参考資料：OpenIDの概要

- OpenIDConnect/OAuth2.0 とは？
- アイデンティティ＆アクセスマネジメント（IAM）の定義と分類
- IAMの進化は何をもたらすのか

OpenIDConnect/OAuth2.0 とは？

グローバルでの技術標準化が普及を後押し

ID連携

ID情報（認証結果と属性情報）を安全にサービス間でやりとりするための仕様

OpenID 2.0
（オープンアイディー）

API連携

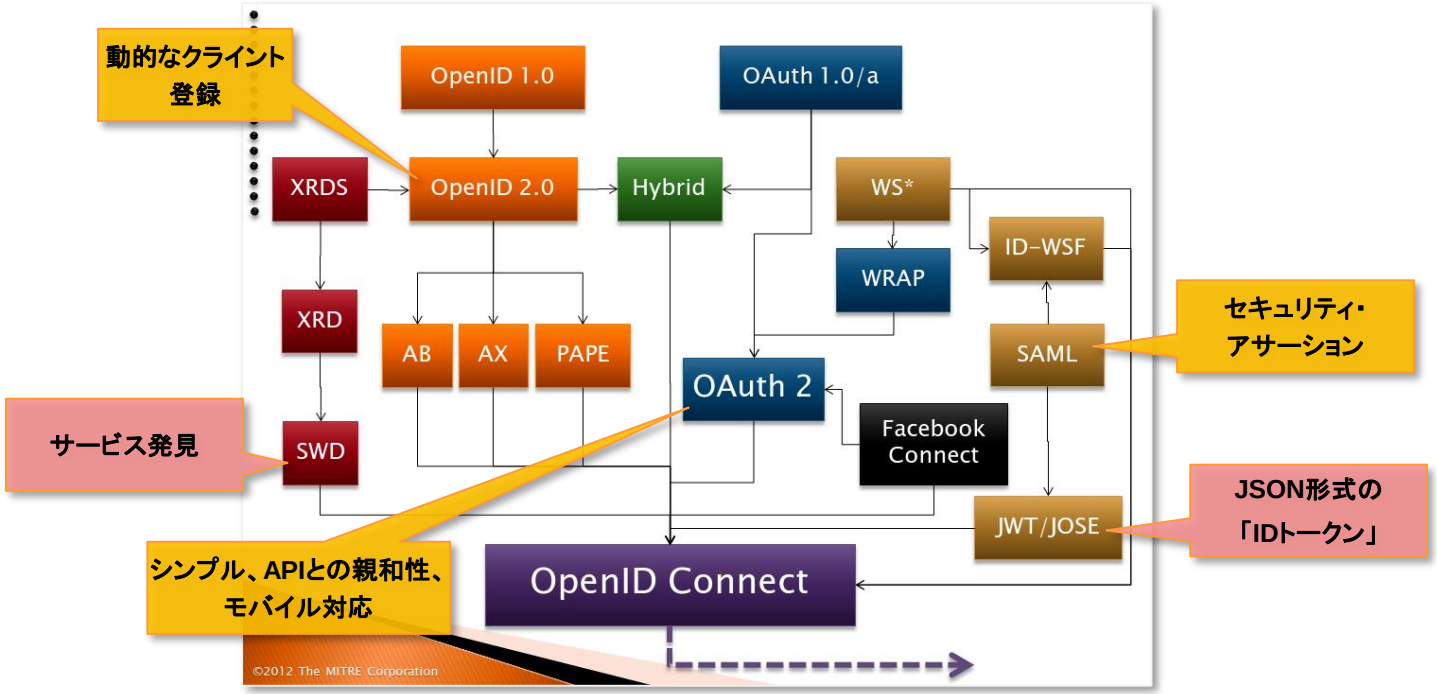
あるサービスのAPIアクセスの許可を、別のサービスに安全に与えるための方法

OAuth 2.0
（オーオース）

OpenID Connect

主要ID/API連携仕様がすべてOpenID Connectに収斂

- とくにOpenID Connectの中核であるIDトークンは、SAMLのセキュリティ・アサーション（XML）を参考にしつつ、JSONをベースとするJWT
  （JSON Web Token）ことにより、さまざまな言語や実行環境において対応しやすいものとなっている



Source: http://civics.com/OpenID-connect-webinar/

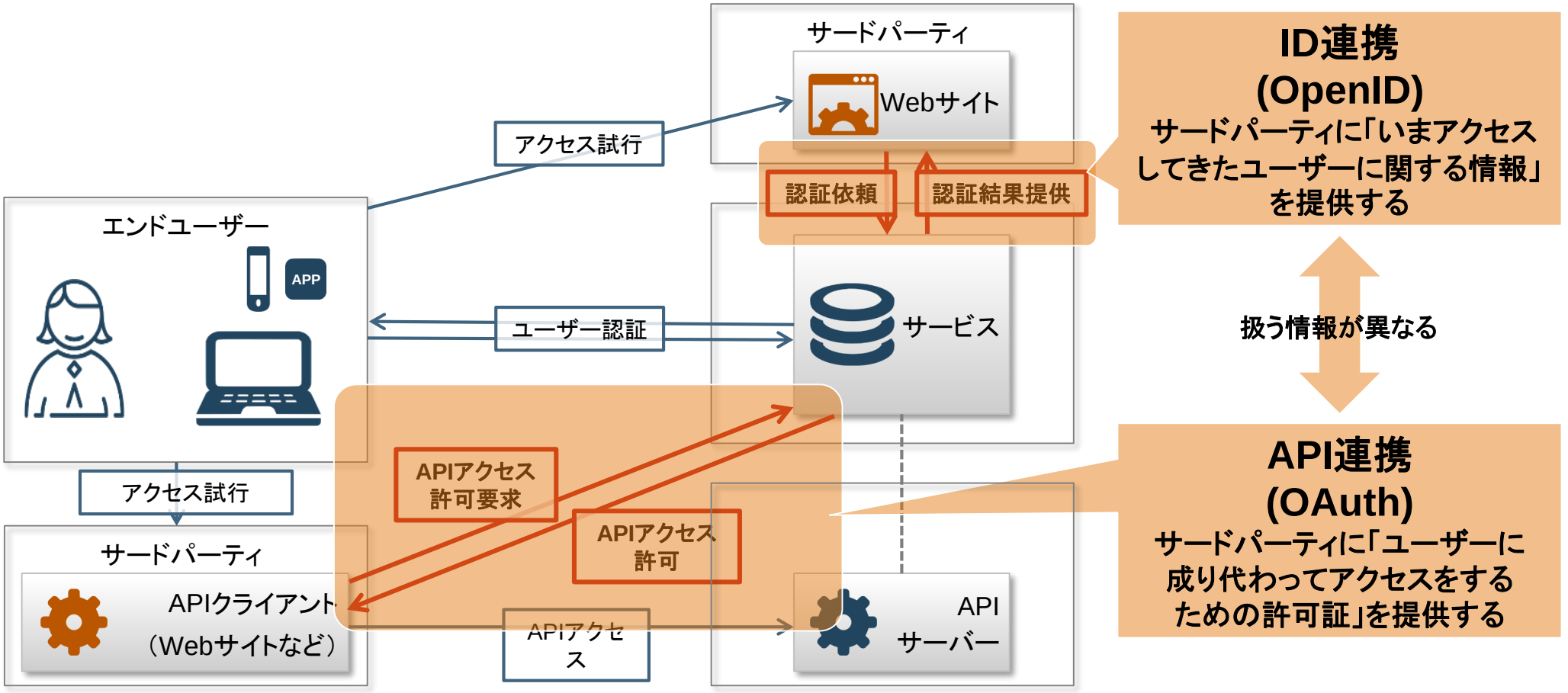
ID連携とAPI連携（認可）を支えてきた技術

ID情報を積極的に外部に提供しているサービス事業者を中心に、標準技術の採用が一巡。一方でOpenIDConnectが世に広まる前は、OAuthでID連携をカスタム実装する（またはOpenIDでAPI連携）などの「正しくない使い方」が散見されていた。

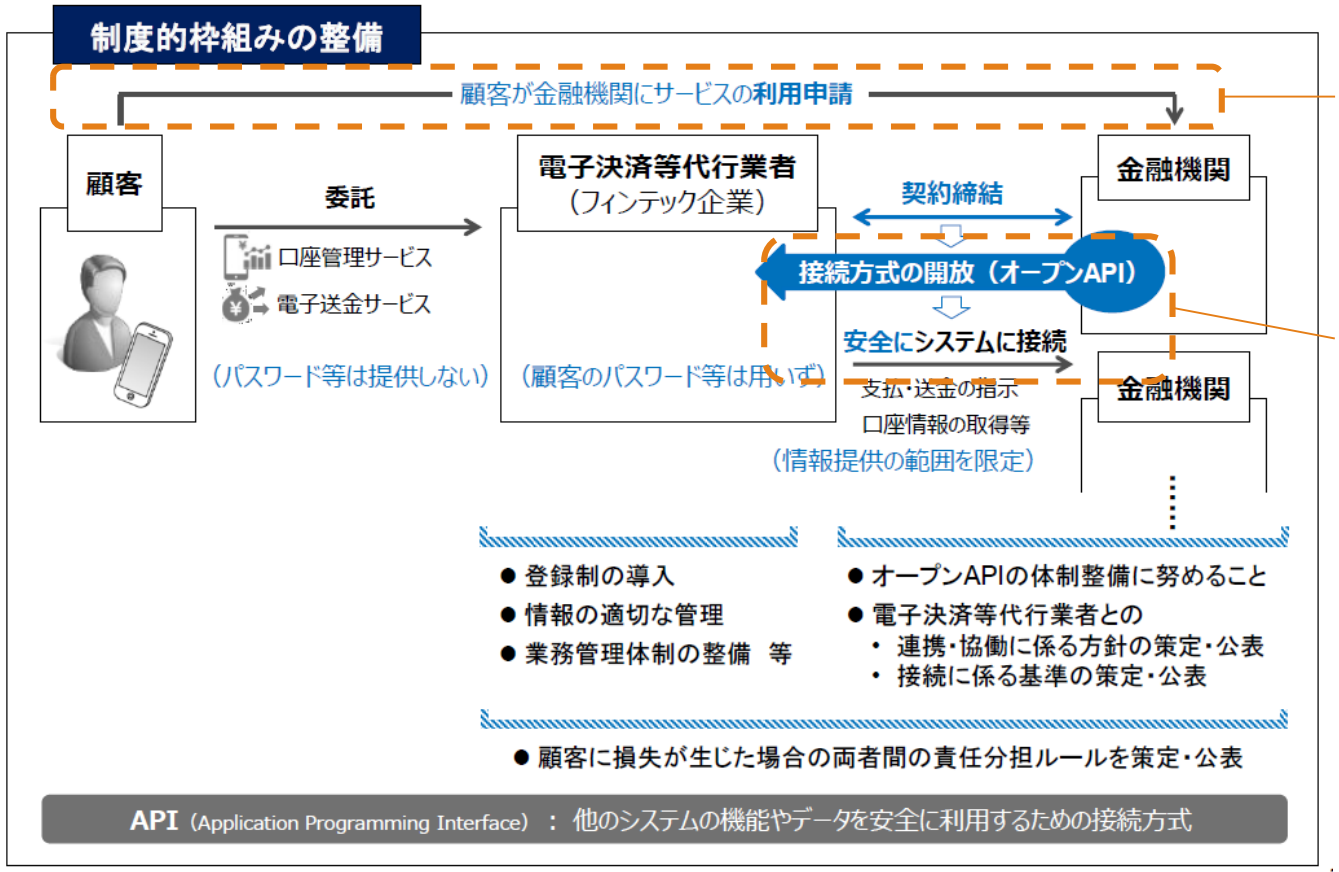
事業者	会員数	ID連携に対応したサービス	ID連携・API連携プロトコル
Yahoo! JAPAN	総アカウント数：約2億人 （アクティブユーザー数： 約2,641万人）	認証、属性、決済、ポイント	OpenID 2.0, OAuth 1.0a, OAuth 2.0, OpenID Connect
Google	Gmail アカウント数：約4.2億人	認証、属性、決済、ソーシャル他	OpenID 2.0, OAuth 2.0, OpenID Connect
Twitter	約2億人	認証、属性、ソーシャル	OAuth 1.0a
Facebook	約11.1億人　（国内推定 約1,378万人）	認証、属性、ソーシャル	OAuth 2.0
ミクシィ	2,700万人 (アクティブユーザー 1,402万人)	認証、属性、ソーシャル	OpenID 2.0, OAuth2.0、 OpenID Connect
NTT IDログイン	延べ 7,000万		
NTTドコモ	契約者数：6,154万人	認証、決済、ストレージ、電話帳	OpenID 2.0, OAuth 2.0
OCN	約844万人	認証	OpenID 2.0
goo	約1,000万人強	認証	OpenID 2.0
KDDI	契約者数：3,792万人 ネット接続サービス：3,043万人 au ID会員数：1,000万人	認証、決済、年齢認証他	OpenID 2.0他
SoftBank	契約者数： 約3,274万人	認証、決済	OpenID 2.0他
楽天	楽天会員数：7,500万人	認証、決済、ブックマーク他	OpenID 2.0、OAuth 2.0
JAL	JALマイレージバンク会員数：約2,500万人	認証、属性	OpenID 2.0
PayPal	総アカウント数：2.2億万	認証、属性、決済	OpenID 2.0、OAuth 2.0、 OpenID Connect
日本経済新聞社	日経電子版 会員数（有料）：30万人	認証、属性	OpenID Connect
サントリー	非公開	認証、属性、ポイント	OpenID 2.0, OAuth 2.0
東京急行電鉄	東急ポイント会員数：295万人	認証、属性	OpenID Connect

2012年当時。上記公表企業以外にも、多数企業が採用

OAuthとOpenIDとの関係



改正銀行法におけるID、API連携の枠組み　～OpenID Connect（OAuth）仕様を前提～



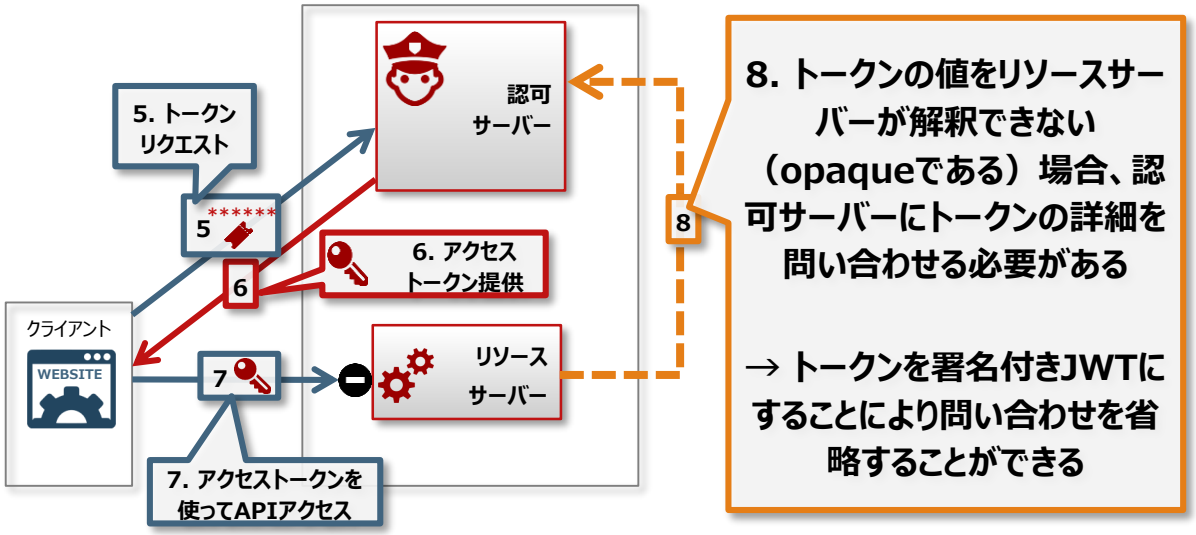
金融審議会・金融制度ワーキング・グループ報告書（3月3日）より抜粋

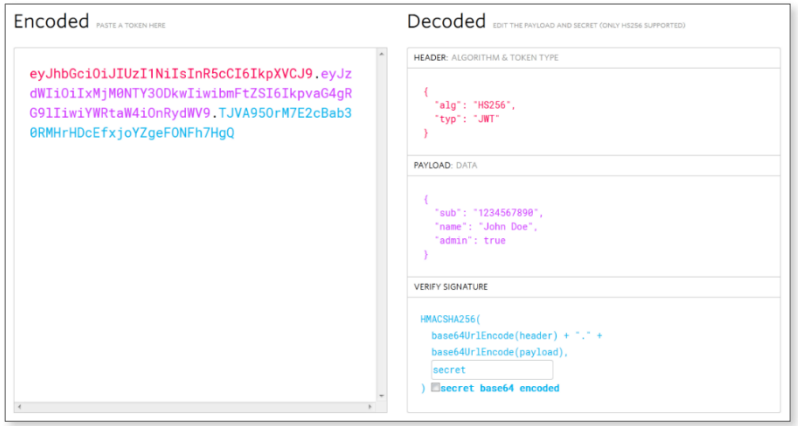
それぞれの仕様はカバレッジが異なり、ID連携とAPI連携の双方を提供するならば、OpenIDConnectがベスト

		比較項目 ＼ 仕様	SAML 2.0	OpenID 2.0	OAuth 2.0	OpenID Connect
ID提供側（IDP）	ID連携	認証結果の連携	○	○	×（仕様外であり、OAuth採用事業者の独自拡張が乱立）	○
		属性情報の連携	○	○	×（仕様外であり、OAuth採用事業者の独自拡張が乱立）	○
		認証結果・属性情報への署名や暗号化	○	△（共有鍵による署名のみ可能。暗号化は仕様のスコープ外）	×（そもそも、認証結果・属性情報の連携は仕様のスコープ外）	○
		サイト間のセッション管理	○	×（仕様のスコープ外）	×（仕様のスコープ外）	○
		Webブラウザ以外（デスクトップアプリ、スマートフォンアプリなど）への対応	△（仕様上はWebブラウザ以外にも対応可能だが、実際に広く利用されているのはWeb SSOのみ）	×（Webブラウザのリダイレクト機能に依存したプロトコルであり、Webブラウザ以外への対応は不可能）	○	○
	API連携	APIアクセス認可（アクセストークン配布）への対応	×（仕様のスコープ外）	△（OAuth 1.0とのハイブリッドにより対応）	○	○
		複数のアクセストークンへの対応	×（仕様のスコープ外）	×（仕様のスコープ外）	×（仕様のスコープ外）	○
		PC以外のデバイス（スマートフォン等）への対応	×	×	○	○
ID受入側		ID受け入れ側（リライング・パーティ）の実装・運用の容易さ	×（XMLやSOAP、PKIなどのスキルが必要であり、通常はSAML処理ライブラリやミドルウェアの導入が必須）	△（鍵交換や署名処理が必要であり、通常はOpenID処理ライブラリなどの導入が必須）	○（ライブラリ等の導入が不要）	○（ライブラリ等の導入が不要）
備考			さまざまなユースケースに適用可能なフレームワークであるが、それゆえに複雑であり、結果的には歴史的経緯による企業向けSaaSのSSOや、同一サービス内でのID連携に利用されるにとどまっている。	Webサイト間の認証結果・属性情報の交換に特化したプロトコルであり、従前の仕様（SAML 2.0)に比較して単純なプロトコルであるが、鍵交換や署名処理など、まだ実装が容易ではない点が残る。	OAuth 1.0をベースに、より容易に実装できるように仕様を簡略化。Web APIのアクセス認可フレームワークとして広く普及。しかしID連携に関してはスコープ外であり、独自仕様が乱立している。	OAuth 2.0をベースに、ID連携のためのプロトコルを定義。OAuth 2.0の実装のしやすさを活かしつつ、ID連携に十分な機能を定義している。

JWT（JSON Web Token）/ JWS（JSON Web Signature）

- OAuth 2.0ではトークンの授受・利用を定義する一方、トークンの形式については仕様のスコープ外としている
- ユースケースによっては、トークン自体に認証情報を記録し、電子署名を付加して改ざんを防止するといった、トークン自体の構造化を行うことで、効率的なアクセス認可を行いたい場合もある
- JWT/JWSはJSONオブジェクトとして記述された認証情報、および電子署名を、コンパクトに表現するための標準(RFC 7519, 7515)
- 現在JWT/JWSはトークンとしてだけではなく、クライアントがサーバーに自身を証明するための認証情報や、APIリクエスト/レスポンスそのものなど、「JSON+電子署名」の標準として広く使われている





JWTの例

Source: JSON Web Tokens - jwt.io
https://jwt.io/

異なるサブシステム間でID連携、API連携プロトコルにおいて「標準」を使うことの意義

■セキュリティ・プライバシー対策と答責性

・認証サーバ、認可サーバ、クライアントの成りすまし、中間者攻撃、再生攻撃、トークン置換攻撃等各種攻撃に対する耐性を考慮された設計。
・プライバシーコントロールを考慮したプロトコルユースケースの設計。（同意管理、スコープ管理）
・外部に対して、セキュリティ標準を遵守していること、一定のセキュリティ対策水準を満たしていることのアカウンタビリティ

■ユースケース拡張性

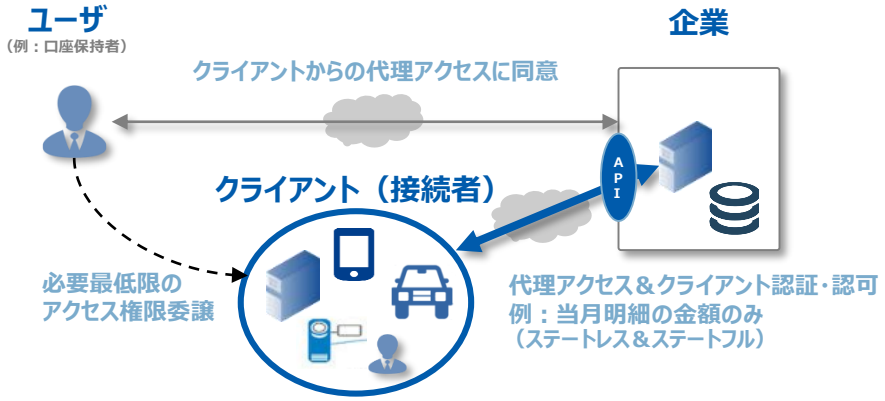
・デバイスタイプ、アプリケーションタイプの拡張性を意識した設計。サードパーティー（信頼度のさがるクライアント）からのサービス連携に対して、適切な最小限の認可範囲、有効期限をグラントするための枠組み。

■相互接続容易性

・標準であるため仕様がオープンであり多くの製品ライブラリが対応しているため、サードパーティーからの接続性が確保しやすいこと。接続容易性は相互セキュリティ維持性（カスタム仕様の場合誤認識、誤用途による脆弱性を生む可能性）

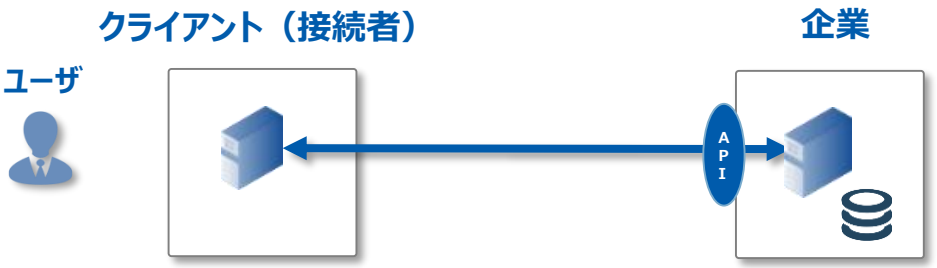
大きく二つの通信モデルが存在する。今日のFintech界隈で注目されるのはユーザ同意ベースのAPI

ユーザ同意ベースのAPI通信モデル



ユーザ認証
ユーザ同意取得
アクセストークン・権限委譲
クライアント認証／サーバ認証
メッセージ認証／署名
クライアント認可
クレデンシャルライフサイクル管理
トークンライフサイクル管理

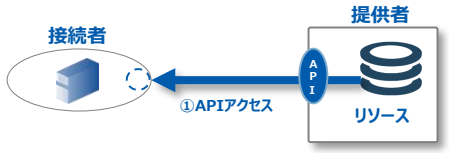
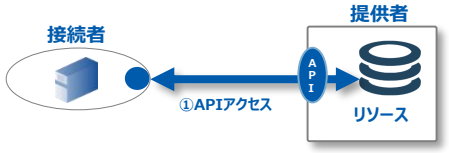
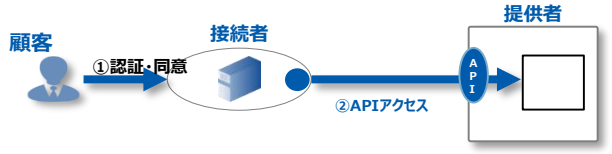
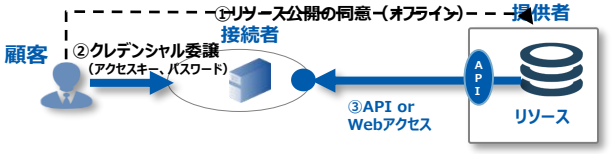
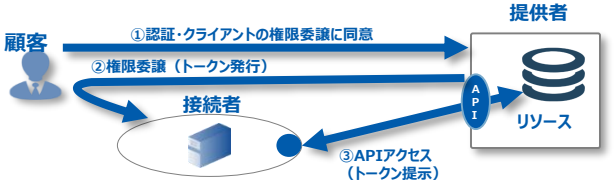
分散システム間のAPI通信モデル



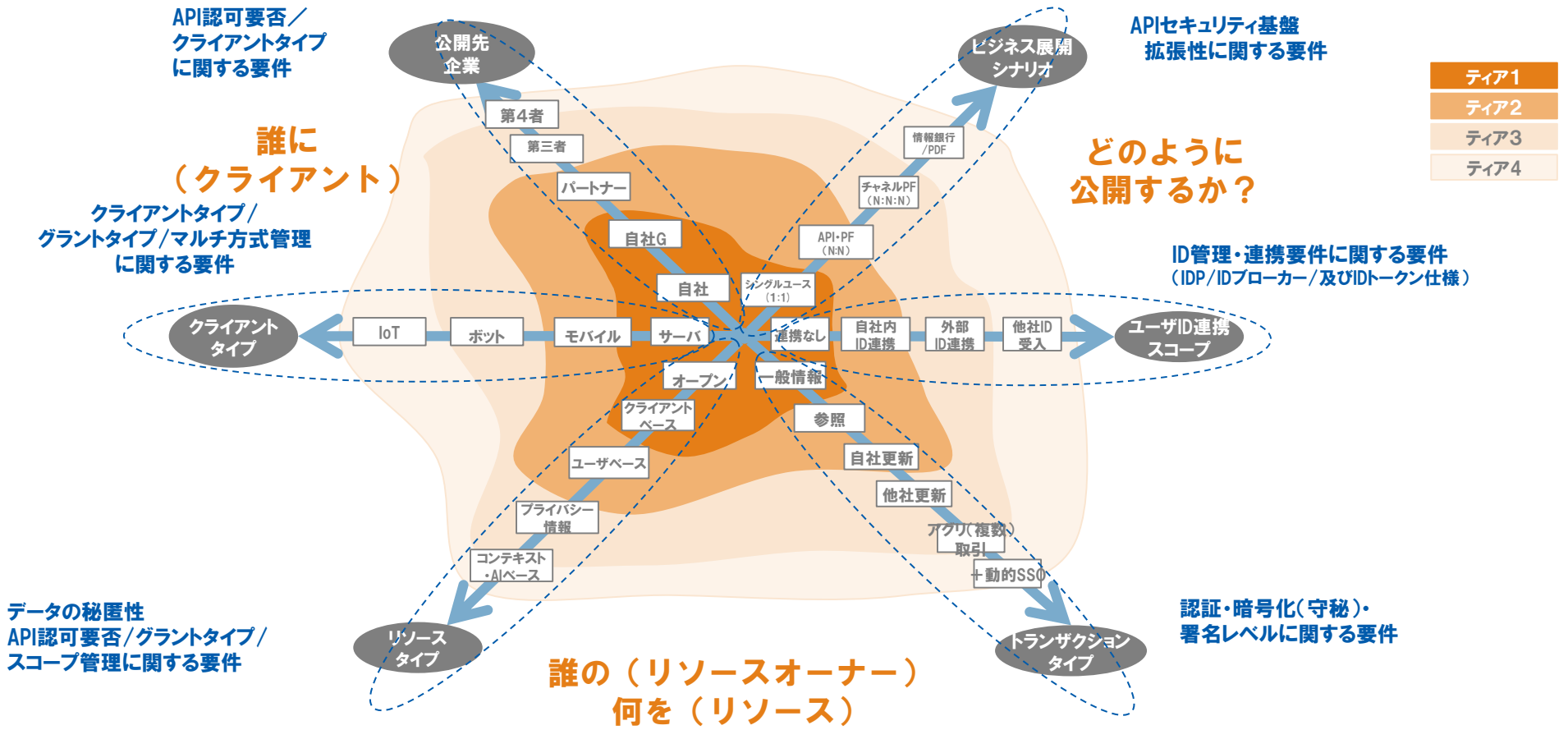
ユーザ認証
ユーザ同意取得
アクセストークン・権限委譲
クライアント認証／サーバ認証
メッセージ認証／署名
クライアント認可
クレデンシャルライフサイクル管理
トークンライフサイクル管理

APIユースケースの分類

ケース5がより複雑なモデル

			情報の性質	情報の方向	ユースケースの特徴 → オンライン
分散システム間	ケース1		公知情報	←参照	商品、営業所の所在地情報のAPI
	ケース2		事業者情報 事業者重要情報	←参照 →更新	企業間契約・発注情報等の事業者向けAPI
	ケース3		個人情報 個人機微情報	→登録	職域、団体者向けサービス他、接続者と顧客の2社間で同意した顧客情報の登録。
ユーザ同意ベース	ケース4）		個人情報	←参照	オフラインで事前同意（提供者、顧客）した顧客情報の公開。 ただし①、②、③の同意ベースのアクセスコントロールをオフラインToオンラインで実現するのは難しい（①が存在しない③勝手WEBアクセスはいわゆるスクレイプ問題） →ケース５に移行すべき
	ケース5		個人情報 個人機微情報	←参照 →更新	ユーザ同意ベースのAPIアクセス。 オンラインで顧客との同意、権限委譲、接続者による代理APIアクセスを実現するモデル。Oauth2.0フレームワークを採用が事実上必須。

API認証・認可系の要求レベル軸例



OAuth2.0/OpenID Connectには、ユースケースによって多様なグラントタイプが存在する

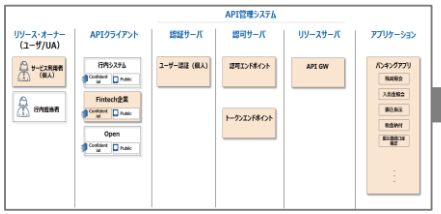
仕様定義	グラントタイプ	適用する環境条件	ビジネスユースケース例
OAuth2.0標準	認可コード	最も基本となるグラントタイプ。セキュリティ面からのメリットも高い。リソースオーナーとクライアントがリダイレクトで通信可能な前提。	・BtoCサービスでの活用前提 ・PFMシステム、サードパーティーアプリ等
	インプリシット	Webブラウザアプリケーション（JavaScript等がクライアントとなるケース）での適用。またはマルウェアによるアクセス権限の窃取等リスクが小さい（または許容できる）前提となる。	軽量アプリで利便性を重視する場合。例えばポイント数参照等、仮に漏えいした場合でも被害が限定的なリソース連携時等
	クライアントクレデンシャル	リソースオーナーがクライアントシステム管理者である場合等。加えてリソースオーナーとクライアントがリダイレクト通信不可能なケース。	・事業者間のデータ連携 （SCM、ASP事業/クラウド事業間連携） ・M2M、IoT連携
	リソースオーナー パスワードクレデンシャル	特権のあるクライアントであることが前提。 クライアントとリソース管理企業が同一、またはビジネス上、非常に密接な環境における連携等。	・グループポータルサイト（クライアント）とリソース連携等 ・金融機関に委託で提供しているアグリゲーションサービス等
OAuth2.0拡張	アサーション	クライアント側にてAPIクライアントの認証を行うサーバーを、API提供側の認可サーバーが信用していること	・法人顧客や、企業グループ内の関連会社からの「代表アカウント（個人ではない）」を用いたAPIアクセス等
OpenID Connect拡張	ハイブリッド	モバイルアプリのフロントエンド（端末側）とモバイルアプリのバックエンド（サーバー側）が双方クライアントとして認可権限を有してAPIアクセスをする場合（それぞれにアクセストークンを払い出す）	・ビジネスロジックはサーバサイド、プレゼンテーションロジックはモバイルアプリ等の機能配置を前提としたサービス等

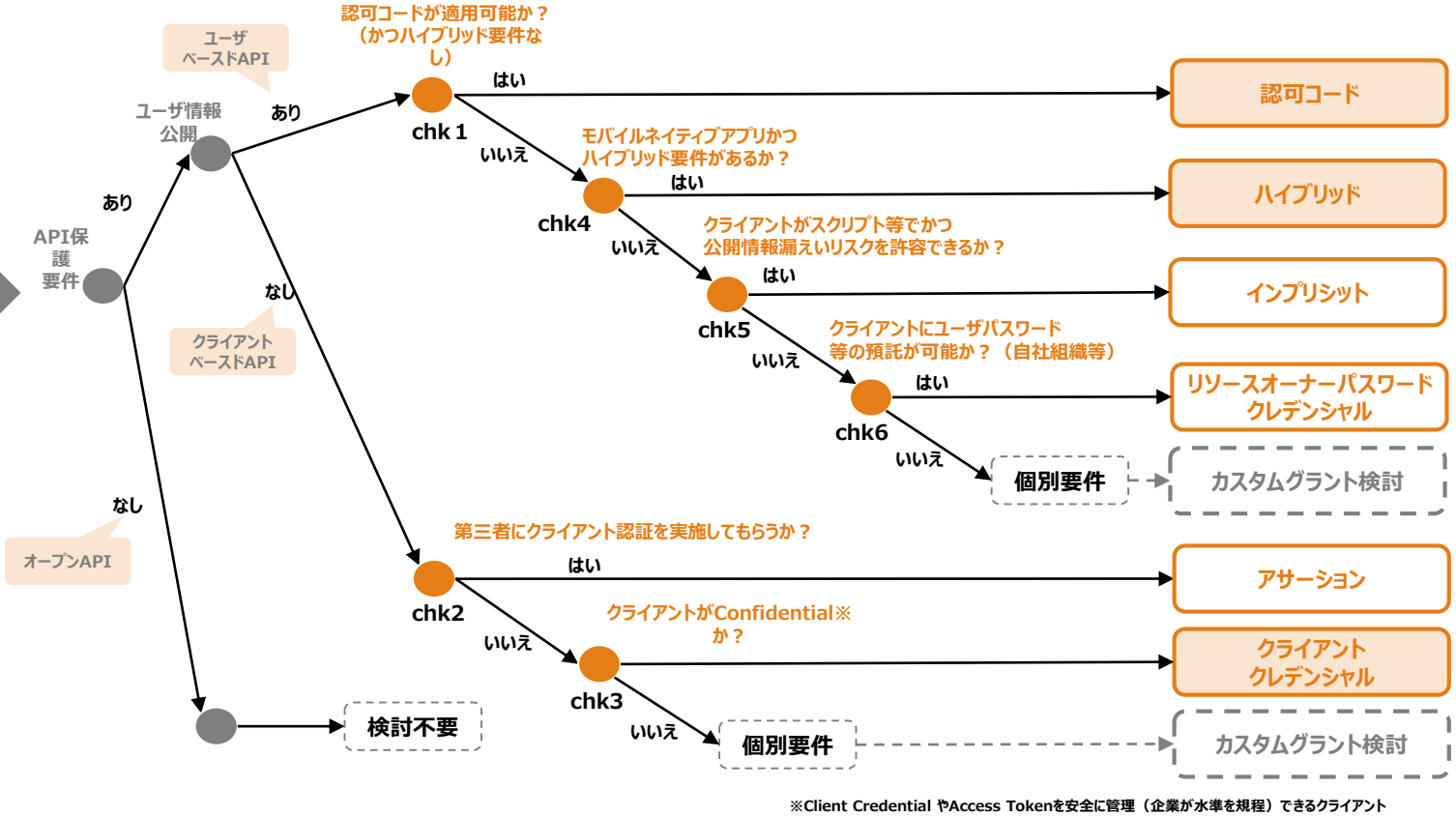
OAuthとは？

適用すべきグラントタイプはユースケースを前提としつつ、セキュアな方式を優先したディシジョンツリーにより決定

API認可の要否決定 ▶ **グラントタイプの決定** ▶ クライアント管理レベルの決定 ▶ トークン管理レベル/暗号・署名鍵レベルの決定 ▶ ID管理との連携要否の決定

ユースケース/API利用環境

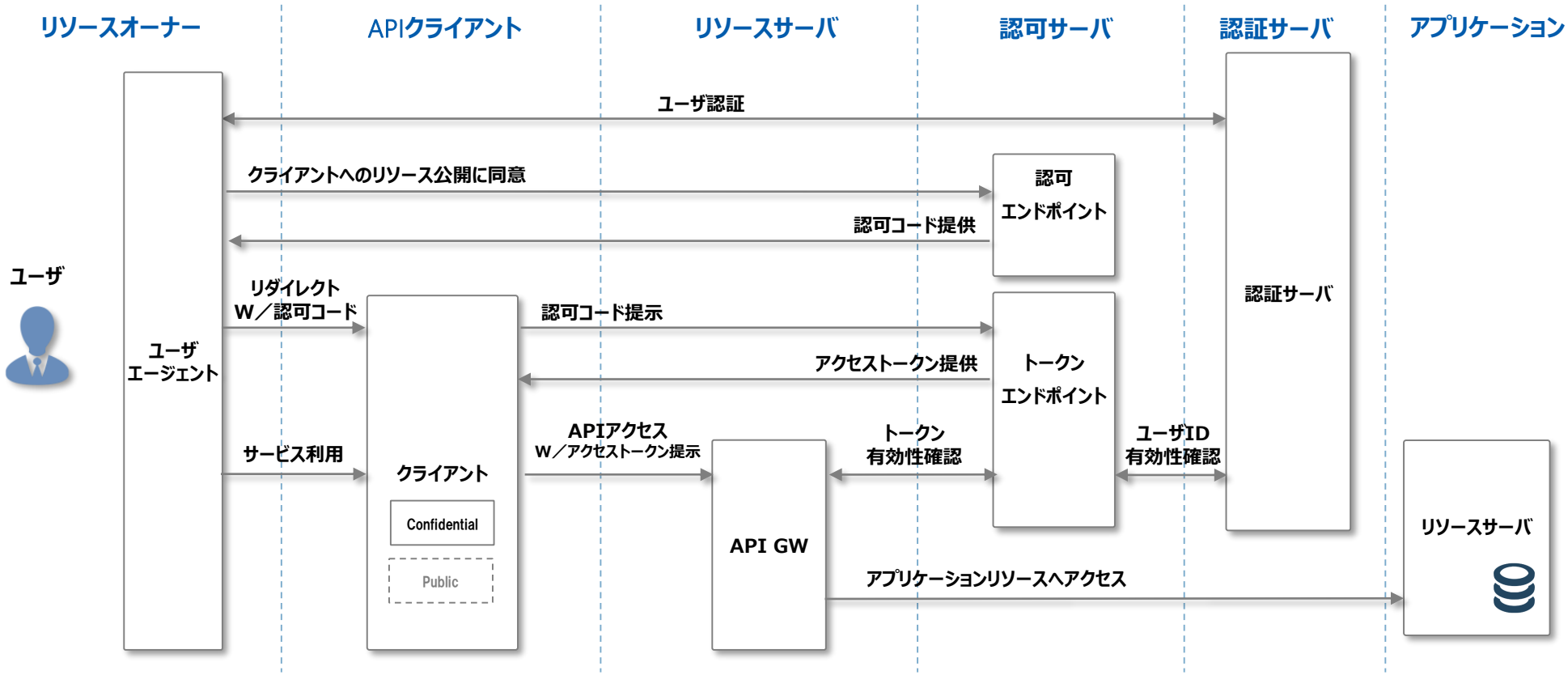




※Client Credential やAccess Tokenを安全に管理（企業が水準を規程）できるクライアント

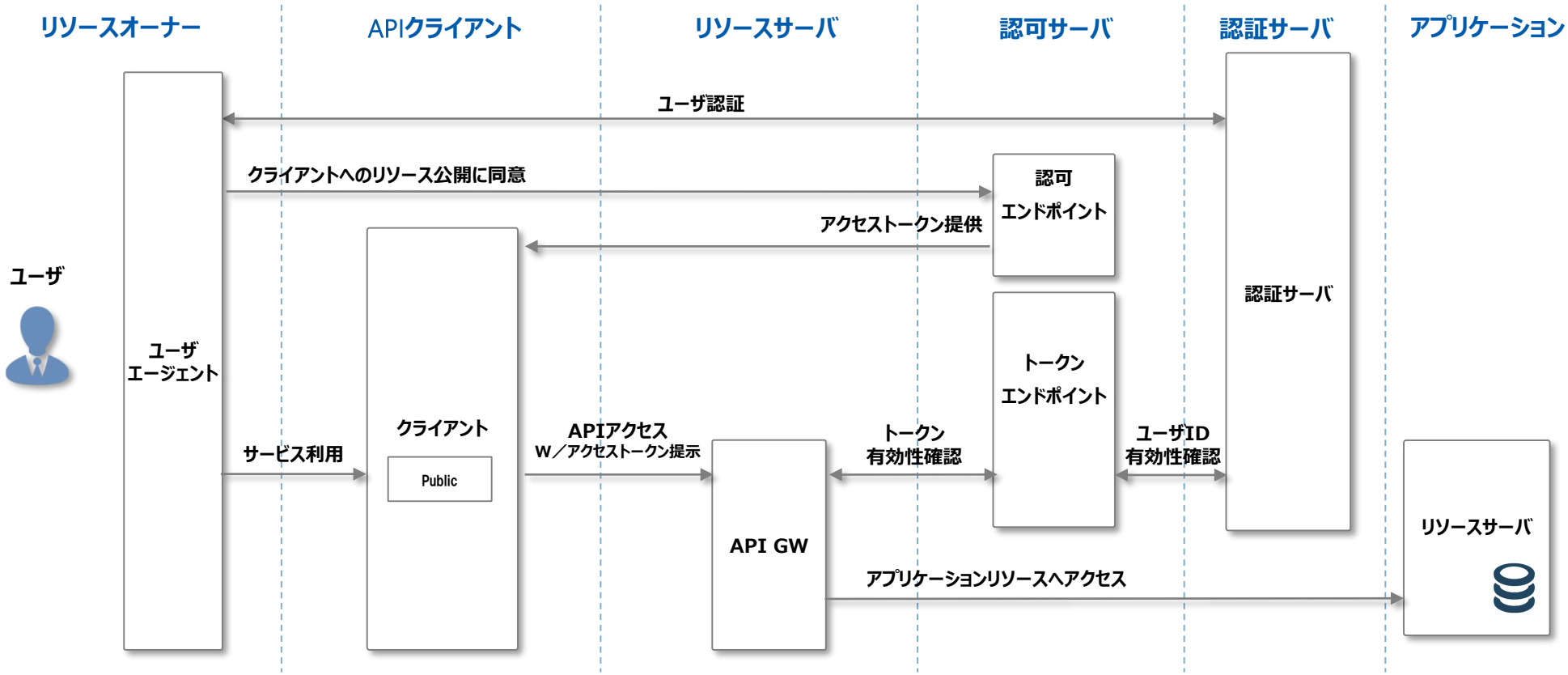
グラントタイプ

認可コードフロー



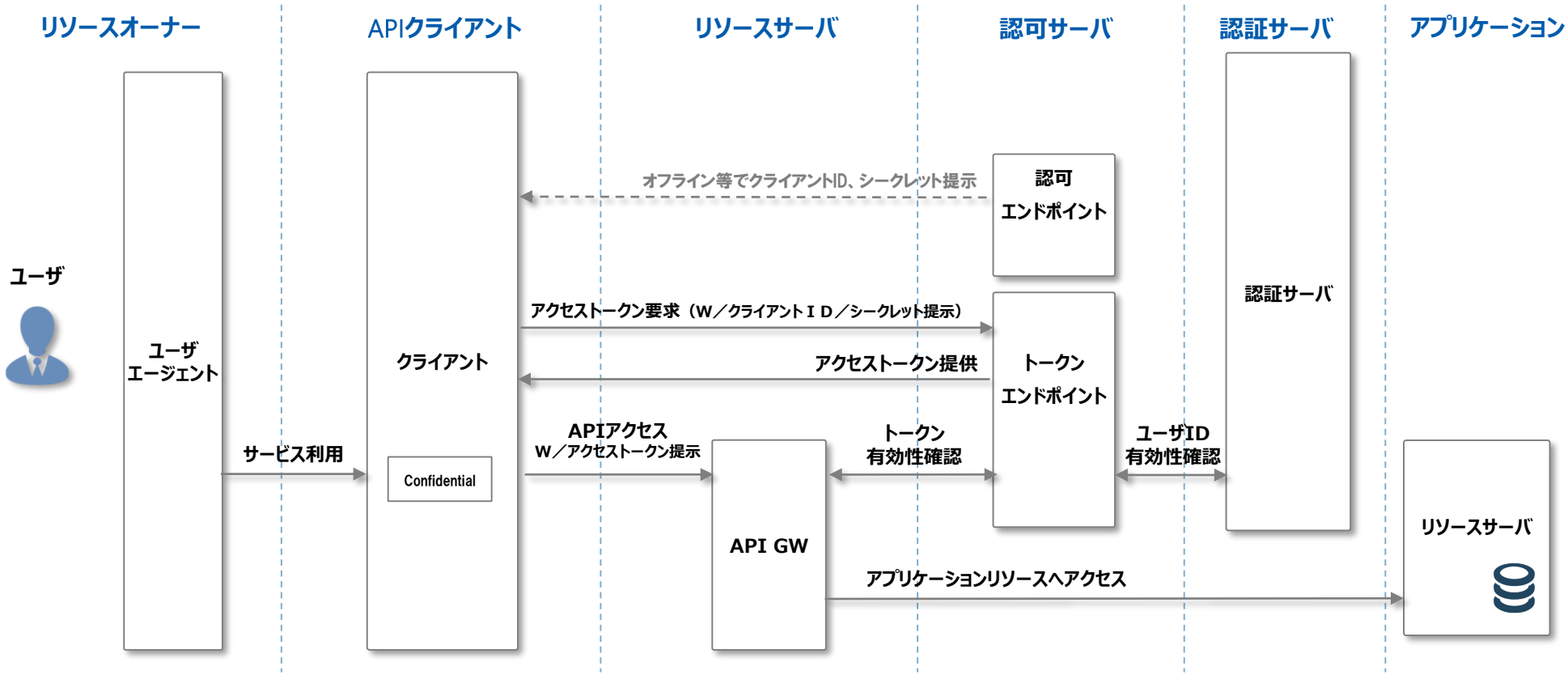
グラントタイプ

インプリシット



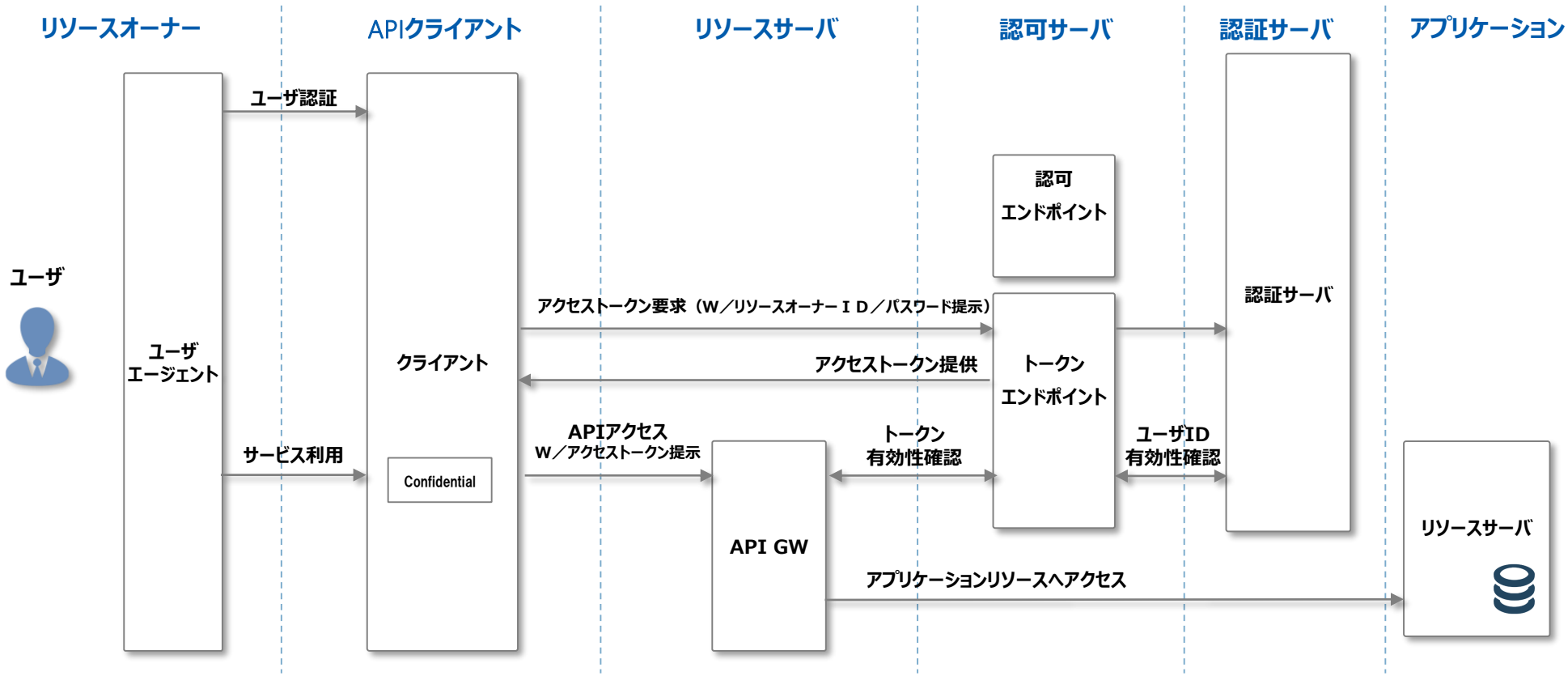
グラントタイプ

クライアントクレデンシャル



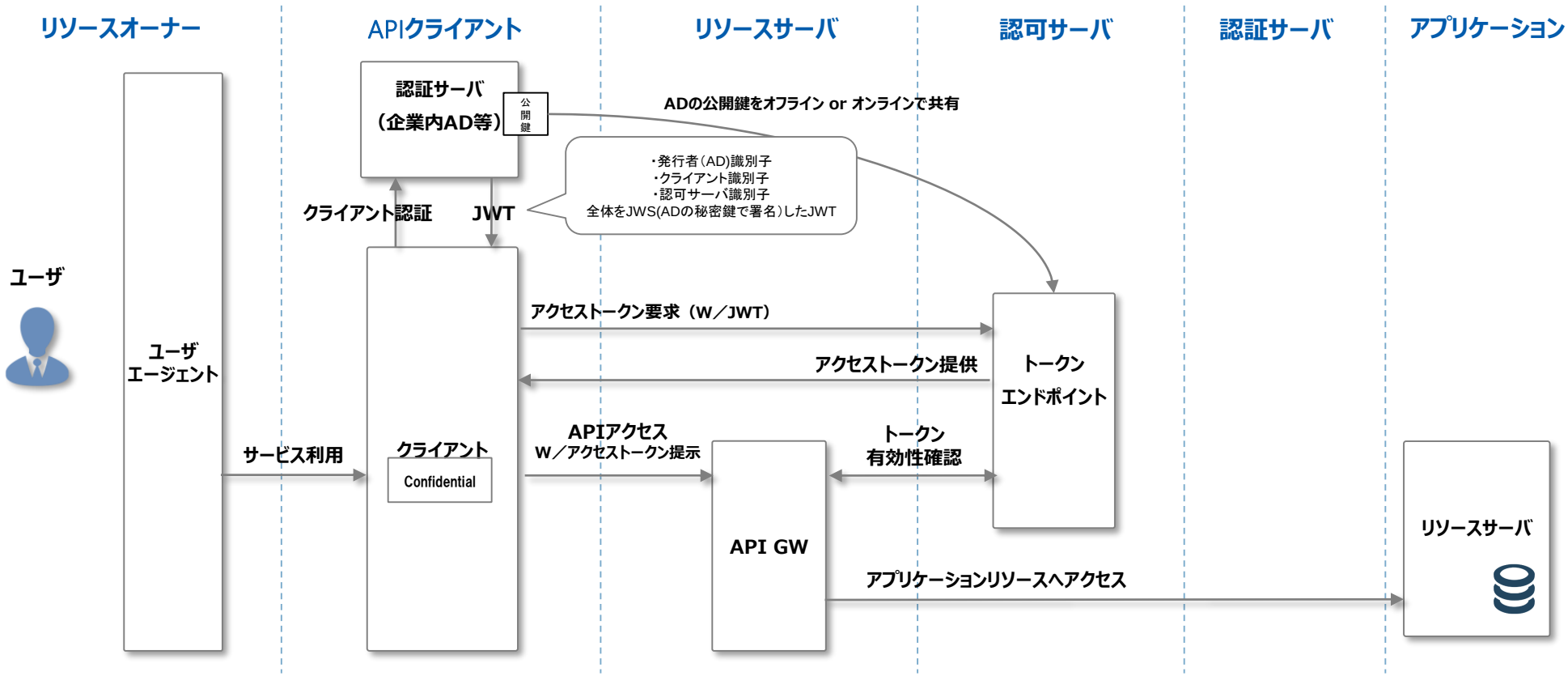
グラントタイプ

リソースオーナーパスワードクレデンシャル



グラントタイプ

アサーション

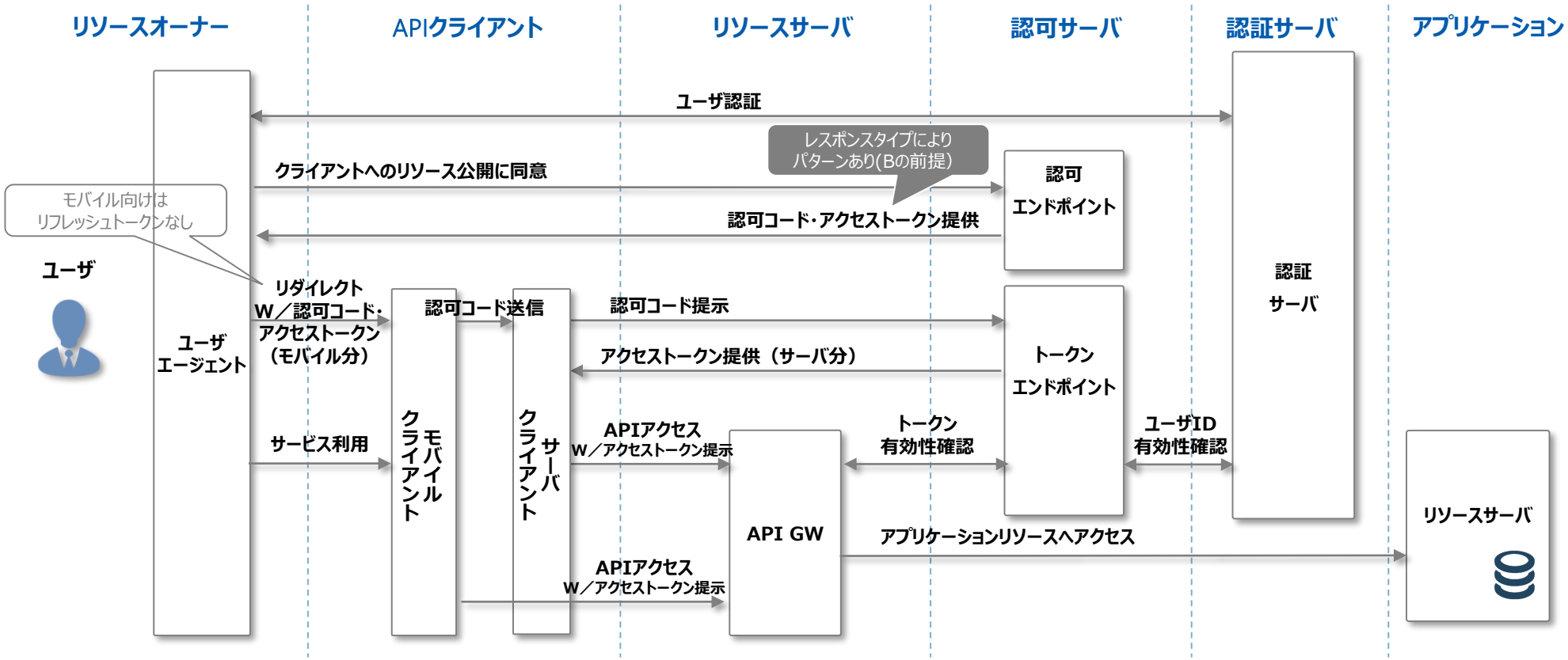


ハイブリッドグラントフロー（モバイルアプリケーション対策）

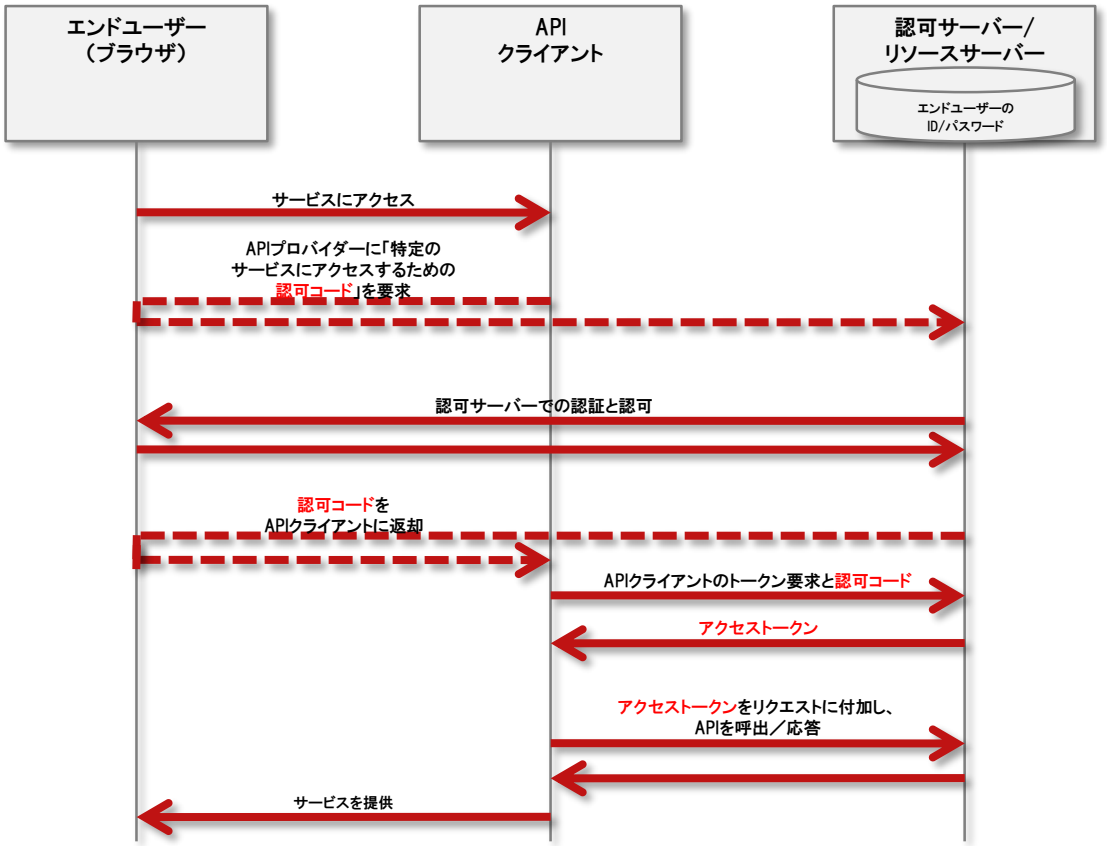
ハイブリッド

レスポンスタイプ	認可コード	アクセストークン	IDトークン
レスポンスタイプA	○		
レスポンスタイプB	○	○	
レスポンスタイプC	○		○
レスポンスタイプD	○	○	○

サーバもモバイルクライアントを同時に認可。モバイルクライアント向けには最小権限のアクセストークンを認可するという思想



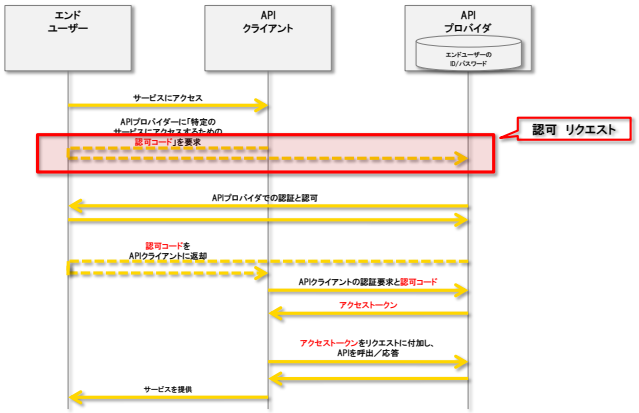
認可コード・グラント・フローのシーケンス例



認可リクエスト

■ 認可サーバへのクエリとして，次のパラメーターを付与する

パラメータ	項目	概要
response_type	必須	code ： 固定
cliend_id	必須	クライアント識別子
redirect_uri	任意	認可サーバからのリダイレクト先URI。 クライアントと認可サーバ間でリダイレクト先のURIが決められていない場合は必須
scope	任意	要求するアクセス権限のスコープを指定。
state	任意	リクエストとコールバックの間で状態を維持するために使用するランダムな値



■ クライアントはHTTPリダイレクト※によってリソースオーナーを認可サーバのURIに送る

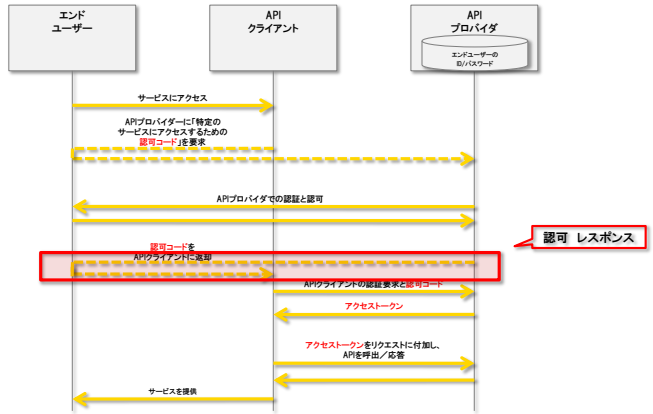
※HTTPリダイレクトに限定されていない

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

認可レスポンス（正常系）

■ 認可サーバはクエリコンポーネントに次のパラメーターを付与する

パラメータ	項目	概要
code	必須	認可サーバにより発行される認可コード。 有効期限は最大10分（推奨）で1回限り有効
state	任意	認可リクエストに含まれていた場合は必須。 クライアントから受け取った値をそのまま返却。



■ 認可サーバはHTTPリダイレクト※によってリソースオーナーをクライアントのURIに送る

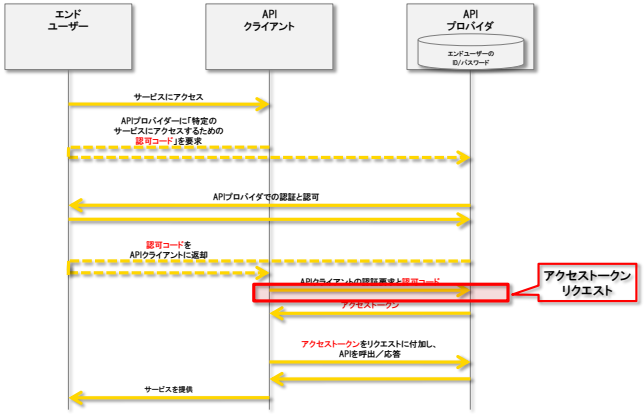
※HTTPリダイレクトに限定されていない

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=SplxlOBeZQQYbYS6WxSbIA &state=xyz
```

アクセストークンリクエスト

■ 認可サーバへのクエリとして，次のパラメーターを付与する

パラメータ	項目	概要
grant_type	必須	authorization_code ： 固定
code	必須	認可サーバから受け取った認可コード
redirect_uri	任意	認可リクエスト時に含まれていた場合は必須
client_id	任意	認可サーバによってクライアント認証されていない場合は必須
client_secret	原則	クライアント認証情報。但し他方式にてクライアント認証（相互TLS認証等）を実施している場合はその限りではない。



■ クライアントより認可サーバにリクエスト

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

client_id、client_secretをBASE64エンコードした値

アクセストークンの更新リクエスト

■ 認可サーバへのクエリとして，次のパラメーターを付与する

パラメータ	項目	概要
grant_type	必須	refresh_token ： 固定
Refresh_token	必須	認可サーバから受け取ったリフレッシュトークン
scope	任意	要求するアクセス権限のスコープを指定。 認可リクエスト時に含んでいない値は不可。
client_id	任意	認可サーバによってクライアント認証されていない場合は必須
client_secret	原則	クライアント認証情報。但し他方式にてクライアント認証（相互TLS認証等）を実施している場合はその限りではない。

■ クライアントより認可サーバにリクエスト

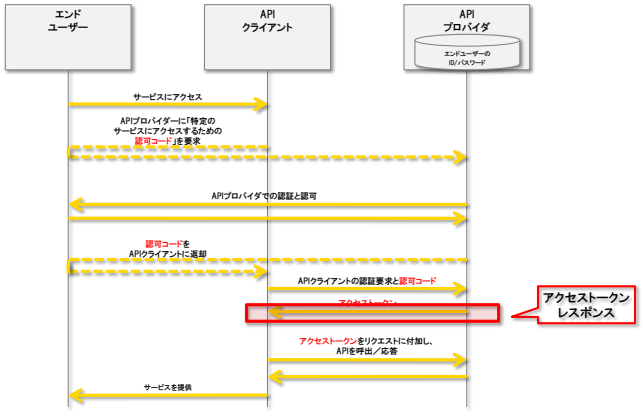
```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
```

client_id、client_secretをBASE64エンコードした値

アクセストークンレスポンス（正常系）

■ 認可サーバはクエリコンポーネントに次のパラメーターを付与する

パラメータ	項目	概要
access_token	必須	認可サーバが発行するアクセストークン
token_type	必須	トークンのタイプ
expires_in	推奨	アクセストークンの有効期限（期限までの絶対秒数）
refresh_token	任意	同じ認可グラントを用いて新しいアクセストークンを取得する際に利用される
scope	任意	クライアントから同一のスコープ要求された場合は任意。その他は必須



■ 認可サーバよりクライアントに HTTPステータスコード 200（OK）を返す。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
 "access_token":"2YotnFZFEjr1zCsicMWpAA",
 "token_type":"example",
 "expires_in":3600,
 "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
 "example_parameter":"example_value"
}
```

アクセストークンの利用（Bearerトークン）と授受の仕様

- Authorizationヘッダに入れる

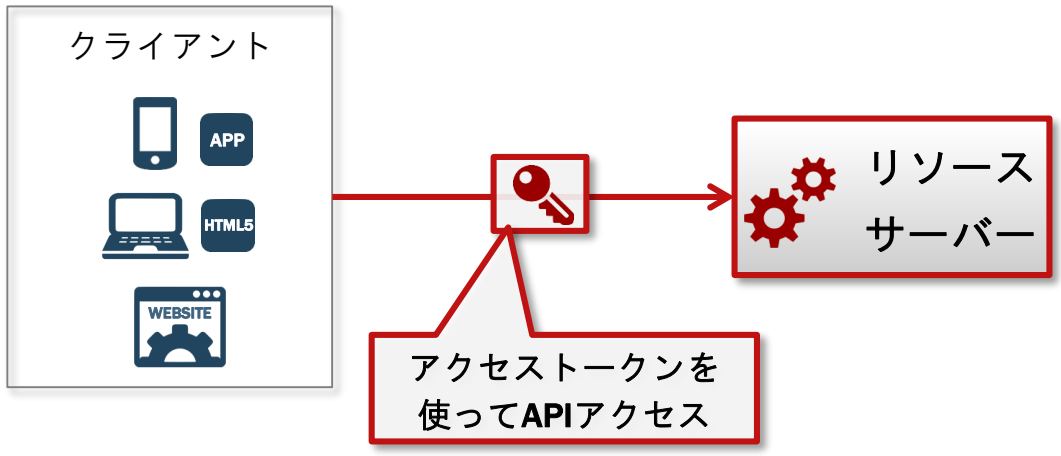
```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

- ボディに入れる

```
POST /resource HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-
urlencoded

access_token=mF_9.B5f-4.1JqM
```

- クエリパラメーターに入れる

```
https://server.example.com/resource?
access_token=mF_9.B5f-4.1JqM&p=q
```



RFCに関するセキュリティ考慮事項の全体像（詳細リストはエクセル別紙に記載）

RFC 6819 の OAuth 2.0 脅威モデルをベースに、同 RFC にてスコープ外となっている関連仕様のSecurity Considerations と、発行後に発見された脅威/対策を踏まえる。尚、本ガイドラインはOpenID Connect及びJWT関連仕様は対象外としている。

■ **RFC 6819: OAuth 2.0 Threat Model and Security Considerations (2013年1月) の脅威モデル**
- クライアント
- 認可エンドポイント
- トークン・エンドポイント
- 認可コード・グラント
- インプリシット・グラント
- リソース・オーナー・パスワード・クレデンシャル
- クライアント・クレデンシャル
- アクセス・トークンのリフレッシュ
- 保護リソースへのアクセス

■ **関連仕様**
- RFC 7009: OAuth 2.0 Token Revocation (2013年8月)
- OpenID Connect Core 1.0 incorporating errata set 1 (2014年11月) ※
- RFC 7519: JSON Web Token (JWT) (2015年5月) ※
- RFC 7521: Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants (2015年5月)
- RFC 7591: OAuth 2.0 Dynamic Client Registration Protocol (2015年7月) ※
- RFC 7636: Proof Key for Code Exchange by OAuth Public Clients (2015年9月)
- RFC 7662: OAuth 2.0 Token Introspection (2015年10月)
- RFC 7800: Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs) (2016年4月)
- OpenID Connect Session Management 1.0 - draft 27 (2016年8月) ※
- OpenID Connect Front-Channel Logout 1.0 - draft 01 (2016年8月) ※

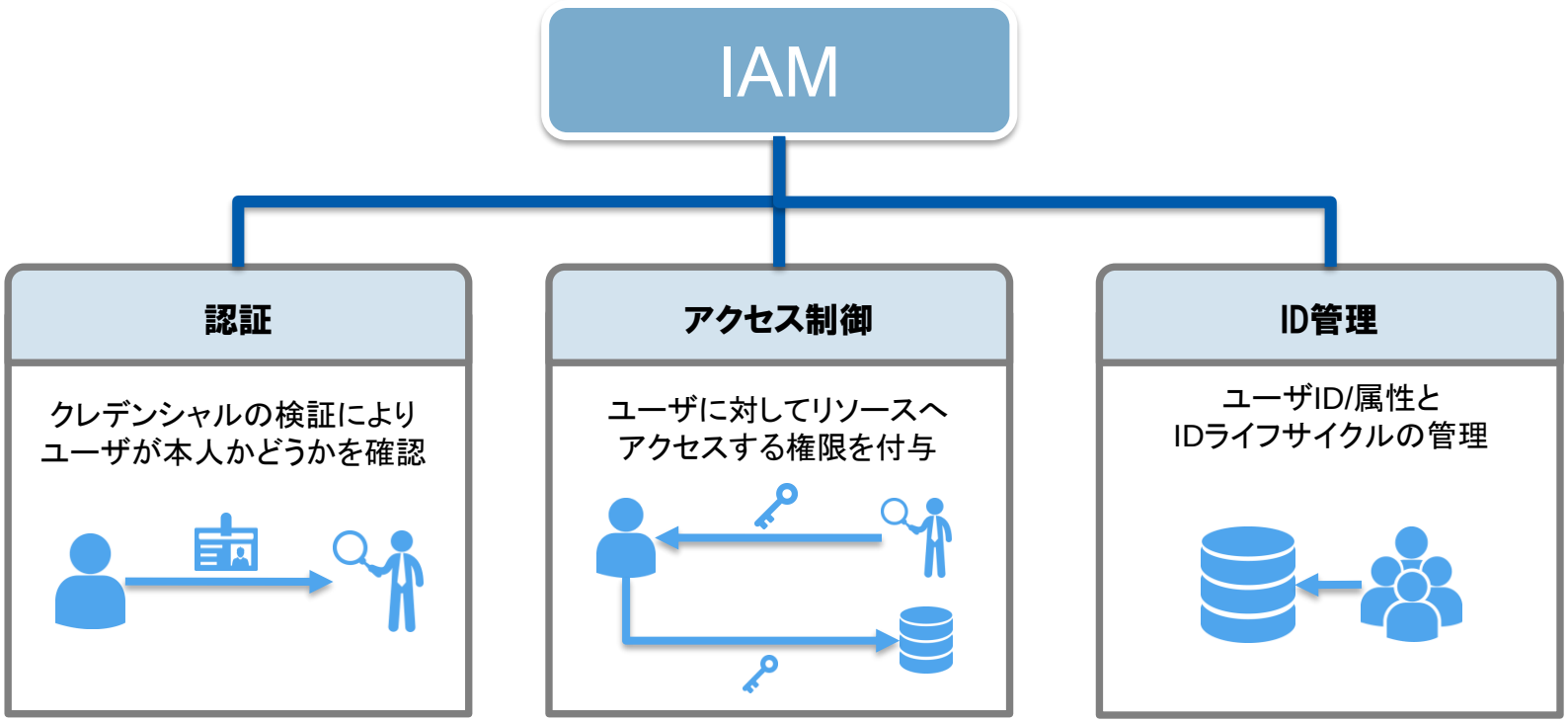
■ **発行後に発見された脅威/対策**
- draft-ietf-oauth-mix-up-mitigation-01: OAuth 2.0 Mix-Up Mitigation (2016年7月)

※ガイドラインの対象外

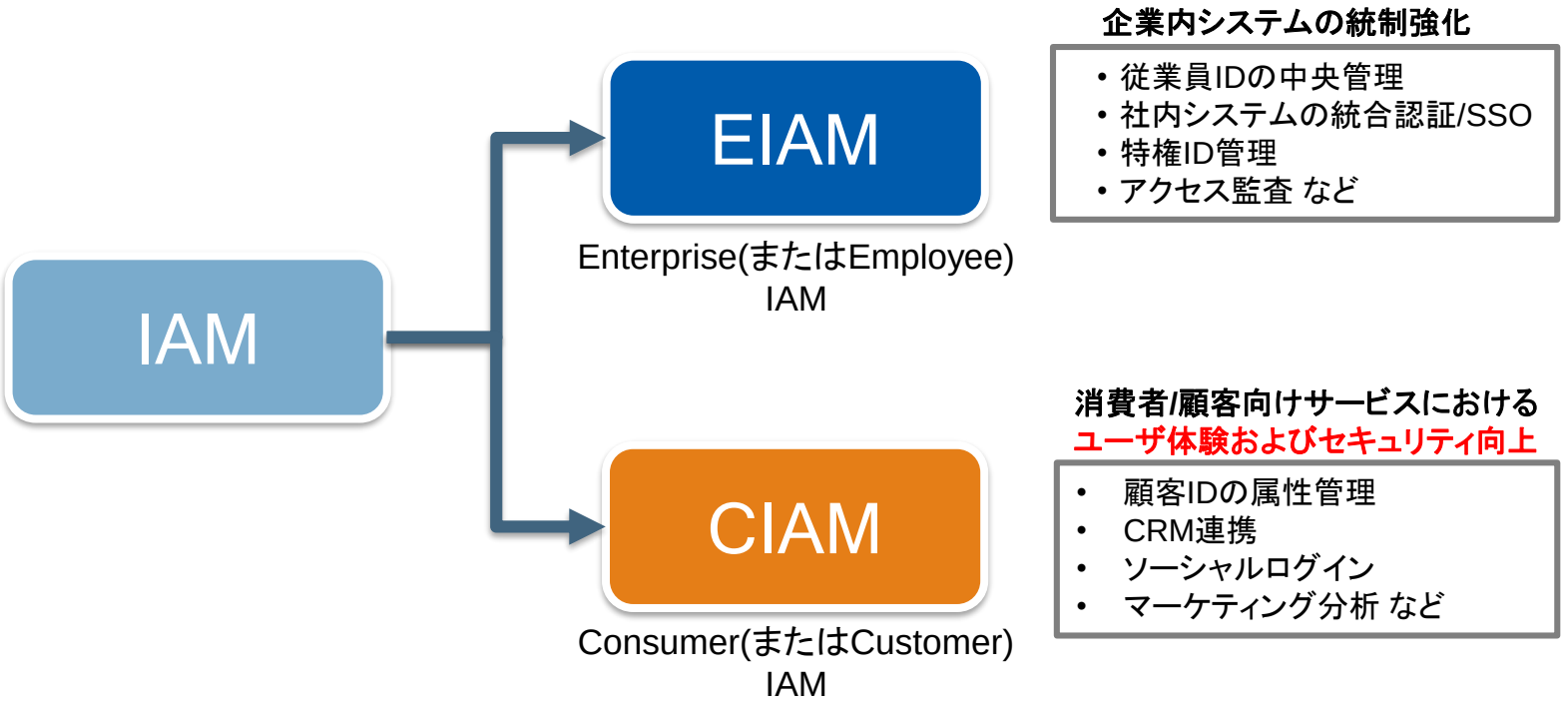
アイデンティティ＆アクセスマネジメント（Identity and Access Management：IAM）の定義と分類

アイデンティティ＆アクセスマネジメント(Identity and Access Management:IAM)とは

ユーザーを認証し、各種データ・機能へのアクセスをコントロールする考え方。
認証・アクセス制御・ライフサイクル管理などを指す。

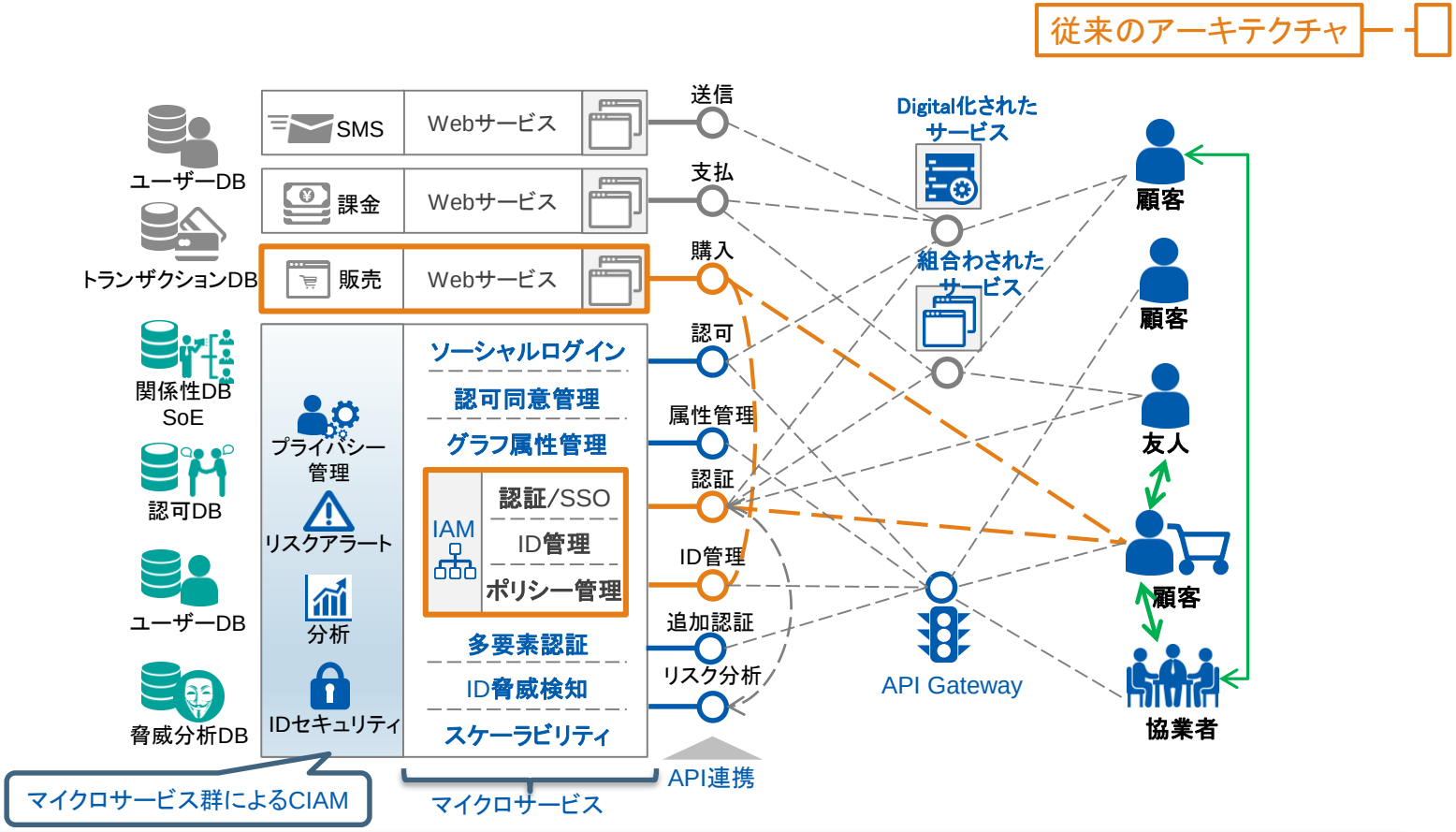


Enterprise IAM（EIAM）とConsumer IAM（CIAM）



IAMの進化は何をもたらすのか

CIAMで再構築したToBeアーキテクチャ



従来は、IAM＋個別Webサービスがモノリシックな状態で提供されていたが、分散アーキテクチャとIAM技術の進化によって、より高度かつセキュアなサービスが提供可能に


NRI
未来創発
Dream up the future.