションを使用する場合 Windows32/64 Linux32/64

Windows32/64

PATHとCLASSPATHをシステム環境変数に設定してください。なお、設定後はOSの再起動が必要になります。

Linux32/64

PATHとCLASSPATHを環境変数に設定してください。なお、設定はInterstageを起動する前に行ってください。

参照

connector利用時にエラーが発生した場合は、“29.9 EJBサービス使用時の異常”を参照してください。

4.9 deployment descriptorファイルへの記述

deployment descriptorファイルに、参照するオブジェクトの情報を記述します。
ここでは、オブジェクトの参照に関するタグについて説明します。

参照

deployment descriptorファイルの詳細については、以下を参照してください。

- J2EEアプリケーションクライアントは、“4.13 J2EEアプリケーションクライアントのdeployment descriptorファイルの詳細設定”
- Webアプリケーションは、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”
- EJBアプリケーションは、Interstage StudioのXMLエディタを使用します。詳細は、“Interstage Studio ユーザーズガイド”を参照してください。

各オブジェクトの情報はdeployment descriptorの以下のタグに記述します。

deployment descriptorのタグ	指定値
ejb-ref	EJB Homeオブジェクト
ejb-local-ref	EJB Local Homeオブジェクト
service-ref	Webサービス
resource-ref	JDBCデータソース
	JMSコネクションファクトリ
	JavaMailメールセッション
	URL(Uniform Resource Locator)
	connectorコネクションファクトリ
resource-env-ref	JMS Destination
message-destination-ref	JMS Destination
env-entry	環境エントリ

deployment descriptorのタグ	指定値
指定不要	UserTransaction
	ORB

タグの説明

ejb-ref

タグ	説明
ejb-ref	EJBの参照に関する定義をします。複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
ejb-ref-name	Enterprise Beanの参照名を、以下のPrefixを付加した名前で指定します(xxxxxは任意の文字列)。 ・ ejb/xxxxx
ejb-ref-type	Enterprise Beanのアプリケーション種別を、次のいずれかで指定します。 ・ Entity ・ Session
home	homeインタフェース名を指定します。homeインタフェース名には、限定名(パッケージ付インタフェース名)を指定します。
remote	remoteインタフェース名を指定します。remoteインタフェース名には、限定名(パッケージ付インタフェース名)を指定します。
ejb-link	Enterprise Bean名を指定します。省略可能です。

ejb-local-ref

タグ	説明
ejb-local-ref	localインタフェースのEJBの参照に関する定義をします。複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
ejb-ref-name	Enterprise Beanの参照名を、以下のPrefixを付加した名前で指定します(xxxxxは任意の文字列)。 ・ ejb/xxxxx
ejb-ref-type	Enterprise Beanのアプリケーション種別を、次のいずれかで指定します。 ・ Entity ・ Session
local-home	local homeインタフェース名を指定します。local homeインタフェース名には、限定名(パッケージ付インタフェース名)を指定します。
local	localインタフェース名を指定します。localインタフェース名には、限定名(パッケージ付インタフェース名)を指定します。
ejb-link	Enterprise Bean名を指定します。省略可能です。

service-ref

タグ	説明
service-ref	Webサービス参照に関する定義をします。複数回指定可能です。 詳細は“18.6.4 service reference記述”を参照してください。

注) J2EE1.4以降のdeployment descriptorファイルで設定可能です。

resource-ref

タグ	説明
resource-ref	リソース参照に関する定義をします。複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
res-ref-name	参照名を、リソースごとのPrefixを付加した名前で指定します(xxxxxは任意の文字列)。 ・ JDBCの場合 :jdbc/xxxxx ・ JMSの場合 :jms/xxxxx ・ JavaMailの場合 :mail/xxxxx ・ URLの場合 :url/xxxxx ・ connectorの場合 :eis/xxxxx
res-type	lookupの際に受け取る型を限定名で指定します。限定名には、オブジェクトのクラス名またはインタフェース名を指定します。 ・ JDBCの場合 :javax.sql.DataSource ・ JMSの場合 :javax.jms.TopicConnectionFactoryまたはjavax.jms.QueueConnectionFactory ・ JavaMailの場合 :javax.mail.Session ・ URLの場合 :java.net.URL ・ connectorの場合 :javax.resource.cci.ConnectionFactory
res-auth	リソース接続者を、次のいずれかで指定します。 ・ Application :アプリケーションで設定された接続情報を使用する ・ Container :リソース定義で設定された接続情報を使用する

resource-env-ref

タグ	説明
resource-env-ref	リソース環境参照に関する定義をします。複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
resource-env-ref-name	参照名を、リソースごとのPrefixを付加した名前で指定します(xxxxは任意の文字列)。 ・ JMSの場合 :jms/xxxxx
resource-env-ref-type	lookupの際に受け取る型を限定名で指定します。限定名には、オブジェクトのクラス名またはインタフェース名を指定します。 ・ JMSの場合 :javax.jms.Topicまたはjavax.jms.Queue

message-destination-ref

タグ	説明
message-destination-ref	Destination参照に関する定義をします。複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
message-destination-ref-name	Destination参照名を、以下のPrefixを付加した名前で指定します(xxxxxは任意の文字列)。

タグ	説明
	・ jms/xxxxx
message-destination-type	lookupの際に受け取る型を限定名で指定します。限定名には、オブジェクトのクラス名またはインタフェース名を指定します。 ・ javax.jms.Topic ・ javax.jms.Queue
message-destination-usage	利用者にアプリケーション内でのリソースの使用方法を以下のいずれかで指定します。 ・ メッセージを受信する場合:Consumes ・ メッセージを送信する場合:Produces ・ 送信も受信もする場合:ConsumesProduces
message-destination-link	message-destination のmessage-destination-name に指定したJMS Destination名を指定します。省略可能です。

注) J2EE1.4以降のdeployment descriptorファイルで設定可能です。J2EE1.3以前の場合、JMS Destinationの設定はresource-env-refに指定してください。

message-destination

タグ	説明
message-destination	Destination参照の対応関係を定義します。 message-destination-linkを指定する場合、本定義は必須です。 複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
message-destination-name	JMS Destination名を指定します。

注) J2EE1.4以降のdeployment descriptorファイルで設定可能です。J2EE1.3以前の場合、JMS Destinationの設定はresource-env-refに指定してください。

env-entry

タグ	説明
env-entry	環境エントリの参照に関する定義をします。複数回指定可能です。
description	利用者に伝えたい任意の情報を指定します。省略可能です。
env-entry-name	環境エントリの参照名を指定します。
env-entry-type	環境エントリ値の型を、次のいずれかで指定します。 ・ java.lang.Boolean ・ java.lang.Byte ・ java.lang.String ・ java.lang.Short ・ java.lang.Integer ・ java.lang.Long ・ java.lang.Float ・ java.lang.Double ・ java.lang.Character

タグ		説明
	env-entry-value	lookupで取得したい環境エントリ値を指定します。省略可能です。Webアプリケーションでenv-entry-valueタグを省略した場合は下記となります。 ・ env-entry-type 値 が java.lang.Boolean の 場合:Boolean.FALSE ・ env-entry-type値が以下の場合:値0の各オブジェクト — java.lang.Byte — java.lang.Character — java.lang.Short — java.lang.Integer — java.lang.Long — java.lang.Float — java.lang.Double ・ env-entry-type値がjava.lang.Stringの場合:lookup時にjavax.naming.NamingExceptionまたはそのサブクラスが投げられます。 Webアプリケーション以外の場合は、lookup時にjavax.naming.NamingExceptionまたはそのサブクラスが投げられます。

注意

名前空間プレフィックスつきのタグは指定しないでください。
指定した場合、配備に失敗したり、名前変換機能が使用できなくなることがあります。

例:<pfx:ejb-ref>

message-destination-refとmessage-destinationについて

- message-destination-ref-nameタグには、アプリケーションでJNDIのlookupメソッドで引数に指定する名前を指定してください。例えばアプリケーションで以下のようにlookupする場合には、message-destination-ref-nameタグには“jms/MyTopic”を指定してください。
  ctx.lookup("java:comp/env/jms/MyTopic")
- message-destination-linkタグにはmessage-destinationタグのmessage-destination-nameタグに指定した値を指定してください。本指定は必須ではありません。
- message-destinationのmessage-destination-nameタグにはJMS Destination名を指定できます。例えば対象となるDestinationが同一のアプリケーションが1つのejb-jarに含まれていた場合、lookupする名前が異なっている場合にはmessage-destinationを定義してmessage-destination-refからリンクすることで、lookupする名前が変更になったり、逆にDestination名が変更になっても変更箇所を局所化することができます。

記述例

EJB Homeオブジュクト“ejb/EJB1”の設定例

```
 ・・・
<ejb-ref>
  <description>EJB Information</description>
  <ejb-ref-name>ejb/EJB1</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
```

```
  <home>sample.ejbHome</home>
  <remote>sample.ejbRemote</remote>
  <ejb-link>SessionBean</ejb-link>
</ejb-ref>
・・・
```

EJB Local Homeオブジュクト“ejb/SampleBMP”の設定例

```
・・・
<ejb-local-ref>
    <ejb-ref-name>ejb/SampleBMP</ejb-ref-name>
    <ejb-ref-type>Entity</ejb-ref-type>
    <local-home>SampleBMPHome</local-home>
    <local>SampleBMPLocal</local>
    <ejb-link>SampleBMP</ejb-link>
</ejb-local-ref>
・・・
```

JDBCデータソース“jdbc/DB1”の設定例

```
・・・
<resource-ref>
  <description>JDBC Information</description>
  <res-ref-name>jdbc/DB1</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
・・・
```

JMSコネクションファクトリ“jms/JMS1”およびJMS Destination“jms/JMS2”の設定例

J2EE1.4以降の場合

```
・・・
<resource-ref>
  <description>JMS Information</description>
  <res-ref-name>jms/JMS1</res-ref-name>
  <res-type>javax.jms.TopicConnectionFactory</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
<message-destination-ref>
  <description>JMS Information2</description>
  <message-destination-ref-name>jms/JMS2</message-destination-ref-name>
  <message-destination-type>javax.jms.Topic</message-destination-type>
  <message-destination-usage>Consumes</message-destination-usage>
  <message-destination-link>Topic001</message-destination-link>
</message-destination-ref>
・・・
<message-destination>
  <description>JMS Destination</description>
  <message-destination-name>Topic001</message-destination-name>
</message-destination>
・・・
```

J2EE1.3以前の場合

```
・・・
<resource-ref>
  <description>JMS Information</description>
  <res-ref-name>jms/JMS1</res-ref-name>
  <res-type>javax.jms.TopicConnectionFactory</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
<resource-env-ref>
  <description>JMS Information2</description>
```

```
  <resource-env-ref-name>jms/JMS2</resource-env-ref-name>
  <resource-env-ref-type>javax.jms.Topic</resource-env-ref-type>
</resource-env-ref>
・・・
```

ポイント

この設定例はJ2EEアプリケーションクライアントのdeployment descriptorです。
Webアプリケーション、およびEJBアプリケーションの場合は<resource-ref>タグと<resource-env-ref>タグの定義順を逆にしてください。

JavaMailメールセション“mail/Mail”の設定例

```
・・・
<resource-ref>
  <res-ref-name>mail/Mail</res-ref-name>
  <res-type>javax.mail.Session</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
・・・
```

URL“url/SVURL”の設定例

```
・・・
<resource-ref>
  <res-ref-name>url/SVURL</res-ref-name>
  <res-type>java.net.URL</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
・・・
```

connectorコネクションファクトリ“eis/RA01”の設定例

```
・・・
<resource-ref>
  <res-ref-name>eis/RA01</res-ref-name>
  <res-type>javax.resource.cci.ConnectionFactory</res-type>
  <res-auth>Application</res-auth>
</resource-ref>
・・・
```

環境エントリSValueの設定例

```
・・・
<env-entry>
  <description>EnvProp</description>
  <env-entry-name>SValue</env-entry-name>
  <env-entry-type>java.lang.Short</env-entry-type>
  <env-entry-value>1024</env-entry-value>
</env-entry>
・・・
```

注意

deployment descriptorファイルに参照するオブジェクトの情報を記述しない場合、以下の注意事項があります。

- コンテナが同じ名前のオブジェクトを自動的に検索します。このとき異なるリソースに同名のオブジェクトが存在すると誤動作する場合があるため注意してください。
- 名前変換機能は使用できません。
- サブコンテキストの取得を行うことができません。

・ 以下の条件を満たす場合は、Local Homeインタフェースが返却されます。

－ EJBアプリケーションを参照している

－ 参照するEJBアプリケーションがHomeインタフェース/Local Homeインタフェースを両方実装している

Local Homeインタフェース/Homeインタフェースの両方を経由したlookup処理を行う場合は、EJBアプリケーション開発時、および運用時に以下の作業を行って下さい。

EJBアプリケーション開発時

1. deployment descriptorの編集
   [参照EJBタグ]または[参照LocalEJBタグ]の“ejb-ref-name”に定義された“EnterpriseBeanの参照名”を重複しないよう編集して下さい。
2. EJBアプリケーションの開発
   lookup時の引数に指定するEJBアプリケーション名がLocal Homeインタフェースを経由する場合と、Homeインタフェースを経由する場合とで重複しないよう記述して下さい。1で設定した参照EJBまたは参照LocalEJBに定義されたEnterpriseBeanの参照名に対応するEJBアプリケーション名を記述して下さい。

EJBアプリケーション運用時

－ 名前変換ファイルの編集
  名前変換ファイルを使用し、1で変更したEnterpriseBeanの参照名と、EJBアプリケーション内にてlookup時の引数で指定したEJBアプリケーション名との対応付けを行って下さい。

4.10 オブジェクトの参照方法

オブジェクトの参照は、以下の手順で行います。

1. javax.naming.Contextクラスオブジェクトを作成します。
2. lookup()メソッドを使用して、参照するオブジェクトに合わせたクラスオブジェクトを獲得します。
   lookup()メソッドの引数には、以下を指定します。
   － EJBの場合、“java:comp/env/ejb/EJBアプリケーション名”
   － JDBCの場合、“java:comp/env/jdbc/JDBCリソースアクセス定義名”
   － JMSの場合、“java:comp/env/jms/JMSリソースアクセス定義名”
   － JavaMailの場合、“java:comp/env/mail/JavaMailリソースアクセス定義名”
   － connectorの場合、“java:comp/env/eis/connectorリソースアクセス定義名”
   － 環境エントリの場合、“java:comp/env/環境エントリ名”
   － 名前変換を利用してアクセスする場合、“java:comp/env/deployment descriptorの参照名”
3. EJB Homeオブジェクトの参照の場合、narrow処理を行います。

記述例を以下に示します。
以下は、例外javax.naming.NamingExceptionの対処を省略してあります。NamingExceptionの対処方法については“lookupで例外が発生した場合”を参照してください。

EJBアプリケーション名が“EJB214ETY”、Homeクラスが“EJB214ETYHome”のEJB Homeオブジェクトの参照例

```
// EJB Homeオブジェクトのlookup処理
java.lang.Object ejbobj = null;
EJB214ETYHome home = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  ejbobj = (java.lang.Object)nctx.lookup("java:comp/env/ejb/EJB214ETY");
  home = (EJB214ETYHome)javax.rmi.PortableRemoteObject.narrow(ejbobj, EJB214ETYHome.class);
} catch(javax.naming.NamingException ex) { }
```

注) Enterprise Beanの参照名を定義するときは、“ejb/Bean名”という形式で定義することを推奨します。

EJBアプリケーション名が“EJB214EmpCBM”、LocalHomeクラスが“EJB214EmpCBMLocalHome”のEJB Local Homeオブジェクトの参照例

```
// EJB Local Homeオブジェクトのlookup処理
EJB214EmpCBMLocalHome home = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  home = (EJB214EmpCBMLocalHome)nctx.lookup("java:comp/env/ejb/EJB214EmpCBM");
} catch(javax.naming.NamingException ex) { }
```

注) Enterprise Beanの参照名を定義するときは、“ejb/Bean名”という形式で定義することを推奨します。

JDBCリソースアクセス定義名が“DB1”のJDBCデータソースの参照例

```
// JDBCデータソースのlookup処理
javax.sql.DataSource dataSource = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  dataSource = (javax.sql.DataSource)nctx.lookup("java:comp/env/jdbc/DB1");
} catch(javax.naming.NamingException ex) { }
```

JMSリソースアクセス定義名が“Topic”および“Queue”のJMSコネクションファクトリの参照例

JMSコネクションファクトリがjavax.jms.TopicConnectionFactoryの場合

```
// JMSコネクションファクトリのlookup処理
javax.jms.TopicConnectionFactory topic = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  topic = (javax.jms.TopicConnectionFactory)nctx.lookup("java:comp/env/jms/Topic");
} catch(javax.naming.NamingException ex) { }
```

JMSコネクションファクトリがjavax.jms.QueueConnectionFactoryの場合

```
// JMSコネクションファクトリのlookup処理
javax.jms.QueueConnectionFactory queue = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  queue = (javax.jms.QueueConnectionFactory)nctx.lookup("java:comp/env/jms/Queue");
} catch(javax.naming.NamingException ex) { }
```

JavaMailリソースアクセス定義名が“MailSession”のJavaMailメールセションの参照例

```
//JavaMailメールセションのlookup処理
javax.mail.Session session = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  session = (javax.mail.Session)nctx.lookup("java:comp/env/mail/MailSession");
} catch(javax.naming.NamingException ex) { }
```

deployment descriptorの参照名が“url/SVURL”のURLの参照例

```
//URLのlookup処理
java.net.URL url = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  url = (java.net.URL)nctx.lookup("java:comp/env/url/SVURL");
} catch(javax.naming.NamingException ex) { }
```

connectorリソースアクセス定義名が“RA01”のconnectorコネクションファクトリの参照例

```
//connectorコネクションファクトリのlookup処理
javax.resource.cci.ConnectionFactory cf = null;
try {
```

```
    javax.naming.Context nctx = new javax.naming.InitialContext();
    cf =(javax.resource.cci.ConnectionFactory)nctx.lookup("java:comp/env/eis/RA01");
  } catch(javax.naming.NamingException ex) { }
```

JMSリソースアクセス定義名が“Topic”および“Queue”のJMS Destinationの参照例

JMSのDestinationがTopicの場合

```
// JMS Destination(javax.jms.Topic)のlookup処理
javax.jms.Topic topic = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  topic = (javax.jms.Topic)nctx.lookup("java:comp/env/jms/Topic");
} catch(javax.naming.NamingException ex) { }
```

JMSのDestinationがQueueの場合

```
// JMS Destination(javax.jms.Queue)のlookup処理
javax.jms.Queue queue = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  queue = (javax.jms.Queue)nctx.lookup("java:comp/env/jms/Queue");
} catch(javax.naming.NamingException ex) { }
```

環境エントリ名が“SValue”の環境エントリの参照例

```
//環境エントリのlookup処理
java.lang.Short val = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  val = (java.lang.Short)nctx.lookup("java:comp/env/SValue");
} catch(javax.naming.NamingException ex) { }
```

注) 引数に“java:comp/env/”を指定しないでlookup()メソッドを実行することも可能ですが、アプリケーションの移行性を重視する場合には推奨しません。

ORBオブジェクトの参照例

また、RMI over IIOP通信でコンテナが使用するORBオブジェクトも以下のように参照できます。

```
org.omg.CORBA.ORB orb = null;
try {
  javax.naming.Context nctx = new javax.naming.InitialContext();
  orb = (org.omg.CORBA.ORB)nctx.lookup("java:comp/ORB");
} catch(javax.naming.NamingException ex) { }
```

lookupで例外が発生した場合

lookup()メソッドを発行した結果、例外javax.naming.NamingExceptionが返却される場合があります。
NamingExceptionの原因となる例外情報を取得する場合、NamingExceptionに対してgetMessage()メソッドを発行してください。getCause()メソッドでは取得できません。

```
try {
  ・・・
  nctx.lookup(<参照名>);
  ・・・
} catch(javax.naming.NamingException ex) {
  ex.printStackTrace();
  String msg = ex.getMessage();
  throw new MyException(msg);
}
```

EJBアプリケーションを参照する場合、CORBAサービスから例外が返却される可能性があります。CORBAサービスの例外は、getMessage()でマニュアル“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”の“CORBA

サービスのマイナーコード”に記載されているような文字列が取得できます。
マイナーコードを解析する場合は、“minor code: ”の後の数値を利用してください。

注) 上記は、lookupの引数(<参照名>)に“java:comp/env/ejb/”で始まる文字列を指定した場合の解析方法です。

4.11 名前変換機能

アプリケーションで指定するJNDIの名前と、運用環境の実名をマッピングする機能です。JNDIの名前と、運用環境の実名が異なる場合でも、名前変換機能を使用することでアプリケーションソースのJNDIの名前を変更することなく対応することができます。

Webアプリケーション、EJBアプリケーションの場合

名前変換は、モジュールごとに、Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > “モジュール名” > [名前変換]タブで設定します。詳細は、Interstage管理コンソールのヘルプを参照してください。
なお、interstage.xmlファイルを直接編集することもできます。interstage.xmlファイルは、以下に格納されます。interstage.xmlファイルの詳細は、“4.11.2 interstage.xmlファイル”を参照してください。ただし、サーバ上の任意の位置で実行するWebアプリケーションの場合は、interstage.xmlファイルを直接編集できませんので、配備後にInterstage管理コンソールを使用して設定してください。
なお、ワークユニットが起動している状態で定義を変更した場合は、下記の操作により定義内容が有効になります。

- ワークユニットを再起動する
- Interstage管理コンソールの[ワークユニット] > “ワークユニット名” > [アプリケーション状態/配備解除]タブで配備モジュールを選択して再活性ボタンを押下する

Windows32/64

J2EE共通ディレクトリ¥ijserver¥[IJServer名]¥apps¥[モジュール名]¥META-INF¥interstage.xml
(J2EE共通ディレクトリのデフォルトは、C:¥Interstage¥J2EE¥var¥deployment です。)

Solaris64 Linux32/64

J2EE共通ディレクトリ/ijserver/[IJServer名]/apps/[モジュール名]/META-INF/interstage.xml
(J2EE共通ディレクトリのデフォルトは、/opt/FJSVj2ee/var/deployment です。)

J2EEアプリケーションクライアントの場合

名前変換ファイルを以下のディレクトリに配置し、環境プロパティでファイル名を指定します。名前変換ファイルの詳細は、“4.11.1 名前変換ファイル”を参照してください。

Windows32/64

C:¥Interstage¥J2EE¥etc

Solaris64

/etc/opt/FJSVj2ee/etc

Linux32/64

/etc/opt/FJSVj2ee/etc

注意

deployment descriptorファイルに参照するオブジェクトの情報を記述しない場合、名前変換機能は使用できません。

4.11.1 名前変換ファイル

記述形式

名前変換ファイルの記述形式はXML形式です。名前変換ファイルの記述形式を以下に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE fujitsu-ebe-definition SYSTEM 'fujitsu-ebe-definition.dtd'>
<fujitsu-ebe-definition>
```

```
  <client>
    <app-name>app-name</app-name>
    <ejb-ref-entry>
      <ejb-ref-name>ejb-ref-name</ejb-ref-name>
      <jndi-name>jndi-name</jndi-name>
    </ejb-ref-entry>
    <res-entry>
      <res-ref-name>res-ref-name</res-ref-name>
      <datasource-name>datasource-name</datasource-name>
    </res-entry>
    <res-env-entry>
      <res-env-ref-name>res-env-ref-name</res-env-ref-name>
      <environment-name>environment-name</environment-name>
    </res-env-entry>
    <message-destination-entry>
      <message-destination-name>message-destination-name</message-destination-name>
      <jndi-name>jndi-name</jndi-name>
    </message-destination-entry>
  </client>
</fujitsu-ebe-definition>
```

- 1行目、2行目は、XML宣言、およびDTD(文書型定義)を記述しているため、名前変換ファイルの先頭で必ず記述してください。
  J2EEアプリケーションクライアント用の定義で、値の文字列に日本語を使用する場合は、エンコード形式(「encoding=」部分)に、“Shift_JIS”等の適切な値を指定してください。
- 3行目、最終行の<fujitsu-ebe-definition>、</fujitsu-ebe-definition>は、XMLファイルの開始と終了を示すルートタグです。必ず指定してください。
- 各タグの記載順序は、上記の記載順序に従ってください。
- 太字部分は、必ず指定する必要があります。**<app-name>**は必須です。
- 斜体文字の部分は任意の文字列を指定します。空白、タブ、改行などの制御文字は使用できません。なお、斜体文字部分は、大文字小文字が区別されます。
- XMLファイルで、特別な意味をもつ文字(<、>、&)を使用する場合は、変換定義に従って、以下のように記述してください。

使用したい文字	名前変換ファイルでの記述
<	<
>	>
&	&

- 値の文字列に"'"、および"""を記述した場合は、それぞれシングルクォーテーション(')、ダブルクォーテーション(")と解釈されます。

タグの説明

タグ	説明
<client>	J2EEアプリケーションクライアントの場合に指定します。複数回指定可能です。
<app-name>	名前変換を行うアプリケーション名を指定します。
<ejb-ref-entry>	EJBオブジェクトの名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<ejb-ref-name>	deployment descriptorの参照名を指定します。

タグ	説明
<jndi-name>	<ejb-ref-name>に対応するEJBアプリケーション名(運用環境の実名)を指定します。
<res-entry>	JDBCデータソース、JMS(QueueConnectionFactory,TopicConnectionFactory)、JavaMail、connector、URLの名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<res-ref-name>	deployment descriptorの参照名を指定します。
<datasource-name>	<res-ref-name>に対応するリソースアクセス定義名(運用環境の実名)を指定します。
<res-env-entry>	JMS Destination(Queue,Topic)の名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<res-env-ref-name>	deployment descriptorの参照名を指定します。
<environment-name>	<res-env-ref-name>に対応するリソースアクセス定義名(運用環境の実名)を指定します。
<message-destination-entry>	JMS Destinationの名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<message-destination-name>	deployment descriptorの参照名を指定します。
<jndi-name>	<message-destination-name>に対応するJMS Destination定義名(運用環境の実名)を指定します。

注) 参照名を重複して定義した場合は、最後の定義が有効になります。

記述例(J2EEアプリケーションクライアント)

deployment descriptorの参照名と運用環境の実名が以下の場合について、名前変換ファイルの記述例を示します。

	deployment descriptorの参照名	運用環境の実名
EJB	ejb/EntBean	EB1
リソース参照(JDBC)	jdbc/DataSource	DS1
リソース参照(JMS)	jms/TopicCF	CF1
リソース環境参照	jms/Topic	DN1

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE fujitsu-ebe-definition SYSTEM 'fujitsu-ebe-definition.dtd'>
<fujitsu-ebe-definition>
  <client>
    <app-name>GetBeans</app-name>
    <ejb-ref-entry>
      <ejb-ref-name>ejb/EntBean</ejb-ref-name>
      <jndi-name>EB1</jndi-name>
    </ejb-ref-entry>
    <res-entry>
      <res-ref-name>jdbc/DataSource</res-ref-name>
      <datasource-name>DS1</datasource-name>
    </res-entry>
```

```
      <res-entry>
        <res-ref-name>jms/TopicCF</res-ref-name>
        <datasource-name>CF1</datasource-name>
      </res-entry>
      <res-env-entry>
        <res-env-ref-name>jms/Topic</res-env-ref-name>
        <environment-name>DN1</environment-name>
      </res-env-entry>
    </client>
  </fujitsu-ebe-definition>
```

4.11.2 interstage.xmlファイル

記述形式

interstage.xmlファイルの記述形式はXML形式です。interstage.xmlファイルの記述形式を以下に示します。

```
<?xml version="1.0"?>
<!DOCTYPE fujitsu-ebe-definition SYSTEM 'fujitsu-ebe-definition.dtd'>
<fujitsu-ebe-definition>
  <web> または <ejb>
    <app-name>app-name</app-name> または <jndi-name>jndi-name</jndi-name> または<module-name>module-name</
module-name>
    <ejb-ref-entry>
      <ejb-ref-name>ejb-ref-name</ejb-ref-name>
      <jndi-name>jndi-name</jndi-name>
    </ejb-ref-entry>
    <res-entry>
      <res-ref-name>res-ref-name</res-ref-name>
      <datasource-name>datasource-name</datasource-name>
    </res-entry>
    <res-env-entry>
      <res-env-ref-name>res-env-ref-name</res-env-ref-name>
      <environment-name>environment-name</environment-name>
    </res-env-entry>
    <message-destination-entry>
      <message-destination-name>message-destination-name</message-destination-name>
      <jndi-name>jndi-name</jndi-name>
    </message-destination-entry>
  </web> または </ejb>
</fujitsu-ebe-definition>
```

- 1行目、2行目は、XML宣言、およびDTD(文書型定義)を記述しているため、名前変換ファイルの先頭で必ず記述してください。
- 3行目、最終行の<fujitsu-ebe-definition>、</fujitsu-ebe-definition>は、XMLファイルの開始と終了を示すルートタグです。必ず指定してください。
- 各タグの記載順序は、上記の記載順序に従ってください。
- 太字部分は、必ず指定する必要があります。**<jndi-name>**または**<module-name>**タグのどちらかがEJBアプリケーションの場合必須です。**<app-name>**タグはWEBアプリケーションの場合必須です。
- 斜体文字の部分は任意の文字列を指定します。空白、タブ、改行などの制御文字は使用できません。なお、斜体文字部分は、大文字小文字が区別されます。
- XMLファイルで、特別な意味をもつ文字(<、>、&)を使用する場合は、変換定義に従って、以下のように記述してください。

使用したい文字	名前変換ファイルでの記述
<	<
>	>
&	&

- 値の文字列に"'"、および"""を記述した場合は、それぞれシングルクォーテーション(')、ダブルクォーテーション(")と解釈されます。
- <web>、および<ejb>タグ以外は編集しないでください。

タグの説明

タグ	説明
<web>	Webアプリケーションの場合に指定します。複数回指定可能です。
<ejb>	EJBアプリケーションの場合に指定します。複数回指定可能です。
<app-name>	Webアプリケーションの場合、名前変換を行うアプリケーション名を指定します。
<jndi-name>	EJBアプリケーションで<ejb-ref-entry>、<res-entry>、<res-env-entry>を指定する場合、名前変換を行うEJBアプリケーション名を指定します。
<module-name>	EJBアプリケーションで<message-destination-entry>を指定する場合、名前変換を行うEJBモジュール名を指定します。
<ejb-ref-entry>	EJBオブジェクトの名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<ejb-ref-name>	deployment descriptorの参照名を指定します。
<jndi-name>	<ejb-ref-name>に対応するEJBアプリケーション名(運用環境の実名)を指定します。
<res-entry>	JDBCデータソース、 JMS(QueueConnectionFactory,TopicConnectionFactory)、 JavaMail、connector、URLの名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<res-ref-name>	deployment descriptorの参照名を指定します。
<datasource-name>	<res-ref-name>に対応するリソースアクセス定義名(運用環境の実名)を指定します。
<res-env-entry>	JMS Destination(Queue,Topic)の名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<res-env-ref-name>	deployment descriptorの参照名を指定します。
<environment-name>	<res-env-ref-name>に対応するリソースアクセス定義名(運用環境の実名)を指定します。
<message-destination-entry>	JMS Destinationの名前変換を定義します。複数回指定可能です。 このタグの1つの定義に対し、以下の2つのタグを1つずつ定義してください。
<message-destination-name>	deployment descriptorの参照名を指定します。
<jndi-name>	<message-destination-name>に対応するJMS Destination定義名(運用環境の実名)を指定します。

注) 参照名を重複して定義した場合は、最後の定義が有効になります。

記述例(Webアプリケーションの場合)

deployment descriptorの参照名と運用環境の実名が以下の場合について、interstage.xmlファイルの記述例を示します。

	deployment descriptorの参照名	運用環境の実名
EJB	ejb/EntBean	EB1
リソース参照(JDBC)	jdbc/DataSource	DS1
リソース参照(JMS)	jms/TopicCF	CF1
リソース環境参照	jms/Topic	DN1

```
<?xml version="1.0"?>
<!DOCTYPE fujitsu-ebe-definition SYSTEM 'fujitsu-ebe-definition.dtd'>
<fujitsu-ebe-definition>
  <web>
    <app-name>GetBeans</app-name>
    <ejb-ref-entry>
      <ejb-ref-name>ejb/EntBean</ejb-ref-name>
      <jndi-name>EB1</jndi-name>
    </ejb-ref-entry>
    <res-entry>
      <res-ref-name>jdbc/DataSource</res-ref-name>
      <datasource-name>DS1</datasource-name>
    </res-entry>
    <res-entry>
      <res-ref-name>jms/TopicCF</res-ref-name>
      <datasource-name>CF1</datasource-name>
    </res-entry>
    <res-env-entry>
      <res-env-ref-name>jms/Topic</res-env-ref-name>
      <environment-name>DN1</environment-name>
    </res-env-entry>
  </web>
</fujitsu-ebe-definition>
```

記述例(EJBアプリケーションの場合)

deployment descriptorの参照名と運用環境の実名が以下の場合について、interstage.xmlファイルの記述例を示します。

	deployment descriptorの参照名	運用環境の実名
EJB	ejb/CallBean	AccountBean
リソース参照(JMS)	jms/TopicCF	CatalogCF

EJBアプリケーションの記述例

```
...
javax.naming.Context ic = new javax.naming.InitialContext();

Object obj = (Object)ic.lookup("java:comp/env/ejb/CallBean");
CallBeanHome beanHome =
    (CallBeanHome)javax.rmi.PortableRemoteObject.narrow(obj, CallBeanHome.class);
...
javax.jms.TopicConnectionFactory cf =
    (javax.jms.TopicConnectionFactory)ic.lookup("java:comp/env/jms/TopicCF");
...
```

interstage.xmlファイルの記述例

EJBアプリケーションで名前変換を行う場合のinterstage.xmlファイルの記述例を以下に示します。配備モジュール名は“CatalogEJB.jar”、EJBアプリケーション名は“OperationBean”、“EmployeeBean”です。

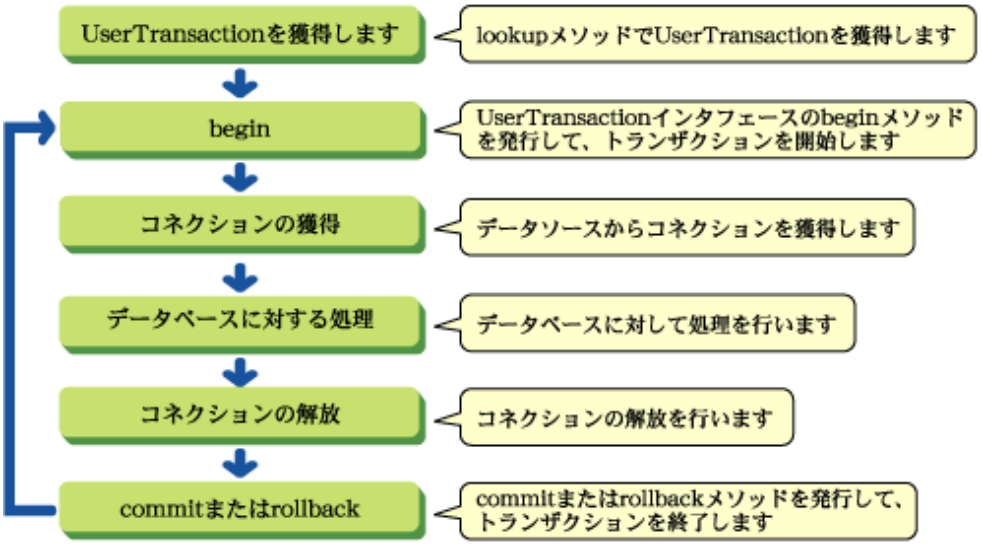
```
<?xml version="1.0"?>
<!DOCTYPE fujitsu-ebe-definition SYSTEM 'fujitsu-ebe-definition.dtd'>
<fujitsu-ebe-definition >
  <ejb>
    <jndi-name>OperationBean</jndi-name>
    <ejb-ref-entry>
      <ejb-ref-name>ejb/CallBean</ejb-ref-name>
      <jndi-name>AccountBean</jndi-name>
    </ejb-ref-entry>
    <res-entry>
      <res-ref-name>jms/TopicCF</res-ref-name>
      <datasource-name>CatalogCF</datasource-name>
    </res-entry>
  </ejb>
  <ejb>
    <jndi-name>EmployeeBean</jndi-name>
    <ejb-ref-entry>
      <ejb-ref-name>ejb/CallBean</ejb-ref-name>
      <jndi-name>AccountBean</jndi-name>
    </ejb-ref-entry>
    <res-entry>
      <res-ref-name>jms/TopicCF</res-ref-name>
      <datasource-name>CatalogCF</datasource-name>
    </res-entry>
  </ejb>
  <ejb>
    <module-name>CatalogEJB.jar</module-name>
    <message-destination-entry>
      <message-destination-name>Topic001</message-destination-name>
      <jndi-name>CatalogTopic</jndi-name>
    </message-destination-entry>
  </ejb>
</fujitsu-ebe-definition>
```

4.12 UserTransactionインタフェースを使用したトランザクション制御

アプリケーションが、トランザクションを制御する場合、UserTransactionインタフェースを使用して制御します。以下に、UserTransactionインタフェースを使用したトランザクション機能の処理の流れなど、アプリケーションの開発について説明します。

UserTransactionインタフェースを使用したトランザクション機能の流れ

以下に、UserTransactionインタフェースを使用したトランザクション機能を使用する場合の処理の流れを示します。



UserTransactionインタフェースのメソッド

UserTransactionインタフェースには以下のメソッドがあります。
使用できるメソッドの一覧を示します。

メソッド名	内容
begin	新しいトランザクションを生成し、現在のスレッドに関連付けます。
commit	現在のスレッドと関連付けられたトランザクションを完了させます。
getStatus	現在のスレッドと関連付けられたトランザクションの状態を獲得します。
rollback	現在のスレッドと関連付けられたトランザクションをロールバックします。
setRollbackOnly	現在のスレッドと関連付けられたトランザクションを、トランザクションの結果がロールバックだけになるように変更します。
Windows32/64 Linux32/64 setTransactionTimeout	現在のスレッドでbeginメソッドにより開始されたトランザクションと関連するタイムアウト値を変更します。

注意

UserTransactionインタフェースのsetTransactionTimeoutメソッドは分散トランザクションを使用する場合のみ有効になります。
ローカルトランザクションで本メソッドを使用する場合、設定された値は無効になります。

トランザクション制御範囲

デフォルトのトランザクションを使用する場合、J2EEアプリケーションでトランザクションを開始し、同一Java VM上の他のJ2EEアプリケーションにアクセスする場合、同一のトランザクションで動作させることができます。
また、WebアプリケーションとEJBアプリケーションを同一JavaVMで運用する場合、WebアプリケーションがUserTrasanctionを使用して、Webアプリケーションからアクセスした EJBアプリケーション処理をトランザクション連携させることも可能です。

コネクションの獲得、解放

コネクションの獲得は、データソースに対してgetConnection()メソッドを発行することによって行います。これ以外の方法で獲得されたコネクションは、トランザクション配下のコネクションとして扱われません。

コネクションの解放は、獲得したコネクションに対してclose()メソッドを発行することによって行います。
データソースを使用する場合は、最初にデータソースのlookupを行います。

例

記述例

```
...
javax.transaction.UserTransaction userTransaction = null ;
  /* JNDIのlookupメソッドを使用して、UserTransactionを獲得します */
  try {
      javax.naming.Context initialContext = new
      javax.naming.InitialContext();
      userTransaction =
      (UserTransaction)initialContext.lookup("java:comp/UserTransaction");
  } catch(NamingException ex) {
      /* 例外処理 */
      ...
  }

  /* トランザクション処理を開始します */
  try {
      userTransaction.begin();
  } catch(javax.transaction.NotSupportedException e) {
      /* 例外処理 */
      ...
  } catch(javax.transaction.SystemException e) {
      /* 例外処理 */
      ...
  }

  try {
      /* EJBアプリケーションの呼出し、もしくは、JDBCデータソースへのアクセス */
      ...
  } catch( Throwable e ) {
      /* 例外が発生した場合などはトランザクションをロールバック*/
      userTransaction.rollback();
      throw e;
  }

  try {
      /* 処理を完結したい場合などにはトランザクションをコミット*/
      userTransaction.commit();
  } catch(javax.transaction.HeuristicMixedException e ) {
      /* HeuristicMixedExceptionが発生した場合はコミットもしくはロールバックを実行してください*/
      userTransaction.rollback();
      throw e;
  } catch(java.lang.IllegalStateException e ) {
      userTransaction.rollback();
      throw e;
  } catch(java.lang.SecurityException e ) {
      userTransaction.rollback();
      throw e;
  } catch(javax.transaction.SystemException e ) {
      userTransaction.rollback();
      throw e;
  } catch( Throwable e ) {
      throw e;
  }
...
```

注意

UserTransactionインタフェースのcommit処理でjavax.transaction.HeuristicMixedExceptionが発生した場合、commitもしくはrollback処理を実行してください。commitもしくはrollback処理を実行しない場合、トランザクションが開始中の状態のまま扱われます。
また、UserTransactionインタフェースのcommit処理で以下の例外が発生した場合、rollback処理を実行してください。rollback処理を実行しない場合、トランザクションが開始中の状態のまま扱われます。
UserTransactionオブジェクトを再利用した場合に、トランザクションが開始中の状態のままbegin()メソッドを発行すると、javax.transaction.NotSupportedExceptionが発生する可能性があります。

- java.lang.IllegalStateException
- java.lang.SecurityException
- javax.transaction.SystemException

4.13 J2EEアプリケーションクライアントのdeployment descriptorファイルの詳細設定

以下に、J2EEアプリケーションクライアントのdeployment descriptorファイルの記述形式を説明します。
deployment descriptorのファイル名は任意で拡張子を.xmlとします。
deployment descriptorファイルは、任意のディレクトリに配置し、環境プロパティでファイル名をフルパスで指定します。

記述形式

deployment descriptorの記述形式はXML形式です。deployment descriptorの記述例を以下に示します。

J2EE1.4のdeployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<application-client version="1.4"
xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/application-client_1_4.xsd">
  <icon>
    <small-icon>small_icon</small-icon>
    <large-icon>large_icon</large-icon>
  </icon>
  <display-name>display_name</display-name>
  <description>description</description>
  <env-entry>
    <description>description</description>
    <env-entry-name>name</env-entry-name>
    <env-entry-type>type</env-entry-type>
    <env-entry-value>value</env-entry-value>
  </env-entry>
  <ejb-ref>
    <description>description</description>
    <ejb-ref-name>name</ejb-ref-name>
    <ejb-ref-type>type</ejb-ref-type>
    <home>home</home>
    <remote>remote</remote>
    <ejb-link>link</ejb-link>
  </ejb-ref>
  <resource-ref>
    <description>description</description>
    <res-ref-name>name</res-ref-name>
    <res-type>type</res-type>
    <res-auth>auth</res-auth>
```

```
  </resource-ref>
  <resource-env-ref>
    <description>description</description>
    <resource-env-ref-name>name</resource-env-ref-name>
    <resource-env-ref-type>type</resource-env-ref-type>
  </resource-env-ref>
</application-client>
```

J2EE1.3のdeployment descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application-client PUBLIC
"-//Sun Microsystems, Inc.//DTD J2EE Application Client 1.3//EN"
"http://java.sun.com/dtd/application-client_1_3.dtd">
<application-client>
  <icon>
    <small-icon>small_icon</small-icon>
    <large-icon>large_icon</large-icon>
  </icon>
  <display-name>display_name</display-name>
  <description>description</description>
  <env-entry>
    <description>description</description>
    <env-entry-name>name</env-entry-name>
    <env-entry-type>type</env-entry-type>
    <env-entry-value>value</env-entry-value>
  </env-entry>
  <ejb-ref>
    <description>description</description>
    <ejb-ref-name>name</ejb-ref-name>
    <ejb-ref-type>type</ejb-ref-type>
    <home>home</home>
    <remote>remote</remote>
    <ejb-link>link</ejb-link>
  </ejb-ref>
  <resource-ref>
    <description>description</description>
    <res-ref-name>name<res-ref-name>
    <res-type>type</res-type>
    <res-auth>auth</res-auth>
  </resource-ref>
  <resource-env-ref>
    <description>description</description>
    <resource-env-ref-name>name</resource-env-ref-name>
    <resource-env-ref-type>type</resource-env-ref-type>
  </resource-env-ref>
</application-client>
```

- J2EE1.4のdeployment descriptorでは、先頭の<?xml. . . >は、XML宣言を記述しているため、deployment descriptorファイルの先頭で必ず記述してください。
  J2EE1.3のdeployment descriptorでは、先頭の<?xml. . . >と<!DOCTYPE application-client...>は、XML宣言、およびDTD(文書型定義)を記述しているため、deployment descriptorファイルの先頭で必ず記述してください。
  また、値の文字列に日本語を使用する場合は、エンコード形式(「encoding=」部分)に、“Shift_JIS”等の適切な値を指定してください。
- <application-client>、</application-client>は、XMLファイルの開始と終了を示すルートタグです。必ず指定してください。
- 各タグの記載順序は、上記の記載順序に従ってください。
- タグの文字列は、大文字小文字が区別されます。
- 斜体文字の部分は任意の文字列を指定します。空白、タブ、改行などの制御文字は使用できません。なお、斜体文字部分は、大文字小文字が区別されます。

・ 名前空間プレフィックスつきのタグは指定しないでください。
指定した場合、名前変換機能が使用できなくなることがあります。
例：<pfx:ejb-ref>

タグの説明

タグ	説明
icon	
small-icon	GUI上でJ2EEアプリケーションクライアントを表現する、小さい(16×16)アイコン(GIF/JPEG形式)へのURIを指定します。URIは、パッケージのルートからの相対で設定します。省略可能です。
large-icon	GUI上でJ2EEアプリケーションクライアントを表現する、大きい(32×32)アイコン(GIF/JPEG形式)へのURIを指定します。URIは、パッケージのルートからの相対で設定します。省略可能です。
display-name	J2EEアプリケーションクライアント表示名を指定します。J2EEアプリケーションクライアント表示名は、GUIなどで表示されます。省略不可です。
description	J2EEアプリケーションクライアントの詳細情報を指定します。詳細情報には、利用者に伝えたい任意の情報を指定します。省略可能です。

オブジェクトの参照に関する以下のタグについては、“4.9 deployment descriptorファイルへの記述”を参照してください。

・ env-entry
・ ejb-ref
・ resource-ref
・ resource-env-ref
・ service-ref
・ message-destination-ref
・ message-destination

第5章 J2EEアプリケーションのセキュリティ

本章では、J2EEアプリケーションのセキュリティについて、以下を説明します。

- セキュリティ機能
  セキュリティ機能の種類と内容について説明します。
- セキュリティ機能の組み込み方法
  セキュリティ機能を組み込む方法について説明します。
- セキュリティ機能の認証のログ採取
  セキュリティ機能の認証のログについて、採取するための設定方法と、メッセージの書式について説明します。
- セキュリティ機能の異常時の対処
  セキュリティ機能で異常が発生した場合の対処方法について説明します。

5.1 セキュリティ機能

セキュリティ機能は、J2EEアプリケーションの資源に対する不当なアクセスを防止するための機能です。

運用形態

J2EEアプリケーションのセキュリティ機能は、次の運用形態を想定しています。

- J2EEアプリケーションクライアント－EJBアプリケーション
- J2EEアプリケーションクライアント－EJBアプリケーション－EJBアプリケーション
- Webアプリケーション－EJBアプリケーション
- Webアプリケーション－EJBアプリケーション－EJBアプリケーション

セキュリティの種類

J2EEアプリケーションのセキュリティ機能には、以下があります。

- ユーザ認証
- アクセス制限
- メソッドパーミッション
- セキュリティ関連のメソッド
- リソース接続者管理機能
- run-as security機能

5.1.1 ユーザ認証

ユーザ認証とは

ユーザ認証は、ユーザIDとパスワードによって、正当なユーザであるかをチェックする機能です。これにより、不当なユーザからのアクセスを防止することができます。

セキュリティロール

セキュリティロールとは、ユーザに割り当てられている権限であり、ユーザのグルーピングとしての働きを持ちます。
(例:Administrator、Guest、Managerなど)
セキュリティロールは、ユーザ認証により得ることができます。

セキュリティロールにより、ユーザをグループ化してアクセス制限を設定することができます。
例えば、「AdministratorまたはManagerのセキュリティロールに所属しているユーザのみアクセスを許可する」といった指定が可能です。

他のセキュリティ機能との関係

他のセキュリティ機能(アクセス制限、メソッドパーミッション等)では、ユーザ認証で得られたユーザ情報(ユーザ名、パスワード、セキュリティロール)を参照し、そのユーザが持つ権限の範囲内でアクセスを許可します。

ディレクトリサービス

Interstage Application Serverでは、ユーザ/セキュリティロールの管理簿として、Interstage ディレクトリサービス(以下ディレクトリサービスと記述します)を使用します。
J2EEアプリケーションのセキュリティ機能を利用するためには、ディレクトリサービスを準備し、ユーザを登録しておく必要があります。

注意

Interstage ディレクトリサービスは、以下の製品で同梱されています。同梱されてない製品を利用する場合は、別途、準備してください。

- Interstage Application Server Enterprise Edition
- Interstage Application Server Standard-J Edition

ユーザ認証を行うアプリケーション

J2EEアプリケーションクライアント

JNDI環境プロパティに設定されたユーザIDとパスワードが、ディレクトリサービスに作成されている場合、ユーザが認証されます。

Webアプリケーション

認証画面から入力されたユーザID名とパスワードが、ディレクトリサービスに作成されている場合、ユーザが認証されます。
ユーザ認証の方法として、以下を使用することができます。

- HTTP Basic認証
  Webブラウザが提供する認証画面(ダイアログ)を使用する
- フォームベース認証
  認証画面として作成した任意のページ(HTML、JSPなど)を使用する

5.1.2 アクセス制限

アクセス制限は、Webアプリケーションのリソースごとに以下の単位で設定することができます。

- セキュリティロール
- 転送方法

アクセス制限の結果により、ServletコンテナはWebサーバ経由でWebブラウザに以下の応答を返却します。

- アクセス可能:HTTP ステータスコード 200
- アクセス不可/許可されないユーザ:HTTP ステータスコード 401
- アクセス不可/許可されない転送方法、セキュリティロール:HTTP ステータスコード 403

セキュリティロール

ユーザ認証によって得られたセキュリティロールによりアクセスを制限します。

転送方法

クライアントーWebサーバ間の転送方法(通信方法)に対してアクセスを制限します。
アクセスを制限できる転送方法として以下の種類があります。

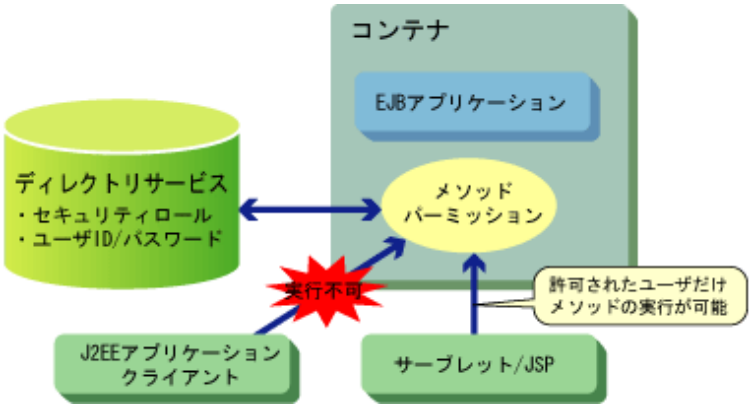
- NONE:データの転送保証は必要としません。
- INTEGRAL:データの転送保証を必要とします。
- CONFIDENTIAL:データの盗聴防止を必要とします。

INTEGRAL、CONFIDENTIALはSSLによる通信の場合にアクセスを許可します。
例えば、「SSLを使用している場合にのみアクセスを許可する」といった指定が可能です。

5.1.3 メソッドパーミッション

メソッドパーミッションは、EJBアプリケーションのメソッドに対するアクセスを制限する機能です。
メソッドパーミッションのしくみを以下に示します。

1. EJBアプリケーションのメソッドに、アクセス可能なセキュリティロールを定義しておきます。
2. EJBアプリケーションのメソッドにアクセスがあると、コンテナは、実行者のユーザIDからセキュリティロールを取得します。
3. 取得したセキュリティロールが、メソッドに定義されていれば、アクセスを許可します。



メソッドパーミッションでは、実行者のユーザ情報を使用します。
そのため、J2EEアプリケーションクライアント、または、Webアプリケーションで、ユーザ認証が行われている必要があります。

5.1.4 セキュリティ関連のメソッド

EJBアプリケーションでは、次のセキュリティ関連のメソッド(javax.ejb.EJBContextインタフェースのメソッド)を使用することが可能です。

- getCallerPrincipal()
- isCallerInRole(java.lang.String roleName)

このメソッドを使用することで、EJBアプリケーションのビジネスメソッド内での認証情報の獲得や、アクセス承認をすることが可能です。
メソッドの詳細については、“Javadoc集”を参照してください。

5.1.5 リソース接続者管理機能

リソース接続者管理機能は、リソース接続者を指定することで、不当なリソースアクセスを防ぐための機能です。
この機能は、リソースマネージャがJDBCとconnectorのときだけ有効です。

リソース接続者の指定方法は、各J2EEアプリケーションのdeployment descriptorの“リソース接続者の指定”(resource-refタグ内のres-authタグ)で定義します。
以下の値が指定可能です。

- Container :リソース定義で設定された接続情報を使用する。
- Application:各アプリケーションで設定された接続情報を使用する。

Container指定について

リソース定義で設定された接続情報とは、Interstage管理コンソールで指定したユーザIDとパスワードです。Interstage管理コンソールの詳細はヘルプを参照してください。

Application指定について

各アプリケーションで設定された情報とは、以下の2つのパターンがあります。

ー EJBアプリケーションからリソース接続する場合で、かつ、
EJBアプリケーションにリソース接続者が指定されている場合
EJBアプリケーションで指定したユーザIDとパスワード

ー その他の場合
J2EEアプリケーションクライアントでユーザ認証されたユーザIDとパスワード

注意

WebアプリケーションではApplication指定をサポートしていません。Container指定されたものとして動作します。

例

J2EEアプリケーションクライアントでの、リソース接続者の指定例を以下に示します。
以下の例では、jdbc/DB1というリソースにアクセスするために、J2EEアプリケーションクライアントでユーザ認証されたユーザIDとパスワードを使用することを定義しています。

```
・・・
<resource-ref>
    <description>JDBC Information</description>
    <res-ref-name>jdbc/DB1</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Application</res-auth>
</resource-ref>
・・・
```

5.1.6 run-as security機能

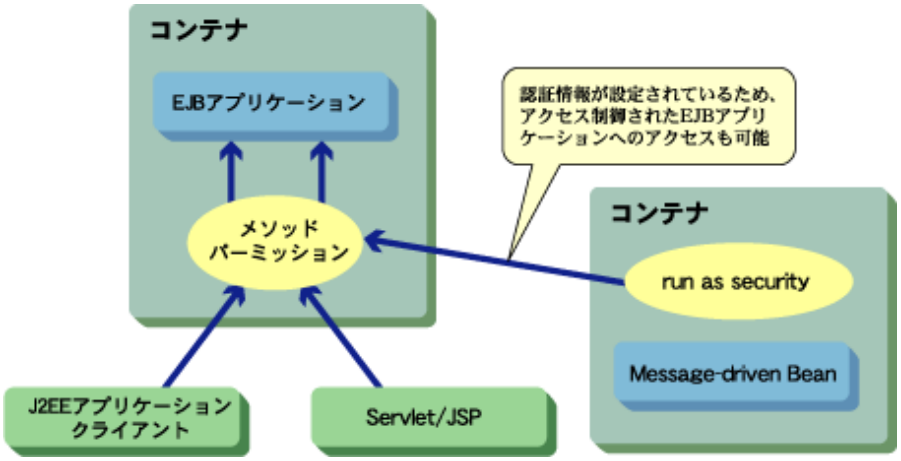
run-as security機能は、EJBアプリケーションに認証情報を指定できる機能です。

通常のEJBアプリケーションで使用される認証情報(セキュリティロール、ユーザID、パスワード)はクライアント(J2EEアプリケーションクライアント、Webアプリケーション)で認証された情報です。しかし、run-as security機能を使用することによってEJBアプリケーションで使用する認証情報を変更することが可能となります。
この機能はMessage-driven Beanから以下のようなセキュリティ機能が使用されたEJBアプリケーションにアクセスする場合に有効な機能です。

- メソッドパーミッションが指定されたEJBアプリケーション
- リソース接続者管理機能に“Application”が指定されたEJBアプリケーション

Message-driven Beanは通常クライアントから直接アクセスされることがないため、認証情報を持ちません。このため、上記のようなEJBアプリケーションにアクセスしようとした場合には、必ずセキュリティチェックされてエラーとなります。このような場合に

run-as security機能を使用してMessage-driven Beanに認証情報を指定することによって、アクセス制御されたEJBアプリケーションへMessage-driven Beanからアクセス可能となります。



run-as security機能を使用するには以下の設定が必要です。

- deployment descriptorの設定
- ユーザID・パスワードの設定
- ディレクトリサービスの設定

deployment descriptorの設定

run-as security機能の設定は、deployment descriptorで行います。security-identity(セキュリティアイデンティティ)のrun-asタグにセキュリティロール名を設定する必要があります。この設定をすることにより、EJBアプリケーションは設定されたセキュリティロールで動作することになります。
deployment descriptorへの設定、および変更は、Interstage StudioまたはInterstage管理コンソールによって行います。

例

run-as securityのdeployment descriptorへの指定例を以下に示します。以下の例ではEJBアプリケーションに“Admin”というセキュリティロールを設定しています。

```
・・・
   <enterprise-beans>
       <security-identity>
           <run-as>
               <role-name>Admin</role-name>
           </run-as>
       </security-identity>
   </enterprise-beans>
・・・
```

ユーザID・パスワードの設定

run-asタグに指定したセキュリティロール名に対応するユーザID・パスワードを、Interstage管理コンソールで指定します。以下のような場合には起動時に警告メッセージEJB1078が出力されます。警告が発生した場合、メソッドパーミッション機能を使用したEJBアプリケーションを呼び出した場合には承認エラーとなります。

- パスワードが誤っている場合
- セキュリティロールがディレクトリサービスに登録されていない場合
- 指定したセキュリティロールに、指定したユーザIDが登録されていない場合

ディレクトリサービスの設定

詳細については、“ディレクトリサービスの設定”を参照してください。

注意

run-as security機能を使用する場合は、以下に注意してください。

当機能は、IJServer外からだけ呼び出される、EJBアプリケーションだけで使用してください。
IJServerに配備されたEJBアプリケーションから、呼び出されるEJBアプリケーションで使用した場合、以下の動作に不具合が生じる可能性があります。

- メソッドパーミッション機能
- リソース接続者管理機能

5.2 セキュリティ機能の組み込み方法

セキュリティ機能の組み込み手順を説明します。

1.ディレクトリサービスの設定

ディレクトリサービスの設定として、以下を設定します。

1. セキュリティ管理環境定義ファイルの設定
   ディレクトリサービスの動作環境を設定します。
2. ユーザ、セキュリティロールの設定
   ディレクトリサービスにユーザ、セキュリティロールを設定します。
   具体的な設定方法については、“5.2.3 ディレクトリサービスの作業手順”を参照してください。

2.アプリケーションごとの設定

アプリケーションごとの設定については、以下を参照してください。

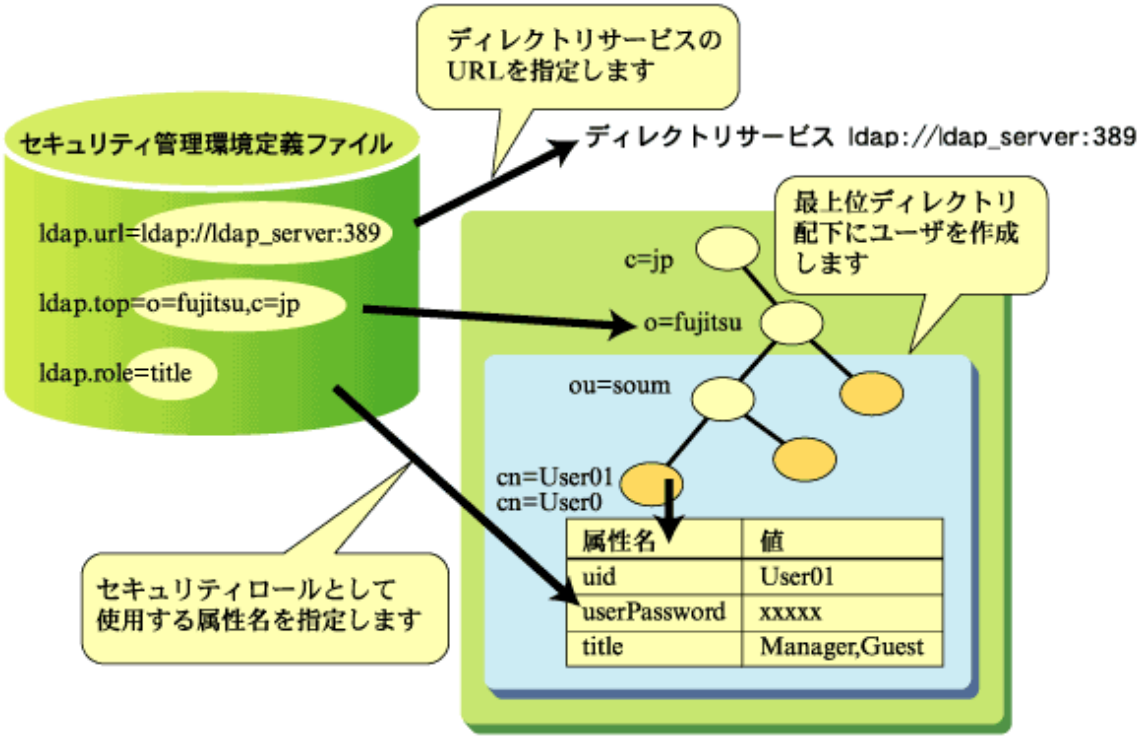
- J2EEアプリケーションクライアントの設定
- Webアプリケーションの設定
- EJBアプリケーションの設定

5.2.1 セキュリティ管理環境定義ファイルの設定

セキュリティ管理環境定義ファイルで、ディレクトリサービスの動作環境を定義する方法を説明します。

セキュリティ管理環境定義ファイルとディレクトリサービスとの関連

セキュリティ管理環境定義ファイルとディレクトリサービスとの関連は下図のとおりです。



セキュリティ管理環境定義ファイルの格納先

セキュリティ管理環境定義ファイルは、J2EEパッケージをインストールすると、次のファイル名で格納されます。

Windows32/64

C:¥Interstage¥J2EE¥etc¥security.properties

Solaris64 Linux32/64

/etc/opt/FJSVj2ee/etc/security.properties

サーバ環境のセキュリティ管理環境定義ファイルと、使用するクライアント環境のセキュリティ管理環境定義ファイルの設定をそれぞれ行ってください。
クライアント環境では、クライアント環境ごとにセキュリティ管理環境定義ファイルがインストールされます。クライアント環境ごとにセキュリティ管理環境定義ファイルの設定を行ってください。
サーバ環境のセキュリティ管理環境定義ファイルとクライアント環境のセキュリティ管理環境定義ファイルは同じ値を設定してください。

セキュリティ管理環境定義ファイルの設定項目

セキュリティ管理環境定義ファイルで設定する項目は下表のとおりです。

項目名	設定値	省略値
ldap.url	ディレクトリサービスのサーバURLを指定します。 “ldap://ホスト名:ポート番号”の形式で指定してください。 「ホスト名」の省略値は「localhost」、「:ポート番号」の省略値は「:389」となります。ただし、この値全体を省略することはできません。	省略不可
ldap.top	ユーザを格納する最上位ディレクトリのDN名を指定します。 ディレクトリサービスで指定した、トップエントリDN以下のDN名を指定してください。 最上位ディレクトリ配下に作成されたユーザが、セキュリティ機能で使用することができます。	省略不可

項目名	設定値	省略値
ldap.role	セキュリティロールとして使用するユーザオブジェクトの属性名を指定します。	省略不可

セキュリティ管理環境定義ファイルは、テキストエディタを利用して編集してください。
1行に“項目名=設定値”の形式で指定し、存在しない項目名が指定された場合は、コメント行とみなします。
以下に指定例を示します。

```
ldap.url=ldap://ldap_server:389
ldap.top=o=fujitsu,c=jp
ldap.role=title
```

注意

ディレクトリサービスで日本語を含むDNを使用し、セキュリティ管理環境定義ファイルのldap.topに日本語を指定する場合は、UNICODEで指定する必要があります。次の手順でUNICODEへのコード変換を行ってください。
変換元ファイルにセキュリティ管理環境定義ファイル名を指定します。

Windows32/64

native2ascii ファイル名 保存先のファイル名
copy 保存先のファイル名 C:¥Interstage¥J2EE¥etc¥security.properties

Solaris64 Linux32/64

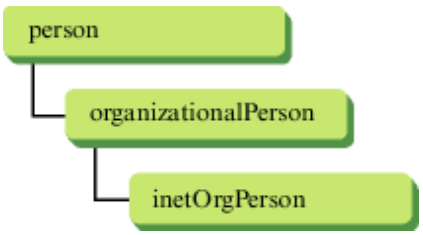
native2ascii ファイル名 保存先のファイル名
cp 保存先のファイル名 /opt/FJSVj2ee/etc/security.properties

なお、セキュリティ管理環境定義ファイルの定義項目を変更した場合は、各コンテナを再起動してください。

5.2.2 ユーザ、セキュリティロールの設定

ディレクトリサービスのディレクトリに、セキュリティ機能で利用するユーザを作成し、ユーザID、パスワード、セキュリティロールを設定する方法を説明します。
ユーザの作成は、ディレクトリサービスが提供する管理ツールを使用します。
手順の詳細については、“5.2.3 ディレクトリサービスの作業手順”を参照してください。

ユーザは次のオブジェクトクラスで作成します。



ユーザID、パスワード、セキュリティロールは下表の属性に設定します。

属性名	属性値	備考
uid	ユーザID	最上位ディレクトリ配下で、ユニークなユーザIDを設定してください。(最上位ディレクトリはセキュリティ管理環境定義ファイルで定義します) ユーザIDが重複する場合、セキュリティが正しく適用され

属性名	属性値	備考
		ません。 2バイトコード文字は指定できません。
userPassword	パスワード	2バイトコード文字は指定できません。
任意に定義	セキュリティロール	セキュリティロールを複数指定する場合は、コンマ‘,’で区切って指定してください。 2バイトコード文字は指定できません。

セキュリティロールの属性名について

セキュリティロールとして使用する属性名は、セキュリティ管理環境定義ファイルで定義します。ディレクトリサービスの既存の属性に、セキュリティロールをマッピングしてください。
例えば、セキュリティ管理環境定義ファイルに、「ldap.role=title」を定義した場合、title属性に設定した値がセキュリティロールとして扱われます。

変更の反映について

ディレクトリサービスへのアクセスはキャッシュされます。
そのため、ディレクトリサービスでのパスワードの変更、セキュリティロールの変更、ユーザの削除は、キャッシュの保持時間である30分間は反映されない場合があります。
変更を直ちに反映したい場合は、セキュリティ機能を使用する機能を再起動してください。

5.2.3 ディレクトリサービスの作業手順

ここでは、Interstage管理コンソールおよびディレクトリサービスが提供する管理ツールを使用して、J2EEアプリケーションのセキュリティ機能を利用するための作業手順について説明します。
Interstage ディレクトリサービスの機能の詳細は、“Interstage ディレクトリサービス運用ガイド”を参照してください。

- Interstage管理コンソールでの操作

  1. リポジトリを作成する
  2. リポジトリを起動する

- エントリ管理ツールでの操作

  3. 接続先リポジトリの設定をする
  4. リポジトリへログインする
  5. ユーザを登録する

1. リポジトリを作成する

Interstage管理コンソールで、リポジトリを作成します。

1. Interstage管理コンソールを起動します。
2. [サービス] > [リポジトリ] > [新規作成]タブで、管理者用DNのパスワードを入力して、リポジトリを作成します。管理者用DNのパスワード以外の入力項目に関しては初期値を使用します。詳細はInterstage管理コンソールのヘルプを参照してください。

2. リポジトリを起動する

Interstage管理コンソールで、リポジトリを起動します。
[サービス] > [リポジトリ]を選択して、1.で作成したリポジトリのチェックボックスを選択して、起動ボタンを押します。

3. 接続先リポジトリの設定をする

エントリ管理ツールで接続先リポジトリの設定を行います。

Windows32/64

C:¥Interstage¥bin¥irepeditentコマンドを実行し、エントリ管理ツールを表示させます。

Solaris64 Linux32/64

Xウィンドウが動作する環境で、/opt/FJSVirep/gui/bin/irepeditentコマンドを実行し、エントリ管理ツールを表示させます。

1. [エントリ管理ツール]ウィンドウの[接続]-[接続情報設定]を選択します。
   → [接続先一覧]ウィンドウが表示されます。
2. 新しい接続先ボタンを押します
   → 接続先名の入力ウィンドウが表示されます。
3. 接続先名を入力して、OKボタンを押します。[接続先一覧]ウィンドウに作成したリポジトリのホスト名、ポート、公開ディレクトリ、管理者用DNを入力して保存ボタンを押します。
   → 確認ダイアログが表示されます。
4. OKボタンを押します。閉じるボタンを押します。

4. リポジトリへログインする

リポジトリへログインします。リポジトリへのログインはリポジトリ管理者の権限で行います。

1. [エントリ管理ツール]ウィンドウの[接続]-[ログイン]を選択します。
   → 接続先名とパスワードの入力ダイアログが表示されます。
2. 接続先名を選択して、管理者DNのパスワードを入力してログインボタンを押します。

5. ユーザを登録する

エントリとしてユーザを登録します。

1. [エントリ管理ツール]ウィンドウの“ディレクトリ”に表示されているトップエントリをダブルクリックします。トップエントリは、セキュリティ管理環境定義ファイル(security.properties)のldap.urlの設定と同一です。
   → 組織単位“User”が表示されます。
   ※組織単位“User”はリポジトリの新規作成時にデフォルトツリーとして作成されます。組織単位“User”が表示されない場合はリポジトリ作成時のデフォルトツリーの作成の設定を確認してください。
2. ユーザ登録先エントリとして組織単位“User”を選択します。その場合、セキュリティ管理環境定義ファイル(security.properties)以下のような設定になります。

```
ldap.top=ou=User,ou=interstage,o=fujitsu,dc=com
```

3. ユーザ登録先のエントリを選択した状態で右クリックし、[追加]を選択します。
4. [オブジェクトクラス一覧]パネルでインターネットユーザを選択します。
5. 右フレームへ以下の情報を入力します。

項目	説明
cn	ユーザを特定するユニークな名称(従業員番号など)を入力します。
sn	ユーザの姓を入力します。
givenName	ユーザの名を入力します。
uid	認証で使用するログイン名を入力します。
employeeNumber	従業員番号を入力します。
userPassword	認証で使用するパスワードを入力します。
ou	ユーザの属する組織単位を入力します。

6. [属性追加]ボタンでロール名の情報を入力します。

属性名	属性値
title	ユーザに該当するロール名を入力します。

※セキュリティ管理環境定義ファイル(security.properties)の“ldap.role”に設定するスキーマ名を“title”から変更することにより、他のスキーマ名をロール名の保存先として使用することも可能です。

7. [OK]ボタンを押します。
   → 選択したエントリに、追加したユーザが追加されます。

5.2.4 J2EEアプリケーションクライアントの設定

ユーザ認証の設定

J2EEアプリケーションクライアントでユーザ認証を行うには、JNDI環境プロパティに次の設定が必要です。

- FJUserID :ディレクトリサービスのユーザ認証で使用するユーザIDを指定します。
- FJPassword:ディレクトリサービスのユーザ認証で使用するパスワードを指定します。

FJUserIDとFJPasswordは、次のいずれかの方法で設定してください。
なお、環境プロパティが重複して指定された場合は、以下の順で上書きされます(“3”で指定された環境プロパティが最も優先されます)。

1. FJjndi.propertiesファイル
2. new javax.naming.InitialContext(Hashtable environment)の引数environment
3. アプリケーション起動時のコマンドラインでの引数(-D)

例

JNDI環境プロパティの設定例を次に示します。

- FJjndi.propertiesファイルによる設定

```
FJUserID=user01
FJPassword=pass01
com.fujitsu.interstage.j2ee.DeploymentDescriptorClient=/export/home/j2eeapl/application-client.xml
```

- new InitialContextの引数による設定

```
・・・
Context ctx = null;
try {
  Hashtable env = new Hashtable (5);
  env.put ("java.naming.factory.initial",
    "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient");
  env.put ("FJUserID", "user01");
  env.put ("FJPassword", "pass01");
  env.put ("com.fujitsu.interstage.j2ee.DeploymentDescriptorClient",
    "/export/home/j2eeapl/application-client.xml");
  ctx = new InitialContext(env);
}
catch (NamingException ne) {
  ne.printStackTrace();
}
・・・
```

- アプリケーション起動時の引数による設定

```
java -
Djava.naming.factory.initial=com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient -
```

```
Dcom.fujitsu.interstage.j2ee.DeploymentDescriptorClient=/export/home/j2eeapl/application-client.xml -DFJUserID=user01 -DFJPassword=pass01 ClientAPP
```

リソース接続者管理機能の設定

リソース接続者管理機能の設定については、“5.1.5 リソース接続者管理機能”を参照してください。

5.2.5 Webアプリケーションの設定

ユーザ認証の設定

Webアプリケーションでユーザ認証を行うには、Webアプリケーション環境定義ファイル(deployment descriptor)の“7.5.14 ユーザ認証”を設定します。

アクセス制限の設定

Webアプリケーションでアクセス制限を行うには、Webアプリケーション環境定義ファイル(deployment descriptor)の以下のタグを設定します。

- “アクセス制限の定義(security-constraintタグ)”
- “セキュリティロールの定義(security-roleタグ)”
- “サーブレットの属性の定義(servletタグ)”のsecurity-role-refタグ

例

Webアプリケーション環境定義ファイル(deployment descriptor)の設定例を次に示します。

```
<servlet>
    :
    <security-role-ref>
        <role-name>
            ADM
        </role-name>
        <role-link>
            Administrator →(*1)
        </role-link>
    </security-role-ref>
    :
</servlet>
:
<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            Shop
        </web-resource-name>
        <url-pattern>
            /Shop/* →(*2)
        </url-pattern>
        <http-method>
            POST →(*3)
        </http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>
            Administrator →(*4)
        </role-name>
    </auth-constraint>
    <user-data-constraint>
```

```
        <transport-guarantee>
            CONFIDENTIAL →(*5)
        </transport-guarantee>
    </user-data-constraint>
</security-constraint>
:
<login-config>
    <auth-method>
        BASIC →(*6)
    </auth-method>
    <realm-name>
        name
    </realm-name>
</login-config>
:
<security-role>
    <role-name>
        Administrator →(*7)
    </role-name>
</security-role>
:
```

上記例は、(*2)に示されるリソースに対し、(*3)に示される方法でアクセスした場合の設定です。
ユーザ認証は、(*6)で示されるようにHTTP Basic認証が行われます。
セキュリティロールは、(*4)に示されるようにAdministratorが許可しています。
(*4)の値は、(*7)で定義します。
(*1)の値は、(*7)で定義します。
(*7)の値は、セキュリティロールとして使用するディレクトリサービスのtitle属性として設定した値を指定してください。
転送方法は、(*5)に示されるように「データの盗聴防止を必要とする」ため、SSLによる通信の場合を許可しています。

例

フォームベース認証の場合の例を以下に示します。

```
<login-config>
    <auth-method>
        FORM
    </auth-method>
    <form-login-config>
        <form-login-page>
            /login.jsp
        </form-login-page>
        <form-error-page>
            /error.jsp
        </form-error-page>
    </form-login-config>
</login-config>
```

リソース接続者管理機能の設定

リソース接続者管理機能の設定については、“5.1.5 リソース接続者管理機能”を参照してください。

5.2.6 EJBアプリケーションの設定

メソッドパーミッションの設定

メソッドパーミッションを使用するためには、deployment descriptorに以下の定義をします。

- セキュリティロール名
- 参照セキュリティ
- メソッドパーミッション

設定の詳細については、“Interstage Studio ユーザーズガイド”を参照してください。

リソース接続者管理機能の設定

リソース接続者管理機能の設定については、“5.1.5 リソース接続者管理機能”を参照してください。

5.3 セキュリティ機能の認証のログ採取

セキュリティ機能の認証のログを採取することができます。これにより、不正なアクセスがないかチェックを行うことができます。

注意

以下のいずれかの条件の場合、ログは出力されません。

- ユーザIDまたはパスワードが指定されていない場合
- ユーザIDまたはパスワードに空文字("")が指定されている場合

設定方法

Java VMの起動時パラメタとして以下の指定を行います。

- ログファイル名:
  -Dcom.fujitsu.interstage.j2ee.security.logfile= ログファイル名
- ログサイズ:
  -Dcom.fujitsu.interstage.j2ee.security.logsize= ログサイズ

J2EEアプリケーションクライアントは、アプリケーション起動時のコマンドラインでの引数で指定します。

Webアプリケーション、EJBアプリケーションは、IJServerのワークユニット定義のJava Command Optionで指定します。

ログファイル名、ログサイズの指定についての説明を以下に示します。

指定項目	説明
ログファイル名	ログのファイル名を指定します。 相対パスで指定した場合は、Java VMの動作するカレントディレクトリからの相対パスとなります。 ログファイル名の指定がない、あるいは、ログファイル名としてディレクトリ名を指定された場合は、ログを採取しません。 注)起動するJava VMごと異なるログファイル名を設定してください。 同一のファイルを指定した場合、複数のJava VMから出力されたログが混在あるいは、部分的に欠損する可能性があります。
ログサイズ	ログの最大サイズを、MB単位で指定します。 サイズを超えた場合は、以下の名前のバックアップファイルを作成し、ログのバックアップを残します。すでにバックアップファイルがある場合、上書きします。 (ログファイル名).old 指定可能な範囲は、1～2147483647です。数値以外を指定された場合、または、省略された場合は、ログサイズは1MBとなります。

メッセージの書式

以下の書式でログが出力されます。

```
[日/月/年 時:分:秒] authentication trace (認証方法, 認証結果, 拒否理由) uid="ユーザ名" role="セキュリティロール"
```

各項目には、以下の要素が出力されます。

項目	説明
[日/月/年 時:分:秒]	事象が発生した日時です。
authentication trace	セキュリティ認証でのアクセス・トレースであることを識別する識別子です。
認証方法	認証の可否を判断した方法です。 ・ ldap : ディレクトリサービスによる認証 ・ cache: キャッシュによる認証
認証結果	認証の結果です。 ・ true : 認証成功 ・ false: 認証拒否
拒否理由	認証が拒否された場合の理由です。 ・ no data: 認証方法に示された場所にデータが存在しない ・ different password: パスワードが異なる ・ over time: キャッシュの有効期限外である 認証された場合は、'-'が出力されます。
ユーザ名	認証を行うユーザ名
セキュリティロール	認証成功の場合、ユーザに対応するセキュリティロールです。 認証が拒否された場合、この項目は出力されません。

以下に出力例を示します。

```
[09/01/2001 12:00:00.000] authentication trace (ldap, true, -) uid="Fujitsu" role="Administrator"
[09/01/2001 12:01:01.000] authentication trace (ldap, false, no data) uid="Fujitsu"
```

5.4 セキュリティ機能の異常時の対処

セキュリティ機能でエラーが発生した場合、各アプリケーションでは、それぞれ次のところにエラーメッセージが出力されます。

- J2EEアプリケーションクライアント
  標準出力、標準エラー出力のログ
- Webアプリケーション、EJBアプリケーション
  IJServerのコンテナログ

参照

エラーの対処に際しては、必要に応じて以下を参照してください。

- セキュリティ管理環境定義ファイル
  “5.2.1 セキュリティ管理環境定義ファイルの設定”

- Interstage ディレクトリサービスに関するエラー
  “メッセージ集”の“メッセージ番号がirepで始まるメッセージ”、または“Interstage ディレクトリサービスが出力するメッセージ”
- セキュリティ機能のトレース機能
  “5.3 セキュリティ機能の認証のログ採取”

エラーメッセージの内容と詳細については、“メッセージ集”の“J2EEアプリケーションのセキュリティ機能で出力するメッセージ”を参照してください。

第2部 Servlet/JSP編

第6章 Servletサービスの機能

本章では、Servletサービスの機能について説明します。

以下は、Servletサービスの基本機能です。

- サーブレットの制御
- JSPの制御

以下は、Servletサービス固有の機能です。

- セションリカバリ機能

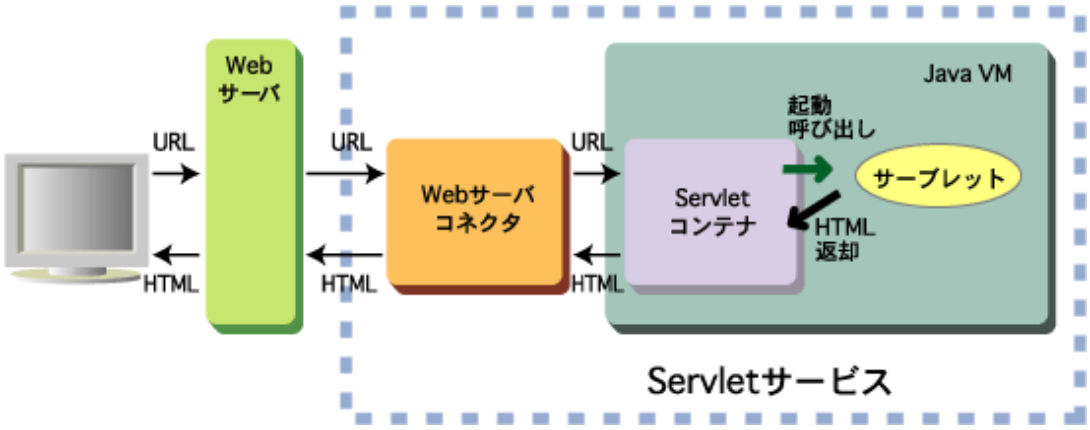
6.1 サーブレットの制御

Servletサービスは、サーブレットの起動(クラスのローディング、インスタンス化)、サーブレットの呼び出しといったサーブレットの制御を行います。
サーブレットが動作する上での基本機能です。
Java™ Servlet API 2.4に準拠しているサーブレットを実行できます。

サーブレットの起動時には、サーブレットをローディングし、サーブレットの初期化(initメソッド)を実行します。
その後、サーブレットをServletコンテナ上に常駐化(インスタンス化)し、Webブラウザからの呼び出しを待ちます。
Webブラウザから呼び出されたときは、サービスの実行(serviceメソッド)を行います。
サーブレットの停止は、Servletコンテナの停止と同時に行います。
このとき、サーブレットの消滅(destroyメソッド)を処理します。

処理の流れを以下に示します。

1. WebブラウザからURLを入力すると、Webサーバを経由してServletサービスがそのURLを受け取ります。
2. Servletサービスでは、URLを解析し、サーブレットの起動と呼び出しを行います。
3. サーブレットはHTMLを返却します。
   Servletサービスは、出力結果であるHTMLを、Webサーバを経由して、Webブラウザに返却します。



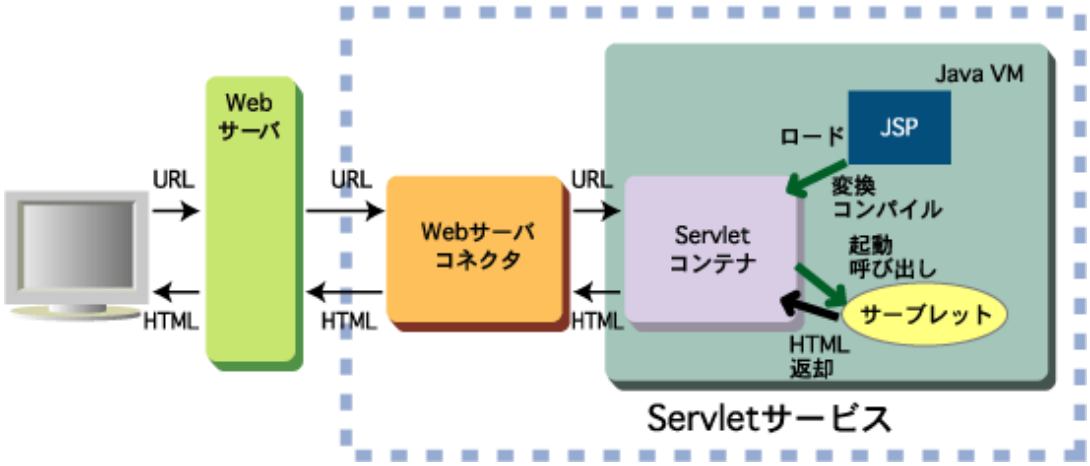
6.2 JSPの制御

Servletサービスは、JSPの制御を行います。
JavaServer Pages 2.0 Specificationに準拠しているJSPの表示ができます。

JSPはサーブレットとして動作します。
Servletサービスは、JSPをサーブレットのソースファイルに変換し、コンパイルします。
その後、生成されたサーブレットを起動します。
Servletコンテナが停止するときは、サーブレットの消滅(destroyメソッド)を処理します。

処理の流れを以下に示します。

1. Webブラウザからurlを入力すると、Webサーバを経由してServletサービスがそのURLを受け取ります。
2. Servletサービスでは、URLを解析し、JSPをロードします。
3. Servletサービスは、JSPをサーブレットのソースファイルに変換し、コンパイルします。
4. コンパイルしたサーブレットを起動します。
5. サーブレットの呼び出し時には、サーブレットはHTMLを返却します。
   Servletサービスは、出力結果であるHTMLを、Webサーバを経由して、Webブラウザに返却します。



6.3 セションリカバリ機能

セションリカバリ機能は、Servletコンテナやマシンがダウンした場合、運用中のServletコンテナがセション情報を引き継ぎ、運用を継続することを可能にする機能です。
セションリカバリ機能を利用することでServletコンテナ、あるいはサーバマシンが複数に分散された環境下において、異常発生時に利用中のセション情報を引き継いで利用することが可能となります。

セション情報は、Servletコンテナ内で、Webアプリケーション単位で管理しています。
Servletコンテナがダウンした場合、そのコンテナ内に格納されたセション情報は破棄されてしまいます。

セションリカバリ機能を有効にすることにより、各Servletコンテナで生成されたセション情報を、Session Registry Serverにバックアップしておき、運用中のServletコンテナにセション情報をリカバリして引き継ぎ、継続して処理を行うことができます。

注意

- セションリカバリ機能は、Web Packageでは使用できません。

第7章 Webアプリケーションの開発

Webアプリケーションは、HTMLファイル、イメージファイル、サーブレット、JSPファイルなどのWebリソースと、Webアプリケーション環境定義ファイル(deployment descriptor)から構築します。
機能を、1つのWebアプリケーションのパッケージ単位として開発できます。

ここでは、以下について説明します。

- Webアプリケーションのディレクトリ構成
- サーブレットの開発
- JSPの開発
- Webアプリケーションの開発上の注意事項
- Webアプリケーション環境定義ファイル(deployment descriptor)

なお、デバッグにつきましては、“3.12 アプリケーションのデバッグ”を参照してください。

7.1 Webアプリケーションのディレクトリ構成

Webアプリケーションのディレクトリ構成について説明します。Webアプリケーションの各ファイルは、ルートとなるディレクトリ(以降、Webアプリケーションのルートディレクトリと呼びます。)配下に以下のディレクトリ構成で作成します。

ディレクトリ名	説明
ルートディレクトリ直下 または WEB-INF以外のディレクトリ	HTMLファイル、イメージファイル、JSPファイルなどを格納します。ディレクトリは自由に作成できます。
WEB-INF	Webアプリケーション環境定義ファイル(deployment descriptor)(web.xml)を格納します。
WEB-INF/classes	サーブレットなどのJavaのクラスファイルを格納します。
WEB-INF/lib	JavaクラスファイルをJAR形式で圧縮した場合の、JARファイルを格納します。

注意

“WEB-INF/classes”配下と“WEB-INF/lib”配下のJARファイルに同名のJavaクラスファイルが存在する場合、“WEB-INF/classes”配下のJavaクラスファイルを優先します。

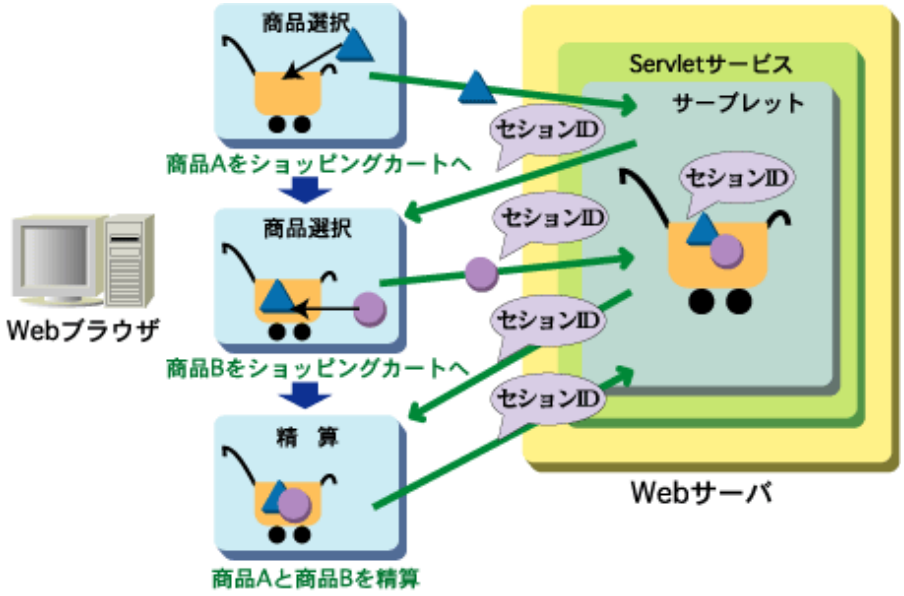
7.2 サーブレットの開発

サーブレットは、Java™ Servlet APIを使用し、自由に作成することができます。
ここでは、代表的なサーブレットの作成方法について説明します。

7.2.1 セション管理

セション管理を利用することで、同一のWebブラウザからの複数回のリクエストを、サーブレットで同一のWebブラウザからのアクセスとして処理することができます。
この機能を利用することで、サーブレットで前回の処理結果を引き継ぐことができ、継続的な処理が可能となります。

セション管理を利用したショッピングの例を示します。



セション管理を利用することで、複数の商品をショッピングカートに入れ、まとめて精算ができるようになります。
セション管理は、HttpSessionインタフェースを使用します。
セションの生成/取得には、getSessionメソッドを使用します。
また、セションへの情報の設定/取得は、setAttributeメソッドとgetAttributeメソッドを使用します。

セションは、encodeURLメソッドによりURLパラメタに埋め込まれるIDで管理されます。
Cookieを使用しないクライアントや、運用環境でCookieを使用しない場合に対応するため、encodeURLメソッドを使用してアプリケーションを作成しておくことを推奨します。

注意

- セションタイムアウトの発生後に処理を継続した場合、新規のセションが確立されます。
- セションIDの一意性の範囲、桁数はServletサービスの実装により異なる可能性があります。セションIDをセション識別以外の目的で使用しないようにしてください。
- Cookieによるセション管理はServletコンテナにより自動で行われます。
  アプリケーションでセション管理用Cookieの追加などの操作は行わないでください。
  セション継続やセキュリティ上の問題が発生する場合があります。

7.2.2 スレッド・モデル

Java™ Servlet APIでは、マルチスレッド・モデルとシングルスレッド・モデルの2つのスレッド・モデルが用意されています。
通常はマルチスレッド・モデルでサーブレットを作成します。
2つのスレッド・モデルの違いを説明します。

マルチスレッド・モデル

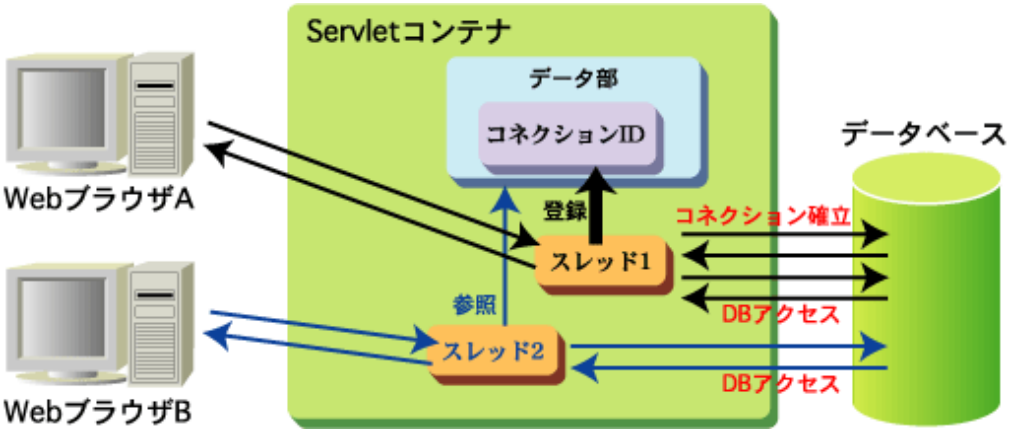
同時に複数の実行スレッドが動作することができますので、多重アクセスの対応に適しています。
メソッドやインスタンス変数をスレッド間で共有し、Servletコンテナによる排他制御が行われません。
そのため、変数の競合を避けたい場合は、メソッドやインスタンス変数をsynchronized宣言する、ローカル変数とするなどのアプリケーションによる制御を行ってください。

シングルスレッド・モデル

同時に複数の実行スレッドは動作せず、メソッドやインスタンス変数の競合を避けるため、Servletコンテナによる排他制御が行われます。
そのため、後続のリクエストは実行中のサーブレットの処理が終了するまで待ち状態となります。
多重アクセスに対応する場合には不向きです。

マルチスレッド・モデルの適用例

以下にマルチスレッド・モデルを利用したDBアクセスの適用例を示します。



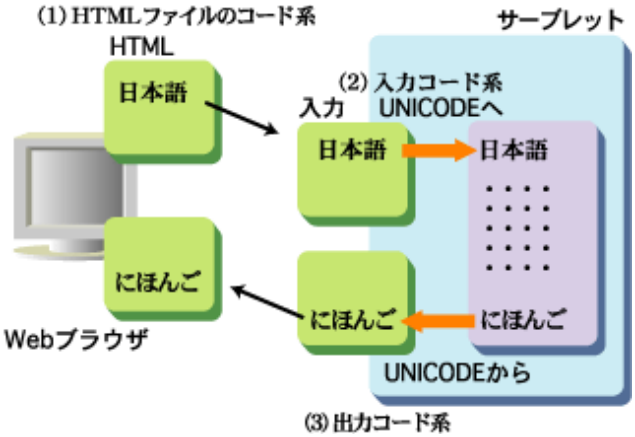
WebブラウザAからのリクエストで確立したコネクションIDをインスタンス変数に格納します。
WebブラウザBからのリクエストでは、インスタンス変数に格納されているコネクションIDを利用します。
コネクションIDを共有することにより、2回目以降のデータベースへのアクセスの高速化が実現します。

7.2.3 日本語コード系

サーブレットで日本語文字を処理するためには、システム全体のコード系を考慮する必要があります。

サーブレットはJava言語で作成するため、サーブレット内部で処理する日本語コード系はUNICODEです。
そこで、Webブラウザから入力された日本語文字はUNICODEへ、Webブラウザへ表示する文字はUNICODEから日本語のコード系に変換する必要があります。

日本語コード変換の流れを以下に示します。



入力コード系と出力コード系に指定できる文字コードセットについては、JDKのドキュメントを参照してください。
JDKのバージョンで異なりますが、EUCJIS、EUC_JP、SJIS、Shift_JIS、ISO-2022-JPなどを日本語コード系として指定することができます。

また、CORBAアプリケーションのメソッドを呼び出す場合は、UNICODEのデータをパラメタに設定してください。CORBAサーバアプリケーションからも日本語文字はUNICODEで返却されます。

注意

- コード系に存在しない文字がある場合、コード変換が正しく行われないことがあります。HTMLファイルのコード系、入力コード系、出力コード系は統一することをお勧めします。
- 入力コード系に“JISAutoDetect”(自動変換)が設定されている場合、コード変換が正しく行われないことがあります。HTMLファイルのコード系と同じコード系を設定することをお勧めします。
- CORBAアプリケーションとの連携、EJBアプリケーションとの連携、トランザクションアプリケーションとの連携を行う場合には、必要に応じて入出力データのコード変換をサーブレットで行ってください。

7.3 JSPの開発

JSPは、JavaServer Pages™を使用し、自由に作成することができます。
ここでは、代表的なJSPの作成方法について説明します。

7.3.1 ビジネスロジックの埋め込み

JSPでは、ビジネス(アプリケーション)ロジックとプレゼンテーションロジックとを分離することができます。
ビジネスロジックはサーブレットやJavaBeansに任せ、JSPはプレゼンテーションロジックを担当します。

ビジネスロジックは、以下の2とおりの方法で埋め込むことができます。

- Java言語の埋め込み
- JavaBeansの呼び出し

Java言語の埋め込み

JSPにサーブレットと同等の処理を埋め込みます。“<%”と“%>”の中に記述します。

```
<HTML>
・・・
<BODY>
<%
for(counter=0; counter<5; counter++) {
    out.println(counter+"<BR>");
}
%>
・・・
```

JavaBeansの呼び出し

JavaBeansを開発し、JSPからはJavaBeansを呼び出します。

JavaBeansを呼び出す場合に使用するタグについて説明します。
なお、その他にも指定可能な属性があります。
タグの記述形式や属性の詳細については、JavaServer Pages™ Specification Version 2.0を参照してください。

タグ名	説明
<jsp:useBean>	JavaBeansのインスタンスを生成します。 以下の属性を指定します。 ・ id属性 JSPファイルの中で使用するインスタンス名を指定します。 ・ class属性 JavaBeansのクラス名を指定します。

タグ名	説明
	・ scope属性 インスタンスの有効範囲を指定します。 page、request、session、applicationから選択します。
<jsp:getProperty>	インスタンスが持っているプロパティの値(インスタンス変数に設定されている値)を取り出して、ブラウザに表示します。 以下の属性を指定します。 ・ name属性 インスタンス名を指定します。 ・ property属性 プロパティ名(インスタンス変数名)を指定します。
<jsp:setProperty>	インスタンスが持っているプロパティの値(インスタンス変数に設定されている値)を設定します。 以下の属性を指定します。 param属性とvalue属性はどちらかを指定します。 ・ name属性 インスタンス名を指定します。 ・ property属性 プロパティ名(インスタンス変数名)を指定します。 "*"を指定した場合は、Webブラウザから送信されたすべてのパラメタの値を設定します。 ・ param属性 <form>タグから入力された値を設定する場合にパラメタ名を指定します。 ・ value属性 設定する値を指定します。

サンプル

JavaBeansを呼び出す簡単なサンプルを以下に示します。

sample.jsp

```
<HTML>
・・・
<BODY>
<jsp:useBean id="sample" scope="request" class="MyApp.AccessName"/>
・・・
<jsp:setProperty name="sample" property="name" value="TARO"/>
・・・
<jsp:getProperty name="sample" property="name"/>
・・・
```

AccessName.java

```
package MyApp;
public class AccessName {
・・・
  String name;
  public AccessName() {
  }
  public String getName() {
    return name; }
  public void setName(String inputname) {
    name = inputname; }
}
```

7.3.2 セション管理

JSPでも、同一のWebブラウザからの複数回のリクエストを、同一のWebブラウザからのアクセスとして処理することができます。

サーブレットのセション管理の処理をJava言語の埋め込みとして、JSPに埋め込むことができます。また、JavaBeansを利用した方法でも実現できます。
<%@ page>タグのsession属性に"true"と、<jsp:useBean>タグのscope属性に"session"を指定します。JavaBeansの呼び出しについては、“7.3.1 ビジネスロジックの埋め込み”を参照してください。

セションはServletコンテナによりセションIDを使用して自動的に管理され、セションIDは、CookieまたはURLパラメタに埋め込まれます。URLパラメタを使用する場合は、リンク等を出力するロジックで、encodeURL メソッドを使用してください。
Webブラウザ、運用環境共にCookieを使用可能な場合はセションはCookieで管理されますが、encodeURLメソッドを使用することで、Cookieを使用可能でない場合でもセションを使用することができます。

注意

- セションタイムアウトの発生後に処理を継続した場合、新規のセションが確立されます。
- セションIDの一意性の範囲、桁数はServletサービスの実装により異なる可能性があります。セションIDをセション識別以外の目的で使用しないようにしてください。
- Cookieによるセション管理はServletコンテナにより自動で行われます。
  アプリケーションでセション管理用Cookieの追加などの操作は行わないでください。
  セション継続やセキュリティ上の問題が発生する場合があります。

サンプル

セション管理にCookieを使用したセション管理を利用した簡単なサンプルを以下に示します。
本サンプルでは、1ページ目で設定した名前を2ページ目で表示しています。

sample1.jsp

```
<HTML>
・・・
<BODY>
<FORM action="sample2.jsp">
<INPUT type="submit" value="NEXT">
</FORM>
<%@ page ・・・ session="true"%>
<jsp:useBean id="sample" scope="session" class="MyApp.AccessName"/>
・・・
<jsp:setProperty name="sample" property="name" param="username"/>
・・・
```

sample2.jsp

```
<HTML>
・・・
<BODY>
<%@ page ・・・ session="true"%>
<jsp:useBean id="sample" scope="session" class="MyApp.AccessName"/>
・・・
<jsp:getProperty name="sample" property="name"/>
・・・
```

AccessName.java

```
package MyApp;
public class AccessName {
・・・
  String name;
  public AccessName() {
```

```
  }
  public String getName() {
    return name; }
  public void setName(String inputname) {
    name = inputname; }
}
```

7.3.3 日本語コード系

JSPで日本語のページを扱うためには、文字コードセットを指定します。
以下のサンプルで説明します。

```
<HTML>
<!--All Rights Reserved, Copyright (C) 富士通株式会社 2003-->
<%@ page session="false" contentType="text/html; charset=SJIS" %>
<% response.setContentType("text/html; charset=EUC_JP"); %>
<TITLE>日本語テスト(EUC_JP 出力)</TITLE>
<BODY BGCOLOR="black" TEXT="yellow">
<H1>日本語テスト(EUC_JP 出力)<BR></H1>
<HR>
<H2>このページは「EUC_JP」です。</H2>
</BODY>
</HTML>
```

<%@ page>タグのcharset属性では、JSPを記述したコード系を指定します。
日本語コードを扱うためには、charset属性は必ず指定してください。

出力コード系は、“<%.....%>”内でsetContentTypeメソッドのより指定します。
JSPを記述したコード系と同じときには、省略することができます。

コード系の"charset"に設定する文字コードセットについては、JDKのドキュメントを参照してください。
JDKのバージョンで異なりますが、EUCJIS、EUC_JP、SJIS、Shift_JIS、ISO-2022-JPなどを日本語コード系として指定することができます。

7.4 Webアプリケーションの開発上の注意事項

ここでは、Webアプリケーションを開発する上での注意事項について説明します。

7.4.1 Cookie使用時の注意

一部のWebブラウザでは、ポート番号に80(SSL通信の場合はポート番号443)を指定した場合と省略された場合とが別サーバであると判断され、Cookieヘッダが送信されない場合があります。
そのため、アプリケーションでCookieによる制御を行う場合は、Webブラウザからの呼び出し方がどちらかに統一されるようにHTML、アプリケーションを構築することをお勧めします。

7.4.2 Cross-site-Scriptingの脆弱性の問題

Webブラウザから入力された値をそのままWebブラウザに返却したり、JavaのErrorやExceptionの内容を返却したりするアプリケーションは、セキュリティホール(Cross-site-Scriptingの脆弱性の問題)となる可能性があります。
このようなアプリケーションは作成しないことをお勧めします。
Cross-site-Scriptingの説明については、“使用上の注意”の“Interstage共通の注意事項”の“Cross-Site Scripting問題について”を参照してください。

7.4.3 ErrorやExceptionについて

アプリケーションで発生したErrorやExceptionをWebブラウザに返却するアプリケーションは、内部の情報の漏洩につながるため、作成しないことを推奨します。
また、サーブレットやJSPで処理していない(catchしていない)ErrorやExceptionに対して、Webアプリケーション環境定義ファ

イルまたはJSP内でエラーページを指定していない場合、Servletコンテナのもつエラーページが表示されます。この場合、ExceptionやErrorのスタックトレースは出力されません。

7.4.4 HTTPエラーステータスコードに対するエラーページの指定について

ここでは、HTTPエラーステータスコードに対するエラーページの指定箇所と使用されるエラーページについて説明します。

HTTPエラーステータスコードに対するエラーページの指定箇所について

HTTPエラーステータスコードに対するエラーページの指定箇所を以下に示します。

- Webアプリケーション環境定義ファイル(deployment descriptor)
- Webサーバの環境設定

どの指定箇所で指定したエラーページが使用されるかは、問題の発生箇所により異なります。必要に応じてエラーページの表示内容を変更または統一してください。

Webアプリケーション環境定義ファイル(deployment descriptor)

<error-page>タグで指定します。指定方法については“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”を参照してください。
本エラーページはWebアプリケーションで発生したHTTPエラーステータスコードに対して有効です。
HTTPエラーステータスコードに該当するエラーページを指定していない場合は、Servletコンテナによってデフォルトのエラーページが使用されます。
本エラーページが使用される場合、HTTPステータスコードは変更されません。例えばExceptionに対するerror-pageを設定した場合、HTTPステータスコードは、500となります。
なお、Webブラウザにレスポンスヘッダが送信済みの場合は、送信済みの情報は取り戻すことはできないので、送信済みのHTTPステータスコードになります。

Webサーバの環境設定

以下の場合は、Servletサービスに制御が渡らないためWebサーバの環境設定で指定したエラーページが使用されます。

- リクエストURLに含まれるWebアプリケーション識別子に誤りがある
- Webアプリケーションへのリクエストではない

例

以下のHTTPエラーステータスコードを例に、各指定箇所で指定したエラーページが使用される場合の説明をします。

- HTTPエラーステータスコード 404(Not Found)
- HTTPエラーステータスコード 500(Internal Server Error)

HTTPエラーステータスコード 404(Not Found)

- Webアプリケーション環境定義ファイルで指定したエラーページが使用される場合
  - Webアプリケーション配下のコンテンツが存在しない場合
  - アプリケーションがServlet APIでHTTPエラーステータスコードに404を設定した場合
- Webサーバのエラーページが使用される場合
  - リクエストURLに含まれるWebアプリケーション識別子に誤りがある場合
  - Webアプリケーションへのリクエストではなく、Webサーバ上にコンテンツが存在しない場合

HTTPエラーステータスコード 500(Internal Server Error)

- Webアプリケーション環境定義ファイルで指定したエラーページが使用される場合
  - サーブレットまたはJSPアプリケーション実行中にExceptionまたはErrorが発生した場合
    注)アプリケーション内で発生したエラーを捕捉(catch)したり、JSPのエラーページを設定するなどし、明示的にエラーハンドリングして正常動作させている場合を除きます。

- アプリケーションがServlet APIでHTTPエラーステータスコードに500を設定した場合

ー Webサーバのエラーページが使用される場合

以下の場合に使用されることがあります。

- Webサーバコネクタで異常を検出した場合
  例:IJServerに接続できない場合
  Webサーバコネクタのタイムアウトが発生した場合

注意

Webブラウザの種類や設定によっては意図したエラーページが表示されず、Webブラウザで用意されたエラーページが表示されることがあります。

- 例:Microsoft(R) Internet Explorer 11
  「ツール」→「インターネット オプション」→「詳細設定」→「HTTP エラー メッセージを簡易表示する」が有効 (デフォルト値) の場合。

7.5 Webアプリケーション環境定義ファイル(deployment descriptor)

Webアプリケーション環境定義ファイル(deployment descriptor)は、Webアプリケーションの動作環境を設定します。
Webアプリケーションが複数の場合は、Webアプリケーションの数だけ用意してください。

Webアプリケーション環境定義ファイル(deployment descriptor)の記述形式

deployment descriptorの記述形式はXML形式(XMLスキーマベース)です。deployment descriptorの記述形式を以下に示します。

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
  <display-name>display_name</display-name>
  <context-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </context-param>
  <filter>
    <filter-name>name</filter-name>
    <filter-class>class</filter-class>
    <init-param>
      <param-name>name</param-name>
      <param-value>value</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>value</filter-name>
    <servlet-name>name</servlet-name>
    <dispatcher>value</dispatcher>
  </filter-mapping>
  <filter-mapping>
    <filter-name>value</filter-name>
    <url-pattern>pattern</url-pattern>
    <dispatcher>value</dispatcher>
  </filter-mapping>
  <listener>
```

```
  <listener-class>class</listener-class>
</listener>
<servlet>
  <servlet-name>name</servlet-name>
  <servlet-class>class</servlet-class> または <jsp-file>file-name</jsp-file>
  <init-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </init-param>
  <load-on-startup>priority</load-on-startup>
  <security-role-ref>
    <role-name>name</role-name>
    <role-link>name</role-link>
  </security-role-ref>
</servlet>
<servlet-mapping>
  <servlet-name>name</servlet-name>
  <url-pattern>pattern</url-pattern>
</servlet-mapping>
<session-config>
  <session-timeout>time</session-timeout>
</session-config>
<mime-mapping>
  <extension>ext</extension>
  <mime-type>mime</mime-type>
</mime-mapping>
<welcome-file-list>
  <welcome-file>filename</welcome-file>
</welcome-file-list>
<error-page>
  <error-code>code</error-code> または <exception-type>type</exception-type>
  <location>resource</location>
</error-page>
<resource-env-ref>
  <resource-env-ref-name>env-ref-name</resource-env-ref-name>
  <resource-env-ref-type>type</resource-env-ref-type>
</resource-env-ref>
<resource-ref>
  <res-ref-name>ref-name</res-ref-name>
  <res-type>type</res-type>
  <res-auth>signon</res-auth>
</resource-ref>
<security-constraint>
  <web-resource-collection>
    <web-resource-name>resource-name</web-resource-name>
    <url-pattern>pattern</url-pattern>
    <http-method>method</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>name</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>guarantee-type</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config>
  <auth-method>method</auth-method>
  <realm-name>name</realm-name>
  <form-login-config>
    <form-login-page>login-page</form-login-page>
    <form-error-page>error-page</form-error-page>
  </form-login-config>
</login-config>
```

```
  <security-role>
    <role-name>name</role-name>
  </security-role>
  <env-entry>
    <env-entry-name>entry-name</env-entry-name>
    <env-entry-type>entry-type</env-entry-type>
    <env-entry-value>entry-value</env-entry-value>
  </env-entry>
  <ejb-ref>
    <ejb-ref-name>ref-name</ejb-ref-name>
    <ejb-ref-type>ref-type</ejb-ref-type>
    <home>ejb-home</home>
    <remote>ejb-remote</remote>
    <ejb-link>name</ejb-link>
  </ejb-ref>
  <ejb-local-ref>
    <description>description</description>
    <ejb-ref-name>name</ejb-ref-name>
    <ejb-ref-type>type</ejb-ref-type>
    <local-home>home</local-home>
    <local>remote</local>
    <ejb-link>link</ejb-link>
  </ejb-local-ref>
  <message-destination-ref>
    <description>description</description>
    <message-destination-ref-name>name</message-destination-ref-name>
    <message-destination-type>type</message-destination-type>
    <message-destination-usage>usage</message-destination-usage>
    <message-destination-link>link</message-destination-link>
  </message-destination-ref>
  <message-destination>
    <description>description</description>
    <message-destination-name>name</message-destination-name>
  </message-destination>
  <service-ref>
    <service-ref-name>name</service-ref-name>
    <service-interface>interface</service-interface>
    <wsdl-file>file</wsdl-file>
    <jaxrpc-mapping-file>file</jaxrpc-mapping-file>
  </service-ref>
  <locale-encoding-mapping-list>
    <locale-encoding-mapping>
      <locale>locale</locale>
      <encoding>encoding</encoding>
    </locale-encoding-mapping>
  </locale-encoding-mapping-list>
  <jsp-config>
    <taglib>
      <taglib-uri>uri</taglib-uri>
      <taglib-location>location</taglib-location>
    </taglib>
    <jsp-property-group>
      <url-pattern>pattern</url-pattern>
      <el-ignored>true または false</el-ignored>
      <page-encoding>value</page-encoding>
      <scripting-invalid>true または false</scripting-invalid>
      <is-xml>true または false</is-xml>
      <include-prelude>uri</include-prelude>
      <include-coda>uri</include-coda>
    </jsp-property-group>
  </jsp-config>
</web-app>
```

注意

記述にあたっての注意事項

- マニュアルに記載した定義以外はサポートしていません。
- 先頭の<?xml...>と<web-app xmlns...>は、XML宣言、およびschemaLocationを記述しているため、deployment descriptorファイルの先頭で必ず記述してください。
- deployment descriptorにマルチバイト文字を使用(コメントも含む)する場合は、<?xml...>のエンコード形式(「encoding=」部分)にUTF-8を指定してください。
- 各タグの記載順序は、上記の記載順序に従ってください。
- 省略が“×”であるタグは省略できません。
  省略した場合、タグの定義が無効、またはエラーとなります。
- 複数の指定が“×”であるタグを重複して指定した場合は、最後に指定したタグが有効、またはエラーとなります。
- 大文字・小文字は区別します。
- マニュアルに記載した定義以外を指定した場合、エラーメッセージが出力されずにServletサービスが起動する場合がありますので注意してください。
- Servlet2.2または2.3のdeployment descriptorの記述形式も使用できます。
- 配備済みのアプリケーションのdeployment descriptorの記述形式を、Servlet2.2または2.3からServlet2.4の記述形式に変更する場合は、アプリケーションを配備しなおしてください。
- 名前空間プレフィックスつきのタグは指定しないでください。
  指定した場合、配備に失敗したり、名前変換機能が使用できなくなることがあります。
  例:<pfx:servlet>

Windows32/64

- パス名に指定できる文字は以下のとおりです。
  英数字、'+'、'-'、'_'、'.'、' '、'$'、'%'、':'、'¥'、'~'
- パス名は最大255バイトまでです。

Solaris64 Linux32/64

- パス名に指定できる文字は以下のとおりです。
  英数字、'+'、'-'、'_'、'.'、'$'、'%'、'/'、'~'
- パス名は最大1023バイトまでです。

Webアプリケーション環境定義ファイル(deployment descriptor)のタグ

Webアプリケーション環境定義ファイル(deployment descriptor)には、以下のタグを指定することができます。
web-app以外のタグは省略可能です。必要に応じて定義してください。
各タグの開始タグと終了タグの間に下位階層のタグを設定し、詳細を定義することができます。

タグ名	説明	タグの省略	複数の指定
web-app	Webアプリケーション環境定義ファイル(deployment descriptor)の開始と終了を定義します。	×	×
display-name	サーブレットコンテキストの名前を定義します。	○	×
context-param	サーブレットコンテキストに設定する初期化パラメタを定義します。サーブレットコンテキストには、Webアプリケーションのすべてのサーブレットに共通の情報を設定し、取り出すことができます。	○	○

タグ名	説明	タグの省略	複数の指定
filter	フィルタクラスを定義します。	○	○
filter-mapping	フィルタクラスを適用する対象を定義します。	○	○
listener	Webアプリケーションで発生したイベントに応じて何らかの処理を行いたい場合、ここに実装クラス名を定義します。	○	○
servlet	サーブレットの初期化パラメタ、エイリアスなどの、サーブレットの属性を定義します。	○	○
servlet-mapping	URLをサーブレットやJSPに対応するためのサーブレット・マッピングの定義をします。	○	○
session-config	セション管理を使用する場合、セションパラメタの定義をします。	○	×
mime-mapping	サーブレットAPIで取り出すmimeタイプを定義します。	○	○
welcome-file-list	URLにファイル名を指定しなかった場合に表示するwelcome fileを定義します。	○	○
error-page	エラーコードやJavaの例外タイプに対応するリソースを定義します。	○	○
security-constraint	Webアプリケーションのアクセス制限を定義します。	○	○
login-config	ユーザ認証の方法を定義します。	○	×
security-role	アクセス制限で使用するセキュリティロールを定義します。	○	○
locale-encoding-mapping-list	ロケールと文字コードの対応を定義します。	○	○
jsp-config	Webアプリケーション内のJSPで共通の値を定義します。	○	×

参照

以下のタグについては、“4.9 deployment descriptorファイルへの記述”を参照してください。

- resource-env-ref
- resource-ref
- env-entry
- ejb-ref
- ejb-local-ref
- message-destination-ref
- message-destination
- service-ref

ポイント

各タグ説明の記述例では、説明しているタグ以外のタグの記述は省略しています。
なお、記述例は、Solarisシステムの場合で記述しています。
Windows(R)システムの場合は、適宜、パスを読み替えてください。

7.5.1 Webアプリケーション環境定義ファイル(deployment descriptor)の開始と終了

Webアプリケーション環境定義ファイル(deployment descriptor)の開始と終了は、web-appタグで定義します。

記述形式

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  ・・・
</web-app>
```

記述例

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
  <context-param>
  ・・・
  </context-param>
</web-app>
```

7.5.2 サーブレットコンテキストの名前

サーブレットコンテキストの名前は、display-nameタグで定義します。

設定したサーブレットコンテキストの名前は、以下のメソッドで取得することができます。

- javax.servlet.ServletContext.getServletContextName ()メソッド

記述形式

```
  <display-name>name</display-name>
```

記述例

```
  <display-name>Example Security Constraint</display-name>
```

7.5.3 サーブレットコンテキストの初期化パラメタ

サーブレットコンテキストの初期化パラメタは、context-paramタグで定義します。
サーブレットコンテキストには、Webアプリケーションすべてのサーブレットに共通の情報を設定し、取り出すことができます。
javax.servlet.ServletContext.getInitParameterNames()メソッドとjavax.servlet.ServletContext.getInitParameter()メソッドを使用します。
重複する初期化パラメタ名は定義できません。

記述形式

```
  <context-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </context-param>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
param-name	サーブレットコンテキストの初期化パラメタ名を定義します。パラメタ名を必ず記述してください。値を省略した場合、空文字に対してパラメタ値が設定されます。	×	×
param-value	サーブレットコンテキストの初期化パラメタに指定する値を定義します。値を省略した場合、空文字が設定されます。	×	×

記述例

```
<context-param>
  <param-name>E-mail</param-name>
  <param-value>taro@fujitsu.co.jp</param-value>
</context-param>
```

Webアプリケーションの記述例

```
public doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException,IOException {
String mail = getServletContext().getInitParameter("E-mail");
}
```

7.5.4 フィルタクラス

フィルタ機能を利用するには、“フィルタクラス”と“フィルタクラスを適用する対象”を定義します。ここでは、フィルタクラスの定義について説明します。
フィルタクラスは、filterタグで定義します。
設定したフィルタの初期値はjavax.servlet.FilterConfig.getInitParameter() メソッドで取り出すことができます。

記述形式

```
<filter>
  <filter-name>name</filter-name>
  <filter-class>class</filter-class>
  <init-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </init-param>
</filter>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
filter-name	フィルタクラスの別名です。一意な名前を指定します。 この名前を使用してfilter-mappingで定義したサーブレット名やURIパターンとマッチングされます。	×	×
filter-class	マッピングするフィルタクラスの完全なクラス名を指定します。	×	×
init-param	初期化パラメタとして名前と値のペアをセットします。 パラメタが複数ある場合、パラメタごとに別々の<init-param>タグを使用します。	○	○
param-name	フィルタクラスの初期化パラメタの名前をセットします。 <init-param>タグを使用する場合、必須です。	×	×
param-value	フィルタクラスの初期化パラメタの値をセットします。 <init-param>タグを使用する場合、必須です。	×	×

記述例

“7.5.5 フィルタクラスを適用する対象”の記述例を参照してください。

7.5.5 フィルタクラスを適用する対象

フィルタ機能を利用するには、“フィルタクラス”と“フィルタクラスを適用する対象”を定義します。ここでは、フィルタクラスを適用する対象の定義について説明します。

フィルタクラスを適用する対象は、filter-mappingタグで定義します。
filter-mappingタグでは、リクエストに対して、どのフィルタをどの順番で使えばよいかをかWebコンテナに指示します。
filter-mappingタグは複数定義することができます。フィルタを実行する順番は、以下のように決まります。

1. <url-pattern>要素が定義されている<filter-mapping>の設定が有効になります。
   web.xmlに<url-pattern>要素が定義されている<filter-mapping>の設定が複数存在する場合には、定義されている順番でフィルタを実行する順番が決まります。
2. <servlet-name>要素が定義されている<filter-mapping>の設定が有効になります。
   web.xmlに<servlet-name>要素が定義されている<filter-mapping>の設定が複数存在する場合には、定義されている順番でフィルタを実行する順番が決まります。

記述形式

```
<filter-mapping>
  <filter-name>value</filter-name>
  <servlet-name>name</servlet-name>
  <dispatcher>value</dispatcher>
</filter-mapping>
<filter-mapping>
  <filter-name>value</filter-name>
  <url-pattern>pattern</url-pattern>
  <dispatcher>value</dispatcher>
</filter-mapping>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
filter-name	一意であるフィルタ名を指定します。 この名前がfilterタグ内のfilter-nameタグの値としてマッチしている必要があります。	×	×
url-pattern	フィルタとマッピングするURLパターンを指定します。 servlet-nameタグと同時に指定することはできません。 URLは以下のように記述します。 ・ 単独のURLの場合 URLで呼び出す場合の名前を記述します。 例) /servlet/servlet1 ・ 特定のプレフィックス(パス、識別子)で始まるURLの場合 最後に"/*"を付加し、記述します。 例) /prefix/* ・ 特定の拡張子をもつURLの場合 "*.xxx"を用いて記述します。 例) *.do "*.xxx"を用いて特定の拡張子をもつファイルを指定する場合には、プレフィックスと一緒に指定することはできません。 例) /path/*.do は指定できません。	×	×

タグ名	説明	タグの省略	複数の指定
	特定の拡張子をもつファイルを指定した場合は、Webアプリケーション全体のファイルが対象となります。		
servlet-name	フィルタとマッピングするサーブレット名を指定します。 url-patternタグと同時に指定することはできません。 サーブレット名には、servletタグのservlet-nameタグで指定した名前を記述します。指定していないサーブレット名を記述した場合、または、値を省略した場合は、無効または定義エラーとなります。	×	×
dispatcher	フィルタの呼出しのタイミングを指定します。 以下の値を指定します。 ・ REQUEST(デフォルト値) クライアントからのリクエストに対して、フィルタクラスが有効になります。 ・ FORWARD RequestDispatcherのforward()メソッドやJSPのforwardアクション時に、フィルタクラスが有効になります。 ・ INCLUDE RequestDispatcherのinclude()メソッドやJSPのincludeアクションなど、動的なinclude時に、フィルタクラスが有効になります。 ・ ERROR エラーページ遷移時に、フィルタクラスが有効になります。	○	○ 異なる値で4つ指定可能です。

記述例

特定のサーブレットに対するフィルタの定義を以下に示します。

```
<filter>
  <filter-name>helloWorld</filter-name>            ← フィルタクラスの別名です。
  <filter-class>MyHelloWorldFilter</filter-class>  ← フィルタクラス名です。
  <init-param>
    <param-name>company</param-name>
    <param-value>Fujitsu Ltd.</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>helloWorld</filter-name>            ← フィルタクラスの別名です。
  <servlet-name>MyHelloWorld</servlet-name>        ← サーブレット名です。
</filter-mapping>
```

単独のURL(以下の例ではJSPファイルのURL)に対するフィルタの定義を以下に示します。

```
<filter>
  <filter-name>helloWorld</filter-name>            ← フィルタクラスの別名です。
  <filter-class>MyHelloWorldFilter</filter-class>  ← フィルタクラス名です。
  <init-param>
    <param-name>company</param-name>
    <param-value>Fujitsu Ltd.</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>helloWorld</filter-name>            ← フィルタクラスの別名です。
  <url-pattern>/filter.jsp</url-pattern>
```

```
                                        ← フィルタとマッピングするURIパターンです。
  </filter-mapping>
```

URIパターンに対するフィルタの定義を以下に示します。
以下の例では、ワイルドカードを使って“/jsp/”以下のリソースのすべてがフィルタ機能の対象になることを示しています。

```
  <filter>
    <filter-name>helloWorld</filter-name>            ← フィルタクラスの別名です。
    <filter-class>MyHelloWorldFilter</filter-class>  ← フィルタクラス名です。
    <init-param>
      <param-name>company</param-name>
      <param-value>Fujitsu Ltd.</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>helloWorld</filter-name>            ← フィルタクラスの別名です。
    <url-pattern>/jsp/*</url-pattern>    ← フィルタとマッピングするURIパターンです。
  </filter-mapping>
```

Webアプリケーションの記述例

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyHelloWorldFilter implements Filter {
  String company;
  public void doFilter(ServletRequest req, ServletResponse res,
                       FilterChain chain)
                       throws IOException, ServletException {
    HttpServletResponseWrapper wrapper = new MyResponseWrapper(
                                         (HttpServletResponse)res);
    chain.doFilter(req, wrapper);
    CharArrayWriter writer = new CharArrayWriter();
    writer.write(wrapper.toString().substring(0,
                 wrapper.toString().indexOf("</body>")-1));
    writer.write("<hr>¥n");
    writer.write("Copyrights &copy; " + company + "¥n");
    writer.write("</body>¥n</html>¥n");

    PrintWriter out = res.getWriter();
    res.setContentLength(writer.toString().length());
    out.write(writer.toString());
    out.close();
  }
  public void init(FilterConfig config) throws ServletException {
    company = config.getInitParameter("company");
  }

  public void destroy(){}

  class MyResponseWrapper extends HttpServletResponseWrapper {
    private CharArrayWriter output;
    public String toString() {
      return output.toString();
    }
    public MyResponseWrapper(HttpServletResponse response){
      super(response);
      output = new CharArrayWriter();
    }
    public PrintWriter getWriter(){
      return new PrintWriter(output);
    }
```

```
    }
  }
```

7.5.6 リスナクラス

リスナクラスとは、ライフサイクルイベントが発生したタイミングで呼び出されるクラスのことです。Webアプリケーションでライフサイクルイベントが発生すると、定義したリスナクラスが自動的に起動されます。
リスナクラスは、listenerタグで定義します。
なお、TLD(Tag Library Description file)の<listener>を指定している場合は、両方のリスナが有効になります。

記述形式

```
<listener>
  <listener-class>class</listener-class>
</listener>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
listener-class	以下のイベントを処理する為のクラスの完全なクラス名を指定します。 ・ コンテキストの起動・停止 (javax.servlet.ServletContextListenerインタフェース実装クラス) ・ ServletContextの属性を追加・置換・削除 (javax.servlet.ServletContextAttributeListener インタフェース実装クラス) ・ セションの新規作成・削除 (javax.servlet.http.HttpSessionListenerインタフェース実装クラス) ・ セションに属性を追加・置換・削除 (javax.servlet.http.HttpSessionAttributeListener インタフェース実装クラス) ・ リクエストの開始・終了 (javax.servlet.ServletRequestListenerインタフェース実装クラス) ・ リクエストに属性を追加・置換・削除 (javax.servlet.ServletRequestAttributeListener インタフェース実装クラス) 存在しないクラス名を記述した場合はWebアプリケーションの起動に失敗します。	×	×

記述例

```
<listener>
  <listener-class>listeners.ContextListener</listener-class>
</listener>
```

7.5.7 サーブレットの属性

サーブレットやJSPの属性は、servletタグで定義します。

サーブレットやJSPの属性は、エイリアス、初期化パラメタ、スタートアップを定義することができます。設定した初期化パラメタは、javax.servlet.ServletConfig.getInitParameterNames()メソッドとjavax.servlet.ServletConfig.getInitParameter()メソッドを使用して取り出します。

記述形式

サーブレットを定義する場合

```
<servlet>
  <servlet-name>name</servlet-name>
  <servlet-class>class</servlet-class>
  <init-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </init-param>
  <load-on-startup>priority</load-on-startup>
  <security-role-ref>
    <role-name>name</role-name>
    <role-link>name</role-link>
  </security-role-ref>
</servlet>
```

JSPファイルを定義する場合

```
<servlet>
  <servlet-name>name</servlet-name>
  <jsp-file>file-name</jsp-file>
  <init-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
  </init-param>
  <load-on-startup>priority</load-on-startup>
  <security-role-ref>
    <role-name>name</role-name>
    <role-link>name</role-link>
  </security-role-ref>
</servlet>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
servlet-name	サーブレットやJSPの名前を定義します。 この値は、servlet-mappingタグやfilter-mappingタグで対象サーブレット名を指定する場合にも使用されます。 (補足参照)	×	×
servlet-class	サーブレットの完全なクラス名を定義します。 サーブレットを定義する場合は必須です。	×	×
jsp-file	JSPファイル名をWebアプリケーションのルートディレクトリからの相対パスで定義します。先頭に'/'を付加します。 JSPファイルを定義する場合は必須です。 Windows32/64 相対パスにディレクトリを記述する場合、ディレクトリの間は'¥'でなく、'/'で区切ります。	×	×
init-param	サーブレットまたはJSPの初期化パラメタを定義します。	○	○
param-name	サーブレットまたはJSPの初期化パラメタ名を定義します。 init-paramタグを定義する場合は必須です。 パラメタ名を必ず記述してください。	×	×

タグ名	説明	タグの省略	複数の指定
param-value	サーブレットまたはJSPの初期化パラメタに指定する値を定義します。 init-paramタグを定義する場合は必須です。	×	×
load-on-startup	Servletコンテナ起動時のスタートアップを定義します。 サーブレットやJSPをロードする順序を-2147483648～2147483647で定義します。数値以外は指定できません。 小さい数から順にロードされます。 0を指定した場合は、最初にロードされます。 負の数を指定した場合は、Servletコンテナ起動時にロードされません。 省略した場合は、サーブレットやJSPを呼び出したときにロードされます。 以下の値を指定した場合、そのServletまたはJSPは0を指定した場合と同様に最初にロードされます。 1) -2147483648より小さい値を指定した場合、または 2) 2147483647より大きい値を指定した場合 同じ値が複数定義された場合は、同じ値内ではロード順番は保障されません。	○	×
security-role-ref	サーブレットコードで使用するセキュリティロールの参照を定義します。	○	○
role-name	サーブレットコードで使用されるセキュリティロール名を指定します。 security-role-refタグを定義する場合は必須です。 javax.servlet.http.HttpServletRequest.isUserInRole()メソッドのパラメタとして利用することができます。	×	×
role-link	<security-role>で指定されたセキュリティロール名を定義します。 security-role-refタグを定義する場合は必須です。	×	×

補足)「マッピングがなくてもサーブレットが動作する」を有効にしている場合に限り、サーブレット名を使用してWebブラウザからアクセス可能になります。この場合には以下の文字が使用可能です。

- 英数字、'+'、'-'、'.'、'_'、'$'

記述例

サーブレットを定義する場合

```
<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>com.fujitsu.jservlet.xxx.HelloWorldServlet</servlet-class>
  <init-param>
    <param-name>message</param-name>
    <param-value>I'm a Hello servlet</param-value>
  </init-param>
  <load-on-startup>10</load-on-startup>
  <security-role-ref>
    <role-name>Administrator</role-name>
    <role-link>Manager</role-link>
  </security-role-ref>
</servlet>
<security-role>
  <role-name>Manager</role-name>
</security-role>
```

JSPファイルを定義する場合

```
<servlet>
  <servlet-name>present</servlet-name>
  <jsp-file>/jsp/present.jsp</jsp-file>
  <init-param>
    <param-name>message</param-name>
    <param-value>I'm a Hello JSP</param-value>
  </init-param>
  <load-on-startup>11</load-on-startup>
  <security-role-ref>
    <role-name>Administrator</role-name>
    <role-link>Manager</role-link>
  </security-role-ref>
</servlet>
<security-role>
  <role-name>Manager</role-name>
</security-role>
```

7.5.8 サーブレット・マッピング

指定したURLのファイルやサーブレットを表示せずに異なったサーブレットやJSPに対応させることができます。
このサーブレット・マッピングは、servlet-mappingタグで定義します。
servlet-mappingタグは、サーブレットやJSPの名前を定義した、servletタグより後に記述してください。servletタグより前に記述した場合、Webアプリケーションの起動に失敗します。
url-patternタグに同じURLを複数定義した場合は、最後に定義したサーブレット・マッピングが有効となります。

指定したURLが複数のサーブレット・マッピングで有効である場合、以下の順で優先されます。

- url-patternタグがファイルやサーブレットの名前の場合
- url-patternタグがプレフィックス(パス、識別子)の場合(長い名前の方が優先されます。)
- url-patternタグが拡張子の場合

例

“/index.html”と“*.html”のURLが定義されていて、“/index.html”へのアクセスがあった場合、拡張子“*.html”の定義よりファイルの名前“/index.html”の定義が優先されます。

記述形式

```
<servlet-mapping>
  <servlet-name>name</servlet-name>
  <url-pattern>pattern</url-pattern>
</servlet-mapping>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
servlet-name	リクエストをマッピングするサーブレットやJSPの名前を定義します。 名前には、servletタグのservlet-nameタグで指定した名前を記述します。指定していない名前を記述した場合は、Webアプリケーションの起動に失敗します。	×	×
url-pattern	サーブレットやJSPにマッピングするURLを定義します。 URLは以下のように記述します。	×	×

タグ名	説明	タグの省略	複数の指定
	・ 単独のURLの場合 URLで呼び出す場合の名前を記述します。 例) /servlet/servlet1 ・ 特定のプレフィックス(パス、識別子)で始まるURLの場合 最後に"/*"を付加し、記述します。 例) /prefix/* ・ 特定の拡張子をもつURLの場合 "*.xxx"を用いて記述します。 例) *.do "*.xxx"を用いて特定の拡張子をもつファイルを指定する場合には、プレフィックスと一緒に指定することはできません。 例) /path/*.do は指定できません。 特定の拡張子をもつファイルを指定した場合は、Webアプリケーション全体のファイルが対象となります。		

記述例

単独のURLに対するマッピング定義を以下に示します。

```
<servlet>
  <servlet-name>SendMailServlet</servlet-name>
  <servlet-class>SendMailServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>SendMailServlet</servlet-name>
  <url-pattern>/SendMailServlet</url-pattern>
</servlet-mapping>
```

URLのパス情報が“director”のプレフィクスを持つリクエストに対するマッピング定義を以下に示します。

```
<servlet>
  <servlet-name>director</servlet-name>
  <servlet-class>xxx.yyy.DirectorServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>director</servlet-name>
  <url-pattern>/director/*</url-pattern>
</servlet-mapping>
```

URLが“.do”で終了するリクエストに対するマッピング定義を以下に示します。

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>xxx.yyy.ActionServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

7.5.9 セションパラメタ

セションパラメタは、session-configタグで定義します。
セションパラメタとして、セションタイムアウト時間を設定できます。
設定したセションタイムアウト時間は、javax.servlet.http.HttpSession.getMaxInactiveInterval()メソッドを使用して取り出します。

記述形式

```
<session-config>
  <session-timeout>time</session-timeout>
</session-config>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
session-timeout	セションタイムアウト時間を定義します。 単位は分で、指定可能な範囲は0～35791394です。タグを省略した場合は、30が設定されます。 0または負の値を記述した時にはタイムアウトしません。	○	×

記述例

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

7.5.10 mimeタイプ

mimeタイプは、mime-mappingタグで定義します。
mimeタイプは、Servletコンテナがデフォルトの値を定義しています。
本タグを定義することにより、Webアプリケーション独自のmimeタイプが設定できます。
本タグで設定したmimeタイプは、デフォルトのmimeタイプより、優先されます。
また、重複して定義した場合は、あとから指定したmimeタイプが有効となります。

設定したmimeタイプは、javax.servlet.ServletContext.getMimeType()メソッドを使用して取り出します。

記述形式

```
<mime-mapping>
  <extension>ext</extension>
  <mime-type>mime</mime-type>
</mime-mapping>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
extension	mimeタイプを定義するファイルの拡張子を定義します。	×	×
mime-type	mimeタイプを定義します。	×	×

記述例

```
<mime-mapping>
  <extension>jpg</extension>
  <mime-type>image/jpeg</mime-type>
</mime-mapping>
```

デフォルトのmimeタイプ

extension	mime-type
abs	audio/x-mpeg

extension	mime-type
ai	application/postscript
aif aifc aiff	audio/x-aiff
aim	application/x-aim
art	image/x-jg
asf asx	video/x-ms-asf
au	audio/basic
avi	video/x-msvideo
avx	video/x-rad-screenplay
bcpio	application/x-bcpio
bin	application/octet-stream
bmp	image/bmp
body	text/html
cdf	application/x-netcdf
cer	application/x-x509-ca-cert
class	application/java
cpio	application/x-cpio
csh	application/x-csh
css	text/css
dib	image/bmp
doc	application/msword
dtd	application/xml-dtd
dv	video/x-dv
dvi	application/x-dvi
eps	application/postscript
etx	text/x-setext
exe	application/octet-stream
gif	image/gif
gtar	application/x-gtar
gz	application/x-gzip
hdf	application/x-hdf
hqx	application/mac-binhex40
htc	text/x-component
htm html	text/html
ico	image/x-icon
ief	image/ief
jad	text/vnd.sun.j2me.app-descriptor

extension	mime-type
jar	application/java-archive
java	text/plain
jnlp	application/x-java-jnlp-file
jpe jpeg jpg	image/jpeg
js	text/javascript
jsf	text/plain
jspf	text/plain
kar	audio/midi
latex	application/x-latex
m3u	audio/x-mpegurl
mac	image/x-macpaint
man	application/x-troff-man
mathml	application/mathml+xml
me	application/x-troff-me
mid	audio/midi
midi	audio/midi
mif	application/vnd.mif
mov	video/quicktime
movie	video/x-sgi-movie
mp1 mpa	audio/x-mpeg
mp2 mp3	audio/mpeg
mpe mpeg mpega mpg	video/mpeg
mpv2	video/mpeg2
ms	application/x-troff-ms
nc	application/x-netcdf
oda	application/oda
odb	application/vnd.oasis.opendocument.database
odc	application/vnd.oasis.opendocument.chart
odf	application/vnd.oasis.opendocument.formula
odg	application/vnd.oasis.opendocument.graphics
odi	application/vnd.oasis.opendocument.image
odm	application/vnd.oasis.opendocument.text-master
odp	application/vnd.oasis.opendocument.presentation
ods	application/vnd.oasis.opendocument.spreadsheet

extension	mime-type
odt	application/vnd.oasis.opendocument.text
ogg	application/ogg
otg	application/vnd.oasis.opendocument.graphics-template
oth	application/vnd.oasis.opendocument.text-web
otp	application/vnd.oasis.opendocument.presentation-template
ots	application/vnd.oasis.opendocument.spreadsheet-template
ott	application/vnd.oasis.opendocument.text-template
pbm	image/x-portable-bitmap
pct	image/pict
pdf	application/pdf
pgm	image/x-portable-graymap
pic pict	image/pict
pls	audio/x-scpls
png	image/png
pnm	image/x-portable-anymap
pnt	image/x-macpaint
ppm	image/x-portable-pixmap
pps	application/vnd.ms-powerpoint
ppt	application/vnd.ms-powerpoint
ps	application/postscript
psd	image/x-photoshop
qt	video/quicktime
qti qtif	image/x-quicktime
ras	image/x-cmu-raster
rdf	application/rdf+xml
rgb	image/x-rgb
rm	application/vnd.rn-realmedia
roff	application/x-troff
rtf	text/rtf
rtx	text/richtext
sh	application/x-sh
shar	application/x-shar
smf	audio/x-midi
sit	application/x-stuffit
snd	audio/basic
src	application/x-wais-source
sv4cpio	application/x-sv4cpio
sv4crc	application/x-sv4crc

extension	mime-type
svg	image/svg+xml
svgz	image/svg
swf	application/x-shockwave-flash
t	application/x-troff
tar	application/x-tar
tcl	application/x-tcl
tex	application/x-tex
texi texinfo	application/x-texinfo
tif tiff	image/tiff
tr	application/x-troff
tsv	text/tab-separated-values
txt	text/plain
ulw	audio/basic
ustar	application/x-ustar
vrml	model/vrml
vsd	application/x-visio
vxml	application/voicexml+xml
wav	audio/x-wav
wbmp	image/vnd.wap.wbmp
wml	text/vnd.wap.wml
wmlc	application/vnd.wap.wmlc
wmls	text/vnd.wap.wmlscript
wmlscriptc	application/vnd.wap.wmlscriptc
wrl	model/vrml
xbm	image/x-xbitmap
xht	application/xhtml+xml
xhtml	application/xhtml+xml
xls	application/vnd.ms-excel
xml	application/xml
xpm	image/x-xpixmap
xsl	application/xml
xslt	application/xslt+xml
xul	application/vnd.mozilla.xul+xml
xwd	image/x-xwindowdump
Z z	application/x-compress
zip	application/zip

7.5.11 welcome file

URLにファイル名を入力しなかった場合に表示するファイル(welcome file)を定義できます。
welcome fileは、URLにWebアプリケーション名まで指定した場合や、Webアプリケーションのルートディレクトリからの相対パスとしてディレクトリ名まで指定した場合のどちらにも有効です。
welcome fileを省略した場合、デフォルト設定が使用されます。デフォルト設定のファイルは以下のとおりです。

- index.html
- index.htm
- index.jsp

welcome file(省略時はデフォルト設定)に該当するファイルがない場合はInterstage管理コンソールの[Servletコンテナ設定] > [ファイルの一覧表示]で設定されている値によって、ステータスコード404(ファイルが存在しない)、またはその実体となるディレクトリ配下のディレクトリやファイルの一覧が表示されます
welcome fileは、welcome-file-listタグで定義します。welcome fileは複数指定ができ、記述した順に有効となります。

記述形式

```
<welcome-file-list>
  <welcome-file>filename</welcome-file>
</welcome-file-list>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
welcome-file	welcome fileを定義します。	×	○

記述例

```
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.htm</welcome-file>
</welcome-file-list>
```

7.5.12 エラー発生時のリソース

HTTPエラーやJavaで例外が発生した場合に対応するリソース(HTMLファイル、サーブレット)を定義することができます。
エラー発生時のリソースの定義は、error-pageタグで定義します。
同じHTTPのエラーコードやJavaの例外タイプで複数定義した場合は、最後に定義したリソースの定義が有効となります。

記述形式

HTTPエラーの場合

```
<error-page>
  <error-code>code</error-code>
  <location>resource</location>
</error-page>
```

Javaで発生する例外の場合

```
<error-page>
  <exception-type>type</exception-type>
  <location>resource</location>
</error-page>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
error-code	HTTPのエラーコードを定義します。 HTTPエラーコードを定義する場合に指定します。	×	×
exception-type	Javaの例外タイプの完全クラス名を定義します。 例外タイプを定義する場合に指定します。	×	×
location	エラーが発生した場合に対応するリソース(HTML文書、サーブレットなど)を定義します。Webアプリケーションのルートディレクトリからの相対パスで指定します。このとき、先頭に'/'を付加します。 リソースを省略した場合は、定義エラーとなります。 Windows32/64 相対パスにディレクトリを記述する場合、ディレクトリの間は'¥'でなく、'/'で区切ります。 注) Webブラウザの設定内容によって、Webブラウザ内蔵のエラーページが表示される場合があります。	×	×

error-codeタグまたはexception-typeタグのいずれかを定義します。
どちらも指定されていない場合は、定義エラーとなります。

記述例

HTTPエラーの場合

```
<error-page>
  <error-code>500</error-code>
  <location>/error/http/code500.html</location>
</error-page>
```

Javaで発生する例外の場合

```
<error-page>
  <exception-type>java.lang.IllegalStateException</exception-type>
  <location>/error/exception/IllegalState.html</location>
</error-page>
```

7.5.13 アクセス制限

アクセス制限の定義は、security-constraintタグで定義します。

記述形式

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>resource-name</web-resource-name>
    <url-pattern>pattern</url-pattern>
    <http-method>method</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>name</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>guarantee-type</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
web-resource-collection	Webリソースコレクションを定義します。	×	○
web-resource-name	webリソースのコレクション名を定義します。 web-resource-collectionタグの定義時は、必須です。	×	×
url-pattern	URLパターンを定義します。 Webアプリケーションのルートディレクトリからの対象パスで定義します。このとき、先頭に“/”を追加します。	×	○
http-method	HTTPのメソッド(GET、POSTなど)を定義します。 定義したメソッドに対してだけアクセスが制限されます。 省略した場合は、全メソッドがアクセス制限の対象となります。	○	○
auth-constraint	Webリソースコレクションにアクセス可能なセキュリティロールを定義します。 指定したロールを持つユーザだけ、リソースコレクションにアクセス可能となります。 省略した場合は、すべてのユーザがアクセス可能となります。	○	×
role-name	セキュリティロール名を定義します。 セキュリティロール名が指定された場合、ユーザの識別が必要となるため、ユーザ認証が行われます。 role-nameタグまたは値を省略した場合は、どのユーザもアクセスできません。	○	○
user-data-constraint	データ保護属性を定義します。クライアントーコンテナ間で通信するデータをどう保護すべきであるかを定義します。	○	×
transport-guarantee	クライアントーサーバ間の転送方法を定義します。 user-data-constraintタグの定義時は、必須です。 以下の値を指定します。 ・ NONE アプリケーションが、転送保証を必要としないことを意味します。 ・ INTEGRAL クライアントーサーバ間で送信されるデータの転送保証を必要とすることを意味します。 ・ CONFIDENTIAL データの盗聴の防止が必要であることを意味します。 INTEGRALまたはCONFIDENTIALを指定された場合は、SSLによる通信であることを保証します。	×	×

記述例

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Hello</web-resource-name>
    <url-pattern>/Hello.jsp</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Administrator</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>NONE</transport-guarantee>
```

```
    </user-data-constraint>
  </security-constraint>
```

7.5.14 ユーザ認証

ユーザ認証方法の定義は、login-configタグで定義します。

記述形式

```
  <login-config>
    <auth-method>method</auth-method>
    <realm-name>name</realm-name>
    <form-login-config>
      <form-login-page>login-page</form-login-page>
      <form-error-page>error-page</form-error-page>
    </form-login-config>
  </login-config>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
auth-method	認証方法を定義します。以下の値を指定します。 ・ BASIC HTTP Basic認証 ・ FORM フォームベース認証 auth-methodタグが省略された場合は、“BASIC”が省略値となります。値の省略はできません。	○	×
realm-name	HTTP Basic認証で使用する領域名を定義します。 領域名は、ユーザ認証を行う画面(ダイアログボックス)に表示されます。 HTTP Basic認証を使用しない場合、指定は無視されます。 HTTP Basic認証を使用する場合は、必ず指定してください。	○	×
form-login-config	フォームベース認証定義の開始/終了を定義します。 フォームベース認証で使用するログインページやエラーページを指定します。 フォームベース認証を使用しない場合、指定は無視されます。 フォームベース認証を使用する場合は、必ず指定してください。	○	×
form-login-page	フォームベース認証で使用するログインページを定義します。 form-login-configタグ定義時は、必須です。 指定されたログインページは、フォームベース認証が行われる際にWebブラウザに表示されます。 注)ログインページでは、ユーザ名/パスワードをServletコンテナに受け渡すため、以下のインタフェースを使用しなければなりません。 ・ アプリケーション名:j_security_check ・ パラメタ名->ユーザ名:j_username ・ パラメタ名->パスワード:j_password	×	×

タグ名	説明	タグの省略	複数の指定
	例) : <FORM ACTION="j_security_check" METHOD="POST"> UserName: <INPUT TYPE="text" NAME="j_username"> Password: <INPUT TYPE="password" NAME="j_password"> <input type="submit" value="送信"></FORM> : auth-methodタグでフォームベース認証を指定した場合は、本タグでログインページを必ず指定してください。		
form-error-page	フォームベース認証失敗時に表示するエラーページのロケーションを定義します。 form-login-configタグ定義時は、必須です。 指定されたエラーページは、フォームベース認証失敗時にWebブラウザに表示されます。 フォームログイン失敗時に表示するエラーページのロケーションを指定します。	×	×

記述例

HTTP Basic認証の場合

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Welcome Page</realm-name>
</login-config>
```

フォームベース認証の場合

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.jsp</form-login-page>
    <form-error-page>/error.jsp</form-error-page>
  </form-login-config>
</login-config>
```

留意事項

フォームベース認証は、Webアプリケーションのコンテキスト設定に指定する“ブラウザでセションを保存する設定”によって、認証の継続についての動作が異なります。
Webアプリケーションのコンテキスト設定はInterstage管理コンソールで設定します。

- “ブラウザでセションを保存する設定”を有効にした場合
  ユーザ名/パスワードの入力処理は、クライアントの初回起動時だけ必要です。
  クライアントを終了しても認証は継続します。
  認証の継続時間は、以下によって決まります。

  ー セションのタイムアウト
    セションがタイムアウトするまでの間、認証は継続します。
    セションのタイムアウト時間は、Webアプリケーション環境定義ファイル(deployment descriptor)(web.xml)のsession-configタグ、またはHttpSessionクラスのsetMaxInactiveInterval(int interval)メソッドで定義することができます。
    session-configタグの詳細は、“7.5.9 セションパラメタ”を参照してください。

ー アプリケーションがセションを破棄するまで
アプリケーション内の処理で、HttpSessionクラスのinvalidate()メソッドを使用してセションを破棄するまで、認証は継続します。

注意

フォームベース認証における認証継続処理は、Cookieを利用して実装されています。したがって、Cookieをサポートしていないクライアントや、Cookieを無効にするような運用をしている場合には、“ブラウザでセションを保存する設定”を有効にした場合でも、認証は継続されません。

- “ブラウザでセションを保存する設定”を無効にした場合
ユーザ名/パスワードの入力処理は、クライアントを起動するたびに必要です。
クライアントを終了すると、認証は無効になります。

フォームベース認証使用時には、認証情報はServletのセションに格納されます。
そのため、以下の場合はセションは新規となり、再度認証が必要となることがあります。

- セションリカバリ機能有効、かつ
- Session Registry Client(Servletコンテナ)の[セションリカバリ設定]において、[セションを使用しないURLの末尾]として拡張子を指定している、かつ
- クライアントから、指定した拡張子のコンテンツに対するリクエストを行った、かつ
- IJServerの再起動等の操作を行った、またはIJServerの停止、異常終了等で、別のServletコンテナにリクエストが振り分けられた、かつ
- 該当コンテンツが認証の対象になっている。

7.5.15 セキュリティロール

セキュリティロールの定義は、security-roleタグで定義します。

記述形式

```
<security-role>
  <role-name>name</role-name>
</security-role>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
role-name	セキュリティロール名を定義します。 ロール名には、セキュリティ機能の運用設定(Interstage ディレクトリサービス)で定義したセキュリティロール名を指定してください。	×	×

記述例

```
<security-role>
  <role-name>Manager</role-name>
</security-role>
```

7.5.16 ロケールと文字エンコーディングの対応

ロケールと文字エンコーディングの対応は、locale-encoding-mapping-listタグで定義します。

記述形式

```
<locale-encoding-mapping-list>
  <locale-encoding-mapping>
    <locale>locale</locale>
    <encoding>encoding</encoding>
  </locale-encoding-mapping>
</locale-encoding-mapping-list>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
locale-encoding-mapping	ロケールと文字コードの対応を定義します。	×	○
locale	ISO-639-1とISO-3166で定義されたロケールを指定します。	×	×
encoding	ロケールに対応させる文字コードを指定します。	×	×

記述例

```
<locale-encoding-mapping-list>
    <locale-encoding-mapping>
        <locale>ja</locale>
        <encoding>Shift_JIS</encoding>
    </locale-encoding-mapping>
</locale-encoding-mapping-list>
```

7.5.17 Webアプリケーション内のJSPの共通定義

Webアプリケーション内のJSPで共通の定義は、jsp-configタグで定義します。

記述形式

```
<jsp-config>
  <taglib>
    <taglib-uri>uri</taglib-uri>
    <taglib-location>location</taglib-location>
  </taglib>
  <jsp-property-group>
    <url-pattern>pattern</url-pattern>
    <el-ignored>true | false</el-ignored>
    <page-encoding>value</page-encoding>
    <scripting-invalid>true | false</scripting-invalid>
    <is-xml>true | false</is-xml>
    <include-prelude>uri</include-prelude>
    <include-coda>uri</include-coda>
  </jsp-property-group>
</jsp-config>
```

タグの内容

タグ名	説明	タグの省略	複数の指定
taglib	JSPで独自タグを埋め込む場合、タグライブラリを定義します。	○	○

タグ名	説明	タグの省略	複数の指定
taglib-uri	Webアプリケーションが使用するJSPのタグライブラリのURIを定義します。JSPファイル内の<taglib>指定のuriに定義すべきURIを指定します。	×	×
taglib-location	タグライブラリのTLD(Tag Library Description file)ファイル名を定義します。 Webアプリケーションのルートディレクトリからの相対パスで指定します。このとき、先頭に'/'を付加します。 存在しないパスを記述した場合は、Webアプリケーションの起動に失敗します。 Windows32/64 相対パスにディレクトリを記述する場合、ディレクトリの間は'¥'でなく、'/'で区切ります。	×	×
jsp-property-group	<url-pattern>タグで設定されたURLパターンを満たすJSPに対して、定義や制限を指定します。	○	○
url-pattern	<jsp-property-group>を適用するJSPのURLパターンを指定します。	×	○
el-ignored	Expression LanguageをJSP内で無視するかどうかを指定します。 ・ true: Expression Languageは無視されます(エラーではない) ・ false: Expression Languageは評価されます。 省略値はfalseです。	○	×
page-encoding	JSPファイルの文字コードを指定します。 JSPファイルのpageディレクティブのpageEncodingで指定された値と違う場合はJSPのコンパイルに失敗します。	○	×
scripting-invalid	JSPのスクリプトの記述を非許可とするかどうかを指定します。 ・ true :スクリプトの記述を非許可とします。 JSP内にスクリプトが記述されている場合は、JSPのコンパイルに失敗します。 ・ false:スクリプトの記述を許可します。 scripting-invalidを省略した場合は、スクリプトの記述は許可されます。	○	×
is-xml	JSPファイルを、JSP Documents(XML構文)として処理するかどうかを指定します。 ・ true: JSPをJSP Documents(XML構文)として処理します。 JSPがJSP Documents(XML構文)として記述されていない場合は、JSPのコンパイルに失敗します。 ・ false: JSPをregular JSP page(非XML構文)として処理します。 デフォルトでは、拡張子が「.jspx」の場合にJSP Documents(XML構文)として処理されます。	○	×
include-prelude	JSPの先頭にインクルードするファイルのURIを指定します。 JSPの先頭にincludeディレクティブを指定するのと同じ動作になります。	○	○

タグ名	説明	タグの省略	複数の指定
include-coda	JSPの最後にインクルードするファイルのURIを指定します。JSPの最後にincludeディレクティブを指定するのと同じ動作になります。	○	○

記述例

```
<jsp-config>
    <taglib>
        <taglib-uri>http://www.fujitsu.com/interstage/jservlet/example-taglib</taglib-uri>
        <taglib-location>/WEB-INF/tld/example-taglib.tld</taglib-location>
    </taglib>
    <jsp-property-group>
        <url-pattern>/jsp/*</url-pattern>
        <el-ignored>true</el-ignored>
        <page-encoding>UTF-8</page-encoding>
        <scripting-invalid>false</scripting-invalid>
        <is-xml>false</is-xml>
        <include-prelude>/WEB-INF/jsp/include_pre.jspf</include-prelude>
        <include-coda>/WEB-INF/jsp/include_coda.jspf</include-coda>
    </jsp-property-group>
</jsp-config>
```

JSPファイルの記述例

```
<html>
<body>
<%@ taglib uri="http://www.fujitsu.com/interstage/jservlet/example-taglib" prefix="eg" %>
Radio stations that rock:
<ul>
<eg:foo att1="98.5" att2="92.3" att3="107.7">
<li>
<%= member %>
</li>
</eg:foo>
</ul>
  :
  :
```

注意

タグライブラリ・ディスクリプタファイルの<tag-class>タグを変更した場合の注意

タグライブラリ・ディスクリプタファイルの<tag-class>タグを変更した場合、変更したタグライブラリ・ディスクリプタに対応するタグライブラリを使用するJSPファイルの再コンパイルが必要です。
JSPの再コンパイルは、JSPファイルと対応するjavaのソースファイル、クラスファイルがIJServerディレクトリ直下のworkディレクトリ内に存在しない場合に実行されます。
したがって、JSPファイルと対応するjavaのソースファイル、クラスファイルを削除することによって、JSPの再コンパイルが実行されます。

例えば、Webアプリケーションのルートディレクトリからの JSPファイルのパスが"/jsp/HelloJSP.jsp"の場合、以下のように生成されます。

- ソースファイル名: jsp¥HelloJSP_jsp.java
  クラスファイル名: jsp¥HelloJSP_jsp.class

第8章 Webアプリケーションの呼び出し方法

本章では、Webアプリケーションの呼び出し方法について説明します。

本文中のパスの記述は、Solarisシステムでの表現となっています。
Windows(R)システムの場合は、パスの記述は適宜読み替えてください。

8.1 サーブレットの呼び出し

サーブレットは、WebブラウザのURLや、HTML文書の中のリンクにURLを指定して呼び出します。
サーブレットの呼び出し方について、IJServer単位に以下の指定ができます。

- マッピングが必要な呼び出し方
  マッピングがないとサーブレットを動作させない
- マッピングが不要な呼び出し方
  マッピングがなくてもサーブレットを動作させる

セキュリティの面から、通常はマッピングが必要な呼び出し方を使用することを推奨します。
マッピングが不要な呼び出し方は既定の設定では無効になっています。必要な場合は、Interstage管理コンソールで、[ワークユニット] > “ワークユニット名” > [環境設定] > [詳細設定] > [Servletコンテナ設定] > [マッピングがなくてもサーブレットが動作する]の設定を変更してください。

8.1.1 マッピングが必要な呼び出し方

サーブレットURL

サーブレットURLは、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”の“7.5.8 サーブレット・マッピング”(servlet-mapping)のurl-patternタグで定義したURLパターンに該当するサーブレットのURLです。

サーブレットの格納先

サーブレットは、以下のどちらかに格納します。

- Webアプリケーションのルートディレクトリ/WEB-INF/classes配下
- Webアプリケーションのルートディレクトリ/WEB-INF/lib配下のJARファイル

URLで指定して呼び出す場合

```
http://サーバホスト名:ポート番号/Webアプリケーション名/サーブレットURL
```

“:ポート番号”は省略できます。省略したときのポート番号は80になります。

HTML文書の中で呼び出す場合

```
<A HREF="/Webアプリケーション名/サーブレットURL">Click Here</A>
```

HTML文書の中に入力フィールド(HTMLのFORMタグ)を作って入力した情報を渡すこともできます。
サーブレットは以下のように呼び出します。METHODはGETも指定できます。

```
<FORM ACTION="/Webアプリケーション名/サーブレットURL" METHOD=POST>
```

サーブレットに情報を渡す方法

サーブレットにパラメタを渡したい場合は、サーブレットURLの後ろに次の形式で指定します。

```
サーブレットURL?パラメタ名1=値1&パラメタ名2=値2&・・・
```

また、CGIと同じようにPATH_INFOを使用して、サーブレットにパス情報を渡すことができます。
サーブレットURLの後ろに“/”で始まるパス情報を指定します。
この場合には、サーブレット・マッピングの定義のurl-patternタグの最後に"/*"を付加し、記述します。

```
サーブレットURL/パス情報?パラメタ名1=値1&パラメタ名2=値2&・・・
```

注意

- Webブラウザにサーブレットの実行結果が表示されず、ステータスコードやメッセージが表示される場合は、Webサーバの環境設定やサーブレットの呼び出し方法に誤りがあるなどの原因が考えられます。
  “第29章 J2EEアプリケーション開発・運用時の異常”を参照して、エラーの原因を取り除いてください。
- “WEB-INF/classes”配下と“WEB-INF/lib”配下のJARファイルに同名のサーブレットが存在する場合、“WEB-INF/classes”配下のサーブレットが呼び出されます。
- Windows32/64
  Windows(R)システムでは、指定したサーブレット名の大文字/小文字が間違っている場合、Javaの例外java.lang.NoClassDefFoundErrorが発生します。
  この場合、Webブラウザに“500 Internal Server Error”が表示されます。
  大文字/小文字が間違っていることを伝えるエラーページや“404 Not Found”を伝えるエラーページを作成し、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”の“7.5.12 エラー発生時のリソース”(error-pageタグ)で定義することをお勧めします。

8.1.2 マッピングが不要な呼び出し方

サーブレット名

サーブレットは、サーブレット名を指定して呼び出します。
サーブレット名は、ファイル名から拡張子(.class)を除いた名前です。
サーブレット名の大文字、小文字は区別されます。

サーブレットの格納先

サーブレットは、以下のどちらかに格納します。

- Webアプリケーションのルートディレクトリ/WEB-INF/classes配下
- Webアプリケーションのルートディレクトリ/WEB-INF/lib配下のJARファイル

パッケージとして作成されている場合

サーブレット名にはパッケージ名を付加して呼び出します。
パッケージは、サーブレットのソースコードにpackage宣言をすることで作成できます。
パッケージ名org.xxx.zzzz、サーブレット名HelloWorldServletのときは、以下のようになります。

```
org.xxx.zzzz.HelloWorldServlet
```

URLで指定して呼び出す場合

```
http://サーバホスト名:ポート番号/Webアプリケーション名/servlet/サーブレット名
```

“:ポート番号”は省略できます。省略したときのポート番号は80になります。

HTML文書の中で呼び出す場合

```
<A HREF="/Webアプリケーション名/servlet/サーブレット名">Click Here</A>
```

HTML文書の中に入力フィールド(HTMLのFORMタグ)を作って入力した情報を渡すこともできます。
サーブレットは以下のように呼び出します。METHODはGETも指定できます。

```
<FORM ACTION="/Webアプリケーション名/servlet/サーブレット名" METHOD=POST>
```

サーブレットに情報を渡す方法

サーブレットにパラメタを渡したい場合は、サーブレット名の後ろに次の形式で指定します。

```
サーブレット名?パラメタ名1=値1&パラメタ名2=値2&・・・
```

また、CGIと同じようにPATH_INFOを使用して、サーブレットにパス情報を渡すことができます。
サーブレット名の後ろに“/”で始まるパス情報を指定します。

```
サーブレット名/パス情報?パラメタ名1=値1&パラメタ名2=値2&・・・
```

エイリアスによる指定

サーブレット名は、エイリアスで指定することができます。
エイリアスは、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”の“7.5.7 サーブレットの属性”(servletタグ)で定義します。

例

エイリアスを使用しない例(サーブレット名での指定)

```
http://hostname/webapl1/servlet/HelloWorldServlet
```

エイリアスを使用した例("HelloWorldServlet"をエイリアス"Hello"で定義)

```
http://hostname/webapl1/servlet/Hello
```

注意

- Webブラウザにサーブレットの実行結果が表示されず、ステータスコードやメッセージが表示される場合は、Webサーバの環境設定やサーブレットの呼び出し方法に誤りがあるなどの原因が考えられます。
“第29章 J2EEアプリケーション開発・運用時の異常”を参照して、エラーの原因を取り除いてください。
- “WEB-INF/classes”配下と“WEB-INF/lib”配下のJARファイルに同名のサーブレットが存在する場合、“WEB-INF/classes”配下のサーブレットが呼び出されます。
- Windows32/64
Windows(R)システムでは、指定したサーブレット名の大文字/小文字が間違っている場合、Javaの例外java.lang.NoClassDefFoundErrorが発生します。
この場合、Webブラウザに“500 Internal Server Error”が表示されます。

大文字/小文字が間違っていることを伝えるエラーページや“404 Not Found”を伝えるエラーページを作成し、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”の“7.5.12 エラー発生時のリソース”(error-pageタグ)で定義することをお勧めします。

8.2 JSPの呼び出し

JSPは、WebブラウザのURLや、HTML文書の中のリンクにURLを指定して呼び出します。

JSPの相対パス名

JSPは、ファイル名を指定して呼び出します。
また、URLにはWebアプリケーションのルートディレクトリからの相対パスで指定します。
以降、“JSPの相対パス名”と記述します。

例

JSPのフルパス名とJSPの相対パス名の例を、以下に示します。

JSPのフルパス名が以下の場合

```
Webアプリケーションのルートディレクトリ/jsp/Hello/HelloJSP.jsp
```

JSPの相対パス名は以下のようになります。

Windows(R)システムの場合も、ディレクトリの間は“¥”でなく、“/”で区切ります。

```
jsp/Hello/HelloJSP.jsp
```

URLで指定して呼び出す場合

```
http://サーバホスト名:ポート番号/Webアプリケーション名/JSPの相対パス名
```

“:ポート番号”は省略できます。省略したときのポート番号は80になります。

例

以下に呼び出し例を示します。

```
http://hostname/webapl1/jsp/Hello/HelloJSP.jsp
```

HTML文書の中で呼び出す場合

```
<A HREF="/Webアプリケーション名/JSPの相対パス名">Click Here</A>
```

注意

- WebブラウザにJSPの実行結果が表示されず、ステータスコードやメッセージが表示される場合は、Webサーバの環境設定やJSPの呼び出し方法に誤りがあるなどの原因が考えられます。
  “第29章 J2EEアプリケーション開発・運用時の異常”を参照して、エラーの原因を取り除いてください。

8.3 HTMLやイメージファイルなどのファイルの呼び出し

HTMLやイメージファイルなどのファイルは、WebブラウザのURLや、HTML文書の中のリンクにURLを指定して呼び出します。

ファイルの相対パス名

URLにはWebアプリケーションのルートディレクトリからの相対パスで指定します。
以降、“ファイルの相対パス名”と記述します。

例

ファイルのフルパス名が以下の場合

```
Webアプリケーションのルートディレクトリ/apl/Hello/index.htm
```

ファイルの相対パス名は以下のようになります。
Windows(R)システムの場合も、ディレクトリの間は“¥”でなく、“/”で区切ります。

```
apl/Hello/index.htm
```

URLで指定して呼び出す場合

```
http://サーバホスト名:ポート番号/Webアプリケーション名/ファイルの相対パス名
```

“:ポート番号”は省略できます。省略したときのポート番号は80になります。

例

以下に呼び出し例を示します。

```
http://hostname/webapl1/apl/Hello/index.htm
```

HTML文書の中で呼び出す場合

```
<A HREF="/Webアプリケーション名/ファイルの相対パス名">Click Here</A>
```

第9章 セションリカバリ機能

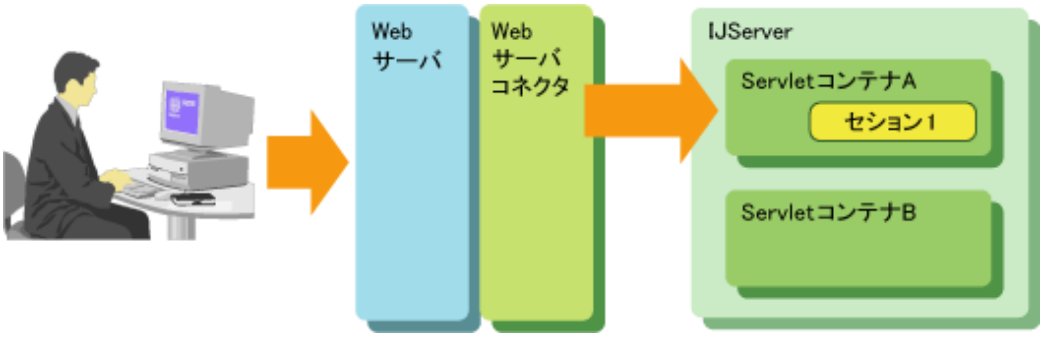
本章では、Servletサービスのセションリカバリ機能について説明します。

9.1 セションリカバリ機能について

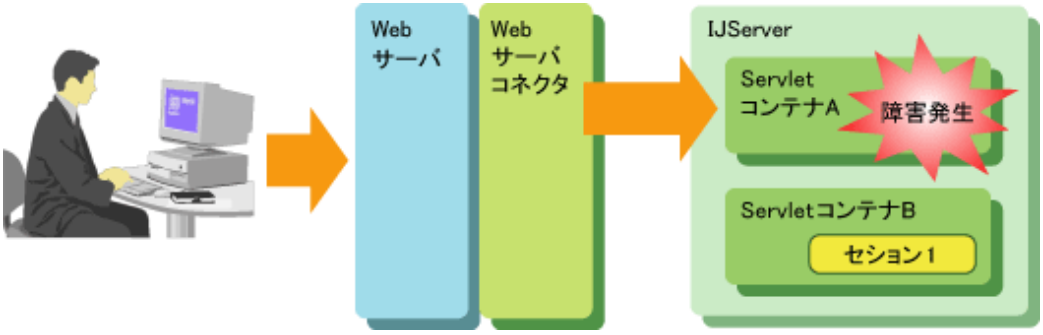
セションリカバリ機能とは

セションリカバリ機能は、Servletコンテナのプロセスダウンまたは、マシンダウンの場合に他の運用中のServletコンテナでServletのセション情報を引き継ぎ、Webアプリケーションの運用を継続して可能にする機能です。
また、IJServerを再起動した場合に、再起動前のServletのセションを破棄せずに継続して使用することも可能になります。

障害発生前



障害発生後



セションリカバリ機能の構成要素

セションリカバリ機能は、以下によって構成されています。

Session Registry Server

Session Registry Serverは、IJServerで使用しているServletのセション情報を保存するサーバです。
障害発生時はSession Registry Serverからセションをリカバリすることで障害発生前のセション情報を使用し、Webアプリケーションを継続して運用することができます。

IJServer(Servletコンテナ)

Webアプリケーションの実行環境です。
障害発生時にWebアプリケーションを継続して運用するためには多重プロセス、または、複数マシンで環境を構築する必要があります。

Session Registry Client

Session Registry Clientは、セションリカバリ機能有効時、IJServer(Servletコンテナ)にアドインして従来のセション管理モジュールにかわって動作し、セションの管理とSession Registry Serverとの通信を行います。

Webサーバコネクタ

WebサーバコネクタはWebサーバが受けたリクエストをServletコンテナに転送する役割を持っています。
通常、Webサーバコネクタはクライアントからのリクエストがservletのセションを持っている場合は、そのセションを作成したServletコンテナへリクエストを振り分けますが、何らかの原因でServletコンテナにリクエストを振り分けることができなかった場合は、他の運用可能なServletコンテナにリクエストを振り分けることでWebアプリケーションの継続運用を実現しています。

9.1.1 セションのバックアップ

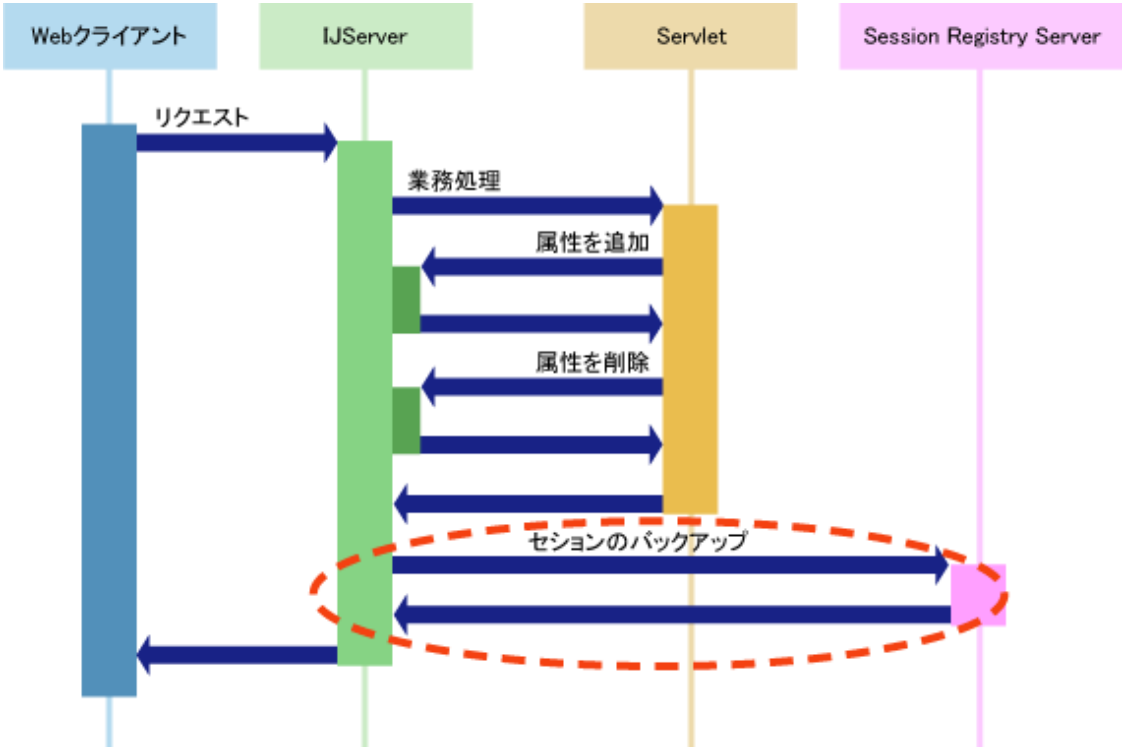
セションはSession Registry Serverにバックアップされ、Servletコンテナの異常発生時にSession Registry Serverにバックアップされたセションを他のServletコンテナにリカバリすることで、他のServletコンテナでセションを引き継ぎます。

セションのバックアップの契機

セションをバックアップする契機は以下の2種類から選択できます。

リクエストごと(完了後)にバックアップします

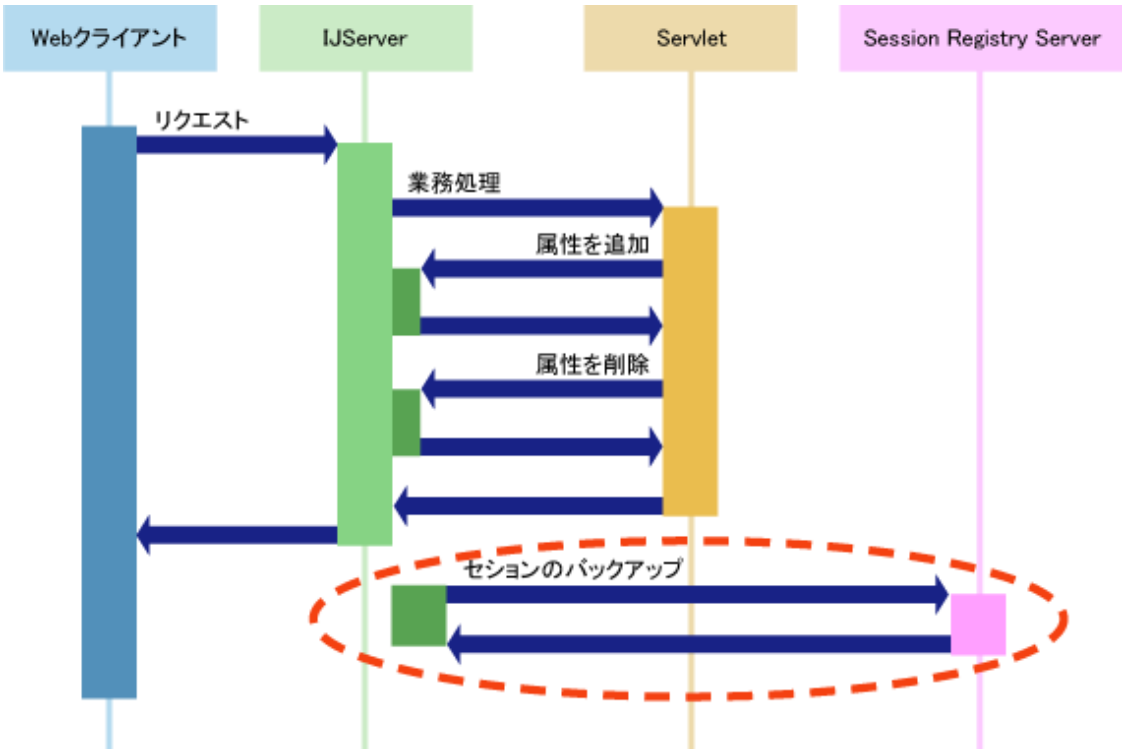
リクエストの完了後にセションをバックアップします。また、クライアントへのレスポンスはセションがSession Registry Serverにバックアップされるのを待ってから完了します。



ー クライアントからのリクエストの処理時間にセションのバックアップ時間も含まれるため、クライアントからのリクエストを処理する時間は一定間隔でバックアップする場合と比較して長くなります。

一定間隔でバックアップします・・・デフォルト値

セションはリクエスト処理が完了した後にServletコンテナによって定期的にバックアップされます。



― リクエスト処理とは独立してバックアップが行われるため、個々のリクエスト返却までの性能には直接影響しません。

― セションがSession Registry Serverにバックアップされる前にServletコンテナがダウンした場合、そのセションはリカバリすることができません。

設定方法

バックアップの契機は、Session Registry Client側で、Interstage管理コンソールの以下の項目で設定します。デフォルト値は“一定間隔でバックアップする”で、バックアップする間隔のデフォルト値は**5**秒です。

- [ワークユニット] > “ワークユニット名” > [環境設定] > [セションリカバリ設定] > [バックアップの契機]

Session Registry Clientの設定については、“9.4 Session Registry Clientの設定”を参照してください。また、Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドを使用して、上記の設定を行うこともできます。isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

バックアップの各契機について、以下の関係が成り立ちます。

- リカバリ時のセションの最新性(より新しい状態のセションのバックアップ保証)
  リクエストごと ＞ 一定間隔
- 性能
  一定間隔 ＞ リクエストごと

セションのバックアップに失敗した場合

セションのバックアップに失敗した場合でもクライアント、および、Webアプリケーションへエラーの通知は行われません。これは、セションのバックアップに失敗した場合、クライアント、および、Webアプリケーションにエラーを通知するとセションリカバリ機能を使用することがアプリケーションの稼働率を下げる要因となってしまうためです。

また、セションのバックアップに失敗した場合は、以降Session Registry Serverは使用不可とマークされ、使用可能となるまで通信(バックアップ、リカバリ)を行いません。
Session Registry Serverが使用可能になる契機については“9.1.4 セションリカバリ機能の監視”を参照してください。

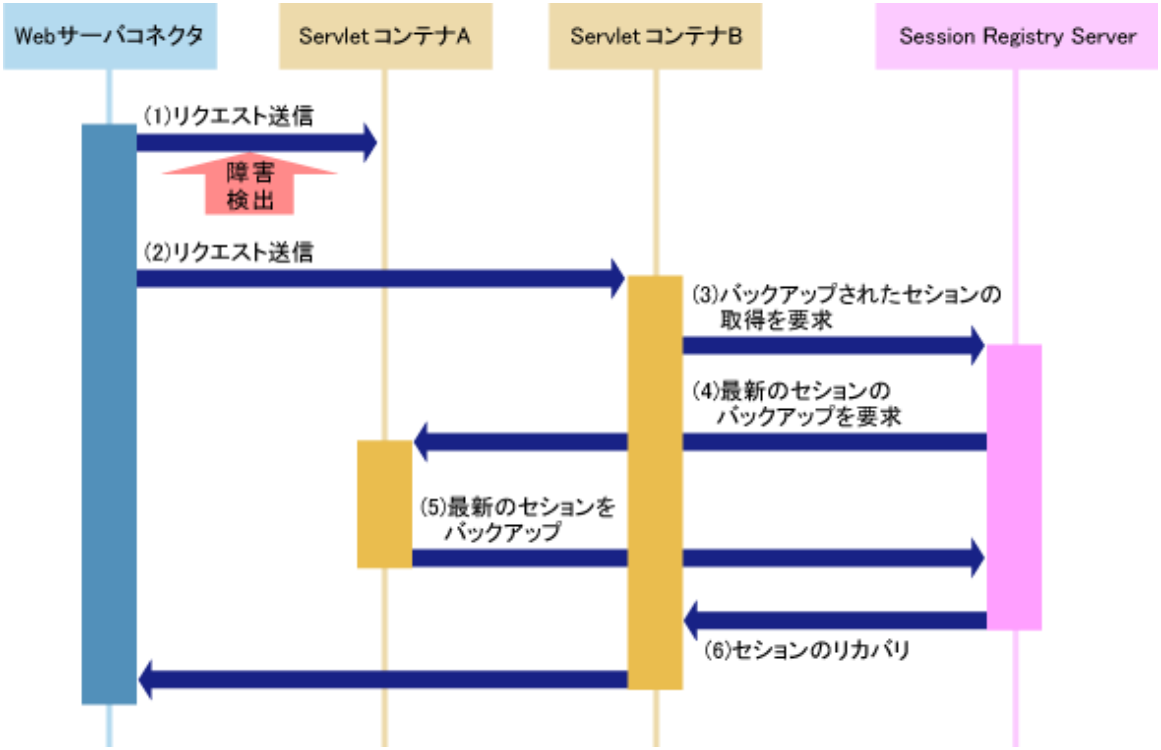
複数のリクエストでセションを処理する場合

同じ1つのセションを扱うすべてのリクエストは同じ時間には1つのJavaVMで処理しなければいけないと規約化されています。したがって、いったん他のServletコンテナにセションがリカバリされた場合は、もとあったServletコンテナのセションは無効にする必要があります。
セションを他のServletコンテナにリカバリする際にSession Registry Serverは可能な限り、もとあったServletコンテナのセションを破棄しようとしますが、ネットワーク異常などで破棄できない可能性があります。その場合、もとあったServletコンテナのセションは、そのServletコンテナとSession Registry Server間の通信の復旧時に破棄されます。

9.1.2 セションのリカバリ

セションのリカバリ処理

Servletコンテナは、クライアントから通知されてきたセションIDに該当するセションを保持していない場合(もともとそのセションを扱っていたServletコンテナがダウンした場合など)、他のコンテナによりバックアップされているセションをSession Registry Serverから取得し、リクエストを処理します。
Session Registry Serverは、Servletコンテナからセションの取得要求があった場合、単にバックアップされたセションを返すのではなく、以下の図のように可能な限り最新のセションを返して、もとのServletコンテナのセションを破棄します。



1. WebサーバコネクタはServletコンテナAにリクエストを送ろうとしましたが、ServletコンテナAに障害が発生し、リクエストの送信に失敗しました。
2. WebサーバコネクタはServletコンテナBにリクエストを振り分けます。
3. ServletコンテナBは送られてきたセションは自分が所有するセションではないため、Session Registry Serverにバックアップされたセションの取得要求を行います。
4. Session Registry Serverはセションの元々の所有者であるServletコンテナAに最新のセションを要求します。
5. ServletコンテナAはセションをSession Registry Serverに受け渡します。
   Servletコンテナ A上のセションは受け渡しにより破棄されます。
   この時、HttpSessionListener.sessionDestroyed、HttpSessionBindingListener.valueUnBoundは呼び出されません。

6. Session Registry Serverはこの受け取った最新のセションをServletコンテナBにリカバリします。

ServletコンテナAとの通信に異常がある場合など、最新のセションを取得できないときは、Session Registry ServerはすでにバックアップされているセションをコンテナBにリカバリします。
また(4)において、該当の(セションの元々の所有者である)Servletコンテナとの通信ができない場合や、ひとつのServletコンテナに対する最新セションの要求処理が高多重となった場合、Session Registry Serverはそれ以上の最新セションの要求を行わずにSession Registry Serverにバックアップされているセションを返します。そのため、必ずしも最新のセションを取得(リカバリ)できるとは限りませんが、指定した契機でバックアップ済みのセションのリカバリは保証されます。
なお、(5)においてServletコンテナAでセションを使用中であった場合、使用終了するまでセションはバックアップされません。どれくらいの時間使用する可能性があるかはアプリケーション処理に依存するため、この最大待ち時間を指定することが可能です。
待ち時間を経過してもセションを使用中である場合、セションはServletコンテナBにリカバリされません。
このとき、getSession()、getSession(true)の場合は新規セション、getSession(false)の場合はnullがアプリケーションに返却されます。

HttpSessionActivationListener

セションに結びつけられているオブジェクトは、セションの非活性化や活性化といったコンテナのイベントからの通知を受けることができます。VM間でセションを移動させたりセションを持続させたりするコンテナは、セションに結びつけられている属性のうちHttpSessionActivationListener を実装しているすべての属性に通知することを要求されます。

イベント	メソッド	
	sessionWillPassivate	sessionDidActivate
リカバリ時	元のServletコンテナで実行	リカバリ先のServletコンテナで実行
Servletコンテナ停止時	セションをバックアップする前に実行	ー
Servletコンテナ起動時	ー	セションがリカバリされた後に実行

セションをリカバリできない場合

セションリカバリ機能を使用していても、以下の場合、セションがSession Registry Serverに存在しないため、セションのリカバリはできません。

- [Session Registry Serverで保持するセションの上限数]を超えてバックアップが行われ、バックアップから破棄された(古い)セションについてのリカバリが発生した場合。
- セションのバックアップの契機を[一定間隔]としている場合に、バックアップされる前にServletコンテナがダウンしたとき。
- IJServerおよびセションの永続化機能を無効としている場合に、Session Registry Serverを再起動したとき。
- セションの永続化機能を有効としている場合でも、指定した間隔で永続化が行われる前にSession Registry Serverがダウンし、かつServletコンテナを再起動したとき。

リカバリが保証される範囲については、“9.2 セションリカバリ機能の補償範囲”も参照してください。

9.1.3 URIでのセションリカバリ機能の有効・無効

Webブラウザからのリクエストについて、拡張子を指定することにより、静的コンテンツなどに対してセションリカバリ機能を無効にすることが可能です。

設定方法

セションリカバリ機能を無効とする拡張子は、Session Registry Client側で、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [環境設定] > [セションリカバリ設定] > [セションを使用しないURLの末尾]

Session Registry Clientの設定については、“9.4 Session Registry Clientの設定”を参照してください。また、Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドを使用して、上記の設定を行うこともできます。isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

使用場面

この機能を使用する場面について説明します。

Webブラウザから、明にセションを使用する動的コンテンツ(ServletやJSP)以外、静的コンテンツ(htmlや、htmlに埋め込んでいる画像ファイル等全般を含む)へのリクエストがあった場合でも、静的コンテンツもWebアプリケーションの一部として扱われるため、セションが生成されている場合はセションのlastAccessedTime(セションに関連付けられた要求をクライアントが最後に送信した時刻)が更新されます。(Cookieを使用する設定の場合。)
そのため、静的コンテンツに対するリクエストであってもセションのバックアップが発生します。また、サーバダウンなどが発生した場合には、静的コンテンツに対してもセションのリカバリが行われます。

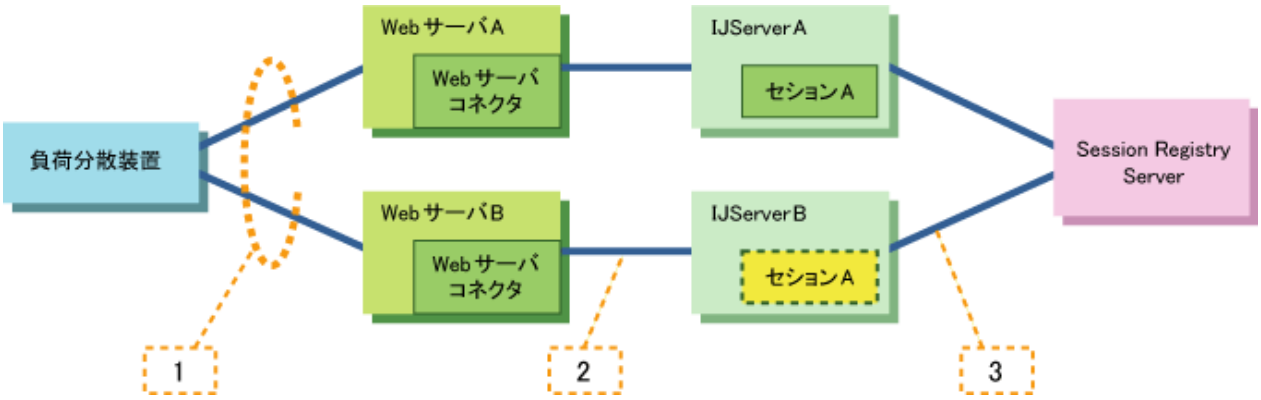
しかし、アプリケーションにとって静的コンテンツへのアクセスによるlastAccessedTimeの更新が特に重要ではない場合、性能への影響の防止および不要なリカバリ処理を発生させないため、拡張子を指定することによりセションのバックアップやリカバリを無効とすることが可能です。

注意

- この定義を行ったリソースへアクセスした場合、Session Registry Serverへのバックアップは行われません。
  したがって、この定義を行ったリソースのみにアクセスしている場合は、Session Registry Server上ではセションがタイムアウトし、バックアップから削除される可能性があります。
- フォームベース認証使用時には、認証情報はServletのセションに格納されます。
  そのため、以下の場合では、セションは新規となり、再度認証が必要となることがあります。
  - セションリカバリ機能有効、かつ
  - Session Registry Client(Servletコンテナ)の[セションリカバリ設定]において、[セションを使用しないURLの末尾]として拡張子を指定している、かつ
  - クライアントから、指定した拡張子のコンテンツに対するリクエストを行った、かつ
  - IJServerの再起動等の操作を行った、またはIJServerの停止、異常終了等で、別のServletコンテナにリクエストが振り分けられた、かつ
  - 該当コンテンツが認証の対象になっている。

Webアプリケーションの独自ヘッダ(独自のURLへの埋め込み文字列)で負荷分散する場合の注意

次の図のような構成で、Webアプリケーションの独自ヘッダ(または独自のURLへの埋め込み文字列)によって負荷分散を行う場合、静的コンテンツへのリクエストには、これら独自の値が付加されないため、負荷分散装置はServletのセションを生成したIJServer以外のIJServerにリクエストを振り分けることがあります。この場合、IJServerは正常に動作しているにもかかわらずセションが他のIJServerにリカバリされることとなります(Cookieによるセション管理時)。
そのため、セションのリカバリが頻繁に動作し、レスポンスの低下を引きおこす原因となります。



1. アプリケーションの独自ヘッダで振り分けを行っている場合、静的コンテンツへのリクエストはセションAがついたリクエストであっても、WebサーバBにリクエストが振り分けられる可能性があります。
2. リクエストのセションAはIJServerBのものではないため、Webサーバコネクタは振り分け先がありません。その場合任意のIJServer(この場合IJServerB)に振られます。
   (システム構成によっては内部定義が同じでそのままIJServerBに振られる場合もあります)
3. セションAはIJServerBには存在しないため、Session Registry Serverからセション Aを取得します。ServletコンテナAのセションAは破棄されます。

上記のような負荷分散の場合、該当のコンテンツについては、セションリカバリ機能を無効とする必要があります。

9.1.4 セションリカバリ機能の監視

セションリカバリ機能を使用する場合は、IJServer、Session Registry Serverは相互に使用可能かどうかを監視します。
障害を検出した場合は、そのIJServer、および、Session Registry Serverは使用不可にマークされ使用可能になるまで、処理対象からはずされます。

IJServerはSession Registry Serverのポートに対して一定間隔(10秒)で監視用のメッセージを送付し、正常にレスポンスが返ることを確認します。
応答待ち時間(20秒)内にレスポンスが返らない場合、Session Registry Serverは使用不可にマークされセションのバックアップ対象からはずされます。その後、使用不可にマークしたSession Registry Serverから正常にレスポンスが返るようになった場合、そのSession Registry Serverは使用可能とし、セションのバックアップが行われます。

Session Registry ServerはIJServerから一定間隔(30秒)以内ごとに監視用のメッセージが送付されてくることを確認します。
IJServerから監視用メッセージが送付されない場合は、そのIJServerは使用不可にマークされセションリカバリ時の処理対象からはずされます。

9.1.5 Webサーバコネクタの故障監視

Webサーバコネクタはクライアントからのリクエストを Servletコンテナに振り分ける時に初めてServletコンテナの異常を検出します。(Webサーバコネクタが異常を検出するには数秒から数分かかる場合があります。)
また、Webサーバコネクタは Servletコンテナが復旧したことを検出することはできませんので、異常の検出から1分後に再度Servletコンテナへリクエストの振り分けを行います。この時、Servletコンテナが復旧していない場合は再度接続に失敗します。

Webサーバコネクタの故障監視を使用することで上記問題を回避することができます。
Webサーバコネクタの故障監視はIJServerとWebサーバをそれぞれ別のサーバマシンに分離して負荷分散運用を行う場合に、分散先のIJServerの稼動状況を監視し、故障したIJServerを自動的に振り分けの対象から除外することや、故障から復旧したIJServerを自動的に振り分けの対象に戻すことができます。

セションリカバリ機能を使用する際はWebサーバコネクタの故障監視とあわせて使用することをお勧めします。
Webサーバコネクタの故障監視については、“3.7.2 Webサーバコネクタの故障監視”を参照してください。

9.1.6 Session Registry Serverで保持するセションの上限数

メモリ不足等のリソース不足によるシステムダウンを回避するためにSession Registry Serverで保持するセション数を制限することができます。
Session Registry Serverで保持するセション数は、Session Registry Server環境定義ファイルの定義“backup.limit”で指定します。詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

Session Registry Serverで保持しているセションの数が指定された値を超えた場合は、最も長い間使用されていないセションのバックアップを破棄し、代わりにバックアップ要求のあったセションをSession Registry Serverに保持します。
バックアップが破棄されたセションは、その後バックアップ要求があるまでSession Registry Serverにバックアップが存在しませんので、そのセションに対するリカバリ要求があった場合(Servletコンテナダウン時など)は、そのセションはリカバリされません。
バックアップが破棄された場合でもセション自体が破棄されたわけではないため、セションのリカバリが必要な障害が発生しない限りはそのセションについての業務は継続可能です。このとき、このセションは使用されることにより再度バックアップされ、代わりに別の長時間未使用なセションがバックアップから破棄されます。

9.1.7 セションの永続化

Session Registry Serverにバックアップされたセション情報を永続化することができます。永続化の有効・無効は、Session Registry Server環境定義ファイルの定義“session.store”で指定します。詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

セションの永続化を有効にした場合、Session Registry Serverにバックアップされたセションは永続化ファイルに書き出されます。
Session Registry Serverを再起動した場合は、永続化ファイルからセションを読み込み、セションを復元します。
また、セションの永続化を有効にした場合はSession Registry Serverを停止する際にもセションを永続化ファイルに書き出します。

セションを永続化するタイミング

セションを永続化するタイミングを指定することができます。
Session Registry Serverは指定された間隔ごとにセションを永続化します。

永続化ファイルの保管先

セションの永続化ファイルの保管先はSession Registry Serverごとに任意のディレクトリを指定することができます。
相対パスで指定した場合は、Session Registry Serverを配備したディレクトリからの相対となります。
指定したディレクトリ配下に“sessionrecovery”というディレクトリが作成され、このディレクトリ配下に永続化ファイルが出力されます。すでに該当ディレクトリが存在する場合は、そのディレクトリをそのまま使用します。

永続化ファイルの読み込み

Session Registry Serverは、セションの永続化が有効である場合、起動後IJServerから通信があった時点でこれら永続化ファイルを読み込みます。

注意

- Session Registry Serverをクラスタサービス機能で利用する場合はセションの永続化を有効にし、セションの永続化ファイルの保管先として各ノードから利用できる共有ディスクのパスを指定してください。
- セションの永続化が行われる前にSession Registry Serverを再起動した場合は、最後にバックアップされたセションの永続化ファイルからセションが復元されます。
- 永続化に必要な時間は、以下に依存します。
  また、前回の永続化の完了から指定された間隔の経過後、次の永続化が行われます。
  - セションの属性に格納したオブジェクトのサイズ
  - 生成/更新したセションの数
  - ファイルシステムの性能
- 永続化機能使用時、以下の場合は、あらかじめ永続化セション消去コマンドにより永続化ファイルを削除してください。
  - アプリケーションの運用を終了(アプリケーションを配備解除、ワークユニットを削除)する場合

- ー 永続化ファイルの保管先を変更する場合
- ー Session Registry Serverを削除する場合
  ただし、永続化ファイルの保管先ディレクトリがSession Registry Serverを配備したディレクトリ配下(デフォルト値含む)の場合は、自動的に削除されます。
- ー Interstage Application Serverをアンインストールする場合

- Windows32/64
  永続化ファイルの保管先にネットワークドライブやUNCパスは指定できません。

9.1.8 セションの全消去

業務の初期化やメモリ等の資源を解放するために業務システム上のセションをすべて消去したい場合、IJServerおよびSession Registry Serverをいったんすべて停止し、再起動してください。
セションリカバリの機能上、個々に停止/再起動を行っただけでは、再バックアップや、バックアップされているセションのリカバリが発生するため、すべてを消去することができません。
また、セションの永続化の機能を有効としている場合は、再起動前に永続化セション消去コマンドを実行してください。
コマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“jssrsadmin”を参照してください。

9.1.9 セションリカバリ機能のログ

セションリカバリ機能が出力するログは以下の2ヵ所に出力されます。

- IJServerのコンテナログに出力されるログ
- Session Registry Serverのログ

設定方法

それぞれのログの出力先は、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [環境設定] > [ワークユニット設定] > [ログ出力ディレクトリ]

また、ログファイルのロールオーバや、ログを保管する世代数は、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [ログ定義]

isj2eeadminコマンドを使用して、上記の設定を行うこともできます。isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

IJServerのコンテナログに出力されるログ

IJServerのコンテナログに出力されるログはエラーメッセージ、警告メッセージ、情報、アクセスログが出力されます。

アクセスログについては設定により出力するか出力しないかを選択することができます。
アクセスログの出力については、Session Registry Client側で、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [環境設定] > [セションリカバリ設定] > [アクセスログを出力する]

Session Registry Clientの設定については、“9.4 Session Registry Clientの設定”を参照してください。また、Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。

Session Registry Serverのログ

Session Registry Serverのログはエラーメッセージ、警告メッセージ、情報、アクセスログが出力されます。
アクセスログについては設定により出力するか出力しないかを選択することができます。
Session Registry Serverのアクセスログについては、Session Registry Server環境定義ファイルの定義“access_log”で指定します。詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

9.1.10 Session Registry Serverが保持する期限切れ(タイムアウト)セションの破棄

通常、Session Registry ServerはIJServerからセションが無効になったことを通知された時に保持しているセションを破棄します。
しかし、IJServerがダウンしている場合はセションが無効になったことがIJServerから通知されないため、不要なセションがSession Registry Serverに残り、Session Registry Serverのリソースを圧迫します。
そのため、Session Registry Serverは保持しているセションを一定間隔で監視し、期限切れ(タイムアウト)となっているセションを破棄することでリソースの圧迫を防ぎます。この監視間隔はデフォルトでは**60**秒です。

監視間隔は、Session Registry Server環境定義ファイルの定義“clean.interval”で指定します。詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

この監視機能では、期限切れセションを単に削除しているだけであるため、IJServerがダウンしてSession Registry Serverにのみセションが存在する場合には、以下のセション破棄関連処理が実行されません。

- javax.servlet.http.HttpSessionListener#sessionDestroyed(javax.servlet.http.HttpSessionEvent)
- javax.servlet.http.HttpSessionAttributeListener#attributeRemoved(javax.servlet.http.HttpSessionBindingEvent)
- javax.servlet.http.HttpSessionBindingListener#valueUnbound(javax.servlet.http.HttpSessionBindingEvent)

上記の処理を実行する必要のある場合は、Session Registry Server環境定義ファイルに以下を指定することにより、Session Registry Serverのみに存在する期限切れセションを、生存しているIJServerに送信し、IJServerでセション破棄関連処理を実行させることができます。

- invalid.session.send
- invalid.session.waiting.time

詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

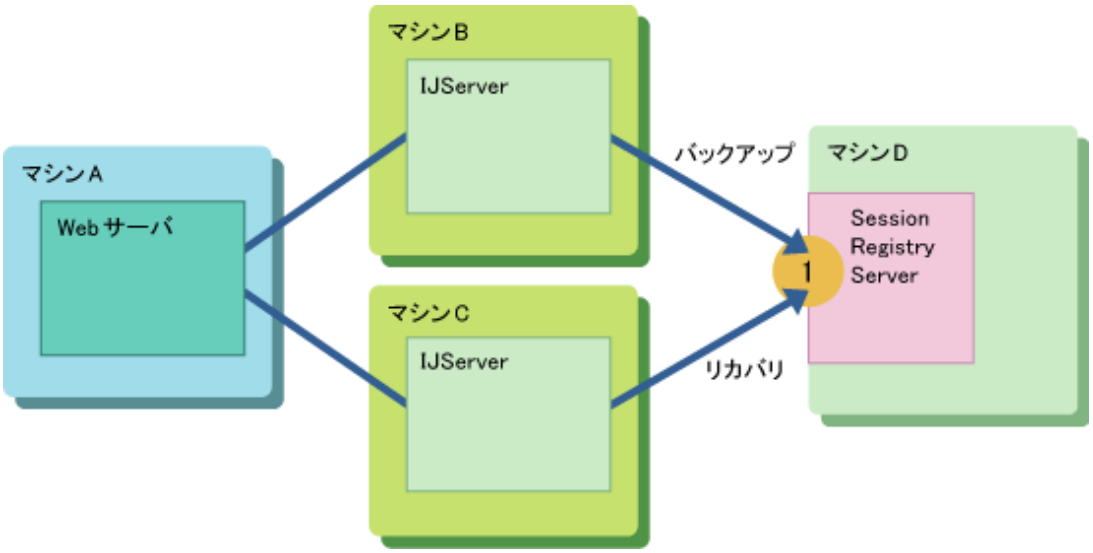
9.1.11 セションIDについて

以下のServletAPIにより取得されるセションIDは、ServletコンテナがWebブラウザとの間でセション管理を行う際に付加するCookieの値や、URLエンコードでURLに設定される値とは異なります。Webアプリケーションでセション IDを参照する場合は、CookieやURLにエンコードされた値ではなく、下記ServletAPIにより取得できる値を使用してください。

- javax.servlet.http.HttpSession#getId()
- javax.servlet.http.HttpServletRequest#getRequestedSessionId()

9.1.12 Session Registry Serverへのアクセスの制限

Session Registry Serverには、Servletコンテナからのバックアップやリカバリ要求の通信を受け付けるためのIPアドレス(図中1・マシンDのIPアドレス)とポート番号を指定します。このとき、通信を許可する相手(Servletコンテナ)のIPアドレス(図中マシンB、CのIPアドレス)を指定することにより、指定以外のIPアドレスからの不正な通信を受け付けないようIPアドレスによるアクセス制限を行うことが可能です。



設定方法

通信を許可するIPアドレスは、Session Registry Server側で、Interstage管理コンソールの以下の項目で設定します。

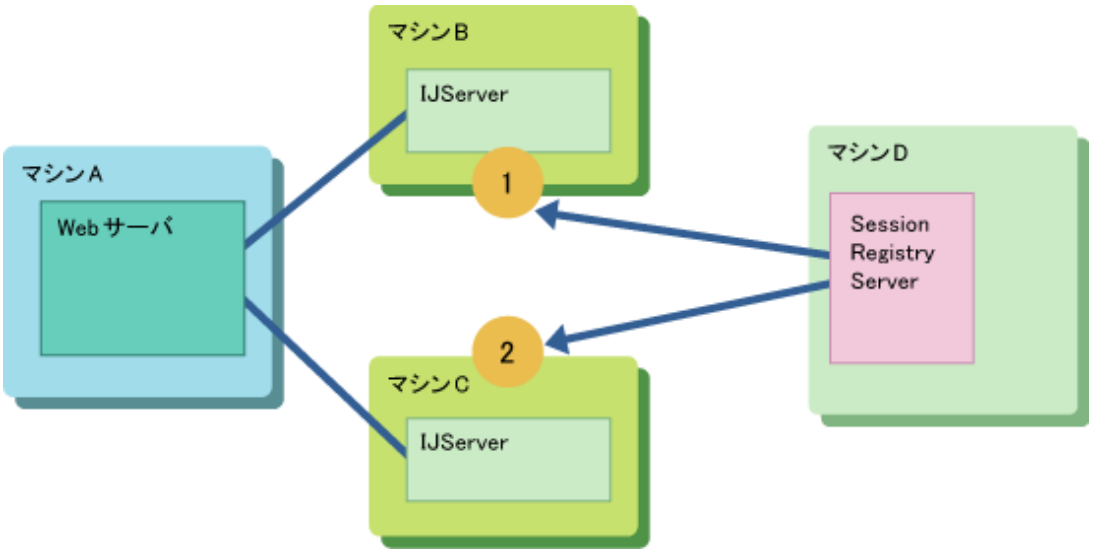
- [ワークユニット] > “ワークユニット名” > [環境設定] > [Webサーバコネクタ(コネクタ)設定] > [WebサーバのIPアドレス]

Session Registry Serverの設定については、“9.3 Session Registry Serverの設定”を参照してください。また、Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。

isj2eeadminコマンドを使用して、上記の設定を行うこともできます。isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

9.1.13 Servletコンテナの制御用ポートの指定

セションリカバリ機能を使用する場合はServletコンテナに制御用のポート(図中1、2)を設定する必要があります。
制御用のポートはSession Registry Serverからの要求を処理するために使用します。
また、制御用のポートは通信を許可する相手(本機能の場合はSession Registry Server)のIPアドレス(図中のマシンDのアドレス)を指定することにより、指定以外のIPアドレスからの不正な通信を受け付けないようIPアドレスによるアクセス制限を行うことが可能です。



設定方法

制御用のポートやアクセスを許可するIPアドレスは、Session Registry Client側で、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [環境設定] > [Servletコンテナ設定] > [制御用ポート]
- [ワークユニット] > “ワークユニット名” > [環境設定] > [Servletコンテナ設定] > [アクセス許可IPアドレス]

Session Registry Clientの設定については、“9.4 Session Registry Clientの設定”を参照してください。また、Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。

isj2eeadminコマンドを使用して、上記の設定を行うこともできます。isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

9.2 セションリカバリ機能の補償範囲

異常発生時のセション情報の保証範囲は、以下のとおりです。

Servletコンテナ異常終了時のセションのリカバリ保証範囲

Servletコンテナが異常終了した場合であっても、Session Registry Serverにセションがバックアップされていれば、運用中のServletコンテナでそのセションをリカバリすることができます。
したがって、リカバリの保証範囲として、Session Registry Serverにセションがバックアップされたかどうかが重要なポイントとなります。
バックアップの契機は、Session Registry Client側で、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [環境設定] > [セションリカバリ設定] > [バックアップの契機]

Session Registry Clientの設定については、“9.4 Session Registry Clientの設定”を参照してください。また、Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドを使用して、上記の設定を行うこともできます。isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

例

以下に例を示します。

- [リクエスト終了時]指定の場合
  レスポンスをクライアントに返信するごとにバックアップを行います。
  このため、クライアントに返信した直後に異常終了した場合でもリカバリすることができます。
- [一定間隔 5 秒ごと]指定の場合
  5秒ごとにセション情報をバックアップします。IJServerが異常終了した場合、最後にバックアップされたセションの情報をリカバリすることができます。

Session Registry Server異常終了時のセションのリカバリ保証範囲

Session Registry Serverが異常終了した場合であっても、セションに格納されたデータがファイルに永続化されていれば、再度Session Registry Serverを起動したときに永続化されたファイルからそのセションをリカバリすることができます。
したがって、リカバリの保証範囲として、Session Registry Serverにおいてセションが永続化されたかどうかが重要なポイントとなります。
永続化の間隔は、Session Registry Server環境定義ファイルの定義“serialize.interval”で指定します。詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

例

以下に例を示します。

- serialize.intervalに60を設定した場合
  60秒ごとにセションの情報をファイルに書き出し(永続化)します。
  永続化されたタイミングによりますが、異常終了時から60秒前までのセションは保証されません。
  異常終了時から60秒前までにSession Registry Serverにバックアップされたセションは保証されます。

永続化について

永続化を行うことにより、信頼性は向上しますが、レスポンスが低下します。
永続化の有効・無効は、Session Registry Server環境定義ファイルの定義“session.store”で指定します。詳細は“Session Registry Server環境定義ファイルの設定”を参照してください。

以下にその動作の違いを説明します。

- session.storeに“on”を設定した場合
  Session Registry Serverは、セションの情報を定期的に永続化します。
  予期しない障害によりSession Registry Serverがダウンした場合、永続化された情報からダウン前の最後に永続化された時点でのセションの状態を復元することが可能です。
  Session Registry Serverは、Session Registry Server環境定義ファイルの“serialize.file.path”のディレクトリに、“serialize.interval”の間隔で、セションの情報を永続化します。

- session.storeに“off”を設定した場合
  Session Registry Serverは、セションの情報を永続化しません。
  したがって、予期しない障害によりSession Registry Serverがダウンした場合、セションのデータを復元することはできません。

9.3 Session Registry Serverの設定

セションリカバリを使用するためには、以下の設定をします。

- Session Registry Serverの設定
- Session Registry Clientの設定

ここでは、Session Registry Serverの設定について説明します。
Session Registry Clientの設定については、“9.4 Session Registry Clientの設定”を参照してください。

概要

セションリカバリを使用するためにはSession Registry Serverを用意する必要があります。
IJServerではセションをバックアップするSession Registry Serverを指定し、セションリカバリ機能を使用します。

注意

- Webアプリケーションの動作するIJServerとSession Registry Serverを別マシンで運用する場合、Session Registry Serverを運用する環境では、“Webサーバとワークユニットを同一のマシンで運用しない”設定とする必要があります。

設定方法

Session Registry Serverを作成します。
Session Registry ServerはIJServer上で動作します。Session Registry Server用のIJServerを作成し、Session Registry Serverを配備します。

注意

ユーザアプリケーションを運用するIJServerと、Session Registry Serverを運用するIJServerは別に用意します。

Session Registry Serverは、以下の手順で作成します。

1. システムの環境設定
2. Session Registry Server用のワークユニットの作成
3. Session Registry Serverの配備
4. Session Registry Server環境定義ファイルの設定

1. システムの環境設定

Session Registry ServerをWebアプリケーションの動作するIJServerと別マシンで運用する場合、Interstage管理コンソールのシステムの環境設定で以下を設定する必要があります。

- [システム] > [環境設定] > [詳細設定] > [Servletサービス詳細設定] > [Webサーバとワークユニットを同一のマシンで運用する]で“運用しない”を設定

2. Session Registry Server用のワークユニットの作成

Session Registry Server用のワークユニットを、以下のどちらかの方法で作成します。

- Interstage管理コンソールを使用して作成する
  詳細は、“9.3.1 Session Registry Server用のワークユニットの作成(Interstage管理コンソールを使用)”を参照してください。
- isj2eeadminコマンドを使用して作成する
  詳細は、“9.3.2 Session Registry Server用のワークユニットの作成(isj2eeadminコマンドを使用)”を参照してください。

3. Session Registry Serverの配備

Session Registry Serverを、以下のどちらかの方法で配備します。

- Interstage管理コンソールを使用して配備する
  詳細は、“9.3.3 Session Registry Serverの配備(Interstage管理コンソールを使用)”を参照してください。
- ijsdeploymentコマンドを使用して配備する
  詳細は、“9.3.4 Session Registry Serverの配備(ijsdeploymentコマンドを使用)”を参照してください。

注意

Session Registry Server用のIJServerには、他のWebアプリケーションは配備しないでください。

4. Session Registry Server環境定義ファイルの設定

Session Registry Server環境定義ファイルの設定を行います。
Session Registry Server環境定義ファイルは以下に保管されています。

Windows32/64

[IJServerディレクトリ]¥apps¥srs.ear¥srs.war¥WEB-INF¥web.xml

Solaris64 Linux32/64

[IJServerディレクトリ]/apps/srs.ear/srs.war/WEB-INF/web.xml

設定内容については、“9.3.5 Session Registry Server環境定義ファイルの設定内容”を参照してください。

9.3.1 Session Registry Server用のワークユニットの作成(Interstage管理コンソールを使用)

Session Registry Server用のワークユニットを、Interstage管理コンソールを使用して作成する方法を説明します。
Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。

1. Interstage管理コンソールで、[ワークユニット] > [新規作成]画面を表示します。
   以下で説明していない項目については、設定する必要はありません。

2. [簡易設定]の以下の項目を設定します。

項目名	設定内容
ワークユニット名	Session Registry Serverを識別する名前を指定します。
ワークユニットタイプ	“IJServer”を選択します。

3. [詳細設定] > [IJServer設定]の以下の項目を設定します。

項目名	設定内容
IJServerタイプ	“Webアプリケーションのみ運用”を選択します。

4. [詳細設定] > [ワークユニット設定]の以下の項目を設定します。

項目名	設定内容
JavaVMオプション	Session Registry Serverのヒープサイズを指定します。設定値については、“チューニングガイド”の“メモリ容量”を参照してください。 デフォルト値:**-Xms16m -Xmx256m**
ワークユニット自動起動	Interstage起動時にSession Registry Serverを起動するかどうかを指定します。 Solaris64 Linux32/64 “自動起動する”を選択した場合に、起動ユーザ名を指定します。指定したユーザ名の権限で、Session Registry Serverを起動します。起動ユーザの変更については、“9.6.2 Session Registry Serverの起動ユーザの変更について ”を参照してください。
アプリケーション最大処理時間	Session Registry Serverの最大処理時間の監視時間を指定します。 “9.5.1 各タイムアウト値の設定について”を参照してください。
アプリケーション最大処理時間超過時の制御	Session Registry Serverの最大処理時間を経過しても処理が終了しない場合の動作を指定します。
ワークユニット起動待ち時間	Session Registry Serverの起動が完了するまでの監視時間を指定します。通常、デフォルト値で問題ありません。
リトライカウント	Session Registry Serverの異常終了時の再起動回数を指定します。1、または小さな値を推奨します。 注) “9.6.5 Session Registry Serverの再起動について”を参照してください。
リトライカウントリセット時間	Session Registry Serverの異常終了回数をリセットする時間を指定します。 注) “9.6.5 Session Registry Serverの再起動について”を参照してください。
ログ出力ディレクトリ	Session Registry Serverのログ出力先のディレクトリを指定します。

5. [詳細設定] > [Webサーバコネクタ(コネクタ)設定]の以下の項目を設定します。

項目名	設定内容
WebサーバのIPアドレス	アクセス許可IPアドレス。このSession Registry Serverが対象とするServletコンテナのIPアドレスをxxx.xxx.xxx.xxxの形式で指定します。改行で区切ることで複数指定可能です。 “9.1.12 Session Registry Serverへのアクセスの制限”を参照してください。 Session Registry ServerとIJServerを同一マシンに構築する場合(WebサーバとIJServerを同一マシンで運用する設定となっている場合)、本項目は表示されません。

6. [詳細設定] > [Servletコンテナ設定]の以下の項目を設定します。

項目名	設定内容
ServletコンテナのIPアドレス	Session Registry ServerのIPアドレスをxxx.xxx.xxx.xxxの形式で指定します。 Session Registry ServerとIJServerを同一マシンに構築する場合(WebサーバとIJServerを同一マシンで運用する設定となっている場合)、本項目は表示されません。
タイムアウト	このSession Registry Serverが対象とするServletコンテナから通信が途絶えた場合に通信を切断するまでの時間を指定します。 “9.5.1 各タイムアウト値の設定について”を参照してください。
ポート番号	このSession Registry Serverが対象とするServletコンテナとの接続に使用するポート番号を指定します。
同時処理数 最大値	Session Registry Serverの同時処理最大数を指定します。 “9.5.2 多重度(同時処理数)の設定について”を参照してください。

7. [作成]を実行します。

9.3.2 Session Registry Server用のワークユニットの作成(isj2eeadminコマンドを使用)

Session Registry Server用のワークユニットを、isj2eeadminコマンドを使用して作成する方法を説明します。
isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

1. IJServer定義ファイルのサンプルを任意のディレクトリにコピーします。

   サンプルの保管場所:

   Windows32/64

   C:¥Interstage¥F3FMjssrs¥sample¥srs

   Solaris64 Linux32/64

   /opt/FJSVjssrs/sample/srs/

ファイル名	文字コード
sample_utf-8.xml	UTF-8
sample_euc-jp.xml	EUC-JP
sample_shift_jis.xml	Shift_jis

注意

IJServer定義ファイルのサンプルは、IJServerの定義内容はすべて同じですが、記述された文字コードごとにファイルが分かれています。適切な文字コードでファイルの編集を行ってください。

例

Shift_jisのサンプルファイルを“srs.xml”としてコピーします。

Windows32/64

```
copy C:¥Interstage¥F3FMjssrs¥sample¥srs¥sample_shift_jis.xml srs.xml
```

Solaris64 Linux32/64

```
cp /opt/FJSVjssrs/sample/srs/sample_shift_jis.xml srs.xml
```

2. コピーしたIJServer定義ファイルを運用するSession Registry Serverの環境に合わせて編集します。詳細は、“IJServer定義ファイル”を参照してください。

3. isj2eeadminコマンドを使用してIJServerを作成します。

例

isj2eeadmin ijserver -a -f srs.xml

IJServer定義ファイル

IJServer定義ファイルについて説明します。

記述形式

IJServer定義ファイルは、以下の形式で記載されています。
注) 太字箇所は、必ず、環境に合わせて変更してください。

```
<?xml version="1.0" encoding="Shift_JIS" standalone="yes" ?>
<Isj2eeIjserverDefinition>
  <IJServer>
    <Name>[ijserver name]</Name>
    <Type>WEB</Type>
    <AutomaticStart>
      <Mode>YES</Mode>
      <User>root</User>
    </AutomaticStart>
    <ApplicationRetry>
      <AbnormalTerminationCounts>1</AbnormalTerminationCounts>
      <RetryCountResetTime>600</RetryCountResetTime>
    </ApplicationRetry>
    <CurrentDirectory>
      <NumberOfRevisionDirectories>1</NumberOfRevisionDirectories>
    </CurrentDirectory>
    <Common>
      <JavaCommandOptions>-Xms16m -Xmx256m</JavaCommandOptions>
      <ProcessingTime>
        <MaximumProcessingTime>400</MaximumProcessingTime>
        <TerminateProcessModeForTimeout>NO</TerminateProcessModeForTimeout>
      </ProcessingTime>
    </Common>
    <Web>
      <IPAddress/>
      <Timeout>60</Timeout>
      <Ports>
        <Number>[port number]</Number>
      </Ports>
      <ThreadConcurrency>
        <MaxThreads>64</MaxThreads>
      </ThreadConcurrency>
      <Www>
        <AcceptedHosts>
          <Address/>
        </AcceptedHosts>
      </Www>
    </Web>
    <Log>
      <Directory/>
      <Mode>SIZE</Mode>
      <Size>1</Size>
      <StartTime/>
      <Interval/>
      <HistorySize>1</HistorySize>
```

```
    </Log>
  </IJServer>
</Isj2eeIjserverDefinition>
```

定義内容

各タグの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

Name :ワークユニットに関する設定

Session Registry Serverを識別する名前を指定します。
必ず、環境にあわせて変更してください。

Type :ワークユニットに関する設定

“WEB”のまま、変更しないでください。

Mode :ワークユニットに関する設定

Interstage起動時にSession Registry Serverを起動するかどうかを指定します。
必要に応じて、変更してください。

User :ワークユニットに関する設定

Solaris64 Linux32/64
<Mode>タグにYESを指定した場合に、起動ユーザ名を指定します。指定したユーザ名の権限で、Session Registry Serverを起動します。
起動ユーザの変更については、“9.6.2 Session Registry Serverの起動ユーザの変更について ”を参照してください。
必要に応じて、変更してください。

AbnormalTerminationCounts :ワークユニットに関する設定

Session Registry Serverの異常終了時の再起動回数を指定します。1、または小さな値を推奨します。
“9.6.5 Session Registry Serverの再起動について”を参照してください。
必要に応じて、変更してください。

RetryCountResetTime :ワークユニットに関する設定

Session Registry Serverの異常終了回数をリセットする時間を指定します。
“9.6.5 Session Registry Serverの再起動について”を参照してください。
必要に応じて、変更してください。

NumberOfRevisionDirectories :ワークユニットに関する設定

カレントディレクトリをバックアップする世代数を指定します。
必要に応じて、変更してください。

JavaCommandOptions :ワークユニットに関する設定

Session Registry Serverのヒープサイズを指定します。
必要に応じて、変更してください。

MaximumProcessingTime :ワークユニットに関する設定

Session Registry Serverの最大処理時間の監視時間を指定します。
“9.5.1 各タイムアウト値の設定について”参照してください。
必要に応じて、変更してください。

TerminateProcessModeForTimeout :ワークユニットに関する設定

Session Registry Serverの最大処理時間を経過しても処理が終了しない場合の動作を指定します。
必要に応じて、変更してください。

IPAddress :Servletコンテナに関する設定

Session Registry ServerのIPアドレスをxxx.xxx.xxx.xxxの形式で指定します。
この項目はWebサーバとワークユニットを同一マシンで運用しない場合のみ有効です。
必要に応じて、変更してください。

Timeout :Servletコンテナに関する設定

このSession Registry Serverが対象とするServletコンテナから通信が途絶えた場合に通信を切断するまでの時間を指定します。
“9.5.1 各タイムアウト値の設定について”を参照してください。
必要に応じて、変更してください。

Number :Servletコンテナに関する設定

このSession Registry Serverが対象とするServletコンテナとの接続に使用するポート番号を指定します。
必ず、環境にあわせて変更してください。

MaxThreads :Servletコンテナに関する設定

Session Registry Serverの同時処理最大数を指定します。
“9.5.2 多重度(同時処理数)の設定について”を参照してください。
必要に応じて、変更してください。

Address :Webサーバコネクタに関する設定

アクセス許可IPアドレス。このSession Registry Serverが対象とするServletコンテナのIPアドレスをxxx.xxx.xxx.xxxの形式で指定します。複数指定する場合は1つずつタグを分けて指定してください。
“9.1.12 Session Registry Serverへのアクセスの制限”を参照してください。
この項目はWebサーバとワークユニットを同一マシンで運用しない場合のみ有効です。
必要に応じて、変更してください。

Directory :Logに関する設定

Session Registry Serverのログ出力先のディレクトリを指定します。
必要に応じて、変更してください。

Mode :Logに関する設定

ロールオーバをサイズとログ収集時間のどちらで行うかを指定します。
必要に応じて、変更してください。

Size :Logに関する設定

ロールオーバするサイズを指定します。
必要に応じて、変更してください。

StartTime :Logに関する設定

ロールオーバを開始する時刻を指定します。
必要に応じて、変更してください。

Interval :Logに関する設定

ロールオーバする間隔を指定します。
必要に応じて、変更してください。

HistorySize :Logに関する設定

ロールオーバしたログファイルを保管する世代数を指定します。
必要に応じて、変更してください。

注意

- 上記の定義内容に記載していないタグは、Session Registry Server用のIJServer定義ファイルには定義しないでください。

9.3.3 Session Registry Serverの配備(Interstage管理コンソールを使用)

Session Registry Serverを、Interstage管理コンソールを使用して配備する方法を説明します。
Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。

1. Interstage管理コンソールで、[ワークユニット] > “Session Registry Server名” > [配備]画面を表示します。

2. 以下を設定します。以下の項目以外は、設定しないでください。

[配備]画面の項目名	設定内容
[配備ファイル]	“サーバ上に格納されているファイルを配備する”を選択し、以下のファイルを指定します。 Windows32/64 C:¥Interstage¥F3FMjssrs¥ear¥srs.ear Solaris64 Linux32/64 /opt/FJSVjssrs/ear/srs.ear
[起動指定]	“配備完了後、ワークユニットを起動する”は**チェックしません**。
[詳細設定] > [Webアプリケーション設定] > [Webアプリケーション名]	以下から変更しないでください。 ・ ROOT

3. [配備]を実行します。

9.3.4 Session Registry Serverの配備(ijsdeploymentコマンドを使用)

Session Registry Serverを、ijsdeploymentコマンドを使用して配備する方法を説明します。
ijsdeploymentコマンドの詳細については、リファレンスマニュアル(コマンド編)の“ijsdeployment”を参照してください。

ijsdeploymentコマンドを使用してSession Registry Serverを配備します。

例

Session Registry Server名(ワークユニット名)を“SRSV01”とした場合

Windows32/64

```
ijsdeployment -n SRSV01 -f C:¥Interstage¥F3FMjssrs¥ear¥srs.ear
```

Solaris64 Linux32/64

```
ijsdeployment -n SRSV01 -f /opt/FJSVjssrs/ear/srs.ear
```

9.3.5 Session Registry Server環境定義ファイルの設定内容

Session Registry Server環境定義ファイルの設定内容を説明します。

記述形式

Session Registry Server環境定義ファイルは、以下の形式で記載されています。

```
<web-app>
  <context-param>
    <param-name>backup.limit</param-name>
    <param-value>0</param-value>
  </context-param>
  <context-param>
    <param-name>clean.interval</param-name>
    <param-value>60</param-value>
  </context-param>
  <context-param>
    <param-name>session.store</param-name>
    <param-value>off</param-value>
  </context-param>
  <context-param>
    <param-name>serialize.file.path</param-name>
```

```
    <param-value>serializedata</param-value>
  </context-param>
  <context-param>
    <param-name>serialize.interval</param-name>
    <param-value>60</param-value>
  </context-param>
  <context-param>
    <param-name>pop.timeout</param-name>
    <param-value>360</param-value>
  </context-param>
  <context-param>
    <param-name>access_log</param-name>
    <param-value>off</param-value>
  </context-param>
  <context-param>
    <param-name>invalid.session.send</param-name>
    <param-value>off</param-value>
  </context-param>
  <context-param>
    <param-name>invalid.session.waiting.time</param-name>
    <param-value>2000</param-value>
  </context-param>
</web-app>
```

定義内容

backup.limit：保持するセション数の制限

Session Registry Serverでセションを保持する上限数を、**0～2147483647の整数値**で指定します。デフォルト値は、**0**です。
0を指定した場合、上限はなくなります。

clean.interval：期限切れセションの監視間隔

タイムアウトしたセションの監視の間隔(秒)を、**30～2147483647の整数値**で指定します。デフォルト値は、**60**です。

session.store：セションの永続化

Session Registry Serverがセションの永続化(ファイルに保管)を行うかどうかを、以下から選択します。大文字・小文字は区別しません。
クラスタサービス機能を利用する場合は、必ず“on”を選択してください。

- on：有効
- **off**: 無効・・・デフォルト値

serialize.file.path：セションの永続化ファイルの保存先

session.storeでonを指定した場合は必須です。
セション情報の永続化ファイルの保存先ディレクトリを以下の範囲で指定します。

Windows32/64

1～90文字

Solaris64 Linux32/64

1～860文字

相対パスの場合、Session Registry Serverを配備したディレクトリからのパスに置き換えられます。相対パスを指定する場合は、絶対パスに置き換えたときに上限の長さ以内となるようにしてください。
本項目が初期値“serializedata”(相対パス)で、Interstageのインストールディレクトリ¥IJServerディレクトリがデフォルト、Session Registry Server名が“SRS001”の場合、以下のようになります。

Windows32/64

C:¥Interstage¥J2EE¥var¥deployment¥ijserver¥SRS001¥apps¥srs.ear¥srs.war¥serializedata
同じ条件で本項目を指定する場合、19文字まで指定可能です。

Solaris64 Linux32/64

/opt/FJSVj2ee/var/deployment/ijserver/SRS001/apps/srs.ear/srs.war/serializedata
同じ条件で本項目を指定する場合、798文字まで指定可能です。

本定義で指定したディレクトリ配下にサブディレクトリ“sessionrecovery”が生成され利用されます。
指定値については、以下に注意してください。

- 存在しないディレクトリの場合は起動時にエラーとなり、運用できません。
- クラスタサービス機能を利用する時は、各ノードから利用できる共有ディスクのパスを指定してください。
- Windows32/64
  ネットワークドライブやUNCパスは指定できません。
- Solaris64 Linux32/64
  Session Registry Serverをroot権限のあるユーザ以外で運用する場合、指定したディレクトリにそのユーザでの全権限がある必要があります。必要に応じてオーナや権限を変更してください。

serialize.interval：セションを永続化する間隔

セションを永続化する間隔(秒)を、**1～2147483647の整数値**で指定します。デフォルト値は、**60**です。

pop.timeout：最新セション取得待ち時間

最新セション取得待ち時間(秒)を、**1～86400の整数値**で指定します。デフォルト値は、**360**です。
セションのリカバリ時、もともとセションを保持していたコンテナから最新のセションの取得を行いますが、このときの最大待ち時間です。
セションがリクエスト処理中で使用されている場合の待ち時間です。待ち時間をオーバーしてもセション使用中の場合、該当のセションはリカバリされません。
タイムアウト値については、“9.5.1 各タイムアウト値の設定について”を参照してください。

access_log：アクセスログを出力する

アクセスログの出力の有効無効を、以下から選択します。大文字・小文字は区別しません。
アクセスログには、バックアップやリカバリ、生存監視の通信の詳細ログを出力します。

- on：出力する
- **off**: 出力しない・・・デフォルト値

invalid.session.send：期限切れセションをIJServerへ送信

Session Registry Serverに存在するタイムアウト時刻を経過したセションのうち、IJServerからセションの破棄が通知されないものを、生存しているSession Registry Clientに送信するかどうか設定します。この定義を有効にすることにより、Session Registry Serverのみに存在するセションに対して、セションの破棄関連処理を実行させることができます。

- on：有効
- **off**: 無効・・・デフォルト値

invalid.session.waiting.time：セションの無効通知の待ち時間

invalid.session.sendがonの場合、Session Registry Serverでタイムアウト時刻を経過したセションが、この定義で指定した待ち時間(秒)を経過してもSerssion Registry Clientからセションが無効になったことが通知されない場合に、生存しているSession Registry Clientに期限切れセションを送信します。待ち時間(秒)は**60～2147483647の整数値**で指定します。デフォルト値は、**2000**です。

注意

- Session Registry Server環境定義ファイルに2バイトコードを記載する場合(コメントやserialize.file.pathなど)、UTF-8で記載してください。
- Session Registry Server環境定義ファイルには、上記の定義内容に記載していないタグもありますが、それらは更新・削除しないでください。

9.4 Session Registry Clientの設定

セションリカバリを使用するためには、以下の設定をします。

- Session Registry Serverの設定
- Session Registry Clientの設定

ここでは、Session Registry Clientの設定について説明します。
Session Registry Serverの設定については、“9.3 Session Registry Serverの設定”を参照してください。

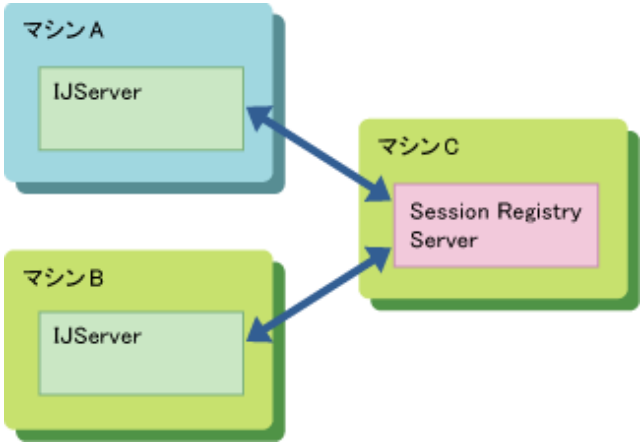
構成

Session Registry ServerとSession Registry Clientの構成例を以下に示します。

複数のマシンで1つのSession Registry Serverを使用する場合

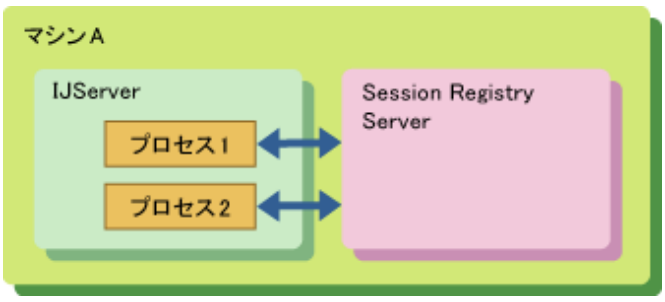
この構成の場合、IJServerとSession Registry Serverがそれぞれ別々のマシンで運用されていますので、IJServerが動作しているマシンA、マシンBのどちらかがダウンした場合でも継続してWebアプリケーションを運用することができます。ただし、Session Registry Serverが動作しているマシンCがダウンしている間はセションリカバリ機能が使用できません。(セションのバックアップ/リカバリはできませんが、通常の業務継続は可能です。)
そのため、この構成で運用する場合、以下のように費用面での欠点があります。

- Session Registry Server用に1台マシンを用意する必要があります
- Session Registry Serverには信頼性の高いマシンを使用する必要があります



IJServerを多重プロセスで運用する場合

この構成の場合、IJServerの1つのプロセスがダウンしている間は、ダウンしていないプロセスにセションがリカバリされるため、プロセスがダウンしている間もWebアプリケーションを継続して運用することができます。
この構成の場合、IJServerが1つのマシンで運用されているため、マシンがダウンした場合はWebアプリケーションを継続して運用することはできません。



設定方法

Session Registry Clientの設定では、Servletコンテナ側から使用するSession Registry Serverを指定します。
Session Registry Client(Servletコンテナ)の設定について説明します。

注意

- Session Registry Client(Servletコンテナ)の設定は、Session Registry Clientのパッケージがインストールされている環境でのみ設定可能です。
- IJServerのタイプがEJB Onlyの場合は、[セションリカバリ設定]、[Servletコンテナ設定]は表示されません。
- Windows32/64
  Session Registry ServerがWindowsの場合、Webアプリケーション名のアルファベットの大文字と小文字は区別されません。そのため、アルファベットの大文字と小文字だけが異なるアプリケーションのセションのバックアップ先には、それぞれ別のSession Registry Serverを指定してください。

Session Registry Clientとして使用する場合は、通常のIJServerの設定に加えて、以下の設定が必要です。
Interstage管理コンソールを使用して、以下を設定します。設定内容の詳細については、Interstage管理コンソールのヘルプを参照してください。

- [ワークユニット] > [新規作成] > [Servletコンテナ設定]
  または
  [ワークユニット] > “ワークユニット名” > [環境設定] > [Servletコンテナ設定]

  ー 制御用ポート
  セションリカバリ機能を使用する場合、必須です。

  ー アクセス許可IPアドレス
  Session Registry Serverを運用するマシンに複数のIPアドレスが設定されている場合は、すべてを記載してください。

- [ワークユニット] > [新規作成] > [セションリカバリ設定]
  または
  [ワークユニット] > “ワークユニット名” > [環境設定] > [セションリカバリ設定]

  ー セションリカバリ
  IJServerでSession Registry Server機能を使用するかしないかを選択します。

  ー セションのバックアップ先Session Registry Serverのアドレス:ポート
  セションのバックアップ先のSession Registry Serverを指定します。

  ー バックアップの契機
  セションをバックアップする契機を選択します。

  ー Session Registry Serverからの応答待ち時間
  Session Registry Serverからの応答待ち時間を設定します。設定値については、“9.5.1 各タイムアウト値の設定について”を参照してください。

  ー セションを使用しないURLの末尾
  セションを使用しないコンテンツを、URLの末尾に付く拡張子で指定します。

  ー アクセスログを出力する
  アクセスログを出力するかどうかを指定します。

isj2eeadminコマンドを使用して、上記の設定を行うこともできます。
isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

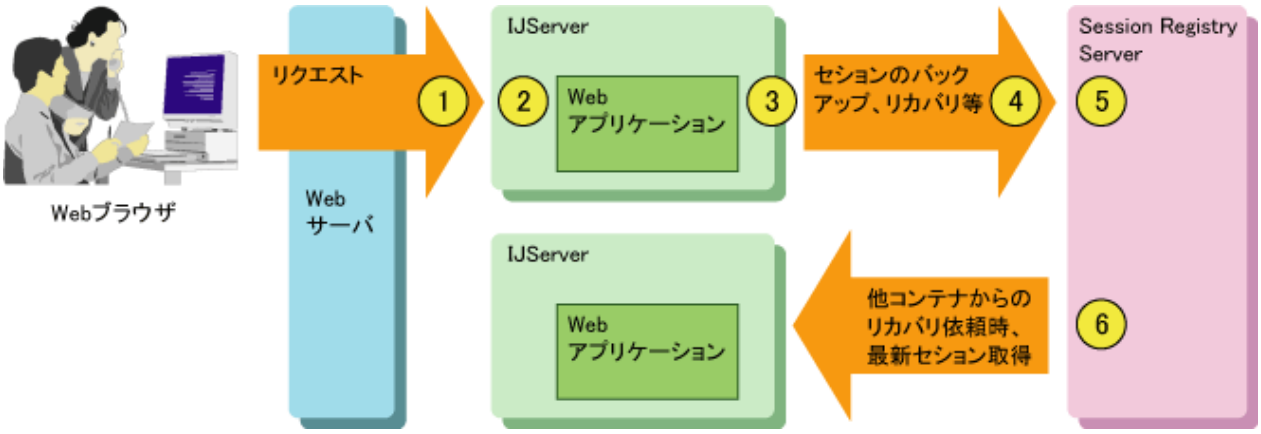
9.5 セションリカバリ機能に関する設定について

セションリカバリ機能に関する設定について説明します。

9.5.1 各タイムアウト値の設定について

セションリカバリ機能で行う、通信のタイムアウト値の設定について説明します。
セションリカバリ機能に関係して、以下のタイムアウト値を設定します。

設定箇所 [項目名]		値域/デフォルト値 Session Registry Serverでの例	内容
(1)	IJServer:Servletコンテナ設定 [タイムアウト]	1～2147483 / 60 60	クライアントとの通信が途絶えた場合に通信を切断するまでの時間を指定します。
(2)	IJServer:ワークユニット設定 [アプリケーション最大処理時間]	0～86400 / 480 480	アプリケーションの最大処理時間の監視時間を指定します。
(3)	IJServer:セションリカバリ設定 [Session Registry Serverからの応答待ち時間]	1～86400 / 440 440	Session Registry Serverからの応答待ち時間を設定します。
(4)	Session Registry Server:Servletコンテナ設定 [タイムアウト]	1～2147483 / 60 60	このSession Registry Serverが対象とするServletコンテナからの通信が途絶えた場合に通信を切断するまでの時間を指定します。
(5)	Session Registry Server:ワークユニット設定 [アプリケーション最大処理時間]	0～86400 / 480 400	Session Registry Serverの最大処理時間の監視時間を指定します。
(6)	Session Registry Server:Session Registry Server環境定義ファイル [最新セション取得待ち時間(pop.timeout)]	1～86400 / 360 360	セションのリカバリ時、もともとセションを保持していたコンテナから最新のセションの取得を行いますが、このときの最大待ち時間を設定します。 セションがリクエスト処理中で使用されている場合の待ち時間です。 セション使用中のまま待ち時間をオーバーすると、該当のセションはリカバリされません。



各値は、以下の関係を満たすように設定してください。

（2）＞（3）＞（5）＞（6）
（1）＜＝（2）
（4）＜＝（5）

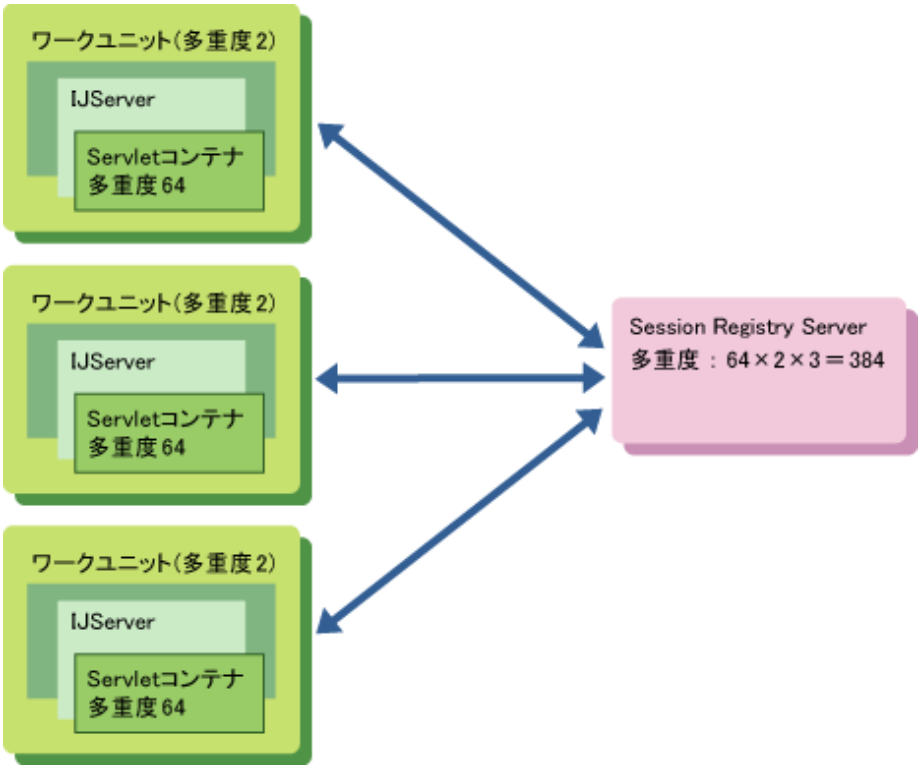
9.5.2 多重度(同時処理数)の設定について

Session Registry Server同時処理数には、そのSession Registry Serverをセションのバックアップ先とするワークユニット(IJServer)の以下の値の合計値を、Session Registry Serverの同時処理数の上限値2048以内で指定してください。

Servletコンテナの同時処理数 × ワークユニット(IJServer)のプロセス多重度

例

以下の例では、Servletコンテナの多重度64、多重度2のワークユニット3つがSession Registry Serverをバックアップ先としていますので、
64×2×3 ＝ 384 を設定します。



9.5.3 IPアドレスとポート番号の設定例

以下のような運用を行う場合について、IPアドレスとポート番号の設定例を説明します。

- Session Registry ServerをWebアプリケーションの動作するIJServerと別マシンで運用する場合
- Session Registry ServerをWebアプリケーションの動作するIJServerと同一のマシンで運用する、かつ、WebサーバとIJServerを同一のマシンで運用する場合
- Session Registry ServerをWebアプリケーションの動作するIJServerと同一のマシンで運用する、かつ、WebサーバとIJServerを別マシンで運用する場合

注意

Session Registry ServerをWebアプリケーションの動作するIJServerと別マシンで運用する場合、Session Registry Serverを運用するマシンで、Interstage管理コンソールのシステムの環境設定で以下を設定する必要があります。

- [システム]＞[環境設定]＞[詳細設定]＞[Servletサービス詳細設定]＞[Webサーバとワークユニットを同一のマシンで運用する]で“運用しない”を設定

例1: Session Registry ServerをWebアプリケーションの動作するIJServerと別マシンで運用する場合

Session Registry Server(192.0.2.111)の環境設定

	項目		設定値例	備考
(1)	Servletコンテナ設定	ServletコンテナのIPアドレス	192.0.2.111	(3)、(5)のIPアドレス、ポートと同じとします。
		ポート番号	5678	
(2)	Webサーバコネクタ(コネクタ)設定	WebサーバのIPアドレス	192.0.2.222 192.0.2.223	Webアプリケーションの動作するIJServerを運用するマシンA、BのIPアドレスを指定します。複数ある場合はすべて設定します。

IJServer(マシンA:192.0.2.222)の環境設定

	項目			設定値例	備考
(3)	セションリカバリ設定	セションのバックアップ先Session Registry Serverのアドレス:ポート		192.0.2.111:5678	(1)のIPアドレス、ポートと同じとします。
(4)	Servletコンテナ設定	制御用ポート	ポート番号	15000	
			アクセス許可IPアドレス	192.0.2.111	アクセス制限を行う場合、(1)のIPアドレスと同じとします。ただし、Session Registry Serverを運用するマシンに複数のIPアドレスが設定されている場合はすべて設定します。

IJServer(マシンB:192.0.2.223)の環境設定

	項目			設定値例	備考
(5)	セションリカバリ設定	セションのバックアップ先Session Registry Serverのアドレス:ポート		192.0.2.111:5678	(1)のIPアドレス、ポートと同じとします。
(6)	Servletコンテナ設定	制御用ポート	ポート番号	15000	
			アクセス許可IPアドレス	192.0.2.111	アクセス制限を行う場合、(1)のIPアドレスと同じとします。ただし、Session Registry Serverを運用するマシンに複数のIPアドレス

	項目			設定値例	備考
					が設定されている場合はすべて設定します。

例2: Session Registry ServerをWebアプリケーションの動作するIJServerと同一のマシンで運用する、かつ、WebサーバとIJServerを同一のマシンで運用する場合

Session Registry Serverの環境設定

	項目		設定値例	備考
(1)	Servletコンテナ設定	ポート番号	5678	(2)のポートと同じとします。

IJServerの環境設定

	項目			設定値例	備考
(2)	セションリカバリ設定	セションのバックアップ先Session Registry Serverのアドレス:ポート		127.0.0.1:5678	IPアドレスは127.0.0.1とします。ポートは(1)と同じとします。
(3)	Servletコンテナ設定	制御用ポート	ポート番号	15000 15001	(Webアプリケーションを運用するIJServer(ワークユニット)のプロセス多重度が2のとき)
			アクセス許可IPアドレス	127.0.0.1	アクセス制限を行う場合、127.0.0.1とします。

例3: Session Registry ServerをWebアプリケーションの動作するIJServerと同一のマシンで運用する、かつ、WebサーバとIJServerを別マシンで運用する場合

Session Registry Serverの環境設定

	項目		設定値例	備考
(1)	Servletコンテナ設定	ServletコンテナのIPアドレス	127.0.0.1	IPアドレスは127.0.0.1(注)とします。ポートは(3)と同じとします。
		ポート番号	5678	
(2)	Webサーバコネクタ(コネクタ)設定	WebサーバのIPアドレス	127.0.0.1	127.0.0.1(注)とします。

IJServerの環境設定

	項目			設定値例	備考
(3)	セションリカバリ設定	セションのバックアップ先Session Registry Serverのアドレス:ポート		127.0.0.1:5678	IPアドレスは127.0.0.1(注)とします。ポートは(1)と同じとします。
(4)	Servletコンテナ設定	制御用ポート	ポート番号	15000 15001	(Webアプリケーションを運用するIJServer(ワークユニット)のプロセス多重度が2のとき)

	項目			設定値例	備考
			アクセス許可IPアドレス	127.0.0.1	アクセス制限を行う場合、127.0.0.1(注)とします。

注) 各IPアドレスは、127.0.0.1のかわりに実際のマシンのIPアドレス(192.0.2.111など)を指定することも可能です。ただし、127.0.0.1との混在はできません。いずれかで統一してください。

9.6 セションリカバリ機能の運用方法

セションリカバリ機能の運用方法について説明します。

9.6.1 Session Registry Serverの操作・参照

Session Registry Serverの操作や、各種情報の参照は、Interstage管理コンソールからワークユニットの操作や参照として行います。

Session Registry Serverの状態表示と操作

[ワークユニット] > “Session Registry Server名” > [操作]

Session Registry Serverの環境設定

[ワークユニット] > “Session Registry Server名” > [環境設定]
[ワークユニット] > “Session Registry Server名” > [ログ定義]

isj2eeadminコマンドを使用して、環境設定を行うこともできます。
isj2eeadminコマンドの詳細については、リファレンスマニュアル(コマンド編)の“isj2eeadmin”を参照してください。

Session Registry Serverの一覧表示

[ワークユニット] > [状態]

Session Registry Serverのモニタ表示

[ワークユニット] > “Session Registry Server名” > [モニタ]

ログ参照

[ワークユニット] > “Session Registry Server名” > [ログ参照]
ログはIJServer(Servletコンテナ)のログと同じファイルに出力されますので、コンテナログ/起動情報を参照します。

9.6.2 Session Registry Serverの起動ユーザの変更について Solaris64 Linux32/64

Session Registry Serverを運用するワークユニットは、任意のユーザで起動可能です。ただし、セションの永続化機能使用時、初回起動以降に起動ユーザを変更した場合は、永続化したセションの引き継ぎは保証されません。

永続化機能使用時に、以前起動していたユーザ以外のユーザで運用を行う場合は、変更前に以下のどちらかの対処を行ってください。なお、以下の対処を行うためには、“serialize.file.path”で指定したディレクトリが存在しており、また変更後のユーザに全権限がある必要があります。

- 永続化ファイル消去コマンドで、永続化ファイルを消去します。
- Session Registry Server環境定義ファイルの“serialize.file.path”に指定するディレクトリを、別のディレクトリに変更します。

上記対処を行わずに異なるユーザで起動した場合、永続化に失敗することがあります。その場合は、以下のいずれかの対処を行ってください。

- 元のユーザで起動します。

注意

一度でもroot権限で起動した場合は、この対処はできません。以降いずれかの対処を行ってください。

- 永続化ファイル消去コマンドで、永続化ファイルを消去します。
- serialize.file.pathを変更後のユーザに全権限のあるディレクトリに変更します。

9.6.3 マシン切り離し

Session Registry Serverが動作しているマシンをメンテナンス等で切り離す場合は、以下の手順で行う必要があります。

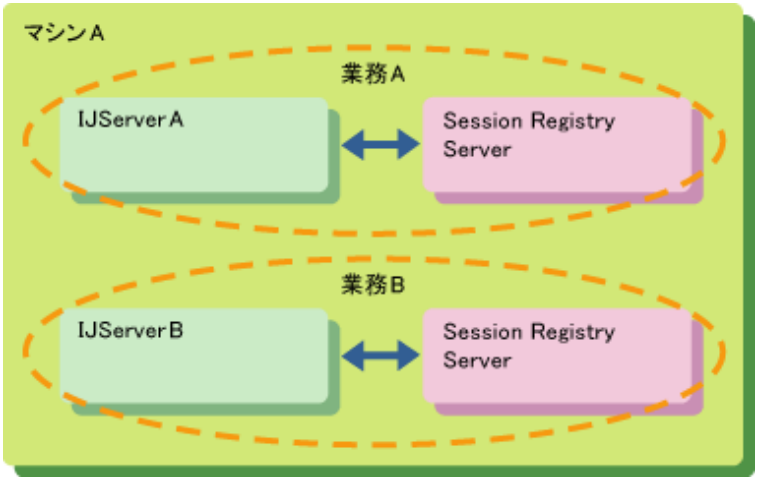
1. Session Registry Serverの停止。
2. 停止したSession Registry Serverにバックアップを行っているIJServerのコンテナログ、またはシステムログより以下を確認します。
   - “9.1.4 セションリカバリ機能の監視”によりSession Registry Serverが使用不可にマーク(メッセージJSSR32001が出力)されたこと。
3. マシンを停止。

上記手順をふまずにマシンの停止を行った場合は、運用中のWebアプリケーションのレスポンスが一時的に劣化する場合や、クライアントからのリクエストがタイムアウトする可能性があります。
なお、マシンを切り離した場合、セションのバックアップ/リカバリはできませんが、通常の業務継続は可能です。

9.6.4 Session Registry Serverの複数運用

Session Registry Serverは同一マシン内で複数運用することができます。
使用するSession Registry Serverを業務ごとに分けることによって、他の業務からの影響(パフォーマンスの影響、障害発生時の影響など)を受けることなく運用することができます。

Session Registry Serverを業務ごとに分ける例



9.6.5 Session Registry Serverの再起動について

Session Registry Server異常終了時、Session Registry Server(ワークユニット)の定義[リトライカウント]、[リトライカウントリセット時間]の設定内容によっては、Session Registry Serverの再起動を行った場合でも、再度異常終了する可能性があります。
また、異常終了の原因を取り除いていない場合も、再度異常終了する可能性があります。
たとえば使用資源量の見積もり不足による資源不足で異常終了した場合、再起動を行っても(再度見積もり資源量を超えるセションがバックアップされ)連続して異常終了することがあります。

異常終了後に再起動して、Session Registry ServerとServletコンテナの通信が回復した場合、双方の保持するセションの情報の整合性をとるため、この間一時的にWebブラウザからのリクエストを処理できません。原因を取り除かずに再起動を繰り返すと、この整合性をとる処理が繰り返されることとなり、レスポンスに影響を与える場合があります。
そのため、[リトライカウント]には1、または小さな値を設定することを推奨します。Session Registry Serverが起動していない間は、Servletコンテナはセションのバックアップやリカバリは行えませんが、通常の業務処理は可能です。異常の原因を解決

した後、Session Registry Serverを再起動してください。
なお、整合性をとる処理にかかる時間はネットワーク/マシン性能、有効なセションの数、およびサイズに依存します。

9.6.6 Session Registry Serverの資源のバックアップ・リストアについて

Session Registry Serverは、IJServerのワークユニットとして作成し、運用します。
したがって、Session Registry Serverの資源のバックアップ・リストアは、IJServer資源のバックアップ・リストアと同様の手順で実施してください。
IJServerのバックアップ・リストアにつきましては、“運用ガイド(基本編)”の“メンテナンス(資源のバックアップ/他サーバへの資源移行/ホスト情報の変更)”を参照してください。

注意

セションの永続化を有効としている場合に出力される永続化ファイルは、運用上の一時的な資源であり、バックアップ・リストアでは不要な資源ですが、セションの永続化ファイルの保存先に指定したディレクトリ(※)によっては、他の資源と共にリストアされる場合があります。
そのため、リストア後、Session Registry Serverの起動前に、jssrsadminコマンドのclearsessionサブコマンドにより、永続化されているセションの情報を消去してください。
(※)初期値の場合や、相対パスで指定した場合など。
jssrsadminコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“jssrsadmin”を参照してください。

9.7 アプリケーション作成方法

セションリカバリ機能を使用するためのWebアプリケーションの条件を説明します。

- セションの属性に格納するオブジェクトはjava.io.Serializableインタフェースをimplementsしている(オブジェクト直列化可能である)必要があります。
  この条件を満たしていない場合、setAttributeを行ったときにIllegalArgumentExceptionが発生します。
- セションの属性に格納するオブジェクトが保持する直列化可能でないインスタンス変数には、transient修飾子を宣言してください(バックアップ、リカバリの対象外となります)。
- OSおよびプロセス(Java VM)固有の資源を表すオブジェクトをセションに格納した場合、動作の保証はできません。たとえば、Threadクラスや入出力に関するクラス(InputStreamクラスやWriterクラスなどを継承したクラス)、CORBAやEJBのクライアント(スタブ、org.omg.CORBA.ORBやjavax.naming.Context等、およびこれに類するオブジェクトへの参照を保持するクラス)などはJava VMに固有の資源です。ユーザ定義オブジェクトを開発する場合には、このようなクラスを継承したり、このようなクラスへの参照を保持したりしないようにしてください。
- セションの属性にはアプリケーションのリクエストを復元するために必要な情報をすべて格納する必要があります。
  セションのリカバリ処理ではセション属性に格納された情報だけが復元され、セションとは別のオブジェクトやクラスに格納した情報は復元されません。そのため、アプリケーションのリクエスト処理がセション属性以外の動的情報に依存している場合にセションのリカバリ処理をしても、リカバリ先のServletコンテナでアプリケーションがリクエストを正しく処理できないことがあります。
- セション属性に格納するオブジェクトクラスを変更し、かつ変更前にバックアップされたセションを継続して使用する場合、“セション属性に格納するオブジェクトクラスを変更する方法”に示す対応方法に従ってオブジェクトクラスを作成、または修正する必要があります。
  この条件を満たさない場合、クラス修正前に保存されたオブジェクトをクラス修正後に復元させようとするとjava.io.InvalidClassExceptionがログ出力され、アプリケーションのリクエストは新規セションとして処理されます。

注意

セションの属性に格納するオブジェクトのサイズが大きくなればなるほど、実行速度が遅くなります。

セション属性に格納するオブジェクトクラスを変更する方法

セション属性に格納するオブジェクトクラスを変更する場合、以下の2つの方法があります。

- オブジェクトクラスの作成時に、将来の修正に備えておく方法
- オブジェクトクラスの作成時に、修正に対する備えをしていなかった場合の変更方法

オブジェクトクラスの作成時に、将来の修正に備えておく方法

オブジェクトクラスの作成時に将来属性の変更があることを見越し、かつ修正前に格納した情報を修正後も使用したい場合、以下の対応が必要です。

a) 作成時

オブジェクトクラスの作成時にserialVersionUIDをクラス変数として宣言し、任意の値(long整数)を設定しておきます。

修正前のオブジェクトクラス例

```
package com.fujitsu.sample;
import java.io.*;

public class Company implements Serializable {

  private String  longName;
  private String  shortName;
  private String  address;
  static final long serialVersionUID = 10203040506070809L;  ........(1)

  public Company(String longName,
                 String shortName,
                 String address) {
              :
              :
  }
}

(1):任意(一意)のlong整数値を設定します。
```

b) 修正時

オブジェクトクラスを変更(属性追加)します。

修正後のオブジェクトクラス例

```
package com.fujitsu.sample;
import java.io.*;

public class Company implements Serializable {

  private String  longName;
  private String  shortName;
  private String  address;
  private String  telNumber;                        ........(2)
  static final long serialVersionUID = 10203040506070809L;

  public Company(String longName,
                 String shortName,
                 String address,
                 String telNumber ) {
              :
              :
  }
}

(2):telNumber属性を追加します。
```

オブジェクトクラスの作成時に、修正に対する備えをしていなかった場合の変更方法

オブジェクトクラスの作成時に修正に対する備えをしていなく、かつ修正前に格納した情報を修正後も使用したい場合、以下の対応を行う必要があります。

a) 作成時

修正前のオブジェクトクラス例
修正前のオブジェクトクラスではserialVersionUIDの対応はされていません。

```
package com.fujitsu.sample;
import java.io.*;

public class Company implements Serializable {

  private String  longName;
  private String  shortName;
  private String  address;

  public Company(String longName, String shortName, String address) {
              :
              :
  }
}
```

b) 修正時

1. 修正前クラスに対して、serialverコマンドを実行してserialVersionUIDを取得します。

   serialVersionUIDの取得方法

```
  > serialver com.fujitsu.sample.Company    ←クラス名
    com.fujitsu.sample.Company:    static final long
    serialVersionUID = 7242562793746307240L;  ←serialVersionUIDが通知されます。
  >
```

2. 修正後のクラスに取得したserialVersionUIDをクラス変数として設定します。

   修正後のオブジェクトクラス例

```
package com.fujitsu.sample;
import java.io.*;

public class Company implements Serializable {

  private String  longName;
  private String  shortName;
  private String  address;
  private String  telNumber;                                    ....(3)
  static final long serialVersionUID = 7242562793746307240L;  ....(4)

  public Company(String longName, String shortName, String address, String telNumber ) {
              :
              :
  }
}

(3):telNumber属性を追加します。
(4):serialverコマンドで取得したsevialVersionUIDをクラス変数として設定します。
```

補足

- 修正前と修正後のオブジェクトクラスを一意なserialVersionUIDをつけることで互換性を持たせることができます。
- 修正前のオブジェクトクラスでバックアップしたセション情報を修正後のオブジェクトクラスでリカバリした場合、追加された属性値にはデフォルト値(String型:null、Int型:0 等)が通知され、エラーは通知されません。

・ オブジェクト直列化の詳細につきましては、以下のJavaの仕様についてのドキュメントを参照してください。英語 : https://docs.oracle.com/javase/8/docs/technotes/guides/serialization/index.html

第3部 EJB編

第10章　EJBサービスの機能

本章では、EJBサービスの機能について説明します。

EJBアプリケーションの形態

EJBアプリケーションには、以下の形態があります。

Session Bean

“Session Bean”は、クライアントとの対話処理を実現するオブジェクトです。一般的には、ビジネスロジックを実現しています。

Entity Bean

“Entity Bean”は、業務データ(ビジネスデータ)を表現するオブジェクトです。一般的には、“Entity Bean”のクラスは、リレーショナルデータベースのテーブルに、“Entity Bean”のインスタンスは、そのテーブル内のレコードに、それぞれマッピングされます。

Message-driven Bean

“Message-driven Bean”は、JMSメッセージを受信して処理するオブジェクトです。コンテナはJMSメッセージを受信して“Message-driven Bean”のインスタンスにマッピングしてビジネスロジックを実現しています。

通常、“Session Bean”または、“Message-driven Bean”から“Entity Bean”を呼び出して、データベースへアクセスします。

以下に、それぞれの形態の実行環境について説明します。

- Session Beanの実行環境
- Entity Beanの実行環境
- Message-driven Beanの実行環境

EJBサービスが提供する機能

EJBサービスでは、上記の実行環境とともに以下の機能を提供しています。機能の詳細については、各説明を参照してください。

- EJBサービスのトランザクション制御
- EJBサービスで使用できる時間監視機能

EJBサービス機能のチューニング

EJBサービスの機能でチューニングを行う場合は、“27.4 EJBコンテナのチューニング”を参照してください。

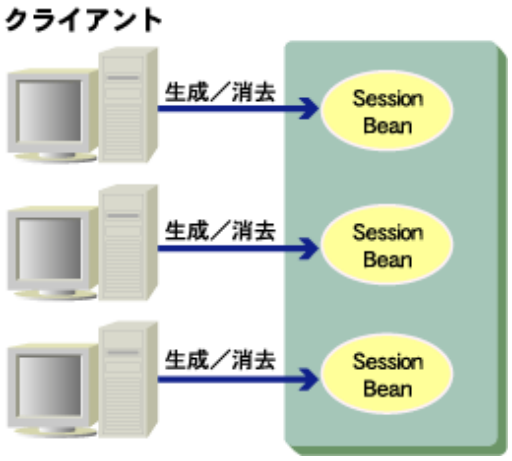
注意

EJBアプリケーションをIJServer上に配備して使用する際の注意事項を、以下に記載しています。

- EJBサービス機能における注意事項

10.1 Session Beanの実行環境

Session Beanは、クライアントとの対話処理を実現するオブジェクトです。
Session Beanのオブジェクトは、クライアントと1対1に対応して存在します。このため、サーバで実行される処理は、クライアント処理の延長として考えることができます。一般的には、ビジネスロジックを実現しています。
Session Beanは、Entity Beanを呼び出すか、または、DBMSが提供するデータベース操作命令(JDBCドライバ)を直接呼び出すことにより、データベース処理を行います。クライアントが、Session Beanのライフサイクル(生成および削除)を制御します。



次項以降で、Session Beanの実行環境について説明します。

Session Beanの時間監視については、“10.5.3 STATEFUL Session Beanの無通信監視機能”を参照してください。
STATELESS Session BeanはEJB objectを再利用するため、無通信監視機能は必要ありません。

10.1.1 Session Beanの形態

Session Beanは、クライアントとの対話処理を行うためのオブジェクトです。Session BeanにはSTATEFULとSTATELESSの2種類が存在します。
それぞれの特徴を以下に示します。

STATEFUL

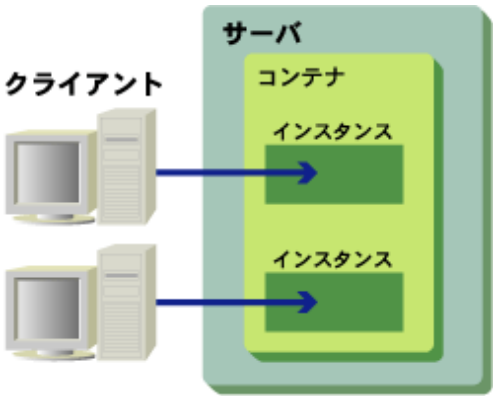
クライアントと1対1に対応し、クライアントとの対話中にトランザクション状態やアプリケーション変数を保持します。クライアントの仕事の一部を、サーバ上に分散させるための機構です。
クライアントからは、あたかもクライアント上に存在するオブジェクトのようにアクセスできます。
EJBアプリケーションのインスタンスは、クライアントからのcreate要求により生成されますが、システム異常などが発生するとSTATEFUL Session Beanの無通信監視機能によって自動的に消滅します。STATEFUL Session Beanの無通信監視機能については、“10.5.3 STATEFUL Session Beanの無通信監視機能”を参照してください。

STATELESS

複数のクライアントから共有されるオブジェクトです。
クライアントn個に対してEJBアプリケーションのインスタンスはn個以下を設定できます。
クライアントとの対話中、トランザクション状態やアプリケーション変数を保持しません。そのため、対話中の状態は、クライアント側で管理してください。
ビジネスメソッドが複数のクライアントから呼び出された場合、クライアント要求数がEJBアプリケーション定義で設定されたインスタンス数を超えると、クライアントからの要求はシリアルにキューイングされます。
EJBアプリケーションのインスタンスは、処理スレッドの数だけ作成され再利用されます。
STATEFULに比べて、サーバ側の資源を小さくしたい場合に利用します。

STATEFUL Session

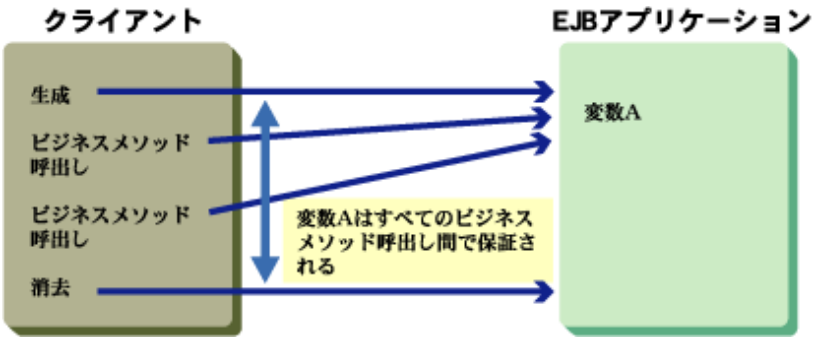
STATEFULは、STATELESSに比べて、EJBアプリケーションおよび、クライアントアプリケーションを容易に記述できるメリットがあります。



STATEFULでは、クライアントとEJBアプリケーションのインスタンスが1対1となります。
そのため、インスタンスを消去しないかぎり、クライアントとEJBアプリケーションの対話の状態を保持します。

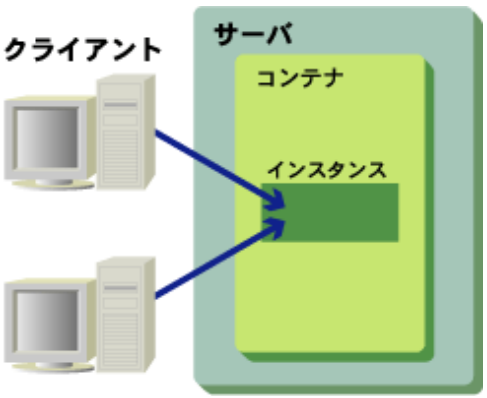
ビジネスメソッドで使用する変数は、ビジネスメソッドを終了しても保証されます。
そのため、複数のメソッドにまたがる処理ができます。

また、EJBアプリケーションで使用するトランザクションに関しても、トランザクションの開始から終了までの間に、クライアントから複数のメソッド呼出しができます。



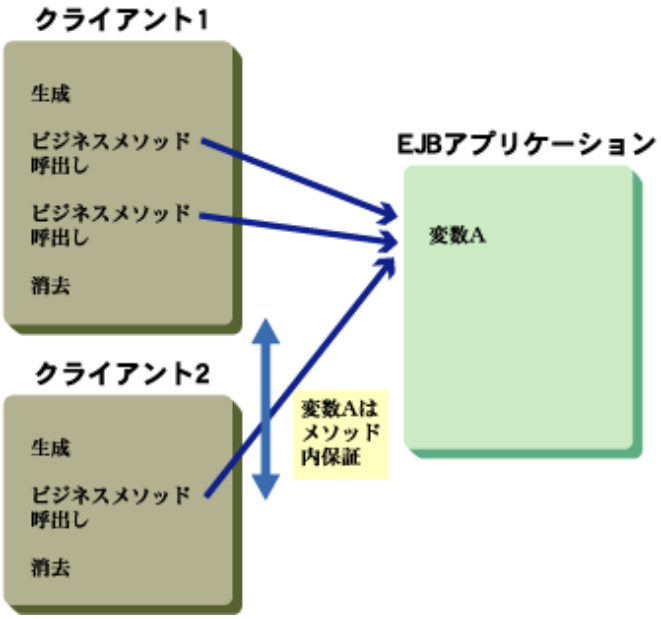
STATELESS Session

STATELESSは、STATEFULに比べて、メモリ消費量が少ないというメリットがあります。
STATELESSは、クライアント数をM個、EJBアプリケーションのインスタンス数をN個とすると、M>=Nにできます。
これにより、STATEFULに比べて、EJBアプリケーションのインスタンス数を少なくできます。



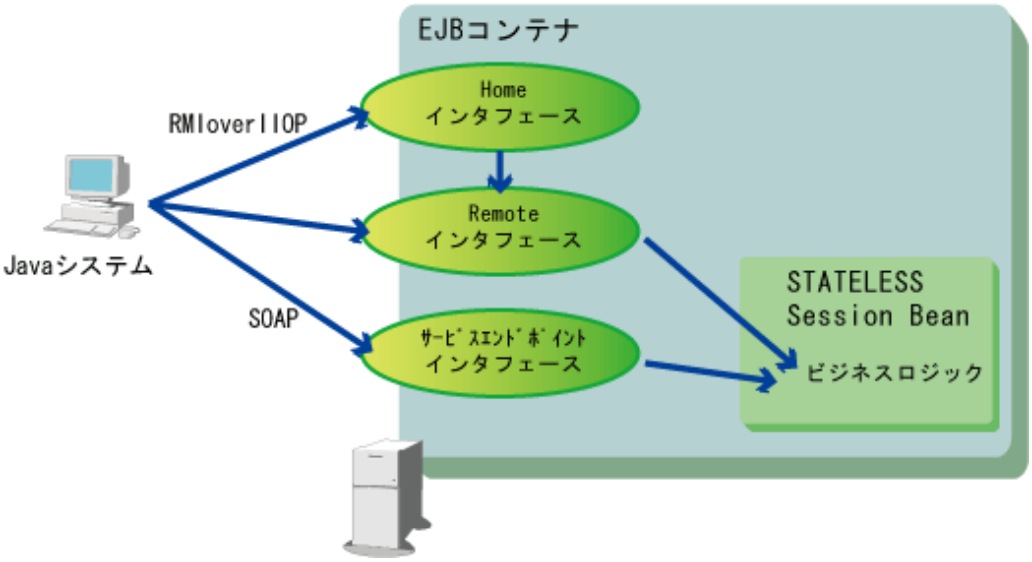
STATELESSでは、インスタンスは複数のクライアントから共用されます。
そのため、複数のビジネスメソッド間では、クライアントとEJBアプリケーションの対話の状態を保持しません。

ビジネスメソッドで使用される変数は、ビジネスメソッドを終了すると保証されません。
また、EJBアプリケーションで使用するトランザクションに関しても、トランザクションの開始と終了は、メソッド内に閉じていなくてはなりません。



10.1.2 STATELESS Session BeanのWebサービス化

STATELESS Session BeanをWebサービスのエンドポイントとして公開することができます。
Webサービスとして公開するメソッドを定義したサービスエンドポイントインタフェースを、STATELESS Session Beanのdeployment descriptorファイルに定義することでSOAPでの呼び出しが可能となります。



Webサービスアプリケーションの開発を行う場合にはEJBアプリケーションの開発方法と合わせて“第18章 Webサービスの開発”を参照してください。また、Webサービスの運用については“第19章 Webサービスの運用”を参照してください。

10.2 Entity Beanの実行環境

ここでは、Entity Beanの実行環境について説明します。

Entity Beanとは

Entity Beanは、業務データ(顧客情報、注文情報など)を表現するオブジェクトです。
Entity Beanのクラスは、リレーショナルデータベースのテーブルに、Entity Beanのインスタンスは、そのテーブル内の行(レコード)に、それぞれマッピングされます。

以下に、Entity Beanの特徴を示します。

永続性

Session Beanと異なり、Entity Beanの状態はデータベースのレコードとしてディスクに保存されるため、永続性があります。
システムの異常やサーバ停止でも消滅しません。

共有アクセス

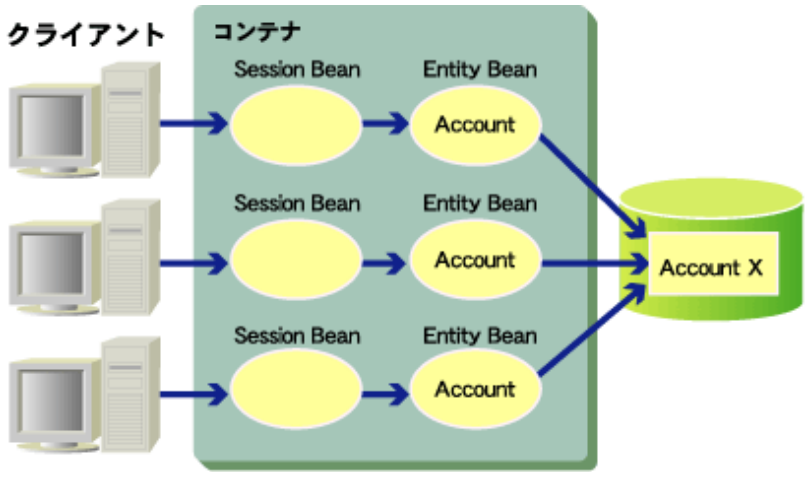
Entity Beanは複数のクライアントから共有され、同時にアクセスされることがあります。
この場合、データの整合性保証は、トランザクション管理により実現されます。

Primary Key

Entity Beanはそれぞれ一意なオブジェクト識別子(Primary Key)を持っています。
この識別子によって、クライアントが特定のEntity Beanを検索できます。

以下に、Entity Beanの例を示します。

データベースの表“Account X”を、Entity Bean“Account”で実現する例



10.2.1 Entity Beanの形態

Entity BeanにはBMP(Bean-managed persistence)とCMP (Container-managed persistence)の2種類が存在します。また、EJB QLというクエリ言語を使用して、データベースの操作ができます。
それぞれの特徴を以下に示します。

BMP(Bean-managed persistence)

EJBアプリケーション内に適切なデータベース操作文を発行する処理を記述することにより、EJBアプリケーション自身がデータの永続化を行います。
状況に応じたきめ細かいデータベース管理ができます。

CMP(Container-managed persistence)

CMP1.1

コンテナがデータの永続化を行います。このため、EJBアプリケーション自身にデータベース操作文を記述することなく、データベースにアクセスできます。
EJBアプリケーションにデータベース操作文を記述する必要がないため、ポータビリティ性の高いアプリケーションが容易に開発できます。

CMP2.0

CMP2.0ではCMP1.1の機能に加え、Entity Beanが別のEntity Beanとの関係を保持する機能を備えたことにより、CMP1.1と比べてはるかに複雑化したデータをEntity Beanに関連づけることができます。また、CMP2.0を使用することでデータベースのテーブル間のマッピングや、データベースへのSQLクエリおよびEJB QLなどに対する可搬性が向上します。

10.2.2 Entity Beanのインスタンス管理

- インスタンス数の設定
- インスタンス管理モード
- インスタンス生成モード

ポイント

インスタンス数、インスタンス管理モード、およびインスタンス生成モードを、Entity Bean単位に設定できます。
設定は、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義] > [Interstage拡張情報]で行います。

インスタンス数の設定

EJBサービスでは、Entity Beanのインスタンス用領域を仮想メモリ上でプール管理しています。
ユーザは、プール管理するインスタンスの数を設定できます。

インスタンスは、インスタンス生成モードに設定したタイミングで生成され、停止するまで保持されます。
設定したインスタンス数以上のデータ操作をした場合、インスタンスが再利用されてデータベースへのアクセスが行われるので処理性能に若干影響します。インスタンスの利用についての詳細は“15.4 Enterprise Beanインスタンス／EJB object／EJB homeの関係”を参照してください。

インスタンス管理モード

EJBサービスでは、処理性能やメモリ性能の向上を図るため、ユーザの用途に合わせたきめ細かいインスタンス管理モードを用意しています。
インスタンス管理の種類とそれぞれの用途を以下に示します。

インスタンス管理モード	用途
ReadWrite(デフォルト)	トランザクション単位にインスタンスをキャッシュすることで、同一トランザクション内でデータアクセスの性能向上を図っています。検索および更新処理を行うオンライン処理に有効です。
ReadOnly	トランザクションにまたがってインスタンスをキャッシュすることで、検索処理に特化して高速化を図っています。例えば、オンライン中に更新されないマスタ情報の検索処理などに有効です。

インスタンス管理モード	用途
Sequential	データを順番に取り出して操作するような処理に対して、メモリ性能の向上を図っています。大量データを操作するバッチ処理に有効です。ただし、上記モードと比べてレスポンス性能が若干低下します。

インスタンス生成モード

Entity Beanに対して、「インスタンス数」に指定したインスタンスの生成タイミングを選択できます。
インスタンスを生成するタイミングを、以下の表に示します。

インスタンス生成モード	タイミング
At Start-Up	Entity Bean起動処理の延長で、インスタンス数だけEntity Beanインスタンスを生成します。
At First Access	Entity Bean起動完了後の初回アクセス時に、インスタンス数だけEntity Beanインスタンスを生成します。
As Required	Entity Bean起動完了後、アクセス時にEntity Beanインスタンスがプールに存在しない場合、毎回生成を行います。生成したインスタンスは非活性化の延長(passivate時)でプーリングされます。 本モードがデフォルトとして設定されます。

注意

インスタンスの生成は、「Entity Beanのインスタンス数」を上限として行われます。

Entity Beanのインスタンス生成モードを選択する基準を、以下の表に示します。

インスタンス生成モード	選択基準
At Start-Up	1. Entity Beanの運用開始直後の処理性能を向上させたい場合
At First Access	1. Entity Beanの起動性能を向上させたい場合
As Required	1. Entity Beanの起動性能を向上させたい場合 2. Entity Beanにアクセスする頻度が低い場合

10.2.3 Entity Beanの最適化処理

Interstage Studioを使用してBMPのEntity Beanを作成した場合、Enterprise Beanの生成ウィザードで“Entity Beanの最適化処理”を選択することによって、高性能なEntity Beanを作成できます。
Entity Beanの最適化処理は、以下のいずれかの条件に当てはまる場合に使用できます。

- Entity Beanのトランザクション属性が、“Mandatory”または“Required”の場合
- 複数件検索実行中に、トランザクションを完了しない場合
- 複数件検索実行中に、コネクションを切断しない場合
- ローカル呼出しを使用している場合、または同一JavaVMのIJServerを使用している場合

上記に一致しない条件で使用した場合、Entity Beanの処理で誤動作することがありますので注意してください。
トランザクション属性については、“10.4.1 トランザクション管理種別とトランザクション属性”を参照してください。

注意

Windows32/64 Linux32/64
最適化処理を行ったアプリケーションでは、分散トランザクション機能を使用できません。

分散トランザクション機能を使用した場合、以下のすべての条件を満たすときにSQLException(“ORA-01002:フェッチ順序が無効です。”)が返却されます。

- 復帰値がEnumerationもしくはCollectionのfinderメソッドを実行
- 復帰値Enumerationに対してnextElementメソッド、もしくは復帰値CollectionのIteratorに対してnextメソッドを実行
- Oracleで1度にフェッチされるレコード数(デフォルト値 10)分だけnextElementもしくはnextを実行

10.2.4 CMP2.0の複数件検索時の高速化

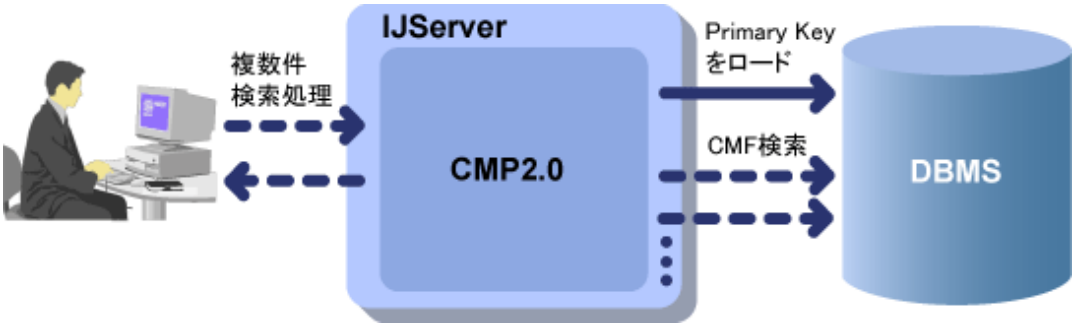
以下について説明します。

CMP2.0の複数件検索時の高速化機能とは

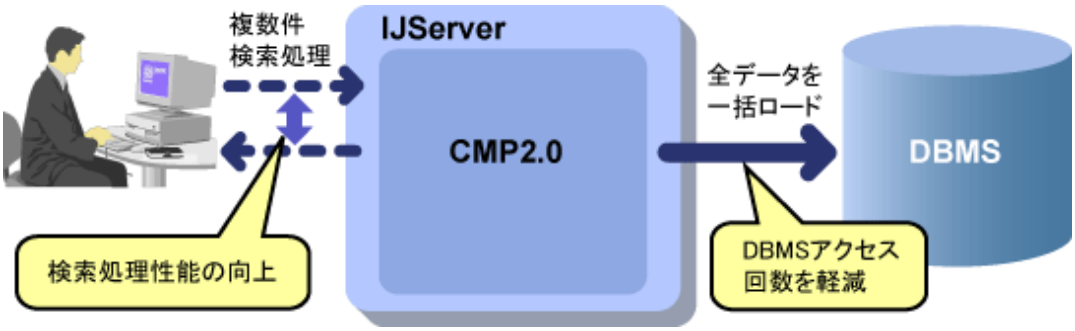
CMP2.0の複数件検索時の高速化機能とは、以下のメソッド実行時にPrimary Keyだけでなく、すべてのカラムデータを一括で検索するSQL文を実行し、SQL文の発行回数を削減するオプションのことです。

- CMP2.0 Entity Beanの複数件finderメソッド
- 1:多 relationshipのCMRに対するgetアクセッサメソッド
- 多:多 relationshipのCMRに対するgetアクセッサメソッド

V7.0以前と8.0での、複数件検索処理の処理イメージは以下のとおりです。



V7.0以前でCMP2.0 Entity Beanの複数件finderメソッドを実行すると、Primary Keyだけをデータベースからロードして、Primary Keyに対応するEJB objectをクライアントに返却しています。
Primary Key以外のデータ(CMF)については、各EJB objectに対して、CMFの初回アクセス時にデータベースからデータをロードしていました。この処理は、特定のPrimary Keyのデータだけにアクセスする場合には、余分なデータをロードすることがないため、効率よく処理することができますが、すべてのデータをデータベースからロードするような場合には、データベースへのアクセス回数が多くなります。



8.0でCMP2.0 Entity Beanの複数件finderメソッドを実行した場合は、レコードのデータを1度にすべてロードする「CMP2.0の複数件検索時の高速化機能」を使用することで、すべてのデータをデータベースからロードするような処理の場合にも、高速にデータベースからデータをロードすることができます。
この機能は1:多、または多:多のrelationshipを持つEntity Beanに対してCMRのgetアクセッサメソッドを実行する場合に、relationshipを持つ複数のEntity Beanのデータをロードする時にも有効です。

設定方法

設定は、Interstage管理コンソール、またはコマンドを使用します。
Interstage管理コンソールについては、Interstage管理コンソールヘルプを参照してください。
コマンドについて詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

システムの複数件検索の高速化設定

Interstage管理コンソール

すべてのCMP2.0 Entity Beanとrelationshipの複数件検索を高速化する場合、[システム] > [環境設定]タブで、「CMP2.0の複数件検索高速化」に“全Bean、全relationshipに適用する”を設定してください。
デフォルトは“Bean、relationshipごとに設定する”です。この場合、CMP2.0 Entity Bean、relationshipごとの設定ができます。

コマンド

isj2eeadminコマンドを使用します。

CMP2.0 Entity Bean単位の複数件の高速化設定

Interstage管理コンソール

CMP2.0 Entity Bean単位で複数件検索を高速化するかを設定する場合は、[システム] > [ワークユニット] > “ワークユニット名” > “モジュール名” > “Bean名” > [アプリケーション環境定義] > [Interstage拡張情報]で行います。

コマンド

ejbdefimportを使用します。

relationship単位のCMFアクセスの高速化設定

Interstage管理コンソール

CMP2.0 Entity Bean単位で複数件検索を高速化するかを設定する場合は、[システム] > [ワークユニット] > “ワークユニット名” > “モジュール名” > “Bean名” > [アプリケーション環境定義] > [CMRマッピング定義]で行います。

コマンド

ejbdefimportを使用します。

効果

SQL文の発行回数を削減することによって、データベースとの通信回数が削減されるため、検索処理性能が向上します。

- CMP2.0 Entity Beanの複数件finderメソッド
- 1:多 relationshipのCMRに対するgetアクセッサメソッド
- 多:多 relationshipのCMRに対するgetアクセッサメソッド

以下に、複数件検索の高速化オプションを使用しない場合と使用した場合に、発行されるSQL文の違いについて説明します。

複数件検索finderメソッド実行時

ここでは、以下の場合を例にして説明します。

- 複数件finderメソッドがfindAll
- CMFがidとname
- CMFに対応するカラムがidとname
- DBMSのテーブル名がEMPLOYEE

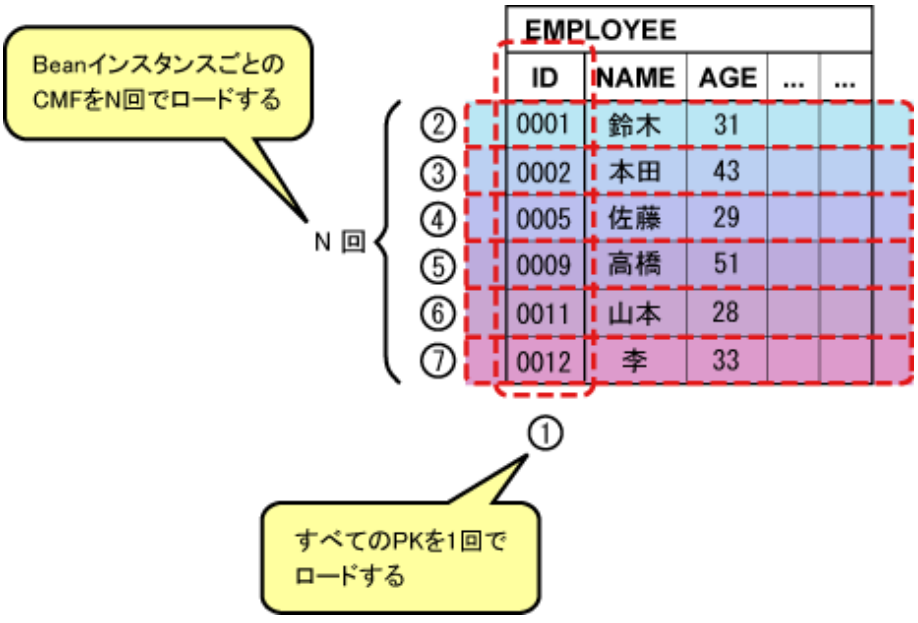
複数件検索の高速化オプションを使用しない場合

SQL文は、以下のタイミングで発行されます。

複数件finderメソッド(findAll)の使用例	SQL文発行のタイミング
Collection employees = employeeHome.**findAll**();	SELECT **id** FROM EMPLOYEE
Iterator iter = employees.iterator();	
while (iter.hasNext()) {	
EmployeeLocal e = (EmployeeLocal)iter.next();	
System.out.println("Id: " **+ e.getId**());	SELECT **name**,**age**,・・・FROM EMPLOYEE WHERE id = ?
System.out.println("Name: " + e.getName());	
}	

複数件finderメソッド(findAll)の実行時に、Primary Keyフィールド(id)をロードします。
各Beanインスタンスの最初のCMFアクセス時(getIdメソッド実行時)に、CMFフィールドをすべてロードします。

SQL文発行数の総計は、以下の図に示すようにN＋1回になります(「＋1」は(1)の検索が必要なため)。



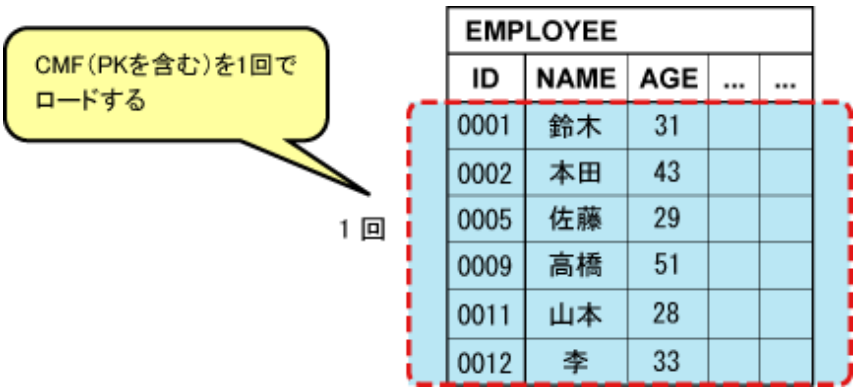
複数件検索の高速化オプションを使用する場合

SQL文は、以下のタイミングで発行されます。

複数件finderの使用例	SQL文発行のタイミング
Collection employees = employeeHome.**findAll**();	SELECT **id**,**name**,**age**,・・・FROM EMPLOYEE
Iterator iter = employees.iterator();	
while (iter.hasNext()) {	
EmployeeLocal e = (EmployeeLocal)iter.next();	
System.out.println("Id: " + e.getId());	
System.out.println("Name: " + e.getName());	
}	

複数件finderメソッド(findAll)の実行時に、すべてのカラムデータを取得するSQL文を発行します。
コンテナは、ResultSetからレコードをすべて取得しメモリ上にキャッシュします。CMFアクセス時には、値をキャッシュから取得します。

複数件finderメソッドを呼び出すと、CMF(PKフィールドを含む)をすべてロードするので、SQL文発行数の総計は、以下の図に示すように1回となります。



1:多、多:多relationshipのgetアクセッサメソッド実行時

ここでは、以下の場合を例にして説明します。

- getアクセッサメソッドがgetEmployeesメソッド
- データベースのテーブル名がEMPLOYEE

複数件検索の高速化オプションを使用しない場合

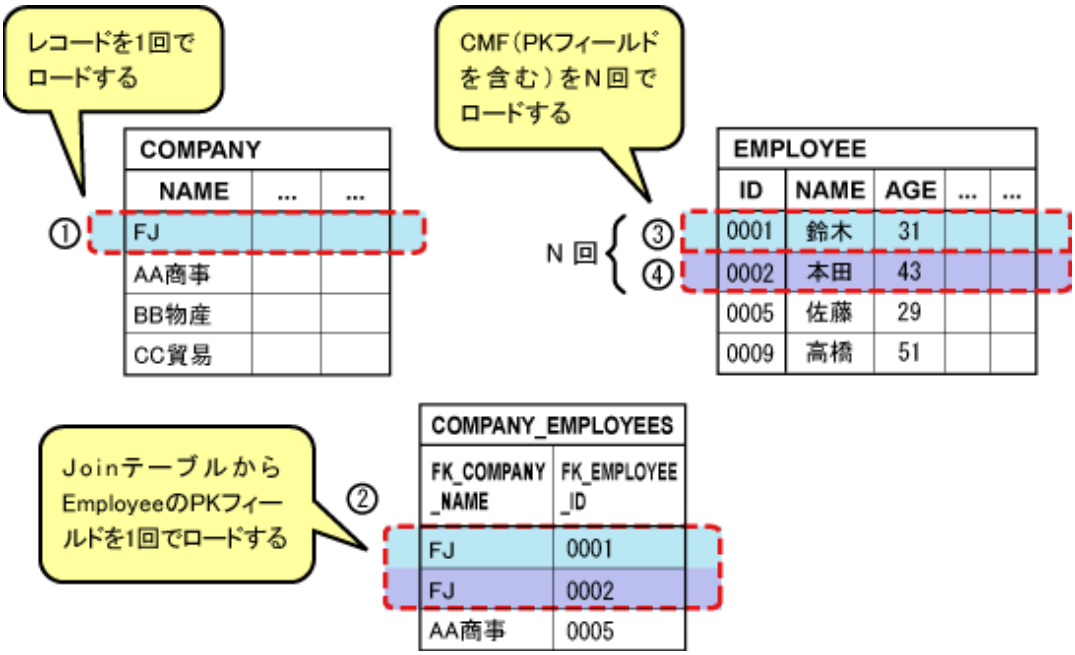
SQL文は、以下のタイミングで発行されます。

1:多、多:多 relationshipのアクセッサ使用例	SQL文発行のタイミング
CompanyLocal fj = companyHome.**findByPrimaryKey**("FJ");	SELECT name FROM COMPANY WHERE name = ?

1:多、多:多 relationshipのアクセッサ使用例	SQL文発行のタイミング
Collection employees = fj.getEmployees();	SELECT **fk_employee_id** FROM COMPANY_EMPLOYEE WHERE fk_company_name = ?
Iterator iter = employees.iterator();	
while (iter.hasNext()) {	
EmployeeLocal e = (EmployeeLocal)iter.next();	
System.out.println("Id: " + e.**getId**());	SELECT **name**,**age**,・・・FROM EMPLOYEE WHERE id = ?
System.out.println("Name: " + e.getName());	
}	

1:多、多:多relationshipのCMRのgetアクセッサメソッドを呼び出すと、Joinテーブルから Primary Keyフィールドをロードします。
各Beanインスタンスの、最初のCMFアクセス時に、CMFをすべてロードします。

テーブルへのSQL文発行数の総計は、以下の図に示すようにN＋2回になります(「＋2」は(1)と(2)の検索が必要なため)。



複数件検索の高速化オプションを使用する場合

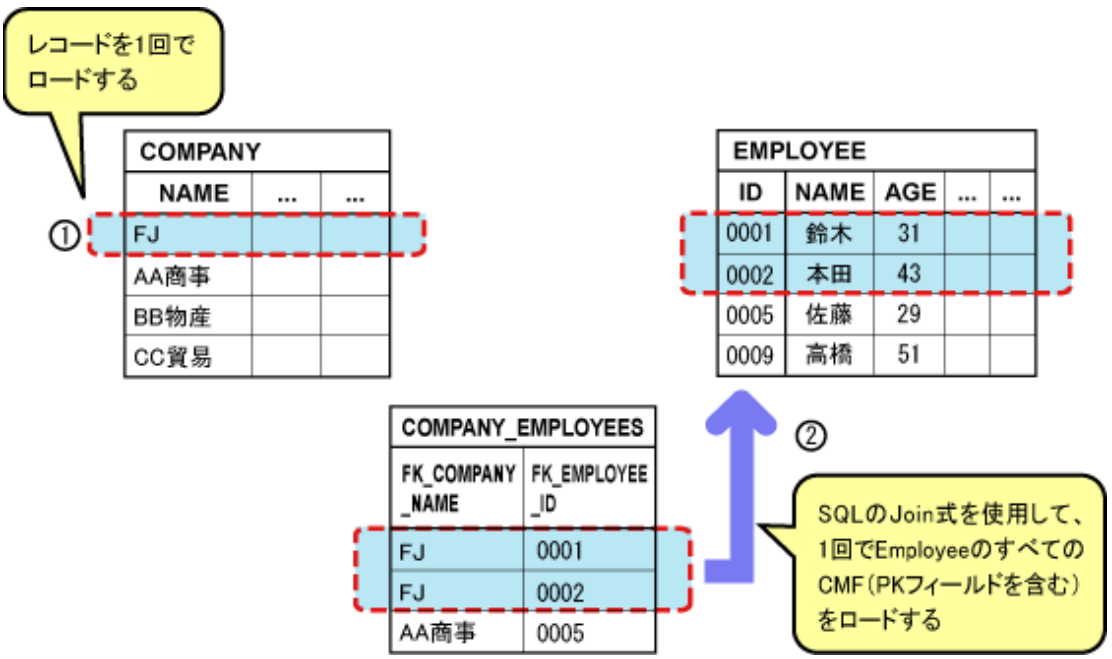
SQL文は、以下のタイミングで発行されます。

1:多、多:多 relationshipのアクセッサの使用例	SQL文発行のタイミング
CompanyLocal fj = companyHome.**findByPrimaryKey**("FJ");	SELECT name FROM COMPANY WHERE name = ?
Collection employees = **fj.getEmployees**();	SELECT **e.id**,**e.name**,**e.age**, ・・・ FROM EMPLOYEE e LEFT OUTER JOIN COMPANY_EMPLOYEE ce ON e.id =

1:多、多:多 relationshipのアクセッサの使用例	SQL文発行のタイミング
	ce.fk_employee_id WHERE ce.fk_company_name = ?
Iterator iter = employees.iterator();	
while (iter.hasNext()) {	
EmployeeLocal e = (EmployeeLocal)iter.next();	
System.out.println("Id: " + e.getId());	
System.out.println("Name: " + e.getName());	
}	

1:多、多:多relationshipのアクセッサメソッドの実行時に、すべてのカラムデータを取得するSQL文を発行します。
コンテナは、ResultSetからレコードをすべて取得しキャッシュします。CMFアクセス時には、値をキャッシュから取得します。

relationshipのアクセッサメソッドを呼び出すと、CMF(PKフィールドを含む)をすべてロードするので、SQL文発行数の総計は、以下の図に示すように2回となります。



注意

- インスタンス管理モードがReadOnlyの場合、プロセス内で一度アクセスしたレコードデータはキャッシュされます。同じレコードにアクセスした場合には、プライマリキーが分かればキャッシュされているデータが使用されます。
  このため、複数件検索時にプライマリキーだけを検索するデフォルトの設定より、本オプションを設定した場合に、性能が劣化する可能性があるため、本オプションの設定は無効となります。
- EJB QLクエリでDISTINCT句(検索結果の重複している行を取り除いて結果を返す)を設定しているfinderメソッドに本機能を設定した場合、設定は有効になりません。
  これは、DISTINCT句が設定されているfinderメソッドに本機能を適用した場合、Primary Keyフィールドだけでなく、CMFフィールドの重複確認も行ってしまい、本機能を適用せずにPrimary Keyフィールドのみ重複確認を行った場合の方が性能が良い場合があるためです。

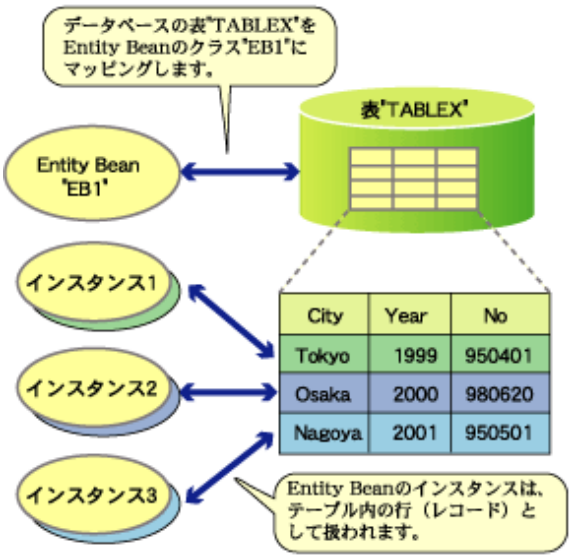
データベースによっては、DISTINCT句との併用がサポートされていないデータ型が存在する場合もあります。
DISTINCT句が使用されたEJB QLに本機能を適用しようとした場合には、起動時に警告が出力されます。

10.2.5 Entity Beanとデータベースの対応

- Entity Beanとリレーショナルデータベースのマッピング
- オブジェクトの操作とデータベース処理の対応
- CMP2.0の更新データの一貫性保証

Entity Beanとリレーショナルデータベースのマッピング

一般的に、Entity Beanのクラスは、リレーショナルデータベースのテーブルに、Entity Beanのインスタンスは、そのテーブル内の行(レコード)に、それぞれマッピングされます。
以下にEntity Beanとリレーショナルデータベースのマッピング例を示します。



オブジェクトの操作とデータベース処理の対応

Entity Beanのオブジェクトを操作することにより、対応するデータベース処理を行うことができます。
以下にオブジェクトの操作とデータベース処理の対応を示します。

オブジェクトの操作	対応するデータベース処理	発行するデータベース操作文
オブジェクトの生成	新しいレコードが追加される	INSERT文
オブジェクトの削除	レコードが削除される	DELETE文
オブジェクトの呼出し	レコードの検索または、更新 (レコードの更新は、トランザクションの終了時に行われます。)	SELECT文(検索の場合) UPDATE文(更新の場合)

検索メソッド

以下のメソッドを使用して、データベースの操作を行います。

finderメソッド

- EJBアプリケーションのHomeインタフェースに定義され、単一または複数のEJBアプリケーションに対するリファレンスを取得します。

- CMP2.0では戻り値が複数の場合の型は、java.util.Collection型だけサポートされています(java.util.EnumerationはCMP1.1およびBMPではサポートされています)。

ejbSelectメソッド

- EJBアプリケーション内部でだけ使用できるプライベートメソッドです。
- ejbSelectメソッドを使用してCMFやCMRフィールドの値や、Local/Remoteインタフェースを取得できます。
- ejbHomeメソッドやビジネスメソッドからも呼び出せることにより、finderメソッドよりも柔軟性に優れていますが、Homeインタフェースではなく Enterprise Beanクラスで抽象メソッドとして定義されるので、クライアントには公開されません(クライアントから呼び出すことができません)。
- CMP2.0では戻り値が複数の場合の型は、java.util.Collection型だけサポートされています(java.util.Set型は現在未サポートです)。

Entity Beanのインスタンスについては、“27.4 EJBコンテナのチューニング”を参照してください。

CMP2.0の更新データの一貫性保証

CMP2.0 Entity Beanで、同一データに対して複数から更新処理を実行する場合に、データの整合性が保証できず、デッドロックエラーが発生することを抑止するCMP2.0の更新データの一貫性保証機能を使用できます。

デッドロックエラーを抑止するためにデータベースでは、テーブルを検索するSELECT文にFOR UPDATE句を付加して検索した行に更新ロックをかけることで、同一レコードに複数トランザクションからアクセスする場合の検索から更新までのデータの一貫性を保証します。
更新ロックとは、検索処理時に参照した行にロックをかけることで、トランザクション処理が完了するまでデータの一貫性を保証するものです。
更新ロックされた行は、別のトランザクションでは参照できません。

CMP2.0 Entity Bean単位で行の更新ロックを使用すると、コンテナがデータベースに検索処理を実行する以下の処理で、更新ロックがかかります。

- finderメソッド
- ejbSelectメソッド
- CMRへのアクセッサメソッド
- CMFのロード

設定は、Interstage管理コンソールの[システム] > [ワークユニット] > “ワークユニット名” > “モジュール名” > “Bean名” > [アプリケーション環境定義]の「CMFマッピング定義」で行います。デフォルトは“しない”です。

詳細はInterstage管理コンソールのヘルプを参照してください。

注意

- ejbSelectで実行される集合関数(MAXなど)を使用した検索処理については、FOR UPDATEの使用はデータベースの仕様により禁止されているため、更新ロックはかかりません。
  FOR UPDATE句がサポートされていないデータベースを使用して本機能を使用すると、起動時にエラーメッセージが出力され、起動または活性化に失敗します。
  FOR UPDATE句については、使用するデータベースのマニュアルを参照してください。
- データベースにSymfowareを使用している、および、EJBアプリケーションにrelationshipを設定している場合、本機能を使用すると起動時にエラーメッセージが出力され、起動または活性化に失敗します。

本機能使用時にデッドロックエラーが発生した場合は、“29.9.11 デッドロックが発生する場合”を参照してください。

10.2.6 relationshipの管理

- relationship定義について
- CMPのrelationshipのイメージ
- 外部キー
- Joinテーブル
- cascade-delete

relationship定義について

CMP2.0を使用する場合、Bean間/オブジェクト間のrelationshipが定義できるようになったため、データベースのテーブル間の関連づけが簡単に行えます。
このデータベースのテーブルとテーブルの間の関係が、relationship(リレーション)です。
relationship定義を行うことによって、各Bean間の参照関係が定義できます。1つのrelationship定義は2つのEntity Bean間の関係をモデル化します。

あるEntity Beanが、もう一方のEntity BeanのCMRフィールドをリファレンスとして保持する場合、これら2つのEntity Bean間は単方向のリレーションとなります。また、2つのBeanが互いにリファレンスを持ち、どちら側からも参照可能な場合は、双方向のリレーションとなります。このように参照状態をリレーションの方向(リファレンス)として表します。

またEntity Bean間のリレーションには、1:1、1:多、多:多という形態がサポートされます。
各Entity Beanに対して、1(One)、または多(Many)のいずれかを“多重度”として指定します。
リレーションを持っているA BeanとB Beanを例とした場合、Aの多重度が「1」の場合、BはAと1リレーションだけ持つことができます。
一方、Bの多重度が「多」の場合には、AはBと複数のリレーションを持つことができます。したがって、A BeanとB Beanは「1:多」という形態のリレーションになります。

以下の定義をdeployment descriptorファイルに記述します。

- CMRフィールド(container managed relationship field)
- 方向(リファレンス)
- 多重度

CMRフィールド(container managed relationship field)

CMRフィールドとは、Bean間の永続的な関係を保持するフィールドです。relationship定義内にCMRフィールドが記述されているBeanは、もう一方のBeanを参照できます。

方向(リファレンス)

Beanの関係をあらわす方法として単方向または双方向があります。

種類	説明
単方向	Bean AはBean Bを参照できるが、Bean BはBean Aを参照できない。
双方向	Bean AはBean Bを参照でき、逆にBean BはBean Aを参照できる。

多重度

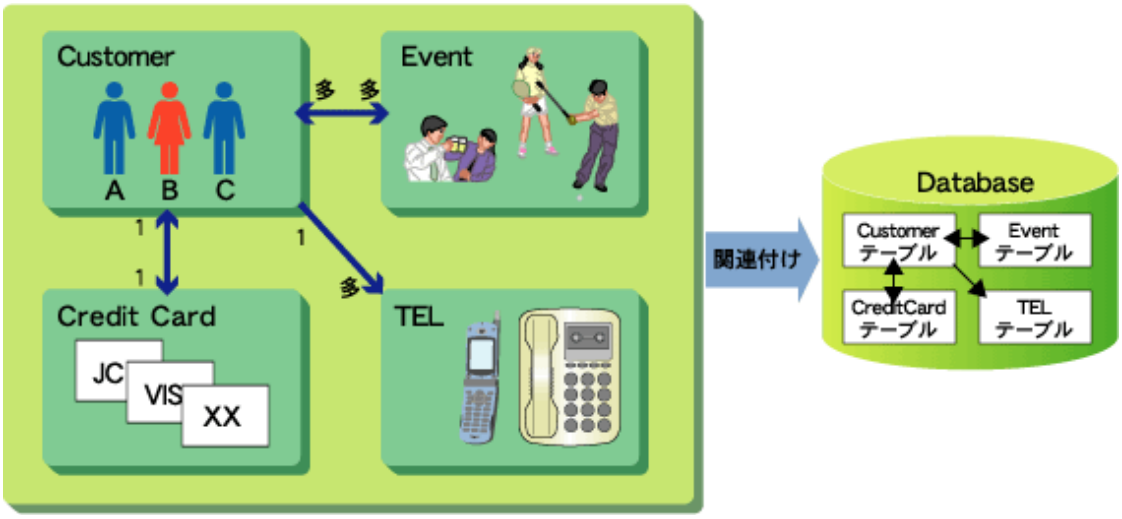
Bean間の多重度は以下の3種類です。以下にそれぞれの多重度の例を説明します。

種類	例
1:1	顧客とクレジットカードの関係。 顧客1人とクレジットカード1枚は対応している。

種類	例
1:多	顧客と電話番号の関係。 1人の顧客が多くの電話番号(自宅、携帯、職場など)を持っている状態。
多:多	顧客とイベントの関係。 1人の顧客は何度もイベントに参加でき、1回のイベントには複数の顧客が参加する。

また、Enterprise BeanクラスでCMRフィールドに対するget/setアクセッサメソッドを定義し、定義したアクセッサメソッドを使用して、relationshipの設定や変更を行います。
コンテナは、deployment descriptorに定義されたrelationshipを参照して、Bean間の関係を管理します。

CMPのrelationshipのイメージ



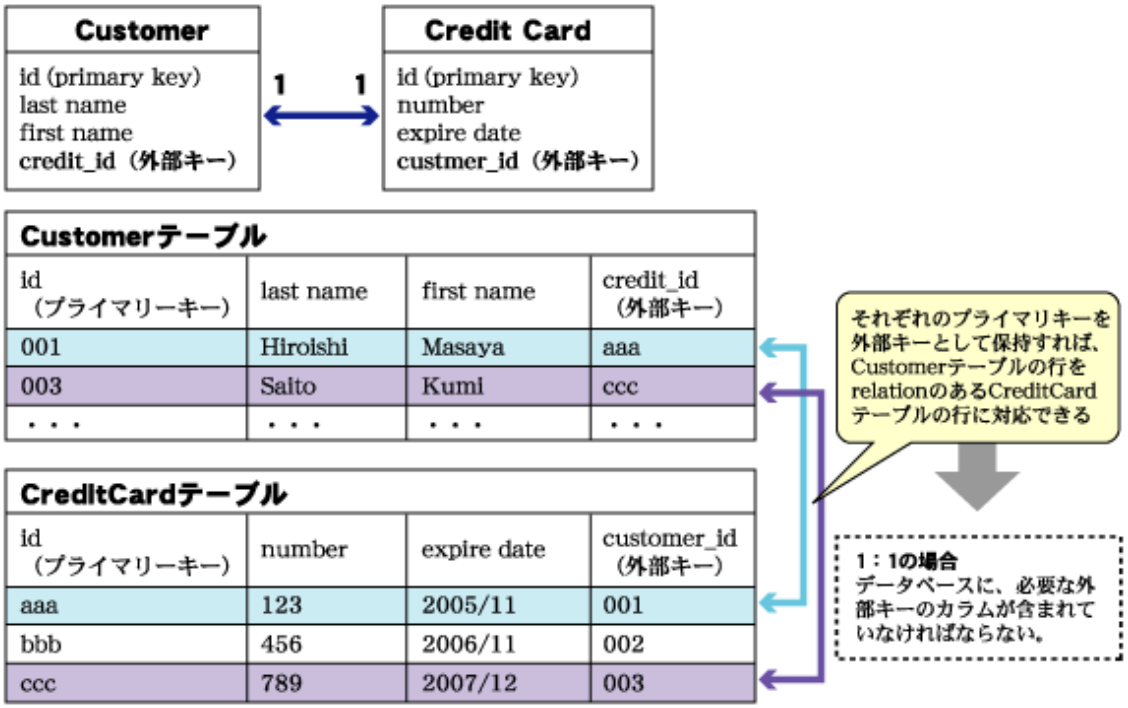
外部キー

コンテナデータベーステーブルのrelationshipを自動的に判断するために、外部キー(データベーステーブルにおけるカラムのうち、他のテーブルにおけるプライマリキーに対応するもの)を使用します。

外部キーは、1:1で双方向のリレーションの場合だけでなく、1:1で単方向のリレーションの場合においても必要です。例えば、1:1で単方向のリレーションを持つEJBアプリケーションを、途中で1:1の双方向のリレーションに変更した場合、始めから外部キーを定義していないとデータベースを再構築する必要があります。このため、1:1で単方向のリレーションの場合においても、あらかじめ外部キーを定義しておきます。

例

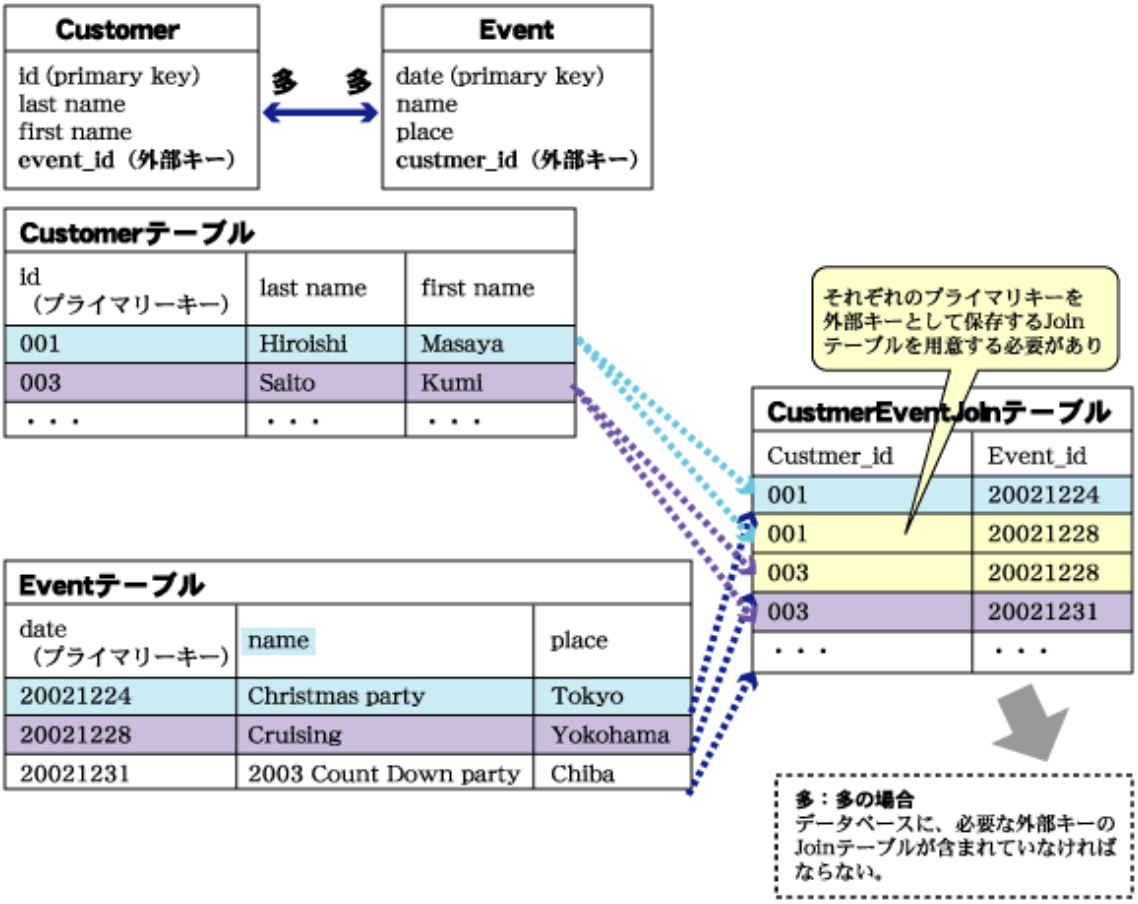
以下に1:1 relationshipの例として、Customer(顧客)とCredit Card(クレジットカード)の関係を示します。
例のようなデータベーステーブルがある場合、例えばIDが001であるCustomerのクレジットカードについての情報をたどれます。



Joinテーブル

例

以下に多:多 relationshipの例として、Customer(顧客)とEvent(イベント)の関係を示します。
例えば2002/12/24(プライマリキー)のクリスマスパーティイベントにID(プライマリキー)が001であるCustomerが参加する場合、多:多の関係が更新され、Joinテーブルが更新されます。



cascade-delete

CMP2.0Beanのレコードが破棄されると、コンテナはそのBeanが保持しているrelationshipから関連づけられたBeanのレコードを削除します。例えばCustomerレコードの破棄に伴い、CreditCardBeanレコードの破棄もできます。
これを“cascade-delete”と呼び、1:1または1:多の関係においてdeployment descriptorには<cascade-delete>要素を指定することで有効となります。多:多の場合、例えばCustomer:Eventでは、1人の顧客がイベントの参加をキャンセルしてもイベントが開催中止にはならないため、cascade-deleteは意味を持ちません。

注意

relationshipのあるアプリケーションは、Enterprise Bean名で関係付けられます。EJBアプリケーション名(JNDI名)を変更して複数配備した場合でもEnterprise Bean名は変わらないために以下に注意してください。
同名のEnterprise Beanが存在した場合には、relationshipの整合性を保つために、一方のEJBアプリケーションのCMFマッピング定義／Relation定義を変更すると双方の定義が更新されます。

10.2.7 EJB QL

- EJB QLとは
- EJB QL文
- EJB QLで使用できる機能
- EJB QL基本
- EJB QL拡張

EJB QLとは

EJB QLとは、finderメソッドまたはejbSelectメソッドなど検索メソッドのクエリ文を定義するためのSQL92ベースの言語です。一般的に使用するSQLとの最大の違いは、SELECT型にEntity Beanの識別子だけが単独で指定されている場合に指定する**OBJECT**キーワードです。

EJB QLは抽象スキーマ名に基づいて記述されるので、特定のデータベースに依存しない互換性があります。抽象スキーマ名は、CMP/CMRフィールドに並んで抽象永続スキーマを構成する重要な構成要素です。

注意

抽象スキーマ名は、ejb-jar内で一意になるようにしてください。

EJB QL文

EJB QLを使用するには、deployment descriptor内の<query>タグにEJB QL文を記述します。
EJBアプリケーションのdeployment descriptorの情報が定義されているファイル(ejb-jar.xml)は、XML形式で記述する必要があります。このため、ejb-jar.xmlを編集する場合は、XML形式の仕様に従ってください。
特に以下の文字については、定義済み実体参照(暗黙定義エンティティ)で記述してください。
EJB QLが使用できるのは、CMP2.0のEntity Beanです。

ejb-jar.xmlを編集する場合

編集する文字	定義済み実体参照
<	<
>	>
&	&
'	'
"	"

EJB QLでは、以下のEJB規約が使用できます。

- EJB2.0で規定しているEJB QL(EJB QL基本)
- EJB2.1で規定しているEJB QL(EJB QL拡張)

EJB QL文とは、SELECT句、FROM句、WHERE句を使用して1つまたは複数のEJBオブジェクトを検索するクエリ文であり、SELECT句とFROM句の定義は必須です。このEJB QL文はコンテナによって解析され、適切なデータベースクエリ言語に翻訳されます。

SELECT句

返却値の値としてOBJECT(Entity Bean)、またはパス式を使ってCMF、CMRフィールドを指定します。

FROM句

EJBアプリケーションのクラス名ではなく、Entity Bean内に記述された一意の“抽象スキーマ名”を指定します。

WHERE句

検索条件の指定を行います。例えば ?1は、検索メソッドのパラメタの1番目を意味します。
Entity Beanの検索メソッドにはfinderメソッド、またはejbSelectメソッドがあります。それぞれの検索メソッドにおいてメソッド名、パラメタの型を指定してください。
また、メソッドの返却値の型をマッピングするために、結果のタイプにLocalまたはRemote要素を指定します。デフォルト値はLocalです。

EJB QLで使用できる機能

EJB QLおよびEJB QL拡張で使用できる機能は、以下です。

EJB QL構文			基本	拡張	説明
データ操作文	SELECT句		○	○	クエリに返却するオブジェクトまたは値を指定します。
	FROM句		○	○	SELECT句およびWHERE句内で指定した値の取得場所を指定します。
	WHERE句		○	○	クエリによって返却される値を制限するための条件を指定します。
	ORDER BY句		×	○	クエリによって返却される結果をソートします。
SELECT句の返却値	パス式(CMF/CMR)		○	○	単一またはコレクションの値を持つ式です。
	OBJECT(Entity Bean)		○	○	SELECT句にEntity Beanの識別子だけが単独で指定されている場合に指定します。
	集合関数	MAX	×	○	集合で一番大きな値を返却します。
		MIN	×	○	集合で一番小さな値を返却します。
		SUM	×	○	集合の合計値を返却します。
		AVG	×	○	集合の平均値を返却します。
		COUNT	×	○	集合の要素数を返却します。
条件式	比較式		○	○	指定した2つの値を比較して評価します。
	BETWEEN式		○	○	値が指定した範囲内に含まれるかを評価します。
	LIKE式		○	○	LIKEに続く文字列が合致するかを評価します。
	IN式		○	○	値が含まれているかどうかを評価します。
	NULL比較式		○	○	値がNULL値かを評価します。
	空コレクション比較式(EMPTY比較式)		○	○	コレクション値を持つパス式が要素をもつかを評価します。
	コレクションメンバ式(MEMBER式)		○	○	値がコレクションのメンバであるかを評価します。
演算子	ナビゲーション演算子	. (ピリオド)	○	○	関係を辿るシンボルです。
	算術演算子	+ - (単項)	○	○	+ : 指定した値は正の値です。 - : 指定した値は負の値です。
		* / (乗算 除算)	○	○	* : 指定した値を乗算します。 / : 指定した値を除算します。

EJB QL構文			基本	拡張	説明
		+ - (加算 減算)	○	○	+ : 指定した値を加算します。 - : 指定した値を減算します。
	比較演算子	=	○	○	“=”の左辺の値と右辺の値が等しい場合に真となり、等しくない場合に偽となります。
		<>	○	○	“<>”の左辺の値と右辺の値が等しくない場合に真となり、等しい場合に偽となります。
		<	○	○	“<”の左辺の値が右辺の値より小さい場合に真となり、大きいまたは等しい場合に偽となります。
		>	○	○	“>”の左辺の値が右辺の値より大きい場合に真となり、小さいまたは等しい場合に偽となります。
		<=	○	○	“<=”の左辺の値が右辺の値より小さいまたは等しい場合に真となり、大きい場合に偽となります。
		>=	○	○	“>=”の左辺の値が右辺の値より大きいまたは等しい場合に真となり、小さい場合に偽となります。
	論理演算子	NOT	○	○	“NOT”に導かれる式が真の場合に偽となり、偽の場合には真となります。
		AND	○	○	“AND”の左辺および右辺が真の場合に真となり、そうでない場合には偽となります。
		OR	○	○	“OR”の左辺または右辺のいずれかが真の場合に真となり、左辺および右辺が偽の場合には偽となります。
関数式	文字列式	CONCAT	○	○	引数で与えられた文字列を結合した値を求めます。
		SUBSTRING	○	○	指定した文字列の部分列を求めます。
		LOCATE	○	○	条件に指定した文字列の位置を求めます。
		LENGTH	○	○	文字数またはバイト数を求めます。
	算術式	ABS	○	○	引数に指定した値に対する“絶対値”を求ます。
		SQRT	○	○	検索条件に指定した値の平方根を求めます。
		MOD	×	○	検索条件に指定した値の余剰を求めます。

○:指定可能　×:指定不可

注意

- 使用するDBMSがSymfowareの場合、以下に注意してください。
  - ー SQRT関数、MOD関数は使用できません。
  - ー LOCATE関数の第三引数は指定できません。
  - ー LENGTH関数に、入力パラメタを指定することはできません。
  - ー SUBSTRING関数の第一引数に、入力パラメタを指定することはできません。
  - ー NULL比較式に、入力パラメタを指定することはできません。

・ 使用するDBMSがPostgreSQLの場合、以下に注意してください。

  ― LOCATE関数の第三引数は指定できません。

  ― LIKE述語のエスケープ文字に、“¥”の指定はできません。

EJB QL基本

EJB QL基本ではSELECT句、FROM句、WHERE句を使用したデータ操作ができます。また、データを編集して返却する演算子が使用できます。
より詳細なEJB QLの使用方法については、EJB2.0の規約を参照してください。

以下に演算子を使用したEJB QL文の例を示します。

・ SELECT句

  ― 例 SELECT句-1) 単純なEJB QL文

  ― 例 SELECT句-2) パス式を使用したEJB QL文

・ WHERE句
FROM句で検索された値に対しWHERE句を使用して、いろいろな条件テストが指定できます。

  ― 例 WHERE句-1) リテラルを使用したEJB QL文

  ― 例 WHERE句-2) パラメタを使用したEJB QL文

  ― 例 WHERE句-3) 比較式を使用したEJB QL文

  ― 例 WHERE句-4) BETWEEN式を使用したEJB QL文

  ― 例 WHERE句-5) NULL比較式を使用したEJB QL文

例 SELECT句-1) 単純なEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
```

すべてのCustomerのCollectionを返します。
WHERE句がなく識別子が抽象スキーマ1つしか使われないもので、上記のEJB QLの例では、Customerという抽象スキーマから検索します。FROM句の後に抽象スキーマ名(Customer)を指定して、ASの後のcはCustomerをあらわす識別子です(ASは省略可)。
上記のようにSELECT型にEntity Beanの識別子だけが単独で指定されている場合は、OBJECT()演算子が必須です。

例 SELECT句-2) パス式を使用したEJB QL文

```
SELECT c.lastName FROM Customer AS c
```

全Customerの姓の、Collectionを返します。
Customerという抽象スキーマよりCustomerすべての姓を検索するEJB QLの記述です。この例では、OBJECT()演算子は不要です。
SELECT句はCMPフィールドまたはCMRフィールドも返却できます。

例 WHERE句-1) リテラルを使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.name = 'Fujitsu'
```

'Fujitsu'という名前のCustomerを返します。
WHERE句で指定されたリテラル(定数)に一致するBeanを検索するためのEJB QLの記述です。指定したリテラルが文字列リテラルの場合は、シングルクォーテーション(')で囲みます。上記のは、名前がFujitsuであるCustomerを検索するEJB QLの記述です。

留意事項

― EJB QLでは文字列リテラル、数値リテラルおよびboolean型のリテラルが指定できます。

ー 日時型など他のデータ型を指定する場合は、リテラルではなくパラメタを使用してください。

ー EJBアプリケーションのdeployment descriptorの情報が定義されているファイル(ejb-jar.xml)を直接編集する場合は、“ejb-jar.xmlを編集する場合”を参照して、XML形式の仕様に従った記述をしてください。

例 WHERE句-2) パラメタを使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.name = ?1
```

パラメタに指定された名前のCustomer Collectionを返します。
WHERE句の後に指定したパラメタに対して、疑問符(?)とパラメタが最初(1)から数えて何番目になるかを指定します。上記は、Customerという抽象スキーマからパラメタの名前に指定されたCustomerから検索するEJB QLの記述です。

例 WHERE句-3) 比較式を使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.age >= 25
```

WHERE句では比較式が使用できます。上記の例では比較式(>=)を使用して年齢が25歳以上のCustomerを検索するEJB QLの例を記述します。

留意事項

ー 左辺にリテラルおよびパラメタの指定はできません。

ー 以下のデータ型を比較する場合、左辺および右辺に指定するデータ型は同じ型を指定してください。

- java.lang.String
- charまたはjava.lang.Character
- booleanまたはjava.lang.Boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

ー 以下のデータ型は比較できません。

- byte配列
- Serializableオブジェクト

例 WHERE句-4) BETWEEN式を使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.id BETWEEN 100 and 200
```

idが100～200のCustomerのCollectionを返します。
BETWEEN式(NOT BETWEEN)の使用方法は、SQLと同様です。BETWEEN演算子の指定区間について検索できます。例ではIDが100～200のCustomerを検索するEJB QLです。

留意事項

ー 以下のデータ型を比較する場合、指定するデータ型は同じデータ型を指定してください。

- java.lang.String
- charまたはjava.lang.Character
- booleanまたはjava.lang.Boolean
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

ー 以下のデータ型は比較できません。

- byte配列
- Serializableオブジェクト

例 WHERE句-5) NULL比較式を使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.age IS NOT NULL
```

年齢情報があるCustomerのCollectionを返します。
NULL比較式演算子は、NULL値の検索に使用します。例えば年齢がNULL値ではないCustomerを検索するEJB QLは上記のように記述します。

EJB QL拡張

EJB QL拡張は、EJB2.0で使用できる機能に加えて以下が拡張されています。

- ORDER BY句の指定ができる
- 関数式MODが指定できる
- LIKE式のパターンに変数が指定できる
- LIKE式のエスケープ文字に変数が指定できる
- NULL比較式に変数が指定できる
- IN式に変数と数値が指定できる
- 集合関数の以下が指定できる
  - ー MIN、MAX、AVG、SUM、COUNT

より詳細なEJB QL拡張の使用方法については、EJB2.1の規約を参照してください。

以下に演算子を使用したEJB QL文の例を示します。

- 例1) ORDER BY句を使用した基本的なEJB QL文
- 例2) 集合関数を使用したEJB QL文
- 例3) 関数式MODを使用したEJB QL文
- 例4) LIKE式を使用したEJB QL文
- 例5) IN式を使用したEJB QL文

例1) ORDER BY句を使用した基本的なEJB QL文

```
SELECT c.address.city FROM Customer c
ORDER BY c.address.city DESC
```

ORDER BY句を使用して値式にCMFを指定すると、検索結果がソートされます。ORDER BY句には昇順「ASC」か、降順「DESC」を指定できます。省略された場合、昇順になります。
上記の例では、返却される値をc.address.cityの降順で返却します。

留意事項

ー 以下のデータ型のCMFは指定できません。

- booleanまたはjava.lang.Boolean
- byte配列
- Serializableオブジェクト

例2) 集合関数を使用したEJB QL文

```
SELECT SUM(c.employeeCount)
FROM Company c WHERE c.type = 'IT'
```

集合関数は、値の合計や平均などの集計を行う関数です。
上記の例では、c.typeが'IT'のものをc.employeeCountの値を合計して返却します。

留意事項

- 集合関数はejbSelectメソッドのクエリ文だけに指定可能であり、SELECT句の返却値としてだけ指定できます。
- 集合関数のMAXおよびMINに以下のデータ型は指定できません。
  - booleanまたはjava.lang.Boolean
  - byte配列
  - Serializableオブジェクト
- 集合関数のSUMおよびAVGに以下のデータ型は指定できません。
  - java.lang.String
  - charまたはjava.lang.Character
  - booleanまたはjava.lang.Boolean
  - java.sql.Date
  - java.sql.Time
  - java.sql.Timestamp
  - byte配列
  - Serializableオブジェクト

例3) 関数式MODを使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer c
WHERE MOD(c.age, 10) = 0
```

MODを使用すると剰余を得ることができます。年齢を10で割った余りを返却します。

留意事項

- 関数式MODに指定可能なデータ型は以下です。
  - byteまたはjava.lang.Byte
  - shortまたはjava.lang.Short
  - intまたはjava.lang.Integer

例4) LIKE式を使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.name LIKE ?1 ESCAPE ?2
```

LIKE式の使用方法はSQLと同様です。上記の例では、名前がパターン('Fujitsu_'から始まる)文字列に合致するCustomerを検索するEJB QLの例を記述します。
パターン文字列は'Fujitsu¥_'を指定します。また、エスケープ文字は'¥'を指定します。

留意事項

- パターン文字列に指定可能なデータ型は以下です。
  - java.lang.String
- エスケープ文字に指定可能なデータ型は以下です。
  - charまたはjava.lang.Character

例5) IN式を使用したEJB QL文

```
SELECT OBJECT(c) FROM Customer AS c
WHERE c.age IN (?1, ?2, ?3)
```

IN式の使用方法は、SQLと同様です。上記の例では年齢がパラメタで指定した値に合致するCustomerを検索するEJB QLの例を記述します。

留意事項

ー IN式に指定可能なデータ型は以下です。

- byteまたはjava.lang.Byte
- shortまたはjava.lang.Short
- intまたはjava.lang.Integer
- floatまたはjava.lang.Float
- doubleまたはjava.lang.Double
- java.lang.BigDecimal
- java.lang.String

ー IN式の左辺および右辺に指定するデータ型は、同じデータ型を指定してください。

10.3 Message-driven Beanの実行環境

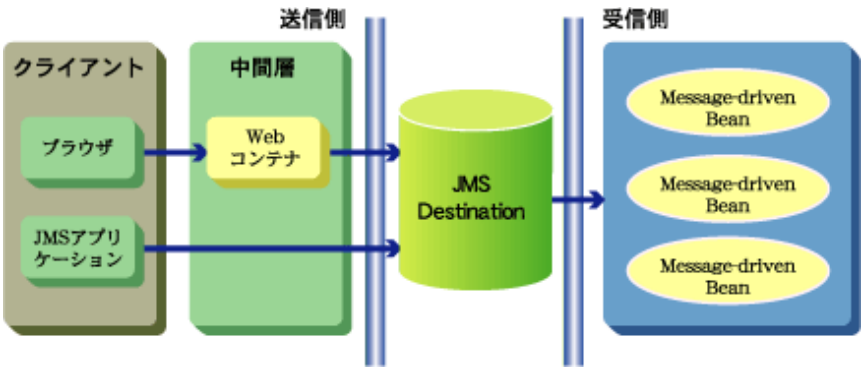
Message-driven Beanとは

ここでは、Message-driven Beanの実行環境について説明します。

EJB2.0規約から追加されたEnterprise JavaBeansが、Message-driven Beanです。
Message-driven Beanは、JMSメッセージもしくはリソースアダプタのメッセージを受信して処理するオブジェクトであり、受信メッセージをEJBアプリケーションで処理できます。

コンテナは、メッセージを受信してから“Message-driven Bean”のインスタンスにマッピングを行い、ビジネスロジックを実現します。
“Message-driven Bean”に記述されたメッセージリスナメソッド(JMSの場合はonMessageメソッド)でメッセージを処理します。
Message-driven Beanは、Session BeanまたはEntity Beanを呼び出すか、DBMSが提供するデータベース操作命令(JDBCドライバ)を直接呼び出してデータベース処理を行います。

以下にMessage-driven Beanの位置づけを示します。以下はJMSメッセージを受信する場合の構成図です。



Message-driven Beanはメッセージを受信する対象(これを受信対象種別と呼びます)としてJMSからメッセージを受信するか、リソースアダプタからメッセージを受信するか選択できます。

受信対象種別がJMSの場合

メッセージモデル

Message-driven Beanは、以下のJMSメッセージングモデルを実現しています。
それぞれのモデルには、以下の特長があります。

Point-To-Pointモデル

特定の受信者に対してメッセージを配信する1対1のメッセージングモデルです。

Publish/Subscribeモデル

複数の受信者に対して同一のメッセージを配信する1対nのメッセージングモデルです。

JMS DestinationとJMS ConnectionFactoryの定義

Message-driven Beanを動作させるには、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義] > [Message-driven Bean拡張情報]で“Destination名”と“JMSコネクションファクトリ”を定義してください。

各定義に何も指定しなかった場合は以下が有効になります。

定義名		デフォルト値
JMSコネクションファクトリ名	Topicの場合	TopicCF001
	Queueの場合	QueueCF001
Destination名		EJBアプリケーション名

受信対象種別がリソースアダプタの場合

Message-driven Beanを動作させるには、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義] > [Message-driven Bean拡張情報]で“リソースアダプタ名”を定義してください。
何も指定しなかった場合は以下が有効になります。

定義名	デフォルト値
リソースアダプタ名	EJBアプリケーション名

10.3.1 durable Subscription機能

- durable Subscriptionとは
- durable Subscriber定義の登録と削除

durable Subscriptionとは

本機能は受信対象種別がJMSの場合に有効な機能です。
Message-driven Beanが起動していない状態でDestinationに送信されたメッセージをJMSが保持し、Message-driven Beanが起動した後にメッセージを受信できる機能をdurable Subscription機能と呼びます。
この機能は、複数のメッセージ受信者に対して配信するPublish/Subscribeモデルで使用します。

本機能を使用する場合は、deployment descriptorのsubscription-durability(サブスクライバの永続性)に“**NonDurable**”または“**Durable**”を設定してください。deployment descriptorへの設定、および変更は、Interstage Studioで行います。EJB2.0以前のEJBアプリケーションの場合は、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義] > [Message-driven Bean拡張情報]の[サブスクライバの永続性]でも設定することができます。

NonDurable

Message-driven Beanの停止中にDestinationへ送信されたメッセージはMessage-driven Beanを再起動しても配信されません。

Durable

Message-driven Beanがメッセージを受信するまでメッセージが保持されます。Message-driven Beanの停止中にDestinationへ送信されたメッセージも、Message-driven Bean再起動時にメッセージが配信されます。

durable Subscriber定義の登録と削除

“サブスクライバの永続性”に“**Durable**”を指定した場合、“サブスクライバの識別名”を設定してください。IJServerはMessage-driven Beanの初回起動時に“サブスクライバの識別名”に設定された名前でdurable Subscriberを登録します。
“サブスクライバの識別名”はInterstage管理コンソールで設定します。設定方法の詳細は、Interstage管理コンソールのヘルプを参照してください。
1度登録されたdurable SubscriberはMessage-driven Beanが停止しても存在し、メッセージを保持します。保持されたメッセージは再度Message-driven Beanが起動されたときに配信されます。

durable Subscriberが必要なくなった場合は、以下のコマンドにより削除してください。コマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“JMS運用コマンド”を参照してください。

例

サブスクライバの識別名が“dsub”、クライアント識別子が“client1”というdurable Subscriberを削除する場合

```
jmsrmds -n dsub -i client1
```

10.3.2 メッセージ・セレクタ機能

本機能は受信対象種別がJMSの場合に有効な機能です。
メッセージ・セレクタ機能とは、Message-driven Beanで処理するメッセージの条件を指定できる機能です。
この機能を使用するには、deployment descriptorのmessage-selector(メッセージ セレクタ定義)にSQL文を指定します。
Message-driven Beanには、メッセージ・セレクタに定義されたSQL文の条件に合致するメッセージだけが配信されます。
記述する条件式の詳細は、“24.10.1 メッセージセレクタ条件式”を参照してください。
deployment descriptorへの設定、および変更は、Interstage Studioで行います。EJB2.0以前のEJBアプリケーションの場合はInterstage管理コンソールでも行うことができます。

例

以下に、JMSTypeが'car'で、かつcolorが'blue'のメッセージを受信する場合のメッセージ・セレクタ定義の例を示します。

```
JMSType = 'car' AND color = 'blue'
```

10.3.3 プロセス多重度のサポート

- 複数メッセージの受信処理性能を向上させる方法
- プロセス起動多重
- スレッド多重でのメッセージ受信
- 利用シーン

複数メッセージの受信処理性能を向上させる方法

トランザクション連携を行う非同期メッセージングモデルの場合に、一般的に受信アプリケーションが受信メッセージの通番管理を行うために従来では未サポートだった、Message-driven Beanを利用したトランザクションと連携した同時多重スレッドでのメッセージ受信についてサポートします。

受信対象種別がリソースアダプタの場合、もしくは受信対象種別がJMSでPoint-To-Pointメッセージングモデルの場合に、以下の機能が使用できます。

- プロセス起動多重
  Message-driven Beanをプロセス多重で起動します
- スレッド多重でのメッセージ受信
  トランザクションと連携した同時多重スレッドでのメッセージ受信ができます

上記により、トランザクションと連携してメッセージ受信を行う以下のMessage-driven Beanの、複数メッセージ受信処理性能が向上します。

- トランザクション管理種別にBeanを指定したMessage-driven Bean
- トランザクション属性にContainerを指定、およびトランザクション属性にRequiredを指定したMessage-driven Bean

プロセス起動多重

以下の表に、受信対象種別がJMSの場合のメッセージングモデルとトランザクション種別におけるプロセス起動多重の使用可否を説明します。受信対象種別がリソースアダプタの場合には特に制約はありません。

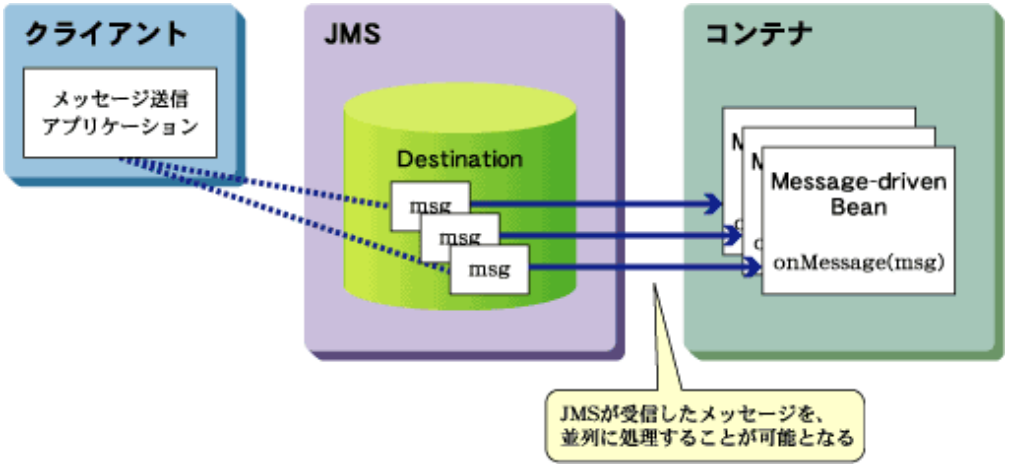
メッセージングモデル	トランザクション種別	使用可否
Point-To-Point	分散トランザクションを使用しない	○
	Windows32/64 Linux32/64 分散トランザクションを使用する (注)	―
Publish/Subscribe	分散トランザクションを使用しない	×
	Windows32/64 Linux32/64 分散トランザクションを使用する (注)	―

注) IJServerのプロセス多重を行うことはできません。

プロセス多重度の設定

Interstage管理コンソールの[システム] > [ワークユニット]からIJServerを選択し、環境設定画面のワークユニット設定でプロセス多重度を設定してください。
isj2eeadminコマンドを使用して、変更することもできます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

スレッド多重でのメッセージ受信



以下の表にトランザクション管理種別と、トランザクション属性における使用可否を説明します。受信対象種別がリソースアダプタの場合には特に制約はありません。

トランザクション管理種別	トランザクション属性	Point-To-Point	Publish/Subscribe
Bean	－	○	×
Container	NotSupported	○	○
	Required	○	×

使用できないアプリケーションは以下で説明する初期起動インスタンス数の設定に関係なく、初期起動インスタンス数は1で動作し、1メッセージずつ処理されます。

同時実行スレッド数の設定

メッセージの受信処理を同時に行う最大数を[ワークユニット]＞[IJServer名]＞[EJBアプリケーション]＞[アプリケーション環境定義]の[初期起動インスタンス数]の項目に設定します。詳細は、Interstage管理コンソールのヘルプを参照してください。

初期起動インスタンス数の設定

Interstage管理コンソールの[EJBアプリケーション環境定義]＞[Message-driven Bean拡張情報]で、初期起動インスタンス数を設定してください。
初期起動インスタンス数は、以下の範囲で設定してください。

項目	値
初期設定値	8
最大値	10000000
最小値	1

利用シーン

以下に該当する場合で複数メッセージの受信処理性能を向上させたいときは、プロセス多重度または初期起動インスタンス数(同時実行スレッド数)を設定してください。

- 受信メッセージの通番管理を行わないシステム
- 受信メッセージの通番管理は行うが、通番通りのメッセージ受信の順序性を保証しないシステム

受信したメッセージを同時に受信処理したい場合には、同時に受信するメッセージの最大数を見積もってプロセス多重度、および初期起動インスタンス数(同時実行スレッド数)を設定してください。また、必要に応じてメモリ・CPUを増設してください。

10.3.4 異常時のメッセージ退避機能

異常時のメッセージ退避機能とは

本機能は受信対象種別がJMSの場合に有効な機能です。
トランザクション管理種別がContainerで、かつ、トランザクション属性がRequiredの場合、Message-driven Bean内でjava.lang.RuntimeExceptionやjava.lang.Error、またはそれらのサブクラスのシステム例外が発生すると、コンテナはトランザクションをロールバックします。
この場合、ロールバックされたメッセージは再度Message-driven Beanに送信され、処理がループする場合があります。
この現象を回避するために、異常時のメッセージ退避機能を使用してください。

Message-driven Bean内でjava.lang.RuntimeExceptionやjava.lang.Error、またはそれらのサブクラスのシステム例外がリトライ回数＋1回連続して発生すると、コンテナがメッセージをバックアップ用のDestinationへ送信します。
異常時ループ対処用のJMSコネクションファクトリ名とDestination名が指定されていない場合や、指定が間違えているなどの

場合には、メッセージはシリアライズされ、シリアライズファイルが作成されます。
シリアライズに失敗した場合は、プロセスを停止します。

設定方法

[異常時メッセージ退避定義]の“リトライカウント”、“JMSコネクションファクトリ名”、“Destination名”は、Interstage管理コンソールまたはejbdefimportコマンドで設定します。
設定方法の詳細は、Interstage管理コンソールのヘルプ、または、“第16章 運用コマンドを使用してカスタマイズする方法”を参照してください。

シリアライズファイル

以下に、シリアライズファイルのファイル名と格納先を示します。

シリアライズファイルのファイル名

MSG_[EJBアプリケーション名]_[プロセスID]_[スレッドID]_[数字].ser

シリアライズファイルの格納先

Windows32/64

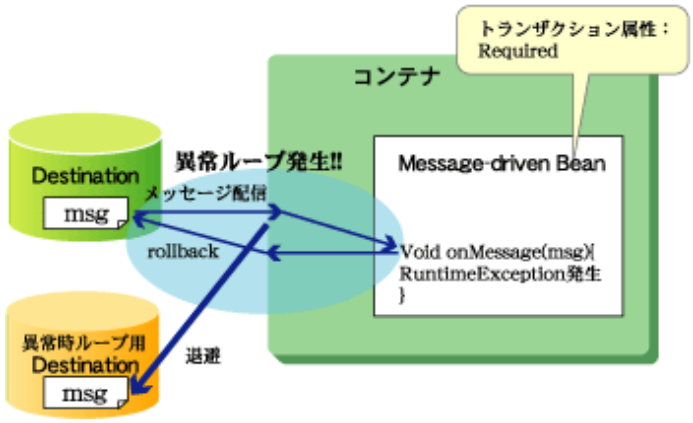
C:¥Interstage¥EJB¥var

Solaris64 Linux32/64

/opt/FJSVejb/var

処理概要

以下に、この機能を使用した場合の処理イメージを示します。



例

シリアライズファイルの復元方法

シリアライズされたJMSメッセージを復元するための記述例を示します。

```
FileInputStream fis = new FileInputStream("シリアライズファイル名");
try {
  ObjectInputStream ois = new ObjectInputStream(fis);
  Message msg = (Message)ois.readObject();   //このmsgが退避されたメッセージです。
} finally {
  fis.close();
}
```

注意

- 以下の場合には、異常が発生したメッセージではないメッセージが退避の対象となる可能性がありますので、使用しないでください。
  - Point-To-Pointモデルモデルを使用し、Message-driven BeanのDestinationに他の受信アプリケーションが存在する場合
  - JMSメッセージに以下のJMSヘッダーフィールドが指定された場合
    - JMSPriority
    - JMSExpiration
- 以下に該当する場合、リトライ回数を超過してもメッセージ受信を繰り返す可能性があります。
  - Point-To-Pointモデルメッセージングモデル
  - 不揮発化チャネルを利用

イベントサービスの設定について

イベントチャネルのイベントデータをメモリにキャッシュする数は、以下の式が成立するように設定してください。

```
イベントチャネルのイベントデータをメモリにキャッシュする数　＞
　　イベントチャネルに蓄積できるイベントデータの最大値
```

設定はイベントサービス運用コマンドを使用して行います。詳細は“リファレンスマニュアル(コマンド編)”の“essetcnf”および“essetcnfchnl”を参照してください。

10.4 EJBサービスのトランザクション制御

EJBサービスが提供するトランザクション制御を使用すると、トランザクション管理種別やトランザクション属性を定義することにより、トランザクションの制御方法や実行時の制御内容などを詳細に定義できます。

また、javax.ejb.SessionSynchronizationインタフェースを使用して、インスタンスが参加したトランザクションの開始と終了を受信できる機能があります。

この機能を“**Session Beanのsynchronization機能**”といい、STATEFULのSession Beanで、特定のトランザクション属性をビジネスメソッドに指定している場合に限り使用できます。

Windows32/64 Linux32/64
トランザクション機能などを使用し、複数のトランザクション処理を行っている場合、データベース連携サービス(Object Transaction Service)と連携した“**分散トランザクション機能**”を使用することにより、以下のことが実現します。

- 複数のトランザクション処理を1つのトランザクションとして扱う
- 複数のリソースマネージャ(DBMS、JMSなど)へのアクセス処理のトランザクション制御を行う
- 複数のJava VM間で動作するEJBアプリケーションを1つのトランザクションで使用する

10.4.1 トランザクション管理種別とトランザクション属性

ここでは、以下について説明します。

- トランザクション管理種別
- トランザクション属性
- トランザクション属性を指定するときの注意事項
- EJBサービスが提供するトランザクション機能を使用する場合の注意事項

トランザクションの制御方法や実行時の制御内容などを詳細に決定するために、トランザクション管理種別とトランザクション属性を定義します。

トランザクション管理種別

トランザクション管理種別とは、EJBアプリケーションのトランザクションを制御するための定義を行うもので、EJBアプリケーションごとに定義します。
トランザクション管理種別には、“**Bean**”と“**Container**”の2つがあります。以下に、各トランザクション管理種別の詳細を示します。

トランザクション管理種別	内容
Bean	UserTransactionインタフェースを使用して、EJBアプリケーションがトランザクションの制御を行います。 この管理種別を指定できるEJBアプリケーションは、Session BeanとMessage-driven Beanです。 Session Beanでは、呼出し元でトランザクションが開始されている場合は、コンテナがそのトランザクションを中断します。処理が終了した後、中断していた呼出し元のトランザクションを再開します。 Message-driven Beanでは、メッセージの受信はトランザクションに含まれません。受信対象種別がJMSの場合、Destinationに送信されたメッセージはonMessageメソッドが終了した時点でJMSが自動的に受信確認(acknowledge)を行い、メッセージの受信を完了します。
Container	トランザクション制御はコンテナが行います。 EJBアプリケーションにトランザクション処理の記述をしなくてよいため、EJBアプリケーションのポータビリティが向上し、最小単位の部品としてEJBアプリケーションを作成できます。 この管理種別を指定した場合、実行時のトランザクション制御内容を指定するために、“トランザクション属性”を指定してください。

注意

トランザクション管理種別がContainerの場合、javax.transaction.UserTransactionインタフェースを使用してトランザクションを制御しようとすると、IllegalStateExceptionが発生します。

トランザクション属性

トランザクション属性とは、実行時にコンテナが行うトランザクション制御の内容を指定するもので、トランザクション管理種別に“Container”を指定した場合に指定します。
EJBアプリケーションごと、または、EJBアプリケーションのメソッド単位で指定します。
以下に、各トランザクション属性の詳細を示します。

トランザクション属性	内容	Session Bean	Entity Bean	Message-driven Bean
Mandatory	この属性を指定したEJBアプリケーション(メソッド)は、常に呼出し元で開始されているトランザクションで実行されます。 ・ 呼出し元でトランザクションが開始されている場合 呼出し元のトランザクションで実行されます。 ・ 呼出し元でトランザクションが開始されていない場合 呼出し元にTransactionRequiredExceptionが返却されます。	○	○	－
Required	この属性を指定したEJBアプリケーション(メソッド)は、常にトランザクションが開始された状態で実行されます。呼出し元でトランザクションが開始されている場合と開始されていない場合で、トランザクションが異なります。	○	○	○

トランザクション属性	内容	Session Bean	Entity Bean	Message-driven Bean
	・ 呼出し元のトランザクションで開始されている場合 呼出し元のトランザクションで実行されます。 ・ 呼出し元でトランザクションが開始されていない場合 コンテナがトランザクションを開始し、そのトランザクションで実行されます。開始されたトランザクションはメソッド実行完了後、コンテナにより終了(commit/rollback)されます。 通常、メソッド実行完了後、トランザクションがcommitされます。当属性が指定されたBean(メソッド)内でトランザクションをrollbackする必要がある場合、SessionContext/EntityContext/MessageDrivenContextインタフェースのsetRollbackOnlyメソッドを使用してトランザクションをrollbackするようマーク(宣言)できます。 一般的に、Entity Beanでこの属性を使用すると、可搬性の高いEJBアプリケーションを作成できます。 Message-driven Beanの場合、メッセージの受信もトランザクションに含まれます。Message-driven Beanにメッセージが配信されると、コンテナがトランザクションを開始してからメッセージリスナメソッドにメッセージを渡します。 注) Message-driven Beanでこの属性を指定して、無条件にsetRollbackOnlyメソッドを使用してメッセージ受信をロールバックすると、メッセージの配信が繰り返されてループする可能性があります。処理を継続できないようなエラーが発生した場合には、setRollbackOnlyメソッドを実行するのではなく、EJBExceptionをthrowするなどして異常時のメッセージ退避機能を使用してください。			
Supports	この属性を指定したEJBアプリケーション(メソッド)は、呼出し元でトランザクションが開始されている場合と開始されていない場合で、実行される状態が異なります。 ・ 呼出し元のトランザクションで開始されている場合 呼出し元のトランザクションで実行されます。 ・ 呼出し元でトランザクションが開始されていない場合 トランザクションが開始されていない状態のまま処理が実行されます。 この属性は、トランザクションの一貫性・整合性が保たれない可能性があることから、EJB規約では推奨していません。	○	○	－
RequiresNew	この属性を指定したEJBアプリケーション(メソッド)は、常にコンテナで開始されたトランザクションで実行されます。 ・ 呼出し元のトランザクションで開始されている場合 コンテナが、呼出し元のトランザクションを中断します。その後、コンテナで開始されたトランザクションで実行されます。開始されたトランザクションは、メソッド実行完了後、コンテナにより終了(commit/rollback)されます。処理が終了した後、コンテナは、中断していた呼出し元のトランザクションを再開します。通常、メソッド実行完了後、トランザクションがcommitされます。当属性が指定されたBean(メソッド)内でトランザクションをrollbackする必要がある場合、SessionContext/EntityContextインタフェースのsetRollbackOnlyメソッドを使用してトランザクションをrollbackするようマーク(宣言)できます。	○	○	－

トランザクション属性	内容	Session Bean	Entity Bean	Message-driven Bean
	・ 呼出し元でトランザクションが開始されていない場合 コンテナがトランザクションを開始し、そのトランザクションで実行されます。開始されたトランザクションは、メソッド実行完了後、コンテナにより終了(commit/rollback)されます。通常、メソッド実行完了後、トランザクションがcommitされます。当属性が指定されたBean(メソッド)内でトランザクションをrollbackする必要がある場合、SessionContext/EntityContextインタフェースのsetRollbackOnlyメソッドを使用してトランザクションをrollbackするようマーク(宣言)できます。 この属性を指定すると、トランザクション排他処理の範囲を縮小してデータベースの占有を防止できます。			
NotSupported	この属性を指定したEJBアプリケーション(メソッド)は、常にトランザクションが開始されていない状態で実行されます。 ・ 呼出し元のトランザクションで開始されている場合 コンテナが、そのトランザクションを中断します。処理が終了した後、コンテナは、中断していた呼出し元のトランザクションを再開します。 ・ 呼出し元でトランザクションが開始されていない場合 トランザクションが開始されていない状態のまま処理が実行されます。 Message-driven Beanの場合、メッセージが配信された時点でコンテナがメッセージの受信を完了します。	○	○	○
Never	この属性を指定したEJBアプリケーション(メソッド)は、常にトランザクションが開始されていない状態で実行されます。 ・ 呼出し元のトランザクションで開始されている場合 呼出し元に例外が返却されます。 ・ 呼出し元でトランザクションが開始されていない場合 トランザクションが開始されていない状態のまま処理が実行されます。	○	○	ー

○:サポート　ー:用途なし

呼出し先のBeanに指定するトランザクション属性により、呼出し先のBeanと呼出し元のトランザクションの制御状態は以下のようになります。

呼出し先のBeanに指定したトランザクション属性	呼出し元でのトランザクションの状態	呼出し先でのトランザクションの状態
NotSupported	none	none
	T1	none
Required	none	T2
	T1	T1
Supports	none	none
	T1	T1
RequiresNew	none	T2

呼出し先のBeanに指定したトランザクション属性	呼出し元でのトランザクションの状態	呼出し先でのトランザクションの状態
	T1	T2
Mandatory	none	error
	T1	T1
Never	none	none
	T1	error

上記の状態の意味を下表に示します。

状態	意味
none	トランザクションが開始されていない状態
T1	呼出し元で開始したトランザクションで動作している状態
T2	呼出し先で開始したトランザクションで動作している状態
error	エラーが発生

Mandatory、RequiredおよびSupportsを使用して、呼出し元のトランザクション(T1)で呼出し先のBeanを動作させる場合、以下の機能を使用してください。以下の機能を使用しない場合には、呼出し元のトランザクションは開始されていない状態(none)と判断して動作します。

- 分散トランザクションを使用しない
  呼出し元のEJBアプリケーションと、呼出し先のEJBアプリケーションを同一JavaVM上で動作させる。
- 分散トランザクションを使用する Windows32/64 Linux32/64
  呼出し元が、クライアントアプリケーションまたはJava VM外のEJBアプリケーションの場合、分散トランザクションを使用してトランザクションを連携する。

トランザクション属性を指定するときの注意事項

トランザクション属性を指定するときは、以下の点に注意してください。

- トランザクション属性をメソッド単位で指定する場合、指定できるメソッドは限られています。以下に、トランザクション属性が指定できるメソッドを示します。

Bean種別	Homeインタフェースメソッド	Remoteインタフェースメソッド	LocalHomeインタフェースメソッド	Localインタフェースメソッド
Session Bean	なし	すべてのビジネスメソッド	すべてのビジネスメソッド	すべてのビジネスメソッド
Entity Bean	create() findByPrimaryKey() finderメソッド ejbHomeメソッド remove(java.lang.Object) remove(javax.ejb.Handle)	すべてのビジネスメソッド remove()	create() findByPrimaryKey() finderメソッド ejbHomeメソッド remove(java.lang.Object)	すべてのビジネスメソッド remove()

- Message-driven Beanは、メッセージリスナメソッドにトランザクション属性を設定するか、またはEJBアプリケーション(Bean単位)でトランザクション属性を指定してください。
- EJBアプリケーション(Bean単位)でトランザクション属性を指定し、そのEJBアプリケーションの各メソッドすべてに対してトランザクション属性を指定した場合、メソッドの種類によっては重複した設定となります。このような場合はメソッドへの指定が優先されます。

- 例外が発生した場合、呼出し元に例外の通知を行います。例外発生時の詳細は、“15.5.2 EJBサービスが提供するトランザクション制御の例外処理”を参照してください。
- EJB規約により、トランザクション種別に“Container”が指定されたSTATEFULのSession Beanのインスタンスが1度に参加できるトランザクションは1つだけです。呼出し元で開始しているトランザクションに参加しているインスタンスに対して、“NotSupported”または、“RequiresNew”のどちらかのトランザクション属性が指定されたメソッドが呼び出された場合、java.rmi.RemoteExceptionが呼出し元へ返却されます。
- トランザクション属性が指定されていないメソッドが存在する場合、コンテナは自動的に“Required”が指定されたものとしてトランザクションの制御を行います。
- トランザクション管理種別に“Container”、トランザクション属性にMandatory/Required/Supportsを指定したSTATEFUL Session Beanを使用するEJBアプリケーションを開発する場合、STATEFUL Session Beanの開放(remove)について以下の点に注意してください。
  - STATEFUL Session Beanを呼び出すEJBアプリケーションでトランザクションの操作を行う場合、STATEFUL Session Beanの開放処理(removeメソッド)は、必ずビジネスメソッド実行時に開始されているトランザクション処理が、完了したあとに行ってください。トランザクション処理中に開放処理(removeメソッド)を行った場合は、以下のエラーが発生します。
    1) javax.ejb.RemoveExceptionが呼び出し元に返却される
    2) イベントログ、または、システムログにエラーメッセージ“**IJServer: エラー: IJServer21061: トランザクションが開始されています**”が出力される

EJBサービスが提供するトランザクション機能を使用する場合の注意事項

EJBサービスが提供するトランザクション機能を使用する場合、以下の点に注意してください。

- 分散トランザクションを使用しない場合、javax.transaction.UserTransactionインタフェースでJMSのトランザクション制御を行うことはできません。
  Windows32/64 Linux32/64
  JMSのトランザクション制御を行う場合は、分散トランザクション機能を使用して、javax.transaction.UserTransactionインタフェースでJMSのトランザクションを制御するか、または、JMSのトランザクション機能を使用してJMSの制御を行ってください。
- javax.transaction.UserTransactionインタフェースで、JDBCのトランザクション制御を行う場合、UserTransaction.begin()メソッド発行後、複数のデータソースにアクセスすると、データソースごとに別々のトランザクションとなります。
  この状態でUserTransaction.commit()メソッドを発行すると、関連するすべてのトランザクションが順番にコミットされます。
  コミットは、コンテナによって1フェーズコミットメントプロトコルで処理されるため、コミットされるトランザクションとロールバックされるトランザクションが混在する可能性があります。
  このため、javax.transaction.UserTransactionインタフェースのbegin()メソッドからcommit()メソッドの間で、更新するデータベースを1つのデータソースに限定するようにしてください。
  データソースは、Interstage管理コンソールまたはisj2eeadminコマンドで設定します。
  Windows32/64 Linux32/64
  分散トランザクション機能を使用することにより、複数のトランザクション処理を1つのトランザクションとして扱うことができます。

Windows32/64 Linux32/64
分散トランザクションの指定はIJServerの定義時に指定します。
指定方法について、Interstage管理コンソールを使用する場合は、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドを使用する場合は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

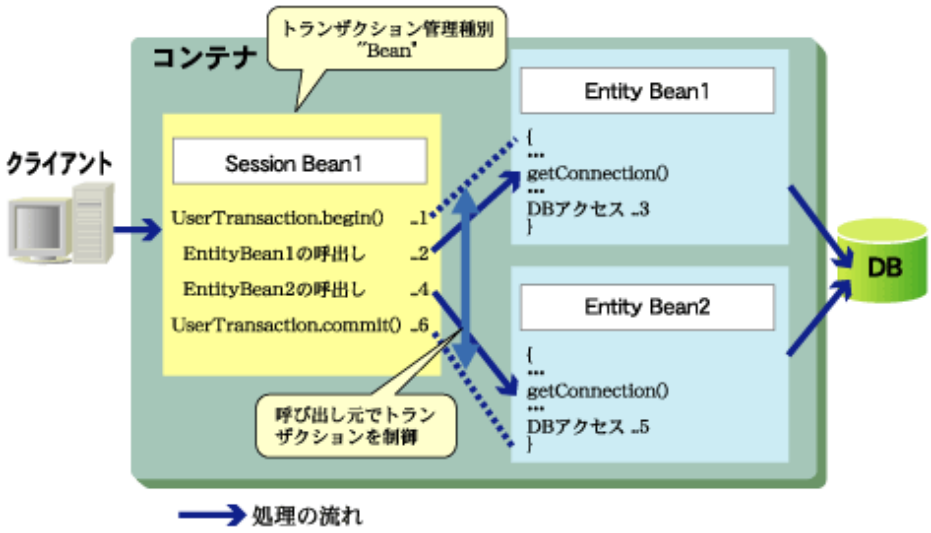
“分散トランザクション機能”についての詳細は、“第5部 JTS/JTA編 ”を参照してください。

10.4.2 各トランザクション管理種別と各トランザクション属性の制御例

以下に、トランザクション管理種別を指定した場合の制御例を示します。

- トランザクション管理種別に“Bean”を指定した場合の制御例
- トランザクション管理種別に“Container”を指定した場合の制御例

トランザクション管理種別に“Bean”を指定した場合の制御例



図の説明

1. Session Bean1は、UserTransaction.begin()メソッドを発行し、トランザクションを開始します。
2. Session Bean1は、Entity Bean1を呼び出します。
3. Entity Bean1がDBに対して処理を行います。
4. Session Bean1は、Entity Bean2を呼び出します。
5. Entity Bean2がDBに対して処理を行います。
6. Session Bean1は、UserTransaction.commit()メソッドを発行し、トランザクションをコミットします。このときコンテナは、高速に呼び出されるBeanがアクセスしたDBに対してコミットを発行します。

トランザクション管理種別に“Container”を指定した場合の制御例

トランザクション管理種別に“Container”を指定した場合、トランザクション属性によって、制御方法は異なります。

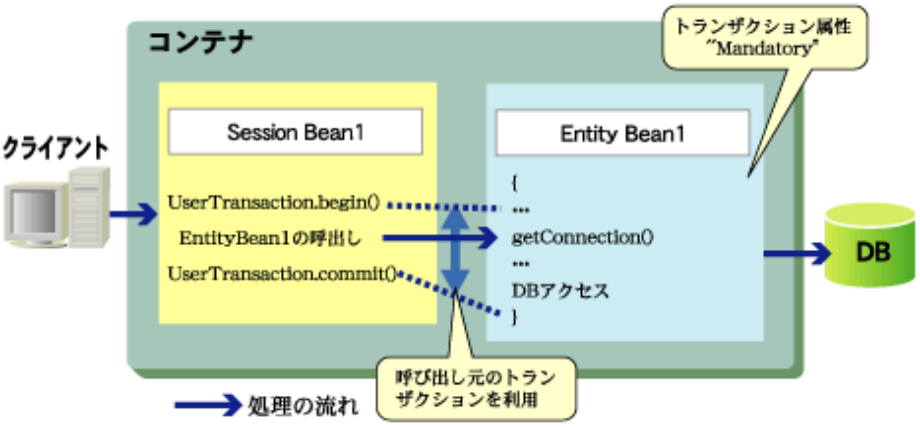
以下に、各トランザクション属性を指定した場合の制御例を示します。

- “Mandatory”を指定した場合
- “Required”を指定した場合
- Message-driven Beanで“Required”を指定した場合
- “RequiresNew”を指定した場合
- “NotSupported”を指定した場合
- “Never”を指定した場合

Supportsは、推奨されない属性であるため省略します。

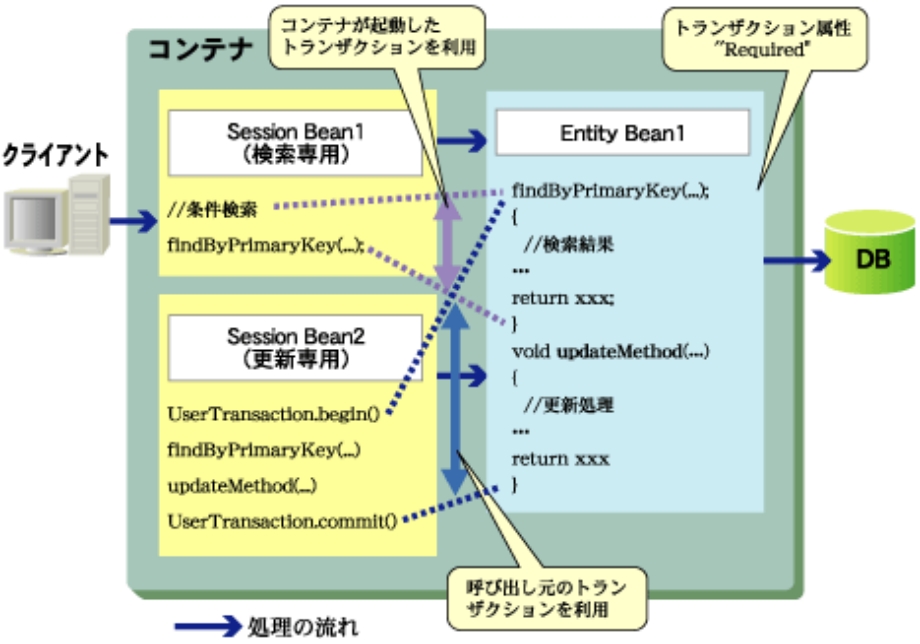
“Mandatory”を指定した場合の制御例

呼出し元で開始されたトランザクションで動作させるEJBアプリケーション(メソッド)に指定します。この属性を使用し、呼出し元でトランザクションが開始されていない状態でEJBアプリケーション(メソッド)が呼び出された場合、例外が発生します。



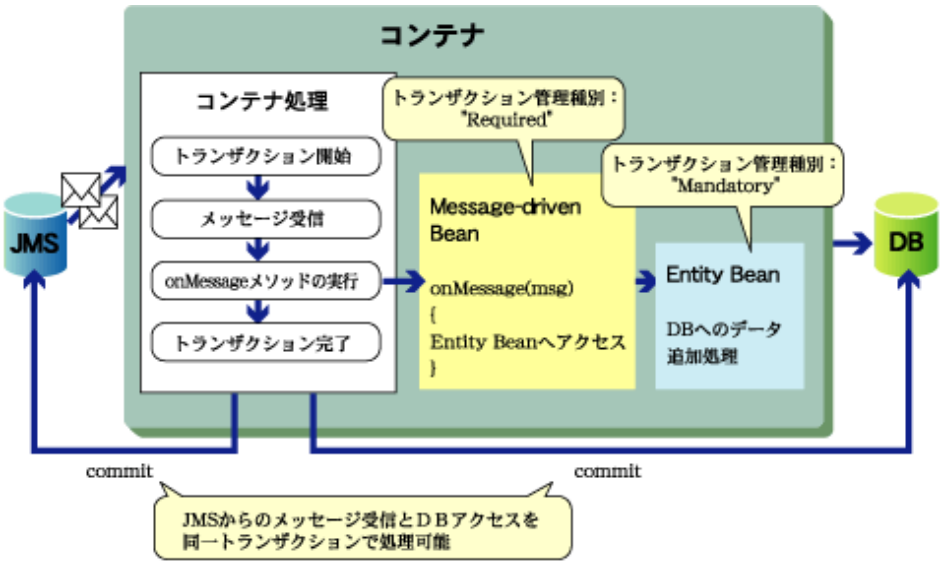
“Required”を指定した場合の制御例

呼出し元でのトランザクションの動作の状態に影響をうけることなく、常にトランザクションが開始されている状態で処理を行うことができます。
Message-driven Beanで“Required”を指定した場合の制御例は、“Message-driven Beanで“Required”を指定した場合の制御例”を参照してください。



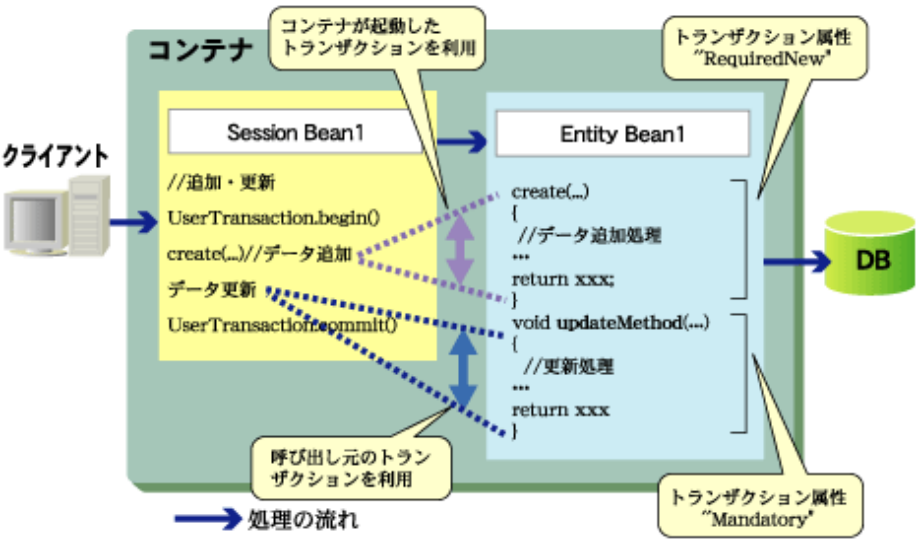
Message-driven Beanで“Required”を指定した場合の制御例

JMSからメッセージを受信して、受信したメッセージの情報をDBに格納する場合の制御例を示します。受信したメッセージとDBアクセスを同一トランザクション内で処理するため、Message-driven Beanのトランザクション種別は“Container”、トランザクション属性には“Required”を指定します。



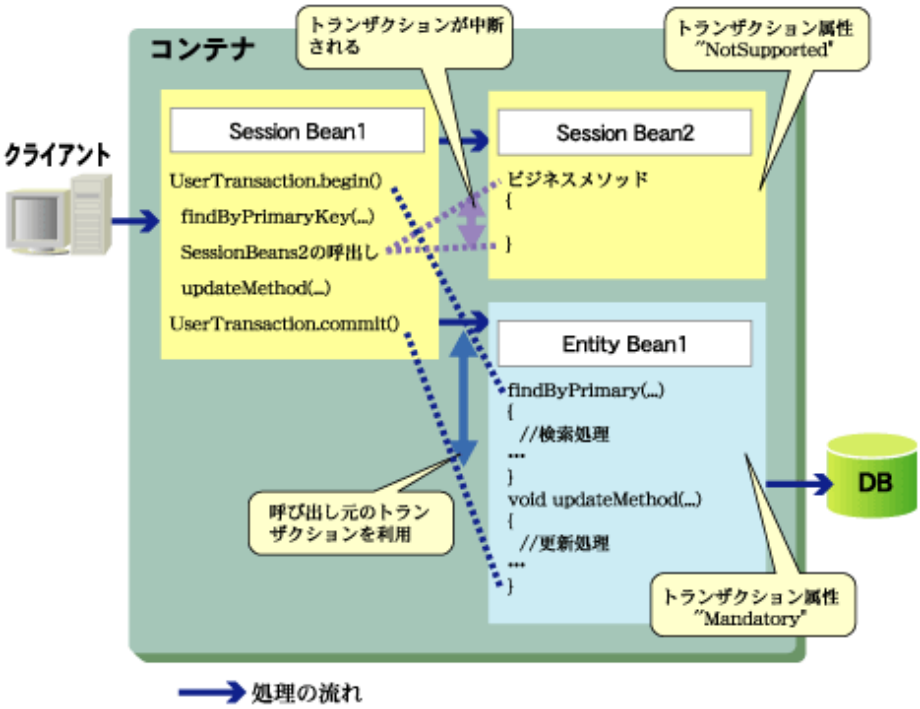
“RequiresNew”を指定した場合の制御例

呼出し元で開始されたトランザクションと異なるトランザクションで実行することができます。このため、呼出し元のトランザクション処理とは異なるトランザクション処理を行うことができます。



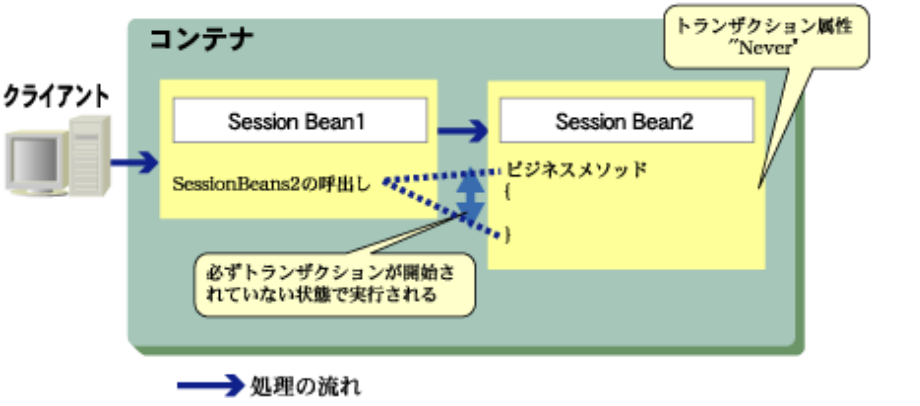
“NotSupported”を指定した場合の制御例

呼出し元でのトランザクションの動作の状態に影響されることなく処理が行えます。EJBサービスが提供するトランザクション機能を使用しない場合に指定します。



“Never”を指定した場合の制御例

呼出し元のトランザクションから影響をうけることなく処理が行えるため、EJBサービスが提供するトランザクション機能を使用しない場合に指定します。この属性を使用すると、呼出し元でトランザクションが開始された状態で呼び出されることを防ぐことができます(呼び出された場合は、例外が発生します)。



10.4.3 トランザクション管理種別と属性の設定方法

トランザクション管理種別と属性は、EJBアプリケーションの開発時にdeployment descriptorファイルに設定します。

Interstage Studioを使用して開発を行う場合、deployment descriptorファイルの設定はXMLエディタで行います。
設定方法の詳細については、“Interstage Studio ユーザーズガイド”を参照してください。
EJB2.0以前のEJBアプリケーションの場合、トランザクション管理種別と属性は、Interstage管理コンソールを使用して設定することもできます。詳細は、Interstage管理コンソールのヘルプを参照してください。

10.4.4 Session Beanのsynchronization機能

Session Beanのsynchronization機能とは、javax.ejb.SessionSynchronizationインタフェースを使用して、インスタンスが参加したトランザクションの開始と終了を受信できる機能です。
この機能は、STATEFULのSession Beanで、トランザクション属性(Mandatory、Required、RequiresNewのいずれか)をビジネスメソッドに指定している場合に限り使用できます。

トランザクション属性	内容
Mandatory	呼出し元が開始したトランザクションの開始と終了を受信します。
Required	呼出し元、または、コンテナが開始したトランザクションの開始と終了を受信します。
RequiresNew	コンテナが開始したトランザクションの開始と終了を受信します。

注意

Session Beanのsynchronization機能使用時に例外が発生した場合、ビジネスメソッドの呼出し元に例外の通知を行います。例外発生時の詳細は、“15.5.2 EJBサービスが提供するトランザクション制御の例外処理”を参照してください。

10.5 EJBサービスで使用できる時間監視機能

EJBサービス時間監視機能について

EJBサービスでは、以下の時間監視機能が使用できます。

- アプリケーションの最大処理時間の時間監視機能
- クライアントにサーバメソッドが復帰するまでの待機時間の監視機能
- STATEFUL Session BeanのEJBオブジェクト無通信監視機能
- Entity Bean EJB objectのタイマ削除機能
- EJBタイマーサービス

以下の表で、各機能の違いを説明します。

機能	監視対象	説明
アプリケーションの最大処理時間の時間監視機能	EJBアプリケーションメソッド実行時間(サーバ処理時間)を監視します。監視時間を超過した場合、アプリケーションが存在するプロセスを強制停止するかしないかを選択できます。	EJBアプリケーション処理中に、ハングや処理遅延が発生したことを検知できます。
クライアントにサーバメソッドが復帰するまでの待機時間の監視	クライアントからサーバへのリクエスト送信が一定時間超過しても返信がない場合、クライアントにタイムアウトが通知されます。	EJBアプリケーションプロセスの処理中に、ハングアップなどの現象を防止できます。
STATEFUL Session BeanのEJBオブジェクト無通信監視機能	STATEFUL Session BeanのEJB objectに対して、一定時間超過してもビジネスメソッドが実行されなかった場合、コンテナが該当のインスタンスと対応するEJB objectを削除します。	ejbRemoveメソッドを発行し忘れている、不要なEJB objectが削除されるため、使用するメモリの最適化ができます。

機能	監視対象	説明
Entity Bean EJB objectのタイマ削除機能	Entity Beanをプロセス外から呼び出した場合、createまたはfinderメソッドにより生成されたEJB objectのインスタンスは、removeメソッドが発行されない限りメモリ上に残存してしまいます。 そのため、EJB objectへの最終アクセスから一定時間が超過した場合、自動的にEJB objectを削除します。	プロセス外からEntity Beanを呼び出す場合に、残存したEJB objectを自動的に削除することにより、不当なメモリの増加を抑止できます。
EJBタイマーサービス	EJBアプリケーション内で任意の監視対象を監視することができます。	アプリケーション内でタイマーを作成し、任意の時刻にEJBコンテナからコールバック処理を実行させることができます。

各機能のタイムアウト値の設定内容

時間監視の各機能の設定値は以下のとおりです。また、EJBタイマーサービスについては他の時間監視機能と異なり、EJBアプリケーション内でタイマーを作成します。タイマーの作成方法については、“10.6 EJBタイマーサービス”を参照してください。

機能	設定場所	デフォルト値 最大値 最小値	備考
アプリケーションの最大処理時間の時間監視機能	アプリケーションの最大処理時間および、最大処理時間を超過した場合にアプリケーションが存在するプロセス強制停止を行うか行わないかの設定は、Interstage管理コンソールで行います。詳細は、Interstage管理コンソールのヘルプを参照してください。 また、isj2eeadminコマンドでもIJServerの設定できます。詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。	0 86400 0	0を設定すると時間監視を行いません。設定する数値の単位は秒です。
クライアントにサーバメソッドが復帰するまでの待機時間の監視	CORBAサービスの動作環境ファイル(config)のperiod_receive_timeoutに待機時間を設定します。詳細は、“チューニングガイド”の“CORBAサービスの動作環境ファイル”を参照してください。	72(360秒) 20000000 0	0を設定すると時間監視を行いません。設定する数値の単位は秒です。設定した値に、5を乗じた値が実際の設定値になります。
STATEFUL Session BeanのEJBオブジェクト無通信監視機能	Interstage管理コンソールのSTATEFUL Session Beanのアプリケーション環境定義画面で設定します。詳細は、Interstage管理コンソールのヘルプを参照してください。	1800 2147483647 0	0を設定すると時間監視を行いません。設定する数値の単位は秒です。
Entity Bean EJB objectのタイマ削除機能	Interstage管理コンソールのEJB objectタイムアウト値に設定します。詳細は、Interstage管理コンソールのヘルプを参照してください。	120 2147483647 1	設定する数値の単位は秒です。

各時間監視機能の設定値について

各種時間監視機能を併用する場合の設定方法について説明します。
EJBサービスで使用できる各種時間監視機能を併用する場合、各時間監視機能の時間の設定は以下のように設定してください。

```
T(s)＞T(c)＞T(a)
```

T(s):STATEFUL Session Beanの無通信監視機能に設定するタイムアウト時間
T(c):クライアントにサーバメソッドが復帰するまでの待機時間
T(a):アプリケーションの最大処理時間

10.5.1 アプリケーションの最大処理時間の時間監視機能

この機能は、IJServerを利用して、アプリケーションを動作させる場合に使用できます。
この機能を使用することにより、EJBアプリケーションの無限ループを検出できます。

以下にサーバ処理中にアプリケーションの最大処理時間を超過した場合の処理について説明します。
アプリケーションが存在するプロセスを強制停止するように設定している場合と、設定していない場合で処理内容が異なります。

アプリケーションが存在するプロセスを強制停止する設定の場合

プロセスが強制停止されます。
トランザクション中の場合は、プロセス終了の結果、トランザクションがロールバックされます。
最大処理時間を超過した場合、サーバのシステムログに以下のメッセージが出力されます。

```
extp: エラー: EXTP4365: アプリケーションの処理時間が監視時間を超過しました
```

クライアントには、以下の例外が通知されます。

```
java.rmi.RemoteException:CORBA UNKNOWN
```

アプリケーションが存在するプロセスを強制停止しない設定の場合

プロセスは強制停止されません。
最大処理時間を超過した場合、サーバのシステムログに以下のメッセージが出力されます。

```
extp: 警告: EXTP4366: アプリケーションの処理時間が監視時間を超過しました
```

クライアントには、通知されません。

最大処理時間を超過した後、クライアントからサーバに処理の要求が行われた場合の処理について説明します。
アプリケーションが存在するプロセスを強制停止するように設定している場合と、設定していない場合で処理内容が異なります。

アプリケーションが存在するプロセスを強制停止する設定の場合

処理の実行はできません。サーバのシステムログには、出力されません。
クライアントには、以下の例外が通知されます。

```
java.rmi.RemoteException: CORBA NO_IMPLEMENT
```

アプリケーションが存在するプロセスを強制停止しない設定の場合

通常通り、処理は実行されます。

注意

Interstage管理コンソールまたはisj2eeadminコマンドで“アプリケーション最大処理時間超過時の制御”に「警告メッセージを出力する」を選択した場合、最初のタイムアウト発生時にメッセージを出力してから10分間は同一プロセスからのタイムアウト時間超過メッセージの出力は抑止されます。
また、「プロセスを強制停止する」を選択した場合、プロセス強制停止処理はスレッドダンプの出力前にプロセスが強制停止されることを防止するために、2回目のスレッドダンプ出力処理の10秒後に実施します。
そのため、タイムアウト時間超過メッセージが出力されてから10秒後に2回目のスレッドダンプが出力され、さらに10秒後に

プロセスが強制停止されるため少なくとも20秒間プロセスは停止しません。そのことにより、メッセージが出力された場合でもプロセス停止までの間に正常にアプリケーションが復帰し、その後にプロセスが強制停止される場合があります。

10.5.2 クライアントにサーバメソッドが復帰するまでの待機時間の監視機能

CORBAサービスでは、アプリケーションの稼働状況を監視するためのタイムアウト監視機能を備えており、クライアント/サーバアプリケーションの動作中に、クライアントでサーバメソッドが発行されてから、そのメソッドがクライアントに復帰するまでの時間を監視します。
この機能の詳細は、“OLTPサーバ運用ガイド”の“CORBAアプリケーション運用時のタイマ監視”を参照してください。
クライアントにサーバメソッドが復帰するまでの待機時間が設定値を超えた場合、クライアントとサーバの通信が切断されます。

クライアントへは、以下の例外が通知されます。

```
java.rmi.MarshalException: CORBA COMM_FAILURE 1179255041
```

クライアントにサーバメソッドが復帰するまでの待機時間が設定値を超えた後、クライアントからサーバに処理の要求を行う場合は、createメソッドからやり直してください。

10.5.3 STATEFUL Session Beanの無通信監視機能

STATEFUL Session Beanの無通信監視機能とは、STATEFUL Session BeanのEJB objectに対して、一定時間超過してもビジネスメソッドが実行されなかった場合、コンテナが該当のインスタンスと対応するEJB objectを削除する機能です。
この機能を使用することによって、ejbRemoveメソッドを発行し忘れている不要なEJB objectが削除されるため、使用するメモリの最適化できます。デフォルトは30分です。
削除されたEJB objectに対応するインスタンスに対してクライアントから要求が来た場合、コンテナはアクセスで経由するインタフェースの種類によって、以下の例外を返却します。

- 経由するインタフェースがRemoteインタフェースの場合

  java.rmi.NoSuchObjectExceptionをクライアントへ返却します。

- 経由するインタフェースがLocalインタフェースの場合

  javax.ejb.NoSuchObjectLocalExceptionをクライアントへ返却します。

注意

- タイムアウトが発生した時、コンテナはejbRemoveメソッドを呼び出します。その時に、ejbRemoveメソッド内で例外が発生しても、EJB objectを自動的に削除します。
- トランザクション管理種別が“Bean”でタイムアウトが発生した時に、コンテナはejbRemoveメソッドを呼び出し、トランザクション処理中である場合は、コンテナがトランザクションを自動的にロールバックします。

10.5.4 EJB objectのタイマ削除機能

create/finderメソッドにより生成されたEJB objectのインスタンスは、removeメソッドが発行されるとメモリ上から消滅します。Entity Beanは、通常removeメソッドが発行されないBeanであるため、Entity Beanをプロセス外から呼び出した場合、create/finderメソッドにより生成されたEJB objectのインスタンスがメモリ上に残存してしまいます。

こうした状況が発生するのを防ぐために、Entity Beanへの最終アクセスから一定時間後に、放置されたEJB objectのインスタンスを削除する機能があります。この機能を、“**EJB objectのタイマ削除機能**”といい、その時間を、“**Entity BeanのEJB objectタイムアウト値**”といいます。

Entity Beanをプロセス内で呼び出した場合は、以下のタイミングでEJB objectがJavaのガーベジコレクションの対象となり、自動的に削除されます。そのため、このEJB objectのタイマ削除機能は、プロセス外から呼び出されるEntity Beanで有効です。

- Entity Beanを呼び出すBeanがremoveされたとき
- Entity Beanを呼び出すBean(Session Beanの場合)のSTATEFUL Session Beanの無通信監視機能でタイムアウトが発生したとき

- Entity Beanを呼び出すBean(Entity Beanの場合)のEJB objectのタイムアウトが発生したとき

Entity BeanのEJB objectタイムアウト値は、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義] > [Interstage拡張情報]のEJB objectタイムアウト値に設定します。設定方法の詳細は、Interstage管理コンソールのヘルプを参照してください。

注意

- トランザクション中にEJB objectが削除されないように、EJB objectタイムアウト値はトランザクションタイムアウト値より大きく設定してください。
- Entity Beanをプロセス外から呼び出した場合、EJB objectがタイマー削除されない間でメモリに残存したり、データベースのレコードを大量に扱う場合にはCORBAの通信路を頻繁に経由することになるため、Entity Beanはできるだけ同一JavaVM内で呼び出されるBeanに設定してください。

10.6 EJBタイマーサービス

EJBタイマーサービスとは

ここでは、EJBサービスの時間監視機能のひとつであるEJBタイマーサービスについて説明します。
EJBタイマーサービスは、EJB2.1規約から追加された機能です。本機能では、以下のようなタイマーを作成し、任意の時刻にEJBコンテナからコールバック処理を実行させることができます。

- 一定時間経過後にコールバック
- 一定時間経過後にコールバック、その後定期間隔でコールバック
- 指定日時にコールバック
- 指定日時にコールバック、その後定期間隔でコールバック

したがって、例えば買い物をするとポイントが貯まるサービスで、毎日0:00にポイントを集計し、ポイントが貯まったお客様に販促メールを配信するといったタイマーを利用したイベント処理を作成することが可能になります。

注意

本機能はEJBアプリケーションでタイマーを作成する機能です。タイマーを作成したIJServerプロセスが停止した場合(メモリ不足などにより強制停止した場合を含む)には、タイマーも停止します。このため、タイマーが予期せず停止する場合がありますので、常時タイマーを起動しておきたい場合には他の時間監視機能を使用することをお勧めします。

使用可能なEJBアプリケーション

EJBタイマーサービスはSTATELESS Session Bean、Message-driven Beanで使用可能です。

- 使用可能なEnterprise Bean
  - STATELESS Session Bean
  - Message-driven Bean
- 使用不可能なEnterprise Bean
  - STATEFUL Session Bean
- 利用に制限があるEnterprise Bean
  - Entity Bean (注)
    注) 本バージョンではEntity Beanはタイマーによるコールバック処理(ejbTimeoutメソッド)は実行されません。また、EJBタイマーサービスを使用するEntity Beanを配備したIJServerを起動するとIJServer21149の警告メッセージが出力されます。

基本機能

タイマーの生成方法

タイマーはEJBタイマーサービスのcreateTimerメソッドを実行して生成します。createTimerメソッドの引数に実行時刻や実行間隔を指定します。EJBタイマーサービスは、Enterprise BeanにEJBコンテナより渡されるEJBコンテキストのgetTimerServiceメソッドを実行して取得します。

タイマーのキャンセル方法

生成したタイマーをキャンセルするには、Enterprise Beanより生成したタイマーのcancelメソッドを実行します。タイマーは、Enterprise BeanにてgetTimerServiceメソッドを実行して取得したEJBタイマーサービスに対し、getTimersメソッドを実行することにより取得できます。また、コールバック処理(ejbTimeoutメソッド)の引数で渡されます。

コールバック処理の実行

タイマー生成時に指定した時刻に、EJBタイマーサービスより各Enterprise Beanのコールバック処理(ejbTimeoutメソッド)を実行します。Enterprise Beanはjavax.ejb.TimedObjectインタフェースを実装している必要があります。

EJBタイマーサービスの有効範囲

タイマーは生成したプロセス内でのみ使用できます。そのためEnterprise BeanのRemoteインタフェースやWebサービスのエンドポイントを通してリモート側で取得しないでください。また、クラスタ構成やプロセス多重環境での使用は未サポートです。EJBタイマーサービスを実装したEJBアプリケーションを配備してプロセス多重度を2以上に設定したIJServerで起動した場合は、IJServer21148のエラーが発生します。

EJBタイマーサービスで生成するタイマーは、配備したモジュールが非活性になると削除されます。そのため以下の操作を行った場合、必要に応じてタイマーを再登録してください。

- IJServerを再起動した場合
- ホットデプロイ機能を使用した配備解除や上書き配備を行った場合
- IJServerが異常終了後、プロセスが自動再起動された場合

タイマーの正確性と処理遅延時の動作

負荷が集中するとタイマーの開始時間がずれる場合があります。
タイマーの処理がすぐに完了せず、後続のタイマーの実行時間が来た場合、後続のタイマーは実行中のタイマーの完了を待ちます。実行中のタイマーが完了後、実行予定時刻が超過しているタイマーはただちに実行されます。一定間隔で実行するタイマーの場合、実行予定時刻を越えている処理分連続して実行されます。

トランザクションとの関係

タイマー生成時

トランザクションの範囲内で生成したタイマーは、トランザクションがロールバックされると削除されます。

タイマーキャンセル時

トランザクションの範囲内でタイマーをキャンセルした場合、トランザクションがロールバックされるとキャンセルは無効になります。新規タイマーの場合は削除されます。既存のタイマーの場合は継続して動作します。

コールバック処理実行時

コールバック処理実行時のトランザクションは、EJBアプリケーションに定義されたトランザクション属性により以下のように動作します。

トランザクション属性	EJBコンテナの処理
RequiresNew Required	EJBコンテナはトランザクションを新しく開始してからコールバック処理を実行します。 トランザクション内でロールバックされた場合、もしくは例外が発生した場合、EJBタイマーサービスはコールバック処理を1回だけリトライします。リトライ中に再びロールバックされた場合、もしくは例外が発生した場合、警告メッセージIJServer21147をコンテナログに出力し、タイマーの処理を継続します。
Supports NotSupported Never	EJBコンテナはトランザクションを開始せず、コールバック処理を実行します。
Mandatory	指定できません。 指定された場合、EJBアプリケーション活性時にエラーメッセージIJServer21095をコンテナログに出力します。

トランザクション属性の詳細は“10.4.1 トランザクション管理種別とトランザクション属性”を参照してください。

コールバック処理が実行されるインスタンス

コールバック処理(ejbTimeoutメソッド)はそれぞれ以下のインスタンス上で実行されます。

- STATELSS Session Beanの場合

  プーリングされているインスタンス上でコールバック処理が実行されます。

- Message-driven Beanの場合

  新しいインスタンス上でコールバック処理が実行されます。
  (インスタンスはコールバック処理実行後削除されます。)

- Entity Beanの場合

  未サポートです。コールバック処理は実行されません。

EJBタイマーサービスを利用するアプリケーションの作成方法

以降で、EJBタイマーサービスを利用するアプリケーションの作成方法を説明します。

10.6.1 EJBタイマーサービスのアクセス方法

EJBアプリケーションからEJBタイマーサービスにアクセスするには、EJBコンテナから渡されるEJBコンテキスト(javax.ejb.EJBContext)のインタフェースに規定されているgetTimerServiceメソッドを用います。
EJBコンテキストは各EJBアプリケーション(javax.ejb.SessionBean、javax.ejb.EntityBean、javax.ejb.MessageDrivenBean)のインタフェースに規定されているsetSessionContextメソッド、setEntityContextメソッド、setMessageDrivenContextメソッドにてEJBコンテナから渡されます。

10.6.2 監視の開始方法

EJBタイマーサービス(javax.ejb.TimerService)のインタフェースに規定されているcreateTimerメソッドを実行して、タイマーオブジェクト(javax.ejb.Timer)を生成します。タイマーオブジェクトが生成された時から監視時間のカウントダウンが始まります。
createTimerメソッドには以下の表に記載した種類があります。指定時間の単位はミリ秒です。

javax.ejb.TimerServiceインタフェースに規定されているAPI

項番	メソッド名	説明
1	public javax.ejb.Timer createTimer (java.util.Date initialExpiration, long intervalDuration, java.io.Serializable info)	指定した日時(initialExpiration)に1回、その後一定間隔(intervalDuration)で繰り返し実行するタイマーを生成します。 実行時に渡したい情報がある場合は、infoに指定します(ない場合はnullを指定します)。
2	public javax.ejb.Timer createTimer (java.util.Date expiration, java.io.Serializable info)	指定した日時(expiration)に1回だけ実行するタイマーを生成します。 実行時に渡したい情報がある場合は、infoに指定します(ない場合はnullを指定します)。
3	public javax.ejb.Timer createTimer (long initialDuration, long intervalDuration, java.io.Serializable info)	監視開始から指定された監視持続時間(initialDuration)経過後に1回、その後一定間隔(intervalDuration)で繰り返し実行するタイマーを生成します。 実行時に渡したい情報がある場合は、infoに指定します(ない場合はnullを指定します)。
4	public javax.ejb.Timer createTimer (long duration, java.io.Serializable info)	監視開始から指定された監視持続時間(duration)経過後に1回だけ実行するタイマーを生成します。 実行時に渡したい情報がある場合は、infoに指定します(ない場合はnullを指定します)。
5	public java.util.Collection getTimers ()	このメソッドを実行したEJBアプリケーションに紐付けられている活動中のタイマーオブジェクトをすべて取得します。

補足

javax.ejb.TimerServiceインタフェースにはcreateTimerメソッドの他にgetTimersメソッドが規定されています。getTimersメソッドを実行するとそのEJBアプリケーションが生成したタイマーのコレクションを取得することができます。タイマーオブジェクトが同一かどうかを確認する場合は、タイマー(javax.ejb.Timer)オブジェクトが持つequalsメソッドを使用してください。("==”演算子での比較はオブジェクトの同一性を保証しません。)

10.6.3 時間監視処理の実行方法

EJBタイマーサービスを利用したいEJBアプリケーションは、Enterprise Beanクラスにjavax.ejb.TimedObjectインタフェースを実装する必要があります。EJBタイマーサービスを利用するEnterprise Bean自身かその親クラスで実装してください。

javax.ejb.TimedObjectインタフェースにはejbTimeoutメソッドが一つのみ宣言されています。

以下のようにEnterprise Beanに実装してください。

実装例(Stateless Session Beanの場合)

```
public class  Enterprise Bean名
    implements javax.ejb.SessionBean, javax.ejb.TimedObject
{
    ・・・
    public void ejbTimeout(javax.ejb.Timer timer) {
        // 予定時刻に実行するビジネスロジックを
        // ここに記述します
    }
}
```

EJBタイマーサービスに登録した予定時刻になるとEJBコンテナはejbTimeoutメソッドを呼び出します。予定時刻に実行したいビジネスロジックをejbTimeoutメソッドに記述してください。

ejbTimeoutメソッドから実行可能なメソッドは以下を参照してください。

- STATELESS Session Beanの場合
  “第12章 Session Beanの実装”の“12.6.3 Enterprise Beanクラスのメソッドが実行可能な操作”
- Entity Beanの場合
  “第13章 Entity Beanの実装”の“13.6.14 Enterprise Beanクラスのメソッドが実行可能な操作”
- Message-driven Beanの場合
  “第14章 Message-driven Beanの実装”の“14.2.3 Enterprise Beanクラスのメソッドが実行可能な操作”

10.6.4 タイマーのキャンセル・状況参照方法

タイマー(javax.ejb.Timer)インタフェースには以下のAPIが用意されていて、時間監視のキャンセルや状況参照をすることができます。

javax.ejb.Timerインタフェースに規定されているAPI

項番	メソッド名	説明
1	public void cancel ()	タイマー監視をキャンセルします。キャンセルが実行されたタイマーはEJBコンテナにより削除されます。
2	public javax.ejb.TimerHandle getHandle ()	タイマーハンドルオブジェクトを取得します。後でタイマーオブジェクトを再取得したい時にTimerHandleインタフェースのgetTimerメソッドを利用して得ることができます。
3	public java.io.Serializable getInfo()	タイマー生成時に渡した情報(createTimerメソッドに渡したinfoオブジェクト)を取得します。
4	public java.util.Date getNextTimeout ()	次のejbTimeoutメソッドが実行される予定日時を取得します。
5	public long getTimeRemaining ()	次のejbTimeoutメソッドが実行されるまでのタイマーの残時間を取得します。単位はミリ秒です。

javax.ejb.TimerHandleインタフェースに規定されているAPI

項番	メソッド名	説明
1	public javax.ejb.Timer getTimer()	このTimerHandleオブジェクトに対応するTimerオブジェクトを取得できます。

補足

- 一度のみ実行するタイマーの場合、コールバック処理(ejbTimeoutメソッド)正常完了後、そのタイマーはコンテナより削除されます。
- 削除されたタイマーオブジェクトに対してメソッドを実行するとNoSuchObjectLocalExceptionが返却されます。

10.6.5 その他

ejbTimeoutメソッドはローカルで利用するメソッドであり、クライアントからリモートで呼び出されるわけではないのでクライアント識別子を持っていません。このため、ejbTimeoutメソッドでgetCallerPrincipalメソッドを実行すると、コンテナは認証情報のない認証オブジェクトを返却します。

10.7 EJBサービス機能における注意事項

EJBアプリケーションをIJServer上に配備して使用する場合、以下に注意してください。

- 配備するEJBアプリケーション名の最大長は255バイトです。それ以上長い名前のEJBアプリケーションを配備しないでください。
- 複数のIJServer上に、同一のEJBアプリケーションを配備して運用できません。(ServletとEJBが同一JavaVMであるIJServerの場合は運用できます)。
- Message-driven Beanを運用する場合、イベントサービスが停止するなど、運用が継続できない状況となった場合にはJava VMを停止します。
  このため、異常が発生したMessage-driven Beanと同一の、IJServerに配備された他のEJBアプリケーションの運用も停止します。
- IJServerに配備されたEJBアプリケーションが、同一のRemoteインタフェース、Homeインタフェースを使用していると、起動時にエラーとなります(ローカル呼出しを使用した場合、またはServletとEJBが同一JavaVMであるIJServerの場合はエラーとなりません)。
- クライアントからEJBアプリケーションのメソッドが呼び出された場合、クライアント要求数が最大処理スレッド数(デフォルト値:64)に設定された数を超えると、クライアントからの要求は、IJServer単位にシリアルへキューイングされます。
- Windows32/64 Linux32/64
  分散トランザクションを使用する場合、IJServerのプロセスは多重で起動できません。
- EJB2.1規約に準拠したEJBアプリケーションはIJServerタイプが以下の場合のみ使用できます。EJB2.1規約に準拠したEJBアプリケーションを以下以外のIJServerタイプのIJServerに配備しようとした場合にはエラーとなります。

  ー Webアプリケーションと EJBアプリケーションを同一JavaVMで運用

第11章　EJBアプリケーションの開発

本章では、EJBアプリケーションとクライアントアプリケーションの開発について説明します。

なお、EJBアプリケーションの形態別の実装方法については、第12章 Session Beanの実装、第13章 Entity Beanの実装、第14章 Message-driven Beanの実装をそれぞれ参照してください。
また、EJBアプリケーションの呼び出し方法の詳細については、第15章 EJBアプリケーションの呼出し方法を参照ください。

EJBアプリケーションおよびクライアントアプリケーションを開発する前に、業務の内容、連携方法、サーバの環境などを十分に分析し、アプリケーションの内容を考慮してください。

11.1 EJBアプリケーション形態の選択

EJBアプリケーションには、以下に示す3つの形態があります。ユーザの業務にあった形態を選択してください。

Session Bean

Session Beanは、クライアントとの対話処理を行うアプリケーションの形態です。
Session Beanについての詳細は、“10.1 Session Beanの実行環境”を参照してください。

Entity Bean

Entity Beanは、データベース処理を行うアプリケーションの形態です。
Entity Beanについての詳細は、“10.2 Entity Beanの実行環境”を参照してください。

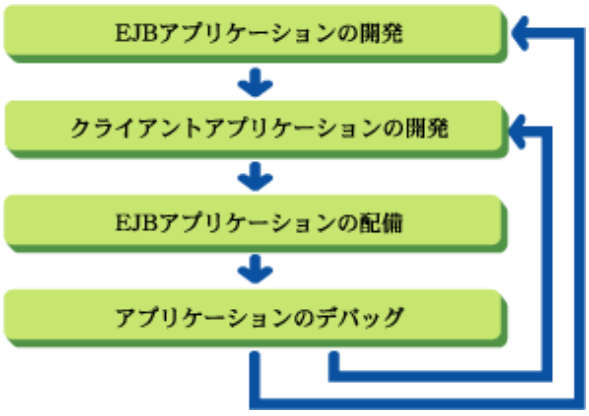
Message-driven Bean

Message-driven Beanは、JMSメッセージを処理するアプリケーションの形態です。
Message-driven Beanについての詳細は、“10.3 Message-driven Beanの実行環境”を参照してください。

11.2 アプリケーションの開発の流れ

アプリケーションを開発し、Interstage管理コンソールを利用してアプリケーションのデバッグを行うまでの流れを以下に示します。
アプリケーションのデバッグ完了後、IJServerを起動してEJBアプリケーションを運用します。

ワークユニットについては、“運用ガイド(基本編)”を参照してください。



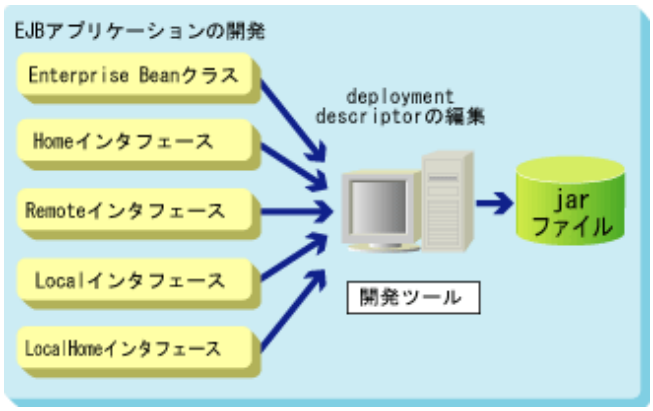
当社のコンポーネント指向の統合開発支援ツールであるInterstage Studioを使用することにより、一連の手順を統合した使いやすいビュー操作によって操作できます。

Interstage Studioでは、さまざまなEJBアプリケーションおよびクライアントアプリケーションの開発支援機能を提供しているため、アプリケーション開発の生産性を向上できます。

11.3 EJBアプリケーションの開発

EJBアプリケーションの開発は、以下の手順で行います。

1. Home/LocalHome/Remote/Localインタフェース、Enterprise Bean/Primary Keyクラスの開発
2. deployment descriptorの編集
3. ソースコードのコンパイル
4. EJBアプリケーションのパッケージ化



Interstage Studioを利用してEJBアプリケーションを開発する場合の詳細は、“Interstage Studio ユーザーズガイド”を参照してください。
他社の開発ツールを利用して開発する場合は、利用する開発ツールのマニュアルを参照してください。

クラスファイル

以下にSession Bean、Entity Bean、Message-driven Beanを構成する各クラスファイルについて説明します。

Homeインタフェース(Message-driven Beanでは不要)

Session Bean、Entity Beanへのアクセスインタフェースです。
EJBアプリケーションの生成の制御手段を定義します。

LocalHomeインタフェース(Message-driven Beanでは不要)

Session Bean、Entity Beanへのアクセスインタフェースです。
EJBアプリケーションの生成の制御手段を定義します。
同一JavaVM内で呼び出されるインタフェースです。

Remoteインタフェース(Message-driven Beanでは不要)

Session Bean、Entity Beanへのアクセスインタフェースです。
ユーザのビジネスメソッドを呼び出すためのインタフェースを定義します。

Localインタフェース(Message-driven Beanでは不要)

Session Bean、Entity Beanへのアクセスインタフェースです。
EJBアプリケーションの生成の制御手段を定義します。
同一JavaVM内で呼び出されるインタフェースです。

Enterprise Beanクラス

目的とする処理を実行するサーバプログラムです。
ユーザの業務用メソッド(ビジネスメソッド)を実装します。

Primary Keyクラス(Session Bean, Message-driven Beanでは不要)

Entity Beanクラスのインスタンスの一意性を表すためのクラスです。
EJBアプリケーションの作成にあたって、当社の統合開発支援ツールInterstage Studioを使用すると、EJBアプリケーションを構成する各クラスファイルのひな形が自動生成され、ビジネスメソッドの処理を記述するだけで、EJBアプリケーションが完成します。

プログラムとして作成した以下のクラスファイルは、jarファイルに格納しパッケージ化します。このjarファイルが最小の流通単位となります。

- Enterprise Beanクラス [.class]
- Homeインタフェース [.class] (Message-driven Beanでは不要)
- Remoteインタフェース (Message-driven Beanでは不要)[.class]
- LocalHomeインタフェース(Message-driven Beanでは不要)[.class]
- Localインタフェース(Message-driven Beanでは不要)[.class]
- Primary Keyクラス (Session Bean, Message-driven Beanでは不要)[.class]

jarファイルは、さらに以下のファイルを含みます。これらはEJBアプリケーション作成時にInterstage Studioにより自動的に作成されます。

- META-INF/MANIFEST.MF
  jarの格納物を示す manifestファイルです。
- ejb-jar.xml
  EJBアプリケーションのdeployment descriptorの情報が定義されているファイルです。EJB1.1規約よりXML形式で記述されます。

注意

- EJBアプリケーションをサーバ間で連携する場合は、サーバごとにEJBアプリケーションをパッケージ化してください。
- Message-driven Beanとその他のEnterprise Beanを使用する場合で、IJServerを複数プロセスで起動して運用したい場合には、以下を別々にパッケージ化してください。
  - Message-driven BeanとMessage-driven Beanからのみ呼び出されるEnterprise Bean(1プロセスでのみ運用)
  - 上記以外(複数プロセスで運用)
- EJBアプリケーションのdeployment descriptorの情報が定義されているファイル(ejb-jar.xml)は、XML形式で記述する必要があります。このため、ejb-jar.xmlを編集する場合は、XML形式の仕様に従ってください。
  特に以下の文字については、定義済み実体参照(暗黙定義エンティティ)で記述してください。

編集する文字	定義済み実体参照
<	<
>	>
&	&
'	'
"	"

名前空間プレフィックスつきのタグは指定しないでください。
指定した場合、配備に失敗したり、名前変換機能が使用できなくなることがあります。
例:<pfx:session>

11.4 クライアントアプリケーションの開発

ここでは、Javaで記述するクライアントアプリケーションの開発方法を説明します。

Session BeanとEntity Beanの場合

Session BeanとEntity Beanに対するクライアントアプリケーションとは、Remoteインタフェースおよびホームインタフェースを通じてEJBアプリケーションにアクセスする、クライアント側のアプリケーションです。

Remoteインタフェース/LocalHomeインタフェース/Localインタフェースとホームインタフェースに対する、EJBクライアントの形で記述します。

クライアントアプリケーションを構築するときには、ejb-jar ファイルをCLASSPATHに設定してください。

Message-driven Beanの場合

Message-driven Beanに対してのクライアントアプリケーションとは、JMSの送信アプリケーションです。JMSの送信アプリケーションの詳細については第6部 JMS編を参照してください。

11.5 EJBアプリケーションの配備

開発したEJBアプリケーションを実行可能な状態にするために、配備を行います。
Interstage管理コンソールを使用して配備を行う場合は、Interstage管理コンソールのヘルプを参照してください。

11.6 EJBアプリケーションのデバッグ

EJBアプリケーションのデバッグは、IJServerのデバッグ機能を利用します。
IJServerのデバッグ方法についての詳細は、“3.12 アプリケーションのデバッグ”を参照してください。

11.7 他社開発環境の利用

ここでは、他社の開発環境を利用して作成したEJBアプリケーションを、Interstage上で運用する際に必要となる作業について説明します。

作業手順について

他社の開発環境を利用する場合でも、開発から運用までの大きな作業手順に違いはありません。

他社の開発環境を利用して、“11.3 EJBアプリケーションの開発”に記載されている手順でEJBアプリケーションのパッケージ化まで行ってください。
それ以降の作業については、“11.5 EJBアプリケーションの配備”を参照してください。

CMPのEntity Beanを開発する場合

CMPのEntity Beanを開発する場合は、EJBアプリケーションを配備後、Interstage管理コンソールを使用してCMP定義を設定してください。設定方法の詳細は、Interstage管理コンソールのヘルプを参照してください。

第12章　Session Beanの実装

本章では、Session Beanのプログラミング方法について説明します。

12.1 Session Beanの概要

以下に、Session Beanの概要について説明します。

また、Webサービス化するSTATELESS Session Beanの開発方法については“第18章 Webサービスの開発”を参照してください。

クラスファイルの構成

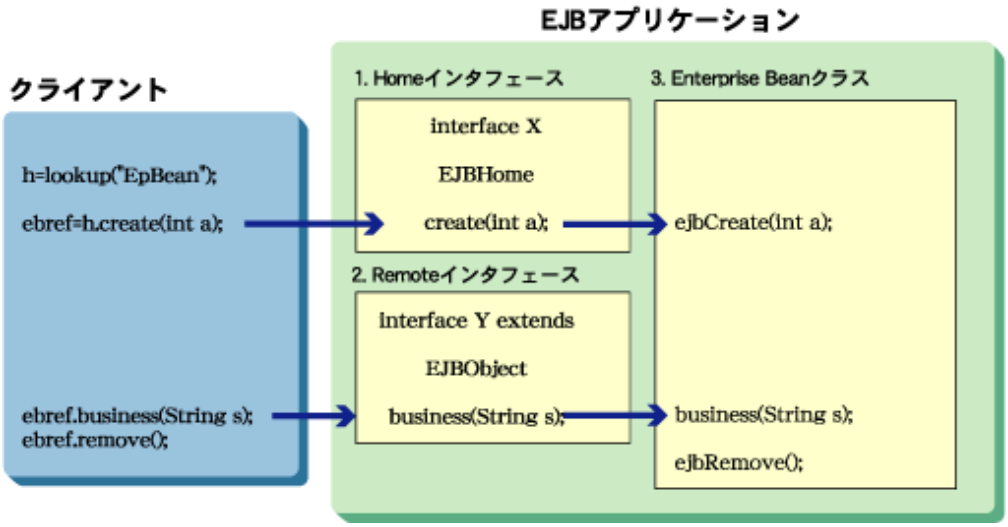
Session BeanのEJBアプリケーションは、以下の5つのクラスファイルから成り立っています。それぞれのクラス名はユーザの任意の名前を付けることができます。

下表1～5のクラスファイルはEnterprise Beanを使用するためのインタフェースです。

	クラスファイル名	内容
1	Homeインタフェース	EJB objectの生成を行うインタフェースを定義します。
2	LocalHomeインタフェース	同一JavaVM上で、EJB objectの生成を行うインタフェースを定義します。
3	Remoteインタフェース	ユーザのビジネスメソッドを呼び出すためのインタフェースを定義します。
4	Localインタフェース	同一JavaVM上で、ユーザのビジネスメソッドを呼び出すためのインタフェースを定義します。
5	サービスエンドポイントインタフェース	Webサービスでユーザのビジネスメソッドを呼び出すためのインタフェースを定義します。 このインタフェースはSTATELESS Session Beanでのみ定義できます。
6	Enterprise Beanクラス	サーバで実際に処理を行うクラスで、ユーザの開発するメソッドを実装します。

相関図

以下に例として、クラスファイルとクライアントアプリケーションの相関図を示します。



[図の説明]

1. Homeインタフェースは EJB objectを作成するためのcreateメソッドを定義します。
   クライアントより、createメソッドを呼び出すと、EJB objectを作成し、同じ形式で記述されているEnterprise Beanクラスの ejbCreateメソッドを呼び出します。
   また、クライアントからHomeインタフェースのメソッドを呼び出すには、lookup()メソッドで、EJBHomeを検索してください。
2. Remoteインタフェースもしくはサービスエンドポイントインタフェースは、Enterprise Beanクラスのビジネスメソッドを定義します。
   インタフェースで定義するビジネスメソッドの形式は、Enterprise Beanクラスに実装されるビジネスメソッドと同じ形式にしてください。
   Remoteインタフェースはjavax.ejb.EJBObjectインタフェースを継承しており、javax.ejb.EJBObjectインタフェースのメソッドを使用できます。
   SOAPとRMIoverIIOPの双方から呼び出されるSTATELESS Session Beanを作成する場合には、RMIoverIIOP用のインタフェース(Remoteインタフェース/Homeインタフェース)とSOAP用のサービスエンドポイントインタフェースの両方を作成してください。
   SOAPからのみ呼び出されるSTATELESS Session Beanの場合には、サービスエンドポイントインタフェースのみ用意する(Remoteインタフェースなどを用意しない)ことも可能です。
3. ユーザは実際に業務を行うejbCreate、ejbRemoveおよびビジネスメソッドをEnterprise Beanクラスに実装します。

12.2 Homeインタフェースの作成

Homeインタフェースは、RMIインタフェースの形で記述します。

Homeインタフェースには、クライアントより EJB objectを createするためのメソッドを定義します。ビジネスメソッドで利用する固有の型も定義できます。

また、EJB2.0規約以降に準拠したEJBアプリケーションでは、STATEFUL Session Beanだけcreateで始まる任意の名前を記述できます。
これにより、createメソッド名に意味のある名前を記述できるため、EJBアプリケーションをわかりやすく作成できます。

Interstage Studioを使用して開発を行った場合、Homeインタフェースのひな形が自動生成されます。

以下に、Homeインタフェースの記述形式を示します。

記述形式

```
public interface  Homeインタフェース名  extends  javax.ejb.EJBHome
{
    public  Remoteインタフェースタイプ  create(任意の引数<STATEFULのみ>)
        throws javax.ejb.CreateException,
        java.rmi.RemoteException;
}
```

規約

Homeインタフェースは以下の規約を満たしていなければなりません。

- Homeインタフェースは、1つ以上の createメソッドを定義します。
  各 createメソッドは createで始まる任意の名前でなければなりません。
  また、Enterprise Beanクラスに定義されたejbCreateメソッドの1つに一致しなくてはなりません。
  一致するejbCreateメソッドは引数の数と型が同じでなければなりません(返却値は違っていてもかまいません)。
- createメソッドの返却値のタイプは、Enterprise BeanのRemoteインタフェースタイプでなければなりません(Enterprise Beanクラスのejbcreateメソッドの返却値はvoid固定です)。
- createメソッドは、引数のタイプと数が、Enterprise Beanクラスに定義されたejbCreateメソッドの1つに一致しなくてはなりません(返却値のタイプは違っていてもかまいません)。
- STATELESSの場合、createメソッドに引数は指定できません。
- Homeインタフェースは javax.ejb.EJBHomeインタフェースを継承しなければなりません。

- Homeインタフェースで定義されたメソッドは、RMI over IIOPの規約に従わなくてはなりません。
  引数と戻り値がRMI/IIOPの有効な型であり、throws節がjava.rmi.RemoteExceptionを含まなければなりません。
- Enterprise Beanクラスのejb Createメソッドに定義されたすべての例外は、一致する createメソッドのthrows句に定義されなければなりません。
  また、throws句には、javax.ejb.CreateExceptionを含まなければなりません。
- throws句には、javax.ejb.CreateExceptionを含まなければなりません。
- throws句には、java.rmi.RemoteExceptionを含まなければなりません。
- Homeインタフェースは、スーパーインタフェースをもつことができます。
  インタフェース継承の仕様は、RMI over IIOP上の規約におけるRemoteインタフェースの定義に従わなくてはなりません。

注意

Homeインタフェースの作成における注意事項

throws句に宣言する例外の数は、5以下にしてください。多数宣言されている場合、配備時間に影響します。

12.2.1 記述例

Homeインタフェースの記述例

SampleHomeという名前のインタフェースを設定した場合の例を以下に示します。太字部分は、ユーザの任意の指定ができます。

```
SampleHome.java

    import javax.ejb.*;
    import java.rmi.*;
    public interface SampleHome extends EJBHome      /* EJBHomeを継承します*/
    {
        public SampleRemote create(String sn)       /*  createの形式を定義      */
        throws RemoteException,                     /*  引数はSTATEFULのみ設定可*/
        CreateException;
    }
```

Enterprise Beanクラスの記述例

任意に指定された createメソッドがクライアントより発行された時点で、サーバ上の起動されているEJBアプリケーションにおいて一致する ejbCreateメソッドが起動されます。
両者のメソッドは、引数のタイプや数が一致している必要があります。

ejbCreateでは、EJBアプリケーションが使用するインスタンス変数の初期化やリソース(データベース、ファイル)のオープンなど、各種の初期化処理を行います。

SampleBeanという名前のEJBアプリケーションを作成した場合の例を以下に示します。太字部分は、ユーザの任意の指定ができます。

```
SampleBean.java(部分)

    ・・・
    // startup work
    public void ejbCreate(String sn)        /*  EJBアプリケーションの初期化処理を行います*/
        throws EJBException,
        CreateException {
            System.out.println("SampleBean: ejbCreate called");
        }
    }
    ・・・
```

12.2.2 使用できるメソッド

Homeインタフェースは、EJBHomeインタフェースを継承しています。クライアントアプリケーションより使用できるメソッドの一覧を以下に示します。

メソッド名	内容
getEJBMetaData()	Enterprise BeanのEJBMetaDataインタフェースを取得します。
getHomeHandle()	Home objectのハンドルを取得します。
remove(javax.ejb.Handle)	ハンドルに対応するEJB objectを消去します。

12.3 LocalHomeインタフェースの作成

LocalHomeインタフェースは、Java VM内で呼ばれるインタフェースであるためRMIの形で記述する必要はありません。

LocalHomeインタフェースには、クライアントよりEJB local objectをcreateするためのメソッドを定義します。
ビジネスメソッドで利用する固有の型も定義できます。

また、EJB2.0規約以降に準拠したEJBアプリケーションでは、STATEFUL Session Beanだけcreateで始まる任意の名前を記述できます。
これにより、createメソッド名に意味のある名前を記述できるため、EJBアプリケーションをわかりやすく作成できます。

以下に、LocalHomeインタフェースの記述形式を示します。

記述形式

```
public interface  LocalHomeインタフェース名  extends  javax.ejb.EJBLocalHome
{
    public  Localインタフェースタイプ  create(任意の引数<STATEFULのみ>)
        throws javax.ejb.CreateException;
}
```

規約

LocalHomeインタフェースは以下の規約を満たしていなければなりません。

- LocalHomeインタフェースはjavax.ejb.EJBLocalHomeインタフェースを継承しなければなりません。
- Localインタフェースに定義されたメソッドのthrows句にはjava.rmi.RemoteExceptionを定義してはなりません。
- Localインタフェースは、スーパーインタフェースを持つことができます。
- LocalHomeインタフェースは、1つ以上のcreateメソッドを定義します。
  各createメソッドは、createで始まる任意の名前でなければなりません。
  また、Enterprise Beanクラスに定義されたejbCreateメソッドの1つに一致しなくてはなりません。
  一致するejbCreateメソッドは引数の数と型が同じでなければなりません(返却値は違っていてもかまいません)。
- createメソッドの返却値のタイプは、Enterprise BeanのLocalインタフェースタイプでなければなりません(Enterprise BeanクラスのejbCreateメソッドの返却値はvoid固定です)。
- Enterprise BeanクラスのejbCreateメソッドに定義されたすべての例外は、一致するcreateメソッドのthrows句に定義されなければなりません。また、throws句には、javax.ejb.CreateExceptionを含まなければなりません。

12.3.1 記述例

LocalHomeインタフェースの記述例

SampleLocalHomeという名前のインタフェースを設定した場合の例を以下に示します。太字部分は、ユーザの任意の指定ができます。

```
SampleLocalHome.java

    import javax.ejb.*;
    import java.rmi.*;
    public interface SampleLocalHome extends EJBLocalHome       /* EJBLocalHomeを継承します*/
    {
        public SampleLocal create(String s)        /*  createの形式を定義       */
        throws CreateException;                      /*  引数はSTATEFULのみ設定可*/
    }
```

Enterprise Beanクラスの記述例

任意に指定されたcreateメソッドがクライアントより発行された時点で、EJBアプリケーションにおいて一致するejbCreateメソッドが起動されます。

両者のメソッドは、引数のタイプや数が一致している必要があります。

ejbCreateでは、EJBアプリケーションが使用するインスタンス変数の初期化やリソース(データベース、ファイル)のオープンなど、各種の初期化処理を行います。

SampleLocalBeanという名前のEJBアプリケーションを作成した場合の例を、以下に示します。太字部分は、ユーザが任意で指定できます。

```
SampleLocalBean.java(部分)

    ・・・
    // startup work
    public void ejbCreate(String sn)         /*  EJBアプリケーションの初期化処理を行います*/
        throws EJBException,
        CreateException {
            System.out.println("SampleLocalBean: ejbCreate called");
        }
    }
    ・・・
```

12.3.2 使用できるメソッド

LocalHomeインタフェースは、EJBLocalHomeインタフェースを継承しています。クライアントアプリケーションより使用できるメソッドの一覧を以下に示します。

メソッド名	内容
remove()	EJB local objectを消去します。

12.4 Remoteインタフェースの作成

Remoteインタフェースは、RMIインタフェースの形で記述します。

Remoteインタフェースには、ユーザがその業務内容に応じて自由に定義できるビジネスメソッドの形式を定義します。また、ビジネスメソッドで利用する固有の型も定義できます。

Interstage Studioを使用して開発を行った場合、Remoteインタフェースのひな形が自動生成されます。

以下に、Remoteインタフェースの記述形式を示します。

記述形式

```
    public interface  Remoteインタフェース名  extends  javax.ejb.EJBObject
    {
        public 返却値 ビジネスメソッド(任意の引数)
            throws java.rmi.RemoteException;
    }
```

規約

Remoteインタフェースは以下の規約を満たしていなければなりません。

- 定義するビジネスメソッドは、Enterprise Beanクラスのビジネスメソッドと同じ名前でなければなりません。
- また、Enterprise Beanクラスのビジネスメソッドのメソッド名、引数の数と型、返却値の型が同じでなければなりません。
- また、Enterprise Beanクラスの一致したビジネスメソッドの throws句に定義されたすべての exceptionは、Remoteインタフェースのメソッドの throws句に定義されなければなりません。
- Remoteインタフェースは javax.ejb.EJBObjectインタフェースを継承しなければなりません。
- Remoteインタフェースで定義されたメソッドは RMI over IIOPの規約に従わなくてはなりません。
- throws句には、java.rmi.RemoteExceptionを含まなければなりません。
- Remoteインタフェースは、スーパーインタフェースをもつことができます。
  インタフェース継承の仕様は、RMI over IIOP上の規約におけるRemoteインタフェースの定義に従わなくてはなりません。

注意

Remoteインタフェースの作成における注意事項

throws句に宣言する例外の数は、5以下にしてください。多数宣言されている場合、配備時間に影響します。

12.4.1 記述例

Remoteインタフェースの記述例

SampleRemoteという名前のインタフェースを設定した場合の記述例を以下に示します。この例では、businessというメソッドにString型のパラメタを渡しています。太字部分はユーザの任意の指定ができます。

```
SampleRemote.java

    import javax.ejb.*;
    import java.rmi.*;
    public interface SampleRemote extends EJBObject    /* EJB objectを継承します  */
    {
        public String business(String s)               /* ユーザの業務に応じて設定*/
        throws RemoteException;
    }
```

Enterprise Beanクラスの記述例

Enterprise Beanクラスのビジネスメソッドの記述例を以下に示します。ビジネスメソッドでは、ユーザがサーバで行う任意の処理を記述します。太字部分はユーザの任意の指定ができます。

```
SampleBean.java(部分)

    ・・・
    // business method
    public String business(String s)           /* ビジネスメソッドの処理を記述 */
        throws EJBException {
            System.out.println("SampleBean: business");
            return SessionName + ":" + s.toUpperCase();
    }
    ・・・
```

12.4.2 使用できるメソッド

Remoteインタフェースは、EJBObjectインタフェースを継承しています。クライアントアプリケーションより使用できるメソッドの一覧を以下に示します。

以下に使用できるメソッドの一覧を示します。

メソッド名	内容
getEJBHome()	Enterprise Bean のHomeインタフェースを取得します。
getHandle()	EJB objectのハンドルを取得します。
isIdentical(javax.ejb.EJBObject)	パラメタとして与えられたEJB objectが、呼び出されたEJB objectと一致するか判定します。
remove()	EJB objectを消去します。

12.5 Localインタフェースの作成

Localインタフェースは、Java VM内で呼ばれるインタフェースであるためRMIの形で記述する必要はありません。

Localインタフェースには、ユーザがその業務内容に応じて自由に定義できるビジネスメソッドの形式を定義します。
また、ビジネスメソッドで利用する固有の型も定義できます。

以下に、Localインタフェースの記述形式を示します。

記述形式

```
public interface  Localインタフェース名  extends  javax.ejb.EJBLocalObject
{
    public 返却値 ビジネスメソッド(任意の引数) ;
}
```

規約

Localインタフェースは以下の規約を満たしていなければなりません。

- Localインタフェースはjavax.ejb.EJBLocalObjectインタフェースを継承しなければなりません。
- Localインタフェースに定義されたメソッドのthrows句にはjava.rmi.RemoteExceptionを定義してはなりません。
- Localインタフェースは、スーパーインタフェースを持つことができます。
- Localインタフェースに定義するメソッドは、Enterprise Beanクラスのメソッドと同じ名前でなければなりません。また、Enterprise Beanクラスのメソッドのメソッド名、引数の数と型、返却値の型が同じでなければなりません。
- Enterprise Beanクラスの一致したメソッドのthrows句に定義されたすべてのexceptionは、Localインタフェースのメソッドのthrows句に定義してください。

12.5.1 記述例

Localインタフェースの記述例

SampleLocalという名前のインタフェースを設定した場合の記述例を以下に示します。この例では、businessというメソッドにString型のパラメタを渡しています。太字部分はユーザが任意で指定できます。

```
SampleLocal.java

public interface SampleLocal extends javax.ejb.EJBLocalObject {    /* EJBLocalObjectを継承します  */
    public String business(String s);              /* ユーザの業務に応じて設定*/
}
```

Enterprise Beanクラスの記述例

Enterprise Beanクラスのビジネスメソッドの記述例を以下に示します。ビジネスメソッドでは、ユーザがサーバで行う任意の処理を記述します。太字部分はユーザが任意で指定できます。

```
SampleBean.java(部分)

   ・・・
   // business method
   public String business(String s)          /* ビジネスメソッドの処理を記述 */
       throws EJBException {
           System.out.println("SampleBean: business");
           return SessionName + ":" + s.toUpperCase();
   }
   ・・・
```

12.5.2 使用できるメソッド

Localインタフェースは、EJB Local Objectインタフェースを継承しています。クライアントアプリケーションより使用できるメソッドの一覧を以下に示します。

メソッド名	内容
getEJBLocalHome()	Enterprise Bean のLocalHomeインタフェースを取得します。
isIdentical(javax.ejb.EJBLocalObject)	パラメタとして与えられた EJB local objectが、呼び出されたEJB local objectと一致するか判定します。
remove()	EJB local objectを消去します。

12.6 Enterprise Beanクラスの作成

Enterprise Beanクラスは、ユーザがインタフェースで定義した ejbCreateメソッドおよびビジネスメソッドを実装します。Interstage Studioを使用して開発を行った場合、Enterprise Beanクラスのひな形が自動生成されます。

以下に、Enterprise Beanクラスの記述の形式を示します。

記述形式

```
public class   Enterprise Bean名
       extends Object
       implements javax.ejb.SessionBean
{
    ・・・
    // startup work
    public void ejbCreate(任意の引数)
    throws javax.ejb.EJBException,
    javax.ejb.CreateException {
            ・・・
    }

    // business method
    public  任意の返却値  ビジネスメソッド(任意の引数)
    throws javax.ejb.EJBException {
           ・・・
    }
    ・・・
}
```

規約

Enterprise Beanクラスは以下の規約を満たしていなければなりません。

- javax.ejb.SessionBeanインタフェースを実装していなければなりません。
- publicとして定義されていなければなりません。
- パラメタを取らないpublicコンストラクタをもつ必要があります。
- ejbCreateメソッドとビジネスメソッドが実装されていなければなりません。
- Enterprise Beanクラスは、Enterprise BeanのRemoteインタフェースを実装してもかまいませんが、メソッド引数または結果として、不用意なthisの受渡しを防ぐRemoteインタフェースを実装しないことが推奨されます。
- Enterprise Beanクラスは、EJB仕様に定められたメソッドのほかにも、メソッド(ビジネスメソッドが内部で呼び出すヘルパーメソッドなど)を実装できます。

ejbCreateとビジネスメソッドには、以下の規約があります。

ejbCreateの規約

ejbCreateは以下の規約を満たしていなければなりません。

- メソッド名は ejbCreateでなければなりません。
  EJB2.0規約以降に準拠したEJBアプリケーションでは、Homeインタフェースにcreateで始まる任意の名前を記述できます。
  この場合は、ejbCreate<任意の名前>でなければなりません。
- メソッドは publicとして定義されていなければなりません。
- 返却値は voidでなければなりません。
- メソッドの引数は、RMI over IIOPの規約に従わなくてはなりません。
- throws句には任意のアプリケーション固有の例外を定義できます。
- throws句は、javax.ejb.CreateExceptionを含むことができます。

ビジネスメソッドの規約

ビジネスメソッドは以下の規約を満たしていなければなりません。

- メソッド名は任意ですが、それらは、EJBのAPIで定義されているメソッドの名前と重複してはなりません。
- メソッドは publicとして宣言されていなければなりません。
- メソッド引数と返却値は、RMI over IIOPの規約に従わなくてはなりません。
- throws句には任意のアプリケーション固有の例外を定義できます。
- Webサービス化するSTATELESS Session Beanを開発する場合には、上記以外にメソッドのパラメタ(引数・返り値)には、“18.3 Javaのデータ型とXMLのデータ型との対応”に示されたJavaデータ型だけを使用する必要があります。この場合には、java.util.Vectorやjava.util.Hashtableをメソッドのパラメタ(引数・返り値)に指定できません。配列型などを使用してください。

12.6.1 記述例

以下に Enterprise Beanの記述例を示します。太字部分は、ユーザの任意の指定ができます。

SampleBean.java

```
    package Sample;

    import javax.ejb.*;
    import java.rmi.*;

    public class SampleBean
```

```
        extends Object implements SessionBean
{
    // constructor
    public void SampleBean() {
            ・・・
            /*  Enterprise Bean自身のコンストラクタの処理を記述します  */
    }
    // receive SessionContext
    public void setSessionContext(SessionContext ctx)
                throws EJBException {
            ・・・
            /* コンテナによって保守されているコンテキストへの  */
            /* アクセスを行い、必要な情報を取得します          */
    }
    // startup work
    public void ejbCreate(String sn)
        throws EJBException,
        CreateException {
            ・・・
            /* インスタンス変数の初期化や、データベースや          */
            /* ファイルのopenなど、Enterprise Beanのインスタンスが */
            /* createされたときの処理を記述します                  */
    }
    // business method
   public String business(String s)
        throws EJBException {
            ・・・          /* ビジネスメソッドの処理を記述します */
    }
    // termination work
    public void ejbRemove()
        throws EJBException {
            ・・・     /* インスタンスがremoveされるときの処理を記述します */
                       /* openしているリソースはclose処理が必要です        */
    }
    // work for passivation
    public void ejbPassivate()
        throws EJBException {
            ・・・                        /* 本バージョンでは呼ばれません */
    }
    // work for activation
    public void ejbActivate()
        throws EJBException {
            ・・・                        /* 本バージョンでは呼ばれません */
    }
}
```

12.6.2 使用できるメソッド

その他のメソッドについて

ejbCreateとビジネスメソッド以外に、Enterprise Beanクラスには、コンテナが処理の各フェーズで呼び出す以下のメソッドがあります。
以下のメソッドにはユーザが任意の処理を指定できます。以下のメソッドはすべて省略できません。

以下に使用できるメソッドの一覧を示します。

メソッド名	内容
setSessionContext	コンテナによって保守されているコンテキストへのアクセスを与えます。
ejbPassivate	EJBアプリケーションのインスタンスが二次領域(注1)に退避されるときに呼び出されます(ただし本バージョンでは呼ばれません)。

メソッド名	内容
ejbActivate	EJBアプリケーションのインスタンスが二次領域から復元されるときに呼び出されます（ただし本バージョンでは呼ばれません）。
ejbRemove	EJBアプリケーションのインスタンスが終了するときに呼び出されます。

注1) EJBアプリケーションのインスタンスを退避するための領域

SessionContextインタフェースのメソッド

SessionContextは、コンテナによって保守されているコンテキストへのアクセスを与えます。
このインタフェースは、setSessionContextのパラメタとして指定することにより取得できます。それによりSessionContextより拡張されるEJBContextインタフェースのメソッドを使用できます。

以下に使用できるメソッドの一覧を示します。

メソッド名	内容
getEJBObject()	現在インスタンスに関連付けられているEJB objectの参照を取得します。
getEJBLocalObject()	現在インスタンスに関連付けられているEJB local objectの参照を取得します。
getEJBHome()	Enterprise BeanのHomeインタフェースを取得します。
getEJBLocalHome()	Enterprise BeanのLocalHomeインタフェースを取得します。
getCallerPrincipal()	呼出し側のjava.security.Principalを取得します。
isCallerInRole(java.lang.String)	呼出し側がセキュリティーロールに割り当てられているか判定します。
setRollbackOnly()	カレントトランザクションを“rollback”にマークします。
getRollbackOnly()	そのトランザクションが“rollback only”でマークされているか判定します。
getUserTransaction()	トランザクション区別インタフェースを取得します。

上記以外のメソッドは本バージョンでは使用上制限があります。

ejbPassivateとejbActivateについて

ejbPassivateとejbActivateは、本バージョンでは呼ばれません。ただし、省略できないのでメソッドの記述はしてください。

ejbRemoveについて

EJBアプリケーションのインスタンスが終了するため、ファイルやデータベースのclose処理を行ってください。

EJBサービスでは、createメソッドで作成したSTATEFUL Session Beanのインスタンスを再利用することにより、サーバへの接続時間を短縮しています。そのため、STATEFUL Session Beanのcreate実行時に、コンストラクタの実行とフィールドの初期化がされないことがあります。
初期化処理はejbCreateメソッド内に記述してください。

12.6.3 Enterprise Beanクラスのメソッドが実行可能な操作

以下の操作については、Beanクラスのメソッドごとに実行できる操作が異なります。

- javax.ejb.SessionContextインタフェースのメソッド実行
- javax.transaction.UserTransactionインタフェースのメソッド実行
- Enterprise Bean Environmentの利用
- データベースへのアクセス
- 他のEJBアプリケーションへのアクセス

以下に実行できる操作についてまとめます。以下に示す以外の操作を実行した場合は、java.lang.IllegalStateExceptionが発生する場合があります。

- STATEFUL Session Beanの場合
- STATELESS Session Beanの場合

STATEFUL Session Beanの場合

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
コンストラクタ	なし	なし
setSessionContext	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome • Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス (注1) • リソースマネージャ(データベースなど)へのアクセス (注1)	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome • Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス (注1) • リソースマネージャ(データベースなど)へのアクセス (注1)
ejbCreate ejbRemove ejbActivate ejbPassivate	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome — getCallerPrincipal — isCallerInRole — getEJBObject — getEJBLocalObject • Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス • リソースマネージャ(データベースなど)へのアクセス	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome — getCallerPrincipal — isCallerInRole — getEJBObject — getEJBLocalObject — getUserTransaction • javax.transaction.UserTransactionメソッド • Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス • リソースマネージャ(データベースなど)へのアクセス
ビジネスメソッド	• javax.ejb.SessionContextメソッド — getEJBHome	• javax.ejb.SessionContextメソッド — getEJBHome

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
	ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getEJBObject ー getEJBLocalObject ・ Enterprise Bean Environmentの利用 ・ 他のEJBアプリケーションへのアクセス ・ リソースマネージャ(データベースなど)へのアクセス	ー getEJBLocalHome ー getCallerPrincipal ー isCallerInRole ー getEJBObject ー getEJBLocalObject ー getUserTransaction ・ javax.transaction.UserTransactionメソッド ・ Enterprise Bean Environmentの利用 ・ 他のEJBアプリケーションへのアクセス ・ リソースマネージャ(データベースなど)へのアクセス ・ データベースへのアクセス
afterBegin beforeCompletion	・ javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getEJBObject ー getEJBLocalObject ・ Enterprise Bean Environmentの利用 ・ 他のEJBアプリケーションへのアクセス ・ リソースマネージャ(データベースなど)へのアクセス	なし (トランザクション種別がBeanの場合、SessionSynchronizationインタフェースのAPIは使用できません)。(注2)
afterCompletion	・ javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー isCallerInRole ー getEJBObject ー getEJBLocalObject	

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
	・Enterprise Bean Environmentの利用	

注1) EJB規約では許可されていないため、アプリケーションの移行性を重視する場合には推奨しません。
注2) 詳細については、“10.4.4 Session Beanのsynchronization機能”を参照してください。

STATELESS Session Beanの場合

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
コンストラクタ	なし	なし
setSessionContext	・javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal (注1) ー isCallerInRole (注1) ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス (注1) ・リソースマネージャ(データベースなど)へのアクセス (注1)	・javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal (注1) ー isCallerInRole (注1) ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス (注1) ・リソースマネージャ(データベースなど)へのアクセス (注1)
ejbCreate	・javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal (注1) ー isCallerInRole (注1) ー getEJBObject ー getEJBLocalObject ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス (注1)	・javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal (注1) ー isCallerInRole (注1) ー getEJBObject ー getEJBLocalObject ー getUserTransaction ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・javax.transaction.UserTransactionメソッド

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
	• リソースマネージャ(データベースなど)へのアクセス (注1)	• Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス (注1) • リソースマネージャ(データベースなど)へのアクセス (注1)
ejbRemove	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome — getEJBObject — getEJBLocalObject — getTimerService • Enterprise Bean Environmentの利用	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome — getEJBObject — getEJBLocalObject — getUserTransaction — getTimerService • javax.transaction.UserTransactionメソッド • Enterprise Bean Environmentの利用
ビジネスメソッド(Remoteインタフェースもしくは Localインタフェースで呼び出された場合)	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome — getCallerPrincipal — getRollbackOnly — isCallerInRole — setRollbackOnly — getEJBObject — getEJBLocalObject — getTimerService • javax.ejb.TimerServiceメソッド • javax.ejb.Timerメソッド • Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス • リソースマネージャ(データベースなど)へのアクセス	• javax.ejb.SessionContextメソッド — getEJBHome — getEJBLocalHome — getCallerPrincipal — isCallerInRole — getEJBObject — getEJBLocalObject — getUserTransaction — getTimerService • javax.ejb.TimerServiceメソッド • javax.ejb.Timerメソッド • javax.transaction.UserTransactionメソッド • Enterprise Bean Environmentの利用 • 他のEJBアプリケーションへのアクセス • リソースマネージャ(データベースなど)へのアクセス
ビジネスメソッド(サービスエンドポイ	• javax.ejb.SessionContextメソッド — getEJBHome	• javax.ejb.SessionContextメソッド — getEJBHome

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
ントインタフェースで呼び出された場合)	ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getEJBObject ー getEJBLocalObject ー getMessageContext ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス ・リソースマネージャ(データベースなど)へのアクセス	ー getEJBLocalHome ー getCallerPrincipal ー isCallerInRole ー getEJBObject ー getEJBLocalObject ー getUserTransaction ー getMessageContext ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・javax.transaction.UserTransactionメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス ・リソースマネージャ(データベースなど)へのアクセス
ejbTimeout	・javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー isCallerInRole ー getRollbackOnly ー setRollbackOnly ー getEJBObject ー getEJBLocalObject ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス ・リソースマネージャ(データベースなど)へのアクセス	・javax.ejb.SessionContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー isCallerInRole ー getEJBObject ー getEJBLocalObject ー getUserTransaction ー getTimerService ・javax.ejb.TimerServiceメソッド ・javax.ejb.Timerメソッド ・javax.transaction.UserTransactionメソッド ・Enterprise Bean Environmentの利用 ・他のEJBアプリケーションへのアクセス ・リソースマネージャ(データベースなど)へのアクセス

注1) EJB規約では許可されていないため、アプリケーションの移行性を重視する場合には推奨しません。

注意

SessionContextインタフェースのgetRollbackOnlyメソッドとsetRollbackOnlyメソッドは、トランザクション内で実行されるEnterprise Beanメソッドの中でだけ使用してください。

インスタンスがトランザクションに関連付けられていないときにこれらのメソッドが呼び出された場合は、java.lang.IllegalStateExceptionが発生する場合があります。

第13章　Entity Beanの実装

本章では、Entity Beanプログラミング方法について説明します。

13.1 Entity Beanの概要

以下に、Entity Beanの概要について説明します。

13.1.1 Entity Beanの形態

Entity BeanにはBMP(Bean-managed persistence)とCMP(Container-managed persistence)の2種類が存在し、それぞれ以下の特長があります。作成するEntity Beanの形態を選択してください。

BMP（Bean-managed persistence）

EJBアプリケーション内に適切なデータベース操作文を発行する処理を記述することにより、EJBアプリケーション自身がデータの永続化を行います。
状況に応じたきめ細かいデータベース管理ができます。

CMP（Container-managed persistence）

CMP1.1

コンテナがデータの永続化を行います。このため、EJBアプリケーション自身にデータベース操作文を記述することなく、データベースにアクセスできます。

CMP2.0

CMP2.0ではCMP1.1の機能に加えて、Entity Beanが別のEntity Beanとの関係を保持する機能を備えたことにより、CMP1.1と比べてはるかに複雑化したデータをEntity Beanに関連づけることができます。
EJBアプリケーションにデータベース操作文を記述する必要がないため、ポータビリティ性の高いアプリケーションが容易に開発できます。

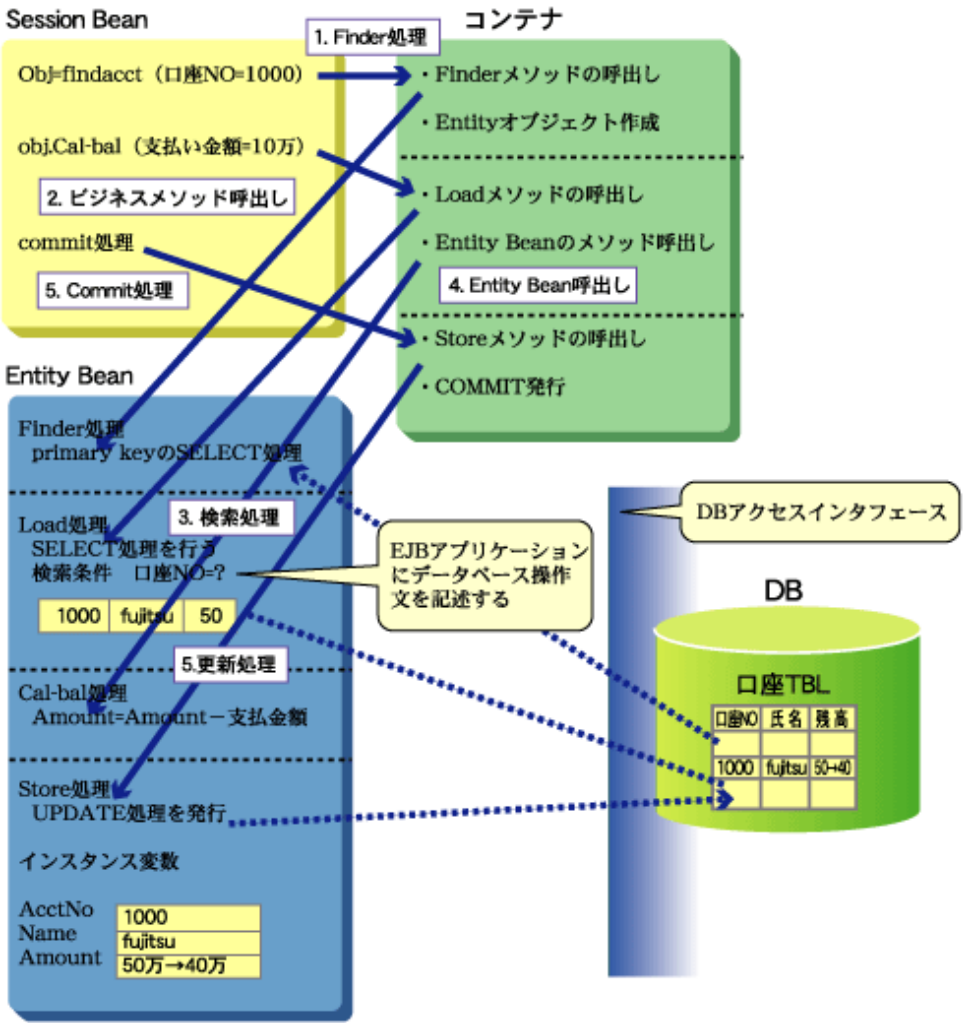
CMP1.1とCMP2.0の違いについては、CMP1.1とCMP2.0の差異を参照してください。

処理イメージ

以下に、口座引き落とし処理を例にして、BMPとCMPのEntity Beanを利用した処理イメージを示します。

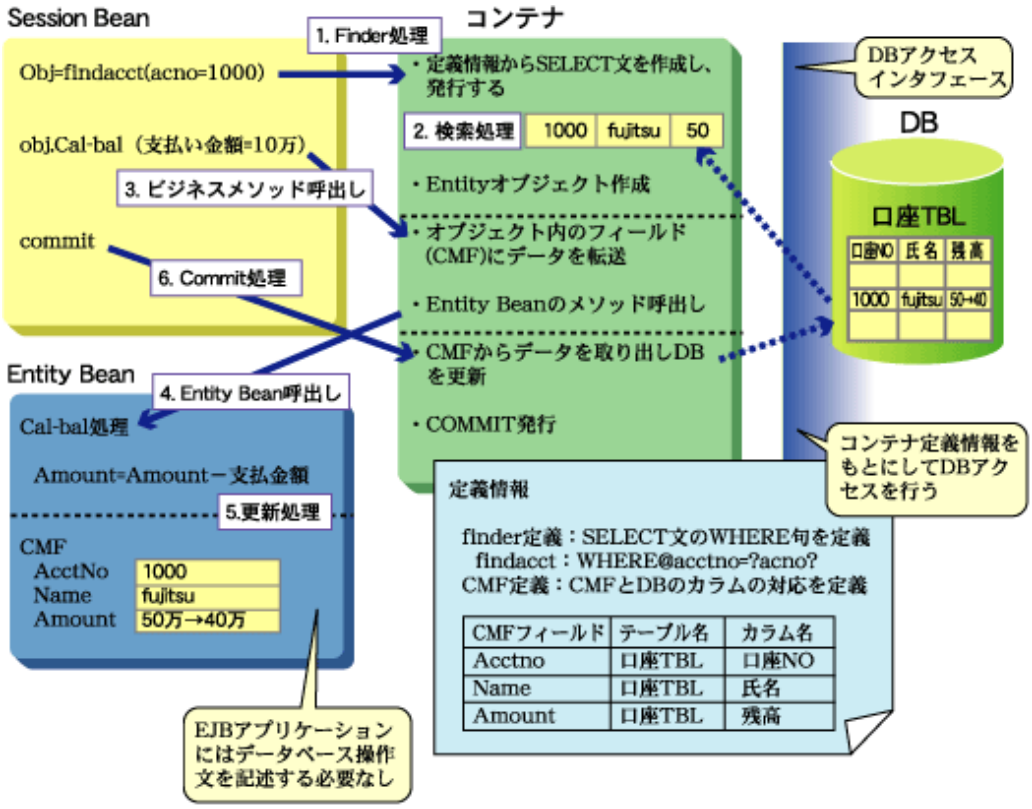
- BMPの処理イメージ
- CMP1.1の処理イメージ
- CMP2.0の処理イメージ

BMPの処理イメージ



1. 呼出し元Beanは、口座NOを指定してfinderメソッドを呼び出します。コンテナは、Entity Beanのfinder処理を呼び出し、Entityオブジェクトを検索します。
2. 呼出し元Beanは、支払金額に10万を設定し、ビジネスメソッド(Cal_bal)を呼び出します。コンテナは、Entity Beanのビジネスメソッドを呼び出す前にEntity BeanのLoad処理を呼び出します。
3. Load処理では、データベースの検索を実施し、検索結果をEntityオブジェクトとして保持します。
4. コンテナは、ビジネスメソッドを呼び出します。
5. ビジネスメソッドは、Amount（残高）を更新します。
6. 呼出し元Beanは、commitメソッドを呼び出します。コンテナは、Entity BeanのStore処理を呼び出し、Store処理ではデータベースの更新を実施します。

CMP1.1の処理イメージ

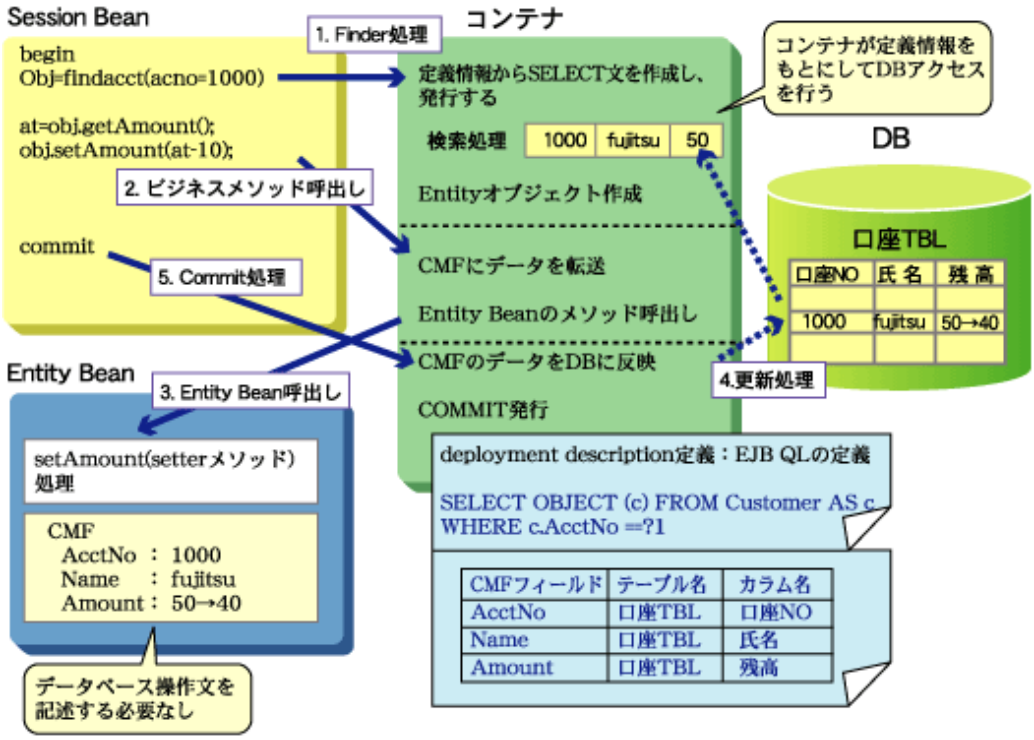


1. 呼出し元Beanは、口座NOを指定してfinderメソッドを呼び出し、Entityオブジェクトの検索をコンテナに依頼します。
2. 検索結果は、コンテナにより、Entityオブジェクトとして保持されます。
3. 呼出し元Beanは、支払金額に10万を設定し、ビジネスメソッド(Cal_bal)を呼び出します。
4. コンテナは、2.で保持した検索結果データをCMFへ転送し、ビジネスメソッドを呼び出します。
5. ビジネスメソッドは、Amount（残高）を更新します。
6. 呼出し元Beanからのcommit指示により、コンテナはデータベースの更新を実施します。

注意

CMPマッピング定義の定義情報についての詳細はInterstage管理コンソールのヘルプを参照してください。

CMP2.0の処理イメージ



1. 呼出し元Beanは、口座NOを指定してfinderメソッドを呼び出し、Entityオブジェクトの検索をコンテナに依頼します。検索結果は、コンテナにより、Entityオブジェクトとして保持されます。
2. 呼出し元Beanは、アクセッサメソッド(setAmount)を呼び出して、残高から10万を引いた値を新たに設定します。
3. コンテナは、コンテナが実装するアクセッサメソッドを呼び出します。
4. アクセッサメソッドは、Amount（残高）を更新します。
5. 呼出し元Beanからのcommit指示により、コンテナはデータベースの更新を実施します。

注意

CMP2.0の定義情報についての詳細はInterstage管理コンソールのヘルプを参照してください。

CMP1.1とCMP2.0の差異

以下に、CMP1.1とCMP2.0の差異を表で示します。

	CMP1.1	CMP2.0
relationshipの管理	アプリケーション内で解決してください。	deployment descriptorファイルにリレーションを定義することで、CMP2.0のBean間の関係を定義できます。定義された関係はコンテナが管理します。
finderメソッドの検索条件	Interstage StudioでEnterprise Java Bean開発時に指定します。また、Interstage管理コンソールまたはejbdefimportコマンドで編集できます。Interstage Studio以外で開発したEnterprise Java Beanの場合にはInterstage管理コンソールまたはejbdefimportコマンドで指定します。	Enterprise Java Bean開発時にdeployment descriptorファイルにEJB QLを定義します。

	CMP1.1	CMP2.0
CMFの更新	ビジネスメソッド内でEnterprise Beanクラスに定義したCMFを直接更新します。	deployment descriptorに定義したCMFに対応する抽象アクセッサメソッドをEnterprise Beanクラスに定義して、必ず抽象アクセサメソッドでCMFを更新します。
finderメソッドの復帰値の型	java.util.Collection型とjava.util.Enumeration型を使用できます。	java.util.Collection型が使用できます。
ejbSelectメソッドの使用	使用できません。	使用できます。

13.1.2 クラスファイルの作成

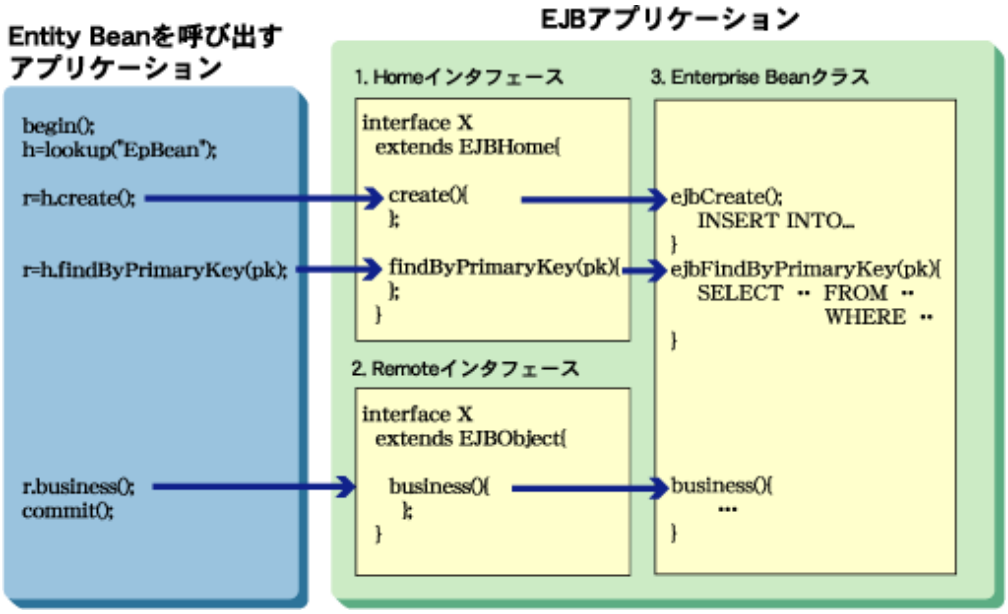
Entity BeanのEJBアプリケーションは、以下の6つのクラスファイルから成り立っています。
それぞれのクラス名はユーザの任意の名前を付けることができます。

下表1～4のクラスファイルはEnterprise Beanを使用するためのインタフェースです。

	クラスファイル名	内容
1	Homeインタフェース	EJB objectの生成・検索を行うインタフェースを定義します。
2	LocalHomeインタフェース	同一JavaVM上で、EJB objectの生成を行うインタフェースを定義します。
3	Remoteインタフェース	ユーザのビジネスメソッドを呼び出すためのインタフェースを定義します。
4	Localインタフェース	同一JavaVM上で、ユーザのビジネスメソッドを呼び出すためのインタフェースを定義します。
5	Enterprise Beanクラス	サーバで実際に処理を行うクラスで、ユーザの開発するメソッドを実装します。
6	Primary Keyクラス	EJB objectの一意性を表すためのクラスです。

相関図

以下に例として、クラスファイルとクライアントアプリケーションの相関図を示します。



[図の説明]

1. HomeインタフェースはEJB objectを作成するためのcreateメソッド、EJB objectまたはEJB objectのコレクションを検索するためのfinderメソッドを定義します。
   クライアントより、createメソッドを呼び出すと、EJB objectを作成し、同じ形式で記述されているEnterprise BeanクラスのejbCreateメソッド、続いて、ejbPostCreateメソッドを呼び出します。
   クライアントより、finderメソッドを呼び出すと、対応するEnterprise BeanクラスのejbFind<METHOD>を呼び出し、その検索結果より、EJB objectまたはEJB objectのコレクションを返却します。
2. Remoteインタフェースは、Enterprise Beanクラスのビジネスメソッドを使用するためのインタフェースを定義します。インタフェースで定義するビジネスメソッドの形式は、Enterprise Beanクラスに実装されるビジネスメソッドと同じ形式にしてください。
3. Entity Beanクラスを実装するクラスであり、ejbCreateメソッド、ejbFind<METHOD>メソッドおよびビジネスメソッドを実装します。

CMPのEntity Beanでは、6つのクラスファイルを作成するほかに、データベースとアクセスするための情報を定義してください。これをCMP定義と呼びます。

CMP定義の詳細については、“13.1.3 CMP定義”を参照してください。

13.1.3 CMP定義

CMPのEntity Beanでは、データベースとアクセスするための以下の情報を定義します。
CMP定義を行うことで、CMPのEnterprise Beanにデータベース操作文を記述する必要がなくなります。

定義情報	内容
CMFマッピング定義	Entity Beanの永続化フィールドとデータベースのカラムの対応づけを行います。この永続化フィールドをCMF(Container managed field)と呼びます。
finder定義 （CMP1.1の場合）	finderメソッドの検索条件をSQLのWHERE句の形式で定義します。検索条件は、単一のテーブルを対象とした検索とORDER BY句を使用した検索が利用できます。
CMRマッピング定義 （CMP2.0の場合）	データベースのテーブル間の関連づけを行います。このデータベースのテーブルとテーブルの間の関係を、relationship（リレーション）と呼び、relationshipフィールドをCMR(Container managed relationship)と呼びます。

定義情報	内容
クエリ定義 （CMP2.0の場合）	FROM句やSELECT句（WHERE句）を使用した、1つまたは複数のEJBオブジェクトを検索する文をEJB QL文として定義できます。

定義する項目

以下に、それぞれ定義する項目について表で示します。

CMFマッピング定義

項目	内容
データソース名	登録してあるデータソース名を指定します。
スキーマ名	使用するデータベースのスキーマ名を指定します。
デーブル名	使用するテーブル名を指定します。
DBカラム名	永続化フィールドと対応付けるDBカラム名を指定します。

finder定義（CMP1.1の場合）

項目	内容
検索条件	finderメソッドの検索条件を指定します。

CMRマッピング定義（CMP2.0の場合）

項目	内容
スキーマ名	リレーションが1:多、多:多の場合に使用するJoinテーブルのスキーマ名を指定します。
テーブル名	リレーションが1:多、多:多の場合に使用するJoinテーブル名を指定します。
カラム名	リレーションのあるEJBアプリケーションの外部キーと対応付けるDBカラム名を指定します。

クエリ定義（CMP2.0の場合）

項目	内容
検索メソッド名	検索するメソッドの（finder/select）を指定します。
検索メソッドパラメータ	クエリ内のパラメタの型を指定します。
返却EJBオブジェクト型	ejbSelectメソッドが返すオブジェクト（LocalまたはRemote）を指定します。デフォルト値はLocalです。
EJB QL文	検索メソッドに対応するEJB QL文を指定します。

定義方法

CMP1.1の場合

CMP定義は、Interstage管理コンソールの［ワークユニット］>［IJServer名］>［EJBアプリケーション］>［アプリケーション環境定義]の［CMFマッピング定義］と［finder定義］で設定します。

設定方法の詳細は、Interstage管理コンソールのヘルプを参照してください。

Interstage Studioを使用してEJBアプリケーションの開発を行う場合、XMLエディタを使用して定義できます。

詳細については、“Interstage Studio ユーザーズガイド”を参照してください。

CMP2.0の場合

CMP定義は、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義]の[CMFマッピング定義]と[CMRマッピング定義]で設定します。
クエリ定義はEJBアプリケーション作成時にあらかじめ定義しておいてください。

設定方法の詳細は、Interstage管理コンソールのヘルプを参照してください。

13.1.4 オブジェクト操作とデータベース操作の関係

以下に、Entity Beanを呼び出すアプリケーションからのオブジェクト操作とEntity Beanでのデータベース操作の対応例を示します。

- BMPの場合
- CMP1.1の場合
- CMP2.0の場合

BMPの場合

オブジェクト操作	コンテナから呼び出されるEntity Beanのメソッド	Entity Beanのメソッドで行うデータベースの操作
インスタンスの生成 ・ create()	ejbCreate()	インスタンスに対応する行を表へINSERT
	ejbPostCreate()	なし
インスタンスの消去 ・ remove()	ejbActivate()	なし
	ejbLoad()	インスタンスに対応する行を表からSELECT
	ejbRemove()	インスタンスに対応する行を表からDELETE
インスタンスのプライマリキー値による検索 ・ findByPrimaryKey()	ejbFindByPrimaryKey()	プライマリキー値を検索条件として表からSELECT
インスタンスの条件値による検索 ・ find<METHOD>()	ejbFind<METHOD>()	条件値を検索条件として表から1～複数のプライマリキーをSELECT
ejbHomeメソッドの呼出し	ejbActivate()	なし
	ejbHome<METHOD>()	任意
	ejbPassivate()	なし
ビジネスメソッドの呼出し	ejbActivate()	なし
	ejbLoad()	インスタンスに対応する行を表からSELECT
	ビジネスメソッド()	なし
トランザクションのコミット ・ commit()	ejbStore()	インスタンスの変更がある場合は、表中の対応する行をUPDATE
	ejbPassivate()	なし
トランザクションのロールバック ・ rollback()	ejbPassivate()	なし

CMP1.1の場合

オブジェクト操作	コンテナから呼び出されるEntity Beanのメソッド	Entity Beanのメソッドで行うデータベースの操作
インスタンスの生成 ・ create()	ejbCreate()	なし
	ejbPostCreate()	なし
インスタンスの消去 ・ remove()	ejbActivate()	なし
	ejbLoad()	なし
	ejbRemove()	なし
インスタンスのプライマリキー値による検索 ・ findByPrimaryKey()	ejbActivate()	なし
	ejbLoad()	なし
インスタンスの条件値による検索 ・ find<METHOD>()	ejbActivate()	なし
	ejbLoad()	なし
ビジネスメソッドの呼出し	ejbActivate()	なし
	ejbLoad()	なし
	ビジネスメソッド()	なし
トランザクションのコミット ・ commit()	ejbStore()	なし
	ejbPassivate()	なし
トランザクションのロールバック ・ rollback()	ejbPassivate()	なし

CMP2.0の場合

オブジェクト操作	コンテナから呼び出されるEntity Beanのメソッド	Entity Beanのメソッドで行うデータベースの操作
インスタンスの生成 ・ create()	ejbCreate()	なし
	ejbPostCreate()	なし
インスタンスの消去 ・ remove()	ejbActivate()	なし
	ejbLoad()	なし
	ejbRemove()	なし
インスタンスのプライマリキー値による検索 ・ findByPrimaryKey()	ejbActivate()	なし
	ejbLoad()	なし
インスタンスの条件値による検索 ・ find<METHOD>()	ejbActivate()	なし
	ejbLoad()	なし
ejbHomeメソッドの呼出し	ejbActivate()	なし
	ejbHome<METHOD>()	なし
	ejbPassivate()	なし
抽象アクセッサメソッドの呼出し	ejbActivate()	なし
	ejbLoad()	なし

オブジェクト操作	コンテナから呼び出されるEntity Beanのメソッド	Entity Beanのメソッドで行うデータベースの操作
ejbSelectメソッドの呼出し	ejbActivate()	なし
	ejbLoad()	なし
ビジネスメソッドの呼出し	ejbActivate()	なし
	ejbLoad()	なし
	ビジネスメソッド()	なし
トランザクションのコミット ・ commit()	ejbStore()	なし
	ejbPassivate()	なし
トランザクションのロールバック ・ rollback()	ejbPassivate()	なし

上記以外に、以下の場合にはejbPassivateメソッドが呼び出されることがあります。

- インスタンスを活性化した後にエラーや例外が発生した場合
- メソッド実行時にインスタンスを活性化したが、同一のプライマリキーを持つインスタンスがすでに活性化されていた場合
- 活性化されたインスタンス数が初期起動インスタンス数に達し、コンテナがEJB objectとインスタンスの切離し(非活性化)が必要と判断した場合

13.2 Homeインタフェースの作成

Homeインタフェースには、Entity Beanの生成および検索を行うためのメソッドを定義します。

Homeインタフェースは、RMIインタフェースの形で記述します。
また、EJB2.0規約以降に準拠したEJBアプリケーションでは、BMPだけejbHomeメソッドと呼ばれるビジネスメソッドをHomeインタフェースに記述できます。Homeインタフェースには、“create”、“find”、“remove”以外で始まる任意の名前を記述できます。

例えば、データベースのレコード件数を取得するビジネスメソッドなどが記述できるため、データベース操作をより自由にできます。
Interstage Studioを使用して開発を行った場合、Homeインタフェースのひな形が自動生成されます。

以下に、Homeインタフェースの記述形式を示します。

記述形式

```
public interface Homeインタフェース名 extends javax.ejb.EJBHome
{
    public Remoteインタフェースタイプ create(任意の引数)
                throws javax.ejb.CreateException,
                       java.rmi.RemoteException;

    public Remoteインタフェースタイプ findByPrimaryKey(primary key)
                throws java.rmi.RemoteException,
                       javax.ejb.FinderException;

    public Remoteインタフェースタイプ find<METHOD>(任意の引数)
                throws java.rmi.RemoteException,
                       javax.ejb.FinderException;

    public java.util.Enumeration find<METHOD>(任意の引数)
                throws java.rmi.RemoteException,
                       javax.ejb.FinderException;

    public java.util.Collection find<METHOD>(任意の引数)
```

```
                throws java.rmi.RemoteException,
                       javax.ejb.FinderException;

}
```

規約

Homeインタフェースは以下の規約を満たしていなければなりません。

- インタフェースはjavax.ejb.EJBHomeインタフェースを継承しなければなりません。
- このインタフェースで定義されたメソッドはRMI over IIOPの規約に従わなくてはなりません。
- Homeインタフェースには、以下のメソッドを定義します。
  - 0個以上のcreateメソッド
  - 1個以上のfinderメソッド
  - 0個以上のejbHomeメソッド

createメソッドの規約

createメソッドは、インスタンスを生成するためのメソッドです。createメソッドが呼び出されることによって、INSERT文が実行され、データベースに新しい行が追加されます。

なお、EJB2.0規約以降に準拠したEJBアプリケーションでは、BMPだけcreateで始まる任意の名前を記述できます。これにより、createメソッド名に意味のある名前を記述できるため、EJBアプリケーションをわかりやすく作成できます。

createメソッドは以下の規約を満たしていなければなりません。

- Homeインタフェースは、0個以上のcreateメソッドを定義します。
- 各createメソッドはcreateという名前で始まらなければなりません。
- 各createメソッドは、引数のタイプと数が、Enterprise Beanクラスに定義された ejbCreateメソッドの1つに一致しなくてはなりません。
- createメソッドの返却値のタイプは、Enterprise BeanのRemoteインタフェースタイプでなければなりません。
- throws句には、以下の例外をすべて定義する必要があります。
  - 一致するEnterprise BeanクラスのejbCreateメソッドに定義されたすべての例外
  - javax.ejb.CreateException
  - java.rmi.RemoteException

finderメソッドの規約

finderメソッドは、特定のインスタンス、またはインスタンスの集合を検索するためのメソッドです。finderメソッドに指定した引数を検索条件にしたSELECT文が実行されます。

finderメソッドは以下の規約を満たしていなければなりません。

- Homeインタフェースに、以下の条件を満たすfindByPrimaryKeyメソッドを定義しなければなりません。
  - findByPrimaryKeyという名前であること
  - 引数は1つでプライマリキータイプであること
  - 返却値はEnterprise BeanのRemoteインタフェースタイプであること
- 各finderメソッドは、find<METHOD>という名前(例:findLargeAccount)でなければなりません。
- BMPの場合、各finderメソッドは、引数のタイプと数が、Enterprise Beanクラスに定義されたejbFind<METHOD>の1つに一致しなくてはなりません。

- 返却値のタイプは、以下のどれかでなければなりません。
  - Enterprise BeanのRemoteインタフェースタイプ
  - java.util.Enumerationインタフェース(BMP、CMP1.1のみ)
  - java.util.Collectionインタフェース
- throws句には、以下の例外をすべて定義する必要があります。
  - 一致するEnterprise BeanクラスのejbFind<METHOD>メソッドに定義されたすべての例外
  - javax.ejb.FinderException
  - java.rmi.RemoteException

注意

Homeインタフェースの作成における注意事項

throws句に宣言する例外の数は、5以下にしてください。多数宣言されている場合、配備時間に影響します。

13.2.1 記述例

SampleHomeという名前のインタフェースを設定した場合の例を以下に示します。

```
package Sample;

  import javax.ejb.*;
  import java.rmi.RemoteException;
  import java.util.*;

  public interface SampleHome extends EJBHome
  {

      /* createメソッドの定義 */
      public SampleRemote create(Integer code, String name, String desc  )
                          throws CreateException, RemoteException;

      /* findByPrimaryKeyメソッドの定義 */
      public SampleRemote findByPrimaryKey(SampleBeanPrimaryKey primaryKey)
                          throws FinderException, RemoteException;

      /* find<METHOD>メソッドの定義 */
      public Enumeration findSampleBigCode(Integer bigcode)
                         throws FinderException, RemoteException;

      /* find<METHOD>メソッドの定義 */
      public Collection findSampleByName(String name)
                         throws FinderException, RemoteException;
  }
```

13.2.2 使用できるメソッド

Homeインタフェースは、EJBHomeインタフェースを継承しているため、Entity Beanを呼び出すアプリケーションはcreateメソッド、finderメソッド以外にも以下のメソッドが使用できます。

メソッド名	内容
getEJBMetaData()	Enterprise BeanのEJBMetaDataインタフェースを取得します。
getHomeHandle()	Home objectのハンドルを取得します。

メソッド名	内容
remove(javax.ejb.Handle)	ハンドルに対応するEJB objectを消去します。
remove(java.lang.Object)	プライマリキーに対応するEJB objectを消去します。

13.3 LocalHomeインタフェースの作成

LocalHomeインタフェースには、Entity Beanの生成および検索を行うためのメソッドを定義します。

LocalHomeインタフェースは、Java VM内で呼ばれるインタフェースであるためRMIの形で記述する必要はありません。また、EJB2.0規約以降に準拠したEJBアプリケーションでは、BMPとCMP2.0においてejbHomeメソッドと呼ばれるビジネスメソッドをLocalHomeインタフェースに記述できます。LocalHomeインタフェースには、“create”、“find”、“remove”以外で始まる任意の名前を記述できます。

例えば、データベースのレコード件数を取得するビジネスメソッドなどが記述できるため、データベース操作をより自由にできます。

以下に、LocalHomeインタフェースの記述形式を示します。

記述形式

```
public interface LocalHomeインタフェース名 extends javax.ejb.EJBLocalHome
{
    public Localインタフェースタイプ create(任意の引数)
                throws javax.ejb.CreateException;

    public Localインタフェースタイプ create<METHOD>(任意の引数)
                throws javax.ejb.CreateException;

    public Localインタフェースタイプ findByPrimaryKey(primary key)
                throws javax.ejb.FinderException;

    public Localインタフェースタイプ find<METHOD>(任意の引数)
                throws javax.ejb.FinderException;

    public java.util.Enumeration find<METHOD>(任意の引数)
                throws javax.ejb.FinderException;

    public java.util.Collection find<METHOD>(任意の引数)
                throws javax.ejb.FinderException;

    public 返却値 ビジネスメソッド(任意の引数) ;

}
```

規約

LocalHomeインタフェースは以下の規約を満たしていなければなりません。

- インタフェースはjavax.ejb.EJBLocalHomeインタフェースを継承しなければなりません。
- LocalHomeインタフェースには、以下のメソッドを定義します。
  - 0個以上のcreateメソッド
  - 1個以上のfinderメソッド
  - 0個以上のejbHomeメソッド

createメソッドの規約

createメソッドは、インスタンスを生成するためのメソッドです。createメソッドが呼び出されることによって、INSERT文が実行され、データベースに新しい行が追加されます。

なお、EJB2.0規約以降に準拠したEJBアプリケーションでは、BMPだけcreateで始まる任意の名前を記述できます。これにより、createメソッド名に意味のある名前を記述できるため、EJBアプリケーションをわかりやすく作成できます。

createメソッドは以下の規約を満たしていなければなりません。

- LocalHomeインタフェースは、0個以上のcreateメソッドを定義します。
- 各createメソッドはcreateという名前で始まらなければなりません。
- 各createメソッドは、引数のタイプと数が、Enterprise Beanクラスに定義された ejbCreateメソッドの1つに一致しなくてはなりません。
- throws句には、以下の例外をすべて定義する必要があります。
  - 一致するEnterprise BeanクラスのejbCreateメソッドに定義されたすべての例外
  - javax.ejb.CreateException

finderメソッドの規約

finderメソッドは、特定のインスタンス、またはインスタンスの集合を検索するためのメソッドです。finderメソッドに指定した引数を検索条件にしたSELECT文が実行されます。

finderメソッドは以下の規約を満たしていなければなりません。

- Homeインタフェースに、以下の条件を満たすfindByPrimaryKeyメソッドを定義しなければなりません。
  - findByPrimaryKeyという名前であること
  - 引数は1つでプライマリキータイプであること
  - 返却値はEnterprise BeanのLocalインタフェースタイプであること
- 各finderメソッドは、find<METHOD>という名前(例:findLargeAccount)でなければなりません。
- BMPの場合、各finderメソッドは、引数のタイプと数が、Enterprise Beanクラスに定義されたejbFind<METHOD>の1つに一致しなくてはなりません。
- 返却値のタイプは、以下のどれかでなければなりません。
  - Enterprise BeanのLocalインタフェースタイプ
  - java.util.Enumerationインタフェース(BMP、CMP1.1のみ)
  - java.util.Collectionインタフェース
- throws句には、以下の例外をすべて定義してください。
  - 一致するEnterprise BeanクラスのejbFind<METHOD>メソッドに定義されたすべての例外
  - javax.ejb.FinderException

13.3.1 記述例

SampleLocalHomeという名前のインタフェースを設定した場合の例を以下に示します。

```
package Sample;

  import javax.ejb.*;
  import java.util.*;

  public interface SampleLocalHome extends EJBLocalHome
  {

      /* createメソッドの定義 */
      public SampleLocal create(Integer code, String name, String desc  )
```

```
                        throws CreateException;

    /* findByPrimaryKeyメソッドの定義 */
    public SampleLocal findByPrimaryKey(SampleBeanPrimaryKey primaryKey)
                        throws FinderException;

    /* find<METHOD>メソッドの定義 */
    public Enumeration findSampleBigCode(Integer bigcode)
                       throws FinderException;

    /* find<METHOD>メソッドの定義 */
    public Collection findSampleByName(String name)
                       throws FinderException;
}
```

13.3.2 使用できるメソッド

LocalHomeインタフェースは、EJBLocalHomeインタフェースを継承しているため、Entity Beanを呼び出すアプリケーションはcreateメソッド、finderメソッド、ejbHomeメソッド以外にも以下のメソッドが使用できます。

メソッド名	内容
remove(java.lang.Object)	プライマリキーに対応するEJB local objectを消去します。

13.4 Remoteインタフェースの作成

Remoteインタフェースには、ビジネスメソッドを定義します。ビジネスメソッドは、業務処理に応じて自由に定義できます。Remoteインタフェースは、RMIインタフェースの形で記述します。

Interstage Studioを使用して開発を行った場合、Remoteインタフェースのひな形が自動生成されます。

以下に、Remoteインタフェースの記述形式を示します。

記述形式

```
public interface  Remoteインタフェース名  extends  javax.ejb.EJBObject
{
    public 返却値 ビジネスメソッド(任意の引数)
        throws java.rmi.RemoteException;

    その他、任意のビジネスメソッドの定義;
}
```

規約

Remoteインタフェースは、以下の規約を満たしていなければなりません。

- インタフェースはjavax.ejb.EJBObjectインタフェースを継承しなければなりません。
- このインタフェースで定義されたメソッドはRMI over IIOPの規約に従わなくてはなりません。
- 定義するビジネスメソッドは、Enterprise Beanクラスのビジネスメソッドと同じ名前でなければなりません。
- Enterprise Beanクラスのビジネスメソッドのメソッド名、引数の数と型、返却値の型が同じでなければなりません。
- throws句には、以下の例外をすべて定義してください。
  - 一致するEnterprise Beanクラスのビジネスメソッドに定義されたすべての例外
  - java.rmi.RemoteException

注意

Remoteインタフェースの作成における注意事項

throws句に宣言する例外の数は、5以下にしてください。多数宣言されている場合、配備時間に影響します。

13.4.1 記述例

SampleRemoteという名前のインタフェースを設定した場合の例を以下に示します。

```
package Sample;

import            java.rmi.RemoteException;
import            javax.ejb.*;

public interface SampleRemote extends EJBObject
{

    /* ビジネスメソッドの定義 */
    public String  getName()                    throws RemoteException;
    public String  getDesc()                    throws RemoteException;
    public void    setName(String NameValue)    throws RemoteException;
    public void    setDesc(String DescValue)    throws RemoteException;
}
```

13.4.2 使用できるメソッド

Remoteインタフェースは、EJBObjectインタフェースを継承しているので、Entity Beanを呼び出すアプリケーションは、以下のメソッドも使用できます。

メソッド名	内容
getEJBHome()	Enterprise Bean のHomeインタフェースを取得します。
getHandle()	EJB objectのハンドルを取得します。
getPrimaryKey()	EJB objectのプライマリキーを取得します。
isIdentical(javax.ejb.EJBObject)	パラメタとして与えられたEJB objectが、呼び出されたEJB objectと一致するか判定します。
remove()	EJB objectを消去します。

13.5 Localインタフェースの作成

Localインタフェースには、ビジネスメソッドを定義します。ビジネスメソッドは、業務処理に応じて自由に定義できます。LocalHomeインタフェースは、Java VM内で呼ばれるインタフェースであるためRMIの形で記述する必要はありません。

以下に、Localインタフェースの記述形式を示します。

記述形式

```
public interface  Localインタフェース名  extends  javax.ejb.EJBLocalObject
{
    public 返却値 ビジネスメソッド(任意の引数) ;
    その他、任意のビジネスメソッドの定義;
}
```

規約

Localインタフェースは以下の規約を満たしていなければなりません。

- インタフェースはjavax.ejb.EJBLocalObjectインタフェースを継承しなければなりません。
- 定義するビジネスメソッドは、Enterprise Beanクラスのビジネスメソッドと同じ名前でなければなりません。
- Enterprise Beanクラスのビジネスメソッドのメソッド名、引数の数と型、返却値の型が同じでなければなりません。
- throws句には、以下の例外をすべて定義してください。
  - 一致するEnterprise Beanクラスのビジネスメソッドに定義されたすべての例外

注意

Localインタフェースの作成における注意事項

CMRを使用したEntity Beanは、Localインタフェースを使用して呼び出されなければならないため、必ずLocalインタフェースを使用してください。

13.5.1 記述例

Localインタフェースの記述例

SampleLocalという名前のインタフェースを設定した場合の例を以下に示します。

```
package Sample;

import            javax.ejb.*;

public interface SampleLocal extends EJBLocalObject
{

    /* ビジネスメソッドの定義 */
    public String  getName();
    public String  getDesc();
    public void    setName(String NameValue);
    public void    setDesc(String DescValue);
}
```

13.5.2 使用できるメソッド

Localインタフェースは、EJBLocalObjectインタフェースを継承しています。クライアントアプリケーションより使用できるメソッドの一覧を以下に示します。

以下に使用できるメソッドの一覧を示します。

メソッド名	内容
getEJBLocalHome()	Enterprise Bean のLocalHomeインタフェースを取得します。
getPrimaryKey()	EJB local objectのハンドルを取得します。
isIdentical(javax.ejb.EJBLocalObject)	パラメタとして与えられた EJB local objectが、呼び出された EJB local objectと一致するか判定します。
remove()	EJB local objectを消去します。

13.6 BMPのEnterprise Beanクラスの作成

BMPのEnterprise Beanクラスには、ユーザの開発するビジネスメソッドだけでなく、オブジェクトの永続化処理を実現するためのメソッドにデータベースアクセス処理を記述してください。

Interstage Studioを使用して開発を行った場合、Enterprise Beanクラスのひな形が自動生成されます。

13.6.1 BMPのEnterprise Beanクラスの概要

BMPのEnterprise Beanクラスの規約

BMPのEnterprise Beanクラスは以下の規約を満たしていなければなりません。

- javax.ejb.EntityBeanインタフェースを実装していなければなりません。
- publicとして定義されていなければなりません。
- ejbCreateメソッド、ejbPostCreateメソッド、ejbFind<METHOD>メソッドおよびビジネスメソッドが実装されていなければなりません。
- ejbFindByPrimaryKeyメソッドを定義しなければなりません。
- 他のEnterprise BeanのRemoteインタフェースをこのEnterprise Beanの中で指定してもかまいません。

BMPのEnterprise Beanクラスに実装するメソッド

BMPのEnterprise Beanクラスには、ビジネスメソッドのほかに、コンテナが処理の各フェーズで呼び出す以下のメソッドを実装します。

- setEntityContextメソッド
- unsetEntityContextメソッド
- ejbCreateメソッド
- ejbPostCreateメソッド
- ejbFindByPrimaryKeyメソッド
- ejbFind<METHOD>メソッド（※<METHOD>は任意の文字列）
- ejbRemoveメソッド
- ejbLoadメソッド
- ejbStoreメソッド
- ejbActivateメソッド
- ejbPassivateメソッド
- ejbHomeメソッド

記述構成

以下にBMPのEnterprise Beanクラスの記述構成を示します。それぞれの記述内容については、“永続化フィールドの記述”以降を参照してください。

Interstage Studioを使用すると、永続化フィールド、各メソッドおよびビジネスメソッドのひな形などが生成されます。

public class <Enterprise Beanクラス名> implements javax.ejb.EntityBean
{

- 永続化フィールドの記述
- setEntityContextおよびunsetEntityContextメソッドの記述
- ejbCreateおよびejbPostCreateメソッドの記述
- ejbFindByPrimaryKeyメソッドの記述
- ejbFind<METHOD>メソッドの記述
- ejbRemoveメソッドの記述
- ejbLoadおよびejbStoreメソッドの記述

- ejbActivateおよびejbPassivateメソッドの記述
- ビジネスメソッドの記述
- ejbHomeメソッドの記述

}

13.6.2 永続化フィールドの記述

データベースとデータをやりとりするための格納域を定義します。データベースのカラムに対応するようにインスタンス変数として記述します。これを永続化フィールドと呼びます。

記述例

```
private Integer code;
private String  name;
private String  desc;
```

13.6.3 setEntityContextメソッドおよびunsetEntityContextメソッドの記述

setEntityContextメソッドは、インスタンス生成時に呼び出されます。

unsetEntityContextメソッドは、インスタンス消去時に呼び出されます。

一般的には、setEntityContextメソッドでは、1回だけ初期化するような処理(例えばデータソースの獲得処理)、unsetEntityContextでは、回収処理などの後処理を記述します。また、setEntityContextメソッドでは、引数で渡されるEntityContextのリファレンスをインスタンス内に保持するように記述します。

lookupを使用したデータソース獲得については、“4.10 オブジェクトの参照方法”を参照してください。

Entity Beanから別のEntity Beanを呼び出す場合、呼出し元のEntity BeanのsetEntityContextメソッド内に、呼出し先のEntity Beanをlookupする処理を記述することはできません。他のメソッド内に記述してください。

記述例

```
public void setEntityContext(javax.ejb.EntityContext ctx) throws javax.ejb.EJBException
{
    context = ctx;
    // データソースの獲得
    try {
        javax.naming.Context ic = new javax.naming.InitialContext();
        dataSource = (DataSource)ic.lookup("java:comp/env/jdbc/SYMFO");
    }
    catch( Exception ex ) {
        throw new javax.ejb.EJBException(ex);
    }
}
public void unsetEntityContext() throws javax.ejb.EJBException
{
    context = null;
    dataSource = null;
}
```

13.6.4 ejbCreateメソッドおよびejbPostCreateメソッドの記述

コンテナがデータベースにデータを追加するときに呼び出すメソッドです。

Entity Beanを呼び出すEJBアプリケーションがcreateメソッドを呼び出すと、コンテナはcreateメソッドに対応するejbCreateメソッドを呼び出します。ejbPostCreateメソッドは、ejbCreateメソッドが呼び出された後コンテナによって呼び出されます。

記述する処理の概要

一般的に、ejbCreateメソッドには、以下の処理を記述します。

- 永続化フィールドを入力引数で初期化する処理
- データベースにレコードを挿入（INSERT）する処理
- プライマリキーオブジェクトを作成して返却する処理

ejbPostCreateメソッドには通常、処理を記述する必要はありません。

ejbCreateメソッドの規約

ejbCreateメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbCreateでなければなりません。
  EJB2.0規約以降に準拠したEJBアプリケーションでは、Homeインタフェースにcreateで始まる任意の名前を記述できます。
  この場合、ejbCreate<任意の名前>でなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はプライマリキータイプでなければなりません。
- メソッドの引数は、RMI over IIOPの規約に従わなくてはなりません。
  また、引数のタイプと数はHomeインタフェースのcreateメソッドに一致していなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.CreateException
  - javax.ejb.DuplicateKeyException

ejbPostCreateメソッドの規約

ejbPostCreateメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbPostCreateでなければなりません。
  EJB2.0規約以降に準拠したEJBアプリケーションでは、Homeインタフェースにcreateで始まる任意の名前を記述できます。
  この場合、ejbPostCreate<任意の名前>でなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- メソッド引数は、対応するejbCreateメソッドの引数と同じでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.CreateException
  - javax.ejb.DuplicateKeyException

記述例

```
public SampleBeanPrimaryKey ejbCreate(Integer codeValue,
                                      String nameValue,
                                      String descValue)
        throws javax.ejb.CreateException, javax.ejb.EJBException
{
    // 永続化フィールドの初期化
    code = codeValue;
    name = nameValue;
    desc = descValue;
    Connection connection = null;
    PreparedStatement psInsert = null;
    int rows = 0;
    try
    {
        // データベースへレコードを追加する
        connection = dataSource.getConnection();
        psInsert = connection.prepareStatement(
                   "INSERT INTO SAMPLESCM.SAMPLETBL(ID,NAME,DESC) VALUES(?,?,?)");
        psInsert.setObject(1,code);
        psInsert.setString(2,name);
        psInsert.setString(3,desc);
        rows = psInsert.executeUpdate();
    }
    catch(SQLException e)
    {
        throw new EJBException(e.getMessage());
    }
    finally
    {
        try
        {
            if (psInsert != null)
                psInsert.close();
            if (connection != null)
                connection.close();
        }
        catch(Exception e) {}
    }
    // プライマリキーオブジェクトの作成
    SampleBeanPrimaryKey pk = new SampleBeanPrimaryKey();
    pk.code = code;
    return pk;
}

public void ejbPostCreate(Integer codeValue, String nameValue, String descValue)
        throws javax.ejb.CreateException, javax.ejb.EJBException
{
}
```

13.6.5 ejbFindByPrimaryKeyメソッドの記述

ejbFindByPrimaryKeyメソッドは、プライマリキーを返却するメソッドです。

Entity Beanを呼び出すEJBアプリケーションがfindByPrimaryKeyメソッドを呼び出すと、コンテナはfindByPrimaryKeyメソッドに対応するejbFindByPrimaryKeyメソッドを呼び出します。

記述する処理の概要

ejbFindByPrimaryKeyメソッドには、以下の処理を記述します。引数にはプライマリキーオブジェクトを指定します。

- 引数(プライマリキーオブジェクト)を検索キーにしてプライマリキーを検索(SELECT)する処理

・ プライマリキーオブジェクトを作成する処理

ejbFindByPrimaryKeyメソッドの規約

ejbFindByPrimaryKeyメソッドは以下の規約を満たしていなければなりません。

・ メソッドはpublicとして定義されていなければなりません。

・ 返却値はEnterprise Beanのプライマリキータイプでなければなりません。

・ メソッドの引数は、RMI over IIOPの規約に従わなくてはなりません。

・ throws句には以下の例外を定義することができます。

　－ 任意のEJBアプリケーション固有の例外

　－ javax.ejb.EJBException

　－ javax.ejb.FinderException

　－ javax.ejb.ObjectNotFoundException

記述例

```
public SampleBeanPrimaryKey ejbFindByPrimaryKey(SampleBeanPrimaryKey SampleBeanPrimaryKeyValue)
        throws javax.ejb.FinderException,
               javax.ejb.ObjectNotFoundException,
               javax.ejb.EJBException
{
    Connection connection = null;
    PreparedStatement psSelect = null;
    ResultSet rs = null;
    int rows = 0;
    SampleBeanPrimaryKey pk = null;
    try
    {
        // プライマリキーの検索
        connection = dataSource.getConnection();
        psSelect = connection.prepareStatement("SELECT ID FROM SAMPLESCM.SAMPLETBL WHERE ID = ?");
        psSelect.setObject(1,SampleBeanPrimaryKeyValue.code);
        rs = psSelect.executeQuery();
        while(rs.next())
        {
            rows++;
            // プライマリキーオブジェクトの作成
            pk = new SampleBeanPrimaryKey();
            pk.code = new Integer(rs.getInt(1));
        }
    }
    catch(SQLException e)
    {
        throw new EJBException(e.getMessage());
    }
    finally
    {
        try
        {
            if (rs != null)
                rs.close();
            if (psSelect != null)
                psSelect.close();
            if (connection != null)
                connection.close();
        }
        catch(Exception e) {}
```

```
    }
    if (rows == 0)
    {
        throw new ObjectNotFoundException ("No Record Found");
    }
    else if (rows > 1)
    {
        throw new FinderException ("Many Records Found");
    }
    return pk;
}
```

13.6.6 ejbFind<METHOD>メソッドの記述

ejbFind<METHOD>メソッドは、プライマリキーを返却するメソッドです。複数のプライマリキーを返却できます。

Entity Beanを呼び出すEJBアプリケーションがfind<METHOD>メソッドを呼び出すと、コンテナはfind<METHOD>メソッドに対応するejbFind<METHOD>メソッドを呼び出します。

記述する処理の概要

ejbFind<METHOD>メソッドには、以下の処理を記述します。引数には、任意の検索キーを指定します。

- 引数を検索キーにしてプライマリキーを検索(SELECT)する処理
- プライマリキーオブジェクトを作成する処理

また、複数のプライマリキーを返却するメソッドで検索結果が0件の場合、要素が0個のEnumerationやCollectionを返却するようにしてください。

ejbFind<METHOD>メソッドの規約

ejbFind<METHOD>メソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbFindプレフィックスで始まらなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はEnterprise Beanのプライマリキータイプかまたはプライマリキータイプのオブジェクトのコレクションでなければなりません。
- メソッドの引数は、RMI over IIOPの規約に従わなくてはなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.FinderException

記述例

```
public Enumeration ejbFindSampleBigCode(Integer bigcode)
        throws javax.ejb.FinderException, javax.ejb.EJBException
{
    Connection connection = null;
    PreparedStatement psSelect = null;
    ResultSet rs = null;
    int rows = 0;
    SampleBeanPrimaryKey pk = null;
    Vector v = new Vector();
    try
    {
        // プライマリキーの検索
        connection = dataSource.getConnection();
```

```
            psSelect = connection.prepareStatement("SELECT ID FROM SAMPLESCM.SAMPLETBL WHERE ID > ?");
            psSelect.setObject(1,bigcode);
            rs = psSelect.executeQuery();
            while(rs.next())
            {
                rows++;
                // プライマリキーオブジェクトの作成
                pk = new SampleBeanPrimaryKey();
                pk.code = new Integer(rs.getInt(1));
                v.addElement(pk);
            }
        }
        catch(SQLException e)
        {
            throw new EJBException(e.getMessage());
        }
        finally
        {
            try
            {
                if (rs != null)
                    rs.close();
                if (psSelect != null)
                    psSelect.close();
                if (connection != null)
                    connection.close();
            }
            catch(Exception e) {}
        }

        return v.elements();
    }
```

13.6.7 ejbRemoveメソッドの記述

コンテナがデータベースからデータを削除するときに呼び出すメソッドです。

Entity Beanを呼び出すEJBアプリケーションがremoveメソッドを呼び出すと、コンテナはremoveメソッドに対応するejbRemoveメソッドを呼び出します。

記述する処理の概要

ejbRemoveメソッドには、以下の処理を記述します。

- 該当するインスタンスのプライマリキーオブジェクトをキーにして、データベースからレコードを削除(DELETE)する処理

ejbRemoveメソッドの規約

ejbRemoveメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- throws句には以下の例外を定義することができます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.RemoveException

記述例

```
public void ejbRemove() throws javax.ejb.EJBException, javax.ejb.RemoveException
{
    Connection connection = null;
    PreparedStatement psDelete = null;
    SampleBeanPrimaryKey pk = null;
    int rows = 0;
    try
    {
        // レコードの削除
        connection = dataSource.getConnection();
        psDelete = connection.prepareStatement("DELETE FROM SAMPLESCM.SAMPLETBL WHERE ID = ?");
        pk = (SampleBeanPrimaryKey)context.getPrimaryKey();
        psDelete.setObject(1,pk.code);
        rows = psDelete.executeUpdate();
    }
    catch(SQLException e)
    {
        throw new EJBException(e.getMessage());
    }
    finally
    {
        try
        {
            if (psDelete != null)
                psDelete.close();
            if (connection != null)
                connection.close();
        }
        catch(Exception e) {}
    }
    if (rows != 1)
        throw new RemoveException("ejbRemove failed");
}
```

13.6.8 ejbLoadメソッドおよびejbStoreメソッドの記述

ejbLoadメソッドおよびejbStoreメソッドは、インスタンスの内容をデータベースと同期させるとき(例えば、ビジネスメソッドを呼び出す前やトランザクションをコミットする前)に、コンテナから呼び出されます。

記述する処理の概要

ejbLoadメソッドには、以下の処理を記述します。

- 該当するインスタンスのプライマリキーのレコードをデータベースから検索(SELECT)する処理
- 検索した結果を永続化フィールドに設定する処理

ejbStoreメソッドには、以下の処理を記述します。

- 永続化フィールドのデータを使用して、データベースのレコードを更新(UPDATE)する処理永続化フィールドが初期化された状態(create時)や、更新されていない場合は、更新する処理(UPDATE)を実行しないようにすることで性能向上が図れます。

ejbLoadメソッドおよびejbStoreメソッドの規約

ejbLoadメソッドおよびejbStoreメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。

・ throws句には以下の例外を定義できます。

　－ 任意のEJBアプリケーション固有の例外

　－ javax.ejb.EJBException

　－ javax.ejb.NoSuchEntityException

記述例

```
public void ejbLoad() throws javax.ejb.EJBException,javax.ejb.NoSuchEntityException
{
    Connection connection = null;
    PreparedStatement psSelect = null;
    ResultSet rs = null;
    SampleBeanPrimaryKey pk = null;
    int rows = 0;
    try
    {
        // レコードの検索
        connection = dataSource.getConnection();
       psSelect = connection.prepareStatement("SELECT ID,NAME,DESC FROM SAMPLESCM.SAMPLETBL WHERE ID = ?");
        pk = (SampleBeanPrimaryKey)context.getPrimaryKey();
        psSelect.setObject(1,pk.code);
        rs = psSelect.executeQuery();
        while(rs.next())
        {
            rows++;
            // 検索結果を永続化フィールドに設定
            code = new Integer(rs.getInt(1));
            name = rs.getString(2);
            desc = rs.getString(3);
        }
    }
    catch(SQLException e)
    {
        throw new EJBException(e.getMessage());
    }
    finally
    {
        try
        {
            if (rs != null)
                rs.close();
            if (psSelect != null)
                psSelect.close();
            if (connection != null)
                connection.close();
        }
        catch(Exception e) {}
    }
    if (rows == 0)
    {
        throw new NoSuchEntityException("No Record Found");
    }
    else if (rows > 1)
    {
        throw new EJBException("Many Records Found");
    }
}

public void ejbStore() throws javax.ejb.EJBException,javax.ejb.NoSuchEntityException
{
    Connection connection = null;
```

```
      PreparedStatement psUpdate = null;
      SampleBeanPrimaryKey pk = null;
      int rows = 0;
      try
      {
          // レコードの更新
          connection = dataSource.getConnection();
          psUpdate = connection.prepareStatement("UPDATE SAMPLESCM.SAMPLETBL SET NAME = ?,DESC = ? WHERE ID
= ?");
          pk = (SampleBeanPrimaryKey)context.getPrimaryKey();
          psUpdate.setString(1,name);
          psUpdate.setString(2,desc);
          psUpdate.setObject(3,pk.code);
          rows = psUpdate.executeUpdate();
      }
      catch(SQLException e)
      {
          throw new EJBException(e.getMessage());
      }
      finally
      {
          try
          {
              if (psUpdate != null)
                  psUpdate.close();
              if (connection != null)
                  connection.close();
          }
          catch(Exception e) {}
      }

      if (rows != 1)
          throw new EJBException("ejbStore failed");
  }
```

13.6.9 ejbActivateメソッドおよびejbPassivateメソッドの記述

ejbActivateメソッドおよびejbPassivateメソッドは、コンテナが、EJB objectとインスタンスの関連付け(活性化)や切離し(非活性化)が必要と判断したときに呼び出します。

コンテナは、活性化を行った後、ejbActivateメソッドを呼び出します。また、非活性化時は、コンテナは、ejbPassivateメソッドを呼び出した後、非活性化を行います。

記述する処理の概要

ejbActivateメソッドおよびejbPassivateメソッドには、特に処理を記述する必要はありません。

活性化または非活性化時に、必要であれば任意の処理を記述してください。

ejbActivateメソッドおよびejbPassivateメソッドの規約

ejbActivateメソッドおよびejbPassivateメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException

記述例

```
public void ejbActivate() throws javax.ejb.EJBException
{
}

public void ejbPassivate() throws javax.ejb.EJBException
{
}
```

13.6.10 ejbHomeメソッドの記述

ejbHomeメソッドは、クライアントからejbHomeメソッドを呼び出したときに呼び出されます。

コンテナは、Entity Beanのインスタンスプールからインスタンスを1つ選択してejbHomeメソッドを実行し、ejbHomeメソッドの実行が終了した後にインスタンスをプーリングに返却します。

記述する処理の概要

ejbHomeメソッドには、特定のEntity Beanインスタンスを対象としないビジネスロジックを記述します。

ejbHomeの規約

ejbHomeメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- メソッドはstaticとして定義してはいけません。
- Enterprise BeanクラスのejbHome<METHOD>メソッドは、HomeインタフェースまたはLocalHomeインタフェースの<METHOD>メソッドの引数の数と型、返却値の型が同じでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException

記述例

```
public void ejbHomeTotalReservation() throws javax.ejb.EJBException
{
    // ビジネスロジックを記述する。
}
```

13.6.11 ビジネスメソッドの記述

Remoteインタフェースで宣言したメソッドを実装します。

また、Homeインタフェースで宣言したejbHomeメソッドを実装します。

ビジネスメソッドの規約

ビジネスメソッドは以下の規約を満たしていなければなりません。

- メソッド名は任意ですが、EJBのAPIで定義されているメソッドの名前と重複してはなりません。
  また、Homeインタフェースで宣言したejbHomeメソッドは、“ejbHome”で始まるメソッド名にしてください。
- メソッドはpublicとして宣言されていなければなりません。
- メソッドの引数と返却値は、RMI over IIOPの規約に従わなくてはなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外

- javax.ejb.EJBException
- javax.ejb.NoSuchEntityException

記述例

データベースのカラム“NAME”(対応する永続化フィールドは“name”)の値を取り出すビジネスメソッドの例

```
public  String  fgetName() throws javax.ejb.EJBException
{
    return  name;
}
```

13.6.12 例外処理

BMPのEnterprise Beanクラスで記述する例外の種類には以下の2つがあります。

- Application exceptions
- System exceptions

Enterprise Beanクラスに実装する各メソッドでは、以下の指針で例外処理を記述してください。

Application exceptions

Application exceptionsは、アプリケーションレベルの例外を通知する場合に使用します。ビジネスメソッドの入力引数が不正な値である場合や、createメソッド実行時に一意性制約違反が発生した場合など、アプリケーションのロジックで例外とする場合です。

Exception	記述するメソッド	指針
javax.ejb.CreateException	• ejbCreate • ejbPostCreate	createメソッドの引数が不正であるなどユーザロジックで例外を発生させる場合
javax.ejb.DuplicateKeyException	• ejbCreate • ejbPostCreate	INSERTの実行で一意性制約違反が発生した場合
javax.ejb.FinderException	• ejbFindByPrimaryKey • ejbFind<METHOD>	単一の検索結果を返却するメソッドで検索結果が複数件の場合や、ユーザロジックで例外とする場合
javax.ejb.ObjectNotFoundException	• ejbFindByPrimaryKey • ejbFind<METHOD>	検索結果が0件の場合 (単一の検索結果を返却するメソッドの場合のみ、このexceptionを使用できます。)
javax.ejb.RemoveException	• ejbRemove	ユーザロジックで例外とする場合
ユーザ例外	全メソッド	ユーザロジックで例外とする場合

System exceptions

System exceptionsは、アプリケーションで処理できないシステムレベルの例外を通知する場合に使用します。Application exceptions以外のデータベースのエラーや、他のEJBアプリケーションの呼出しから予期しないRemoteExceptionが返却された場合です。

Enterprise Beanクラスからjavax.ejb.EJBExceptionを返却すると、コンテナはjava.rmi.RemoteExceptionに変換して呼出し元に返却します。

Exception	記述するメソッド	指針
javax.ejb.NoSuchEntityException	・ ejbLoad() ・ ejbStore() ・ ビジネスメソッド	インスタンスに対応するデータがデータベースから削除されている場合
javax.ejb.EJBException	全メソッド	NoSuchEntityException以外の場合 例えば、データベース操作文でエラーになった場合など

13.6.13 使用できるメソッド

Enterprise Beanクラスの各メソッドでは、以下のインタフェースのメソッドを使用できます。

EntityContextインタフェースのメソッド

EntityContextは、コンテナによって保守されているコンテキストへのアクセスを与えます。このインタフェースは、setEntityContextのパラメタとして指定することにより取得できます。それによりEntityContextより拡張されるEJBContextインタフェースのメソッドを使用できます。

メソッド名	内容
getEJBObject()	EJB objectを返却します。
getPrimaryKey()	EJB objectのプライマリキーを返却します。
getCallerPrincipal()	呼出し側のjava.security.Principalを取得します。
getEJBHome()	Enterprise BeanのHomeインタフェースを取得します。
getRollbackOnly()	そのトランザクションが“rollback only”でマークされているか判定します。
getUserTransaction()	トランザクション区別インタフェースを取得します。
isCallerInRole(java.lang.String)	呼出し側がセキュリティーロールに割り当てられているか判定します。
setRollbackOnly()	カレントトランザクションを“rollback”にマークします。

13.6.14 Enterprise Beanクラスのメソッドが実行可能な操作

以下の操作については、Beanクラスのメソッドごとに実行できる操作が異なります。

- javax.ejb.EntityContextインタフェースのメソッド実行
- javax.transaction.UserTransactionインタフェースのメソッド実行
- Enterprise Bean Environmentの利用
- データベースへのアクセス
- 他のEJBアプリケーションへのアクセス

以下に実行できる操作についてまとめます。以下に示す以外の操作を実行した場合は、java.lang.IllegalStateExceptionが発生する場合があります。

メソッド名	実行可能な操作
コンストラクタ	なし
setEntityContext	・ javax.ejb.EntityContextメソッド 　－ getEJBHome

メソッド名	実行可能な操作
	ー getEJBLocalHome ・ Enterprise Bean Environmentの利用 ・ 他のEJBアプリケーションへのアクセス (注1)
unsetEntityContext	・ javax.ejb.EntityContextメソッド ー getEJBHome ー getEJBLocalHome ・ Enterprise Bean Environmentの利用
ejbCreate	・ javax.ejb.EntityContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getTimerService ・ javax.ejb.TimerServiceメソッド ・ javax.ejb.Timerメソッド ・ Enterprise Bean Environmentの利用 ・ リソースマネージャ(データベースなど)へのアクセス ・ 他のEJBアプリケーションへのアクセス
ejbPostCreate	・ javax.ejb.EntityContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getEJBObject ー getEJBLocalObject ー getPrimaryKey ー getTimerService ・ javax.ejb.TimerServiceメソッド ・ javax.ejb.Timerメソッド ・ Enterprise Bean Environmentの利用 ・ リソースマネージャ(データベースなど)へのアクセス ・ 他のEJBアプリケーションへのアクセス
ejbRemove	・ javax.ejb.EntityContextメソッド

メソッド名	実行可能な操作
	ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getEJBObject ー getEJBLocalObject ー getPrimaryKey ー getTimerService ・ javax.ejb.TimerServiceメソッド ・ javax.ejb.Timerメソッド ・ Enterprise Bean Environmentの利用 ・ リソースマネージャ(データベースなど)へのアクセス ・ 他のEJBアプリケーションへのアクセス
ejbFind<METHOD>	・ javax.ejb.EntityContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ・ Enterprise Bean Environmentの利用 ・ リソースマネージャ(データベースなど)へのアクセス ・ 他のEJBアプリケーションへのアクセス
ejbHome<METHOD>	・ javax.ejb.EntityContextメソッド ー getEJBHome ー getEJBLocalHome ー getCallerPrincipal ー getRollbackOnly ー isCallerInRole ー setRollbackOnly ー getTimerService ・ javax.ejb.TimerServiceメソッド ・ javax.ejb.Timerメソッド ・ Enterprise Bean Environmentの利用 ・ リソースマネージャ(データベースなど)へのアクセス

メソッド名	実行可能な操作
	・ 他のEJBアプリケーションへのアクセス
ejbActivate ejbPassivate	・ javax.ejb.EntityContextメソッド 　― getEJBHome 　― getEJBLocalHome 　― getEJBObject 　― getEJBLocalObject 　― getPrimaryKey 　― getTimerService ・ Enterprise Bean Environmentの利用
ejbLoad ejbStore	・ javax.ejb.EntityContextメソッド 　― getEJBHome 　― getEJBLocalHome 　― getCallerPrincipal 　― getRollbackOnly 　― isCallerInRole 　― setRollbackOnly 　― getEJBObject 　― getEJBLocalObject 　― getPrimaryKey 　― getTimerService ・ javax.ejb.TimerServiceメソッド ・ javax.ejb.Timerメソッド ・ Enterprise Bean Environmentの利用 ・ リソースマネージャ（データベースなど）へのアクセス ・ 他のEJBアプリケーションへのアクセス
ビジネスメソッド	・ javax.ejb.EntityContextメソッド 　― getEJBHome 　― getEJBLocalHome 　― getCallerPrincipal 　― getRollbackOnly 　― isCallerInRole 　― setRollbackOnly 　― getEJBObject 　― getEJBLocalObject 　― getPrimaryKey 　― getTimerService ・ javax.ejb.TimerServiceメソッド

メソッド名	実行可能な操作
	• javax.ejb.Timerメソッド • Enterprise Bean Environmentの利用 • データベースへのアクセス • 他のEJBアプリケーションへのアクセス

注1) EJB規約では許可されていないため、アプリケーションの移行性を重視する場合には推奨しません。

注意

EntityContextインタフェースのgetRollbackOnlyメソッドとsetRollbackOnlyメソッドは、トランザクション内で実行されるEnterprise Beanメソッドの中でだけ使用してください。

インスタンスがトランザクションに関連付けられていないときにメソッドが呼び出された場合は、java.lang.IllegalStateExceptionが発生する場合があります。

13.7 CMP1.1のEnterprise Beanクラスの作成

CMP1.1のEnterprise Beanクラスには、ユーザの開発するビジネスメソッドだけでなく、オブジェクトの永続化処理を実現するためのメソッドを記述します。

CMP1.1のEnterprise Beanでは、CMP定義にfinderメソッドの検索条件や、永続化フィールドとデータベースのカラムの対応を定義するため、メソッドにはデータ操作文を記述する必要がありません。このため、ポータビリティ性の高いアプリケーションが容易に開発できます。

CMP定義の詳細については、“13.1.3 CMP定義”を参照してください。

Interstage Studioを使用して開発を行った場合、Enterprise Beanクラスのひな形が自動生成されます。

13.7.1 CMP1.1のEnterprise Beanクラスの概要

CMP1.1のEnterprise Beanクラスの規約

CMP1.1のEnterprise Beanクラスは以下の規約を満たしていなければなりません。

- javax.ejb.EntityBeanインタフェースを実装していなければなりません。
- publicとして定義されていなければなりません。
- ejbCreateメソッド、ejbPostCreateメソッドおよびビジネスメソッドが実装されていなければなりません。
- 他のEnterprise BeanのRemoteインタフェースをこのEnterprise Beanの中で指定してもかまいません。

CMP1.1のEnterprise Beanクラスに実装するメソッド

CMP1.1のEnterprise Beanクラスには、ビジネスメソッドのほかに、コンテナが処理の各フェーズで呼び出す以下のメソッドを実装します。

- setEntityContextメソッド
- unsetEntityContextメソッド
- ejbCreateメソッド
- ejbPostCreateメソッド
- ejbRemoveメソッド
- ejbLoadメソッド
- ejbStoreメソッド

- ejbActivateメソッド
- ejbPassivateメソッド

記述構成

以下にCMP1.1のEnterprise Beanクラスの記述構成を示します。それぞれの記述内容については、“永続化フィールド(CMF)の記述”以降を参照してください。

Interstage Studioを使用すると、永続化フィールド、各メソッドおよびビジネスメソッドのひな形などが生成されます。

public class <Enterprise Beanクラス名> implements javax.ejb.EntityBean
{

- 永続化フィールド(CMF)の記述
- setEntityContextおよびunsetEntityContextメソッドの記述
- ejbCreateおよびejbPostCreateメソッドの記述
- ejbRemoveメソッドの記述
- ejbLoadおよびejbStoreメソッドの記述
- ejbActivateおよびejbPassivateメソッドの記述
- ビジネスメソッドの記述

}

13.7.2 永続化フィールド(CMF)の記述

データベースとデータをやりとりするための格納域を定義します。データベースのカラムに対応するようにインスタンス変数として記述します。これを永続化フィールド(CMF)と呼びます。

CMP1.1では、コンテナがこの永続化フィールドを参照して、データベースとのアクセスを行います。

Interstage管理コンソールのCMFマッピング定義でデータベースのカラムと永続化フィールドの対応づけを行います。Interstage Studioを使用している場合も、対応づけができます。

記述例

```
public Integer code;
public String  name;
public String  desc;
```

13.7.3 setEntityContextメソッドおよびunsetEntityContextメソッドの記述

setEntityContextメソッドは、インスタンス生成時に呼び出されます。unsetEntityContext()メソッドは、インスタンス消去時に呼び出されます。

一般的には、setEntityContextメソッドでは、1回だけ初期化するような処理、unsetEntityContextでは、回収処理などの後処理を記述します。また、setEntityContextメソッドでは、引数で渡されるEntityContextのリファレンスをインスタンス内に保持するように記述します。

注意

Entity Beanから別のEntity Beanを呼び出す場合、呼出し元のEntity BeanのsetEntityContextメソッド内に、呼出し先のEntity Beanをlookupする処理を記述することはできません。他のメソッド内に記述してください。

記述例

```
public void setEntityContext(javax.ejb.EntityContext ctx) throws javax.ejb.EJBException
{
    context = ctx;
}
public void unsetEntityContext() throws javax.ejb.EJBException
{
    context = null;
}
```

13.7.4 ejbCreateメソッドおよびejbPostCreateメソッドの記述

コンテナがデータベースにデータを追加するときに呼び出すメソッドです。

Entity Beanを呼び出すEJBアプリケーションがcreateメソッドを呼び出すと、コンテナはcreateメソッドに対応するejbCreateメソッドを呼び出します。コンテナはejbCreateメソッドが完了すると、データベースに対してINSERT文を発行し、ejbPostCreateメソッドを呼び出します。

記述する処理の概要

ejbCreateメソッドには、以下の処理を記述します。

- 永続化フィールドを入力引数で初期化する処理

ejbPostCreateメソッドには通常、処理を記述する必要はありません。

ejbCreateメソッドの規約

ejbCreateメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbCreateでなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はプライマリキータイプでなければなりません。
- メソッドの引数は、RMI over IIOPの規約に従わなくてはなりません。
  また、引数のタイプと数はHomeインタフェースのcreateメソッドに一致していなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.CreateException
  - javax.ejb.DuplicateKeyException

ejbPostCreateメソッドの規約

ejbPostCreateメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbPostCreateでなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- メソッド引数は、対応するejbCreateメソッドの引数と同じでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.CreateException

－ javax.ejb.DuplicateKeyException

記述例

```
public SampleBeanPrimaryKey ejbCreate(Integer codeValue, String nameValue, String descValue)
        throws javax.ejb.EJBException
{
    // 永続化フィールドを入力引数の値で初期化
    code = codeValue;
    name = nameValue;
    desc = descValue;
    return( null );
}

public void ejbPostCreate(Integer codeValue, String nameValue, String descValue)
        throws javax.ejb.EJBException
{
}
```

13.7.5 ejbRemoveメソッドの記述

コンテナがデータベースからデータを削除するときに呼び出すメソッドです。

Entity Beanを呼び出すEJBアプリケーションがremoveメソッドを呼び出すと、コンテナはremoveメソッドに対応するejbRemoveメソッドを呼び出します。

コンテナはejbRemoveメソッドが完了すると、データベースに対してDELETE文を発行します。

ejbRemoveメソッドの規約

ejbRemoveメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- throws句には以下の例外を定義できます。
  － 任意のEJBアプリケーション固有の例外
  － javax.ejb.EJBException
  － javax.ejb.RemoveException

記述する処理の概要

ejbRemoveメソッドには、特に処理を記述する必要はありません。

記述例

```
public void ejbRemove() throws javax.ejb.EJBException
{
    // 以下に削除される前に行いたい処理を記述してください。
}
```

13.7.6 ejbLoadメソッドおよびejbStoreメソッドの記述

ejbLoadメソッドおよびejbStoreメソッドは、インスタンスの内容をデータベースと同期させるとき（例えば、ビジネスメソッドを呼び出す前やトランザクションをコミットする前）に、コンテナから呼び出されます。

ejbLoadメソッドは、ビジネスメソッドが実行される前などに、コンテナから呼び出されます。コンテナは、ejbLoadメソッドを呼び出す前にデータベースに対してSELECT文を発行して永続化フィールドにデータを設定します。

ejbStoreメソッドは、トランザクションをコミットしてデータを更新するときなどに、コンテナから呼び出されます。コンテナはejbStoreメソッドが完了すると、データベースに対してUPDATE文を発行します。

記述する処理の概要

ejbLoadメソッドおよびejbStoreメソッドには、特に処理を記述する必要はありません。

ejbLoadメソッドおよびejbStoreメソッドの規約

ejbLoadメソッドおよびejbStoreメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.NoSuchEntityException

記述例

```
public void ejbLoad() throws javax.ejb.EJBException
{
    // 以下にデータが読み込まれた後に行いたい処理を記述してください。
}

public void ejbStore() throws javax.ejb.EJBException
{
    // 以下にデータを更新する前に行いたい処理を記述してください。
}
```

13.7.7 ejbActivateメソッドおよびejbPassivateメソッドの記述

ejbActivateメソッドおよびejbPassivateメソッドは、コンテナが、EJB objectとインスタンスの関連付け(活性化)や切離し(非活性化)が必要と判断したときに呼び出します。

コンテナは、活性化を行った後、ejbActivateメソッドを呼び出します。また、非活性化時は、コンテナは、ejbPassivateメソッドを呼び出した後、非活性化を行います。

記述する処理の概要

ejbActivateメソッドおよびejbPassivateメソッドには、特に処理を記述する必要はありません。

活性化または非活性化時に、必要であれば任意の処理を記述してください。

ejbActivateメソッドおよびejbPassivateメソッドの規約

ejbActivateメソッドおよびejbPassivateメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException

記述例

```
public void ejbActivate() throws javax.ejb.EJBException
{
}

public void ejbPassivate() throws javax.ejb.EJBException
{
}
```

13.7.8 ビジネスメソッドの記述

Remoteインタフェースで宣言したメソッドを実装します。

ビジネスメソッドの規約

ビジネスメソッドは以下の規約を満たしていなければなりません。

- メソッド名は任意ですが、EJBのAPIで定義されているメソッドの名前と重複してはなりません。
- メソッドはpublicとして宣言されていなければなりません。
- メソッドの引数と返却値は、RMI over IIOPの規約に従わなくてはなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.NoSuchEntityException

記述例

データベースのカラム“NAME”(対応する永続化フィールドは“name”)の値を取り出すビジネスメソッドの例

```
public  String  fgetName() throws javax.ejb.EJBException
{
    return  name;
}
```

13.7.9 例外処理

例外処理は、特に記述する必要はありません。ユーザロジックで例外を発生させる場合は、“13.6.12 例外処理”の指針を参考にしてください。

データベースアクセスエラーなどコンテナ内で発生した例外は、コンテナから呼出し元アプリケーションに例外を通知します。通知する例外および発生契機を以下に示します。

Exception	発生契機	
javax.ejb.DuplicateKeyException	• createメソッド	一意性制約違反が発生した場合
javax.ejb.FinderException	• findByPrimaryKey メソッド • find<METHOD> メソッド	単一の検索結果を返却するメソッドで検索結果が複数件の場合
javax.ejb.ObjectNotFoundException	• findByPrimaryKey メソッド • find<METHOD> メソッド	検索結果が0件の場合 (単一の検索結果を返却するメソッドの場合のみ、このexceptionが通知されます。)

Exception	発生契機	
java.rmi.RemoteException	全メソッド	データベースアクセスエラーやシステムエラーが発生した場合
java.rmi.NoSuchObjectException	ビジネスメソッド	インスタンスに対応するデータがデータベースから削除されている場合

13.7.10 使用できるメソッド

作成するEnterprise Beanクラス内では、以下のインタフェースのメソッドを使用できます。

EntityContextインタフェースのメソッド

EntityContextは、コンテナによって保守されているコンテキストへのアクセスを与えます。このインタフェースは、setEntityContextのパラメタとして指定することにより取得できます。それによりEntityContextより拡張されるEJBContextインタフェースのメソッドを使用できます。

メソッド名	内容
getEJBObject()	EJB objectを返却します。
getEJBLocalObject()	同一JavaVM内のEJB objectを返却します。
getPrimaryKey()	EJB objectのプライマリキーを返却します。
getCallerPrincipal()	呼出し側のjava.security.Principalを取得します。
getEJBHome()	Enterprise BeanのHomeインタフェースを取得します。
getEJBLocalHome()	同一Java内のEnterprise Bean内のHomeインタフェースを取得します。
getRollbackOnly()	そのトランザクションが“rollback only”でマークされているか判定します。
getUserTransaction()	トランザクション区別インタフェースを取得します。
isCallerInRole(java.lang.String)	呼出し側がセキュリティーロールに割り当てられているか判定します。
setRollbackOnly()	カレントトランザクションを“rollback”にマークします。

13.7.11 Enterprise Beanクラスのメソッドが実行可能な操作

Enterprise Beanクラスのメソッドが実行可能な操作については、“13.6.14 Enterprise Beanクラスのメソッドが実行可能な操作”を参照してください。

13.8 CMP2.0のEnterprise Beanクラスの作成

CMP2.0のEnterprise Beanクラスには、ユーザの開発するビジネスメソッドだけでなく、オブジェクトの永続化処理を実現するためのメソッドを記述します。

CMP2.0のEnterprise Beanでは、CMP定義にfinderメソッドの検索条件や、永続化フィールドとデータベースのカラムの対応を定義するため、メソッドにはデータ操作文を記述する必要がありません。このため、ポータビリティ性の高いアプリケーションが容易に開発できます。

CMP定義の詳細については、“13.1.3 CMP定義”を参照してください。

Interstage Studioを使用して開発を行った場合、Enterprise Beanクラスのひな形が自動生成されます。

13.8.1 CMP2.0のEnterprise Beanクラスの概要

CMP2.0のEnterprise Beanクラスの規約

CMP2.0のEnterprise Beanクラスは以下の規約を満たしていなければなりません。

- javax.ejb.EntityBeanインタフェースを実装していなければなりません。
- publicとして定義されていなければなりません。
- ejbCreateメソッド、ejbPostCreateメソッドおよびビジネスメソッドが実装されていなければなりません。
- 他のEnterprise BeanのRemoteインタフェースをこのEnterprise Beanの中で指定してもかまいません。

CMP2.0のEnterprise Beanクラスに実装するメソッド

CMP2.0のEnterprise Beanクラスには、ビジネスメソッドのほかに、コンテナが処理の各フェーズで呼び出す以下のメソッドを実装します。

- setEntityContextメソッド
- unsetEntityContextメソッド
- ejbCreateメソッド
- ejbPostCreateメソッド
- ejbRemoveメソッド
- ejbLoadメソッド
- ejbStoreメソッド
- ejbActivateメソッド
- ejbPassivateメソッド
- ejbHomeメソッド
- 抽象アクセッサメソッド
- ejbSelectメソッド

ポイント

Beanの永続フィールド(CMP／CMRフィールド)に対するアクセスや、更新を行うメソッドはabstract(抽象)で宣言してください。
抽象アクセッサメソッド名は、getまたはsetで始まる永続フィールド名を表します。

記述構成

以下にCMP2.0のEnterprise Beanクラスの記述構成を示します。それぞれの記述内容については、“setEntityContextおよびunsetEntityContextメソッドの記述”以降を参照してください。

Interstage Studioを使用すると、永続化フィールド、各メソッドおよびビジネスメソッドのひな形などが生成されます。

```
public class <Enterprise Beanクラス名> implements javax.ejb.EntityBean
{
```

- setEntityContextおよびunsetEntityContextメソッドの記述
- ejbCreateおよびejbPostCreateメソッドの記述
- ejbRemoveメソッドの記述
- ejbLoadおよびejbStoreメソッドの記述
- ejbActivateおよびejbPassivateメソッドの記述
- ejbHomeメソッド
- 抽象アクセッサメソッド

- ejbSelectメソッド
- ビジネスメソッド

}

13.8.2 setEntityContextメソッドおよびunsetEntityContextメソッドの記述

CMP2.0のsetEntityContextメソッドおよびunsetEntityContextメソッドの記述は、CMP1.1の場合と違いはありません。

CMP1.1の“13.7.3 setEntityContextメソッドおよびunsetEntityContextメソッドの記述”を参照してください。

13.8.3 ejbCreateメソッドおよびejbPostCreateメソッドの記述

コンテナがデータベースにデータを追加するときに呼び出すメソッドです。
Entity Beanを呼び出すEJBアプリケーションがcreateメソッドを呼び出すと、コンテナはcreateメソッドに対応するejbCreateメソッドを呼び出します。
コンテナはejbCreateメソッドが完了すると、データベースに対してINSERT文を発行し、ejbPostCreateメソッドを呼び出します。

記述する処理の概要

ejbCreateメソッドには、以下の処理を記述します。

- 永続化フィールドを入力引数で初期化する処理

ejbPostCreateメソッドには通常、処理を記述する必要はありません。

ejbCreateメソッドの規約

ejbCreateメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbCreateでなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はプライマリキータイプでなければなりません。
- メソッドの引数は、RMI over IIOPの規約に従わなくてはなりません。
  また、引数のタイプと数はHomeインタフェースのcreateメソッドに一致していなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.CreateException
  - javax.ejb.DuplicateKeyException

ejbPostCreateメソッドの規約

ejbPostCreateメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbPostCreateでなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- 返却値はvoidでなければなりません。
- メソッド引数は、対応するejbCreateメソッドの引数と同じでなければなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.CreateException

－ javax.ejb.DuplicateKeyException

記述例

```
public SampleBeanPrimaryKey ejbCreate(Integer code, String name, String desc)
        throws javax.ejb.EJBException
{
    // 永続化フィールドを入力引数の値で初期化
    setCode(code);
    setName(name);
    setDesc(desc);
    return( null );
}

public void ejbPostCreate(Integer codeValue, String nameValue, String descValue)
        throws javax.ejb.EJBException
{
}
```

13.8.4 ejbRemoveメソッドの記述

CMP2.0のejbRemoveメソッドの記述は、CMP1.1の場合と違いはありません。

CMP1.1の“13.7.5 ejbRemoveメソッドの記述”を参照してください。

13.8.5 ejbLoadメソッドおよびejbStoreメソッドの記述

CMP2.0のejbLoadメソッドおよびejbStoreメソッドの記述は、CMP1.1の場合と違いはありません。

CMP1.1の“13.7.6 ejbLoadメソッドおよびejbStoreメソッドの記述”を参照してください。

13.8.6 ejbActivateメソッドおよびejbPassivateメソッドの記述

CMP2.0のejbActivateメソッドおよびejbPassivateメソッドの記述は、CMP1.1の場合と違いはありません。

CMP1.1の“13.7.7 ejbActivateメソッドおよびejbPassivateメソッドの記述”を参照してください。

13.8.7 ejbHomeメソッドの記述

ejbHomeメソッドは、クライアントからejbHomeメソッドを呼び出したときに呼び出されます。

コンテナは、Entity Beanのインスタンスプールからインスタンスを1つ選択してejbHomeメソッドを実行し、ejbHomeメソッドの実行が終了した後にインスタンスをプーリングに返却します。

記述する処理の概要

ejbHomeメソッドには、特定のEntity Beanインスタンスを対象としないビジネスロジックを記述します。ejbHomeメソッドでデータベースにアクセスする場合は、ejbSelectメソッドを使用して行うのが一般的です。

ejbHomeの規約

ejbHomeメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- メソッドはstaticとして定義してはいけません。
- Enterprise BeanクラスのejbHome<METHOD>メソッドは、HomeインタフェースまたはLocalHomeインタフェースの<METHOD>メソッドの引数の数と型、返却値の型が同じでなければなりません。
- throws句には以下の例外を定義できます。
  － 任意のEJBアプリケーション固有の例外

－ javax.ejb.EJBException

記述例

```
public void ejbHomeTotalReservation() throws javax.ejb.EJBException
{
    // ビジネスロジックを記述する。
}
```

13.8.8 抽象アクセッサメソッドの記述

deployment descriptorに定義したCMFとCMRフィールドに対して、抽象アクセッサメソッド（get/setメソッド）を定義します。

記述する処理の概要

抽象アクセッサメソッドはabstractとして定義するため、メソッドの定義以外に記述する処理はありません。

抽象アクセッサメソッドの規約

抽象アクセッサメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして定義されていなければなりません。
- メソッドはabstractとして定義されていなければなりません。
- 抽象アクセッサメソッドは、CMFとCMRフィールドの先頭を大文字として、その前にget/setを付加して定義します（CMFとCMRフィールドをdeployment descriptorに定義するときは、先頭は小文字で定義してください）。
- throws句には例外を定義できません。

記述例

```
public abstract class CustomerBean implement EntityBean{

    public abstract void setId(Integer id);
    public abstract Integer getId();

    public abstract void setLastName(String lastName);
    public abstract String getLastName();

    public abstract void setFirstName(String firstName);
    public abstract String getFirstName();

    public abstract void setAddress(AddressLocal address);
    public abstract AddressLocal getAddress();
}
```

13.8.9 ejbSelectメソッドの記述

ejbSelectメソッドを使用することで、finderメソッドと同様に特定のインスタンス、またはインスタンスの集合を検索できます。

ejbSelectメソッドの特徴

- ejbSelectメソッドは、Finderメソッドと異なりHomeインタフェースに宣言しないため、クライアントアプリケーションまたはほかのEJBアプリケーションから呼び出すことはできません。
- ejbSelectメソッドはCMPフィールド値を返却することができます。複数のリファレンスを返却する場合は、戻り値の型としてjava.util.Collection型を使用できます。
- ejbSelectメソッドはrelationshipがあるBeanのインタフェース、またはそのコレクションを返却できます。
- ejbSelectメソッドは集合関数の結果を返却できます。

- ejbSelectメソッド内からデータベースアクセスを行うには、deployment descriptorに定義されたEJB QLを使用します。
- ejbHomeメソッド内よりejbSelectメソッドを使用することにより、データベースへのアクセスをより簡単に行うことができます。

ejbSelectメソッドの返却値について

Enterprise BeanのRemoteまたはLocalインタフェースタイプを返却できます。そのインタフェースのコレクションを返却する場合は、インタフェースタイプ（RemoteまたはLocal）をdeployment descriptorに指定します。それ以外の場合は、インタフェースタイプを指定する必要はありません。

注意

java.util.Set型を戻り値に持つejbSelectメソッドを使用することはできません。

ejbSelectメソッドの利用個所について

ejbSelectメソッドはEnterprise Beanクラス内ならejbHomeメソッドだけでなくビジネスメソッド、ejbLoadメソッドおよびejbStoreメソッドからも利用できます。

ejbSelectメソッドの規約

ejbSelectメソッドは以下の規約を満たしていなければなりません。

- メソッド名はejbSelectで始まらなければなりません。
- メソッドはpublicとして定義されていなければなりません。
- メソッドはabstractとして定義されていなければなりません。
- throws句には、javax.ejb.FinderExceptionを定義してください。

記述例

Enterprise Beanには以下のように記述します。

```
public abstract class CustomerBean implement javax.ejb.EntityBean{
    public abstract java.util.Collection ejbSelectOrders(int priority)
            throws javax.ejb.FinderException;
}
```

13.8.10 ビジネスメソッドの記述

Remoteインタフェースで宣言したメソッドを実装します。

ビジネスメソッドの規約

ビジネスメソッドは以下の規約を満たしていなければなりません。

- メソッド名は任意ですが、EJBのAPIで定義されているメソッドの名前と重複してはなりません。
- メソッドはpublicとして宣言されていなければなりません。
- メソッドの引数と返却値は、RMI over IIOPの規約に従わなくてはなりません。
- throws句には以下の例外を定義できます。
  - 任意のEJBアプリケーション固有の例外
  - javax.ejb.EJBException
  - javax.ejb.NoSuchEntityException

記述例

参加するイベントを追加するビジネスメソッドの例

```
public void assignEvent(Event e) {
    Collection events = getEvents();
    events.add(e);
}
```

13.8.11 例外処理

CMP2.0の例外処理は、CMP1.1の場合と違いはありません。

CMP1.1の“13.7.9 例外処理”を参照してください。

13.8.12 使用できるメソッド

CMP2.0の使用できるメソッドは、CMP1.1の場合と違いはありません。

CMP1.1の“13.7.10 使用できるメソッド”を参照してください。

13.8.13 Enterprise Beanクラスのメソッドが実行可能な操作

Enterprise Beanクラスのメソッドが実行可能な操作については、“13.6.14 Enterprise Beanクラスのメソッドが実行可能な操作”を参照してください。

13.9 Primary Keyクラスの作成

Primary Keyクラスは、Entity Beanのインスタンスの識別子となるクラスで、特定のインスタンスを検索するのに使用されます。

EJBコンテナは、データベースから取得したプライマリキーに対応するインスタンスを、キャッシュします。

finderメソッドが実行された場合には、equalsメソッドとhashcodeメソッドを使用して、同じプライマリキーのインスタンスがキャッシュされているかを判定して、プライマリキーが合致する場合には、キャッシュされているEJB objectを返却します。

Interstage Studioを使用している場合は、自動的に生成されます。

Primary Keyクラスの規約

Primary Keyクラスは以下の規約を満たしていなければなりません。

- java.io.Serializableインタフェースを実装した非リモートJavaオブジェクトでなければなりません。
- パラメタなしの空コンストラクタを用意しなければなりません。
- ObjectクラスのhashCodeメソッドとequalsメソッドを再定義しなければなりません。

記述する処理

1. プライマリキー値を格納する変数を記述します。複数記述することもできます。
2. パラメタなしの空コンストラクタを記述します。
3. equalsメソッドを記述します。引数で渡されるプライマリキーオブジェクトと同じものかどうか判定します。
4. hashCodeメソッドを記述します。プライマリキー値のhashcodeを返却します。

equalsメソッドとhashcodeメソッド

equalsメソッドとhashcodeメソッドには以下のような特徴があります。

- equalsメソッド
  コンテナがインスタンスを検索する際に、プライマリキーの値が同一かを判断するために実行します。
  また、EJBObjectクラスのisIdenticalメソッドを実行した際に実行されます。

- hashcodeメソッド
  equalsメソッドを実行する前に、インスタンスの絞り込みを行うためのメソッドです。異なるインスタンスに対してできるだけ異なる値を返却するようにhashcodeメソッドを定義しておくと、コンテナがインスタンスを検索する際の性能を向上できます。

注意

メソッド定義時の注意事項

- equalsメソッドで同じインスタンスと判断される場合は、hashcodeメソッドで同じ整数値を返却するように定義してください。また、equalsメソッドで同じインスタンスと判断されない場合でも、hashcodeメソッドが異なる整数値を返却する必要はありません。
- メソッド定義を誤った場合、CMFの更新処理が反映されないなどの現象が発生する場合がありますので、基本的にはInterstage Studioが出力するひな形を使用してください。
- プライマリキーに固定長文字列のDBカラムを指定した場合、データベースから取得されるデータは文字列長分だけ後方空白が補完されて取得されます。
  同じキー項目かどうかをequalsメソッドとhashcodeメソッドで判定する際に、これらのメソッドが、補完された後方空白を考慮(後方空白は無視)して実装しないとキー項目を異なるものとして判定するため、データベースにアクセスして返却されるのは、意図しないEJB objectとなります。

  例えば、以下のようにfindByPrimaryKeyメソッドを実行した場合、初回のfindByPrimaryKeyメソッドでDBアクセス(SELECT文発行)して、取得したデータをPrimary Keyの変数に格納します。
  項目長分のデータが入っていない固定長文字列のDBカラムからデータを取り出した場合、Primary Keyの変数にはスペースを補完した値が設定されて、EJB objectがキャッシュされます。
  2回目のfindByPrimaryKeyでは、findByPrimaryKeyの入力パラメタであるPrimary Keyをキーに、すでに検索されているEJB objectであるかを、コンテナ内のキャッシュから検索します。

  1. UserTransaction.begin
  2. SampleRemote EO1 = SampleHome.findByPrimaryKey(PK1) ※
  3. 更新処理
  4. SampleRemote EO2 = SampleHome.findByPrimaryKey(PK1) ※
  5. 更新処理
  6. UserTransaction.rollback

  ※PK1は同じPrimaryKey値

  上記EO1とEO2では、異なるEJB objectが返却されるため、3.や5.の更新処理が反映されないといった問題が発生する可能性があります。
  固定長文字列のDBカラムをプライマリキーとして使用する場合は、以下を考慮してください。

  ー DBカラムを固定長項目から可変長項目に変更する。

  ー PrimaryKeyクラスのhashcodeメソッド、および、equalsメソッド内でDBの固定長項目にマッピングしている変数を参照する場合、後方スペースを削除する。

CMPでプライマリキーのフィールドが1つの場合

CMPでプライマリキーのフィールドが1つの場合で、フィールドが以下の型の場合には、Primary Keyクラスを作成しないことも可能です。

- java.lang.Boolean
- java.lang.Byte
- java.lang.Character
- java.lang.Short
- java.lang.Integer

- java.lang.Long
- java.lang.Float
- java.lang.Double
- java.math.BigDecimal
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.lang.String

この場合、通常はPrimary Keyクラスを指定するHomeインタフェースのcreateメソッドやfindByPrimaryKeyメソッドなどの引数には以下のように、プライマリキーのフィールドを指定します。

 例

プライマリキーがString型の場合

```
public SampleRemote create(String pkField)
        throws javax.ejb.DuplicateKeyException,
               javax.ejb.CreateException,
               java.rmi.RemoteException;
public SampleRemote findByPrimaryKey(String pkValue)
        throws javax.ejb.FinderException,
               javax.ejb.ObjectNotFoundException,
               java.rmi.RemoteException;
```

また、deployment descriptorの“Primary Keyクラス名”にはプライマリキーのフィールドの型(java.lang.Stringなど)を指定し、“Primary Keyフィールド名”にはフィールド名を指定してください。

13.9.1 CMPの記述例

```
package Sample;

public class SampleBeanPrimaryKey implements java.io.Serializable
{
    public Integer code;

    public SampleBeanPrimaryKey()
    {
    }

    public boolean equals(Object obj)
    {
        if (obj == null)
            return false;
        if (!(obj instanceof SampleBeanPrimaryKey))
            return false;
        SampleBeanPrimaryKey pk = (SampleBeanPrimaryKey)obj;
        boolean b = (code == null)?(pk.code == null):code.equals(pk.code);
        return b;
    }
    public int hashCode()
    {
        String temp = String.valueOf(code);
        return temp.hashCode();
    }
}
```

13.10 インスタンス管理モードでの注意事項

Entity Beanの実行環境では、ユーザの用途に合わせて、インスタンス管理のモードが選択できます。選択するインスタンス管理モードによっては、以下に注意してください。

インスタンス管理モードの詳細については、“10.2.2 Entity Beanのインスタンス管理”を参照してください。

注意事項	該当インスタンス管理モード	対処方法
Entity Beanを呼び出すアプリケーションからcreateおよびremoveメソッドを発行するとエラーとなります。また、永続化フィールドの値を更新してもデータベースには反映されません。	ReadOnly	更新処理を行う場合は、ReadOnly以外のモードを指定してください。
deployment descriptorのreentrant属性に“動作不可能(false)”を設定し、EJBアプリケーションのメソッドがreentrant呼出しを行った場合、正常に動作します。	ReadOnly	Reentrant呼出し不可能にしたい場合はReadOnly以外のインスタンス管理モードを指定してください。
Windows32/64 Linux32/64 分散トランザクション機能を使用し、かつOracleのデータベースを使用する場合、複数インスタンス検索(Enumeration型またはCollection型)を実行すると、Oracleのデータベースから“ORA-01002: フェッチ順序が無効です”という例外が返却される場合があります。	Sequential	更新処理を行う場合はReadWriteを、検索処理だけ行う場合はReadOnlyを指定してください。
トランザクション属性にMandatory以外を指定して、複数インスタンス検索(Enumeration型またはCollection型)を実行すると、例外が発生する場合があります。	Sequential	EJBアプリケーション(Bean単位)にトランザクション属性Mandatoryを指定して実行してください。
以下のEJBアプリケーションはインスタンス管理モードにSequentialが設定できません。 ・ IJServerに配備されている、およびInterstage管理コンソールで、“ローカル呼出し”を“しない”に設定しているEntity Bean ・ IJServerに配備されていて、かつ、カストマイズで“ローカル呼出し”に“しない”が設定されているEntity Bean	Sequential	インスタンス管理モードにSequentialを指定してEntity Beanを動作させたい場合は“ローカル呼出し”を“する”に設定してください。 “ローカル呼出し”に“しない”を設定して運用する場合は、以下のようにインスタンス管理モードを変更してください。 ・ 更新処理を行う場合は、ReadWrite指定する。 ・ 検索処理だけを行う場合は、ReadOnlyを指定する。

13.11 CMPで定義するJavaのデータ型とDBMSのSQLデータ型との対応

CMPで定義する以下のデータ型は、コンテナがDBMSのSQLデータ型に対応づけます。

- CMFのデータ型
- finderメソッドのパラメタのデータ型

上記のデータ型と、DBMSのSQLデータ型との対応について説明します。

サポートするデータ型は以下です。

- 標準データ型
- その他のデータ型

13.11.1 標準データ型

使用できる標準データ型

以下に、使用できる標準データ型を示します。

- boolean
- java.lang.Boolean (注)
- byte
- java.lang.Byte (注)
- byte[]
- char
- java.lang.Character (注)
- double
- java.lang.Double (注)
- float
- java.lang.Float (注)
- int
- java.lang.Integer (注)
- long
- java.lang.Long (注)
- short
- java.lang.Short (注)
- java.lang.String
- java.math.BigDecimal
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

注) コンテナが、対応するJavaのプリミティブデータ型に変換してからデータベースに設定します。

nullが使用可能なCMFのデータ型

以下に、nullが使用できるデータ型を示します。

- java.lang.Boolean
- java.lang.Byte
- byte[]
- java.lang.Character
- java.lang.Double

- java.lang.Float
- java.lang.Integer
- java.lang.Long
- java.lang.Short
- java.lang.String
- java.math.BigDecimal
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

推奨パターン

以下に、Javaのデータ型とDBMSのSQLデータ型の組合せ(推奨パターン)を示します。

Javaのデータ型を、以下の表に示す以外のDBMSのSQLデータ型と組み合わせたい場合は、JDBCドライバの変換規則にしたがってください。
ただし、“使用できる標準データ型”で、(注)となっているデータ型については、コンテナが変換した後のJavaのプリミティブデータ型でJDBCの変換規則にしたがってください。

なお、Interstage EJBではJDBC1.0の範囲でサポートしています。

注意

DBMSのSQLデータ型でCHAR型のような固定長文字列型を使用した場合、JDBCドライバの仕様により、返却される文字列データ値について後方空白削除、または空白パディングを行うためデータ値の不整合が起こる場合があります。
特にプライマリキーフィールドにCHAR型のカラムをマッピングする場合に注意してください。

このような場合、CHAR型のような固定長文字列型ではなく、VARCHAR型やVARCHAR2型といった可変長文字列型を使用することでこの問題は解決されます。

Javaのデータ型	DBMSのSQLデータ型			
	Symfoware	Oracle	SQL Server	Windows32/64 Linux32/64 PostgreSQL
boolean/ java.lang.Boolean	－	NUMBER	BIT	BOOLEAN
byte/ java.lang.Byte	－	NUMBER	－ (注9)	SMALLINT
char/ java.lang.Character	－	－	－	－
double/ java.lang.Double	FLOAT DOUBLE	NUMBER	FLOAT	DOUBLE PRECISION (注7)
float/ java.lang.Float	REAL	NUMBER	REAL	REAL (注7)
int/ java.lang.Integer	INT	NUMBER	INT	INTEGER SERIAL

Javaのデータ型	DBMSのSQLデータ型			
	Symfoware	Oracle	SQL Server	Windows32/64 Linux32/64 PostgreSQL
long/ java.lang.Long	－	NUMBER	BIGINT	BIGINT BIGSERIAL
short/ java.lang.Short	SMALLINT	NUMBER	SMALLINT TINYINT (注9)	SMALLINT
java.lang.String	CHAR NCHAR VARCHAR NVARCHAR	CHAR (注2) VARCHAR2 (注2) LONG (注2) CLOB (注8)	CHAR VARCHAR TEXT NCHAR NVARCHAR NTEXT	CHARACTER CHARACTER VARYING TEXT
java.math.BigDecimal	NUMERIC DECIMAL	NUMBER	DECIMAL NUMERIC MONEY SMALLMONEY	NUMERIC DECIMAL
byte[]	BLOB (注1)	RAW LONG RAW (注2)(注3) BLOB (注8)	BINARY VARBINARY IMAGE	BYTEA
java.sql.Date	DATE	DATE	－	DATE
java.sql.Time	TIME	DATE	－	TIME
java.sql.Timestamp	TIMESTAMP	DATE TIMESTAMP(注4)	DATETIME (注5)(注6) SMALLDATETIME (注5)	TIMESTAMP TIMESTAMP WITH TIMEZONE

－：推奨するデータ型は特にありません。

注1)

以下の場合、32KB以上のBLOB型の列に対して取り出し/設定はできません。

－ バージョンレベルがV5.0L10より前のSymfowareを使用する場合

－ バージョンレベルがV5.0L10以降のSymfowareでRDA-SV連携を使用する場合

オプション「CMPデータのstream転送」に“する”を設定している場合は、本注意事項は該当しません。
オプションの設定は、Interstage管理コンソールの[システム]＞[ワークユニット]＞“ワークユニット名”＞“モジュール”＞“EJBアプリケーション名”＞[アプリケーション環境定義]のCMFマッピング定義で行います。

注2)

Oracleを利用する場合、以下のデータ型については、取り扱い可能なデータ長に制限がある場合があります。

－ LONG RAW

－ LONG

詳細は、OracleのJDBCのマニュアルで、setBytesメソッドとsetStringメソッドについて参照してください。
上記制限値より大きいデータ量を取り扱う場合は、BMPのEntity Beanを使用してください。
オプション「CMPデータのstream転送」に“する”を設定している場合は、本注意事項は該当しません。
オプションの設定は、Interstage管理コンソールの[システム]＞[ワークユニット]＞“IJServer名”＞“EJBモジュール”＞“EJBアプリケーション名”＞[アプリケーション環境定義]のCMFマッピング定義で行います。

注3)

OracleのLONG RAWデータ型を使用する場合、CMP1.1のEntity Beanのトランザクション属性にMandatoryを指定して動作させるとOracleのJDBCドライバより以下のメッセージが出力される場合があります。
「ORA-17027:ストリームはすでにクローズ済みです」
トランザクション属性にRequiredを指定することで上記問題は回避できますが、処理性能が劣化する場合があります。

注4)

Oracle10gよりサポートされたデータ型です。

注5)

datetime型とsmalldatetime型のデータを更新する場合は、以下の形式でデータを指定してください。

記述形式:

```
YYYY-MM-DD hh:mm:ss
```

YYYY:年、MM:月、DD:日、hh:時、mm:分、ss:秒（ミリ秒は未サポートです。）

例:

```
2001-09-22 14:23:40
```

注6)

datetime型に対して、以下のAPIを使用する場合、smalldatetime型の範囲内の値だけ有効となります。

- PreparedStatement.setTimestamp(int parameterIndex, Timestamp x)

注7)

PostgreSQLのREALデータ型とDOUBLE PRECISIONデータ型は不正確な精度が変動する数値データ型です。したがって、正確な値ではなく、正確な値に近い値として保存されることがあるため、保存しようとする値と保存された値に多少の誤差がある可能性があります。
なお、ゼロに限りなく近い値で、ゼロとは区別されていない数値は、アンダーフローエラーを引き起こします。正確な記録や計算が必要な場合は、NUMERICデータ型を使用してください。

注8)

OracleのBLOB/CLOBデータ型を利用する場合は、以下の条件を満たしてください。

－ Oracle10g R2以降のデータベースサーバを使用する

－ Oracle10g R2以降のThinドライバ、またはociドライバを使用する。

なお、本製品のJ2EE機能がサポートしているJavaのバージョンをサポートしているJDBCドライバを使用してください。

注9)

TINYINTは、SQL Serverではshortが推奨型です。

13.11.2 その他のクラス

定義可能なクラス

CMP1.1の場合、以下のクラスを定義できます。

- java.io.Serializableインタフェースを直接的または間接的に実装するクラス
- 上記クラスの配列

これらが定義された場合はbyte[]に変換してDBMSに設定します。nullの指定もできます。byte[]とDBMSのSQLデータ型とのマッピングについては“13.11.1 標準データ型”を参照してください。

CMFにHomeインタフェース、Remoteインタフェースのデータ型は使用できません。

第14章　Message-driven Beanの実装

本章では、Message-driven Beanのプログラミング方法について説明します。

14.1 Message-driven Beanの概要

以下に、Message-driven Beanの概要を説明します。

クラスファイルの構成

Message-driven Beanは、クライアントアプリケーションから直接呼び出されることがないため、Session BeanやEntity Beanと違って、HomeインタフェースとRemoteインタフェースを作成する必要がありません。

Message-driven Beanは、Enterprise Beanクラスファイルから成り立っています。

クラス名はユーザ任意の名前にできます。

クラスファイル名	内容
Enterprise Beanクラス	サーバで実際に処理を行うクラスで、ユーザの開発するメソッドを実装します。

14.2 Enterprise Beanクラスの作成

Enterprise Beanクラスは、ユーザがインタフェースで定義した ejbCreateメソッドおよびビジネスメソッドを実装します。

Interstage Studioを使用して開発を行った場合、Enterprise Beanクラスのひな形が自動生成されます。

以下に、Enterprise Beanクラスの記述の形式を示します。

記述形式

```
public class Enterprise Bean名
implements javax.ejb.MessageDrivenBean, メッセージリスナインタフェース {
・・・

    public void ejbCreate() {
    ・・・
    }
    public void ejbRemove() {
    ・・・
    }
    ・・・
    メッセージリスナメソッド {
    ・・・
    }
}
```

規約

Enterprise Beanクラスは以下の規約を満たしていなければなりません。

- javax.ejb.MessageDrivenBeanインタフェースを実装していなければなりません。
- 受信対象種別ごとのメッセージリスナインタフェースを実装しなければなりません。
  - 受信対象種別がJMSの場合、メッセージリスナインタフェースはjavax.jms.MessageListenerとなります。
  - 受信対象種別がresourceadapterの場合、RARファイルのdeployment descriptorファイル(ra.xml)に定義された、messagelistener-typタグに指定されているメッセージリスナインタフェースを実装してください。
- javax.ejb.TimedObjectインタフェースを実装することができます。

- publicとして定義されていなければなりません。finalやabstractとして定義してはいけません。
- パラメタを取らないpublicコンストラクタをもつ必要があります。
- finalizeメソッドを定義してはなりません。
- ejbCreateメソッドを実装しなければなりません。
- スーパークラスを持ったり、インタフェースを実装したりすることができます。
  スーパークラスにejbCreateメソッドやその他のMessageDrivenBeanインタフェースやMessageListenerのインタフェースを実装することができます。
- Enterprise Beanクラスは、EJB仕様に定められたメソッドのほかにも、メソッド(ビジネスメソッドが内部で呼び出すヘルパーメソッドなど)を実装できます。

ejbCreate、ejbRemove、メッセージリスナメソッドには、以下の規約があります。

ejbCreateの規約

ejbCreateは以下の規約を満たしていなければなりません。

- メソッド名は ejbCreateでなければなりません。
- メソッドは publicとして定義されていなければなりません。
- メソッドはfinalやstaticで定義してはなりません。
- 返却値は voidでなければなりません。
- 引数を持ってはなりません。
- throws句には任意のアプリケーション固有の例外を定義してはなりません。

メッセージリスナメソッドの規約

メッセージリスナメソッドは以下の規約を満たしていなければなりません。

- メソッドはpublicとして宣言されていなければなりません。
- メソッドはfinalやstaticで定義してはなりません。

ejbRemoveの規約

ejbRemoveは以下の規約を満たしていなければなりません。

- メソッド名は ejbRemoveでなければなりません。
- メソッドは publicとして定義されていなければなりません。
- メソッドはfinalやstaticで定義してはなりません。
- 返却値は voidでなければなりません。
- 引数を持ってはなりません。
- throws句には任意のアプリケーション固有の例外を定義してはなりません。

14.2.1 記述例

以下に Enterprise Beanの記述例を示します。太字部分は、ユーザ任意の指定ができます。

```
package Sample;

import javax.ejb.*;
import javax.jms.*;
```

```
public class SampleBean
  implements javax.ejb.MessageDrivenBean, javax.jms.MessageListener {

    private javax.ejb.MessageDrivenContext context;

    public SampleBean() {
        // 以下に初期化する手続きを記述してください。
    }

    public void ejbCreate() {
        // 以下に作成された場合の手続きを記述してください。
    }

    public void ejbRemove() {
        // 以下に削除される場合の手続きを記述してください。
    }

    public void setMessageDrivenContext(javax.ejb.MessageDrivenContext ctx) {
        // 以下にコンテキストが通知された場合の手続きを記述してください。
        context = ctx;
    }

    public void onMessage(javax.jms.Message msg) {
        // 以下にメッセージが通知された場合の手続きを記述してください。
        try {
            TextMessage textmsg = (TextMessage)msg;
            String str_msg = textmsg.getText();
            System.out.println("¥tMessage = " + str_msg);
        }
        catch (JMSException ex) {
            System.out.println("onMessage Exception occured.:" + ex.toString());
        }
    }
}
```

14.2.2 使用できるメソッド

MessageDrivenContextインタフェースのメソッド

MessageDrivenContextは、コンテナによって保守されているコンテキストへのアクセスを与えます。このインタフェースは、setSessionContextのパラメタとして指定することにより取得できます。それによりSessionContextより拡張されるEJBContextインタフェースのメソッドを使用できます。

以下に使用できるメソッドの一覧を示します。

メソッド名	内容
setRollbackOnly()	カレントトランザクションを“rollback”にマークします。
getRollbackOnly()	そのトランザクションが“rollback only”でマークされているか判定します。
getUserTransaction()	トランザクション区別インタフェースを取得します。

その他のメソッドについて

ejbCreate、ejbRemove、メッセージリスナメソッド以外に、Enterprise Beanクラスには、コンテナが処理の各フェーズで呼び出す以下のメソッドがあります。このクラスにはユーザが任意の処理を指定できます。このメソッドは省略できません。

以下に使用できるメソッドを示します。

メソッド名	内容
setMessageDrivenContext	コンテナによって保守されているコンテキストへのアクセスを与えます。

14.2.3 Enterprise Beanクラスのメソッドが実行可能な操作

以下の操作は、Beanクラスのメソッドごとに実行できる操作が異なります。

- javax.ejb.MessageDrivenContextインタフェースのメソッド実行
- javax.transaction.UserTransactionインタフェースのメソッド実行
- Enterprise Bean Environmentの利用
- データベースへのアクセス
- 他のEJBアプリケーションへのアクセス

実行できる操作は以下です。以下に示す以外の操作を実行した場合は、java.lang.IllegalStateExceptionが発生する場合があります。

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
コンストラクタ	なし	なし
setMessageDrivenContext	・javax.ejb.MessageDrivenContextメソッド ー getTimerService (注1) ・Enterprise Bean Environmentの利用	・javax.ejb.MessageDrivenContextメソッド ー getUserTransaction (注1) ー getTimerService (注1) ・Enterprise Bean Environmentの利用
ejbCreate	・javax.ejb.MessageDrivenContextメソッド ー getTimerService ・Enterprise Bean Environmentの利用	・javax.ejb.MessageDrivenContextメソッド ー getUserTransaction ー getTimerService ・Enterprise Bean Environmentの利用
ejbRemove	・javax.ejb.MessageDrivenContextメソッド ー getRollbackOnly (注1) ー setRollbackOnly (注1) ー getTimerService ・リソースマネージャ(データベースなど)へのアクセス (注1) ・他のEJBアプリケーションへのアクセス (注1) ・Enterprise Bean Environmentの利用	・javax.ejb.MessageDrivenContextメソッド ー getUserTransaction ー getTimerService ・javax.transaction.UserTransactionメソッド (注1) ・リソースマネージャ(データベースなど)へのアクセス (注1) ・他のEJBアプリケーションへのアクセス (注1) ・Enterprise Bean Environmentの利用

メソッド名	実行可能な操作	
	トランザクション管理種別がContainerの場合	トランザクション管理種別がBeanの場合
メッセージリスナメソッド	• javax.ejb.MessageDrivenContextメソッド 　－ getRollbackOnly 　－ setRollbackOnly 　－ getTimerService • javax.ejb.TimerServiceメソッド • javax.ejb.Timerメソッド • Enterprise Bean Environmentの利用 • リソースマネージャ(データベースなど)へのアクセス • 他のEJBアプリケーションへのアクセス	• javax.ejb.MessageDrivenContextメソッド 　－ getUserTransaction 　－ getTimerService • javax.transaction.UserTransactionメソッド • javax.ejb.TimerServiceメソッド • javax.ejb.Timerメソッド • Enterprise Bean Environmentの利用 • リソースマネージャ(データベースなど)へのアクセス • 他のEJBアプリケーションへのアクセス
ejbTimeout	• javax.ejb.MessageDrivenContextメソッド 　－ getRollbackOnly 　－ setRollbackOnly 　－ getTimerService • javax.ejb.TimerServiceメソッド • javax.ejb.Timerメソッド • Enterprise Bean Environmentの利用 • リソースマネージャ(データベースなど)へのアクセス • 他のEJBアプリケーションへのアクセス	• javax.ejb.MessageDrivenContextメソッド 　－ getUserTransaction 　－ getTimerService • javax.transaction.UserTransactionメソッド • javax.ejb.TimerServiceメソッド • javax.ejb.Timerメソッド • Enterprise Bean Environmentの利用 • リソースマネージャ(データベースなど)へのアクセス • 他のEJBアプリケーションへのアクセス

注1) EJB規約では許可されていないため、アプリケーションの移行性を重視する場合には推奨しません。

注意

MessageDrivenContextインタフェースのgetRollbackOnlyメソッドとsetRollbackOnlyメソッドは、トランザクション属性がRequiredのときだけ使用してください。
トランザクション属性がNotSupportedのときにこれらのメソッドが呼び出された場合は、java.lang.IllegalStateExceptionが発生する場合があります。

第15章　EJBアプリケーションの呼出し方法

本章では、EJBアプリケーションの呼出し方法について説明します。

EJBアプリケーションをクライアントアプリケーションから呼び出す場合、クライアントアプリケーションの以下のインタフェースに対して処理を行います。

- Homeインタフェース
- Remoteインタフェース
- LocalHomeインタフェース
- Localインタフェース

15.1 Session Beanの呼出し方法

呼出し手順

Session Beanを呼び出すクライアントアプリケーションは、以下の手順で作成します。

1. Homeインタフェースの検索
   Session Beanのオブジェクトの所在をネーミングサービスに問い合わせるlookup処理を行います。lookup処理の詳細は、“4.10 オブジェクトの参照方法”を参照してください。
2. Session Beanのインスタンスの生成
   Homeインタフェースに定義したcreateメソッドを使用してSession Beanのインスタンスを生成します。
3. ビジネスメソッドの呼出し
   Session BeanのRemoteインタフェースに定義したビジネスメソッドを呼び出して必要な処理を行います。
4. Session Beanのインスタンスの削除
   Session BeanのRemoteインタフェースに定義したremoveメソッドを呼び出してSession Beanのインスタンスを消去します。

注意

STATEFUL Session Beanを呼び出すEJBアプリケーションでトランザクションの操作を行う場合、STATEFUL Session Beanの開放処理(removeメソッド)は、必ずビジネスメソッド実行時に開始されているトランザクション処理の完了後に行ってください。トランザクション処理中に行った場合、以下のエラーが発生します。

- javax.ejb.RemoveExceptionが呼び出し元に返却される
- イベントログファイル、または、システムログファイルに以下のエラーメッセージが出力される
  IJServer: エラー: IJServer21061: トランザクションが開始されています

15.1.1 Session Beanを呼び出す場合の記述例

以下は、Session Beanを呼び出すクライアントアプリケーションの記述例です。

SampleClient.java

```
package Sample;

import javax.ejb.*;
import java.rmi.*;
import java.util.*;
import com.fujitsu.interstage.ejb.jndi.*;

class SampleClient
{

  public static void main(String args[]){
```

```
SampleHome   h = null;
SampleRemote r = null;

// lookupの処理を行います
try{
  // InitialContext獲得
  // デフォルトコンストラクタ指定で獲得する(注1)
  javax.naming.Context ic = new javax.naming.InitialContext();

  // lookup            (1)
  java.lang.Object Obj = (java.lang.Object)ic.lookup("SampleBean");
                  /* Enterprise Beanオブジェクトのlookup処理を行います */
  // homeのnarrow()  _(2)
  h = (SampleHome)javax.rmi.PortableRemoteObject.narrow(Obj, SampleHome.class);

}catch (Exception e){
  System.err.println("lookup Failed");
  e.printStackTrace();
  System.exit(1);
}

// create
try{
  /* EJBアプリケーションのcreate処理を行います */
  r = (SampleRemote)h.create("create sample");
}catch (CreateException e){
  System.err.println("CreateException Catched");
  e.printStackTrace();
  System.exit(1);
}catch (RemoteException e){
  System.err.println("RemoteException Catched");
  e.printStackTrace();
  System.exit(1);
}catch (Exception e){
  System.err.println("create Failed");
  e.printStackTrace();
  System.exit(1);
}

// invoke business method
try{
  int ret;
  /* ビジネスメソッドの処理を行います */
  ret = r.business1(1,2);
  System.out.println(ret);
  ret = r.business2(2,8);
  System.out.println(ret);
}catch (RemoteException e){
  System.err.println("RemoteException Catched");
  e.printStackTrace();
  System.exit(1);
}catch (Exception e){
  System.err.println("business method Failed");
  e.printStackTrace();
  try{
    r.remove();
  }catch (Exception ex){
    System.err.println("remove Failed");
    ex.printStackTrace();
  }
  System.exit(1);
}
```

```
        // remove
        try{
          /* EJBアプリケーションのremove処理を行います */
          r.remove();
        }catch (RemoveException e){
          System.err.println("RemoveException Catched");
          e.printStackTrace();
          System.exit(1);
        }catch (RemoteException e){
          System.err.println("RemoteException Catched");
          e.printStackTrace();
          System.exit(1);
        }catch (Exception e){
          System.err.println("remove Failed");
          e.printStackTrace();
          System.exit(1);
        }
      }
    }
```

注意

EJBクライアントを使用してクライアントアプリケーションを起動する際に、以下の環境プロパティを指定し起動してください。
J2EEアプリケーションクライアントと、サーブレット/JSPが提供する各機能を使用する場合には、“JNDIサービスプロバイダの環境設定”を参照してください。

```
-Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
```

例)

```
java -Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient Javaアプリケーション名
```

LocalHomeインタフェースを経由した場合

記述例はHomeインタフェースの例ですが、LocalHomeインタフェースを経由してSession Beanにアクセスする場合には、記述例の**(1)**と**(2)**の処理が異なります。
以下のようにlookupで取得したオブジェクトを直接キャストして、narrowメソッドを実行する必要はありません。

```
Context initialContext = new InitialContext();
CartHome cartHome = (CartHome)initialContext.lookup("java:comp/env/ejb/cart");
```

15.2 Entity Beanの呼出し方法

Entity Beanを利用することにより、SQLなどのデータベース操作言語を意識せずデータベースアクセスができます。
Entity Beanを呼び出すクライアントアプリケーションは、BMPおよびCMPの区別なく呼び出せます。
ここでは、Entity Beanの呼出し方法およびEntity Beanを使用したデータベースアクセス処理の記述例を説明します。

呼出し手順

Entity Beanを呼び出す場合に必要な手順は以下です。

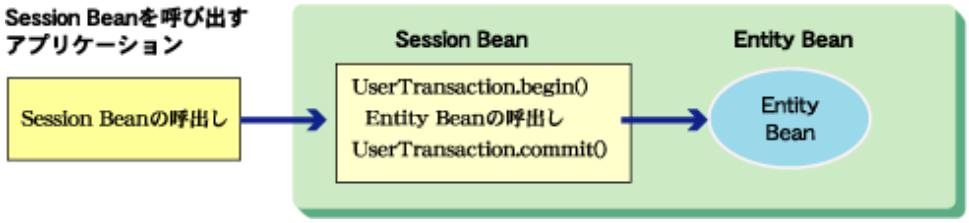
1. Homeインタフェースの検索
   Session beanの場合と同様に、Entity Beanのオブジェクトの所在をネーミングサービスに問い合わせるlookup処理を行います。lookup処理の詳細は、“4.10 オブジェクトの参照方法”を参照してください。
2. Entity Beanのインスタンスの検索または生成
   Homeインタフェースに定義したcreateメソッドまたはfinderメソッドを使用してEntity Beanのインスタンスを取得します。

3. ビジネスメソッドの呼出し
   Remoteインタフェースに定義したビジネスメソッドを呼び出して必要な処理を行います。

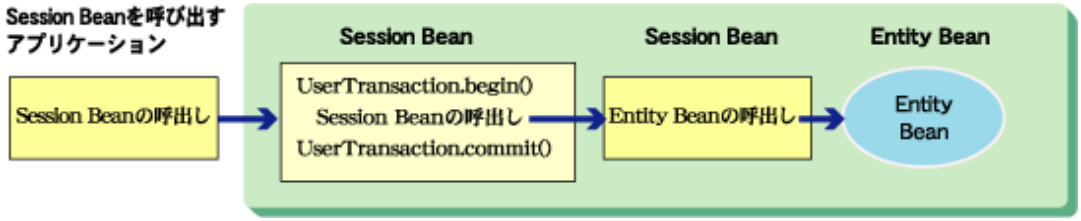
15.2.1 トランザクション機能を使用する場合

Entity Beanでは、データベースアクセス処理を行いますが、EJBサービスが提供するトランザクション配下でEntity Beanを利用することにより、データベースのデータを安全に処理できます。
EJBサービスが提供するトランザクション機能の詳細は、“10.4 EJBサービスのトランザクション制御”を参照してください。
以下にトランザクション機能を使用した場合のEntity Beanの呼出し例を示します。

Session Bean - Entity Beanの場合



Session Bean - Session Bean - Entity Beanの場合



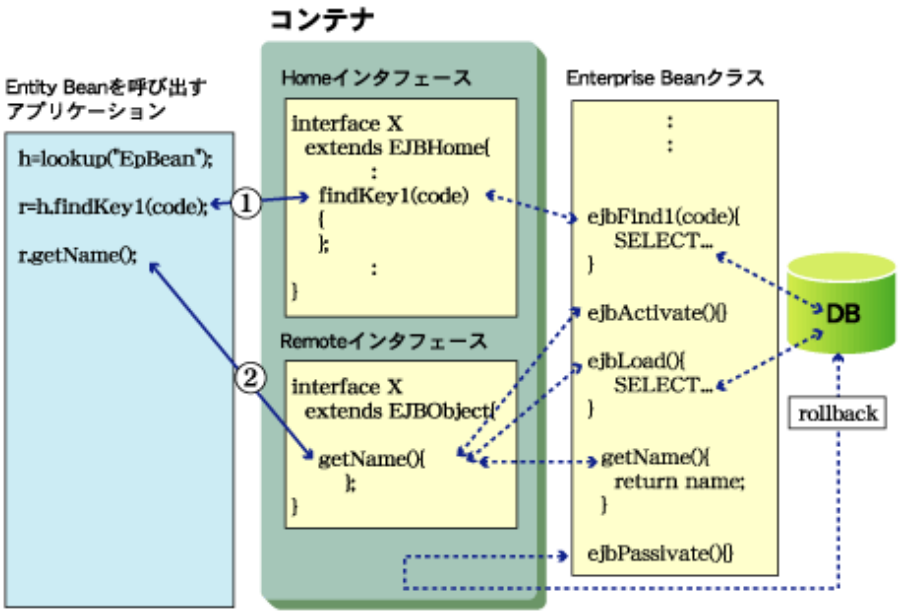
15.2.2 検索処理の記述

Entity Beanのインスタンスを検索するときのEntity Beanを呼び出すアプリケーション、コンテナおよびEnterprise Beanクラス間の処理の流れとEntity Beanを呼び出すアプリケーションの記述例について説明します。

- 処理の流れ
- 1件インスタンス検索の記述例
- 複数インスタンス検索の記述例(collection インタフェースの場合)
- 複数インスタンス検索の記述例(enumeration インタフェースの場合)
- ejbSelectメソッドの記述例

処理の流れ

Entity Beanのインスタンスを検索しデータを取り出すときの処理の流れを示します。
以下の例はBMPのEntity Beanです。CMPはデータベースアクセス処理をコンテナが行いますが、処理の流れは同じです。



1. 呼出し元のアプリケーションから、findkey1メソッドを呼び出します。
   findkey1メソッドが発行されると、コンテナからEnterprise Beanクラスの ejbFindKey1メソッドが呼び出されて、ejbFindKey1に記述したSELECT文を実行し、プライマリキーオブジェクトを作成、返却します。
2. 呼出し元のアプリケーションから、ビジネスメソッド(getNameメソッド)を呼び出します。
   ビジネスメソッドが呼び出されると、コンテナが以下の処理を行います。
   - Enterprise Beanクラスの ejbActivateメソッドを呼び出す。
   - Enterprise Beanクラスの ejbLoadメソッドを呼び出す。ejbLoadメソッドに記述したSELECT文が実行され、永続化フィールドにデータを設定する。
   - Enterprise Beanクラスのビジネスメソッドを呼び出す。

1件インスタンス検索の記述例

記述する処理概要

1. 呼び出すEntity Beanのlookup処理を行い、Entity BeanのEJB homeを取得する。
2. findByPrimaryKeyメソッドを呼び出し、プライマリキーオブジェクトを取得する。
3. ビジネスメソッドを呼び出す。

記述例

EmployeeEVは、1レコード分のデータを格納するためのユーザ定義クラスを表しています。

```
    (lookup処理でEJB homeの取得)
                                    :
EmployeeEntityRemote etyRemote = null;
EmployeeEntityPrimaryKey pk = new EmployeeEntityPrimaryKey();
pk.ID = empEV.getId();

try {
    // findByPrimaryKeyの呼出し
    etyRemote = etyHome.findByPrimaryKey(pk);
} catch ( FinderException ex ) {
    throw new UserException( ex.getMessage() );
} catch ( RemoteException ex ) {
    throw new EJBException( ex.getMessage() );
}
```

```
EmployeeEV view = new EmployeeEV();
// ビジネスメソッド／抽象アクセッサメソッド(getId,getName,getDept,getAge)の呼出し
view.setId(etyRemote.getId());
view.setName(etyRemote.getName());
view.setDept(etyRemote.getDept());
view.setAge(etyRemote.getAge());

                                    :
```

複数インスタンス検索の記述例(collection インタフェースの場合)

記述する処理概要

1. 呼び出すEntity Beanのlookup処理を行い、EJB homeを取得する。
2. find<METHOD>メソッドを呼び出し、プライマリキーオブジェクトを取得する。
3. ビジネスメソッドを呼び出す。

記述例

```
   (lookup処理でEJB homeの取得)
                                    :
Collection enum = null;
try {
    // find<METHOD>の呼出し
    enum = etyHome.findByKey2(empEV);
} catch ( FinderException ex ) {
    throw new UserException( ex.getMessage() );
} catch ( RemoteException ex ) {
    throw new EJBException( ex.getMessage() );
}

if ( enum == null ) {
    examDebug( "## EmployeeCBM : empSearchN2() record not found" );
    return null;
}

int count = 0;
Vector vec = new Vector();
EmployeeEntityRemote etyRemote = null;
Iterator iterator = enum.iterator();

java.lang.Object obj = null;
while( iterator.hasNext() ) {
    EmployeeEV view = new EmployeeEV();
    obj = iterator.next();
    etyRemote =
        (EmployeeEntityRemote)javax.rmi.PortableRemoteObject.narrow(obj,
        EmployeeEntityRemote.class);
    // ビジネスメソッド／抽象アクセッサメソッド(getId,getName,getDept,getAge)の呼出し
    view.setId(etyRemote.getId());
    view.setName(etyRemote.getName());
    view.setDept(etyRemote.getDept());
    view.setAge(etyRemote.getAge());
    count++;
    vec.addElement( view );
}

                                    :
```

注意

以下の[条件]に当てはまらない場合、finderメソッドから返却されるCollectionインタフェースで、以下の[使用できないメソッド]は使用できません。使用した場合には、java.lang.UnsupportedOperationExceptionが返却されます。

条件

- CMP1.1およびBMPの場合
  IJServerタイプ別に以下のように異なります。以下の条件を満たさない場合、使用範囲はV6.0以前の範囲となります。
  - WebアプリケーションとEJBアプリケーションを同一JavaVMで運用
    トランザクション属性に“Mandatory”以外のトランザクション属性を指定してください。
  - WebアプリケーションとEJBアプリケーションを別JavaVMで運用または、EJBアプリケーションのみ運用
    トランザクション属性に“Mandatory”以外のトランザクション属性を指定するか、トランザクション属性に“Mandatory”を指定する場合は、Interstage管理コンソールの[アプリケーション環境定]で、“ローカル呼出し”に“しない”を設定してください。
- CMP2.0の場合、条件はありません。

使用できないメソッド

- add(Object o)
- addAll(Collection c)
- clear()
- contains(Object o)
- containsAll(Collection c)
- remove(Object o)
- removeAll(Collection c)
- retainAll(Collection c)
- size()
- toArray()
- toArray(Object[] a)
- 同一Collectionに対してiterator()メソッドを2回以上実行する

トランザクション属性の設定について詳細は、“10.4.1 トランザクション管理種別とトランザクション属性”を参照してください。

また、sizeメソッドについては以下の場合には使用できません。使用した場合にはjava.lang.UnsupportedOperationExceptionが返却されます。

- インスタンス管理モードにSequentialを指定している場合
- Interstage V3の処理モードを指定している場合
- Interstage V3以前で配備したEJBアプリケーションを使用している場合

複数インスタンス検索の記述例(enumeration インタフェースの場合)

記述する処理概要

1. 呼び出すEntity Beanのlookup処理を行い、EJB homeを取得する。
2. find<METHOD>メソッドを呼び出し、プライマリキーオブジェクトを取得する。
3. ビジネスメソッドを呼び出す。

記述例

```
    (lookup処理でEJB homeの取得)
                                    :
Enumeration enum = null;
try {
    // find<METHOD>の呼出し
    enum = etyHome.findByKey(empEV);
} catch ( FinderException ex ) {
    throw new UserException( ex.getMessage() );
} catch ( RemoteException ex ) {
    throw new EJBException( ex.getMessage() );
}

if ( enum == null ) {
    examDebug( "## EmployeeCBM : empSearchN() record not found" );
    return null;
}

int count = 0;
Vector vec = new Vector();
EmployeeEntityRemote etyRemote = null;
java.lang.Object obj = null;
while( enum.hasMoreElements() ) {
    EmployeeEV view = new EmployeeEV();
    obj = enum.nextElement();
    etyRemote =
        (EmployeeEntityRemote)javax.rmi.PortableRemoteObject.narrow(obj,
            EmployeeEntityRemote.class);
    // ビジネスメソッド／抽象アクセッサメソッド(getId,getName,getDept,getAge)の呼出し
    view.setId(etyRemote.getId());
    view.setName(etyRemote.getName());
    view.setDept(etyRemote.getDept());
    view.setAge(etyRemote.getAge());
    count++;
    vec.addElement( view );
 }

                                    :
```

ejbSelectメソッドの記述例

記述する処理概要

EntityBeanメソッド内よりejbSelectメソッドを呼び出します。クライアントアプリケーション、または、他のEJBアプリケーション内からejbSelectメソッドへアクセスできません。

記述例

以下にビジネスメソッド内からejbSelectメソッドを使用したCMP2.0のEnterprise Beanの記述例を示します。

```
public abstract class OrderBean implements javax.ejb.EntityBean {

                                    :

  public int getSameDayOrderCount() {
    Collection c;
    try {
      // ejbSelectメソッドの呼出し
      c = ejbSelectOrdersOn(getDate());
    } catch (FinderException fe) {
      fe.printStackTrace();
      return 0;
    }
```

```
        return c.size();
    }

                                    :
    // ejbSelectメソッドをabstractで宣言
    public abstract java.util.Collection ejbSelectOrdersOn(java.lang.String date)
                                                throws javax.ejb.FinderException;
                                    :
}
```

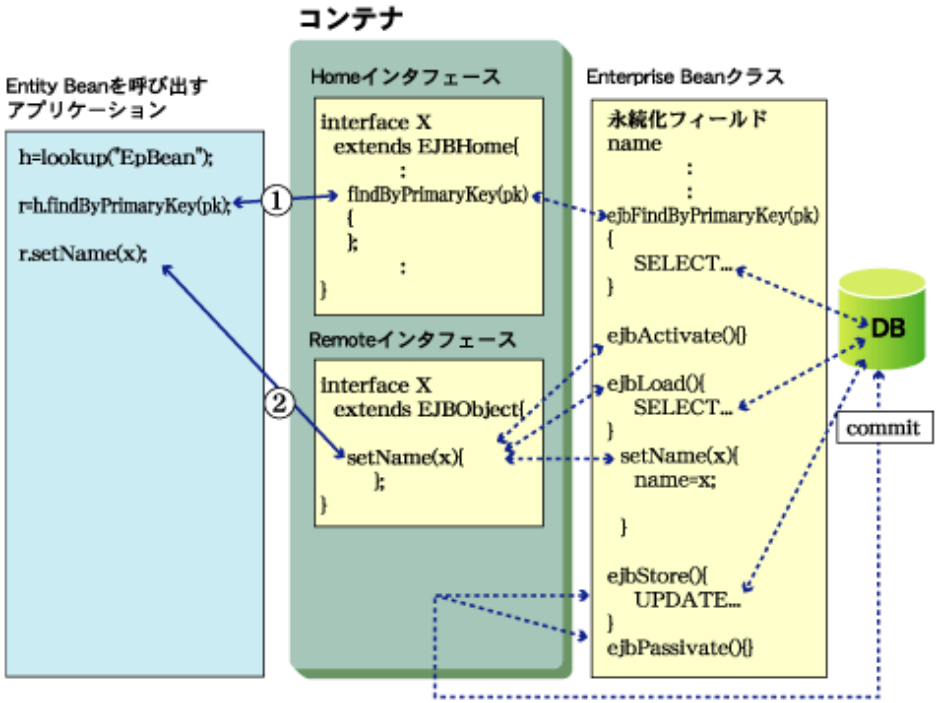
15.2.3 更新処理の記述

Entity Beanのインスタンスを更新するときのEntity Beanを呼び出すアプリケーション、コンテナおよびEnterprise Beanクラス間の処理の流れとEntity Beanを呼び出すアプリケーションの記述例について説明します。

- 処理の流れ
- 更新処理の記述例

処理の流れ

プライマリキーでEntity Beanのインスタンスを検索しデータを更新するときの処理の流れを示します。
以下の例はBMPのEntity Beanです。CMPはデータベースアクセス処理をコンテナが行いますが、処理の流れは同じです。



1. 呼出し元のアプリケーションから、findByPrimarykeyメソッドを呼び出します。
   findByPrimarykeyメソッドが発行されると、コンテナからEnterprise BeanクラスのejbFindByPrimaryKeyメソッドが呼び出されて、ejbFindByPrimaryKeyに記述したSELECT文を実行し、プライマリキーオブジェクトを作成、返却します。
2. 呼出し元のアプリケーションから、ビジネスメソッド(setNameメソッド)を呼び出します。
   ビジネスメソッドが呼び出されると、コンテナが以下の処理を行います。
   ビジネスメソッドには、永続化フィールドに更新データを設定する処理を記述します。

   ー Enterprise BeanクラスのejbActivateメソッドを呼び出す。

－ Enterprise Beanクラスのejbloadメソッドを呼び出す。ejbLoadメソッドに記述したSELECT文が実行され、永続化フィールドにデータを設定する。

－ Enterprise Beanクラスのビジネスメソッドを呼び出す。

更新処理の記述例

記述する処理概要

1. 呼び出すEntity Beanのlookup処理を行い、EJB homeを取得する。
2. findByPrimaryKeyメソッドを呼び出し、プライマリキーオブジェクトを取得する。
3. ビジネスメソッドを呼び出す。

記述例

```
(lookup処理でEJB homeの取得)
                                :
EmployeeEntityRemote etyRemote = null;
EmployeeEntityPrimaryKey pk = new EmployeeEntityPrimaryKey();
pk.ID = empEV.getId();

try {
    // findByPrimaryKeyの呼出し
    etyRemote = etyHome.findByPrimaryKey(pk);
} catch ( FinderException ex ) {
    throw new UserException( ex.getMessage() );
} catch ( RemoteException ex ) {
    throw new EJBException( ex.getMessage() );
}

// ビジネスメソッド／抽象アクセッサメソッド(setName,setDept,setAge)の呼出し
etyRemote.setName( empEV.getName() );
etyRemote.setDept( empEV.getDept() );
etyRemote.setAge( empEV.getAge() );
                                :
```

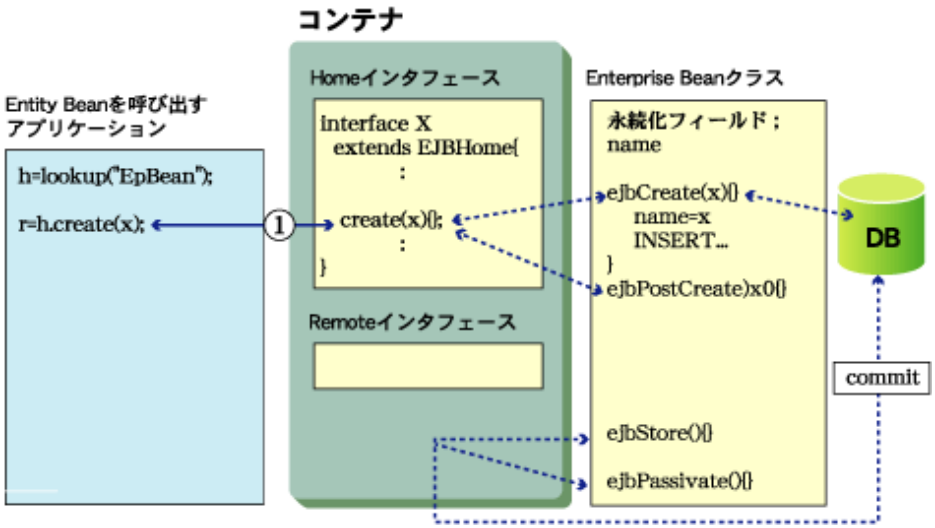
15.2.4 追加処理の記述

Entity Beanのインスタンスを追加するときのEntity Beanを呼び出すアプリケーション、コンテナおよびEnterprise Beanクラス間の処理の流れとEntity Beanを呼び出すアプリケーションの記述例について説明します。

- 処理の流れ
- 追加処理の記述例

処理の流れ

Entity Beanのインスタンスを追加するときの処理の流れを示します。
以下の例はBMPのEntity Beanです。CMPはデータベースアクセス処理をコンテナが行いますが、処理の流れは同じです。



1. 呼出し元のアプリケーションから、createメソッドを呼び出します。
   createメソッドが発行されると、コンテナが以下の処理を行います。
   - Enterprise Beanクラスの ejbCreateメソッドを呼び出す。ejbCreateメソッドに記述したINSERT文が実行され、データをINSERTする。
   - ejbPostCreateメソッドを呼び出す。

追加処理の記述例

記述する処理概要

1. 呼び出すEntity Beanのlookup処理を行い、EJB homeを取得する。
2. createメソッドを呼び出す。

記述例

```
    (lookup処理でEJB homeの取得)
                                :
try {
    etyHome.create( empEV.getId(), empEV.getName(), empEV.getDept(), empEV.getAge());
} catch ( CreateException ex ) {
throw new UserException( ex.getMessage() );
} catch ( RemoteException ex ) {
throw new EJBException( ex.getMessage() );
}
                                :
```

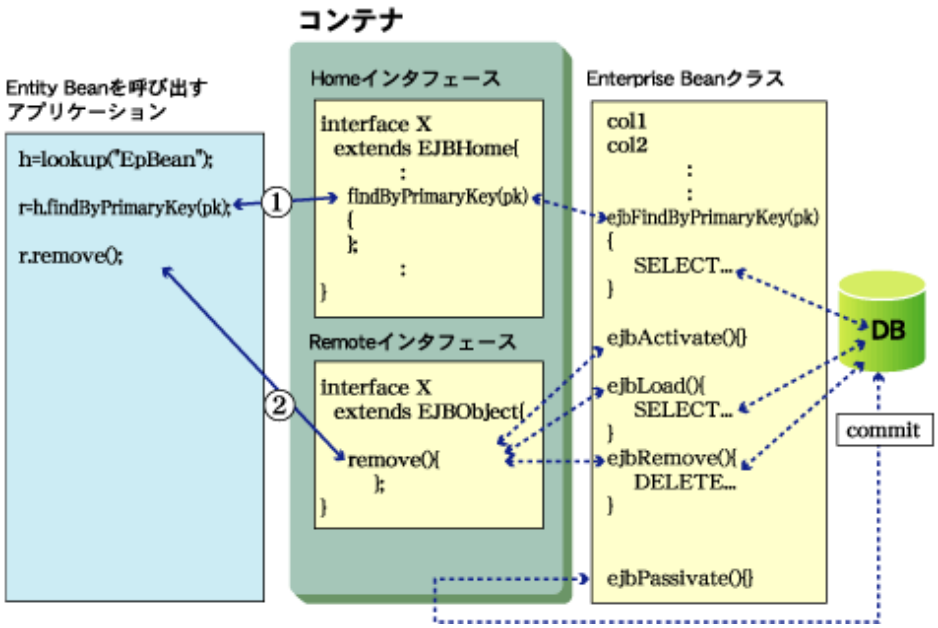
15.2.5 削除処理の記述

Entity Beanのインスタンスを削除するときのEntity Beanを呼び出すアプリケーション、コンテナおよびEnterprise Beanクラス間の処理の流れとEntity Beanを呼び出すアプリケーションの記述例について説明します。

- 処理の流れ
- 削除処理の記述例

処理の流れ

Entity Beanのインスタンスを削除するときの処理の流れを示します。
以下の例はBMPのEntity Beanです。CMPはデータベースアクセス処理をコンテナが行いますが、処理の流れは同じです。



1. 呼出し元のアプリケーションから、findByPrimarykeyメソッドを呼び出します。
   findByPrimarykeyメソッドが発行されると、コンテナからEnterprise BeanクラスのejbFindByPrimaryKeyメソッドが呼び出され、ejbFindByPrimaryKeyに記述したSELECT文を実行し、プライマリキーオブジェクトを作成、返却します。
2. 呼出し元のアプリケーションから、removeメソッドを呼び出します。
   removeメソッドが呼び出されると、コンテナが以下の処理を行います。
   - Enterprise BeanクラスのejbActivateメソッドを呼び出す。
   - Enterprise BeanクラスのejbLoadメソッドを呼び出す。ejbLoadメソッドに記述したSELECT文が実行され、永続化フィールドにデータを設定する。
   - Enterprise BeanクラスのejbRemoveメソッドを呼び出す。ejbRemoveメソッドに記述したDELETE文が実行され、データを削除する。

削除処理の記述例

記述する処理概要

1. 呼び出すEntity Beanのlookup処理を行い、EJB homeを取得する。
2. findByPrimaryKeyメソッドを呼び出し、プライマリキーオブジェクトを取得する。
3. removeメソッドを呼び出す。

記述例

```
    (lookup処理でEJB homeの取得)
                                    :
EmployeeEntityRemote etyRemote = null;
EmployeeEntityPrimaryKey pk = new EmployeeEntityPrimaryKey();
pk.ID = empEV.getId();
try {
    etyRemote = etyHome.findByPrimaryKey(pk);
} catch ( FinderException ex ) {
    throw new UserException( ex.getMessage() );
} catch ( RemoteException ex ) {
    throw new EJBException( ex.getMessage() );
}

try {
    etyRemote.remove();
```

```
    } catch ( RemoveException ex ) {
        throw new UserException( ex.getMessage() );
    } catch ( RemoteException ex ) {
        throw new EJBException( ex.getMessage() );
    }
                                    :
```

15.2.6 例外処理

Entity Beanを呼び出したときの例外処理は、以下の指針で記述してください。

返却されるException	指針
・ java.rmi.RemoteException ・ java.rmi.NoSuchObjectException	アプリケーションで対処できないシステムレベルの例外が通知されたことを意味します。EJBアプリケーションでこの例外をcatchした場合は、javax.ejb.EJBExceptionをthrowしてください。
・ javax.ejb.CreateException ・ javax.ejb.DuplicateKeyException ・ javax.ejb.FinderException ・ javax.ejb.ObjectNotFoundException ・ javax.ejb.RemoveException ・ ユーザ例外	ユーザロジックの例外などアプリケーションで対処可能な例外が通知されたことを意味します。ユーザ任意の例外処理を行ってください。

15.2.7 Entity Beanを呼び出す場合の記述例

以下は、Entity Beanを呼び出すクライアントアプリケーションの記述例です。

SampleClient.java

```
package sample;

import javax.ejb.* ;
import java.rmi.*;

public class SampleClient
{

  public static void main(String args[]){

    EmployeeHome   home = null;
    EmployeeRemote remote = null;

    // lookupの処理を行います
    try{
      // InitialContext獲得
      // デフォルトコンストラクタ指定で獲得する(注1)
      javax.naming.Context ic = new javax.naming.InitialContext();

      // lookup            (1)
      java.lang.Object Obj = (java.lang.Object)ic.lookup("Employee");

      /* Enterprise Beanオブジェクトのlookup処理を行います */
      // homeのnarrow()    (2)
      home = (EmployeeHome)javax.rmi.PortableRemoteObject.narrow( Obj, EmployeeHome.class);

    }catch (Exception e){
      System.err.println("lookup Failed");
```

```
            e.printStackTrace();
            System.exit(1);
        }

        // findByPrimaryKeyの呼出し
        try{
            String ID = "100" ;
            EmployeePrimaryKey pk = new EmployeePrimaryKey(ID);
            remote = home.findByPrimaryKey(pk);
        }catch (FinderException e){
            System.err.println("FinderException Catched");
            e.printStackTrace();
            System.exit(1);
        }catch (RemoteException e){
            System.err.println("RemoteException Catched");
            e.printStackTrace();
            System.exit(1);
        }

        // invoke business method
        try{
            System.out.println(remote.getID());
            System.out.println(remote.getNAME());
            System.out.println(remote.getDEPT());
            System.out.println(remote.getAGE());
        }catch (RemoteException e){
            System.err.println("RemoteException Catched");
            e.printStackTrace();
            System.exit(1);
        }catch (Exception e){
            System.err.println("create Failed");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

注意

EJBクライアントを使用してクライアントアプリケーションを起動する際に、以下の環境プロパティを指定し起動してください。
J2EEアプリケーションクライアントと、サーブレット/JSPが提供する各機能を使用する場合には、“JNDIサービスプロバイダの環境設定”を参照してください。

```
-Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
```

例)

```
java -Djava.naming.factory.initial
=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient Javaアプリケーション名
```

LocalHomeインタフェースを経由した場合

記述例はHomeインタフェースの例ですが、LocalHomeインタフェースを経由してSession Beanにアクセスする場合には、記述例の**(1)**と**(2)**の処理が異なります。
以下のようにlookupで取得したオブジェクトを直接キャストして、narrowメソッドを実行する必要はありません。

```
Context initialContext = new InitialContext();
AccountHome accountHome =
    (AccountHome)initialContext.lookup("java:comp/env/ejb/accounts");
```

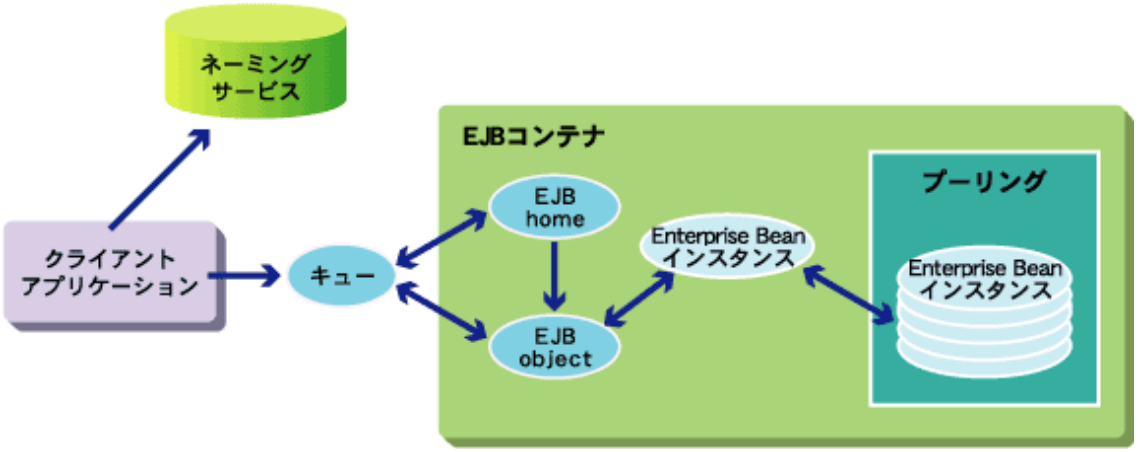
15.3 Message-driven Beanの呼出し方法

Message-driven Beanに対してのクライアントアプリケーションは、JMSの送信アプリケーションです。JMSの送信アプリケーションからMessage-driven Beanを呼び出す方法の詳細や記述例については、“第6部 JMS編”を参照してください。

15.4 Enterprise Beanインスタンス／EJB object／EJB homeの関係

Enterprise Beanインスタンス／EJB object／EJB homeの関係について説明します。

クライアントアプリケーションは、始めにネーミングサービスからEJB homeのオブジェクトリファレンスを取得します。EJB homeのメソッド(createメソッドやfinderメソッドなど)を実行して、EJB objectのオブジェクトリファレンスを取得します。EJB objectに対してメソッドを実行すると、コンテナは必要に応じてEnterprise Beanインスタンスに処理を渡します。
以下にSTATELESS Session Beanの場合の関係図を示します。



以下について説明します。

- EJB objectとEnterprise Beanインスタンスの生成タイミング
- Enterprise Beanインスタンス生成／削除時に呼び出されるメソッド

EJB objectとEnterprise Beanインスタンスの生成タイミング

Enterprise Beanの種類によって、EJB objectとEnterprise Beanインスタンスの生成タイミングや生成される数が異なります。

STATELESS Session Bean

STATELESS Session Beanは、トランザクション状態やアプリケーション変数をEnterprise Beanインスタンスに保持しないため、Enterprise Beanインスタンスをプーリングして利用します。
クライアントからリクエストが来ると、EJBコンテナはプールからインスタンスを1つ取得して、そのインスタンス上で処理を実行します。プールにインスタンスがない場合、インスタンスを1つ作成し、そのインスタンス上で処理を実行します。
処理が終了したらインスタンスをプールに戻して、クライアントに処理結果を返却します。

STATELESS Session Beanのインスタンスを、プロセスへの初回アクセス時に同時接続数に指定した数だけ生成します。
初期起動インスタンス数が指定された場合、STATELESS Session Beanのインスタンスは、IJServer起動時に指定された数分生成されます。
また、クライアントのリクエストをEnterprise Beanインスタンスに渡すEJB objectを、起動時に1つだけ生成します。
EJB objectとEnterprise Beanインスタンスは、IJServerの停止時に削除されます。

STATEFUL Session Bean

STATEFUL Session Beanは、トランザクション状態やアプリケーション変数をEnterprise Beanインスタンスに保持できます。
EJB homeに対してcreateメソッドが実行されるたびに、EJB objectとEnterprise Beanインスタンスを生成して、同一のEJB

objectにアクセスすると、同一のEnterprise Beanインスタンスで処理を実行します。処理終了後にEJB objectに対してremoveメソッドを実行して、EJB objectとインスタンスを削除します。

Entity Bean

Entity BeanのEJB objectは、EJB homeメソッドに対してメソッドを実行するタイミングで、必要に応じてコンテナが生成します。

Entity Beanのインスタンスを初期起動インスタンス数だけプーリングします。Enterprise Beanインスタンスの生成タイミングは、インスタンス生成モードによって異なります。STATELESS Session BeanのEnterprise Beanインスタンスとは位置付けが異なり、DBMSのレコードにEntity BeanのEnterprise Beanインスタンスをマッピングします。1クライアント(1トランザクション)でアクセスしたDBMSのレコード数分だけEnterprise Beanインスタンスがプールから使用され、トランザクションを完了した際にEnterprise Beanインスタンスをプールに返却します。

プールにEnterprise Beanインスタンスが存在しない場合、同一トランザクション内で使用したEnterprise Beanインスタンスを1つ選択してインスタンスに格納されているレコードデータをDBMSに反映し、そのインスタンスを別のレコードデータを格納するインスタンスとして再利用します。同一トランザクション内で使用したEnterprise Beanインスタンスが存在しない場合には、Enterprise Beanインスタンスを1つだけ動的に生成して使用します。使用したEnterprise Beanインスタンスはトランザクション完了後にプールに返却しますが、動的に使用したインスタンスについてはトランザクションが完了した際に破棄します。Enterprise Beanインスタンスが頻繁に再利用されると、性能に影響があります。

Entity Beanの初期起動インスタンス数を定義します。設定方法については、Interstage管理コンソールのヘルプを参照してください。

レコードデータの反映はBMPの場合にはejbStoreメソッドを実行して行います。CMP1.1もしくはCMP2.0の場合には、コンテナがデータベースに対してUPDATE文を実行して行います。
複数トランザクションで同時に処理を実行した場合、SELECT文とUPDATE文が同時に実行されるとデッドロックが発生することがあります。
このような場合にはSELECT文にFOR UPDATEを指定するようにしてください。
詳細は“29.9.11 デッドロックが発生する場合”を参照してください。

Message-driven Bean

Message-driven Beanは、トランザクション状態やアプリケーション変数をEnterprise Beanインスタンスに保持しないため、Enterprise Beanインスタンスをプーリングして利用します。
Destinationからメッセージが配信されると、EJBコンテナはプールからインスタンスを1つ取得してそのインスタンス上で処理を実行します。処理が終了したらインスタンスをプールに戻して処理を終了します。

Enterprise Beanインスタンス生成／削除時に呼び出されるメソッド

Enterprise Beanインスタンスの生成や削除のタイミングで、Enterprise Beanインスタンスの各種メソッドが実行されます。

Enterprise Beanインスタンス生成時

Bean種別	メソッド名
STATELESS Session Bean	setSessionContext ejbCreate
STATEFUL Session Bean	setSessionContext ejbCreate
Entity Bean	setEntityContext
Message-driven Bean	setMessageDrivenContext ejbCreate

Enterprise Beanインスタンス削除時

Bean種別	メソッド名
STATELESS Session Bean	ejbRemove
STATEFUL Session Bean	ejbRemove

Bean種別	メソッド名
Entity Bean	unsetEntityContext
Message-driven Bean	ejbRemove

STATELESS Session Bean／Entity Bean／Message-driven Beanの場合、インスタンスを停止時に削除するため、上記メソッドを停止時に呼び出します。
ただし、強制停止の場合にはメソッドは実行されずに強制的に停止されます。

15.5 トランザクションを使用する場合

トランザクションを使用したアプリケーションについて説明します。

15.5.1 SessionSynchronizationインタフェースを使用したトランザクション機能

トランザクションをコンテナで制御する場合、STATEFULのSession Beanでは、javax.ejb.SessionSynchronizationインタフェースを使用できます。
以下に、SessionSynchronizationインタフェースのメソッドと、Session BeanのSynchronization機能を使用したEJBアプリケーションの作成方法を説明します。

参考

Interstage StudioでEJBアプリケーションを作成する場合、SessionSynchronizationインタフェースを自動生成できます。

SessionSynchronizationインタフェースのメソッド

メソッド名	内容
afterBegin	トランザクション内で最初のビジネスメソッド呼出しの前にコンテナより呼び出され、必要なデータベース処理を行います。
beforeCompletion	トランザクションコミット時、リソースマネージャに対してコミットを行う前にコンテナより呼び出されます。 当メソッドでは、キャッシュされたデータベースの更新を書き出すことができます。 また、setRollbackOnlyメソッドを呼び出して、トランザクションにロールバックを設定できます。
afterCompletion	トランザクション完了時にコンテナより呼び出されます。 引数がTrueの場合はトランザクションがコミットしたことを示し、Falseの場合はRollbackしたことを示します。

15.5.2 EJBサービスが提供するトランザクション制御の例外処理

EJBサービスが提供するトランザクションで、トランザクション管理種別を“Container”に指定した場合の例外処理と、SessionSynchronizationインタフェースを使用した場合の例外処理について説明します。
EJBサービスが提供するトランザクションの詳細は、“10.4.1 トランザクション管理種別とトランザクション属性”を参照してください。
以下に、各属性を指定した場合の例外処理について示します。

- トランザクション管理種別を“Container”に指定した場合の例外処理
- SessionSynchronizationインタフェースを使用した場合の例外処理

トランザクション管理種別を“Container”に指定した場合の例外処理

トランザクション属性		例外処理の内容	
		ユーザ例外発生時	システム例外発生時
Mandatory		呼出し元に、メソッド内で返却されたユーザ例外が返却されます。	トランザクションをrollbackするように宣言(マーク)し、呼出し元へは“TransactionRolledBackException”が返却されます。呼出し元でトランザクションがrollbackされます。
Required	呼出し元でトランザクションが開始されている場合	トランザクションがrollbackにマークされている場合、呼出し元へは“TransactionRolledBackException”例外が返却され、呼出し元で、トランザクションがrollbackされます(rollbackされなくてはなりません)。 上記以外の場合、ユーザ例外が返却されます。	コンテナがトランザクションをrollbackするようマーク(宣言)します。呼出し元へは“TransactionRolledBackException”例外が返却され、呼出し元で、トランザクションがrollbackされます(rollbackされなくてはなりません)。
	呼出し元でトランザクションが開始されていない場合	コンテナは、トランザクションがrollbackにマークされている場合はrollbackを行い、マークされていない場合はcommitを行います。 呼び出し元へは、メソッド内で返却されたユーザ例外が返却されます。	コンテナにより、トランザクションがrollbackされ、呼出し元へは“RemoteException”例外が返却されます。
Supports	呼出し元でトランザクションが開始されている場合	当属性が指定されたメソッドは、SessionContext/EntityContextインタフェースのsetRollbackOnlyメソッドを使用することはできません。呼出し元へは、メソッド内で返却されたユーザ例外が返却されます。	コンテナは、トランザクションをrollbackするようマーク(宣言)し、呼出し元へは“TransactionRolledBackException”が返却されます。呼出し元でトランザクションがrollbackされます。
	呼出し元でトランザクションが開始されていない場合		呼出し元へは“RemoteException”を返却します。
RequiresNew	呼出し元でトランザクションが開始されている場合	トランザクションをrollbackするようマークされている場合は、rollbackを行います。 マークされていない場合は、commitを行います。中断されたトランザクションは、コンテナにより再開され、呼出し元へユーザ例外が返却されます。	トランザクションは、rollbackされます。中断されたトランザクションをコンテナにより再開され、呼出し元へは“RemoteException”が返却されます。
	呼出し元でトランザクションが開始されていない場合	トランザクションをrollbackするようマークされている場合は、rollbackを行います。 マークされていない場合は、commitを行います。呼出し元へはユーザ例外が返却されます。	トランザクションは、rollbackされます。呼出し元へは“RemoteException”が返却されます。
NotSupported		停止されたトランザクションはコンテナによって再開され、当メソッド呼出し元へはユーザ例外が返却されます。	停止されたトランザクションは、コンテナによって再開されます。呼出し元へは“RemoteException”例外が返却されます。

トランザクション属性	例外処理の内容	
	ユーザ例外発生時	システム例外発生時
Never	呼出し元へは、メソッド内で返却されたユーザ例外が返却されます。	呼出し元へは、“RemoteException”が返却されます。

SessionSynchronizationインタフェースを使用した場合の例外処理

メソッド名	例外処理の内容
afterBegin	トランザクションにロールバックが指定され、“TransactionRolledbackException”例外がビジネスメソッドの呼出し元へ通知されます。
beforeCompletion	トランザクションがロールバックされ、コミットの発行元に対して“HeuristicRolledbackException”例外が通知されます。
afterCompletion	例外は通知されません。

例

記述例

以下に、SessionSynchronizationインタフェースを使用した場合の記述例を示します。
SessionSynchronizationインタフェース関連の処理は太字で示しています。

SampleBean.java

```
package Sample;

import javax.ejb.*;
import java.rmi.*;

public class SampleBean
    extends Object implements SessionBean,SessionSynchronization
{

    // constructor
    public void SampleBean() {
            ・・・
        /*  Enterprise Bean自身のコンストラクタの処理を記述します  */
    }

    // receive SessionContext
    public void setSessionContext(SessionContext ctx)
        throws EJBException {
            ・・・
        /* コンテナによって保守されているコンテキストへの  */
        /* アクセスを行い、必要な情報を取得します          */
    }

    // startup work
    public void ejbCreate(String sn)
        throws EJBException,
        CreateException {
            ・・・
        /* インスタンス変数の初期化や、データベースや          */
        /* ファイルのopenなど、Enterprise Beanのインスタンスが */
        /* createされたときの処理を記述します                  */
    }

    // business method
```

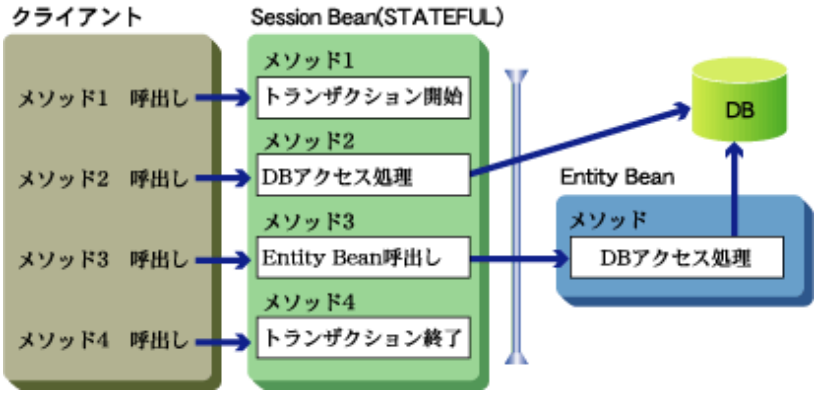
```
    public String business(String s)
        throws EJBException {
            ・・・          /* ビジネスメソッドの処理を記述します */
    }

    // termination work
    public void ejbRemove()
        throws EJBException {
            ・・・     /* インスタンスがremoveされるときの処理を記述します */
                       /* openしているリソースはclose処理が必要です         */
    }

    // work for passivation
    public void ejbPassivate()
        throws EJBException {
            ・・・                        /* 本バージョンでは呼ばれません */
    }

    // work for activation
    public void ejbActivate()
        throws EJBException {
            ・・・                        /* 本バージョンでは呼ばれません */
    }
    // work after transaction begin
    public void afterBegin()
        throws EJBException {
            ・・・
            /* トランザクション内で最初に呼び出されるビジネスメソッド  */
            /* を実行する前に呼び出されます。                          */
    }

    // work before transaction commit
    public void beforeCompletion()
        throws EJBException {
             ・・・
            /* トランザクションのコミット時、リソースに対して     */
            /* コミットを実行する前に呼び出されます。             */
    }

    // work after transaction commit
    public void afterCompletion ()
        throws EJBException {
            ・・・
        /* トランザクションのコミット時、リソースに対して     */
        /* コミットを実行した後に呼び出されます。             */
    }
}
```

15.5.3 トランザクション使用時の注意事項

Session BeanからDB連携を行う場合、Session Beanの属性(STATEFUL、STATELESS)により注意すべき事項について説明します。

STATEFULの場合

1対1の対話の状態が保持される STATEFULの場合、ビジネスメソッドを越えてトランザクションの処理ができます。1トランザクション内で、複数メソッドの処理ができます。

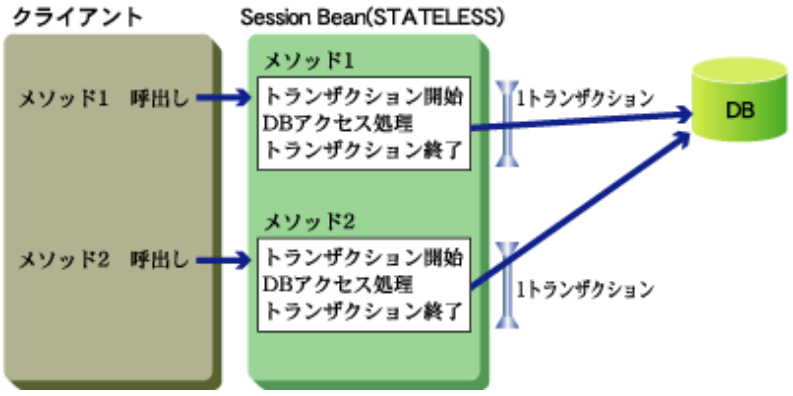


注意

Entity Beanを使用する場合、EJBサービスが提供するトランザクション機能でトランザクションを開始していないと正しく動作しません。

STATELESSの場合

1対1の対話の状態が保持されないSTATELESSの場合、いったんビジネスメソッドを抜けると、そのトランザクションを引き継いで処理を行うことができません。ビジネスメソッドを抜ける前にトランザクションが完結するようにしてください。
1メソッドを1トランザクションで処理します。ビジネスメソッドを抜けたときにトランザクションが完結していない場合、トランザクションは自動的にロールバックされます。



15.6 Javaアプレットを使用する場合（プレインストール型Javaライブラリ）

ここでは、EJBアプリケーションを呼び出すJavaアプレットの開発手順について説明します。

15.6.1 開発手順

以下について説明します。

- Javaクラスファイルのアーカイブ
- アプレットのプログラミング
- Javaクラスファイルのアーカイブ

HTMLファイルの記述

アプレットを実行するには、HTMLファイルで<applet>タグを使用してアプレットを指定します。

注意

JBKプラグインを使用してください。ブラウザのJavaVMやJava Plug-inは使用できません。

例

記述例

以下にHTMLファイルの記述例を示します(タイトル: Java sample Applet、Javaアプレット: Sample)。

JBKプラグインを使用する場合

JBKプラグインを使用する場合は、HTMLファイルを以下のように記述してください。詳細については、Interstage Studioの“J Business Kit オンラインマニュアル”を参照してください。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67"
WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
  NAME="Sample" CODE="Sample.class" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

JBKプラグインを使用する場合(jar形式アーカイブファイル使用)

<PARAM>タグのARCHIVE指定、および<EMBED>タグのARCHIVE指定でSample.jarをダウンロードするように指定します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67"
WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
  CODE="Sample.class" ARCHIVE="Sample.jar" WIDTH=300 HEIGHT=250>
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

アプレットのプログラミング

EJBクライアントを使用してクライアントアプリケーションをJavaアプレットとして開発する場合は、Javaアプリケーションと以下が違います。また、HTMLファイルの<applet>タグで記述したクラス名でクラス宣言します。

- EJBアプリケーションのオブジェクトの所在をネーミングサービスに問い合わせるlookup処理

例

以下に、Javaアプレットでのlookup処理の記述例を示します。

```
import java.awt.*;              // 抽象ウィンドウツールキットクラス

public class Sample extends java.applet.Applet //appletクラスの宣言
{
・・・
// InitialContext獲得
Hashtable env = new Hashtable();                                    ・・・・1
env.put("java.naming.factory.initial",
        "com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient");  ・・・・1
env.put("java.naming.applet", this);                                ・・・・1
javax.naming.Context ic = new javax.naming.InitialContext(env);     ・・・・2
// lookup
java.lang.Object Obj = (java.lang.Object)ic.lookup("SampleBean");      ・・・3
// homeのnarrow()
h = (SampleHome)javax.rmi.PortableRemoteObject.narrow(Obj, SampleHome.class); ・・4
}
```

1. Contextを作成するときの環境情報を設定します。このとおりに記述してください。
2. lookupするためのContextを作成します。このとおりに記述してください。
3. lookupを行います。引数にEJBアプリケーション名を指定してください。lookupに失敗した場合、javax.naming.NameNotFoundException例外が発生します。
   失敗の理由は当例外の詳細メッセージとして通知されます。
   lookupに失敗したときの対処方法については“メッセージ集”の“J2EE使用時に出力される例外”の“lookup処理で例外が発生した場合の対処”を参照してください。
4. lookupしたオブジェクトをnarrowします。javax.rmi.PortableRemoteObject.narrowを発行してください。

注意

- InitialContextの獲得(上記1～2)は、アプリケーション内で1回だけ実行するようにコンストラクタなどで実行してください。
- プレインストール型の環境変数の設定方法については、“4.2.1 クライアント環境での環境設定”を参照してください。
- Javaアプレットのファイル名は、アプレット内で宣言したクラス名と同じ(大文字／小文字の区別あり)にしてください。

Javaクラスファイルのアーカイブ

以下のクラスファイルをjarファイル化します。

- アプレット

複数のクラスファイルをjarファイル化することにより、Webサーバからのダウンロードにかかる時間を短縮できます。

アプレットのjarファイル化

Interstage Studioでアプレットを作成する場合は自動的にjarファイル化されます。Interstage Studioを使用したアプレットの開発についての詳細は、“Interstage Studio ユーザーズガイド”を参照してください。
Interstage Studioを使用しない場合は、jarコマンドを使用してjarファイル化してください。

jarコマンドの使用方法

jarコマンドの使用方法については、JDKのドキュメントを参照してください。

例

以下に、jarコマンドの使用例を示します。

Windows32/64

```
jar cvf SampleApplet.jar *.class samplepkg¥*.class
```

Solaris64 Linux32/64

```
jar cvf SampleApplet.jar *.class samplepkg/*.class
```

作成するjarファイル名とjarファイル化するクラスファイルを指定します。作成されたjarファイルにはサブフォルダ内のファイルが含まれています。

注意

Portable-ORBを使用しないでアプレットを利用する場合は、使用するEJBアプリケーションのクライアント配布物をWebサーバからダウンロードして使用することはできません。クライアント配布物はクライアント環境に複写して、複写先のjarまたはフォルダをクラスパスに設定して使用してください。

15.6.2 クライアント環境の設定

以下について説明します。

- Javaライブラリに対する権限の設定

Javaライブラリに対する権限の設定

Javaアプレットを実行する場合は、Javaライブラリに対する権限を設定してください。
policytool（JDK添付）によるJavaライブラリに対する権限の設定方法を以下に示します。

1. policytoolを起動します。
   policytool（コマンド入力）
2. 起動後、最初に表示される[Policy Tool]の画面上で[Add Policy Entry]ボタンを押下します。
3. 表示された[Policy Entry]の画面上で、各項目の値を以下のように設定します。%JAVA_HOME%にはJDKもしくはJREをインストールしたディレクトリ(例: C:/Interstage/JDK8)を指定してください。

項目	設定値
CodeBase:	[JDKを使用する場合] file:<CORBAサービスクライアントインストールディレクトリ>/etc/class/ODjava4.jar (注) file:/%JAVA_HOME%/jre/lib/-
	[JREを使用する場合] file:<CORBAサービスクライアントインストールディレクトリ>/etc/class/ODjava4.jar (注) file:/%JAVA_HOME%/lib/-
signed by:	(なし)

注) 区切り文字として'/'を使用してください。

4. [Policy Entry]の画面上で[Add Permission]ボタンを押下します。

5. 表示された[Permissions]の画面上で、各項目の値を以下の表のように設定します。[Permissions]画面上での権限の設定は、一組の[Permission:／Target Name:／Actions:]の値を設定し、[OK]ボタンを押下します。この結果、[Policy Entry]の画面に戻ります。次の一組の値を設定するには、再度、[Add Permission]ボタンを押下します。これを繰り返し、必要な情報を設定します。
設定後、[Policy Entry]の画面上で[Done]ボタンを押下します。

通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、この権限以外は設定しないでください。

権限種別	設定を行う権限		
	Permission	Target Name	Actions
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	loadLibrary.DLL名 (注)	設定不要
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read

注) インストールしている機能、および使用するJDK/JREにより以下のダイナミックリンクライブラリ(DLL)名を指定します(拡張子は指定不要)。

インストールしている機能	ダイナミックリンクライブラリ
CORBAサービスクライアント(クライアント機能)	ODjava4
CORBAサービス(サーバ機能)	ODjavas4

CORBAサービスの内部ログを採取する場合に必要な権限 (注1)

権限種別	設定を行う権限		
	Permission	Target Name	Actions
プロパティ権限	PropertyPermission(java.util.PropertyPermission)	user.dir	read
		java.class.path	read
ファイル権限	FilePermission(java.io.FilePermission)	${user.dir}¥*	read, write
		%OD_HOME%¥etc¥config (注2)	read

注1) CORBAサービスの内部ログの詳細は、“チューニングガイド”の“config”を参照してください。なお、ログ採取後は追加した権限を削除してください。
注2) %OD_HOME%は、CORBAサービス、CORBAサービスクライアントのインストールパスを指定します(デフォルトは、C:¥Interstage¥ODWIN)。

6. [Policy Entry]の画面上で、メニュー[File]→[Save]を選択します。
7. メニュー[File]→[Exit]を選択し、policytoolを終了します。

注意

policytoolの初回実行時は、policytool終了前に、[Policy Tool]の画面からメニュー[File]→[Save As]を選択し、ポリシーファイルの名前と格納位置を指定する必要があります。ポリシーファイル指定の詳細については、“15.8.1 デジタル署名”を参照してください。

15.7 Javaアプレットを使用する場合（Portable-ORB）

ここでは、EJBアプリケーションを呼び出すJavaアプレットの開発手順について説明します。

Portable-ORBを使用する場合

EJBサービスでは、クライアントアプリケーションをJavaアプレットとして開発した場合、Portable-ORBを使用できます。

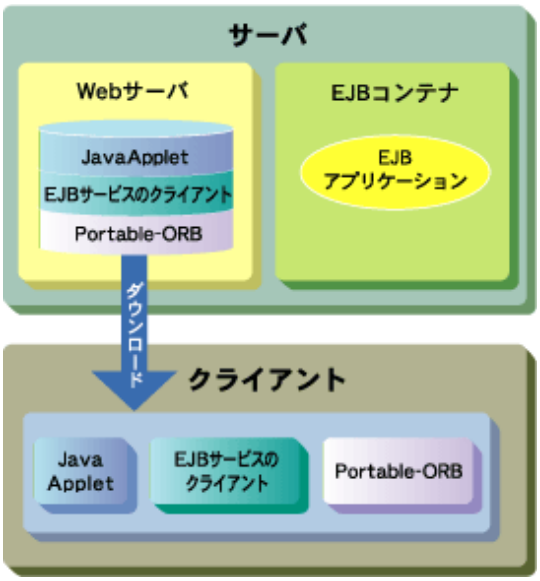
Portable-ORBには以下の特長があります。

インターネット/イントラネットへの適応が可能

Webサーバからダウンロードして運用できるため、ネットワークコンピュータ/モバイル端末などのThinクライアントをInterstageのクライアントとして活用できます。

優れた運用性/保守性

Portable-ORBは個々のクライアント端末に事前にインストールする必要がないため、クライアントの運用・保守コストを削減できます。



注意

EJBサービスでPortable-ORBを使用する場合は、JBKプラグインを使用してください。ブラウザのJavaVMやJava Plug-inは使用できません。
JBKプラグインの詳細については、Interstage Studioの“J Business Kit オンラインマニュアル”を参照してください。

15.7.1 開発手順

以下について説明します。

- HTMLファイルの記述
- アプレットのプログラミング
- Javaクラスファイルのアーカイブ
- WebサーバへのJavaアプレットの登録
- Webサーバで行うPortable-ORBの環境設定

HTMLファイルの記述

アプレットを実行するには、HTMLファイルで<applet>タグを使用してアプレットを指定します。

また、動作するJava VMによってPortable-ORBのファイルを使用します。
Portable-ORBをダウンロードする運用では、Javaアプレットを実行するHTMLファイルで、<APPLET ARCHIVE>タグまたは<PARAM>タグ(cabbase)を記述します。

ダウンロードするファイル

アプレット実行時にWebサーバからダウンロードするjarファイルを以下に示します。これらのjarファイルは、デジタル署名を行いWebサーバに格納します。

アプレットのjarファイル

以下のjarファイルです。

- クライアントアプリケーションとなるアプレットをjarファイル化したもの
- 対象となるEJBアプリケーションのクライアント配布物をjarファイル化したもの

Portable-ORB用のjarファイル

Portable-ORB用のjarファイル名と格納先を以下に示します。

ファイル名	格納先
ODporbROI4_plugin.jar	Windows32/64 C:¥Interstage¥Porb¥lib Solaris64 Linux32/64 /opt/FJSVporb/lib

参考

ODporbROI4_plugin.jarの代わりに、以下のファイルも使用できます。

- ODporb4_plugin.jar
- CosNaming4_plugin.jar
- ODroi4_plugin.jar

EJBサービスのクライアント用jarファイル

EJBサービスのクライアント用のjarファイル名と格納先を以下に示します。

ファイル名	格納先
・ fjcontainer94_plugin.jar ・ fjentity94_plugin.jar (注)	Windows32/64 C:¥Interstage¥EJB¥lib、または、C:¥Interstage¥EJBCL¥lib Solaris64 Linux32/64 /opt/FJSVejb/lib、または、C:¥Interstage¥EJBCL¥lib

注) クライアントからEntity Beanに直接アクセスする場合だけ必要です。

注意

JBKプラグインを使用してください。ブラウザのJavaVMやJava Plug-inは使用できません。

例

記述例

以下に、各Java VMでHTMLファイルの記述例を示します(タイトル: Java sample Applet、Javaアプレット名: Sample)。

JBKプラグインを使用する場合

<PARAM>タグのARCHIVE指定、および<EMBED>タグのARCHIVE指定でSample.jarをダウンロードするように指定します。

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67"
WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="SampleApplet.jar,SampleClient.jar,
        ODporbROI4_plugin.jar,fjcontainer94_plugin.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
  CODE="Sample.class" WIDTH=300 HEIGHT=250
  ARCHIVE="Sample.jar" PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

Portable-ORBをダウンロードしない場合(JBKプラグインを使用する場合)

```
<HTML>
<HEAD><!--demo.html-->
<TITLE>Java sample Applet </TITLE>
</HEAD>
<BODY>
<OBJECT CLASSID="CLSID:BEA62964-C40B-11D1-AACA-00A0C9216A67"
WIDTH=300 HEIGHT=250>
<PARAM NAME="TYPE" VALUE="application/x-JBK-Plugin">
<PARAM NAME="CODE" VALUE="Sample.class">
<PARAM NAME="ARCHIVE" VALUE="Sample.jar">
<PARAM NAME="PORB_HOME" VALUE="PORBDIR">
<COMMENT>
<EMBED TYPE="application/x-JBK-Plugin"
  CODE="Sample.class" WIDTH=300 HEIGHT=250
  ARCHIVE="Sample.jar" PORB_HOME="PORBDIR">
</EMBED>
</COMMENT>
</OBJECT>
</BODY>
</HTML>
```

参考

- 上記のPortable-ORBをダウンロードする場合の例では、各Java VMで使用するPortable-ORBのファイルがHTMLファイルと同一の格納先ディレクトリを仮定したものです。同一ディレクトリにない場合は、ARCHIVE/VALUEでパスを指定します。Solaris/LinuxシステムをWebサーバとして使用する場合は、パスを設定する代わりにリンクファイルを作成することで代行できます。
- <PARAM NAME>タグで指定するPORB_HOMEには、動作環境ファイルを検索する際のサブディレクトリ名を指定します。上記の例での動作環境ファイルの格納ディレクトリは、以下のとおりです。

Webサーバのドキュメントルートディレクトリ/PORBDIR/etc
パスの指定については、“Portable-ORB動作環境ファイルの指定”を参照してください。

- JBKプラグインを使用してアプレットを実行する場合には、<APPLET>タグのかわりにJBKプラグイン用のタグを使用した記述してください。記述方法は使用するブラウザにより異なります。
  上記の例はInternet Explorerで使用できるHTMLファイルの記述例です。記述方法の詳細については、Interstage Studioの“J Business Kit オンラインマニュアル”を参照してください。

アプレットのプログラミング

EJBクライアントを使用してクライアントアプリケーションをJavaアプレットとして開発する場合は、Javaアプリケーションと以下が違います。また、HTMLファイルの<applet>タグで記述したクラス名でクラス宣言します。

- EJBアプリケーションのオブジェクトの所在をネーミングサービスに問い合わせる lookup処理

例

以下に、Javaアプレットでのlookup処理の記述例を示します。

```
import java.awt.*;              // 抽象ウィンドウツールキットクラス

public class Sample extends java.applet.Applet //appletクラスの宣言
{
・・・
// InitialContext獲得
Hashtable env = new Hashtable();                                    ・・・・1
env.put("java.naming.factory.initial",
        "com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient");  ・・・・1
env.put("java.naming.applet", this);                                ・・・・1
javax.naming.Context ic = new javax.naming.InitialContext(env);     ・・・・2
// lookup
java.lang.Object Obj = (java.lang.Object)ic.lookup("SampleBean");     ・・・3
// homeのnarrow()
h = (SampleHome)javax.rmi.PortableRemoteObject.narrow(Obj, SampleHome.class); ・・4
}
```

1. Contextを作成するときの環境情報を設定します。このとおりに記述してください。
2. lookupするためのContextを作成します。このとおりに記述してください。
3. lookupを行います。引数にEJBアプリケーション名を指定してください。lookupに失敗した場合、javax.naming.NameNotFoundException例外が発生します。
   失敗の理由は当例外の詳細メッセージとして通知されます。
   lookupに失敗したときの対処方法については“メッセージ集”の“J2EE使用時に出力される例外”の“lookup処理で例外が発生した場合の対処”を参照してください。
4. lookupしたオブジェクトをnarrowします。javax.rmi.PortableRemoteObject.narrowを発行してください。

注意

- InitialContextの獲得(上記1～2)は、アプリケーション内で1回だけ実行するようにコンストラクタなどで実行してください。
- Javaアプレットのファイル名は、アプレット内で宣言したクラス名と同じ(大文字／小文字の区別あり)にしてください。

Javaクラスファイルのアーカイブ

JavaクラスファイルをWebサーバに登録する場合、複数のファイルを一度にダウンロードしてダウンロード時間を短縮できるよう、クラスファイルをまとめたアーカイブファイルを作成します。アーカイブファイルは、jarコマンド(Java Development Kit(以降JDK)に含まれる)を使用して作成します。

jarコマンドで作成したjarアーカイブファイルは、JBKプラグインで使用できます。

ここで作成したJavaクラスファイルのアーカイブファイル(アプレット)をWebサーバからダウンロードして実行する場合は、アーカイブファイルに対して署名を行ってください。署名については、“15.8 アプレットのデジタル署名”を参照してください。

参照

jarコマンドの使用方法等については、JDKのドキュメントを参照してください。

以下のクラスファイルをjarファイル化します。

- アプレット
- クライアント配布物

複数のクラスファイルをjarファイル化することにより、Webサーバからのダウンロードにかかる時間を短縮できます。

アプレットのjarファイル化

Interstage Studioでアプレットを作成する場合は自動的にjarファイル化されます。Interstage Studioを使用したアプレットの開発についての詳細は、“Interstage Studio ユーザーズガイド”を参照してください。
Interstage Studioを使用しない場合は、jarコマンドを使用してjarファイル化してください。

クライアント配布物のjarファイル化

EJBアプリケーションの配備時に生成されるクライアント配布物を、jarコマンドを使用してjarファイル化します。
サブディレクトリを含むクラスファイルからアーカイブファイルを作成する場合のコマンド使用例を以下に示します。

例

jarコマンド使用例

作成するjarファイル名とjarファイル化するクラスファイルを指定します。作成されたjarファイルにはサブディレクトリ内のファイルが含まれています。

```
jar cvf Sample.jar *.class Samplemod¥*.class
  adding: Samplemod/_SampleintfStub.class (in=1282) (out=704) (deflated 45%)
  adding: Samplemod/Sampleintf.class (in=302) (out=215) (deflated 28%)
  adding: Samplemod/SampleintfHelper.class (in=2175) (out=994) (deflated 54%)
  adding: Samplemod/SampleintfHolder.class (in=907) (out=461) (deflated 49%)
```

注意

- アーカイブファイルとするクラスファイルを指定する場合、パス指定の情報もアーカイブファイル内に含まれます。カレントディレクトリをアーカイブファイルとするクラスファイルが格納される最上位ディレクトリに移動後、クラスファイルの構成にしたがった指定を行い、アーカイブファイルを作成してください。
- HTMLファイル内の<PARAM>タグでcabbaseを指定する場合、<APPLET>タグのARCHIVE指定は無効です。

WebサーバへのJavaアプレットの登録

HTMLファイル、Javaアプレット、IDL生成ファイルクラスをWebサーバ上の同じディレクトリに格納します。

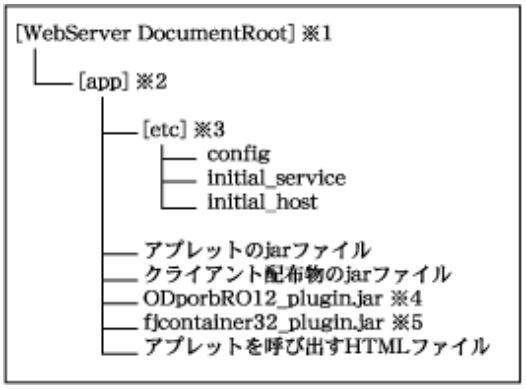
以下のjarファイルを、Webサーバに格納します。

- アプレットのjarファイル
- クライアント配布物のjarファイル
- Portable-ORB用jarファイル
- EJBサービスのクライアント用jarファイル

・ アプレットを呼び出すHTMLファイル

クライアントは、ブラウザからWebサーバを参照することにより、上記ファイルをダウンロードします。

各ファイルをWebサーバに格納するときの構成例を以下に示します。



※1 [WebServer DocumentRoot]は、Webサーバの設定でユーザが任意に指定します。
※2 [app]は、当該アプリケーションを格納するディレクトリです。ユーザが任意に設定します。
※3 [etc]は、Portable-ORBが使用する動作環境設定ファイルを格納します。詳細は"Portable-ORB動作環境ファイルの指定"を参照してください。
※4 Portable-ORB用のjarファイル
※5 EJBサービスのクライアント用jarファイル

Webサーバで行うPortable-ORBの環境設定

"Portable-ORB動作環境ファイルの指定"を参照し、Portable-ORB/JavaアプレットをWebサーバからダウンロードする場合の環境設定手順にしたがって環境を設定してください。

15.7.2 クライアント環境の設定

Portable-ORBを使用したクライアントアプリケーション(アプレット)を運用する場合、クライアント環境で以下の設定をしてください。

- ORB(Object Request Broker)の指定
- Portable-ORB動作環境ファイルの指定
- JBKプラグインの設定ファイルの編集

ORB(Object Request Broker)の指定

アプレットを実行するための環境設定として、使用するORBを選択してください。
orb.propertiesファイルまたはJBKプラグインに付属されているjbkplugin.propertiesファイルに、JavaVMの起動オプションとしてJava実行環境の以下のプロパティ情報を設定します。

プロパティ名	設定値
org.omg.CORBA.ORBClass	com.fujitsu.ObjectDirector.CORBA.ORB
javax.rmi.CORBA.StubClass	com.fujitsu.ObjectDirector.rmi.CORBA.StubDelegateImpl
javax.rmi.CORBA.UtilClass	com.fujitsu.ObjectDirector.rmi.CORBA.UtilDelegateImpl
javax.rmi.CORBA.PortableRemoteObjectClass	com.fujitsu.ObjectDirector.rmi.CORBA.PortableRemoteObjectDelegateImpl

注意

Portable-ORBを使用したアプレット運用を行う場合は、以下のプロパティを指定しないでください。

- org.omg.CORBA.ORBSingletonClass

Portable-ORB動作環境ファイルの指定

Portable-ORBを使用する場合、動作環境ファイルの格納位置を指定するため、HTMLファイルでPORB_HOMEパラメタを設定してください。
Portable-ORBには、以下の動作環境ファイルがあります。

動作環境ファイル	ファイル名
環境定義ファイル	config
オブジェクトリファレンス情報格納ファイル	initial_services
オブジェクトリファレンス検索情報ファイル	initial_hosts

上記の動作環境ファイルは、Webサーバのドキュメントルート配下に格納してください。

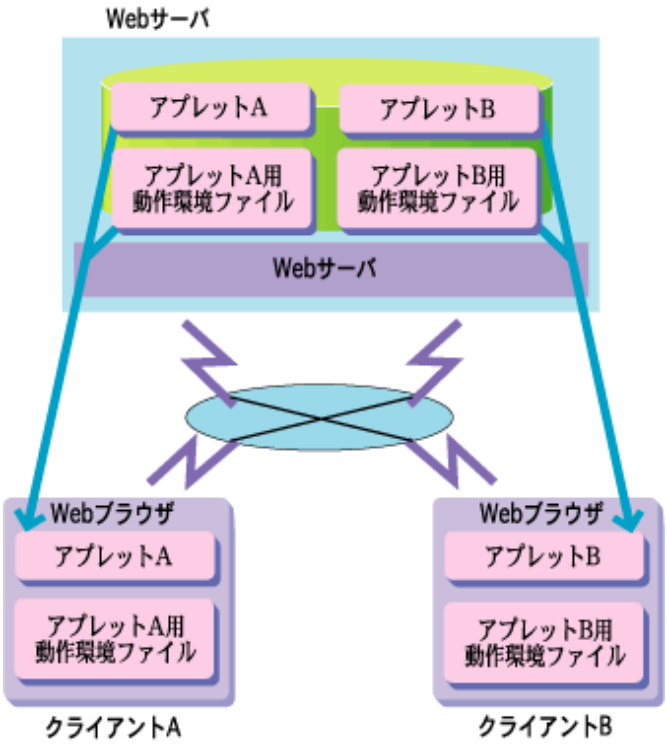
注意

Webサーバでユーザ名とパスワードによる認証を行う場合、ユーザ認証ディレクトリ配下に動作環境ファイルを格納しないでください。

動作環境ファイルとして、アプレットごとに異なる動作環境ファイルを指定する方法と、複数のアプレットで同一の動作環境ファイルを指定する方法があります。以下に、それぞれの場合における動作環境ファイルの使用イメージと指定方法を示します。

アプレットごとに異なる動作環境を使用する場合

動作環境ファイルの使用イメージは以下のとおりです。



クライアントAで動作するアプレットAは、アプレットA用の動作環境ファイルを、クライアントBで動作するアプレットBは、アプレットB用の動作環境ファイルを使用します。
アプレット単位に動作環境ファイルを指定する方法はPORB_HOMEを指定する方法と、PORB_HOMEを指定しない2種類の方法があります。

PORB_HOMEを指定する場合

PORB_HOMEを指定する場合、各アプレットの動作環境ファイルはetcディレクトリを作成してその配下に格納します。動作環境ファイルの格納位置として、HTMLファイルの<PARAM>タグで、PORB_HOMEパラメタにWebサーバのドキュメントルートからの相対パスを指定します。この指定では、etcディレクトリは含めません。
以下にディレクトリ構成例を示します。Webサーバのドキュメントルートディレクトリが/WWWで、アプレットAの動作環境ファイルはaplAenv/etc配下、アプレットBの動作環境ファイルはaplBenv/etc配下に格納されています。

```
/-  ・・・
 - [WWW] - [envfile] - [aplAenv] - [etc] - アプレットA用動作環境ファイル
                     - [aplBenv] - [etc] - アプレットB用動作環境ファイル
         - [applet ] - [appletA] - アプレットA.html
                                 - アプレットA.class
                       [appletB] - アプレットB.html
                                 - アプレットB.class
```

上記の例で、それぞれのアプレットのPORB_HOMEパラメタは<PARAM>タグで以下のように指定します。

アプレットAの<PARAM>タグ

```
<HTML>
<HEAD><!--demo.html--></HEAD>
<TITLE>Java sample Applet </TITLE>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットA.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=envfile/aplAenv>
</applet><BR>
```

```
</BODY>
</HTML>
```

アプレットBの<PARAM>タグ

```
<HTML>
<HEAD><!--demo.html--></HEAD>
<TITLE>Java sample Applet </TITLE>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットB.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=envfile/aplBenv>
</applet><BR>
</BODY>
</HTML>
```

PORB_HOMEを指定しない場合

PORB_HOMEを指定しない場合、各アプレットを格納したディレクトリにetcディレクトリを作成し、動作環境ファイルを格納します。
以下にディレクトリ構成例を示します。

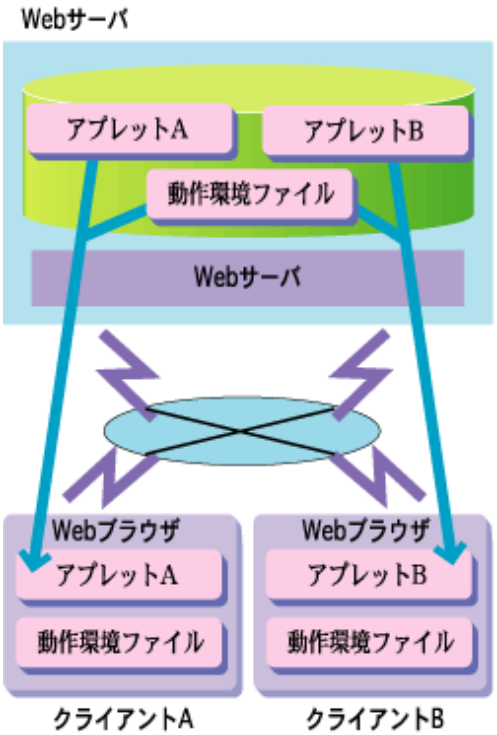
```
/- ・・・
 - [WWW] - [applet  ] - [appletA] - アプレットA.html
                                  - アプレットA.class
                                  - [etc] - アプレットA用動作環境ファイル
                       [appletB] - アプレットB.html
                                  - アプレットB.class
                                  - [etc] - アプレットB用動作環境ファイル
```

参考

fileプロトコルを使用して動作させる場合は、“PORB_HOMEを指定しない場合”の方法で動作させます。ブラウザ、およびJDKのツールであるappletviewerを使用し、Webサーバを使用せずローカルディスクに格納されたHTMLファイルを直接指定した場合は、fileプロトコルとなります。

複数のアプレットで同一の動作環境を使用する場合

動作環境ファイルの使用イメージは以下のとおりです。



クライアントAで動作するアプレットA、クライアントBで動作するアプレットBで共通の動作環境ファイルを使用します。
動作環境ファイルの格納位置を指定するには、アプレットAのHTMLファイルおよびアプレットBのHTMLファイルの<PARAM>タグで、PORB_HOMEパラメタにWebサーバのドキュメントルートからの相対パスを設定します。この指定では、etcディレクトリは含めません。
以下にディレクトリの構成例を示します。/WWW(Webサーバのドキュメントルートディレクトリ)配下のporbディレクトリにPortable-ORBがインストールされ、その配下のetcディレクトリに動作環境ファイルが格納されています。

```
/- ・・・
 - [WWW] - [porb   ] - [lib] - ODporb4.jar...
                     - [etc] - 動作環境ファイル
         - [applet ] - [appletA] - アプレットA.html
                                 - アプレットA.class
                       [appletB] - アプレットB.html
                                 - アプレットB.class
```

アプレットAのHTMLファイル、およびアプレットBのHTMLファイルに/WWW/porb/etc配下の動作環境ファイルを使用するように/WWWから/WWW/porbまでのパスを指定します。
それぞれの<PARAM>タグは以下のとおりです。

アプレットAの<PARAM>記述

```
<HTML>
<HEAD><!--demo.html--></HEAD>
<TITLE>Java sample Applet </TITLE>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットA.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=porb>
</applet><BR>
</BODY>
</HTML>
```

アプレットBの<PARAM>記述

```
<HTML>
<HEAD><!--demo.html--></HEAD>
<TITLE>Java sample Applet </TITLE>
<BODY>
<H1>Java sample Applet</H1>
<applet code="アプレットB.class" width=300 height=250>
<PARAM NAME=PORB_HOME VALUE=porb>
</applet><BR>
</BODY>
</HTML>
```

参考

- 動作環境ファイルは、環境設定コマンド(porbeditenv)を使用して作成および更新を行います。
- 動作環境ファイルを格納するetcディレクトリを作成する場合は、必ず英小文字で作成してください。

JBKプラグインの設定ファイルの編集

JBKプラグインに付属されているjbkplugin.propertiesファイルに、JavaVMの起動オプションとして以下のクラスパスを指定してください。

- JDKを使用している場合
  %JAVA_HOME%¥jre¥lib¥isejb.jar
- JREを使用している場合
  %JAVA_HOME%¥lib¥isejb.jar

例

設定例

```
jbk.plugin.vmoption=-classpath C:¥Interstage¥JBKDI¥jre8¥lib¥isjeb.jar
-Dorg.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
-Djavax.rmi.CORBA.StubClass=com.fujitsu.ObjectDirector.rmi.CORBA.StubDelegateImpl
-Djavax.rmi.CORBA.UtilClass=com.fujitsu.ObjectDirector.rmi.CORBA.UtilDelegateImpl
-Djavax.rmi.CORBA.PortableRemoteObjectClass=com.fujitsu.ObjectDirector.rmi.CORBA.
PortableRemoteObjectDelegateImpl
```

15.8 アプレットのデジタル署名

Javaアプレットをダウンロードして運用する場合、アプレット本体またはPortable-ORBに対してデジタル署名を行ってください。CORBAクライアントとしてのJavaアプレットはWebサーバからダウンロードされて、CORBAのサーバアプリケーションが動作するリモート上のマシンにネットワークを介してアクセスします。したがって、プレインストール型Javaライブラリ、Portable-ORBのいずれを使用する際も、Javaアプレットをダウンロードして運用する場合には、デジタル署名が必要です。

デジタル署名が必要なjarファイルを以下に示します。

- アプレットのjarファイル
- クライアント配布物のjarファイル
- Portable-ORB用のjarファイル
- EJBサービスのクライアント用jarファイル

権限の設定については、JBKプラグインが使用するポリシーファイルを使用することもできます。JBKプラグインが使用するポリシーファイルについての詳細は、Interstage Studioの“J Business Kit オンラインマニュアル”を参照してください。

15.8.1 デジタル署名

J Business Kitのプラグイン（以降JBKプラグイン）の使用したデジタル署名手順を説明します。署名ツールとしてkeytool/jarsigner/policytool（JDKのツール）を使用します。

注意

Portable-ORBをダウンロードしない運用で使用する場合、またはプレインストール型Javaクライアントを使用する場合は、それぞれの環境のJavaライブラリに対しても、policytoolによる権限の設定をしてください。Javaライブラリに対する権限の設定については、“Javaライブラリに対する権限の設定”を参照してください。

参照

署名ツールの詳細については、JDKのドキュメントを参照してください。J Business Kitをご使用の場合は、Interstage Studioの“J Business Kit オンラインマニュアル”を参照してください。

デジタル署名は、以下の手順で行います。

(1)鍵のペア（証明書）の作成

証明書に付加する情報、証明書にアクセスするための別名を指定して鍵のペア（証明書）を作成します。以下の実行例では別名samplesigner、証明書有効期限を365日としています。

```
keytool  -genkey -alias samplesigner -dname
  "cn=samplesigner, ou=JAVA PROJECT, o=FUJITSU, c=JA" -validity 365
  -genkey     :鍵のペア（証明書）の生成を指定
  -alias      :作成する鍵のペア（証明書）にアクセスするために別名を指定
  -dname      :署名者名、組織名、会社名、国名などを指定
  -validity   :証明書の有効期限を指定
```

コマンド入力時、キーストアのパスワード入力、作成する鍵のペア（証明書）のパスワード入力が求められます、これらのパスワードは以降のキーストアへのアクセス、および作成した鍵のペア（証明書）へのアクセスで必要です。
キーストアは鍵のペア（証明書）情報等を管理するデータベースであり、JDK(JRE)インストール時には存在せず、keytoolコマンドの初回実施時に作成されます。
ここで作成された証明書と各クライアントマシンでインポートする証明書の作成時刻は、同一である必要があります。-dnameオプションに指定した内容が同一でも異なる時刻に作成された証明書は、別の証明書として認識されるため注意してください。

(2)jarアーカイブファイルへの適用

作成した証明書を使用して、jarアーカイブファイルへデジタル署名を適用します。以下の実行例では(1)で作成した証明書をSample.jarに署名します。

```
jarsigner  -signedjar Sample.jar.sig  Sample.jar  samplesigner
  -signedjar    :署名を適用したjarファイルの名前を指定
                 （省略時は署名元jarファイルと同一名）
  Sample.jar    :証明元のjarファイルを指定
  samplesigner  :署名を行う証明書の別名を指定
```

コマンド入力時、キーストアのパスワード入力、鍵のペア（証明書）のパスワード入力が求められます。(1)で指定したパスワードを指定します。
jarアーカイブファイルの作成方法はについては、“Javaクラスファイルのアーカイブ”（インストール型Javaライブラリ）／“Javaクラスファイルのアーカイブ”（Portable-ORB）を参照してください。
同一の署名者で複数のjarアーカイブファイルを作成する場合は、(2)の処理を繰り返してください。

(3)jarアーカイブファイルへの署名適用の確認

署名を適用したjarアーカイブファイルに正常に署名が実施されていることを確認する場合は以下のコマンドを使用します。

```
jarsigner  -verify  Sample.jar.sig
  -verify       :jarアーカイブファイルのデジタル署名の検証を指定
  Sample.jar.sig :デジタル署名の検証を行うjarアーカイブファイルを指定
```

正常にデジタル署名が実施されている場合は「jar verified」、また正常にデジタル署名が実施されていない場合は「jar is unsigned.(signatures missing or not parsable)」というメッセージが表示されます。
デジタル署名が正常に実施されていることを確認した後、jarファイルとして使用するために、拡張子を*.jarに変更してください（例：署名を施す前のSample.jarを削除し、Sample.jar.sigをSample.jarに変名します）。
詳細なデジタル署名情報を表示する場合は、-verbose/-certsオプションを指定してコマンドを実行してください。

(4)証明書のエクスポート

アプレットをダウンロードするクライアントマシンにインポートするための証明書のエクスポートを行います。(1)鍵のペア（証明書）の作成で指定した別名を指定します。

```
keytool -export -alias samplesigner -file samplesign.cer
  -export     :証明書の取り出しを指定
  -alias      :取り出しを行う証明書の別名を指定
  -file       :取り出した証明書を格納するファイル名を指定
```

コマンド入力時、キーストアのパスワード入力が求められます。(1)で指定したパスワードを指定します。

(5)証明書のインポート

アプレットをダウンロードするクライアントマシンに証明書をインポートします。この作業はクライアントマシンで行います。証明書samplesign.cerを事前にクライアントマシンにコピーしてください。

```
keytool -import -alias sampleuser -file samplesign.cer
  -import     :証明書のインポートを指定
  -alias      :インポートする証明書の別名を指定
  -file       :インポートする証明書のファイル名を指定
```

コマンド入力時、キーストアのパスワード入力が求められます。これらのパスワードは以降のキーストアへのアクセスで必要です。
また、インポートする証明書を信頼するかどうかの入力が求められるので「yes」と入力します。
コマンド入力時に指定した別名(-aliasで指定した別名)は、以降の操作で、証明書(-fileで指定した証明書)を指定するために必要です。“(6)証明書への権限の設定”で権限を設定する際に、権限を設定する証明書の別名を指定してください。

(6)証明書への権限の設定

クライアントマシンにインポートした証明書に対して、権限を設定します。権限の設定はpolicytoolコマンドを使用して行います。この作業はクライアントマシンで行います。
policytoolコマンドは、JDK/JRE1.4系以降に付属しているセキュリティポリシを定義するGUIツールです。
このコマンド使用して署名されているJavaクラス、および任意の場所に格納されているJavaクラスの権限の設定または変更を行います。policytoolコマンドの設定については、“15.8.2 policytoolコマンドの設定”を参照してください。

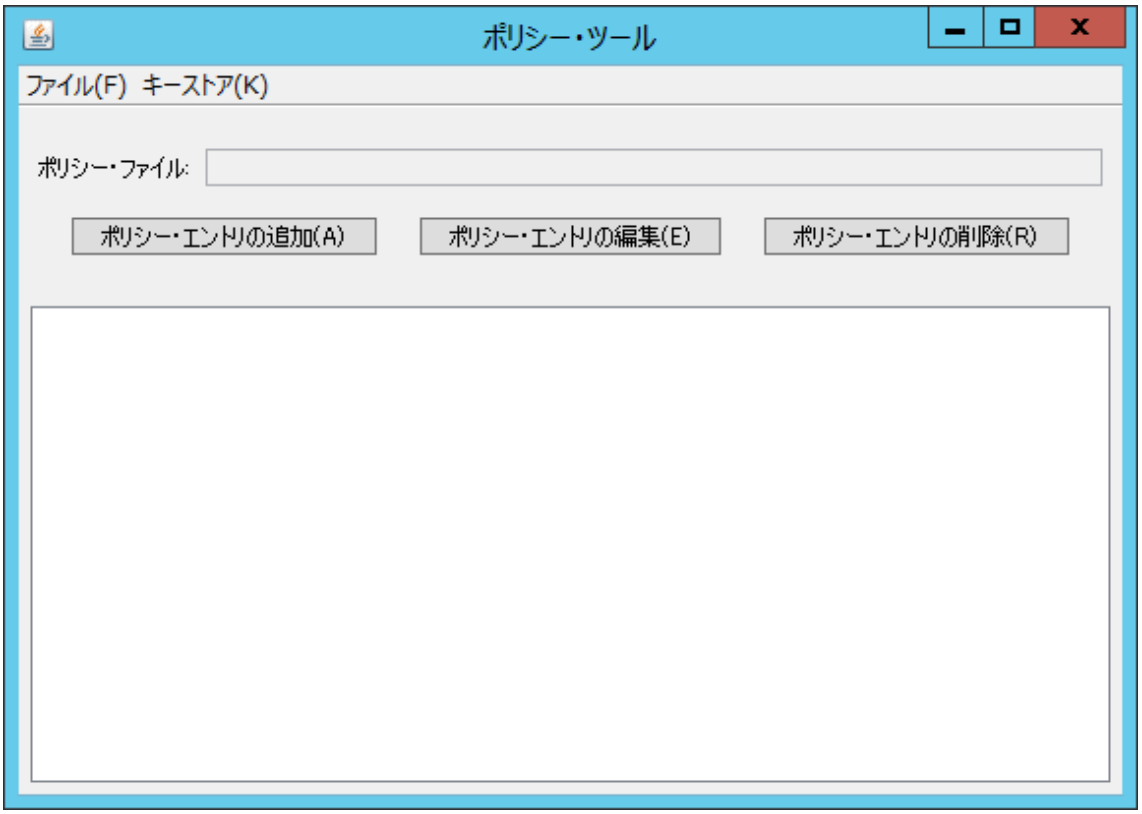
15.8.2 policytoolコマンドの設定

policytoolコマンドの具体的な設定方法について、以下の環境を例に説明します。

使用ライブラリ	プレインストール型Javaライブラリ
オペレーティングシステム	Microsoft(R) Windows Server(R) 2012
Javaのバージョン	JDK/JRE 8
CORBAクライアントのインストールディレクトリ	C:¥Interstage¥ODWIN
ユーザ名	guest
インポートした証明書の別名	sampleuser

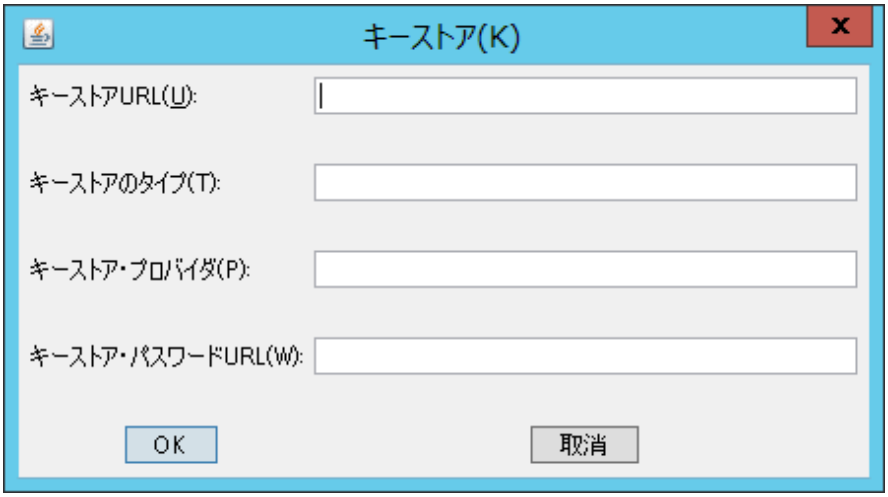
(1)policytoolコマンドの起動

policytoolコマンドを起動します。
表示される「Policy Tool」画面を以下に示します。



(2)キーストア（証明書データベース）の格納位置の指定

キーストアファイルの格納位置、およびキーストアのタイプを指定します。
policytool起動後最初に表示される「Policy Tool」画面内で、プルダウンメニュー「編集」→「キーストアの変更」を選択すると、以下のような「キーストア」画面が表示されます。



「新規キーストアのURL」フィールドへ使用するキーストアの格納位置を指定し、「新規キーストアのタイプ」フィールドにはキーストアタイプを指定します。それぞれのフィールドへ指定する内容を以下に示します。
設定後は、「キーストア」画面上の「了解」ボタンを押下してください。

[キーストアのURL]フィールド

[キーストアURL]フィールドに、“(5)証明書のインポート”で作成されたキーストアの格納位置をキーストアファイル名も含めてURL形式で指定します。
デフォルトの場合の格納先、キーストアおよびポリシーファイルのファイル名を以下に示します。

	格納先	キーストアファイル	ポリシーファイル
Windows32/64	ログインユーザのプロファイルディレクトリ 例 「Microsoft(R) Windows Server(R) 2012」およびログイン名「guest」の場合 **キーストア格納位置** file:c:/Users/guest/.keystore **ポリシーファイル格納位置** c:¥Users¥guest¥.java.policy	.keystore	.java.policy
Solaris64 Linux32/64	ログインユーザのホームディレクトリ		

ポイント

キーストアの格納位置には、URL上のキーストアを指定することも可能です。URL上のキーストアを指定する場合は、“(4)証明書のエクスポート”、“(5)証明書のインポート”の手順は不要となります。

[キーストアタイプ:]フィールド

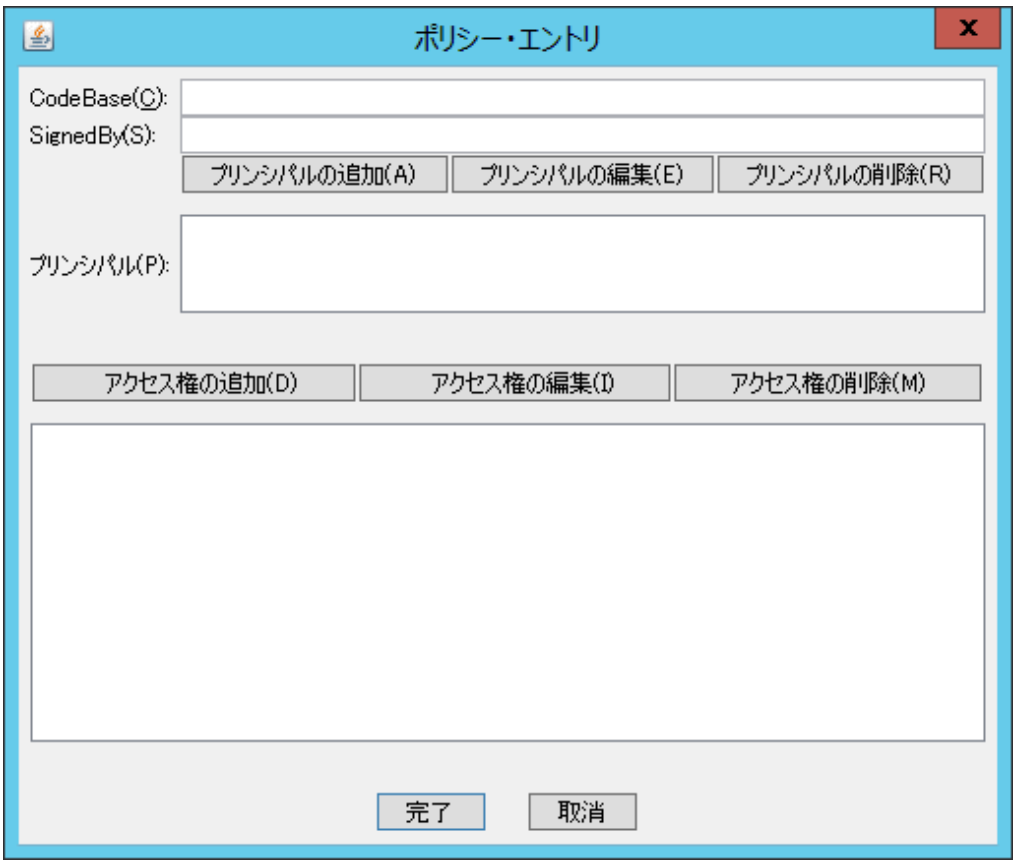
[キーストアタイプ]フィールドに、「jks」を指定します。

設定画面の例を以下に示します。

(3)エントリの作成

[ポリシー・ツール]画面の[ポリシー・エントリの追加]ボタンをクリックすると、以下の[ポリシー・エントリ]画面が表示されます。

本画面で各権限を設定します。



(4)権限の設定

「ポリシーエントリ」画面上の「アクセス権の追加」ボタンを押下し、「アクセス権」画面を表示します。この画面上で権限を設定します。

1組の[アクセス権:]、[ターゲット名:]、および[アクション:]の値を指定し、[OK]ボタンをクリックして権限を設定します。[OK]ボタンをクリックすると、[ポリシー・エントリ]画面に戻ります。次の1組の値を設定する場合は、[アクセス権の追加]ボタンを再度クリックして、本操作を繰り返し、必要な情報を設定します。

なお、設定する権限は、「プレインストール型Javaライブラリを使用する場合」と「Portable-ORBを使用する場合」で異なります。

設定が必要な権限を以下に示します。

プレインストール型Javaライブラリを使用する場合

◇通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、この権限以外は設定しないでください。

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	loadLibrary.<ライブラリ名> (注1)	設定不要
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read

注1）インストールしている機能により以下のダイナミックリンクライブラリ(DLL)名を指定します。なお、拡張子を指定する必要はありません。

インストールしている機能	ダイナミックリンクライブラリ
CORBAサービスクライアント(クライアント機能)	ODjava4
CORBAサービス(サーバ機能)	ODjavas4

◇CORBAサービスの内部ログを採取する場合に必要な権限 (注2)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	user.dir	read
		java.class.path	read
ファイル権限	FilePermission (java.io.FilePermission)	${user.dir}¥*	read, write
		%OD_HOME%¥etc¥config (注3)	read

注2) CORBAサービスの内部ログの詳細は、“チューニングガイド”の“config”を参照してください。なお、ログ採取後は追加した権限を削除してください。
注3) %OD_HOME%は、CORBAサービス、CORBAサービスクライアントのインストールパスを指定します(デフォルトは、C:¥Interstage¥ODWIN)。

Portable-ORBを使用する場合

◇通常運用で必要な権限

通常の運用では、セキュリティ上の安全性を確保するため、この権限以外は設定しないでください。

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
通信権限	SocketPermission (java.net.SocketPermission)	通信先サーバ名 (注1)	connect

注1) Javaアプレットをダウンロードした Webサーバ以外のサーバマシンと通信する場合に、通信先サーバ分の通信先サーバ名を設定します。通信するサーバマシンがJavaアプレットをダウンロードしたWebサーバだけの場合、設定する必要はありません。
通信先サーバ名として、以下のホスト名を指定します。

- porbeditenvコマンドの「ホスト情報」で設定したホスト名
  詳細については、“リファレンスマニュアル(コマンド編)”の“porbeditenv”を参照してください。
- 通信するサーバアプリケーションのオブジェクトリファレンスに設定されているホスト名
  odlistnsコマンド(-lオプション指定)で表示される「オブジェクトのホスト名」です。

◇EJBアプリケーションを使用する場合に必要な権限 (注2)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	com.fujitsu.*	read
		java.class.path	read
ランタイム権限	RuntimePermission (java.lang.RuntimePermission)	getClassLoader	(指定不要)

注) EJBアプリケーションの詳細については、“第3部 EJB編”を参照してください。

◇Portable-ORBのログ情報を採取する場合に必要な権限 (注3)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ファイル権限	FilePermission (java.io.FilePermission)	ログ採取ファイル (注4)	read, write, delete
		ログ採取ディレクトリ (注5)	read

注3) Portable-ORBのログ情報の詳細については、“リファレンスマニュアル(コマンド編)”の“porbeditenv”を参照してください。なお、ログを採取する場合は、事前にporbeditenvコマンドの「ログ格納ディレクトリ」で指定したディレクトリを作成しておく必要があります。また、「ログ格納ディレクトリ」にはログ採取ファイル以外のユーザ資源などを格納しないでください。
ログ採取後は、追加した権限を削除してください。
注4) porbeditenvコマンドの[ログ格納ディレクトリ]で指定したディレクトリ名に「¥*」を付加したパスを指定します。[ログ格納ディレクトリ]に「c:¥log¥porb」と指定した場合は、「c:¥log¥porb¥*」です。
注5) porbeditenvコマンドの[ログ格納ディレクトリ]で指定したディレクトリ名を指定します。

◇SSL連携機能を使用する場合に必要な権限 (注6)

権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
ファイル権限	FilePermission (java.io.FilePermission)	keystoreディレクトリ (注7)	read

注6) SSL連携機能の詳細については、“セキュリティシステム運用ガイド”の“Portable-ORBでSSLを利用する方法”を参照してください。
注7) porbeditenvコマンドの[キーストア格納位置]で指定した格納位置、または-ORB_FJ_PORB_SSLPathパラメタで指定した格納位置を指定します。例えば、格納位置が「C:¥Interstage¥PORB¥etc¥keystore」の場合は、「C:¥Interstage¥PORB¥etc¥keystore」です。キーストアの格納位置がネットワーク上(HTTP指定)である場合、指定する必要はありません。

◇サーバアプリケーションでユーザ情報獲得用のAPIを使用する場合に必要な権限 (注8)

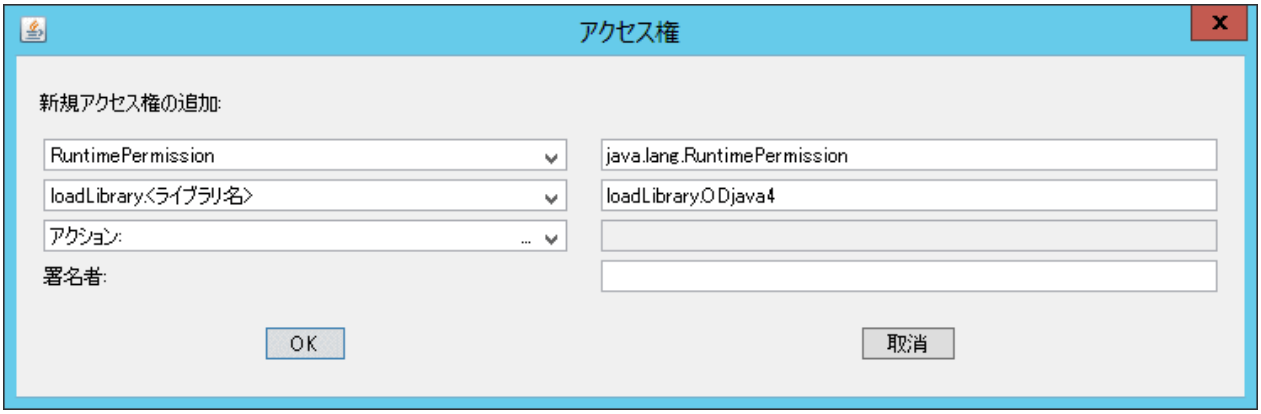
権限種別	設定する権限		
	アクセス権	ターゲット名	アクション
プロパティ権限	PropertyPermission (java.util.PropertyPermission)	user.name	read

注8）ユーザ情報獲得用のAPIの詳細については、“リファレンスマニュアル（API編）”の“TD_get_user_information”、“TD::get_user_information”、“TDGETUSERINFORMATION”を参照してください。

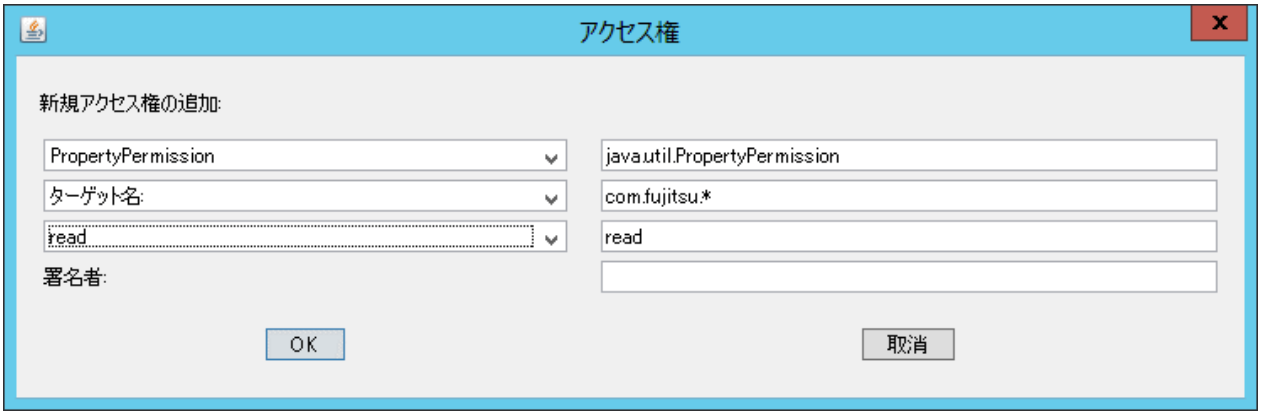
例

「アクセス権」画面の設定例

ランタイム権限の設定画面の例を以下に示します。



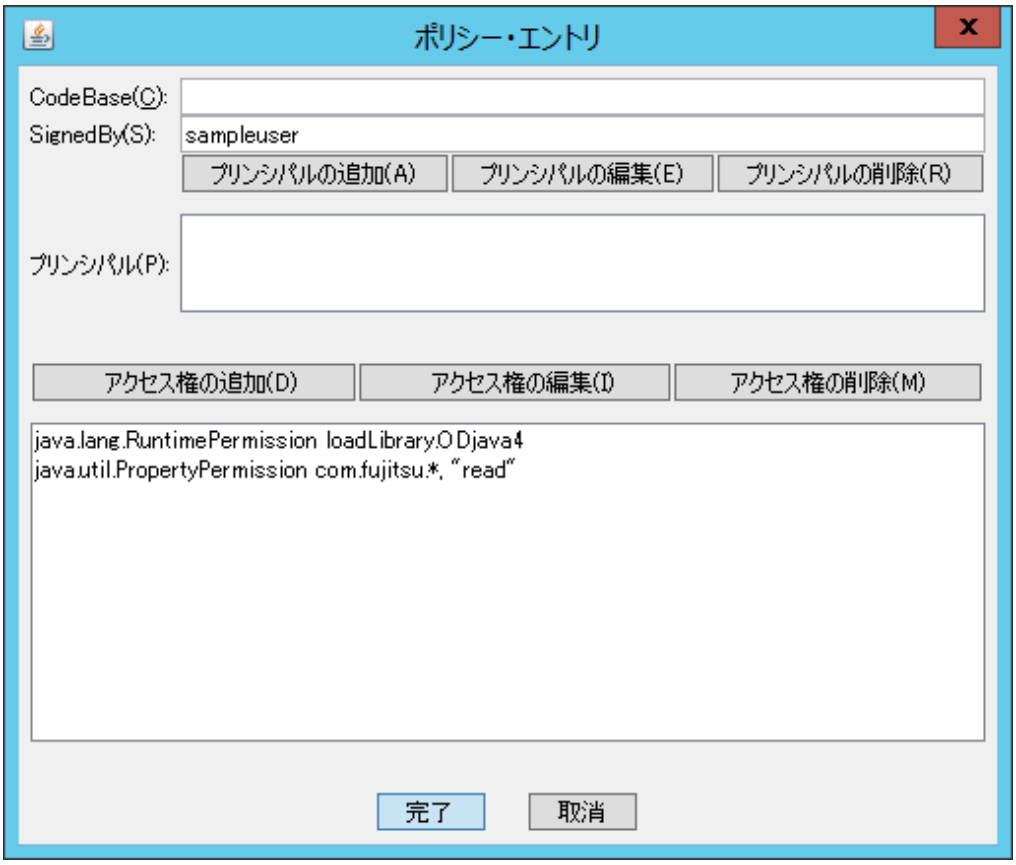
プロパティ権限の設定画面の例を以下に示します。



(5)署名付きJavaクラスのための証明書の指定

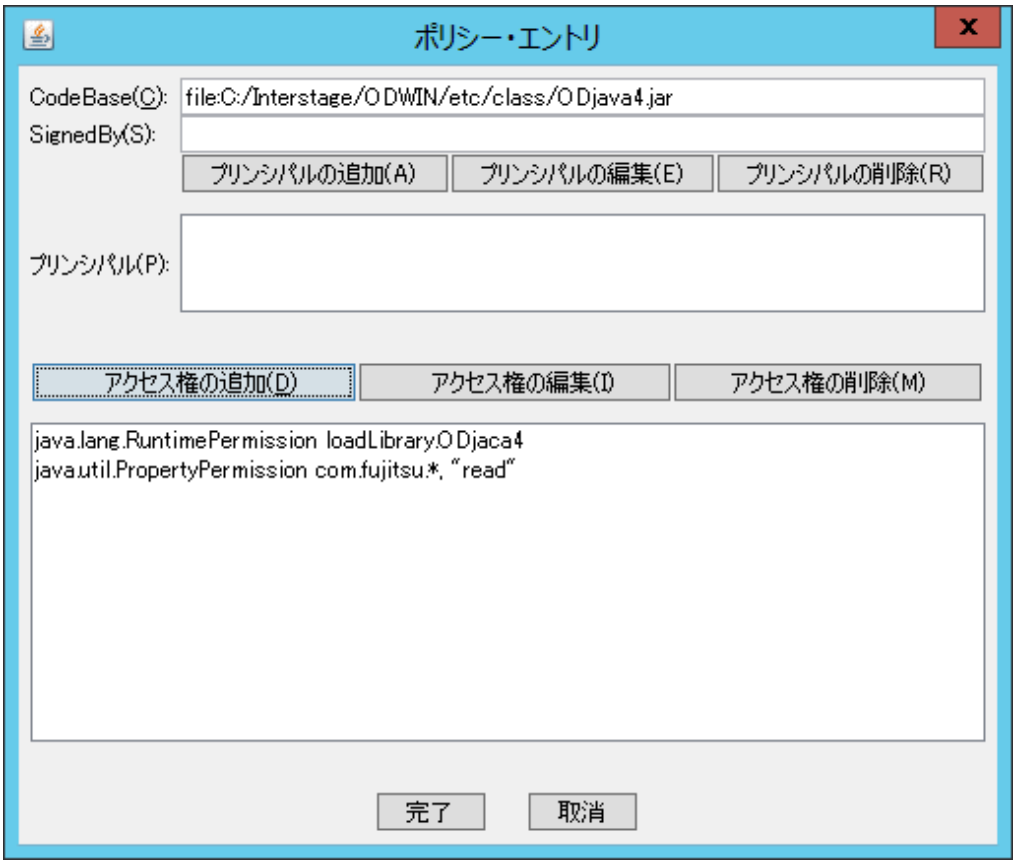
どの証明書に対する権限かを指定します。

[SignedBy:]フィールドに、キーストアにインポートした名前（別名）を指定し、[完了]ボタンをクリックします。



(6)プレインストール型Javaライブラリの権限の設定

プレインストール型Javaライブラリの権限を設定します。
[ポリシー・ツール]画面の[ポリシー・エントリの追加]ボタンをクリックし、ランタイム権限、プロパティ権限、およびファイル権限をそれぞれ設定します(“(3)エントリの作成”、“(4)権限の設定”)。
各権限の設定後、[ポリシーエントリ]画面の[CodeBase:]フィールドにプレインストール型Javaライブラリを指定し、[完了]ボタンをクリックします。



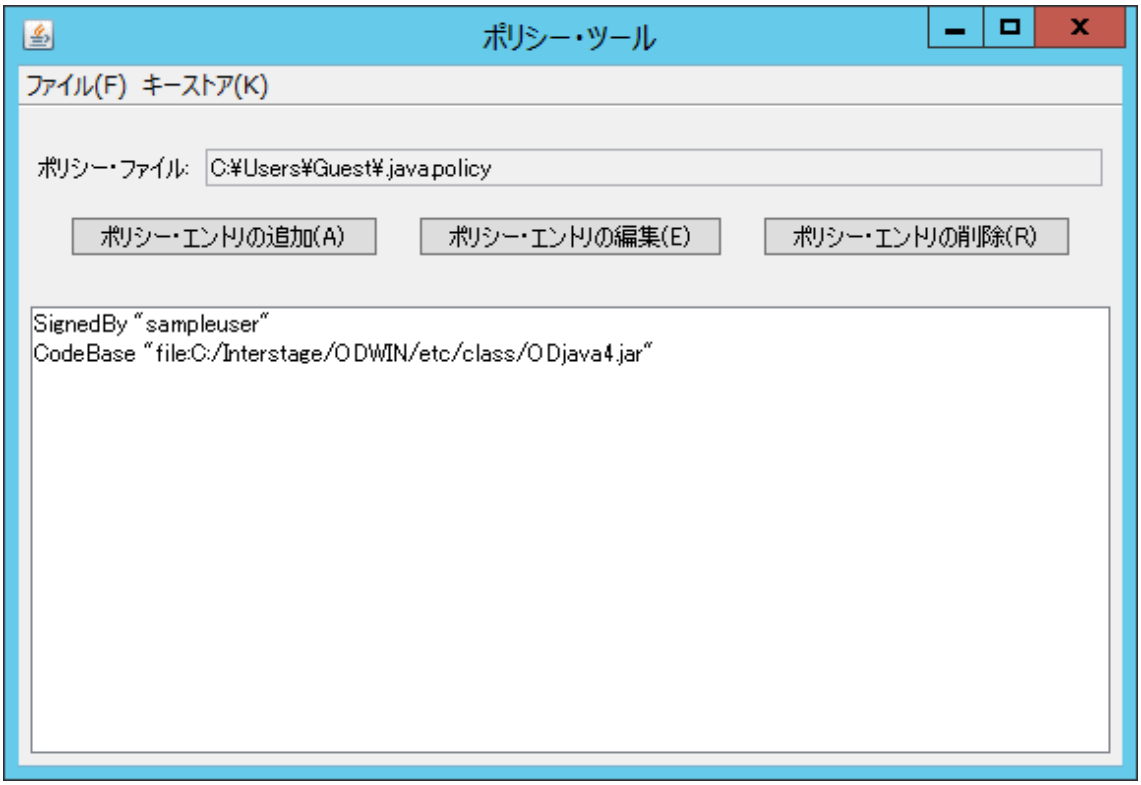
(7)保存

設定した権限をセキュリティポリシーファイルとして保存します。
初回作成時は、プルダウンメニューの[ファイル]の[別名保存]をクリックし、ポリシーファイルの名前および格納位置を指定します。2回目以降は、[ポリシー・ツール]画面の[ポリシーファイル:]フィールドで指定します。
デフォルトで指定するディレクトリは使用するオペレーティングシステムにより異なります。格納ディレクトリについては、[キーストアのURL]フィールドの表を参照してください。

プルダウンメニューの[ファイル]の[保存]をクリックすると、ポリシーファイルから以下のダイアログメッセージが表示されます。

(8)policytoolコマンドの終了

[ポリシーファイル:]フィールドに保存されたポリシーファイルが表示されていることを確認し、[ポリシー・ツール]画面で、プルダウンメニューの[ファイル]の[終了]をクリックします。



15.9 Java以外の言語からの呼出し方法

本節では、Java以外の言語で実装するCORBAクライアントアプリケーションからEJBアプリケーションを利用するための方法について説明します。

本節の機能は、以下の製品で利用できます。

- Interstage Application Server Enterprise Edition

Java以外の言語で実装するCORBAクライアントアプリケーションからEJBアプリケーションを呼び出すためには、その呼出しを仲介するアプリケーションを作成してください。
このアプリケーションを、“**EJBゲートウェイ・アプリケーション**”と呼びます。

EJBゲートウェイ・アプリケーションは、CORBAクライアントアプリケーションとEJBアプリケーションの間に位置し、IIOPおよびRMIの変換を行うアプリケーションです。EJBゲートウェイ・アプリケーションは、Javaにより実装するCORBAサーバアプリケーションで、EJBクライアントアプリケーションです。

EJBゲートウェイ・アプリケーション機能構成

EJBゲートウェイ・アプリケーションは以下に示す機能から構成されます。

CORBAサーバ機能

クライアントに対する分散オブジェクトとしてのインタフェース機能を実現するモジュールです。

EJB-CORBA変換機能

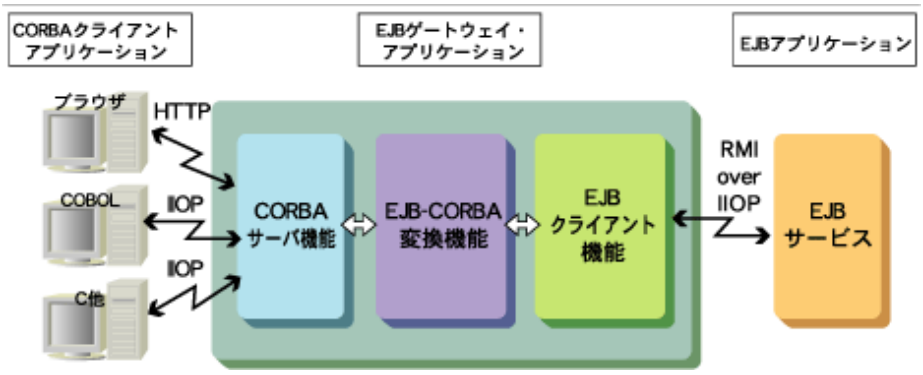
EJBの例外からCORBAの例外への変換や、CORBAオブジェクトおよびJavaオブジェクトへの相互変換の機能を実現するモジュールです。

EJBクライアント機能

CORBAクライアントからのリクエストに応じて、EJBアプリケーションの生成と削除や、EJBのビジネスメソッドの呼出しを行う機能を実現するモジュールです。

各機能の詳細については、“15.9.1 EJBゲートウェイ・アプリケーションの機能”を参照してください。

以下に、EJBゲートウェイ・アプリケーションの機能構成のイメージを示します。



15.9.1 EJBゲートウェイ・アプリケーションの機能

以下について説明します。

- CORBAサーバ機能
- EJB-CORBA変換機能
- EJBクライアント機能

CORBAサーバ機能

CORBAサーバ機能は、IDL言語を用いて定義する分散オブジェクトのインタフェース機能です。
IDL定義では、クライアントに公開するビジネスメソッドを定義します。EJBアプリケーションのビジネスメソッドの引数や復帰値がJavaオブジェクトの場合には、CORBAクライアントと受渡しができるデータ型も定義します。
また、必要であれば例外も定義します。EJBの生成と削除をクライアントアプリケーションで制御する場合には、EJBの生成/削除メソッドを定義してください。

EJB-CORBA変換機能

EJB-CORBA変換機能は、EJB例外からCORBA例外への変換（例外変換）や、CORBAデータ型とJavaデータ型の相互変換（オブジェクト変換）の機能を実現する機能です。

例外変換

EJBで送出した例外は、CORBAクライアントアプリケーションではCORBAにおけるUNKNOWN例外として通知されます。このため、CORBAクライアントアプリケーションに例外情報を通知するには、EJBゲートウェイ・アプリケーションでEJBが送出した例外を、CORBAの例外に変換して送出してください。

EJBビジネスメソッドでの送出例外宣言

ビジネスメソッド定義の例を以下に示します。メソッド定義のthrows句で記述された例外を送出します。

```
public interface SampleBMPSessionRemote extends javax.ejb.EJBObject {
    public int MultipleAddRecords() throws java.rmi.RemoteException;
}
```

IDLビジネスメソッドでの送出例外宣言

ビジネスメソッド定義の例を以下に示します。メソッド定義のraises句で記述された例外を送出します。

```
interface SampleBMPSession {
    long MultipleAddRecords() raises (ejbException);
};
```

IDL例外オブジェクト定義

IDLでの例外オブジェクト定義の例を以下に示します。exceptionで例外オブジェクトを定義します。

```
exception ejbException {
    string reason;
};
```

例外変換処理

例外の変換処理の例を以下に示します。EJBから送出された例外を受け取り、IDLで定義した例外ejbExceptionを送出します。

```
public int MultipleAddRecords() throws EJBGateway.ejbException {
    int retval = 0;
    try {
        retval = r.MultipleAddRecords();
    } catch (Throwable e) {
        throw new ejbException(e.getMessage());
    }
    return retval;
}
```

上記の例では、Javaのすべてのエラーと例外のスーパークラスであるThrowable型を用いています。アプリケーションの要件に応じて適切な例外型によるハンドラを記述してください。

オブジェクト変換

EJBのメソッドの復帰値は、Javaにおけるオブジェクトです。CORBAクライアントアプリケーションでは、このJavaオブジェクトを受け取ることができません。このため、CORBAクライアントアプリケーションに復帰値を返すためには、EJBゲートウェイ・アプリケーションでJavaオブジェクトを受け取り、CORBAのオブジェクトに変換して返却してください。
EJBのメソッドの引数にJavaオブジェクトが指定されていた場合には、EJBゲートウェイ・アプリケーションでCORBAオブジェクトを受け取り、Javaオブジェクトに変換して受け渡してください。以下に復帰値の変換の例を示します。

EJBビジネスメソッド定義

メソッドの復帰値の型がVectorであるメソッド定義の例を以下に示します。

```
public interface SampleBMPSessionRemote extends javax.ejb.EJBObject
{
        public Vector MultipleFindRecords() throws java.rmi.RemoteException;
}
```

IDLビジネスメソッド定義

Vectorに対して、IDLで定義するresultSet型として定義した例を以下に示します。

```
interface SampleBMPSession
{
        resultSet MultipleFindRecords() raises (ejbException);
};
```

IDLオブジェクト定義

IDLでのresultSet定義の例を以下に示します。ただし、EJBで返却されるVectorオブジェクトは、文字列の配列を要素とします。IDLでは2つの文字列をメンバとして持つ構造体とその構造体のシーケンスとして定義します。

```
struct result {
    string id;
    string name;
```

```
};
typedef sequence<result> resultSet;
```

変換処理

オブジェクトの変換処理の例を以下に示します。EJBから送出されたVectorオブジェクトを受け取り、IDLで定義したresultの配列を返却します。
IDLのJavaへのマッピングの詳細については、“Interstage Application Server アプリケーション作成ガイド (CORBAサービス編)”の“アプリケーションの開発(Java言語)”を参照してください(注:Interstage Application Server Standard-J Editionでは提供していません)。

```
public result[] Vector2ResultSet(int lines, Vector vecSearchResult) {
    result[] retval = null;
    retval = new result[lines];
    for ( int i = 0; i < lines; i++ ) {
        String[] vecData = (String[])vecSearchResult.elementAt(i);
        retval[i] = new result(vecData[0], vecData[1]);
    }
    return retval;
}
```

EJBクライアント機能

EJBのクライアントアプリケーション機能です。
クライアントアプリケーション機能は、HomeインタフェースとRemoteインタフェースに対するEJBクライアントの形で記述します。
クライアントアプリケーションの開発方法の詳細については、“第11章 EJBアプリケーションの開発”を参照してください。

15.9.2 環境設定

ここでは、EJBゲートウェイ・アプリケーションを利用するための環境設定について説明します。

EJBゲートウェイ・アプリケーションの実装は、JavaによるCORBAサーバアプリケーションであるとともにEJBクライアントアプリケーションです。したがって、EJBゲートウェイ・アプリケーションを利用するために以下の環境設定をしてください。

- JavaによるCORBAサーバアプリケーションの環境設定
- Interstageがインストールされているサーバマシン上でEJBクライアントアプリケーションを動作させるための環境設定

CORBAサーバアプリケーションの環境設定については“運用ガイド(基本編)”の“Interstageの環境設定”を参照してください。
EJBクライアントの環境設定については、“4.2 EJBを参照する場合の環境設定”を参照してください。

15.9.3 EJBゲートウェイ・アプリケーションの開発方法

ここでは、Javaを用いたCORBAサーバアプリケーションとしてのEJBゲートウェイ・アプリケーションの開発およびテスト方法について説明します。

- EJBゲートウェイ・アプリケーションの開発前に必要な作業
- 開発の流れ
- EJBゲートウェイ・アプリケーションの開発
- クライアントアプリケーションの開発
- EJBゲートウェイ・アプリケーションのデバッグ
- Interstage Studioでの開発方法

参照

EJBゲートウェイ・アプリケーションを開発する場合は、以下のマニュアルも参照してください。

CORBAサーバ機能およびEJB-CORBA変換機能を開発する場合

JavaによるCORBAサーバ機能の開発方法については、以下を参照してください。

－ “Interstage Application Server アプリケーション作成ガイド（CORBAサービス編）”の“アプリケーションの開発（Java言語）”（注：Interstage Application Server Standard-J Editionでは提供していません）。

－ “Interstage Studio ユーザーズガイド”

EJBクライアント機能を開発する場合

EJBクライアント機能の開発方法については、以下を参照してください。

－ “第11章 EJBアプリケーションの開発”

－ “Interstage Studio ユーザーズガイド”

クライアントアプリケーションを開発する場合

クライアントアプリケーションの開発方法については、以下を参照してください。

－ “Interstage Application Server アプリケーション作成ガイド（CORBAサービス編）”の“アプリケーションの開発（Java言語）”（注：Interstage Application Server Standard-J Editionでは提供していません）。

－ “Interstage Studio ユーザーズガイド”

EJBゲートウェイ・アプリケーションの開発前に必要な作業

EJBゲートウェイ・アプリケーションを開発する前に、連携方法、サーバの環境などを十分に分析し、アプリケーションの内容を考慮してください。

ここでは、アプリケーションの開発前に必要な作業について説明します。

アプリケーション形態を選択する

EJBゲートウェイ・アプリケーションは、Servantのインスタンスの管理方法により、以下の形態をとることができます。以下の形態からユーザの業務にあった形態を選択してください。

形態	STATEFUL Session Beanと連携する場合	STATELESS Session Beanと連携する場合
デフォルトインスタンス方式	×	○
Factory-1方式	○	○
Factory-2方式	○	○
ユーザインスタンス管理方式	○	○

○：選択可　×：選択不可

Session Beanとの対応形態を選択する

Session Beanに対するEJBゲートウェイ・アプリケーションの実装方法として、以下の形態があります。

－ 1つのEJBゲートウェイ・アプリケーションで1つのSession Beanに対応するインタフェースを実装する

－ 1つのEJBゲートウェイ・アプリケーションで複数のSession Beanに対応する複数のインタフェースを実装する

上記の方式からユーザの業務にあった形態を選択してください。

開発の流れ

EJBゲートウェイ・アプリケーションを開発し、デバッグを行うまでの流れを以下に示します。

1. EJBゲートウェイ・アプリケーションの開発
2. クライアントアプリケーションの開発
3. アプリケーションのデバッグ

デバッグにより問題が検出された場合は、問題が発生したアプリケーションの開発から繰り返します。

EJBゲートウェイ・アプリケーションの開発

ここでは、EJBゲートウェイ・アプリケーション開発方法について説明します。

1. **IDLファイルの作成とコンパイル**
   EJBゲートウェイ・アプリケーションが提供するインタフェースを、IDL言語を用いて定義し、IDLcコマンドでIDLファイルをコンパイルします。この結果、IDL定義をjavaにマッピングした、複数のファイル（拡張子が.javaのファイル）が出力されます。

2. **EJBゲートウェイ・アプリケーションの作成**
   EJBゲートウェイ・アプリケーションは「CORBAサーバ機能」と「EJB-CORBA変換機能」および「EJBクライアント機能」に分けられます。
   「CORBAサーバ機能」では、ORBの初期化や、Servantオブジェクトの生成/登録などを行う初期化処理を実装します。
   「EJB-CORBA変換機能」および「EJBクライアント機能」はIDLで定義したインタフェースを実装するServantとして実装します。
   各機能の詳細については、“15.9.1 EJBゲートウェイ・アプリケーションの機能”を参照してください。

3. **Javaファイルのコンパイル**
   IDLコンパイラにより生成されたファイル（拡張子が.javaのファイル）とユーザが作成したEJBゲートウェイ・アプリケーションを合わせてjavacコマンドでコンパイルします。
   コンパイル時には、CLASSPATH変数にEJBアプリケーションを配備したejb-jarファイルを指定してください。

クライアントアプリケーションの開発

ここでは、クライアントアプリケーション開発方法について説明します。

1. **IDLファイルのコンパイル**
   IDLcコマンドでIDLファイルをコンパイルします。この結果、IDL定義を目的とする言語にマッピングした、複数のスタブとスケルトンが出力されます。

2. **クライアントアプリケーションの作成**
   クライアントアプリケーションを作成します。

3. **翻訳と結合**
   作成したクライアントアプリケーションとIDLcコマンドで生成したスケルトン以外のソースファイルを翻訳し、実行ファイルを作成します。

EJBゲートウェイ・アプリケーションのデバッグ

EJBアプリケーションのデバッグ方法には、**デバッガを利用したデバッグ**と、**アプリケーションのデバッグ情報を利用したデバッグ**があります。それぞれについて説明します。

デバッガを利用したデバッグ

Interstage Studioが提供するデバッガを利用してデバッグする方法です。
デバッガを利用すると、開発したアプリケーションを実行させながら、処理の論理的な誤りを検出することができます。
通常、プログラムソース上にブレークポイントを設定し、中断点で停止した状態でプログラム中の変数を参照あるいは、変更しながらデバッグを行います。
デバッガを利用したデバッグの詳細は、“Interstage Studio ユーザーズガイド”を参照してください。

アプリケーションのデバッグ情報を利用したデバッグ

EJBゲートウェイ・アプリケーションの開発時に、あらかじめデバッグ情報を出力する処理を記述しておき、その情報をもとにデバッグする方法です。

デバッグ情報について

アプリケーションのデバッグ情報は、標準出力または、標準エラー出力を使用します。

デバッグ時のEJBゲートウェイ・アプリケーション実行方法

EJBゲートウェイ・アプリケーションの実行方法について説明します。

注意

EJBゲートウェイ・アプリケーションを実行する前に、EJBアプリケーションを実行してください。

1. Interstageの起動
   Interstageを起動します。

2. 起動用ファイルの作成
   EJBゲートウェイ・アプリケーションを起動するには、通常、バッチファイルやシェルスクリプトを用意します。
   起動時には、環境変数OD_IMPLIDへインプリメンテーションリポジトリIDを設定してください。また、EJBゲートウェイ・アプリケーションを起動する際に、以下の環境プロパティを指定し起動してください。

```
-Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
```

   例

   以下に起動ファイルの例を示します。

```
set OD_IMPLID=IDL:EJBGateway/Factory:1.0
java -Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
BMPSessionGateway
```

3. **インプリメンテーションリポジトリへの登録**
   インプリメンテーションリポジトリにゲートウェイ・アプリケーション情報を登録するために、OD_impl_instコマンドを使用します。OD_impl_instコマンドおよび指定する情報については、“Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)”の“インプリメンテーションリポジトリへの登録”を参照してください。(注:Interstage Application Server Standard-J Editionでは提供していません)。

4. **オブジェクトリファレンスの作成とネーミングサービスへの登録**
   作成したEJBゲートウェイ・アプリケーションのオブジェクトリファレンスの作成およびネーミングサービスに登録するために、OD_or_admコマンドを使用します。
   OD_or_admコマンドおよび指定する情報については、“Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)”の“オブジェクトリファレンスの生成方法”を参照してください。(注:Interstage Application Server Standard-J Editionでは提供していません)。

5. **EJBゲートウェイ・アプリケーションの起動**
   2.で作成した起動用ファイルによりEJBゲートウェイ・アプリケーションを起動します。

6. **クライアントアプリケーションの実行**
   クライアントアプリケーションを実行します。

Interstage Studioでの開発方法

当社のコンポーネント指向の統合開発支援ツールであるInterstage Studioを使用することにより、一連の手順を統合された使いやすいビュー操作によって操作することができます。
Interstage Studioでは、さまざまなEJBアプリケーションおよびクライアントアプリケーションの開発支援機能を提供しているため、アプリケーション開発の生産性を向上できます。

15.9.4 運用方法

ここでは、EJBゲートウェイ・アプリケーションを使用したEJBサービスの業務運用について説明します。

事前準備

EJBゲートウェイ・アプリケーションを使用したEJBサービスの業務運用を行う場合、事前に以下の作業を行ってください。

EJBアプリケーションの起動準備

EJBアプリケーションを運用可能な状態にします。
EJBアプリケーションの運用方法については、“第3章 J2EEアプリケーションの運用”を参照してください。

EJBゲートウェイ・アプリケーションの配置

EJBゲートウェイ・アプリケーションを以下のサービスがインストールされた環境に配置してください。

- CORBAサービス
- EJBサービス
- Java実行環境(Linuxの場合)

詳細は、“システム設計ガイド”の“ソフトウェア条件”を参照してください。
なお、EJBゲートウェイ・アプリケーションを構成する以下のクラスファイルも合わせて配置し、環境変数CLASSPATHに設定してください。

- EJBアプリケーションを配備した結果、生成されるクライアント配布物
- IDLcコマンドにより生成されるスケルトンのクラスファイル

運用の流れ

EJBゲートウェイ・アプリケーションを介したEJBサービスの業務を行う場合の、運用の流れを以下に示します。

1. **DBMSの起動**
   EJBアプリケーションがデータベースを使用する場合は、DBMSを起動してください。データベースの運用の詳細については、各DBMSのマニュアルを参照してください。
2. **Interstageの起動**
   Interstage管理コンソールを使用してInterstageを起動してください。Interstageの起動の詳細は“運用ガイド(基本編)”の“Interstageの起動”を参照してください。
3. **IJServerの起動**
   業務運用を開始するために、IJServerを起動してください。詳細は、Interstage管理コンソールのヘルプを参照してください。
4. **EJBゲートウェイ・アプリケーションの起動**
   起動用ファイルによりEJBゲートウェイ・アプリケーションを起動します。
   起動用ファイルは、「スタートメニュー」の「ファイル名を指定して実行」ダイアログで起動するか、コマンドプロンプト画面から起動します。

   注意

   「ファイル名を指定して実行」ダイアログで起動、またはコマンドプロンプト画面から起動する場合は、オペレーティングシステムにログインしてください。
5. **業務運用**
   クライアントからデータを入力し、業務を運用します。
6. **EJBゲートウェイ・アプリケーションの停止**
   EJBゲートウェイ・アプリケーションを停止してください。
7. **IJServerの停止**
   業務運用を停止するために、IJServerを停止してください。詳細は、Interstage管理コンソールのヘルプを参照してください。
8. **Interstageの停止**
   Interstageを停止してください。
9. **DBMSの停止**
   データベースを使用した場合は、DBMSを停止してください。

15.10 RMI over IIOPについて

EJBサービスでは、クライアント/サーバ間の通信形態に、RMI over IIOPを採用しています。
ここでは、RMI over IIOPの概要およびインタフェースに使用できるデータ型について説明します。

15.10.1 RMI over IIOPとは

RMI（Remote Method Invocation）はJava間でのリモート呼出しを実現するために規定された技術です。そのためJavaのアプリケーションにとって非常に使い勝手の良いものであり、アプリケーションをRMIで記述できることで、分散環境におけるアプリケーションの流通性を高めることができます。

IIOP（Internet Inter-ORB Protocol）はOMGが規定したCORBAのプロトコルであり、他社ORB（Object Request Broker）間での相互接続ができます。このIIOPを介すことによって異なった言語で記述されたアプリケーション間でのリモート呼出しが可能となりますが、JavaプログラマはCORBAの知識を習得する必要性がありました。

RMI over IIOPとは、RMIの通信層にIIOPを利用するものです。したがって、Javaプログラマから見るとRMIのAPIを使用できるため、CORBAの知識を隠ぺいすることができるとともに、RMIの特性を生かすこともできます。

15.10.2 インタフェースに使用できるデータ型

クライアント/サーバ間のインタフェースで使用できるデータ型について説明します。

- Home/Remoteインタフェースで定義できる型
- アプリケーション実行時に通信できる型

Home/Remoteインタフェースで定義できる型

Home/Remoteインタフェースで、メソッドの復帰値およびパラメタに定義できる型を以下に示します。定数（public static final フィールド）にはプリミティブ型またはString型が定義できます。
ここで定義した型の中でアプリケーション実行時に通信できる型については、“アプリケーション実行時に通信できる型”を参照してください。

プリミティブ型

以下のJavaプリミティブ型はすべて定義できます。

- void
- boolean
- char
- byte
- short
- int
- long
- float
- double

インタフェース型

すべてのインタフェースを定義できます。ただし、java.rmi.Remoteインタフェースを継承する場合は、定義したすべてのメソッド（継承メソッドを含む）がjava.rmi.RemoteExceptionまたはjava.rmi.RemoteExceptionのスーパークラスを例外としてthrowするように定義してください。
また、インタフェースを複数継承する場合、継承されるインタフェース間で同名のメソッドが存在しないように定義してください。

クラス型

すべてのクラスを定義できます。ただし、実装するインタフェースにはjava.rmi.Remoteインタフェースを直接的にも間接的にも継承しないインタフェースを定義してください。

配列型

Home/Remoteインタフェースで定義できる型の配列も定義できます。

例外

すべての例外を定義できます。

アプリケーション実行時に通信できる型

アプリケーション実行時に通信できる型を以下に示します。ここで示す型以外を通信すると実行時に通信エラーとなります。

プリミティブ型

以下のJavaプリミティブ型はすべて定義できます。

- void
- boolean
- char
- byte
- short
- int
- long
- float
- double

インタフェース型

以下のインタフェースを通信できます。

- java.rmi.Remoteインタフェースを継承し、すべてのメソッド(継承メソッドを含む)がjava.rmi.RemoteException、またはjava.rmi.RemoteExceptionのスーパークラスを例外としてthrowするインタフェース

クラス型

以下のクラスを通信できます。

- java.io.Serializableインタフェースを直接的または間接的に実装するクラスで、java.rmi.Remoteインタフェースを直接的にも間接的にも実装しないクラス

java.io.Serializableインタフェースを直接実装していないクラスを通信した場合、ClassCastExceptionが発生します。
java.io.Serializableインタフェースを間接的にも実装していないクラスを通信した場合、org.omg.CORBA.MARSHALが発生します。

配列型

アプリケーション実行時に通信できる型の配列も通信できます。

例外

以下の例外(java.rmi.RemoteExceptionとそのサブクラスを除く)を通信できます。

- java.lang.ErrorまたはjavaLang.RuntimeExceptionを直接的にも間接的にも継承しない例外

15.10.3 注意事項

以下に、EJBアプリケーション作成時の制限事項を示します。

- EJBアプリケーション名の最大長は255バイトです。それ以上長い名前のEJBアプリケーションを作成しないでください。
- ビジネスメソッド名に、256文字より長い名前を使用しないでください。また、JavaからIDLに変換するときの変換規則を適用する場合は、変換後のビジネスメソッド名を256文字以内でビジネスメソッドの定義をしてください。
  変換規則の詳細は、“15.11 IDL変換規則に関する補足資料”を参照してください。
- Homeインタフェース名とRemoteインタフェース名は、234文字以内(パッケージ名を含む)で定義してください。

15.11 IDL変換規則に関する補足資料

EJBアプリケーションは外部との通信にRMI over IIOPを使用しますが、この通信を行うために必要なJavaのクラスは、Interstage管理コンソールでEJBアプリケーションを配備する際に行われるIDL変換処理によって生成されます。
この変換処理ではユーザが作成したHomeインタフェース/Remoteインタフェースクラスから通信用のクラスを生成しますので、その記述内容によってはIDL変換処理でエラーが発生する場合があります。

ここでは、このような場合に参考となるIDL変換規約について説明するとともに、IDL変換時処理で発生するエラーの例について説明いたします。

「Homeインタフェース/Remoteインタフェースで定義できる型」や「アプリケーション実行時に通信できる型」などの情報については“15.10 RMI over IIOPについて”を参照してください。
Interstage管理コンソールで発生するエラーなどについては、Interstage管理コンソールのヘルプを参照してください。

15.11.1 変換規則

JavaからIDLに変換する際の変換規則を以下に示します。変換規則の詳細はhttps://docs.oracle.com/javase/jp/1.4/guide/rmi-iiopを参照してください。
なお、備考欄に○がついているJavaの記述方法を使用する場合には注意してください。JavaクラスからIDLに変換する際に名前が重複し、配備時にエラーとなることがあります。

項	Java	IDL	備考
1	パッケージ(package) 例) a.b.c	モジュール(module) 例) ::a::b::c	―
2	IDLキーワード 例) oneway	先頭に'_'を付加します。 例) _oneway	○
3	先頭が'_' 例) _fred	先頭に'J'を付加します。 例) J_fred	○
4	'$'やユニコード文字などのIDLで定義できない文字 例) a$b, x¥u03bCy	'U'とUnicode値を表す4桁の16進数(大文字)で置換します。 例) aU0024b, xU03BCy	○
5	内部クラス 例) クラスBertの内部クラスFred	外部クラス名と内部クラス名を2つの'_'で連結します。 例) Bert__Fred	○
6	オーバーロードされたメソッド 例) void hello(); void hello(int x, a.b.c y, int z);	以下の手順で変換します。 1. メソッド名に2つの'_'を付加する。 2. IDL変換後の型を2つの'_'で連結する。 3. スペースを'_'で置換する。 例) void hello__(); void hello__long_a_b_c__long(in long x, in ::a::b::c y, in long z); 継承元と継承先で同名のメソッドが存在した場合(オーバーライドを含む)は継承先のメソッド名を変換します。	○
7	大文字/小文字だけが異なる識別子 例) jack, Jack, jAcK	以下の手順で変換します。 1. 末尾に'_'を付加する。	○

項	Java	IDL	備考
		2. 大文字の位置を示す10進インデックスのリストを'_'で連結する。(インデックスは0ベース) 例) jack_, Jack_0, jAcK_1_3	
8	メソッド名が他の識別子(定数、フィールド)の名前と重複した場合 例) 定数"foo"とメソッド"foo"	定数、フィールドの名前の末尾に'_'を付加し、メソッド名は変換しません。 例) 定数"foo_"とメソッド"foo"	○

9	変換後に名前の重なる場合
	例)
	void foo();
	void foo(int x);
	void foo__long();
		エラーとして扱います。	－

10	void	void	－
11	boolean	boolean	－
12	char	char	－
13	byte	byte	－
14	short	short	－
15	int	long	－
16	long	long long	－
17	float	float	－
18	double	double	－
19	リモートインタフェース ・ java.rmi.Remoteインタフェースを直接的または間接的に継承したインタフェース(java.rmi.Remoteインタフェースは除く) ・ すべてのメソッド(継承メソッドを含む)がjava.rmi.RemoteExceptionもしくはそのスーパークラスを生成するように定義されている。	インタフェース(interface)	－

20	java.rmi.Remoteインタフェース	::java::rmi::Remote
		
		//IDL
		module java {
		module rmi {
		typedef Object Remote;
		};
		};
			－

項	Java	IDL	備考
21	リモートインタフェースの継承関係	IDLの継承関係	―
22	以下のいずれかの条件に当てはまるリモートインタフェースのメソッド 1. get<name>メソッドとset<name>メソッドのペアがあり次の条件に当てはまる場合。 ― get<name>メソッドの引数がない。 ― set<name>メソッドの引数が1つで、戻り値がvoidである。 ― get<name>メソッドの復帰値の型がset<name>メソッドの引数の型と同じ。 ― get<name> と set<name> がjava.rmi.RemoteExceptionとそのサブクラス以外の検査例外クラス(*2)を生成しない。 2. get<name>メソッドが次の条件に当てはまる場合。 ― 引数がない。 ― 復帰値の型がvoidでない。 ― java.rmi.RemoteExceptionとそのサブクラス以外の検査例外クラス(*2)を生成しない。 3. is<name>メソッドとset<name>メソッドのペアがあり次の条件に当てはまる場合。 ― is<name>メソッドの引数がない。 ― set<name>メソッドの引数が1つで、戻り値がvoidである。 ― is<name>メソッドの復帰値の型とset<name>メソッドの引数の型がbooleanである。 ― is<name> と set<name> がjava.rmi.RemoteExceptionとそのサブクラス以外の検査例外クラス(*2)を生成しない。 4. is<name>メソッドが次の条件に当てはまる場合。 ― 引数がない。 ― 復帰値の型がbooleanである。 ― java.rmi.RemoteExceptionとそのサブクラス以外の検査例外クラス(*2)を生成しない。	attribute 以下のように変換します。番号(1、2、...)は左欄の番号に対応します。 1. set<name>メソッドの引数の型に対応する読取り書込み可能attribute 2. get<name>メソッドの復帰値の型に対応する読取り専用attribute 3. set<name>メソッドの引数の型に対応する読取り書込み可能attribute 4. is<name>メソッドの復帰値の型に対応する読取り専用attribute attribute名の取得手順を以下に示します。 1. メソッド名先頭の"get"、"is"、または"set"に続く文字を展開する。 2. 最初の文字を小文字に変換する。ただし、1文字目と2文字目の両方が大文字の場合は変換しない。 3. 他の識別子名と名前が重複した場合は語尾に2つの'_'を付加する。	○

項	Java	IDL	備考
23	リモートインタフェースの項目22以外のメソッド	以下の手順で変換します。 1. 項目6、7の変換規則を適用する。 2. 引数を対応するIDL型のinパラメタに変換する。 3. 例外は対応するIDLの例外に変換する。また、java.rmi.RemoteExceptionまたはそのサブクラスと、非検査例外クラス(*1)はIDLにマッピングしない。	―
24	リモートインタフェースの定数(public final staticフィールド)	Javaと同じ値を持ち、対応するIDL型の定数(const)に変換します。wstringとwcharの値はエスケープします。 (*3)	―
25	リモートインタフェースのバージョンチェック	リポジトリID 例) #pragma ID RMInterface "RMI:pack.RMInterface: 0000000000000000"	―
26	直列化可能クラス ・ java.io.Serializableインタフェースを直接的または間接的に継承する。 ・ java.io.Remoteインタフェースを直接的にも間接的にも実装しない。 ・ 以下のクラスを除く。 ― java.lang.String ― java.lang.Class ― org.omg.CORBA.portable.IDLEntity または org.omg.CORBA.portable.ValueBaseを直接的または間接的に実装するクラス	valuetype	―
27	直列化可能クラスで、org.omg.CORBA.portable.IDLEntityまたはorg.omg.CORBA.portable.ValueBaseを直接的または間接的に実装するクラス(*4)	IDLにマッピングしません。	―
28	直列化可能クラスの継承クラス関係	IDLの継承関係	―
29	直列化可能クラスが実装するインタフェースとの関係(java.io.Serializableとjava.io.Externalizableは除く)	・ 実装するインタフェースが項目53に当てはまる場合はIDLの継承関係に変換します。 ・ 実装するインタフェースが項目48に当てはまる場合はIDLのsupports関係に変換します。	―
30	直列化可能クラスのメソッド	IDLにマッピングしません。	―

項	Java	IDL	備考
31	直列化可能クラスのコンストラクタ	IDLにマッピングしません。	－
32	直列化可能クラスの定数(public final staticフィールド)	Javaと同じ値を持ち、対応するIDL型の定数に変換します。wstringとwcharの値はエスケープします。 (*3)	－
33	直列化可能クラスでjava.io.Externalizableを実装しているクラス	custom valuetype	－
34	直列化可能クラスで以下の条件に当てはまる場合 ・ java.io.Externalizableを実装しない。 ・ writeObjectメソッドが定義されている。	custom valuetype	－
35	直列化可能クラスで以下の条件に当てはまる場合 ・ java.io.Externalizableを実装しない。 ・ java.io.ObjectStreamField[]型で名前に serialPersistentFields の private static finalフィールドが宣言されている。	左欄のフィールドserialPersistentFieldsに対応するフィールドだけIDLにマッピングします。 (*3)	－
36	直列化可能クラスのフィールド。 1. 非staticで非transientなpublicフィールド 2. 非 static で 非 transient な private フィールド 3. 上記以外	以下のように変換します。番号(1、2、...)は左欄の番号に対応します。 1. publicデータメンバ 2. privateデータメンバ 3. IDLにマッピングしない。	－
37	直列化可能クラスのバージョンチェック	リポジトリID 例) #pragma ID Hedgehog "RMI:alpha.bravo.Hedgehog: 12345678ABCDEF00:0123456789ABC DEF" (*3)	－
38	java.lang.Stringクラス 1. パラメタ、復帰値、データメンバ 2. 定数	以下のように変換します。番号(1、2)は左欄の番号に対応します。 1. ::CORBA::WStringValue 2. wstring	－
39	java.lang.Classクラス	::javax::rmi::CORBA::ClassDesc	－
40	配列 例1) boolean[] 例2) long[] 例3) a.b.C[] 例4) x.Y[][]	以下のように org.omg.CORBA..boxedRMIモジュールに展開します。 ・ 例 1) モジュール::org::omg::boxedRMIを定義し、以下のvaluetypeを展開する。 valuetype seq1_boolean sequence<boolean>	○

項	Java	IDL	備考
		・例 2) モジュール::org::omg::boxedRMIを定義し、以下のvaluetypeを展開する。 valuetype seq1_long_long sequence<long long>; ・例 3) モジュール::org::omg::boxedRMI::a::bを定義し、以下のvaluetypeを展開する。 valuetype seq1_C sequence<::a::b::C>; ・例 4) モジュール ::org::omg::boxedRMI::xを定義し、以下のvaluetypeを展開する。 valuetype seq1_Y sequence<::x::Y>; valuetype seq2_Y sequence<seq1_Y>;	
41	例外	exceptionとvaluetypeの両方に変換し、exceptionには対応するvaluetypeのフィールドを1つ定義します。 ・exception 以下の手順で変換し、メソッド名のraises節に定義する。 1. 末尾部分が"Exception"の場合は削除する。 2. 名前の末尾に"Ex"を付加する。 3. 他の変換規則を適用する。 ・valuetype 項目26～37と同様に変換され、対応するexceptionに名前が"value"の単一のデータメンバで定義される。	○
42	CORBAオブジェクト参照型 org.omg.CORBA.Objectインタフェースを直接的または間接的に継承し、IDLtoJavaの規約に準拠しているJavaインタフェース(org.omg.CORBA.Objectは除く) (*4)	既存のinterfaceに変換される。新規にIDLは出力しません。	―
43	org.omg.CORBA.Objectインタフェース	Object	―
44	CORBAオブジェクト参照型以外のIDLエンティティ型インタフェース (*4)	IDLエンティティを含む、"boxed"なvaluetypeに変換します。	―

45	java.io.Serializableインタフェース	::java::io::Serializable
		
		//IDL
		module java {
		module io {
		typedef any Serializable;
		};
		};
			―

項	Java	IDL	備考

46	java.io.Externalizableインタフェース	::java::io::Externalizable
		
		//IDL
		module java {
		module io {
		typedef any Externalizable;
		};
		};
			―

47	java.lang.Objectクラス	::java::lang::_Object
		
		//IDL
		module java {
		module lang {
		typedef any _Object;
		};
		};
			○

48	以下の条件に当てはまるインタフェース ・ java.rmi.Remoteインタフェースを直接的にも間接的にも継承しない。 ・ すべてのメソッド定義(継承メソッドを含む)がjava.rmi.RemoteExceptionまたはjava.rmi.RemoteExceptionのスーパークラスを生成する。	abstract interface	―
49	項目48に当てはまるインタフェースの継承インタフェース	継承abstract interface	―
50	項目48に当てはまるインタフェースのメソッド	項目22、23と同様に変換します。	―
51	項目48に当てはまるインタフェースの定数	項目24と同様に変換します。	―
52	リモートインタフェースを実装するクラス 1. 単一リモートインタフェースを実装 2. 複数リモートインタフェースを実装	以下のように変換します。番号(1、2)は左欄の番号に対応します。 1. IDLを出力しない。 2. リモートインタフェースを継承するインタフェース(interface)に変換する。	―
53	項目19、20、42、43、44、45、46、48のどれにも当てはまらないインタフェース	abstract value	―
54	項目26、27、33、34、38、39、47、52のどれにも当てはまらないクラス	abstract value	―

*1) 非検査例外クラス:java.lang.RuntimeExceptionとそのサブクラス、およびjava.lang.Errorとそのサブクラス
*2) 検査例外クラス:非検査例外クラス以外の例外クラス
*3) RMI over IIOPの規約を完全に実装していません。動作には問題ありません。
*4) 本バージョンでは使用しないでください。

15.11.2 EJBアプリケーションの配備でエラーとなる例

以下ではJavaクラスからIDLに変換した際に、EJBアプリケーションの配備でエラーとなる例について説明します。

- 変換規則適用後、クラス名が重なる例

- 変換規則適用後、メソッド名が重なる例
  - 適用した変換規則:4
  - 適用した変換規則:7
- 変換規則適用後、フィールド名と他の識別子名とが重複する例
- 変換規則適用後、attribute名と他の識別子名が重複する例
- 変換規則適用後、例外名（exception）が重複する例

エラーが発生した場合は、前項の変換規則や、本項のエラー事例を参考にしながら、下記のポイントをチェックして対処してください。

- エラーとなったキーワードを含むJavaアプリケーションを確認してください。
- 変換規則の説明をもとに名前が重複する可能性がないかを確認してください。

変換規則適用後、クラス名が重なる例

適用した変換規則

3

エラーメッセージ例

D: ￥ EJB ￥ deployment ￥ work ￥ EB0919350001 ￥ pack001 ￥ _class002Remote_Stub.java:37: class pack001._class002Remote_Stub で宣言された メソッド pack001._under meth001() は 、interface pack001.class002Remote で宣言された同じシグニチャーのメソッドをオーバーライドできません。同じ型の戻り値を持たなければなりません。
public pack001._under meth001() throws RemoteException

D:¥EJB¥deployment¥work¥EB0919350001¥pack001¥_FJclass002RemoteImpl_Tie.java:74: この型は declaration には不適合です。pack001.J_under から pack001._under には変換できません。
pack001._under _result = _target.meth001();

エラー 2 個

Java記述例

```
public pack001.J_under meth001() throws java.rmi.RemoteException;
public pack001._under meth002() throws java.rmi.RemoteException;
```

IDLマッピング例

```
::pack001::J_under meth001();
::pack001::J_under meth002();
```

エラー原因

クラス名“_under”が“J_under”に変換された結果、クラス名が重複しました。

変換規則適用後、メソッド名が重なる例(適用した変換規則:4)

適用した変換規則

4

エラーメッセージ例

FJJ2IDL0003S 変名後のメソッド名と同名のメソッドが定義されています。 InterfaceName = pack.test007Remote MethodName = foo__long

Java記述例

```
public void foo() throws java.rmi.RemoteException;
public void foo(int x) throws java.rmi.RemoteException;
public void foo__long() throws java.rmi.RemoteException;
```

IDLマッピング例

エラーが発生したためIDLは出力されません。

エラー原因

メソッド“foo”がオーバーロードされたために変換され、メソッド“foo_long”と名前が重複しました。

適用した変換規則

4

エラーメッセージ例

<D4105S-02-002-1701> 展開処理中に異常が発生しました。
詳細メッセージに従い対処してください。
<D4298S-04-010-0307> IDLcでエラーを検出しました。
IDLc -roi -info"D:¥EJB¥deployment¥work¥EB1558300003¥j2idl.xml" -I"D:¥EJB¥deployment¥idl" -I"D:¥EJB¥deployment¥work¥EB1558300003" "D:¥EJB¥deployment¥work¥EB1558300003¥pack¥test001Home.idl"
OD: エラー: od51116:IDLroijavacg:aU0024bは識別子として二重定義されました。
OD: エラー: od51221:IDLc:IDLroijavacgコマンドエラー. エラーコード=4

Java記述例

```
// pack.Exclass001
package pack;
public interface Exclass001 {
   public int a$b(int param1, int param2) throws java.rmi.RemoteException;
}

// pack.test001Remote
package pack;
import java.rmi.*;
import javax.ejb.*;
public interface test001Remote extends javax.ejb.EJBObject,Exclass001 {
   public int aU0024b(int param1, int param2) throws java.rmi.RemoteException;
}
```

IDLマッピング例

```
//pack/Exclass001.idl
module pack{
   abstract interface Exclass001 {
      long aU0024b(
         in long arg0,
         in long arg1);
   };
#pragma ID Exclass001 "RMI:pack.Exclass001:0000000000000000"
};

//pack/test001Remote.idl
module pack{
   interface test001Remote: ::javax::ejb::EJBObject, ::pack::Exclass001 {
      long aU0024b(
         in long arg0,
         in long arg1);
   };
#pragma ID test001Remote "RMI:pack.test001Remote:0000000000000000"
};
```

エラー原因

継承元のメソッド“a$b”が“aU0024b”に変換されて、継承先のメソッド“aU0024”と名前が重複しました。

変換規則適用後、メソッド名が重なる例(適用した変換規則:7)

適用した変換規則

7

エラーメッセージ例

<D4105S-02-002-1701> 展開処理中に異常が発生しました。
詳細メッセージに従い対処してください。
<D4298S-04-010-0307> IDLcでエラーを検出しました。
IDLc -roi -info"D:¥EJB¥deployment¥work¥EB1339510001¥j2idl.xml" -I"D:¥EJB¥deployment¥idl" -I"D:¥EJB¥deployment¥work¥EB1339510001" "D:¥EJB¥deployment¥work¥EB1339510001¥pack002¥meth001Home.idl"
OD: エラー: od51116:IDLroijavacg:meth1_は識別子として二重定義されました。
OD: エラー: od51221:IDLc:IDLroijavacgコマンドエラー. エラーコード=4

Java記述例

```
//pack002.meth001Remote
package pack002;
public interface meth001Remote extends SubRemote,javax.ejb.EJBObject {
   public void meth1() throws java.rmi.RemoteException;
   public void meth1(long param1) throws java.rmi.RemoteException;
}

//pack002.SubRemote
package pack002;
public interface SubRemote extends java.rmi.Remote {
   public void meth1() throws java.rmi.RemoteException;
   public void meth1(int param1,int param2) throws java.rmi.RemoteException;
}
```

IDLマッピング例

```
//pack002/meth001Remote.idl
module pack002{
   interface meth001Remote: ::pack002::SubRemote, ::javax::ejb::EJBObject {
      void meth1__();
      void meth1__long_long(
         in long long arg0);
   };
#pragma ID meth001Remote "RMI:pack002.meth001Remote:0000000000000000"
};

//pack002/SubRemote.idl
module pack002{
   interface SubRemote {
      void meth1__();
      void meth1__long__long(
         in long arg0,
         in long arg1);
   };
#pragma ID SubRemote "RMI:pack002.SubRemote:0000000000000000"
};
```

エラー原因

継承元と継承先でそれぞれメソッド名が変換されて、メソッド名が重複しました。

適用した変換規則

7

エラーメッセージ例

<D4105S-02-002-1701> 展開処理中に異常が発生しました。
詳細メッセージに従い対処してください。
<D4298S-04-010-0307> IDLcでエラーを検出しました。
IDLc -roi -info"D:¥EJB¥deployment¥work¥EB1523460001¥j2idl.xml" -I"D:¥EJB¥deployment¥idl" -I"D:¥EJB¥deployment¥work¥EB1523460001" "D:¥EJB¥deployment¥work¥EB1523460001¥pack002¥method003Home.idl"
OD: エラー: od51116:IDLroijavacg:method1_は識別子として二重定義されました。
OD: エラー: od51221:IDLc:IDLroijavacgコマンドエラー. エラーコード=4

Java記述例

```
// pack002.Subclass1
package pack002;
public interface Subclass1 extends java.rmi.Remote {
   public void method1() throws java.rmi.RemoteException;
}

// pack002.Subclass2
package pack002;
public interface Subclass2 extends Subclass1 {
   public void method1() throws java.rmi.RemoteException;
}

// pack002. method003Remote
package pack002;
public interface method003Remote extends Subclass2,javax.ejb.EJBObject {
   public void method1() throws java.rmi.RemoteException;
}
```

IDLマッピング例

```
// pack002/Subclass1.idl
module pack002{
   interface Subclass1 {
      void method1();
   };
#pragma ID Subclass1 "RMI:pack002.Subclass1:0000000000000000"
};

// pack002/Subclass2.idl
module pack002{
   interface Subclass2: ::pack002::Subclass1 {
      void method1__();
   };
#pragma ID Subclass2 "RMI:pack002.Subclass2:0000000000000000"
};

// pack002/method003Remote.idl
module pack002{
   interface method003Remote: ::pack002::Subclass2, ::javax::ejb::EJBObject {
      void method1__();
   };
#pragma ID method003Remote "RMI:pack002.method003Remote:0000000000000000"
};
```

エラー原因

継承元と継承先でそれぞれメソッド名が変換されて、メソッド名が重複しました。

変換規則適用後、フィールド名と他の識別子名とが重複する例

適用した変換規則

7

エラーメッセージ例

FJJ2IDL0005S 変名後のフィールド名と同名のフィールドが定義されています。 ClassName = pack.Valueclass FieldName = jack_

Java記述例

```
public int jack_;
public int jAcK;
public int jack;
```

IDLマッピング例

エラーが発生したためIDLは出力されません。

エラー原因

定義されたフィールド“jAcK”と“jack”が大文字/小文字だけ違うため、“jack”が“jack_”と変換された結果、他のフィールド“jack_”と名前が重複しました。

変換規則適用後、attribute名と他の識別子名が重複する例

適用した変換規則

22

エラーメッセージ例

```
<D4105S-02-002-1701> 展開処理中に異常が発生しました。
詳細メッセージに従い対処してください。
<D4298S-04-010-0307> IDLcでエラーを検出しました。
IDLc -roi -info"D:￥EJB￥deployment￥work￥EB1711200001￥j2idl.xml" -I"D:￥EJB￥deployment￥idl" -I"D:￥EJB
￥deployment￥work￥EB1711200001" "D:¥EJB¥deployment¥work¥EB1711200001¥pack¥test002Home.idl"
OD: エラー: od51116:IDLroijavacg:ATTRは識別子として二重定義されました。
OD: エラー: od51221:IDLc:IDLroijavacgコマンドエラー. エラーコード=4
```

Java記述例

```
// pack.ExIntf002
package pack;
public interface ExIntf002 extends java.rmi.Remote {
   public int getATTR() throws java.rmi.RemoteException;
}

// pack.test002Remote
package pack;
import java.rmi.*;
import javax.ejb.*;
public interface test002Remote extends javax.ejb.EJBObject,ExIntf002 {
   public void ATTR() throws java.rmi.RemoteException;
}
```

IDLマッピング例

```
// pack/ExIntf002.idl
module pack{
   interface ExIntf002 {
      readonly attribute long ATTR;
   };
#pragma ID ExIntf002 "RMI:pack.ExIntf002:0000000000000000"
};
```

```
// pack/test002Remote
module pack{
   interface test002Remote: ::javax::ejb::EJBObject, ::pack::ExIntf002 {
      void ATTR();
   };
#pragma ID test002Remote "RMI:pack.test002Remote:0000000000000000"
};
```

エラー原因

継承元のメソッドがattributeに変換され、継承先のメソッドと名前が重複しました。

適用した変換規則

22

エラーメッセージ例

FJJ2IDL0032S 同名のアトリビュートが定義されています。InterfaceName = pack.test010Remote AttributeName = ATTR

Java記述例

```
public int getATTR() throws java.rmi.RemoteException;
public void setATTR(int param1) throws java.rmi.RemoteException;
public boolean isATTR() throws java.rmi.RemoteException;
```

IDLマッピング例

エラーが発生したためIDLは出力されません。

エラー原因

“getATTR”メソッドと“setATTR”メソッドがattributeに変換され、“isATTR”メソッドもattributeに変換された結果、attribute名が重複しました。

適用した変換規則

22

エラーメッセージ例

FJJ2IDL0033S アトリビュート名と同名のメソッドが定義されています。 InterfaceName = pack.test011Remote AttributeName = ATTR__

Java記述例

```
public long getATTR() throws java.rmi.RemoteException;
public void setATTR(long param1) throws java.rmi.RemoteException;
public void ATTR() throws java.rmi.RemoteException;
public void ATTR__() throws java.rmi.RemoteException;
```

IDLマッピング例

エラーが発生したためIDLは出力されません。

エラー原因

“getATTR”メソッドと“setATTR”メソッドがattributeに変換され、“ATTR”メソッドとattribute名が重複したため、語尾に“__”を付加した結果、“ATTR__”メソッドと名前が重複しました。

変換規則適用後、例外名（exception）が重複する例

適用した変換規則

41

エラーメッセージ例

<D4105S-02-002-1701> 展開処理中に異常が発生しました。
詳細メッセージに従い対処してください。
<D4296S-04-010-0305> IDLcでシンタックスエラーを検出しました。
IDLc -roi -info"D:¥EJB¥deployment¥work¥EB0936420001¥j2idl.xml" -I"D:¥EJB¥deployment¥idl" -I"D:¥EJB¥deployment¥work¥EB0936420001" "D:¥EJB¥deployment¥work¥EB0936420001¥pack001¥class003Home.idl" "D:¥EJB¥deployment¥work¥EB0936420001¥pack001/class003Remote.idl",
OD: エラー: od51108:IDLejbparser:raises句で::pack001::TestExEx例外が二重定義されました。
FILE=D:¥EJB¥deployment¥work¥EB0936420001¥pack001/class003Remote.idl LINE=17
OD: エラー: od51240:IDLejbparser:致命的エラーが発見されました。エラー数 = 1
OD: エラー: od51221:IDLc:IDLejbparserコマンドエラー. エラーコード=2

Java記述例

```
public void meth001() throws TestExException, TestEx, java.rmi.RemoteException;
```

IDLマッピング例

```
void meth001() raises(
   ::pack001::TestExEx,
   ::pack001::TestExEx);
```

エラー原因

例外クラスの“TestExException”が“TestExEx”にマッピングされ、“TestEx”が“TestExEx”に変換された結果、例外名が重複しました。

適用した変換規則

41

エラーメッセージ例

D: ¥ EJB ¥ deployment ¥ work ¥ EB0946180001 ¥ pack001 ¥ _class004Remote_Stub.java:69: class pack001._class004Remote_Stub で宣言されたメソッド void meth002() は、interface pack001.class004Remote で宣言された同じシグニチャーのメソッドをオーバーライドできません。それらの throws 節には互換性がありません。
Public void meth002() throws RemoteException, pack001.TestEx

D:¥EJB¥deployment¥work¥EB0946180001¥pack001¥_FJclass004RemoteImpl_Tie.java:96: 例外 pack001.TestEx は対応する try 文の本体でスローされることはありません。
catch (pack001.TestEx _exp){

エラー 2 個

Java記述例

```
public void meth001() throws TestEx, java.rmi.RemoteException;
public void meth002() throws TestExException, java.rmi.RemoteException;
```

IDLマッピング例

```
void meth001() raises(
   ::pack001::TestExEx);
void meth002() raises(
   ::pack001::TestExEx);
```

エラー原因

例外クラスの“TestExException”が“TestExEx”にマッピングされ、“TestEx”が“TestExEx”に変換された結果、例外名が重複しました。

第16章　運用コマンドを使用してカスタマイズする方法

EJBアプリケーションの実行環境を、カストマイズツールの運用コマンドを使用してカスタマイズする方法について説明します。この方法では、利用者が更新したXML形式の定義ファイルを元にしてコマンドを実行することにより、EJBアプリケーションの実行環境定義の内容(“**Enterprise Bean定義情報**”)を編集できます。

カスタマイズに使用するEJBサービス運用コマンドには、大きくわけて以下の機能があります。

- 定義ファイルのexport
  EJBアプリケーションの実行環境定義の内容を、XML形式の定義ファイルに移出する
- 定義ファイルのimport
  更新されたXML形式の定義ファイルの内容を、EJBアプリケーションの実行環境定義に移入する

以下に、EJBサービスの運用コマンドを使用したカスタマイズの流れ、各定義ファイルのexportとimportの手順、および各定義ファイルの内容と記述例を説明します。

16.1 カスタマイズの流れ

コマンドを使用したカスタマイズでは、ejbdefexportコマンドを使用してEJBサービスからexportした定義ファイル、または新規に作成した定義ファイルを元に、ejbdefimportコマンドを実行し、各定義情報を編集します。

各定義ファイルと、カスタマイズに使用するコマンドについて説明します。

定義ファイル

カスタマイズで使用する定義ファイルを以下に示します。

定義ファイル	内容
Enterprise Bean定義ファイル	Enterprise Bean定義情報をXML形式で表現したファイル

カスタマイズに使用するコマンド

カスタマイズで使用するコマンドを以下に示します。

コマンド	機能
ejbdefexport	Enterprise Bean定義情報をEnterprise Bean定義ファイルにexportします。
ejbdefimport	Enterprise Bean定義ファイルをEnterprise Bean定義情報にimportします。

各コマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。

16.2 Enterprise Bean定義情報のexportとimport

Enterprise Bean定義情報のexportとimportについて説明します。

Enterprise Bean定義情報のexport

ejbdefexportコマンドを実行すると、定義済のEnterprise Bean定義情報の内容がEnterprise Bean定義ファイルにexportされます。
このとき、IJServer内のすべてのEnterprise Bean定義ファイルを一括してexportすることもできます。


例

Windows32/64

- “TestIJServer”というIJServerに配備されている、SampleEBの情報をexportする場合

```
ejbdefexport SampleEB -i TestIJServer -f c:¥ejb¥deffile.xml
```

- “TestIJServer”というIJServerに配備されている、すべてのEJBアプリケーションの情報を、c:¥ejbというディレクトリにexportする場合

```
ejbdefexport -i TestIJServer -all c:¥ejb
```

Solaris64 Linux32/64

- “TestIJServer”というIJServerに配備されている、SampleEBの情報をexportする場合

```
ejbdefexport SampleEB -i TestIJServer -f /tmp/ejb/deffile.xml
```

- “TestIJServer”というIJServerに配備されている、すべてのEJBアプリケーションの情報を、/tmp/ejbというディレクトリにexportする場合

```
ejbdefexport -i TestIJServer -all /tmp/ejb
```

Enterprise Bean定義情報のimport

ejbdefimportコマンドを実行してEnterprise Bean定義ファイルのimportを行い、定義ファイルの内容をEnterprise Bean定義情報に反映します。Enterprise Bean定義ファイルは、ejbdefexportコマンドでexportしたファイルを編集するか、または新規に作成します。
このとき、複数のEnterprise Bean定義ファイルを一括してimportすることもできます。

例

Windows32/64

- c:¥ejb¥deffile.xmlに定義されている、“TestIJServer”というIJServerに配備されたEnterprise Bean定義情報をimportする場合

```
ejbdefimport -i TestIJServer -f c:¥ejb¥deffile.xml
```

- c:¥ejbというフォルダにある、“TestIJServer”というIJServerに配備されたすべてのEnterprise Bean定義ファイルの情報をimportする場合

```
ejbdefimport -i TestIJServer -all c:¥ejb
```

Solaris64 Linux32/64

- /tmp/ejb/deffile.xmlに定義されている、“TestIJServer”というIJServerに配備されたEnterprise Bean定義情報をimportする場合

```
ejbdefimport -i TestIJServer -f /tmp/ejb/deffile.xml
```

- /tmp/ejbというフォルダにある、“TestIJServer”というIJServerに配備されたすべてのEnterprise Bean定義ファイルの情報をimportする場合

```
ejbdefimport -all /tmp/ejb
```

注意

ejbdefexportコマンドでは、定義されている値がそのまま抽出されます。配備時に作成したEnterprise Bean定義情報に矛盾がある場合、ejbdefexportコマンドでexportしたEnterprise Bean定義情報を使ってejbdefimportを実行すると、動作上は無視される矛盾でもejbdefimportコマンドが矛盾をチェックしてimportに失敗する場合があります。

16.3 Enterprise Bean定義ファイルの内容

Enterprise Bean定義ファイルについて

Enterprise Bean定義ファイルはXML形式で記述します。

Enterprise Bean定義ファイルは、以下の上位タグと、それぞれの上位タグの配下にある下位タグで構成されています。
下位タグについては、それぞれの上位タグの表を参照してください。

```
<ejbdef>
  <ejb-jar>
    <!--ejb-jarタグの下位タグ-->
  </ejb-jar>
  <fujitsu-bean-definition>
    <!--fujitsu-bean-definitionタグの下位タグ-->
  </fujitsu-bean-definition>
  <fujitsu-cmp-definition(またはfujitsu-cmp2x-mapping-definition)>
    <!--fujitsu-cmp-definitionタグ(またはfujitsu-cmp2x-mapping-definitionタグ)の下位タグ-->
  </fujitsu-cmp-definition (または/fujitsu-cmp2x-mapping-definition)>
</ejbdef>
```

注意

- fujitsu-cmp-definitionタグとfujitsu-cmp2x-mapping-definitionタグは、同時に定義できません。
- EJB2.1に準拠したEJBアプリケーションの場合には、ejb-jarタグ配下の定義の移出入ができません。配備前に、開発環境においてejb-jar.xmlファイルを編集してください。

タグの構成と内容

- ejb-jarタグの構成と内容
- fujitsu-bean-definitionタグの構成と内容
- fujitsu-cmp-definitionタグの構成と内容
- fujitsu-cmp2x-mapping-definitionタグの構成と内容

表の見方

- タグ名:定義ファイルの中のXMLタグ名
- 値:定義ファイル内のXMLの値
- 意味:タグの意味(Interstage管理コンソールで表示される項目は、Interstage管理コンソールでの項目名です。)
- 編集:値の編集可否(○:編集可能 ×:編集不可能)
- Interstage管理コンソール画面との対応:Interstage管理コンソール画面との対応
- ?付きのタグは省略できます。
- +付きのタグは繰り返し指定できるタグです。
- 編集可能な項目は、値を選択するものについては、指定された値が選択肢に存在するかチェックします。数値型の場合は、値の型と、範囲のチェックを行います。
- 編集可能な項目で、Enterprise Beanの定義情報に項目が存在する場合は、Enterprise Beanの定義ファイルの値に編集されます。
- 編集可能な項目で、Enterprise Beanの定義情報に項目が存在しない場合は、Enterprise Beanの定義情報に項目が追加されます。
- 編集できない項目は、定義ファイルの値とEnterprise Beanの定義情報の値を比較し、異なっていた場合はエラーとなります。

注意

- Interstage管理コンソールで“<”、“>”、“&”を入力した場合、exportで出力した定義ファイルには、それぞれ“<”、“>”、“&”と表示されます。
  また定義ファイル内に、“<”、“>”、“&”を入力して、importすると以下のエラーが表示されます。定義ファイル内では、“<”、“>”、“&”は、それぞれ“<”、“>”、“&”と記述してください。

```
定義ファイル(ファイル名)の読込み中にエラーが発生しました。
不明なマークアップがあります。
file:///(XX) : 行 XX, 桁 XX で上記エラーを検出しました。
EJB3504S-20-093-0203
```

- Enterprise Bean定義情報をexportした場合、Enterprise Bean定義ファイルの“password”には、パスワードの文字数分“*”が出力されます。この値を変更しないでimportした場合、そのままの文字列がEnterprise Bean定義情報に反映されるので注意してください。

ejb-jarタグの構成と内容

ejb-jarタグはEJB2.0規約以前に準拠したEJBアプリケーションのみ有効となります。EJB2.0規約のEJBアプリケーションを使用する場合のタグについて以下に説明します。

表16.1 ejb-jarタグ

タグ名	意味
ejb-jar?	
enterprise-beans	
session+	Session Beanに関する定義
entity+	Entity Beanに関する定義
message-driven+	Message-driven Beanに関する定義
relationships?	CMRマッピングに関する定義
assembly-descriptor?	セキュリティやトランザクションに関する定義

表16.2 <ejb-jar><enterprise-beans><session>の下位タグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
ejb-name?	任意の文字列	Enterprise Bean名	×	EJBアプリケーション情報
home?	任意の文字列	Homeインタフェース名	×	
remote?	任意の文字列	Remoteインタフェース名	×	
local-home?	任意の文字列	LocalHomeインタフェース名	×	
local?	任意の文字列	Localインタフェース名	×	
ejb-class?	任意の文字列	Enterprise Beanクラス名	×	
session-type?	以下の値から選択 ・ Stateful ・ Stateless	セション種別	×	
transaction-type?	以下の値から選択	トランザクション管理種別	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
	・ Bean ・ Container			
env-entry+				環境プロパティ
description?	任意の文字列	説明	×	
env-entry-name	任意の文字列 export時に、定義ファイルとEnterprise Bean定義情報の間で、env-entry-nameの整合性がとれていない場合、エラーとなります。	プロパティ名	×	
env-entry-type	以下の値から選択 ・ java.lang.Boolean ・ java.lang.String ・ java.lang.Integer ・ java.lang.Double ・ java.lang.Byte ・ java.lang.Short ・ java.lang.Long ・ java.lang.Float ・ java.lang.Character	型	○	
env-entry-value?	以下の範囲の値 ・ java.lang.Boolean: True/False 注) 値を省略(空文字列を指定)することはできません。 ・ java.lang.String: 任意の文字列 ・ java.lang.Integer: -2147483648～2147483647 ・ java.lang.Double: -1.7976931348623157E308～1.7976931348623157E308 ・ java.lang.Byte: -128～127 ・ java.lang.Short: -32768～32767	値	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
	• java.lang.Long: -9223372036854775808～9223372036854775807 • java.lang.Float: -3.4028234663852886E38～3.4028234663852886E38 • java.lang.Character: 任意の文字			
ejb-ref+				参照EJB
description?	任意の文字列	説明	×	
ejb-ref-name	任意の文字列	Enterprise BeanのJNDI名	×	
ejb-ref-type	以下の値から選択 • Session • Entity	Enterprise Bean種別	×	
home	任意の文字列	Homeインタフェース名	×	
remote	任意の文字列	Remoteインタフェース名	×	
ejb-link?	任意の文字列	Enterprise Bean名	×	
ejb-local-ref+				参照LocalEJB
description?	任意の文字列	説明	×	
ejb-ref-name	任意の文字列	Enterprise BeanのJNDI名	×	
ejb-ref-type	以下の値から選択 • Session • Entity	Enterprise Bean種別	×	
local-home	任意の文字列	LocalHomeインタフェース名	×	
local	任意の文字列	Localインタフェース名	×	
ejb-link?	任意の文字列	Enterprise Bean名	×	
security-role-ref+				参照セキュリティロール
description?	任意の文字列	説明	×	
role-name	任意の文字列	コード化セキュリティロール名	×	
role-link?	任意の文字列	セキュリティロール名	×	
security-identity? (注1)				セキュリティアイデンティティ
description? (注1)	任意の文字列	説明	×	
use-caller-identity	なし(run-asとは排他指定)	呼出し側のセキュリティを使用	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
run-as				
description?	任意の文字列	説明	○	
role-name	任意の文字列	セキュリティロール名	○	
resource-ref+				参照リソース
description?	任意の文字列	説明	×	
res-ref-name	任意の文字列	リソースマネージャー名	×	
res-type	任意の文字列	クラス/インタフェース名	×	
res-auth	以下の値から選択 ・ Application ・ Container	リソース接続者	×	
resource-env-ref+				参照環境リソース
description?	任意の文字列	説明	×	
resource-env-ref-name	任意の文字列	リソースマネージャー名	×	
resource-env-ref-type	任意の文字列	クラス/インタフェース名	×	

注1) run-asが指定された場合のみ有効です。

表16.3 <ejb-jar><enterprise-beans><entity>の下位タグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
ejb-name?	任意の文字列	Enterprise Bean名	×	EJBアプリケーション情報
home?	任意の文字列	Homeインタフェース名	×	
remote?	任意の文字列	Remoteインタフェース名	×	
local-home?	任意の文字列	LocalHomeインタフェース名	×	
local?	任意の文字列	Localインタフェース名	×	
ejb-class?	任意の文字列	Enterprise Beanクラス名	×	
persistence-type?	以下の値から選択 ・ Bean ・ Container	Persistenceタイプ	×	
prim-key-class?	任意の文字列	PrimaryKeyクラス名	×	
reentrant?	以下の値から選択 ・ True ・ False	リエントラント種別	×	
cmp-version?	以下の値から選択	CMPバージョン	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
	・1.x ・2.x			
abstract-schema-name?	任意の文字列	抽象スキーマ名	○	
primkey-field?	任意の文字列	PrimaryKeyフィールド名	×	
env-entry+	sessionタグのenv-entryを参照			
ejb-ref+	sessionタグのejb-refを参照			
ejb-local-ref+	sessionタグのejb-local-refを参照			
security-role-ref+	sessionタグのsecurity-role-refを参照			
security-identity?	sessionタグのsecurity-identityを参照			
resource-ref+	sessionタグのresource-refを参照			
resource-env-ref+	sessionタグのresource-env-refを参照			
query+				クエリ定義
description?	任意の文字列	説明	×	
query-method				
method-name	任意の文字列	メソッド名	×	
method-params				
method-param+	任意の文字列	パラメタ	×	
result-type-mapping?	以下の値から選択 ・Local ・Remote	結果タイプ	×	
ejb-ql	任意の文字列	EJB QL	×	

表16.4 <ejb-jar><enterprise-beans><message-driven>の下位タグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
ejb-name?	任意の文字列	Enterprise Bean名	×	EJBアプリケーション情報
ejb-class?	任意の文字列	Enterprise Beanクラス名	×	
transaction-type?	以下の値から選択 ・Bean ・Container	トランザクション管理種別	○	
message-selector?	任意の文字列	メッセージセレクタ	○	
message-driven-destination?				
destination-type	以下の値から選択 ・javax.jms.Topic	Destinationタイプ	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
	・ javax.jms.Queue			
subscription-durability?	以下の値から選択 ・ Durable ・ NonDurable	サブスクライバの永続性	○	
env-entry+	sessionタグのenv-entryを参照			
ejb-ref+	sessionタグのejb-refを参照			
ejb-local-ref+	sessionタグのejb-local-refを参照			
security-identity?	sessionタグのsecurity-identityを参照			
resource-ref+	sessionタグのresource-refを参照			
resource-env-ref+	sessionタグのresource-env-refを参照			

表16.5 <ejb-jar><relationships>の下位タグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
description?	任意の文字列	説明	×	CMRマッピング定義
ejb-relation+				
description?	任意の文字列	説明	×	
ejb-relation-name?	任意の文字列	EJBリレーション名	×	
ejb-relationship-role				
description?	任意の文字列	説明	×	
ejb-relationship-role-name?	任意の文字列	EJBリレーションロール名	×	
multiplicity	以下の値から選択 ・ One ・ Many	多重度	×	
cascade-delete?	なし	レコード削除	×	
relationship-role-source				
description?	任意の文字列	説明	×	
ejb-name	任意の文字列	Enterprise Bean名	×	
cmr-field?				
description?	任意の文字列	説明	×	
cmr-field-name	任意の文字列	CMRフィールド名	×	
cmr-field-type?	任意の文字列	型	×	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
ejb-relationship-role	上記参照			

表16.6 <ejb-jar><assembly-descriptor>の下位タグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
security-role+				セキュリティロール
description?	任意の文字列	説明	×	
role-name	任意の文字列	セキュリティロール名	×	
method-permission+				メソッドパーミッション
role-name+	任意の文字列 指定する文字列は、security-roleタグのrole-nameに定義されている必要があります。	セキュリティロール名	○	
unchecked	ー	メソッドパーミッションのチェック可否	○	
method+				
description?	任意の文字列	説明	×	
ejb-name	任意の文字列	Enterprise Bean名	×	
method-intf?	以下の値から選択 ・ Home ・ Remote	インタフェース種別	×	
method-name	任意の文字列	メソッド名	×	
method-params?				
method-param+	任意の文字列	メソッドパラメタリスト	×	
container-transaction+ (注1)				トランザクション
method+				
description?	任意の文字列	説明	×	
ejb-name	任意の文字列	Enterprise Bean名	×	
method-intf?	以下の値から選択 ・ Home ・ Remote	インタフェース種別	×	
method-name	任意の文字列	メソッド名	×	
method-params?				

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
method-param+	任意の文字列	メソッドパラメタリスト	×	
trans-attribute	以下の値から選択 ・NotSupported ・Required ・Supports ・RequiresNew ・Mandatory ・Never	トランザクション属性	○	

注1) <transaction-type>がContainerの場合のみ指定できます。

fujitsu-bean-definitionタグの構成と内容

表16.7 fujitsu-bean-definitionタグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
fujitsu-bean-definition?				
description?	任意の文字列	メモ	○	Interstage拡張情報
version-entry? (注1)				
deploy-ejb-version?	以下の値固定 ・1.1 ・2.0 ・2.1 (注8)	準拠EJB規約バージョン	×	
deploy-java-version?	以下の値から選択 ・1.1 ・1.2 ・1.3 ・1.4	Deploy時の使用JDKバージョン	×	
base?				
jndi-name? (注2)	任意の文字列	Enterprise BeanのJNDI名	×	EJBアプリケーション名
tran-timeout?	以下の範囲の値 ・0～2147483647 デフォルト値は0です。	トランザクションタイムアウト値	○	－

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
Windows32/64 Linux32/64 tran-kind? (注1)	以下の値から選択 ・ Local :デフォルト値 ・ Global	分散トランザクション	○	
local-mode?	以下の値から選択 ・ False :デフォルト値 ・ True	ローカル呼出しを使用	○	Interstage拡張情報
redirect? (注1)				―
redirect-mode?	以下の値から選択 ・ False :デフォルト値 ・ True	標準出力、標準エラー出力取得モード	○	
redirect-path?	任意の文字列	標準出力、標準エラー出力ファイル	○	
trace? (注1)				―
trace-mode?	以下の値から選択 ・ None ・ Buffer ・ File	トレースの取得モード	×	
trace-level?	以下の範囲の値 ・ 0～2	トレースの取得レベル	×	
trace-buffer?	0以上	トレースの最大取得レコード数	×	
trace-path?	任意の文字列	トレースファイルを出力するディレクトリ名	×	
impl?				EJBアプリケーション情報
impl-home-intfrep?	任意の文字列	Homeインタフェースリポジトリ ID	×	
impl-remote-intfrep?	任意の文字列	Remoteインタフェースリポジトリ ID	×	
session-eb? (注2)				―
session-timeout? (注3)	以下の範囲の値 ・ 0～2147483647 デフォルト値は0です。	セションタイムアウト値	○	
session-idle-timeout? (注3)	以下の範囲の値 ・ 0～2147483647 デフォルト値は1800です。 IJServerワークユニットのJava VMオプションで、無通信監視時間を設定することもできます。Java	無通信監視時間	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
	VMオプションで無通信監視時間が設定されている場合、本設定値は無視されます。詳細は、”リファレンスマニュアル(コマンド編)”の”IJServer定義ファイル”を参照してください。			
max-ejbobject? (注3)	以下の範囲の値 ・ 1～2147483647 デフォルト値は1024です。	StatefulBean同時接続数	○	
initial-stateless-instance-count? (注4)	以下の範囲の値 ・ 0～2147483647 デフォルト値は0です。	Stateless Beanの初期起動インスタンス数	○	
entity-eb? (注5)				Interstage拡張情報
entity-timeout?	以下の範囲の値 ・ 1～2147483647 デフォルト値は120です。	Entity BeanのEJB objectタイムアウト値	○	
entity-instance-type?	以下の値から選択 ・ ReadWrite :デフォルト値 ・ ReadOnly ・ Sequential	Entity Beanのインスタンス管理モード	○	
entity-instance-size?	以下の範囲の値 ・ 1～2147483647 デフォルト値は100です。	Entity Beanのインスタンス数	○	
entity-instance-create-type?	以下の値から選択 ・ At Start-Up ・ At First Access ・ As Required :デフォルト値	Entity Beanのインスタンス生成モード	○	
entity-batch-operations? (注7)	以下の値から選択 ・ True :デフォルト値 ・ False	CMP1.1の複数レコードの一括更新	○	
message-driven-eb? (注6)				Message-driven Bean拡張情報
message-type?	以下の値から選択 ・ JMS ・ resourceadapter	受信対象種別	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
jms?				
max-messages? (注1)	以下の範囲の値 ・1～10000000 デフォルト値は1です。	メッセージバッファ最大値	×	
subscription-name?	任意の文字列 注) 値を省略(空文字列を指定)することはできません。	サブスクライバの識別名	○	
connection-factory-name?	任意の文字列 注) 値を省略(空文字列を指定)することはできません。	JMSコネクションファクトリ名	○	
bean-pool-size?	以下の範囲の値 ・1～10000000 デフォルト値は8です。	初期起動インスタンス数	○	
destination-name?	任意の文字列 注) 値を省略(空文字列を指定)することはできません。	Destination名	○	
retry-count?	以下の範囲の値 ・0～50 デフォルト値は0です。importを行う時に、以下の場合にだけ値が編集できます。 ・message-drivenタグの transaction-typeに“Container”を指定 ・container-transactionタグの trans-attributeに“Required”を指定	リトライカウント	○	異常時メッセージ退避定義
backup-connection-factory-name?	任意の文字列 importを行う時に、以下の場合にだけ値が編集できます。 ・message-drivenタグの transaction-typeに“Container”を指定 ・container-transactionタグの	異常時ループ対処用JMSコネクションファクトリ名	○	

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
	trans-attribute に“Required”を指定			
backup-destination-name?	任意の文字列 importを行う時に、以下の場合にだけ値が編集できます。 ・message-drivenタグの transaction-type に“Container”を指定 ・container-transaction タグの trans-attribute に“Required”を指定	異常時ループ対処用Destination名	○	
resourceadapter?				Message-driven Bean拡張情報
resourceadapter-name?	任意の文字列 注) 値を省略(空文字列を指定)することはできません。	リソースアダプタ名	○	
bean-pool-size?	以下の範囲の値 ・1～10000000 デフォルト値は8です。	初期起動インスタンス数	○	
runas-entry?				セキュリティアイデンティティ
userid	任意の文字列	ユーザID	○	
password	任意の文字列	パスワード	○	

注1) 本バージョンでは定義しても無効なタグです。
注2) Session Beanで指定された場合のみ有効です。
注3) STATEFUL Session Beanで指定された場合のみ有効です。
注4) STATELESS Session Beanで指定された場合のみ有効です。
注5) Entity Beanで指定された場合のみ有効です。
注6) Message-driven Beanで指定された場合のみ有効です。
注7) CMP1.1で指定された場合のみ有効です。
注8) EJB2.1に準拠したEJBアプリケーションの場合には、ejb-jarタグ配下の定義の移出入ができません。配備前に、開発環境においてejb-jar.xmlファイルを編集してください。

fujitsu-cmp-definitionタグの構成と内容

表16.8 fujitsu-cmp-definitionタグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
fujitsu-cmp-definition? (注1)				

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
datasource-name?	任意の文字列	データソース名	○	CMFマッピング定義
schema-name?	任意の文字列	スキーマ名	○	
table-name?	任意の文字列	テーブル名	○	
select-for-update?	以下の値から選択 ・ False :デフォルト値 ・ True	findByPrimaryKeyメソッドにFOR UPDATE句を付加	○	finderメソッド定義
stream-data?	以下の値から選択 ・ False :デフォルト値 ・ True	CMPデータのstream転送	○	CMFマッピング定義
field-map+				
field-map-entry+				
field-name	任意の文字列 export時に、定義ファイルとEnterprise Bean定義情報の間で、field-nameの整合性がとれていない場合、エラーとなります。	フィールド名	×	
field-type?	任意の文字列	型	×	
dbcolumn-name	任意の文字列	DBカラム名	○	
finder-map+				finderメソッド定義
finder-map-entry+				
finder-key-name	任意の文字列 export時に、定義ファイルとEnterprise Bean定義情報の間で、finder-key-nameの整合性がとれていない場合、エラーとなります。	メソッドシグネチャ	×	
finder-query-string	任意の文字列	検索条件	○	

注1) fujitsu-cmp2x-mapping-definitionとは排他指定です。また、CMP1.1で指定された場合のみ有効です。

fujitsu-cmp2x-mapping-definitionタグの構成と内容

表16.9 fujitsu-cmp2x-mapping-definitionタグ

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
fujitsu-cmp2x-mapping-definition? (注1)				
datasource-name?	任意の文字列	データソース名	○	CMFマッピング定義
ejb?				
ejb-name	任意の文字列	Enterprise Bean名	×	
schema-name	任意の文字列 注) 値を省略(空文字列を指定)することはできません。	スキーマ名	○	
table-name	任意の文字列	テーブル名	○	
optimize-finders?	以下の値から選択 ・ False :デフォルト値 ・ True	複数件検索の高速化	○	
concurrency-mode?	以下の値から選択 ・ for-update ・ normal :デフォルト値	同時更新データの一貫性保証	○	
stream-data?	以下の値から選択 ・ False :デフォルト値 ・ True	CMPデータのstream転送	○	
field-map2x				
field-map-entry2x+				
default-dbcolumn-name	任意の文字列	内部フィールド名	×	
is-primary-key	以下の値から選択 ・ True ・ False	プライマリキー	×	
foreign-key?				CMRマッピング定義
foreign-ejb-name	任意の文字列	Enterprise Bean名	×	
field-name	任意の文字列	CMPフィールド名	×	CMFマッピング定義
dbcolumn-name	任意の文字列	DBカラム名	○	
join-object+				CMRマッピング定義

タグ名	値	意味	編集	Interstage管理コンソール画面との対応
join-name	任意の文字列	内部クラス名	×	
schema-name	任意の文字列	スキーマ名	○	
table-name	任意の文字列	テーブル名	○	
optimize-relationships?	以下の値から選択 ・ False :デフォルト値 ・ True	relationshipのgetアクセッサの高速化	○	
source				
ejb-name	任意の文字列	Enterprise Bean名	×	
cmr-field-name?	任意の文字列	CMRフィールド名	×	
sink				
ejb-name	任意の文字列	Enterprise Bean名	×	
cmr-field-name?	任意の文字列	CMRフィールド名	×	
field-map2x	ejbタグのfield-map2xを参照			

注1) fujitsu-cmp-definitionとは排他指定です。また、CMP2.0で指定された場合のみ有効です。

16.4 Enterprise Bean定義ファイルのサンプル

Enterprise Bean定義ファイルのサンプルを以下に示します。

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE ejbdef SYSTEM 'ejbdef.dtd'>

<ejbdef>
  <ejb-jar>
    <enterprise-beans>
      <entity>
        <ejb-name>EjbCmp11</ejb-name>
        <home>com.fujitsu.interstage.j2eesamples.ejb.cmp11.EjbCmp11Home</home>
        <remote>com.fujitsu.interstage.j2eesamples.ejb.cmp11.EjbCmp11Remote</remote>
        <ejb-class>com.fujitsu.interstage.j2eesamples.ejb.cmp11.EjbCmp11</ejb-class>
        <persistence-type>Container</persistence-type>
        <prim-key-class>com.fujitsu.interstage.j2eesamples.ejb.cmp11.EjbCmp11PrimaryKey</prim-key-class>
        <reentrant>False</reentrant>
        <cmp-version>1.x</cmp-version>
      </entity>
    </enterprise-beans>
    <assembly-descriptor>
      <container-transaction>
        <method>
          <ejb-name>EjbCmp11</ejb-name>
          <method-name>*</method-name>
        </method>
        <trans-attribute>Required</trans-attribute>
      </container-transaction>
    </assembly-descriptor>
  </ejb-jar>
```

```
  <fujitsu-bean-definition>
    <base>
      <jndi-name>EjbCmp11</jndi-name>
      <tran-timeout>0</tran-timeout>
      <local-mode>False</local-mode>
    </base>
    <impl>
      <impl-home-intfrep>RMI:com.fujitsu.interstage.j2eesamples.ejb.cmp11.EjbCmp11Home:0000000000000000</
impl-home-intfrep>
     <impl-remote-intfrep>RMI:com.fujitsu.interstage.j2eesamples.ejb.cmp11.EjbCmp11Remote:0000000000000000</
impl-remote-intfrep>
    </impl>
    <session-eb>
      <session-timeout>0</session-timeout>
      <session-idle-timeout>1800</session-idle-timeout>
      <max-ejbobject>1024</max-ejbobject>
      <initial-stateless-instance-count>0</initial-stateless-instance-count>
    </session-eb>
    <entity-eb>
      <entity-timeout>120</entity-timeout>
      <entity-instance-type>ReadWrite</entity-instance-type>
      <entity-instance-size>100</entity-instance-size>
      <entity-instance-create-type>As Required</entity-instance-create-type>
      <entity-batch-operations>True</entity-batch-operations>
    </entity-eb>
  </fujitsu-bean-definition>
  <fujitsu-cmp-definition>
    <datasource-name>EjbCmp11</datasource-name>
    <schema-name>EJB</schema-name>
    <table-name>EMPLOYEE</table-name>
    <select-for-update>False</select-for-update>
    <field-map>
      <field-map-entry>
        <field-name>ID</field-name>
        <field-type>int</field-type>
        <dbcolumn-name>ID</dbcolumn-name>
      </field-map-entry>
      <field-map-entry>
        <field-name>NAME</field-name>
        <field-type>String</field-type>
        <dbcolumn-name>NAME</dbcolumn-name>
      </field-map-entry>
      <field-map-entry>
        <field-name>ADDRESS</field-name>
        <field-type>String</field-type>
        <dbcolumn-name>ADDRESS</dbcolumn-name>
      </field-map-entry>
      <field-map-entry>
        <field-name>TELEPHONE</field-name>
        <field-type>String</field-type>
        <dbcolumn-name>TELEPHONE</dbcolumn-name>
      </field-map-entry>
    </field-map>
    <finder-map>
      <finder-map-entry>
        <finder-key-name>findALL()</finder-key-name>
        <finder-query-string></finder-query-string>
      </finder-map-entry>
    </finder-map>
  </fujitsu-cmp-definition>
</ejbdef>
```

第4部 Webサービス編

第17章　Interstage Webサービスの機能

本章では、Interstage Webサービスの機能について説明します。

Webサービスとは、任意の方法で実現されたサーバプログラム(Webサービスアプリケーション)を、インターネットの標準技術を利用して、ネットワーク経由でアクセスできるように構築したシステムのことです。
通信プロトコルにはSOAPを用い、利用(アクセス)するためのインタフェース情報の記述にはWSDLを用います。

SOAP

W3C(World Wide Web Consortium)で規定された、XMLベースのプログラム連携用プロトコルです。プラットフォームに依存せず、柔軟性・拡張性に優れています。

WSDL

W3C(World Wide Web Consortium)で規定された、Webサービスのインタフェース情報を定義するためのXMLベースの記述言語、または、その言語で記述されたインタフェース情報です。Webサービスごとに、固有のアクセスする際のSOAP上のデータ形式や、アクセスのためのアドレス情報などが記述されています。

ポイント

WSDLは、インポート機能により複数のファイルで構成することができます。本製品では、注意事項などの記載がない場合は、インポート先も含めて1つのWSDLと呼んでいます。

17.1 Webサービスの標準規約

Webサービスの開発・運用に関連する標準規約を以下に示します。

Web Services for J2EE 1.1

Webサービスアプリケーション、およびWebサービスクライアントで利用するAPI/deployment descriptor/モジュールの構成物など、J2EEにおけるWebサービスのアプリケーション作成方法や配備処理について規定したものです。
本規約で、WebアプリケーションやEJBアプリケーションと同様に、Webサービスアプリケーションを標準に従った方法で開発可能になり、また、同じ仕様に準拠した製品への配備ができます。

Webサービスアプリケーションの実装を、Webサービスエンドポイントと呼びます。

APIには、JAX-RPC1.1およびSAAJ1.2を使用します。これら個々のWebサービスアプリケーションの実装と、それに対してアクセスするアドレス(URL)を定義したものをポートと呼びます。1つのWebサービスは1つ以上のポートから成り立ちます。

Webサービスクライアントは、J2EEアプリケーションクライアントとしても動作します。

注意

Interstage Application Serverでは、以下の機能をサポートしています。

1. ServletコンテナまたはEJBコンテナ上でWebサービスエンドポイントを、標準の方法でWebサービスとして提供できます。
2. ServletコンテナまたはEJBコンテナ上のアプリケーションやJ2EEアプリケーションクライアントがWebサービスクライアントとなり、Webサービスを利用できます。

SOAP Messages with Attachments

SOAPを使って添付ファイルを送受信するためのメッセージの形式について定めた規約です。添付の仕組みとしてMIMEの仕様を利用しています。

JAX-RPC 1.1

Webサービスアプリケーション、およびWebサービスクライアントを実装するためのJavaの標準APIです。Ver.1.1からWS-I Basic Profile 1.0に対応しています。

JAX-RPC 1.1の詳細については、JAX-RPCのWebサイトを参照してください。

- https://jcp.org/en/jsr/summary?id=101

注意

Interstage Application Serverでは、Enumerationをサポートしていません。

SAAJ 1.2

Webサービスで送受されるSOAPのメッセージ内容を、XMLレベルで参照/更新するJavaの標準APIです。SOAPの添付ファイルの参照/更新も可能です。Ver.1.2からXML部分をDOM Level2としても利用できます。

SAAJ 1.2の詳細については、SAAJのWebサイトを参照してください。

- https://jcp.org/aboutJava/communityprocess/mrel/jsr067/index2.html

SAAJ-APIの使用方法については“付録C SOAPメッセージの低レベル処理”を参照してください。

WS-I Basic Profile 1.0

Webサービスの標準仕様であるSOAP/WSDL/UDDI、および同時に用いられるHTTPなどの仕様について、Webサービスとしての相互接続性向上の観点から以下のように詳細化・制約付加を規定した規約です。

- 仕様上曖昧な点の明確化
- 仕様上許されるが使用すべきでない機能の規定(使用すべき範囲の限定)

詳細は、“18.5 WS-I Basic ProfileおよびAttachments Profileに準拠した開発”を参照してください。

注意

Interstage Application ServerによるWS-I Basic Profile 1.0準拠の範囲にUDDIは含まれていません。

WS-I Attachments Profile 1.0

添付ファイルを利用した場合のWebサービスの相互接続性向上のために、SOAP、WSDL、等の仕様について、仕様の明確化や使用範囲の限定を行った規約です。SOAP Messages with Attachmentsをベースに、仕様の明確化や使用範囲の限定、WSDL定義を定めています。
WebサービスをAttachments Profileに準拠させることにより、添付ファイル付きSOAPメッセージを送受信する際のシステム間での相互運用性が向上します。Interstage Application Serverでは、添付ファイルを利用するWebサービスの開発において、“18.5 WS-I Basic ProfileおよびAttachments Profileに準拠した開発”に従うことで、WebサービスをAttachments Profileにも準拠させることができます。

17.2 Interstage Webサービスの基本機能

Interstage Webサービスでは、以下の機能を提供しています。

Webサービスサーバ(Webサービスを提供する)機能

JAX－RPCに準拠したWebサービスアプリケーションをIJServerに配備することで、SOAPを利用して呼出し可能なWebサービスを構築できます。提供するWebサービスのインタフェース情報としてWSDLも生成します。

Webサービスクライアント(Webサービスを利用する)機能

SOAPを利用してWebサービスを利用する(呼び出す)、JAX-RPCに準拠したWebサービスクライアントアプリケーションを開発・運用できます。

Webサービス開発コマンド(iswsgen)

サービスエンドポイントインタフェースやWSDLファイルから、Webサービスアプリケーションおよびサービスクライアントの開発資産(Serviceインタフェース、実装クラス、および定義ファイル)を自動生成します。
ファイルなどが自動生成されることで、利用者は効率的な開発ができます。

17.3 Webサービスの実行環境

Webサービスの実行環境は以下です。

- IJServer上でWebサービスアプリケーションを配備して動作させることができます。
- IJServerおよびJ2EEアプリケーションクライアントの実行環境で、Webサービスクライアントを動作させることができます。

Webサービスアプリケーションの配備については、“3.5 J2EEアプリケーションの配備と設定”を参照してください。

17.3.1 Webサービスアプリケーションの実行環境

Webサービスを提供する側(サーバ側)の実行環境は、以下です。

IJServer環境

- Interstage管理コンソール、コマンドでWebサービスアプリケーションの配備/配備解除ができます。
- Interstage管理コンソール、コマンドでWebサービスアプリケーションの運用ができます。
  - Webサービスアプリケーションのモニタリング(Interstage管理コンソールのみ)
  - Webサービスアプリケーションの活性化・非活性化
- WS-I Basic Profile 1.0に準拠しています。
- WS-I Attachments Profile 1.0 に準拠しています。

注意

- IJServerのタイプは“WebアプリケーションとEJBアプリケーションを同一JavaVMで運用”だけとなります。
- 使用できるXMLパーサは、JAXP1.2以上をサポートしているものだけです。

17.3.2 Webサービスクライアントの実行環境

Webサービスを呼び出す側(クライアント側)の実行環境は、以下です。

- IJServer環境
- J2EEアプリケーションクライアント実行環境

環境設定について詳細は、“19.2 Webサービス(クライアント機能)の運用方法”を参照してください。

注意

- Javaバージョン1.3以前の環境では動作しません。
- XMLパーサは、JAXP1.2以上をサポートしているものを使用してください。

第18章　Webサービスの開発

本章では、WebサービスアプリケーションおよびWebサービスクライアントアプリケーションの開発について説明します。

Webサービスアプリケーション/Webサービスクライアントアプリケーションの開発

WebサービスとWebサービスクライアントの通信は、WSDLファイルに定義されたインタフェース情報に基づいて行われます。

WSDLファイルは、Webサービスアプリケーションの開発で作成されます。Webサービスクライアントアプリケーションは、このWSDLファイルを使用してスタブなど生成し、これを利用するアプリケーションとして開発します。
Webサービスアプリケーションの開発とWebサービスクライアントアプリケーションの開発は、WSDLファイルを介して、独立して行うことができます。

ツールによるファイル生成

WebサービスアプリケーションとWebサービスクライアントアプリケーションの開発では、iswsgenコマンドを使用して、WSDLやスタブなどのファイル生成を行います。
iswsgenコマンドについて詳細は、“リファレンスマニュアル(コマンド編)”の“iswsgen”を参照してください。

開発用途に応じて、iswsgenコマンドの以下のサブコマンドを使用します。

- Webサービスアプリケーションの開発 － wsdlサブコマンド
  iswsgenコマンドにwsdlサブコマンドを指定して実行します。
  サービスエンドポイントインタフェースから、WSDLファイルおよびWebサービスアプリケーション開発に必要なファイルが生成されます。
- Webサービスクライアントアプリケーションの開発 － clientサブコマンド
  iswsgenコマンドにclientサブコマンドを指定して実行します。
  WSDLファイルから、Webサービスクライアント開発に必要なファイルが生成されます。

当社のコンポーネント指向の統合開発支援ツールであるInterstage Studioを使用することにより、一連の手順を統合した使いやすいビュー操作によって、操作できます。

詳細は、Interstage Studioのマニュアルを参照してください。

WS-I Basic Profileへの準拠

本製品では、WS-I Basic Profile1.0およびWS-I Attachments Profile1.0に準拠したWebサービスを開発・運用できます。本規約に準拠したWebサービスの開発については、“18.5 WS-I Basic ProfileおよびAttachments Profileに準拠した開発”を参照してください。

注意

- “添付ファイル”を除いた分の通信データサイズが大きい場合、受信時に大量にメモリを消費し、メモリ不足や処理時間超過を招く恐れがあります。
- 目安として、通信データサイズが数100Kバイト(XMLでのタグなどを含む)を超える場合、“添付ファイル”を使用することを検討してください。
- ただし、通信データサイズが同じでもメモリ消費量は内容形式により大きく異なり、また、同時処理多重度や性能要件もシステムにより異なります。上記サイズを目安としつつ、条件に応じて、実際の業務用のデータ内容形式を使用して要件に対する事前検証を行うことをお勧めします。

18.1 Webサービス(サーバ機能)の開発

ここでは、Webサービス(サーバ機能)を開発する時の開発手順や操作について説明します。

Webサービスを提供するには、WebサービスアプリケーションのWARファイルもしくはSTATELESS Session Beanを含むejb-jarファイルを開発し、実行環境に配備します。

Webサービスアプリケーションの開発では、以下の資材を開発し、WARファイルもしくはejb-jarファイルへのパッケージングを行います。

- サービスエンドポイントインタフェースおよび、Webサービスエンドポイント
  JAX-RPCに準拠したJavaアプリケーション資材です。サービスエンドポイントインタフェースはWebサービスとして提供するインタフェースを定義するJavaインタフェース、Webサービスエンドポイントは Webサービスの実際の処理ロジックを実装したJavaクラスです。
  STATELESS Session Beanの場合、Webサービスエンドポイントは Enterprise Beanクラスです。
- WSDLファイル
  WebサービスをSOAP通信で呼び出すためインタフェース情報が定義されたファイルです。
- deployment descriptor
  Webサービス/Webサービスアプリケーションの各種定義を記述するファイルです。

以下に、Webサービスを提供する場合の手順と主な生成物について説明します。

Webサービス開発の流れ

1. Webサービスアプリケーションを開発する
   Webサービスとして提供するインタフェースを定義するJavaインタフェース(サービスエンドポイントインタフェース)を作成し、WSDLファイルを生成します。また、Webサービスの実際の処理ロジックを実装したJavaクラス(Webサービスエンドポイント)を作成します。
2. deployment descriptorを編集する
   deployment descriptorを編集します。
3. WARファイルもしくはejb-jarファイル/EARファイルへパッケージングする
   jarコマンドを使用し、パッケージングします。
4. 配備する
   IJServerへ配備します。
5. アプリケーションのデバッグ
   アプリケーションを修正した時は、1～5の手順を繰り返します。

Interstage Studioを使用する場合は、1～5までの操作をすべてInterstage Studioで行えます。

18.1.1 WebサービスアプリケーションのWAR/ejb-jarファイルの構成

WebサービスアプリケーションにWebサービスエンドポイントクラスを使用する場合に、パッケージングするモジュールの構成例を下記に示します。

WARファイルの場合

```
------------------------------------------------------------
WARファイル
 |__ JSPファイル
 |__ HTMLファイル
 |__ ・・・
 |__ WEB-INF
     |__ classes            ※
         |__ クラスファイル ※
         |__ ・・・         ※
     |__ lib                ※
         |__ JARファイル    ※
         |__ ・・・         ※
     |__ web.xml
     |__ webservices.xml
     |__ wsdl
         |__ WSDLファイル
     |__ <WSDLファイル名>_mapping.xml
------------------------------------------------------------
```

注意

- Webサービスアプリケーションの Java資材は、上記の“※”の位置に格納してください。
- webservices.xml、<WSDLファイル名>_mapping.xmlは、“WEB-INF”配下に格納してください。
- WSDLファイル、およびWSDLファイルがインポートしているスキーマファイルなどは、“WEB-INF/wsdl”配下に格納してください。
- 上記以外のファイルについては、“7.1 Webアプリケーションのディレクトリ構成”を参照してください。
- “WEB-INF/wsdl”配下のファイルは、公開用WSDLに含められます。公開用WSDLについては、“公開用WSDLの取得”を参照してください。

ejb-jarファイルの場合

```
------------------------------------------------------------
ejb-jarファイル
 |__ クラスファイル
 |__ META-INF
     |__ ejb-jar.xml
     |__ webservices.xml
     |__ wsdl
         |__ WSDLファイル
     |__ <WSDLファイル名>_mapping.xml
------------------------------------------------------------
```

注意

- webservices.xml、<WSDLファイル名>_mapping.xmlは、“META-INF”配下に格納してください。
- WSDLファイル、およびWSDLファイルがインポートしているスキーマファイルなどは、“META-INF/wsdl”配下に格納してください。
- 上記以外のファイルについては、“11.3 EJBアプリケーションの開発”を参照してください。
- “META-INF/wsdl”配下のファイルは、公開用WSDLに含められます。公開用WSDLについては、“公開用WSDLの取得”を参照してください。

18.1.2 Webサービスアプリケーションを開発する

以下の手順で、Webサービスアプリケーションを作成してください。

1 Webサービスのインタフェースを定義する

1) サービスエンドポイントインタフェースの定義

サービスエンドポイントインタフェースは、Webサービスとして提供するWebサービスアプリケーションの公開インタフェースを定義する、Javaインタフェースです。

Webサービスアプリケーションのインタフェースとして、以下の条件を満たすJavaインタフェースを作成します。
このJavaインタフェースを作成することで、どのようなメソッドをWebサービスとして提供するかを定義します。

- java.rmi.Remoteインタフェースを継承している。
- メソッドはjava.rmi.RemoteExceptionをthrowする。
- メソッドのパラメタ(引数・返り値)には、“18.3 Javaのデータ型とXMLのデータ型との対応”に示されたJavaデータ型だけを使用する。
- メソッドのオーバーロード(メソッド名が同じで引数の数やデータ型のみ異なる複数のメソッドの定義)をしない。
- メソッドのパラメタ(引数・返り値)にEJBObjectやEJBLocalObjectを含まない。

ー メソッドのパラメタ(引数・返り値)に使用する配列型のデータや、構造体型・Bean型のメンバにEJBObjectやEJBLocalObject、EJBアプリケーションのLocalインタフェース、Remoteインタフェース、Local Homeインタフェース、Remote Homeインタフェース、Timerインタフェース、TimerHandleインタフェース、CMPのfinderメソッドの復帰値のCollectionを含まない。

また、WebサービスとRMIoverIIOPの双方から呼び出されるSTATELESS Session Beanを作成する場合には、Remoteインタフェース/Homeインタフェースとサービスエンドポイントインタフェースの両方を作成してください。
Webサービスからのみ呼び出されるSTATELESS Session Beanの場合には、サービスエンドポイントインタフェースのみ用意する(Remoteインタフェースなどを用意しない)ことも可能です。

Webサービスとして Stringを引数に取りfloatを返す、getLastTradePriceというメソッドの例です。

```
package com.example;
public interface StockQuoteProvider extends java.rmi.Remote {
     float getLastTradePrice (String tickerSymbol) throws java.rmi.RemoteException;
}
```

2) WSDLの生成

iswsgen wsdlコマンドを使用してWSDLを生成します。“インタフェース名”には、1) サービスエンドポイントインタフェースの定義で作成したサービスエンドポイントインタフェースのクラス名を指定します。
iswsgenコマンドの詳細については、“リファレンスマニュアル(コマンド編)”のiswsgenコマンドを参照してください。

```
iswsgen wsdl [オプション] "インタフェース名"
```

コマンド実行後、以下のファイルが生成されます。

ー WSDLファイル
Webサービスのインタフェース情報が定義されています。ただし、接続先のURL情報はデフォルトでは、仮の値が記述されています。Webサービスアプリケーションを IJServerに配備したあと、正しいURL情報が記述されたWSDLファイルを取得できます。

ー <WSDLファイル名>_mapping.xml
最後にパッケージングする時に、アプリケーションとともに含めます(プロトコル(SOAP)で使用するXMLとアプリケーションの対応付けが記述されています)。

WARファイルの場合

```
iswsgen wsdl com.example.StockQuoteProvider
```

ejb-jarファイルの場合

```
iswsgen wsdl -module ejb com.example.StockQuoteProvider
```

2 Webサービスエンドポイントを実装する

Webサービスエンドポイントは、Webサービスとして提供するWebサービスアプリケーションの処理ロジックを実装したJavaクラスです。
STATELESS Session Beanの場合、WebサービスエンドポイントはEnterprise Beanクラスです。

Webサービスアプリケーションの処理ロジックとして、以下の条件を満たすJavaクラスを作成します。Webサービスが呼び出されたときに、このJavaクラスのオブジェクトに対して、呼び出しに対応するメソッドが呼び出されます。

- publicだが、finalおよびabstractでないクラス

- publicデフォルトコンストラクタを持つ
- finalize()メソッドを定義していない
- WARファイルの場合、作成したサービスエンドポイントインタフェースを実装(implements)している
- ejb-jarファイルの場合、サービスエンドポイントインタフェースの各メソッドの以下を満たすビジネスメソッド(注)がEnterprise Beanクラスに定義されている
  - 同名
  - 同数・同型の引数と復帰値である
  - サービスエンドポイントインタフェースと同一の例外をスローする

  注) Enterprise Beanクラスに実装するビジネスメソッドについては“12.6 Enterprise Beanクラスの作成”の“ビジネスメソッドの規約”を参照してください。

初期化と後処理のカスタマイズが必要な場合

WARファイルの場合、Webサービスエンドポイントでインタフェースを実装することで、起動時の初期化・終了時の後処理が定義できます。

```
package javax.xml.rpc.server;
public interface ServiceLifecycle {
  public void init(Object context) throws javax.xml.rpc.ServiceException;
  public void destroy();
}
```

JAX-RPC Service Endpointのインスタンス化時にinitメソッドが呼び出され、インスタンスが解放されるときにdestroyメソッドが呼び出されます。
initメソッドのパラメタには、下記のServletEndpointContextオブジェクトが渡されます。

```
package javax.xml.rpc.server;
public interface ServletEndpointContext {
  public java.security.Principal getUserPrincipal();
  public javax.xml.rpc.handler.MessageContext getMessageContext();
  public javax.servlet.http.HttpSession getHttpSession();
  public javax.servlet.ServletContext getServletContext();
}
```

ポイント

サービスエンドポイントインタフェースおよびWebサービスエンドポイントをコンパイルする際は、クラスパスおよびJava VMオプションに以下を設定してください。

クラスパス

Windows32/64

C:¥Interstage¥J2EE¥lib¥isws.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isws.jar

JavaVMオプション

Windows32/64

-J-XX:EndorsedClassPath=C:¥Interstage¥J2EE¥lib¥isws-saaj-api.jar

Solaris64 Linux32/64

-J-XX:EndorsedClassPath=/opt/FJSVj2ee/lib/isws-saaj-api.jar

添付ファイルを利用する場合や初期化と後処理のカスタマイズを行う場合には、上記に加えて以下もクラスパスに追加してください。

Windows32/64

C:¥Interstage¥J2EE¥lib¥isj2ee.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isj2ee.jar

18.1.3 deployment descriptorを編集する

以下のdeployment descriptorを記述します。

- webservices.xml
- web.xml (WARファイルの場合)
- ejb-jar.xml (ejb-jarファイルの場合)

webservices.xmlについては、“18.6.1 webservices.xmlの記述形式”、“18.6.2 webservices.xmlのタグ”を参照してください。
web.xmlについては、“18.6.3 web.xml”を参照してください。

ejb-jar.xmlでは以下のようにdeployment descriptorファイルに作成したサービスエンドポイントインタフェース名を定義してください。

例

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar version="2.1" xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
 http://java.sun.com/xml/ns/j2ee/ejb-jar_2_1.xsd">

  <display-name>BankJAR</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>CreditCardEndpointBean</ejb-name>
      <home>com.fujitsu.j2ee.bank.creditcardservice.StatelessHome</home>
      <remote>com.fujitsu.j2ee.bank.creditcardservice.StatelessRemote</remote>
      <service-endpoint>
        com.fujitsu.j2ee.bank.creditcardservice.CreditCardIntf
      </service-endpoint>
      <ejb-class>
        com.fujitsu.j2ee.bank.creditcardservice.CreditCardEndpointBean
      </ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>

  <assembly-descriptor>
    <container-transaction>
      <method>
        <ejb-name>CreditCardEndpointBean</ejb-name>
        <method-intf>ServiceEndpoint</method-intf>
        <method-name>validateCreditCard</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
    </container-transaction>
  </assembly-descriptor>

</ejb-jar>
```

18.1.4 WARファイルもしくはejb-jarファイル/EARファイルへパッケージングする

“18.1.3 deployment descriptorを編集する”でwebservices.xmlに記述したパスの構成で、WARファイルもしくはejb-jarファイル/EARファイルにパッケージングします。従来のWARと同様に、jarコマンドなどでパッケージングします。

WARファイルの構成については、“7.1 Webアプリケーションのディレクトリ構成”を参照してください。
ejb-jarファイルの構成については、“11.3 EJBアプリケーションの開発”を参照してください。また、Webサービス対応したejb-jarファイルは必ずEARファイルでパッケージングしてください。Webサービス対応したejb-jarファイルを単体で配備した場合には配備時にエラーが発生します。

例

jar cvf StockService.war .

注意

通常、Webサービスアプリケーションのクラス、サービスエンドポイントインタフェースのクラス、および、引数・返り値で使用するクラス(Bean/構造体の場合はそのメンバのクラスを含む)は、以下に格納してください。

- WARファイルの場合
  WEB-INF/classes配下にクラスを格納、または、WEB-INF/lib配下にJarファイルとして格納
- ejb-jarファイルの場合
  ejb-jarファイル直下にクラスを格納

これらのクラスを、WARファイルの場合にはWebappクラスローダ以外、ejb-jarファイルの場合にはApplicationクラスローダ以外のクラスローダでロードさせる場合、Webサービスアプリケーションに対してHotDeploy機能を使用しないでください。
IJServerのクラスローダの構成については、“2.3.1 クラスローダの構成”を参照してください。

18.1.5 HTTP接続に関する設定

セション管理の利用

WARファイルの場合、セション管理を行うことで、同じWebサービスクライアントアプリケーションからの複数回のリクエストの処理で、前回の処理結果を引き継ぐことが可能となり、継続的な処理が行えます。セション管理は、Servlet APIであるjavax.servlet.http.HttpSessionオブジェクトを利用します。ServletEndpointContextオブジェクトのgetHttpSessionメソッドで取得できます。

ServletEndpointContextオブジェクトの取得方法については、“初期化と後処理のカスタマイズが必要な場合”を参照してください。

例

```
.....

public class SamplePortImpl implements Sample, ServiceLifecycle {

    private ServletEndpointContext context;

    public void init(Object context) throws ServiceException {
        this.context = (ServletEndpointContext) context;
    }

    public String sayHello(String name) {
        HttpSession session = context.getHttpSession();
        .....
    }
```

. . . . .

セションタイムアウトの設定

セション管理のタイムアウト時間は30分(省略値)です。タイムアウト時間を変更する場合は、Webアプリケーション(web.xml)ファイルを編集してください。
詳細は“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”を参照してください。

HTTP Basic認証の利用

Webサービスの呼出しに対してHTTP Basic認証を設定することができます。設定する場合はWebアプリケーション(web.xml)ファイルを編集してください。詳細は“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”を参照してください。

注意

- セション管理は、Webサービスクライアントが HTTP Cookieを利用したセション管理に対応している場合に、有効です。
- WebサービスがHTTP Cookieを利用したセション管理を行っているかどうかは、WSDLの情報には含まれません。Webサービスクライアント側と、HTTP Cookieを利用したセション管理を行うかどうかについて任意の方法で取り決めを行ってください。
- WebサービスのHTTP Basic認証については、WSDLの情報には含まれません。Webサービスクライアント側と、HTTP Basic認証について任意の方法で取り決めを行ってください。

18.1.6 Webサービスのインタフェース情報を提供する

IJServerにWebサービスアプリケーションを配備すると、そのWebサービスの公開用WSDLを取得することができます。必要に応じて、このWSDLをWebサービスの利用者に任意の方法で提供します。

公開用WSDLの取得については、“公開用WSDLの取得”を参照してください。

18.2 Webサービスを呼び出す場合(クライアント機能)の開発

ここでは、Webサービスクライアント(クライアント機能)を開発する時の開発手順や操作について説明します。

Webサービスを呼び出すには、利用する(呼び出す)WebサービスのWSDLからWebサービス呼び出し機能を持つスタブを生成して、このスタブのメソッドを呼び出すアプリケーションを開発、実行します。

クライアントアプリケーション開発の流れ

クライアントアプリケーションの開発の流れは、以下のアプリケーションで差異があります。

- IJServer上で動作するアプリケーション(Servletなど)
- 上記以外のアプリケーション

アプリケーションごとの手順は、以下のとおりです。

共通の手順

1.Webサービスのインタフェース情報を入手する

利用するWebサービスのWSDLを入手します。

2.スタブを生成する

iswsgen client コマンドを使用して、インタフェース情報(WSDL)から開発時に必要なファイルを生成します。

3.Webサービスクライアントアプリケーションを開発する

JAX-RPCのAPIを使用して、クライアントアプリケーションを実装します。

IJServer上で動作するアプリケーション

4.WARファイルもしくはejb-jarファイル/EARファイルへパッケージングする

共通の手順で生成したファイルをパッケージングします。

WARファイルの構成については、“18.1.1 WebサービスアプリケーションのWAR/ejb-jarファイルの構成”を参照してください。

5.配備する

IJServerへ配備します。

6.アプリケーションのデバッグ

アプリケーションを修正した時は、1～5の操作を繰り返します。

上記以外のアプリケーション

4.アプリケーションのデバッグ

アプリケーションを修正した時は、1～3の操作を繰り返します。

18.2.1 Webサービスのインタフェース情報を入手する

利用するWebサービスのWSDLを入手します。入手方法は、Webサービスごとに任意です。Webサービスとセットで開発している場合は、“18.1.6 Webサービスのインタフェース情報を提供する”を参照してください。

18.2.2 スタブを生成する

ファイルの生成

iswsgen client コマンドを使用して、インタフェース情報(WSDL)から必要ファイルを生成します。
iswsgenコマンドの詳細については、“リファレンスマニュアル(コマンド編)”のiswsgenコマンドを参照してください。

```
iswsgen client [オプション] "WSDLファイル"
```

コマンド実行後、以下のファイルが生成されます。

サービスエンドポイントインタフェースソース

Webサービスとして提供するWebサービスアプリケーションの公開インタフェースを定義するJavaインタフェースのソースです。
java.rmi.Remoteインタフェースを継承しています。
インタフェース名はWSDLのPortType要素の名前が使用されます。

Serviceインタフェースソース

WSDLファイルに記述されたWebサービスを表すJavaインタフェースのソースです。Webサービスアプリケーションのスタブを取得できます。
javax.xml.rpc.Serviceインタフェースを継承しています。
インタフェース名はWSDLのService要素の名前が使用されます。

ユーザ定義型クラスソース

WSDLのoperationのパラメタとしてユーザ定義型が指定されている場合に、対応するJavaクラスソースを生成します。
クラス名はユーザ定義型の型名が使用されます。

スタブ/サービス実装

サービスエンドポイントインタフェースを実装したクラスおよびServiceインタフェースを実装したクラスのソースです。クラス名はすべて“_isws_”で始まります。

生成されたJavaソースファイルのコンパイル

生成されたJavaソースファイルをすべてコンパイルします。コンパイルする際は、クラスパスおよびJava VMオプションに以下を設定してください。

クラスパス

Windows32/64

C:¥Interstage¥J2EE¥lib¥isws.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isws.jar

Java VMオプション

Windows32/64

-J-XX:EndorsedClassPath=C:¥Interstage¥J2EE¥lib¥isws-saaj-api.jar

Solaris64 Linux32/64

-J-XX:EndorsedClassPath=/opt/FJSVj2ee/lib/isws-saaj-api.jar

クライアントアプリケーションから添付ファイルを利用するWebサービスを呼び出す場合は、上記に加えて以下もクラスパスに追加してください。

Windows32/64

C:¥Interstage¥J2EE¥lib¥isj2ee.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isj2ee.jar

iswsgen clientコマンドで生成されるサービスエンドポイントインタフェースやユーザ定義型クラスは、Webサービスアプリケーションのものと異なる場合があります。
クライアントアプリケーションの開発では、必ずiswsgen clientコマンドで生成したものを使用してください。

18.2.3 Webサービスクライアントアプリケーションを開発する

Webサービスクライアントは、WSDLから生成したスタブを利用してWebサービスを呼び出す処理を行う任意のクラスです。
WebサービスクライアントがWebサービスを呼び出す方法は、以下の二通りあります。

- JAX-RPCのServiceFactoryクラスを使用してServiceオブジェクトを取得する方法
- JNDIを使用してServiceオブジェクトをlookupする方法

Webサービスを呼び出す処理では、以下のオブジェクトを利用します。

JAX-RPCで提供されるjavax.xml.rpc.ServiceFactoryオブジェクト

ServiceFactoryクラスを使用する場合に利用します。
Serviceオブジェクトの生成を行います。

javax.naming.InitialContextオブジェクト

JNDIを使用する場合に利用します。
Interstageが提供するJNDIサービスプロバイダを利用し、Serviceオブジェクトを取得します。

WSDLから生成された、ServiceインタフェースおよびServiceオブジェクト

呼び出すWebサービス全体に対応するクラスです。Serviceオブジェクトは、Serviceインタフェースをimplementsし、スタブオブジェクトの生成機能を備えています。WSDLのtargetNamespaceに基づいたパッケージ名と、WSDLのserviceのnameに基づいたクラス名でアクセスします。

WSDLから生成された、サービスエンドポイントインタフェースおよびスタブオブジェクト

呼び出すWebサービスアプリケーションに対応するクラスです。スタブオブジェクトは、サービスエンドポイントインタフェースをimplementsし、Webサービスの呼び出し機能を備えています。WSDLのtargetNamespaceに基づいたパッケージ名と、WSDLのportTypeのnameに基づいたクラス名でアクセスします。

JAX-RPCのServiceFactoryクラスを使用してServiceオブジェクトを取得する方法

ServiceFactoryクラスを使用してServiceオブジェクトを取得する方法は任意のJavaアプリケーションで利用可能です。この場合、以下の手順でWebサービスを呼び出します。

1. ServiceFactoryオブジェクトを生成する
2. ServiceFactoryオブジェクトからServiceオブジェクトを取得する
3. Serviceオブジェクトからスタブオブジェクトを取得する
4. スタブオブジェクトのメソッドを呼び出す

 例

Serviceインタフェースのクラス名が“StockQuoteProviderService”、サービスエンドポイントインタフェースのクラス名が“StockQuoteProvider”、呼び出すWebサービスのオペレーションの名前が“getLastTradePrice”の場合についての処理例です。

```
ServiceFactory sf = ServiceFactory.newInstance();//(1)
StockQuoteProviderService sqs =
(StockQuoteProviderService) sf.loadService(StockQuoteProviderService.class);//(2)
StockQuoteProvider sqp = sqs.getStockQuoteProviderPort();//(3)
float price = sqp.getLastTradePrice(tickerID);//(4)
```

JNDIを使用してServiceオブジェクトをlookupする場合

WebアプリケーションまたはEJBアプリケーション、J2EEアプリケーションクライアントからWebサービスを呼び出す場合、JNDIを使用してServiceオブジェクトをlookupすることができます。その他のJavaアプリケーションでServiceオブジェクトを取得する場合は、ServiceFactoryクラスを使用して取得してください。
JNDIを使用する場合は以下の手順でWebサービスを呼び出します。

1. InitialContextオブジェクトを生成する
2. InitialContextのlookupメソッドを使用し、Serviceオブジェクトを取得する
3. Serviceオブジェクトからスタブオブジェクトを取得する
4. スタブのメソッドを呼び出す

lookupの引数には、以下の文字列を指定します。

java:comp/env/[deployment descriptorの<service-ref-name>に指定した値]

deployment descriptorについては、“18.6.4 service reference記述”を参照してください。

 例

```
InitialContext ic = new InitialContext ();   //(1)
StockQuoteProviderService sqs =
 (StockQuoteProviderService)ic.lookup("java:comp/env/service/StockQuote");  //(2)
StockQuoteProvider sqp = sqs.getStockQuoteProviderPort();  //(3)
float price = sqp.getLastTradePrice(tickerID);  //(4)
```

注意

- JAX-RPCのServiceFactoryオブジェクトでは、以下のメソッドを使用してください。

  ー loadService(java.lang.Class class1)

  以下のメソッドは使用できません。

  ー createService(QName serviceName)

  ー createService(java.net.URL wsdlDocumentLocation, QName serviceName)

  ー loadService(java.net.URL url, java.lang.Class class1,java.util.Properties properties)

  ー loadService(java.net.URL url, QName qname, java.util.Properties properties)

- Serviceオブジェクトでは、呼び出すWebサービスに対応してWSDLに基づいて生成された、対象のWebサービス固有のメソッドを使用してください。JAX-RPC APIで提供されるjavax.xml.rpc.Serviceインタフェースに定義されている汎用メソッドは使用できません。

- セション管理を利用している場合、同じ接続先に対して新たなセションを 開始するときは、Serviceオブジェクトを新たに取得します。セション管理の利用については“18.2.5 HTTP接続に関する設定”を参照してください。

- Serviceオブジェクトの取得では、内部的に多くの処理が行われます。アプリケーションの性能向上のため、Serviceオブジェクトをリクエストごとに取得するのではなく、取得したServiceオブジェクトを保持して再利用することを推奨します。

- 単一のスタブオブジェクトは、複数のスレッドで同時に使用できません。スレッドの排他を行って利用する、または、リクエストごとに毎回取得するなどの対応が必要です。

- JNDIを利用してServiceオブジェクトをlookupする場合は、単一のInitialContextオブジェクトを複数のスレッドで使用することはできません。

- 通信異常などが発生した場合のため、これらの処理で発生する可能性のある例外については、これをすべてcatchし、すべての情報とスタックトレースをログなどに残る出力処理を行うことを推奨します。
  以下は標準出力・標準エラーがファイルに出力されている場合の例外処理の例です。

```
    } catch( javax.xml.rpc.ServiceException e ){
      e.printStackTrace();
      if( e.getLinkedCause() != null ){
        e.getLinkedCause().printStackTrace();
      }
    } catch( javax.xml.rpc.soap.SOAPFaultException e ){
      e.printStackTrace();
      System.out.println("  Code = " + e.getFaultCode());
      System.out.println("  String = " + e.getFaultString());
      System.out.println("  Actor = " + e.getFaultActor());
      if( e.getDetail() != null ){
        Detail detail = e.getDetail();
        Iterator it = detail.getDetailEntries();
        if( it != null ){
          while(it.hasNext()){
            SOAPElement elm = (SOAPElement)it.next();
            System.out.println("  Detail = " + elm);
          }
        }
      }
    } catch( javax.xml.rpc.JAXRPCException e ){
      e.printStackTrace();
      if( e.getLinkedCause() != null ){
        e.getLinkedCause().printStackTrace();
      }
    } catch( java.rmi.RemoteException e ){
      e.printStackTrace();
      if( e.detail != null ){
        e.detail.printStackTrace();
      }
```

```
    } catch( Throwable e ) {
      e.printStackTrace();
    }
```

- WebサービスとのHTTP通信においてエラーが発生したり、Webサービスとの通信に至る前に発生したエラーの場合、WebサービスからのFaultの返却はありませんが、例外のメッセージに“ISWSFault”や“faultCode:”、“faultString:”といった文字列が含まれる場合があります。

18.2.4 deployment descriptorを編集する

JNDIを使用してServiceオブジェクトを取得する場合、deployment descriptorの編集が必要です。
Webサービスクライアントアプリケーションのdeployment descriptorの編集についての詳細は“18.6.4 service reference記述”を参照してください。
ServiceFactoryを使用してServiceオブジェクトを取得する場合、この作業は不要です。

18.2.5 HTTP接続に関する設定

Webサービスクライアントアプリケーションで追加のHTTP接続情報を設定する方法について以下に説明します。

接続先URLを指定する方法

スタブオブジェクトはWSDLで定義されたデフォルトの接続先URLに対してリモート呼び出しを行います。明示的に接続先URLを指定したい場合は、スタブオブジェクトに対してプロパティで接続先のURLを指定します。
スタブオブジェクトにプロパティで指定するには、javax.xml.rpc.Stubインタフェースの_setPropertyメソッドを使用して、プロパティとして接続先URLを設定します。

```
import javax.xml.rpc.Stub;

      .....
  StockQuoteProviderService abf =
      (StockQuoteProviderService) sf.loadService(StockQuoteProviderService.class);
  StockQuoteProvider quoteProvider = abf.getStockQuoteProviderPort();
  ((Stub)quoteProvider)._setProperty("javax.xml.rpc.service.endpoint.address",
      "http://anotherhost/StockService/services/StockQuoteProvider");//接続先URLを指定

  float quote = quoteProvider.getLastTradePrice(tickerID);//指定されたURLに接続
```

接続先URLに関するプロパティは、以下のとおりです。

キー	値(java.lang.String)
javax.xml.rpc.service.endpoint.address	接続先のURL(デフォルトはWSDLで指定されたURL)

Webサービスのユーザ名/パスワードを設定する方法

WebサービスでHTTP Basic認証を行っている場合は、スタブオブジェクトに対してプロパティとして認証情報を設定します。
Webサービスのユーザ名/パスワードに関するプロパティは、以下のとおりです。

キー	値(java.lang.String)
javax.xml.rpc.security.auth.username	ユーザ名
javax.xml.rpc.security.auth.password	パスワード

応答タイムアウト時間を指定する方法

接続先から応答がない場合に接続を切断するタイムアウト時間を指定する場合は、スタブオブジェクトに対してプロパティとしてタイムアウト時間をミリ秒で設定します。
タイムアウト時間に関するプロパティは、以下のとおりです。

キー	値(java.lang.Integer)
com.fujitsu.interstage.isws.client.connect.timeout	タイムアウト時間(ミリ秒) (デフォルトは10分)

セション管理の利用を指定する方法

HTTP Cookieを利用したセション管理の利用を指定する場合は、スタブオブジェクトに対してプロパティとして「利用する・しない」をjava.lang.Booleanオブジェクトで設定します。
セション管理の利用に関するプロパティは、以下のとおりです。

キー	値(java.lang.Boolean)
javax.xml.rpc.session.maintain	Boolean.TRUE(利用する) Boolean.FALSE(利用しない＝デフォルト)

注意

- セション管理は、呼び出すWebサービスがHTTP Cookieを利用したセション管理を行っている場合に有効です。
- Webサービスクライアントアプリケーション終了時、セション管理情報を含むcookieは保存されません。
- セションのタイムアウトが発生した後に処理を継続した場合、新規のセションが確立されます。

ポイント

スタブオブジェクトのプロパティは、プログラムでの設定以外に外部ファイルでも設定できます。詳細については“19.2.2 スタブ設定ファイル”を参照してください。

プロキシに関する設定

“19.2.3 プロキシを経由した接続”を参照してください。

18.3 Javaのデータ型とXMLのデータ型との対応

ここでは、Webサービスアプリケーションおよびクライアントアプリケーションで使用するJavaデータ型と、対応するXMLのデータ型について説明します。

参考

本節で説明に使用しているXMLのネームスペースのprefixは、以下のとおりです。アプリケーションやコマンドの実行時には本節で説明するネームスペースに対して下記以外のprefixが使用されることがあります。

- “xsd”が表すネームスペースは“http://www.w3.org/2001/XMLSchema”です。
- “soapenc”が表すネームスペースは“http://schemas.xmlsoap.org/soap/encoding/”です。
- “wsdl”が表すネームスペースは“http://schemas.xmlsoap.org/wsdl/”です。
- “mime”が表すネームスペースは“http://schemas.xmlsoap.org/wsdl/mime/”です。
- “wsibp”が表すネームスペースは“http://ws-i.org/profiles/basic/1.1/xsd”です。
- “apachesoap”が表すネームスペースは“http://xml.apache.org/xml-soap”です。

18.3.1 単純型

Webサービスアプリケーションのパラメタ(引数、返り値)で使用できるデータ型と、それに対応してXMLで使用される型を以下に示します。

プリミティブ型

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型
int	xsd:int
short	xsd:short
long	xsd:long
byte	xsd:byte
float	xsd:float
double	xsd:double
boolean	xsd:boolean

参照型

単純型の参照型は、一部のJavaのデータ型について、XMLで使用されるデータ型が選択したスタイル方式によって異なります。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型	
	デフォルト(literal利用の場合)	encoded利用の場合
java.lang.Integer	xsd:int	soapenc:int
java.lang.Short	xsd:short	soapenc:short
java.lang.Long	xsd:long	soapenc:long
java.lang.Byte	xsd:byte	soapenc:byte
java.lang.Float	xsd:float	soapenc:float
java.lang.Double	xsd:double	soapenc:double
java.lang.Boolean	xsd:boolean	soapenc:boolean
java.lang.String	xsd:string	soapenc:string
java.math.BigDecimal	xsd:decimal	soapenc:decimal
java.math.BigInteger	xsd:integer	soapenc:integer
java.util.Calendar	xsd:dateTime	xsd:dateTime
javax.xml.namespace.QName	xsd:QName	xsd:QName
java.net.URI	xsd:anyURI	xsd:anyURI
byte[]	xsd:base64Binary	soapenc:base64

そのほかのXMLの単純型

以下のXMLのデータ型は、Javaベースの開発では使用されませんが、WSDLおよびSOAP通信では利用できます。利用した場合、データの対応は以下の表のとおりです。

これらのデータ型を使用する場合、SOAP通信でXML上の値がXMLのデータ型に適合することを、アプリケーションで保証する必要があります。

XMLで使用するデータ型	対応するJavaのデータ型
xsd:nonPositiveInteger	java.math.BigInteger
xsd:negativeInteger	java.math.BigInteger
xsd:nonNegativeInteger	java.math.BigInteger
xsd:unsignedLong	java.math.BigInteger
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:positiveInteger	java.math.BigInteger
xsd:normalizedString	java.lang.String
xsd:duration	java.lang.String
xsd:time	java.util.Calendar
xsd:date	java.util.Calendar
xsd:gYearMonth	java.lang.String
xsd:gYear	java.lang.String
xsd:gMonthDay	java.lang.String
xsd:gDay	java.lang.String
xsd:gMonth	java.lang.String
xsd:token	java.lang.String
xsd:language	java.lang.String
xsd:NMTOKEN	java.lang.String
xsd:Name	java.lang.String
xsd:NCName	java.lang.String
xsd:ID	java.lang.String
xsd:hexBinary	byte[]

nillableについて

literalのWSDLのスキーマ定義でnillable属性が付いていない、またはnillable属性値が偽(false/0)のelementは、実際の通信でnil（Javaのnullに相当）になることはない要素、という意味の定義になります。
このようなelementに対応するJavaの変数は、値をnullで送信しないでください。

また、Javaで値がnullにならないデータ型（プリミティブ型）に対して、値がnillの要素を受信した場合、該当の変数は、Java言語規定で定義された初期値が設定される場合があります。

18.3.2 構造体型・Bean型

構造体型とBean型について、説明します。

- 構造体型
- Bean型

構造体型

構造体型は、任意の数のデータ(メンバと呼びます)をpublicインスタンスフィールドとして持つデータ型です。メンバは、サポートされているデータ型であれば自由に定義できます。

構造体型と、それに対応してXMLで使用される型を以下に示します。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型
構造体型として定義した任意のクラス	構造体 (xsd:sequenceを内容とするxsd:complexTypeとして定義されます)

構造体型は、次の条件を満たすpublicクラスとして定義します。

- 各メンバをpublicインスタンスフィールドとして持つ(transientまたはfinal修飾されていないこと)
- publicインスタンスフィールドのデータ型は、すべてサポートされているデータ型である
- publicデフォルトコンストラクタを持つ
- java.rmi.Remoteインタフェースをimplementsしていない

例

構造体型として定義したクラスの例

```
package com.example;  // パッケージ名
 public class Person {
   public String name;    // 各メンバをpublicインスタンスフィールドとして宣言する
   public int age;        // 同上
   public Person (){        // public デフォルト(引数なし)コンストラクタ
     name="nobody";
     age=0;
   }
   public Person (String myname, int myage){  // publicコンストラクタ(省略可)
     name = myname;
     age = myage;
   }
 }
```

上記のJavaに対応する、XMLの構造体のWSDL定義例を以下に示します。

```
<xsd:complexType name="Person">
   <xsd:sequence>
     <xsd:element name="age" type="xsd:int"/>
     <xsd:element name="name" type="xsd:string"/>
   </xsd:sequence>
 </xsd:complexType>
```

注意

- publicインスタンスフィールドで定義するメンバのほかに、Bean型と同様にset/getメソッドを提供することでもメンバを定義することもできます。この場合、set/getメソッドで定義するメンバと、publicインスタンスフィールドで定義するメンバで、メンバ名(プロパティ名)が重ならない様に定義してください。メンバ名(プロパティ名)の違いは、大文字小文字のみの違いではなく、スペルが異なる様に定義してください。
- 構造体型がBean型や他の構造体型を継承しており、構造体型とそのスーパークラスの両方で同じ名前のpublicインスタンスフィールドが宣言されている場合、WSDLの生成やSOAP通信には、スーパークラスで宣言されたフィールドのみが使用されます。
- Javaのクラスではメンバの順番が保障されません。そのためiswsgen wsdlコマンドは、JavaからWSDLを生成する際に、メンバを名前順に並べ替えることでWSDL上でメンバの順番の一貫性を保ちます。

Bean型

Bean型は、任意の数のデータ(メンバと呼びます)をプロパティとして持つ、構造体に類似したデータ型です。メンバは、サポートされているデータ型であれば自由に定義できます。
Bean型では、set/getメソッドを提供することでメンバを定義します。

Bean型と、それに対応してXMLで使用される型を以下に示します。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型
Bean型として定義した任意のクラス	構造体 (xsd:sequenceを内容とするxsd:complexTypeとして定義されます)

Bean型は、次の条件を満たすpublicクラスとして定義します。

- 各メンバをインスタンスのプロパティ(set/getメソッドのペア)として持つ
- インスタンスのプロパティ(set/getメソッドのペア)のデータ型は、すべてサポートされているデータ型である
- publicデフォルトコンストラクタを持つ
- java.rmi.Remoteインタフェースをimplementsしていない

例

Bean型の定義例

```
package com.example;    // パッケージ名
 public class PersonBean {
   // 各メンバをプロパティ (set/getメソッドのペア) として宣言する
   private String _name;//privateのため、メンバとして直接参照されない
   public void setName(String name) { _name = name; }
   public String getName() { return _name; }
   // 同上
   private int _age; //privateのため、メンバとして直接参照されない
   public void setAge(int age) { _age = age; }
   public int getAge() { return _age; }

   public PersonBean() {     // public デフォルト(引数なし)コンストラクタ
      _name="nobody";
      _age = 0;
   }
   public PersonBean(String name, int age){   // publicコンストラクタ(省略可)
     _name=name;
     _age=age;
   }
 }
```

上記のJavaに対応する、XMLの構造体のWSDL定義例を以下に示します。

```
<xsd:complexType name="Person">
   <xsd:sequence>
     <xsd:element name="age" type="xsd:int"/>
     <xsd:element name="name" type="xsd:string"/>
   </xsd:sequence>
 </xsd:complexType>
```

注意

- set/getメソッドで定義するメンバのほかに、構造体と同様にpublicインスタンスフィールドを提供することでもメンバを定義することもできます。この場合、set/getメソッドで定義するメンバと、publicインスタンスフィールドで定義するメンバで、メンバ名（プロパティ名）が重ならない様に定義してください。メンバ名（プロパティ名）の違いは、大文字小文字のみの違いではなく、スペルが異なる様に定義してください。
- Bean型が他のBean型や構造体型を継承しており、Bean型とそのスーパークラスの両方で同じ名前のpublicインスタンスフィールドが宣言されている場合、WSDLの生成やSOAP通信には、スーパークラスで宣言されたフィールドのみが使用されます。
- Javaのクラスではプロパティの順番が保障されません。そのためiswsgen wsdlコマンドは、JavaからWSDLを生成する際に、プロパティを名前順に並べ替えることでWSDL上でメンバの順番の一貫性を保ちます。

そのほかのXMLの構造体

以下のXMLのデータ型は、Javaベースの開発では使用されませんが、WSDLおよびSOAP通信で利用は可能です。

- xsd:allを内容とするxsd:complexTypeとして定義された型
- メンバをxsd:attributeとして定義した構造体

18.3.3 配列型

サポートされているデータ型は、配列にして配列型として使用する事ができます。

配列型は、選択したスタイル方式によってXMLで使用されるデータ型が異なります。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型	
	デフォルト(literal利用の場合)	encoded利用の場合
配列	maxOccurs="unbounded"が指定されたxsd:element	soapenc:Arrayをベース型としてsoapenc:arrayType属性に要素の型を指定

例

- Javaのint型の配列（int[]）に対応する、literal利用時のXMLの配列のWSDL定義例を以下に示します。

```
<xsd:element name="age" maxOccurs="unbound" type="xsd:int"/>
```

- Javaのint型の配列（int[]）に対応する、encoded利用時のXMLの配列のWSDL定義例を以下に示します。

```
<xsd:complexType name="ArrayOf_xsd_int">
   <xsd:complexContent>
      <xsd:restriction base="soapenc:Array">
         <xsd:attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int[]"/>
      </xsd:restriction>
   </xsd:complexContent>
</xsd:complexType>
```

そのほかのXMLの配列

以下のXMLのデータ型は、Javaベースの開発では使用されませんが、WSDLおよびSOAP通信では利用できます。

- maxOccursが2以上に指定されたxsd:element
- soapenc:Arrayをベース型とし、内容をmaxOccursが"unbounded"または2以上が指定された単一のxsd:elementと定義されたcomplexType

18.3.4 添付ファイル型

添付ファイル型は、実際の通信においてXMLへ変換されず、SOAPメッセージの添付ファイルとしてそのままのデータで送受信されるデータ型です。
通常のデータ型と同様に、Webサービスアプリケーションのパラメタ(引数、返り値)、Bean型・構造体型のメンバなどに利用できます。

添付ファイル型に使用するクラスと、それに対応してXMLで使用される型を以下に示します。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型
javax.activation.DataHandler	wsibp:swaRef (またはMIMEバインディングのpart要素に任意のMIME型としてマップ)

ポイント

- wsibp:swaRefは、SOAPメッセージから添付ファイルを参照する要素のXMLのデータ型です。
- 実際のデータは、SOAPメッセージの添付ファイルとして送受信されます。
- iswsgen wsdlコマンドのオプションにより、XMLで使用されるデータ型をapachesoap:DataHandlerとすることもできます。
- 以下の場合、WSDLでMIMEバインディングのpartに任意のMIME型として直接マップされ、XMLのデータ型wsibp:swaRefは使用されません。
  - iswsgen wsdlコマンドで、-styleuseオプションにDOCUMENTLITERALWRAPPED以外を指定し、かつ、
  - javax.activation.DataHandlerがサービスエンドポイントインタフェースの引数または返り値に(Bean/構造体型のメンバや配列ではなく)直接宣言されている場合

javax.activation.DataHandlerは、JAF(Java Activation Framework)で提供されるクラスです。MIMEの各種情報およびデータ内容を保持します。本クラスの詳細については、JavaDocを参照してください。

そのほかの添付ファイルのデータ型

下記のJavaクラスも添付ファイルを表すデータ型として利用することができます。ただし、これらのJavaクラスを使用する場合、バイト列とJavaオブジェクトへの変換が行われるなど、オリジナルのバイト列データが正確に保持されない場合があります。また、これらのクラスを使用したサービスエンドポイントインタフェースからは作成するWSDLはWS-I Attachments Profileに準拠できない場合があります。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型
java.awt.Image	apachesoap:Image (またはMIMEバインディングのpart要素MIME型image/jpegとしてマップ)
javax.mail.internet.MimeMultipart	apachesoap:MimeMultipart (またはMIMEバインディングのpart要素にMIME型multipart/*としてマップ)
javax.xml.transform.Source (注)	apachesoap:Source (またはMIMEバインディングのpart要素にMIME型text/xmlとしてマップ)

注) javax.xml.transform.Sourceインタフェースの実装クラスには、javax.xml.transform.source.StreamSourceクラスを使用してください。それ以外のクラスのオブジェクトはサポートされません。

mime:contentからJavaクラスへのマッピング

WSDL中のmime:content要素で添付ファイルが指定されている場合、次の表に示すようにJavaクラスにマップされます。

mime:content要素のtype属性	Javaクラス
image/jpeg	java.awt.Image
text/plain	java.lang.String
multipart/*	javax.mail.internet.MimeMultipart
text/xml	javax.xml.transform.Source
application/xml	javax.xml.transform.Source
上記以外のMIME型 (注)	javax.activation.DataHandler

注) image/gif型は未サポートです。

mime:content要素が使用された場合、mime:content要素のpart属性から参照されたwsdl:part要素のtype及びelement属性は、生成されるサービスエンドポイントインタフェースにマップされません。替わりに上記のJavaクラスが使用されます。

ポイント

- apachesoap:Image、apachesoap:MimeMultipart、apachesoap:Sourceは、SOAPメッセージから添付ファイルを参照する要素のXMLのデータ型です。
- 実際のデータは、SOAPメッセージの添付ファイルとして送受信されます。
- これらのJavaクラスを使用する場合は、iswsgen wsdlコマンドのオプションで、apachesoapのネームスペースを使用する指定が必要です。
- 以下の場合、WSDLのMIMEバインディングのpartに直接マップされ、上記のXMLのデータ型は使用されません。
  - iswsgen wsdlコマンドで、-styleuseオプションにDOCUMENTLITERALWRAPPED以外を指定し、かつ、
  - 添付ファイルのJavaのデータ型がサービスエンドポイントインタフェースの引数または返り値に(Bean/構造体型のメンバや配列ではなく)直接宣言されている場合
- Solaris64 Linux32/64
  Javaアプリケーションでjava.awt.Imageクラスを使用する場合は、X Windowシステムのアプリケーションが動作する環境を用意するか、ヘッドレスモードでのアプリケーションの実行が必要になる場合があります。詳細はJavaのドキュメントを参照して下さい。

なお、MIMEバインディングのpart要素が上記に記載されたMIME型以外の場合は、Webサービスアプリケーションのパラメタとしてjavax.activation.DataHandlerが使用されます。ただし、MIME型image/gifは利用できません。

注意

一定以上のサイズの添付ファイルを受信する際、メモリ節約のため、内部的に一時ファイルが作成されます。また、添付ファイルに関するAPIを使用した場合も、一時ファイルが作成される場合があります。
一時ファイルの生成に関するチューニングは、Webサービス設定ファイルで行います。
詳細については、“Webサービスの運用”の“19.3 Webサービス設定ファイル”を参照してください。

18.3.5 out/inoutパラメタとしての利用

サポートされているデータ型を、outパラメタまたはinoutパラメタとして利用する場合は、Java上ではHolderクラスを利用してください。

ポイント

- outパラメタとは、クライアントからは値を送信せず、サーバから値が返信される引数です。
- inoutパラメタとは、クライアントから値を送信するとともに、サーバからも値が返信される引数です。

Holderクラス

Holderクラスの形式を、以下に示します。

クラス名	クラス本体要素	説明
“内容型名”Holder	public 内容型 value	public インスタンスフィールド。 インスタンスが保持している値です。
	public “内容型名”Holder()	publicデフォルトコンストラクタ。 valueフィールドを初期値(プリミティブ型は0またはfalse、参照型はnull)としてインスタンスを構築します。
	public “内容型名” Holder(内容型 initValue)	publicコンストラクタ。 valueフィールドをinitValueに設定したインスタンスを構築します。

Webサービスアプリケーション、Webサービスクライアントでは、Webサービスのout/inoutパラメタに対応するJava上の引数として、Holderクラスを使用します。
Webサービスアプリケーションでは、引数で受け取ったHolderオブジェクトのvalueフィールドに値を設定することで、その値がoutパラメタとしてサービスクライアントへ返信されます。

Webサービス呼び出し後に、Webサービスクライアントでは、引数に使用したHolderオブジェクトのvalueフィールドに返信された値が、設定されています。

Webサービスクライアントでは、Webサービス呼び出しから復帰すると、引数に使用したHolderオブジェクトのvalueフィールドに、Webサービスがoutパラメタとして返信した値が設定されています。

```
//Webサービスのメソッド
 public int serverMethod(int  inParam,  IntHolder  inoutParam, StringHolder  outParam)
     throws RemoteException {
   int  number = inoutParam.value;  // inoutパラメタは入力値を参照し必要な処理を行う
      :
   // out/inout パラメタのvalueフィールドに値を代入(return時、クライアントに返信される)
   inoutParam.value = 10;
   outParam.value = "abc";
   return 0;
 }
```

例

Holderクラスの使用例(Webサービスクライアント)

```
//パラメタの準備
int inParam = 123;
IntHolder inoutParam = new IntHolder(987);
StringHolder outParam = new StringHolder();

//Webサービスの呼出し(サーバが上記の例の場合、resultには0が入る)
int result = portStub.serverMethod(inParam, inoutParam, outParam);

//返信されたout/inoutパラメタ値の利用(out/inoutパラメタのvalueフィールドから値を取得)
int inoutResult = inoutParam.value; //(サーバが上記の例の場合、10)
String outResult = outParam.value;  //(サーバが上記の例の場合、"abc")
      :
```

標準でHolderクラスが提供されているデータ型

単純型の多くのデータ型については、javax.xml.rpc.holders パッケージに標準のHolderクラスが提供されていますので、これらのHolderクラスを使用します。

- StringHolder, BooleanHolder, BooleanWrapperHolder, ByteHolder, ByteWrapperHolder, DoubleHolder, DoubleWrapperHolder, FloatHolder, FloatWrapperHolder, IntHolder, IntegerWrapperHolder, LongHolder, LongWrapperHolder, CalendarHolder, QNameHolder, ShortHolder, ShortWrapperHolder, BigDecimalHolder, BigIntegerHolder, ByteArrayHolder

名前に“Wrapper”が含まれるHolderクラスは、プリミティブ型に対応するラッパクラスのHolderクラスです（例：BooleanWrapperHolderは、java.lang.BooleanクラスのHolderクラスです。boolean型のHolderクラスはBooleanHolderです）。

Interstage Webサービスでは、以下のHolderクラスを利用できます。

Javaクラス	Holderクラス
java.net.URI	com.fujitsu.interstage.isws.apis.holders.URIHolder
javax.activation.DataHandler	com.fujitsu.interstage.isws.apis.holders.DataHandlerHolder
java.awt.Image	com.fujitsu.interstage.isws.apis.holders.ImageHolder
javax.mail.internet.MimeMultipart	com.fujitsu.interstage.isws.apis.holders.MimeMultipartHolder
javax.xml.transform.Source	com.fujitsu.interstage.isws.apis.holders.SourceHolder

標準でHolderクラスが提供されていないデータ型

配列や構造体など、標準でHolderクラスが提供されていない場合、以下の条件を満たすように固有のHolderクラスを用意して使用します。

- Holderクラスに記載されたHolderクラスの形式を満たすpublicクラスである
- javax.xml.rpc.holders.Holder インタフェースをimplementsしている
- 利用するデータ型クラスのパッケージに“holders”を付加したパッケージに属している（例：com.example.Personの場合、com.example.holders.PersonHolderになります）

 例

Holderクラスの定義例(内容型がcom.example.Personクラスの場合)

```
package com.example.holders;
final public class PersonHolder implements javax.xml.rpc.holders.Holder {
    public com.example.Person value;
    public PersonHolder() {}
    public PersonHolder(com.example.Person initial) { value = initial; }
}
```

Webサービスアプリケーションでのoutパラメタの使用

サービスエンドポイントインタフェースでパラメタにHolderクラスを使用した場合、iswsgen wsdl コマンドなどで生成されるWSDLでは、該当のパラメタはinoutパラメタになります。
inoutパラメタではなくoutパラメタとする必要がある場合は、WSDLを修正してoperationの該当パラメタがoutパラメタになるように、inputメッセージから該当のパラメタを削除してください。

WSDLを修正する場合は、WSDLの仕様などを参照して該当部分のみを正しく修正してください。

18.4 使用できるWSDLについて

Interstage Webサービスで使用するWSDLが、WSDLの仕様以外に従う必要のある事項について説明します。

参考

本節で使用しているXMLのネームスペースのprefixは、以下のとおりです。

- “wsdl”が表すネームスペース:“http://schemas.xmlsoap.org/wsdl/”
- “soap”が表すネームスペース:“http://schemas.xmlsoap.org/wsdl/soap/”
- “mime”が表すネームスペース:“http://schemas.xmlsoap.org/wsdl/mime/”
- “wsibp”が表すネームスペース:“http://ws-i.org/profiles/basic/1.1/xsd”

型定義

- wsdl:typesでの型定義には、XMLスキーマのみ使用できます。個々のデータ型については、“18.3 Javaのデータ型とXMLのデータ型との対応”を参照してください。
- XMLインスタンスの解釈が不定となる、スキーマ定義(例:出現回数が固定でなくかつ名前が同じ複数のelement定義が連続したsequenceモデル)に注意してください。そのような定義のデータは、実際の通信において、受信時の解釈も不定となります。
- 1つのcomplexType定義に、複数のany定義が含まれないようにしてください。

メッセージ

- styleが“document”の場合、wsdl:messageに含まれるwsdl:partは1つ以下でなければなりません。
- wsdl:partは、type属性またはelement属性のいずれか一方だけを指定できます。

ポートタイプ

- wsdl:portTypeには、request-response型のオペレーションだけを使用できます。
- operation要素のname属性には、1つのwsdl:portType内でユニークな値を指定しなければなりません。

バインディング

- wsdl:bindingには、SOAPバインディングおよびMIMEバインディングだけを使用できます。

サービス

- 1つのWSDLに、wsdl:serviceは1つ以上存在するようにしてください。

WSDL拡張

- WSDLの拡張要素/属性には、SOAPバインディングおよびMIMEバインディングで規定されたものだけが使用できます。

SOAPバインディング

- soap:bindingのtransport属性の値は、“http://schemas.xmlsoap.org/soap/http”でなければなりません。
- soap:headerは使用できません。
- 1つのWSDLで、styleの指定は“document”または“rpc”のいずれかに統一されていなければなりません。
- styleが“document”の場合、useには“literal”のみ指定できます。
- 1つのWSDLで、useの指定は“literal”または“encoded”のいずれかに統一されていなければなりません。
- useの指定は省略できません。
- encodingStyle属性を指定する場合は、値は“http://schemas.xmlsoap.org/soap/encoding/”でなければなりません。

MIMEバインディング

以下の内容のWSDLはサポートされません。

- mime:part要素が複数のmime:content要素を含む場合
- wsibp:swaRefデータ型がxsd:element要素のtype属性またはwsdl:part要素のtype属性以外の場所から参照されている場合

その他 Windows32/64

以下のいずれかがDOSデバイス名と一致するWSDLは使用できません。

- 型名
- ポートタイプ名
- サービス名
- ネームスペースの構成要素('.'や'/'、':'で区切られた文字列)

該当する場合、iswsgen serverコマンドまたはiswsgen clientコマンドがisws15208やisws10443、isws10314などのエラーになる場合があります。
ただし、ネームスペースの構成要素のみが該当する場合は、iswsgenコマンドを以下のように実行することでWSDLを使用できます。

- -PkgNSmappingFileオプションでJavaパッケージ名の構成要素がDOSデバイス名を含まないように指定する

18.5 WS-I Basic ProfileおよびAttachments Profileに準拠した開発

異なるプラットフォーム間での相互運用性を保証する手段として、WS-Iが定めるProfileに準拠したシステムを構築することが挙げられます。本製品では、WS-I Basic Profile1.0およびWS-I Attachments Profile 1.0に準拠したシステムを構築することが可能です。

これらのProfileに準拠したシステムを構築する場合は、以下の点に注意してください。

iswsgen wsdlコマンドでWSDLファイルを生成する

WS-I Basic Profile1.0の規約では、WSDLファイルのuse属性を“literal”にする必要があります。iswsgen wsdl コマンドのデフォルトでuse属性が“literal”のWSDLファイルが生成されます。-styleuse(-y)オプションで指定を変更する場合は注意してください。iswsgen wsdl コマンドのオプション設定方法については、“リファレンスマニュアル(コマンド編)”の“iswsgen”を参照してください。

WSDLファイルを作成、または編集する

ユーザが自分でWSDLファイルを作成、またはiswsgen wsdl コマンドで生成したWSDLファイルを編集する場合は、WSDLファイルの最終的な作成者がWS-I Basic Profile1.0の規約の準拠を保証する必要があります。
最新バージョンのWS-I Basic Profile1.0の規約およびエラッタを参照して、規約の範囲内でWSDLファイルを作成、または編集してください。

最新バージョンのWS-I Basic Profile1.0の規約およびエラッタは、WS-I OrganizationのWebサイトを参照してください。

注意

日本語版の規約は最新版ではない場合があります。必ず最新版とエラッタを参照してください。

- http://www.oasis-ws-i.org/

アプリケーション実行時のパラメタについて

WSDL作成時に-styleuse(-y)オプションでRPCLITERALを指定してアプリケーションを作成している場合、またはstyle属性が“rpc”、use属性が“literal”のWSDLファイルを使用してアプリケーションを作成している場合、以下の点に注意してください。

- Webサービスアプリケーションのパラメタの値としてnullを指定して送信しないでください。
- Webサービスアプリケーションのjavax.activation.DataHandlerの添付ファイル型のパラメタには、ひとつの通信内でほかの箇所と同一インスタンスを指定せず、それぞれ個別のインスタンスを指定してください。

セションの利用について

Webサービスアプリケーションではセションを利用可能です。しかし、WS-I Basic Profile1.0では、Webサービスの正常動作がHTTP Cookieに依存しない事が推奨されています。セションを利用する場合は、補助的な利用にとどめ、セションが継続されなくても正常動作できる設計にする事が推奨されます。

BOMを使用したUTF-8でエンコーディングされたXMLデータの扱い

WS-I Basic Profile1.0では、通信に使用するSOAPメッセージのエンコーディングに、BOM（Byte Order Mark）を使用したUTF-8の使用を認めています。ユーザが、BOMが付加されているUTF-8でエンコーディングされたXMLデータを使用して、SOAPエンベロープを作成する場合は、以下のSourceインタフェースを実装したクラスを使用してください。

- SAXSource
- StreamSource

ユーザが使用するパーサの種類によっては、上記以外のSourceインタフェースを実装したクラスを使用しても問題がない場合があります。詳細は使用するパーサの仕様を確認してください。

なお、XMLデータを使用してSOAPエンベロープを作成する方法の詳細については、“付録C SOAPメッセージの低レベル処理”を参照してください。

WS-I Attachments Profile 1.0に準拠したアプリケーションの開発

WS-I Attachments Profile 1.0に準拠したアプリケーションを開発する際には、以下に注意してください。

- iswsgen wsdlサブコマンド実行時に-attachmentsTypeオプションで“apache”と指定した場合、非標準のXMLデータ型がWSDLファイルに出力されます。Attachments Profileに準拠したアプリケーションを開発する場合は-attachmentsTypeオプションを省略するか、オプション値に“swaref”と指定して、Javaクラスがwsibp:swaRef型またはmime:content要素にマップされるようにアプリケーションを作成してください。

また、本節の他の説明も参照し、Basic Profileにも準拠するようにアプリケーションを作成してください。

18.6 Webサービス環境定義ファイル(deployment descriptor)

Webサービス環境定義ファイル(webservices.xml)は、Webサービスの動作環境を設定します。また、Webサービスが動作するには、Webアプリケーション環境定義ファイル(web.xml)も必要です。

Webサービスクライアントアプリケーションでは JNDIを使用してServiceをlookupする場合、service reference記述をdeployment descriptorに追加する必要があります。

ここでは、これらの記述方法について説明します。

18.6.1 webservices.xmlの記述形式

webservices.xmlの記述

webservices.xmlは、Webサービスアプリケーションについてのdeployment descriptorです。

アプリケーションのWebサービス関連の構成物のパスや、提供するWebサービスの実装を記述します。パスは、WARにパッケージングした時のモジュール内のパスを記述します。

deployment descriptorは、XML形式で記述します。以下にdeployment descriptorの記述形式を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                                 http://java.sun.com/xml/ns/j2ee/j2ee_web_services_1_1.xsd"
             version="1.1">
```

```
  <webservice-description>
    <webservice-description-name>name</webservice-description-name>
    <wsdl-file>file-path</wsdl-file>
    <jaxrpc-mapping-file>file-path</jaxrpc-mapping-file>
    <port-component>
      <port-component-name>name</port-component-name>
      <wsdl-port>qname</wsdl-port>
      <service-endpoint-interface>interface</service-endpoint-interface>
      <service-impl-bean>
        <servlet-link>servlet-name</servlet-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```

注意

記述にあたっての注意事項

- 本マニュアルに記載した定義以外は、使用できません。
- 先頭の<?xml...>はXML宣言を記述しているため、必ず、deployment descriptorファイルの先頭で記述してください。
- deployment descriptorに日本語2バイト文字を使用する場合は、<?xml...>のエンコード形式(「encoding=」部分)にUTF-8を指定してください。日本語2バイト文字を使用している場合に、UTF-8以外を指定すると配備できません。この制限はコメントにも適用されます。
- <webservices・・・>、</webservices>は、XMLファイルの開始と終了を示すルートタグです。必ず指定してください。
- 各タグの記載順序は、上記の記載順序に従ってください。
- 大文字・小文字は区別します。
- 本マニュアルに記載した定義以外を指定した場合でも、エラーメッセージが出力されずにIJServerが起動する場合がありますので、注意してください。

18.6.2 webservices.xmlのタグ

Webサービス環境定義ファイル(webservices.xml)には、以下のタグを指定できます。

定義する内容

タグ名	説明	タグの省略	複数の指定
webservices	webservices.xmlの開始と終了を定義します。	×	×
webservice-description	一つのWebサービスに関する定義を記述します。	×	○
webservice-description-name	Webサービスの名前を定義します。 webservices.xml内で一意の値とする必要があります。 注) 以下の文字は使用できません。 ','、'='、':'、'*'、'?'、""の記号	×	×
wsdl-file	モジュール内のWSDLのパスを定義します。	×	×
jaxrpc-mapping-file	モジュール内の<WSDLファイル名>_mapping.xmlのパスを定義します。	×	×
port-component	1つのポートに関する定義を記述します。	×	○

タグ名	説明	タグの省略	複数の指定
port-component-name	ポートの名前を定義します。 webservices.xml内で一意の値とする必要があります。 注) 以下の文字は使用できません。 ','、'='、':'、'*'、'?'、""の記号	×	×
wsdl-port	WSDL内の対応するポートのQNameを定義します。wsdl-fileタグで指定したWSDLファイルに定義されたポートである必要があります。 ポートのQNameは、WSDLのport要素のname属性で指定され、その値はWSDLのdefinitions要素のtargetNamespace属性で指定されたネームスペースに属します。 注) webservices.xml内の複数のwsdl-portタグで、WSDLの同一のポートを指定することはできません。	×	×
service-endpoint-interface	サービスエンドポイントインタフェースの完全修飾名を定義します。	×	×
service-impl-bean	Webサービス実装クラスの定義を記述します。	×	×
servlet-link	Webサービスに対応するweb.xml内のサーブレット名を定義します。 注) webservices.xml内の複数のservlet-linkタグで同一のサーブレットを指定することはできません。 注) servlet-linkまたはejb-linkのいずれか一方のみを必ず指定する必要があります。	○	×
ejb-link	Webサービスに対応するejb-jar.xml内のSTATELESS Session Beanのejb-nameを指定します。 注) webservices.xml内の複数のejb-linkタグで同一のEJBアプリケーションを指定することはできません。 注) servlet-linkまたはejb-linkのいずれか一方のみを必ず指定する必要があります。	○	×

注意

- 省略が“×”であるタグは、省略すると配備できません。
- 複数の指定が“×”であるタグを重複して指定すると配備できません。

例

モジュール構成が、以下の場合の例です。

```
WARファイル
 |__ WEB-INF
     |__ classes
         |__ com
             |__ example
                 |__ StockQuoteProvider.class
```

```
                |__ StockQuoteProviderImpl.class
    |__ web.xml
    |__ webservices.xml
    |__ wsdl
        |__ StockQuoteProviderPort.wsdl
    |__ StockQuoteProviderPort_mapping.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<webservices xmlns="http://java.sun.com/xml/ns/j2ee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                                 http://java.sun.com/xml/ns/j2ee/j2ee_web_services_1_1.xsd"
             version="1.1">
    <webservice-description>
        <webservice-description-name>StockQuoteProviderService</webservice-description-name>
        <wsdl-file>WEB-INF/wsdl/StockQuoteProviderPort.wsdl</wsdl-file>
        <jaxrpc-mapping-file>WEB-INF/StockQuoteProviderPort_mapping.xml</jaxrpc-mapping-file>
        <port-component>
            <port-component-name>StockQuoteProvider</port-component-name>
            <wsdl-port xmlns:pfx="http://example.com">pfx:StockQuoteProviderPort</wsdl-port>
            <service-endpoint-interface>com.example.StockQuoteProvider</service-endpoint-interface>
            <service-impl-bean>
                <servlet-link>StockQuoteServlet</servlet-link>
            </service-impl-bean>
        </port-component>
    </webservice-description>
</webservices>
```

18.6.3 web.xml

Webサービスの動作には、Webサービス環境定義ファイル(webservices.xml)の他に、Webアプリケーション環境定義ファイル(web.xml)も必要です。

web.xmlの記述

“Webアプリケーション環境定義ファイル(deployment descriptor)の記述形式”に記載の形式でweb.xmlを作成します。新規追加のタグはありません。
以下の表にないタグについては、“Webアプリケーション環境定義ファイル(deployment descriptor)のタグ”を参照してください。

タグ名	説明
servlet-name	webservices.xmlのservlet-linkタグから参照されるサーブレット名を定義します。
servlet-class	Webサービス実装クラスの完全修飾クラス名を定義します。
url-pattern	Webサービスにマッピングするurlを定義します。省略値は以下となります。 ・ "/services/" + webservices.xmlのport-component-nameタグの値

例

モジュール構成が、以下の場合の例です。

```
WARファイル
 |__ WEB-INF
     |__ classes
         |__ com
             |__ example
                 |__ StockQuoteProvider.class
                 |__ StockQuoteProviderImpl.class
     |__ web.xml
     |__ webservices.xml
```

```
    |__ wsdl
        |__ StockQuoteProviderPort.wsdl
    |__ StockQuoteProviderPort_mapping.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
      <servlet-name>StockQuoteServlet</servlet-name>
      <servlet-class>com.example.StockQuoteProviderImpl</servlet-class>
   </servlet>
</web-app>
```

18.6.4 service reference記述

Webサービスクライアントアプリケーションから JNDIを使用してServiceをlookupする場合、クライアントアプリケーションの形態に応じて以下のdeployment descriptorにservice reference記述を追加する必要があります。

- WebアプリケーションからServiceをlookupする場合・・・web.xml
- EJBアプリケーションからServiceをlookupする場合・・・ejb-jar.xml
- J2EEアプリケーションクライアントからServiceをlookupする場合・・・application-client.xml

対象となるdeployment descriptorに<service-ref>タグを追加し、Webサービス参照についての定義を記述してください。

以下の表にないタグについては、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”、“4.13 J2EEアプリケーションクライアントのdeployment descriptorファイルの詳細設定”、または“4.9 deployment descriptorファイルへの記述”を参照してください。

定義する内容

要素名	説明	タグの省略	複数の指定
service-ref	Webサービス参照についての定義を記述します。	○	○
service-ref-name	JNDIに登録する名前を定義します。“service/”で始まるように定義してください。	×	×
service-interface	Serviceインタフェースの完全修飾クラス名を定義します。	×	×
wsdl-file	iswsgenコマンドまたはInterstage Studioを使用してスタブおよびクライアントアプリケーションを作成した場合、本要素は省略してください。 WebアプリケーションまたはEJBアプリケーションで本要素が指定された場合、運用時にInterstageが利用するスタブ/サービスの実装クラスをIJServerへの配備時に生成します。本要素はWSDLファイルをモジュールのルートからの相対パスで定義します。	○	×
jaxrpc-mapping-file	<WSDLファイル名>_mapping.xmlをモジュールのルートから相対パスで定義します。wsdl-fileを指定した場合は本要素は省略不可です。	○	×

注意

- <wsdl-file>を指定した場合に生成されるクラス名は、“_isws_パッケージ名を除いたクラス名”となります(例:stock.server._isws_StockQuoteProviderPortSoapBindingStub)。配備するモジュール内に同名のクラスが存在した場合は上書きされます。
- <service-interface>要素に“javax.xml.rpc.Service”を指定することはできません。WSDLから生成されたサービス固有のServiceインタフェース(javax.xml.rpc.Serviceを継承したインタフェース)のクラス名を指定してください。
- JDKがインストールされていない場合、<wsdl-file>を指定することはできません。

例

WebアプリケーションからServiceをlookupする場合(web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd version="2.4">
  <display-name>StockQuote Client</display-name>
      ...
  <service-ref>
      <description>WSDL Service StockQuote </description>
      <service-ref-name>service/StockQuote</service-ref-name>
      <service-interface>stock.server.StockQuoteProviderService</service-interface>
  </service-ref>
</web-app>
```

18.7 サンプルアプリケーションの格納先

Webサービスのサンプルアプリケーションは、以下のディレクトリに格納されています。

実行手順については、サンプルが格納されている以下のディレクトリにドキュメントが同梱されていますので、そちらを参照してください。

Windows32/64

C:¥Interstage¥J2EE¥sample¥isws

Solaris64 Linux32/64

/opt/FJSVj2ee/sample/isws

第19章　Webサービスの運用

本章では、Webサービスの運用について説明します。

19.1 Webサービス(サーバ機能)の運用方法

Interstage管理コンソール、またはコマンドを使用してWebサービスアプリケーションの運用操作を行います。

IJServer作成、環境設定

Webサービスを使用する場合、同一VMタイプのIJServerを作成します。

注意

- IJServerのタイプは同一VMだけとなります。
- 使用できるXMLパーサは、JAXP1.2以上をサポートしているものだけです。

アプリケーションの配備/配備解除

Webサービスアプリケーションを含むWAR/EARを、従来のWAR/EARと同様に配備/配備解除します。
アプリケーションの配備/配備解除について詳細は、“3.5 J2EEアプリケーションの配備と設定”を参照してください。

WSDLファイル、およびWSDLファイルから参照されているリソース読み込み処理の、デフォルトのタイムアウト時間は45秒です。
デフォルトのタイムアウト時間を変更する場合、Interstage JMXサービスが使用するjavaプロセスのシステムプロパティに以下の値を設定してください。

```
-Dcom.fujitsu.interstage.isws.deploy.wsdl.timeout=タイムアウト時間(単位:ミリ秒)
```

“0”を指定した場合、タイムアウト時間は無制限になります。

例

タイムアウト時間を90秒にする場合

```
-Dcom.fujitsu.interstage.isws.deploy.wsdl.timeout=90000
```

また、プロキシを経由してWSDLファイルから参照されているリソースを取得する場合、Interstage JMXサービスが使用するjavaプロセスのシステムプロパティに設定します。

システムプロパティに設定する値については、“19.2.3 プロキシを経由した接続”を参照してください。
Interstage JMXサービスが使用するjavaプロセスのシステムプロパティの設定方法は、“運用ガイド(基本編)”の“Interstage管理コンソール環境のカスタマイズ”の“Interstage JMXサービスのカスタマイズ”を参照してください。

Webサービスの公開URL

WARファイルの場合、Servletの仕様に従ってWebサービスの公開URLをカスタマイズすることができます。詳細は、“8.1.1 マッピングが必要な呼び出し方”の“URLで指定して呼び出す場合”および“18.6.3 web.xml”を参照してください。

公開用WSDLの取得

IJServerにWebサービスアプリケーションを配備すると、そのWebサービスの公開用WSDLを取得できます。
必要に応じて、このWSDLをWebサービスの利用者に任意の方法で提供します。

公開用WSDLを取得する場合は、配備後にInterstage管理コンソールのワークユニット>“IJServer名”>[アプリケーション状態/配備解除]タブでWebモジュールを選択して、[Webサービス環境定義]タブより行ってください。

WebサービスアプリケーションがSTATELESS Session Beanの場合、Interstage管理コンソールの[ワークユニット] > “IJServer名” > [アプリケーション状態/配備解除]タブでEARモジュール、ejb-jarモジュールと選択して、[Webサービス環境定義]タブより取得してください。

ポイント

- 上記で取得できるWSDLは、Webサービスを呼び出すためのURL情報が記述されているほか、J2EEのアプリケーションモジュール標準の形式に整えられています。
- 公開用WSDLでは、一部のインデントなどが整形される場合があります。
- 公開用WSDLには、配備したWARファイルの“WEB-INF/wsdl”配下のファイル、またはejb-jarファイルの“META－INF/wsdl”配下のファイルも含められます。
  WAR/ejb-jarファイルの構成については、“18.1.1 WebサービスアプリケーションのWAR/ejb-jarファイルの構成”を参照してください。

Webサービス環境定義

Webサービス環境定義の項目は、deployment descriptorに記載する項目です。
Webサービスのdeployment descriptorについて詳細は、“18.6.1 webservices.xmlの記述形式”を参照してください。

注意

設定を変更するには、再配備を行います。配備後の設定変更はできません。

Webサービスアプリケーションのモニタリング

Interstage管理コンソールを使用して、Webサービスアプリケーションのメソッドごとに処理時間を参照できます。

Webサービスアプリケーションのモニタリングについて詳細は、Interstage管理コンソールのヘルプを参照してください。

また、SOAPエンジン処理を含めた処理時間をServletの処理時間として参照できますが、こちらはメソッドごとではなくポートごとになります(ポート内のすべてのメソッドに関する合計や平均となります)。

受信するリクエストメッセージサイズ(添付ファイルおよびSOAPメッセージ込み)の上限指定

下記の方法で、Webサービスで受け付けるリクエストのサイズ(添付ファイルおよびSOAPメッセージの合計)を制限できます。
なお、この条件を超えたサイズのリクエストを受信した場合、以下となります。

- Webサービスアプリケーションは呼び出されません。
- Webサービスクライアントには以下のエラーが返却されます(リクエストの形式に依存)。
  - HTTPのエラーメッセージが返却される場合(通常)
    ステータスコード413のエラーメッセージが返却されます。
  - SOAPのエラーメッセージ(Faultメッセージ)が返却される場合
    Faultのエラーメッセージ(FaultString)は、不定。
    container.logに、下記が出力されます。
    - ISWS: ERROR: isws11201: Error occurred.; nested exception is:
      org.xml.sax.SAXParseException: Premature end of file. %s(詳細情報) または
    - その他

ただし、Webサービスで、2ギガバイト(2147483647バイト)以上のリクエストメッセージは受信できません。2ギガバイトを越えるリクエストメッセージを受信した場合は、ステータスコード400でHTTPのエラーメッセージが返却されます。

Interstage管理コンソールからWebサーバのリクエストサイズ制限を設定する

Interstage管理コンソールを使用して、以下の設定項目で、リクエストの上限サイズを指定してください。

ー [サービス] > [Webサーバ] > “Webサーバ名” > [環境設定] > [詳細設定] > [リクエストメッセージ本体の最大サイズ制限]

この制限指定は、Webサービスだけでなく、Webサーバで受け付けるすべてのリクエストに対して有効です。問題がある場合は、次の方法で制限指定を行ってください。

WebサービスのWebアプリケーション単位で、リクエストサイズ制限を設定する

Interstage HTTP Serverの環境定義ファイルの<Location>ディレクティブおよび<LimitRequestBody>ディレクティブを使用して、制限指定を行ってください。
<Location>ディレクティブでWebアプリケーションのパスを指定したうえで、<LimitRequestBody>ディレクティブでリクエストサイズの上限値を指定します。詳細については、“Interstage HTTP Server 運用ガイド”を参照してください。

例

WebサービスのWebアプリケーション(パス:“myws”)のリクエストサイズを制限するInterstage HTTP Serverの環境定義ファイルの記述例(一部抜粋)

```
<Location  /myws>
    LimitRequestBody  1048576
</Location>
```

19.2 Webサービス(クライアント機能)の運用方法

Webサービスクライアントの場合、クライアントアプリケーションの形態によって、運用方法が違います。

Webサービスで運用するクライアントアプリケーションには、以下があります。

- IJServer上で動作するアプリケーション
- J2EEアプリケーションクライアント
- 上記以外のアプリケーション

IJServer上で動作するアプリケーション

Servlet/EJBアプリケーションと同様に、IJServer上で動作するアプリケーションの場合は、作成したアプリケーションとスタブなどをパッケージ化し配備(配備解除)します。

また、IJServerのタイプに応じて以下の設定も必要です。

同一VMの場合

IJServerの環境設定で、コンテナのWebサービス機能を“有効”にします。

上記以外の場合

IJServerの環境設定で、以下のjarファイルをクラスパスおよびJava VMオプションに設定します。

クラスパス

Windows32/64

C:¥Interstage¥J2EE¥lib¥isws.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isws.jar

Java VMオプション

Windows32/64

-XX:EndorsedClassPath=C:¥Interstage¥J2EE¥lib¥isws-saaj-api.jar

Solaris64 Linux32/64

-XX:EndorsedClassPath=/opt/FJSVj2ee/lib/isws-saaj-api.jar

J2EEアプリケーションクライアント

“4.1 JNDIサービスプロバイダの環境設定”および“4.2.1 クライアント環境での環境設定”を参照し、JNDIを利用するために必要な設定を行い、アプリケーションを実行します。

また、クラスパスおよびJava VMオプションに以下を設定する必要があります。

クラスパス

作成したアプリケーション、スタブなど

以下のjarファイル

Windows32/64

C:¥Interstage¥J2EE¥lib¥isws.jar
C:¥Interstage¥J2EE¥lib¥isws-lib.jar
C:¥Interstage¥J2EE¥lib¥isj2ee.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isws.jar
/opt/FJSVj2ee/lib/isws-lib.jar
/opt/FJSVj2ee/lib/isj2ee.jar

Java VMオプション

Windows32/64

-XX:EndorsedClassPath=C:¥Interstage¥J2EE¥lib¥isws-saaj-api.jar

Solaris64 Linux32/64

-XX:EndorsedClassPath=/opt/FJSVj2ee/lib/isws-saaj-api.jar

上記以外のアプリケーション

クラスパスおよびJava VMオプションに以下を設定して、実行します。

クラスパス

作成したアプリケーション、スタブなど

以下のjarファイル

Windows32/64

C:¥Interstage¥J2EE¥lib¥isws.jar

C:¥Interstage¥J2EE¥lib¥isws-lib.jar

Solaris64 Linux32/64

/opt/FJSVj2ee/lib/isws.jar

/opt/FJSVj2ee/lib/isws-lib.jar

Java VMオプション

Windows32/64

-XX:EndorsedClassPath=C:¥Interstage¥J2EE¥lib¥isws-saaj-api.jar

Solaris64 Linux32/64

-XX:EndorsedClassPath=/opt/FJSVj2ee/lib/isws-saaj-api.jar

注意

- Javaバージョン1.3以前の環境では動作しません。
- IJServer上でWebサービスクライアントを運用する場合、該当アプリケーションにHotDeploy機能・オートリロード機能は使用しないでください。

- XMLパーサは、JAXP1.2以上をサポートしているものを使用してください。
- “isws-lib.jar”をクラスパスに設定する場合、以下のjarファイルも自動的にクラスパスに設定されます。

  Windows32/64

  C:¥Interstage¥J2EE¥lib¥xerces¥xercesImpl.jar

  C:¥Interstage¥J2EE¥lib¥xerces¥xml-apis.jar

  C:¥Interstage¥J2EE¥lib¥isj2ee.jar

  Solaris64 Linux32/64

  /opt/FJSVj2ee/lib/xerces/xercesImpl.jar

  /opt/FJSVj2ee/lib/xerces/xml-apis.jar

  /opt/FJSVj2ee/lib/isj2ee.jar

- J2EEアプリケーションクライアントの場合は、“isj2ee.jar”が“isws-lib.jar”より後になるようにクラスパスを設定してください。また、「上記以外のアプリケーション」の場合は“isj2ee.jar”はクラスパスに設定しないでください。

19.2.1 クライアント機能のログ

Interstage Webサービスのクライアント機能は例外をスローすることでエラー情報をユーザアプリケーションに通知します。例外をスローできない場合、IJServer外では標準エラーへの出力またはWebサービスクライアントログファイルにエラー内容を出力します。

注意

IJServer上のWebサービスのログはコンテナログに出力されます。コンテナログについては“3.12 アプリケーションのデバッグ”の“IJServerのログ”を参照してください。

Webサービスクライアントログファイル

IJServer外において、Interstage WebサービスのJavaクライアント機能がエラー情報(例外をスローする場合を除く)を出力するファイルです。このファイルはプロセスごとに用意します。

注意

- 異なるプロセスのWebサービスクライアントが同一ファイルにログを出力する設定となっている場合、ファイルのローテーションに失敗したり、出力メッセージが乱れたりするなど、正常にログが出力されない場合があります。
- 本ファイルの初期化に失敗したとき、(たとえば、Webサービスクライアントログファイルパスとして指定されたパスにアクセス権がない場合や、ディレクトリがない場合など)は、標準エラーにエラー情報を出力します。この場合、以後のログ出力を停止します。

Webサービスクライアントログファイルの指定

Webサービスクライアントログファイルは、Webサービス設定ファイルの以下の項目で設定します。

com.fujitsu.interstage.isws.log.file.path

設定内容は“19.3 Webサービス設定ファイル”の“Webサービスクライアントログファイルパス”を参照してください。

注意

複数のプロセスで共通のWebサービス設定ファイルファイルを利用した場合、複数のプロセスに対し共通のWebサービスクライアントログファイルパスが指定されるため、“Webサービスクライアントログファイル”の注意に記載したとおり、Webサー

ビスクライアントログファイルのローテーションに失敗するなどの異常が発生します。ただし、Webサービスクライアントログファイルパスを指定しない場合(標準エラーで出力する場合)は問題ありません。

19.2.2 スタブ設定ファイル

スタブオブジェクトのプロパティは、スタブのAPIを使用して設定する以外に、スタブ設定ファイルによって設定することもできます。
スタブ設定ファイルによる設定方法について以下に説明します。

なお、スタブオブジェクトに設定できるプロパティについては、"18.2.5 HTTP接続に関する設定"を参照してください。

ポイント

- スタブ設定ファイルで指定したプロパティがスタブオブジェクトに設定されるのは、Serviceオブジェクトからスタブオブジェクトを取得するときです。そのため、アプリケーションはjavax.xml.rpc.Stubインタフェースの_setPropertyメソッドを使用してプロパティを上書きできます。
- Serviceオブジェクトからスタブオブジェクトを取得する際に、メソッドの引数に接続先URLを指定した場合は、スタブ設定ファイルで指定した値より引数に指定した値が優先されます。

スタブ設定ファイルの指定方法

Serviceインタフェースごとに、以下のシステムプロパティでスタブ設定ファイルのパスを指定します。

システムプロパティ名:

```
<Serviceインタフェースの完全修飾クラス名(FQCN)>.configuration
```

例

Serviceインタフェースがcom.example.TestServiceの場合の設定例

```
-Dcom.example.TestService.configuration=C:¥temp¥test.properties
```

スタブ設定ファイルの記述形式

スタブ設定ファイルはプロパティファイル形式で記述します。

プロパティ名、プロパティ値は以下の形式で記述します。ポート名については"ポート名について"を参照してください。

サービス内の特定のポートに対する設定

```
<ポート名>.<スタブに設定するプロパティ名>=<スタブに設定するプロパティ値>
```

サービス内のすべてのポートに対する設定(上記設定が優先されます)

```
default.<スタブに設定するプロパティ名>=<スタブに設定するプロパティ値>
```

例

```
Test.javax.xml.rpc.service.endpoint.address=http://www.example.com/services/Test
Test.javax.xml.rpc.session.maintain=true
```

注意

- プロパティの型がjava.lang.Booleanの場合、値が文字列“true”に等しい(大文字と小文字は区別しない)場合はtrue、それ以外はfalseとなります。
- プロパティの型がjava.lang.Integerの場合、整数値として不正な値を指定した場合は警告メッセージが出力され、そのプロパティは設定されません。
- 存在しないポート名やプロパティ名を指定した場合は無視されます。
- プロパティとして、HTTP Basic認証の認証情報なども指定できます。スタブ設定ファイルに記述する情報に応じて、ファイルのアクセス権限を適切に設定してください。

ポート名について

プロパティ名に使用するポート名は、WSDLに記述されているものではなく、生成されたServiceインタフェースのget<ポート名>というメソッドに使われるポート名の文字列と同じ文字列としてください(Javaで使用可能な名前に変換されたものを使用してください)。

19.2.3 プロキシを経由した接続

Webサービスクライアントアプリケーションからプロキシを経由してWebサービスに接続する場合、以下の項目をシステムプロパティ、またはスタブオブジェクトにString型のプロパティで設定してください。スタブオブジェクトへの設定はシステムプロパティの指定より優先されます。スタブオブジェクトの設定は、“18.2.5 HTTP接続に関する設定”および“19.2.2 スタブ設定ファイル”を参照してください。

キー	設定する値	備考
http.proxyHost	ホスト名	値が設定されない場合は、プロキシを経由せずに接続を行います。
http.proxyPort	ポート番号	
http.nonProxyHosts	プロキシを経由せずに接続するホスト名	「|」区切りで複数のホストを指定できます。
http.proxyUser	ユーザ名	プロキシがベーシック認証を行っている場合に設定してください。
http.proxyPassword	パスワード	

19.3 Webサービス設定ファイル

Webサービス設定ファイルは、Webサービスの設定を行うプロパティファイルです。プロセスごとに本プロパティファイルを用意します。ただし、IJServer上においてプロセス多重の設定を行っている場合、同一IJServerの各プロセスに対して共通のWebサービス設定ファイルを用意します。

設置パスの指定

Webサービス設定ファイルを以下のシステムプロパティに設定したパスに設置します。

システムプロパティ名

com.fujitsu.interstage.isws.configuration

指定可能な値:

Webサービス設定ファイルのパス(カレントディレクトリからの相対パス、またはフルパスで指定)

省略時:

Webサービス設定ファイルをロードしません。

例

システムプロパティ使用例

Webサービス設定ファイル(config.properties)をカレントディレクトリに設置した場合

```
-Dcom.fujitsu.interstage.isws.configuration=config.properties
```

Windows32/64

Webサービス設定ファイル(config.properties)をC:¥tmpに設置した場合

```
-Dcom.fujitsu.interstage.isws.configuration=C:¥tmp¥config.properties
```

Solaris64 Linux32/64

Webサービス設定ファイル(config.properties)を/tmpに設置した場合

```
-Dcom.fujitsu.interstage.isws.configuration=/tmp/config.properties
```

注意

設定したパスにWebサービス設定ファイルが存在しない場合あるいはアクセスできない場合、警告メッセージを出力し、すべての設定項目に省略値を利用して処理を続行します。

設定項目

以下に設定項目を示します。

No.	設定項目	key名	省略可否	IJServer上で有効かどうか
1	Webサービスクライアントログファイルパス	com.fujitsu.interstage.isws.log.file.path	省略可	無効(注)
2	Webサービスクライアントログファイルの最大サイズ	com.fujitsu.interstage.isws.log.file.maxfilesize	省略可	無効(注)
3	Webサービスクライアントログファイルの最大世代数	com.fujitsu.interstage.isws.log.file.maxbackupindex	省略可	無効(注)
4	Webサービスクライアントで使用するSSL定義	com.fujitsu.interstage.isws.client.ssl.configname	省略可	有効
5	添付ファイルの一時ファイル作成場所	com.fujitsu.interstage.isws.attachment.tmpdir	省略可	有効
6	添付ファイル受信時に一時ファイルを生成せずメモリのみで扱うサイズの上限	com.fujitsu.interstage.isws.attachment.memory.cachesize	省略可	有効
7	レスポンス返却時の、Webサービスアプリケーションで受信した添付ファイルデータ削除(資源解放)	com.fujitsu.interstage.isws.attachment.tempfile.keepmode	省略可	有効
8	WSDLでtext/plainに指定された添付ファイルのデフォルト文字コード	com.fujitsu.interstage.isws.attachment.plaintext.charset	省略可	有効
9	非ascii文字の送信形式	com.fujitsu.interstage.isws.utf8encoder.notascii.charref	省略可	有効

No.	設定項目	key名	省略可否	IJServer上で有効かどうか
10	ディレクトリサービスによる認証の有効化	com.fujitsu.interstage.isws.enable.directory.authentication	省略可	有効
11	Webサービスクライアントの接続時タイムアウト時間	com.fujitsu.interstage.isws.client.socket.connection.timeout	省略可	有効

注) IJServer上ではWebサービスクライアントログファイルに関する設定は無効です。

注意

Webサービス設定ファイルは、Javaのプロパティファイルの仕様に従って記述してください。

例) 日本語などの ISO 8859-1 文字エンコーディングで直接表現できない文字はUnicodeエスケープを使用して記載する。

19.3.1 Webサービスクライアントログファイルパス

Key名

```
com.fujitsu.interstage.isws.log.file.path
```

指定可能な値:

出力ファイルパス(カレントディレクトリからの相対パス、またはフルパスで指定)

省略値:

標準エラーに出力します。

使用例:

カレントディレクトリのlog.txtに出力

```
com.fujitsu.interstage.isws.log.file.path=log.txt
```

Windows32/64
C:¥tmp¥log.txtに出力

```
com.fujitsu.interstage.isws.log.file.path=C:¥¥tmp¥¥log.txt
```

Solaris64 Linux32/64
/tmp/log.txtに出力

```
com.fujitsu.interstage.isws.log.file.path=/tmp/log.txt
```

注意

「Webサービスクライアントログファイルパス」の注意事項

- Webサービスクライアントログファイルパスはプロセスごとに異なるファイルを指定してください。詳細は“Webサービスクライアントログファイル”を参照してください。
- ファイル出力先のディレクトリを用意する必要があります。
- 本項目で指定されたパスにアクセス権がない場合、あるいはディレクトリがない場合、標準エラーにエラーメッセージを出力し、処理を続行します。エラーメッセージ出力後のWebサービスクライアントログファイルおよび標準出力へのログ出力はおこないません。

19.3.2 Webサービスクライアントログファイルの最大サイズ

Key名

com.fujitsu.interstage.isws.log.file.maxfilesize

指定可能な値:

1～2048(単位はMB)

省略値:

10

使用例:

Webサービスクライアントログファイルの最大サイズを1024MB(1GB)とします。

```
com.fujitsu.interstage.isws.log.file.maxfilesize=1024
```

注意

「Webサービスクライアントログファイルの最大サイズ」の注意事項

- 本項目はcom.fujitsu.interstage.isws.log.fileが指定されたときのみ有効です。
- 有効でない値を設定した場合、警告メッセージを標準エラー出力し、省略値の10を採用します。

19.3.3 Webサービスクライアントログファイルの最大世代数

Key名

com.fujitsu.interstage.isws.log.file.maxbackupindex

指定可能な値:

1～100

省略値:

5

使用例:

Webサービスクライアントログファイルの最大世代数を10とします。

```
com.fujitsu.interstage.isws.log.file.maxbackupindex=10
```

注意

「Webサービスクライアントログファイルの最大世代数」の注意事項

- 本項目はcom.fujitsu.interstage.isws.log.fileが指定されたときのみ有効です。
- 有効でない値を設定した場合、警告メッセージを標準エラー出力し、省略値の5を採用します。

19.3.4 Webサービスクライアントで使用するSSL定義

Key名

com.fujitsu.interstage.isws.client.ssl.configname

指定可能な値:

Interstage証明書環境に定義されたSSL定義名

省略値:

なし

使用例:

Webサービスクライアントの SSL通信に、定義名“WSClientSSL”のSSL定義を使用します。

```
com.fujitsu.interstage.isws.client.ssl.configname=WSClientSSL
```

19.3.5 添付ファイルの一時ファイル作成場所

Key名

```
com.fujitsu.interstage.isws.attachment.tmpdir
```

説明:

一定以上のサイズの添付ファイルを受信する際、メモリ節約のため、内部的に一時ファイルが作成されます。また、添付ファイルに関するAPIを使用した場合も、一時ファイルが作成される場合があります。

指定可能な値:

ディレクトリの絶対パス
(ローカルシステムでJavaVMプロセスに書き込み権限があり十分な容量があること)

省略値:

IJServer上で動作するアプリケーションの場合: JavaVMのカレント(システムプロパティ:user.dir)
IJServer外で動作するアプリケーションの場合: JavaVMのTEMPフォルダ(システムプロパティ:java.io.tmpdir)
これらのシステムプロパティの値がパスとして不正な場合、添付ファイルの受信に失敗します。

使用例:

Windows32/64
D:¥tmp¥wsdataにプロセスごとにディレクトリが作成され、配下に一時ファイルが作成されます

```
com.fujitsu.interstage.isws.attachment.tmpdir=D:¥¥tmp¥¥wsdata
```

Solaris64 Linux32/64
/var/tmp/wsdataにプロセスごとにディレクトリが作成され、配下に一時ファイルが作成されます

```
com.fujitsu.interstage.isws.attachment.tmpdir=/var/tmp/wsdata
```

注意

「添付ファイルの一時ファイル作成場所」の注意事項

- 作成場所には、呼量と添付ファイルのサイズ上限値に応じて十分なディスク容量を確保して下さい。
- サイズが大きいまたは大量の添付データを受信する場合は、容量が不足しても他に処理に影響を与えないよう、作成場所は独立したドライブまたはパーティションなどを用意する事を推奨します。
- Javaプロセスが異常終了した場合、作成場所に一時ファイルが残ることがあります。その場合は、手動で削除して下さい(Webサービスアプリケーションについては、IJServerで作成場所をカスタマイズしていない場合、カレントディレクトリがワークユニット機能管理下であれば(デフォルト)、一定回数再起動後に自動的に削除されます)。
- 作成場所は、Javaプロセスで書き込み可能なディレクトリである必要があります。またローカルシステム内のディレクトリである必要があります(リモートの共有ディレクトリなどは使用できません)。

・ 指定したディレクトリ配下に“_iswsatttmp*”ディレクトリが作成され、一時ファイルが格納されます。“_iswsatttmp*”ディレクトリ配下に一時ファイル以外のファイルを格納しないでください。

19.3.6 添付ファイル受信時に一時ファイルを生成せずメモリのみで扱うサイズの上限

Key名

com.fujitsu.interstage.isws.attachment.memory.cachesize

指定可能な値:

0～2147483647までの整数(Kバイト単位)。
負の値が指定された場合は0、それ以外の不正な値が指定された場合は、省略値となります。

省略値:

16(Kバイト)

使用例:

64Kバイトまでの添付ファイルは一時ファイルを生成せずメモリのみで扱います。

```
com.fujitsu.interstage.isws.attachment.memory.cachesize=64
```

19.3.7 レスポンス返却時の、Webサービスアプリケーションで受信した添付ファイルデータ削除(資源解放)

Key名

com.fujitsu.interstage.isws.attachment.tempfile.keepmode

説明:

Webサービスアプリケーションが、Webサービスクライアントから受信した添付ファイルデータをレスポンス返却後も保持するかどうかを指定します。

指定可能な値:

trueまたはfalse(大文字と小文字は区別しません)。
true以外の値が指定された場合はfalseになります。
false(または省略値)を指定した場合、WebサービスアプリケーションがWebサービスクライアントから受信したDataHandlerオブジェクトと、そのデータ内容であるDataSourceオブジェクトは、レスポンス返却時に削除(資源解放)されることがあります。その場合、これらのオブジェクトは利用できなくなります。
trueを選択した場合、レスポンス返却後も添付ファイルデータのオブジェクトは保持されます。

省略値:

false

使用例:

レスポンス返却後も添付ファイルデータのオブジェクトを解放しません。

```
com.fujitsu.interstage.isws.attachment.tempfile.keepmode=true
```

注意

「レスポンス返却時の、Webサービスアプリケーションで受信した添付ファイルデータ削除(資源解放)」の注意事項

・ この項目をtrueに設定すると、添付ファイルデータのオブジェクトがGCによって回収されるまでは、一時ファイルは削除されません。そのため、一時ファイル作成場所に十分なディスク容量を確保するとともに、アプリケーションでは添付ファイルデータのオブジェクトの参照を不要に保持しないでください。

- DataSourceオブジェクトが解放される前に、DataHandlerオブジェクトのgetInputStreamメソッドで読み出したデータは、資源解放の対象とはなりません。アプリケーションで任意に利用及び解放が可能です。
- Webサービスクライアントでは、DataSourceオブジェクトの削除(資源解放)は、GCに任されます。

19.3.8 WSDLでtext/plainに指定された添付ファイルのデフォルト文字コード

Key名

com.fujitsu.interstage.isws.attachment.plaintext.charset

説明:

WSDL中のmime:content要素でtype属性がtext/plainとなっている添付ファイルについて、どの文字コードで送信するか、および、受信時にも文字コードが不明な場合にどの文字コードで受信するかを指定します。

指定可能な値:

US-ASCII
ISO-8859-1
UTF-16
など使用するJDKでサポートしている文字コード

省略値:

UTF-8

使用例:

ISO-8859-1で送信し、受信時に文字コードが不明な場合はISO-8859-1として解釈します。

com.fujitsu.interstage.isws.attachment.plaintext.charset=ISO-8859-1

19.3.9 非ascii文字の送信形式

Key名

com.fujitsu.interstage.isws.utf8encoder.notascii.charref

説明:

SOAP通信における文字列データ中の非ascii文字を、そのままの文字で送信するか、XMLの文字参照形式に変換してascii文字を用いて送信するかを指定します。
XML規約に従っている限り、受信側でこの変更によるデータの違いは発生しません。通常は本設定項目を変更する必要はありません。

指定可能な値:

trueまたはfalse(大文字と小文字は区別しません)。
true以外の値が指定された場合はfalseになります。

省略値:

false

使用例:

通信における文字列データ中の非ascii文字をXMLの文字参照形式に変換してascii文字を用いて送信します。

com.fujitsu.interstage.isws.utf8encoder.notascii.charref=true

「非ascii文字の送信形式」の注意事項

- 本設定項目をtrueに指定した場合、JIS X 0213:2004においてJIS X 0208から追加された文字の一部など、Unicodeの基本多言語面以外に割り当てられた文字(Javaのchar型においてサロゲートペアによって表現される文字)は正しく送信できません。
- 本設定項目は、添付ファイル型の添付ファイルデータ内の文字列データの送信形式には影響しません。

19.3.10 ディレクトリサービスによる認証の有効化

Key名

com.fujitsu.interstage.isws.enable.directory.authentication

説明:

JNDIを使用してServiceをlookupするWebサービスクライアントアプリケーションが、ディレクトリサービスが認証した情報を使用してHTTPリクエストの送信を行うかを指定します。本項目にtrueが設定された場合は、“18.2.5 HTTP接続に関する設定”の“Webサービスのユーザ名/パスワードを設定する方法”による設定のかわりに以下をBasic認証のユーザ名/パスワードとして使用します。

- J2EEアプリケーションクライアントの場合:
  JNDI環境プロパティに設定され、JNDIサービスプロバイダで認証されたユーザ名/パスワードを使用します。
- IJServer上のアプリケーションの場合:
  Webアプリケーションで認証されたユーザ名/パスワードを使用します。

指定可能な値:

trueまたはfalse(大文字と小文字は区別しません)。
true以外の値が指定された場合はfalseになります。

省略値:

false

使用例:

ディレクトリサービスが認証したユーザID/パスワードを使用します。

com.fujitsu.interstage.isws.enable.directory.authentication=true

注意

「ディレクトリサービスによる認証の有効化」の注意事項

- ServiceFactoryを使用してServiceオブジェクトを取得する場合は本設定項目は有効になりません。

19.3.11 Webサービスクライアントの接続時タイムアウト時間

Key名

com.fujitsu.interstage.isws.client.socket.connection.timeout

説明:

Webサービスクライアントの接続時のタイムアウト時間を指定します。

指定可能な値:

-2147483648～2147483647までの整数(単位はミリ秒)。
有効でない値を指定した場合は、省略値を採用します。

注意

- 指定値がシステムで設定されているタイムアウト時間を超える場合は、システムの設定値を優先します。
- 0を指定した場合は、システムで設定されているタイムアウト時間を採用します。
- 負の値を指定した場合は、応答タイムアウト時間で指定された値を採用します。ただし、応答タイムアウト時間がシステムで設定されているタイムアウト時間を超える場合は、システムの設定値を優先します。
- システムで設定されているタイムアウト時間を変更する場合は、TCP/IPのパラメタをチューニングしてください。
- 応答タイムアウト時間で指定された値については、“18.2.5 HTTP接続に関する設定”の“応答タイムアウト時間を指定する方法”を参照してください。

省略値:

SSLを使用しない場合は、応答タイムアウト時間で指定された値。
SSLを使用する場合は、システムで設定されているタイムアウト時間。

使用例:

接続時のタイムアウト時間を30秒に設定します。

```
com.fujitsu.interstage.isws.client.socket.connection.timeout=30000
```

第5部 JTS/JTA編 Windows32/64 Linux32/64

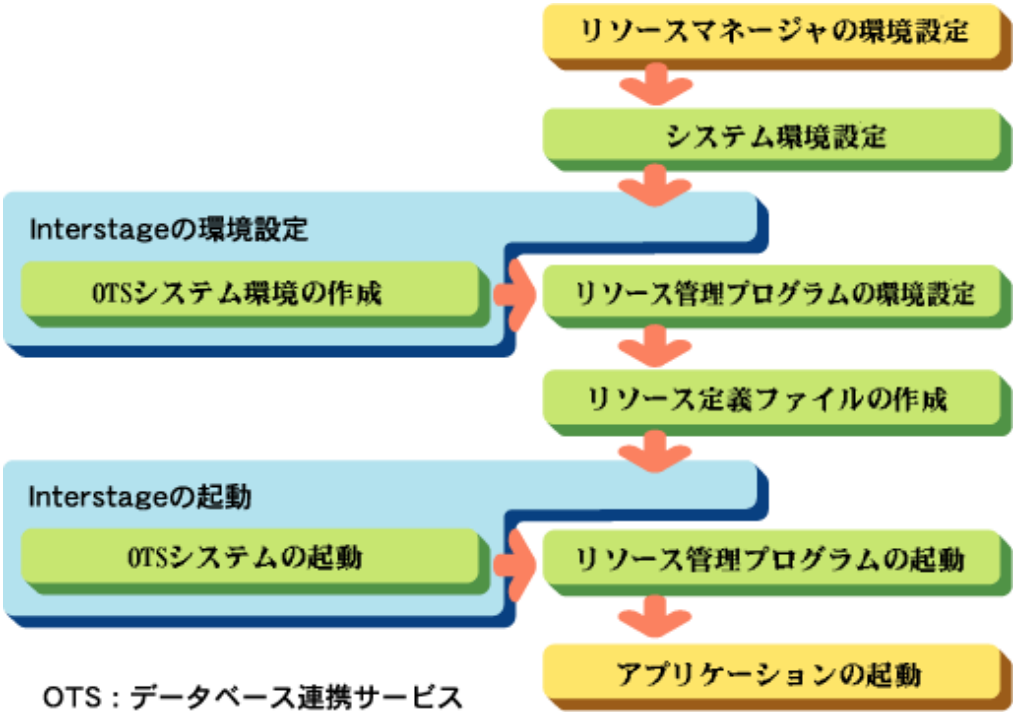
第20章　JTSの運用 Windows32/64 Linux32/64

JTSを使用するためにはデータベース連携サービス(OTSシステム)の環境設定が必要です。
JTSを使用するための環境設定、運用は、Interstage管理コンソールを利用して操作することができます。詳細については、“3.8 JTSを利用する場合の手順 ”を参照してください。

本章では、従来の方法を利用する場合の、環境設定、運用手順について説明します。

20.1 JTSを利用する場合の運用手順

JTSを使用するには、以下の流れにそって環境設定、運用を行います。
JTSは、データベース連携サービスが提供する機能です。以降、環境設定手順に合わせて説明します。



20.2 システム環境設定

以下について説明します。

- システムチューニング
- ディスクパーティションの獲得

システムチューニング

Windows32/64

JTSを使用するには、以下の資源チューニングが必要です。JTSで使用する資源のチューニングは、iniファイル（Interstageインストールフォルダ¥ots¥etc¥ots.ini）で行います。

- 共用メモリ
- セマフォ資源

ー メッセージキュー

ー Windows(R)固有パラメタ

詳細については、“チューニングガイド”の“データベース連携サービスのチューニング”を参照してください。

Linux32/64

以下の資源チューニングが必要です。

ー 共用メモリ

ー セマフォ資源

ー メッセージキュー

詳細については、“チューニングガイド”および“インストールガイド”を参照してください。

ディスクパーティションの獲得 Linux32/64

JTSを使用するためのシステムログファイルを作成するには、ディスクパーティションを作成する必要があります。
ローデバイスの作成法については、各OSのマニュアルを参照してください。
また、システムログファイルについては、“チューニングガイド”の“データベース連携サービスの環境定義”を参照してください。

必要な領域サイズは、以下の計算式で算出してください。

```
トランザクション最大数× X ＋ 1 （ K byte）
```

Xは、次のように求めます。

- 1トランザクションに参加可能なリソース数が4以下の場合：X = 4
- 1トランザクションに参加可能なリソース数が4以上の場合：X = 参加リソース数

20.3 Interstageの環境設定

Interstageの環境設定の詳細については、“運用ガイド(基本編)”を参照してください。
ここでは、データベース連携サービス(OTSシステム)を運用する際に必要な環境設定について説明します。

20.3.1 Interstageの初期化

ここでは、以下について説明します。

- Interstage動作環境定義の生成
- isinitコマンドとotssetupコマンドについて
- ローカルのネーミングサービスを利用する場合(推奨)
- リモートのネーミングサービスを利用する場合

Interstage動作環境定義の生成

Interstageシステム定義ファイルを登録することにより、Interstage動作環境定義が自動的に生成されます。
Interstage動作環境定義の詳細については、“運用ガイド(基本編)”を参照してください。

Interstage動作環境定義の格納先

Windows32/64

```
C:¥Interstage¥td¥etc¥isreg¥isinitdef.txt
```

Linux32/64

```
/opt/FJSVtd/etc/isreg/isinitdef.txt
```

isinitコマンドとotssetupコマンドについて

Interstageの初期化は、isinitコマンドで行います。
isinitコマンドを使用してInterstageの初期化を行うには、Interstage動作環境定義を生成し、必要項目を設定しておく必要があります。
Interstage動作環境定義については、“運用ガイド(基本編)”を参照してください。

データベース連携サービスを利用するには、type2で初期化する必要があります。EJBアプリケーションを利用する場合は、ejbを指定する必要があります。

```
isinit type2 ejb
```

リモートのネーミングサービスを利用する場合は、isinitコマンドにtype3を指定して実行する必要があります。type3では、OTSシステムの環境設定は行われません。
そのため、OTSシステムやリソース管理プログラムの環境設定は、otssetupコマンドを使用して行います。
ここでは、otssetupコマンドによる動作環境の設定と動作環境の削除について説明します。

```
otssetup -f セットアップ情報ファイル
```

OTSシステムを削除する場合は、以下のように指定してコマンドを実行します。

```
otssetup -d
```

ローカルのネーミングサービスを利用する場合(推奨)

データベース連携サービス(OTSシステム)が動作するホストと同じホストのネーミングサービスを利用する場合の設定方法を説明します。
データベース連携サービスに必要な以下の定義項目は、Interstageがシステム規模に合わせて値を設定します。運用環境に合わせて値を変更する場合は、以下の項目の値を変更してください。
ただし、セットアップ種別とシステムログファイル名は、必ず設定する必要があります。

1. Interstage動作環境定義の設定

```
# OTSのセットアップ種別の設定
OTS Setup mode = sys

# OTSシステムのスレッド多重度
OTS Multiple degree=5

# データベース連携サービスのリカバリプロセスの多重度
OTS Recovery=2

# データベース連携サービスで使用するシステムログファイル名を
#必ず設定してください
OTS path for system log=

# データベース連携サービスのトランザクション最大数
# システム規模が small の場合
# システム規模により最大接続クライアント数を指定します
OTS maximum Transaction=50
```

2. Interstageの初期化
   isinitコマンドでInterstageを初期化します。

```
isinit type2 ejb
```

リモートのネーミングサービスを利用する場合

データベース連携サービス(OTSシステム)が動作するホストと異なるホストのネーミングサービスを利用する場合の設定方法を説明します。
リモートのネーミングサービスを利用する場合、isinitコマンドでOTSシステムの初期化は行えません。
最初にネーミングサービスの環境設定を行った後、otssetupコマンドでOTSシステムを初期化する必要があります。

1. Interstage動作環境定義の設定

```
NS USE=remote
NS Host Name=利用するネーミングサービスが動作するホスト名
NS Port Number=利用するネーミングサービスのPort番号
```

2. Interstageの初期化
   isinitコマンドでInterstageを初期化します。

```
isinit type3
```

3. セットアップ情報ファイルの作成

```
# OTSのセットアップ種別の設定
MODE=SYS

# OTSシステムのスレッド多重度
OTS_FACT_THR_CONC=5

# データベース連携サービスのリカバリプロセスの多重度
OTS_RECV_THR_CONC=2

# データベース連携サービスで使用するシステムログファイル名を
#必ず設定してください
LOGFILE=

# データベース連携サービスのトランザクション最大数
# システム規模が small の場合
# システム規模により最大接続クライアント数を指定します
TRANMAX=10
```

4. データベース連携サービスの環境設定
   otssetupコマンドでOTSを初期化します。

```
otssetup -f セットアップ情報ファイル
```

20.3.2 データベース連携サービスの環境定義の設定

データベース連携サービスの環境定義は、運用に合わせて変更する必要があります。

データベース連携サービスの環境定義ファイル格納先

Windows32/64

```
C:¥Interstage¥ots¥etc¥config
```

Linux32/64

```
/opt/FJSVots/etc/config
```

データベース連携サービスの環境定義の詳細については、“チューニングガイド”を参照して下さい。

20.4 Interstageの起動・停止

Interstageの運用の詳細については、“運用ガイド(基本編)”を参照してください。

Interstageの起動

```
isstart
```

Interstageの停止

```
isstop
```

ポイント

Interstageがtype2の運用形態で初期化されている場合は、Interstageの起動により、データベース連携サービスも同時に起動します。

20.5 リソース管理プログラムの環境設定

20.5.1 リソース管理プログラムの環境設定

ここでは、以下について説明します。

- 環境変数
- Java環境の設定

環境変数

JTS用のリソース管理プログラムを使用する場合は、環境変数に以下のクラスライブラリを設定する必要があります。

システムの環境変数classpath(注1)に、以下が指定されていることを確認してください。

- データベースのJDBCドライバ(注2)
- リソースアダプタのクラスライブラリ(注3)

注1)

Windows32/64

環境変数classpathは、必ずシステム環境変数に設定してください。ユーザ環境変数に設定しても動作しません。

Linux32/64

環境変数classpathは、isstartコマンドを実行する前に必ず設定してください。

注2)データベースを利用する場合に設定する必要があります。

注3)Connectorを利用してリソースアダプタと連携する場合に設定する必要があります。

他の環境変数、またはライブラリの格納パスを指定する場合は、RMPプロパティファイルで指定します。RMPプロパティファイルの詳細については、“チューニングガイド”の“データベース連携サービスの環境定義”－“RMPプロパティ”を参照してください。データベース使用時に必要な環境変数の詳細については、“4.3 JDBC(データベース)を参照する場合の環境設定”を参照してください。

Java環境の設定

JTS用のリソース管理プログラムを使用する場合は、以下のconfigファイルにjavaのバージョンとjavaコマンドへのパスをフルパスで指定する必要があります。

格納場所

Windows32/64

```
C:¥Interstage¥ots¥etc¥config
```

Linux32/64

```
/opt/FJSVots/etc/config
```

例

Windows32/64

```
JAVA_VERSION=14
PATH=C:¥Interstage¥JDK8¥bin¥java.exe
```

Linux32/64

```
JAVA_VERSION=14
PATH=/opt/FJSVawjbk/jdk8/bin/java
```

注意

- JAVA_VERSION項目を省略した場合は、自動的に14が設定されます。
- PATH項目は、省略できません。
- JDK/JREと同時にインストールした場合は、自動的に設定されます。

詳細については“チューニングガイド”の“データベース連携サービスの環境定義”を参照してください。

20.5.2 OTSシステムと別ホストで動作するための環境設定

リソース管理プログラムをOTSシステムが動作するホストと別ホスト上で動作させる場合の環境設定について説明します。

リソース管理プログラムは、分散されたアプリケーションを想定して、そのアプリケーションと同じホストで動作することが可能です。

リソース管理プログラムをOTSシステムが動作するホストとは別ホスト上で動作させるには、以下の手順で行ってください。

- ネーミングサービスを共有させる場合(推奨)
- ネーミングサービスを共有させない場合

ネーミングサービスを共有させる場合(推奨)

OTSシステムが動作するホストとリソース管理プログラムを動作させるホストでネーミングサービスを共有して利用する場合

Windows32/64

1. リソース管理プログラムが動作するホストで、Interstage動作環境定義に以下を指定します。

```
NS USE=remote
NS Host Name=利用するネーミングサービスが動作するホスト名
NS Port Number=利用するネーミングサービスのPort番号
```

2. リソース管理プログラムが動作するホストで、isinitコマンドにてtype3を選択して初期化します。
3. リソース管理プログラムが動作するホストで、isstartコマンドにてInterstageを起動します(OTSシステムは、起動されません)。
4. リソース管理プログラムが動作するホストで、netコマンドなどを使用して「ObjectTransactionService」サービスを起動します。

```
例) net start ObjectTransactionService
```

5. OTSシステムが動作するホストで、isstartコマンドにてOTSシステムを起動します。(注1)
6. リソース管理プログラムが動作するホストで、セットアップ情報ファイルに以下を指定します。

```
MODE=RMP (注2)
```

7. リソース管理プログラムが動作するホストで、otssetupコマンドにて初期化します。
8. リソース管理プログラムが動作するホストで、otsstartrscコマンドにてリソース管理プログラムを起動します。

注1)OTSシステムがローカル以外のネーミングサービスを利用する場合は、リソース管理プログラム同様にInterstage動作環境定義に利用するネーミングサービスを設定してInterstageを初期化する必要があります。詳細については、"リモートのネーミングサービスを利用する場合"を参照してください。

注2)HOST／PORTは、指定しないでください。

ネーミングサービスを共有させない場合

OTSシステムが動作するホストとリソース管理プログラムを動作させるホストで異なるネーミングサービスを利用する場合(リソース管理プログラムがローカルのネーミングサービスを利用する場合)

1. リソース管理プログラムが動作するホストで、Interstage動作環境定義ファイルに以下を指定します。

```
OTS Setup mode=rmp
OTS Host= OTSシステムの動作するホスト名
OTS Port= OTSシステムが動作するノードのCORBAサービスのポート番号
OTS Locale=OTSシステムが動作するホストのロケール
```

2. リソース管理プログラムが動作するホストで、isinitコマンドにてtype2を選択して初期化します。
3. リソース管理プログラムが動作するホストで、isstartコマンドにてInterstageを起動します(OTSシステムは、起動されません)。
4. OTSシステムが動作するホストで、isstartコマンドにてOTSシステムを起動します。
5. リソース管理プログラムが動作するホストで、otsstartrscコマンドにてリソース管理プログラムを起動します。

OTSシステムが動作するホストとリソース管理プログラムを動作させるホストで異なるネーミングサービスを利用する場合。(リソース管理プログラムがリモートのネーミングサービスを利用する場合)

1. リソース管理プログラムが動作するホストで、Interstage動作環境定義ファイルに以下を指定します。

```
NS USE=remote
NS Host Name=利用するネーミングサービスが動作するホスト名
NS Port Number=利用するネーミングサービスのPort番号
```

2. リソース管理プログラムが動作するホストで、isinitコマンドにてtype3を選択して初期化します。

3. リソース管理プログラムが動作するホストで、isstartコマンドにてInterstageを起動します(OTSシステムは起動されません)。
4. リソース管理プログラムが動作するホストで、セットアップ情報ファイルを作成し、以下を設定します。

```
OTS Setup mode=rmp
OTS Host= OTSシステムの動作するホスト名
OTS Port= OTSシステムが動作するホストのCORBAサービスのポート番号
OTS Locale=OTSシステムが動作するホストのロケール
```

5. (Windows(R)の場合のみ)リソース管理プログラムが動作するホストで、netコマンドなどを使用して「ObjectTransactionService」サービスを起動します。

```
例) net start ObjectTransactionService
```

6. リソース管理プログラムが動作するホストで、otssetupコマンドにてセットアップ情報ファイルを指定しセットアップします。
7. リソース管理プログラムが動作するホストで、isstartコマンドにてInterstageを起動します(OTSシステムは、起動されません)。
8. OTSシステムが動作するホストで、isstartコマンドにてOTSシステムを起動します。
9. リソース管理プログラムが動作するホストで、otsstartrscコマンドにてリソース管理プログラムを起動します。

ポイント

- ここでは、リソース管理プログラムを別ホストで動作させるために必要な情報だけを記載しています。Interstage動作環境定義ファイルとセットアップ情報ファイルの詳細については、“チューニングガイド”を参照してください。
- OTSシステムがリモートのネーミングサービスを利用する場合の設定は、“リモートのネーミングサービスを利用する場合”を参照してください。

20.6 リソース定義ファイルの作成

リソース定義ファイルは、リソース(データベースなど)ごとに作成し、リソースに接続するための情報など(リソース定義)をテキスト形式で記述します。

20.6.1 JDBC用リソース定義ファイルの作成

例

リソース定義ファイル例

JDBCを利用してデータベースと連携する場合のリソース定義ファイルの設定例を以下に示します。

Windows32/64

```
# database1
name=resource1
rscType=JTS
type=JDBC
lookUpName=jdbc/xads1
initialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
providerURL=file://c:/tmp/JNDI
user=dbuser
password=dbpass
logfileDir=c:¥temp
```

Linux32/64

```
# database1
name=resource1
rscType=JTS
type=JDBC
lookUpName=jdbc/xads1
initialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
providerURL=file:/tmp/JNDI
user=dbuser
password=dbpass
logfileDir=/tmp
```

リソース定義ファイルはテキストエディタで作成します。リソース定義ファイルの書式を以下に説明します。なお、1文字目に“#”を記述された場合はコメント行とみなします。

name=リソース定義名

otssetrscコマンドによって登録された際に、ここに記載されたリソース定義名として登録されます。一度登録されたリソース定義ファイルは、すべてリソース定義名で扱うことが可能になります。リソース定義名は、32文字以内で記述する必要があります。
"JTSRMP"は予約語ですので、リソース定義名に使用できません(一部またはすべてを小文字にしても使用できません)。
通常は、isj2eeadminコマンドで登録するJ2EEリソース定義の接続対象となるリソースの"定義名"を指定することを推奨します。
省略することはできません。

rscType=JTS

"JTS"を指定してください。
省略することはできません。省略した場合は、"OTS"が指定されたものとして扱われるため、JTS用のリソース定義ファイルとして正しく動作しません。

type=JDBC

"JDBC"または"DBMS"(旧バージョンでの指定方法)を指定してください。
省略することはできません。

lookUpName=データソースをバインドした名前

データベースが提供するデータソースをバインドした名前を指定します。
isj2eeadminコマンドで登録するJ2EEリソース定義で設定したデータソース名と同じ値を指定してください。

initialContextFactory=initialContextFactory名

バインドされたデータソースを参照する時に使用するinitialContextFactoruy名を指定します。isj2eeadminコマンドで登録するJ2EEリソース定義で設定したクラス名と同じ値を指定してください。

providerURL=プロバイダURL

バインドされたデータソースを参照する時に使用するprovider URLを指定します。
isj2eeadminコマンドで登録するJ2EEリソース定義で設定したURLと同じ値を指定してください。

user=ユーザ名

リソースと接続する際にユーザ名が必要な場合に指定します。
isj2eeadminコマンドで登録するJ2EEリソース定義によって設定したユーザ名を指定してください。

password=パスワード

リソースと接続する際にパスワードが必要な場合に指定します。
isj2eeadminコマンドで登録するJ2EEリソース定義によって設定したパスワードを指定してください。

logfileDir=ログファイルの格納先ディレクトリ

接続したリソースのトラブル調査を行う場合は、トレースログを採取するディレクトリを指定してください。ディレクトリ名の最後にセパレータは付加しないでください。
通常は指定しません。

ポイント

name、lookUpName、initialContextFactory、providerURL、user、passwordは、isj2eeadminコマンドで登録するJ2EEリソース定義情報と同じものを指定してください。

20.6.2 Connector用リソース定義ファイルの作成(J2EE Connector Architectureを利用する場合)

例

リソース定義ファイル例

リソースアダプタを利用してEISと連携する場合のリソース定義ファイルの設定例を以下に示します。

Windows32/64

```
# resource adapter1
name=RA01
rscType=JTS
type=JCA
lookupName=myEIS
user=rauser
password=rapass
logfileDir=c:¥temp
```

Linux32/64

```
# resource adapter1
name=RA01
rscType=JTS
type=JCA
lookupName=myEIS
user=rauser
password=rapass
logfileDir=/tmp
```

リソース定義ファイルは、テキストエディタで作成します。リソース定義ファイルの書式を以下に説明します。なお、1文字目に“#”を記述された場合は、コメント行とみなします。

name=リソース定義名

otssetrscコマンドによって登録された際に、ここに記載されたリソース定義名として登録されます。一度登録されたリソース定義ファイルは、すべてリソース定義名で扱うことが可能になります。リソース定義名は、32文字以内で記述する必要があります。
"JTSRMP"は予約語ですので、リソース定義名に使用できません(一部またはすべてを小文字にしても使用できません)。
通常は、isj2eeadminコマンドで登録するJ2EEリソース定義の接続対象となるリソースの"定義名"を指定することを推奨します。
省略することはできません。

rscType=JTS

"JTS"を指定してください。
省略することはできません。

type=接続するリソースの種類

"JCA"を指定してください。
省略することはできません。

lookUpName=リソース名

リソースアダプタを配備する際に設定した"リソース名"を指定してください。

user=ユーザ名

リソースと接続する際にユーザ名が必要な場合に指定します。リソースアダプタ配備時または定義変更時に設定したユーザ名を指定してください。

password=パスワード

リソースと接続する際にパスワードが必要な場合に指定します。リソースアダプタ配備時または定義変更時に設定したパスワードを指定してください。

logfileDir=ログファイルの格納先ディレクトリ

接続したリソースのトラブル調査を行う場合は、トレースログを採取するディレクトリを指定してください。ディレクトリ名の最後にセパレータは付加しないでください。
通常は指定しません。

ポイント

name、lookUpName、initialContextFactory、providerURL、user、passwordは、isj2eeadminコマンドで登録するJ2EEリソース定義情報と同じものを指定してください。

20.7 リソース管理プログラムの運用

20.7.1 リソース管理プログラムの起動・停止

リソース管理プログラムを起動するには、otstartrscコマンドを実行します。このコマンドはリソース管理プログラムを起動するマシン上で行う必要があります。以下のコマンドを実行する前に、CORBAサービスが起動している必要があります。
また、リソース管理プログラムを停止するには、otsstoprscコマンドを実行します。

例

起動例

```
otsstartrsc -j
```

停止例

```
otsstoprsc -j
```

20.7.2 リソース管理プログラムの運用状態の確認

リソース管理プログラムの運用状態は、otsaliveコマンドを使用して確認します。

```
otsalive
```

本コマンドにより、本コマンドを実行したマシンで動作している以下のサービスに関する情報を表示します。

- OTSシステム
- リソース管理プログラム

例

動作中の場合

```
#otsalive
------------------------------------------------------------
OTS system                      START-TIME 2003/04/01 16:23:14
JTS Resource   JTSRMP           START-TIME 2003/04/01 16:24:59
------------------------------------------------------------
```

動作していない場合

```
#otsalive
------------------------------------------------------------
Nothing
------------------------------------------------------------
```

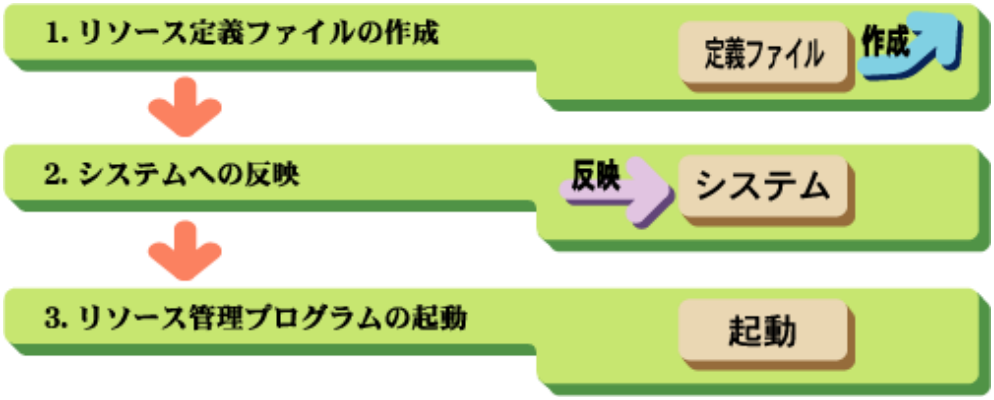
20.7.3 リソース定義ファイルの変更

リソース定義ファイルの変更には、以下の作業があります。

- リソース定義ファイルの追加
- リソース定義ファイルの削除
- リソース定義ファイルの変更

リソース定義ファイルの追加

リソース定義ファイルを追加する手順について説明します。



1. リソース定義ファイルの作成
   新規に追加するリソース定義ファイルを作成します。
   リソース定義ファイルの作成方法の詳細については、“20.6 リソース定義ファイルの作成”を参照してください。

2. システムへの反映
   otssetrscコマンドにより、リソース定義ファイルをシステムに反映します。

```
otssetrsc -a -rf リソース定義ファイル
```

3. リソース管理プログラムの起動
   リソース管理プログラムを起動します。

```
otsstartrsc -j
```

リソース定義ファイルの削除

リソース定義ファイルを削除する手順について説明します。

1. リソース定義ファイルの削除
   対象となるリソース定義ファイルを削除します。

```
otssetrsc -d -n リソース定義名
```

リソース定義ファイルの変更

リソース定義ファイルの変更方法を以下に示します。

リソース定義ファイルを修正した後、otssetrscコマンドを使用してリソース定義を再登録してください。再登録を行わない場合、リソース定義ファイルの修正は反映されません。

```
otssetrsc -o -rf リソース定義ファイル
```

ポイント

登録済みのリソース定義ファイルのリストを表示するには、otssetrscコマンドの-lオプションを利用します。

```
otssetrsc -l
```

各リソース定義ファイルの情報を参照するには、-nオプションにリソース定義名を指定して確認します。

```
otssetrsc -l -n リソース定義ファイル名
```

第21章　JTAの使用方法 Windows32/64 Linux32/64

JTA（Java Transaction API）は、トランザクションサービスによって提供される分散トランザクション操作APIです。
Interstageではトランザクションサービスとして JTS(Java Transaction Service)を提供します。

21.1 JTAについて

JTAの各インタフェース

JTA仕様は、標準Javaインタフェースとして旧Sun(現オラクル社)によって提唱されました。
分散トランザクションに参加するコンポーネントは、J2EEアプリケーションやリソースマネージャになります。
JTAを利用することによって、複数のコンポーネントを連携させて一つのトランザクションとして扱うことが可能になります。

JTAには以下のインタフェースが定義されています。

インタフェース	機能概要
javax.transaction.UserTransaction	トランザクションの開始や完了を指示するためのインタフェースです。このインタフェースはアプリケーションによって使用することが可能です。
javax.transaction.TransactionManager	トランザクションマネージャによって使用されるインタフェースです。通常、このインタフェースはアプリケーションから使用することができません。
javax.transaction.Transaction	トランザクションを論理的に扱うためのインタフェースです。通常、このインタフェースはアプリケーションから使用することができません。
javax.transaction.Synchronization	トランザクションの完了と同期処理を行うためのインタフェースです。通常、このインタフェースはアプリケーションから使用することができません。
javax.transaction.Status	トランザクションの状態を定義するインタフェースです。UserTransactionのgetStatusメソッドで返される値は、このインタフェースで定義されています。
javax.transaction.xa.XAResource	トランザクションマネージャとリソースマネージャ間のインタフェースです。通常、このインタフェースはアプリケーションから使用する必要はありません。
javax.transaction.xa.Xid	リソースマネージャがトランザクションを扱うための識別子インタフェースです。通常、このインタフェースはアプリケーションから使用する必要はありません。

注意

アプリケーションで利用できるJTAは、UserTransactionインタフェースになります。
アプリケーション内でTransactionManagerインタフェースは利用できません。

21.2 UserTransactionインタフェース

UserTransactionインタフェースは、JTA(Java Transaction API)に含まれるインタフェースです。

21.2.1 UserTransactionインタフェースの機能

UserTransactionインタフェースには、以下の機能があります。

メソッド名	メソッド概要
begin	トランザクションを開始します。また、トランザクションは、このメソッドを発行したスレッドと関連付けられます。すでにトランザクションがスレッドに関連付けられている場合は、NotSupportedExceptionが発生します。
commit	スレッドに関連付けられた該当トランザクションをコミットします。 コミット発行後に、処理を取り消すことはできません。
getStatus	スレッドに関連付けられたトランザクションの状態を取得します。
rollback	スレッドに関連付けられた該当トランザクションをロールバックします。 ロールバックを発行することにより、処理前の状態に戻ります。
setRollbackonly	該当トランザクションをロールバックだけ可能な状態にします。
setTransactionTimeout	トランザクションタイムアウトを設定します。 トランザクションタイムアウトは、トランザクション開始前に設定します。

注意

- setTransactionTimeoutメソッドは、分散トランザクションを使用する場合に有効となります。ローカルトランザクションで本メソッドを使用する場合、設定値は無効となります。
- setTransactionTimeoutメソッドでトランザクションタイムアウトを設定する場合、トランザクション開始前に設定してください。トランザクション開始後は、トランザクションタイムアウトを設定しても、開始済みのトランザクションに適用されません。

21.2.2 UserTransactionインタフェースを利用するための環境設定

EJBアプリケーション以外のJ2EEアプリケーション内でUserTransactionインタフェースを利用するには、クラスパスに以下のクラスライブラリを設定する必要があります。

- fjtsclient.jar (JTSクライアント用クラスライブラリ)
- isj2ee.jar (J2EEのクラスライブラリ)
- CORBAサービスのクラスライブラリ
- EJBサービスのクラスライブラリ(EJBクライアントアプリケーションを作成する場合)

注意

JTSのクライアント用クラスライブラリについて

JTSのクライアント用クラスライブラリは、サーバ機能にインストールされます。クライアント機能ではインストールされません。クライアント機能をインストールした環境でJTAを利用したアプリケーションを運用するには、サーバ機能でインストールした環境から、JTSのクライアント用クラスライブラリをコピーしてくる必要があります。
サーバ機能をインストールした環境では、クライアント用クラスライブラリは以下に格納されています。

Windows32/64

```
C:¥Interstage¥ots¥lib¥fjtsclient.jar
```

Linux32/64

```
/opt/FJSVots/lib/fjtsclient.jar
```

注意

UserTransactionインタフェースを利用する場合について

EJBアプリケーション内でUserTransactionインタフェースを利用する場合は、環境変数クラスパスにJTS用クラスライブラリを設定しないでください。

21.2.3 UserTransactionインタフェースの獲得方法

ここでは、UserTransactionインタフェースを利用するために、JNDIからUserTransactionオブジェクトを獲得する方法を説明します。
UserTransactionオブジェクトをJNDIから獲得するには、JNDI環境プロパティを設定する必要があります。
JNDI環境プロパティの詳細については、“第4章 JNDI”を参照してください。

JNDIからUserTransactionオブジェクトを獲得するためのJNDI名は、以下になります。

```
java:comp/UserTransaction
```

例

```
InitialContext ic = new InitialContext();
UserTransaction ut = ic.lookup("java:comp/UserTransaction");
```

注意

JNDI環境プロパティが正しく設定されていない場合、lookup処理が失敗します。
lookup処理で失敗する場合は、JNDI環境プロパティの設定を確認してください。

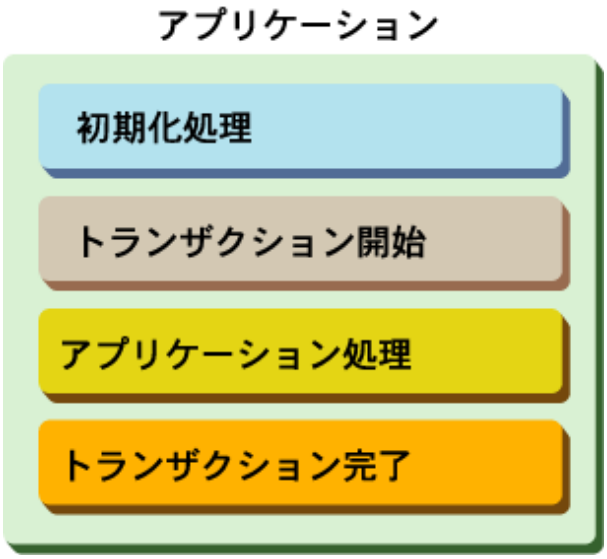
21.3 JTAを利用したアプリケーションの作成

UserTransactionインタフェースの使用方法

ここでは、UserTransactionインタフェースを利用したアプリケーションを作成する方法について説明します。

21.3.1 アプリケーションの構成

これらの処理を考慮したクライアントアプリケーションの構成について以下に示します。



初期化処理

JNDIを利用してUserTransactionオブジェクトを獲得します。

トランザクションの開始

トランザクションを開始します。

アプリケーション処理部

ビジネスロジックを記述します。

トランザクションの完了

トランザクションをコミット、またはロールバックします。

21.3.2 初期化処理とUserTransactionオブジェクトの獲得

InitialContextの生成し、javax.transaction.UserTransactionオブジェクトを取得します。

JNDIからUserTransactionオブジェクトを獲得するためのJNDI名は、以下になります。

```
java:comp/UserTransaction
```

例

処理の記述例を以下に示します。

```
/* 初期化処理  */
javax.transaction.UserTransaction ut = null;
javax.naming.Context ic = null;

// InitialContextの生成
try{
    ic = new InitialContext();
} catch( NamingException e ) {
    System.out.println( "error: new InitialContext()" );
    System.exit(1);
```

```
}

// UserTransactionの獲得
try {
    ut = (UserTransaction)ic.lookup("java:comp/UserTransaction");
} catch( NamingException e ) {
    System.out.println( "error: lookup UserTransaction" );
    System.exit(1);
}
```

21.3.3 トランザクションの開始から完了まで

UserTransactionインタフェースを利用してトランザクションを開始するには、beginメソッドを発行します。

UserTransactionインタフェースを利用してトランザクションを完了するには、commitメソッド／rollbackメソッドを発行します。commitメソッドは処理を確定し、rollbackメソッドは処理を取り消します。

例

処理の記述例を以下に示します。

```
// UserTransactionの獲得
try {
    ut = (UserTransaction)ic.lookup("java:comp/UserTransaction");
} catch( NamingException e ) {
    System.err.println( "error: lookup UserTransaction" );
    System.exit(1);
}
try {
    ut.begin();
} catch (NotSupportedException e) {
    System.err.println("The thread is already associated with a transaction ");
    System.exit(1);
} catch(SystemException e) {
    System.err.println("The system error has been encountered");
    System.exit(1);
}

try {
    //ビジネスロジックを記述します。
} catch(Throwable e) {
    // ここでは例として処理が失敗した場合にフラグをfalseに設定しています。
    commitRequest = false;
}

if(commitRequest) {
    try {
        ut.commit();
    } catch(RollbackException e) {
        System.err.println("The transaction has been rolled back rather than committed");
    } catch(SystemException e) {
        System.err.println("The system error has been encountered.");
    }
} else {
    try {
        ut.rollback();
    } catch(java.lang.IllegalStateException e) {
        System.err.println("The current thread is not associated with a transaction.");
    } catch(SystemException e) {
        System.err.println("The system error has been encountered.");
```

```
    }
}
```

21.3.4 JTAを利用したアプリケーション例

```
/* 初期化処理  */
javax.transaction.UserTransaction ut = null;
javax.naming.Context ic = null;

// InitialContextの生成
try{
    ic = new InitialContext();                                          (1)
} catch( NamingException e ) {
    System.out.println( "error: new InitialContext()" );
    System.exit(1);
}

// UserTransactionの獲得
try {
    ut = (UserTransaction)ic.lookup("java:comp/UserTransaction");       (2)
} catch( NamingException e ) {
    System.err.println( "error: lookup UserTransaction" );
    System.exit(1);
}
try {
    ut.begin();                                                         (3)
} catch (NotSupportedException e) {
    System.err.println("The thread is already associated with a transaction ");
    System.exit(1);
} catch(SystemException e) {
    System.err.println("The system error has been encountered");
    System.exit(1);
}

try {
    // ビジネスロジックを記述します。                                   (4)
} catch(Throwable e) {
    // ここでは、例として処理が失敗した場合にフラグをfalseに設定しています。
    commitRequest = false;
}

if(commitRequest) {                                                     (5)
    try {
        ut.commit();                                                    (6)
    } catch(RollbackException e) {
        System.err.println("The transaction has been rolled back rather than committed");
    } catch(SystemException e) {
        System.err.println("The system error has been encountered.");
    }
} else {
    try {
        ut.rollback();                                                  (7)
    } catch(java.lang.IllegalStateException e) {
        System.err.println("The current thread is not associated with a transaction.");
    } catch(SystemException e) {
        System.err.println("The system error has been encountered.");
    }
}
```

1. JNDIを使用するために、InitialContextを生成します。
2. JNDIからjavax.transaction.UserTransactionオブジェクトを獲得します。

3. javax.transaction.UserTransaction.beginメソッドでトランザクションを開始します。
4. ビジネス処理を記述します。
5. コミット可能かを確認して、トランザクションの状態を決定します。
6. トランザクションを確定終了させる場合は、commitメソッドでトランザクションをコミットさせます。
7. 異常によりコミットすべきではない場合、rollbackメソッドでトランザクションをロールバックさせます。

ポイント

JNDIの詳細については、“第4章 JNDI”を参照してください。

21.3.5 注意事項

クライアントアプリケーションの起動に失敗した場合

環境変数に必要な情報が設定されているかを確認してください。

- fjtsclient.jar (JTSクライアント用クラスライブラリ)
- isj2ee.jar (J2EEのクラスライブラリ)
- CORBAサービスのクラスライブラリ
- EJBサービスのクラスライブラリ(EJBクライアントアプリケーションを作成する場合)

JNDIが必要とする環境プロパティについては、“第4章 JNDI”を参照してください。

クライアントアプリケーションがエラーを検出した場合

クライアントアプリケーションが以下のようなエラーを検出した場合、rollbackを発行してトランザクションを終了してください。

- 自プログラム内の異常を検出した場合
- サーバアプリケーションからエラーが通知された場合

第6部 JMS編

第22章　Interstage JMSの基本機能

本章では、Interstage JMSの基本機能について説明します。

Interstage JMSで使用する主な用語について、以下に説明します。

Message（メッセージ）

JMSで送受信する利用者のデータをメッセージと呼びます。

Publish/Subscribeメッセージングモデル

複数の受信者に対して同一のメッセージを配信する1対nのメッセージングモデルです。

Publisher（パブリッシャ）

JMSでは、Publish/Subscribeメッセージングモデルにおけるメッセージの送信者をパブリッシャと呼びます。

Subscriber（サブスクライバ）

JMSでは、Publish/Subscribeメッセージングモデルにおけるメッセージの受信者をサブスクライバと呼びます。

Durable Subscription機能

アプリケーションがアクティブではない間に送信されたメッセージを、アプリケーションがアクティブになった後に受信できる機能です。

durable Subscriber（持続性のあるサブスクライバ）

Durable Subscription機能を持つサブスクライバを持続性のあるサブスクライバと呼びます。

Message Listener（メッセージリスナ）

メッセージが受信者に到達した際、自動的に処理する機能です。

Point-To-Pointメッセージングモデル

特定の受信者に対してメッセージを配信する1対1のメッセージングモデルです。

Sender（センダ）

JMSでは、Point-To-Pointメッセージングモデルにおけるメッセージの送信者をセンダと呼びます。

Receiver（レシーバ）

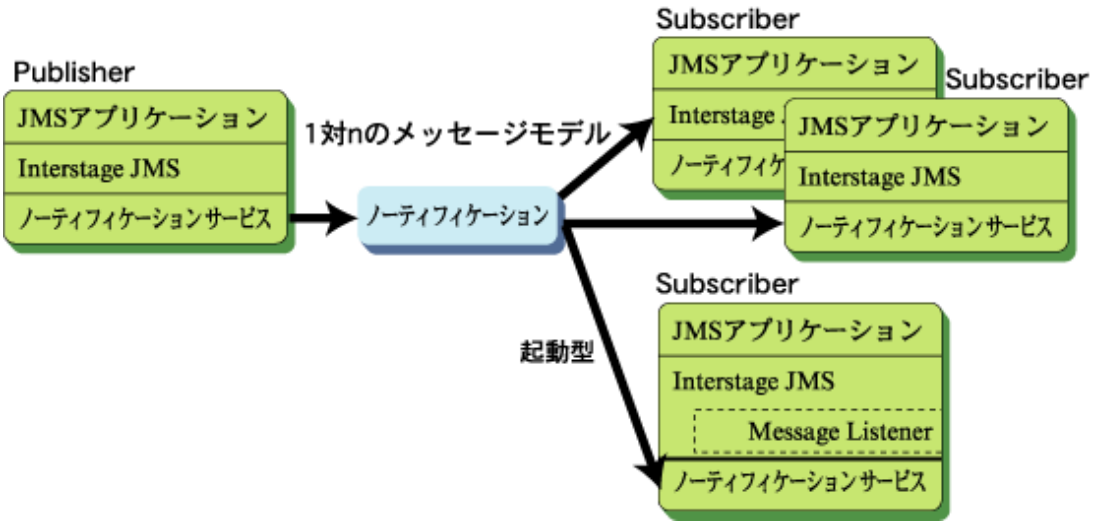
JMSでは、Point-To-Pointメッセージングモデルにおけるメッセージの受信者をレシーバと呼びます。

22.1 Publish/Subscribeメッセージングモデル（1対nメッセージングモデル）

Interstage JMSでは、Publish/Subscribeメッセージングモデルにおいて、以下の機能を提供しています。

- 複数の受信者に対して同一のメッセージを配信
  ノーティフィケーションサービスを介して、送信アプリケーション（Publisher）と受信アプリケーション（Subscriber）が連携します。
- 受信アプリケーションの形態として待機型とMessage Listenerを使用した起動型をサポート
- ノーティフィケーションサービスによるメッセージの優先度および生存時間の制御

なお、Publish/Subscribeメッセージングモデル、Point-To-Pointメッセージングモデルは、イベントチャネル作成時（esmkchnl）に指定できます。



Publish/Subscribeメッセージングモデルを使用した場合の効果を以下に示します。

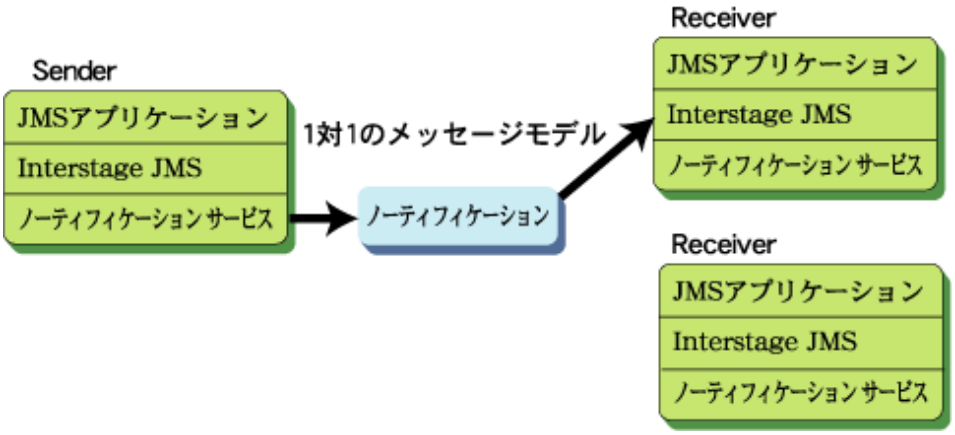
- 複数のSubscriberに配信
  Publisherは、1つの宛先にメッセージを送信することで、接続しているSubscriberの数を意識することなく、複数のSubscriberに同一のメッセージを配信できます。
- 運用変更によるアプリケーションへの影響が少ない
  Subscriber数の増減に対して、Publisherの変更は不要です。

22.2 Point-To-Pointメッセージングモデル（1対1メッセージングモデル）

Interstage JMSでは、Point-To-Pointメッセージングモデルにおいて、以下の機能を提供しています。

- 1拠点から1拠点への非同期通信
  ノーティフィケーションサービスを介して、送信アプリケーション（Sender）と受信アプリケーション（Receiver）が連携します。
  複数のReceiverが接続している場合、メッセージを受信できるのは1つのReceiverだけで、メッセージは自動的に振り分けられます。
- 受信アプリケーションの形態として待機型とMessage Listenerを使用した起動型をサポート
- ノーティフィケーションサービスによるメッセージの優先度および生存時間の制御

なお、Publish/Subscribeメッセージングモデル、Point-To-Pointメッセージングモデルは、イベントチャネル作成時（esmkchnl）に指定できます。



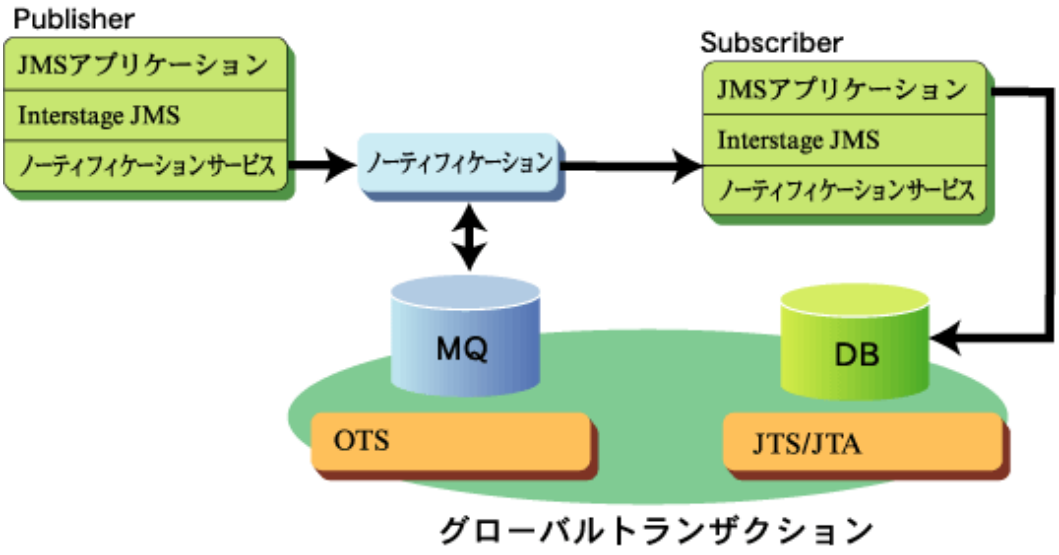
Point-To-Pointメッセージングモデルを使用した場合の効果を以下に示します。

- システム処理性能の向上
  プロセス多重またはスレッド多重でReceiverの多重度を増加させることにより、業務アプリケーションのシステム性能を向上させることができます。Receiverがメッセージを受信した後にデータベース処理などを並行して動作させることで、システム性能の向上につながります。
- 運用性向上
  Publish/Subscribeメッセージングモデルでは、createSubscriberによりSubscriberを作成する前に配信されたメッセージは破棄されます。しかし、Point-To-Pointメッセージングモデルでは、Receiverの状態にかかわらず配信されたメッセージが蓄積され、createReceiverによりReceiverを作成した後に受信できます。

22.3 メッセージ保証

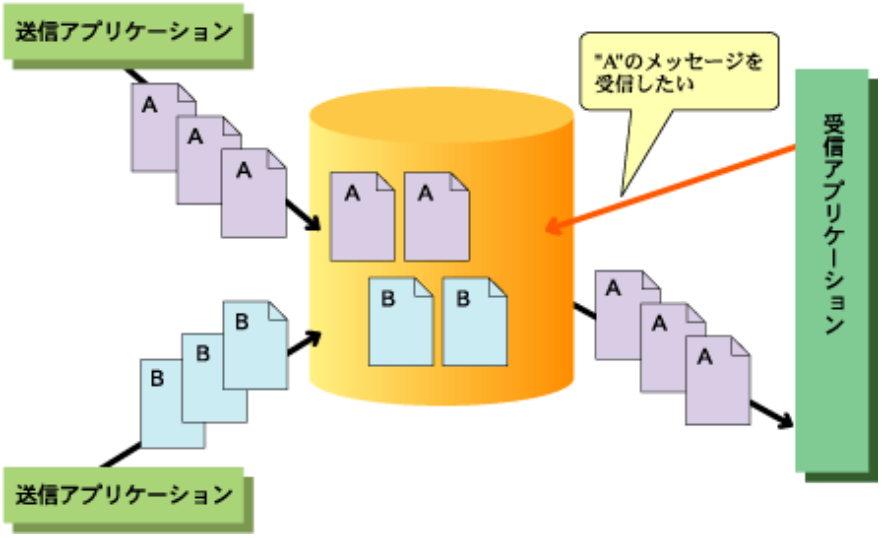
イベントチャネルの不揮発化機能およびローカルトランザクション機能により、メッセージの重複／欠落を防止します。

グローバルトランザクション機能により、メッセージの送受信とDB更新処理の整合性を保証します。



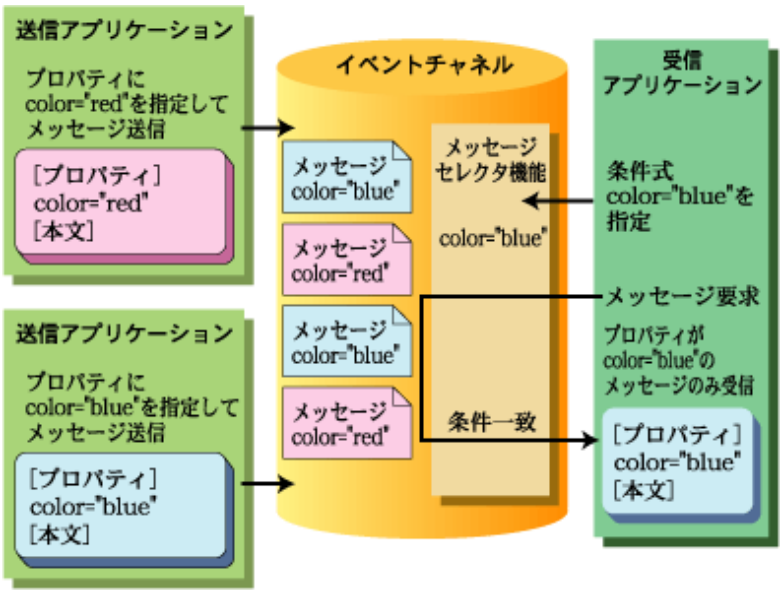
22.4 メッセージセレクタ機能

メッセージセレクタ機能は、送信アプリケーションから送られる多様なメッセージの中から、受信アプリケーションが受信したいメッセージを指定して受け取ることができる機能です。



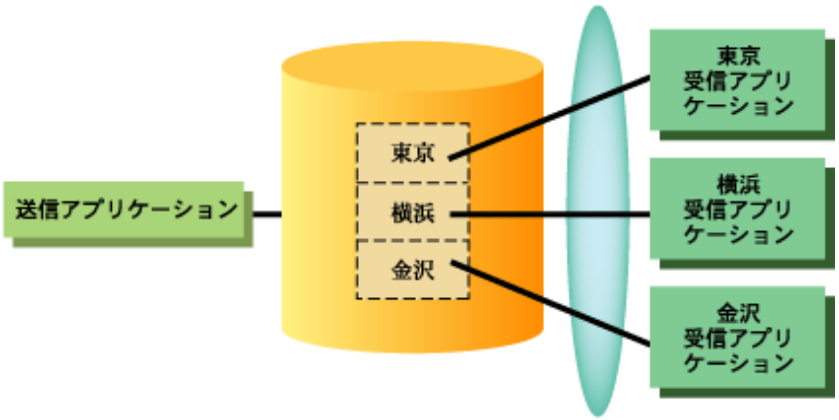
メッセージセレクタ機能の動作を以下に説明します。

- 送信アプリケーションでは、メッセージを識別/分類するための情報をメッセージのプロパティフィールドに設定し、メッセージを送信します。
- 受信アプリケーションでは、受信したいメッセージを特定するための条件式を受信開始時に指定します。以降、受信アプリケーションは、条件式に一致したプロパティを持つメッセージだけを受信します。
  条件式については、“24.10.1 メッセージセレクタ条件式”を参照してください。
- Publish/Subscribeメッセージングモデルでは、条件式に一致しないメッセージは破棄され、Point-To-Pointメッセージングモデルでは、条件式に一致しないメッセージは読み飛ばされます。



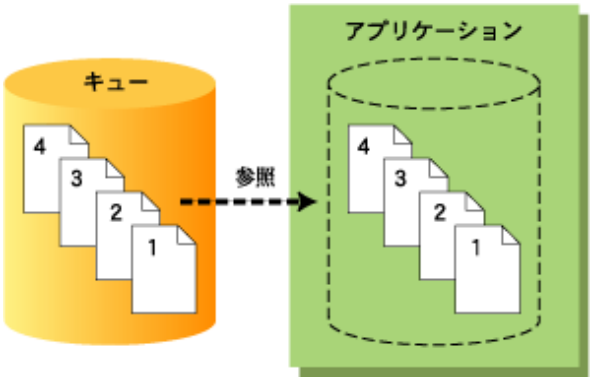
メッセージセレクタ機能を使用した場合の効果を以下に示します。

- 受信アプリケーションが興味のある情報だけを取得可能
  送信アプリケーションが送信したすべてのメッセージを受信することなく、受信アプリケーションが興味のあるメッセージだけを受信でき、運用性が向上します。
  たとえば、送信アプリケーションが国際/社会/政治/経済/天気予報/スポーツ/芸能等のジャンルのニュースを送信する場合、ある受信アプリケーションは、天気予報/スポーツなどの特定の情報だけを受信できます。
- イベントチャネル数の削減およびシステム資源の削減
  従来、あるメッセージを特定の拠点だけに送信する場合、拠点ごとにイベントチャネルを作成しておく必要があり、拠点が増加するにつれ、システム資源を使用してシステムを圧迫してきました。
  メッセージセレクタ機能を使用して各拠点の受信アプリケーションが受信メッセージを選択できるようになるため、以下のようにイベントチャネルを1つにすることができます。これにより、拠点が増加した場合のシステム資源を削減できます。



22.5 キューブラウザ機能

キューブラウザ機能は、アプリケーションがキューに蓄積されているメッセージをブラウジングするための機能です。

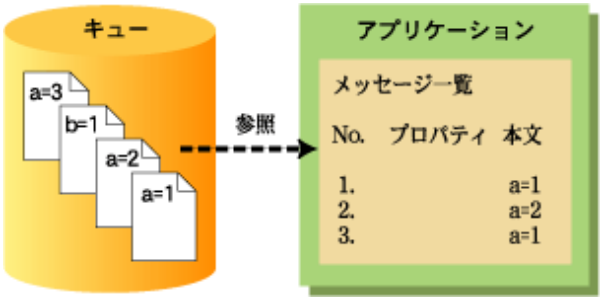


キューブラウザ機能の動作を以下に説明します。

- アプリケーションでは、キューに蓄積されているメッセージを参照するためにBrowserを作成します。
- その後、順にメッセージを取り出すことができ、キューの内容をブラウジングできます。

・ ブラウジング中は、キューからメッセージを取り出しても、メッセージは削除されません。

キューブラウザ機能を使用することにより、キューの内容を表示する機能を持ったアプリケーションの開発が容易になります。



第23章　Interstage JMSの環境設定

本章では、Interstage JMSを使用するための環境設定について説明します。

ポイント

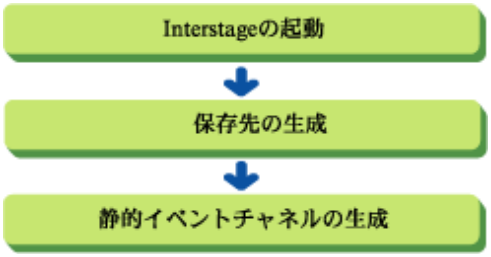
イベントサービス(ノーティフィケーションサービス)で使用する用語とInterstage JMSで使用する用語の対応関係を下表に示します。

イベントサービス	Interstage JMS
イベントチャネル	トピック／キュー
静的生成運用のイベントチャネル	トピック／キュー
動的生成運用のイベントチャネル	テンポラリトピック／テンポラリキュー
コンシューマ	サブスクライバ／レシーバ／コンシューマ
サプライヤ	パブリッシャ／センダ／プロデューサ
イベントデータ	メッセージ

23.1 イベントチャネル運用マシンの運用前の環境設定

ここでは、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルの運用を行うマシンの環境設定について説明します。

運用前の環境設定の手順を以下に示します。



23.1.1 Interstageの起動

Interstage管理コンソールまたはisstartコマンドで、Interstageを起動してイベントサービスを起動します。

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

注意

Interstage管理コンソールの[システム] > [状態]で[Interstage構成サービス]に“イベントサービス”が登録されていない場合は、[システム] > [環境設定]で[JMS]または[イベントサービス]を“使用する”に変更し、構成を変更してください。

■isstartコマンドを使用する場合

isstartコマンドで、Interstageを起動してイベントサービスを起動します。

```
isstart
```

isstartコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“Interstage統合コマンド”－“isstart”を参照してください。

23.1.2 保存先の生成

以下の機能を使用する場合、Interstage管理コンソールまたはesmkunitコマンドで、メッセージの保存先(ユニット)を作成します。

- Durable Subscription機能
- イベントチャネルの不揮発化機能(メッセージ保証)
- ローカルトランザクション機能(メッセージ保証)
- グローバルトランザクション機能(メッセージ保証)

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

■esmkunitコマンドを使用する場合

esmkunitコマンドで、メッセージの保存先(ユニット)を作成します。

例

ユニット定義ファイル“unit1.def”で、ユニットを生成する場合

```
esmkunit -uf unit1.def
```

ユニット定義ファイルは、以下のディレクトリにデフォルトのファイルが格納されています。ユニット定義ファイルの設定項目を必要に応じて編集して、メッセージの保存先(ユニット)を作成してください。

Windows32/64 (インストールパスはデフォルト)

```
C:¥Interstage¥eswin¥etc¥def
```

Solaris64 Linux32/64 (インストールパスはデフォルト)

```
/opt/FJSVes/etc/def
```

esmkunitコマンドおよびユニット定義ファイルの項目の詳細については、“リファレンスマニュアル(コマンド編)”の“イベントサービス運用コマンド”－“esmkunit”を参照してください。

注意

ユニット定義ファイルの以下の設定項目は、正確に見積もった値を設定するように注意してください。

設定項目	内容
unitmode	ユニットのモードを指定します。 ・ std:標準ユニット ・ ext:拡張ユニット
tranmax	ユニット内のイベントチャネルに対して、同時にトランザクション運用を行う多重度を指定します。
tranunitmax Solaris64 Linux32/64	1トランザクション内で操作することができる最大メッセージサイズをブロック数で指定します(1ブロック:16Kバイト(固定))。
syssize	システム(ユニット制御)用ファイルの容量(Mバイト)を指定します。
sysqnum	システム(ユニット制御)用データ格納域の数を指定します。
usersize	イベントデータ(メッセージ)用ファイルの容量(Mバイト)を指定します。
userqnum	イベントデータ(メッセージ)用データ格納域の数を指定します。
shmmax	ユニットで使用する共用メモリサイズ(Mバイト)を指定します。

23.1.3 静的イベントチャネルの生成

Interstage管理コンソールまたはesmkchnlコマンドで、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルを作成します。

注意

- グローバルトランザクション機能を使用する場合は、データベース連携サービスが起動されている必要があります。データベース連携サービスの起動については、“第20章 JTSの運用 ”を参照してください。
- データベース連携サービスを使用する場合、SSL連携は使用できません。
- イベントチャネルの作成時、初期状態としてイベントチャネルの自動起動が設定され、Interstageの起動(イベントサービスの起動)時に自動的に起動されます。イベントチャネルの自動起動は、Interstage管理コンソールを使用して変更できます。
- SSLアクセラレータを使用してイベントチャネルが存在するマシンにSSL環境を構築せずに運用する場合、Interstage管理コンソールを使用してイベントチャネルを作成できません。esmkchnlコマンドを使用してイベントチャネルを作成してください。esmkchnlコマンドについては、“リファレンスマニュアル(コマンド編)”を参照してください。

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

■esmkchnlコマンドを使用する場合

esmkchnlコマンドで、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルを作成します。

例

グループ“mygroup”のイベントチャネルの不揮発運用・ローカルトランザクション運用を行うイベントチャネル“mychannel”を生成する場合

[Publish/Subscribeメッセージングモデル]

```
esmkchnl -g mygroup -c mychannel -notify -persist all -tran
```

[Point-To-Pointメッセージングモデル]

```
esmkchnl -g mygroup -c mychannel -notify -ptp -persist all -tran
```

esmkchnlコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“イベントサービス運用コマンド”－“esmkchnl”を参照してください。

esmkchnlのオプションの概要を以下に示します。

オプション	設定内容
-g group	グループ名を指定します。
-c channel	グループに含まれるイベントチャネル名を、64バイト以内の文字列で指定します。
-m number	グループに含まれるイベントチャネルに接続するプロデューサ・コンシューマの合計値(最大接続数)を指定します。Message-driven Beanを運用する場合、以下の計算式を使用して最大接続数を見積ってください。 最大接続数 ＝（初期起動インスタンス数 × 2）× 1.2 ＋ クライアントアプリケーションの多重度
-notify	JMSのイベントチャネルとして生成する場合に指定します。 注）必ず指定してください。
-ptp	Point-To-Pointメッセージングモデルの場合に指定します。本オプションを省略した場合は、Publish/Subscribeメッセージングモデルとなります。
-persist all	以下の機能を使用する場合に指定します。イベントデータ(メッセージ)、接続情報が不揮発化の対象となります。 ・ Durable Subscription機能 ・ イベントチャネルの不揮発化機能 ・ ローカルトランザクション機能 ・ グローバルトランザクション機能
-tran	ローカルトランザクション運用を行う場合に指定します。
-ots	グローバルトランザクション運用を行う場合に指定します。
-l locale	日本語メッセージを送受信する場合に、イベントチャネルが動作するマシンのコード系を指定します。 ・ SJIS(コード系:ShiftJIS) ・ EUC(コード系:EUC)

オプション	設定内容
	・ UNICODE(コード系:UNICODE) Solaris64 Linux32/64 ・ UTF8(コード系:UTF8) Solaris64 Linux32/64
-autodiscon	アプリケーションの異常終了やIJServerの強制停止時なにより、プロデューサ・コンシューマが静的生成イベントチャネルに対してConnection／Sessionのcloseメソッドを発行しないで終了した際に、イベントチャネルに残ったコネクション情報を自動回収する場合に指定します。 また、-tranオプションと同時に本オプションを指定すると、コネクション情報の自動回収時にトランザクション中のローカルトランザクション(未完了のローカルトランザクション)が存在した場合、ローカルトランザクションをロールバックします。 注)Durable Subscription機能を使用する場合は、本オプションを指定しないでください。
-ssl	SSL通信を行う場合に指定します。

23.1.4 イベントチャネル動作環境の変更

Interstage管理コンソールまたはessetcnfコマンド／essetcnfchnlコマンドで、イベントチャネルの動作環境を変更します。

- 構成情報の設定
- イベントチャネルの環境設定

注意

不揮発チャネル運用中のイベントチャネルの動作環境を変更する場合は、以下の構成情報／環境情報だけ変更できます。

- イベントデータ待ち合せ時間
- エラーログファイルサイズ
- ローカルトランザクションのタイムアウト時間
- 2フェーズコミットタイムアウト監視時間
- コンシューマ未接続時のエラー復帰モード
- イベントチャネル自動起動

上記以外の構成情報／環境情報を変更した場合は、不揮発チャネル運用中のイベントチャネルの動作環境が変更されることがあります。この場合、不揮発化情報の整合性が取れなくなる可能性があるため、不揮発チャネル運用中のイベントチャネルを再作成する必要があります。

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

■essetcnfコマンド／essetcnfchnlコマンドを使用する場合

essetcnfコマンドでイベントサービスの構成情報を設定／変更します。また、essetcnfchnlコマンドでイベントチャネルの環境情報を設定／変更します。

essetcnfコマンドおよびessetcnfchnlコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“イベントサービス運用コマンド”－“essetcnf”または“essetcnfchnl”を参照してください。

essetcnfコマンド／essetcnfchnlコマンドの-sオプションに指定する設定項目の概要を以下に示します。

オプション	構成情報／動作環境項目
-schmax	トピックまたはキューが使用する静的イベントチャネルの最大起動数
-dchmax	テンポラリトピックまたはテンポラリキューが使用する動的イベントチャネルの最大起動数
-edmax	イベントチャネルに蓄積できるメッセージ数の最大値
-wtime	メッセージ(イベントデータ)の待ち合わせ時間(秒)(注)
-logsize	イベントサービスで発生したエラー情報を出力するログファイルのサイズ(Kバイト単位)
-gtrnmax	同時実行可能なグローバルトランザクション数
-ltrntime	ローカルトランザクションのタイムアウト時間(秒)
-2pctime	2フェーズコミットタイムアウト監視時間(秒)
-retrytime	リカバリ時のリトライ間隔(秒)
-retrymax	リカバリ時のリトライ回数
-chkcon	サブスクライバ未接続時のエラー復帰モード
-autostart	イベントサービス起動時にイベントチャネルを自動起動する

注)メッセージ(イベントデータ)の待ち合わせ時間には、10秒以上の値を設定することを推奨します。10秒より小さい値を設定する場合は、receiveNoWait()メソッドを使用してメッセージの待ち合わせを行わない運用を検討してください。
また、CORBAサービスの動作環境ファイル(config)のパラメタ“period_receive_timeout”(クライアントのタイムアウト時間)より小さい値を指定してください。値の差が20秒以上になるように指定することを推奨します。

23.2 イベントチャネル運用マシンの運用後の環境削除

ここでは、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルの運用を行うマシンの環境削除について説明します。

運用後の環境削除の手順を以下に示します。

23.2.1 静的イベントチャネルの削除

Interstage管理コンソールまたはesrmchnlコマンドで、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルを削除します。

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

■esrmchnlコマンドを使用する場合

esrmchnlコマンドで、JMSアプリケーションがメッセージの送受信に使用するイベントチャネルを削除します。

例

グループ“mygroup”に属するすべてのイベントチャネルを削除する場合

```
esrmchnl -g mygroup
```

esrmchnlコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“イベントサービス運用コマンド”－“esrmchnl”を参照してください。

23.2.2 保存先の削除

Interstage管理コンソールまたはesrmunitコマンドで、メッセージの保存先（ユニット）を削除します。

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド（基本編）”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

■esrmunitコマンドを使用する場合

esrmunitコマンドで、メッセージの保存先（ユニット）を削除します。なお、ユニットを削除する前に、esstopunitコマンドでユニットを強制停止する必要があります。

例

ユニットID“unit1”を削除する場合

1. esstopunitコマンドで、ユニットID“unit1”を強制停止します。

   ```
   esstopunit -unit unit1 -o off
   ```

2. esrmunitコマンドで、ユニットID“unit1”を削除します。

   ```
   esrmunit -unit unit1
   ```

esrmunitコマンドおよびesstopunitコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“イベントサービス運用コマンド”－“esrmunit”または“esstopunit”を参照してください。

23.2.3 Interstageの停止

Interstage管理コンソールまたはisstopコマンドで、Interstageを停止してイベントサービスを強制停止します。

■Interstage管理コンソールを使用する場合

Interstage管理コンソールの使用方法については、“運用ガイド（基本編）”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

■isstopコマンドを使用する場合

isstopコマンドで、Interstageを停止してイベントサービスを停止します。

```
isstop -f
```

isstopコマンドの詳細については、“リファレンスマニュアル(コマンド編)”の“Interstage統合コマンド”－“isstop”を参照してください。

23.3 JMSアプリケーション運用マシンの運用前の環境設定

ここでは、JMSアプリケーションの運用を行うマシンの環境設定について説明します。

ユーザがJMSアプリケーションを運用するためには、以下の定義が必要です。

- JNDI環境定義
  JMSアプリケーションがJNDIのネーミングサービスにアクセスするために必要な定義です。
- ConnectionFactory定義
  Interstage JMSプロバイダと接続するために必要な定義です。
- Destination定義
  JMSアプリケーションがメッセージを送受信する宛先の定義情報です。

注意

パス／クラスファイルの設定について

JMSアプリケーションの運用に必要なパス／クラスファイルが以下の環境変数に設定されていることを確認してください。

Windows32/64 (インストールパスはデフォルト)

環境変数名	パス／クラスファイル
PATH	JDKのパス(注1) C:¥Interstage¥bin
CLASSPATH	C:¥Interstage¥ODWIN¥etc¥Class¥ODjava4.jar C:¥Interstage¥eswin¥lib¥esnotifyjava4.jar(注2) C:¥Interstage¥J2EE¥lib¥isj2ee.jar C:¥Interstage¥jms¥lib¥fjmsprovider.jar C:¥Interstage¥ots¥lib¥fjtsclient.jar(注3)(注4)

注1)
JDKが複数インストールされている場合は、使用するJDKのパスが有効になるよう設定してください。

注2)
Interstageのクライアント機能がインストールされている場合は、以下のクラスファイルを設定します。

```
C:¥Interstage¥ODWIN¥etc¥Class¥esnotifyjava4.jar
```

注3)
グローバルトランザクション機能を使用する時に必要となります。

注4)
Interstageのクライアント機能がインストールされている場合は、データベース連携サービスがインストールされたホストから取り出す必要があります。

Solaris64 (インストールパスはデフォルト)

環境変数名	パス／クラスファイル
PATH	JDKのパス(注1) /opt/FJSVj2ee/bin /opt/FJSVjms/bin
CLASSPATH	/opt/FSUNod/etc/class/ODjava4.jar /opt/FJSVes/lib/esnotifyjava4.jar /opt/FJSVj2ee/lib/isj2ee.jar /opt/FJSVjms/lib/fjmsprovider.jar
LD_LIBRARY_PATH	/opt/FSUNod/lib(注2) /opt/FJSVjms/lib

注1)
JDKが複数インストールされている場合は、使用するJDKのパスが有効になるよう設定してください。

注2)
環境変数LD_LIBRARY_PATHに/opt/FSUNod/lib/ntを指定しないでください。指定した場合、JMSは動作しません。

Linux32/64

環境変数名	パス／クラスファイル
PATH	JDKのパス(注1) /opt/FJSVj2ee/bin /opt/FJSVjms/bin
CLASSPATH	/opt/FJSVod/etc/class/ODjava4.jar /opt/FJSVes/lib/esnotifyjava4.jar /opt/FJSVj2ee/lib/isj2ee.jar /opt/FJSVjms/lib/fjmsprovider.jar /opt/FJSVots/lib/fjtsclient.jar(注2)
LD_LIBRARY_PATH	/opt/FJSVod/lib(注3) /opt/FJSVjms/lib

注1)
JDKが複数インストールされている場合は、使用するJDKのパスが有効になるよう設定してください。

注2)
グローバルトランザクション機能を使用する時に必要となります。

注3)
環境変数LD_LIBRARY_PATHに/opt/FJSVod/lib/ntを指定しないでください。指定した場合、JMSは動作しません。

注意

ORBの指定について

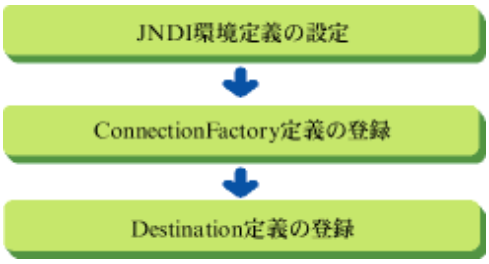
使用するORBとしてCORBAサービス(ObjectDirector)が環境設定ファイルで指定されていることを確認してください。
使用するORBを記述したテキストファイル(ファイル名:orb.properties)を作成し、Javaのシステムプロパティ“java.home”に設定されているディレクトリ配下のlibに格納してください。
orb.propertiesファイルの設定例を以下に示します。

```
org.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
org.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
```

ConnectionFactory定義について

ConnectionFactory定義を大量に登録した場合、Interstage管理コンソールのログイン処理に時間を要することがあります。jmsinfofactコマンドを使用して、不要なConnectionFactory定義が登録されていないかを確認してください。不要なConnectionFactory定義が登録されている場合は、必要に応じて、jmsrmfactコマンドを使用して、不要なConnectionFactory定義を削除してください。ConnectionFactory定義の削除方法については、“23.4.1 ConnectionFactory定義の削除”を参照してください。

運用前の環境設定の手順を以下に示します。



23.3.1 JNDI環境定義の設定

JMSアプリケーションからJNDIのネーミングサービスにアクセスするため、環境プロパティの設定やdeployment descriptorの参照リソース情報にリソースマネージャ名を設定する必要があります。
環境プロパティの設定については、“4.1 JNDIサービスプロバイダの環境設定”を参照してください。deployment descriptorの設定については、“4.9 deployment descriptorファイルへの記述”を参照してください。

23.3.2 ConnectionFactory定義の登録

Interstage管理コンソールでJMSアプリケーションが参照するConnectionFactory定義を登録します。
ConnectionFactory定義は、Interstage JMSプロバイダと接続するために必要な定義です。クライアント識別子、トランザクション種別など接続設定パラメタの集合を持ちます。

Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

Interstageのクライアント機能がインストールされている場合は、jmsmkfactコマンドでConnectionFactory定義を登録します。

例

[Publish/Subscribeメッセージングモデルの場合]
クライアント識別子“client”、JNDI“java:comp/env/jms/TestTopicConnectionFactory”のTopicConnectionFactoryタイプのConnectionFactory定義を登録する場合

```
jmsmkfact -t -i client TestTopicConnectionFactory
```

[Point-To-Pointメッセージングモデルの場合]
クライアント識別子“client”、JNDI“java:comp/env/jms/TestQueueConnectionFactory”のQueueConnectionFactoryタイプのConnectionFactory定義を登録する場合

```
jmsmkfact -q -i client TestQueueConnectionFactory
```

jmsmkfactコマンド実行時は、以下のオプションを任意に指定します。

-t

ConnectionFactoryのタイプが“TopicConnectionFactory”の場合に指定します。

-q

ConnectionFactoryのタイプが“QueueConnectionFactory”の場合に指定します。

-x

グローバルトランザクション機能を使用する場合に指定します。

注意

JNDI“TopicCF001”のConnectionFactory定義は、TopicConnectionFactoryタイプのデフォルト定義として登録されています。
また、JNDI“QueueCF001”のConnectionFactory定義は、QueueConnectionFactoryタイプのデフォルト定義として登録されています。
デフォルト定義は、ConnectionFactoryの定義情報画面、isj2eeadminコマンド、またはjmsmkfactコマンドで、以下の設定内容を変更できます。

- クライアント識別子(クライアントID)
- グローバルトランザクション機能の使用の有無

23.3.3 Destination定義の登録

Interstage管理コンソールでJMSアプリケーションが参照するDestination定義を登録します。
Destination定義は、JMSアプリケーションがメッセージを送受信する宛先の定義情報です。イベントサービス(ノーティフィケーションサービス)のイベントチャネル名、グループ名を持ちます。
Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

Interstageのクライアント機能がインストールされている場合は、jmsmkdstコマンドでDestination定義を登録します。

例

[Publish/Subscribeメッセージングモデルの場合]
グループ“mygroup”のイベントチャネル“mychannel”を、JNDI“java:comp/env/jms/TestTopic”のTopicタイプのDestination定義に関連付ける場合

```
jmsmkdst -t -g mygroup -c mychannel TestTopic
```

[Point-To-Pointメッセージングモデルの場合]
グループ“mygroup”のイベントチャネル“mychannel”を、JNDI“java:comp/env/jms/TestQueue”のQueueタイプのDestination定義に関連付ける場合

```
jmsmkdst -q -g mygroup -c mychannel TestQueue
```

jmsmkdstコマンド実行時は、以下のオプションを任意に指定します。

-t

Destinationのタイプが“Topic”の場合に指定します。

-q

Destinationのタイプが“Queue”の場合に指定します。

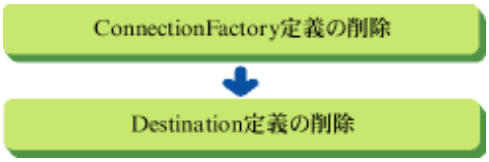
注意

- 他のサーバのイベントチャネルと関連付けるDestinationを定義する場合は、以下の方法でJMSアプリケーションを動作させるサーバ上に、イベントチャネルが登録されている“ホスト名またはIPアドレス”および“ポート番号”を指定し、Destinationを作成してください。
  なお、JMSアプリケーションを動作させるサーバのネーミングサービスの“ホスト定義”および“ポート番号”に設定されている設定情報が、イベントチャネルが登録されているサーバと同一である場合は、Destination作成時に“ホスト名またはIPアドレス”および“ポート番号”を指定する必要はありません。
  - Interstage管理コンソール
    - [システム] > [リソース] > [JMS] > [Destination] > [新規作成]
  - isj2eeadminコマンド
  - jmsmkdstコマンド
- “ホスト名またはIPアドレス”を指定する場合は、hostsファイルの内容またはDNSの設定により名前解決（IPアドレス解決）が可能であるかを確認してください。

23.4 JMSアプリケーション運用マシンの運用後の環境削除

ここでは、JMSアプリケーションの運用を行うマシンの環境削除について説明します。

運用後の環境削除の手順を以下に示します。



23.4.1 ConnectionFactory定義の削除

Interstage管理コンソールでConnectionFactory定義を削除します。
Interstage管理コンソールの使用方法については、“運用ガイド（基本編）”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

Interstageのクライアント機能がインストールされている場合は、jmsrmfactコマンドでConnectionFactory定義を削除します。

例

JNDI“java:comp/env/jms/TestTopicConnectionFactory”のConnectionFactory定義を削除する場合

```
jmsrmfact TestTopicConnectionFactory
```

注意

JNDI“TopicCF001”のConnectionFactory定義は、TopicConnectionFactoryタイプのデフォルト定義として登録されています。また、JNDI“QueueCF001”のConnectionFactory定義は、QueueConnectionFactoryタイプのデフォルト定義として登録されています。なお、デフォルト定義は、削除できません。

23.4.2 Destination定義の削除

Interstage管理コンソールでDestination定義を削除します。
Interstage管理コンソールの使用方法については、“運用ガイド(基本編)”の“Interstage管理コンソールによるInterstage運用”およびInterstage管理コンソールのヘルプを参照してください。

Interstageのクライアント機能がインストールされている場合は、jmsrmdstコマンドでDestination定義を削除します。

例

JNDI“java:comp/env/jms/TestTopic”のDestination定義を削除する場合

```
jmsrmdst TestTopic
```

23.4.3 durable Subscriberの削除

Durable Subscription機能を使用していた場合、アプリケーションでunsubscribeメソッドを使用して、durable Subscriberを削除していないときは、jmsrmdsコマンドでdurable Subscriberを削除します。

例

Durable Subscription“dsub”、クライアント識別子“client”のdurable Subscriberを削除する場合

```
jmsrmds -n dsub -i client
```

注意

durable Subscriberを削除する場合、イベントチャネルを起動しておく必要があります。

ポイント

jmsinfodsコマンドで、Durable Subscription名およびクライアント識別子を確認できます。

第24章　JMSアプリケーションの開発

本章では、JMSを使用したアプリケーションの開発について説明します。

24.1 設計方法

JMSの利用目的に応じて、以下のように設計します。

利用目的	設計方法
メッセージが宛先に到達すると、自動的に処理する	Message Listenerを使用します。
Publish/Subscribeメッセージングモデルにおいて、受信するJMSアプリケーションが停止中に配信されたメッセージを受信する	Durable Subscription機能を使用します。
メッセージの欠落を防止する	メッセージの不揮発化機能、およびローカルトランザクション機能を使用します。
メッセージおよびデータベースの処理を一貫して保証する	グローバルトランザクション機能を使用します。
受信アプリケーションが興味のある情報だけを取得する	メッセージセレクタ機能を使用します。
資源削減のため、送信拠点/受信拠点ごとにイベントチャネルを作成することなく、1つのイベントチャネルでメッセージを送受信する	メッセージセレクタ機能を使用します。
送信拠点/受信拠点が頻繁に変更されるシステム形態において、変更されるたびにイベントサービスの環境を変更することなく、メッセージを送受信する	メッセージセレクタ機能を使用します。
Point-To-Pointメッセージングモデルにおいて、キューに蓄積されているメッセージを参照する	キューブラウザ機能を使用します。
キューにメッセージを蓄積する処理のスループットを向上させる	送信アプリケーション（Sender）の多重度を上げます。

ポイント

オラクル社より公開されている、Java Message Service 1.1規約に従ったアプリケーションを作成してください。
Java Message Service 1.1規約では、Java Message Service 1.0.2規約に従ったアプリケーションも正常に動作することを保証しています。

24.2 Publish/SubscribeメッセージングモデルとPoint-To-Pointメッセージングモデルの作成方法

Publish/SubscribeメッセージングモデルとPoint-to-Pointメッセージングモデルのアプリケーションの作成に、同一のAPIを使用できます。

MessageProducer（送信アプリケーション）とMessageConsumer（受信アプリケーション）を使用します。
MessageProducerはイベントチャネルにメッセージを送信し、MessageConsumerはイベントチャネルにメッセージを要求します。

24.2.1 MessageProducerの作成

MessageProducerは、イベントチャネルにメッセージを送信します。
MessageProducerがイベントチャネルにメッセージを送信する手続き例と処理の流れを以下に説明します。

[MessageProducer]

```
import javax.jms.*;
import javax.naming.*;

public class Producer {
  public static void main( String[] args) {
    Connection connection = null;
    ...
    try {
      java.util.Hashtable env = new java.util.Hashtable( );
      env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
             "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
      InitialContext initialContext = new InitialContext(env);                    /* 1 */
      ConnectionFactory connectionFactory = (ConnectionFactory)
          initialContext.lookup("java:comp/env/jms/ConnectionFactory");           /* 2 */
      Destination destination = (Destination)
          initialContext.lookup("java:comp/env/jms/Destinaiton");                 /* 3 */
      connection = connectionFactory.createConnection();                          /* 4 */
      Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); /* 5 */
      MessageProducer messageProducer = session.createProducer(destination);     /* 6 */
      TextMessage Message = session.createTextMessage( );
      Message.setText( "Message" );
      messageProducer.send(Message);                                              /* 7 */
    } catch( Exception e ) {
        ...
    }
    finally {
      if( null != connection ) {
        try {
          connection.close();                                                     /* 8 */
        }
        catch( Exception e ) {
          ...
        }
      }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
4. Connectionを作成します。
5. Sessionを作成します。
6. MessageProducerを作成します。
7. メッセージを送信します(MessageタイプがTextMessageで送信メッセージが“Message”の場合)。
8. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.2.2 MessageConsumerの作成

MessageConsumerは、イベントチャネルにメッセージを要求します。
MessageConsumerがイベントチャネルにメッセージを要求する手続き例と処理の流れを以下に説明します。

[MessageConsumer]

```
import javax.jms.*;
import javax.naming.*;

public class Consumer {
  public static void main( String[] args) {
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
            "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                    /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");           /* 2 */
        Destination destination = (Destination)
            initialContext.lookup("java:comp/env/jms/Destination");                 /* 3 */
        connection = connectionFactory.createConnection();                          /* 4 */
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); /* 5 */
        MessageConsumer messageConsumer = session.createConsumer(destination);      /* 6 */
        connection.start();                                                         /* 7 */
        Message message = messageConsumer.receive();                                /* 8 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                                 /* 9 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
4. Connectionを作成します。
5. Sessionを作成します。
6. MessageConsumerを作成します。
7. 接続によるメッセージの配信を開始します。
8. メッセージを受信します。
9. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.3 Message Listenerの作成方法

Message Listenerを使用した受信アプリケーションの開発について説明します。
メッセージが受信者に到達した際、自動的に処理を行う機能を使用します。

24.3.1 Message Listenerを使用したMessageConsumerの作成

受信したメッセージを処理するために、MessageListenerオブジェクトを登録します。メッセージが到達すると、Message Listenerが呼出されます。
Message Listenerを使用してメッセージを受信する手続き例と処理の流れを以下に説明します。

[Message Listenerを使用したMessageConsumer]

```
import javax.jms.*;
import javax.naming.*;

public class ConsumerA implements MessageListener,ExceptionListener {                /* 1 */
  public static void main( String[] args ) {
    ...
    Connection connection = null;
    ConsumerA consumerA = new ConsumerA ();                                           /* 2 */
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
             "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                      /* 3 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");             /* 4 */
        Destination destination = (Destination)
            initialContext.lookup("java:comp/env/jms/Destination");                   /* 5 */
        connection = connectionFactory.createConnection();                            /* 6 */
        connection.setExceptionListener( consumerA ) ;                                /* 7 */
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); /* 8 */
        MessageConsumer consumer = session.createConsumer(destination);              /* 9 */
        consumer.setMessageListener(consumerA);                                       /* 10 */
        connection.start();                                                           /* 11 */
        /* 待ち合わせ処理 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                                   /* 12 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }

  public void onMessage(Message message) {
```

```
    ...
    try {
    } catch( JMSException e ) {
        ...
    }
    ...
  }

  public void onException( JMSException jmsEx ) {
    ...
  }
}
```

1. MessageListenerインタフェースおよびExceptionListenerインタフェースを実装します。
2. クラスのインスタンスを生成します。
3. JNDIの開始コンテキストを構築します。
4. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
5. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
6. Connectionを作成します。
7. ExceptionListenerオブジェクトを登録します。
8. Sessionを作成します。
9. MessageConsumerを作成します。
10. MessageListenerオブジェクトを登録します。
11. 接続によるメッセージの配信を開始します。
12. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.4 Durable Subscription機能の作成方法

Durable Subscription機能を使用した受信アプリケーションの開発について説明します。
アプリケーションがアクティブではない間に送信されたメッセージを、アプリケーションがアクティブになった後に受信できる機能を使用します。

24.4.1 Durable Subscription機能を使用したMessageConsumerの作成

durable Subscriberを作成します。durable Subscriberは、イベントチャネルにメッセージを要求します。
Durable Subscription機能を使用してメッセージを受信する手続き例と処理の流れを以下に説明します。

[Durable Subscription機能を使用したMessageConsumer]

```
import javax.jms.*;
import javax.naming.*;

public class SubscriberD {
  public static void main( String[] args ) {
```

```
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
                  "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                          /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");                 /* 2 */
        Topic topic = (Topic)initialContext.lookup("java:comp/env/jms/Topic");            /* 3 */
        connection = connectionFactory.createConnection();                                /* 4 */
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);      /* 5 */
        TopicSubscriber topicSubscriber = session.createDurableSubscriber(topic, "dsub"); /* 6 */
        connection.start();                                                               /* 7 */
        Message message = topicSubscriber.receive();                                      /* 8 */
        topicSubscriber.close();                                                          /* 9 */
        session.unsubscribe("dsub");                                                      /* 10 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                                       /* 11 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Topic オブジェクトを取得します(JNDI名が“Topic”の場合)。
4. Connectionを作成します。
5. Sessionを作成します。
6. Durable TopicSubscriberを作成します(Durable Subscription名が“dsub”の場合)。
7. 接続によるメッセージの配信を開始します。
8. メッセージを受信します。
9. Durable TopicSubscriberをクローズします。
10. Durable TopicSubscriberを解除します(アプリケーション中断中に送信されたメッセージを、アプリケーション再起動後に受信しない場合)。
11. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。
Durable TopicSubscriberを解除すると、JMSプロバイダがアプリケーションに代わって持続している接続状態が削除される

ため、アプリケーション中断中に送信されたメッセージをアプリケーション再起動後に受信することはできません。Durable Subscription機能が必要なくなった場合にだけDurable TopicSubscriberを解除してください。

24.4.2 Durable Subscription機能使用時の注意事項

durable Subscriberは、Durable Subscription名とクライアント識別子の2つの情報で同定されます。
同一マシン上で複数のアプリケーションを起動する場合は、それぞれ異なるDurable Subscription名を使用するか、または異なるクライアント識別子を指定したConnectionFactory定義を使用する必要があります。

24.5 メッセージの優先度と生存時間の作成方法

メッセージの優先度や生存時間は、メッセージの送信側で設定します。
MessageProducerインタフェースのsend(Message message, int deliveryMode, int priority, long timeToLive)メソッドによって、設定できます。
本メソッドを使用しない場合は、デフォルトの優先度、生存時間が設定されます。
デフォルトの優先度は4、生存時間は無限ですが、MessageProducerインタフェースのsetPriority(int defaultPriority)、setTimeToLive(long timeToLive) を使用して変更できます。

- 優先度は、priorityに(低)0～9(高)の値を設定します。
- 生存時間は、timeToLiveに時間(ミリ秒単位)を設定します。

注意

timeToLiveはミリ秒単位までサポートしますが、イベントチャネルのメッセージタイムアウト時間の精度は秒単位であるため、timeToLive値は最も近い値に丸められます。

24.6 メッセージの不揮発化機能の作成方法

メッセージの不揮発化機能を使用するには、以下の手順が必要になります。

1. esmkchnlコマンドで不揮発チャネル作成します。
2. MessageProducerインタフェースのsend()メソッドの引数deliveryModeに、javax.jms.DeliveryMode.PERSISTENTを指定します。

24.7 ローカルトランザクションの作成方法

24.7.1 ローカルトランザクションを使用したMessageProducerの作成

[ローカルトランザクションのMessageProducer]

```
import javax.jms.*;
import javax.naming.*;

public class Producer {
  public static void main( String[] args ) {
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
                "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                    /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");           /* 2 */
```

```
        Destination destination = (Destination)
            initialContext.lookup("java:comp/env/jms/Destination");          /* 3 */
        connection = connectionFactory.createConnection();                   /* 4 */
        Session session = connection.createSession(true, 0);                 /* 5 */
        MessageProducer producer = session.createProducer(destination);     /* 6 */
        TextMessage Message = session.createTextMessage( );
        Message.setText( "Message" );
        producer.send(Message);                                              /* 7 */
        session.commit();                                                    /* 8 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                          /* 9 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
4. Connectionを作成します。
5. Sessionを作成します。
6. MessageProducerを作成します。
7. メッセージを送信します(MessageタイプがTextMessageで、送信メッセージが“Message”の場合)。
8. コミットします。
9. Connectionをクローズします。

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.7.2 ローカルトランザクションを使用したMessageConsumerの作成

[ローカルトランザクションのMessageConsumer]

```
import javax.jms.*;
import javax.naming.*;

public class Consumer {
  public static void main( String[] args ) {
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
```

```
                "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                    /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");           /* 2 */
        Destination destination = (Destination)
            initialContext.lookup("java:comp/env/jms/Destination");                 /* 3 */
        connection = connectionFactory.createConnection();                          /* 4 */
        Session session = connection.createSession(true, 0);                        /* 5 */
        MessageConsumer consumer = session.createConsumer(destination);            /* 6 */
        connection.start();                                                         /* 7 */
        Message message = consumer.receive();                                       /* 8 */
        session.commit();                                                           /* 9 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                                 /* 10 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
4. Connectionを作成します。
5. Sessionを作成します。
6. MessageConsumerを作成します。
7. 接続によるメッセージの配信を開始します。
8. メッセージを受信します。
9. コミットします。
10. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.8 グローバルトランザクションの作成方法

24.8.1 グローバルトランザクションを使用したMessageProducerの作成

[グローバルトランザクションのMessageProducer]

```
import javax.jms.*;
import javax.naming.*;
```

```
public class Producer {
  public static void main( String[] args ) {
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
                "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");       /* 2 */
        Destination destination = (Destination)
            initialContext.lookup("java:comp/env/jms/Destination");             /* 3 */
        javax.transaction.UserTransaction ut =(javax.transaction.UserTransaction)
            initialContext.lookup("java:comp/UserTransaction");                 /* 4 */
        connection = connectionFactory.createConnection();                      /* 5 */
        Session session = connection.createSession(true, 0);                    /* 6 */
        MessageProducer producer = session.createProducer(destination);        /* 7 */
        TextMessage Message = session.createTextMessage( );
        Message.setText( "Message" );
        ut.begin();                                                             /* 8 */
        producer.send(Message);                                                 /* 9 */
        ut.commit();                                                            /* 10 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                             /* 11 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
4. UserTransactionオブジェクトを取得します。
5. Connectionを作成します。
6. Sessionを作成します。
7. MessageProducerを作成します。
8. グローバルトランザクションを開始します。
9. メッセージを送信します(Messageタイプが TextMessageで送信メッセージが“Message”の場合)。
10. グローバルトランザクションを完了します。
11. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.8.2 グローバルトランザクションを使用したMessageConsumerの作成

[グローバルトランザクションのMessageConsumer]

```
import javax.jms.*;
import javax.naming.*;

public class Consumer {
  public static void main( String[] args ) {
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
                  "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                    /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");           /* 2 */
        Destination destination = (Destination)
            initialContext.lookup("java:comp/env/jms/Destination");                 /* 3 */
        javax.transaction.UserTransaction ut =(javax.transaction.UserTransaction)
            initialContext.lookup("java:comp/UserTransaction");                     /* 4 */
        connection = connectionFactory.createConnection();                          /* 5 */
        Session session = connection.createSession(true, 0);                        /* 6 */
        MessageConsumer consumer = session.createConsumer(destination);            /* 7 */
        connection.start();                                                         /* 8 */
        ut.begin();                                                                 /* 9 */
        Message message = consumer.receive();                                       /* 10 */
        ut.commit();                                                                /* 11 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                                 /* 12 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Destinationオブジェクトを取得します(JNDI名が“Destination”の場合)。
4. UserTransactionオブジェクトを取得します。
5. Connectionを作成します。
6. Sessionを作成します。
7. MessageConsumerを作成します。

8. 接続によるメッセージの配信を開始します。
9. グローバルトランザクションを開始します。
10. メッセージを受信します。
11. グローバルトランザクションを完了します。
12. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

24.8.3 アプリケーション起動時の注意事項

アプリケーションを起動する場合、以下の環境プロパティに“True”を指定する必要があります。

- com.fujitsu.ObjectDirector.CORBA.GlobalTransactionMode

環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。

例

コマンドラインで指定する場合

```
java -Dcom.fujitsu.ObjectDirector.CORBA.GlobalTransactionMode=True Publisher
```

24.9 CORBAアプリケーションとの連携機能の作成方法

24.9.1 JMSアプリケーションからCORBAアプリケーションへの通信

JMSアプリケーションからノーティフィケーションサービスを介したCORBAアプリケーションへの通信時には、JMS MessageからStructuredEventへ変換します。
Interstage JMSがサポートするCORBAアプリケーションと連携可能なJMSのメッセージ型は、BytesMessageとTextMessageです。

JMS MessageのStructuredEventへの変換について、以下に示します。

	JMS Message	StructuredEvent
ヘッダ	JMSPriority JMSExpiration	QoSプロパティ ・ Priority ・ Timeout FixedHeader：JMS-Providerで自動的に追加 ・ domain_name="JMS" ・ type_name="v6.2" ・ event_name=""
プロパティ	変換を行いません	
ボディ	BytesMessage	octet型のsequence型として取り出してください

	JMS Message	StructuredEvent
	TextMessage	wstring型として取り出してください

JMSPriorityとQoSプロパティのPriorityの対応表

JMSPriority	0	1	2	3	4	5	6	7	8	9
Priority(QoS)	-3	-3	-2	-1	0	1	2	2	3	3

24.9.2 CORBAアプリケーションからJMSアプリケーションへの通信

CORBAアプリケーションからノーティフィケーションサービスを介したJMSアプリケーションへの通信時には、StructuredEventのremainder_of_bodyからJMS Messageのボディへの変換だけが可能です。
Interstage JMSでサポートするCORBAアプリケーションと連携可能なStructuredEventのデータ型は、以下のようになっています。

StructuredEvent	JMS Message
string型	TextMessage
wstring型	
octet型の配列	BytesMessage
octet型のsequence型	

24.9.3 注意事項

日本語メッセージを送受信する場合、CORBAアプリケーション側で環境変数OD_CODE_SETにコード系を設定してください。

例

Windows32/64
コード系がShiftJISの場合

```
set OD_CODE_SET=SJIS
```

Solaris64 Linux32/64
コード系がEUCの場合

```
setenv OD_CODE_SET EUC
```

ポイント

CORBAアプリケーションがJavaの場合、自動的にコード系をUNICODEと判断します。そのため、環境変数OD_CODE_SETを設定する必要はありません。

24.10 メッセージセレクタ機能の作成方法

メッセージセレクタ機能を使用するには、アプリケーションを以下のように作成する必要があります。

■送信アプリケーション

送信アプリケーションは、メッセージの作成時にメッセージセレクタの対象となるプロパティ値を指定します。

例

メッセージにプロパティ名“NAME”で、プロパティ値“FUJITSU”を設定する場合

```
String name = "FUJITSU";
message.setStringProperty("NAME", name)
```

■受信アプリケーション

受信アプリケーションは、Consumerの作成時、Durable Subscriberの作成時、またはBrowserの作成時にパラメタとしてメッセージセレクタ文を指定します。
メッセージセレクタ文は、SQL文のWHERE句で使用されるクエリ文字列です。

例

Consumerの作成時に、プロパティ名“NAME”のプロパティ値が“FUJITSU”のメッセージを受信するためのメッセージセレクタを設定する場合

```
String selector = "NAME = 'FUJITSU'";
session.createConsumer(destination, selector)
```

24.10.1 メッセージセレクタ条件式

条件式は、SQL文のWHERE句で使用されるSQL-92の条件式に準拠しています。

例

```
color = 'blue'
```

上記の条件式の左辺のcolorは、識別子です。フィルタリング時にJMS Messagesのプロパティ名と比較されます。
右辺の'blue'は、リテラルです。フィルタリング時にJMS Messagesのプロパティ値と比較されます。
リテラルには、文字列型リテラル、完全数値リテラル、および近似数値リテラルがあります。

文字列型リテラル

シングルクォートで囲む必要があります。

完全数値リテラル

小数点なしの数値（例：57、-957、+62）です。

近似数値リテラル

指数表記された数値（例：7E3、-57.9E2）、または小数点を含む数値（例：7.、-95.7、+6.2）です。

なお、条件式の文法の詳細については、Javadoc集の“パッケージ javax.jms”の“インタフェース Message”を参照してください。

条件式の例を以下に示します。

・ 比較演算条件式
以下の比較演算子が使用できます。

－ =(等しい)

－ >(大きい)

－ >=(以上)

－ <(小さい)

－ <=(以下)

－ <>(等しくない)

例:プロパティ名NAMEのプロパティ値が“FUJITSU”のJMS Messagesを受信する場合

```
NAME = 'FUJITSU'
```

例:プロパティ名NUMBERのプロパティ値が1000以上のJMS Messagesを受信する場合

```
NUMBER >= 1000
```

例:プロパティ名NUMBERのプロパティ値が230以下のJMS Messagesを受信する場合

```
NUMBER <= 10 * 20 + 30
```

上記のように、算術演算子(+,-,*,/)を使用できます。

・ BETWEEN条件式
BETWEEN条件式により範囲付きの検索ができます。

例:プロパティ名NUMBERのプロパティ値が100以上かつ1000以下のJMS Messagesを受信する場合

```
NUMBER BETWEEN 100 AND 1000
```

例:プロパティ名NUMBERのプロパティ値が100未満または1000より大きいJMS Messagesを受信する場合

```
NUMBER NOT BETWEEN 100 AND 1000
```

・ LIKE条件式
LIKE条件式によりパターン検索を使用した検索ができます。
文字列リテラルに、以下の記号を指定することによりパターン検索を行います。

－ _:任意の1文字

－ %:任意の文字列

オプションであるエスケープ文字は、単独の文字リテラルです。'_'と'%'を単なる文字列として扱う場合に使用します。

例:プロパティ名PROPERTYのプロパティ値が“任意の文字+C”のJMS Messagesを受信する場合

```
PROPERTY LIKE '%C'
```

プロパティ値が“ABC”、“CCC”、または“C”の場合は、受信できます。“AB”の場合は、受信できません。

例:プロパティ名PROPERTYのプロパティ値が“任意の文字+C”以外のJMS Messagesを受信する場合

```
PROPERTY NOT LIKE '%C'
```

プロパティ値が“AB”の場合は、受信できます。“ABC”、“CCC”、または“C”の場合は、受信できません。

例:プロパティ名PROPERTYのプロパティ値が“任意の1文字+C”のJMS Messagesを受信する場合

```
PROPERTY LIKE '_C'
```

プロパティ値が“AC”または“CC”の場合は、受信できます。“ABC”または“C”の場合は、受信できません。

例:プロパティ名PROPERTYのプロパティ値が“任意の1文字+本”のJMS Messagesを受信する場合

```
PROPERTY LIKE '__本'
```

プロパティ値が“日本”の場合は、受信できます、“本”の場合は、受信できません。
なお、日本語は1バイトであるため、1文字に付き'_'が2個必要です。

例:プロパティ名PROPERTYのプロパティ値が“任意の文字+%+C”のJMS Messagesを受信する場合

```
PROPERTY LIKE '%#%C' ESCAPE '#'
```

プロパティ値が“A%C”の場合は、受信できます。“AAC”の場合は、受信できません。

- NULL条件式
  NULL条件式により、プロパティがあるかどうかを検索できます。

  例:プロパティ名PROPERTYのプロパティが存在しないJMS Messagesを受信する場合

```
PROPERTY IS NULL
```

  例:プロパティ名PROPERTYのプロパティが存在するJMS Messagesを受信する場合

```
PROPERTY IS NOT NULL
```

- IN条件式
  IN条件式により、リスト内の項目を検索できます。

  例:プロパティ名PROPERTYのプロパティ値が“AAA”、“BBB”、または“CCC”ならば、JMS Messagesを受信する場合

```
PROPERTY IN ('AAA','BBB','CCC')
```

  例:プロパティ名PROPERTYのプロパティ値が“AAA”、“BBB”、または“CCC”以外の場合にJMS Messagesを受信する場合

```
PROPERTY NOT IN ('AAA','BBB','CCC')
```

- 条件式の混在
  上記の条件式を、NOT、AND、ORにより混在して指定できます。

  例:プロパティ名NUMBERのプロパティ値が100以上かつ1000以下、またはプロパティ名PROPERTYのプロパティ値が“任意の文字+C”の場合にJMS Messagesを受信する場合

```
(NUMBER BETWEEN 100 AND 1000) OR (PROPERTY LIKE '%C')
```

  例:プロパティ名NUMBERのプロパティ値が100以上かつ1000以下、かつプロパティ名PROPERTYのプロパティ値が“任意の文字+C”の場合にJMS Messagesを受信

```
(NUMBER BETWEEN 100 AND 1000) AND (PROPERTY LIKE '%C')
```

  例:プロパティ名NUMBERのプロパティ値が100以上かつ1000以下以外、またはプロパティ名PROPERTYのプロパティ値が“任意の文字+C”以外の場合にJMS Messagesを受信する場合

```
NOT ((NUMBER BETWEEN 100 AND 1000) AND (PROPERTY LIKE '%C'))
```

注意

- 指定可能なメッセージセレクタ文の最大長は、4096バイトです。
- 条件式に指定可能な識別子および文字列型リテラルの最大長は、1024バイトです。
- 条件式内に指定可能な識別子と文字列型リテラルの合計の上限は、512個です。
- IN条件式に指定可能なリスト数の上限は、256個です。
- 識別子には、日本語を指定できません。

・ LIKE条件式のESCAPE文字には、日本語を指定できません。

24.11 キューブラウザ機能の作成方法

アプリケーションでは、キューに蓄積されているメッセージを参照するため、Browserを作成します。その後、順にメッセージを取り出して、キューの内容をブラウジングできます。
キューブラウザの手続き例と処理の流れを以下に示します。

[Browser]

```
import javax.jms.*;
import javax.naming.*;

public class Browser {
  public static void main( String[] args ) {
    Connection connection = null;
    ...
    try {
        java.util.Hashtable env = new java.util.Hashtable( );
        env.put ( javax.naming.Context.INITIAL_CONTEXT_FACTORY ,
                  "com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient" );
        InitialContext initialContext = new InitialContext(env);                    /* 1 */
        ConnectionFactory connectionFactory = (ConnectionFactory)
            initialContext.lookup("java:comp/env/jms/ConnectionFactory");           /* 2 */
        Queue queue = (Queue)initialContext.lookup("java:comp/env/jms/Queue");      /* 3 */
        connection = connectionFactory.createConnection();                          /* 4 */
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE); /* 5 */
        QueueBrowser queueBrowser = session.createBrowser(queue);                   /* 6 */
        connection.start();                                                         /* 7 */
        java.util.Enumeration e = queueBrowser.getEnumeration();                    /* 8 */
        Message message;
        while ( e.hasMoreElements() ) {
            message = (Message)e.nextElement();                                     /* 9 */
            ...
        }
        queueBrowser.close();                                                       /* 10 */
    } catch( Exception e ) {
        ...
    }
    finally {
        if( null != connection ) {
            try {
                connection.close();                                                 /* 11 */
            }
            catch( Exception e ) {
                ...
            }
        }
    }
  }
}
```

1. JNDIの開始コンテキストを構築します。
2. ConnectionFactoryオブジェクトを取得します(JNDI名が“ConnectionFactory”の場合)。
3. Queue オブジェクトを取得します(JNDI名が“Queue”の場合)。
4. Connectionを作成します。

5. Sessionを作成します。
6. QueueBrowserを作成します。
7. 接続によるメッセージの配信を開始します。
8. ブラウジングを開始します。
9. 1メッセージを取得します。
10. QueueBrowserをクローズします。
11. Connectionをクローズします。

注意

JNDIの開始コンテキスト構築時の環境プロパティの指定については、“J2EEアプリケーションクライアント”を参照してください。
1つのイベントチャネルに対し、複数のキューブラウザを使用することはできません。

24.12 アプリケーション実行時の注意事項

■TopicRequestor/QueueRequestor使用時について

TopicRequestorクラス、またはQueueRequestorクラスのrequest()メソッドは、Topic、またはQueueに要求メッセージを送り、その応答を待ちます。受信アプリケーションが接続していない場合、または受信アプリケーションが応答メッセージを返さない場合など、アプリケーションの構成によってrequest()メソッドから復帰しないことがあります。
上記のような無限の待ち合わせを回避する方法として、システムプロパティを使用することにより、一定時間内に応答が返らない場合にrequest()メソッドからの復帰を指示することができます。

[システムプロパティ名]

```
com.fujitsu.interstage.jms.receive_timeout
```

[指定方法]

ミリ秒単位でタイムアウト時間を設定します。

例

```
-Dcom.fujitsu.interstage.jms.receive_timeout=10000
```

ポイント

システムプロパティに、イベントサービス動作環境の「メッセージの待ち合わせ時間(-wtimeの指定値)」より小さい値を設定した場合、-wtimeに指定された時間だけ待ち合わせます。
タイムアウト発生時、request()メソッドは、nullで復帰します。

■メッセージの欠落の防止について

イベントチャネルのメッセージ蓄積可能なデータ数の上限値を超過した場合、メッセージが欠落することがあります。
メッセージの欠落を回避する方法として、システムプロパティを使用することにより、エラー通知を指示することができます。

[システムプロパティ名]

```
com.fujitsu.interstage.jms.queue_max_err
```

[指定方法]

“yes”を設定します。

例

```
-Dcom.fujitsu.interstage.jms.queue_max_err=yes
```

■JMS1.0.2規約のAPIについて

本製品の8.0以前に作成したJava Message Service 1.0.2規約に従ったアプリケーションは、システムプロパティを使用することにより、8.0以前と同じように動作させることができます。

[システムプロパティ名]

```
com.fujitsu.interstage.jms.exception_version_lower
```

[指定方法]

“yes”を設定します。

例

```
-Dcom.fujitsu.interstage.jms.exception_version_lower=yes
```

■メッセージの受信タイムアウトについて

メッセージを受信できない場合に受信タイムアウトが発生するタイミングは、receive()メソッドに指定したタイムアウト時間より長くなることがあります。メッセージの受信タイムアウトの詳細については、“24.13.4 パッケージjavax.jmsのAPI一覧（その4）”のreceive()メソッドの（注1）を参照してください。

24.13 インタフェース

Interstage JMSがサポートするJMSのインタフェースについて説明します。
パッケージjavax.jmsのAPI一覧を以下に示します。

APIの詳細については、Javadoc集の“パッケージ javax.jms”を参照してください。

24.13.1 パッケージjavax.jmsのAPI一覧（その1）

インタフェース名／クラス名	メソッド	サポート
BytesMessage	getBodyLength()	○（注1）
	readBoolean()	○
	readByte()	○
	readBytes(byte[] value)	○

インタフェース名／クラス名	メソッド	サポート
	readBytes(byte[] value, int length)	○
	readChar()	○
	readDouble()	○
	readFloat()	○
	readInt()	○
	readLong()	○
	readShort()	○
	readUnsignedByte()	○
	readUnsignedShort()	○
	readUTF()	○
	reset()	○
	writeBoolean(boolean value)	○
	writeByte(byte value)	○
	writeBytes(byte[] value)	○
	writeBytes(byte[] value, int offset, int length)	○
	writeChar(char value)	○
	writeDouble(double value)	○
	writeFloat(float value)	○
	writeInt(int value)	○
	writeLong(long value)	○
	writeObject(java.lang.Object value)	○
	writeShort(short value)	○
	writeUTF(java.lang.String value)	○
Connection	close()	○
	createConnectionConsumer(Destination destination, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)	○（注1）
	createDurableConnectionConsumer(Topic topic, java.lang.String subscriptionName, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)	○（注1）
	createSession(boolean transacted, int acknowledgeMode)	○（注1）
	getClientID()	○
	getExceptionListener()	○
	getMetaData()	○
	setClientID(java.lang.String clientID)	○
	setExceptionListener(ExceptionListener listener)	○

インタフェース名／クラス名	メソッド	サポート
	start()	○
	stop()	○
ConnectionConsumer	close()	○
	getServerSessionPool()	○
ConnectionFactory	createConnection()	○（注1）
	createConnection(java.lang.String userName, java.lang.String password)	○（注1）

○：サポート
×：未サポート

注1）JMS1.1規約で追加されました。

24.13.2 パッケージjavax.jmsのAPI一覧（その2）

インタフェース名／クラス名	メソッド	サポート
ConnectionMetaData	getJMSMajorVersion()	○
	getJMSMinorVersion()	○
	getJMSProviderName()	○（注1）
	getJMSVersion()	○
	getJMSXPropertyNames()	○
	getProviderMajorVersion()	○
	getProviderMinorVersion()	○
	getProviderVersion()	○
DeliveryMode	メソッドなし	
Destination	メソッドなし	
ExceptionListener	onException(JMSException exception)	○
MapMessage	getBoolean(java.lang.String name)	○
	getByte(java.lang.String name)	○
	getBytes(java.lang.String name)	○
	getChar(java.lang.String name)	○
	getDouble(java.lang.String name)	○
	getFloat(java.lang.String name)	○
	getInt(java.lang.String name)	○
	getLong(java.lang.String name)	○
	getMapNames()	○
	getObject(java.lang.String name)	○
	getShort(java.lang.String name)	○
	getString(java.lang.String name)	○
	itemExists(java.lang.String name)	○
	setBoolean(java.lang.String name, boolean value)	○

インタフェース名／クラス名	メソッド	サポート
	setByte(java.lang.String name, byte value)	○
	setBytes(java.lang.String name, byte[] value)	○
	setBytes(java.lang.String name, byte[] value, int offset, int length)	○
	setChar(java.lang.String name, char value)	○
	setDouble(java.lang.String name, double value)	○
	setFloat(java.lang.String name, float value)	○
	setInt(java.lang.String name, int value)	○
	setLong(java.lang.String name, long value)	○
	setObject(java.lang.String name, java.lang.Object value)	○
	setShort(java.lang.String name, short value)	○
	setString(java.lang.String name, java.lang.String value)	○

○：サポート
×：未サポート

注1）“Interstage Application Server”が返ります。

24.13.3 パッケージjavax.jmsのAPI一覧（その3）

インタフェース名／クラス名	メソッド	サポート
Message	acknowledge()	○
	clearBody()	○
	clearProperties()	○
	getBooleanProperty(java.lang.String name)	○
	getByteProperty(java.lang.String name)	○
	getDoubleProperty(java.lang.String name)	○
	getFloatProperty(java.lang.String name)	○
	getIntProperty(java.lang.String name)	○
	getJMSCorrelationID()	○
	getJMSCorrelationIDAsBytes()	×
	getJMSDeliveryMode()	○
	getJMSDestination()	○
	getJMSExpiration()	○
	getJMSMessageID()	○
	getJMSPriority()	○
	getJMSRedelivered()	○
	getJMSReplyTo()	○
	getJMSTimestamp()	○
	getJMSType()	○
	getLongProperty(java.lang.String name)	○
	getObjectProperty(java.lang.String name)	○

インタフェース名／クラス名	メソッド	サポート
	getPropertyNames()	○
	getShortProperty(java.lang.String name)	○
	getStringProperty(java.lang.String name)	○
	propertyExists(java.lang.String name)	○
	setBooleanProperty(java.lang.String name, boolean value)	○
	setByteProperty(java.lang.String name, byte value)	○
	setDoubleProperty(java.lang.String name, double value)	○
	setFloatProperty(java.lang.String name, float value)	○
	setIntProperty(java.lang.String name, int value)	○
	setJMSCorrelationID(java.lang.String correlationID)	○
	setJMSCorrelationIDAsBytes(byte[] correlationID)	×
	setJMSDeliveryMode(int deliveryMode)	○
	setJMSDestination(Destination destination)	○
	setJMSExpiration(long expiration)	○
	setJMSMessageID(java.lang.String id)	○
	setJMSPriority(int priority)	○
	setJMSRedelivered(boolean redelivered)	○
	setJMSReplyTo(Destination replyTo)	○
	setJMSTimestamp(long timestamp)	○
	setJMSType(java.lang.String type)	○
	setLongProperty(java.lang.String name, long value)	○
	setObjectProperty(java.lang.String name, java.lang.Object value)	○
	setShortProperty(java.lang.String name, short value)	○
	setStringProperty(java.lang.String name, java.lang.String value)	○

○：サポート
×：未サポート

24.13.4 パッケージjavax.jmsのAPI一覧（その4）

インタフェース名／クラス名	メソッド	サポート
MessageConsumer	close()	○
	getMessageListener()	○
	getMessageSelector()	○
	receive()	○
	receive(long timeout)	○（注1）
	receiveNoWait()	○（注2）
	setMessageListener(MessageListener listener)	○
MessageListener	onMessage(Message message)	○
MessageProducer	close()	○

インタフェース名／クラス名	メソッド	サポート
	getDeliveryMode()	○
	getDestination()	○（注3）
	getDisableMessageID()	○
	getDisableMessageTimestamp()	○
	getPriority()	○
	getTimeToLive()	○
	send(Destination destination, Message message)	○（注3）
	send(Destination destination, Message message, int deliveryMode, int priority, long timeToLive)	○（注3）
	send(Message message)	○（注3）
	send(Message message, int deliveryMode, int priority, long timeToLive)	○（注3）
	setDeliveryMode(int deliveryMode)	○（注4）
	setDisableMessageID(boolean value)	○
	setDisableMessageTimestamp(boolean value)	○
	setPriority(int defaultPriority)	○
	setTimeToLive(long timeToLive)	○（注5）
ObjectMessage	getObject()	○
	setObject(java.io.Serializable object)	○

○：サポート
×：未サポート

注1）メッセージの待ち合わせを行う必要がない場合は、receive()メソッドではなく、receiveNoWait()メソッドを使用して、メッセージの待ち合わせを行わない運用を行ってください。
receive()メソッドを使用すると、メッセージの受信を待合わせて、受信タイムアウト時間に達すると、nullで復帰します。受信タイムアウト時間は、receive()メソッドの指定値、およびイベントチャネルの動作環境の設定値「メッセージ(イベントデータ)の待ち合わせ時間」により以下のように算出します。なお、イベントチャネル動作環境の設定方法については、“23.1.4 イベントチャネル動作環境の変更”を参照してください。

```
受信タイムアウト時間 ＝ メッセージの待ち合わせ時間 × n
n ＝ receive()メソッドの指定値 ／ （メッセージの待ち合わせ時間 × 1000）
n： 小数点以下、整数に切り上げ
```

例

receive()メソッドの指定値“50000ミリ秒”、メッセージの待ち合わせ時間“40秒”の場合

```
n ＝ 50000 ／ （40 × 1000）≒ 2 （小数点以下、整数に切り上げ）
受信タイムアウト時間 ＝ 40 × 2 ＝ 80（秒）
```

注2）受信できるメッセージがない場合は、nullで即時に復帰します。

注3）JMS1.1規約で追加されました。

注4）イベントチャネルの配信モードおよび同一モードだけをサポートします。

注5）timeToLiveは、ミリ秒単位までサポートします。しかし、イベントチャネルのメッセージタイムアウト時間の精度は秒単位であるため、timeToLive値は最も近い値に丸められます。

24.13.5 パッケージjavax.jmsのAPI一覧（その5）

インタフェース名／クラス名	メソッド	サポート
Queue	getQueueName()	○
	toString()	○（注1）
QueueBrowser	close()	○
	getEnumeration()	○
	getMessageSelector()	○
	getQueue()	○
QueueConnection	createConnectionConsumer(Queue queue, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)	○
	createQueueSession(boolean transacted, int acknowledgeMode)	○
QueueConnectionFactory	createQueueConnection()	○
	createQueueConnection(java.lang.String userName, java.lang.String password)	○（注2）
QueueReceiver	getQueue()	○
QueueRequestor	close()	○
	request(Message message)	○
QueueSender	getQueue()	○
	send(Message message)	○
	send(Message message, int deliveryMode, int priority, long timeToLive)	○（注3）
	send(Queue queue, Message message)	○
	send(Queue queue, Message message, int deliveryMode, int priority, long timeToLive)	○（注3）
QueueSession	createBrowser(Queue queue)	○
	createBrowser(Queue queue, java.lang.String messageSelector)	○
	createQueue(java.lang.String queueName)	○
	createReceiver(Queue queue)	○
	createReceiver(Queue queue, java.lang.String messageSelector)	○
	createSender(Queue queue)	○
	createTemporaryQueue()	○

○：サポート
×：未サポート

注1）“com.fujitsu.interstage.jms:Queue名::イベントチャネルのグループ名::イベントチャネルのチャネル名”が返ります。

注2）userNameおよびpasswordは、無視されます。

注3）timeToLiveは、ミリ秒単位までサポートします。しかし、イベントチャネルのメッセージタイムアウト時間の精度は秒単位であるため、timeToLive値は最も近い値に丸められます。

24.13.6 パッケージjavax.jmsのAPI一覧（その6）

インタフェース名／クラス名	メソッド	サポート
ServerSession	getSession()	○
	start()	○
ServerSessionPool	getServerSession()	○
Session	close()	○
	commit()	○
	createBrowser(Queue queue)	○（注1）
	createBrowser(Queue queue, java.lang.String messageSelector)	○（注1）
	createBytesMessage()	○
	createConsumer(Destination Destination)	○（注1）
	createConsumer(Destination destination, java.lang.String messageSelector)	○（注1）
	createConsumer(Destination destination, java.lang.String messageSelector, boolean NoLocal)	○（注1）
	createDurableSubscriber(Topic topic, java.lang.String name)	○（注1）
	createDurableSubscriber (Topic topic, java.lang.String name, java.lang.String messageSelector, boolean noLocal)	○（注1）
	createMapMessage()	○
	createMessage()	○
	createObjectMessage()	○
	createObjectMessage(java.io.Serializable object)	○
	createProducer(Destination destination)	○（注1）
	createQueue(java.lang.String queueName)	○（注1）
	createStreamMessage()	○
	createTemporaryQueue()	○（注1）
	createTemporaryTopic()	○（注1）
	createTextMessage()	○
	createTextMessage(java.lang.String text)	○
	createTopic(java.lang.String topicName)	○（注1）
	getAcknowledgeMode()	○（注1）
	getMessageListener()	○
	getTransacted()	○
	recover()	○
	rollback()	○
	run()	○
	setMessageListener(MessageListener listener)	○（注2）
	unsubscribe()	○（注1）

インタフェース名／クラス名	メソッド	サポート
	unsubscribe(java.lang.String name)	○（注1）
StreamMessage	readBoolean()	○
	readByte()	○
	readBytes(byte[] value)	○
	readChar()	○
	readDouble()	○
	readFloat()	○
	readInt()	○
	readLong()	○
	readObject()	○
	readShort()	○
	readString()	○
	reset()	○
	writeBoolean(boolean value)	○
	writeByte(byte value)	○
	writeBytes(byte[] value)	○
	writeBytes(byte[] value, int offset, int length)	○
	writeChar(char value)	○
	writeDouble(double value)	○
	writeFloat(float value)	○
	writeInt(int value)	○
	writeLong(long value)	○
	writeObject(java.lang.Object value)	○
	writeShort(short value)	○
	writeString(java.lang.String value)	○
TemporaryQueue	delete()	○
TemporaryTopic	delete()	○

○：サポート
×：未サポート

注1）JMS1.1規約で追加されました。

注2）1つのMessageConsumerで同時に同期、非同期のメッセージ受信を行うことはできません。

24.13.7 パッケージjavax.jmsのAPI一覧（その7）

インタフェース名／クラス名	メソッド	サポート
TextMessage	getText()	○
	setText(java.lang.String string)	○
Topic	getTopicName()	○
	toString()	○（注1）

インタフェース名／クラス名	メソッド	サポート
TopicConnection	createConnectionConsumer(Topic topic, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)	○
	createDurableConnectionConsumer(Topic topic, java.lang.String subscriptionName, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)	○
	createTopicSession(boolean transacted, int acknowledgeMode)	○
TopicConnectionFactory	createTopicConnection()	○
	createTopicConnection(java.lang.String userName, java.lang.String password)	○（注2）
TopicPublisher	getTopic()	○
	publish(Message message)	○
	publish(Message message, int deliveryMode, int priority, long timeToLive)	○（注3）
	publish(Topic topic, Message message)	○
	publish(Topic topic, Message message, int deliveryMode, int priority, long timeToLive)	○（注3）
TopicRequestor	close()	○
	request(Message message)	○
TopicSession	createDurableSubscriber(Topic topic, java.lang.String name)	○
	createDurableSubscriber(Topic topic, java.lang.String name, java.lang.String messageSelector, boolean noLocal)	○
	createPublisher(Topic topic)	○
	createSubscriber(Topic topic)	○
	createSubscriber(Topic topic, java.lang.String messageSelector, boolean noLocal)	○
	createTemporaryTopic()	○
	createTopic(java.lang.String topicName)	○
	unsubscribe(java.lang.String name)	○
TopicSubscriber	getNoLocal()	○
	getTopic()	○

○：サポート
×：未サポート

注1）“com.fujitsu.interstage.jms:Topic名::イベントチャネルのグループ名::イベントチャネルのチャネル名”が返ります。

注2）userNameおよびpasswordは、無視されます。

注3）timeToLiveは、ミリ秒単位までサポートします。しかし、イベントチャネルのメッセージタイムアウト時間の精度は秒単位であるため、timeToLive値は最も近い値に丸められます。

24.13.8 パッケージjavax.jmsのAPI一覧（その8）

インタフェース名／クラス名	メソッド	サポート
XAConnection	createSession(boolean transacted, int acknowledgeMode)	×

インタフェース名／クラス名	メソッド	サポート
	createXASession()	×
XAConnectionFactory	createXAConnection()	×
	createXAConnection(java.lang.String userName, java.lang.String password)	×
XAQueueConnection	createQueueSession(boolean transacted, int acknowledgeMode)	×
	createXAQueueSession()	×
XAQueueConnectionFactory	createXAQueueConnection()	×
	createXAQueueConnection(java.lang.String userName, java.lang.String password)	×
XAQueueSession	getQueueSession()	×
XASession	commit()	×
	getSession()	×
	getTransacted()	×
	getXAResource()	×
	rollback()	×
XATopicConnection	createTopicSession(boolean transacted, int acknowledgeMode)	×
	createXATopicSession()	×
XATopicConnectionFactory	createXATopicConnection()	×
	createXATopicConnection(java.lang.String userName, java.lang.String password)	×
XATopicSession	getTopicSession()	×

○：サポート
×：未サポート

注）XAインタフェースは、未サポートです。

第7部 connector編

第25章　Interstage connectorの基本機能

本章では、Interstage connectorの基本機能について説明します。

注意

- connector1.5規約に準拠したリソースアダプタはIJServerタイプが以下の場合のみ使用できます。connector1.5規約に準拠したリソースアダプタを以下以外のIJServerタイプのIJServerに配備しようとした場合にはエラーとなります。

  ー　WebアプリケーションとEJBアプリケーションを同一JavaVMで運用

- IJServerに対して配備する方法とIJServerを指定せずにリソースとして配備する方法がありますが、connector1.5規約に準拠したリソースアダプタはIJServerに対して配備して使用してください。リソースとして配備した場合にはエラーとなります。

25.1 リソースアダプタの種別

リソースアダプタにはJ2EEアプリケーションからの要求をEISに通知する同期型のリソースアダプタ(これを**アウトバウンド・リソースアダプタ**と呼びます)と、EISから非同期で通知されるメッセージをJ2EEアプリケーションで処理することが可能なリソースアダプタ(これを**インバウンド・リソースアダプタ**と呼びます)があります。リソースアダプタによってはリソースアダプタ固有の情報を管理する**管理対象オブジェクト**が定義されている場合があります。

以下にそれぞれについて説明します。

アウトバウンド・リソースアダプタ

アウトバウンド・リソースアダプタはアプリケーションと同期して動作するリソースアダプタです。リソースアダプタはアプリケーションからの処理要求を受けてEISの処理を実行します。
アウトバウンド・リソースアダプタを使用する場合には、Interstage管理コンソールを使用してConnectionFactoryの定義名を指定してください。アプリケーションはJNDIのlookupメソッドに定義名を指定して実行することでConnectionFactoryを取得し、EISへの処理要求を実行します。JNDIを使用することで処理要求はIJServerによって接続管理、トランザクション管理、セキュリティ管理が行われます。

インバウンド・リソースアダプタ

connector1.5規約以降で追加されたリソースアダプタです。インバウンド・リソースアダプタはアプリケーションとは非同期で動作し、リソースアダプタが必要に応じてアプリケーションに処理を要求します。リソースアダプタはEISからの処理要求を受けて必要なタイミングでMessage-driven Beanを呼び出してアプリケーションサーバに処理要求を配信します。配信されたメッセージはMessage-driven Beanの各種定義に従ってEJBコンテナによって管理されます。

Message-driven Beanとの連携

インバウンド・リソースアダプタの受信アプリケーションはMessage-driven Beanで作成できます。EJB2.1規約以降のMessage-driven Beanは受信するメッセージの形態(これを**受信対象種別**と呼びます)として“JMS”または“resourceadapter”を選択できます。“resourceadapter”を選択した場合には、受信対象とするリソースアダプタのリソースアダプタ名を選択します。Message-driven Beanは選択したリソースアダプタのメッセージリスナインタフェースに合わせて実装されている必要があります。Message-driven Beanの実装方法については“第14章 Message-driven Beanの実装”を参照してください。

管理対象オブジェクト

connector1.5規約以降のリソースアダプタには管理対象オブジェクトが定義されている場合があります。リソースアダプタはConnectionFactoryとは別にリソースアダプタ固有の情報を管理することが必要な場合があります。リソースアダプタにはこの固有の情報を管理するオブジェクトを管理対象オブジェクトとして定義することができます。管理対象オブジェクトを使用する場合には、Interstage管理コンソールを使用して管理対象オブジェクトの定義名を指定してください。アプリケーションはJNDIのlookupメソッドに定義名を指定して実行することで管理対象オブジェクトを取得し、処理を実行します。

注意

定義名の設定時の注意事項

定義名を設定する際には、以下の名称間で一意の名前をつける必要があります。

- connector1.0規約のリソースアダプタの定義名
- connector1.5規約のConnectionFactoryの定義名
- connector1.5規約の管理対象オブジェクトの定義名

また、connector1.0規約のリソースアダプタの定義名は、connector1.5規約のリソースアダプタのリソースアダプタ名とも重複しないような一意の名前を設定してください。使用可能な文字種は、Interstage管理コンソールのヘルプ画面を参照してください。

25.2 リソースアダプタの起動/停止

リソースアダプタによってIJServerに連動してリソースアダプタの起動と停止処理が実行される場合があります。

connector1.5規約以降のリソースアダプタではdeployment descriptorファイル(ra.xml)にResourceAdapterクラスを定義することができます。ResourceAdapterクラスにはstartメソッド、stopメソッド、endpointActivationメソッド、endpointDeactivationメソッドが実装されます。
ResourceAdapterクラスが定義されたリソースアダプタを配備した場合、IJServerの起動または停止時(HotDeploy機能使用時には配備、配備解除または再活性時)に、それぞれのメソッドが呼び出されて、非同期メッセージの受信処理の開始または停止などを行います。

以下にResourceAdapterクラスのメソッドが実行されるタイミングを記載します。各モジュールが活性化もしくは非活性化されるタイミングについては、“3.5.3 J2EEのHotDeploy機能”を参照してください。

メソッド名	呼び出しタイミング
start	・ IJServer起動時 ・ RARファイルに含まれるクラスがオートリロードされた時 以下はHotDeploy機能を使用した場合 ・ 起動しているIJServerに対してRARファイル(またはRARファイルを含むEARファイル)を配備した時 ・ 起動しているIJServerにおいてRARファイル(またはRARファイルを含むEARファイル)が活性化された時
stop	・ IJServer停止時 ・ RARファイルに含まれるクラスがオートリロードされた時 以下はHotDeploy機能を使用した場合 ・ 起動しているIJServerに対してRARファイル(またはRARファイルを含むEARファイル)を配備解除した時 ・ 起動しているIJServerにおいてRARファイル(またはRARファイルを含むEARファイル)が非活性化された時
endpointActivation	・ IJServer起動時 ・ リソースアダプタを受信対象とするMessage-driven Beanを含むejb-jarファイルのクラスがオートリロードされた時 以下はHotDeploy機能を使用した場合 ・ 起動しているIJServerに対してリソースアダプタを受信対象とするMessage-driven Beanを含むejb-jarファイル(またはejb-jarファイルを含むEARファイル)を配備した時 ・ 起動しているIJServerにおいてリソースアダプタを受信対象とするMessage-driven Beanを含むejb-jarファイル(またはejb-jarファイルを含むEARファイル)が活性化された時

メソッド名	呼び出しタイミング
endpointDeactivation	・ IJServer停止時 ・ リソースアダプタを受信対象とするMessage-driven Beanを含むejb-jarファイルのクラスがオートリロードされた時 以下はHotDeploy機能を使用した場合 ・ 起動しているIJServerに対してリソースアダプタを受信対象とするMessage-driven Beanを含むejb-jarファイル(またはejb-jarファイルを含むEARファイル)を配備解除した時 ・ 起動しているIJServerにおいてリソースアダプタを受信対象とするMessage-driven Beanを含むejb-jarファイル(またはejb-jarファイルを含むEARファイル)が非活性化された時

注意

再配備時の注意事項

再配備を行った場合には、ConnectionFactoryおよび管理対象オブジェクトの定義名を再定義してください。

25.3 接続管理

Interstage connectorはJNDIよりconnectorのコネクションファクトリを取得して、connectorのリソースに接続する機能があります。

Interstage connectorは一度接続したconnectorへの接続情報をプール管理します。これにより、多数のクライアントからのリソースアクセス、またはリソースへの頻繁なアクセスが必要なアプリケーション環境の構築ができます。

プーリングしたコネクションの時間監視

プールされたコネクションはコンテナが時間監視(タイムアウト)します。使用されずにタイムアウト時間を超過したコネクションはコンテナが自動的に解放します。
以下のファイルに値を設定することによってタイムアウト時間を変更できます。単位は秒、デフォルト値は600(秒)です。

Windows32/64

C:¥Interstage¥J2EE¥etc¥JCA¥jca.properties

Solaris64 Linux32/64

/opt/FJSVj2ee/etc/jca/jca.properties

上記ファイルに以下のプロパティを設定してください。デフォルトで600が設定されています。

プロパティ名	プロパティ値
idle.resource.threshold	タイムアウト値

例

タイムアウト時間を300秒に設定する場合の例を以下に記述します。

```
idle.resource.threshold=300
```

25.4 トランザクション管理

Interstage connectorはEJBが提供するトランザクション機能を使用して、リソースのトランザクションを管理する機能があります。この機能を使用して、複数のリソースマネージャにまたがるトランザクションを管理できます。

サポートされているトランザクション サポートレベル

Interstage connectorではresource adapterの各トランザクションレベルをサポートしています。

resource adapterのトランザクションサポートレベルはdeployment descriptorのtransaction-supportタグに定義されています。ここに指定された値によって以下のようにresource adapterがサポートするトランザクションレベルに違いがあります。

トランザクション種別	使用用途
Windows32/64 Linux32/64 XA トランザクションサポート	transaction-supportタグに“XATransaction”が指定されます。 EJBの分散トランザクション機能を使用して、resource adapter以外のリソースとも連携したトランザクション管理ができます。トランザクションは2フェーズコミットプロトコルで処理されます。
ローカル トランザクションサポート	transaction-supportタグに“LocalTransaction”が指定されます。 XA トランザクションと違い、2 フェーズ コミットプロトコル(2PC)に関わることができません。必ず1フェーズコミットプロトコルで処理されるため、1トランザクションでは1つのリソースのみアクセスすることが推奨されます。
トランザクション非サポート	transaction-supportタグに“NoTransaction”が指定されます。 このトランザクション種別がサポートされたresource adapterはトランザクション連携されません。

XAトランザクションまたはローカルトランザクションをサポートするresource adapterを使用する場合は、コンテナが制御するトランザクション(トランザクション属性に“Container”を指定)と連携できます。

注意

トランザクション機能を使用する場合の注意事項

XAトランザクションを使用する場合、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBコンテナ設定] > [分散トランザクションを使用する]で“使用する”に設定してください。ただし、IJServerのタイプが“WebアプリケーションとEJBアプリケーションを同一JavaVMで運用”の場合は、選択できません。

25.5 セキュリティ管理

Interstage connectorではEISへの安全なセキュリティ管理機能をサポートします。この機能によりEISに対する安全が保障され、EISが管理するリソースが保護されます。この機能はEJBのリソース接続者管理機能を使用します。

サインオン方法	リソース接続者の指定	動作
アプリケーション管理によるサインオン	Application	アプリケーションで指定したユーザID/パスワードでリソースにアクセスします。アプリケーションで設定した情報は、以下の2つのパターンがあります。 ・ EJBアプリケーションからリソース接続する場合で、かつ、EJBアプリケーションにリソース接続者が指定されている場合 EJBアプリケーションで指定したユーザIDとパスワードが使用されます。 ・ その他の場合 J2EEアプリケーションクライアント、または、Webアプリケーションでユーザ認証されたユーザIDとパスワードが使用されます。
コンテナ管理によるサインオン	Container	コンテナが以下で設定されたユーザID/パスワードで接続します。 ・ Interstage管理コンソール

25.6 Work管理

Interstage connectorはWork(アプリケーションの呼び出し、ネットワーク・エンドポイントの監視、入力データの処理など)を同期/非同期に実行するためのWork管理機能をサポートします。Work管理機能の使用により、Interstageの提供するスレッド制御機構を利用したアプリケーション構築が可能となります。この機能はconnector1.5規約に準拠したリソースアダプタで利用可能です。
リソースアダプタはIJServerが提供するWorkManagerクラスの実装に対して、scheduleWorkメソッドなどを使用してWorkを登録します。詳細はconnector1.5規約を参照してください。

Workに割り当て可能な同時実行スレッド数

1つのWorkに対し1つのスレッドが割り当てられます。その同時実行可能なスレッド数のチューニングパラメタとして、最小値・最大値・アイドルタイムアウトが指定できます。
以下のファイルに値を設定することによって、Workに割り当て可能な同時実行スレッド数を変更できます。

Windows32/64

C:¥Interstage¥J2EE¥etc¥JCA¥jca.properties

Solaris64 Linux32/64

/opt/FJSVj2ee/etc/jca/jca.properties

上記ファイルに以下のプロパティを設定してください。パラメタを指定しなかった場合、および指定可能範囲外の値を設定した場合には、デフォルト値が設定されます。

チューニングパラメタ	プロパティ名	意味
最小値	min-thread-size	Workに割り当て可能な同時実行スレッド数の最小値です。 指定可能な値は1～2147483647の整数値。デフォルト値は1。
最大値	max-thread-size	Workに割り当て可能な同時実行スレッド数の最大値です。 指定可能な値は1～2147483647の整数値。デフォルト値は64。 なお、最小値よりも小さい値を指定した場合、最小値と同じ値が最大値として設定されます。
アイドルタイムアウト	thread-idle-timeout	アイドルタイムアウト値(秒)です。プールに返却されたスレッドが、本項目で指定した時間を超過しても使用されない場合に破棄されます。ただし、初期起動スレッド数分は破棄対象外です。 指定可能な値は0～2147483647の整数値。デフォルト値は600。 0を指定した場合、タイムアウトしません。

例

Workに割り当て可能な同時実行スレッド数の最小値を2、最大値を10、アイドルタイムアウトを1000に設定した場合の例を以下に記述します。

```
min-thread-size=2
max-thread-size=10
thread-idle-timeout=1000
```

第26章　connectorアプリケーションの開発

リソースアダプタへアクセスするEJBアプリケーションまたはWebアプリケーションを作成します。

EISとのコネクションを確立するために、javax.naming.InitialContextクラスのlookupメソッドを使用してコネクションファクトリを獲得します。このlookupメソッドの引数には、“java:comp/env/eis/＊＊＊”という文字列を渡してください。＊＊＊には、配備時に指定したリソース名を指定します。

以下に、CCI（ユーザアプリケーションとresource adapterとの間の規約化されたインタフェース）のインタフェース（javax.resource.cciパッケージのインタフェース）を使用したユーザアプリケーションの例を記載します。

```
// JNDIからConnectionFactoryオブジェクトを取得
javax.naming.InitialContext ic = new javax.naming.InitialContext();
javax.resource.cci.ConnectionFactory cf =
(javax.resource.cci.ConnectionFactory) ic.lookup("java:comp/env/eis/RA01");

// ConnectionFactoryオブジェクトからConnectionオブジェクトを取得
javax.resource.cci.Connection con = cf.getConnection();

// Connectionオブジェクトから Interactionオブジェクトを作成
javax.resource.cci.Interaction ix = con.createInteraction();

CciInteractionSpec iSpec = new CciInteractionSpec();  //各リソース提供者もしくは
iSpec.setFunctionName("INSERTCOFFEE");          //サードベンダが提供している
iSpec.setSchema(user);                                  //resource adapterによって
iSpec.setCatalog(null);                                   //処理が変わります

// 処理に必要な情報を設定
javax.resource.cci.RecordFactory rf = cf.getRecordFactory();
javax.resource.cci.IndexedRecord iRec = rf.createIndexedRecord("InputRecord");
iRec.add(name);
iRec.add(new Integer(qty));

// 処理の実行
javax.resource.cci.Record rec = ix.execute( iSpec, iRec );

// Interactionの開放
ix.close();

// Connectionの開放
con.close();
```

26.1 インタフェース

connectorの標準インタフェースについてはJ2EE規約を参照してください。

resource adapterのインタフェースについてはresource adapterの仕様書を参照してください。

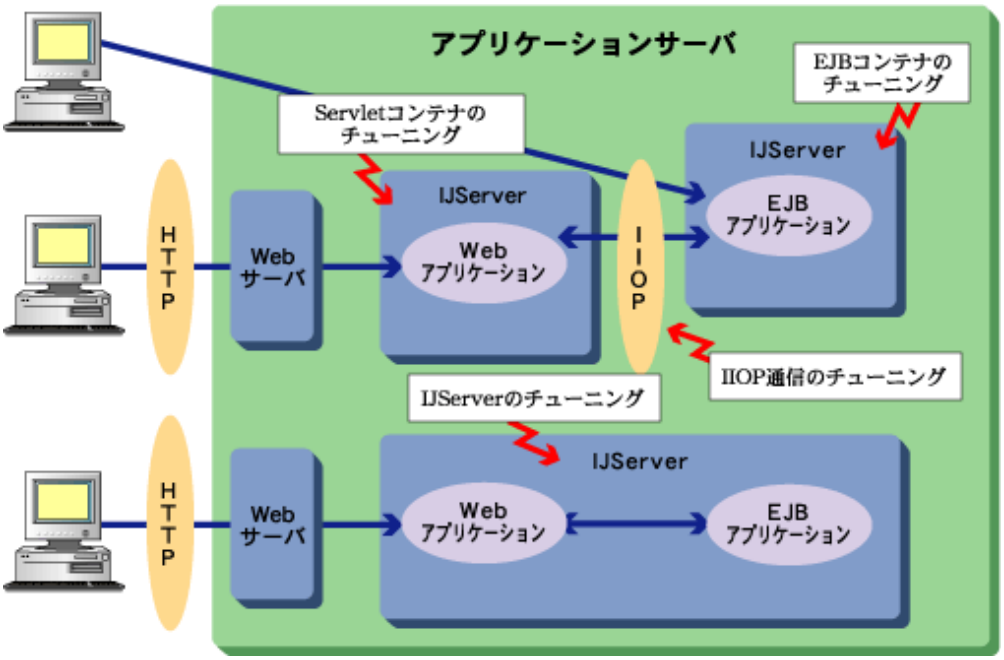
第8部 チューニング

第27章　J2EEのチューニング

Interstageはシステム規模の指定だけで、システム運用が可能となるようなモデルケースを設定して各サービス定義を登録しています。J2EEアプリケーションを動作させるためには、これらの定義に加え、J2EEを構成する各コンポーネントでチューニングが必要となります。

ここでは、下記のようなJ2EEアプリケーションの形態を例に、チューニングに関する設定を説明します。また、J2EEアプリケーションのチューニングに役立つ“27.1 J2EEモニタロギング機能”についても説明します。

- IJServerのチューニング
- Servletコンテナのチューニング
- EJBコンテナのチューニング
- IIOP通信のチューニング
  詳細は、「チューニングガイド」－「CORBAサービスの動作環境ファイル」を参照してください。
- CORBAサービスのチューニング
- LDAPサーバとしての、ディレクトリサービスのチューニング



27.1 J2EEモニタロギング機能

J2EEモニタロギング機能とは、IJServerの性能情報をロギングする機能です。この機能を利用して、JavaVMやJDBCデータソースなどの性能情報を定期的に採取し、結果をログファイルへ出力することができます。ログファイルはCSV形式のファイルに出力されるため、容易にMicrosoft(R) Excelなどで読み込んで性能情報を分析することができ、統計情報の蓄積にも役立ちます。

isj2eemonitorコマンドを実行すると、isj2eemonitorコマンドはInterstage JMXサービスにロギングの要求を通知します。Interstage JMXサービスではロギングの開始要求を受け付けた場合には、指定された時間間隔で定期的にIJServerにログ出力要求を通知し、IJServerプロセス上で性能情報をCSV形式のファイルに出力します。ロギングの停止要求を受け付けた場合にはロギング処理を停止します。

27.1.1 J2EEモニタロギングの操作手順

J2EEモニタロギングの操作手順について、以下を説明します。

- J2EEモニタロギングの起動操作

■J2EEモニタロギングの起動操作

isj2eemonitorコマンドを以下のように実行して、J2EEモニタロギングを起動します。

```
isj2eemonitor -start -n IJServer名
```

性能監視対象のIJServerの起動前/起動後のどちらでもJ2EEモニタロギングを起動し監視を行うことが可能です。また、コマンド実行時にログ採取間隔や採取する情報などを指定することもできます。詳細は“リファレンスマニュアル(コマンド編)”を参照してください。

IJServer起動前にロギングを開始した場合、初回採取データとしては、IJServer起動時から次回のログ採取間隔経過時の集計情報が出力されます。また、IJServerが停止されると、その時点でログ出力は停止します。

■J2EEモニタロギングの停止操作

isj2eemonitorコマンドを以下のように実行して、J2EEモニタロギングを停止します。

```
isj2eemonitor -stop -n IJServer名
```

また、以下のサービスを停止した場合も、J2EEモニタロギングが停止します。

Windows32/64

“Interstage Operation Tool”サービス

Solaris64 Linux32/64

Interstage JMXサービス

■監視操作の流れ

◆特定時刻のログのみ採取する場合

トラブル調査などのため、ある特定の時間のみ性能情報を採取したい場合は、以下のようにIJServer起動後に採取したいタイミングでJ2EEモニタロギング機能を開始して性能情報を分析してください。

1. IJServerの起動
   Interstage管理コンソール、またはisstartwuコマンドでIJServerを起動します。
2. J2EEモニタロギングの開始
   isj2eemonitorコマンドでJ2EEモニタロギングを開始します。
3. 性能情報の分析
   出力された情報をMicrosoft(R) Excelなどで分析します。
4. J2EEモニタロギングの停止
   isj2eemonitorコマンドでJ2EEモニタロギングを停止します。
5. 2.～4.を繰り返します。
6. IJServerの停止
   Interstage管理コンソール、またはisstopwuコマンドでIJServerを停止します。

◆継続的にログを採取する場合

継続的にログを採取して性能チューニングの妥当性の検証を行いたい場合は、以下のようにJ2EEモニタロギング機能の起動後に、IJServerを起動して性能情報を分析してください。

1. J2EEモニタロギングの開始
   isj2eemonitorコマンドでJ2EEモニタロギングを開始します。

2. IJServerの起動
   Interstage管理コンソール、またはisstartwuコマンドでIJServerを起動します。
3. 性能情報の分析
   出力された情報をMicrosoft(R) Excelなどで分析します。
4. IJServerの停止
   Interstage管理コンソール、またはisstopwuコマンドでIJServerを停止します。
5. J2EEモニタロギングの停止
   isj2eemonitorコマンドでJ2EEモニタロギングを停止します。
6. 1.～5.を繰り返します。

27.1.2 J2EEモニタロギングのログファイル

J2EEモニタロギングのログファイルについて、以下を説明します。

- ログファイル名
- ログファイルの出力条件
- ロールオーバ後のファイル名
- ログファイルのライフサイクル
- ファイルのアクセス権
- 出力ディレクトリ

■ログファイル名

モニタ情報を出力するログファイルは、プロセスごと、かつ、ロギング対象ごとに出力されます。ログファイルのファイル名は以下の命名規則で出力されます。

monitor-[ロギング対象名].log

ファイル名一覧を以下に記載します。

ファイル	ファイル名
JavaVM情報ログファイル	monitor-JavaVM.log
データソース情報ログファイル	monitor-DataSource.log
トランザクション情報ログファイル	monitor-Transaction.log
Servletコンテナ情報ログファイル	monitor-ServletContainer.log
EJBコンテナ情報ログファイル	monitor-EJBContainer.log

■ログファイルの出力条件

ログファイルの作成と性能情報の出力は、指定されたIJServerが起動している場合のみ行われます。
また、IJServerのタイプにより、出力可能なログファイルの種類が異なります。IJServerタイプごとの出力可能ログファイルを以下に示します。

	Webアプリケーションと EJBアプリケーションを同一JavaVMで運用	Webアプリケーションと EJBアプリケーションを別JavaVMで運用	Webアプリケーションのみ運用	EJBアプリケーションのみ運用
JavaVM情報ログファイル	○	○	○	○

	Webアプリケーションと EJBアプリケーションを同一JavaVMで運用	Webアプリケーションと EJBアプリケーションを別JavaVMで運用	Webアプリケーションのみ運用	EJBアプリケーションのみ運用
データソース情報ログファイル	○ (注1)	○ (注1)	○ (注1)	○ (注1)
トランザクション情報ログファイル	○	○	○	○
Servletコンテナ情報ログファイル	○	○ (注2)	○	－
EJBコンテナ情報ログファイル	○	○ (注3)	－	○

注1) データソースを使用している場合のみ、データソース情報ログファイルが出力されます。
注2) Webアプリケーションを運用するJavaVMでのみ、Servletコンテナ情報ログファイルが出力されます。
注3) EJBアプリケーションを運用するJavaVMでのみ、EJBコンテナ情報ログファイルが出力されます。

■ロールオーバ後のファイル名

出力されたログファイルは一定間隔でロールオーバされます。ロールオーバ後のログファイルは、以下のようにロールオーバした日時情報を付加してバックアップされます。ログファイル名のロギング対象名(例:“JavaVM”)と拡張子“.log”の間に、日時を示す文字列が挿入されます。また、ロギング対象名と日時文字列の間はハイフン(“-”)で区切られます。

```
monitor-[ロギング対象名]-YYYY_MM_DD-hh_mm_ss.log
```

日時情報について以下に説明します。

YY YY	年を4桁の数字(0000～9999)で表示します。
MM	月を2桁の数字(01～12)で表示します。
DD	日を2桁の数字(01～31)で表示します。
hh	時を2桁の数字(00～23)で表示します。
mm	分を2桁の数字(00～59)で表示します。
ss	秒を2桁の数字(00～59)で表示します。

例) monitor-JavaVM-2006_06_24-01_00_00.log

■ログファイルのライフサイクル

ログファイルのライフサイクルについて説明します。
IJServerプロセスがログを出力する場合、ログ出力ディレクトリに“■ログファイル名”で説明しているファイルが存在しなければファイルを新規に作成します。また、ログファイルは以下の条件の場合にロールオーバされます。

ロールオーバ条件

- ロールオーバ時刻にIJServerプロセスが起動している場合
- ログが採取されるタイミングで、以前のログファイルがログ出力ディレクトリに残存しており、かつ、そのファイルの更新日時が前回のロールオーバ日時より前の場合

ロールオーバ開始時刻はJ2EEモニタロギングを開始する時に指定できます。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

ロールオーバは以下のように行われます。

1. バックアップされているファイルのファイル数がログファイルの世代数以上となっている場合にファイルの更新日時が古いファイルを削除します。バックアップされたファイルのファイル数が“世代数－1”となるまで削除します。
2. 既存のログファイルを“■ロールオーバ後のファイル名”で説明している名前に変名してバックアップします。
3. 新規に“■ログファイル名”で説明している名前の新規ファイルを作成します。

■ファイルのアクセス権 Solaris64 Linux32/64

出力されるファイルの所有者はIJServerの起動ユーザとなり、ファイルの権限は「644」となります。

■出力ディレクトリ

デフォルトのログファイル出力ディレクトリは以下です。

Windows32/64

[J2EE共通ディレクトリ]¥ijserver¥[IJServer名]¥log¥[プロセス通番]

Solaris64 Linux32/64

[J2EE共通ディレクトリ]/ijserver/[IJServer名]/log/[プロセス通番]

出力先はisj2eeadminコマンドのIJServer定義、またはInterstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [ログ出力ディレクトリ]で変更できます。

27.1.3 性能情報の分析と対処

ログファイルに採取した性能情報の分析方法と対処方法について説明します。

■出力情報

モニタ情報ログファイルは、以下のようにCSV形式(カンマ区切りで並べた形式)で出力します。

```
D1,D2,D3,D4,D5,・・・
```

■性能情報の項目内容

Interstage管理コンソールで参照可能なモニタ情報はIJServer起動時からの集計データを表示していましたが、J2EEモニタロギング機能で採取されるデータは性能ボトルネックなどの解析に使用可能とするためデータ採取区間内での集計データ(データ採取時間帯での最大値など)を採取します。

また、Interstage管理コンソールの場合、一部のデータ(JavaVMなど)を除いてIJServer全体のデータを集計してモニタ情報として参照できましたが、J2EEモニタロギング機能では以下のようにIJServerプロセスごとのデータを採取可能であるため、より詳細な分析が可能となります。

	Interstage管理コンソール	J2EEモニタロギング
上限値	IJServer全体で使用可能な上限値	各プロセスで使用可能な上限値
最大値	IJServer全体の最大値	各プロセスの最大値
最小値	IJServer全体の最小値	各プロセスの最小値
平均値	IJServer全体の平均値	各プロセスの平均値
現在値	IJServer全体の現在値	各プロセスの現在値

性能情報中の日時情報は、以下の値が“DD/MM/YYYY hh:mm:ss:SSS”のフォーマットで出力されます。

DD　日を2桁の数字(01～31)で表示します。

MM　月を2桁の数字(01～12)で表示します。

YY　年を4桁の数字(0000～9999)で表示します。
YY

hh　時を2桁の数字(00～23)で表示します。

mm　分を2桁の数字(00～59)で表示します。

ss　秒を2桁の数字(00～59)で表示します。

SSS　ミリ秒を3桁の数字(000～999)で表示します。

例) 24/06/2006 01:00:00:200

性能情報中の「ミリ秒」単位は、以下の値が“h:mm:ss:SSS”のフォーマットで出力されます。

h　時を可変桁の数字(0～)で表示します。値が0の場合は“0”と表示します。

mm　分を2桁の数字(00～59)で表示します。

ss　秒を2桁の数字(00～59)で表示します。

SSS　ミリ秒を3桁の数字(000～999)で表示します。

例) 0:00:00:200

性能情報として出力される項目について説明します。各表の項番に書かれているD1、D2、・・・は、CSV形式で出力されるD1、D2、・・・に対応しています。

1)JavaVM情報

項番	項目名	単位	内容
D1	データ採取開始日時	－	当該レコードの性能情報の測定を開始した日時
D2	データ採取終了日時	－	当該レコードの性能情報の測定を終了した日時
D3	プロセス通番	－	IJServerが起動したプロセスの通番
D4	プロセスID	－	測定対象のIJServerのプロセスID
D5	コンテナタイプ	－	JavaVMのコンテナタイプ。以下のいずれかを出力します。 ・1VM(ServletとEJBを運用するプロセス) ・Web(Servletのみ運用するプロセス) ・EJB(EJBのみ運用するプロセス)
D6	JavaVMの運用時間	－	IJServer起動時からの総時間
D7	ヒープ現在使用量	KB	ログ採取時のJavaVMのヒープ使用量
D8	ヒープ最小使用量	KB	測定時間内でのJavaVMの最小ヒープ使用量(注1)
D9	ヒープ最大使用量	KB	測定時間内でのJavaVMの最大ヒープ使用量(注1)
D10	ヒープ上限値	KB	ヒープサイズの上限値(-Xmxオプションで指定した値とほぼ同等の値)
D11	メタスペース現在使用量	KB	ログ採取時のJavaVMのメタスペース使用量
D12	メタスペース最小使用量	KB	測定時間内でのJavaVMの最小メタスペース使用量 (注1)
D13	メタスペース最大使用量	KB	測定時間内でのJavaVMの最大メタスペース使用量 (注1)
D14	メタスペース上限値	KB	メタスペースサイズの上限値(-XX:MaxMetaspaceSizeオプションで指定した値とほぼ同等の値)(注2)
D15	ガーベジコレクション発生回数	回	測定時間内でのガーベジコレクションの発生回数 (注3)

項番	項目名	単位	内容
D16	ガーベジコレクション処理トータル時間	ミリ秒	測定時間内でのガーベジコレクションの処理時間の合計値 (注3)

注1) JavaVMヒープ/メタスペースの最小値/最大値については、Interstage管理コンソールのモニタ表示画面と同様に、3秒間隔でサンプリングした値の最小値/最大値を出力します。
注2) -XX: MaxMetaspaceSizeオプションが指定されていない場合は0が出力されます。0は無制限を意味します。
注3) ガーベジコレクションは、Full GCの情報を使用しています。

2)データソース情報

項番	項目名	単位	内容
D1	データ採取開始日時	―	当該レコードの性能情報の測定を開始した日時
D2	データ採取終了日時	―	当該レコードの性能情報の測定を終了した日時
D3	プロセス通番	―	IJServerが起動したプロセスの通番
D4	プロセスID	―	測定対象のIJServerのプロセスID
D5	データソース名	―	データソース名
D6	物理コネクション現在数	個	ログ採取時に確立されている物理コネクション数 (注1)
D7	物理コネクション最小数	個	測定時間内で同時に確立したデータベースの最小物理コネクション数 (注1)
D8	物理コネクション最大数	個	測定時間内で同時に確立したデータベースの最大物理コネクション数 (注1)
D9	物理コネクション上限数	個	確立可能なデータベースへの上限物理コネクション数 (注1)
D10	使用コネクション現在数	個	ログ採取時に使用されているコネクション数 (注1)
D11	使用コネクション最小数	個	測定時間内でプーリングされているコネクションで同時に使用された最小コネクション数 (注1)
D12	使用コネクション最大数	個	測定時間内でプーリングされているコネクションで同時に使用された最大コネクション数 (注1)
D13	累積コネクション待ち回数	回	測定時間内でコネクション待ちとなった回数(コネクション待ちタイムアウトが発生した場合は測定対象外) (注2)
D14	平均コネクション待ち時間	ミリ秒	測定時間内でコネクション待ちとなった時の平均待ち時間(コネクション待ちタイムアウトが発生した場合は測定対象外) (注2)
D15	コネクション待ち最小時間	ミリ秒	測定時間内でコネクション待ちとなった時の最小待ち時間(コネクション待ちタイムアウトが発生した場合は測定対象外) (注2)
D16	コネクション待ち最大時間	ミリ秒	測定時間内でコネクション待ちとなった時の最大待ち時間(コネクション待ちタイムアウトが発生した場合は測定対象外) (注2)
D17	コネクション待ち現在スレッド数	個	ログ採取時にコネクション待ちとなっているスレッド数 (注2)
D18	コネクション待ち最小スレッド数	個	測定時間内でコネクション待ちとなった時の最小待ちスレッド数 (注2)

項番	項目名	単位	内容
D19	コネクション待ち最大スレッド数	個	測定時間内でコネクション待ちとなった時の最大待ちスレッド数 (注2)
D20	コネクション待ちタイムアウト回数	回	測定時間内でコネクション待ちとなった時のコネクション待ちタイムアウトが発生した回数 (注2)
D21	物理コネクション確立回数	回	測定時間内でデータベースの物理コネクションを確立した回数 (注2)
D22	物理コネクション最大確立時間	ミリ秒	測定時間内でプーリングしていたコネクションのデータベースへの最大確立時間 (注2)
D23	アイドルタイムアウトによるクローズ回数	回	測定時間内でアイドルタイムアウトによりクローズした物理コネクション数 (注2)
D24	例外発生によるクローズ回数	回	測定時間内でコネクション接続時もしくはクローズ時に例外発生によりクローズした物理コネクション数 (注2)
D25	コネクション獲得回数	回	測定時間内でアプリケーションがgetConnectionメソッドを実行してコネクションを獲得した回数 (注3)
D26	コネクションクローズ回数	回	測定時間内でアプリケーションがcloseメソッドを実行してコネクションを解放した回数 (注3)
D27	通信待ちタイムアウト回数	回	測定時間内で通信待ちタイムアウトが発生した回数 (注3)
D28	コネクション平均使用時間	ミリ秒	測定時間内にアプリケーションでコネクションを使用した平均時間 (注3)
D29	コネクション最小使用時間	ミリ秒	測定時間内にアプリケーションでコネクションを使用した最小時間 (注3)
D30	コネクション最大使用時間	ミリ秒	測定時間内にアプリケーションでコネクションを使用した最大時間 (注3)
D31	コネクション使用タイムアウト回数	回	測定時間内でコネクション使用タイムアウトが発生した回数 (注3)

注1) InterstageでJDBCコネクションをプーリングしている、または、データベースがOracle10g以降かつJDBCドライバでJDBCコネクションをプーリングしている場合に、値を出力します。その他の場合は、“-”(ハイフン)が出力されます。
注2) InterstageでJDBCコネクションをプーリングしている場合のみ値を出力します。JDBCドライバでJDBCコネクションをプーリングしている場合、“-”(ハイフン)が出力されます。
注3) アプリケーションの前後で動作するコンテナ制御処理で使用されるコネクションの情報も合わせて表示されます。

※ 未使用のデータソースの性能情報は出力されません。

3)トランザクション情報

項番	項目名	単位	内容
D1	データ採取開始日時	－	当該レコードの性能情報の測定を開始した日時
D2	データ採取終了日時	－	当該レコードの性能情報の測定を終了した日時
D3	プロセス通番	－	IJServerが起動したプロセスの通番
D4	プロセスID	－	測定対象のIJServerのプロセスID
D5	実行トランザクション総数	個	測定時間内にアプリケーションで実行したJ2EEトランザクション数 (注1)

項番	項目名	単位	内容
D6	コミットトランザクション数	個	測定時間内にアプリケーションでコミットしたJ2EEトランザクション数 (注1)
D7	ロールバックトランザクション数	個	測定時間内にアプリケーションでロールバックしたJ2EEトランザクション数 (注1)
D8	トランザクション平均時間	ミリ秒	測定時間内にアプリケーションで実行したJ2EEトランザクションの平均時間 (注1)
D9	トランザクション最小時間	ミリ秒	測定時間内にアプリケーションで実行したJ2EEトランザクションの最小時間 (注1)
D10	トランザクション最大時間	ミリ秒	測定時間内にアプリケーションで実行したJ2EEトランザクションの最大時間 (注1)
D11	現在実行トランザクション数	個	ログ採取時の実行中であったJ2EEトランザクション数 (注1)
D12	最小同時実行トランザクション数	個	測定時間内に同時に実行された最小J2EEトランザクション数 (注1)
D13	最大同時実行トランザクション数	個	測定時間内に同時に実行された最大J2EEトランザクション数 (注1)

注1) コンテナで制御されるトランザクションの情報も合わせて表示されます。

4)Servletコンテナ情報

項番	項目名	単位	内容
D1	データ採取開始日時	－	当該レコードの性能情報の測定を開始した日時
D2	データ採取終了日時	－	当該レコードの性能情報の測定を終了した日時
D3	プロセス通番	－	IJServerが起動したプロセスの通番
D4	プロセスID	－	測定対象のIJServerのプロセスID
D5	現在使用スレッド数		ログ採取時に使用中であったスレッド数
D6	最小同時使用スレッド数	個	測定時間内に同時に使用した最小スレッド数 (注1)
D7	最大同時使用スレッド数	個	測定時間内に同時に使用した最大スレッド数 (注2)
D8	現在トータルスレッドプール数	個	ログ採取時にプールされていたスレッド数 (注3)
D9	最小トータルスレッドプール数	個	測定時間内にプールされていたスレッドの最小数 (注3)
D10	最大トータルスレッドプール数	個	測定時間内にプールされていたスレッドの最大数 (注3)

注1) クライアントからのリクエスト受け付け用にスレッドは1個使用されるため、クライアントからのリクエストがまったくない場合でも最小同時使用スレッド数は1が出力されます。
注2) クライアントからのリクエストを受付けるServletコンテナのServerSocketは数秒おきに待ち状態から復帰し、再度クライアントからのリクエストの待ち状態になります。そのため、クライアントからのリクエストがまったくない場合でも最大同時使用スレッド数は2が出力されます。
注3) クライアントからのリクエストが一定時間ない場合は、Servletコンテナの待機中の最大値に設定された値にしたがって余分な処理スレッドが破棄されます。このとき、クライアントからのリクエストを受け付けるスレッドは破棄対象となりませんので、待機中の最大値＋1 が最小値となります。

5)EJBコンテナ情報

項番	項目名	単位	内容
D1	データ採取開始日時	－	当該レコードの性能情報の測定を開始した日時
D2	データ採取終了日時	－	当該レコードの性能情報の測定を終了した日時
D3	プロセス通番	－	IJServerが起動したプロセスの通番
D4	プロセスID	－	測定対象のIJServerのプロセスID
D5	Message-driven Bean 最大同時処理数	個	Message-driven Beanでプーリングできるスレッド数の最大値
D6	Message-driven Bean 現在スレッドプールサイズ	個	ログ採取時にMessage-driven Beanでプーリングしていたスレッド数
D7	Message-driven Bean 最小スレッドプールサイズ	個	Message-driven Beanでプーリングした最小スレッド数
D8	Message-driven Bean 最大スレッドプールサイズ	個	Message-driven Beanでプーリングした最大スレッド数
D9	Message-driven Bean 現在アイドルスレッド数	個	ログ採取時にMessage-driven Beanでプーリングしたスレッドで未使用であったスレッド数
D10	Message-driven Bean 最小アイドルスレッド数	個	Message-driven Beanでプーリングしたスレッドで未使用であったスレッドの最小スレッド数
D11	Message-driven Bean 最大アイドルスレッド数	個	Message-driven Beanでプーリングしたスレッドで未使用であったスレッドの最大スレッド数
D12	Message-driven Bean アイドルタイムアウト発生回数	個	Message-driven Beanでプーリングしたスレッドをアイドルタイムアウトで破棄したスレッド数

■評価方法と対処方法

1)JavaVM情報

項番	評価方法	対応/処置
1	途中ガーベジコレクションが発生しているにもかかわらず、ヒープ使用量が増加傾向にある。	メモリリークの可能性があります。 不要になったオブジェクトが参照され続けていることがないかなど、サーバアプリケーションの見直しを実施してください。 また、Webアプリケーションのセションタイムアウト時間を極端に長く設定している場合、設定値が妥当か見直しを実施してください。
2	ヒープ最小使用量がヒープ上限値に近い状態が続いている。	運用を継続するとヒープ不足になる可能性があります。 指定しているヒープサイズの上限を増やすなど、IJServerの設定を見直してください。
3	メタスペース最小使用量がメタスペース上限値に近い値となっている。	運用を継続するとメタスペース不足になる可能性があります。 指定しているメタスペースサイズを増やす等、IJServerの設定を見直してください。

項番	評価方法	対応/処置
4	ガーベジコレクション発生回数が多い。	ガーベジコレクションが多く発生することでアプリケーションの処理性能が劣化している可能性があります。 JavaVMヒープの最小使用量が上限値と近い値となっている場合、ヒープ不足によりガーベジコレクションが多発している可能性があります。ヒープの上限を増やすなど、IJServerの設定を見直してください。
5	ガーベジコレクションの処理トータル時間が長いときがある。	IJServerを運用するサーバのCPU負荷が高い状態が続いている可能性があります。 不要なアプリケーションを停止するなどしてCPU負荷を軽減するか、IJServerを運用するサーバのCPU性能が適正か見直してください。 また、ヒープサイズを大きくすると、1回のガーベジコレクションで回収されるオブジェクトが増えるため、1回あたりの処理時間が長くなる傾向にあります。 ヒープの使用状況を確認し、余裕がある場合はヒープサイズの上限を小さくすることを検討してください。

※ JavaVMのチューニングについては、以下も参照してください。

― “27.2.2 JavaVMのヒープ領域サイズ”

― “27.2.3 ガーベジコレクション発生回数”

― “チューニングガイド”－“JDK/JRE 8のチューニング”－“チューニング方法”

2)データソース情報

項番	評価方法	対応/処置
1	物理コネクション最大数が物理コネクション上限数に近い値となっている。	運用を継続するとコネクション待ちとなる可能性があります。 指定している最大コネクション数の指定が適切か見直してください。
2	使用コネクション最大数が物理コネクション最小数よりかなり小さい値となっている。	プーリングされているコネクションと比較して、使用しているコネクションが少ないため、無駄なコネクションがプーリングされている可能性があります。 指定している事前コネクト数とアイドルコネクトタイムアウト値を見直してください。事前コネクト数を小さくすることで常時接続している物理コネクションを少なくすることできます。また、アイドルコネクトタイムアウトを小さくすることで無駄なコネクションを解放することが可能です。
3	累積コネクション待ち回数が多い。	コネクション待ちが発生することでアプリケーションの処理性能を劣化させる原因になっている可能性があります。 指定している最大コネクション数の指定が適切か見直してください。
4	平均コネクション待ち時間が長い。	コネクション待ちが発生することでアプリケーションの処理性能を劣化させる原因になっている可能性があります。 指定している最大コネクション数もしくはコネクション待ち時間の指定が適切か見直してください。
5	コネクション待ち最小時間が短く、コネクション待ち最大時間が長い。	時間帯によりコネクション待ち時間にばらつきがあるため、アプリケーションの処理性能のばらつきが発生している可能性があります。

項番	評価方法	対応/処置
		コネクション待ち時間の設定を短くして待ち時間が長時間に及ぶ場合にはエラーを返却するなどの対策を検討してください。
6	コネクション待ち最大スレッド数が増加している。	アプリケーションで同時に使用するコネクションが多くなっているため、運用を継続するとアプリケーションの処理性能を劣化させる可能性があります。指定している最大コネクション数の指定が適切か見直してください。
7	コネクション待ちタイムアウト回数が増加している。	コネクション待ちタイムアウトが発生してエラーとなるアプリケーションが増加しています。 指定している最大コネクション数もしくはコネクション待ち時間の指定が適切か見直してください。
8	物理コネクション確立回数が多い。アイドルタイムアウトによるクローズ回数が多い。	アイドルタイムアウトにより物理コネクション数が解放されるために物理コネクト回数が多くなり、アプリケーションの処理性能を劣化させる原因になっている可能性があります。 アイドルタイムアウト値が適切か見直してください。また、事前コネクト数を指定することで物理コネクションを常時確立することもできます。
9	物理コネクション確立回数が多い。例外発生によるクローズ回数が多い。	アイドルタイムアウトにより物理コネクション数が解放されるために物理コネクト回数が多くなり、アプリケーションの処理性能を劣化させる原因になっている可能性があります。 アプリケーションを見直して例外発生原因を取り除いてください。
10	コネクション獲得回数とコネクションクローズ回数の差が大きい。	コネクションのクローズ漏れが発生している可能性があります。 アプリケーションを見直してください。
11	コネクション平均使用時間が長い。	データベースアクセス処理によりアプリケーションの処理性能を劣化させる原因になっている可能性があります。 アプリケーションもしくはデータベースの設定を見直してください。
12	コネクション最大使用時間とコネクション最小使用時間の差が大きい。	データベースアクセス処理の何かしらの理由により、アプリケーションの処理性能のばらつきが発生している可能性があります。 アプリケーションもしくはデータベースの設定を見直してください。
13	通信待ちタイムアウトの発生回数が多い。	データベースアクセス処理によりアプリケーション処理性能が劣化している可能性があります。 データベースの処理時間、ハングの有無をチェックしてください。

※ JDBCデータソースのチューニングについては、以下も参照してください。

－ “27.2.5 JDBCのコネクション”

－ “27.2.6 Statementキャッシュ機能”

3)トランザクション情報

項番	評価方法	対応/処置
1	実行トランザクション総数と最大同時実行トランザクション数が増加し、トランザクション平均時間も増加している。	同時に実行されるトランザクションが増加したことにより、データベースの排他制御などでアプリケーションの処理性能が劣化している可能性があります。 IJServerに指定した同時処理数を小さくして処理性能を安定させるか、IJServerサーバを運用するサーバの処理性能を見直してください。
2	実行トランザクション総数は変わらないが、最大同時実行トランザクション数とトランザクション最大時間が大きい。	一定時間帯にトランザクション処理が集中したことにより、データベースの排他制御などでアプリケーションの処理遅延が発生した可能性があります。 IJServerに指定した同時処理数を小さくして処理性能を安定させるか、IJServerサーバを運用するサーバの処理性能を見直してください。
3	コネクション最大使用時間とコネクション最小使用時間の差が大きい。	データベースアクセス処理の何かしらの理由により、アプリケーションの処理性能のばらつきが発生している可能性があります。 アプリケーションもしくはデータベースの設定を見直してください。 もしくはIJServerを運用するサーバのCPU負荷が高い状態が一時的に発生している可能性があります。不要なアプリケーションを停止するなどしてCPU負荷を軽減するか、IJServerを運用するサーバのCPU性能が適正か見直してください。
4	ロールバックトランザクション数に比べてコミットトランザクション数が多い。	コミット処理はロールバック処理に比べて一般的に負荷は高いため、アプリケーションの処理性能が劣化している可能性があります。 無駄にコミット処理を実行していないかアプリケーションを見直してください。

※ JDBCデータソースのチューニングについては、以下も参照してください。

― “27.2.4 トランザクションアイソレーションレベル”

4)Servletコンテナ情報

項番	評価方法	対応/処置
1	アクティブ数が、長時間トータルに近い値を示している。	[CPUに余裕がある場合] ・ Servletコンテナの同時処理数が小さすぎる可能性があります。 Servletコンテナの同時処理数を増やしてください。ただし、JavaVMのヒープ使用量に余裕がない場合にそのまま同時処理数を増やすとOutOfMemoryErrorが発生する可能性があります。この場合は、JavaVMのヒープ使用量の最大値を増やすなどの対処を同時に行ってください。 ・ プロセス内の要因によってスレッド間の競合が発生している可能性があります。 ワークユニットのプロセス多重度を増やしてください。
		[CPUに余裕がない場合] ・ サーバの処理能力を超えている可能性があります。 サーバの増設または、より性能の良いサーバへのリプレースを検討してください。

※ Servletコンテナのチューニングについては、以下も参照してください。

― “27.3 Servletコンテナのチューニング”

5)EJBコンテナ情報

項番	評価方法	対応/処置
1	Message-driven Bean最大同時処理数とMessage-driven Bean最大スレッドプールサイズが近い値となっている。	運用を継続するとスレッド獲得待ちとなる可能性があります。 指定しているMessage-driven Beanの同時処理数が適切か見直してください。
2	Message-driven Bean最小スレッドプールサイズとMessage-driven Bean最小スレッドプールサイズの差が大きく、Message-driven Beanアイドルタイムアウト発生回数が多い。	Message-driven Beanのスレッド生成処理でアプリケーションの処理性能が劣化している可能性があります。 指定しているMessage-driven Beanの最小同時処理数が適切か見直してください。
3	Message-driven Bean最大アイドルスレッド数が大きい。	指定しているMessage-driven Beanの最小同時処理数が大きいため、メモリ資源を無駄に消費している可能性があります。 指定しているMessage-driven Beanの最小同時処理数が適切か見直してください。

※ EJBコンテナのチューニングについては、以下も参照してください。

― “27.4 EJBコンテナのチューニング”

27.2 IJServerのチューニング

IJServerをチューニングする時に考慮するポイントは以下です。ここに記述されたチューニングは、ServletコンテナとEJBコンテナの両方に有効です。

- プロセス多重度
- JavaVMのヒープ領域サイズ
- ガーベジコレクション発生回数
- トランザクションアイソレーションレベル
- JDBCのコネクション
- Statementキャッシュ機能
- モニタリング情報
- IPCOM連携時の注意事項

27.2.1 プロセス多重度

1つのIJServerを、複数のプロセスで起動できます。これにより、負荷を分散できます。
IJServerのプロセス多重度は、Interstage管理コンソールのワークユニット設定またはisj2eeadminコマンドで指定できます。詳細については、Interstage管理コンソールのヘルプを参照してください。isj2eeadminコマンドについては、“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

27.2.2 JavaVMのヒープ領域サイズ

Interstage管理コンソールまたはisj2eeadminコマンドを使用して、ワークユニット設定のJavaVMオプションを指定することで、IJServerが動作するJavaVMのパラメタを変更して動作させることができます。
パラメタを変更して、JavaVMヒープ領域サイズなどを変更できます。

JDK 8の場合における最大ヒープ領域サイズの例を次に示します。
最大ヒープ領域のサイズの省略値は、JavaVMによって異なりますので、JDKのドキュメントを参照してください。
java.lang.OutOfMemoryErrorが多発する場合には、本定義項目で、JavaVMの最大ヒープ領域を増加させてください。

例

Windows32 Linux32

JavaVMの最大ヒープ領域を512メガバイトとする場合の設定

```
-Xmx512m
```

例

Windows64 Solaris64 Linux64

JavaVMの最大ヒープ領域を1024メガバイトとする場合の設定

```
-Xmx1024m
```

なお、Interstageではヒープ領域の問題を警告メッセージで通知する、予兆監視機能を提供しています。
警告メッセージが出力された場合、そのまま業務を継続すると、メモリ不足やレスポンス低下などの問題が発生する可能性があります。これらの問題を解決するために、警告メッセージに記載されている不足リソースの情報を元に、チューニングを実施してください。
JavaVMで問題となる異常の原因は、ヒープ領域またはメタスペースの不足です。これを回避するために、現在の上限値を20%増加させて運用を再開します。それでも警告が出力される場合は、上限値を更に20%増加させて、警告が出力されなくなるまで繰り返しチューニングを実施してください。チューニングを繰り返し行い、警告メッセージが出力されない状態にすることで、安定稼動するシステムを構築することができます。
予兆監視機能については、“3.3.11 予兆監視”を参照してください。

27.2.3 ガーベジコレクション発生回数

IJServerでは、JavaのRMI機能による自動ガーベジコレクションが1時間隔(デフォルトの場合)で動作します。
RMI機能による自動ガーベジコレクションの発生間隔は、Interstage管理コンソールの“ワークユニット名” > [環境設定]タブ > [ワークユニット設定]のJavaVMオプションに、以下を設定します。

“-Dsun.rmi.dgc.client.gcInterval=発生間隔”および
“-Dsun.rmi.dgc.server.gcInterval=発生間隔”

発生間隔: ミリ秒単位で数値を指定します。デフォルト値は、3600000です。

isj2eeadminコマンドを使用して、設定することもできます。isj2eeadminコマンドについては、“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

なお、RMI機能による自動ガーベジコレクションの発生間隔をチューニングしても、ガーベジコレクション発生回数が削減されない場合、JavaVMのヒープ領域サイズが不足している可能性がありますので、JavaVMのヒープ領域サイズのチューニングを行うことで削減される場合があります。“27.2.2 JavaVMのヒープ領域サイズ”を参照してください。

27.2.4 トランザクションアイソレーションレベル

EJBアプリケーションからデータベースにアクセスする場合、EJBアプリケーションの実行多重度を上げるには、トランザクションアイソレーションレベル(以降、アイソレーションレベルと呼びます)を考慮する必要があります。アイソレーションレベルとは、データベースに対する排他整合性水準のことです。

使用できるアイソレーションレベルを以下に示します。アイソレーションレベルの詳細は、使用するデータベースのマニュアルを参照してください。

- Transaction-read-committed
- Transaction-read-uncommitted
- Transaction-repeatable-read
- Transaction-serializable

アイソレーションレベルの設定は、UserTransaction.begin()メソッドを発行してから、UserTransaction.commit()メソッドまたはUserTransaction.rollback()メソッドを発行するまでの間有効です。

■設定方法

アイソレーションレベルは、Interstage管理コンソールまたはisj2eeadminコマンドで設定します。設定方法の詳細については、Interstage管理コンソールのヘルプを参照してください。isj2eeadminコマンドについては、“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

注意

DBMSにOracleを使用している場合

「ORA-8177:このトランザクションのアクセスを逐次化できません。」というエラーは、トランザクションアイソレーションレベルにTransaction-serializableが設定されているにもかかわらず、複数のユーザが同時に同一の表を更新した場合など、トランザクションのシリアル化を保障できない場合に出力され、ユーザにその旨を伝えています。
トランザクションアイソレーションレベルにTransaction-serializableを設定して、エラー「ORA-8177」が発生した場合は、アプリケーション側で単に「異常終了」と判断するのではなく、トランザクションのロールバック後に「リトライ」させるなどの対処が必要になります。
なお、トランザクションアイソレーションレベルがTransaction-read-committed(Oracleのデフォルト)の場合は、「ORA-8177」エラーが発生することはありません。特にTransaction-serializableの設定が必須ではない場合、Transaction-read-committedを設定することによって、同時実行性が向上し、「ORA-8177」エラーも発生しなくなります。

27.2.5 JDBCのコネクション

ここでは、JDBCのコネクションの以下について説明します。

- コネクションプーリングの種別
- コネクションプーリングのチューニングパラメタ

InterstageのJNDIサービスプロバイダから取得したJDBCデータソースを使用した場合、JDBCのコネクションはプーリングされて再利用されます。

■コネクションプーリングの種別

コネクションプーリングには、以下の2種類があります。

- Interstageでコネクションプーリング
  Interstageでコネクションのプーリング制御を行うため、Interstage管理コンソールでコネクションプーリングの詳細設定ができます。プーリングされている情報を、Interstage管理コンソールのモニタ機能で参照できます。
  Windows32/64 Linux32/64
  Oracleの分散トランザクションを使用する場合もInterstageでコネクションプーリングを行います。

- JDBCドライバでコネクションプーリング
  JDBCドライバでコネクションのプーリング制御を行うため、使用する各JDBCドライバの機能を使用して、コネクションプーリングの設定を行います。設定方法の詳細は、JDBCドライバのマニュアルを参照してください。
  JDBCドライバ側でプーリング制御を行うため、Interstage管理コンソールで参照可能なJDBCデータソースのモニタ情報は、Oracleの場合は「コネクションプール情報」と「アプリケーションからのコネクション確立情報」、Oracle以外のデータベースは「アプリケーションからのコネクション確立情報」のみです。詳細はInterstage管理コンソールのヘルプを参照してください。

コネクションプーリングの機能概要については、“4.4 JDBC(データベース)のコネクション”を参照してください。

以下に、各データベースとコネクションプーリングの対応について記載します。
サポートしているデータベースは、使用しているプラットフォームによって異なります。“システム設計ガイド”の“アプリケーション実行時に必要なソフトウェア”を参照してください。

DB種別	コネクションプーリング
Oracle	データソースの種類に“Interstageのコネクションプーリングを使用する”を選択している場合、Interstageでコネクションプーリングを行います。 “Oracleのコネクションプーリングを使用する”を選択している場合、JDBCドライバがコネクションプーリングを行います。 Oracleのプーリングは暗黙的接続キャッシュを使用します。
Symfoware	データソースの種類に“Interstageのコネクションプーリングを使用する”を選択している場合、Interstageでコネクションプーリングを行います。 “Symfowareのコネクションプーリングを使用する”を選択している場合、JDBCドライバでコネクションプーリングを行います。
SQL Server	データソースの種類にかかわらずInterstageでコネクションプーリングを行います。
Windows32/64 Linux32/64 PostgreSQL	データソースの種類に“Interstageでコネクションプーリングを行う”を選択している場合に、Interstageでコネクションプーリングを行います。 “PostgreSQLでコネクションプーリングを行う”の場合は、JDBCドライバがコネクションプーリングを行います。
汎用定義	データソースクラスがjava.sql.ConnectionPoolDataSourceインタフェースを実装している場合、Interstageでコネクションプーリングを行います。 上記以外の場合、JDBCドライバがコネクションプーリングを行うかどうかはJDBCドライバの実装に依存します。

■コネクションプーリングのチューニングパラメタ

IJServerの環境設定に指定可能なDBコネクション設定はデータベースタイプおよびデータソース種別によって設定が有効となる項目が異なります。以下の表を参照してください。

	Oracle			Symfoware		SQL Server	PostgreSQL		汎用定義	
	Interstage (*1)	JDBC (*2)	XA (*3)	Interstage (*1)	JDBC (*2)	Interstage (*1)	Interstage (*1)	JDBC (*2)	Interstage (*4)	JDBC (*5)
トランザクションアイソレーションレベル	○	○	○	○	○	○	○	○	○	○
事前コネクト数	○	○	○	○	○	○	○	○	○	○
最大コネクション数	○	○	○	○	×	○	○	×	○	×

	Oracle			Symfoware		SQL Server	PostgreSQL		汎用定義	
	Interstage (*1)	JDBC (*2)	XA (*3)	Interstage (*1)	JDBC (*2)	Interstage (*1)	Interstage (*1)	JDBC (*2)	Interstage (*4)	JDBC (*5)
コネクションタイムアウト	○	○	○	○	×	○	○	×	○	×
アイドルタイムアウト	○	○	○	○	×	○	○	×	○	×
コネクション使用監視時間	○	○	○	○	○	○	○	○	○	○
Statementキャッシュサイズ	×	○	×	○	×	×	×	×	×	×
Statement自動クローズ	×	×	×	○	×	×	×	×	×	×
通信待ち時間	○	○	○	○	○	○	○	○	○	○
異常時の再接続	○	×	×	○	×	○	○	×	×	×
インターバル時間	○	×	×	○	×	○	○	×	×	×
リトライ回数	○	×	×	○	×	○	○	×	×	×

○:有効　×:無効

*1) Interstageのコネクションプーリングを使用する場合
*2) JDBCのコネクションプーリングを使用する場合
*3) 分散トランザクションを使用する場合 Windows32/64 Linux32/64
*4) データソースクラスがjava.sql.ConnectionPoolDataSourceインタフェースを実装している場合
*5) データソースクラスがjava.sql.ConnectionPoolDataSourceインタフェースを実装していない場合（JDBCでコネクションプーリングを行うかはJDBCドライバの実装に依存します。）

以下に、Interstage管理コンソールまたはisj2eeadminコマンドを使用して設定可能なパラメタを示します。
パラメタは、データソースごとにIJServerの環境設定で設定します。

パラメタ	説明	設定値
事前コネクト数 (注1)(注5)	運用で必要なコネクションを、起動時にあらかじめ取得することにより、初回疎通時から2回目以降と同等の処理速度が得られます。 データベースタイプがOracle、データソースの種類が「Oracleのコネクションプーリングを使用する」で、暗黙的接続キャッシュの接続キャッシュ・プロパティにInitialLimitを指定し、かつ事前コネクト数が1以上の場合、事前コネクト数とInitialLimitの値が大きい方が有効となります。(注4)	最大値: 2147483647 最小値:0 初期設定値:0
最大コネクション数	最大コネクション数を抑止することにより、メモリ資源を抑止することが可能です。 最大コネクション数すべてをJ2EEアプリケーションが使用している状態で接続リクエストがあった場合、コンテナは“コネクションタイムアウト”における設定時間の期間内で、プールにコネクションが返却されるのを待ちます。プールにコネクションが返却された場合にはそのコネクションを使用し、返却されなかった場合にはSQLExceptionを返却します。	最大値: 2147483647 最小値:1 初期設定値:64

パラメタ	説明	設定値
	データベースタイプがOracle、データソースの種類が「Oracleのコネクションプーリングを使用する」の場合、以下の注意があります。 ・ 本項目は暗黙的接続キャッシュの接続キャッシュ・プロパティ MaxLimitの設定となります。 ・ JDBCリソース定義時にキャッシュ・プロパティMaxLimitを指定しても値は有効になりません。本項目で指定してください。	
コネクションタイムアウト	最大コネクション数分のコネクションすべてがJ2EEアプリケーションで使用中の状態で、コネクションの接続要求が来た場合に、プールにコネクションが返却されるのを待つ時間を指定します。 時間が超過してもコネクションが返却されなかった場合は、SQLExceptionが返却されます。0を指定した場合、コネクション待ち状態になるとすぐにSQLExceptionが返却されます。 データベースタイプがOracle、データソースの種類が「Oracleのコネクションプーリングを使用する」の場合、以下の注意があります。 ・ 本項目は暗黙的接続キャッシュの接続キャッシュ・プロパティ ConnectionWaitTimeout の設定となります。 ・ JDBCリソース定義時にキャッシュ・プロパティConnectionWaitTimeoutを指定しても値は有効になりません。本項目で指定してください。	最大値: 2147483647 最小値:0 初期設定値:5 (単位:秒)
アイドルタイムアウト	使用されていないコネクションをタイムアウトで破棄することにより、無駄なメモリ資源を解放することができます。 ただし、事前コネクトで接続されたコネクションはアイドルタイムアウトの対象となりません。 データベースタイプがOracle、データソースの種類が「Oracleのコネクションプーリングを使用する」の場合、以下の注意が必要です。 ・ 本項目は、暗黙的接続キャッシュの接続キャッシュ・プロパティ InactivityTimeout の設定となります。 ・ JDBCリソース定義時にキャッシュ・プロパティInactivityTimeout を指定しても値は有効になりません。本項目で指定してください。 ・ タイムアウトを確認する時間間隔は、暗黙的接続キャッシュの接続キャッシュ・プロパティPropertyCheckInterval の設定値となります。データソースの環境設定に PropertyCheckInterval を定義してください。プロパティの詳細はOracleのマニュアルを参照してください。 ・ Oracleのコネクションプーリングを使用する場合、アイドルタイムアウト発生時のコネクション破棄はOracleのJDBCドライバで行われます。また事前コネクトで接続されたコネクションもアイドルタイムアウトの対象となります。	最大値: 2147483647 最小値:0 初期設定値:600 (単位:秒)
コネクション使用監視時間 (注2)	トランザクション開始前にアプリケーションが獲得したコネクションに対してクローズするまでの使用時間を監視します。 コネクションの使用時間が指定した時間を超過してもクローズされない場合、警告メッセージがコンテナログに出力されます。	最大値: 2147483647 最小値:0 初期設定値:60 (単位:分)

パラメタ	説明	設定値
	0を指定した場合、コネクション使用時間の監視は行いません。	
タイムアウトしたコネクションはクローズする	コネクション使用監視時間使用時(コネクション使用監視時間が1以上)にタイムアウトしたコネクションを自動回収する場合はONにします。OFFにした場合、タイムアウトしたコネクションは自動回収されません。	・ する ・ しない(デフォルト)
Statementキャッシュサイズ	Statementのキャッシュサイズを指定します。 アプリケーションが実行したStatementをクローズせずにキャッシュするサイズを指定します。 0を指定した場合、Statementのキャッシュは行いません。 Statementキャッシュ機能の詳細は“27.2.6 Statementキャッシュ機能”を参照してください。	最大値:32000 最小値:0 初期設定値:10
Statement自動クローズ	ステートメントキャッシュ機能の使用時(Statementキャッシュサイズが1以上の場合)に、ステートメントのクローズをJDBCドライバが自動的に行うかどうかを指定します。	・ する ・ しない(デフォルト)
通信待ち時間(注2)	以下のSQL文実行の開始から終了までの時間を監視します。 ・ java.sql.Statement — execute(String) — execute(String, int) — execute(String, int[]) — execute(String, String[]) — executeBatch() — executeQuery(String) — executeUpdate(String) — executeUpdate(String, int) — executeUpdate(String, int[]) — executeUpdate(String, String[]) ・ java.sql.PreparedStatement — execute() — executeQuery() — executeUpdate() SQL文の実行時間が一定時間超過しても復帰しなかった場合、警告メッセージがコンテナログに出力されます。 0を指定した場合、通信待ち時間の監視は行いません。	最大値:2147483647 最小値:0 初期設定値:400 (単位:秒)
ログにSQL文を出力する	通信待ち時間使用時(通信待ち時間が1以上)に、ログに出力する警告メッセージにSQL文を出力する場合はONにします。OFFにした場合、SQL文は出力されません。	・ ON ・ OFF(デフォルト)
異常時の再接続	JDBCコネクションの自動再接続機能(注3)を使用するかどうかを指定します。 自動再接続機能を使用する場合、プーリングされているJDBCのコネクションが使用可能なコネクションであるかを判定し、使用できないコネクションの場合には自動的にDBMSに再接続します。	・ する(デフォルト) ・ しない

パラメタ	説明	設定値
インターバル時間	JDBCコネクションの自動再接続機能(注3)において、プーリングされているJDBCのコネクションが使用できない場合、またはDBMSへの接続に失敗した場合、再度接続を行うまでのインターバル時間を指定します。 異常時の再接続を[する]にした場合のみ、指定した値が有効になります。	最大値: 2147483647 最小値:1 初期設定値:10 (単位:秒)
リトライ回数	JDBCコネクションの自動再接続機能(注3)において、プーリングされているJDBCのコネクションが使用できない場合、またはDBMSへの接続に失敗した場合、再度接続を試みる回数を指定します。 異常時の再接続を[する]にした場合のみ、指定した値が有効になります。	最大値: 2147483647 最小値:1 初期設定値:10 (単位:回数)

注1)

CMP2.0のEJBアプリケーションを配備したIJServerを起動する場合、起動時にDBMSの識別子長の最大値をチェックします。このため、事前コネクト機能を使用しない場合にも1コネクションだけDBMSへ接続します。
Interstageでコネクションをプーリングする場合には、起動処理完了後にコネクションが切断されます。

注2)

コネクション使用監視時間、および通信待ち時間のタイムアウト値は以下の計算式を参考に設定してください。

アプリケーション最大処理時間 ＞ コネクション使用監視時間 ＞ 通信待ち時間

注3)

Symfowareの場合は、Connection Managerの機能を使用することで、データベースサーバのダウンおよび通信回線の異常発生時にも同等の運用を行うことが可能です。Connection Managerの詳細については、Symfoware Serverのマニュアル“Connection Managerユーザーズガイド”を参照してください。

注4)

Oracleの場合、事前コネクトの獲得タイミングは以下の表のようになります。

Oracleのデータソース種別および設定		事前コネクト取得タイミング
Interstageでコネクションプーリング	DBコネクション設定の“事前コネクト数”に1以上の値を設定	IJServer起動時
	上記設定をしない場合	事前にコネクションは確立されません
Oracleでコネクションプーリング	DBコネクション設定の“事前コネクト数”に1以上の値を設定 ただし暗黙的接続キャッシュのプロパティに“InitialLimit”も設定されている場合は双方の値の大きい方の値分確立されます	IJServer起動時
	DBコネクション設定の“事前コネクト数”を設定せず(デフォルト値:0)、暗黙的接続キャッシュのプロパティの“InitialLimit”に1以上の値を設定	初回getConnection時
	上記以外の場合	事前にコネクションは確立されません

注5)

事前コネクト数を1以上に設定した場合、IJServer起動時にDBMSへ接続しますので対象のDBMSは起動されている必要があります。

27.2.6 Statementキャッシュ機能

以下の場合に、Statementキャッシュ機能を使用できます。

- データベースタイプが“Oracle”で、データソースの種類が“Oracleのコネクションプーリングを使用する”の場合
  この場合、OracleのStatementキャッシュ機能が使用可能になります。
  Windows32/64 Linux32/64
  “Oracleのコネクションプーリングを使用する”データソースでは、分散トランザクションが使用できないため、本機能は分散トランザクション環境では使用できません。
- データベースタイプが“Symfoware”で、データソースの種類が“Interstageのコネクションプーリングを使用する”の場合

Statementをキャッシュすることによって、SQL文の解析、作成によるオーバーヘッドの軽減や、データベースとの通信回数を削減する効果があります。

キャッシュ対象となるStatementは以下です。

- Connection.prepareStatement(String)メソッドで取得したPreparedStatementオブジェクト
- Connection.prepareStatement(String, int, int)メソッドで取得したPreparedStatementオブジェクト
- Connection.prepareStatement(String, int)メソッドで取得したPreparedStatementオブジェクト
- Connection.prepareStatement(String, int[])メソッドで取得したPreparedStatementオブジェクト
- Connection.prepareStatement(String, int, int, int)メソッドで取得したPreparedStatementオブジェクト
- Connection.prepareStatement(String, String[])メソッドで取得したPreparedStatementオブジェクト
- Connection.prepareCall(String)メソッドで取得したCallableStatementオブジェクト
- Connection.prepareCall(String, int, int)メソッドで取得したCallableStatementオブジェクト
- Connection.prepareCall(String, int, int, int)メソッドで取得したCallableStatementオブジェクト

■チューニング方法

Statementキャッシュサイズ

Statementキャッシュサイズは、使用するデータソースに対して、Interstage管理コンソールまたはisj2eeadminコマンドで設定します。
Interstage管理コンソールでは、[IJServer] > [環境設定] > [DBコネクション設定] > [Statementキャッシュサイズ]で設定します。
[Statementキャッシュサイズ]に0を設定した場合は、Statementキャッシュは行なわれません。
isj2eeadminコマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。
なお、JDBCリソース定義画面の接続オプションでStatementキャッシュサイズを設定し、同時にDBコネクション設定でキャッシュサイズを指定した場合は、DBコネクション設定で行ったキャッシュサイズの値が有効になります。

設定の指針

Statementはコネクション単位でキャッシュされ、コネクションは実行する同一IJServerプロセス内のアプリケーション全体で共通に使用されます。
アプリケーションで発行されるStatement数がStatementキャッシュサイズより大きくなるとキャッシュから削除されるStatementが増加します。アプリケーションが要求するStatementがキャッシュから削除されていると、Statementの再作成のために、データベースとの通信、SQLステートメント文の解析によるオーバーヘッドが発生します。
キャッシュからのStatement削除回数を削減するため、StatementキャッシュサイズをIJServerプロセスで発行されるStatementの総数もしくはそれに近い値を設定することを推奨します。ただし、Statementのキャッシングはマシン資源を消費します。マシンスペックを考慮の上でStatementのキャッシュサイズの設定を行う必要があります。

また、アプリケーションサーバ(コンテナ)側でStatementを発行するCMPアプリケーションの場合は、次の数の合計値をアプリケーションで発行するStatementの代わりにキャッシュサイズへ追加します。

・ CMP1.1 Entity Beanの場合

　ー Bean単位で算出するStatement

　　- 4つ(挿入、削除、更新、検索(プライマリキー))

　　- 1件検索、複数件検索メソッド数

・ CMP2.0 Entity Beanの場合

　ー Bean単位で算出するStatement

　　- 4つ(挿入、削除、更新、検索(プライマリキー))

　　- deployment descriptorに定義したEJB QLクエリ数

　ー relationship単位で算出するStatement

　　- CMP2.0 Entity Bean間の単方向、かつ、1:1のrelationship数

　　- CMP2.0 Entity Bean間の双方向、かつ、1:1のrelationship数×2

　　- CMP2.0 Entity Bean間の単方向、かつ、1:多、多:多のrelationship数×3

　　- CMP2.0 Entity Bean間の双方向、かつ、1:多、多:多のrelationship数×4

OPEN_CURSORSの設定(Oracleを使用する場合)

Statementキャッシュ機能を使用する場合、コネクションのクローズ時にStatementは破棄されず、1コネクションで同時に発行しているStatementは増加します。このため、“1コネクション(トランザクション)で同時に発行可能なStatement数の上限値(以降OPEN_CURSORS/デフォルト:50)”を超える可能性があります。Statement数がOPEN_CURSORSを超えた場合、SQLExceptionが発生します。
このような場合は、OPEN_CURSORSが、Statementキャッシュサイズ以上となるように設定してください。OPEN_CURSORSの設定方法は、Oracleのマニュアルを参照して下さい。

■キャッシュされたStatementの削除契機

キャッシュされたStatementは、以下の契機で削除されます。また、実行したSQL文の数が、「Statementキャッシュサイズ」に設定した値に達した場合、それ以降実行するSQL文の扱いについては使用するJDBCドライバの仕様に依存します。(JDBCドライバの仕様によりキャッシュされたStatementが削除される場合があります。)JDBCドライバの仕様については、各JDBCドライバのマニュアルを参照してください。

・ JDBCプーリングコネクションの時間監視機能
　Statementは接続インスタンス(コネクション)ごとにキャッシングされるため、物理接続が接続キャッシュ内にアイドル状態で存続できる最大期間を超過したコネクションオブジェクトを開放する時に、そのコネクションでキャッシュされていたStatementオブジェクトも削除されます。

・ コネクションの物理的開放
　Statementは接続インスタンス(コネクション)ごとにキャッシングされるため、コネクションがプーリングされている時にデータベースダウンなどによりコネクションが無効となった場合、そのコネクションを開放する際にキャッシュされていたStatementオブジェクトも削除されます。

27.2.7 モニタリング情報

Interstage管理コンソールでは、運用中のIJServerの稼動情報を表示します。出力される情報により、性能のボトルネックの検出や、性能チューニングの効果の確認ができます。
以下の情報が出力されます。出力される情報の詳細は、Interstage管理コンソールのヘルプを参照してください。

・ JavaVM情報

・ Servletコンテナ情報

・ データソース情報

・ トランザクション情報

- キュー情報
- Message Driven Beanスレッド情報

なお、V9.0以降のIJServerを使用している場合は、性能情報を一定間隔でログファイルへ出力することができます。詳細は、“27.1 J2EEモニタロギング機能”を参照してください。

27.2.8 IPCOM連携時の注意事項

IPCOMを利用して、IJServerとWebサーバを分離して運用するシステムの負荷分散をする場合、故障監視用のコネクション(スレッド)が必要となります。
この場合、IJServerの同時処理数を設定するときは実際の同時処理数に監視用の数を考慮してください。設定は以下になります。

設定する同時処理数 ＝ 実際の同時処理数 ＋ 1(監視用)

なお、IPCOMのWebアクセラレーション機能を使用する場合は、実際の同時処理数を設定してください(監視用の数は加算しないでください)。Webアクセラレーション機能については、IPCOMのマニュアルを参照してください。

27.3 Servletコンテナのチューニング

Servletコンテナをチューニングする時に考慮するポイントは以下です。

- 同時処理数
- 接続数
- タイムアウト

■同時処理数

同時処理数およびプロセス多重度を増やすと、Webアプリケーションの同時実行多重度を増やせます。
同時処理数を増やすと、1プロセスあたりの実行多重度を増やせますが、同時処理数が増えることによる負荷や資源の増加により効果はみられない可能性があります。通常はデフォルト値以下で運用することを推奨しています。
アプリケーションから呼び出すEJBの同時処理数や、JDBCのコネクション数にあわせてチューニングしてください。
同時処理数は、Interstage管理コンソールのServletコンテナ設定で指定します。また、isj2eeadminコマンドを使用して設定することもできます。

同時処理数については、以下が設定できます。

- 初期値(増分値)
- 待機中(アイドル状態)の処理スレッドの最大値
- 最大値

処理スレッドの処理完了後、待機中となり使用されていない処理スレッドは、1分間隔の監視によって、待機中の最大値をこえている分が解放されます。
これにより、一時的に負荷が高くなった場合でも、負荷が下がればサーバ資源を解放し節約することができます。

■接続数

接続数に関する設定項目としては以下があります。

- WebサーバコネクタのServletコンテナへの最大接続数

  Interstage管理コンソールで指定します。

  ー Webサーバとワークユニットを同一マシンで運用する場合
  ワークユニットの環境設定のWebサーバコネクタ(コネクタ)設定

ー Webサーバとワークユニットを同一マシンで運用しない場合
Webサーバコネクタの環境設定

また、isj2eeadminコマンドを使用して設定することもできます。

- Servletコンテナの接続数(最大接続数)

  Interstage管理コンソールのServletコンテナ設定で指定します。
  また、isj2eeadminコマンドを使用して設定することもできます。

上記項目の値を増やすとServletコンテナが受付けるクライアントからのリクエストの数を増やせます。
運用時に一時的に負荷が高くなりServletコンテナの同時処理数を超えるリクエストを受付けることが想定される場合は、Servletコンテナの同時処理数を抑えてServletコンテナの接続数(最大接続数)に大きな値を設定することで、サーバ全体のレスポンスの悪化を防ぐことができます。

接続数は、運用するシステムの要求の違いによって以下のように設定します。

- Servletコンテナの負荷が高く処理できないリクエストを時間がかかってもかまわないので、正常に処理させたい場合

項目	設定値
Servletコンテナの最大接続数	リクエストを処理できるだけの十分に大きな値を設定
WebサーバコネクタのServletコンテナへの接続数の制限	なし(無制限)

- Servletコンテナの負荷が高く処理できないリクエストは長時間待たせるのではなく、エラーとしてクライアントに通知したい場合

項目	設定値
Servletコンテナの最大接続数	リクエストを処理できるだけの十分に大きな値を設定
WebサーバコネクタのServletコンテナへの接続数の制限	Servletコンテナが処理可能な数を設定

注意

- Servletコンテナへの接続数がServletコンテナの同時処理数の最大値を超えた場合、超えた分の接続はKeepAliveされませんので、Servletコンテナの同時処理数の最大値を超えない場合と比較して多くのSocketを消費します。
- Servletコンテナの接続数(最大接続数)に設定された値はServletコンテナが使用するSocketのbacklogに設定されます。
  Socketのbacklogに設定した値はOSによって有効となる範囲が決まっていますので、Servletコンテナの接続数(最大接続数)に設定した値がすべて有効となるわけではありません。
  Socketのbacklogの有効範囲については各OSのドキュメントを参照してください。
  Windows (R)の場合、200より大きな値を設定しても有効になりません。
- Servletセションをご利用でない環境における、Webサーバコネクタの振り分け処理に関する注意事項

  「Servletコンテナへの最大接続数」を設定している場合、Servletコンテナで処理中のリクエスト数が最大接続数以内であっても、WebサーバコネクタによりHTTP 503エラーが返却されることがあります。(注1)

  そのため、リクエスト多重度(クライアント数)が限定されている環境において厳密な運用条件が要求される場合、想定多重度に基づいた最大接続数設定を行う際には、1～3%程度のマージンを取った値とすることをお薦めします。

  実際のマージンはHTTP 503エラーの発生状況により、さらに増やすなどしてください。

  注1)高負荷状態など、何らかの理由によりWebサーバの特定通信プロセスに長時間(数十ms以上)CPUが割り当たらないなど、特殊な状況によります。

  このような状況においては、Webサーバコネクタの内部制御情報の更新が完了する前に次のリクエストが到着する可能性があるためです。

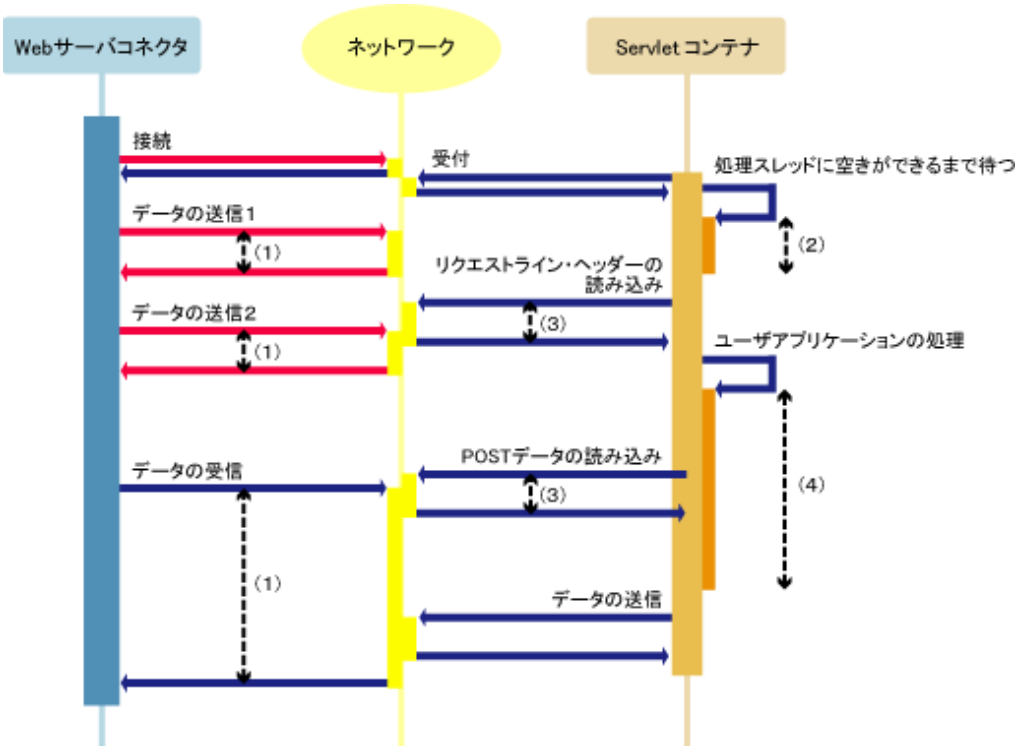
■タイムアウト

タイムアウトの監視項目

タイムアウトに関しては、以下が監視されています。

監視項目	意味
(1)	WebサーバコネクタがServletコンテナとの間でデータパケットを送受信するときに待機する時間
(2)	Servletコンテナに接続されてからリクエストの処理が開始されるまでの待ち時間(値を指定することはできません)
(3)	ServletコンテナがWebサーバコネクタからのデータパケットを受信するときに待機する時間
(4)	ユーザアプリケーションの処理時間

それぞれの項目が監視されるタイミングを下図に示します。



タイムアウトの設定項目

タイムアウトに関する設定項目には以下があります。

設定項目	意味
(a)	Webサーバコネクタの送受信タイムアウト Interstage管理コンソールの以下の項目で指定します。 ・ Webサーバとワークユニットを同一マシンで運用する場合 [ワークユニット] > [Webサーバコネクタ(コネクタ)設定] > [送受信タイムアウト] ・ Webサーバとワークユニットを同一マシンで運用しない場合 [Webサーバコネクタ] > [環境設定] > [送受信タイムアウト]

設定項目	意味
(b)	Servletコンテナのタイムアウト Interstage管理コンソールの[Servletコンテナ設定] > [タイムアウト]で指定します。
(c)	ワークユニットのアプリケーション最大処理時間 Interstage管理コンソールの[ワークユニット設定] > [アプリケーション最大処理時間]で指定します。

上記は、isj2eeadminコマンドを使用して設定することもできます。

ポイント

- 以下の現象が頻繁に発生する場合、(a)、(c)の値を大きくすることで回避できる可能性があります。(a)、(c)の値を大きくしても回避できない場合は、サーバの処理能力を超えている可能性があります。サーバの増設または、より性能の良いサーバへのリプレースを検討してください。
  - コンテナログにIJServer32113が出力される
  - Webサーバコネクタのログに IJServer12035、IJServer12044が出力される
- 以下の現象が頻繁に発生する場合、(b)の値を大きくすることで回避できる可能性があります。回避できない場合はネットワークやクライアントアプリケーションに問題がある可能性があります。発生している問題を解決してください。
  - リクエストデータの読み込みでサーブレットにjava.net.SocketTimeoutExceptionがスローされる
  - Webサーバコネクタのログに IJServer12026、IJServer12027、IJServer12034、IJServer12036が出力される

タイムアウトの設定方法

タイムアウトの設定値は、以下の関係を満たすように値を設定します。

```
設定項目(a) ＞ 監視項目(2) ＋ 監視項目(3) ＋ 監視項目(4)
設定項目(b) ＞ 監視項目(3)
設定項目(c) ＞ 監視項目(4)
```

注意

- 以下の条件に該当する場合、IJServerが強制停止されます。
  - 設定項目(b) ＞ 設定項目(c)に設定した場合、かつ、
  - [ワークユニット設定] > [アプリケーション最大処理時間超過時の制御]を“プロセスを強制停止する”に設定した場合、かつ
  - タイムアウトの監視項目の図における“データの送信2”が設定項目(c)を超えても完了しなかった場合
- 以下に設定した場合、Webブラウザには Webサーバコネクタのタイムアウトが通知されます。
  設定項目(a) ＜ 監視項目(2) ＋ 監視項目(3) ＋ 監視項目(4)
  しかし、Servletコンテナは Webサーバコネクタがタイムアウトしたことを検出することはできないため、アプリケーションを継続して実行し、監視項目(3)の時間内に処理が完了した場合は、Servletコンテナ側で異常を検出することができません。
- 監視項目(4)の時間にはPOSTデータの読み込み時間も含まれます。

27.4 EJBコンテナのチューニング

EJBコンテナをチューニングする時に考慮するポイントは以下です。

- 同時処理数

- Session Bean
- Entity Bean
- Message-driven Bean
- ローカル呼出し機能
- JNDI

27.4.1 同時処理数

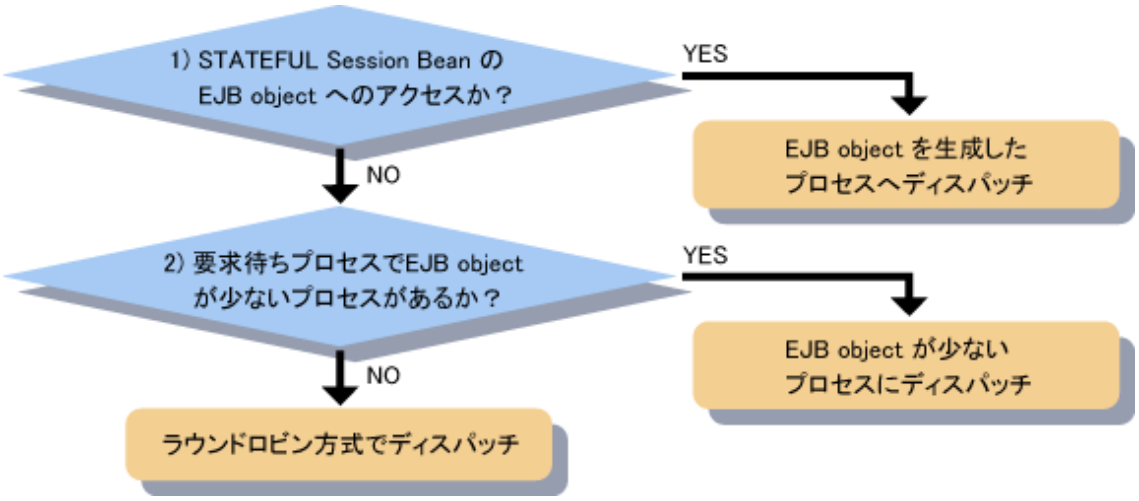
同時処理数は、1プロセスあたりの実行多重度です。IJServerで同時に処理できるクライアントのリクエスト数は、プロセス多重度と同時処理数で決まります。例えばプロセス多重度を2、同時処理数を16に設定すると、合計で2×16＝32の処理が同時に処理できます。クライアントからのリクエストが、処理できるリクエスト数を超えた場合は、キューイングされます。

注意

- 同時処理数を変更することで、1プロセスあたりの実行多重度を増やすことができますが、同時処理数が増えることによる使用資源(CPU使用率、メモリ量など)の増加により、効果が見られない場合があります。
- 資源が不足している場合には、後から実行されたリクエストをキューイングすることにより、レスポンスの安定を図ります。

■プロセスへの処理振り分け

EJBコンテナのプロセス多重度を2以上に設定した場合、各プロセスへのディスパッチ論理は以下のように行われます。



1. STATEFUL Session Beanの場合、createメソッドで取得したEJB objectへのアクセスは、必ず同一のプロセスにディスパッチされます(同一のプロセスにディスパッチされるため、前回の処理をセションの情報として保持し、次回のアクセス時に参照できます)。
2. 1以外の場合には、EJB objectの総数が少ないプロセスに処理を振り分けます。
   各プロセスのEJB objectが同数の場合には、ラウンドロビン方式(処理要求を順番に割り振る方式)でディスパッチするプロセスを決定します。

注意

“ラウンドロビン方式”以外でディスパッチされるリクエストの処理状況が、“ラウンドロビン方式”によるディスパッチ先プロセスの決定に影響する場合があります。その結果、振分けがラウンドロビンにならずリクエストの振分けに偏りが出る場合があります。

各EJBアプリケーション種別で、EJB objectが生成されるタイミングと削除されるタイミングを以下に記載します。
EJBコンテナは、各プロセスに生成されたEJB objectの総数を判断して、適切なプロセスに処理を割り振ります。

EJBアプリケーション種別	EJB objectが生成されるタイミング	EJB objectが削除されるタイミング
STATEFUL Session Bean	createメソッド実行時	・removeメソッド実行時 ・無通信監視タイムアウト時 ・IJServer停止時
STATELESS Session Bean	IJServer起動時に1つ生成	IJServer停止時
Entity Bean	・createメソッド実行時 ・finderメソッド実行時	・removeメソッド実行時 ・Entity BeanのEJB objectタイムアウト時 ・IJServer停止時
Message-driven Bean	ー (Message-driven BeanはRemoteインタフェースがないため、EJB objectは生成されません。)	ー

同時処理数には、以下のように最小値と最大値の設定ができます。
通常はデフォルト値で運用することを推奨します。

定義項目	デフォルト値	説明
最小値	16	IJServer起動直後から、EJBコンテナで同時処理可能なスレッド数です。クライアントのリクエスト数が最小値を超えた場合は、自動的に拡張して動作します。
最大値	64	拡張可能な同時処理数の最大値です。クライアントのリクエスト数が最大値を超えた場合は、リクエストはキューイングされます。

27.4.2 Session Bean

Session Beanのチューニングについて、説明します。
リソースを効果的に利用するために、Session Beanにおいては以下の設定を行います。

- STATEFULとSTATELESSの選択
- STATEFUL Session Beanの無通信監視
- Session Beanのcreateメソッド実行数の上限
- STATELESS Session Beanの起動時インスタンス生成

■STATEFULとSTATELESSの選択

STATELESS Session Beanを使用することで、メモリ資源やオブジェクトの生成回数が抑止されるため、処理性能が向上します。
STATEFULとSTATELESSには以下の違いがありますので、用途に合わせて使い分けてください。

	STATEFUL	STATELESS
対話状態	createからremoveまで同一のオブジェクトにアクセスするため、クライアントとの対話状態を保持することができる。	クライアントとの対話状態を持たないため、クライアントが情報を保持する必要がある。
トランザクション	Session Beanのsynchronization機能を使って、トランザクションとの同期が可能。	1メソッド内でトランザクションを完了させる必要がある。
性能	クライアントごとにEJB objectを生成するため、STATELESSに比べてメモリ使用量が多い。	インスタンスやEJB objectを使い回すため、メモリ使用量を抑止することができる。また、オブジェクトの生成回数が抑止される。

■STATEFUL Session Beanの無通信監視

createメソッドで作成したオブジェクトに対してremoveメソッドを実行せずに終了した場合、残存するオブジェクトを自動的に消去するため、不要なメモリの増加を防ぐことができます。
デフォルト値は、30(分)です。

■Session Beanのcreateメソッド実行数の上限

createメソッド実行数を高負荷の実行環境に合わせて変更できます。
以下の計算式で算出された値が、デフォルト値の1024を超過する場合には、Session Beanのcreateメソッド実行数の上限値を変更してください。createメソッド実行数はInterstage管理コンソールで設定します。

(クライアントアプリケーションのプロセス数)×(1プロセス数あたりの実行スレッド数の平均)

createメソッドで作成したオブジェクトをremoveメソッドで削除しなかった場合、無通信監視機能でタイムアウトが発生するまでオブジェクトが残存するため、メモリの使用量が多くなる可能性があります。create数の上限値は適切な値に設定してください。

設定範囲

種類	設定値
初期設定値	1024
最小値	1
最大値	2147483647

■STATELESS Session Beanの起動時インスタンス生成

IJServer起動時に、STATELESS Session Beanのインスタンスを作成しておくことでアクセス時のインスタンス作成時間が省略され、処理性能が向上します。
設定は、Interstage管理コンソールの[システム] > [ワークユニット] > “ワークユニット名” > “EJBモジュール” > “EJBアプリケーション名” > [アプリケーション環境定義]タブの[Interstage拡張情報]で行います。デフォルトは“しない”です。

設定範囲

種類	設定値
最小値(デフォルト)	0
最大値	2147483647

注意

- 初期起動インスタンス数を増やした場合、使用するヒープ量が増えるため注意してください。

- 初期起動インスタンス数を1以上に指定した場合、setSessionContext、またはejbCreateメソッドから以下の操作、またはアクセスはできません。
  - javax.ejb.TimerServiceメソッド
  - javax.ejb.Timerメソッド
  - 他のEJBアプリケーションへのアクセス
  - リソースマネージャ(データベースなど)へのアクセス
- 作成されたインスタンスは削除されないため、実行時の同時処理数以上に初期起動インスタンス数を設定しないよう、注意してください。
  実行時の同時処理数の上限は、以下の式で算出してください。
  - IJServerタイプが“Webアプリケーションと EJBアプリケーションを同一JavaVMで運用”の場合
    Servletコンテナ設定の同時処理数 ＋ Message-driven Beanの同時処理数の最大値
  - IJServerタイプが“Webアプリケーションと EJBアプリケーションを別JavaVMで運用”または“EJBアプリケーションのみ運用”の場合
    EJBコンテナ設定のIIOP呼び出しの同時処理数の最大値 ＋ Message-driven Beanの同時処理数の最大値

27.4.3 Entity Bean

Entity Beanのチューニングについて、説明します。

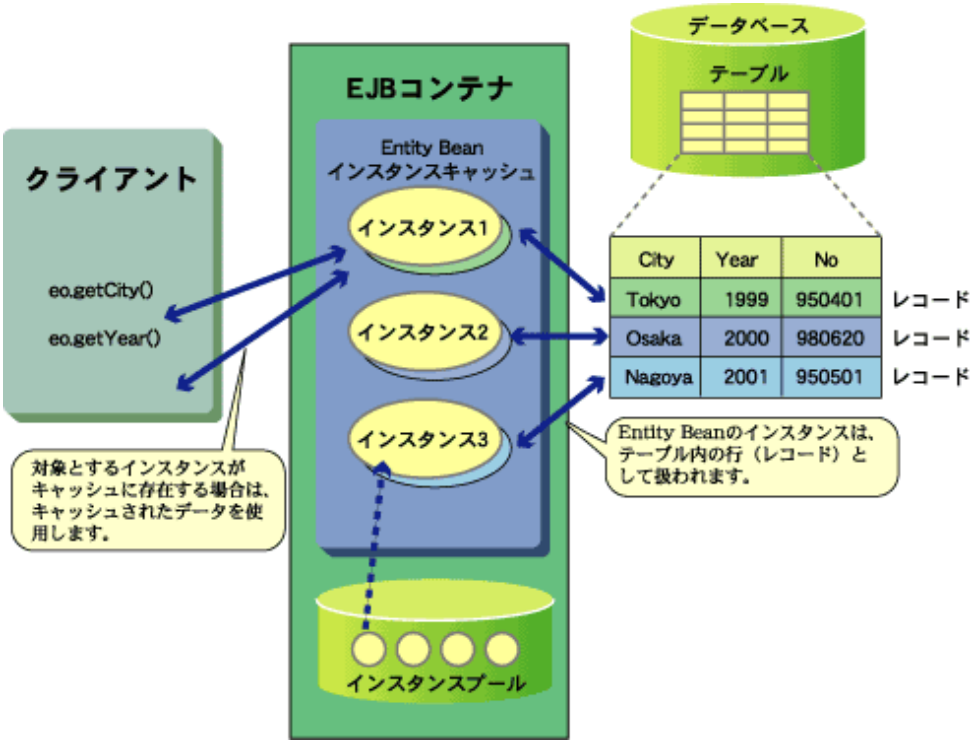
- Entity Bean呼出しの注意
- インスタンス数
- インスタンス管理モード
- CMPデータのstream転送
- CMP2.0の複数件検索高速化
- CMP1.1の複数レコードの一括更新
- CMP1.1のbyte配列更新判定

■Entity Bean呼出しの注意

Entity Beanは、レコードの情報を取得するためにメソッドを頻繁に実行します。このため、プロセス外からEntity Beanを呼び出すとIIOP通信が頻繁に発生することにより、性能が劣化します。
Entity Beanは同一IJServerのアプリケーションから呼び出すことを推奨します。

■インスタンス数

インスタンスはトランザクション内でキャッシュされます。インスタンスプールにインスタンスが存在しない場合、同一トランザクション内で使用したインスタンスのレコードデータをデータベースに反映し、そのインスタンスを別のレコードデータを格納するインスタンスとして再利用します。インスタンスが頻繁に再利用されるとデータベースにアクセスする回数が増加することによって性能に影響がありますので、性能を考慮してインスタンス数を設定してください。



インスタンス数はデータベースの検索レコード数とクライアントの同時接続数に関係します。効果的な値としては、通常検索されるレコード数の1.25倍の値を設定します。
以下に設定値の計算式を表します。

Entityインスタンス数 ＝ a × b × 1.25 (安全率) a: 1度に検索されるレコード数 b: 1プロセスで同時にアクセスするクライアント数

例) 10クライアントが同時に100件を検索した場合
　インスタンス数 ＝ 10 × 100 × 1.25 ＝ 1250

注) インスタンス数を増やすと、使用メモリが増えるので注意してください。

■インスタンス管理モード

Entity Beanのインスタンス管理モードによってデータベースの処理をチューニングすることができます。
以下の表に、それぞれのインスタンス管理モードと最適な処理について示します。

管理モード	最適な処理
ReadWrite	検索を実行する時、またオンラインのデータベースを更新する時に有効
ReadOnly	更新されない主要なデータを検索(参照)する時に有効
Sequential	大量のデータを一括処理する時に有効

■CMPデータのstream転送

CMPで使用するEntity Beanで、JDBCのサイズ制限以上のデータを扱う場合は、Interstage管理コンソールの以下で設定してください。
[システム] > [ワークユニット] > “ワークユニット名” > “EJBモジュール” > “EJBアプリケーション名” > [アプリケーション環境定義]タブを開き、CMFマッピング定義の「CMPデータのstream転送」に“する”を設定します。また、ejbdefexport/

ejbdefimportコマンドを使用して設定することもできます。詳細は“第16章 運用コマンドを使用してカスタマイズする方法”を参照してください。デフォルトは“しない”です。

■CMP2.0の複数件検索高速化

CMP2.0 Entity Beanで複数件finderメソッドを実行した場合に、レコードのデータを一度にすべてロードするオプションを提供しています。
全データを、DBMSからロードするような処理の場合にも、高速にDBMSからデータをロードできます。
詳細は、“10.2.4 CMP2.0の複数件検索時の高速化”を参照してください。

■CMP1.1の複数レコードの一括更新

CMP1.1の複数レコードの一括更新の設定は、Interstage管理コンソールの[システム]>[ワークユニット]>“ワークユニット名”>“EJBモジュール”>“EJBアプリケーション名”>[アプリケーション環境定義]タブの[Interstage拡張情報]で設定します。「複数レコードの一括更新」で“する”を選択してください。デフォルトは“する”です。
また、ejbdefexport/ejbdefimportコマンドを使用して設定することもできます。詳細は“第16章 運用コマンドを使用してカスタマイズする方法”を参照してください。

設定を行ったCMP1.1 Entity Beanはデータベースの更新を行う時に、以下のAPIを利用して一括更新を行います。

- java.sql.PreparedStatementクラスのaddBatchメソッド

注意

- 使用するデータベース、および、JDBCドライバがJDBC2.0バッチ更新機能をサポートしている必要があります。
  JDBC2.0バッチ更新機能が未サポートのデータベース、および、JDBCドライバを使用した場合は、通常のデータベースへの更新処理を行います。
- 「インスタンス管理モード」が“Read-Only”の場合、データの更新自体が行われないため、「複数レコードの一括更新」の設定は無効となります。
- 「複数レコードの一括更新」は、分散トランザクションを使用しない場合にだけ有効です。
  Windows32/64 Linux32/64
  分散トランザクションを使用する場合、通常のデータベースへの更新処理を行います。
- CMP1.1 Entity Beanでstream転送を行う場合、データの更新に失敗する場合があります。その場合は「複数レコードの一括更新」を“しない”に変更してください。

■CMP1.1のbyte配列更新判定

CMP1.1 Entity Beanでbyte配列を使用する場合に、byte配列のデータが更新されているかの判定方法を設定できます。
設定は、Interstage管理コンソールの[システム]>[環境設定]の詳細設定から[EJBサービス詳細設定]タブで「CMP1.1のbyte配列更新判定」で行います。また、isj2eeadminコマンドを使用して設定することもできます。

27.4.4 Message-driven Bean

Message-driven Beanのインスタンス数を設定することにより、同時にメッセージ処理を行うことができます。
目安としては、キューにメッセージが蓄積されない程度のインスタンス数を設定します。ただし、クライアント数とMessage-driven Bean処理時間に依存するため、環境に合わせ試験運用を行い、調整後に設定してください。

■Message-driven Beanのスレッドプール

Point-To-Point、または、Publish/Subscriberメッセージングモデルの Message-driven Beanのメッセージ受信時にプールしたスレッドを利用してメッセージ受信処理を行います。
このプールされたスレッド数のことをMessage-driven Beanの同時処理数と呼びます。

注意

以下の条件に該当するMessage-driven Beanの場合、スレッド多重でのメッセージ受信をサポートしていないため、スレッドプールは使用されません。

	条件A	条件B
メッセージングモデル	Publish/Subscriber	Publish/Subscriber
トランザクション管理種別	Container	Bean
トランザクション属性	Required	－

スレッドプールの単位

IJServerプロセスに1つ作成し、プールしているスレッドはIJServerに配備されているすべてのMessage-driven Beanのメッセージ受信処理で共有されます。

スレッドの作成について

IJServerプロセス起動時にMessage-driven Beanの最小同時処理数のスレッドを作成してプールに格納します。
メッセージが配信されると、メッセージの処理開始前にプールを検索し、プールにスレッドが存在しない場合、および現在利用されているスレッド数がMessage-driven Beanの最大同時処理数より小さい場合に、スレッドを作成して受信処理を行います。
受信完了後、作成したスレッドは削除されることなくプールに返却されます。

注意

プールにスレッドが存在しない場合で、現在利用されているスレッド数がMessage-driven Beanの最大同時処理数を超過している場合は、現在利用されるスレッドがプールに返却されるまで待機し、プールに返却されたスレッドを取得して受信処理を行います。

スレッドの削除について

プーリングされたスレッドは、アイドルタイムアウトによって破棄されます。スレッドがプールに返却されてから指定した時間を超過しても使用されないスレッドが存在する場合には、そのスレッドを破棄します。
ただし、初期起動スレッド数分はプーリングされ続けますので破棄対象外となります。また、IJServerプロセスが停止するとスレッドも削除されます。

スレッドプールのチューニング

Interstage管理コンソールまたはisj2eeadminコマンドで、スレッドプールに対して以下のチューニングを行うことができます。

Message-driven Beanの同時処理数

EJBを運用するプロセス上で、Message-driven Beanで同時に処理できる処理数(スレッド数)の最大値と最小値を指定します。
また、使用されずにプーリングされたスレッドを解放するまでの、タイムアウト時間を指定します。

設定は、Interstage管理コンソールの[ワークユニット]>“ワークユニット名”>[環境設定]>[詳細設定]>[EJBコンテナ設定]で行います。
または、isj2eeadminコマンドで設定することも可能です。詳細は、“リファレンスマニュアル(コマンド編)”を参照してください。

項目	説明	値の説明
最小	IJServerプロセスの起動時に生成するスレッド数を指定します。 生成されたスレッドは、プーリングされてアイドルタイムアウトの対象外となります。	0～2147483647の整数値を指定できます。 デフォルト値は0です。
最大	起動時に最小値だけスレッドが生成され、必要に応じて最大値までスレッドが拡張されます。	1～2147483647の整数値を指定できます。 デフォルト値は64です。
アイドルタイムアウト	アイドルタイムアウト値を指定します。 スレッドが、プールに返却されてから指定した時間を超過しても、使用されないスレッドを破棄します。ただし、最小同時処理数分は破棄対象外となります。 0を指定した場合には、タイムアウトしません。	0～2147483647の整数値を指定できます。 デフォルト値は600(秒)です。

Message-driven Beanの同時処理数をチューニングする場合、以下を参考にしてください。

- 断続的に処理を実行するような場合には、常に同時に実行されるスレッド数を最小値に設定することで、スレッドの作成、破棄の処理が軽減されるためにCPU使用率が軽減されて性能が向上します。
  Message-driven Beanの処理が頻繁に行われない場合には、最小値を小さく設定することで、使用メモリ量を抑えることができます。
- Interstage管理コンソールのモニタ情報を参照して、アイドルタイムアウト回数が断続的に増加している場合、スレッドの生成と破棄が頻繁に発生しています。
  アイドルタイムアウトの設定値が小さい可能性がありますので、アイドルタイムアウトの設定値を大きくすることを検討してください。
  ただし、アイドルタイムアウトの設定値を大きく設定すると生成したスレッドがプールに滞留する時間が長くなるため、メモリの使用量も増加します。Javaのヒープ量やシステム資源も考慮して設定してください。
- 膨大な処理要求を受け付ける可能性がある場合、最大値を小さく指定して同時処理数を抑制することで、CPU負荷を軽減できます。
  CPUの使用率が上限値に対して低い場合には、最大値を大きく設定することで、最大値に指定した値までMessage-driven Beanで同時処理できます。
- 各Message-driven Beanの初期起動スレッド数の合計値が、Message-driven Beanの同時処理数の最大値を上回っている場合、Message-driven Beanの同時処理数の最大値以上のメッセージがIJServerプロセスに配信される可能性があります。
  その場合、使用中のスレッドがプールに返却されるまで待機します。プールに返却されるまで待機させない場合には、各Message-driven Beanの初期起動インスタンス数の合計値をInterstage管理コンソール、またはisj2eeadminコマンドで、Message-driven Beanの同時処理数の最大値に指定してください。

■異常時のメッセージ退避機能

Point-To-Pointメッセージングモデルでかつ不揮発化チャネルを利用する場合、リトライ回数を超過してもメッセージ受信を繰り返す可能性があります。
イベントチャネルのイベントデータをメモリにキャッシュする数を、イベントチャネルに蓄積できるイベントデータの最大値よりも大きく設定してください。
設定はイベントサービス運用コマンドを使用して行います。詳細は“リファレンスマニュアル(コマンド編)”の“essetcnf”および“essetcnfchnl”を参照してください。

27.4.5 ローカル呼出し機能

IJServerに配備されたEJBアプリケーションが、同一のEJBコンテナ内のEJBアプリケーションからだけで呼び出される場合、ローカル呼出し機能を使用できます。
ローカル呼出し機能を使用することによって、ネットワークを介してEJBアプリケーションを呼び出されることを考慮したIJServerの処理が軽減されるため、通常よりも更に性能良く動作します。
本機能は、IJServerが“EJBアプリケーションのみ運用する場合”、または、“WebアプリケーションとEJBアプリケーションを別JavaVMで運用する場合”にだけ有効です。

ローカル呼出し機能の設定はInterstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション] > [アプリケーション環境定義] > [Interstage拡張情報]の“ローカル呼出し”で行います。設定の詳細についてはInterstage管理コンソールのヘルプを参照してください。

注意

- 「ローカル呼出し」を“する”に設定したEJBアプリケーションをプロセス外から呼び出した場合、“CORBA OBJ_ADAPTER”のエラーが発生します。
- Entity Beanをプロセス外から呼び出す場合には、「ローカル呼出し」を“しない”に変更してください。また、ローカル呼出しをしない場合には、EJB objectをタイマで削除してください。EJB objectのタイマ削除については“10.5.4 EJB objectのタイマ削除機能”を参照してください。

27.4.6 JNDI

■オブジェクトの使い回し

lookupメソッドで取得したオブジェクトは、EJBアプリケーション内で保持して使い回すことにより、lookupメソッドの実行回数を軽減できます。

■deployment descriptor定義

deployment descriptorファイルにオブジェクトの情報を定義することができます。

- 定義した場合、定義された情報を元に各ネーミングサービスからIJServer起動時にオブジェクトを取得してメモリ上に保持するため、処理性能が向上します。
- 定義しない場合、アプリケーションでlookupメソッドを実行した時にオブジェクトが各ネーミングサービスに存在しないか確認します。

deployment descriptorファイルにオブジェクトの情報を定義することを推奨しますが、すでに開発済みのEARファイルを使用する場合などでdeployment descriptorファイルの編集ができない場合、以下のオプションを使用することで、ネーミングサービスへのアクセス回数を軽減できます。

項目	設定内容
定義ファイル格納ディレクトリ	Windows32/64 Interstageインストールディレクトリ¥ejb¥etc Solaris64 Linux32/64 /opt/FJSVejb/etc
定義ファイル	FJEJBconfig.properties
指定するキー名	“LookupCache”(固定)
指定する値	• 1:同一オブジェクトへの2回目以降のlookupメソッド実行時にはメモリに保持した情報を返却します。 • 1以外: lookupメソッドを実行する度に、各ネーミングサービスにアクセスします(デフォルト)。

本オプションを使用すると、lookupメソッド実行時に各ネーミングサービスから取得した情報をメモリ上に保持するため、同一のオブジェクトに対する2回目以降のlookupメソッドの処理性能が向上します。
なお、以下のオブジェクトを取得する場合には、本オプションを設定してIJServerを運用してください。

- 他のIJServerに配備されたEJBアプリケーションのHomeオブジェクト
- データソース
- JMSコネクションファクトリ
- JMS Destination

27.5 CORBAサービスのチューニング

以下について説明します。

- Interstageシステム定義ファイルの生成
- CORBAサービス環境定義のチューニング

■Interstageシステム定義ファイルの生成

通常の運用ではカスタマイズの必要はありません。

Interstageの接続クライアント数を変更する場合は、システム規模にあわせてisgendefコマンドのscale-valueの値を設定してください。
isgendefコマンドの詳細については、“運用ガイド(基本編)”の“Interstage統合コマンドによる運用操作”を参照してください。

■CORBAサービス環境定義のチューニング

通常の運用ではカスタマイズの必要はありません。

システム規模の拡張やIJServerの追加によりデフォルト値の設定で資源が不足する場合は、「チューニングガイド」－「Interstageのチューニング」に記載されている「CORBAサービスの動作環境ファイル」の加算値を参照して、Interstageをインストールしたマシンで CORBAサービス環境定義のステートメント値を加算してください。

EJBサービスを使用する場合

IJServerの作成/削除、EJBアプリケーションを配備/配備解除するために、EJBサービスとしてCORBAサービスの資源を使用します。以下の値を加算してください。

ステートメント	加算値
max_processes	1
max_IIOP_resp_con	1

IJServerを追加した場合

IJServerの追加または、IJServerのプロセス多重度を変更する場合には、以下の値を加算してください。

ステートメント	加算値
max_processes	追加するIJServerのプロセス多重度 (注1)
max_exec_instance	追加するIJServer(EJBコンテナ)のプロセス数 × 64(IJServer作成時に指定する同時処理数の最大値) (注2)
max_impl_rep_entries	3 (注2)

注1) すべてのIJServerのタイプの場合で加算してください。

注2) 以下のIJServerのタイプの場合に加算してください。

- WebアプリケーションとEJBアプリケーションを別JavaVMで運用
- EJBアプリケーションのみ運用

クライアントアプリケーションを追加した場合

J2EEアプリケーションクライアントまたは、アプレットからEJBアプリケーションを呼び出す場合には、以下の値を加算してください。

ステートメント	加算値
max_IIOP_resp_con	追加するクライアントアプリケーションのプロセス数

EJBアプリケーションを呼び出すアプリケーションをIJServerに配備した場合

別マシンや別プロセスからEJBアプリケーションを呼び出す場合、以下の値を加算してください。

ステートメント	加算値
max_IIOP_resp_con	EJBアプリケーションを呼び出す側のIJServerのプロセス数

上記のEJBアプリケーションを呼び出すアプリケーションのプロセス数には、以下の表に示すプロセス数の合計値を使用してください。

- WebアプリケーションとEJBアプリケーションを同一JavaVMで運用する場合

	条件	加算値
1	WebアプリケーションやEJBアプリケーションから別のIJServerに配備されているEJBアプリケーションを呼び出す場合	IJServerのプロセス多重度
2	WebアプリケーションやEJBアプリケーションから別のIJServerに配備されているEJBアプリケーションを呼び出さない場合	なし

- WebアプリケーションとEJBアプリケーションを別JavaVMで運用する場合

	条件	加算値
1	Webアプリケーションから同じIJServerに配備されているEJBアプリケーションを呼び出す場合	Servletコンテナのプロセス多重度 (注1)
2	EJBアプリケーションから同じIJServerに配備されているEJBアプリケーションを呼び出す場合	EJBコンテナのプロセス多重度 (注2)(注3)
3	Webアプリケーションから別のIJServerに配備されているEJBアプリケーションを呼び出す場合	Servletコンテナのプロセス多重度 (注1)
4	EJBアプリケーションから別のIJServerに配備されているEJBアプリケーションを呼び出す場合	EJBコンテナのプロセス多重度 (注2)

注1) 1と3の条件を満たす場合は、1回加算してください。
注2) 2と4の条件を満たす場合は、1回加算してください。
注3) アプリケーション環境定義で「ローカル呼出し」が“する”に設定されているEJBアプリケーションの場合、加算する必要はありません。

- EJBアプリケーションのみ運用する場合

	条件	加算値
1	EJBアプリケーションから同じIJServerに配備されているEJBアプリケーションを呼び出す場合	EJBコンテナのプロセス多重度 (注1)(注2)
2	EJBアプリケーションから別のIJServerに配備されているEJBアプリケーションを呼び出す場合	EJBコンテナのプロセス多重度 (注1)

注1) 1と2の条件を満たす場合は、1回加算してください。
注2) アプリケーション環境定義で「ローカル呼出し」が“する”に設定されているEJBアプリケーションの場合、加算する必要はありません。

・ Webアプリケーションのみ運用する場合

	条件	加算値
1	Webアプリケーションから別のIJServerに配備されているEJBアプリケーションを呼び出す場合	Servletコンテナのプロセス多重度

注意

EJBアプリケーションを使用する場合、CORBAサービスの無通信監視は使用できません。

27.6 LDAPサーバとしての、ディレクトリサービスのチューニング

Interstage ディレクトリサービスは、以下の製品で使用することができます。

・ Interstage Application Server Enterprise Edition

・ Interstage Application Server Standard-J Edition

Interstage ディレクトリサービスのチューニングについては、以下を参照してください。

・ 「チューニングガイド」－「Interstage ディレクトリサービスのシステム資源の設定」 Solaris64 Linux32/64

・ “ディレクトリサービス運用ガイド”の“検索のチューニング”

第28章　Systemwalkerとの連携

Interstageでは Systemwalker製品と連携して、以下の機能が使用できます。

- Systemwalker Service Quality Coordinatorと連携したトランザクション内訳分析 Windows32/64 Linux32/64

注意

連携するSystemwalkerのバージョン・レベルにより、使用できる機能を限定される場合があります。「インストールガイド」を参照し、使用できる機能範囲を確認してください。

Systemwalker Service Quality Coordinatorと連携したトランザクション内訳分析 Windows32/64 Linux32/64

Systemwalker Service Quality Coordinatorのトランザクション内訳分析機能を利用して、J2EEアプリケーションのコンポーネントごとの処理時間を測定することができます。
詳しくは、Systemwalker Service Quality Coordinatorのマニュアルを参照してください。

28.1 Systemwalker Service Quality Coordinatorと連携したトランザクション内訳分析 Windows32/64 Linux32/64

Systemwalker Service Quality Coordinatorのトランザクション内訳分析機能を利用して、IJServer上で動作するJ2EEアプリケーションのコンポーネントごとの処理時間を測定することができます。
測定できるアプリケーションは、IJServerタイプによって以下のようになります。

IJServerタイプ	測定できるアプリケーション
WebアプリケーションとEJBアプリケーションを同一JavaVMで運用	Webアプリケーション EJBアプリケーション
WebアプリケーションとEJBアプリケーションを別JavaVMで運用	Webアプリケーション **注意** EJBアプリケーションは測定されません。
Webアプリケーションのみ運用	Webアプリケーション
EJBアプリケーションのみ運用	なし

Webアプリケーションはサーブレット/JSP単位、EJBアプリケーションはメソッド単位に測定されます。
測定を行う場合は、Interstage管理コンソールの [Interstage管理コンソール] > [Interstage Application Server] > [システム] > [ワークユニット] > [(ワークユニット名)]の環境設定タブで、[共通定義]-[トランザクション内訳分析]項目に「使用する」を設定します。また、isj2eeadminコマンドを使用して設定することもできます。
isj2eeadminコマンドを使用する場合の詳細は「リファレンスマニュアル(コマンド編)」の「isj2eeadmin」を参照してください。
その他にSystemwalker Service Quality Coordinatorの設定が必要です。詳しくは、Systemwalker Service Quality Coordinatorのマニュアルを参照してください。

注意

- 本機能を使用するには、Systemwalker Service Quality Coordinator Agentがインストールされている必要があります。詳細は、「システム設計ガイド」の「ソフトウェア条件」をご確認ください。
- Webアプリケーションでは、Servlet/JSP以外のコンテンツ(htmlファイルなど)へのリクエストは測定されません。

第9部 トラブルシューティング

第29章　J2EEアプリケーション開発・運用時の異常

本章では、J2EEアプリケーションの開発や運用中に異常が発生した場合の対処方法を説明します。
異常への対処は、次の手順で行います。

1. 異常情報の参照
2. 異常発生コンポーネントの特定
3. 異常への対処

29.1 異常情報の参照

J2EEアプリケーションの異常情報は次の箇所に出力されます。

- コマンドの実行画面へのメッセージ
- システムログ
- J2EEコンポーネントのログファイル
- Webブラウザへのメッセージ
- アプリケーションの例外情報

それぞれの出力先を下表に示します。

	異常情報	出力先	対象製品
サーバ	コンテナログ ・アプリケーションの標準出力、標準エラー出力 ・GenericServletクラスのlogメソッドの出力 ・ServletContextクラスのlogメソッドの出力 ・EJBコンテナログ/Servletコンテナログ ・EJBコンテナ/Servletコンテナのエラーメッセージ ・EJBのスナップ出力 ・Webサービスのログ	Windows32/64 [J2EE共通ディレクトリ]¥ijserver¥[IJServer名]¥log¥[プロセス通番]¥container.log (注2) Solaris64 Linux32/64 [J2EE共通ディレクトリ]/ijserver/[IJServer名]/log/[プロセス通番]/container.log (注2)	全製品
	コンテナ情報ログ ・Java VMプロセスの起動情報(ARGV、ENV) ・Java VMプロセスの起動エラーメッセージ ・スレッドダンプ ・コンテナログに出力できないメッセージ (注1)	Windows32/64 [J2EE共通ディレクトリ]¥ijserver¥[IJServer名]¥log¥[プロセス通番]¥info.log (注2) Solaris64 Linux32/64 [J2EE共通ディレクトリ]/ijserver/[IJServer名]/log/[プロセス通番]/info.log (注2)	全製品
	メッセージ	システムログ	全製品
クライアント	ユーザアプリケーションの例外情報(Exception)	アプリケーションに通知されます。	全製品
	メッセージ	コマンドの実行画面	全製品

	異常情報	出力先	対象製品
	Webのメッセージおよびステータスコード	Webブラウザのページ	全製品

注1)

テキストエディタの種類によっては、IJServerログファイルを開いたままアプリケーションを実行した場合に、メッセージが出力されないことがあります。その場合、コンテナ情報ログ(info.log)に以下のメッセージが出力されています。コンテナ情報ログ(info.log)に本メッセージを出力できなかった場合は、同じディレクトリに“IJLogger_err.log”というファイルが生成され、このファイルに出力されます。

- ERROR: The output of a message was not processed normally.
  Please check whether the file is opened or there is any authority of writing.

上記メッセージが出力されていた場合は、以下の対処を行ってください。

- IJServerログファイルを閉じる
- IJServerログに対して書き込み権限があるか確認する
- 他のプロセスでIJServerログファイルが使用されていないか確認する

注2)

デフォルトの出力先です。出力先はInterstage管理コンソールの[ワークユニット] > “ワークユニット名” > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [ログ出力ディレクトリ]で変更できます。

29.2 異常発生コンポーネントの特定と対処

異常情報の出力先によって、異常が発生したコンポーネントを特定し、対処する方法を以降に示します。

■コマンドおよびInterstage管理コンソールの実行画面

Servletコンテナ関連の異常が発生した可能性があります。
“29.7 Webアプリケーションの開発・運用時の異常”を参照してください。

以下の操作を実施した時、DEP1863またはDEP1873が出力されて、IJServerのディレクトリ/ファイルが残存する場合があります。

- 配備解除
- IJServerの削除

残存した場合、しばらくしてから手動で削除してください。削除できない場合は、Interstage JMXサービスを再起動してから削除してください。

■システムログ

システムログに出力されたメッセージについては、“メッセージ集”を参照して対処を行ってください。
また、メッセージ集とあわせて、下表の対処方法も参照してください。

メッセージ	対処方法
“IJServer2”で始まるメッセージ、または“EJB”で始まるメッセージの場合	EJBアプリケーションの異常です。“29.9 EJBサービス使用時の異常”を参照してください。
“JMS:”で始まるメッセージの場合	JMSアプリケーションの異常です。“29.12 Interstage JMSの異常時の対処”を参照してください。

メッセージ	対処方法
“OTS:”で始まるメッセージの場合	データベース連携サービスの異常です。“トラブルシューティング集”－“データベース連携サービス使用時の異常”を参照してください。

■Webブラウザ

Webアプリケーションの異常である可能性があります。
ブラウザ(ページ)上に表示されるメッセージおよびステータスコードに従って、“29.7 Webアプリケーションの開発・運用時の異常”を参照してください。

■アプリケーションの例外情報(Exception)

Exceptionの代表的な例を下表に示します。

Exception	発生コンポーネント	対処方法
javax.naming.XXX	JNDI	JNDI関連の異常の可能性があります。“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。
org.omg.CORBA.XXX	CORBAサービス	“メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”を参照してください。
javax.transaction.XXX	コンポーネントトランザクションサービス	“メッセージ集”の“コンポーネントトランザクションサービスから通知される例外情報”を参照してください。
javax.transaction.xa.XXX	データベース連携サービス	“メッセージ集”の“データベース連携サービスから通知される例外情報”を参照してください。
javax.servlet.XXX	Servlet	Servlet関連の異常の可能性があります。“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。“
javax.ejb.XXX	EJB	EJB関連の異常の可能性があります。“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。
javax.sql.XXX	JDBC	発生したExceptionの詳細情報を参照して異常の対処を行ってください。
javax.mail.XXX	JavaMail	発生したExceptionの詳細情報を参照して異常の対処を行ってください。
javax.jms.XXX	JMS	発生したExceptionの詳細情報を参照して異常の対処を行ってください。

表に示すException以外の場合には、次の記事を参照するか、または発生したExceptionの詳細情報を参照して異常の対処を行ってください。

- “メッセージ集”の“J2EE使用時に出力される例外情報”
- “メッセージ集”の“J2EEアプリケーションのセキュリティ機能で出力するメッセージ”
- “メッセージ集”の“CORBAサービスから通知される例外情報/マイナーコード”
- “メッセージ集”の“コンポーネントトランザクションサービスから通知される例外情報”
- “メッセージ集”の“データベース連携サービスから通知される例外情報”
- “EJBサービス使用時の異常”

■セキュリティ機能での異常について

セキュリティ機能で異常が発生した場合は、次の原因が考えられます。
また、各アプリケーションのログに“security”で始まるエラーメッセージが出力される場合は、“メッセージ集”の“J2EEアプリケーションのセキュリティ機能で出力するメッセージ”に従って対処してください。

アプリケーション	原因
共通	セキュリティ管理環境定義ファイルが存在しない、読み込み権限がない、または内容が正しく設定されていない場合
	ディレクトリサービスが使用可能な状態でない場合
	ディレクトリサービスにユーザID、パスワードが登録されてない場合
J2EEアプリケーションクライアント	JNDI環境プロパティのFJUserIDまたはFJPasswordがNULLまたは空の場合
Webアプリケーション	Webアプリケーション環境定義ファイルのユーザ認証の定義が正しく設定されていない場合
EJBアプリケーション	Interstage管理コンソールのメソッドパーミッションタグが正しく設定されていない場合

29.3 IJServer起動時の異常

IJServerの起動で異常が発生した場合は、Interstage管理コンソール(Interstage管理コンソールから起動した場合)、コマンド実行画面(コマンドから起動した場合)、システムログ、コンテナログ、および、コンテナ情報ログについて、異常・問題を示すメッセージをチェックし、“メッセージ集”にしたがって対処してください。特定の場合の対処について、ポイントを以下に示します。

リソース(JDBC、JMSなど)を使用するIJServerの起動が完了したとき、システムログに以下のIJServer2、またはEJBで始まるメッセージが出力されることがあります。

- IJServer21066、またはEJB1066
- IJServer21067、またはEJB1067
- IJServer21069、またはEJB1069
- IJServer21243、またはEJB1243 (注)

上記のメッセージが出力された場合、可変情報に出力されたリソースを当該EJBアプリケーションが使用する場合は、一度IJServerを停止し原因を取り除いてから、再度起動処理を実行してください。
可変情報に出力されたリソースを当該EJBアプリケーションが使用しない場合は、問題なくEJBアプリケーションを動作させることができます。
各メッセージの意味や対処方法などの詳細ついては、“メッセージ集”を参照してください。

注)

- IJServerのワークユニット定義に、使用するJDBCのクラスパスが正しく設定されていない可能性があります。ワークユニット定義に、正しいJDBCのクラスパスを設定後、IJServerを再起動してください。
  ワークユニット定義の詳細は、Interstage管理コンソールのヘルプを参照してください。
- IJServer21243、またはEJB1243のメッセージが出力されたとき、同時にIJServerのログファイルに以下の例外が出力されることがありますが、アプリケーションの動作とは関係ありません。
  例外 ： FJCD_LockFailedException

IJServerをデバッグする場合は、プロセス多重度を確認してください。プロセス多重度を2以上にして起動することはできません。デバッグする際には、ワークユニットの“プロセス多重度”を必ず1に設定してください。

WebサーバコネクタとServletコンテナ間の通信にSSLを使用する設定の場合、Interstage管理コンソールのヘルプを参照してSSLの設定内容を見直してください。

Servletコンテナ起動時にServletの初期化(initメソッドの呼出し)を行うWebアプリケーションで、IJServer起動時に例外が発生する場合は、“29.6 Javaの例外(Exception)と対処方法”を参照してください。

Oracleを使用し、ドライバタイプがociの場合、環境変数ORACLE_HOMEの設定が必要です。“4.3.4 Oracleを使用する場合の環境設定”を参照し、IJServer環境設定の環境変数にORACLE_HOMEを設定してください。

Solaris64 Linux32/64

IJServerの起動に失敗し、以下のメッセージがシステムログに出力される場合は、EJBアプリケーションの転送を行った際に誤りが発生している可能性があります。

- IJServer21035、またはEJB1035

上記のメッセージが出力された場合は、再度、以下の点に注意し、EJBアプリケーションの転送を行ってください。

- 転送モードは、バイナリモードで行ってください。
- ファイル名の大文字、小文字が変換されないように行ってください。

上記の対処を行ったにもかかわらず現象が変わらない場合は、他の原因が考えられます。メッセージのユーザの対処を参照してください。

29.4 IJServer運用時の異常

クライアントからEJBアプリケーション接続時の異常

クライアントからEJBアプリケーション接続時に、以下のメッセージがシステムログに出力される場合は、IJServerのワークユニット定義に、使用するJDBCのパスが正しく設定されていない可能性があります。ワークユニット定義に、正しいJDBCのパスを設定後、IJServerを再起動してください。
ワークユニット定義の詳細は、Interstage管理コンソールのヘルプを参照してください。

- “EJB:エラー:EJB1020:メソッドの実行に失敗しました”
  “IJServer:エラー:IJServer21020:メソッドの実行に失敗しました”

IJServer2で始まるエラーメッセージの埋め込み文字が文字化けする場合 Linux32/64

環境変数LANGにja_JP.UTF-8を指定した場合、システムログ上に出力されるエラーメッセージの可変文字列に日本語文字が含まれていると文字化けします。
IJServerのワークユニット定義で環境変数LANGの設定を変更するか、コンテナログを参照してエラーメッセージを確認してください。

IJServer停止時の異常

IJServerの停止時に、以下のメッセージがシステムログに出力される場合は、クライアントアプリケーションがインスタンスを消去しないで終了した場合や、インスタンスが消去する前に異常終了した可能性があります。この場合は、IJServerを強制停止してください。

- “extp:エラー:EXTP4406:アプリケーションが実行中のためワークユニットが停止できません”

XMLファイル読み込み時の異常

XMLファイルに以下のように定義するエンコードは、各XMLパーサによってサポートするエンコードが異なります。他の環境で動作していたアプリケーションを移行した場合に、XMLファイルの読み込みに失敗した場合には、使用しているXMLパーサが指定しているエンコードをサポートしているか確認してください。ほぼすべてのXMLパーサがサポートする“UTF-8”を使用することを推奨します。

```
<?xml version="1.0" encoding="UTF-8"?>
・・・
```

使用するXMLパーサによって、エンコード以外に機能範囲は異なる場合があります。詳細は使用するXMLパーサのマニュアルを参照してください。

コンテナログのエラー情報

IJServer運用中にInterstage JMXサービスが停止された場合、コンテナログに以下のメッセージが出力されます。
IJServerの運用には影響ないため、特に対処の必要はありません。

- ClientCommunicatorAdmin restart
  警告: Failed to restart: java.io.IOException: Failed to get a RMI stub: javax.naming.ServiceUnavailableException [Root exception is java.rmi.ConnectException: Connection refused to host: [ホスト名またはIPアドレス]; nested exception is: java.net.ConnectException: Connection refused: connect]

J2EEモニタロギング機能の性能情報の異常

JavaVM情報が同じ値で変化しない場合、Javaプロセスの実行が阻害される要因がないか確認し、システムの過負荷過負荷の原因を取り除いた後でIJServerを再起動してください。また、Java VM情報は、jheapおよびGCログでも取得できます。jheapおよびGCログの詳細は“トラブルシューティング集”－“Javaツール機能”を参照してください。

Servletコンテナ情報のスレッド数やスレッドプール数で異常な値(負の値)が出力された場合、IJServerで資源の枯渇などの異常が発生している可能性があります。
コンテナログおよび起動情報(info.log)を参照し、出力されているメッセージに従って対処してください。

29.5 HotDeploy機能使用時の異常

ポイント

Interstage管理コンソールからの配備・配備解除処理一般については、“Interstage管理コンソール操作中の異常”の“その他の異常”にも記載があります。

29.5.1 各操作（配備／再配備／配備解除／再活性）に失敗した場合

■配備モジュールの状態

以下に、配備、再配備、配備解除、再活性の各操作において異常が発生した場合における配備モジュールの状態を示します。

失敗したタイミング	各操作 失敗後のモジュールの状態			
	配備	再配備	配備解除	再活性 (注1)
モジュールの配備	アプリケーションのリクエスト受付を停止しません。	アプリケーションのリクエスト受付を開始しません。	－	
配備モジュールの配備解除	－		非活性	－
配備モジュールの活性化 (注2)	1) 活性化処理中 活性化処理が1分間で完了しなかった場合には、“活性化処理中”の状態で処理が終了します。活性化処理が完了すると“活性”の状態となります。(注3) 2) 非活性 活性化処理中に定義異常などで異常が発生し、非活性状態で処理が終了した場合には“非活性”の状態となります。			
配備モジュールの非活性化	アプリケーションのリクエスト受付を停止しません。 非活性化処理中 (注4) 非活性化処理が1分間で完了しなかった場合には、“非活性化処理中”の状態で			

失敗したタイミング	各操作 失敗後のモジュールの状態			
	配備	再配備	配備解除	再活性 (注1)
	処理が終了します。非活性化処理が完了すると“非活性”の状態となります。(注3)			

注1)

IJServerが起動していない場合、再活性を実行すると以下のエラーが発生します。
is40131:IJServerが起動されていません。(IJServer名=%s)

注2)

活性化で異常が発生したモジュールについては、異常が発生した原因を取り除いてモジュールを再活性することにより、運用可能な状態とすることが可能です。

注3)

単純に活性／非活性処理が長時間かかっているだけであれば、処理が終了するまで待てば自動的に活性／非活性の状態となります。活性／非活性処理でハングしている場合には、活性／非活性の状態のままとなります。この場合にはアプリケーションのデバッグ情報などを参照して、ハング原因を調査して対処してください。
アプリケーションのデバッグ方法については、“3.12 アプリケーションのデバッグ”を参照してください。

注4)

非活性化処理中にメモリ不足が発生した場合には、“非活性化処理中”の状態で配備処理が中断します。他の処理が終了後に再度配備を実行してください。ただし、非活性化処理で実行されるアプリケーションのメソッド(たとえば、Entity BeanのunsetEntityContextメソッド)でエラーが発生した場合、再度配備を実行しても同様のエラーが発生すると考えられるために配備処理を継続します。

■is20711/is20725/is20726/is20727/is20746のメッセージが出力された場合

HotDeploy機能使用時に、以下のメッセージが出力されることがあります。

- IS: エラー: is20711:通信エラーが発生しました エラー情報=%s
- IS: エラー: is20725: サーバとの接続に失敗しました。エラー情報=%s
- IS: エラー: is20726: Interstage JMXサービスとの接続に失敗しました。エラー情報=%s
- IS: エラー: is20727:通信エラーが発生しました 例外情報=%s1 例外メッセージ=%s2
- IS: エラー: is20746:要求を発行した対象のリソースが存在しません。別ユーザからの要求により削除されたか、Interstage JMXサービスが再起動された可能性があります

上記のメッセージが出力された場合の原因と対処を説明します。

- Interstage JMXサービスを業務運用中に停止、および、再起動させた場合が考えられます。HotDeploy機能を有効にするために、IJServerを再起動してください。
  また、業務運用中にInterstage JMXサービスを停止させないようにしてください。
- Windows32/64
  マシン起動時に、ワークユニット定義のワークユニット自動起動が有効になっている場合が考えられます。本現象が発生した場合には、環境変数“IS_JMX_SERVICE_WAIT_TIMEOUT”をシステムの環境変数に追加し、以下の値を設定することにより対処してください。本設定は、次回のマシン起動時より有効となります。

  ー メッセージtd11001の出力時刻とメッセージis20701のメッセージ出力時刻の時間間隔(単位:秒)に180を加算した値

29.5.2 非活性状態のモジュールに対してリクエストを送信した場合

非活性状態のモジュールに対してリクエストを送信した場合、WebアプリケーションとEJBアプリケーションでは以下のエラーが発生します。

アプリケーション種別	エラー
Webアプリケーション	ブラウザ上に404のエラーが発生します。

アプリケーション種別	エラー
EJBアプリケーション (Message-driven Bean 以外)	停止中にクライアントアプリケーションからアクセスすると以下の例外が返却されます。 java.rmi.RemoteException: CORBA OBJ_ADAPTER 1179257480 No また、アプリケーション停止前に取得したEJB objectに対して、アプリケーション再起動後にアクセスした場合には以下の例外が返却されます。 java.rmi.NoSuchObjectException: CORBA OBJECT_NOT_EXIST 0 No; nested exception is: org.omg.CORBA.OBJECT_NOT_EXIST: CORBA_Request_get_response minor code: 0 completed: No
EJBアプリケーション (Message-driven Bean)	メッセージ受信を停止しますが、エラーは発生しません。通常はDestinationにメッセージが滞留します。Publish/Subscribeメッセージングモデル(1対nメッセージングモデル)のDurable Subscription機能を使用しない場合には、メッセージが消滅します。

対処方法

リクエスト送信先の配備モジュールの状態を確認してください。非活性状態となっている場合には、配備処理中、配備解除処理中、再活性中の可性能があります。処理が完了するまで待ってから再度リクエストを送信してください。

上記に該当しない場合、活性化処理に失敗している可能性があります。コンテナログファイルを確認して、以下の例外が出力されていないか確認してください。

- java.rmi.RemoteException: CORBA OBJ_ADAPTER 1179257480 No

また、アプリケーション停止前に取得したEJB objectに対して、アプリケーション再起動後にアクセスした場合には以下の例外が返却されます。

- java.rmi.NoSuchObjectException: CORBA OBJECT_NOT_EXIST 0 No; nestedexception is: org.omg.CORBA.OBJECT_NOT_EXIST: CORBA_Request_get_responseminor code: 0 completed: No

これらの例外が出力されている場合には、その前後に出力されているメッセージからエラー原因を調査して対処してください。

29.5.3 破棄されたServletのセションとSTATEFUL Session Beanのインスタンスにアクセスしようとした場合

破棄されたServletのセション、またはSTATEFUL Session Beanのインスタンスにアクセスしようとした場合、以下のエラーが発生します。

破棄されたもの	エラー
Servletのセション	javax.servlet.http.HttpServletRequestに対してgetSession(boolean)メソッドを実行した場合、以下のように動作します。 ・ 引数がtrueの場合:新たにセションが生成されます。 ・ 引数がfalseの場合:nullが返却されます。
STATEFUL Session Beanのインスタンス	クライアントに以下の例外が返却されます。 java.rmi.NoSuchObjectException: CORBA OBJECT_NOT_EXIST 0 No; nested exception is: org.omg.CORBA.OBJECT_NOT_EXIST: CORBA_Request_get_response minor code: 0 completed: No

対処方法

クライアントアプリケーションでは、createメソッドを実行してEJB objectを再作成してから処理を継続してください。

29.6 Javaの例外(Exception)と対処方法

アプリケーション処理で例外が発生した場合、IJServerのコンテナログにJavaのスタックトレースが出力されます。
スタックトレースの出力例と解析方法を以下に説明します。

■出力例

Webアプリケーションのサーブレットで例外が発生した場合で説明します。
(先頭の“数字:”は、説明の都合上記載しています。)

```
1:  java.lang.NullPointerException
2:    at agency.attestation.CheckLoginInfo.doCheck(CheckLoginInfo.java:150)
3:    at agency.attestation.AttestationServlet.doGet(AttestationServlet.java:96)
4:    at agency.attestation.AttestationServlet.doPost(AttestationServlet.java:161)
5:    at javax.servlet.http.HttpServlet.service(HttpServlet.java:772)
6:    at javax.servlet.http.HttpServlet.service(HttpServlet.java:865)
                        :
```

1. NullPointerExceptionが発生(例外)
2. 発生個所はagency.attestation.CheckLoginInfoクラスのdoCheckメソッド内であり、CheckLoginInfo.javaの150行目にあたる。
3. doCheckメソッドを呼び出した個所はagency.attestation.AttestationServletクラスのdoGetメソッド内であり、AttestationServlet.javaの96行目にあたる。
4. doGetメソッドを呼び出した個所はagency.attestation.AttestationServletクラスのdoPostメソッド内であり、AttestationServlet.javaの161行目にあたる。
5. doPostメソッドを呼び出した個所はjavax.servlet.http.HttpServletクラスのserviceメソッド内であり、HttpServlet.javaの772行目にあたる。
6. serviceメソッドを呼び出した個所はjavax.servlet.http.HttpServletクラスのserviceメソッド内であり、HttpServlet.javaの865行目にあたる。

■解析方法

スタックトレースの先頭行のクラスが自分で作成したクラスであるならば、例外が発生した行がどうなっているのかを、ソースファイルを見て確かめることができます。
先頭行のクラスが自分で作成したクラスでないならば、例外が発生した行を見ることができないかもしれません。その場合、トレースの最も上行にある自分が作成したクラスを探し、ソースファイルを見て確認してください。発生した例外が何であるかを確認して、呼び出し個所のメソッドの引数などを確認してください。
また、スタックトレースに続けて“Caused by:”として別の例外が出力されている場合、根本原因を表していますので、こちらをあわせて確認してください。

スタックトレースで表示される例外(Exception)とErrorについては、“メッセージ集”を参照してください。

補足

- **WebアプリケーションのJSP実行時に例外が発生した場合**

  JSPはjavaファイルに変換しコンパイルして実行されるため、実行時に例外が発生した場合、スタックトレース上では以下の例のように出力されます。

  例:Webアプリケーションのルートディレクトリ(コンテキストルート)配下の以下のJSPを呼び出した場合。

  ー /subdir/sample.jsp

```
                        :
at org.apache.jsp.subdir.sample_jsp._jspService(sample_jsp.java:46)
                        :
```

固定部分“org.apache.jsp.”の後に、サブディレクトリ名、JSPに対応するクラス名、メソッド名、(対応するjavaファイル名:行番号)となります。
なお、対応するjavaファイルの格納先については“2.2.3 IJServerのファイル構成”を参照してください。

29.7 Webアプリケーションの開発・運用時の異常

IJServerを起動し、WebブラウザからWebアプリケーションを呼び出したときに発生する以下のトラブルについて、対処方法を説明します。

- Webブラウザにステータスコードやメッセージが表示される場合
  “メッセージ集”を参照して対処してください。
- JSPのコンパイルに失敗した場合
- ijscompilejspコマンドがタイムアウトする場合
- ijscompilejspコマンドで静的includeするファイルのコンパイルがエラーとなる場合
- JSPを更新したのに更新内容が反映されない場合
- ログファイルにエラーメッセージが出力される場合
- ログファイルにメッセージが出力されない場合
- エラーページとして指定したページがWebブラウザに表示されない場合
- JSP事前コンパイルでコンパイルが正常終了したJSPを含むWebアプリケーションの起動が異常終了する場合
- Webアプリケーション環境定義ファイル(deployment descriptor)の更新内容がWebアプリケーションに反映されない場合
- アプリケーションの再配備時にDEP4112が出力された場合
- 故障監視機能の異常

29.7.1 JSPのコンパイルに失敗した場合

JSPで使用しているクラスが見つからない場合

以下の可能性があります。それぞれの対処を行ってください。

- クラスパスが正しく設定されていません。
  IJServerの環境設定を見直してください。
- 無名パッケージのクラスを使用しています。
  JSPからは無名パッケージのクラスを使用することはできません。
  JSPから使用するクラスにはパッケージ名を指定してください。

クラスの設定方法については、“2.3.4 IJServerで使用するクラスの設定について”を参照してください。

JSPファイルを確認しても構文は間違っていないのに構文エラーとなる場合

【原因】
JSPファイルの文字コードと配備先のIJServerの文字コードが違っているため、JSPファイルのコンパイル時に文字化けしている可能性があります。

【対処】
以下のようにpageディレクティブのpageEncodingでJSPファイルの文字コードを指定してください。

```
<%@ page pageEncoding="文字コード" %>
```

または、以下のようにpageディレクティブのcontentTypeに指定することもできます。

```
<%@ page contentType="text/html; charset=文字コード" %>
```

上記の場合は、クライアントに返されるコンテントデータも指定された文字コードで返されます。

サイズの上限を超える場合

【現象】
以下のメッセージが出力されます。
The code of method _jspService(HttpServletRequest, HttpServletResponse) is exceeding the 65535 bytes limit

【原因】
JSPファイルのサイズが大きいため、変換したJavaファイルのメソッドのサイズが、Javaで扱える上限を超えました。

【対処】

- タグ(アクションやスクリプティング要素)の間の不要な記載(スペース、コメント、改行等)がある場合は削除してください。
- JSPファイルを分割し、動的にincludeを行ってください。

以前のバージョンのServletサービスでコンパイルできていたJSPが、移行(バージョンアップ)によりコンパイルに失敗するようになった場合

“30.1 Servletサービス(Tomcat5.5ベースのサーブレット実行環境)への移行”を参照してください。

29.7.2 ijscompilejspコマンドがタイムアウトする場合

1度に多量のJSPをJSP事前コンパイルする場合、Interstage JMXサービスの通信のタイムアウトを超える可能性があります。
その場合、以下のエラーメッセージが表示されijscompilejspコマンドが中断されます。

- IS: エラー: is20727:通信エラーが発生しました 例外情報=java.rmi.UnmarshalException 例外メッセージ=Error unmarshaling return header; nested exception is:
  java.net.SocketTimeoutException: Read timed out

この場合、ijscompilejspコマンドはエラーで復帰しますが、JSPのコンパイルはバックグラウンドで実行されています。
JSPが正常にコンパイルされたかどうかを確認するには、しばらくたってから-pオプションを指定してijscompilejspを実行してください。
正常にコンパイルが完了したJSPについては“jspファイルは更新されていません”と表示されます。
コンパイルエラーとなったJSPについては再度コンパイルが実行されますので、表示されたコンパイルエラーを取り除いてください。

注意

Interstage JMXサービスの通信タイムアウト後にijscompilejspを実行した場合、バックグラウンドで処理されているJSP事前コンパイルの完了まで、ijscompilejspの実行は待たされます。

Interstage JMXサービスの通信のタイムアウト時間は、環境定義ファイルのtimeoutタグのrmi属性で指定します。

タイムアウト時間の変更方法の詳細は、“運用ガイド(基本編)”の“Interstage管理コンソール環境のカスタマイズ”の“Interstage JMXサービスのカスタマイズ”を参照してください。

29.7.3 ijscompilejspコマンドで静的includeするファイルのコンパイルがエラーとなる場合

JSP事前コンパイルは、ファイルの拡張子でJSPかどうかを判断します。
拡張子が“jsp”のファイルをJSPと判断してコンパイルします。
したがって、静的includeするファイルの拡張子が“jsp”の場合、そのファイルもコンパイルされます。
通常、静的includeするファイルはincludeしているJSPを構成する部品のため、単体ではコンパイルエラーとなる場合がありますが、静的includeするファイルが単体で呼び出されることがない場合は、エラーは無視してかまいません。

ポイント

JSPから静的includeされるファイルの拡張子は、“jsp”ではなく“jspf”とすることを推奨します。

29.7.4 JSPを更新したのに更新内容が反映されない場合

JSPのコンパイラはJSPとコンパイル結果のタイムスタンプを比較し、JSPのタイムスタンプの方が新しい場合にのみJSPをコンパイルします。
JSPを更新した場合はJSPのタイムスタンプをマシンの現在値にあわせてください。

ポイント

ijscompilejspコマンドを使用する場合は、-aオプションを使用すれば、すべてのJSPをコンパイルできます。

29.7.5 ログファイルにエラーメッセージが出力される場合

■Webアプリケーション環境定義ファイルの誤り

Webアプリケーション環境定義ファイルの記述に誤りがある場合には、IJServerログに以下のエラーメッセージが出力されますので、記述を見直してください。たとえば、servlet-mappingタグをservletタグよりも前に定義する等です。
Webアプリケーション環境定義ファイルに関しては、“7.5 Webアプリケーション環境定義ファイル(deployment descriptor)”を参照してください。

- [ERROR] Digester - -Parse Error at line [行数] column [列数]: [原因]

29.7.6 ログファイルにメッセージが出力されない場合

テキストエディタの種類によっては、IJServerログを開いたままアプリケーションを実行した場合に、メッセージが出力されないことがあります。その場合、コンテナ情報ログ(info.log)に以下のメッセージが出力されています。

- IJServer14115: ERROR: The output of a message was not processed normally.
  Please check whether the file is opened or there is any authority of writing.

上記メッセージが出力されていた場合は、以下の対処を行ってください。

- IJServerログファイルを閉じる
- IJServerログに対して書き込み権限があるか確認する
- 他のプロセスでIJServerログファイルが使用されていないか確認する

29.7.7 エラーページとして指定したページがWebブラウザに表示されない場合

Webアプリケーション環境定義ファイルの<error-page>タグで指定したページや、JSPのerrorPage属性で指定したJSPエラーページ(isErrorPage属性に"true"を設定)では、ステータスコードは、発生した現象のステータスコードそのまま、または500が返却されます。

一部のWebブラウザでは、ステータスコード200以外のコンテンツは、Webブラウザの設定内容やコンテンツのサイズによって、Webブラウザ内蔵のエラーページが表示される場合があります。
必要に応じてWebブラウザの設定を変更してください。
または、返却するコンテンツのサイズを大きくすることにより回避できる場合があります。

29.7.8 JSP事前コンパイルでコンパイルが正常終了したJSPを含むWebアプリケーションの起動が異常終了する場合

JSP事前コンパイルでコンパイルが正常終了したJSPを含むWebアプリケーションの起動がIJServer32042のエラーで異常終了する場合は、アプリケーションの配備後にWebアプリケーション環境定義ファイル(deployment descriptor)が更新されたことによって、記述形式に誤りが発生した可能性があります。

Webアプリケーション環境定義ファイル(deployment descriptor)を正しく修正し、JSP事前コンパイルコマンドまたはアプリケーションを実行してください。
このとき、JSP事前コンパイルコマンドに-aオプションを指定してください。

29.7.9 Webアプリケーション環境定義ファイル(deployment descriptor)の更新内容がWebアプリケーションに反映されない場合

Webアプリケーション環境定義ファイル(deployment descriptor)の更新内容がWebアプリケーションに反映されない場合は、JSPのコンパイル結果に反映させるため、JSP事前コンパイルコマンドを実行してください。
このとき、JSP事前コンパイルコマンドに-aオプションを指定してください。

29.7.10 アプリケーションの再配備時にDEP4112が出力された場合

再配備を行うと、配備済みアプリケーションの配備取消が行われたあと、アプリケーションの配備が行われます。

このため、「Webサーバコネクタの制限事項」の項番1の制限により、アプリケーションの配備に失敗する場合があります。

この場合、しばらく時間をおいてから再操作するか、Webサーバの停止後に再配備操作を行ってください。

制限事項の詳細は、「使用上の注意」の「制限事項」－「既知の問題」－「Webサーバコネクタの制限事項」を参照してください。

29.7.11 故障監視機能の異常

故障監視機能を使用しているが、故障が検出されない場合がある

WebサーバコネクタとWebサーバコネクタ(Interstage HTTP Server 2.2用)で、同一のWebサーバ名を使用している可能性があります。

故障監視を行っているWebサーバ名が、WebサーバコネクタとWebサーバコネクタ(Interstage HTTP Server 2.2用)の両方で使用されている場合、正しく故障監視を行うことができません。

以下の方法で、それぞれのWebサーバ名を確認してください。

- Webサーバコネクタの場合

  [Interstage管理コンソール] > [Interstage Application Server] > [システム] > [サービス] > [Webサーバ]

- Webサーバコネクタ(Interstage HTTP Server 2.2用)の場合

  wscadmin list-web-serversサブコマンド
  list-web-serversサブコマンドについては、お使いのJava EE 6または、Java EE 7のマニュアルのwscadmin list-web-serversサブコマンドを参照してください。(注)

故障監視を行っているWebサーバ名が、WebサーバコネクタとWebサーバコネクタ(Interstage HTTP Server 2.2用)の両方で使用されている場合、Webサーバコネクタ(Interstage HTTP Server 2.2用)で使用するWebサーバ名を変更してください。

1. Webサーバ名を変更するWebサーバを停止します。

   Webサーバの停止方法については、「Interstage HTTP Server 2.2 運用ガイド」の「運用・保守」－「起動・停止」－「停止」を参照してください。

2. wscadmin list-web-serversサブコマンドの--detailオプションでWebサーバコネクタの設定を出力し、記録します。
   list-web-serversサブコマンドについては、お使いのJava EE 6または、Java EE 7のマニュアルのwscadmin list-web-serversサブコマンドを参照してください。(注)

3. wscadmin delete-envサブコマンドでWebサーバコネクタの動作環境を削除します。
   delete-envサブコマンドについては、お使いのJava EE 6または、Java EE 7のマニュアルのwscadmin delete-envサブコマンドを参照してください。(注)

4. テキストエディタなどでWebサーバ一覧ファイル(servers.conf)を開いて、変更対象のWebサーバ名を新しいWebサーバ名に変更します。

   Webサーバ一覧ファイル(servers.conf)の設定方法については、「Interstage HTTP Server 2.2 運用ガイド」の「チューニング」－「環境定義」－「Web サーバ一覧ファイル(servers.conf)」を参照してください。

5. wscadmin create-envサブコマンドで、新しいWebサーバ名を指定し、Webサーバコネクタの動作環境を作成します。
   create-envサブコマンドについては、お使いのJava EE 6または、Java EE 7のマニュアルのwscadmin create-envサブコマンドを参照してください。(注)
6. asadmin、またはwscadminコマンドを使用し、2.で記録したWebサーバコネクタの設定と同様になるように、設定を行ってください。
   asadmin、wscadminコマンドについては、お使いのJava EE 6または、Java EE 7のマニュアルを参照してください。(注)
7. Webサーバ名を変更したWebサーバを起動します。

   Webサーバの起動方法については、「Interstage HTTP Server 2.2 運用ガイド」の「運用・保守」－「起動・停止」－「起動」を参照してください。

   注）

EE

   － Java EE 6を使用している場合
   「Java EE運用ガイド(Java EE 6編)」の「Java EE 6運用コマンド」

   － Java EE 7を使用している場合
   「Java EE 7 設計・構築・運用ガイド」の「Java EE 7運用コマンド」

svmondspstatコマンドで表示されない振り分け先がある

WebサーバコネクタとWebサーバコネクタ(Interstage HTTP Server 2.2用)で、同一のWebサーバ名を使用している可能性があります。

「故障監視機能を使用しているが、故障が検出されない場合がある」を参照してください。

29.8 Webアプリケーションで文字化けが発生する場合の対処

文字化けの対処方法、および文字化けに影響するコード変換の設定について説明します。

■文字化け状況

文字化けが発生した場合、原因を追求するために文字化けが起きた状況を確認します。

- 文字化けする文字と文字化けしない文字の区別をする。
- 文字化けが発生している箇所を特定する。
- 文字化けが発生している箇所の文字を16進で表示する。

■チェックポイント

Servletサービスの運用で文字化けが発生した場合のチェックポイントを以下に示します。

1. 日本語がすべて“?”に文字化けする場合、ロケールに必要なパッケージがインストールされているか確認してください。
2. 一部の文字が文字化けする場合、ロケールの設定を確認してください。
   ロケールの設定については、“■ロケールの設定”を参照してください。
3. サーブレットアプリケーションの受取時に文字化けする場合、“Webアプリケーションのエンコーディング”(注)を見直し、正しく設定してください。
4. “Shift_JIS”と“SJIS”が混在している場合、charsetへの設定は統一してください。
   Charsetへの設定については、“■Webアプリケーションのコード系”を参照してください。
5. 特殊文字「～」などが文字化けして“?”になる場合、日本語コード(JIS、EUC、シフトJIS)をUnicodeに変換する際の変換規則が異なるために文字化けが発生している可能性がありますので、コード系は統一してください。統一できない場合は、WebアプリケーションでUnicodeから変換する直前にフィルターをかけ対処してください。

6. コード系に“JISAutoDetect”(自動変換)が設定されている場合、短い日本語文字列などはコード系の認識が正しく行われずコード変換が正しく行われないことがあります。(例として、日本語EUCの文字をShift-JISの半角カタカナだと誤認識した場合、半角カタカナを含む判読不可能な文字列になります。)

7. JDK/JRE 1.4.1からエンコーディング名「Shift_JIS」の扱いが変更されました。このため、“Webアプリケーションのエンコーディング”(注) に“Shift_JIS”を指定している場合、JDK/JRE 1.3で動作していたWebアプリケーションを動作させると文字化けが発生することがあります。
   JDK/JRE 1.3と同様に動作させるには、“■Webアプリケーションのコード系”に“windows-31j”を指定してください。

注) “Webアプリケーションのエンコーディング”は、Interstage管理コンソールの以下の項目で設定します。

- [ワークユニット] > “ワークユニット名” > [配備]タブ > [詳細設定] > [Webアプリケーション設定] > [エンコーディング]
- [ワークユニット] > “ワークユニット名” > “Webアプリケーション名” > [環境設定]タブ > [エンコーディング]

■ロケールの設定

JavaVM、javac、native2asciiなどのツールは、ロケールの設定に基づいて動作します。Servletサービスでも同様にロケールの設定に基づいて動作します。ロケールの設定方法について説明します。

【ロケール】
ロケール(Locale)とは、共有する言語や文化に依存するさまざまな属性をひとまとめにして定義しています。

Windows32/64

[コントロールパネル]の[時計、言語、および地域]でロケールを設定します。
注) Windows Server(R) 2012の場合の説明です。使用するOSにより操作方法は異なる場合があります。

Solaris64 Linux32/64

環境変数LANGにロケールを設定します。

コード系	ロケール
EUC	ja ja_JP.eucJP
シフトJIS	ja_JP.PCK
UTF-8	ja_JP.UTF-8

■Webアプリケーションのコード系

Webアプリケーションのサーブレットは Java言語で作成するため、内部で処理する日本語コード系はUNICODEです。そこで、Webブラウザから入力された日本語文字はUNICODEへ、Webブラウザへ表示する文字はUNICODEから日本語のコードに変換する必要があります。
JSPもJavaファイルに変換され実行されるので、コード変換の設定が必要となります。

コード変換の設定方法を説明します。

サーブレット

[HTMLのコード系]
“charset”を設定します。

記述例(Shift_JIS)

```
<meta http-equiv="Content-type" content="text/html; charset=Shift_JIS">
```

[入力コード系]
ブラウザからの受信時にコンバータを使用し、文字変換を行います。

記述例(SJIS→Unicode)

```
request.setCharacterEncoding("SJIS");
String in_value1 = request.getParameter("param1");
```

Interstage管理コンソールで“Webアプリケーションのエンコーディング”項目を指定している場合は、コード変換を行う必要はありません。

[出力コード系]
処理結果をWebブラウザに返す時、文字変換を行うためにコンバータ指定します。

記述例(Unicode→SJIS)

```
response.setContentType("text/html; charset=SJIS");
PrintWriter out = response.getWriter();
```

setContentType() はgetWriter() を取得する前に呼び出す必要があります。

注意

- コード系に存在しない文字がある場合、コード変換が正しく行われないことがあります。
- HTMLファイルのコード系、入力コード系、出力コード系を統一していない場合、文字化けの原因になります。
- 入力コード系に“JISAutoDetect”(自動変換)が設定されている場合、コード変換が正しく行われないことがあります。HTMLファイルのコード系と同じコード系を設定してください。

JSP

- <%@ page>タグのcharset属性では、JSPを記述したコード系を指定します。日本語コードを扱うためには、charset属性は必ず指定してください。
- 出力コード系は、“<%”と“%>”の中でsetContentTypeメソッドを使用して指定します。JSPを記述したコード系と同じときには、省略することができます。
- コード系の“charset”に設定する文字コードセットについては、JDKのドキュメントを参照してください。JDKのバージョンで異なりますが、EUC_JP、SJIS、ISO-2022-JP(ISO2022JP)などを日本語コード系として指定することができます。

記述例

```
<%@ page session="false" contentType="text/html; charset=SJIS" %>
<%
response.setContentType("text/html; charset=EUC_JP");
%>
<HTML>
<TITLE>日本語テスト(EUC_JP 出力)</TITLE>
<BODY BGCOLOR="black" TEXT="yellow">
<H1>日本語テスト(EUC_JP 出力)<BR></H1>
```

■文字化け特殊文字

以下に、文字化け特殊文字の対応表を掲載します。

JIS X 0208		Unicodeへの変換
33 (‘～’)	EUC:a1c1	Solaris上での変換: U+301c (WAVE DASH)
	SJIS:8160	Windows(R)上での変換: U+ff5e (FULLWIDTH TILDE)
01-34 (‘∥’)	EUC:a1c2	Solaris上での変換: U+2016 (DOUBLE VERTICAL LINE)

JIS X 0208		Unicodeへの変換
	SJIS:8161	Windows(R)上での変換: U+2225 (PARALLEL TO)
01-61 (‘－’)	EUC:a1dd	Solaris上での変換: U+2212 (MINUS SIGN)
	SJIS:817c	Windows(R)上での変換: U+ff0d (FULLWIDTH HYPHEN-MINUS)
01-81 (‘￠’)	EUC:a1f1	Solaris上での変換: U+00a2 (CENT SIGN)
	SJIS:8191	Windows(R)上での変換: U+ffe0 (FULLWIDTH CENT SIGN)
01-82 (‘￡’)	EUC:a1f2	Solaris上での変換: U+00a3 (POUND SIGN)
	SJIS:8192	Windows(R)上での変換: U+ffe1 (FULLWIDTH POUND SIGN)
02-44 (‘￢’)	EUC:a2cc	Solaris上での変換: U+00ac (NOT SIGN)
	SJIS:81ca	Windows(R)上での変換: U+ffe2 (FULLWIDTH NOT SIGN)

29.9 EJBサービス使用時の異常

EJBサービスで異常が発生した場合の対処方法について説明します。

■異常時の対処方法

1. IJServerのログを参照する

   EJBサービスで異常が発生した場合、IJServerのログに発生した例外のスタックトレースやコンテナのエラーメッセージを出力します。
   IJServerのログについては、“3.12 アプリケーションのデバッグ”を参照してください。

2. システムログを参照する

   ワークユニットで異常が発生した場合、異常が発生した際の情報をシステムログに出力します。EJBアプリケーションの起動時、運用時、および停止時にワークユニットで異常が発生した場合には、システムログに出力される、異常の種別や異常を伝えるメッセージを参照してください。

3. EJBアプリケーションの異常情報を取得する(デバッグ情報取得)

   EJBアプリケーションが異常発生時と異なるExceptionをThrowした場合、IJServerのログファイルに出力されているスタックトレース情報に、EJBアプリケーションで発生した異常が直接出力されない場合があります。EJBアプリケーションで異常が発生し、コンソール情報を参照しても異常の原因が判明しない場合は、EJBアプリケーションで発生している異常情報の取得を行い対処してください。EJBアプリケーションの異常情報の取得方法については、“3.12 アプリケーションのデバッグ”を参照してください。

4. スレッドダンプを参照する

   スレッドダンプは、以下の契機でコンテナ情報ログ(info.log)に出力されます。

   ― 起動時のタイムアウト

   ― アプリケーションのタイムアウト

   ― 停止時のタイムアウト

   スレッドダンプ出力についてのは、“3.12.3 スレッドダンプ自動採取”を参照してください。

29.9.1 クライアントアプリケーションの異常

クライアントアプリケーションが異常終了したりループ状態になったりした場合には、原因を取り除いた後、クライアントアプリケーションを再起動してください。また、例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

javaまたはjreコマンドが見つからない場合

javaまたはjreコマンドが見つからない場合は、Interstage Application Serverでインストールされ たJDKがインストールされ、正しく設定されていることを確認してください。

クライアントからのメソッド呼出しで例外が発生した場合

- クライアントからのcreate()メソッドでNO_IMPLEMENT例外が発生した場合は、当該IJServerが起動されていることを確認してください。
  また、ロードバランスを使用して運用を行っている場合は、ロードバランスの設定が正しいかどうか確認してください。
- クライアントからのcreate()メソッド、またはビジネスメソッドで、BAD_OPERATION例外が発生した場合は、当該EJBアプリケーションのHomeインタフェース、またはRemoteインタフェースが変更されている可能性があります。当該EJBアプリケーションのHomeインタフェース、またはRemoteインタフェースを確認し、再度クライアントアプリケーションの構築を行ってください。

アプレットのlookup処理で例外が発生した場合

クライアントアプリケーションをJavaアプレットとして実行し、lookup()処理でorg.omg.CORBA.INITIALIZE例外が発生した場合、デジタル署名の設定および、JBKプラグインが使用するポリシーファイルが正しく設定されているか確認してください。詳細は、以下を参照してください。

- “15.6 Javaアプレットを使用する場合(プレインストール型Javaライブラリ)”
- “15.7 Javaアプレットを使用する場合(Portable-ORB)”
- “メッセージ集”の“J2EE使用時に出力される例外情報”
- “15.8 アプレットのデジタル署名”
- Interstage Studioの“J Business Kit オンラインマニュアル”

その他の例外

- ロードバランス機能を利用し、クライアントアプリケーションで、“java.lang.ClassCastException”例外が発生した場合は、OD_or_admコマンド、またはodadministerlbコマンドで設定したインタフェースリポジトリIDが正しいか確認してください。
- 例外の詳細情報にCORBAサービスの例外情報とマイナーコードが出力された場合は、CORBAサービスでエラーが発生している可能性があります。
  EJBアプリケーションで発生するCORBAメッセージについては“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

Windows32/64 Linux32/64

- 分散トランザクション機能を利用している場合は、データベース連携サービスで異常が発生している可能性があります。データベース連携サービスで発生した異常時の対処方法については、“トラブルシューティング集”－“データベース連携サービス使用時の異常”を参照してください。

Solaris64

- Interstageがインストールされているサーバマシン上でクライアントアプリケーションを動作させ、“*** panic : libthread loaded into green threads”が発生した場合は、環境変数(THREADS_FLAG)が正しく設定されているか確認してください。クライアントの環境設定についての詳細は、“4.2 EJBを参照する場合の環境設定”を参照してください。

- クライアントアプリケーションからEJBアプリケーションのメソッドを呼び出したにもかかわらず、EJBアプリケーションからの応答がない場合は、以下の点を確認してください。

  ー クライアントアプリケーションとEJBアプリケーションで共通のクラスを使用するようなインタフェースとしている場合、同一のクラスファイルを使用しているか(ファイルの作成日付、サイズなどで確認してください)。

- “ERROR! Shared library ioser12 could not be found.”が発生した場合、JDKがインストールされ、環境変数が正しく設定されていることを確認してください。

29.9.2 EJBアプリケーションの異常

EJBアプリケーションが異常終了した場合には、原因を取り除いた後、IJServerを再起動してください。
また、例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

EJBアプリケーションが待機状態またはループ状態になった場合

EJBアプリケーションが待機状態またはループ状態になった場合は、以下の対処を行ってください。

- EJBアプリケーションで、トランザクションの終了漏れがないか確認してください。終了漏れがある場合は、トランザクションを終了するようEJBアプリケーションを修正してください。
- EJBアプリケーションが無限ループしていないか確認してください。無限ループしている場合は、EJBアプリケーションを修正してください。
- Windows32/64 Linux32/64
  分散トランザクション機能を利用している場合は、データベース連携サービスで異常が発生している可能性があります。
  データベース連携サービスで発生した異常時の対処方法については、“トラブルシューティング集”ー“データベース連携サービス使用時の異常”を参照してください。

また、クライアントアプリケーションには、タイムアウトなどのエラーが通知されますので、エラーが通知されたクライアントアプリケーションを終了させてください。

EJBアプリケーション内で、“javax.swing”、“java.awt”を使用している場合

EJBアプリケーションを動作させた時、以下の例外がコンテナログに出力された場合は、EJBアプリケーション内で、“javax.swing”または“java.awt”を使用している可能性があります。EJBアプリケーション内で、“javax.swing”または“java.awt”を使用しないように修正してください。

- java.lang.InternalError: Can't connect to X11 window server using ':0.0' as the value of the DISPLAY variable.

deployment descriptorに参照情報が記述されていない場合

以下の例外がコンテナログに出力されることがありますが、問題ありません。

- com.fujitsu.interstage.j2ee.jndi.resource.FJNotFoundDefinitionException

Message-driven Bean運用時の異常

Message-driven Beanの運用中にInterstage JMSまたはイベントサービスで異常が発生した場合には、“29.12 Interstage JMSの異常時の対処”、または“トラブルシューティング集”ー“イベントサービス運用時の異常”を参照して対処してください。
また、Message-driven BeanをIJServerで運用中にIJServerの強制停止をした場合、再度IJServerを起動してもメッセージが受信できないなどの異常が発生する場合があります。この場合、イベントチャネルに接続情報が残った可能性がありますので、同様に“29.12 Interstage JMSの異常時の対処”、または“トラブルシューティング集”ー“イベントサービス運用時の異常”を参照して対処し、IJServerを再起動してください。

なお、Publisher/Subscriberモデルで、トランザクション属性がRequired、メッセージセレクタ機能を使用するMessage-driven Beanの運用中に、以下のメッセージが出力された場合は、メッセージセレクタの条件に合致しないメッセージを送信している

可能性があります。
この場合Message-driven Beanの動作への影響はないため、特に対処を行う必要はありません。

- com.fujitsu.interstage.ejb.container.messagedriven.FJMDBNullMessageException: "null message from receiveNoWait()"

イベントチャネルが停止または再起動した場合、Message-driven Beanのメッセージ受信処理が停止します。メッセージ受信停止後、メッセージ受信を再開するにはIJServerを再起動させる必要があります。イベントチャネル起動後に、IJServerを再起動してください。Message-driven Beanがメッセージを受信可能な状態かどうかは、Interstage管理コンソールの以下の画面の「メッセージ受信状態」を参照することにより確認できます。

- [ワークユニット] > [IJServer名] > [モジュール名] > [EJBアプリケーション名] > モニタ

connector使用時の異常

resource adapterで例外が発生した可能性があります。IJServerのコンテナログを参照して、原因を取り除いてください。
IJServerのコンテナログのディレクトリは、Interstage管理コンソールまたはisj2eeadminコマンドで変更可能です。
デフォルトのディレクトリは、以下です。

Windows32/64

J2EE共通ディレクトリ¥ijserver¥[IJServer名]¥log¥プロセス通番¥container.log

Solaris64 Linux32/64

J2EE共通ディレクトリ/ijserver/[IJServer名]/log/プロセス通番/container.log

注意事項

EJBアプリケーションが待ち状態またはループ状態になった場合、IJServerを停止することはできません。
この状態で当該IJServerを停止しようとすると、停止処理は行われず、実行結果はコンソールに出力されません。このように、起動しているIJServerに対して実行結果がシステムログに出力されない場合は、IJServerのプロセスを停止してください。

29.9.3 EJBアプリケーション呼び出し時の異常

lookupやnarrowなどでEJBアプリケーションを呼び出す際に異常が発生した場合、以下の例外が出力されることがあります。

- java.lang.ClassCastException
- java.lang.ClassNotFoundException
- java.lang.NoClassDefFoundError

例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”に記述されている各例外情報や“lookup処理で例外が発生した場合の対処”を参照してください。

29.9.4 JavaVMの異常

EJBアプリケーション実行中またはクライアントアプリケーション実行中にjreコマンドまたはjavaコマンドで異常が発生した場合、以下について調査してください。

- 使用するJDBCドライバのバージョンがJavaのバージョンと異なっている可能性があります。JDBCドライバのマニュアルを参照し、Javaのバージョンと一致しているか確認してください。
- 標準出力または標準エラー出力により、異常が発生している箇所を特定してください。該当箇所がJavaのAPIである場合は、オラクル社のホームページからJavaVMの障害情報を参照し、対処してください。該当する障害情報が見つからない場合、または対処方法が不明な場合は、技術員に連絡してください。

29.9.5 Symfowareの強制終了

Symfowareを強制終了すると、Symfowareに接続中のIJServerも、Symfowareによって強制的にプロセスが停止されます。
この結果、EXTP4703メッセージやod10301メッセージが出力されることがあります。
Symfowareの強制終了の詳細は、Symfowareのマニュアルを参照してください。

29.9.6 アプリケーション連携中の通信回線異常

クライアントアプリケーションとEJBアプリケーション間で通信異常が発生した場合は、クライアントアプリケーションからのオペレーション呼出しの復帰値により、エラーが通知されます。
通信回線の回復後、クライアントアプリケーションを再起動してください。

29.9.7 エラーメッセージが通知された場合

EJBのエラーメッセージが通知された場合

業務システム運用中に、システムログまたはIJServerのログファイルにIJServer2またはEJBで始まるエラーメッセージが通知された場合は、“メッセージ集”の“メッセージ番号がIJServer2で始まるメッセージ”、または“メッセージ番号がEJBで始まるメッセージ”を参照し、対処を行ってください。

Windows32/64 Linux32/64
分散トランザクション機能を利用している場合、データベース連携サービスのメッセージが出力されている場合があります。
“メッセージ集”の“メッセージ番号がOTSで始まるメッセージ”を参照し、対処を行ってください。

DBMSのメッセージが出力されている場合

DBMSの運用で異常が発生してDBMSのメッセージが出力されている場合は、DBMSのマニュアルにあるメッセージの説明を参照し、対処を行ってください。

Interstageの運用で異常が発生している場合

Interstageの運用で異常が発生している場合、Interstageのメッセージが出力されている場合があります。
ラベルには、以下のInterstageのラベルか、Interstageのコマンド名が出力されます。

もし、該当するメッセージが出力されている場合は、“メッセージ集”を参照して対処を行ってください。

ラベル	
OD	CORBAサービス(ObjectDirector)が出力するラベル
IS	Interstageが出力するメッセージラベル

29.9.8 システムのメモリ不足

EJBサービスを運用中に、“メモリ不足が発生しました”というメッセージが表示されることがあります。通常は、しばらくしてから再度処理を実行してください。
このようなメッセージが、頻繁に出力される場合は、“チューニングガイド”を参考に、EJBサービスを運用するために必要なメモリ量を再見積もりし、メモリが十分に用意されているか確認してください。

EJBサービスの必要とするメモリ量に対して、メモリが十分に用意されている場合には、他のアプリケーションの使用メモリ量が不足していることが考えられます。同一マシン上で運用している他のアプリケーションについても再見積もりを実施し、メモリ量が適当か調査してください。

再見積もりの結果、メモリ量が不足していることが確認された場合には、システム管理者に連絡してください。システム管理者は、メインメモリの増設または、Windows(R)の場合は仮想メモリのページファイルの拡張を、Solaris/Linuxの場合はスワップ領域の拡張を行ってください。
また、Entity Beanを利用したEJBアプリケーションの場合、EJBサービスが提供するトランザクション機能で、UserTransaction範囲内でEntity Beanを呼び出しているか確認してください。

29.9.9 ライブラリが見つからない場合

Windows32/64

“ダイナミックリンクライブラリが指定されたパスに見つからない”というメッセージが出力された場合には、以下の原因が考えられます。

- EJBサービスの関連製品がインストールされていない可能性があります。“システム設計ガイド”の“ソフトウェア条件”を参照し、関連製品がインストールされているか確認してください。
- EJBサービスが動作するために必要な、環境変数が設定されていない可能性があります。“4.2 EJBを参照する場合の環境設定”を参照し、環境変数が設定されているか確認してください。
- EJBサービスが動作するために必要な環境変数の設定と、インストールされているJDK/JREのバージョンが合っていない可能性があります。“4.2 EJBを参照する場合の環境設定”を参照し、環境変数の設定と、JDK/JREのバージョンが合っているかどうかを確認してください。
  なお、この場合、以下のダイナミックリンクライブラリが見つからないというメッセージが出力されます。

  ー javai.dll

Solaris64 Linux32/64

“ファイルもディレクトリもありません”というメッセージが出力された場合には、以下の原因が考えられます。

- EJBサービスの関連製品がインストールされていない可能性があります。“システム設計ガイド”の“ソフトウェア条件”を参照し、関連製品がインストールされているか確認してください。
- EJBサービスが動作するために必要な、環境変数が設定されていない可能性があります。“4.2 EJBを参照する場合の環境設定”を参照し、環境変数が設定されているか確認してください。
- EJBサービスが動作するために必要な環境変数の設定と、インストールされているJDK/JREのバージョンが合っていない可能性があります。“4.2 EJBを参照する場合の環境設定”を参照し、環境変数の設定と、JDK/JREのバージョンが合っているかどうか確認してください。
  なお、この場合、以下の共用ライブラリが見つからないというメッセージが出力されます。

  ー libjava.so

29.9.10 定義ファイルが更新できない場合

Interstageがインストールされているディスクの空き容量が不足すると、以下の操作時に、EJBサービスの定義ファイルが更新できず、エラーメッセージが出力されます。

- Interstage管理コンソールのヘルプを参照するときに使用するブラウザのパスの設定
- Interstage管理コンソールにおける定義の追加または更新

エラーメッセージが出力されたら、処理を直ちに中断し、ディスクの空き容量を増やすようにしてください。ディスクの空き容量を増やさずに再度上記の操作を行うと、定義ファイルが壊れることがあります。
万一、EJBサービスの定義ファイルが壊れた場合は、ディスクのI/O障害に備えて取得してあるバックアップから復旧するなどしてください。

29.9.11 デッドロックが発生する場合

Entity Beanを利用して複数件のレコードを検索して更新する処理を複数端末から同時に実行すると、デッドロックが発生することがあります。
この場合は、プライマリキーを検索するSELECT文にFOR UPDATEを指定するようにしてください。

BMPの場合はEJBアプリケーションのejbFind<METHOD>メソッドもしくはejbLoadメソッドで、CMP1.1の場合はCMP定義のfinder定義に指定します。
CMP2.xの場合は、CMFマッピング定義の同時更新データの一貫性保証を選択します。
またFOR UPDATE句を指定、および、アプリケーションが1トランザクション内で複数のテーブルにアクセスするなどの場合、シーケンスによっては、複数端末からの同時実行によりデッドロックエラーが発生することがあります。
この場合、アプリケーションを見直し、デッドロックエラー発生時にリトライを行うようなアプリケーションに変更するか、またはアプリケーション内でトランザクションの排他を取るようにアプリケーションを変更してください。
FOR UPDATEを提供していないデータベースを使用する場合は、データベースが提供する代替の排他機能を使用するか、アプリケーション側でデッドロックエラー発生時にリトライするか、またはトランザクションの排他を行ってください。

例

- BMPの場合
  SELECT ID FROM SAMPLESCM.SAMPLETBL WHERE ID > ? FOR UPDATE
- CMP1.1の場合
  WHERE @ID > ?param1? FOR UPDATE

29.9.12 Javaアプレットの異常

Javaアプレットが何らかの異常で起動できない場合の対処方法を説明します。
なお、以下に記述した異常の他に“29.9.1 クライアントアプリケーションの異常”に該当する項目がある場合があります。以下に記述した異常に該当する項目がない場合は“29.9.1 クライアントアプリケーションの異常”を参照してください。
また、Javaコンソールに表示される例外情報については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

■Javaコンソールの表示

JBKプラグインが使用するポリシーファイル(jbkplugin.properties)に、Javaコンソールを表示するように指定してください。

例

設定例

```
jbk.plugin.console.visible=true
```

29.9.13 データベースを使用したときの異常

データベースを使用した場合に以下の異常が発生した場合は、それぞれの説明に従ってください。

Symfowareを使用している場合

メッセージに以下の文字列が出力されている場合は、データベースのエラーが発生しています。Symfowareの“RDB メッセージリファレンス”を参照して対処してください。

- JYPXXXXE　('X'は数字)

また、別サーバにあるSymfowareを使用時(RDB2_TCP接続時)に例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

Oracleを使用している場合

メッセージに以下の文字列が出力されている場合は、データベースのエラーが発生しています。Oracleの“エラーメッセージ”を参照して対処してください。

- ORA-XXXXX　('X'は数字)

また、Oracleデータベースのバージョンとアプリケーションが使用しているJDBCドライバのバージョンが異なる場合、正常にデータが挿入できない等の現象が発生することがあります。
使用するJDBCドライバは、必ずOracleデータベースと同一のバージョンを使用してください。

SQL Serverを使用している場合

SQLServer使用時に例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

Microsoft(R) JDBCドライバでSQL Serverに接続する場合、推奨されていないソフトウェアを使用すると例外情報が文字化けして、正しく表示できないことがあります。“システム設計ガイド”の“ソフトウェア条件”を参照して、推奨しているソフトウェアを使用してください。

例

```
java.sql.SQLException: [Microsoft][SQLServer 2000 Driver for JDBC]??????????????????????
at com.microsoft.jdbc.base.BaseExceptions.createException(Unknown Source)
```

JDBCドライバロギング機能

Microsoft(R) JDBCドライバでは、JDKが提供するjava.util.loggingパッケージのロギング機能を利用してデバッグができます。
JDBCドライバのログを出力したい場合は、JDBCドライバのドキュメントを参照してください。
デフォルトの標準出力、標準エラー出力はコンテナログに出力されます。

注意

Interstageのユーザスナップ情報の出力でも、JDKが提供するjava.util.loggingパッケージのロギング機能を利用しています。
そのため、ユーザスナップ情報を出力すると、デフォルトではJDBCドライバのログも出力されます。
JDBCドライバのログを抑止したい場合は、以下の定義を行ってください。

定義項目	定義内容
定義ファイル格納ディレクトリ	Windows32/64 C:¥Interstage¥EJB¥etc Solaris64 Linux32/64 /opt/FJSVejb/etc
定義ファイル名	FJlogging.properties
追加定義	com.microsoft.sqlserver.jdbc.level = OFF

PostgreSQLを使用している場合 Windows32/64 Linux32/64

PostgreSQL使用時に例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。
なお、下記のメッセージがIJServerのログファイルに出力される場合は、それぞれの対処を実施してください。

- A connection error has occurred: FATAL: No pg_hba.conf entry for host ※ホストIPアドレス, user postgres, database template

  PostgreSQLサーバのpg_hba.confファイルでInterstageサーバのホストIPアドレスのアクセスが許可されていません。詳しくはPostgreSQLのマニュアルを参考してください。

- A connection error has occurred: FATAL: Password authentication failed for user "postgres"

  リソース定義のパスワードを確認してください。

- Backend start-up failed: FATAL: Database "※データベース名" does not exist in the system catalog

  リソース定義のデータベース名を確認してください。

- Backend start-up failed: FATAL: user "postgres1" does not exist

  リソース定義のユーザIDを確認してください。

- Connection refused. Check that the hostname and port are correct and that the postmaster is accepting TCP/IP connections

  リソース定義のポート番号、またはPostgreSQLが起動しているかを確認してください。

・ The connection attempt failed because Exception: java.net.NoRouteToHostException: No route to host.
または
The connection attempt failed because Exception: java.net.NoRouteToHostException: ホストへの経路がありません

リソース定義のホスト名を確認してください。

29.9.14 分散トランザクション機能使用時の異常 Windows32/64 Linux32/64

Entity Beanの最適化処理を行ったアプリケーションで分散トランザクション機能を使用している可能性があります。最適化処理を行ったアプリケーションでは、分散トランザクション機能を使用することはできません。使用した場合、以下のすべての条件を満たすときにSQLException(“ORA-01002:フェッチ順序が無効です。”)が返却されます。

・ 復帰値がEnumerationまたはCollectionのfinderメソッドを実行

・ 復帰値Enumerationに対してnextElementまたは復帰値CollectionのIteratorに対してnextをメソッドを実行

・ Oracleで一度にフェッチされるレコード数(デフォルト値 10)分だけnextElementまたはnextを実行

上記に該当しない場合で、分散トランザクション機能を使用しているときに異常が発生した場合は、“トラブルシューティング集”－“データベース連携サービス使用時の異常”を参照して対処を行ってください。

29.9.15 IPCOMと連携したときの異常

IPCOM連携は、以下の製品で利用可能です。

・ Interstage Application Server Enterprise Edition

EJBアプリケーションの配備、またはIJServerの更新が失敗した場合、詳細なメッセージがInterstage管理コンソールに出力されます。
環境設定の完了したあとで、負荷分散されないなどの問題が発生した場合には、以下の情報を確認してください。

■環境設定の確認

IJServerの設定は、Interstage管理コンソールの[ワークユニット]のEJBコンテナ設定の画面で「IPCOMのメソッド負荷分散」の設定を確認してください。
設定に誤りがある場合は、Interstage管理コンソールから設定値を修正したあと、更新して下さい。
また、isj2eeadminコマンドを使用して設定することもできます。isj2eeadminコマンドについて詳細は“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照してください。

オブジェクトリファレンスの詳細情報の表示

「IPCOMのメソッド負荷分散」の設定値が正しい場合は、ネーミングサービスに登録されたオブジェクトリファレンスの情報について確認してください。
オブジェクトリファレンスの情報の表示は、odlistnsコマンドを使用します。

odlistns -lコマンドを実行して、負荷分散対象のEJBアプリケーションのホスト名とポート番号を確認します。

以下に、オブジェクトリファレンスの詳細情報について説明します。

```
Name(Type)       Object information(detail)
                 Default object information(detail)
EasyBean(o)      RMI:pkgTest.EasyBeanHome:0000000000000000,
  (a)
IDL:com.fujitsu.interstage.j2ee.ijserver/IJServer001:1.0,
(SVv:8002:1.1:UNICODE(UCS2))
 (b)  (c)

(a)EJBアプリケーション名
(b)仮想ホスト名
(c)代表ポート番号
```

仮想ホスト名、代表ポート番号がInterstage管理コンソールまたはisj2eeadminで設定した値になっていない場合は、「環境設定」を更新して下さい。

■EJBアプリケーション

以下のようなクライアントEJBアプリケーションでは、負荷分散されません。

- createメソッドを1回実行して、その後の処理は同一EJB objectに対して処理を実行する。

したがって、新規に処理を実行する場合は、createメソッドから再実行し、処理が分散されるようにプログラミングしてください。

負荷分散されるタイミングについては、“高信頼性システム運用ガイド”の“J2EEアプリケーション”の“プログラミング方法”を参照してください。

29.9.16 ログファイルにメッセージが出力されない場合

テキストエディタの種類によっては、IJServerログを開いたままアプリケーションを実行した場合に、メッセージが出力されないことがあります。その場合、コンテナ情報ログ(info.log)に以下のメッセージが出力されています。

- IJServer14115: ERROR: The output of a message was not processed normally.
  Please check whether the file is opened or there is any authority of writing.

上記メッセージが出力されていた場合は、以下の対処を行ってください。

- IJServerログファイルを閉じる
- IJServerログに対して書き込み権限があるか確認する
- 他のプロセスでIJServerログファイルが使用されていないか確認する

29.9.17 EJBアプリケーションの配備時の異常

HomeインタフェースまたはRemoteインタフェースに宣言されているメソッドのthrows句に例外が多数ある場合、以下のIJServerタイプにEJBアプリケーションを配備すると、時間がかかり応答がない状態になります。

- EJBアプリケーションのみ運用
- WebアプリケーションとEJBアプリケーションを別JavaVMで運用

例外クラスの親クラスを作成し、throws句には親クラスのみ宣言するなどの修正を行い、throws句の例外の数を5以下にしてください。

上記条件に該当し、配備が完了しない場合は、以下の操作を行ってください。

1. Interstage JMXサービスを停止してください。
2. 以下のディレクトリおよびその配下を削除してください。

   Windows32/64

   ― C:¥Interstage¥J2EE¥var¥isdeploy¥ebxxxx (xxxxは数字)

   ― J2EE共通ディレクトリ¥ijserver¥<IJServer名>¥apps¥<配備モジュール名> (注)

   ― J2EE共通ディレクトリ¥deployment¥ijserver¥WU001¥distribute¥<配備モジュール名> (注)

   Solaris64 Linux32/64

   ― /opt/FJSVj2ee/var/isdeploy/ebxxxx (xxxxは数字)

   ― J2EE共通ディレクトリ/ijserver/<IJServer名>/apps/<配備モジュール名> (注)

   ― J2EE共通ディレクトリ/deployment/ijserver/WU001/distribute/<配備モジュール名> (注)

   注) 新規配備のみです。上書き配備の場合は削除しないでください。
3. Interstage JMXサービスを起動してください。
4. throws句を修正したアプリケーションを再度配備してください。

29.9.18 EJBタイマーサービス使用時の異常

EJBタイマーサービスで生成したタイマーが動作しない場合、タイマーがキャンセルされている可能性があります。
EJBタイマーの有効範囲については“10.6 EJBタイマーサービス”を参照してください。

29.10 Webサービスの異常

29.10.1 Webサービスクライアントログファイルが正常に出力されない場合

異なるプロセスのWebサービスクライアントが同一ファイルにログを出力する設定となっている場合、ファイルのローテーションに失敗したり、出力メッセージが乱れるなど正常にログが出力されない場合があります。
Webサービスクライアントログの出力先は、プロセスごとに異なるファイルを指定してください。
Webサービスクライアントログの出力先の設定方法は、以下を参照してください。

- “19.3 Webサービス設定ファイル”

29.10.2 8.0.0からの移行時の注意

8.0.0の一部のプラットフォームのWebサービスから移行した場合に発生しうるトラブルについて説明します。

スタブを生成し直すと、構造体型・Bean型クラスのコンストラクタの引数の順番が変わっている(iswsgen wsdlコマンドで生成されるWSDLでcomplexTypeのメンバの順番が変わる)

8.0.0の一部のプラットフォームにおいて、iswsgen wsdlコマンドがcomplexType内のelementを、開発環境・実行環境などの条件に影響を受ける不定の順番で出力していました。
今版では環境などに影響を受けない特定の順番で出力するため、生成されるWSDLは8.0.0の一部のプラットフォームと差異があります。そのWSDLを使用してスタブを生成しなおすと本問題が発生します。

対応方法

生成し直したスタブにあわせて、アプリケーションを修正してください。

なお、以下の方法により以前と同様の不定順序のWSDLを生成できますが、上記の対応方法で移行を行うことを強く推奨します。

1. 下記の内容のファイルを作成する
   ファイルの内容
   com.fujitsu.interstage.isws.utils.fields.nosort=true
2. 環境変数ISWSGEN_JAVA_OPTに以下のように1.のパスを指定する
   環境変数ISWSGEN_JAVA_OPTの設定値(ファイルのパスが「/tmp/iswsgen.properties」の場合)
   -Dcom.fujitsu.interstage.isws.configuration=/tmp/iswsgen.properties

サーバから返信の通信データの中で、complexType内のelementの順番が変わっている

8.0.0の一部のプラットフォームにおいて、サーバから返信の通信データの中のcomplexType内のelementが、開発環境・実行環境などの条件に影響を受ける不定の順番で出力されていました。
今版では環境などに影響を受けない特定の順番で出力するため、8.0.0の一部のプラットフォームとサーバからの返信の通信データに差異があります。
これにより実際の通信において、complexType内のelementが、WSDLにおける定義と一致しない順番でサーバから返信される場合があります。
ただし、iswsgen serverコマンドを使用して開発したWebサービスアプリケーションでは本問題はありません。

対応方法

WSDLにおける定義と、実際のサーバからの返信データが一致しない場合、WSDLを本プラットフォームで生成し直してください。

なお、以下の方法で、以前と同様の不定順序でのサーバからの返信を行えますが、上記の対応方法で移行を行うことを強く推奨します。

該当のIJServerに対して、Webサービス設定ファイルを指定して以下の項目を設定してください。
Webサービス設定ファイルについては、“第19章 Webサービスの運用”を参照してください。

設定項目:

Webサービスアプリケーション(iswsgen serverコマンドで生成したものを除きます)からの返信において、構造体・Bean型のメンバを特定の順序にソートせずに送信します。

key名:

com.fujitsu.interstage.isws.utils.fields.nosort

指定可能な値:

true

省略時:

構造体・Bean型のメンバを特定の順序にソートして送信します。

サーバにおいて、継承関係にある構造体型・Bean型の子クラス側のメンバが無視される

8.0.0の一部のプラットフォームにおいて、サーバで複数の構造体型・Bean型が継承関係にあった場合、親クラス側に同名のpublicフィールドが存在するにも関わらず、子クラス側のメンバが不正に利用(リクエストのデータが親クラス側のpublicフィールドではなく子クラス側のメンバに反映される・子クラス側のメンバがレスポンスのデータに反映される)される場合がありました。今版では、親クラス側に同名のpublicフィールドが存在する場合、正しく親クラス側のpublicフィールドが利用されます。

対応方法

子クラス側のメンバではなく、親クラス側のフィールドを使用するようにアプリケーションを修正してください。

iswsgen serverおよびiswsgen clientコマンドでスタブなどを再生成すると、メソッドの引数および返り値のクラスが変わっている

8.0.0の一部のプラットフォームにおいて、iswsgen serverおよびiswsgen clientコマンドが、特定の条件に該当するWSDLを入力とした場合、インタフェースが誤ったスタブやサービスエンドポイントクラスおよび関連クラスのソースを生成する場合がありました。今版では正しい生成物を生成するため、生成されるJavaソースのインタフェースが8.0.0の一部のプラットフォームと差異がある場合があります。

対応方法

生成し直したスタブなどにあわせて、アプリケーションを修正してください。

移行前の環境で作成したWebサービスのWARファイルを配備すると、配備がエラー(isws16021)になる、または、リクエスト受信時にエラー(isws11003)となる

上記の「iswsgen serverおよびiswsgen clientコマンドでスタブなどを再生成すると、メソッドの引数および返り値のクラスが変わっている」の問題による誤ったJavaインタフェースを含むWARファイルを、今版の環境に配備すると、インタフェースの不整合が発生し、配備に失敗する、または、リクエストの受信(解析)に失敗する場合があります。

対応方法

今版の環境において、iswsgen serverコマンドで、サービスエンドポイントインタフェースなどを再生成してください。再生成された資材は、上記の「iswsgen serverおよびiswsgen clientコマンドでスタブなどを再生成すると、メソッドの引数および返り値のクラスが変わっている」に該当しますので、必要に応じて対応してください。

iswsgen serverコマンドで、特定のWSDLを入力とした場合にエラー(isws11254またはisws11251)になる

iswsgen serverコマンドにチェック機能が追加されたため、特定の条件に該当し、正しい通信が行うことができない定義内容のWSDLを指定した場合にエラー(isws11254またはisws11251)になります。

条件、および対処方法は、“メッセージ集”を参照してください。

WSDLで定義したユーザ定義型の要素について、SOAPメッセージ上でxsi:type属性の値が変わっている

8.0.0の一部のプラットフォームにおいて、XMLSchemaのsimpleTypeから派生させたユーザ定義型を使用している場合、または、FaultDetailの要素についてWSDLのmessageのpartからelement属性を使用してXMLSchemaの要素定義を参照している場合において、xsi:type属性に誤った値を設定する事がありました。

今版では正しい値を設定するため、一部のプラットフォームとxsi:typeの値に差異がある場合があります。なお、匿名型の要素については、次の様な型名を割り当てます「'>'で始まり、これに、該当要素が所属する型がある場合はその型名に'>'を繋げ、これに、該当要素の名前を繋げた名前」。

対応方法

今版の環境において、iswsgen serverコマンドおよびiswsgen clientコマンドで、サービスエンドポイントインタフェースなどを再生成する事で、正しいxsi:typeが設定される様になります。誤ったxsi:type属性値に依存したアプリケーションがある場合は修正してください。

29.11 スレッドダンプが出力された場合の対処

スレッドダンプの出力方法については、“3.12.3 スレッドダンプ自動採取”を参照してください。

■出力例

以下にスタックトレース部の出力例を示します。

出力例

```
"Thread-29" prio=10 tid=0x12a23d50 nid=0x11e4 waiting on monitor [0..0x1ef4fd6c]
"RMI LeaseChecker" daemon prio=5 tid=0x14831c18 nid=0x1270 waiting on monitor
    [0x1d7ff000..0x1d7ffdf8]
      at java.lang.Thread.sleep(Native Method)
      at sun.rmi.transport.DGCImpl$LeaseChecker.run(DGCImpl.java:294)
      at java.lang.Thread.run(Thread.java:479)
"RMI Reaper" prio=5 tid=0x12f46020 nid=0x11f0 waiting on monitor
    [0x1d77f000..0x1d77fdf8]
      at java.lang.Object.wait(Native Method)
                        (中略)
"main" prio=5 tid=0x287d00 nid=0x1148 runnable [0x6f000..0x6fc20]
      at com.fujitsu.interstage.j2ee.ijserver.util.WUties.extp_is_IJServer_stop
        (Native Method)
      at com.fujitsu.interstage.j2ee.ijserver.util.WUties.stop(WUties.java:220)
      at org.apache.catalina.startup.Catalina.start(Catalina.java:664)
      at org.apache.catalina.startup.Catalina.execute(Catalina.java:442)
      at org.apache.catalina.startup.Catalina.process(Catalina.java:207)
      at java.lang.reflect.Method.invoke(Native Method)
      at com.fujitsu.interstage.j2ee.ijserver.IJServer.main(IJServer.java:407)
"VM Thread" prio=5 tid=0x7184d0 nid=0x1150 runnable
```

```
  "VM Periodic Task Thread" prio=10 tid=0x71ac60 nid=0x11c0 waiting on monitor
  "Suspend Checker Thread" prio=10 tid=0x71b620 nid=0xeac runnable
```

■スタックトレースの見方

スタックトレースは、javaの**java.lang.Throwableクラス**printStackTrace**メソッド**が出力する形式と同様です。メソッドの呼び出し元から呼び出し先に向かって下から上にスタックに積まれ、現在走行中のメソッドが最上位に表示されます。

出力例

```
  "http-9000-Processor15" daemon prio=6 tid=0x287b7008 nid=0x111c waiting on
 condition [0x05ebf000..0x05ebfa98]
  at java.lang.Thread.sleep(Native Method)
  at examples.TestServlet.doGet(Test.java:12)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
```

上記出力例では、TestServletのdoGetメソッドの12行目でsleepしていることがわかります。

解析方法

スタックトレース中に自分で作成したクラスがあれば、意図せずに待ち状態になっていないか確認してください。行番号が表示されている場合は、ソースコードを参照することで実行中の処理を確認することができます。

スタックトレースの解析時には“チューニングガイド”の“スレッドダンプ”もあわせて参照してください。
以下は調査対象となる代表的なスレッドのスタックトレースの出力例です。

【サーブレットのリクエスト処理中のスレッド】

リクエスト処理中のスレッドではフィルター/サーブレット/JSP等が処理されます。

サーブレット処理中のスレッドの出力例

サーブレットの場合はdoGetやdoPost等、オーバーライドして作成したサーブレットのメソッドより上位のスタックが調査対象となります。

```
  "http-9000-Processor15" daemon prio=6 tid=0x56ae0070 nid=0x11c8 runnable [0x5f58f000..0x5f58fb98]
  at examples.TestServlet.doGet(TestServlet.java:27)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:689)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
                              以下省略
```

上記出力例ではTestServletのdoGetメソッドより上位のスタックが調査対象となります。
ここではTestServletのdoGetメソッドが処理中であることがわかります。

JSP処理中のスレッドの出力例

JSPの場合は以下のようにorg.apache.jspから始まるパッケージ名のクラスに変換されています。通常HttpJspBaseのserviceメソッドより上位のスタックが調査対象となります。このスレッドからは、スクリプトレット部分から呼び出されるjavaクラスやカスタムタグクラス等が呼び出されます。

```
  "http-9000-Processor16" daemon prio=6 tid=0x5688f7b8 nid=0x10c8 waiting on condition
[0x619af000..0x619afa98]
  at java.lang.Thread.sleep(Native Method)
  at examples.TestTag.doEndTag(TestTag.java:19)
  at org.apache.jsp.jsp.test_jsp._jspService(test_jsp.java:64)
  at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:97)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:373)
```

```
  at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:329)
  at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:271)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:802)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:252)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
                                以下省略
```

上記出力例ではカスタムタグTestTagのdoEndTagメソッドの19行でsleepしていることがわかります。

フィルター処理中のスレッドの出力例

フィルターの場合はオーバーライドして作成したdoFilterメソッドより上位のスタックが調査対象となります。

```
  "http-9000-Processor16" daemon prio=6 tid=0x56a5a050 nid=0x1230 waiting on condition
[0x6174f000..0x6174fb98]
  at java.lang.Thread.sleep(Native Method)
  at examples.TestFilter2.doFilter(TestFilter2.java:15)      (2)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:202)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
  at examples.TestFilter1.doFilter(TestFilter1.java:18)       (1)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:202)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:173)
                                以下省略
```

上記出力例ではフィルターが2つ定義されているため(1)と(2)の2箇所に作成したフィルターのスタックが出力されており、TestFilter1、TestFilter2の順番で呼び出されTestFilter2のdoFilterメソッドの15行でsleepしていることがわかります。

【サーブレットのセションタイムアウト時のスレッド】

サーブレットのセションタイムアウト時のスレッドではHttpSessionListenerのsessionDestroyed等のセションが破棄された場合の処理が動作します。

```
  "ContainerBackgroundProcessor[StandardEngine[Catalina]]" daemon prio=6 tid=0x569bbfa0 nid=0x111c runnable
[0x60fcf000..0x60fcfb98]
  at examples.TestListener.sessionDestroyed(TestListener.java:45)
  at org.apache.catalina.session.StandardSession.expire(StandardSession.java:701)
  - locked <0x4dc9abe8> (a com.fujitsu.interstage.jservlet.catalina.session.ISSession)
  at org.apache.catalina.session.StandardSession.isValid(StandardSession.java:581)
  - locked <0x4e7e3e80> (a com.fujitsu.interstage.jservlet.catalina.session.ISManager)
  at org.apache.catalina.session.ManagerBase.processExpires(ManagerBase.java:678)
  at org.apache.catalina.session.ManagerBase.backgroundProcess(ManagerBase.java:663)
  at com.fujitsu.interstage.jservlet.catalina.session.ISManager.backgroundProcess(ISManager.java:147)
  at org.apache.catalina.core.ContainerBase.backgroundProcess(ContainerBase.java:1284)
  at org.apache.catalina.core.ContainerBase$ContainerBackgroundProcessor.processChildren(ContainerBase.java:
1569)
  at org.apache.catalina.core.ContainerBase$ContainerBackgroundProcessor.processChildren(ContainerBase.java:
1578)
  at org.apache.catalina.core.ContainerBase$ContainerBackgroundProcessor.processChildren(ContainerBase.java:
1578)
  at org.apache.catalina.core.ContainerBase$ContainerBackgroundProcessor.run(ContainerBase.java:1558)
  at java.lang.Thread.run(Thread.java:595)
                                以下省略
```

上記出力例ではTestListenerのsessionDestroyedメソッドの45行が処理中であることがわかります。

※セションのタイムアウト時に呼び出される処理はスレッドダンプ自動採取の対象外です。

【IJServer起動時のスレッド】

IJServer起動時にはload-on-startupを指定したサーブレットの初期化等が実行されます。

```
"main" prio=6 tid=0x0028cc08 nid=0x1080 runnable [0x0006f000..0x0006fbf8]
at examples.TestServlet.init(TestServlet.java:26)
at javax.servlet.GenericServlet.init(GenericServlet.java:211)
at org.apache.catalina.core.StandardWrapper.loadServlet(StandardWrapper.java:1105)
- locked <0x4cfccef0> (a org.apache.catalina.core.StandardWrapper)
at org.apache.catalina.core.StandardWrapper.load(StandardWrapper.java:932)
- locked <0x4cfccef0> (a org.apache.catalina.core.StandardWrapper)
at org.apache.catalina.core.StandardContext.loadOnStartup(StandardContext.java:3952)
at com.fujitsu.interstage.jservlet.catalina.core.ISContext.loadOnStartup(ISContext.java:324)
at org.apache.catalina.core.StandardContext.start(StandardContext.java:4226)
- locked <0x4e8f1c70> (a com.fujitsu.interstage.jservlet.catalina.core.ISContext)
at com.fujitsu.interstage.jservlet.catalina.core.ISContext.start(ISContext.java:140)
- locked <0x4e8f1c70> (a com.fujitsu.interstage.jservlet.catalina.core.ISContext)
                              中略
at com.fujitsu.interstage.j2ee.ijserver.startup.IJServerBoot.process(IJServerBoot.java:532)
at com.fujitsu.interstage.j2ee.ijserver.startup.IJServerBoot.start(IJServerBoot.java:588)
at com.fujitsu.interstage.j2ee.ijserver.startup.IJServerBoot.main(IJServerBoot.java:199)
```

上記出力例ではTestServletのinitメソッドの26行が処理中であることがわかります。

【IJServer停止時のスレッド】

IJServer停止時のスレッドではサーブレットのdestroy等が処理されます。

```
"Thread-29" prio=6 tid=0x613d62b0 nid=0x11c4 runnable [0x6617f000..0x6617fb18]
at examples.TestServlet.destroy(TestServlet.java:38)
at org.apache.catalina.core.StandardWrapper.unload(StandardWrapper.java:1326)
- locked <0x4e866370> (a org.apache.catalina.core.StandardWrapper)
at org.apache.catalina.core.StandardWrapper.stop(StandardWrapper.java:1654)
at org.apache.catalina.core.StandardContext.stop(StandardContext.java:4346)
- locked <0x4e85e728> (a com.fujitsu.interstage.jservlet.catalina.core.ISContext)
at com.fujitsu.interstage.jservlet.catalina.core.ISContext.stop(ISContext.java:117)
- locked <0x4e85e728> (a com.fujitsu.interstage.jservlet.catalina.core.ISContext)
                              中略
at com.fujitsu.interstage.j2ee.ijserver.context.IJServerContext.stop(IJServerContext.java:536)
at com.fujitsu.interstage.j2ee.ijserver.core.ISServer.syncStop(ISServer.java:943)
at com.fujitsu.interstage.j2ee.ijserver.core.ISServer$1.run(ISServer.java:648)
```

上記出力例ではTestServletのdestroyメソッドの38行が処理中であることがわかります。

29.12 Interstage JMSの異常時の対処

ここでは、JMSアプリケーション運用中にトラブルが発生した場合の対処方法を説明します。

Interstage JMSでは、運用中のJMSアプリケーション情報を、システムログに出力します。JMSアプリケーション運用中に異常が発生した場合には、異常の種別や異常を伝えるメッセージが出力されますので参照してください。また、異常時のメッセージにスタックトレース情報を加えたものを、コンソールログとして出力します。システムログだけでは異常の原因がわからない場合は、コンソールログも同時に参照してください。その場合は、メッセージの出力時間と番号で、当該メッセージを特定してください。

なお、JMSアプリケーションで異常が発生していて、コンソールログを参照しても異常の原因が判明しない場合は、JMSアプリケーション実行中の各種入出力情報をロギングする機能を使用してスナップログを採取して、発生している異常情報の取得を行ってください。

以下にJMSアプリケーションで異常が発生した場合の対処方法を示します。

1. システムログを参照する
2. コンソールログを参照する
3. スナップログを採取する

注意

JMSアプリケーション運用中にトラブルが発生した場合、“トラブルシューティング集”－“イベントサービス運用時の異常”も参考にして対処してください。

29.12.1 システムログ

Interstage JMSでは、運用中のJMSアプリケーション情報を、システムログに出力します。

Windows32/64

運用時にJMSアプリケーションからの応答がなくなった場合など、何らかの異常が発生した場合は、イベントビューアでアプリケーションログを参照して、JMSアプリケーションの状態を確認してください。

Solaris64 Linux32/64

運用時にJMSアプリケーションからの応答がなくなった場合など、何らかの異常が発生した場合は、テキストエディタでシステムログを参照して、JMSアプリケーションの状態を確認してください。

一般にInterstage JMSで発生する例外は次の3種類に分類されます。

- JMS例外
  一般に例外はJMSExceptionのかたちでユーザに通知されます。
  “メッセージ集”を参照して、JMSのエラー番号に対応する「ユーザの対処」を実施してください。
- イベントサービス例外
  イベントサービスと関連する例外の可能性があります。
  “メッセージ集”を参照して、イベントサービスのエラー番号に対応する「ユーザの対処」を実施してください。
  イベントサービスでは、イベント通信に失敗した場合にログが出力されます。
  “メッセージ集”を参照して、イベントサービスが出力するメッセージに対応する「ユーザの対処」を実施してください。
- CORBAサービス例外
  “メッセージ集”を参照して、CORBAサービスのエラー番号に対応する「ユーザの対処」を実施してください。

29.12.2 コンソールログ

Interstage JMSでは、異常時のメッセージにスタックトレース情報を加えたものを、コンソールログとして出力します。

コンソールログは、テキストエディタで開いて参照してください。
コンソールログのファイル名を次に示します。

Windows32/64

```
C:¥Interstage¥jms¥var¥fjmsconsole.log
```

Solaris64 Linux32/64

```
/var/opt/FJSVjms/fjmsconsole.log
```

コンソールログはファイルの大きさが規定値に達するたびに新しく作成され、古いコンソールログは次の名前にリネームされます。

```
fjmsconsole.logn(nは数値)
```

コンソールログは3世代まで管理され、3世代を超えるものは破棄されますので、必要な場合はコピーするなどして退避してください。

29.12.3 スナップログ

スナップログはJMSアプリケーションのデバッグ情報として、JMSアプリケーション実行中の各種入出力情報をロギングする機能です。

スナップログを採取するには、JMSアプリケーション起動時に、下表の定義を付加します。

fjmssnapfile	スナップログが出力されるファイル名をフルパスで指定します。		
fjmssnaplevel	スナップログの出力レベルを指定します。指定値には1と2があり、デフォルトは1です。 それぞれのレベルで出力される情報を次に示します。	1	2
	呼び出された全APIメソッド	○	○
	全APIメソッドのパラメタ値		○
	全APIメソッドからの復帰	○	○
	APIメソッドからの復帰値		○
	APIメソッド呼び出し中に発生した例外	○	○
	Throwされた例外からのスタックトレース		○
	送信したメッセージタイプ	○	○
	送信したメッセージデータの詳細		○
	受信したメッセージタイプ	○	○
	受信したメッセージデータの詳細		○

例

Windows32/64

```
java -Dfjmssnapfile=C:¥Interstage¥jms¥var¥Publisher.log -Dfjmssnaplevel=2 Publisher Factory Topic
```

Solaris64 Linux32/64

```
java -Dfjmssnapfile=/var/opt/FJSVjms/Publisher.log -Dfjmssnaplevel=2 Publisher Factory Topic
```

注意事項

Solaris64 Linux32/64

スナップログが出力されるファイル名のフルパスには、シンボリックリンクを含めることはできません。

29.12.4 よくある問題とその対処方法

下表によくある問題とその対処方法を示します。

問題	主な原因	対処
JNDI定義情報にTopicConnectionFactory、またはQueueConnectionFactoryが見つからない。	JNDIサブコンテキストが正しくない。	正しいJNDIサブコンテキストを使用しているか確認してください。 JMSアプリケーションからJNDI名を参照する場合は“java:comp/env/jms/”を付加してください。

問題	主な原因	対処
	誤ったTopicConnectionFactory名、QueueConnectionFactory名が使用された。	すべての名前は大文字、小文字を区別します。 jmsinfofactコマンドを使用して、ConnectionFactory名を確認してください。
JNDI定義情報にTopic、またはQueueが見つからない。	イベントチャネルが作成されていない。	esmkchnlコマンドを使用して、イベントチャネルが作成されているか確認してください。 esmonitorコマンドにより、イベントチャネルが起動しているか確認してください。
	JNDIサブコンテキストが正しくない。	正しいJNDIサブコンテキストを使用しているか確認してください。 JMSアプリケーションからJNDI名を参照する場合は“java:comp/env/jms/”を付加してください。
	誤ったTopic名、Queue名、チャネル名、グループ名が使用された。	すべての名前は大文字、小文字を区別します。 jmsinfodstコマンドを使用して、Destination名を確認してください。また、イベントチャネル運用マシン上で指定したチャネル名、グループ名と同じか確認してください。
JNDI名、クライアント識別子およびDurable Subscription名の制限。	使用可能な文字種は、アルファベット(大文字小文字ともに)、数字、ピリオド(.)、コロン(:)、アンダースコア(_)、ハイフン(-)、スラッシュ(/)です。スラッシュはJNDI名のアトミック名の区切り文字として認識されます。 アトミック名およびクライアント識別子の先頭文字はアルファベット+数字のみ指定可能です。 指定可能な最大文字列長は255バイトです。ただし、JNDI名は“java:comp/env/jms/”より後ろが255バイト以下となります。	正しい名前を使用してください。
Subscriberが Session.commit()または Message.acknowledge()などでメッセージの受信確認を行っているのに、イベントサービスのesmonitorコマンド画面での QueueCountが減らない。	closeメソッドを呼ばずにSubscriberを終了した(アプリケーションの異常終了などを含む)可能性があります。 ・ Connection.close() ・ Session.close() ・ MessageConsumer.close() これにより、イベントチャネルにプロキシ(接続情報)が残ったため、イベントチャネルからデータが減らないということが考えられます。 Durable Subscription機能を使用していない場合、Subscriberを起動すると新しいプロキシを作成します。Subscriberがclose	esmonitorコマンドを使用して、イベントチャネルの状態(ConsumerCount)を確認してください。 プロキシの数が実際に運用するSubscriberの数より多い場合は、esstopchnl→esrmchnlを行い初期化してからesmkchnl→esstartchnlを行ってください。

問題	主な原因	対処
	メソッドを呼ばずに終了した場合、イベントチャネルにそのプロキシが残り、Subscriberを再起動しても新しいプロキシを作成するため、以前作成したプロキシは使用されません。使用されないプロキシ用にイベントチャネルはデータを保存し続けるため、再起動したSubscriberがメッセージの受信確認を行っても、イベントチャネルからデータの削除は行われません。そのため、esmonitorコマンド画面でのQueueCountが減らないという現象が起こります。 Durable Subscription機能を使用している場合、Subscriber終了後、Subscriberを再起動すると、以前作成したプロキシを使用します。そのため、キューカウントが減らないという現象は起こりません。	
イベントサービスのesmonitorコマンドによりイベントチャネルにメッセージの蓄積が確認できる状態で、JMSアプリケーションを起動してもイベントチャネルからメッセージを受信できない。	Point-To-Pointメッセージングモデルにおいて、メッセージ受信のトランザクション処理中にアプリケーションが終了し、トランザクションタイムアウトまで受信中であったメッセージが配信されないことが考えられます。	“トラブルシューティング集”－“イベントサービス運用時の異常”の“アプリケーション運用中の異常”に記載されている“コンシューマアプリケーションの異常終了(接続情報を保存していない場合)”を参照してください。
NoClassDefFoundErrorが発生する。	動作環境の設定に誤りがあります。	必要な製品が正しくインストールされ、環境変数CLASSPATHが正しく設定されていることを確認してください。
javax.naming.NamingException: .GlobalTransactionMode nothingが発生する。	グローバルトランザクション機能を使用するための設定に誤りがあります。	JNDI環境プロパティcom.fujitsu.ObjectDirector.CORBA.GlobalTransactionModeにTrueが指定されていることを確認してください。
Windows32/64 JMS運用コマンドを実行した場合に“Registry key 'Software¥JavaSoft¥Java Runtime Environment ¥CurrentVersion' has value '1.n', but '1.n' is required.”のメッセージがコンソールに出力される。 注)'1.n'はインストールされているJDKのバージョン	Interstage Application Serverがインストールしたjdk以外のJDKがインスールされているため、JMS運用コマンドが正常に動作しません。	以下のことを確認してください。 ・ JDKが正しくインストールされていることを確認してください。 ・ JDKのパスが環境変数PATHに設定されていることを確認してください。 または、以下のディレクトリ(インストールパスはデフォルト)に格納されているJava環境設定ファイル(java_config)に使用するJavaのインストールディレクトリを設定後、JMS運用コマンドを実行してください。 ・ C:¥Interstage¥jms¥bin 設定形式 JDKDIR=javaのインストールディレクトリ (注)

問題	主な原因	対処
		注)javaのインストールディレクトリは絶対パス形式で設定してください。 設定例 C:¥jdkにインストールしたJDKを使用する場合: JDKDIR=C:¥jdk
JMSException		“メッセージ集”を参照して、ユーザの対処を実施してください。
Interstage管理コンソールの[サービス] > [JMS] > [イベントチャネル]で[イベントチャネル:状態]画面が表示できない(エラーメッセージ“es39996”または“es39999”が出力される)。	イベントチャネルグループ名に以下の使用可能な文字以外が含まれています。 Windows32/64 英数字、イクスクラメーションマーク(!)、ナンバー(#)、ドルマーク($)、アポストロフィー(')、左丸括弧(()、右丸括弧())、プラス(+)、ハイフン(-)、ピリオド(.)、スラッシュ(/)、セミコロン(;)、アットマーク(@)、左角括弧([)、右角括弧(])、アンダースコア(_)、逆クォート(`)、左中括弧({)、右中括弧(})、波線(~) Solaris64 英数字、イクスクラメーションマーク(!)、プラス(+)、ハイフン(-)、ピリオド(.)、スラッシュ(/)、アットマーク(@)、左角括弧([)、右角括弧(])、アンダースコア(_)、左中括弧({)、右中括弧(})、波線(~) Linux32/64 英数字、プラス(+)、ハイフン(-)、ピリオド(.)、スラッシュ(/)、アットマーク(@)、左角括弧([)、右角括弧(])、キャロット(^)、アンダースコア(_)、左中括弧({)、右中括弧(})	Interstage管理コンソールを使用する場合は、esmonitorコマンドを使用してイベントチャネルグループ名を確認し、esrmchnlコマンドを使用して、該当するイベントチャネルを削除してください。 または、イベントサービス運用コマンドで操作してください。
	イベントチャネル名に以下の使用可能な文字以外が含まれています。 Windows32/64 英数字、イクスクラメーションマーク(!)、ナンバー(#)、ドルマーク($)、アポストロフィー(')、左丸括弧(()、右丸括弧())、アスタリスク(*)、プラス(+)、カンマ(,)、ハイフン(-)、ピリオド(.)、スラッシュ(/)、コロン(:)、セミコロン(;)、イコール(=)、クエスチョンマーク(?)、アットマーク(@)、左角括弧([)、右角括弧(])、アンダースコア(_)、逆クォート(`)、左中括弧({)、右中括弧(})、波線(~) Solaris64	Interstage管理コンソールを使用する場合は、esmonitorコマンドを使用してイベントチャネル名を確認し、esrmchnlコマンドを使用して、該当するイベントチャネルを削除してください。 または、イベントサービス運用コマンドで操作してください。

問題	主な原因	対処
	英数字、イクスクラメーションマーク(!)、アスタリスク(*)、プラス(+)、カンマ(,)、ハイフン(-)、ピリオド(.)、スラッシュ(/)、コロン(:)、イコール(=)、クエスチョンマーク(?)、アットマーク(@)、左角括弧([)、右角括弧(])、アンダースコア(_)、左中括弧({)、右中括弧(})、波線(~) Linux32/64 英数字、パーセント(%)、アスタリスク(*)、プラス(+)、カンマ(,)、ハイフン(-)、ピリオド(.)、スラッシュ(/)、コロン(:)、イコール(=)、クエスチョンマーク(?)、アットマーク(@)、左角括弧([)、右角括弧(])、キャロット(^)、アンダースコア(_)、左中括弧({)、右中括弧(})	
Interstage管理コンソールの[サービス] > [JMS] > [保存先]で[保存先:情報]画面が表示できない(エラーメッセージ“es39996”または“es39999”が出力される)。	格納ディレクトリに以下の使用禁止文字が含まれています。 ・ アンパサンド(&) ・ 小なり(<)	Interstage管理コンソールを使用する場合は、ユニット作成時のユニット定義ファイルで格納ディレクトリを確認し、esrmunitコマンドを使用して、該当するユニットを削除してください。 または、イベントサービス運用コマンドで操作してください。
Interstage管理コンソールの[サービス] > [JMS] > [ConnectionFactory]で[ConnectionFactory:一覧]画面が表示できない(エラーメッセージ“jms9997”が出力される)。	JNDI名に以下の使用禁止文字が含まれています。 ・ コロン(:)	Interstage管理コンソールを使用する場合は、jmsinfofactコマンドを使用してJNDI名およびクライアント識別子を確認し、jmsrmfactコマンドを使用して、該当するConnectionFactory定義を削除してください。 または、JMS運用コマンドで操作してください。
Interstage管理コンソールの[サービス] > [JMS] > [イベントチャネル] > [イベントチャネルグループ名::イベントチャネル名] > [Destination]で[Destination:一覧]画面が表示できない(エラーメッセージ“jms9997”が出力される)。	JNDI名に以下の使用禁止文字が含まれています。 ・ コロン(:)	Interstage管理コンソールを使用する場合は、jmsinfodstコマンドを使用してJNDI名、グループ名およびチャネル名を確認し、jmsrmdstコマンドを使用して、該当するDestination定義を削除してください。 または、JMS運用コマンドで操作してください。
JMS運用コマンドを使用して作成したDestination定義がInterstage管理コンソールに表示されない。	イベントチャネルが作成されていない。	Destination定義のグループ名、チャネル名に指定したイベントチャネルが作成されていることを確認してください。 または、JMS運用コマンドで操作してください。
	誤ったチャネル名、グループ名が使用された。	jmsinfodstコマンドを使用して、イベントチャネルのグループ名、チャネル名を確認してください。

問題	主な原因	対処
以下の状態であるにも関わらず、Message-driven Beanアプリケーションにおいてメッセージを受信できない。 ・ Message-driven Beanアプリケーションを配備したワークユニットは、起動中である。 ・ イベントチャネルは、動作中である。 ・ クライアントアプリケーションからイベントチャネルに対し、正常にメッセージを送信できる。 ・ esmonitorコマンドによりイベントチャネルのイベントデータの現蓄積数を確認できる。	ワークユニットの起動後、イベントチャネルが再起動された可能性があります。 イベントチャネルを再起動した場合は、ワークユニットも再起動する必要があります。	Windows(R)システムの場合はイベントログ、Solarisシステムの場合はシステムログに、以下の順でメッセージが出力されている場合、ワークユニットを再起動してください。 ・ 情報メッセージEXTP4401「ワークユニットが起動しました」 ・ 情報メッセージtd11028「ワークユニットを起動しました」 ・ 情報メッセージtd11008「ワークユニットを正常に起動しました」 ・ 情報メッセージes10028「イベントチャネルグループを終了しました」 ・ エラーメッセージjms2550「通信エラーが発生しました」 ・ 情報メッセージes10023「イベントチャネルグループを起動しました」
以下のように操作した場合、イベントチャネルに格納したメッセージの順序で受信されない。 1. Message-driven Beanアプリケーションを配備したワークユニットをメッセージの処理中に強制終了する。 2. ワークユニットを再起動する。	イベントチャネルのローカルトランザクション処理中に、ワークユニットを強制停止した場合、メッセージはローカルトランザクションのタイムアウト時間に達するまで受信できません。 したがって、ローカルトランザクションのタイムアウト時間に達する前にワークユニットが再起動された場合、次にイベントチャネルに格納したメッセージが先に受信されることになります。 なお、ローカルトランザクション処理中に強制終了したメッセージは、ローカルトランザクションのタイムアウト時間に達した後、受信できます。	ワークユニットを[停止(同期停止)]で停止してください。 ワークユニットを強制終了する場合は、イベントサービスおよびイベントチャネルのローカルトランザクションのタイムアウト時間を確認し、タイムアウト時間に達した後、ワークユニットを再起動してください。(注1) なお、esmkchnlコマンドでイベントチャネルを作成する際に-autodisconオプションを指定すると、イベントチャネルに残ったコネクション情報を自動回収する機能が有効となり、本現象を回避することができます。 esmkchnlコマンドの詳細については、“リファレンスマニュアル(コマンド編)”を参照してください。
Message-driven Beanを運用した場合、エラーメッセージ“od10965”が出力される。	イベントチャネルの最大接続数の設定値に、実際に運用中のサプライヤおよびコンシューマの合計値より小さい値が設定されています。	サプライヤおよびコンシューマが正しい運用状態であるかを確認してください。 正しい運用状態である場合、イベントチャネルの最大接続数の設定値を確認してください。(注2) 最大接続数の設定値が実際に運用中のサプライヤおよびコンシューマの合計値より少ない場合は、最大接続数の値を増加してイベントチャネルを再作成してください。
イベントチャネルのメッセージ蓄積可能なデータ数の上限値を超過した場合、最初に格納されたメッセージが受信できない。	イベントチャネルのQoSプロパティ項目“DiscardPolicy”が省略時の“FifoOrder”となっています。詳細については、“アプリケーション作成ガイド(イベントサービス編)”の“QoS機能”―	Sender/Publishアプリケーションを実行する場合は、“-Dcom.fujitsu.interstage.jms.queue_max_err=yes”を指定して実行してください。

問題	主な原因	対処
	“QoSプロパティ”を参照してください。	
java.lang.ClassCastExceptionが発生する。	Interstage JMSで使用できないDestinationが指定されました。	アプリケーションにおいて、指定したDestinationに誤りがないかを確認してください。Destinationには、以下のいずれかの値を設定してください。 ・ Interstageが提供するJNDIサービスプロバイダのlookupで取得したDestination ・ createQueue、createTopic、createTemporaryQueueまたはcreateTemporaryTopicメソッドで作成されたDestination
java.lang.ArrayIndexOutOfBoundsExceptionが発生する。	setJMSCorrelationIDAsBytesメソッドに指定された値に誤りがあります。	アプリケーションにおいて、setJMSCorrelationIDAsBytesメソッドに指定された値に誤りがないかを確認してください。 setJMSCorrelationIDAsBytesメソッドのパラメタ“byte[] correlationID”には、文字列に復号化されるバイトを指定してください。

注1）ローカルトランザクションのタイムアウト時間は、以下を使用して確認してください。

- Interstage管理コンソールで運用している場合
  - [システム] > [リソース] > [JMS] > [構成情報]
  - [システム] > [リソース] > [JMS] > [イベントチャネル] > [イベントチャネルグループ名::イベントチャネル名] > [環境設定]
- イベントサービス運用コマンドで運用している場合
  - essetcnf -d
    local tran timeout (sec) :300
  - essetcnfchnl -d -g イベントチャネルグループ名
    local tran timeout (sec) :default

注2）イベントチャネルの最大接続数は、以下を使用して確認してください。

- Interstage管理コンソールで運用している場合
  - [システム] > [リソース] > [JMS] > [イベントチャネル] > [グループ名::イベントチャネル名] > [設定情報]
- イベントサービス運用コマンドで運用している場合
  - essetcnfchnl -d -g イベントチャネルグループ名
    maximum connection (mixed) :16

29.13 Interstage JNDIの異常時の対処

ここでは、JNDIを使用したJ2EEアプリケーションの運用中やリソース操作時に異常が発生した場合の対処方法を説明します。
J2EEアプリケーションが異常終了したり、ループ状態になったりした場合には、原因を取り除いた後、再度実行してください。

29.13.1 javaコマンドが見つからない場合

javaが見つからない場合は、JDKがインストールされ、正しく設定されていることを確認してください。

29.13.2 J2EEアプリケーションで例外が発生した場合

J2EEアプリケーション運用時に例外が発生した場合の対処方法については、“メッセージ集”の“J2EE使用時に出力される例外情報”を参照してください。

29.13.3 クライアントアプリケーションで例外が発生した場合

lookup処理で例外が発生したときの例外情報や対処方法の詳細については、“メッセージ集”の“J2EE使用時に出力される例外情報”の“lookup処理で例外が発生した場合の対処”を参照してください。

29.13.4 リソース操作時の異常

JDBCデータソース登録時の異常

JDBCデータソースの定義名に不当な文字を用いた場合、以下の例のようなエラーメッセージが出力されて、JDBCデータソースの登録に失敗する場合があります。
Interstage管理コンソールのヘルプまたは“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照し、指定できない文字列を除いて再度実行してください。

- Interstage管理コンソール
  IS: エラー: is40002: Interstage JMXサービス上で異常が発生しました(エラー情報=*****)
- isj2eeadminコマンド
  isj2eeadmin: エラー: isj2ee2052:Interstage JMXサービス上で異常が発生しました ERROR=*****
  isj2eeadmin: エラー: isj2ee2206:JDBCデータソースの登録に失敗しました NAME=*****

JavaMailリソース登録時の異常

JavaMailのリソース名(定義名)に不当な文字を用いた場合、以下の例のようなエラーメッセージが出力されて、リソースの作成に失敗する場合があります。
Interstage管理コンソールのヘルプまたは“リファレンスマニュアル(コマンド編)”の“isj2eeadmin”を参照し、指定できない文字列を除いて再度実行してください。

- Interstage管理コンソール
  IS: エラー: is40002: Interstage JMXサービス上で異常が発生しました(エラー情報=*****)
- isj2eeadminコマンド
  isj2eeadmin: エラー: isj2ee2052:Interstage JMXサービス上で異常が発生しました ERROR=*****
  isj2eeadmin: エラー: isj2ee2306: JavaMailリソースの登録に失敗しました NAME=*****

また、このときリソースとしては登録されていますが、各種定義情報が異常値となり、参照/削除/更新等が正常に操作できないことがあります。
該当のJavaMailリソースは、以下の例のように、isj2eeadmin resourceコマンドに-allオプションを指定して削除してください。

例

```
isj2eeadmin resource -d -all -k javamail <Enter>
Delete "[該当リソース名]", are you sure (a/y/n)? y <Enter>
```

このとき次のメッセージが出力されますが、削除は行われています。

- isj2eeadmin: エラー: isj2ee2052:Interstage JMXサービス上で異常が発生しました ERROR=*****
  isj2eeadmin: エラー: isj2ee2307:JavaMailリソースの削除に失敗しました NAME=*****

注意

該当のリソースのみ、問い合わせに対して“y”を入力して削除してください。

29.14 Javaアプリケーションのメソッドトレースの採取

J2EEアプリケーションの開発・異常時の調査では、Javaメソッドトレースが有効な場合があります。メソッドトレースの定義方法および詳細については、“トラブルシューティング集”－“Javaツール機能”を参照してください。
ここでは、Webアプリケーションに対してfjtrace(メソッドトレース)を適用した場合の採取例、調査例を紹介します。

29.14.1 サーブレットの場合

以下のサーブレットを例として、トレース情報の見方を説明します。

■例:HelloServletのソースコード

```
 1:  import javax.servlet.http.*;
 2:  import javax.servlet.*;
 3:  import java.io.*;
 4:  public class HelloServlet extends HttpServlet{
 5:    public void doGet(HttpServletRequest request, HttpServletResponse response)
 6:                        throws ServletException, IOException {
 7:      response.setContentType("text/html");
 8:      PrintWriter out = response.getWriter();
 9:      out.println("<html>");
10:      out.println("<head>");
・・・(略)・・・
11:    }
12:  }
```

■例:トレース情報

上記サーブレットを呼び出した場合、以下のトレース情報が出力されます。日時、スレッド名および16進出力は、省略しています。また、表示の関係上、改行している箇所があるため、実際の出力形式とは若干の違いがあります。

```
・・・(略)・・・
1: in, org.apache.catalina.connector.RequestFacade, <init>,
   PARAM:(ISRequest):com.fujitsu.interstage.jservlet.catalina.connector.ISRequest@24d517
2: out, org.apache.catalina.connector.RequestFacade, <init>,
   RET:void
・・・(略)・・・
3: in, org.apache.catalina.connector.ResponseFacade, <init>,
   PARAM:(ISResponse):com.fujitsu.interstage.jservlet.catalina.connector.ISResponse@2219b4
4: out, org.apache.catalina.connector.ResponseFacade, <init>, RET:void
・・・(略)・・・
5: in, HelloServlet, doGet,
   PARAM:(RequestFacade):org.apache.catalina.connector.RequestFacade@c58af4:
  (ResponseFacade):org.apache.catalina.connector.ResponseFacade@125b750
6: in, org.apache.catalina.connector.ResponseFacade, setContentType,
   PARAM:(String):text/html
7: in, org.apache.catalina.connector.ResponseFacade, isCommitted, PARAM:
8: out, org.apache.catalina.connector.ResponseFacade, isCommitted,
   RET:(boolean):false
9: out, org.apache.catalina.connector.ResponseFacade, setContentType, RET:void
10: in, org.apache.catalina.connector.ResponseFacade, getWriter, PARAM:
11: in, org.apache.catalina.connector.ResponseFacade, isFinished, PARAM:
12: out, org.apache.catalina.connector.ResponseFacade, isFinished,
   RET:(boolean):false
```

```
13: out, org.apache.catalina.connector.ResponseFacade, getWriter,
   RET:(CoyoteWriter):org.apache.catalina.connector.CoyoteWriter@1262667
14: in, org.apache.catalina.connector.CoyoteWriter, println, PARAM:(String):<html>
15: in, org.apache.catalina.connector.CoyoteWriter, print, PARAM:(String):<html>
16: in, org.apache.catalina.connector.CoyoteWriter, write, PARAM:(String):<html>
・・・(略)・・・
17: out, HelloServlet, doGet, RET:void
18: in, org.apache.catalina.connector.RequestFacade, getAttribute,
   PARAM:(String):javax.servlet.include.context_path
19: out, org.apache.catalina.connector.RequestFacade, getAttribute, RET:(Object):[]
・・・(略)・・・
20: in, org.apache.catalina.connector.RequestFacade, getServletPath, PARAM:
21: out, org.apache.catalina.connector.RequestFacade, getServletPath,
   RET:(String):/HelloServlet
22: in, org.apache.catalina.core.StandardWrapperFacade, getServletName, PARAM:
23: out, org.apache.catalina.core.StandardWrapperFacade, getServletName,
   RET:(String):HelloServlet
24: in, org.apache.catalina.connector.CoyoteWriter, recycle, PARAM:
25: out, org.apache.catalina.connector.CoyoteWriter, recycle, RET:void
```

・ doGetメソッドの呼び出しと復帰 5,17行目

“in”はメソッドの呼び出し、“out”は復帰を表しています。
上の例では正常に処理が行われているため、doGetメソッドは“in”と“out”が対になって出力されています。
引数および戻り値の情報もメソッドトレース機能の書式に従って出力されます。

・ リクエスト前処理 1～4行目

リクエストに先立ち、HttpServletRequest、HttpServletResponseの実装クラスが作成されていることがわかります。
HelloServletがinitメソッドを実装している場合は、初回呼び出し時に、リクエスト前処理に先立ちinitメソッドの呼び出しが出力されます。

・ サービス(doGetメソッド) 5～17行目

各メソッドとも、正常に復帰しているため、“in”と“out”が対になって出力されています。
トレース情報は以下のように出力されます。

例:トレース情報
6～9行目 HelloServletクラス、doGetメソッドからHttpServletResponseのsetContentTypeメソッドを呼び出しています。
setContentTypeメソッドは、内部的にisCommittedメソッドを呼び出した後復帰していることがわかります。

```
in, org.apache.catalina.connector.ResponseFacade, setContentType,
  PARAM:(String):text/html HEX=0074 0065 0078 0074 002f 0068 0074 006d 006c
in, org.apache.catalina.connector.ResponseFacade, isCommitted,
  PARAM:
out, org.apache.catalina.connector.ResponseFacade, isCommitted,
  RET:(boolean):false
out, org.apache.catalina.connector.ResponseFacade, setContentType, RET:void
```

setContentTypeメソッド引数のisCommittedメソッドの戻り値等がそれぞれ出力されています。

・ リクエスト後処理 18～25行目

doGetメソッドが復帰した後で、各種後処理が行われていることがわかります。
HelloServletがdestroyメソッドを実装している場合は、IJServer停止時にこのメソッドが呼び出されます。

29.14.2 JSPの場合

以下のJSPを例として、トレース情報の見方を説明します。

■例:HelloJSP.jspのコード

```
1 : <%@ page contentType="text/html; charset=Shift_JIS" %>
2 : <HTML>
```

```
 3 : <HEAD>
 4 : <META http-equiv="Content-Type" content="text/html; charset=shift_jis">
 5 : <TITLE>HelloJSP</TITLE>
 6 : </HEAD>
 7 : <BODY>
 8 : <H1>HelloJSP</H1>
 9 : <FORM method=post >
10 : <INPUT name="param" size="30">
11 : <INPUT type="submit" value="Submit">
12 : <INPUT type="reset" value="Reset">
13 : </FORM>
14 : <P>
15 : <%
16 : String param=request.getParameter("param");
17 : if(param==null)
18 :   param="";
19 : String encParam=new String(param.getBytes("8859_1"),"Shift_JIS");
20 : %>
21 : 入力データ：<%=encParam%>
22 : </BODY>
23 : </HTML>
```

■例:トレース情報

上記JSPを呼び出した場合、以下のトレース情報が出力されます。日時、スレッド名および16進出力は、省略しています。また、表示の関係上、改行している箇所があるため、実際の出力形式とは若干の違いがあります。

```
・・・(略)・・・
1: in, org.apache.catalina.connector.RequestFacade, getServletPath, PARAM:
2: out, org.apache.catalina.connector.RequestFacade, getServletPath,
   RET:(String):/HelloJSP.jsp
3: in, org.apache.catalina.connector.RequestFacade, getPathInfo, PARAM:
4: out, org.apache.catalina.connector.RequestFacade, getPathInfo, RET:(String):[]
5: in, org.apache.catalina.connector.RequestFacade, getQueryString, PARAM:
6: out, org.apache.catalina.connector.RequestFacade, getQueryString, RET:(String):[]
7: in, org.apache.jsp.HelloJSP_jsp, _jspService,
   PARAM:(RequestFacade):org.apache.catalina.connector.RequestFacade@15ede11:
   (ResponseFacade):org.apache.catalina.connector.ResponseFacade@1455cf4
8: in, org.apache.catalina.connector.ResponseFacade, setContentType,
   PARAM:(String):text/html; charset=Shift_JIS
9: in, org.apache.catalina.connector.ResponseFacade, isCommitted, PARAM:
10: out, org.apache.catalina.connector.ResponseFacade, isCommitted,
   RET:(boolean):false
11: out, org.apache.catalina.connector.ResponseFacade, setContentType, RET:void
・・・(略)・・・
12: in, org.apache.jasper.runtime.JspWriterImpl, write, PARAM:(String):
<HTML>
・・・(略)・・・
<P>

13: in, org.apache.jasper.runtime.JspWriterImpl, write, PARAM:(String):
<HTML>
・・・(略)・・・
<P>
:(int):0:(int):293
14: in, org.apache.jasper.runtime.JspWriterImpl, ensureOpen, PARAM:
15: out, org.apache.jasper.runtime.JspWriterImpl, ensureOpen, RET:void
16: in, org.apache.jasper.runtime.JspWriterImpl, min, PARAM:(int):8192:(int):293
17: out, org.apache.jasper.runtime.JspWriterImpl, min, RET:(int):293
18: out, org.apache.jasper.runtime.JspWriterImpl, write, RET:void
19: out, org.apache.jasper.runtime.JspWriterImpl, write, RET:void
20: in, org.apache.catalina.connector.RequestFacade, getParameter,
```

```
   PARAM:(String):param
・・・(略)・・・
21: out, org.apache.catalina.connector.RequestFacade, getParameter,
   RET:(String):FUJITSU
・・・(略)・・・
22: out, org.apache.jsp.HelloJSP_jsp, _jspService, RET:void
23: in, org.apache.catalina.connector.RequestFacade, getAttribute,
   PARAM:(String):javax.servlet.include.context_path
24: out, org.apache.catalina.connector.RequestFacade, getAttribute, RET:(Object):[]
・・・(略)・・・
25: in, org.apache.catalina.connector.RequestFacade, getServletPath, PARAM:
26: out, org.apache.catalina.connector.RequestFacade, getServletPath,
   RET:(String):/HelloJSP.jsp
27: in, org.apache.catalina.core.StandardWrapperFacade, getServletName, PARAM:
28: out, org.apache.catalina.core.StandardWrapperFacade, getServletName,
   RET:(String):jsp
29: in, org.apache.catalina.connector.CoyoteWriter, recycle, PARAM:
30: out, org.apache.catalina.connector.CoyoteWriter, recycle, RET:void
```

・ リクエスト前処理 1～6行目

HelloJSP.jspの呼び出しに先立ち、HttpServletRequest、HttpServletResponseの実装クラスが作成されていることがわかります。
HelloJSP.jspがjspInitメソッドを実装している場合は、初回呼び出し時に、リクエスト前処理に先立ち、jspInitメソッドの呼び出しが出力されます。

・ サービス(_jspServiceメソッド) 7～22行目

各メソッドとも、呼び出し後、正常復帰しているため、“in”と“out”が対になって出力されています。
JSPコード1行目のpageディレクティブの処理が、トレース情報8行目のsetContentTypeメソッド呼び出しになっています。
JSPコード2～14行目のHTMLタグからPタグまでの出力が、トレース情報12行目のwriteメソッド呼び出しになっています。
JSPコード16行目のパラメタ情報取り出し処理が、トレース情報20行目のgetParameterメソッド呼び出しになっています。

・ リクエスト後処理 23～30行目

_jspServiceメソッドが復帰した後で、各種後処理が行われていることがわかります。
HelloJSP.jspでjspDestroyメソッドを実装している場合は、IJServer停止時にこのメソッドが呼び出されます。

29.14.3 異常のあるサーブレットの例

異常のあるサーブレットを例として、トレース情報の見方について説明します。

■例:応答がなくなるFreezeServlet

```
 1:  import javax.servlet.http.*;
 2:  import javax.servlet.*;
 3:  import java.io.*;
 4:  public class FreezeServlet extends HttpServlet
 5:  {
 6:    public void service (HttpServletRequest request, HttpServletResponse response)
 7:    throws ServletException, IOException {
 8:      try {
 9:        response.setContentType("text/html; charset=iso-2022-jp");
10:        PrintWriter out = response.getWriter();
11:        FreezeClass fc=new FreezeClass();
12:        String outString = fc.getString();
13:        String tmpStr = request.getParameter("SampleParameter");
14:        out.println(outString+tmpStr);
15:      } catch (Exception e) {}
16:    }
17:  }
```

このサーブレットを実行して採取したトレース情報が以下であるときの説明をします。日時、スレッド名、および16進出力は省略しています。

```
1: in, FreezeServlet, service,
  PARAM:(RequestFacade):org.apache.catalina.connector.RequestFacade@1526e3:
  (ResponseFacade):org.apache.catalina.connector.ResponseFacade@ac2d3c
2: in, org.apache.catalina.connector.ResponseFacade, setContentType,
  PARAM:(String):text/html; charset=iso-2022-jp
3: in, org.apache.catalina.connector.ResponseFacade, isCommitted, PARAM:
4: out, org.apache.catalina.connector.ResponseFacade, isCommitted,
  RET:(boolean):false
5: out, org.apache.catalina.connector.ResponseFacade, setContentType, RET:void
6: in, org.apache.catalina.connector.ResponseFacade, getWriter, PARAM:
7: in, org.apache.catalina.connector.ResponseFacade, isFinished, PARAM:
8: out, org.apache.catalina.connector.ResponseFacade, isFinished,
  RET:(boolean):false
9: out, org.apache.catalina.connector.ResponseFacade, getWriter,
  RET:(CoyoteWriter):org.apache.catalina.connector.CoyoteWriter@19f9c7a
   （以降、出力なし）
```

この情報より、以下のことがわかります。

1. FreezeServlet.javaの9行目、10行目で呼び出しているHttpServletResponseのメソッドは、メソッド呼び出し時、メソッドからの復帰時にそれぞれのトレース情報が出力されていることから、呼び出された後、復帰していることがわかります。
2. トレース対象である13行目のHttpServletRequestのgetParameterメソッドのトレース情報が出力されていないことより、この行が実行されていないことがわかります。
3. 1、2より、FreezeServlet.javaの11行目、12行目のどちらかの処理が復帰していない、ということがわかります。

FreezeServletが正常に動作する場合には、上記のトレース情報に続いて以下の情報が出力されます。

```
10: in, org.apache.catalina.connector.RequestFacade, getParameter,
    PARAM:(String):SampleParameter
11: out, org.apache.catalina.connector.RequestFacade, getParameter, RET:(String):[]
・・・(略)・・・
12: out, FreezeServlet, service, RET:void
```

■例:出力が文字化けするCodeMissServlet

```
 1:  import javax.servlet.http.*;
 2:  import javax.servlet.*;
 3:  import java.io.*;
 4:  public class CodeMissServlet extends HttpServlet
 5:  {
 6:    public void service (HttpServletRequest request, HttpServletResponse response)
 7:    throws ServletException, IOException {
 8:      response.setContentType("text/html; charset=iso-2022-jp");
 9:      PrintWriter out = response.getWriter();
10:      out.println("<HTML><HEAD><TITLE>CodeMissServlet</TITLE></HEAD><BODY><H1>");
11:      String str=request.getParameter("param");
12:      out.println("data = "+ str );
13:      out.println("</H1><BR></BODY></HTML>");
14:    }
15:  }
```

下記のHTMLから“富士通太郎”と入力します。

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=shift_jis">
<TITLE>CodeMissServlet</TITLE>
</HEAD>
```

```
<BODY>
<FORM method="post" action="CodeMissServlet">
<INPUT name="param" size="30">
<INPUT type="submit" value="Submit">
<INPUT type="reset" value="Reset">
</FORM>
</BODY>
</HTML>
```

Webブラウザへの出力

```
data = ?x?m?????Y
```

採取したトレース情報が以下であるときの説明をします。日時、スレッド名、および16進出力は省略しています。

```
  1: in, CodeMissServlet, service,
     PARAM:(RequestFacade):org.apache.catalina.connector.RequestFacade@d41f3:
    (ResponseFacade):org.apache.catalina.connector.ResponseFacade@64dd79
  2: in, org.apache.catalina.connector.ResponseFacade, setContentType,
    PARAM:(String):text/html; charset=iso-2022-jp
  3: in, org.apache.catalina.connector.ResponseFacade, isCommitted, PARAM:
  4: out, org.apache.catalina.connector.ResponseFacade, isCommitted,
    RET:(boolean):false
  5: out, org.apache.catalina.connector.ResponseFacade, setContentType, RET:void
    ・・・(略)・・・
  6: in, org.apache.catalina.connector.RequestFacade, getParameter,
    PARAM:(String):param
    ・・・(略)・・・
  7: out, org.apache.catalina.connector.RequestFacade, getParameter,
    RET:(String):?x?m?????Y
  8: in, org.apache.catalina.connector.CoyoteWriter, println,
    PARAM:(String):data = ?x?m?????Y
    ・・・(略)・・・
  9: out, org.apache.catalina.connector.CoyoteWriter, println, RET:void
    ・・・(略)・・・
 10: out, CodeMissServlet, service, RET:void
```

このとき、引数および戻り値の情報の出力が有効になっていた場合、トレース情報の7行目、8行目において、引数および戻り値はそれぞれ次のように出力されます。
なお、getParameterメソッドの戻り値は、Webブラウザから受け取ったデータを%デコードして戻り値としています。
HEXvalue=[0095 0078 008e 006d・・・]は、Shift JISの富(0x9578)、士(0x8e6d)、通(0x92ca)を表しています。

```
  7 :(略)  RET:(String):?x?m?????Y HEX=0095 0078 008e 006d 0092 00ca 0091 00be 0098 0059
  8 :(略)  PARAM:(String):data = ?x?m?????Y HEX=0064 0061 0074 0061 0020 003d 0020 0095
       0078 008e 006d 0092 00ca 0091 00be 0098 0059
```

この値より、サーブレットのソースの12行目(トレース情報の8行目)においてPrintWriterオブジェクトのprintlnメソッドに引数として渡した文字列がUNICODEではなく、Shift JISであることがわかります。この例の場合、サーブレットで受け取ったコード系をUNICODEで処理したいため、たとえば、次のようにします。

- Interstage管理コンソールで、該当するWebアプリケーションの[環境設定]タブ > [エンコーディング]に入力文字コードを指定します。
  この例の場合には、HTMLで指定されている“shift_jis”を指定します。
  この方法では、サーブレットのソースの11行目(トレース情報の7行目)で取得した文字列はUNICODEに変換されています。

- サーブレットで文字コード変換をします。
  サーブレットのソースの11行目と12行目の間に以下の処理を加えます。

```
str=new String(str.getBytes("8859_1"), "shift_jis");
```

上記の処理を行うことによって、サーブレットのソースの12行目に正しくUNICODEの文字列が渡され、トレース情報の8行目の引数の値は以下のようになります。

```
PARAM:(String):data = 富士通太郎 HEX=0064 0061 0074 0061 0020 003d 0020 5bcc 58eb 901a 592a 90ce
```

29.15 Oracle Database使用時の異常

thinドライバでプラガブル・データベース(PDB)に接続できない場合

ドライバタイプが"thin"のJDBCデータソースを使用して以下の例外が出力される場合、PDBにSIDで接続できないことが原因です。

```
java.sql.SQLException: Listener refused the connection with the following error: ORA-12505, TNS:listener does not currently know of SID given in connect descriptor
```

この例外は、Interstageでの操作やアプリケーションの処理に応じて、以下の場所に出力されます。

- Interstage管理コンソールからの"DB接続テスト"が失敗する場合、Interstage管理コンソールの画面右下に表示されます。
- IJServerの起動に失敗する場合、IJServerのコンテナログに出力されます。
- アプリケーションの実行に失敗する場合、IJServerのコンテナログに出力されます。

thinドライバを使用する場合は、Oracle Net Servicesの構成ファイルであるlistener.oraファイルの"USE_SID_AS_SERVICE_*listener_name*"制御パラメータの値を"on"に設定し、Oracle Net ServicesがSIDをサービス名として解釈するように設定してください。

"USE_SID_AS_SERVICE_*listener_name*"制御パラメータの詳細については、データベースのマニュアルを参照してください。

第10部　移行

第30章　旧機能から新機能への移行方法

30.1 Servletサービス(Tomcat5.5ベースのサーブレット実行環境)への移行

Interstage Application Server/Interstage Web Server V9.0以降、Tomcat5.5ベースのサーブレット実行環境(以降、Tomcat5.5ベースのServletサービス)を提供しています。

Windows32 Linux32

本製品では、Tomcat4.1ベースのServletサービスと、Interstage Application Server V5.1以前のサーブレット実行環境であるTomcat3.1ベースのサーブレット実行環境(以降、Tomcat3.1ベースのServletサービス)を提供していません。
それぞれからTomcat5.5ベースのServletサービスに移行する場合の方法について説明します。

30.1.1 Tomcat3.1ベースのServletサービスからの移行について

「Tomcat3.1ベースのServletサービス」と「Tomcat5.5ベースのServletサービス」では運用方法が異なります。
「Tomcat5.5ベースのServletサービス」の操作は、Interstage管理コンソールまたはisj2eeadminコマンドを使用して行います。
Interstage管理コンソールについては「運用ガイド(基本編)」を、isj2eeadminコマンドについては「リファレンスマニュアル(コマンド編)」を参照してください。

配備方法

Interstage管理コンソールでIJServerワークユニットを作成した後、Webアプリケーションを配備しなおします。

環境定義ファイル

Tomcat3.1ベースのServletサービス固有の環境ファイルの移行については、「第32章 V5.1以前のServletサービス環境定義の移行」を参照してください。
その他の環境定義ファイルについては以下を参照してください。

Tomcat3.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
名前変換定義ファイル (注1)	・ [ワークユニット] > 「ワークユニット名」 > 「モジュール名」 > [名前変換]タブ
配備時の名前変換定義(interstage.xml)の<app-name>タグ	「Tomcat3.1ベースのServletサービス」では指定しても意味がありませんでしたが、「Tomcat5.5ベースのServletサービス」では名前変換定義情報の設定対象であるWebアプリケーションの名前を正しく指定してください。app-nameタグを指定しない場合や、タグに間違ったWebアプリケーション名を指定すると名前変換は動作しません。 app-nameタグに指定するWebアプリケーション名は以下のようにしてください。 ・ 以下の文字以外の文字が使用されていないこと 英数字、'+'、'-'、'.'、'_'、'$'、'/' ・ '.'を使用する場合は、他の文字と一緒に使用されていること

注1)

以下のファイルです。

Windows32

C:¥INTERSTAGE¥j2ee¥etc¥FJWebebeProperties.xml

Linux32

/opt/FJSVj2ee/etc/FJWebebeProperties.xml

8.0以前のInterstage HTTP Serverの資源を使用する場合 Windows32 Linux32

以下の操作により8.0以前のInterstage HTTP Serverの資源を使用する場合には、WebサーバからServletサービスへの通信を行うWebサーバコネクタの設定が必要です。

- 8.0以前でバックアップした資源をリストア

上記の場合、Interstage HTTP Serverの環境定義ファイル(httpd.conf)の末尾に以下に示す定義項目を追加してください。

Windows32

LoadModule ihs2_redirector2_module "C:/Interstage/F3FMjs5/gateway/ihs2/mod_ihs2_redirector2.so"

Linux32

LoadModule ihs2_redirector2_module "/opt/F3FMjs5/gateway/ihs2/mod_ihs2_redirector2.so"

ポイント

Interstage HTTP Serverの環境定義ファイル(httpd.conf)に以下のV5.1以前のサーブレット・ゲートウェイの定義が設定されている場合には、その定義情報を削除してください。以下にV5.1以前のサーブレット・ゲートウェイの定義について記載します。

Windows32

LoadModule jsvlt_module "C:/Interstage/F3FMjs2/gateway/jsgw_apapi_is.dll"
include "サーブレット・ゲートウェイ環境定義ファイルのフルパス名"

Linux32

LoadModule jsvlt_module /opt/FJSVjs2/gateway/jsgw_apapi_is.so
include "サーブレット・ゲートウェイ環境定義ファイルのフルパス名"

30.1.2 Servletサービス移行時の注意

Tomcat3.1ベースのServletサービス、またはTomcat4.1ベースのServletサービスから、Tomcat5.5ベースのServletサービスへ移行する場合の注意事項について説明します。
特に表中記載のないものについては、設定や動作上の差分はありません。

- Webアプリケーション環境定義ファイル(deployment descriptor)の記述形式
- サーブレット・マッピング
- JSPのimport
- 配備時のWebアプリケーション名
- HTTPトンネリングの移行
- マルチコンテナ機能の移行
- JSPのコンパイル
- Systemwalker Service Quality Coordinatorと連携したトランザクション内訳分析の対象

- Interstage管理コンソールのWebアプリケーションのモニタの対象
- JSPのリロード
- セション管理用CookieにSecure属性を常に付加する
- リクエストのエンコーディング
- HTTP TRACEメソッド
- 接続先コネクタの制限
- 同時処理数
- JSP Document(XMLベースのJSP)の属性の順番
- リクエストのURIに'¥'を含む場合

Webアプリケーション環境定義ファイル(deployment descriptor)の記述形式

Tomcat5.5ベースのServletサービスでは、Webアプリケーション環境定義ファイル(deployment descriptor)のタグは以下の方法でチェックします。

- Servlet2.2/Servlet2.3のdeployment descriptor

  DTD(文書型定義)ベース

- Servlet2.4のdeployment descriptor

  XMLスキーマベース

Tomcat3.1ベースのServletサービスから移行時の注意

Tomcat3.1ベースのServletサービスでは独自の方式でチェックを行っていたため、これまでチェックされていなかった内容がチェックされる場合があります。

Tomcat4.1ベースのServletサービスから移行時の注意

deployment descriptorをServlet2.4のものに修正した場合、チェック方法(チェック対象)が異なるため、これまで配備時にはチェックされていなかった内容がチェックされる場合があります。

タグの指定方法が誤っている場合には、Interstage管理コンソールで配備を行った際に、画面上のメッセージ領域にエラーメッセージを出力して配備を中断します。
また、IJServerの起動時にタグの指定方法の誤りが検出された場合には、IJServerのコンテナログに以下のエラーメッセージが出力されますので、「7.5 Webアプリケーション環境定義ファイル(deployment descriptor)」を参照して修正してください。
たとえば、servlet-mappingタグをservletタグよりも前に定義した場合にエラーとなります。

```
[ERROR] Digester - -Parse Error at line [行数] column [列数]: [原因]
```

Webアプリケーション環境定義ファイル(deployment descriptor)修正後、IDE(統合開発環境ツール)等に付属しているdeployment descriptorの検証機能を使用して、修正内容の妥当性を検証することを推奨します。
配備済みのアプリケーションのdeployment descriptorの記述形式を、Servlet2.2または2.3からServlet2.4の記述形式に変更する場合は、アプリケーションを配備しなおしてください。

サーブレット・マッピング

Tomcat3.1ベースのServletサービスから移行時の注意

デフォルトでは、サーブレット・マッピングの定義を行わない、以下のURLの形式でのサーブレットの呼び出しはできません。以下のURLの形式でサーブレットの呼び出しを行っていた場合は、Webアプリケーション環境定義ファイルにサーブレット・マッピングの定義を行う必要があります。

```
http://サーバホスト名:ポート番号/Webアプリケーション名/servlet/サーブレット名
```

IJServerワークユニットの環境設定で、サーブレット・マッピングの定義を行わずに、上記URLの形式で呼び出すことができる設定に切り替えることができますが、以下の理由から推奨しません。

- クラス名が分かればすべてのサーブレットが動作できてしまう(セキュリティ)。
- Filterアプリケーションの呼び出しがサポートされない。
- Servletの仕様外の機能である(移植性)。

Tomcat4.1ベースのServletサービスから移行時の注意

特にありません。

JSPのimport

Tomcat5.5ベースのServletサービスでは、JSP1.2/2.0の仕様の以下のJavaパッケージのクラス以外は、importなしにクラス名のみで直接スクリプトレット使用はできません。

- javax.servlet
- javax.servlet.http
- javax.servlet.jsp

Tomcat3.1ベースのServletサービスから移行時の注意

仕様外のクラスを使用している場合は、下記の対処を行ってください。

Tomcat4.1ベースのServletサービスから移行時の注意

JavaVMオプションの以下のプロパティは使用できません。

```
-Dcom.fujitsu.interstage.j2ee.ijserver.jspImplicitImport=true(英大文字・英小文字の区別なし)
```

これにより下記のクラスも使用できなくなります。

- java.io.*
- java.util.*

仕様外のクラスを使用している場合は、下記の対処を行ってください。

Tomcat5.5ベースのServletサービスで仕様外のクラスを使用する場合は、以下のいずれかの対処を行ってください。

- JSPのPageディレクティブで正しくimport宣言を行う。
- Servlet2.4(JSP2.0)の機能を使用する。
  1. deployment descriptor(web.xml)を、Servlet2.4用に修正します。(Servlet2.3以前のdeployment descriptorの場合)

     詳細は「7.5 Webアプリケーション環境定義ファイル(deployment descriptor)」を参照してください。

2. JSPの必要なimport文を含んだファイルを用意します。

header.jspf

```
<%@ page import="java.io.*,java.util.*" %>
```

3. deployment descriptor(web.xml)に<jsp-config>タグを追加し、対象とするJSPに対して、上記header.jspfを設定します。

この例では、header.jspfファイルをアプリケーションのWEB-INF/jspfディレクトリ配下に格納し、すべてのJSPに対して設定しています。

deployment descriptor(web.xml)

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
        http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
・・・
<jsp-config>
  <jsp-property-group>
    <url-pattern>*.jsp</url-pattern>
    <include-prelude>/WEB-INF/jspf/header.jspf</include-prelude>
  </jsp-property-group>
</jsp-config>
・・・
</web-app>
```

配備時のWebアプリケーション名

Tomcat3.1ベースのServletサービスから移行時の注意

Webアプリケーションの配備を行う場合、Webアプリケーション名の省略値には以下の値が使用されます。

- EARファイルの場合
  - application.xmlのcontext-rootに値が設定されている場合には、その設定されている名前が使用されます。
  - application.xmlのcontext-rootに値が設定されていない場合には、WARファイル名から拡張子を除いた名前が使用されます。
- WARファイルの場合

  WARファイル名から拡張子を除いた名前が使用されます。

Tomcat4.1ベースのServletサービスから移行時の注意

特にありません。

HTTPトンネリングの移行

Tomcat4.1ベースのServletサービスから移行時/Tomcat3.1ベースのServletサービスから移行時の共通の注意

CORBA_ORB_init()関数でのHTTPトンネリング機能の設定は有効となりません。
HTTPトンネリング機能を使用する場合は、IJServerのJavaVMオプション、またはFJjndi.propertiesファイルで、環境プロパティとして「HTTPGW」を設定してください。
環境プロパティ「HTTPGW」の値と設定内容については、「4.1.1 J2EEアプリケーションクライアント」を参照してください。
注) アプリケーションでのJNDI機能の使用有無にかかわらず設定してください。
なお、CORBA_ORB_init()関数で設定していた場合でも、アプリケーションの修正は必要ありません。

Tomcat3.1ベースのServletサービスから移行時の注意

HTTPトンネリング機能は、Webアプリケーションのみ運用のタイプのIJServerで使用可能です。以下のタイプのIJServerではHTTPトンネリング機能は使用できません。

- WebアプリケーションとEJBアプリケーションを同一JavaVMで運用
- WebアプリケーションとEJBアプリケーションを別JavaVMで運用

Tomcat4.1ベースのServletサービスから移行時の注意

特にありません。

マルチコンテナ機能の移行

Tomcat3.1ベースのServletサービスから移行時の注意

マルチコンテナ機能を使用していた場合には、WebアプリケーションをIJServerへ配備し直す必要があります。「第32章 V5.1以前のServletサービス環境定義の移行」の「サーブレット・ゲートウェイ環境の移行」を参照してください。

Tomcat4.1ベースのServletサービスから移行時の注意

特にありません。

JSPのコンパイル

Java言語仕様に準拠していないコードや推奨されないコードを含むJSPは、以前のバージョンからの移行により、コンパイルに失敗したり、実行結果が変わったりすることがあります。
移行時にはJSPを見直し、Java言語仕様に従っているか確認してください。

例

JSP内のJavaコードで、ローカル変数名に「org」を使用している。
⇒ユニークなパッケージ名の先頭に使用される文字列(「org」や「com」など)を、ローカル変数名に使用すべきではありません。

Systemwalker Service Quality Coordinatorと連携したトランザクション内訳分析の対象

Tomcat3.1ベースのServletサービスから移行時の注意

未サポートです。

Tomcat4.1ベースのServletサービスから移行時の注意

[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [Servletコンテナ設定]で、「マッピングがなくてもサーブレットが動作する」が設定されている場合、以下の形式で呼び出したサーブレットの情報は、すべてまとめて「invoker」として出力されていましたが、この情報は出力されなくなります。

```
http://servername[:port]/Webアプリケーション名/servlet/サーブレットクラス名
```

Interstage管理コンソールのWebアプリケーションのモニタの対象

Tomcat3.1ベースのServletサービスから移行時の注意

未サポートです。

Tomcat4.1ベースのServletサービスから移行時の注意

[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [Servletコンテナ設定]で、「マッピングがなくてもサーブレットが動作する」が設定されている場合、以下の形式で呼び出したサーブレットの情報は、すべてまとめて「invoker」として出力されていましたが、この情報は出力されなくなります。

```
http://servername[:port]/Webアプリケーション名/servlet/サーブレットクラス名
```

JSPのリロード

クラスのオートリロード機能とは別に設定可能となりました。
[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [JSPのリロード]で設定可能です。

セション管理用CookieにSecure属性を常に付加する

Tomcat5.5ベースのServletサービスでは、Interstage管理コンソールより以下で設定を行ってください。

- [システム] > [ワークユニット] > 「ワークユニット名」 > 「Webアプリケーション名」 > [環境設定]タブ > [コンテキスト設定] > [セション] > クッキーにSecure属性を常に付加する

既存通りJavaVMオプションとして定義することも可能(※)ですが、Interstage管理コンソールからの設定を推奨します。
※) いずれかで設定されていれば有効となります。

リクエストのエンコーディング

Servletサービスのバージョンによって、リクエストを処理するエンコーディングにデフォルトでは以下の動作差分があります。

対象	Tomcat4.1ベースのServlet サービス	Tomcat5.5ベースのServlet サービス
リクエストURI	UTF-8 (注1)	ISO-8859-1(設定可能)
GETメソッド時のリクエストパラメタ(クエリストリング)	リクエストボディを処理するエンコーディング (注2)	以下から選択可能。 ・リクエストURIを処理するエンコーディング ・リクエストボディを処理するエンコーディング (注2)
POSTメソッド時のリクエストパラメタ(リクエストボディ)	リクエストボディを処理するエンコーディング (注2)	リクエストボディを処理するエンコーディング (注2)
リクエストボディを処理するReader ServletRequest#getReader()	リクエストボディを処理するエンコーディング (注2)	リクエストボディを処理するエンコーディング (注2)

注1) PG48587の修正を適用した場合は以下の順で処理されます。

1. UTF-8
2. OSのデフォルトエンコーディング
3. 自動判定(Java言語の自動判定機能(JISAutoDetect)を使用)

注2)「リクエストボディを処理するエンコーディング」は以下が該当します。

- リクエストのContent-Typeヘッダで示されるcharset
- Webアプリケーションでリクエストに setCharacterEncodingメソッドで設定したエンコーディング
- Webアプリケーション環境設定で「エンコーディング」に指定したエンコーディング

Tomcat5.5ベースのServletサービスでTomcat4.1ベースのServletサービスと同じ動作にする場合は以下の設定を行ってください。

- [ワークユニット] > 「ワークユニット名」 > [Servletコンテナ設定] > [リクエストURIのエンコーディング]に「UTF-8」
- [ワークユニット] > 「ワークユニット名」 > [Servletコンテナ設定] > [リクエストボディ処理のエンコーディングをクエリパラメタに使用する]に「する」

また、Tomcat4.1ベースのServletサービスとTomcat5.5ベースのServletサービスでは、Webアプリケーション環境設定で「エンコーディング」を指定した場合の動作に以下の改善が行われています。
これにより、リクエストのContent-Typeヘッダに正しくエンコーディング「charset=」が指定されている場合は、その値が有効となります。

(1)Webアプリケーション環境設定「エンコーディング」	(2)リクエストのContent-Typeのcharset	(3)アプリケーションによるリクエストへのsetCharacterEncoding	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
指定あり	あり	あり	(3)が有効	(3)が有効
		なし	(1)が有効	(2)が有効
	なし	あり	(3)が有効	(3)が有効
		なし	(1)が有効	(1)が有効
指定なし	あり	あり	(3)が有効	(3)が有効
		なし	(2)が有効	(2)が有効
	なし	あり	(3)が有効	(3)が有効
		なし	デフォルト(ISO-8859-1)	デフォルト(ISO-8859-1)

HTTP TRACEメソッド

Tomcat5.5ベースのServletサービスでは、HTTP TRACEメソッドはセキュリティ上の理由から使用できません。
なお、HTTP TRACEメソッドは、Webサイトの利用者が通常使用する機能ではありません。

接続先コネクタの制限

Tomcat4.1ベースのServletサービスで、Webサーバコネクタの証明書のシリアル番号と発行者識別DNを指定して接続先コネクタの制限を行っていた場合、Tomcat5.5ベースのServletサービスでは、Interstage管理コンソールの以下の設定で、Webサーバコネクタ とServletコンテナ間の通信で使用するSSL定義を適切に設定して、接続先コネクタの制限を行ってください。

・ [システム] > [セキュリティ] > [SSL]の新規作成、または、参照・変更の詳細設定項目である「[認証局証明書のニックネーム]の選択」

同時処理数

Tomcat5.5ベースのServletサービスでは、同時処理数に指定した値のうちの1つは、クライアントシステムからのリクエスト受け付け用として、Servletコンテナの内部処理で使用されます。そのため、指定できる最小値は2になります。

JSP Document(XMLベースのJSP)の属性の順番

JSP Documents(XML形式のドキュメント)の場合、XMLでは属性の順番が保証されないため、依存するアプリは移行により動作しない場合があります。
問題が発生する場合は、依存しないよう修正してください。

例

修正前

```
<vsi:loop begin="%= last - s %" end="%= last++ %" />
```

修正後

```
<vsi:loop begin="%= last - s %" end="%= last %" /><jsp:scriptlet>last++;</jsp:scriptlet>
```

リクエストのURIに'¥'を含む場合

Tomcat5.5ベースのServletサービスでは、リクエストのURIに'¥'(エンコードされている場合も該当)を含む場合、Tomcat4.1ベースのServletサービス(PG56136の修正未適用)では呼び出し可能であったコンテンツが呼び出しできなくなることがあります。

30.1.3 アプリケーションの非互換一覧

Servletサービスにおけるアプリケーション動作の非互換項目を以下に示します。該当する機能をWebアプリケーションで使用している場合には、Webアプリケーションを修正してください。
Servlet/JSP仕様の差分については、Java(TM) Servlet SpecificationおよびJavaServer Pages(TM) Specificationをあわせて参照してください。
またシステム移行の際には、運用前に業務の観点で十分な動作検証を実施してください。

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
1	カスタムタグのTag#release()メソッド	カスタムタグの開発で実装するjavax.servlet.jsp.tagext.Tagインタフェースのpublic void release()メソッドは、カスタムタグを使用したページの呼出しが終わるごとに呼び出されます。	カスタムタグの開発で実装するjavax.servlet.jsp.tagext.Tagインタフェースのpublic void release()メソッドは、Webアプリケーション終了時に呼び出されます。 V5.1以前と同じタイミングで処理を行う場合には、カスタムタグを以下のように修正してdoFinally()メソッドに処理を記述してください。 javax.servlet.jsp.tagext.TryCatchFinallyインタフェースをimplementsして、以下のメソッドを実装します。 public void doCatch(Throwable t) public void doFinally()	

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
2	Webブラウザに返却されるステータスコード	Webアプリケーション環境定義ファイルに<error-page>タグが設定されている場合は、ステータスコードは200が返却されます。 また、JSPのerrorPage属性に指定したJSPエラーページでisErrorPage属性に"true"が設定されている場合にも、ステータスコードは200です。	Webアプリケーション環境定義ファイルに<error-page>タグが設定されている場合にも、発生した現象のステータスコードがそのまま返却されます。 また、JSPのerrorPage属性に指定したJSPエラーページでisErrorPage属性に"true"が設定されている場合には、ステータスコードは200です。	Webアプリケーション環境定義ファイルに<error-page>タグが設定されている場合にも、発生した現象のステータスコードがそのまま返却されます。 また、JSPのerrorPage属性に指定したJSPエラーページでisErrorPage属性に"true"が設定されている場合には、ステータスコードは500です。
3	JSPの呼出し	JSPファイルを更新した場合、前回コンパイルされたJSPファイルとファイル更新日時が異なる場合に反映されます。(反映はServletコンテナ再起動、またはリロード機能有効時) Windows32/64 大文字・小文字は区別されません。	JSPまたはJSPから静的includeしているファイルを更新した場合、前回コンパイルされた日時よりも新しい日付のJSPファイルの場合に反映されます。 「使用上の注意」の「Servletサービスの注意事項」も参照してください。 Windows32/64 大文字・小文字は区別されます。	
4	Webアプリケーション環境定義ファイルの<servlet>タグ内の<load-on-startup>タグ	0を指定した場合は、サーブレット・コンテナ起動時にロードされません。	0を指定した場合は最後にロードされます。 負の数を指定した場合はWebアプリケーション起動(活性化)時にロードされません。	0を指定した場合は最初にロードされます。 負の数を指定した場合はWebアプリケーション起動(活性化)時にロードされません。
		-2147483648～2147483647以外の値を指定した場合はWebアプリケーションの起動は失敗します。	-2147483648～2147483647以外の値を指定した場合もWebアプリケーションの起動は成功します。	
5	Webブラウザにスタックトレースを表示	サーブレット・コンテナ環境定義ファイルのContextManagerタグのerrResponse属性で、リクエストの処理でErrorやExceptionが発生した場合、Webブラウザにスタックトレースを表示するかどうかを指定します。	内部の情報が漏洩する可能性があるため、リクエストの処理でErrorやExceptionが発生した場合、Webブラウザにスタックトレースを表示しません。	
6	Webアプリケーション環境定義ファイルの<servlet-mapping>タグ内の<servlet-name>タグ	指定していない名前を記述した場合は、無効となります。	指定していない名前を記述した場合は、Webアプリケーションの起動に失敗します。	
7	Webアプリケーション環境定義ファイルの	welcome fileを省略した場合、サーブレット・コンテナ環境定義ファイルで記述する	welcome fileを省略した場合、デフォルト設定が使用されます。デフォルト設定のファイルは以下の順番で有効になります。	

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
	<welcome-file-list>タグ内の<welcome-file>タグ	Context定義のdirList属性の指定により、その実体となるディレクトリ配下のディレクトリやファイルの一覧、またはステータスコード404(ファイルが存在しない)が表示されます。	1. index.html 2. index.htm 3. index.jsp welcome file(省略時はデフォルト設定)に該当するファイルがない場合は、Interstage管理コンソールの[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [詳細設定] > [Servletコンテナ設定] > [ファイルの一覧表示]に設定されている値によって、ステータスコード404(ファイルが存在しない)、またはその実体となるディレクトリ配下のディレクトリやファイルの一覧が表示されます。	
8	Webアプリケーション環境定義ファイルの<env-entry>タグ内の<env-entry-value>タグ	<env-entry-value>タグを省略して環境エントリをlookupした場合にはjavax.naming.NamingExceptionまたはそのサブクラスが投げられます。	<env-entry-value>タグを省略した場合には、デフォルト値が設定されます。(注1)	
			Tomcat3.1ベースのServletサービスと同じ動作をさせる場合には、Interstage管理コンソールの[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [JavaVMオプション]に「-Dcom.fujitsu.interstage.jservlet.naming.defaultenventry=FALSE」を設定してください。	Tomcat3.1ベースのServletサービスと同じ動作をさせる場合には、deployment descriptor(web.xml)から、該当する<env-entry>タグを削除してください。
9	Webアプリケーション環境定義ファイルの<taglib-location>タグで存在しないパスを指定した場合	無効となります。	Webアプリケーションの起動に失敗します。	無効となります。
10	WEB-INF配下にXMLの文法に誤りのあるTagLibraryDescriptorファイルが存在した場合	「XML parsing error on file」がコンテナの標準出力、標準エラー出力のログに出力されるが、サーブレット・コンテナの起動に成功します。	メッセージ「IJServer15111」がコンテナログに出力され、Webアプリケーションの起動に失敗します。	メッセージは出力されますが、Webアプリケーションの起動は成功します。
11	Webアプリケーション環境定義ファイルの<servlet-mapping>タグ内の<url-pattern>タグ	特定の拡張子をもつURLを指定した場合(例:*.jsp)は、URL内の「/」と「/」で囲まれた箇所もマッピング対象となります。	特定の拡張子をもつURLを指定した場合(例:*.jsp)は、URL内の「/」と「/」で囲まれた箇所もマッピング対象とならず、最後の「/」以降がマッピング対象となります。	
12	Webアプリケーション環境定義ファイルの<welcome-file-list>タグ内の<welcome-file>タグに指定され	URLが「/」で終わっていない場合、URLの最後に「/」を追加した値をLocationヘッダに指定し、ステータスコード:302を返します。	URLが「/」で終わっていない場合、URLの最後に「/」を追加した値をLocationヘッダに指定し、ステータスコード:302を返します。	URLが「/」で終わっていない場合、URLの最後に「/」を追加した値をLocationヘッダに指定し、ステータスコード:302を返します。

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
	たファイルを表示する場合の動作	URLが「/」で終わっている場合、<welcome-file>タグに指定されたコンテンツに内部でforwardします。そのため、指定したコンテンツ(JSP等)先で取得できるリクエストのURLなどのパス情報は、forwardされたコンテンツのものとなります。	URLが「/」で終わっている場合、<welcome-file>タグに指定されたファイル名を追加したURLをLocationヘッダに指定し、ステータスコード:302を返します。そのため、指定したコンテンツ先で取得できるリクエストのURLなどのパス情報は、ブラウザが再度リクエストを発行(リダイレクト)した時の情報となります。	URLが「/」で終わっている場合、<welcome-file>タグに指定されたコンテンツを直接処理して返します。そのため、指定したコンテンツ先で取得できるリクエストのURLなどのパス情報は、ブラウザからリクエストされてきたパスのままとなり、<welcome-file>タグで指定したコンテンツを含むパスとはなりません。(注3)
13	複数のAccept-Languageヘッダを持つリクエストを受け付けた場合のHttpServletRequest#getHeaders()メソッドの動作(注2)	version属性(※)に4.1を指定または省略している場合「,」で区切られた値を1つの要素としてEnumerationが返されます。 例) ja, es version属性(※)に5.0または5.1を指定している場合 1つ1つの値が1つの要素としてEnumerationが返されます。 例) ja es ※サーブレット・コンテナ環境定義ファイルのContextManagerタグ内	「,」で区切られた値を1つの要素としてEnumerationが返されます。 例) ja, es	
14	レスポンスのデフォルトLocale	"en"です。	"en_US"です。	デフォルトではサーバのLocaleとなります。
15	レスポンスのコミット後またはgetWriterメソッド呼出し後の、setLocaleメソッドまたはsetContentTypeメソッド呼出しのレスポンスのエンコーディングへの反映 (注4)	「setLocaleメソッドまたはsetContentTypeメソッド呼出しのレスポンスのエンコーディングへの反映」を参照してください。		
16	レスポンスのsetLocaleメソッドまたはsetContentTypeメソッド(引数のmimeタイプ文字列にcharset属性を含む) (注5)によりレスポンスの	レスポンスのWriterによるデータの書き出しには反映されますが、レスポンスのContent-Typeヘッダにはcharset属性は付加されません。		Servletの仕様に従い、レスポンスのWriterによるデータの書き出し、Content-Typeヘッダのcharset属性ともに、先に設定済みのエンコーディングが反映されます。

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
	エンコーディングを設定済みで、setContentTypeメソッドをcharset属性を含まないmimeタイプ文字列を指定して呼び出した場合の動作 (注6)			
17	レスポンスのsetContentTypeメソッドをcharset属性ありのmimeタイプ文字列を指定して呼び出した (注5)後の、setLocaleメソッドのレスポンスのエンコーディングへの反映 (注6)	反映されます。		Servletの仕様に従い、反映されません。
18	setLocaleメソッドまたはsetContentTypeメソッドでエンコーディングを指定しない場合のレスポンスのエンコーディング	レスポンスのWriterによるデータの書き出しはISO-8859-1となりますが、レスポンスのContent-Typeヘッダに、charset属性は付加されません。		レスポンスのWriterによるデータの書き出し、およびContent-Typeヘッダのcharset属性はISO-8859-1となります。
19	getWriterでcharsetを指定する場合のエラーページ処理のエンコーディングへの反映	getWriterで指定したcharsetの内容が、エラーページ処理に引き継がれます。	getWriterで指定したcharsetの内容が、エラーページ処理に引き継がれません。	getWriterで指定したcharsetの内容が、エラーページ処理に引き継がれます。
20	JSPのpageディレクティブで明示的にpageEncodingまたはcontentType属性を指定しない場合のデフォルトエンコーディング	レスポンスのデータの書き出しはISO-8859-1、Content-Typeヘッダは"text/html"となります。	レスポンスのデータの書き出しはISO-8859-1、Content-Typeヘッダは"text/html;charset=ISO-8859-1"となります。	
21	サーブレットのserviceが、永久的に使用不能な状態であること示すUnavailableExceptionをthrowした時の動作	Servlet2.3では規定されていませんが、基本的にステータスコード500を返却。	Servlet2.3では規定されていませんが、基本的にステータスコード503を返却。	ステータスコード404を返却。

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
22	deployment descriptor(web.xml)のtaglibタグ	<taglib>タグは、<web-app>タグの直下に記載します。	<taglib>タグは、<web-app>タグの直下に記載します。	<taglib>タグは、<jsp-config>タグ配下に記載します。
23	server.xml定義ファイルへのrealmの設定	ー	Tomcat4.1と同様のrealmタグを設定することができます。	できません。ディレクトリサービスを使用してください。
24	「コンテキストの共有」有効時のセション	ー	Webアプリケーション環境設定の「コンテキストの共有」を「する」に設定し、Webアプリケーションから別のWebアプリケーションに、RequestDispatcherを使用してディスパッチした場合、ディスパッチ前と後で使用するセションは同じとなります。	Webアプリケーション環境設定の「コンテキストの共有」を「する」に設定し、Webアプリケーションから別のWebアプリケーションに、RequestDispatcherを使用してディスパッチした場合、Servletの仕様に厳密に従い、セションは別のものとなります。セションの属性は共有されません。ディスパッチ前のセションの情報(属性)をディスパッチ後に使用したい場合は、リクエストの属性として(ServletRequest#setAttributeメソッド等を使用して)受渡を行う等、アプリケーションで対応してください。
25	HttpSessionのgetAttribute(null)呼出し時	NullPointerExceptionとなります。	nullが返却されます。	NullPointerExceptionとなります。 Servletの仕様では、HttpSessionのsetAttributeメソッドのキーにnullは指定できません。 getAttributeしても値の取得はできませんので、キーがnullの場合は呼び出さないようにしてください。
26	JSPのコンパイル<jsp:useBean>タグのclass属性	JSPの仕様に沿っていなくてもコンパイル/実行できる場合がありました。		JSPの仕様に沿っていない場合は動作しない可能性があります。 仕様に沿うよう修正することを推奨します。 以下のような修正が必要です。 フルクラス名で指定する publicなクラスを指定する publicなデフォルトコンストラクタがあるクラスを指定する 修正しない場合は、事前に充分な動作検証を行ってください。 詳細はJSP仕様を参照してください。

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
27	HttpSessionListenerのsessionDestroyed	セションが破棄された後に呼び出されます。		セションが破棄される前に呼び出されます。 詳細はServlet仕様を参照してください。
28	HttpSessionBindingListenerのvalueBound	オブジェクトがHttpSessionのgetAttributeで取得できる状態になってから呼び出されます。		オブジェクトがHttpSessionのgetAttributeで取得できる状態になる前に呼び出されます。 またこの影響により、HttpSessionのsetAttributeメソッドですでに存在するキーで別のオブジェクトをsetした場合、先に新しいオブジェクトのvalueBoundメソッド、後からもとのオブジェクトのvalueUnboundメソッドが呼び出されます。 詳細はServlet仕様を参照してください。
29	Webアプリケーション環境定義ファイルのタグの順番	<env-entry>タグ配下は以下の順で記載します。 <env-entry> <env-entry-name> entry-name </env-entry-name> <env-entry-value> entry-value </env-entry-value> <env-entry-type> entry-type </env-entry-type> </env-entry>		<env-entry>タグ配下は以下の順で記載します。 <env-entry> <env-entry-name> entry-name </env-entry-name> <env-entry-type> entry-type </env-entry-type> <env-entry-value> entry-value </env-entry-value> </env-entry>
30	JSPのタグの書式	以下の例のように、タグ～閉じタグ(ETag)間に子タグやBODYがなく、改行などがあっても動作する場合がありました。 例: <jsp:forward page="next.jsp" > </jsp:forward>		JSP2.0では許可される書式が明示され、左記のような記述はコンパイル時にエラーとなります。JSPの仕様に従い、以下のようにしてください。 例: <jsp:forward page="next.jsp" ></ jsp:forward> または <jsp:forward page="next.jsp" /> 詳細はJSP2.0またはJSP2.1仕様を参照してください。
31	JSPのBeanのプロパティ値がnullの場合の動作	JSPのBeanのプロパティ値がnullの場合、<jsp:getProperty>アクションの出力は""(空文字列)となります。 例:以下のようなJSPで、プロパティ"age"値がnullの場合		JSPの仕様に従い、出力は"null"となります。 以前のServletサービスと同

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
		<jsp:useBean id="eb" class="pkg.EmptyBean" scope="session" /> <jsp:getProperty name="eb" property="age" />		じ出力としたい場合は、以下のいずれかを行ってください。 プロパティの初期値を""とするなど、JSPの仕様に従うようにアプリケーションを修正する。(推奨) Interstage管理コンソールの[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [JavaVMオプション]に「-Dcom.fujitsu.interstage.jservlet.jsp.bean.getProperty.UseEmptyStringForNull=true」を設定する。
32	Webアプリケーション環境定義ファイルの<security-constraint>タグ内の<auth-constraint>タグ内の<role-name>タグ	"*"を指定する場合、<security-role>タグに有効な<role-name>タグが定義されていなければリソースにアクセスできません。	"*"を指定する場合、<security-role>タグを定義しなくてもリソースにアクセスできます。	"*"を指定する場合、<security-role>タグに有効な<role-name>タグが定義されていなければリソースにアクセスできません。
33	JSPのファイルエンコーディング	JSP1.2に従い、JSPファイルを読み込むエンコーディングは、コンパイル単位でした。include先のJSPファイルも、include元とおなじエンコーディングで読み込まれます。include先のJSPがinclude元のJSPファイルのエンコーディングに依存していた場合、Tomcat5.5ベースのServletサービスでは、include先のJSPファイルにエンコーディングを明に設定することで、同様の動作とすることが可能です。 例:deployment descriptor(web.xml)をServlet2.4のものに修正し、JSPプロパティグループで指定する。 <jsp-property-group> <url-pattern>/ja/*</url-pattern> <page-encoding>Shift_JIS</page-encoding> </jsp-property-group> 詳細は、JSP2.0の仕様(JavaServer Pages 2.0 Specification)を参照してください。		JSP2.0に従い、JSPファイルを読み込むエンコーディングは、ファイル単位となります。include先のJSPファイルを読み込むエンコーディングは、include元のJSPファイルを読み込むエンコーディングの影響を受けません。(デフォルトISO-8859-1)
34	JSPでカスタムタグライブラリ使用時の動作	TLD(Tag Library Description file)にて、対象のタグのbody-contentに「TAGDEPENDENT」(大文字)指定でも動作可能です。 XML構文のJSP(JSP Documents)の場合、TLD(Tag Library Description file)にて、対象のタグのbody-contentに「tagdependent」(「TAGDEPENDENT」)を指定し、タグのボディで<jsp:text>などを使用した場合、ボディは処理された後にタグライブラリに渡されます。		TLD(Tag Library Description file)にて、対象のタグのbody-contentに「TAGDEPENDENT」(大文字)は指定できません。仕様に従い、「tagdependent」(小文字)を指定してください。 XML構文のJSP(JSP Documents)の場合、TLD(Tag Library

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
				Description file)にて、対象のタグのbody-contentに「tagdependent」を指定し、タグのボディで<jsp:text>などを使用した場合、ボディは処理されずにそのままタグライブラリに渡されます。仕様通りの正しい動作です。 Tomcat3.1ベース/Tomcat4.1ベースのServletサービスと同じ動作をさせたい場合は、TLD(Tag Library Description file)にて、body-contentに「JSP」を指定してください。 なおServletの仕様に従い、参照するTLDファイルは必ずWebアプリケーション内に存在している必要があります。
35	サーブレットコンテキストの初期化パラメタ	サーブレットコンテキストの初期化パラメタ(<context-param>タグ)の複数指定時、重複したパラメタ名(<param-name>タグ)がある場合、最後に指定したものが有効となります。		重複したパラメタ名は指定できません。定義エラーとなります。
36	無効なセションのgetLastAccessedTimeメソッド呼出し	無効なセション(破棄やタイムアウトしたセション)のgetLastAccessedTimeメソッドを呼出し可能です。		無効なセション(破棄やタイムアウトしたセション)のgetLastAccessedTimeメソッドを呼び出した場合、IllegalStateExceptionとなります。
37	Webアプリケーション環境定義ファイルの<listener-class>タグ	<listener-class>タグに存在しないクラスを指定した場合は、コンテナログにエラーは出力されますがアプリケーションの起動は成功します。	<listener-class>タグに存在しないクラスを指定した場合は、IJServer起動時にコンテナログにエラーが出力され、アプリケーションの状態は"非活性"になります。また、そのアプリケーションにアクセスした場合にはステータスコード404(ファイルが存在しない)が表示されます。	
38	ServletRequestのgetServerName	リクエストを受信したサーバのホスト名。 CGI変数のSERVER_NAMEと同じ値が返ります。		リクエストを受信したサーバのホスト名。 リクエストヘッダのHostヘッダの値が返ります。Hostヘッダがない場合は、CGI変数のSERVER_NAMEと同じ値が返ります。
39	ServletRequestのgetServerPort	リクエストを受信したサーバのポート番号。 CGI変数のSERVER_PORTと同じ値が返ります。		リクエストを受信したサーバのポート番号。 リクエストヘッダのHostヘッダの値が返ります。Hostヘッダがない場合は、CGI変数のSERVER_PORTと同じ値が返ります。
40	HttpServletResponseのsendRedirect	指定された相対URLを用いて、クライアントに一時的なリダイレクトレスポンスを送信します。		指定された相対URLを用いて、クライアントに一時的なリダイレクトレスポンスを送信します。

No.	機能	Tomcat3.1ベースのServletサービス	Tomcat4.1ベースのServletサービス	Tomcat5.5ベースのServletサービス
		Servletコンテナによる絶対URLへの変換は、CGI変数のSERVER_NAMEおよびSERVER_PORTと同じ値が使用されます。		Servletコンテナによる絶対URLへの変換は、リクエストヘッダのHostヘッダの値が使用されます。Hostヘッダがない場合は、CGI変数のSERVER_NAMEおよびSERVER_PORTと同じ値が使用されます。

注1)

以下のデフォルト値が設定されます。

env-entry-type	エントリ値
java.lang.Boolean	Boolean.FALSE
java.lang.Byte	値0のByteオブジェクト
java.lang.Character	値0のCharacterオブジェクト
java.lang.String	なし 注) javax.naming.NamingExceptionまたはそのサブクラスが投げられます。
java.lang.Short	値0のShortオブジェクト
java.lang.Integer	値0のIntegerオブジェクト
java.lang.Long	値0のLongオブジェクト
java.lang.Float	値0のFloatオブジェクト
java.lang.Double	値0のDoubleオブジェクト

注2)

例

Accept-Language: ja
Accept-Language: es

注3)

Tomcat4.1ベースのServletサービスからの移行の場合、<welcome-file>でディレクトリ階層の異なるパス(途中に'/'を含むパス)を指定し、そのパスで示されるコンテンツから相対パスのリンクがあると、リンク先が意図したURIとならず、コンテンツが参照できない場合があります。以下のどちらかの対応を行ってください。

- <welcome-file>でディレクトリ階層の異なるコンテンツを指定しない。
- <welcome-file>で指定したページ中のリンクについて、コンテキストパスを含むパスとする。

例

<welcome-file>が「/jsp/welcome.jsp」、Webアプリケーション名が「sample」、JSPからのリンクが「../next.jsp」の場合、リンク先を、「/sample/next.jsp」とする。

動的コンテンツ(JSPやServlet)の場合は、以下のようにコンテキストパスを動的に取得することも可能です。(推奨)

- リンク先を、request.getContextPath()+"/next.jsp" とする。

※実際には、必要に応じてHttpServletResponseのencodeURLメソッド等をあわせて使用してください。

注4)

仕様に従っていないアプリケーションの場合であり、記載のTomcat3.1ベース/Tomcat4.1ベースのServletサービスの動作が保障されるものではありません。

注5)

JSPのpageディレクティブでpageEncoding属性を指定している場合、または、contentType属性にcharsetを指定している場合も含まれます。

注6)

No.15が前提です。

setLocaleメソッドまたはsetContentTypeメソッド呼出しのレスポンスのエンコーディングへの反映

		Tomcat3.1ベースのServletサービス		Tomcat4.1ベースのServletサービス		Tomcat5.5ベースのServletサービス	
getWriterメソッド	レスポンスのコミット	Content-Typeヘッダのcharset属性	データの書き出し	Content-Typeヘッダのcharset属性	データの書き出し	Content-Typeヘッダのcharset属性	データの書き出し
呼出し前	コミット前	反映される	反映される	反映される	反映される	反映される	反映される
	コミット後	反映されない	反映される	反映されない	反映されない	反映されない	反映されない
呼出し後	コミット前	反映される	反映されない	反映される	反映される	反映されない	反映されない
	コミット後	反映されない	反映されない	反映されない	反映されない	反映されない	反映されない

30.2 EJBサービス(IJServer)への移行方法

Interstage Application Server V9.0以降ではInterstage Application Server V5.1以前から提供していた以下の機能は提供していません。

- EJB1.0規約に準拠したアプリケーション運用
- 高速呼び出し機能
- Light EJBコンテナ機能
- DBアクセス環境定義

EJB1.0規約に準拠したアプリケーションをInterstage V9.0以降に移行する場合には、EJB1.1規約以降に準拠したアプリケーションに変更し、Interstage Application Server V9.0以降が提供するIJServer上での運用に移行してください。
DBアクセス環境定義を使用していた場合にはInterstage管理コンソールもしくはisj2eeadminコマンドでJDBCデータソースを定義する運用に移行してください。DBアクセス環境定義で定義したJDBCデータソースを使用してCMP1.1 Entity Beanで有効だったSQL文のキャッシュ機能は、OracleのJDBCドライバを利用した場合に有効なStatementキャッシュ機能を使用した運用に移行してください。
ここでは、高速呼び出し機能とLight EJBコンテナ機能からIJServerに移行する方法について説明します。

30.2.1 EJBの高速呼び出し機能とLight EJBコンテナ機能からIJServerへの移行

IJServerは高速呼び出し機能とLight EJBコンテナ機能に対して下表のような差異があるため、移行の際は留意が必要です。
なお、本バージョンのIJServerに対するセットアップ、および運用操作はInterstage管理コンソールまたはisj2eeadminコマンドを用いて簡単に行うことができます。
isj2eeadminコマンドでできる操作などについては、「リファレンスマニュアル(コマンド編)」の「isj2eeadminコマンド」を参照してください。

以下に、それぞれの運用方法の違いを表で表します。

	高速呼出し機能	旧バージョンにおけるIJServer (Light EJBコンテナ機能)	本バージョンのIJServer
クライアントの環境設定	以下のファイルをクラスパスに設定します。 Windows32 [JDK1.3の場合] C:¥Interstage¥EJB¥lib¥fjcontainer32.jar [JDK1.4の場合] C:¥Interstage¥EJB¥lib¥fjcontainer34.jar Solaris32 Linux32 [JDK1.3の場合] /opt/FJSVejb/lib/fjcontainer32.jar [JDK1.4の場合] /opt/FJSVejb/lib/fjcontainer34.jar	以下のファイルをクラスパスに設定します。 Windows32 [JDK1.3の場合] C:¥Interstage¥EJB¥lib¥fjcontainer32.jar [JDK1.4の場合] C:¥Interstage¥EJB¥lib¥fjcontainer34.jar Solaris32 Linux32 [JDK1.3の場合] /opt/FJSVejb/lib/fjcontainer32.jar [JDK1.4の場合] /opt/FJSVejb/lib/fjcontainer34.jar	以下のファイルをクラスパスに設定します。 Windows32/64 C:¥Interstage¥EJB¥lib¥fjcontainer94.jar Solaris64 Linux32/64 /opt/FJSVejb/lib/fjcontainer94.jar
配備	以下のツールを使用して配備を実行します。 ・J2EE Deploymentツール(非サーバ管理モード) ・EJB Deploymentツールで展開処理実行後、以下のコマンドを実行してEJBアプリケーションをインストールします。 ejbinstalleb [サーバ用生成物] 配備実行後にカストマイズツールを使用して、Java VM外から呼び出されるBeanを1つだけ「高速に呼び出すBean」に定義し、Java VM内で呼び出されるBeanを「高速に呼び出されるBean」に定義します。	以下のツールを使用してIJServer(またはLight EJBコンテナ)を作成します。 ・J2EE Deploymentツール(isdeployコマンド) ・J2EE管理ツール ・ejbmakecontainerコマンド 以下のツールを使用してIJServerに対してWebアプリケーション/EJBアプリケーションを配備します。 ・J2EE Deploymentツール(isdeployコマンド) ・isdeploybコマンド ・J2EE管理ツール ・EJB Deploymentツールで展開処理実行後、以下のコマンドを実行してEJBアプリケーションをインストールします。	以下のツールを使用してIJServerを作成します。 ・Interstage管理コンソール ・isj2eeadminコマンド IJServerには以下の4種類のIJServerを選択できます。 ・WebアプリケーションとEJBアプリケーションを同一Java VMで運用 ・WebアプリケーションとEJBアプリケーションを別Java VMで運用 ・Webアプリケーションのみ運用 ・EJBアプリケーションのみ運用 以下のツールを使用してIJServerに対してWebアプリケーション/EJBアプリケーションを配備します。同一IJServerに配備されたEJBアプリケー

		ejbinstalleb -s [IJServer名] [サーバ用生成物] IJServerに配備されたEJBアプリケーションは、同一Java VM上で動作します。ServletとEJBは別Java VMで動作します。	ションは同一Java VM上で動作します。 ・ Interstage管理コンソール ・ ijsdeploymentコマンド(配備解除時はijsundeploymentコマンド)
カスタマイズ	以下のツールを使用してEJBアプリケーションをカスタマイズします。 ・ ejbcustx ・ ejbdefimportコマンドとejbdefexportコマンド	以下のツールを使用してEJBアプリケーションをカスタマイズします。 ・ ejbcustx -s [IJServer名] ・ ejbdefimportコマンドとejbdefexportコマンド(-sオプションでIJServer名を指定します。) または以下のツールを起動し、配備済みのEJBアプリケーションを選択してカスタマイズします。 ・ J2EE Deploymentツール(isdeployコマンド)	以下のツールを使用して配備済みのEJBアプリケーションを選択してカスタマイズします。 ・ Interstage管理コンソール ・ ejbdefimportコマンドとejbdefexportコマンド(-iオプションでIJServer名を指定します。)
運用	以下のツールを使用して高速に呼び出されるBeanを登録したワークユニットを定義します。 ・ isaddwudefコマンド(削除時はisdelwudefコマンド) ・ Interstage運用操作ツール 定義したワークユニットは以下のツールを使用して起動します。 ・ isstartwuコマンド(停止時はisstopwuコマンド) ・ Interstage運用操作ツール	IJServerを定義するとワークユニットも自動的に定義されます。 以下のツールを使用してIJServer(またはワークユニット)を起動します。 ・ J2EE管理ツール ・ isstartwuコマンド(停止時はisstopwuコマンド) ・ Interstage運用操作ツール	Interstage V6以降ではIJServerは以下のように位置付けています。 「J2EEアプリケーションを運用するワークユニットをIJServerと呼びます。(IJServerとワークユニットは1対1の関係を持ちます。) IJServerはJ2EEアプリケーションの配備対象であり、起動/停止の単位です。」 以下のツールを使用してIJServerを起動します。 ・ Interstage管理コンソール ・ isstartwuコマンド(停止時はisstopwuコマンド)
リソース定義	以下のツールを使用してJ2EEの各種リソースを定義します。 ・ J2EE管理ツール ・ J2EEリソースアクセス定義 ・ J2EE Deploymentツール(isdeployコマンド)		以下のツールを使用してJ2EEの各種リソースを定義します。 ・ Interstage管理コンソール ・ isj2eeadminコマンド

30.2.2 Interstage Application Server V3.xからの移行

Interstage V3.xと本バージョン・レベルでの動作の違い

EJBアプリケーション内で以下の機能を使用した場合、Interstage V3.xで動作させた場合と本バージョン・レベルで動作させた場合と処理結果が異なります。

機能	本バージョン・レベルでの動作	Interstage V3.xの動作
リエントラント制御	[Session Beanの場合] 常に1つのスレッドだけがSessionオブジェクトに対して処理を実行できる。1つのSessionオブジェクトに対して複数のクライアントから同時にアクセスしたり、ループバック呼出しを行うと、後から受信した要求に対してjava.rmi.RemoteExceptionがthrowされる。	[Session Beanの場合] 1つのSessionオブジェクトに対して複数のクライアントから同時にアクセスしたり、ループバック呼出しを行ってもエラーとならない。
	[Entity Beanの場合] deployment descriptorのリエントラント種別に非リエントラントが指定されている場合、1つのEntityオブジェクトに対して複数のクライアントから同時にアクセスしたり、ループバック呼出しを行うと、後から受信した要求に対してjava.rmi.RemoteExceptionがthrowされる。 deployment descriptorのリエントラント種別にリエントラントが指定されている場合、例外は発生しない。	[Entity Beanの場合] deployment descriptorのリエントラント種別にかかわらず例外は発生しない。
Session BeanのPrimaryKey関連メソッド	Session BeanのEJBObject.getPrimaryKey()、EJBHome.remove(Object primaryKey)を実行すると以下の例外がthrowされる。 ・ EJBObject.getPrimaryKey()の場合 java.rmi.RemoteException ・ EJBHome.remove(Object primaryKey)の場合 javax.ejb.RemoveException	Session BeanのEJBObject.getPrimaryKey()、EJBHome.remove(Object primaryKey)を実行しても正常に動作する。
Session Bean内で発生したシステム例外処理	Sessionオブジェクトは破棄される。	Sessionオブジェクトは破棄されない。
Entity Bean内で発生したシステム例外処理	Entity Beanのインスタンスは破棄される。	Entity Beanのインスタンスは破棄されない。
setRollbackOnly()、getRollbackOnly()メソッド	トランザクション管理種別が「Bean」の場合でインスタンスがトランザクションに関連付けられていないときにEntityContextインタフェース、またはSessionContextインタフェースのsetRollbackOnly()やgetRollbackOnlyを実行するとIIllegalStateExceptionが発生する。 トランザクション管理種別が「Container」でトランザクション属性が「NotSupported」の場合も同様にIllegalStateExceptionがthrowされる。	EntityContextインタフェースまたはSessionContextインタフェースのsetRollbackOnly()やgetRollbackOnlyは、どのトランザクション属性でも正常に動作する。
コンテナのリソース接続者管理	リソース接続者に「Container」が指定された場合、Interstage管理コンソールまたはisj2eeadminコマンドで定義したデータ	リソース接続者に指定された値にかかわらず、アプリケーションにユーザID、パスワードが指定された場合はそのユーザID、

機能	本バージョン・レベルでの動作	Interstage V3.xの動作
	ソースのユーザIDとパスワードがリソース接続時に使用される。 リソース接続者に「Application」が指定された場合、アプリケーションにユーザID、パスワードが指定された場合はそのユーザID、パスワードがリソース接続時に使用される。アプリケーションに指定されなかった場合はクライアントの認証情報が使用される。	パスワードがリソース接続時に使用される。 アプリケーションにユーザID、パスワードが指定されていない場合はDBアクセス環境定義に設定されているユーザIDとパスワードがリソース接続時に使用される。
STATEFUL Session Beanのcreate()、remove()メソッド	[create()の場合] トランザクション内で実行されない。トランザクション内で呼び出した場合は、トランザクションが中断されてから実行される。	[create()の場合] 呼出し元で開始されたトランザクション上で動作する。 システム例外が発生した場合はトランザクションにロールバックをマークしてTransactionRolledBackExceptionがthrowされる。
	[remove()の場合] トランザクション内で実行された場合、javax.ejb.RemoveExceptionがthrowされる。	[remove()の場合] 呼出し元で開始されたトランザクション上で動作する。 システム例外が発生した場合はトランザクションにロールバックをマークしTransactionRolledBackExceptionがthrowされる。
Session Beanの Mandatoryトランザクション属性	トランザクション属性にMandatoryが指定された場合、トランザクションが開始されていない状態でSession Beanが呼び出されると、TransactionRequiredExceptionがthrowされる。	トランザクション属性にMandatoryが指定された場合、トランザクションが開始されていない状態でSession Beanが呼び出されても例外は発生せず、正常に動作する。
UserTransactionの lookup	「12.6.3 Enterprise Beanクラスのメソッドが実行可能な操作」に記述されているUserTransactionをlookupできないメソッドでUserTransactionをlookupした場合、java.lang.IllegalStateExceptionが発生する。	すべてのメソッドでUserTransactionをlookupすることが可能。
remove(PrimaryKey)	[Entity Beanの場合] Home.remove(primarykey)メソッドで、primarykeyに該当するレコード(インスタンス)が存在 しない場合、java.rmi.NoSuchObjectExcetionをthrowする。	[Entity Beanの場合] Home.remove(primarykey)メソッドで、primarykeyに該当するレコード(インスタンス)が存在 しない場合、javax.ejb.RemoveExceptionをthrowする。
Entity Bean Homeインタフェースのremove(primarykey)メソッド	remove(primarykey)メソッドで、primarykeyに該当するレコード(インスタンス)が存在しない場合、RemoteException(NoSuchObjectExcetion)がthrowされる。	remove(primarykey)メソッドで、primarykeyに該当するレコード(インスタンス)が存在しない場合、RemoveException がthrowされる。

対処方法

「Interstage V3.xと本バージョン・レベルでの動作の違い」に記載されている機能を使用している場合、以下のどちらかの対処が必要です。なお、該当する機能を使用していない場合は、対処は不要です。

EJBアプリケーションを修正する

「Interstage V3.xと本バージョン・レベルでの動作の違い」に注意して、EJBアプリケーションを修正してください。

EJBアプリケーションの処理モードを変更する Windows32 Linux32

EJBアプリケーションの修正ができない場合は、EJBアプリケーションの処理モードを変更します。

処理モードの変更方法

Interstage管理コンソールのワークユニット設定に以下の定義をしてください。

設定項目

Java VM オプション(Java Command Option)

設定値

-DFJV3MODE=処理モード

処理モード

yes:Interstage V3の動作
no :本バージョンの動作

注意

以下の場合には本バージョンの動作をします。

- 処理モードを設定しない場合
- 処理モードに上記に示す設定値以外を設定した場合

30.2.3 ロードバランス機能について

Interstage Application Server V5.x(Interstage V5.x)よりTraffic Director、およびInterstage Application Server 8.0.x(Interstage 8.0.x)よりIPCOMがサポートされたため、EJBサービスでロードバランシングする場合に従来使用されていたロードバランス機能が、非推奨となりました。

第31章　J2EEの移行

ここでは、以下について説明します。

- J2EEの追加機能
  Interstage Application Server V7.0以前から移行する場合に参照してください。
- J2EEアプリケーションの移行
  Interstage Application Server V11.x以前から移行する場合に参照してください。
- INTERSTAGE V3.xからJ2EEアプリケーションへの移行
  INTERSTAGE Application Server V3.x以前から移行する場合に参照してください。
- Servletサービスの移行
  Interstage Application Serverの旧バージョン・レベルから移行する場合に参照してください。
- EJBサービスの移行
  Interstage Application Server 9.0以前から移行する場合に参照してください。
- Interstage JMSの移行
  Interstage Application Server V10.x以前から移行する場合に参照してください。
- Interstage Webサービスの移行
  Interstage Application Server V10.1以前から移行する場合に参照してください。
- SOAPサービスの移行
  Interstage Application Server V9.0以前から移行する場合に参照してください。
- ワークユニットの移行
  Interstage Application Server V11.1.0以前から移行する場合に参照してください。
- 移行に関する注意事項
  Interstage Application Serverの旧バージョン・レベルから移行する場合に参照してください。

31.1 J2EEの追加機能

J2EEの以下の追加機能について説明します。

Interstage Application Server 8.0での追加機能

- J2EE定義コマンド
- Webサービス機能
- アプリケーションファイル保護レベル
- Javaヒープ/Java Permanent領域不足時の制御

31.1.1 J2EE定義コマンド

Interstage Application Server 8.0(Interstage 8.0)から、J2EEのすべての定義更新処理がコマンドで実行できるisj2eeadminコマンドが提供されました。
従来のInterstage管理コンソールでの操作や以下のコマンドも提供されますが、サーバパッケージでは、以下のコマンドは互換機能(Interstage 8.0以降の機能エンハンスは行いません)となりますので、本機能(isj2eeadminコマンド)への移行を推奨します。
クライアントパッケージではInterstage JMXサービスが存在しないため、従来通り以下のコマンドを使用してください。

- jmsrmfact

・ jmsmkdst

また、従来クライアントパッケージにfjj2eeadminコマンドを提供していましたが、クライアントからのデータベース接続は非推奨です。IJServer上のJ2EEアプリケーションからデータベース操作を行ってください。

isj2eeadminコマンドの操作範囲を、以下の表に示します。

操作	V7.0以前	8.0以降
IJServer作成	なし	isj2eeadmin
ログ定義	なし	isj2eeadmin
実行クラス	なし	isj2eeadmin
IJServer定義更新	なし	isj2eeadmin
IJServer削除	なし	isj2eeadmin
配備	ijsdeployment	ijsdeployment
配備解除	ijsundeployment	ijsundeployment
J2EEプロパティ定義	定義ファイル公開	isj2eeadmin
Servletサービス設定	なし	isj2eeadmin
EJBサービス設定	定義ファイル公開	isj2eeadmin
Webサーバコネクタ定義	なし	isj2eeadmin
JDBCリソース作成・削除	fjj2eeadmin	isj2eeadmin
JMS ConnectionFactory作成	jmsmkfact	isj2eeadmin
JMS ConnectionFactory削除	jmsrmfact	isj2eeadmin
JMS Destination作成	jmsmkdst	isj2eeadmin
JMS Destination削除	jmsrmdst	isj2eeadmin
connectorリソース定義更新	fjj2eeadmin	isj2eeadmin
JavaMailリソース作成・削除	fjj2eeadmin	isj2eeadmin
EJBアプリケーション環境設定	ejbdefexport/ejbdefimport	ejbdefexport/ejbdefimport
名前変換定義	ijsdeployment	ijsdeployment

isj2eeadminコマンドについては、「リファレンスマニュアル(コマンド編)」の「J2EE運用コマンド」の「isj2eeadmin」を参照してください。

31.1.2 Webサービス機能

Interstage Application Server 8.0から、J2EEの機能の1つとして、新たなWebサービス運用環境であるInterstage Webサービスが提供されました。旧バージョン・レベルのSOAPサービスのアプリケーションは、J2EEのInterstage Webサービスへ移行することができます。本機能は、旧バージョン・レベルのSOAPサービスと異なるWebサービス実行環境を提供しており、機能差異があります。機能差異、移行については、「31.9 SOAPサービスの移行」を参照してください。

31.1.3 アプリケーションファイル保護レベル

Interstage Application Server 8.0から、アプリケーションファイルの権限を変更する機能が提供されました。
アプリケーションファイルのアクセス権は、IJServer定義のアプリケーションファイル保護レベルの設定に応じて、下記のとおりに設定されます。

アプリケーションファイル保護レベル	アプリケーションファイルのアクセス権	
	ディレクトリ	ファイル
高(管理者のみ)	755	644
低(一般ユーザ)	777	666

アクセス権の変更対象となる資源は、下記のとおりです。

- IJServerディレクトリ/appsディレクトリ、および配下の資源
- IJServerディレクトリ/Shared/libディレクトリ
- IJServerディレクトリ/Shared/classesディレクトリ
- IJServerディレクトリ/extディレクトリ

デフォルトの設定では、「高(管理者のみ)」が設定されています。
これにより、従来は一般ユーザでアクセス可能であったejbdefexport、ejbdefimportコマンドで参照するファイルのアクセス権が、デフォルトの設定では管理者でのみアクセス可能な状態となります。
必要に応じてアプリケーションファイル保護レベルの設定を変更してください。

31.1.4 Javaヒープ/Java Permanent領域不足時の制御

以下の製品から、Javaヒープ/Java Permanent領域不足時の制御を変更する機能が提供されました。

- SolarisおよびRHEL-AS4(x86)のInterstage Application Server 8.0
- 上記以外のプラットフォームのInterstage Application Server/Interstage Web Server V9.0

本製品では、この機能を「Javaヒープ領域/メタスペース不足時の制御」と呼びます。

制御方法は、以下から選択できます。

- アプリケーションにjava.lang.OutOfMemoryErrorを返却する
- プロセスを再起動する

本機能サポートバージョン以降で作成したIJServerは、「プロセスを再起動する」がデフォルトで設定されます。本機能サポートバージョン以前では「アプリケーションにjava.lang.OutOfMemoryErrorを返却する」が設定された場合と同様の制御となっていたため、必要に応じて設定を変更してください。
本機能サポートバージョン以前に作成したIJServerをリストアした場合には、「アプリケーションにjava.lang.OutOfMemoryErrorを返却する」がデフォルトで設定されます。
Interstage Application Server V6.0で作成したIJServerをリストアした場合は、本機能は設定できません。
Interstage Application Server V7.0以降で作成したIJServerの設定値は、Interstage管理コンソールまたは、isj2eeadminコマンドで変更することができます。

31.2 J2EEアプリケーションの移行

ここでは、J2EEアプリケーションの移行について、以下を説明します。

- IJServerの移行について
  Interstage Application Server V10/V10.1以前から移行する場合に参照してください。
- コンパイル時・実行時にクラスパスに追加するjarについて
  Interstage Application Server 8.0以前に作成したWebアプリケーションをコンパイルする場合に参照してください。
- XMLパーサのXerces2サポート
  Interstage Application Server 8.0以前から移行する場合に参照してください。
- データベースについて
  データベースを使用する場合に参照してください。
- JDBCドライバ
  Interstage Application Server 8.0以前から移行する場合に参照してください。
- 自動再接続機能について
  Interstage Application Server 8.0以前から移行する場合に参照してください。
- SQL実行の通信待ち時間監視について
  Interstage Application Server 8.0以前から移行する場合に参照してください。
- コネクション使用時間監視について
  Interstage Application Server 8.0以前から移行する場合に参照してください。
- クラスローダの変更について
  Interstage Application Server V6.0以前から移行する場合に参照してください。
- 返却される例外の詳細文字列について
  Interstage Application Server V6.0以前から移行する場合に参照してください。
- JNDIから返却される例外について
  Interstage Application Server 5.x以前から移行する場合に参照してください。
- ORBプロパティ情報ファイル（orb.properties）について
  Interstage Application Server V10/V10.1以前から移行する場合に参照してください。
- Fujitsu XMLプロセッサについて
  Interstage Application Server V11.1/V11.2以前から移行する場合に参照してください。
- J2EEアプリケーションの移行時のその他の注意事項
  Interstage Application Server 8.0以前から移行する場合に参照してください。

31.2.1 IJServerの移行について

IJServerの移行について、以下を説明します。

- バックアップ/リストアについて
- リトライカウントリセット契機の変更について
- XMLパーサの設定について
- Javaバージョンについて
- IJServerのファイル構成について
- [IJServerディレクトリ]/commonについて
- IJServerに対するconnectorの配備について

- 名前変換機能について
- 名前変換定義で参照名を重複して定義した場合について
- Interstage Application Server V5.x以前から移行する場合
- IJServerのクラスパス自動設定
- 配備解除に失敗した場合について
- クラスのオートリロード機能
- クラスローダのトレース機能
- 配備時に指定するモジュール名について

バックアップ/リストアについて

Interstage Application Server V6.0/V7.0/8.0からの移行時

Interstage 8.0以前に作成したIJServerの資産をリストアした場合、そのIJServerは運用できません。IJServerを削除し、再度作成し直してください。

Interstage Application Server V9からの移行時

V8.0互換モードのIJServerは運用できません。IJServerを削除し、再度作成し直してください。

パスワードの暗号化強度について

V10からセキュリティレベルを上げました。V9以前の資産を利用している場合、セキュリティレベルは低いままになります。セキュリティレベルを上げるためには、パスワードを設定しているJDBCデータソース、connectorの定義をInterstage管理コンソールまたはisj2eeadminコマンドで更新する必要があります。

リトライカウントリセット契機の変更について

IJServerのアプリケーション異常終了した場合のリトライカウントのリセット時間は、Interstage 8.0までは、以下のIJServerタイプでは設定できませんでしたが、Interstage V9.0以降では、以下のIJServerタイプでも設定可能になりました。

- EJBアプリケーションのみ運用するIJServer
- WebアプリケーションとEJBアプリケーションを別JavaVMで運用するIJServerのEJBアプリケーション運用

上記の場合、従来の監視方式では、EJBアプリケーションへのビジネスロジック(Remoteインタフェース)へアクセスする前に、必ずHomeインタフェースメソッド(初期処理)実行が正常終了した時点でリトライカウントがクリアされていましたが、Interstage V9.0以降では、リトライカウントリセット時間で指定した時間が経過すると、リトライカウントがリセットされます。

リトライカウントリセット時間はInterstage管理コンソールまたはisj2eeadminコマンドを使用して定義変更が可能です。必要に応じて変更を行ってください。

- Interstage管理コンソールの場合

  以下の画面の「リトライカウントリセット時間」の値を変更してください。
  [ワークユニット] > [IJServer名] > [環境設定]

- isj2eeadminコマンドの場合

  IJServer定義ファイルの以下の項目の値を変更してください。
  IJServer > Common > RetryCountResetTime

XMLパーサの設定について

Interstage 8.0までは、Crimsonが配備時のXMLパーサとして使用されていました。
Interstage V9.0以降では、IJServerの環境設定で指定したXMLパーサが、配備時のXMLパーサとして使用されるようにな

りました。
XMLパーサによりサポートしている文字エンコーディングが異なるため、IJServerの環境設定で指定したXMLパーサが、deployment descriptorを記述している文字エンコーディングをサポートしているか確認してください。
なお、従来通りにCrimsonを使用する場合は、IJServerの環境設定でCrimsonが有効となるように設定してください。

また、Webサービスを利用する場合、JAXP1.2以上をサポートしているXMLパーサを指定する必要があります。
Interstage 8.0までは、IJServerの環境設定で「コンテナのWebサービス機能」の定義を「有効」に設定している場合に「使用するXMLパーサの種別」の「その他」にJAXP1.2をサポートしていないXMLパーサ指定してもチェックが行われていませんでしたが、Interstage V9.0以降では、IJServerの環境設定でチェックするように改善されました。

Webサービスを利用する場合は、JAXP1.2以上をサポートしているXMLパーサを使用するように設定してください。
なお、デフォルトではIJServerの環境設定の「コンテナのWebサービス機能」は「有効」に設定されています。Webサービスを利用しない場合は、この設定を「無効」にすることで、JAXP1.2をサポートしていないXMLパーサを使用することができます。

配備時に使用されるXMLパーサについては、「3.5.1 配備に必要なXMLパーサの設定」を参照してください。

Javaバージョンについて

以下の場合、IJServerが動作するJavaのバージョンが変わります。本製品ではJDK/JRE8で動作します。

- Interstage 8.0以前に作成したIJServer、またはV8.0互換モードのIJServerを本製品へ移行する場合
- Interstage V9からV11.2で作成したIJServerを本製品へ移行する場合

また、Interstage V11.2以前はInterstage管理コンソールやisj2eeadminコマンドの入力となるIJServer定義ファイルで、IJServerが動作するJavaのバージョンを指定しましたが、Interstage V12.0以降は指定不要になります。同様に、設定項目の参照・抽出時にJavaのバージョンは表示・抽出されません。IJServerが動作するJavaのバージョンは、コンテナ情報ログに出力されるIJServerを起動するJavaコマンドのパス名で判断してください。

IJServerのファイル構成について

Interstage V7.0以降、J2EEアプリケーションの配備後の管理単位が、アプリケーション単位からモジュール単位に変更されました。
この変更に伴いIJServerのファイル構成が変更されました。
IJServerのファイル構成については「2.2.3 IJServerのファイル構成」を参照してください。

[IJServerディレクトリ]/commonについて

Interstage V7.0以降、EARファイル内のWARファイル、ejb-jarファイル、RARファイル以外の資源の展開先が、以下のように変更されました。

	Interstage V6.0以前	Interstage V7.0以降
仕様: EARファイル内のWARファイル、ejb-jarファイル、RARファイル以外の資源の展開先	[IJServerディレクトリ]/common以下	[IJServer]ディレクトリ]/apps/[アプリケーション名]以下 ※アプリケーション間の独立性を高めたため
影響: common直下に保管したJARファイルの扱い	IJServerの起動時にクラスパスに設定される	クラスパスに設定されない

Interstage V6.0以前にcommon直下に保管していたJARファイルを、本バージョン・レベルで使用する場合は、以下のいずれかの対応を行う必要があります。

- マニフェストクラスパスを使用する
- EARファイル内のSharedディレクトリにJARファイルを保管する
- [IJServerディレクトリ]/SharedディレクトリにJARファイルを保管する

- ワークユニットのクラスパスに設定する

ただし、J2EEの仕様に準拠したEARファイルの作成を行っている場合はマニフェストクラスパスが指定されていますので、上記の対応は不要です。

IJServerに対するconnectorの配備について

Interstage V6.0ではIJServerにconnectorを配備した場合にも、システム全体で一意な資産として管理され、各IJServerで参照可能でした。
Interstage V7.0以降ではIJServerに配備したconnectorは、IJServer内で管理され、そのIJServer内でのみ参照可能となりました。
Interstage管理コンソールで、[リソース] > [connector]で配備されたconnectorとIJServerに配備されたconnectorが重複した場合は、IJServerのconnectorが優先されます。

名前変換機能について

Interstage V6.0でのWebアプリケーション、およびEJBアプリケーションの名前変換は、IJServer単位での設定となっていました。
Interstage V7.0以降ではモジュール単位での設定に変更されました。設定はInterstage管理コンソールの各モジュールの画面から行うことが可能です。
これにともない、配備時にIJServer単位の名前変換定義の更新は行われないため、Interstage V6.0では出力されていた下記のメッセージは配備時に出力されなくなりますが、名前変換定義は有効になっています。

```
IS: 情報: is40902: 名前変換定義を更新しました
```

Interstage管理コンソールを使用しない場合の設定方法については、「4.11 名前変換機能」を参照してください。

名前変換定義で参照名を重複して定義した場合について

名前変換定義は参照名を重複して定義することが可能ですが、Interstage V9.0.0以前では重複定義された場合にどの定義値を有効にするか明確にしていませんでした。

例

重複定義の例 (jdbc/DataSourceが重複している)

```
  <res-entry>
    <res-ref-name>jdbc/DataSource</res-ref-name>  ★ 参照名が重複している
    <datasource-name>DS1</datasource-name>
  </res-entry>
  <res-entry>
    <res-ref-name>jdbc/DataSource</res-ref-name>
    <datasource-name>DS2</datasource-name>
  </res-entry>
  <res-entry>
    <res-ref-name>jdbc/DataSource</res-ref-name>
    <datasource-name>DS3</datasource-name>
  </res-entry>
```

Interstage V9.0.0以前は、以下のように各モジュールにより有効になる定義値が統一されていませんでした。
Interstage V9.0.1/V9.1以降では重複定義された場合、最後の定義値が有効になります(例ではDS3が有効)。

		V9.0.0以前	V9.0.1/V9.1以降
定義時	管理コンソール	最後の定義値が有効	最後の定義値が有効
実行時	J2EEクライアント	最初の定義値が有効	最後の定義値が有効
	Webアプリケーション	最初の定義値が有効	最後の定義値が有効
	EJBアプリケーション	最後の定義値が有効	最後の定義値が有効

名前変換を行うアプリケーションがJ2EEクライアントまたはWebアプリケーションの場合、有効になる定義値が変更になることがあります。従来通りに動作させたい場合は、以下のオプションを設定してください。

定義場所	JavaVMオプション
定義名	com.fujitsu.interstage.j2ee.jndi.EbeDef_CheckOrder_FromHead
定義値	true
記述例	-Dcom.fujitsu.interstage.j2ee.jndi.EbeDef_CheckOrder_FromHead=true
意味	J2EEクライアントアプリケーションまたはWebアプリケーションで名前変換定義に定義した値は先頭に定義した値が有効になります。

Interstage Application Server V5.x以前から移行する場合

J2EEアプリケーションをInterstage Application Server V5.x(Interstage V5.x)以前のバージョンから移行する場合には、J2EEアプリケーションをIJServerに配備する必要があります。
Interstage管理コンソールの[ワークユニット] > [新規作成]タブ、または、isj2eeadminコマンドを使用して、IJServerワークユニットを作成してください。そのあと、「ワークユニット名」 > [配備]タブで配備を行います。
Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドについては、「リファレンスマニュアル(コマンド編)」の「isj2eeadminコマンド」を参照してください。

IJServerのクラスパス自動設定

IJServerのクラスパスは内部的に自動的に設定されます。
CORBAアプリケーションのライブラリとしてはプレインストール型Javaライブラリがクラスパスに常に設定され、Portable-ORB用のライブラリは使用できませんが、V5.1以前のServletサービスでPortable-ORB用のライブラリで動作させていたアプリケーションも、Interstage V6.0以降のIJServerでそのまま動作させることができます。
CORBAサービスの環境定義を変更する場合にはプレインストール型の設定を変更してください。

以下のクラスパスは、V9.0以降のIJServerでは自動設定されません。

Windows32/64

JDKのインストールディレクトリ¥lib¥tools.jar

Solaris64 Linux32/64

JDKのインストールディレクトリ/lib/tools.jar

アプリケーションで必要な場合は、「2.3.1 クラスローダの構成」および「2.3.4 IJServerで使用するクラスの設定について」を参照して設定を行ってください。

配備解除に失敗した場合について

Interstage Application Server/Interstage Web Server V9.0以降では、配備解除の際にファイルの削除に失敗した場合、手動でファイルの削除を行う必要があります。

配備対象モジュールのファイルのみ残った状態ではモジュールはロードされないため、IJServerの運用に影響を及ぼしませんが、同一モジュールの再配備を行った場合に、モジュールの配備先に不明なファイルが存在することになり、再度削除を促すためのメッセージが出力され、配備処理がエラーとなります。このような場合、不要なファイルを削除してから、配備処理を行ってください。

クラスのオートリロード機能

Interstage Application Server 8.0までは、クラスのオートリロード機能はモジュール(EAR/WAR/ejb-jar)単位での設定でした。
Interstage Application Server/Interstage Web Server V9.0以降では、設定はIJServer単位に変更されています。個々のモジュールごとに設定を行う必要はありません。
モジュールによりオートリロード機能の設定を切り替える必要がある場合には、モジュールを別のIJServerに配備し、各IJServerで必要であればオートリロードの設定を行ってください。
また、クラスローダの分離の設定に「分離しない」を選択している場合、Interstage 8.0までのIJServerではWebアプリケーション(WAR)のみオートリロード機能が使用可能でしたが、Interstage V9.0以降では、オートリロード機能は使用できません。
他の「クラスローダの分離」の設定とするか、またはモジュールを別のIJServerに配備し、各IJServerで必要であればオートリロードの設定を行ってください。

クラスローダのトレース機能

Interstage Application Server 8.0までは、クラスローダのトレース機能の設定は、以下で設定可能でした。

- Interstage管理コンソール

  [ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [共通定義] > [クラスローダのトレース情報の出力]

- isj2eeadminコマンドのIJServer定義ファイル

  IJServerタグ/ClassLoaderタグ/Traceタグ

Interstage Application Server/Interstage Web Server V9.0以降でトレース情報を出力するには、以下でJavaVMオプションを設定します。

- Interstage管理コンソール

  [ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [ワークユニット設定] > [JavaVMオプション]

- isj2eeadminコマンドのIJServer定義ファイル

  IJServerタグ/ClassLoaderタグ/JavaCommandOptionsタグ
  指定値: -Dcom.fujitsu.interstage.j2ee.ijserver.loader.trace=true

配備時に指定するモジュール名について

Interstage 8.0までは、モジュール名の末尾にピリオドを指定した場合、配備資源の展開が不正なディレクトリに行われることがありました。
Interstage V9.0以降では、モジュール名の末尾にピリオドを指定した場合、配備時にエラーとなり、DEP1833のメッセージが出力されます。
Interstage V9.0以降で配備を行う場合、モジュール名の末尾には、ピリオド以外の文字を指定してください。
モジュール名の詳細については、Interstage管理コンソールのヘルプの配備を参照してください。
なお、モジュール名の末尾にピリオドが指定されたモジュールが配備されているInterstage 8.0以前のIJServerをバックアップ、およびリストアにより本バージョンレベルに移行することは可能です。

31.2.2 コンパイル時・実行時にクラスパスに追加するjarについて

Interstage Application Server 8.0以前に、Servlet API/JavaServer Pages APIを使用したJavaアプリケーションをコンパイルまたは実行するために「isj2ee.jar」をクラスパスに指定していた場合は、「isj2ee.jar」のかわりに以下のjarをクラスパスに指定する必要があります。

Windows32/64

C:¥Interstage¥F3FMjs5¥common¥lib¥servlet-api.jar
C:¥Interstage¥F3FMjs5¥common¥lib¥jsp-api.jar

Solaris64 Linux32/64

/opt/FJSVjs5/common/lib/servlet-api.jar
/opt/FJSVjs5/common/lib/jsp-api.jar

31.2.3 XMLパーサのXerces2サポート

Interstage Application Server V7.0では、IJServerで使用するXMLパーサのデフォルトはCrimsonでしたが、本バージョン・レベルでは、デフォルトでXerces2が使用されます。V9.0以降のIJServerではXMLパーサにCrimsonは選択できません。従来通りにCrimsonを使用する場合には、以下の対処を実施してください。Crimsonを使用するには、「コンテナのWebサービス機能」の定義を「無効にする」に設定してください。

V9.0以降のIJServerの場合

Interstage管理コンソールの[ワークユニット]>[IJServer名]>[環境設定]で、使用するXMLパーサの種別にその他を選択して、Apacheのホームページからダウンロードしたcrimsonのjarファイルを格納したディレクトリを指定してください。また、isj2eeadminコマンドでも設定できます。

Xerces2がサポートするエンコード

Xerces2では、以下のエンコードをサポートしています。使用するXMLパーサによってサポートするエンコードが異なるため、使用するXMLパーサのエンコードを確認してください。どのXMLパーサでも共通で使用可能なUTF-8を使用することをお勧めします。

UTF-8	ISO-8859-7	ebcdic-cp-no	ebcdic-cp-ar1	gb2312
ISO-8859-1	ISO-8859-8	ebcdic-cp-fi	ebcdic-cp-he	euc-jp
ISO-8859-2	ISO-8859-9	ebcdic-cp-se	ebcdic-cp-ch	iso-2020-jp
ISO-8859-3	ebcdic-cp-us	ebcdic-cp-it	ebcdic-cp-roece	Shift_JIS
ISO-8859-4	ebcdic-cp-ca	ebcdic-cp-es	ebcdic-cp-yu	Big5
ISO-8859-5	ebcdic-cp-nl	ebcdic-cp-gb	ebcdic-cp-is	euc-kr
ISO-8859-6	ebcdic-cp-dk	ebcdic-cp-fr	ebcdic-cp-ar2	koi8-r

31.2.4 データベースについて

データベースの以下の変更内容について説明します。

- Oracleを使用する場合
  - OCIドライバを使用する場合の変更
  - Oracleのコネクションプーリング
  - グローバルトランザクションについて
  - oci/ipc接続をする場合

Oracleを使用する場合

OCIドライバを使用する場合の変更

Oracle OCIドライバを使用し、データソースの「File System Service Providerを使用する」の項目を「使用しない」(デフォルト)にした場合、PG60150の修正を適用していないInterstage V9.0では、ネットサービス名を指定してもSIDとして動作していました。V9.1以降では、「SID/ネットサービス名」に指定した値はOCIドライバの場合ネットサービス名として動作します。V9.0と同様にOCIドライバでSIDとして動作させたい場合は、J2EEプロパティファイルに以下の行を追加してください。

編集対象のファイル(J2EEプロパティファイル)

Windows32/64

C:¥Interstage¥J2EE¥etc¥isj2ee.properties

Solaris64 Linux32/64

/opt/FJSVj2ee/etc/isj2ee.properties

追加する内容

oracle.oci.sid=yes

Oracleのコネクションプーリング

本製品では「Oracleのコネクションプーリングを使用する」ことができますが、Interstage 8.0以前で定義可能であった「Oracleでコネクションプーリングを行う」とは別機能になります。そのため本製品ではInterstage管理コンソールを使用してInterstage 8.0以前で定義可能であった「Oracleでコネクションプーリングを行う」を定義できません。

Interstage 8.0以前の「Oracleでコネクションプーリングを行う」を定義したデータソースを使用することはできません。データソース定義を削除し再作成を行ってください。Interstage管理コンソールにより参照した場合、データソースの種類はいずれも選択されていない状態で表示され、Interstage管理コンソールでは更新できません。

本製品でサポートしているOracleのJDBCドライバではInterstage 8.0以前の「Oracleでコネクションプーリングを行う」で定義されたデータソース定義で使用されるOracleConnectionCacheImplクラスが未サポートとなっているため、使用できません。

グローバルトランザクションについて Windows32/64 Linux32/64

Interstage Application Server 8.0以前において定義可能であったグローバルトランザクションを使用する場合は、「データソースの種類」で「分散トランザクションを使用する」を選択してください。

また、Interstage Application Server 8.0以前で「グローバルトランザクションを利用する」で定義されたデータソースを本製品へリストアした場合、「分散トランザクションを使用する」が選択された状態で表示されます。

oci/ipc接続をする場合

「ドライバタイプ／ネットワークプロトコル」に"oci/ipc"を指定した場合、V10.0以前のバージョンでは誤って"oci/tcp"で接続してしまうことがありましたが、本バージョンレベルでは、正しく"oci/ipc"で接続します。

Symfowareを使用する場合

コネクションプーリングのデフォルト変更

Interstage V9.0より「データソースの種類」に「Interstageのコネクションプーリングを使用する」が選択可能となりました。
Interstage管理コンソールおよびisj2eeadminコマンドでSymfowareのJDBCデータソースを作成する場合、従来は「データソースの種類」のデフォルトが「Symfowareのコネクションプーリングを使用する」でしたが、Interstage V9.0より「Interstageのコネクションプーリングを使用する」がデフォルトとなります。
isj2eeadminコマンドで、Interstage 8.0以前の定義ファイルをそのまま使用し「Symfowareのコネクションプーリングを使用する」を登録する場合は、isj2eeadminコマンドに「-v 8.0」オプションを付加してください。
詳細は「リファレンスマニュアル(コマンド編)」の「isj2eeadmin」を参照してください。

V4.0互換のJNDIサービスプロバイダについて

V4.0以前のSymfowareはサポートされなくなりました。Interstage 8.0以前はInterstage管理コンソールを使用してSymfowareのデータソースを定義する時に、JNDIサービスプロバイダのクラス名にV4.0以前のSymfowareのクラス名(fujitsu.symfoware.jdbc2.jndisp.SYMContextFactory)を選択できましたが、Interstage V9.0以降では選択できません。
Interstage 8.0以前の資産を利用し、データソース定義にJNDIサービスプロバイダのV4互換のクラス名が設定されている場合は、isj2eeadminコマンドを使用してInitialContextFactoryタグの値をサポートするクラス名に修正してください。
詳細は「リファレンスマニュアル(コマンド編)」の「isj2eeadmin」を参照してください。

PostgreSQLを使用する場合

JDBC 2.0 + Optional Packageについて

PostgreSQLのJDBC 2.0 + Optional PackageのJDBCドライバはサポート対象外となりました。JDBC3.0以降でアプリケーション実行環境のJavaのバージョンに合ったJDBCドライバを使用してください。

クライアントバージョンが"V2"のデータソースについて

クライアントバージョンが"V2"のデータソースは利用できません。データソースのクライアントバージョンを"V5"に変更してください。

データベース共通

サポートするデータベースについて

本製品で動作するIJServerのJavaのバージョンが移行前の製品と異なる場合、使用するJDBCドライバを変更する必要がある場合があります。使用するJDBCドライバを変更する場合、そのJDBCドライバがサポートするデータベースを確認し、サポートしているデータベースを使用するようにしてください。詳細はJDBCドライバの提供元に確認してください。

本製品がサポートしているJDBCドライバについては、「システム設計ガイド」の「データベース関連(J2EE)」を参照してください。

File System Service Providerについて

Interstage V9.0より、File System Service Providerを使用せずにJDBCデータソースを登録することが可能となりました。デフォルトではFile System Service Providerを使用しない場合のデータソースが定義されます。
isj2eeadminコマンドにより、Interstage 8.0以前で抽出した定義ファイルでFile System Service Providerを使用するデータソー

スを登録する場合は、「-v 8.0」オプションを指定してください。詳細は「リファレンスマニュアル(コマンド編)」の「isj2eeadmin」を参照してください。

パスワードの省略について

Interstage V9.0より、データソース定義の登録、更新時にパスワードが省略可能となります。このため、isj2eeadminコマンドでパスワードを省略した場合、エラーとならず登録が完了します。

isj2eeadminコマンドによる定義更新時のDatabaseKind変更不可

Interstage V9.0より、isj2eeadminコマンドによりデータソース定義を更新する際、DatabaseKindは定義更新時に変更できなくなります。
DatabaseKindを変更する場合は、一度削除してから再度作成してください。

SQL文のキャッシュ機能について

Interstage 8.0までサポートしていたCMP1.1の範囲で有効だったSQL文のキャッシュ機能はInterstage V9.0以降では未サポートとなりました。
Interstage V9.0より、データベースタイプが「Oracle」で、データソースの種類が「Oracleのコネクションプーリングを使用する」の場合とデータベースタイプが「Symfoware」で、データソースの種類が「Interstageのコネクションプーリングを使用する」の場合にStatementキャッシュ機能をサポートしましたので、「27.2.6 Statementキャッシュ機能」を参照し使用してください。

データソースのキャッシュについて

Interstage V9.0より、初回に参照したデータソース定義情報、データソースオブジェクトをキャッシュするようになりました。
Interstage 8.0以前では、IJServer起動後にデータソース定義を変更しても有効になる場合がありましたが、Interstage V9.0以降ではIJServer起動前にデータソースの定義登録を行ってください。

31.2.5 JDBCドライバ Windows32/64

Interstage JDBC Driverは非推奨機能のため未サポートとなりました。Microsoft(R) JDBCドライバを使用してください。移行は以下の手順により行います。

1. Microsoft(R) JDBCドライバをMicrosoft Corporationのホームページよりダウンロードしてください。JDBCドライバはSQL Server(TM)のバージョンに合わせて適切なものを使用してください。
2. 旧バージョンの資産をリストアした場合は、以下の点に注意し定義を更新してください。
   - 必須項目「データベース名」を入力する必要があります。
   - PROVIDER_URLがInterstage JDBCドライバに対応した記述形式になっていますので、詳細設定の「File System Service Providerを使用する」のチェックを解除しJDBCデータソース定義の.bindingsファイルへの登録を行わないようにしてください。もしくは管理コンソールのヘルプまたは「リファレンスマニュアル(コマンド編)」の「isj2eeadmin」の「リソース定義ファイル」を参照して正しいPROVIDER_URLを設定してください。
3. J2EEプロパティまたはIJServerの環境設定などに指定されているInterstage JDBCDriverのクラスパスを、Microsoft(R) JDBCドライバのクラスパスへ変更してください。

31.2.6 自動再接続機能について

Interstage V9.0より、JDBCデータソースの自動再接続機能がデフォルトで有効となります。自動再接続機能が有効になることによる運用上の問題はありませんが、データベース側のSQL文のログを参照した場合、接続可能かどうかを確認するために発行しているSQL文が多数出力されています。デバッグする場合などで自動再接続機能を無効にしたい場合は、Interstage管理コンソールまたはisj2eeadminコマンドで以下の定義変更を行い、自動再接続が行われないようにしてください。

・ Interstage管理コンソールの場合

以下の画面の「異常時の再接続」を「しない」に変更。
[ワークユニット] > [IJServer名] > [環境設定]

・ isj2eeadminコマンドの場合

IJServer定義ファイルの以下の項目をFALSEに指定して定義を変更。
IJServer > Datasources > Datasource > AbnormalReconnection

31.2.7 SQL実行の通信待ち時間監視について

Interstage V9.0よりSQL実行処理の通信待ち時間をデフォルトで監視します(デフォルト監視時間:400秒)。そのため、400秒のSQL実行処理時間を許容範囲とする業務システムの場合、アプリケーションが正常動作しているにも関わらずIJServer21265の警告メッセージが出力される場合がありますが、ユーザは出力されるメッセージから問題がないことを確認することができます。

警告メッセージの出力を抑止したい場合は、Interstage管理コンソール、もしくはisj2eeadminコマンドで以下の定義変更を行い、SQL実行の通信待ち時間監視を行わないようにしてください。

・ Interstage管理コンソールの場合

以下の画面の「通信待ち時間」を「0」に変更。
「ワークユニット」>「IJServer名」>「環境設定」

・ isj2eeadminコマンドの場合

IJServer定義ファイルの以下の項目を「0」に指定して定義を変更。
IJServer > Datasources > Datasource > SqlWaitTimeout > Time

31.2.8 コネクション使用時間監視について

Interstage V9.0よりトランザクション開始前に獲得したコネクションがクローズされるまでをデフォルトで監視します(デフォルト監視時間:60分)。そのため、アプリケーションで獲得したコネクションを意図的にクローズせずに使用する業務システムの場合、アプリケーションが正常動作しているにも関わらずIJServer21263の警告メッセージが出力される場合がありますが、ユーザは出力されるメッセージから問題がないことを確認することができます。

警告メッセージの出力を抑止したい場合は、Interstage管理コンソール、もしくはisj2eeadminコマンドで以下の定義変更を行い、コネクション使用時間監視を行わないようにしてください。

・ Interstage管理コンソールの場合

以下の画面の「コネクション使用監視時間」を「0」に変更。
「ワークユニット」>「IJServer名」>「環境設定」

・ isj2eeadminコマンドの場合

IJServer定義ファイルの以下の項目を「0」に指定して定義を変更。
IJServer > Datasources > Datasource > ConnectionUseTimeout > Time

31.2.9 クラスローダの変更について

Interstage Application Server V7.0(Interstage V7.0)以降では、アプリケーションのクラスのロードを行うクラスローダ機能が提供されました。クラスローダ機能の提供により、クラスローダのデフォルト設定が従来と異なります。このため、クラスローダの

分離方法をInterstage Application Server V6.0(Interstage V6.0)までの互換である「分離しない」以外を選択した場合は、以下の点に注意してください。

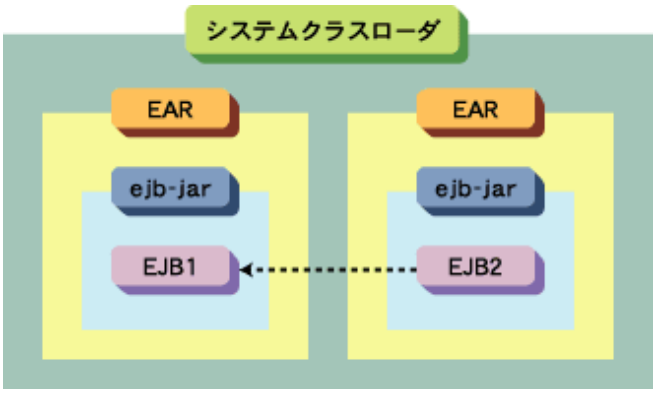
- 配備したモジュール間で参照関係がある場合、他のモジュールのクラスやEJBアプリケーションを正しく参照できないことがあります。
- 異なるクラスローダで同一名のクラスをロードした場合、ClassCastExceptionが発生することがあります。
- 環境変数CLASSPATHが有効となりません。
- メモリ使用量が変動する場合があります。「メモリ使用量について」を参照して、再度見積りを実施してください。

Interstage V6.0との違いについては、以下を参照してください。また、クラスローダの詳細については「2.3 クラスローダ」を参照してください。

IJServerのタイプが「Webアプリケーションと EJBアプリケーションを同一JavaVMで運用」である場合の、アプリケーション間の参照について

Interstage V6.0

EARファイルを配備すると、下図のように、EARファイルに含まれるEJBアプリケーションはシステムクラスローダでロードされました。そのため、同一プロセス内の他のEARファイルに含まれるEJBアプリケーションを参照できました。



Interstage V7.0以降

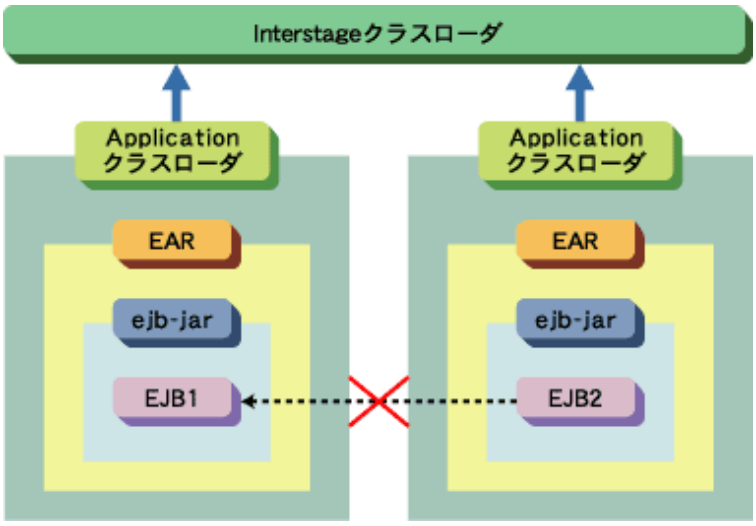
「クラスローダの分離」の設定値やアプリケーションの配備方法によって、アプリケーション間の参照可否が異なります。「クラスローダの分離」には「EAR間で分離」(デフォルト設定)を設定することをお勧めしますが、以下を参考にして、設定内容を検討してください。

1) 「クラスローダの分離」が「EAR間で分離」(デフォルト設定)の場合

EARファイルを配備すると、下図のように、クラスローダはEARごとに割り当てられます。このようにクラスローダを割り当てることにより、アプリケーション間の独立性が高められました。しかし、その一方で、同一プロセス内の他のEARに含まれるEJBアプリケーションを参照できなくなりました。

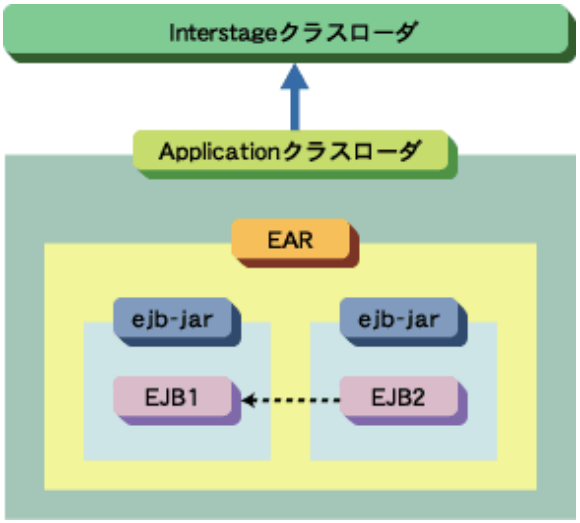
注意

ejb-jarを単体で配備した場合は、V6.0までと同様に、同一プロセス内の他の単体で配備したejb-jarに含まれるEJBアプリケーションを参照できます。



2) 同一プロセス内の他のEARに含まれるEJBアプリケーションを参照する方法(1)

下図のように、関連するEJBアプリケーションを1つのEARファイルに含めてIJServerに配備してください。

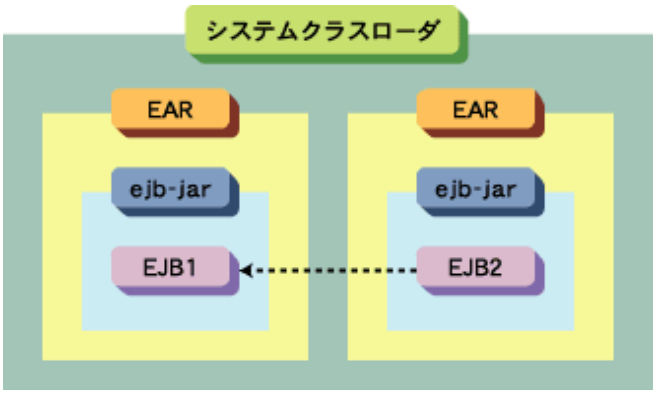


3) 同一プロセス内の他のEARに含まれるEJBアプリケーションを参照する方法(2)

関連するEJBアプリケーションを1つのEARファイルに含めることができない場合は、以下のどちらかの方法で、「クラスローダの分離」に「分離しない」を設定してください。

- [ワークユニット] > [新規作成] > [詳細設定] > [共通定義] > [クラスローダの分離]
- [ワークユニット] > [ワークユニット名] > [環境設定]> [詳細設定] > [共通定義] > [クラスローダの分離]

なお、上記は、isj2eeadminコマンドで設定することもできます。詳細は、「リファレンスマニュアル(コマンド編)」を参照してください。
クラスローダの分離については「2.3.2 クラスローダの分離」を参照してください。

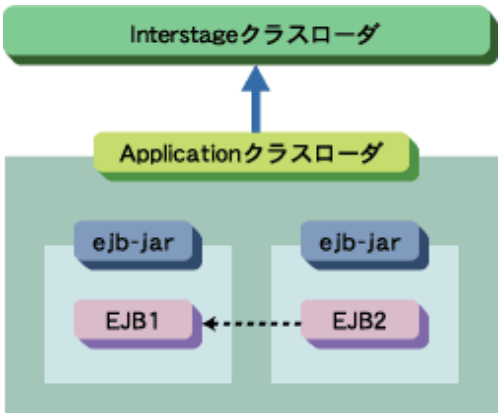


4) HotDeployを行う必要がある場合

「クラスローダの分離」に「分離しない」を設定した場合は、HotDeployができません。
HotDeployを行う必要がある場合は、以下のどちらかの対処をしてください。

- 関連するEJBアプリケーションを1つのEARファイルとしてIJServerに配備
- 下図のようにEARファイル内の個々のejb-jarファイルをIJServerに配備

この場合、「クラスローダの分離」に「EAR間で分離」を設定します。



CLASSPATHについて

Interstage V6.0では環境変数:CLASSPATHに設定された値はIJServer起動時に有効となっていました。
Interstage V7.0以降ではクラスローダの分離が「EAR間で分離」または「すべて分離」の場合、以下の理由により環境変数:CLASSPATHはIJServer起動時に有効となりません。

理由

環境変数:CLASSPATHには、Interstageだけではなくそのサーバ上で動作する他のシステムが使用するライブラリが指定されている場合や、無効なライブラリ(旧バージョンのライブラリなど)が指定されている可能性があります。そのため、環境変数:CLASSPATHをIJServer起動時に有効とすると、IJServerが誤動作する原因となります。

なお、クラスローダの分離が「分離しない」の場合は、Interstage V6.0と同様に環境変数:CLASSPATHはIJServer起動時に有効となります。この場合、環境変数:CLASSPATHはシステムクラスローダでロードされます。
IJServerのクラスの設定方法については、「2.3.4 IJServerで使用するクラスの設定について」を参照してください。

J2EEプロパティおよびワークユニットのクラスパスについて

クラスローダの分離が「EAR間で分離」または「すべて分離」の場合、J2EEプロパティまたはワークユニットのクラスパスに設定されたクラスはInterstageクラスローダでロードされます。
そのためJ2EEプロパティ、またはワークユニットのクラスパスとアプリケーション(EAR、ejb-jar、WAR、RAR)の両方に同じクラス名のクラスが存在する場合、そのクラスをアプリケーションから参照するとアプリケーションのクラスが使用されます。同じクラスをJ2EEプロパティ、またはワークユニットのクラスパスに設定されたクラスから参照すると、J2EEプロパティまたはワークユニットのクラスパスに設定されたクラスが使用されます。
これは何を意味するのかというと、J2EEプロパティまたはワークユニットのクラスパスに設定されたクラスとアプリケーションのクラスが物理的に同じファイルであっても、参照する場所によって別のクラスとして扱われ、ClassCastExceptionが発生する可能性を含んでいるということになります。
これを回避するためには以下の手段が有効となります。

- 同じクラス名のクラスが存在しないようにアプリケーションを設計してください。ワークユニットのクラスからアプリケーションのクラスを参照する必要がある場合は、カレントスレッドのコンテキストクラスローダを使用することでアプリケーションのクラスを動的にロードすることができます。
- クラスローダの分離を「分離しない」としてください。この場合はワークユニットのクラスパスに設定されたクラス、アプリケーションのクラスの両方でワークユニットのクラスパスに設定されたクラスが参照されます。

メモリ使用量について

クラスローダの分離方法に「EAR間で分離」(デフォルト設定)または「すべて分離」を選択した場合、HotDeploy機能などのクラスローダ機能の新機能を使用することができますが、EJBアプリケーションの規模に応じてメモリ使用量が変動します。したがって、クラスローダの分離方法を変更(「分離しない」以外を選択)した場合は、メモリ使用量について再度、見積りを実施してください。

メモリ使用量の概算値は、以下の式より算出してください。

[EJBアプリケーションクラスファイルの合計サイズ]×2.0[安全係数]

Interstage V6.0とクラスローダの構成が同じV6互換モードについて

HotDeployなどのクラスローダ機能のエンハンス機能が必要ない場合、本バージョン・レベルで作成したIJServerに、定義項目「クラスローダの分離」に「分離しない」を指定することで、クラスローダの構成をV6互換モードで動作させることができます。

JNIを使用する場合

同じNativeモジュールは同じクラスローダ上からのみ利用可能です。別々のクラスローダで同じNativeモジュールをロードしようとした場合、java.lang.UnsatisfiedLinkErrorがスローされます。
Interstage V6.0まではEJBアプリケーションはすべてシステムクラスローダでロードされていましたが、本バージョン・レベルではクラスローダの分離の定義によってEJBアプリケーションをロードするクラスローダが分離されます。そのため、別々のEJBアプリケーションから同じNativeモジュールをロードしようとした場合、別々のクラスローダで同じNativeモジュールがロードされることになり、java.lang.UnsatisfiedLinkErrorがスローされる可能性があります。「2.3.7 クラスローダ使用時の注意事項」を参照して対処を行ってください。

31.2.10 返却される例外の詳細文字列について

Interstage Application Server V7.0(Interstage V7.0)以降では、J2EEアプリケーション運用中に開始したトランザクションのコミット処理の延長で、DBMSの更新処理中に例外が発生した場合、DBMSで発生した例外情報をJ2EEアプリケーションに返却します。そのため、トランザクションコミット時に発生したデッドロックを含め、DBMSで発生した例外の検知ができます。
ただし、返却する例外の詳細文字列が変更となるため、詳細文字列を参照しているInterstage Application Server V6.0(Interstage V6.0)以前の製品で運用していたアプリケーションを、本バージョン・レベルに移行したときに問題が発生する可能性があります。Interstage V6.0以前と同様の詳細文字列を返却するには、IJServerのワークユニット設定に以下のように

指定します。なお、以下は、isj2eeadminコマンドで設定することもできます。詳細は、「リファレンスマニュアル(コマンド編)」を参照してください。

設定方法

パラメタ	値
Java VMオプション(Java Command Option)	-DCommitExSQLMsg=off

互換メッセージ内容

- javax.transaction.HeuristicRollbackException例外の詳細文字列

  「STATUS_ROLLEDBACK returned from commit method of UserTransaction.」

- javax.transaction.HeuristcMixedException例外の詳細文字列

  従来どおりメッセージを含まない。

31.2.11 JNDIから返却される例外について

Interstage V6.0より、J2EEアプリケーションクライアント、もしくはWebアプリケーションで行うlookup処理で、該当するオブジェクトが見つからなかった場合に返却される例外が異なります。

- InterstageV5.x以前の場合

  javax.naming.NameNotFoundException例外を返却します。

- InterstageV6.0以降の場合

  javax.naming.NamingException例外を返却します。

注意

以下のようにサブコンテキストを一度獲得し、サブコンテキストに対してlookup処理を実行した場合は従来通りjavax.naming.NameNotFoundException例外が返却されます。

```
// JDBCデータソース」DB1」のlookup例
javax.sql.DataSource dataSource = null;
try {
    javax.naming.Context nctx = new javax.naming.InitialContext();
    javax.naming.Context sctx = (javax.naming.Context)nctx.lookup("java:comp/env/ ");
    dataSource = (javax.sql.DataSource)sctx.lookup("jdbc/DB1");
} catch(javax.naming.NamingException ex) { }
```

javax.naming.NameNotFoundException例外はjavax.naming.NamingException例外のサブクラスのためlookup処理のcatch句はjavax.naming.NamingException例外で行うことを推奨します。

31.2.12 ORBプロパティ情報ファイル(orb.properties)について

Interstage V10.1以前では、ORBプロパティ情報ファイル(orb.properties)がInterstageインストール時に以下のディレクトリにインストールされていましたが、Interstage V11.0以降はインストールされません。

Windows32/64

(JDKの場合)%JAVA_HOME%¥jre¥lib

(JREの場合)%JAVA_HOME%¥lib

Solaris64 Linux32/64

(JDKの場合)${JAVA_HOME}/jre/lib

(JREの場合)${JAVA_HOME}/lib

ORBプロパティ情報ファイルの作成方法、または、javaコマンドのパラメタとしてORBを指定する方法については、以下を参照してください。

- 4.2.1 クライアント環境での環境設定
- 15.7.2 クライアント環境の設定

注意

ORBプロパティ情報ファイルによってORBを指定する場合、JDK6のJavaツールの実行に影響を与える場合があります。ORBプロパティ情報ファイルによるJavaツールへの影響を回避するため方法については、Intnerstage V11の「トラブルシューティング集」－「jvisualvm使用時の注意事項」および「jconsole使用時の注意事項」を参照してください。

なお、本注意事項は本製品には該当しません。

31.2.13 Fujitsu XMLプロセッサについて

本製品ではFujitsu XMLプロセッサを使用できません。Fujitsu XMLプロセッサを使用しているアプリケーションは、Xerces2など他のXMLパーサを使用するようにアプリケーションを修正してください。IJServerで使用できるXMLパーサや設定方法については、「31.2.3 XMLパーサのXerces2サポート」、「3.5.1 配備に必要なXMLパーサの設定」、および「2.3.5 XMLパーサの設定」を参照してください。

31.2.14 J2EEアプリケーションの移行時のその他の注意事項

InfoDirectoryからInterstage ディレクトリサービスへの移行について

本製品では、InfoDirectoryを同梱していません。
ユーザ/セキュリティロールの管理簿として、InfoDirectoryを使用していた場合、Interstage ディレクトリサービスへ移行してください。詳細については、「移行ガイド」－「InfoDirectoryからInterstage ディレクトリサービスへの移行」を参照してください。

配備時向け名前変換定義ファイルについて

Interstage Application Server V5.xで配備時向け名前変換定義ファイル(interstage.xml)を作成していた場合、その移行が必要になることがあります。詳細については「30.1 Servletサービス(Tomcat5.5ベースのサーブレット実行環境)への移行」を参照してください。

Interstage Application Server V5.x以前に作成したJDBCデータソースについて

Interstage Application Server V5.x以前で作成したJDBCデータソースをバックアップ、リストア(または移入)した場合、必須項目が不足しているためInterstage管理コンソールなどで参照した定義をそのまま更新できません。
定義更新が必要な場合は、必須項目を入力してください。

31.3 INTERSTAGE V3.xからJ2EEアプリケーションへの移行

Interstage Application Server V3.x(Interstage V3.x)からのJ2EEアプリケーションの移行について、以下を説明します。

- 運用方法の違い

- J2EEアプリケーションへの移行方法

運用方法の違い

Interstage V3.xでの運用方法と、J2EEとしての運用方法には、以下の違いがあります。

- パッケージ化
- JNDI
- セキュリティ

パッケージ化

Interstage V3.x

以下のように、アプリケーションごとにそれぞれ1つのアーカイブファイルとして流通することができます。

アプリケーション	アーカイブファイル
EJBアプリケーション	EJB JARファイル
Webアプリケーション	WARファイル

J2EE

以下に示すようなアプリケーションごとに作成されたアーカイブファイルを、さらにEnterprise ARchive (EAR)ファイルとしてパッケージ化することによって、運用で使用するアプリケーションすべてを1つのアーカイブファイルとして流通することができます。

アプリケーション	アーカイブファイル
EJBアプリケーション	EJB JARファイル
Webアプリケーション	WARファイル
J2EEアプリケーションクライアント	クライアントJARファイル
resource adapter	RARファイル

JNDI

Interstage V3.x

JavaアプリケーションやWebアプリケーションから、JNDIを使用してEJBやJDBCにアクセスする場合、以下のようにJNDIのサービスプロバイダを使い分ける必要があります。

- EJBにアクセスする場合
  com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient
- JDBCにアクセスする場合
  - Symfowareでは
    com.fujitsu.symfoware.jdbc2.jndisp.SYMContextFactory
  - Oracleでは
    com.sun.jndi.fscontext.RefFSContextFactory

J2EE

J2EEアプリケーションクライアントやWebアプリケーションから、JNDIを使用してEJBやJDBCにアクセスする場合、以下の設定をすることによってJNDIのサービスプロバイダを使い分ける必要はありません。

ー J2EEアプリケーションクライアントの場合
Javaに次のパラメタを指定します。
java.naming.factory.initial=com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient

セキュリティ

Interstage V3.x

認証や承認を実装するには、ユーザがおのおのの手法でセキュリティの機構を実装する必要があります。

J2EE

Interstage ディレクトリサービスを使用したセキュリティ機構によって、J2EEアプリケーションのユーザを一元管理し、J2EE全体で統一された認証や承認の仕組みを実装することができます。

J2EEアプリケーションへの移行方法

以下の場合について、Interstage V3.xからJ2EEアプリケーションに移行する方法について説明します。

- サーブレットーEJB連携の移行
- J2EEアプリケーションクライアントーEJB連携の移行

なお、本製品ではFujitsu XMLプロセッサを使用できません。詳細については「31.2.13 Fujitsu XMLプロセッサについて」を参照してください。

31.3.1 サーブレットーEJB連携の移行

J2EE環境に移行するための以下の手順を説明します。

- サーブレットのソース修正
- 定義の変更
- セキュリティ機能を使用する場合

注意

EJBの場合、Interstage V3.xとは動作に違いがある機能が存在しますので、Enterprise Beanを開発する際には注意が必要です。詳しくは「31.5 EJBサービスの移行」を参照してください。

サーブレットのソース修正

EjbClient.javaのlookup処理で指定する、参照するEJBアプリケーション名をEnterprise Bean Environment形式に変更します。

Interstage V3.x形式のソース

```
・・・
javax.naming.Context ic = new javax.naming.InitialContext();
java.lang.Object obj = (Object)ic.lookup("EjbServer");
home = (EjbServerHome)javax.rmi.PortableRemoteObject.narrow(obj, EjbServerHome.class);
・・・
```

J2EE形式のソース

```
・・・
javax.naming.Context ic = new javax.naming.InitialContext();
```

```
java.lang.Object obj = (Object)ic.lookup("java:comp/env/ejb/EjbServer");
home = (EjbServerHome)javax.rmi.PortableRemoteObject.narrow(obj, EjbServerHome.class);
・・・
```

EJBアプリケーションのlookup処理については、「4.10 オブジェクトの参照方法」を参照してください。

定義の変更

Webアプリケーション環境定義ファイルを設定します。
EJBオブジェクト参照の定義として、ejb-refタグまたはejb-local-refタグを追加します。

```
・・・
<ejb-ref>
    <ejb-name>ejb/EjbServer</ejb-Name>
    ・・・
</ejb-ref>
・・・
```

また、名前変換機能を使用する場合には「FJebeProperties.xmlファイル」に必要な名前変換の定義を指定する必要があります。
オブジェクト参照機能の詳細は「第4章 JNDI」を参照してください。

セキュリティ機能を使用する場合

セキュリティ機能を使用する場合は、Webアプリケーション環境定義ファイルにセキュリティの定義を追加します。

認証方式:HTTP Basic認証
アクセスが許されるセキュリティロール:Administrator
認証の範囲:sample以下のすべてのurl

```
・・・
<servlet>
・・・
    <security-role-ref>
        <role-name>ADM</role-name>
        <role-link>Administrator</role-link>
    </security-role-ref>
・・・
</servlet>
・・・
<security-constraint>
    <web-resource-collection>
        <web-resource-name>sample</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>Administrator</role-name>
    </auth-constraint>
</security-constraint>
・・・
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>sample</realm-name>
</login-config>
・・・
```

```
<security-role>
    <role-name>Administrator</role-name>
</security-role>
・・・
```

セキュリティ機能の詳細は、「第5章 J2EEアプリケーションのセキュリティ」を参照してください。

31.3.2 J2EEアプリケーションクライアントーEJB連携の移行

Enterprise Bean開発を例に、J2EEに移行する場合の手順の相違点を以下に説明します。

注意

EJBの場合、Interstage V3.xとは動作に違いがある機能が存在しますので、Enterprise Beanを開発する際には注意が必要です。詳しくは「31.5 EJBサービスの移行」を参照してください。

Step1:Enterprise Beanを開発する

J2EEのセキュリティ機能を使用する場合には、セキュリティロール定義とメソッドパーミッション定義が必要です。メソッドパーミッション機能を使用することにより、J2EEアプリケーションクライアントやWebアプリケーションの認証機能と連携して、EJBアプリケーションの不当アクセスを制限することができます。J2EEのセキュリティ機能を使用する場合の運用環境の設定は、「第5章 J2EEアプリケーションのセキュリティ」を参照してください。

ここでは、データベースの変更(レコードの追加/削除)を、管理者のみが行うことができるように制御する例を説明します。Session Beanの開発でdeployment descriptorを編集する際に以下のように定義を追加します。

1. Interstage StudioのXMLエディタを起動します。
2. [ソース]タブを選択し、セキュリティロールを追加します。ここでは、管理者権限を持つセキュリティロール「Admin」を追加します。

セキュリティロール名	説明
Admin	管理者

3. アクセスを制御する「TutorialSession」を選択し、「メソッドパーミッション」を定義します。レコードを更新する「insertRecord」メソッドと、レコードを削除する「deleteRecord」メソッドに対して次のように定義を追加します。

セキュリティロール名	メソッド名	インタフェース	パラメタリスト	説明
Admin	insertRecord	Remote	int,java.lang.String	更新
Admin	deleteRecord	Remote	int	削除

上記の設定を行うことによって、メソッドに設定されたセキュリティロールに属するユーザ以外がメソッドを実行した場合には例外が返却されます。

Step2:Enterprise Beanをパッケージ化する～ejb-jarファイルの作成～

この手順には特に相違はありません。

Step3:Enterprise Beanを実行可能な状態にする

EJBアプリケーションの展開～配備についてはInterstage管理コンソールで行うことができます。
また、データソースの登録についても、Interstage管理コンソールで行います。
詳細についてはInterstage管理コンソールのヘルプを参照してください。

Step4:クライアントアプリケーションを開発する

クライアントアプリケーションのlookup処理の記述形式を変更する必要があります。また、J2EEアプリケーションクライアントの機能(リソースアクセスなど)を使用するためにdeployment descriptorファイルの作成が必要です。

1. クライアントアプリケーションの修正

   Tutorial.javaのlookup処理で指定する、参照するEJBアプリケーション名をEnterprise Bean Environment形式に変更します。

   Interstage V3.x形式のソース

```
try{
    InitialContext ic = new InitialContext();
    java.lang.Object obj = ic.lookup("TutorialSession");
    home = (TutorialSessionHome)PortableRemoteObject.narrow
           (obj, TutorialSessionHome.class);
}catch(Exception ex){
・・・
```

   J2EE形式のソース

```
try{
    InitialContext ic = new InitialContext();
    java.lang.Object obj = ic.lookup("java:com/env/ejb/TutorialSession");
    home = (TutorialSessionHome)PortableRemoteObject.narrow
           (obj, TutorialSessionHome.class);
}catch(Exception ex){
・・・
```

   ポイント

   Interstage V3.xまでの記述でも動作可能ですが、J2EEアプリケーションクライアントではEnterprise Bean Environment形式でlookup処理を行うことを推奨しています。
   ただし、リソース接続者管理機能を使用する場合は、必ずEnterprise Bean Environment形式で記述してください。

2. deployment descriptorの作成

   リソースアクセスや名前変換機能を使用する場合にはdeployment descriptorファイルを作成する必要があります。EJBプロジェクトを作成後、以下のようにdeployment descriptorファイルを作成してください。

   1. [ファイル]メニューの[新規]の[その他]を選択します。
   2. [新規]ダイアログの[XML]の[XML]を選択します。
   3. [新規XMLファイル]ウィザードの[XMLファイルの作成]ページで必要な情報を設定し、保存して終了します。

   参照

   deployment descriptorの定義項目についての詳細は「5.2.4 J2EEアプリケーションクライアントの設定」を参照してください。

例

deployment descriptorファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application-client PUBLIC '-//Sun Microsystems, Inc.//DTD J2EE Application Client1.2//EN'
 'http://java.sun.com/j2ee/dtds/application-client_1_2.dtd'>

<application-client>
    <icon>
        <small-icon>smallicon.jpg</small-icon>
        <large-icon>largeicon.jpg</large-icon>
    </icon>
    <display-name>display</display-name>
    <description>Tutorial</description>
    <ejb-ref>
        <description>EJB Information</description>
        <ejb-ref-name>ejb/TutorialSession</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>TutorialSessionHome</home>
        <remote>TutorialSessionRemote</remote>
        <ejb-link>TutorialSession</ejb-link>
        </ejb-ref>
</application-client>
```

Step5:Enterprise Beanを運用する

J2EEアプリケーションクライアントの場合には、環境変数の設定、FJjndi.propertiesファイルの設定およびコマンドラインの変更が必要です。

1) 環境変数の設定

環境変数CLASSPATHに、次の値が設定されていない場合は設定してください。

Windows32/64

```
C:¥Interstage¥J2EE¥lib¥isj2ee.jar
```

Solaris64 Linux32/64

```
/opt/FJSVj2ee/lib/isj2ee.jar
```

2) FJjndi.propertiesファイルの設定

名前変換機能やセキュリティ機能を使用する場合には「FJjndi.propertiesファイル」に必要な環境プロパティを指定する必要があります。このファイルに設定するユーザIDとパスワードはInterstage ディレクトリサービスに設定する必要があります。

例

FJjndi.propertiesファイルの記述例

Windows32/64

```
FJUserID=j2ee
FJPassword=j2ee
com.fujitsu.interstage.j2ee.DeploymentDescriptorClient=C:¥work¥TutorialClient¥application-client.xml
```

Solaris64 Linux32/64

```
FJUserID=j2ee
FJPassword=j2ee
com.fujitsu.interstage.j2ee.DeploymentDescriptorClient=/export/home/j2eeapl/TutorialClient/application-client.xml
```

参照

FJjndi.propertiesファイルの詳細については、「5.2.4 J2EEアプリケーションクライアントの設定」を参照してください。
ユーザIDとパスワードのInterstage ディレクトリサービスへの設定方法については、「ディレクトリサービス運用ガイド」の「エントリの管理」を参照してください。

3) コマンドラインの変更

クライアントアプリケーションを実行するコマンドラインを以下のように変更します。

Interstage V3.xの場合

```
java -Djava.naming.factory.initial=com.fujitsu.interstage.ejb.jndi.FJCNCtxFactoryForClient Tutorial
```

J2EEの場合

```
java -Djava.naming.factory.initial=com.fujitsu.interstage.j2ee.jndi.InitialContextFactoryForClient Tutorial
```

31.4 Servletサービスの移行

ここでは、Servletサービスの移行について説明します。

WebサーバコネクタとServletコンテナ間でSSL通信を行う場合

Interstage Application Server V12.2.0で使用可能な暗号化方法(アルゴリズム)は、以下の通りです。
使用するSSL定義に以下のいずれかの暗号化方法が含まれていることを確認してください。
暗号化方法に以下のいずれも含まれていない場合は、Webアプリケーションに接続できない、またはワークユニットの起動に失敗することがあります。

- 256bitのAES暗号(GCM),AEAD
- 128bitのAES暗号(GCM),AEAD
- 256bitのAES暗号,SHA-256 MAC
- 128bitのAES暗号,SHA-256 MAC
- 256bitのAES暗号,SHA-1 MAC
- 128bitのAES暗号,SHA-1 MAC

リクエストのURIに'¥'を含む場合

リクエストのURIに'¥'(エンコードされている場合も該当)を含む場合、以下の製品では呼び出し可能であったコンテンツが呼び出しできなくなることがあります。リクエストのURIに'¥'を含まないようにアプリケーションを修正してください。

- Windows製品上で動作するInterstage Application Server/Interstage Web Server V9.0

セションIDの長さについて

Interstage Application Server 8.0以降では、セションIDの長さが長くなっています。
セションIDの一意性の範囲、桁数はServletサービスの実装により異なる可能性があります。セションIDをセション識別以外の目的で使用しないようにしてください。

サーブレット実行環境の変更

Interstage Application Server/Interstage Web Server V9.0から、Tomcat5.5ベースのサーブレット実行環境(Servletサービス)を提供しています。
本製品では、Tomcat4.1ベースのServletサービスと、Interstage Application Server V5.1以前のサーブレット実行環境であるTomcat3.1ベースのサーブレット実行環境(以降、Tomcat3.1ベースのServletサービス)を提供していません。以前のサーブレット実行環境からTomcat5.5ベースのサーブレット実行環境への移行については、「30.1 Servletサービス(Tomcat5.5ベースのサーブレット実行環境)への移行」を参照してください。

isj2eeadminコマンドを使用して環境を構築する場合

Interstage Application Server/Interstage Web Server V9.0から、IJServer定義ファイルとWebサーバコネクタ定義ファイルにWebサーバ名の指定が必要になりました。
Interstage Application Server 8.0から抽出したファイルを使用するには、これらのファイルにWebサーバ名の指定を行ってからisj2eeadminコマンドで適用してください。Webサーバ名の指定については、「リファレンスマニュアル(コマンド編)」の、「IJServer定義ファイル」および、「Webサーバコネクタ定義ファイル」を参照してください。

以前のバージョンで発生した障害修正の追加定義

本ソフトウェアでは、互換性維持のために追加定義により修正前後の動作を変更可能な障害修正が含まれています。ただし、以降に示す障害修正については修正後の動作がデフォルトになっているため、以前のバージョンで追加定義を設定していない場合は動作が変わります。

旧バージョンから移行した環境でも修正前の動作で運用する場合、Interstage管理コンソールまたはisj2eeadminコマンドで追加定義を設定してください。
追加定義は以下で設定します。

- Interstage管理コンソール

  [ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [詳細設定] > [ワークユニット設定] > [JavaVMオプション]

- isj2eeadminコマンド

  IJServer定義ファイルの下記タグの値
  /Isj2eeIjserverDefinition/IJServer/Common/JavaCommandOptions

設定方法の詳細については「Interstage管理コンソールヘルプ」または「リファレンスマニュアル(コマンド編)」を参照してください。

以下は、修正後の動作がデフォルトになっている障害修正の追加定義です。

Tomcat5.5ベースのServletサービス

PG62294

書式

-Dcom.fujitsu.interstage.jservlet.api.getRemoteHost.PG62294=true|false
(デフォルトは"true")

true :WebサーバでDNS逆引きに成功した時、getRemoteHostはクライアント端末のリモートホスト名を返却します。
false:getRemoteHostは常にクライアント端末のIPアドレスを返却します。(修正前の動作)

PG87697

書式

-Dcom.fujitsu.interstage.servlet.request.maxContentLength=最大値
(デフォルトは"10240")

- 最大値には、Servletコンテナが受け付けるHTTPリクエストボディサイズを1～2147483647バイトで指定します。
- 0以下を指定すると、「制限なし」になります。(修正前の動作)

31.5 EJBサービスの移行

ここでは、EJBサービスについて、以下を説明します。

31.5.1 Interstage Application Server V11.0.0での変更内容

Interstage Application Server V11.0.0(以降、Interstage V11.0)での変更内容を説明します。

EJBクライアントに指定するクラスパスについて

Interstage V11.0以降、以下のクラスライブラリは提供されなくなりました。

ファイル名	格納先
fjcontainer62.jar fjcontainer64.jar fjcontainer72.jar fjcontainer74.jar fjcontainer62_plugin.jar fjcontainer64_plugin.jar fjcontainer72_plugin.jar	Windows32/64 C:¥Interstage¥EJB¥lib、または、C:¥Interstage¥EJBCL¥lib Solaris32 Linux32/64 /opt/FJSVejb/lib、または、C:¥Interstage¥EJBCL¥lib

ファイル名	格納先
fjcontainer74_plugin.jar fjentity62_plugin.jar fjentity64_plugin.jar fjentity72_plugin.jar fjentity74_plugin.jar	

これらのクラスライブラリをクラスパスに設定している場合は、クラスライブラリのファイル名を以下のように書き換えてください。

Interstage V10.1以前で指定していたファイル名	Interstage V11.0以降で指定するファイル名
fjcontainer62.jar fjcontainer64.jar fjcontainer72.jar fjcontainer74.jar	fjcontainer94.jar
fjcontainer62_plugin.jar fjcontainer64_plugin.jar fjcontainer72_plugin.jar fjcontainer74_plugin.jar	fjcontainer94_plugin.jar
fjentity62_plugin.jar fjentity64_plugin.jar fjentity72_plugin.jar fjentity74_plugin.jar	fjentity94_plugin.jar

31.5.2 Interstage Application Server V9.0.1/V9.1での変更内容

Interstage Application Server V9.0.1(以降、Interstage V9.0.1)(RHEL-AS4(x86)、RHEL-AS4(EM64T)、RHEL5(x86)、RHEL5(Intel64)の場合)、Interstage Application Server V9.1(以降、Interstage V9.1)(Windows、Solarisの場合)の変更内容を説明します。

- [アプリケーション環境定義]の定義項目について
- Message-driven Beanの初期起動インスタンス数について

[アプリケーション環境定義]の定義項目について

EJB2.1に準拠したEJBアプリケーションの場合には、[システム] > [ワークユニット] > [IJServer名] > [EJBモジュール名] > [EJBアプリケーション名] > [アプリケーション環境定義]で表示していた以下の定義については表示されません。
内容を参照したい場合は、[システム] > [ワークユニット] > [IJServer名] > [EJBモジュール名]で表示されるdeployment descriptorの内容を確認してください。また、定義値を変更したい場合は、開発環境で生成されるejb-jar.xmlの内容を変更してください。

非表示となる定義情報	deployment descriptorファイルの対応するタグ
Enterprise Bean名	ejb-name
Homeインタフェース名	home

非表示となる定義情報	deployment descriptorファイルの対応するタグ
Remoteインタフェース名	remote
LocalHomeインタフェース名	local-home
Localインタフェース名	local
Enterprise Bean クラス名	ejb-class
セション種別	session-type
トランザクション管理種別	transaction-type
Persistence タイプ	persistence-type
抽象スキーマ名	abstract-schema-name
Primary Keyクラス名	prim-key-class
Primary Keyフィールド名	primkey-field
リエントラント	reentrant
メッセージセレクタ	message-selector
Destinationタイプ/サブスクライバの永続性	message-driven-destination
環境プロパティ	env-entry
参照EJB	ejb-ref
参照LocalEJB	ejb-local-ref
参照リソース	resource-ref
参照環境リソース	resource-env-ref
トランザクション	container-transaction
参照セキュリティロール	security-role-ref
メソッドパーミッション	method-permission
クエリ定義	query

また、以下の説明についても非表示となりました。
内容を参照したい場合は、対応するdeployment descriptorファイルのタグを参照してください。

非表示となる定義情報	deployment descriptorファイルの対応するタグ
セキュリティアイデンティティの説明	security-identityタグのdescriptionタグ

ポイント

従来のJ2EE1.3ベースで構築されたアプリケーションは従来どおりに運用可能です。J2EE1.3とJ2EE1.4のアプリケーションを同一のIJServer上で混在して運用することも可能です。

Message-driven Beanの初期起動インスタンス数について

Message-driven Beanの初期起動インスタンス数のデフォルト値が8となります。Interstage Application Server V9.0.0以前は1でした。初期起動インスタンス数は同時に処理するメッセージ数です。デフォルト値が変更されたことで、初期設定でのメッセージ処理性能が向上します。
初期起動インスタンス数を変更する場合にはInterstage管理コンソールを使用してEJBアプリケーションのアプリケーション環境設定で変更してください。また、ejbdefimportコマンドでも変更できます。

31.5.3 Interstage Application Server V9.0での変更内容

Interstage Application Server V9.0(以降、Interstage V9.0)での変更内容を説明します。

Message-driven Bean使用時のイベントチャネルの停止について

Message-driven Beanを使用し、JMSメッセージングモデルとトランザクション管理種別が下記表の○の場合、従来はイベントチャネルが停止するとIJServerプロセスを停止していました。しかし、IJServerを停止すると他の業務にも影響があるため、Interstage V9.0以降ではメッセージIJServer21457を出力し、IJServerプロセスの停止は行いません。

JMSメッセージングモデル	トランザクション管理種別		
	Bean	Container	
		トランザクション属性	
		Required	NotSupported
Point-To-Pointモデル	―	―	―
Publish/Subscribeモデル	○	○	―

Entity BeanのPrimary Keyクラスにデータベースの固定長項目をマッピングした場合の動作について

Primary Keyクラスにデータベースの固定長項目をマッピングしている場合、以下の処理シーケンスでビジネスメソッドを実行すると、“IJServer21221:Specified EJB object already deleted: NAME=<Entity Bean名>”というエラーが通知される場合があります。固定長項目をマッピングしているPrimary Keyクラスを実装する場合には、“13.9 Primary Keyクラスの作成”の“メソッド定義時の注意事項”に従って実装してください。

1. キーの文字列に空白を補完したPrimary Keyのインスタンス変数指定して、removeメソッドを実行
2. キーが1と同じ文字列で空白を補完しないPrimary Keyのインスタンス変数を指定して、createメソッドを実行
3. 1と同じPrimary Keyのインスタンス変数を指定して、findByPrimaryKeyを実行
4. 3で取得したEJB Objectのビジネスメソッドを実行

31.5.4 Interstage Application Server 8.0での変更内容

Interstage Application Server 8.0(以降、Interstage 8.0)での変更内容を説明します。

- EJBのプロセスディスパッチ方式の変更
- STATELESS Session Beanのインスタンス生成モード
- Message-driven Beanのスレッドプール
- EJBサービス運用コマンド
- Message-driven Bean機能拡張

EJBのプロセスディスパッチ方式の変更

Interstage Application Server 8.0でEJBコンテナのディスパッチ方式が改善され、ラウンドロビン方式で動作します。Interstage Application Server V7.0はLRU方式で動作しました。
以下に各プロセスディスパッチ方式について説明します。ラウンドロビン方式の場合には各プロセスに処理が均等に割り振られるため、処理が集中した場合の性能改善効果や危険分散効果があります。

プロセスディスパッチ方式	ディスパッチ論理
ラウンドロビン方式 (8.0のディスパッチ方式)	要求待ちのサーバアプリケーションプロセスの中で、最初に要求待ちとなったプロセスに、クライアントからの要求メッセージを振り分けます。 クライアントからの要求数が少ない場合は、各プロセスに均等に要求メッセージが振り分けられます。
LRU方式 (7.0までのディスパッチ方式)	要求待ちのサーバアプリケーションプロセスの中で、最後に要求待ちとなったプロセスに、クライアントからの要求を振り分けます。 クライアントからの要求数が少ない場合は、特定のプロセスのみに要求メッセージが振り分けられます。

STATELESS Session Beanのインスタンス生成モード

以下の条件を満たす場合、STATELESS Session Beanのインスタンス生成タイミングのチューニングが行えません。

- Interstage Application Server V6.0またはV7.0を使用する
- Enterprise Bean定義情報のSTATELESS他Beanアクセス参照機能に「At Start-Up」を指定しているアプリケーションを、8.0に移行する

この場合、At Start-Up定義が有効となり、インスタンス生成タイミングのチューニングが行えません。
インスタンス生成タイミングのチューニングを行いたい場合は、ejbdefexport/ejbdefimportコマンドを使用し、<stateless-instance-create-type?>タグを削除するか、または、タグの値を「At First Access」に変更して下さい。

ejbdefexport/ejbdefimportコマンドについては、「16.2 Enterprise Bean定義情報のexportとimport」、および「リファレンスマニュアル(コマンド編)」の「EJBサービス運用コマンド」を参照してください。

Message-driven Beanのスレッドプール

Interstage Application Server 8.0では、Message-driven Beanはスレッドをプーリングすることが可能となるため、以下を見直してください。

- Message-driven Beanの同時処理数の最大値が、デフォルトで64となります。

  デフォルトで65以上のメッセージがIJServerプロセスに配信されると、使用中のスレッドがプールに返却されるまで待機します。
  プールに返却されるまで待機させない場合には、各Message-driven Beanの初期起動インスタンス数の合計値を、Interstage管理コンソールまたはisj2eeadminコマンドで、Message-driven Beanの同時処理数の最大値に指定してください。

- Message-driven Bean、またはMessage-driven Beanから呼び出すアプリケーションでスレッド変数などを使用していた場合には、動作が変わる可能性があります。

  スレッド変数を使用したアプリケーションを使用している場合には、スレッド変数を使用しないように変更するか、アプリケーション開始または終了時にスレッド変数をクリアするように、アプリケーションを変更してください。
  アプリケーションの変更ができない場合には、以下のオプションを設定して、スレッドプールを使用しないように設定してください。
  FJEJBconfig.propertiesを編集した場合は、設定を有効にするためIJServerを再起動してください。

注意

スレッドプールを使用しないよう設定した場合、Interstage管理コンソールのIJServerモニタ画面の「Message-driven Bean スレッド情報」に表示される値は更新されません。

項目	設定内容
定義ファイル格納ディレクトリ	Windows32/64 Interstage インストールディレクトリ¥ejb¥etc Solaris64 Linux32/64 /opt/FJSVejb/etc
定義ファイル名	FJEJBconfig.properties
指定するキー	"mdb.thread.pool"
指定する値	・ ON(デフォルト) Message-driven Beanでスレッドプールを使用する。 ・ OFF Message-driven Beanでスレッドプールを使用しない。 大文字・小文字は区別しません。

EJBサービス運用コマンド Solaris64 Linux32/64

Interstage Application Server 8.0では、アプリケーションファイルの権限を指定する機能がサポートされたため、以下のコマンドはアプリケーションファイル保護レベルで指定したユーザで使用してください。
アプリケーションファイル保護レベルについては、「3.5 J2EEアプリケーションの配備と設定」を参照してください。

- ejbdefexport
- ejbdefimport

Message-driven Bean機能拡張

異常時のメッセージ退避機能

以下の条件を満たす場合、リトライ回数を超過してもメッセージ受信を繰り返す可能性があります。

- Point-To-Pointメッセージングモデル(1受信者だけにメッセージを配信するモデル)
- 不揮発化チャネルを利用

イベントチャネルのイベントデータをメモリにキャッシュする数を、イベントチャネルに蓄積できるイベントデータの最大値よりも大きく設定してください。
設定はイベントサービス運用コマンドを使用して行います。詳細は「リファレンスマニュアル(コマンド編)」の「essetcnf」および「essetcnfchnl」を参照してください。

Point-To-Pointメッセージングモデルを利用する場合、以下の非互換があります。

- V6.0以前、またはWindows(R)のV7.0のInterstage JMSサービスを使用してノーティフィケーションサービスのイベントチャネルに送信されたメッセージは、リトライ回数を超過しても退避されずにメッセージ受信を繰り返します。

- setRollbackOnlyメソッドを実行してトランザクションをロールバックした場合、V6.0(Windows(R)の場合はV7.0)以前のInterstageではメッセージが退避されませんでしたが、今版よりsetRollbackOnlyメソッドを実行してトランザクションをロールバックした場合もメッセージが退避されます。

上記に該当する場合は、以下の互換オプションを設定して運用してください。ただし、互換オプションを設定した場合、JMSが受信したメッセージは同時に処理されません。

設定ファイルの格納先

Windows32/64

C:¥Interstage¥EJB¥etc¥FJEJBconfig.properties

Solaris64 Linux32/64

/opt/FJSVejb/etc/FJEJBconfig.properties

設定内容

MDB_ReciveType=reciver

31.5.5 Interstage Application Server V7.0での変更内容

Interstage Application Server V7.0(以降、Interstage V7.0)での変更内容を説明します。

- 規約非準拠のEJBアプリケーションの配備について
- STATEFUL Session Beanの無通信監視時間
- finderメソッドの復帰値collectionの各種APIサポート
- IPCOMとの連携
- EJB QL拡張
- セションタイムアウト機能・トランザクションタイムアウト機能
- 同一EJBアプリケーション名の配備

規約非準拠のEJBアプリケーションの配備について

Interstage Application Server V7.0以降では、EJBアプリケーションを配備する際、EJB規約に準拠しているかのチェックが緩和されています。
以下が、V7.0以降で運用可能となるEJB規約の制限です。

No.	Bean種別	チェック対象	チェック内容
1	Enterprise Beanクラス	すべて	ビジネスメソッドがstaticで定義されている。
2	Enterprise Beanクラス	Entity Bean	Enterprise Beanクラスに定義されているejbCreateメソッドの復帰値とdeployment descriptorに設定されている<prim-key-class>は一致しているが、Enterprise Beanクラスの親のクラスに定義されているejbCreateメソッドの復帰値が<prim-key-class>と一致しない。
3	Enterprise Beanクラス	BMP Entity Bean	Enterprise Beanクラスに定義されているejbFind<～>メソッドのthrows句にある例外が、Homeインタフェースに定義されているfind<～>のthrows句にない。

No.	Bean種別	チェック対象	チェック内容
4	Enterprise Beanクラス	Message-driven Bean	ejbCreateメソッドのthrows句にjavax.ejb.CreateExceptionがある。
5	Enterprise Beanクラス	CMP2.0 Entity Bean	ejbCreateメソッドのthrows句にjavax.ejb.CreateExceptionがない。
6	Primary Keyクラス	CMP Entity Bean	deployment descriptorに<primkey-field>が指定されておらず、Primary Keyクラスにpublicではないフィールドがある。
7	Primary Keyクラス	CMP Entity Bean	deployment descriptorに<primkey-field>が指定されておらず、<cmp-field>にないフィールドがPrimary Keyクラスに定義されている。
8	Homeインタフェース	Session Bean Entity Bean	Createメソッドのthrows句にjavax.ejb.CreateExceptionがない。
9	deployment descriptor	CMP2.0 Entity Bean	<multiplicity>に「one」または「many」を指定している

STATEFUL Session Beanの無通信監視時間

無通信監視時間のデフォルト値が、30分に変更になりました。30分以上、STATEFUL Session Beanにアクセスしないアプリケーション構成の場合は、以下のエラーが発生する可能性があるため、無通信監視時間を大きく設定してください。

```
java.rmi.NoSuchObjectException: CORBA OBJECT_NOT_EXIST 0 No; nested
exception is: org.omg.CORBA.OBJECT_NOT_EXIST: CORBA_Request_get_response
minor code: 0 completed: No
```

無通信監視時間は、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [EJBアプリケーション]からSTATEFUL Session Beanを選択して、Interstage拡張情報の「無通信監視時間」に設定してください。

finderメソッドの復帰値collectionの各種APIサポート

Interstage Application Server V7.0より、Entity Beanの復帰値Collection型finderメソッドを実行して取得したjava.util.Collectionオブジェクトに対して、java.util.Collectionインタフェースに定義されているすべてのメソッドが使用可能となりました。

注意

CMP1.1およびBMPで使用する場合には、以下の条件があります。以下の条件で使用しない場合は、V6.0以前の範囲になります。CMP2.0では無条件に使用できます。

- IJServerのタイプが、WebアプリケーションとEJBアプリケーションを同一JavaVMで運用

  トランザクション属性に「Mandatory」以外のトランザクション属性を指定して下さい。

- IJServerのタイプが、WebアプリケーションとEJBアプリケーションを別JavaVMで運用 または、EJBアプリケーションのみ運用

  トランザクション属性にMandatory以外のトランザクション属性を指定するか、トランザクション属性に「Mandatory」を指定する場合は、Interstage管理コンソールの[アプリケーション環境定]より、「ローカル呼出し」に「しない」を設定して下さい。

IPCOMとの連携

Interstage Application ServerでIPCOMと連携する場合、V6.0ではEJBアプリケーションを配備した後に、配備したアプリケーションごとにオブジェクトリファレンスの再登録(オブジェクトリファレンスの確認/削除/再登録)が必要でした。
V7.0以降ではIJServerに負荷分散ホスト名を指定するだけで、EJBアプリケーションの配備時に自動的に環境設定を行い

ますので、EJBアプリケーションごとの操作は不要です。また、配備済みのEJBアプリケーションに対して、「IJServerの環境設定」の内容を更新するだけで環境設定ができます。
IJServer単位で設定した場合には、非互換はありません。

EJBアプリケーションごとにIPCOMと連携する場合、V6.0と同様にCORBAサービスのコマンドを使用して、アプリケーションの配備後にオブジェクトリファレンスを再登録する手順で設定を行ってください。

1. odlistns -lコマンドを実行してオブジェクトリファレンス情報を表示して、負荷分散対象のEJBアプリケーションのインタフェースリポジトリ名とインプリメンテーションリポジトリ名を確認します。

   以下にオブジェクトリファレンスの詳細情報の見方を示します。

```
Name(Type)       Object information(detail)
                 Default object information(detail)
EasyBean(o)      RMI:pkgTest.EasyBeanHome:0000000000000000,
  (a)                              (b)
IDL:com.fujitsu.interstage.j2ee.ijserver/IJServer001:1.0,
(march:8002:1.1:UNICODE(UCS2))
                                 (c)
```

   a) EJBアプリケーション名
   b) インタフェースリポジトリ名
   c) インプリメンテーションリポジトリ名

2. ネーミングサービスに登録されたオブジェクトリファレンスを削除します。

   以下のコマンドを実行して負荷分散したいEJBアプリケーションのオブジェクトリファレンスを削除してください。

```
OD_or_adm -d -n [EJBアプリケーション名]
```

3. オブジェクトリファレンスを登録します。

   以下のように仮想IPアドレスのホスト名を指定して、1.で確認した情報を指定してオブジェクトリファレンスを再登録します。

```
OD_or_adm -c [インタフェースリポジトリID] -a [インプリメンテーションリポジトリID]
 -h [仮想IPアドレスのホスト名] -n [アプリケーション名] -p [ポート番号]
```

例

```
OD_or_adm -c RMI:pkgTest.EasyBeanHome:0000000000000000
 -a IDL:com.fujitsu.interstage.j2ee.ijserver/IJServer001:1.0 -h vhost -p 8002 -n  EJB001
```

   (vhostは、仮想IPアドレス)

注意

オブジェクトリファレンスの再登録後、Interstage管理コンソールの[ワークユニット] > [IJServer名] > [環境設定]からIJServerの更新を行った場合には、再度オブジェクトリファレンスの再登録を行ってください。

EJB QL拡張

Interstage Application Server V7.0以降でEJB QLを使用する場合、デフォルトでV6.0のEJB QLの機能を拡張したEJB QL拡張が設定されています。

EJB QL拡張を使用しない場合

拡張機能を使用しない(EJB2.0規約の範囲内で運用する)場合は、Interstage管理コンソールの[システム] > [環境設定] の詳細設定の下にある[EJBサービス詳細設定] を参照し、「CMP2.0のEJB QL拡張」の定義項目を「使用しない」に変更してください。

EJB QL拡張を使用しない場合、以下の拡張機能が制限されます。

- 比較述語、BETWEEN述語の値式にjava.lang.Stringが指定できる
- LIKE述語のパターンに変数が指定できる
- LIKE述語のエスケープ文字に変数が指定できる
- NULLに変数が指定できる
- INに変数と数値が指定できる
- ORDER BY句が指定できる
- 演算子MODが指定できる
- 集合関数(MIN、MAX、AVG、SUM、COUNT)が指定できる

セションタイムアウト機能・トランザクションタイムアウト機能

Interstage V7.0以降では、セションタイムアウト機能、トランザクションタイムアウト機能は使用できません。
これらの機能は、Interstage V6.0以前では非推奨機能であり旧資産との互換は保証されているため、代替機能への移行を推奨します。代替機能への移行方法については、「非推奨機能」を参照してください。

同一EJBアプリケーション名の配備

1つのIJServerに同じJNDI名のEJBアプリケーションが配備されないように、EJBアプリケーション名の重複チェックを行います。
IJServerのタイプが「WebアプリケーションとEJBアプリケーションを同一JavaVMで運用」の場合、V6.0では異なるモジュールでも上書きしていましたが、V7.0以降ではモジュールが異なる場合に上書きを行わないようチェックを行います。

31.5.6 Interstage Application Server V6.0での変更内容

Interstage Application Server V6.0(以降、Interstage V6.0)での変更内容を説明します。

デフォルト設定の追加

旧バージョンでMessage-driven Beanを運用する場合、EJBアプリケーション環境定義にJMSコネクションファクトリ名とDestination名の定義が必須でした。
8.0では、JMSコネクションファクトリ名とDestination名を定義しなかった場合、以下のデフォルト設定で動作します。

・ JMSコネクションファクトリ名

　― Topicの場合:TopicCF001

　― Queueの場合:QueueCF001

・ Destination名

　EJBアプリケーション名をデフォルトのDestination名とします。

また、旧バージョンにおいて、トランザクション管理種別に「Container」が指定されていてトランザクション属性が設定されていないEJBアプリケーションは、運用することができませんでした。
8.0では、コンテナが自動的に「Required」が指定されたものとしてトランザクションの制御を行います。

非推奨機能

以下の機能群は、V6より非推奨機能となりました。いずれも旧資産との互換は保証されていますが、次期バージョンでは提供されない可能性があるため、代替機能への移行を推奨します。
なお、これらの機能に対する定義操作は、Interstage管理コンソールではサポートされていません。

・ セションタイムアウト機能

　STATEFUL Session Beanの無通信監視機能をサポートしたことにより、セションタイムアウト機能は非推奨機能となっています。
　Interstage管理コンソールからは定義できません。
　以前のバージョン・レベルでセションタイムアウト機能を使用し、本バージョン・レベルで引き続き使用する場合は、以前のバージョン・レベルにてセションタイムアウト機能を使用していたEJBアプリケーション実行環境をejbdefexportコマンドを使用して移出し、本バージョン・レベルへejbdefimportコマンドで移入してください。
　詳細は「第16章 運用コマンドを使用してカスタマイズする方法」を参照してください。
　下位互換性のために、旧資産をそのまま使用する場合には引き続きサポートされます。
　代替機能である「STATEFUL Session Beanの無通信監視機能」については、「10.5.3 STATEFUL Session Beanの無通信監視機能」を参照してください。

注意

セションタイムアウト機能を使用する場合は、STATEFUL Session Beanの無通信監視機能を使用することはできません。

・ トランザクションタイムアウト機能

　EJBのトランザクションタイムアウト機能は非推奨機能となっています。
　Interstage管理コンソールからは定義できません。
　以前のバージョン・レベルでトランザクションタイムアウト機能を使用し、本バージョン・レベルで引き続き使用する場合は、以前のバージョン・レベルにてトランザクションタイムアウト機能を使用していたEJBアプリケーション実行環境をejbdefexportコマンドを使用して移出し、本バージョン・レベルへejbdefimportコマンドで移入してください。
　詳細は「第16章 運用コマンドを使用してカスタマイズする方法」を参照してください。
　トランザクションタイムアウトの代わりにワークユニットの時間監視機能を使用してください。
　ワークユニットの時間監視をIJServerワークユニット作成時に設定する場合は、Interstage管理コンソールを使用する場合、[詳細設定] > [ワークユニット設定]で行います。また作成済みのIJServerワークユニットに対して設定する場合は、[ワークユニット] > [環境設定] > [ワークユニット設定]から「アプリケーション最大処理時間」を設定してください。
　Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
　isj2eeadminコマンドを使用する場合は、「リファレンスマニュアル(コマンド編)」の「isj2eeadminコマンド」を参照して設定してください。

・ 最大メモリ量の設定

IJServerの最大メモリ量をIJServerワークユニット作成時に設定する場合は、Interstage管理コンソールを使用する場合、[詳細設定] > [ワークユニット設定]で行います。また作成済みのIJServerワークユニットに対して設定する場合は、[ワークユニット] > [環境設定] >[ワークユニット設定]から「JavaVMオプション」において以下を指定してください。

－ -Xmx[最大メモリ量]m

Interstage管理コンソールの詳細については、Interstage管理コンソールのヘルプを参照してください。
isj2eeadminコマンドを使用する場合は、「リファレンスマニュアル(コマンド編)」の「isj2eeadminコマンド」を参照して設定してください。
V5で指定可能であったワークユニット定義項目のEJBアプリケーション最大メモリ量(Maximum Memory for EJB Application)は、ワークユニットの設定項目に表示されません。

同時処理数について

V5で提供されたIJServerでは同時処理数のデフォルトは40でしたが、8.0で提供されたIJServerでは同時処理数の最大値が64/最小値が16で動作します。
同時処理数の最大値/最小値は、以下で行います。

・ Interstage管理コンソールの[ワークユニット] > [EJBコンテナ設定]の「同時処理数」

・ isj2eeadminコマンドの「ejb」タグ

isj2eeadminコマンドを使用する場合は、「リファレンスマニュアル(コマンド編)」の「isj2eeadminコマンド」を参照して設定してください。

性能オプションについて

V5でサポートしていた以下の性能オプションについては、デフォルトで動作します。
Interstage管理コンソールで指定する必要はありません。

・ EJBオブジェクトの共用

・ 複数レコードの一括更新

・ SQL文のキャッシュ

STATELESS Session Beanのインスタンス数について

STATELESS Session Beanのインスタンス数を定義する必要がなくなりました。
STATELESS Session Beanのインスタンスは、STATELESS Session Beanへの初回アクセス時に「同時処理数の最大値」に指定された値の数だけ作成されます。

ログ出力について

V5で各ファイルに出力されていた以下の情報については、IJServerのログ(Windows(R)システムの場合、J2EE共通ディレクトリ¥ijserver¥[IJServer名]¥log配下のファイル、Solaris/Linuxシステムの場合、J2EE共通ディレクトリ/ijserver/[IJServer名]/log配下のファイル)に出力されます。
なお、従来から出力しているイベントログのメッセージについては、従来通りイベントログにもメッセージが出力されます。

・ アプリケーションの標準出力、標準エラー出力

・ EJBコンテナログ

・ EJBコンテナのエラーメッセージ

- EJBのスナップ出力
- connectorのログ出力 (以下の注意参照)

注意

connectorのログ出力先を、従来と同じ場所に出力する場合は、以下を設定してください。

定義ファイル格納ディレクトリ	Windows32/64 Interstageインストールディレクトリ¥J2EE¥etc¥jca Solaris64 Linux32/64 /opt/FJSVj2ee/etc/jca
定義ファイル名	jca.properties
指定するキー	log.file.option
指定する値	「V5」を指定します。

例

log.file.option=V5

EJBアプリケーション名について

EJBアプリケーション名に「:」を使用することはできません。V5で「:」を使用していた場合には、配備時にEJBアプリケーション名を変更してください。
なお、名前を変更したEJBアプリケーション名を、アプリケーション名を変更せずにlookupする場合には、名前変換機能を使用してください。

31.5.7 EJBアプリケーション移行時の注意点

本バージョン・レベルで提供する追加機能を使用する場合

旧バージョン・レベルで開発したEJBアプリケーションを、動作させるためには、以下の作業が必要となります。

- Interstage Application Server V4.x以前からの移行時

  EJB2.0規約またはEJB2.1規約に準拠したEJBアプリケーションに変更する場合は、deployment descriptorをEJB2.0またはEJB2.1の形式にする必要があります。

31.5.8 Interstage Application Server V5.x以前からの移行

Interstage Application Server V5.x以前からの移行については、「30.2 EJBサービス(IJServer)への移行方法」を参照してください。

31.6 Interstage JMSの移行

ここでは、Interstage JMSについて、以下を説明します。

- Interstage Application Server V11.0での変更内容
- Interstage Application Server V9.0での変更内容
- Interstage Application Server 8.0での変更内容
- Interstage Application Server V7.0での変更内容

注意

Interstage Application Server V5.x以前からInterstage JMSを移行する場合、イベントサービス運用コマンドの使用について注意すべき点があります。「移行ガイド」－「イベントサービスの移行」を参照して確認してください。

31.6.1 Interstage Application Server V11.0での変更内容

Interstage Application Server V11.0(以降、Interstage V11.0)での変更内容を説明します。

ORB(Object Request Broker)の指定について

Interstage Application Serverのインストール時には、JDK/JREのインストールディレクトリにorb.propertiesファイルが格納されません。JMSアプリケーションを実行する場合は、必ずORB(Object Request Broker)を指定してください。ORB(Object Request Broker)の指定方法については、「23.3 JMSアプリケーション運用マシンの運用前の環境設定」を参照してください。

31.6.2 Interstage Application Server V9.0での変更内容

Interstage Application Server V9.0(以降、Interstage V9.0)での変更内容を説明します。

JMS1.0.2規約からJMS1.1規約への変更について

Interstage 8.0以前までJava Message Service 1.0.2規約をサポートしていましたが、Interstage V9.0以降においてJava Message Service 1.1規約をサポートするように変更となりました。
Java Message Service 1.1規約では、Java Message Service 1.0.2規約に従ったアプリケーションも正常に動作することを保証していますが、Interstage V9.0以降でJMS1.0.2規約のアプリケーションを実行すると、Interstage 8.0以前の環境で実行した場合とエラーチェックおよび例外情報が異なることがあります。
Interstage 8.0以前の環境で作成したアプリケーションを、Interstage V9.0以降の環境で同様の動作で実行する場合は、アプリケーション実行時にシステムプロパティに以下のように指定してください。

システムプロパティ名

com.fujitsu.interstage.jms.exception_version_lower

指定方法

「yes」を設定します。

```
-Dcom.fujitsu.interstage.jms.exception_version_lower=yes
```

31.6.3 Interstage Application Server 8.0での変更内容

Interstage Application Server 8.0(以降、Interstage 8.0)での変更内容を説明します。

EJBのMessage-driven Bean使用時の接続コンシューマ数について Windows32/64

Interstage 8.0以降では、EJBのMessage-driven Beanを使用している場合、イベントチャネルの接続コンシューマ数に以下の値が加算されます。

- Point-To-Pointメッセージングモデル
  接続コンシューマ数の加算値は、通信状態により初期値から最大値まで増加します。

```
process × instance(初期値) ≦ 接続コンシューマ数の加算値 ≦ process × instance × 2(最大値)
```

  process:IJServerの[プロセス多重度]
  instance:Message-driven Beanの[初期起動インスタンス数(同時実行スレッド数)]

- Publish/Subscribeメッセージングモデル

```
接続コンシューマ数の加算値 ＝ 1
```

31.6.4 Interstage Application Server V7.0での変更内容

Interstage Application Server V7.0(以降、Interstage V7.0)での変更内容を説明します。

- V7.0以降のアプリケーションとV6.0以前のアプリケーションを混在して運用する場合
- EJBのMessage-driven Bean使用時の接続コンシューマ数について
- メッセージの出力について

V7.0以降のアプリケーションとV6.0以前のアプリケーションを混在して運用する場合

V7.0以降の送信用アプリケーションからV6.0以前の受信用アプリケーションに対してメッセージを送信することはできません。本運用を行った場合、V6.0以前のアプリケーションにおいてエラーメッセージjms2510が出力されて、異常終了してしまいます。
V7.0以降の送信用アプリケーションからV6.0以前の受信用アプリケーションに対してメッセージを送信する場合は、V7.0以降のシステムで環境変数JMS_RECEIVER_VERSION_LOWERに「ON」を設定してください。

EJBのMessage-driven Bean使用時の接続コンシューマ数について Solaris32 Solaris64 Linux32/64

Interstage V7.0以降では、EJBのMessage-driven Beanを使用している場合、イベントチャネルの接続コンシューマ数に以下の値が加算されます。

- Point-To-Pointメッセージングモデル

  通信状態により、接続コンシューマ数が初期値から最大値まで増加します。

```
接続コンシューマ数(初期値) ＝ ワークユニット(IJServer)の[プロセス多重度] ×
  Message-driven Beanの[初期起動インスタンス数(同時実行スレッド数)]
```

```
接続コンシューマ数(最大値) ＝ ワークユニット(IJServer)の[プロセス多重度] ×
  Message-driven Beanの[初期起動インスタンス数(同時実行スレッド数)] × 2
```

- Publish/Subscribeメッセージングモデル

```
接続コンシューマ数 ＝ 1
```

メッセージの出力について

Solaris32 Solaris64

Interstage V7.0以降において、日本語環境でシステムログに出力されるメッセージを日本語から英語に変更しています。

Linux32/64

Interstage V6.0以降において、日本語環境でシステムログに出力されるメッセージを日本語から英語に変更しています。

以前のバージョン・レベルと同様、日本語のメッセージを出力する場合は、以下のJava(TM)VMのシステムプロパティに設定を行ってください。

プロパティ名	プロパティ値
com.fujitsu.interstage.jms.message_format	「V5」を指定します。 V5を指定した場合、日本語のメッセージが出力されます。 注) UTF-8環境の場合、「V5」は指定できません。

プロパティ値が誤っている場合は、変更後の形式でメッセージが出力されます。

31.7 Interstage Webサービスの移行

Interstage Application Server 8.0から移行する場合の注意事項

非ascii文字の通信形式

Interstage Application Server 8.0のInterstage Webサービスでは、通信における文字列データ中の非ascii文字は、XMLの文字参照形式に変換されてascii文字を用いた形式で送信されていました。
Interstage Application Server V9.0からは、非ascii文字も文字参照形式に変換せずそのまま送信されます。

XML規約に従っている限り、受信側でこの変更によるデータの違いは発生しません。
Interstage Application Server 8.0と同様に、非ascii文字をXMLの文字参照形式に変換してascii文字を用いた形式で送信する必要がある場合は、「19.3 Webサービス設定ファイル」を参照してください。

Interstage Application Server V10.1以前のWebサービスアプリケーションを移行する場合の注意事項

- Webサービスアプリケーション、Webサービスクライアントアプリケーションをコンパイルする場合
  「18.1.2 Webサービスアプリケーションを開発する」および「18.2.2 スタブを生成する」を参照してください。
- Webサービスクライアントアプリケーションを実行する場合
  「19.2 Webサービス(クライアント機能)の運用方法」を参照してください。

注意

Webサービスアプリケーション内でカレントスレッドのコンテキストクラスローダを変更している場合、Webサービスアプリケーションが正常に動作しない可能性があります。カレントスレッドのコンテキストクラスローダは変更しないようにWebサービスアプリケーションを修正してください。

31.8 移行に関する注意事項

Interstage Application Serverの旧バージョン・レベルからの移行やサーバ環境を移行する場合、旧バージョン・レベルとのJ2EE機能の差異だけでなく、移行前後のJavaのバージョン、サーバ性能、負荷等の違いによりアプリケーションの挙動が影響を受ける場合があります。運用に入る前に十分に試験を行い、チューニングを行ってください。

31.9 SOAPサービスの移行

Interstage Application Server 8.0から、J2EEの機能の1つとして、新たなWebサービス運用環境であるInterstage Webサービスが提供されました。SOAPサービスのアプリケーションは、J2EEのInterstage Webサービスへ移行することができます。SOAPサービスから、J2EEのInterstage Webサービスに移行する場合は、以下を参照してください。

- J2EEのWebサービス機能への移行

31.9.1 J2EEのWebサービス機能への移行

ここでは、J2EEのWebサービス機能への移行について説明します。

- 機能の確認
- アプリケーションの移行
- 定義の移行・アプリケーションの管理
- インストール

機能の確認

以下の機能は、J2EEのWebサービス機能では提供されていません。J2EEのWebサービス機能へ移行する場合は、アプリケーションにて相当する機能を実現するか、これらの機能を利用しないように変更してください。

- CORBA/SOAPゲートウェイ機能
- 高信頼性Webサービス(SOAPメッセージの電子署名、XML暗号、SOAPメッセージに対するユーザ認証、送達保証機能)
- Messaging方式のアプリケーション
- V5.0以前のSOAPサービスAPI(パッケージ名が com.fujitsu.interstage.soap. で始まるクラス)
- クライアントパッケージ環境でのSSL機能
- SOAP1.2

アプリケーションの移行

以下の機能は、J2EEのWebサービス機能では提供されていない、または、差異があります。これらの機能を利用している場合は、アプリケーションの修正が必要です。

- 一部のデータ型(Enumerationなど)

- SOAPサービス固有API(名前が com.fujitsu.interstage.soapx. で始まるクラス、またはプロパティ)
- “Application”以外のサーバアプリケーションのライフサイクル
- クライアントアプリケーションの開発方式
- デフォルトのstyle/useなど
- メソッドのオーバーロード
- voidの返り値のダミー返却
- JavaパッケージとXMLの名前空間の対応
- Fault返却時のクライアントアプリケーションでの例外

一部のデータ型

サポートされないデータ型

以下のデータ型は、J2EEのWebサービス機能ではサポートされません。

- 列挙型
- java.util.Vector型
- com.fujitsu.interstage.soapx.typesパッケージに含まれるクラス

SOAPサービスでこれらのデータ型を利用していた場合は、「第18章 Webサービスの開発」を参照して、これらのデータ型を利用しないインタフェースに変更してください。

マッピングに変更があるデータ型

以下のデータ型は、XMLのデータ型とのマッピングがJ2EEのWebサービス機能とSOAPサービスで差異があります。

Webサービスアプリケーションのパラメタに使用するJavaのデータ型	XMLで使用されるデータ型	
	SOAPサービス	J2EEのWebサービス機能
byte[]	xsd:byteの配列	xsd:base64Binary
java.math.BigDecimal	xsd:decimal	soapenc:decimal (注)
java.math.BigInteger	xsd:integer	soapenc:integer (注)

注) encoded利用の場合のデータ型です。

SOAPサービスでこれらのデータ型を利用していた場合は、「第18章 Webサービスの開発」を参照して、アプリケーションまたはインタフェースを変更してください。

構造体型とBean型の混成とみなされるデータ型の注意

構造体型のpublicフィールドと、フィールド名と同名のプロパティのsetメソッドまたはgetメソッドの片方のみを持つデータ型は、J2EEのWebサービス機能とSOAPサービスで扱いに差異があります。以下について確認し、該当する場合は必要に応じてデータ型を修正してください。

- フィールド名と同名のプロパティのsetメソッドのみを持ち、getメソッドを持たない場合は、J2EEのWebサービス機能では該当のフィールド(プロパティ)は値取得不可のプロパティとみなされて、値は送信されません。
- フィールド名と同名のプロパティのgetメソッドのみを持ち、setメソッドを持たない場合は、J2EEのWebサービス機能では該当のフィールド(プロパティ)は値設定不可のプロパティとみなされて、データを受信した場合でも値が設定されません。

例

下記のデータ型を使用した場合、J2EEのWebサービス機能ではnameの値は送信されません。

```
public class Human {
   public String name;                          //nameのpublicフィールド
   public void setName(String n) { name = n; }  //nameのsetメソッド
   ///// ※getName()メソッドなし //////
   public Human() { }
   public Human(String n, int a){ name=n; age=a; }
 }
```

添付ファイル型

添付ファイル型のデータを使用している場合、以下について確認し対応を行ってください。

- SOAPサービスの添付ファイル機能は、WS-I Attachemets Profileに対応していません。移行前のSOAPサービスと同様の通信方法にするには、添付ファイルに対応するXMLのデータ型にはapachesoap名前空間のXMLデータ型を使用してください。添付ファイルに対応するXMLのデータ型の種類は、iswsgen wsdlコマンドの-attachmentsTypeオプションで指定します。詳細については、「リファレンスマニュアル(コマンド編)」の「iswsgen」を参照してください。
- SOAPサーバアプリケーションでは、受信した添付ファイルのパラメタをリクエスト処理完了後も保持することが可能でした。Interstage Webサービスの既定では、リクエスト処理完了後は、添付ファイルのパラメタはリソース解放のため利用できなくなる場合があります。リクエスト処理完了後もパラメタのまま保持する必要がある場合は、「レスポンス返却時の、Webサービスアプリケーションで受信した添付ファイルデータ削除(資源解放)」の設定を変更してください。詳細については、「19.3 Webサービス設定ファイル」を参照してください。

SOAPサービス固有API

プログラム修正	機能	SOAPサービス	J2EEのWebサービス機能
推奨	クライアントでの、プロキシを経由した接続の設定	名前がcom.fujitsu.interstage.soapx.proxyで始まるプロパティをプログラムで設定します。	「第19章 Webサービスの運用」を参照してください。(SOAPサービスでの設定方法は無視されます)
推奨	クライアントでの、接続タイムアウトの設定	com.fujitsu.interstage.soapx.socket.timeoutプロパティをプログラムで設定します。デフォルトは5分です。	「第18章 Webサービスの開発」を参照してください。デフォルトは10分です。(SOAPサービスでの設定方法は無視されます)
必須(該当する場合)	クライアントでの、セション継続の設定	com.fujitsu.interstage.soapx.client.Stub#setMaintainSession()メソッドで設定します。	「第18章 Webサービスの開発」を参照してください。(SOAPサービスでの設定メソッドは提供されません)
必須(該当する場合)	JAX-RPC規定(javax.xml.rpc.holdersパッケージ)以外のHolderクラス	com.fujitsu.interstage.soapx.holdersパッケージでHolderクラスが提供されています。	javax.xml.rpc.holders.Holderインタフェースをimplementsして、アプリケーションで作成してください。

“Application”以外のサーバアプリケーションのライフサイクル

プログラム修正	機能	SOAPサービス	J2EEのWebサービス機能
必須 (該当する場合)	サーバアプリケーションのライフサイクルの設定	Webサービス情報に設定します。“Application”、“Session”、“Request”からサーバアプリケーションのインスタンスの管理方式を選択します。	SOAPサービスにおいてApplicationが指定された場合と同等になります。リクエスト単位でサーバアプリケーションのインスタンス化が必要な場合、アプリケーションをラップする新たなアプリケーションクラスを作成してインスタンス生成と委譲するなどの対処が必要です。セションを利用する場合、アプリケーションのフィールドではなく、HttpSessionオブジェクトに情報を格納します。詳細については、「第18章 Webサービスの開発」を参照してください。 (SOAPサービスでの設定方法は提供されません)

クライアントアプリケーションの開発方式

SOAPサービスでは、統合開発環境ツール(IDE)を利用したスタブ方式と上級者向けの複雑なAPIを使用するDII方式の2つの方式がサポートされています。J2EEのWebサービス機能では、開発容易性に優れたスタブ方式をInterstage Application Serverとしてサポートしています。SOAPサービスでDII方式のクライアントアプリケーションを利用していた場合は、SOAPのクライアント部分をスタブ方式に修正してください。
また、スタブはJ2EEのWebサービス機能で生成したものに置き換える必要があります。

プログラム修正	機能	SOAPサービス	J2EEのWebサービス機能
必須	ロケータオブジェクト(Serviceオブジェクト)の取得	javax.xml.rpc.ServiceFactory#createService(javax.xml.namespace.QName serviceName)メソッドを使用します。	javax.xml.rpc.ServiceFactory#loadService(java.lang.Class generatedServiceClass)メソッドを使用します。

デフォルトのstyle/useなど

SOAPサービスのRPCアプリケーションでは、通信はWSDLのrpc/encoded相当の方式で行われます。J2EEのWebサービス機能は、rpc/encoded、rpc/literal、document/literalのWSDLをサポートしており、デフォルトはdocument/literalです。
またJ2EEのWebサービス機能ではJavaパッケージと名前空間名の対応、ユーザ定義型のJavaクラス名とXMLデータ型のローカル名の対応、および、オペレーションのパラメタの名前にデフォルトを持っています。デフォルトと移行前の値が異なる場合はiswsgenコマンドに対する指定が必要です。オペレーションのパラメタの名前(rpc/encodedのWSDLのpart名)は、WSDLの該当部分を書き換えてください。
詳細については、「リファレンスマニュアル(コマンド編)」の「iswsgen」を参照してください。

メソッドのオーバーロード

J2EEのWebサービス機能では、Webサービスアプリケーションのメソッドのオーバーロード(オペレーション名が同一で引数が異なる複数のメソッド)をサポートしていません。SOAPサービスでメソッドのオーバーロードを利用していた場合は、アプリケーションをラップするなどして、メソッドのオーバーロードを使用しないインタフェースに変更してください。

voidの返り値のダミー返却

SOAPサービスでは、返り値がvoidのアプリケーションについて、プロパティ指定により返り値としてダミーの値が返却するオプションが提供されていました。SOAPサービスで本オプションを使用していた場合は、アプリケーションをラップするなどして、返り値をvoidからintに変更して、ダミーの値0を常に返却するように変更してください。

JavaパッケージとXMLの名前空間の対応

J2EEのWebサービス機能では、Webサービスアプリケーションで使用されるJavaのパッケージとXMLの名前空間が対応付けられます。
移行前の名前空間の利用を継続するには、「リファレンスマニュアル(コマンド編)」の「iswsgen」を参照し、オプションで名前空間名を指定してください。Javaのパッケージと名前空間が1対1に対応していない場合は、既存のクラスをラップするクラスを作成するなどして1対1に対応させてください。

Fault返却時のクライアントアプリケーションでの例外

SOAPサービスでは、WebサービスからFaultが返却されると、クライアントアプリケーションにはFault情報を取得できるAPIを持つjavax.xml.rpc.soap.SOAPFaultExceptionがthrowされました。
それに対して、J2EEのWebサービス機能では、通常、その様なAPIを持たないjava.rmi.RemoteExceptionがthrowされます。同例外では、出力されるスタックトレースにFault情報が含まれています。
クライアントアプリケーションでは、両方の例外をcatchしてログにスタックトレース・Fault情報を出力するようにしてください。

定義の移行・アプリケーションの管理

アプリケーションの形態、配備方法

SOAPサービスでは、サーバアプリケーションは、任意のJARファイルにまとめるなどして、FTPなどでサーバに転送し、IJServer(Servletコンテナ)のクラスパスに設定する必要があります。また、あらかじめIJServerにはSOAPサービスの環境を構築しておく必要があります。
J2EEのWebサービス機能では、WebサービスアプリケーションはWARファイルに含め、通常のWARファイルと同様にInterstage管理コンソールなどからIJServerに配備することで利用可能になります。IJServerに事前の特別の環境構築は不要です。WebサービスアプリケーションのWARファイルの作成については、「第18章 Webサービスの開発」を参照してください。

アプリケーションの定義情報

SOAPサービスでは、Webサービス情報として、アプリケーションの定義情報が管理されており、Webサービス情報編集ツール(GUI)またはsoapmodifyddコマンドと記述ファイルで環境に登録・更新を行います。
J2EEのWebサービス機能では、Webサービス情報に相当する情報をdeployment descriptorとして記述し、アプリケーションのWARファイルに含めます。deployment descriptorの記述およびWARファイルの作成については、「第18章 Webサービスの開発」を参照してください。

インストール

Interstage V8.0、V9.0では、J2EEはInterstage Application Serverのインストールで、標準でインストールされていましたが、Interstage 10.0以降では、標準ではインストールされません。インストーラのメニューから「J2EE互換」を明示的に選択してインストールしてください。

31.10 ワークユニットの移行

ここでは、ワークユニットの移行について、以下を説明します。

31.10.1 予兆監視について

予兆監視について、以下を説明します。

- 予兆監視機能の警告メッセージについて

本製品のV11.1.0以前の環境から移行する場合に参照してください。

予兆監視機能の警告メッセージについて

本製品のV11.1.1から、不要なメッセージの対処を削減するため、予兆監視機能の警告メッセージがデフォルトでイベントログ/システムログに出力されなくなりました。イベントログ/システムログに予兆監視警告メッセージ(EXTP4368)を出力するには、「3.3.11 予兆監視」を参照して、イベントログ/システムログへの出力を有効にします。

第32章　V5.1以前のServletサービス環境定義の移行

本バージョン・レベルでは、V5.1以前のServletサービスを同梱していません。「30.1 Servletサービス(Tomcat5.5ベースのサーブレット実行環境)への移行」を参照して、移行してください。
ここでは、V5.1以前のServletサービスの環境定義をTomcat5.5ベースのServletサービスへ移行する際の定義項目の対応を示します。

Fujitsu XMLプロセッサを使用している場合は、Xerces2へ移行してください。設定方法については、「2.3.5 XMLパーサの設定」を参照してください。

32.1 JServlet環境定義の移行

以下にV5.1以前のServletサービスにおけるJServlet環境定義ファイルの定義項目と、Interstage管理コンソールの対応表を示します。なお、isj2eeadminコマンドを使用して、以下の各設定を行うこともできます。詳細は、「リファレンスマニュアル(コマンド編)」を参照してください。

V5.1以前のServletサービス	Interstage管理コンソール
[containername].containerconf	必要ありません。
[containername].bin	ありません。Tomcat5.5ベースのServletサービスでは、JDK8のみサポートします。
[containername].bin.parameters	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [JavaVMオプション] (注1)(注2)
[containername].env	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [クラスパス]、[パス]および[環境変数] (注1)(注3)
[containername].port	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [ポート番号] (注1)
[containername].log	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [ログ出力ディレクトリ] (注1) [ワークユニット] > 「ワークユニット名」 > [ログ参照]タブ > [起動情報]から参照できます。
[containername].startup	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [ワークユニット自動起動] (注1)
[containername].ha.shutdown	ありません。
[containername].execdir	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [カレントディレクトリ] (注1)
[containername].ipaddress	(注4) [ワークユニット] > [新規作成]タブ > [詳細設定] >[Servletコンテナ設定] > [IPアドレス] (注1)
default.bin default.bin.parameters default.env default.execdir	ありません。
container.checkCount	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [リトライカウント] (注1)

V5.1以前のServletサービス	Interstage管理コンソール
container.checkFrequency	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [リトライカウントリセット時間] (注1)
container.restart	trueの場合:[リトライカウント]を「0」または「2」以上、[リトライカウントリセット時間]を「1」以上に設定してください。 falseの場合:[リトライカウント]を「1」、[リトライカウントリセット時間]を「0」に設定してください。
container.shutdownInterval	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [プロセス強制停止時間] (注1)
container.uid	IJServerワークユニットを起動したユーザの権限でコンテナのプロセスを起動します。 特定ユーザの権限でIJServerワークユニットを自動起動する場合には、以下に示すInterstage管理コンソールの画面で起動ユーザを設定します。 [ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [ワークユニット自動起動] (注1)
container.gid	IJServerワークユニットを起動したユーザのプライマリグループになります。

注1)

[ワークユニット] > [新規作成]タブ > [詳細設定]による操作は、[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [詳細設定]によっても操作できます。

注2)

以下のパラメタは指定する必要はありません。

- -Dcom.fujitsu.interstage.jservlet.j2ee=true
- -Dorg.omg.CORBA.ORBClass=com.fujitsu.ObjectDirector.CORBA.ORB
- -Dorg.omg.CORBA.ORBSingletonClass=com.fujitsu.ObjectDirector.CORBA.SingletonORB
- -Djavax.rmi.CORBA.StubClass=com.fujitsu.ObjectDirector.rmi.CORBA.StubDelegateImpl
- -Djavax.rmi.CORBA.UtilClass=com.fujitsu.ObjectDirector.rmi.CORBA.UtilDelegateImpl
- -Djavax.rmi.CORBA.PortableRemoteObjectClass=com.fujitsu.ObjectDirector.rmi.CORBA.PortableRemoteObjectDelegateImpl

注3)

Servletコンテナ(IJServer)自身の実行に必要なパスは自動的に設定されるため、V5.1以前のServletサービスで環境変数PATH、CLASSPATH、LD_LIBRARY_PATHに設定していた以下の内容は指定する必要はありません。

Windows32/64

- JDKのインストールディレクトリ¥lib¥tools.jar
- C:¥Interstage¥F3FMjs2¥classes¥jsboot.jar
- C:¥Interstage¥F3FMjs2¥classes¥servlet.jar
- C:¥Interstage¥J2EE¥bin
- C:¥Interstage¥ODWIN¥etc¥class¥ODjava2.jar ・・・JDK1.3の場合
- C:¥Interstage¥ODWIN¥etc¥class¥ODjava4.jar ・・・JDK1.4の場合
- C:¥Interstage¥jms¥bin
- C:¥Interstage¥eswin¥lib¥esnotifyjava2.jar ・・・JDK1.3の場合

― C:¥Interstage¥eswin¥lib¥esnotifyjava4.jar ・・・JDK1.4の場合

― C:¥Interstage¥jms¥lib¥fjmsprovider.jar

― オラクル社のホームページからダウンロードしたFile System Service Provider(providerutil.jarおよびfscontext.jar)

― C:¥Interstage¥bin

Solaris64

― JDKのインストールディレクトリ/lib/tools.jar

― /opt/FJSVjs2/classes/jsboot.jar

― /opt/FJSVjs2/classes/servlet.jar

― /opt/FJSVj2ee/bin

― /opt/FSUNod/etc/class/ODjava2.jar ・・・JDK1.3の場合

― /opt/FSUNod/etc/class/ODjava4.jar ・・・JDK1.4の場合

― /opt/FSUNod/lib

― /opt/FJSVjms/bin

― /opt/FJSVes/lib/esnotifyjava2.jar ・・・JDK1.3の場合

― /opt/FJSVes/lib/esnotifyjava4.jar ・・・JDK1.4の場合

― /opt/FJSVjms/lib/fjmsprovider.jar

― /opt/FJSVjms/lib

― オラクル社のホームページからダウンロードしたFile System Service Provider(providerutil.jarおよびfscontext.jar)

Linux32/64

― JDKのインストールディレクトリ/lib/tools.jar

― /opt/FJSVjs2/classes/jsboot.jar

― /opt/FJSVjs2/classes/servlet.jar

― /opt/FJSVj2ee/bin

― /opt/FJSVod/etc/class/ODjava2.jar ・・・JDK1.3の場合

― /opt/FJSVod/etc/class/ODjava4.jar ・・・JDK1.4の場合

― /opt/FJSVod/lib

― /opt/FJSVjms/bin

― /opt/FJSVes/lib/esnotifyjava2.jar ・・・JDK1.3の場合

― /opt/FJSVes/lib/esnotifyjava4.jar ・・・JDK1.4の場合

― /opt/FJSVjms/lib/fjmsprovider.jar

― /opt/FJSVjms/lib

― オラクル社のホームページからダウンロードしたFile System Service Provider(providerutil.jarおよびfscontext.jar)

アプリケーションで必要なクラスについては、「2.3.4 IJServerで使用するクラスの設定について」を参照してください。

注4)

WebサーバコネクタとServletコンテナが別システム([システム]＞[環境設定]タブ＞[Servletサービスの詳細設定]＞[Webサーバとワークユニットを同一のマシンで運用する]で[運用しない]を選択)の場合に設定が必要であり、同一システムの場合は必要ありません。

32.2 サーブレット・ゲートウェイ環境定義の移行

以下にV5.1以前のServletサービスにおけるサーブレット・ゲートウェイ環境定義ファイルの定義項目と、Interstage管理コンソールの対応表を示します。なお、isj2eeadminコマンドを使用して、以下の各設定を行うこともできます。詳細は、「リファレンスマニュアル(コマンド編)」を参照してください。

32.2.1 移行元の環境で使用していたWebサーバがInterstage HTTP Serverの場合

サーブレット・ゲートウェイ	Interstage管理コンソール
ApJServDefaultPort	ありません。
ApJServMount	(注2) [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブ > [ServletコンテナのIPアドレス:ポート番号]および[Webアプリケーション名] (注1)
ApJServLogFile	[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [ログ設定]タブ > [ログ出力ディレクトリ] [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [ログ参照]タブから参照できます。
ApJServLogLevel	[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [ログ設定]タブ > [デバッグ情報]
ApJServRetryAttempts	ありません。
ApJServBalance ApJServHost	複数のServletコンテナを使用する場合は、「1つのWebアプリケーションを複数のServletコンテナで対応する」または「複数のWebアプリケーションを複数のServletコンテナで対応する」を参照してください。
ApJServDefinitionRead	常に有効です。
ApJServSessionRecovery	ありません。

注1)

[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブによる操作は、[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > 「ワークユニット名」によっても操作できます。

注2)

WebサーバコネクタとServletコンテナが別システム([システム] > [環境設定]タブ > [Servletサービス詳細設定] > [Webサーバとワークユニットを同一のマシンで運用する]で[運用しない]を選択)の場合に設定が必要であり、同一システムの場合は必要ありません。

1つのWebアプリケーションを複数のServletコンテナで対応する

Webアプリケーションを配備したIJServerワークユニットの以下の値を変更することで、複数のServletコンテナでWebアプリケーションを実行できるようになります。これにより、1つのServletコンテナに通信が集中することなく、プロセス多重度で指定した多重度数のServletコンテナに負荷を分散させることができます。

・ [ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [プロセス多重度]
または
[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [ワークユニット設定] > [プロセス多重度]

複数のWebアプリケーションを複数のServletコンテナで対応する

複数のIJServerワークユニットを作成し、Webアプリケーションを業務単位に異なるIJServerワークユニットに配備します。IJServerワークユニットは、IJServerワークユニット単位に起動、停止できるため、業務運用中であっても他業務へ影響を与えることなく、該当する業務のWebアプリケーションの変更・保守(復旧)作業を行うことができます。

WebサーバコネクタとServletコンテナが別システムの定義例

WebサーバコネクタとServletコンテナが別システムの定義例は、「3.6.3 IJServerとWebサーバを分離して運用する場合の手順」を参照してください。

32.2.2 移行元の環境で使用していたWebサーバがInterstage HTTP Server以外の場合

サーブレット・ゲートウェイ	Interstage管理コンソール
DefaultPort	ありません。
Timeout	(注2) [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブ > [詳細設定] > [送受信タイムアウト] (注1)
LogFile	[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [ログ設定]タブ > [ログ出力ディレクトリ] [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [ログ参照]タブから参照できます。
Mount	(注2) [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブ > [Webアプリケーション名] (注1)
Container	(注2) [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブ > [ServletコンテナのIPアドレス:ポート番号] (注1) 複数のServletコンテナを使用する場合は、「1つのWebアプリケーションを複数のServletコンテナで対応する」または「複数のWebアプリケーションを複数のServletコンテナで対応する」を参照してください。
DefaultHost	ありません。
ErrorPage	ありません。(注3)
MaxConnection	(注2) [Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブ > [詳細設定] > [Servletコンテナへの最大接続数] (注1)
DefinitionRead	常に有効です。
SessionRecovery	ありません。

注1)

[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > [新規作成]タブによる操作は、[Webサーバ] > 「Webサーバ名」 > [Webサーバコネクタ] > 「ワークユニット名」によっても操作できます。

注2)

WebサーバコネクタとServletコンテナが別システム([システム] > [環境設定]タブ > [Servletサービス詳細設定] > [Webサーバとワークユニットを同一のマシンで運用する]で[運用しない]を選択)の場合に設定が必要であり、同一システムの場合は必要ありません。

注3)

本バージョン・レベルでは、Webサーバコネクタ(サーブレット・ゲートウェイ)のErrorPageをWebサーバのエラーページ機能を使って設定します。

- Interstage HTTP Serverの場合

  Webサーバの環境定義ファイル(httpd.conf)のErrorDocumentディレクティブで設定します。詳細については、「Interstage HTTP Server 運用ガイド」－「ディレクティブ一覧」－「ErrorDocument」を参照してください。

- Internet Information Servicesの場合

  インターネット サービス マネージャを使用して、ISAPIフィルタ・ISAPIエクステンションを設定した、Webサイトおよび仮想ディレクトリのカスタムエラーの設定をします。詳細については、Internet Information Servicesのインターネット サービス マネージャのマニュアルを参照してください。

運用パターンの定義例は、「1つのWebアプリケーションを複数のServletコンテナで対応する」および「複数のWebアプリケーションを複数のServletコンテナで対応する」を参照してください。

WebサーバコネクタとServletコンテナが別システムの定義例

WebサーバコネクタとServletコンテナが別システムの定義例は、「3.6.3 IJServerとWebサーバを分離して運用する場合の手順」を参照してください。

32.3 サーブレット・コンテナ環境定義の移行

以下にV5.1以前のServletサービスにおけるサーブレット・コンテナ環境定義ファイルの定義項目と、Interstage管理コンソールの対応表を示します。なお、isj2eeadminコマンドを使用して、以下の各設定を行うこともできます。詳細は、「リファレンスマニュアル(コマンド編)」を参照してください。

－:該当項目なし

V5.1以前のServletサービス		Interstage管理コンソール
タグ	属性	
Server	－	必要ありません。
Logger	name	必要ありません。 [ワークユニット] > 「ワークユニット名」 > [ログ参照]タブ > [コンテナログ]で参照できます。
	path	[ワークユニット] > [新規作成]タブ > [詳細設定] > [ワークユニット設定] > [ログ出力ディレクトリ] (注1)

V5.1以前のServletサービス		Interstage管理コンソール
タグ	属性	
		[ワークユニット] > 「ワークユニット名」 > [ログ参照]タブ > [コンテナログ]から参照できます。
	backup	[ワークユニット] > 「ワークユニット名」 > [ログ定義]タブ > [ログファイルのロールオーバ]
	size	
	timestamp	必要ありません。
ContextManager	workDir	変更できません。以下のディレクトリになります。 Windows32/64 J2EE共通ディレクトリ¥ijserver¥IJServerワークユニット名¥work Solaris64 Linux32/64 J2EE共通ディレクトリ/ijserver/IJServerワークユニット名/work (注3)
	clientSession	指定できません。
	version	指定できません。
	errResponse	指定できません。
	sessionRecovery	[ワークユニット] > [新規作成]タブ > [詳細設定] > [セションリカバリ設定] (注1) あわせて、「第9章 セションリカバリ機能」を参照してください。
	name	ありません。
	sessionIdPrefix	ありません。
ContextInterceptor	classname	必要ありません。
RequestInterceptor	classname	必要ありません。
Connector	classname	必要ありません。
Parameter	handler	必要ありません。
	port	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [ポート番号] (注1)
	ipaddress	(注4) [ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [ServletコンテナのIPアドレス] (注1)
	acsaddress	(注4) [ワークユニット] > [新規作成]タブ > [詳細設定] > [Webサーバコネクタ(コネクタ)設定] > [要求を受付けるWebサーバのIPアドレス] (注1)
	min_spare_threads	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [同時処理数]の「初期値」 (注1)
	max_spare_threads	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [同時処理数]の「待機中の最大値」 (注1)
	max_threads	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [同時処理数]の「最大値」 (注1)
	connection_timeout	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [タイムアウト] (注1)
Context	path	必要ありません。
	docBase	[ワークユニット] > 「ワークユニット名」 > [配備]タブ > [配備設定]で、[サーバ上の任意の位置で実行するWebアプリケーションを配備す

V5.1以前のServletサービス		Interstage管理コンソール
タグ	属性	
		る]を選択で指定して配備。 上記以外で配備の場合は以下のディレクトリになります。 Windows32/64 J2EE共通ディレクトリ¥ijserver¥IJServerワークユニット名¥webapps¥Webアプリケーション名 Solaris64 Linux32/64 J2EE共通ディレクトリ/ijserver/IJServerワークユニット名/webapps/Webアプリケーション名 (注3)
	reloadable	クラスのオートリロード: [ワークユニット] > [新規作成]タブ > [詳細設定] > [共通定義] > [オートリロード] (注1) JSPのリロード:[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [JSPのリロード] (注1)
	inputcode	[ワークユニット] > 「ワークユニット名」 > [配備]タブ > [Webアプリケーション設定] > [エンコーディング] (注2) [ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [リクエストボディ処理のエンコーディングをクエリパラメタに使用] (注1)
	jsAuthentication	[ワークユニット] > 「ワークユニット名」 > [配備]タブ > [Webアプリケーション設定] > [認証] (注2)
	dirList	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [ファイルの一覧表示] (注1)
	tagPooling	[ワークユニット] > [新規作成]タブ > [詳細設定] > [Servletコンテナ設定] > [カスタムタグプーリングの使用] (注1)
	clientSession	[ワークユニット] > 「ワークユニット名」 > [配備]タブ > [Webアプリケーション設定] > [セション] (注2)
	default	[クッキーに設定する]を選択
	disable	[クッキーに設定しない]を選択
	permanent	[Webブラウザでセションを保存する]をチェック ([クッキーに設定する]を選択要)
	urlEncoding	必要ありません。
APITracer	ー	「トラブルシューティング集」の「Javaツール機能」ー「メソッドトレース機能」を参照してください。
TagPooling	ー	必要ありません。

注1)

[ワークユニット] > [新規作成]タブ > [詳細設定]による操作は、[ワークユニット] > 「ワークユニット名」 > [環境設定]タブ > [詳細設定]によっても操作できます。

注2)

[ワークユニット] > 「ワークユニット名」 > [配備]タブによる操作は、[ワークユニット] > 「ワークユニット名」 > 「Webアプリケーションのモジュール名」 > [環境設定]タブ >によっても操作できます。

注3)

J2EE共通ディレクトリのデフォルトは、Windows(R)ではC:¥Interstage¥J2EE¥var¥deployment、Solaris/Linuxでは/opt/FJSVj2ee/var/deploymentです。

注4)

WebサーバコネクタとServletコンテナが別システム([システム] > [環境設定]タブ > [Servletサービスの詳細設定] > [Webサーバとワークユニットを同一のマシンで運用する]で[運用しない]を選択)の場合に設定が必要であり、同一システムの場合は必要ありません。

32.4 JServletプロパティファイルの移行

以下にV5.1以前のServletサービスにおけるJServletプロパティファイルの定義項目と、Interstage管理コンソールの対応表を示します。なお、isj2eeadminコマンドを使用して、以下の各設定を行うこともできます。詳細は、「リファレンスマニュアル(コマンド編)」を参照してください。

V5.1以前のServletサービス	Interstage管理コンソール
com.fujitsu.interstage.jservlet.sessionrecovery.mode	ありません。
com.fujitsu.interstage.jservlet.session.cookie.secure.mode	[ワークユニット] > 「ワークユニット名」 > [配備]タブ > [Webアプリケーション設定] > [セション] > [クッキーにSecure属性を常に付加する] または、 [ワークユニット] > 「ワークユニット名」 > 「Webアプリケーションのモジュール名」 > [環境設定]タブ > [セション] > [クッキーにSecure属性を常に付加する] V5.1以前のServletサービスで「auto」を指定していた場合は指定する必要はありません。
com.fujitsu.interstage.jservlet.dbcs	必要ありません。

付録A　JDK/JREとFJVM

JDK/JREおよびFJVMを含むJava VMの詳細は、“チューニングガイド”の“JDK/JRE 8のチューニング”を参照してください。

付録B　Oracle Real Application Clustersとの連携

本製品では、Oracleのオプションであるオラクル Real Application Clusters(以降Oracle RAC)との連携をサポートしています。

環境設定

Oracle RACと連携する場合、以下の手順で設定してください。

- Oracle側の設定
- Interstage側の設定

Oracle RACと連携する場合の注意事項は、以下を参照してください。

- Oracle RACと連携の注意事項

注意

分散トランザクション機能(JTS)は、Oracle RACとの連携をサポートしていません。

B.1　Oracle側の設定

Oracle RACが動作可能な環境を作成するために、Oracle側の設定を行ってください。
Oracle側の設定は、ロードバランスの使用有無で異なります。

- ロードバランスを使用しない場合
- ロードバランスを使用する場合

ロードバランスを使用しない場合

アプリケーション(JDBCデータソース)ごとにデフォルトの接続先ノードを決めておき、故障が発生した場合だけに他のノードに接続します。Oracleのサーバ側のlistener.oraに、以下の例のように設定してください。

例

Oracleサーバ1のlistener.ora の例

```
LISTENER =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = smpl)
      (SID_NAME = smpl1)
      (ORACLE_HOME = /opt/app/oracle/product/9.2.0)
    )
  )
```

ociドライバを使用する場合

ociドライバを使用する場合は、Oracleクライアント(Interstage)側のtnsnames.oraに、以下の例のように設定してください。thinドライバを使用する場合は不要です。

例

クライアント(Interstage)側のtnsnames.oraの例

```
SAMPLE.WORLD =
  (DESCRIPTION = (ENABLE = BROKEN)
    (FAILOVER = ON)
    (ADDRESS = (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = server2)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = server3)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = smpl))
  )
```

ロードバランスを使用する場合

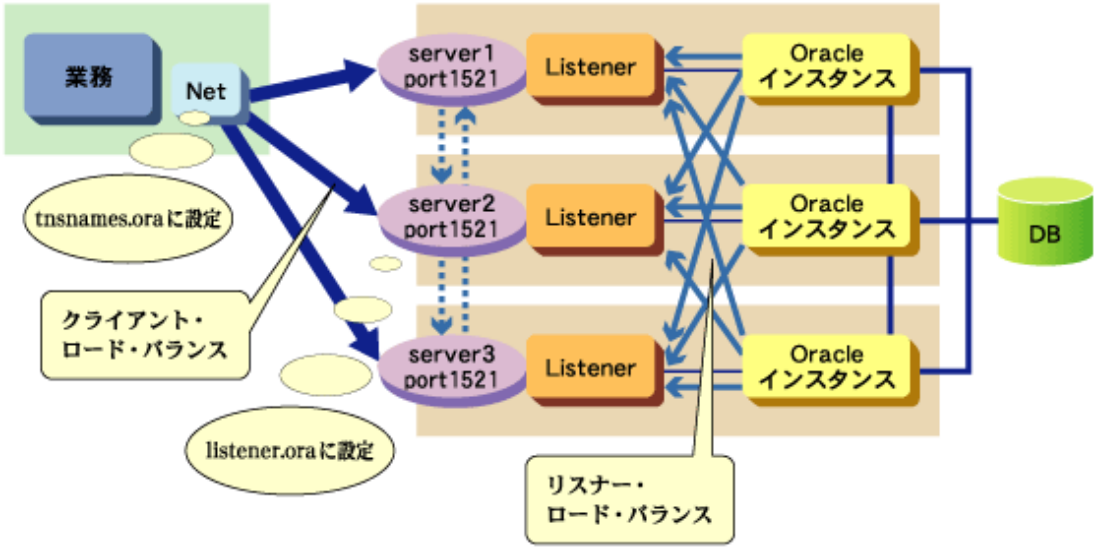
ロードバランス機能により、通常運用時から負荷分散を行います。ロードバランスには、“クライアント・ロード・バランス”と“リスナー・ロード・バランス”があります。

クライアント・ロード・バランス

アプリケーション側から、Oracleのリスナーへの振り分けを行います。

リスナー・ロード・バランス

リスナーから、Oracleインスタンスへの振り分けを行います。



例

“クライアント・ロード・バランス”および“リスナー・ロード・バランス”を使用した設定例を示します。以下の例のように設定してください。

Oracleサーバ1の初期化パラメータの例
注) 以下の例では関係する部分を抜粋しています。

local_listener = "(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(Host = server1)(Port = 1521)))"
remote_listener = "(DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(Host = server2)(Port = 1521)) (ADDRESS = (PROTOCOL = TCP)(Host = server3)(Port = 1521)))"

Oracleサーバ1のlistener.ora の例

```
LISTENER =
  (ADDRESS_LIST =
```

```
    (ADDRESS = (PROTOCOL = TCP)(Host = server1)(Port = 1521))
  )
```

ociドライバを使用する場合

ociドライバを使用する場合は、Oracleクライアント(Interstage)側のtnsnames.oraに、以下の例のように設定してください。thinドライバを使用する場合は不要です。

例

クライアント(Interstage)側のtnsnames.oraの例

```
SAMPLE.WORLD =
  (DESCRIPTION = (ENABLE = BROKEN)
    (LOAD_BALANCE = ON)
    (FAILOVER = ON)
    (ADDRESS = (PROTOCOL = TCP)(HOST = server1)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = server2)(PORT = 1521))
    (ADDRESS = (PROTOCOL = TCP)(HOST = server3)(PORT = 1521))
    (CONNECT_DATA = (SERVICE_NAME = smpl))
  )
```

B.2 Interstage側の設定

thinドライバを使用する場合と、ociドライバを使用する場合で設定方法が異なります。

設定内容

thinドライバを使用する場合

Interstage管理コンソールの[リソース]>[JDBC]>[新規作成]画面より、“RACを使用する”を“使用する”に設定してJDBCデータソースを登録してください。もしくは、isj2eeadminコマンドのresourceサブコマンドでJDBCデータソースを同様に登録してください。
サーバURLにOracle RAC用のサーバURLを指定してください。

例

サーバURLの指定例は以下のとおりです。

```
jdbc:oracle:thin:@(DESCRIPTION=
    (ENABLE=BROKEN)
    (ADDRESS_LIST=
        (FAILOVER=ON)
        (LOAD_BALANCE=ON)
        (ADDRESS=
            (PROTOCOL=tcp)
            (HOST=host1)
            (PORT=1521))
        (ADDRESS=
            (PROTOCOL=tcp)
            (HOST=host2)
            (PORT=1521)))
    (CONNECT_DATA=
        (SERVICE_NAME=service_name)))
```

ociドライバを使用する場合

Interstage管理コンソールの[リソース]>[JDBC]>[新規作成]画面より、“RACを使用する”を“使用する”に設定してJDBCデータソースを登録してください。もしくは、isj2eeadminコマンドのresourceサブコマンドでJDBCデータソースを同様に登

録してください。
サーバURLにOracle RAC用のサーバURLを指定してください。

例

サーバURLの指定例は以下のとおりです。

```
jdbc:oracle:oci:@(DESCRIPTION=
    (ENABLE=BROKEN)
    (ADDRESS=
        (PROTOCOL=tcp)
        (HOST=cluster_alias)
        (PORT=1521))
    (CONNECT_DATA=
        (SERVICE_NAME=service_name)))
```

サーバURLの設定方法

Interstage管理コンソールを使用した場合、データソース定義時の[サーバURL]の項目にある[テンプレート作成]ボタンを押下すると、thinドライバもしくはociドライバに対応した雛型が出力されますので、以下の表に示されている文字列の斜体部分を、使用する環境に合わせて適切な値に変更してください。
下記以外のサーバURLの設定方法については、Oracleのマニュアルを参照してください。

項目	内容
FAILOVER=*ON*	フェイルオーバを使用しない場合はOFFとしてください。 入力例: FAILOVER=OFF
LOAD_BALANCE=*ON*	ロードバランスを使用しない場合はOFFとしてください。 入力例: LOAD_BALANCE=OFF
PROTOCOL=*tcp*	thinドライバの場合はtcpのみサポートされています。ociドライバでipcプロトコルを使用する場合はipcを指定してください。 入力例: PROTOCOL=ipc
HOST=*host* または HOST=*cluster_alias*	使用する環境に合わせてDBサーバのホスト名(ドライバがociの場合はクラスタ名)を入力してください。 入力例: HOST=shop001
PORT=*1521*	Oracleのポート番号が1521以外の場合はこの値を変更してください。通常は変更する必要はありません。
SERVICE_NAME=*service_name*	使用する環境に合わせて、サービス名の設定を行ってください。ociドライバを選択時は、tnsnames.oraファイルに登録したネットサービス名を設定してください。 入力例: SERVICE_NAME=banking

例

サーバURLの設定例

以下にサーバURLの設定例を提示します。

OracleDB1を運用系DBサーバとし、OracleDB2を待機系DBサーバとしてフェイルオーバ機能のみを利用する場合

```
jdbc:oracle:thin:@(DESCRIPTION=
    (ENABLE=BROKEN)
    (ADDRESS_LIST=
        (FAILOVER=ON)
        (LOAD_BALANCE=OFF)
```

```
        (ADDRESS=
            (PROTOCOL=tcp)(HOST=OracleDB1)(PORT=1521))
        (ADDRESS=
            (PROTOCOL=tcp)(HOST=OracleDB2)(PORT=1521)))
    (CONNECT_DATA=
        (SERVICE_NAME=service_name)))
```

OracleDB1、OracleDB2、OracleDB3でロードバランスを行う場合

```
jdbc:oracle:thin:@(DESCRIPTION=
    (ENABLE=BROKEN)
    (ADDRESS_LIST=
        (FAILOVER=ON)
        (LOAD_BALANCE=ON)
        (ADDRESS=
            (PROTOCOL=tcp)(HOST=OracleDB1)(PORT=1521))
        (ADDRESS=
            (PROTOCOL=tcp)(HOST=OracleDB2)(PORT=1521))
        (ADDRESS=
            (PROTOCOL=tcp)(HOST=OracleDB3)(PORT=1521)))
    (CONNECT_DATA=
        (SERVICE_NAME=service_name)))
```

注意

- ociドライバを選択した場合は、Oracle側のlistener.oraおよびクライアント(Interstage)側のtnsnames.oraを編集する必要があります。詳細はOracleのマニュアルを参照してください。
- [サーバURL]のテキストエリア内での改行は、データソース作成時に無視されます。

その他の設定

Oracle10g以降のRAC環境においてRACを使用するデータソース定義で、データソースの種類を“Oracleのコネクションプーリングを使用する”に設定した場合、高速接続フェイルオーバー機能を利用されます。データソースの種類が“Interstageのコネクションプーリングを使用する”の場合、この機能は利用されません。高速接続フェイルオーバー機能の詳細についてはOracleのマニュアルを参照してください。
RACを使用するデータソース定義において、データソースの種類を“Oracleのコネクションプーリングを使用する”に設定した場合、ons.jarをクラスパスに設定する必要があります。ons.jarは以下のOracleをインストールしたディレクトリ(ORACLE_HOME)配下に格納されています。

- ORACLE_HOME¥opmn¥lib¥ons.jar

Interstageのインストールされているサーバの ORACLE_HOME(Oracleをインストールしたディレクトリ)の値を、以下の設定値としてJavaVMオプションに設定してください。

- -Doracle.ons.oraclehome=ORACLE_HOME

JavaVMオプションはInterstage管理コンソールよりシステムの環境設定にあるJ2EEプロパティおよびIJServerの環境設定に設定が可能です。
この環境変数をシステムとIJServerの双方の項目に設定した場合、IJServerに設定した値が有効になります。またJ2EEプロパティに上記の設定行わない場合、正しくDB接続テストが行えません。

定義項目名	定義内容
JavaVMオプション	DB接続テスト機能使用時およびIJServer運用時にJavaコマンドに指定するオプションを設定します。4096バイト以内の文字列を指定します。

定義項目名	定義内容
	・ 1つのオプションに空白が含まれる場合、""(ダブルクォーテーション)で囲んで指定します。 ・ 複数のオプションを指定する場合、空白を区切り文字としてオプションを続けて記載してください。その場合、全体を""(ダブルクォーテーション)で囲むと、全体が1つのオプションとみなされます。""は空白を含むオプションを指定する場合のみ使用してください。 ・ 改行文字を含むことはできません。

またOracleのマニュアルを参照してInterstage側のOracle Notification Serviceを構成しONSデーモンを起動してください。

B.3 Oracle RACと連携の注意事項

- データベースサーバがフェイルオーバした場合について
  Oracle RACまたはサーバがダウンしフェイルオーバした場合は、Oracle RACの機能として縮退運転し業務を継続します。
  その際、ダウンしたサーバへのコネクションはすべて継続運用しているOracleへ振り分けられます。
  この時、継続運用している各サーバへ通常時以上のコネクションが確立され、コネクションがプーリングされます。そのため、ダウンしたサーバが復旧しても必要数のコネクションがすでにプーリングされているため、復旧したサーバへの振り分けが行われない場合があります。
  その場合は、コネクションを再度確立し直すために、ijstune jdbcコマンドによるコネクションの破棄を実行するもしくはIJServerの再起動を行ってください。

- Interstage側の設定について
  Interstageの自動再接続機能は、“使用する”を設定してください。デフォルトは“使用する”です。詳細は、“27.2 IJServerのチューニング”を参照してください。

- Java言語以外のアプリケーションの対応
  Interstageには特別な定義は必要ありません。Oracle RAC側だけの設定になります。

- 分散トランザクション機能(JTS)について
  分散トランザクション機能(JTS)は、Oracle RACとの連携をサポートしていません。

- Interstage管理コンソールの設定について

  ー 記述したサーバURLの構文チェックは、データソース作成時の[DB接続テスト]、もしくは環境設定画面にある[DB接続テスト]ボタンの押下で実施してください。
  ただしロードバランス・フェイルオーバの機能が設定されている場合、アドレスリストのDBサーバにアクセスしコネクションを取得するため、定義されているDBサーバが1つでも正しく接続することができればエラーにはならないためすべてのホスト名の確認にはなりません。

  ー [Oracleのコネクションプーリングを使用する]を選択してデータソースを作成した場合、Interstage管理コンソールの[ワークユニット] > [環境設定] > [DBコネクション設定]の異常時の再接続は有効になりません。

  ー Oracle Notification Serviceの設定に不備がある場合、IJServerの起動時にONSからエラーがコンテナログに出力されますが、IJServerの起動には成功します。Oracleのマニュアルおよび“メッセージ集”の“J2EE使用時に出力される例外情報”を参照し、設定の見直し後IJServerの再起動を行ってください。

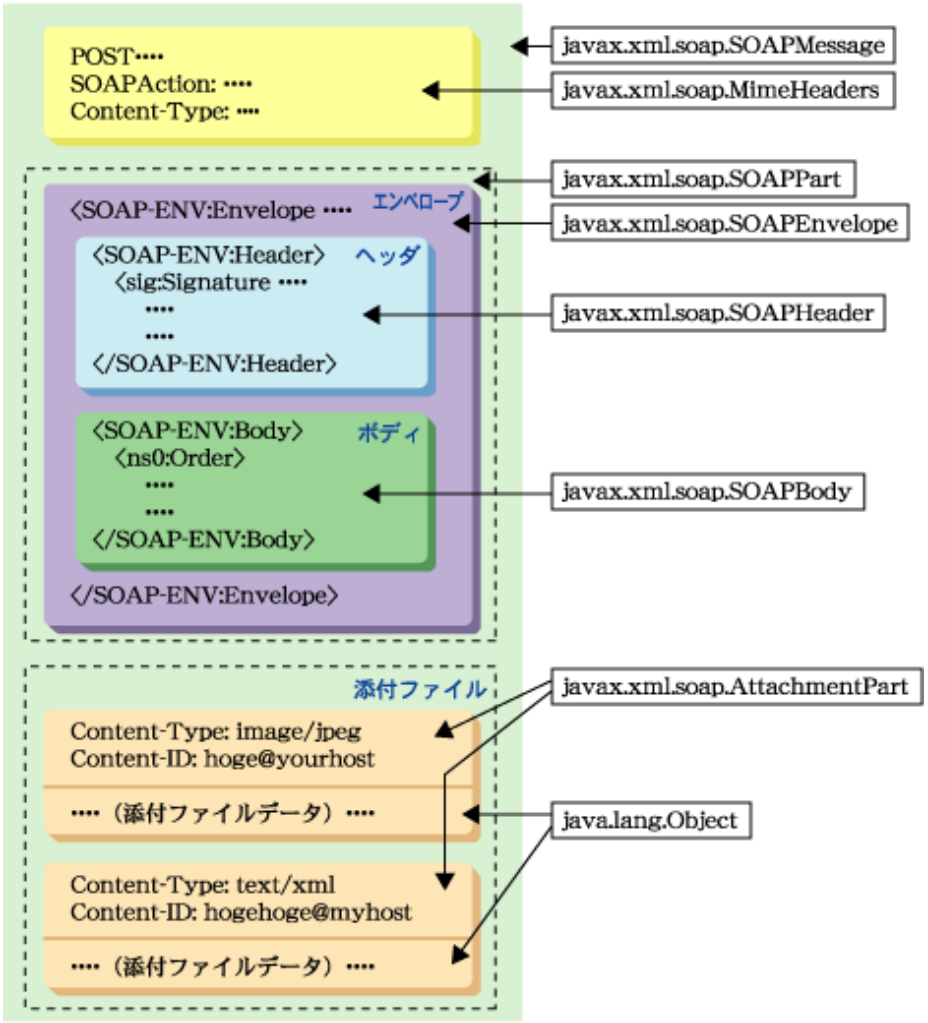
付録C　SOAPメッセージの低レベル処理

ここでは、SOAPメッセージに直接アクセスする高度なAPIを利用した、SOAPメッセージの低レベル処理について説明します。

注意

- SOAPエンベロープのデータサイズが大きい場合、受信時に大量にメモリを消費し、メモリ不足や処理時間超過を招く恐れがあります。
- 目安として、通信データサイズが数100Kバイト(XMLでのタグなどを含む)を超える場合、添付ファイルを使用することを検討してください。
- ただし、通信データサイズが同じでもメモリ消費量は内容形式により大きく異なり、また、同時処理多重度や性能要件もシステムにより異なります。上記サイズを目安としつつ、条件に応じて、実際の業務用のデータ内容形式を使用して要件に対する事前検証を行うことをお勧めします。

C.1　SOAPメッセージの構造



添付ファイルを含むSOAPメッセージは、上図のような構造になっています。SAAJ-APIでは、SOAPメッセージの各要素が、上図の右に示すJavaクラスにマップされています。

C.2 SOAPメッセージの作成

SOAPエンベロープの構成要素を新規作成する場合はSOAPFactoryオブジェクトを使用します。
SOAPFactoryオブジェクトはSOAPFactoryクラスのnewInstanceクラスメソッドを使用して取得します。

```
import javax.xml.soap.*;
    .....
  SOAPFactory sf = SOAPFactory.newInstance();
  SOAPElement el1 = sf.createElement(localName, prefix, nsuri);
  Name name = sf.createName(localName2, prefix, nsuri);
    .....
```

SOAPMessageオブジェクトを新規作成する場合はMessageFactoryオブジェクトを使用します。
MessageFactoryオブジェクトはMessageFactoryクラスのnewInstanceクラスメソッドを使用して取得します。

```
import javax.xml.soap.*;
    .....
  MessageFactory mf = MessageFactory.newInstance();
  SOAPMessage msg = mf.createMessage();
  SOAPPart part = msg.getSOAPPart();
  SOAPEnvelope env = part.getEnvelope();
    .....
```

C.3 SOAPエンベロープの処理

SOAPEnvelopeオブジェクトは、SOAPエンベロープを表すオブジェクトです。1つのSOAPメッセージには、必ず1つのSOAPエンベロープが含まれます。
SOAPEnvelopeオブジェクトは、SOAPPart.getEnvelopeメソッドで取得できます。SOAPヘッダーやSOAPボディを直接操作する場合は、SOAPEnvelopeオブジェクトのAPIを使用します。SOAPエンベロープには0または1つのSOAPヘッダーと1つのSOAPボディが含まれています。

```
import javax.xml.soap.*;
    .....
  MessageFactory mf = MessageFactory.newInstance();
  SOAPMessage msg = mf.createMessage();
  SOAPPart part = msg.getSOAPPart();
  SOAPEnvelope env = part.getEnvelope();

// SOAPHeaderを使用しない場合の処理
  SOAPHeader header = env.getHeader();
  header.detachNode();

// SOAPHeaderを設定する場合の処理
// SOAPHeader header = env.getHeader();
// SOAPHeaderElement helm = header.addHeaderElement(
//       env.createName( "Header1", "ns1", "urn:Sample" ));
// helm.addNamespaceDeclaration("ns1", "urn:Sample");

// SOAPBodyの設定処理
  SOAPBody body = env.getBody();
  SOAPBodyElement belm = body.addBodyElement(
           env.createName( "Body1", "ns1", "urn:Sample" ));
  belm.addNamespaceDeclaration("ns1", "urn:Sample");
    .....
```

SOAPヘッダーを使用しない場合はSOAPHeader.detachNodeメソッドを明示的に呼び出すか、または何もしません。何もしない場合は、空のヘッダー(<xxx:Header/>)が出力されます。

SOAPHeader, SOAPHeaderElement, SOAPBody, SOAPBodyElementの各インタフェースは、SOAPElementインタフェースを継承していますので、そのままSOAPElementインタフェースがもつAPIを使用して子要素の設定・取得などの操作が行えます。

注意

- SOAPEnvelopeオブジェクトおよびその配下で、必要な名前空間宣言を行っていない場合、送信・返信時に、自動的に必要な名前空間宣言が送信・返信されるSOAPエンベロープのデータに追加されます。

SOAPエンベロープを処理するにはSOAPEnvelopeインタフェースのAPIを使用してSOAPHeaderやSOAPBodyのオブジェクトを直接操作することもできますが、XMLデータを使用してXMLオブジェクトをそのままSOAPエンベロープに設定することや、SOAPエンベロープの内容をXMLデータとして取り出すことができます。そのような場合はSOAPPartオブジェクトのsetContent/getContentメソッドを使用します。

```
import javax.xml.soap.*;
import javax.xml.transform.Source;
import javax.xml.transform.sax.SAXSource;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamSource;
    .....
  MessageFactory mf = MessageFactory.newInstance();
  SOAPMessage msg = mf.createMessage();
  SOAPPart part = msg.getSOAPPart();

// SAXSourceオブジェクトを作成
  SAXSource source = .....;

// DOMSourceオブジェクトを作成
  DOMSource source = .....;

// StreamSourceオブジェクトを作成
  StreamSource source = .....;

// SourceオブジェクトからSOAPMessageを作成
  part.setContent(source);
    .....

// SOAPMessageをSourceオブジェクトとして取得
  Source source = part.getContent();
    .....
```

SAXSource, DOMSource, StreamSourceはそれぞれSourceインタフェースを実装したクラスです。Sourceインタフェースを実装したオブジェクトはSOAPPart.setContentメソッドでSOAPエンベロープとして設定できます。同様にSOAPエンベロープの内容をSOAPPartのgetContentメソッドでSourceオブジェクトとして取得できます。具体的な使用方法はサンプルを参照してください。

C.4 Faultの処理

SOAPメッセージを送信する送信アプリケーションは、サーバシステムからの復帰情報としてFaultメッセージを受信する場合があります。SAAJ-APIではFaultはSOAPFaultオブジェクトとして表されます。SOAPFaultオブジェクトはSOAPMessageオブジェクト中のSOAPBodyオブジェクトの子要素として設定されます。
Faultを起こすアプリケーションはSOAPBody.addFaultメソッドでSOAPFaultオブジェクトを生成し、Faultを受けるアプリケーションはSOAPBody.getFaultメソッドでSOAPFaultオブジェクトを取り出すことができます。また送信アプリケーションはSOAPBody.hasFaultメソッドで受信したSOAPMessageオブジェクト中にFaultがあるかどうかを判定できます。

SOAPFaultオブジェクトには以下の情報が保持され、アプリケーションではそれらの情報を設定・取得できます。
Faultコード／Fault説明／Faultアクタの操作をする場合は、必ずSOAPFaultクラスのset系メソッド(setFaultXXXXX)を使用してください。SOAPElement.addTextNodeでは、情報を設定できません。

- Faultコード(java.lang.String)
- Fault説明(java.lang.String)

・ Faultアクタ(java.lang.String)

・ Fault詳細(javax.xml.soap.Detail)

```
import javax.xml.soap.*;
import java.util.Iterator;
    ....
  MessageFactory mf = MessageFactory.newInstance();
  SOAPMessage msg = mf.createMessage();
  SOAPPart part = msg.getSOAPPart();
  SOAPEnvelope env = part.getEnvelope();
  env.getHeader().detachNode();
  SOAPBody body = env.getBody();
    .....
// SOAPFaultを設定する場合
  SOAPFault fault = body.addFault();
  fault.setFaultActor(Faultアクタ);
  fault.setFaultCode(Faultコード);
  fault.setFaultString(Fault説明);
  Detail detail = fault.addDetail();
  DetailEntry entry = detail.addDetailEntry( ..... );
  entry.addChildElement(...);
    .....
  SOAPMessage reply = .....;
// Faultが発生したかを判定
  env = reply.getSOAPPart().getEnvelope();
  body = env.getBody();
  if( body.hasFault() ) {
// Fault発生
    fault = body.getFault();
    String faultActor = fault.getFaultActor();
    String faultCode = fault.getFaultCode();
    String faultString = fault.getFaultString();
    detail = fault.getDetail();
    Iterator it = detail.getDetailEntries();
    while( it.hasNext() ){
      entry = (DetailEntry)it.next();
        ......
    }
  }
  else{
  // 正常処理
      .....
  }
```

SOAPFaultインタフェースはSOAPBodyの子要素を表すSOAPBodyElementインタフェースを継承しています。SOAPFault, SOAPFaultElement, Detail, DetailEntryの各インタフェースは、SOAPElementインタフェースを継承していますので、通常のSOAPメッセージの組み立てと同じようにFault情報を組み立てることができます。

C.4.1 Fault情報の解析

受信アプリケーション(サーバシステム側のアプリケーション)の呼び出し処理で異常が発生した場合、送信アプリケーション(クライアントシステム側のアプリケーション)が受信する返信メッセージにFault情報が設定されます。受信した返信メッセージにFault情報が設定されているかどうかを判定するにはSOAPBody.hasFaultメソッドを使用します。このメソッドの返り値が真(true)の場合、返信メッセージにFault情報が設定されています。
Fault情報は、通常のSOAPメッセージと同様にSOAPボディの子要素として設定されます。Fault情報はSOAPFaultオブジェクトとして表され、SOAPBody.getFaultメソッドで取得できます。返信メッセージの受信からFault情報の取り出しは以下のようなプログラムコードになります。

```
import javax.xml.soap.*;
    .....
// レスポンスデータ受信
```

```
  SOAPMessage response = conn.call( msg, endPoint );
  SOAPEnvelope env = response.getSOAPPart().getEnvelope();
// SOAPBodyの取り出し
  SOAPBody body = env.getBody();

// Fault情報が設定されているかを判定
  if( body.hasFault()) {
    // Fault情報の取り出し
    SOAPFault fault = body.getFault();
    // Faultコードの取得
    String faultCode = fault.getFaultCode();
    // Fault説明の取得
    String faultString = fault.getFaultString();
    // Fault詳細の取得
    Detail detail = fault.getDetail();
    Iterator it = detail.getDetailEntries();
       .....
  }
  else{
// Fault情報は設定されていない(正常系)
       .....
  }
```

Fault情報には以下の情報が含まれています。

情報種別	説明
Faultコード	異常の分類です。以下の文字列が設定されています。 ・ Server ・ Client ・ VersionMismatch ・ MustUnderstand それぞれの意味については下記の“Faultコードの分類”を参照してください。
Fault説明	異常の内容を説明する文字列が設定されています。
Faultアクタ	エラーの発生した場所(受信アプリケーションなど)を表すURIです。
Fault詳細	Webサービスアプリケーション固有のエラー情報が設定されています。

Faultコードの取得

FaultコードはSOAPFaultオブジェクトの以下のメソッドを使用して取り出します。

```
  java.lang.String  getFaultCode()
```

Faultコードの分類

Faultコード	説明
Server	Webサービス・コンテナや受信アプリケーションの処理過程で問題を検出したことを示します。 例えば、Webサービス・コンテナの設定に誤りがある場合などに返ります。
Client	送信アプリケーションが送信したメッセージ内容に誤りを検出したことを示すコードです。 このFaultコードが送信アプリケーションに返された場合は、受信アプリケーションへ送信したメッセージを見直してください。
VersionMismatch	Webサービスが、受信したメッセージのSOAPのバージョンに対応していないことを示します。

Faultコード	説明
MustUnderstand	Webサービスが、受信したメッセージのSOAPヘッダの処理必須である子要素の処理に失敗したことを示します。
その他のFaultコード値	以下のような場合に上記以外のFaultコードが返されることがあります。 ・Webサービスが独自にFaultコードを設定している場合

Fault説明の取得

Fault説明はSOAPFaultオブジェクトの以下のメソッドを使用して取り出します。

```
java.lang.String  getFaultString()
```

Faultアクタの取得

FaultアクタはSOAPFaultオブジェクトの以下のメソッドを使用して取り出します。

```
java.lang.String  getFaultActor()
```

Fault詳細の取得

Fault詳細は、任意の数のFault詳細項目で構成されたWebサービス固有の情報項目です。
Fault詳細はSOAPFaultオブジェクトの以下のメソッドを使用して取り出します。

```
javax.xml.soap.Detail  getDetail()
```

またFault詳細に含まれる個々の詳細項目はjavax.xml.soap.DetailEntryオブジェクトとして表され、Detailオブジェクトの以下のメソッドを使用して取り出します。

```
java.util.Iterator  getDetailEntries()
```

それぞれのFault 詳細項目は、java.util.Iteratorオブジェクトにjavax.xml.soap.DetailEntryオブジェクトとして保持されています。Fault詳細項目の取り出し方は、以下のようなプログラムコードになります。

```
import javax.xml.soap.*;
    .....
  if( body.hasFault()) {
    SOAPFault fault = body.getFault();
    String faultCode = fault.getFaultCode();
    String faultString = fault.getFaultString();
  // Fault詳細の取得
    Detail detail = fault.getDetail();
  // Fault詳細項目の取得
    Iterator it = detail.getDetailEntries();
    while(it.hasNext()) {
      DetailEntry entry = (DetailEntry)it.next();
          .....
    }
  }
  else{
     .....
  }
```

javax.xml.soap.DetailEntryインタフェースはjavax.xml.soap.SOAPElementインタフェースを継承しています。よってFault詳細に含まれる個々の詳細項目は純粋なSOAPElementオブジェクトとして操作できます。

注意

SOAPElement.getChildElements(Name name)メソッドを使用する場合、パラメタのNameオブジェクトは、必ずプレフィックスとURIを指定して取得してください。

付録D　性能監視

性能監視ツールは、業務サーバ上で動作する以下のアプリケーションやコンテナの性能情報を採取します。

- IJServerのEJBコンテナ

注意

IJServerタイプのIJServer(Web ＋ EJB[1VM])およびIJServer(Webのみ)の性能情報は採取できません。
IJServer(Web + EJB[別VM])の場合、EJBアプリケーションが動作するJava VMの情報のみ採取できます。

性能監視ツールの機能

性能監視ツールは、以下の機能をサポートしています。

性能ログファイルへのログ出力機能

指定したオブジェクトの性能情報を、性能ログファイルに蓄積する機能です。蓄積した性能情報は、レポート出力コマンドを使用して、CSV形式で出力できます。

Systemwalker Centric Managerによる性能情報のリアルタイム監視機能(MIBによる監視) Windows32/64 Solaris64

Systemwalker Centric Managerなどの**ネットワーク管理マネージャ**の**MIB**監視機能を利用することにより、指定したオブジェクトの性能情報をリアルタイムに表示および監視できます。ネットワーク管理マネージャを使用して、性能情報を表示、監視することをリアルタイム監視と呼びます。

ネットワーク管理マネージャ

監視サーバ上で性能情報の表示および監視を行うソフトウェアです。
本章では、業務サーバ上でのコマンド操作およびネットワーク管理マネージャとしてSystemwalker Centric Managerを使用した場合の性能情報の表示操作について説明しています。

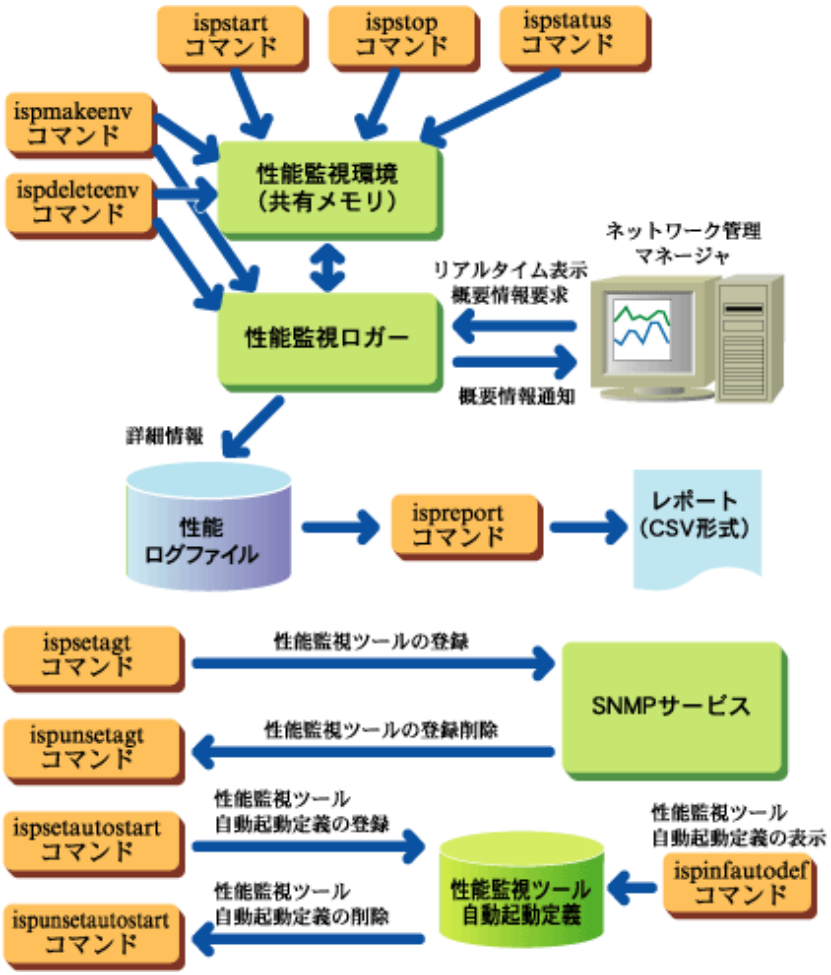
MIB

Management Information Baseの略です。システム情報、TCP/IP情報を管理するために、定義された管理情報領域のことです。

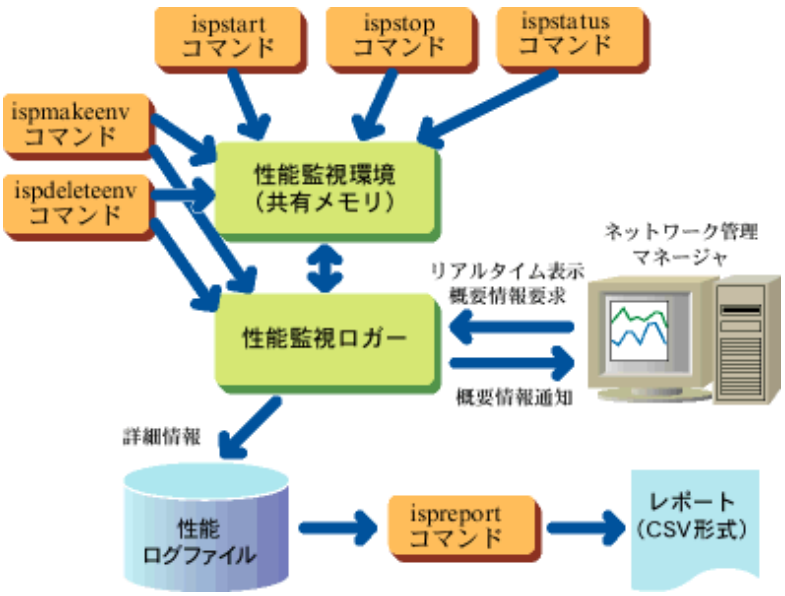
性能監視ツールの構成

性能監視ツールは、性能情報を採取する性能監視ロガーと各種コマンドにより構成されます。
性能監視ツールが提供するコマンドは、以下のような構成になります。

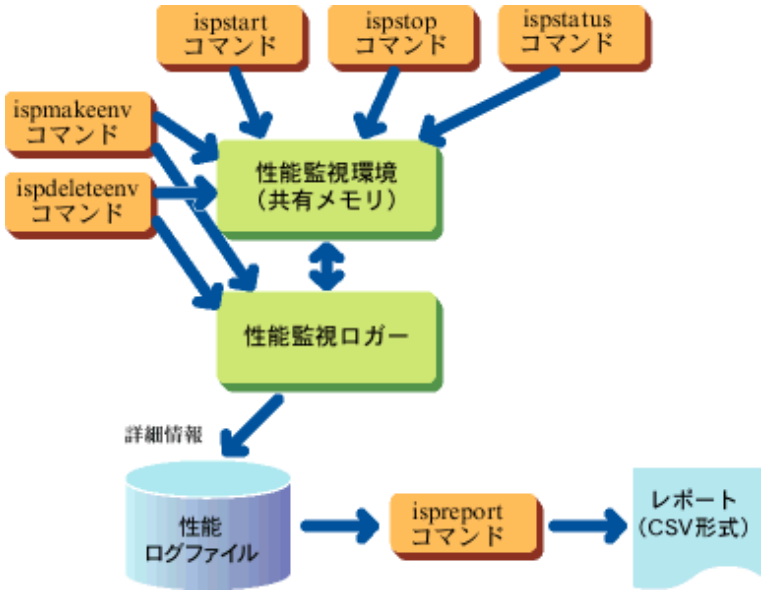
Windows32/64



Solaris64



Linux32/64



性能監視ロガー	性能情報の採取と性能ログファイルの作成、Systemwalker Centric Managerへの性能情報通知(Windows (R)、Solarisのみ)
ispmakeenvコマンド	性能監視環境の作成と性能監視ロガーの起動
ispdeleteenvコマンド	性能監視環境の削除と性能監視ロガーの停止
ispstartコマンド ispstopコマンド	性能監視の開始、停止
ispstatusコマンド	性能監視状態表示
ispreportコマンド	性能ログファイルのレポートを出力
Windows32/64 ispsetagtコマンド ispunsetagtコマンド	性能監視ツールのSNMPサービスへの登録、削除
Windows32/64 ispsetautostartコマンド ispunsetautostartコマンド	性能監視ツール自動起動定義の登録、削除
Windows32/64 ispinfautodefコマンド	性能監視ツール自動起動定義の表示

D.1 性能監視ツールの機能

性能監視ツールの機能について、以下に説明します。

D.1.1 性能ログファイルへのログ出力機能

性能ログファイルへのログ出力機能は、指定したアプリケーションの性能情報を、性能ログファイルに蓄積する機能です。性能情報は、性能監視環境の作成時に指定したインターバル時間の間隔を区切りに蓄積されます。
蓄積した性能情報は、レポート出力コマンドを実行することによりCSV形式で出力できますので、性能情報の分析や統計情報の蓄積に役立ちます。
採取できる情報や分析方法については、「D.3.1 性能ログファイルへのログ出力機能により採取した性能情報」を参照してください。

注意

性能監視対象のIJServerワークユニットを停止する場合は、通常停止してください。

性能監視対象のIJServerワークユニットを強制停止した場合、または"アプリケーション最大処理時間"を超過したためにInterstageによってIJServerワークユニットが強制停止された場合、IJServerワークユニットが停止しても性能ログファイルへのログ出力は停止しません。強制停止したIJServerワークユニットに対しては、ispstopコマンドを実行して性能監視を停止することで、ログ出力を停止することができます。

D.1.2 Systemwalker Centric Managerによる性能情報のリアルタイム監視機能(MIBによる監視) Windows32/64 Solaris64

本機能はWindows(R)版、Solaris版のみ使用可能です。なお、Solaris版はSolaris 10でのみ使用可能です。

リアルタイム監視機能は、指定したアプリケーションの性能情報を、MIB情報として通知する機能を提供します。
Systemwalker Centric Managerには、MIB監視機能がサポートされています。Systemwalker Centric Managerの機能を利用することにより、以下のような運用が可能となります。

- 統計情報のレポート出力
  性能情報のグラフ表示や、CSV形式でのファイル出力が行えます。統計情報の収集に役立ちます。
- 性能異常の監視
  処理待ち要求数などの性能情報に対してしきい値を設定し、監視することによって、性能情報の異常を、事前に察知できるようになります。これにより、異常事象に対して、迅速な対応が行えます。

参考

MIB情報

Management Information Baseの略です。システム情報、TCP/IP情報を管理するために、定義された管理情報領域のことです。

本機能を使用する場合、性能ログファイルへの情報出力も同時に行われますので、オペレーション単位の情報などの、よりきめ細かな性能分析を行いたい場合には、性能ログファイルに蓄積された情報を分析してください。
また、MIB情報を利用した監視方法などについては、Systemwalker Centric Managerのマニュアルを参照してください。

採取できる情報や分析方法については、「D.3.2 Systemwalker Centric Managerによるリアルタイム監視機能により採取した性能情報 」を参照してください。

D.1.3 他製品との連携による性能情報の分析 Windows32/64 Solaris64

本機能はWindows(R)版、Solaris版のみ使用可能です。

性能監視ツールを使用して採取した情報は、他製品を使用することにより、性能情報の解析をより詳細に行うことができます。

Systemwalker Service Quality Coordinatorとの連携

Systemwalker Service Quality Coordinatorは、サービス品質の最適化を支援する製品です。
サービス品質を可視化するとともに、システムを構成する個々のサーバやミドルウェアから収集した性能情報をさまざまな角度で監視・分析・評価する機能を提供します。
本製品と連携することにより、アプリケーション・サーバの性能情報とサーバマシンのCPU 負荷状況やメモリ使用状況等を比較や、相関を分析するなど、より高度な性能情報の解析が可能となります。
詳細は、「Systemwalker Service Quality Coordinator のユーザーズガイド」を参照してください。

Systemwalker PerfMGRとの連携

Systemwalker PerfMGRは、サーバマシン等の性能情報を統計的に解析するための製品です。
Systemwalker PerfMGRで、性能監視ツールが出力するCSVファイルを読み込むことにより、Systemwalker PerfMGRで、アプリケーションの性能情報の解析を行えるようになります。
具体的には、アプリケーションの性能情報と、マシンのCPU負荷状況やメモリ使用状況等と、グラフ表示等による比較が行えます。
マシンの負荷情報も加味した、より詳細な性能情報の解析が可能となります。
詳細は、「Systemwalker PerfMGRのマニュアル」を参照してください。

D.2 性能監視ツールの操作手順

性能監視ツールによりInterstage上の業務アプリケーションの性能を監視および分析するためには、SNMPサービスへの登録操作、性能監視ツール起動操作、監視操作、性能監視ツール停止操作、およびSNMPサービスからの削除操作を行います。以下に、操作の流れを示します。

1. **SNMPサービスへの登録操作** Windows32/64 Solaris64

   リアルタイム監視を行うために必要な操作で、性能監視ツールをSNMPサービスに登録します。
   Interstageのインストール時に行う操作です。
   詳細は、「D.2.1 SNMPサービスへの登録操作 (Windows (R)の場合) 」または「D.2.2 SNMPサービスへの登録操作 (Solarisの場合) 」を参照してください。

2. **性能監視ツール起動操作**

   性能監視ツールを起動する操作です。
   詳細は、「D.2.3 性能監視ツール起動操作」を参照してください。

3. **監視操作**

   性能情報の測定、監視、分析を行う操作です。
   詳細は、「D.2.4 監視操作」を参照してください。

4. **性能監視ツール停止操作**

   性能監視ツールを停止する操作です。
   詳細は、「D.2.5 性能監視ツール停止操作」を参照してください。

5. **SNMPサービスからの削除操作** Windows32/64 Solaris64

   性能監視ツールをSNMPサービスに登録している場合、性能監視ツールをSNMPサービスから削除します。
   Interstageのアンインストール時に行う操作です。
   詳細は、「D.2.6 SNMPサービスからの削除操作 」を参照してください。

D.2.1 SNMPサービスへの登録操作 (Windows (R)の場合) Windows32/64

Systemwalker Centric Managerを使用してリアルタイム監視を行うためには、Interstageのインストール後に以降の操作を行う必要があります。なお、リアルタイム監視を行わない場合は、以降の操作を行う必要はありません。

1) SNMPサービスへの登録

ispsetagtコマンドを実行して、性能監視ツールをSNMPサービスに登録してください。ispsetagtコマンド実行後は、Windowsの「サービス」画面よりSNMPサービスを再起動してください。なお、SNMPサービスがインストールされていなければ、性能監視ツールをSNMPサービスに登録することはできません。ispsetagtコマンドを実行する前に、SNMPサービスをインストールしてください。
Windows Server(R)でSNMPサービスのインストール方法を、以下に示します。

Windows Server(R) 2012以降の場合

1. 「サーバ マネージャ」の「機能」から「機能の追加」を開き、「SNMPサービス」を選択してインストールしてください。

2) MIB定義ファイルの読み込み

Systemwalker Centric Managerから性能情報を採取するには、性能情報のMIB定義ファイルをSystemwalker Centric Managerで読み込まなければなりません。Interstageをインストールしたマシン上にある、以下のMIB定義ファイルをSystemwalker Centric Managerに読み込ませてください。

```
TD_HOME¥isp¥mib¥ispmibNT.my
```

MIB定義ファイルの読み込み方法を、以下に示します。

1. Interstageをインストールしたマシン上にあるMIB定義ファイルを、Systemwalker Centric Managerがインストールされているマシンに、FTPなどを使ってコピーしてください。
2. Systemwalker Centric Managerのシステム監視を起動してください。
3. システム監視画面のメニューから、「操作」→「MIB拡張操作」の順に項目をクリックして、MIB拡張操作画面を表示してください。
4. MIB拡張操作画面で「MIB登録」ボタンをクリックして、拡張MIBファイル選択画面を表示してください。
5. 拡張MIBファイル選択画面でMIBファイルを選択して、「開く」ボタンをクリックしてください。その後、MIB拡張操作画面で「閉じる」ボタンをクリックしてください。

D.2.2 SNMPサービスへの登録操作 (Solarisの場合) Solaris64

Systemwalker Centric Managerを使用してリアルタイム監視を行うためには、Interstageのインストール後に以降の操作を行う必要があります。なお、リアルタイム監視を行わない場合は、以降の操作を行う必要はありません。なお、リアルタイム監視はSolaris 10でのみ使用可能です。

1) 性能監視ツールのコピー

Systemwalker Centric Manager上で、性能情報を表示するには、性能監視ツールの以下のファイルを、所定のディレクトリにコピーしてください。この操作は、Interstageインストール時に行ってください。また、操作完了後は、マシンを再起動してください。

ファイル名	コピー元ディレクトリ	コピー先ディレクトリ
ispsubad8	/opt/FSUNtd/isp/lib	/usr/lib/snmp
ispsuba8.acl	/etc/opt/FSUNtd/snmp/conf	/etc/snmp/conf
ispsuba8.reg	/etc/opt/FSUNtd/snmp/conf	/etc/snmp/conf
ispsuba8.rsrc	/etc/opt/FSUNtd/snmp/conf	/etc/snmp/conf

注意

- 性能監視ツールのリアルタイム監視機能を使用する場合は、必須ソフトウェアをインストールし、必須パッチを適用する必要があります。必須ソフトウェアおよび必須パッチの詳細については、「インストールガイド」を参照してください。
- SEAのマスターエージェントを再起動する場合は、性能監視ツールは必ず停止してください。
- SEAのマスターエージェント(snmpdx)は、ポート番号161を使用し、ポート番号は変更しないでください。SEAのマスターエージェントのポート番号を161以外に変更すると、以下のようなメッセージが出力され、性能情報のリアルタイム監視機能は使用できません。

```
/opt/FSUNtd/isp/lib/ispsubad8: [ID 702911 daemon.error] subagent registration failed
```

・ システム管理エージェント(SMA)を使用したリアルタイム監視機能は使用できません。
Solstice Enterprise Agentsソフトウェア(SEA)を使用してください。

・ Solaris 10でSolstice Enterprise Agentsソフトウェア(SEA)の使用するポート番号が161から変更になっています。従来通りのポート番号161を使用するようにしてください。
この場合、SMAがポート番号161を使用しているため、SMAを使用せずSEAのみを使用するようにする必要があります。

例

SEAのみを使用する場合の環境設定例

以下に、SMAを使用せず、SEAのみを使用する場合の環境設定例を記載します。
詳細は、OSのマニュアルを参照してください。

1. SMAの停止

```
# /etc/init.d/init.sma stop
```

2. SMAおよび関連サービスがブート時に自動的に起動しないように設定

   以下のサービスがブート時に自動的に起動しないように設定します。

   ー svc:/application/management/sma

   ー svc:/application/management/seaport

```
# svcadm disable svc:/application/management/sma
# svcadm disable svc:/application/management/seaport
```

   また、「svc:/application/management/snmpdx」は「svc:/application/management/seaport」と依存関係があるため、「svc:/application/management/seaport」を無効にした場合、「svc:/application/management/snmpdx」が起動されません。依存関係を解消してください。
   サービスの依存関係などの設定に関しては、OSのマニュアルを参照し実施してください。

3. SEAが使用するポート番号の設定を161に変更

```
/etc/snmp/conf/snmpdx.regファイルのport行を16161から161に変更
```

4. SEAの各設定ファイルを作成

```
# cp /etc/snmp/conf/mibiisa.rsrc- /etc/snmp/conf/mibiisa.rsrc
# cp /etc/snmp/conf/snmpdx.acl /etc/snmp/conf/mibiisa.acl
```

5. SEAの再起動

```
# /etc/init.d/init.snmpdx stop
# /etc/init.d/init.snmpdx start
```

2) MIB定義ファイルの読み込み

Systemwalker Centric Managerから性能情報を採取するには、性能情報のMIB定義ファイルをSystemwalker Centric Managerで読み込まなければなりません。Interstageをインストールしたマシン上にある、以下のMIB定義ファイルをSystemwalker Centric Managerに読み込ませてください。

```
TD_HOME¥isp¥mib¥ispmibSol.my
```

MIB定義ファイルの読み込み方法を、以下に示します。

1. Interstageをインストールしたマシン上にあるMIB定義ファイルを、Systemwalker Centric Managerがインストールされている業務管理クライアントに、FTPを使ってコピーしてください。

2. Systemwalker Centric Manager(業務監視)を起動してください。
3. 業務監視画面のメニューから、「操作」→「MIB拡張操作」の順に項目をクリックして、MIB拡張操作画面を表示してください。
4. MIB拡張操作画面で「MIB登録」ボタンをクリックして、拡張MIBファイル選択画面を表示してください。
5. 拡張MIBファイル選択画面でMIBファイルを選択して、「開く」ボタンをクリックしてください。その後、MIB拡張操作画面で「閉じる」ボタンをクリックしてください。
6. 業務監視画面のメニューから、「ポリシー」→「ポリシー配付」の順に項目をクリックして、ポリシーの配付画面を表示してください。
7. ポリシーの配付画面で必要な項目を設定した後、「OK」ボタンをクリックしてください。

3) ポート番号の設定

性能監視ツールは、Systemwalker Centric Managerに性能情報を通知する際に通信を行うため、通信用のポートを使用します。デフォルトでは「7042」番のポートを使用しますので、このポートを他のプログラムで使用している場合は、性能監視ツールのポート番号を変更してください。この操作は、Interstageインストール時に行ってください。また、操作完了後は、マシンを再起動してください。
変更方法は以下のようになります。

1. 環境設定ファイルをエディタで開いてください。

   環境設定ファイルは「/etc/snmp/conf/ispsuba8.reg」です。
   エディタで開くと以下のように表示されます。

```
agents =
  {
    {
      name = "ispsubad8"
          subtrees = { isPerformanceInf }
      timeout = 40000000
      watch-dog-time = 2000000
      port = 7042
    }
  }
```

2. 「port = 7042」の「7042」の部分を、未使用のポート番号に書き換えてください。
3. ファイルを保存して、エディタを終了してください。
4. マシンを再起動してください。

D.2.3 性能監視ツール起動操作

性能監視ツールの起動操作について説明します。

運用パターン

性能監視ツールの起動には、以下の方法があります。

自動運用 Windows32/64

Interstage起動前に性能監視ツール自動起動定義の登録を行うと、Interstage起動時に性能監視ツールを自動的に起動します。自動運用により性能監視ツールを起動した場合は、Interstage停止時にのみ性能監視ツールは停止します。また、性能監視自動起動定義ファイルに「Auto_start=YES」および性能監視対象アプリケーションを記述して自動起動の登録を行った場合は、Interstage起動時に性能監視ツールが自動的に起動し、性能監視対象アプリケーションの性能監視を開始します(自動監視開始機能)。性能監視自動起動定義ファイルに「Auto_start=NO」を記述して自動起動の登録を

行った場合は、Interstage起動時に性能監視ツールが自動的に起動しますが、性能監視は開始されません。自動監視開始機能を指定しない場合は、Interstage起動後に、ispstartコマンドを実行して性能監視を開始してください。

手動運用

ispmakeenvコマンドを実行して、性能監視ツールを起動します。性能監視対象のワークユニットを起動する前であれば、いつでも性能監視ツールを起動し性能監視を行うことが可能です。

注意

- Windows32/64
  手動運用を行った場合、Windowsをログオフすると、性能監視ツールは停止します。性能監視ツール起動後に、Windowsのログオフを行う場合は、自動運用により、性能監視ツールを起動してください。
- Solaris64 Linux32/64
  性能監視ツールは「手動運用」でのみ起動できます。

起動方法

性能監視ツール自動運用の場合 Windows32/64

Interstage起動時に性能監視ツールを起動する操作について説明します。

1. **性能監視ツール自動起動定義の作成**

   性能測定に必要な環境および性能監視を行うオブジェクトを指定する定義ファイルを作成します。性能監視ツール自動起動定義の詳細については、「D.5.2 性能監視自動起動定義ファイル(ispsetautostartコマンド)」を参照してください。

2. **性能監視ツール自動起動定義の登録**

   ispsetautostartコマンドで性能監視ツール自動起動定義を登録します。

3. **Interstageの起動**

   isstartコマンドを実行し、Interstageを起動します。Interstage起動時に自動的に性能監視ツールが起動します。

性能監視ツール手動運用の場合

1. **システム構成設定操作** Solaris64

   性能監視環境を作成し、性能監視ツールを起動するために、システム構成情報ファイル内の以下のシステム構成情報を調整する必要があります。また、操作完了後は、マシンを再起動してください。

   ```
   Semsys:seminfo_semmnu
   ```

   性能監視ツールを起動する際には、必ず上記システム構成情報の設定値を見積もってください。見積もった結果、設定値の変更が不要な場合は、上記のシステム構成情報を設定する必要はありません。設定値の見積もりについては「チューニングガイド」の「システム構成情報の見積もり方法」を参照してください。

2. **Interstageの起動**

   isstartコマンドを実行し、Interstageを起動します。

3. **性能監視環境の作成**

   ispmakeenvコマンドにより性能監視環境を作成し、性能監視ツールを起動します。この際、以下のインターバル時間を指定します。

   ー 性能ログファイル用インターバル時間
   性能ログファイルに対して性能情報を出力する間隔です。1分、5分、10分、20分、30分、1時間、2時間、3時間、4時間のいずれかが指定可能です。省略した場合には、1時間が設定されます。

ー リアルタイム監視用インターバル時間 Windows32/64 Solaris64
Systemwalker Centric Managerに通知する性能情報の採取間隔です。Systemwalker Centric Managerからの性能監視を行う場合に指定しています。1～60分が指定可能です。省略した場合には、5分が設定されます。

注意

- ispmakeenvコマンド実行後に、性能を測定する業務アプリケーション(ワークユニット)を起動してください。ispmakeenvコマンド実行前に起動された業務アプリケーションに対しては、性能は測定されません。
- Solaris64 Linux32/64
  システムを再起動した場合は、性能監視環境を再作成する必要があります。

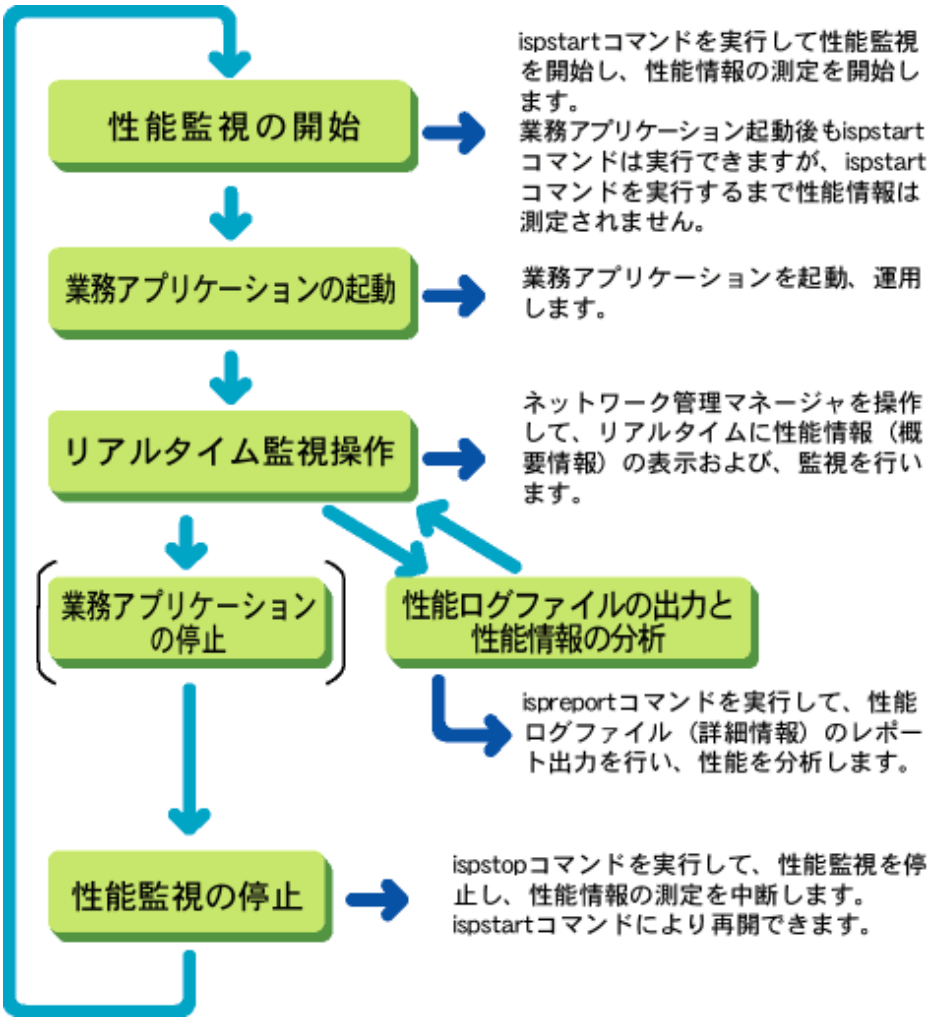
D.2.4 監視操作

性能監視ツール監視操作について説明します。

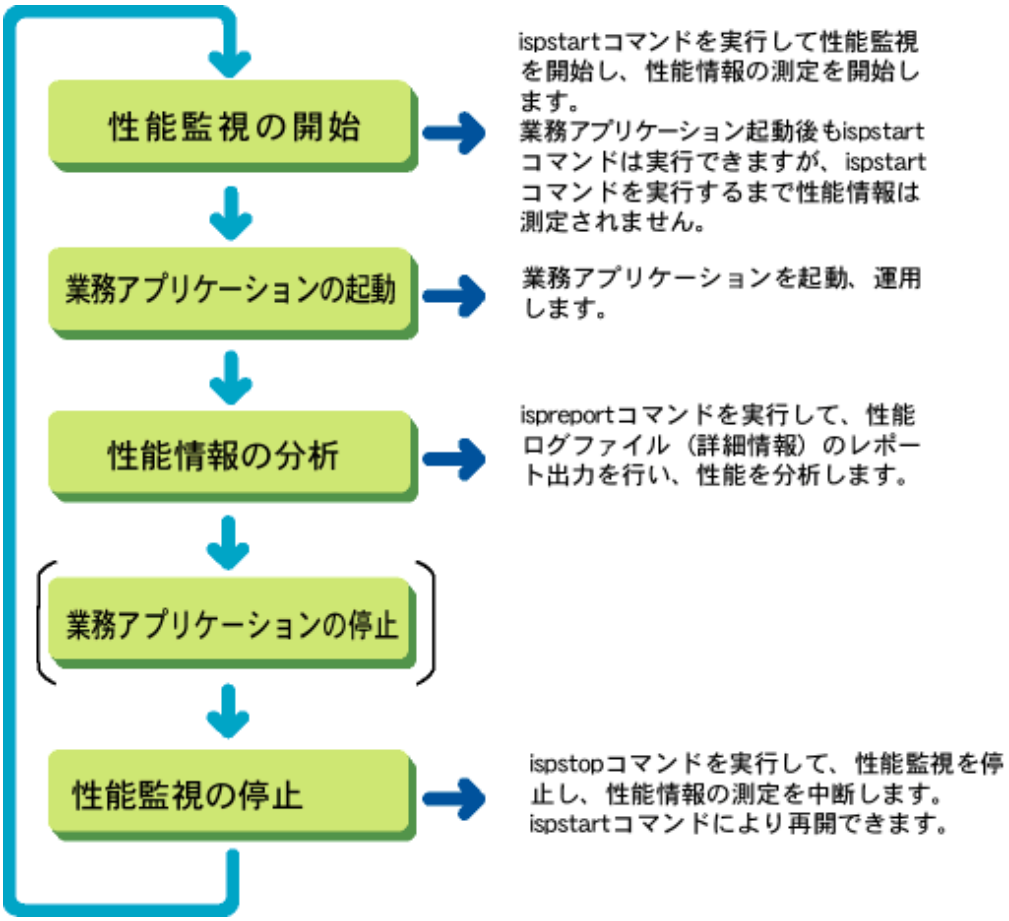
- 監視操作の流れ
- 監視操作の方法
- リアルタイム監視操作手順

監視操作の流れ

Windows32/64 Solaris64


性能監視の開始
ispstartコマンドを実行して性能監視を開始し、性能情報の測定を開始します。
業務アプリケーション起動後もispstartコマンドは実行できますが、ispstartコマンドを実行するまで性能情報は測定されません。
業務アプリケーションの起動
業務アプリケーションを起動、運用します。
リアルタイム監視操作
ネットワーク管理マネージャを操作して、リアルタイムに性能情報（概要情報）の表示および、監視を行います。
業務アプリケーションの停止
性能ログファイルの出力と性能情報の分析
ispreportコマンドを実行して、性能ログファイル（詳細情報）のレポート出力を行い、性能を分析します。
性能監視の停止
ispstopコマンドを実行して、性能監視を停止し、性能情報の測定を中断します。
ispstartコマンドにより再開できます。

Linux32/64



監視操作の方法

1. **性能監視の開始**

   ispstartコマンドにより特定のオブジェクトの性能監視を開始します。これ以降、性能監視ツール自動起動定義ファイル(Windows(R)のInterstage Application Server Enterprise Edition)またはispmakeenvコマンドで指定したインターバル間隔で、性能情報が性能ログファイルに出力されます。
   ispstopコマンドにより性能監視を停止するまで、性能情報が採取されます。

   Windows32/64
   性能監視ツール自動運用で性能監視を開始している場合は、ispstartコマンドを実行する必要はありません。

2. **業務アプリケーションの起動**

   isstartwuコマンドを実行し、ワークユニットを起動します。

3. **リアルタイム監視操作** Windows32/64 Solaris64

   Systemwalker Centric Managerで性能情報を表示し、性能を監視するには、Systemwalker Centric Manager上で性能情報を表示する操作を行います。
   詳細は、「リアルタイム監視操作手順」を参照してください。

4. **性能ログファイルの出力と性能情報の分析**

   ispreportコマンドを使用して性能ログファイルを出力し、性能情報を分析します。詳細は、「D.3.1 性能ログファイルへのログ出力機能により採取した性能情報」を参照してください。

   Windows32/64 Solaris64
   リアルタイム監視で基準値超えを検出し、性能異常の可能性がある場合は、性能ログファイルに保存されている詳細情報を分析してください。

5. **業務アプリケーションの停止**

   isstopwuコマンドを実行し、ワークユニットを停止します。

6. **性能監視の停止**

   ispstopコマンドにより性能監視を停止します。本コマンドが実行された時点で、性能情報の採取および性能ログファイルへの性能情報の出力が停止します。

注意

- 業務アプリケーションを停止しなくても、ispstopコマンドを実行して性能監視を停止することは可能です。ただし、ispstopコマンド実行後は、性能情報は測定されません。性能情報の測定を再開する場合は、ispstartコマンドを実行してください。

Windows32/64 Solaris64

- ispmakeenvコマンド実行後に、性能を測定する業務アプリケーション(ワークユニット)を起動してください。ispmakeenvコマンド実行前に起動された業務アプリケーションに対しては、性能は測定されません。
- Systemwalker Centric Managerを使って性能情報をリアルタイム表示している場合、性能情報を表示したまま、性能監視の停止/開始を行わないでください。性能監視開始前に性能情報を表示している画面を閉じ、性能監視開始後にリアルタイム監視操作を行って、性能情報を表示してください。

Windows32/64

- 性能監視ツール自動運用で性能監視を開始している場合は、ispstartコマンドを実行する必要はありません。
- 性能監視ツール自動運用で性能監視を開始したのち、監視対象アプリケーションを変更する場合は、ispstopコマンド実行後、ispstartコマンドを実行してください。

リアルタイム監視操作手順 Windows32/64 Solaris64

以下に、Systemwalker Centric Manager運用管理サーバを使用した場合の操作概要を示します。
Systemwalker Centric Manager以外のネットワーク管理マネージャを使用する場合は、使用するネットワーク管理マネージャの操作説明書を参照してください。

1) MIB取得画面の表示

Windows32/64

1. Systemwalker Centric Managerのシステム監視を起動します。
2. システム監視画面で、対象となる業務サーバ名をクリックします。
3. システム監視画面のメニューから、「操作」→「指定システム」→「MIB情報」→「MIB取得」の順に項目をクリックして、MIB取得画面を表示します。

Solaris64

1. Systemwalker Centric Manager(業務監視)を起動します。
2. 業務監視画面の「機能選択」に「監視」を設定し、ツリー選択に「ノード一覧」を設定します。業務サーバの一覧が表示されていますので、対象となる業務サーバ名をクリックします。
3. 業務監視画面のメニューから、「操作」→「指定オブジェクト」→「MIB情報」→「MIB取得」の順に項目をクリックして、MIB取得画面を表示します。

2) 監視するオブジェクトのインスタンス番号の確認

オブジェクト名の一覧を表示して、監視するオブジェクトのインスタンス番号を調べます。
MIB取得画面で以下の設定を行い、「取得」ボタンをクリックします。

a. 「ポーリングなし」を指定

b. 取得方法に「DUMP」を指定

c. 表示するMIB(性能情報の測定項目)にispSumObjectNameを指定

なお、MIBは以下の手順で指定します。

1. 「参照」ボタンをクリックして、MIB参照画面を表示します。
2. MIB参照画面で表示されている項目を「iso」→「org」→「dod」→「internet」→「private」→「enterprises」→「fujitsu」→「 application 」→「 aplNetWork 」→「 aplNetFunction 」→「 aplInterstage 」→「 isPerformanceinf 」→「ispSummaryTable」→「ispSummaryTableEntry」の順にダブルクリックします。リアルタイム監視で表示可能な性能情報の項目が表示されますので、その中から「ispSumObjectName」をクリックします。
3. MIB取得画面で「リストへ追加」ボタンをクリックします。

3) 基準値の設定

1. システム監視画面(Windows (R))、または、業務監視画面(Solaris)のメニューから、「ポリシー」→「ポリシーの定義」→「ノード」→「MIB監視」の順に項目をクリックして、MIB監視画面を表示します。
2. MIB監視画面で「追加」ボタンをクリックして、MIBしきい値の設定画面を表示します。
3. MIBしきい値の設定画面で、MIB名(監視する性能情報の表示名)、インスタンス番号、しきい値(基準値)を設定します。MIB名の指定方法は、2)を参照してください。
4. MIB監視画面の「有効」を選択します。

4) リアルタイム表示画面の表示

性能情報を表示します。
MIB取得画面で以下を設定し、「取得」ボタンをクリックします。

a. ポーリング時間に5分以上の値を指定

b. 取得方法に「GET」を指定

c. 監視するオブジェクトのインスタンス番号を指定
オブジェクト名の一覧を表示した際に、以下のように番号が表示されていますので、この番号をインスタンス番号に設定します。
「ispSumObjectName －＞ispSumObjectName. 番号:オブジェクト名」

d. 表示する性能情報の表示名をMIB名の欄に指定
MIB名の指定方法は、2)を参照してください。

リアルタイム監視で表示可能な性能情報については、「D.3.2 Systemwalker Centric Managerによるリアルタイム監視機能により採取した性能情報 」を参照してください。

注意

- オブジェクト名の一覧を表示する場合を除き、性能情報を表示する場合の取得方法には「GET」を指定してください。「DUMP」を指定してリアルタイム表示を行うと、Systemwalker Centric Managerと性能監視ツールの間で、膨大な回数の通信が発生し、ネットワークに多大な負荷を掛ける可能性があります。また、業務サーバへの負荷も増大します。
- 性能情報を表示できるオブジェクトがない場合は、オブジェクト名として「NONE」が表示されます。

D.2.5 性能監視ツール停止操作

性能監視ツールの停止操作について説明します。

性能監視ツール自動運用の場合 Windows32/64

Interstage停止時に性能監視ツールを停止する操作について説明します。

1. Interstageの停止
   isstopコマンドを実行し、Interstageを停止します。Interstage停止時に自動的に性能監視ツールが停止します。
2. 性能監視ツール自動起動定義の削除
   ispunsetautostartコマンドより性能監視ツール自動起動定義を削除します。
   定義の削除を行わない場合は、次回Interstage起動時に登録済みの定義で性能監視ツールが起動します。

注意

- 性能監視ツールを自動起動した場合は、ispdeleteenvコマンドによる性能監視ツールの停止はできません。
- 性能監視ツール自動起動定義の登録内容を変更する場合は、Interstage停止後、定義内容を変更した自動起動定義ファイルを使用して、ispsetautostartコマンドを実行してください。この場合、ispsetautostartコマンド実行前にispunsetautostartコマンドを実行する必要はありません。

性能監視ツール手動運用の場合

ispdeleteenvを実行して、性能監視環境を削除し、性能監視ツールを停止する操作について説明します。

1. 性能監視環境の削除
   ispdeleteenvコマンドにより性能監視ツールを停止し、性能監視環境を削除します。性能監視環境の削除は、性能監視停止後に実施してください。性能監視停止前に性能監視環境を削除した場合は、性能監視環境削除後の性能情報は採取されません。再度、性能監視を行う場合は、Interstageを再起動した後、性能監視環境の作成から行ってください。
2. Interstageの停止
   isstopコマンドを実行し、Interstageを停止します。

注意

- 性能監視環境を再度作成する場合は、性能監視環境の再作成前に、Interstageを再起動してください。
- 性能監視ツールを手動運用で起動した場合は、Interstage停止時に性能監視ツールは停止しません。

D.2.6 SNMPサービスからの削除操作 Windows32/64 Solaris64

Windows (R)の場合 Windows32/64

ispunsetagtコマンドを実行して、性能監視ツールをSNMPサービスから削除してください。ispunsetagtコマンド実行後は、以下の手順でSNMPサービスを再起動してください。

- 「コンピュータの管理」の「サービス」で、「SNMP Service」をクリックし、「停止」および「開始」操作を行ってください。

ispsetagtコマンドで性能監視ツールをSNMPサービスに登録していない場合は、この操作を行う必要はありません。

Solarisの場合 Solaris64

以下のファイルを削除後、マシンを再起動してください。

- /usr/lib/snmp/ispsubad8
- /etc/snmp/conf/ispsuba8.reg
- /etc/snmp/conf/ispsuba8.acl

・ /etc/snmp/conf/ispsuba8.rsrc

D.2.7 注意事項

性能監視ツールを使用する場合、性能監視ツールのコマンド(ispmakeenv/ispdeleteenv/ispstart/ispstop)とInterstageおよびワークユニットの操作コマンド(isstart/isstop/isstartwu/isstopwu)には、以下の順序性があります。

1. isstartとispmakeenvでは、どちらを先に実行してもかまいません。
   ただし、Interstage起動時にワークユニットを自動起動する設定となっている場合は、必ずispmakeenvを先に起動する必要があります。
2. isstartとispstartでは、isstartを必ず先に起動してください。
   isstopによりInterstageを停止した場合、ispstopを実行して性能監視も停止し、Interstage再起動時は、再度、isstart、ispstart の順で性能監視も開始してください。
3. isstartwuとispstartでは、どちらを先に実行してもかまいません。
4. ispmakeenvとisstartwuでは、ispmakeenvを必ず先に起動してください。

以上の関係をまとめると、コマンドの実行順は以下となります。

動作可否	手順1	手順2	手順3	手順4
○	ispmakeenv	isstart	ispstart	isstartwu
○	ispmakeenv	isstart	isstartwu	ispstart
○	ispmakeenv	isstart	(ワークユニット自動起動)	ispstart
×	ispmakeenv	ispstart	isstart	isstartwu
○	isstart	ispmakeenv	ispstart	isstartwu
○	isstart	ispmakeenv	isstartwu	ispstart
×	isstart	isstartwu	ispmakeenv	ispstart
×	isstart	(ワークユニット自動起動)	ispmamkeenv	ispstart

なお、実行順番を誤った場合、コマンドの実行は正常に終了しますが、性能ログファイルへのログ出力が実施されません。
また、InterstageおよびワークユニットをInterstageを再起動する場合は、性能監視ツールも停止し、上記の正しい順番で性能監視ツールも再起動してください。Interstageおよび、ワークユニット、性能監視ツールの停止については、性能監視時に実行した順番と逆の順で停止してください。

D.3 性能情報の分析と対処

性能ログファイルおよびリアルタイム監視で採取した性能情報の分析方法と対処方法について説明します。

D.3.1 性能ログファイルへのログ出力機能により採取した性能情報

性能ログファイルへのログ出力機能を使用した場合に採取できる性能情報と、その評価方法、対処方法について説明します。

・ 性能ログファイルの出力方法
・ 性能情報の項目内容

- 評価方法と対処方法

性能ログファイルの出力方法

ispreportコマンドを使用して、性能ログファイルのレポート出力を行います。ispreportコマンドは、性能ログファイルに保存されている性能情報を1レコードずつ読み出し、以下のようにCSV形式に変換して標準出力に出力します。出力される項目は、アプリケーション種別ごとに異なります。

```
D1, D2, D3, D4, D5, ......
```

ポイント

- 平均処理待ち要求数を出力する場合は、「-a WQUEAVG」オプションを付加してください。
- IJServerのEJBコンテナの性能情報を出力する場合は、「-k EJBAPL」オプションを付加してください。

性能ログファイルをCSV形式に変換してファイルに出力する場合は、ispreportコマンド実行時に、以下のようにして、出力先のファイル名を指定してください。

```
ispreport  オプション >  出力先ファイル名
```

性能情報の項目内容

採取できる性能情報は、アプリケーション種別ごとに項目が異なります。
性能情報として出力される項目について、アプリケーション種別ごとに説明します。各表の項番に書かれているD1、D2、・・・は、CSV形式で出力されるD1、D2、・・・に対応しています。

- IJServerのEJBコンテナの場合

IJServerのEJBコンテナの場合(項目数22)

以降の説明では、「IJServerのEJBコンテナ」を「EJBコンテナまたはアプリケーション」と記載します。

項番	性能情報の項目名	単位	内容
D1	データ採取開始日付	－	当該レコードの性能情報の測定を開始した日付
D2	データ採取開始時刻	－	当該レコードの性能情報の測定を開始した時刻
D3	データ採取終了日付	－	当該レコードの性能情報の測定を終了した日付
D4	データ採取終了時刻	－	当該レコードの性能情報の測定を終了した時刻
D5	EJBアプリケーション名	－	測定対象のEJBアプリケーション名 IJServer名/EJBアプリケーション名(最大288byte)
D6	メソッド名＋シグネチャ	－	測定対象のメソッド名＋シグネチャ(メソッドの引数と戻り値の型)
D7	プロセスID	－	測定対象のEJBコンテナのプロセスID
D8	スレッドID	－	測定対象のメソッドが動作するスレッドID
D9	最大要求処理時間	ミリ秒	当該スレッドにおける当該メソッドの処理時間(インターバル時間内での最大値/最小値/平均値)
D10	最小要求処理時間	ミリ秒	
D11	平均要求処理時間	ミリ秒	
D12	最大要求処理待ち時間	ミリ秒	クライアントアプリケーションからの要求を受け付けてから、メソッドが処理を開始するまでの待ち時間(インターバル時間内での最大値/最小値/平均値)

項番	性能情報の項目名	単位	内容
D13	最小要求処理待ち時間	ミリ秒	配備されたEJBアプリケーションがMessage-driven Beanの場合、「0」
D14	平均要求処理待ち時間	ミリ秒	
D15	処理数	回	当該スレッドにおける当該メソッドの処理回数(インターバル時間内での値)
D16	要求受信数	回	当該EJBコンテナまたはアプリケーションの累積処理回数(性能監視開始時からの累積値)
D17	処理待ち要求数	個	当該EJBコンテナに対して処理待ちとなった要求数(インターバル時間内での最大値) 配備されたEJBアプリケーションがMessage-driven Beanの場合、「0」
D18	EJBオブジェクト数(Session)	個	現在のEJBオブジェクト数(createメソッド実行数とremoveメソッド実行数の差分)(インターバル時間内での最大値)
D19	Entityの最大Passivate数	－	EJBコンテナ(プロセス)のインスタンスのプーリング回数(インターバル時間内での最大値)
D20	VMの最大メモリ使用量	Kバイト	EJBコンテナに対応するVMのメモリ使用量(インターバル時間内での最大値/平均値) インターバル時間にメソッドが処理されない場合、「0」
D21	VMの平均メモリ使用量	Kバイト	
D22	平均処理待ち要求数	個	当該オブジェクトに対して処理待ちとなった要求数(インターバル時間内での平均値) 要求を受信した契機に滞留していた処理待ち要求数の合計を、処理数で割った値

ispreportコマンドは、インターバル時間間隔の情報を、各プロセス上のメソッド＋シグネチャ単位に出力します。出力情報には、各プロセス上のメソッド＋シグネチャ単位の情報と、EJBコンテナ単位の情報があります。

- メソッド＋シグネチャ単位の評価(D9～D11、D15)
  D9～D11、D15は、D7に示すプロセス内の、D6に示すメソッド＋シグネチャに対する要求処理時間、処理数を示します。この情報を用いることにより、プロセスごとのメソッド＋シグネチャ単位の評価を行えます。
  なお、一度集計したD6のメソッド＋シグネチャに対する情報は、次回以降の集計インターバル時間内に、1度も動作しない場合、処理数0として出力されます。
- EJBアプリケーション単位の評価(D12～D14、D16～D18、D22)
  D12～D14、D16～D18、D22は、D5に示すEJBアプリケーションに対する要求処理待ち時間、要求受信数、処理待ち要求数等を示します。EJBアプリケーション単位の評価が行えます。
- プロセス単位の評価(D19～D21)
  D19～D21は、D7に示すプロセスに対するEntityの最大Passivate数、VMのメモリ使用量を示します。プロセス単位の評価が行えます。

評価方法と対処方法

性能ログファイルへのログ出力機能で採取した性能情報の評価方法と対処方法を、以下の一覧にまとめます。
性能異常を検出した場合は、一覧を参考にして対処してください。

項番	評価方法	対応/処置
1	性能監視を実施した全時間帯で、最大要求処理時間が長く、かつ、平均要求処理	要求処理時間が、目標値よりも長くかかっている場合には、以下の要因が考えられます。

項番	評価方法	対応/処置
	時間が、最大要求処理時間に近い時間となっている。	• サーバアプリケーションに性能問題がある • システムの負荷が高い 上記の観点で、サーバアプリケーションおよびシステムを見直してください。
2	特定の時間帯で、最大・平均・最小の各要求処理時間が長くなっている。	特定の時間帯に、システム負荷が高くなっている可能性があります。 他のサーバアプリケーションの性能情報も測定し、負荷状況を確認してください。
3	特定の時間帯で、最大・平均・最小の各要求処理待ち時間が長くなっている。	
4	最大要求処理時間は長いが、平均要求処理時間は短く、最小要求処理時間に近い時間となっている。	以下の要因が考えられます。 • 一時的にシステムの負荷が高くなった • 特定の条件下でサーバアプリケーションに性能問題がある 上記の観点で、システムおよびサーバアプリケーションを見直してください。
5	最大要求処理待ち時間は長いが、平均要求処理待ち時間は短く、最小要求処理待ち時間に近い時間となっている。	
6	性能監視を実施した全時間帯で、最大要求処理待ち時間および平均要求処理待ち時間が長くなっている。	クライアントからの要求数に対して、サーバアプリケーションの処理能力が不足しています。 ワークユニット定義でプロセス多重度を上げるなど、サーバアプリケーションの処理能力を上げる対処を実施してください。
7	特定の時間帯で、処理数・処理待ち要求数が多くなっている。	特定の時間帯にサーバアプリケーションに対する要求数が増加しています。 クライアントからの要求数に対して、サーバアプリケーションの処理能力が不足している場合には、ワークユニット定義でプロセス多重度を上げるなど、サーバアプリケーションの処理能力を上げる対処を行ってください。
8	EJBオブジェクト数がクライアント接続数より多くなっている。 (EJBアプリケーションの場合のみ有効)	EJBオブジェクトの数が増加しています。createメソッドに対するremoveメソッドの実行がされていない可能性があります。 クライアントアプリケーションの見直しを実施してください。
9	Passivate数が多くなっている。 (EJBアプリケーションの場合のみ有効)	Entity初期インスタンス数が不足しています。 Passiivate数の増加を目安にして、Entity初期インスタンス数を増やしてください。
10	VMのメモリ使用量が多くなっている。 (EJBアプリケーションの場合のみ有効)	メモリリークをしている可能性があります。 オブジェクトの削除の観点で、見直しを実施する等、サーバアプリケーションの見直しを実施してください。
11	処理待ち要求数が大きいが、平均要求処理待ち時間が短い。	isinfobjコマンドを使用して、定期的に待ちキューの状態を確認し、インターバル時間内の負荷状態を確認してください。

D.3.2 Systemwalker Centric Managerによるリアルタイム監視機能により採取した性能情報 Windows32/64 Solaris64

リアルタイム監視機能を使用した場合に採取できる性能情報と、評価方法、対処方法について説明します。

- 性能情報の項目内容
- 評価方法と対処方法

性能情報の項目内容

採取できる性能情報は、アプリケーション種別ごとに項目が異なります。
以下に、リアルタイム監視で表示可能な性能情報の項目を、アプリケーション種別ごとに示します。なお、表示名とは、Systemwalker Centric Managerで表示される、性能情報の項目名です。

- IJServerのEJBコンテナの場合

IJServerのEJBコンテナの場合

以下の情報は、IJServerのEJBコンテナ単位で採取されます。

性能情報の項目名	単位	表示名	内容
EJBアプリケーション名	ー	IspSumObjectName	測定対象のEJBアプリケーション名 IJServer名/EJBアプリケーション名(最大288byte)
最大要求処理時間	ミリ秒	IspSumExecTimeMax	当該EJBコンテナの処理時間(インターバル時間内での最大値/最小値/平均値)
最小要求処理時間	ミリ秒	IspSumExecTimeMin	
平均要求処理時間	ミリ秒	IspSumExecTimeAve	
最大要求処理待ち時間	ミリ秒	IspSumWaitTimeMax	クライアントアプリケーションからの要求を受け付けてから、メソッドが処理を開始するまでの待ち時間(インターバル時間内での最大値/最小値/平均値)
最小要求処理待ち時間	ミリ秒	IspSumWaitTimeMin	
平均要求処理待ち時間	ミリ秒	IspSumWaitTimeAve	
要求受信数	個	IspSumRequestNum	当該EJBコンテナの累積処理回数(インターバル時間内での値)
処理待ち要求数	個	IspSumWaitReqNum	当該EJBアプリケーションに対して処理待ちとなった要求数(インターバル時間内での最大値)

評価方法と対処方法

リアルタイム監視機能で採取した性能情報の評価方法と対処方法を、以下の一覧にまとめます。
性能異常を検出した場合は、以下の一覧を参考にして対処してください。また、性能ログファイルに出力された性能情報も評価の参考としてください。

項番	評価方法	対応/処置
1	性能監視を実施した全時間帯で、最大要求処理時間が長く、かつ、平均要求処理時間が、最大要求処理時間に近い時間となっている。	要求処理時間が、目標値よりも長くかかっている場合には、以下の要因が考えられます。 ・ サーバアプリケーションに性能問題がある ・ システムの負荷が高い 上記の観点で、サーバアプリケーションおよびシステムを見直してください。
2	特定の時間帯で、最大・平均・最小の各要求処理時間が長くなっている。	特定の時間帯に、システム負荷が高くなっている可能性があります。 他のサーバアプリケーションの性能情報も測定し、負荷状況を確認してください。
3	特定の時間帯で、最大・平均・最小の各要求処理待ち時間が長くなっている。	
4	最大要求処理時間は長いが、平均要求処理時間は短く、最小要求処理時間に近い時間となっている。	以下の要因が考えられます。 ・ 一時的にシステムの負荷が高くなった ・ 特定の条件下でサーバアプリケーションに性能問題がある 上記の観点で、システムおよびサーバアプリケーションを見直してください。
5	最大要求処理待ち時間は長いが、平均要求処理待ち時間は短く、最小要求処理待ち時間に近い時間となっている。	
6	性能監視を実施した全時間帯で、最大要求処理待ち時間および平均要求処理待ち時間が長くなっている。	クライアントからの要求数に対して、サーバアプリケーションの処理能力が不足しています。 ワークユニット定義でプロセス多重度を上げるなど、サーバアプリケーションの処理能力を上げる対処を実施してください。
7	特定の時間帯で、処理数・処理待ち要求数が多くなっている。	特定の時間帯にサーバアプリケーションに対する要求数が増加しています。 クライアントからの要求数に対して、サーバアプリケーションの処理能力が不足している場合には、ワークユニット定義でプロセス多重度を上げるなど、サーバアプリケーションの処理能力を上げる対処を行ってください。

D.3.3 性能情報評価時の注意事項

以下に、性能情報評価時の注意事項を説明します。

・ サーバアプリケーションが、処理中に異常終了した場合、その要求は、性能情報に反映されません。

D.4 性能ログファイルの運用

ディスク容量の見積り

性能監視ツールを起動する前に、性能ログファイルを作成するための十分なディスク容量が確保されていることを確認してください。ディスク容量の見積もり方法を、以下に示します。

```
ディスク容量　＝性能監視ツール起動時に指定する共有メモリのサイズ ×
　　（ 性能監視ツールを起動してから停止するまでの時間 ÷
　　　性能監視ツール起動時に指定するインターバル時間 ）
```

「性能監視ツール起動時に指定する共有メモリのサイズ」の見積もり方法の詳細については、「チューニングガイド」の「性能監視ツール使用時に必要なシステム資源」を参照してください。

性能ログファイルのバックアップ

性能ログファイルは、定期的にバックアップして、不要になったファイルは削除してください。定期的に削除しない場合、ディスク容量を圧迫する可能性があります。バックアップ、削除したファイルをレポート出力する場合は、そのファイルを任意のディレクトリに格納して、そのファイルを指定してレポートを出力してください。

性能ログファイル名

性能ログファイルが作成されるディレクトリは、以下のようになります。

- ispmakeenvコマンドのパラメタで指定したディレクトリ
- 環境変数ISP_LOGに指定したディレクトリ

両方を指定した場合は、ispmakeenvコマンドで指定したディレクトリ名が優先されます。両方共指定がない場合は、以下のデフォルトのディレクトリ名を使用します。

Windows32/64

C:¥Interstage¥td¥isp¥log

Solaris64

/opt/FSUNtd/isp/log

Linux32/64

/opt/FJSVtd/isp/log

上記のディレクトリ配下に、以下の命名規約に従って性能ログファイルが作成されます。

```
性能ログファイル名:ispYYYYMMDD.log
    YYYYMMDDは、ファイル作成日付
    ー YYYY:西暦
    ー MM:月(01～12)
    ー DD:日(01～31)
```

性能監視ツールは、起動された時点で、その日の日付に対応する性能ログファイルを作成します。また、数日間に渡って性能監視ツールを起動した場合は、日付単位に起動日数分の性能ログファイルを作成します。

注意

ispmakeenvコマンド実行後に、性能監視ツールが作成中の性能ログファイルを削除しないでください。削除した場合は、性能情報が正しく保存されない可能性があります。性能ログファイルを削除する場合は、ispdeleteenvコマンドを実行してください。

D.5 性能監視ツール運用時に使用する定義ファイル

性能監視ツールを使用する場合に、コマンドで指定するファイルについて説明します。

D.5.1 性能監視対象指定ファイル(ispstartコマンド)

性能監視ツールの性能監視対象を指定するファイルです。ispstartコマンドのパラメタとして指定します。

形式

```
［セクション名］
性能監視対象アプリケーション
  :
［セクション名］
性能監視対象アプリケーション
  :
```

指定項目

「セクション名」は[]で囲み、その下に「性能監視対象アプリケーション」を記述します。
性能監視対象は、合計1000個まで指定可能です。

指定可能な「セクション名」と、それぞれのセクションの「性能監視対象アプリケーション」で指定するものを以下に示します。

セクション名	性能監視対象アプリケーション
EJBCONT	IJServerを指定 **注意** IJServerタイプのIJServer(Web + EJB[1VM])およびIJServer(Webのみ)の性能情報は採取できません。 IJServer(Web + EJB[別VM])の場合、EJBアプリケーションが動作するJava VMの情報のみ採取できます。

定義例

以下に、定義ファイルの記述例を記載します。

```
[EJBCONT]
myserver
IJServer_Split
IJServer_EJB
```

D.5.2 性能監視自動起動定義ファイル(ispsetautostartコマンド) Windows32/64

性能監視ツール自動運用時に、性能測定を行う環境および性能監視対象アプリケーションを指定するファイルです。性能監視対象は、合計1000個まで指定可能です。ispsetautostartコマンドのパラメタとして指定します。

形式

```
［セクション名］
定義項目
  :
［セクション名］
定義項目
  :
```

指定項目

「セクション名」は[]で囲み、その下に定義項目を記述します。指定可能な「セクション名」とそれぞれのセクションで指定可能な定義項目について以下に説明します。

セクション名: Control

定義項目	意味
Shmsize	共有メモリのサイズをMbyteで指定します。省略値は1です。 指定可能な最小値は1で、最大値はシステムで定義されている共有メモリ量の最大値(MB)または2046のうち、どちらか小さい値です。 性能監視ツールでは、性能情報採取に共有メモリを使用します。採取される性能情報量から容量を算出し、指定してください。性能情報量の見積り方法は、「チューニングガイド」を参照してください。
Log_path	性能ログファイルの出力先を指定します。 省略値は「C:¥Interstage¥td¥isp¥log」です。
Auto_start	定義ファイルに指定した性能監視対象アプリケーションに対して、Interstage起動時に自動的に性能監視を開始するか否かを指定します。省略値はNOです。 ・ YES:Interstage起動時に、指定した性能監視対象アプリケーションに対して、自動的に性能監視が開始されます。性能監視対象アプリケーションが指定されていない場合、性能監視は開始されません。 ・ NO:Interstage起動時に、性能監視が開始されません。

セクション名: Interval

定義項目	意味
local_interval	性能ログファイル採取用インターバル時間を指定します。省略値は1です。指定可能な範囲は以下のとおりです。 ・ 時間単位:1、2、3、4 ・ 分単位:1m、5m、10m、20m、30m
real_interval	リアルタイム監視用インターバル時間を分単位で指定します。省略値は5です。1～60の範囲で指定できます。

セクション名: EJBCONT

IJServerを指定します。

注意

「Auto_start=YES」かつ性能監視対象アプリケーション名に誤りがある場合は、性能監視ツールの起動のみ行われ、性能監視は開始されません。性能監視対象指定ファイルに正しいアプリケーション名を指定して、ispstartコマンドより性能監視を開始してください。

定義例

以下に、定義ファイルの記述例を記載します。

Windows32

```
[Control]
Shmsize = 10
Auto_Start = NO
Log_Path = c:¥log

[Interval]
```

```
local_interval = 5m
real_interval = 1

[EJBCONT]
myserver
IJServer_Split
IJServer_EJB
```

Windows64

```
[Control]
Shmsize = 10
Auto_Start = NO
Log_Path = c:¥log

[Interval]
local_interval = 5m
real_interval = 1

[EJBCONT]
myserver
IJServer_Split
IJServer_EJB
```

索 引

[た]

[な]

[は]

[ま]

[や]

[ら]

[わ]