ORACLE

マイクロサービスの認証・認可とJWT

Oracle Cloud Hangout Cafe – Season 4 #4

Shuhei Kawamura
Oracle Corporation Japan
July 7th, 2021

自己紹介

- 川村 修平(Shuhei Kawamura)
- 日本オラクル株式会社 (2020/10/15~)
  - ソリューション・アーキテクト本部
- 前職
  - SIerで社内FW(Java)の開発、保守
- 趣味
  - 音楽/料理/サウナ

@shukawam

shukawam

shukawam

Agenda

1. 認証・認可のおさらい
2. Javaで実現するマイクロサービスの認証・認可
3. JSON Web Token(JWT) とその周辺仕様
4. OAuth 2.0/OpenID Connect 1.0
5. MicroProfile – JWT Propagation & デモ
6. まとめ

Agenda

1. **認証・認可のおさらい**
2. Javaで実現するマイクロサービスの認証・認可
3. JSON Web Token(JWT) とその周辺仕様
4. OAuth 2.0/OpenID Connect 1.0
5. MicroProfile – JWT Propagation & デモ
6. まとめ

そもそも認証・認可とは？

認証 (Authentication/AuthN)

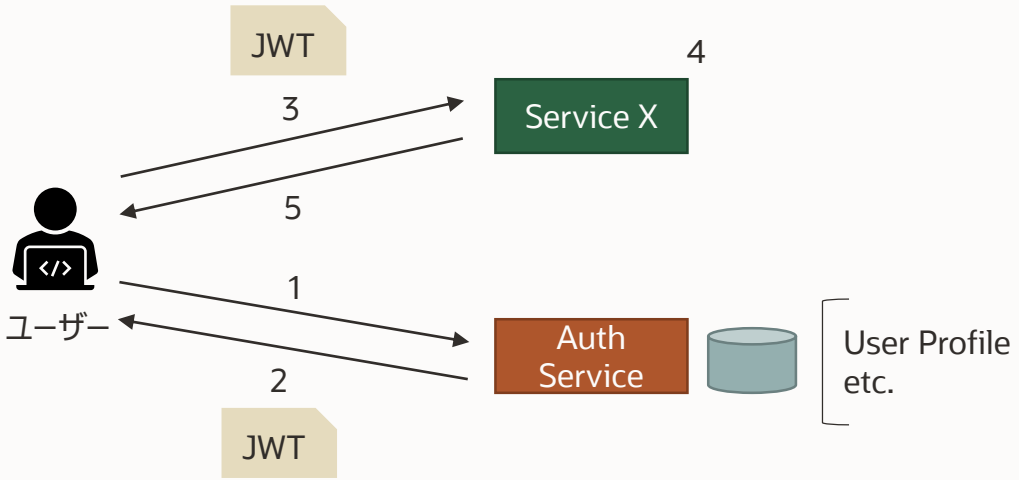
- ID/パスワードに代表される情報を用いてユーザー／システムの本人性を検証すること
- 認証の3要素
  - 記憶: What you know
  - 所有: What you have
  - 生体: What you are
- HTTP 401 Unauthorized - The request requires user authentication.
  - 本来ならUnauthenticatedであるべき…？

認可 (Authorization/AuthZ)

- ユーザー／システムに対して特定の処理を行うことを許可すること
- HTTP 403 Forbidden - The server understood the request, but is refusing to fulfill it.

一般的に広く使われている認証の方式

- JSON Web Tokenを用いたID連携のための標準プロトコルが存在する(OpenID Connect 1.0)
- コンパクトで自己完結型な表現方法がマイクロサービス・アーキテクチャとも好相性
  - 自己完結型のトークンを用いる事で認証サービスとその他サービス間を疎結合に保つ
  - 通信全体のトラフィック量削減が期待できる

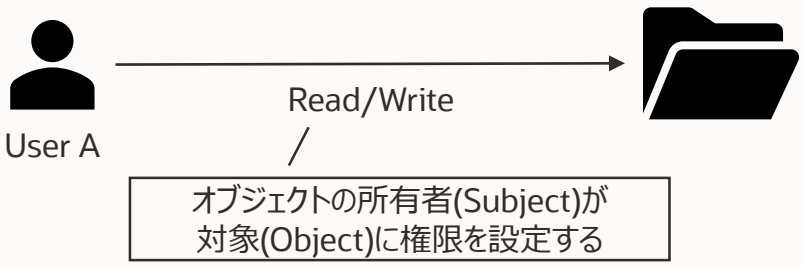


1. ユーザーが認証要求を行う
2. JWTを返却
3. JWTを業務処理を担当するサービスへ送信
4. JWTの検証
5. レスポンス

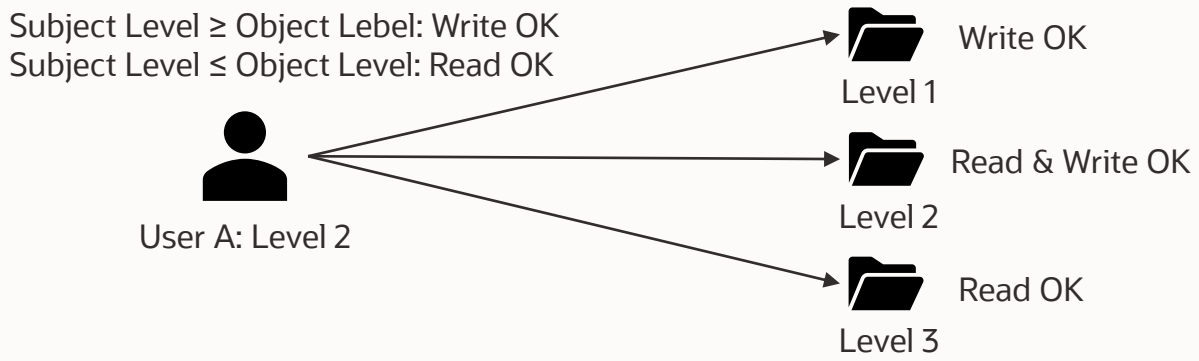
その他の方式等はこちらの論文から；
https://iopscience.iop.org/article/10.1088/1742-6596/910/1/012060

アクセス制御の基本方針
DAC, MAC, RBAC, ABAC

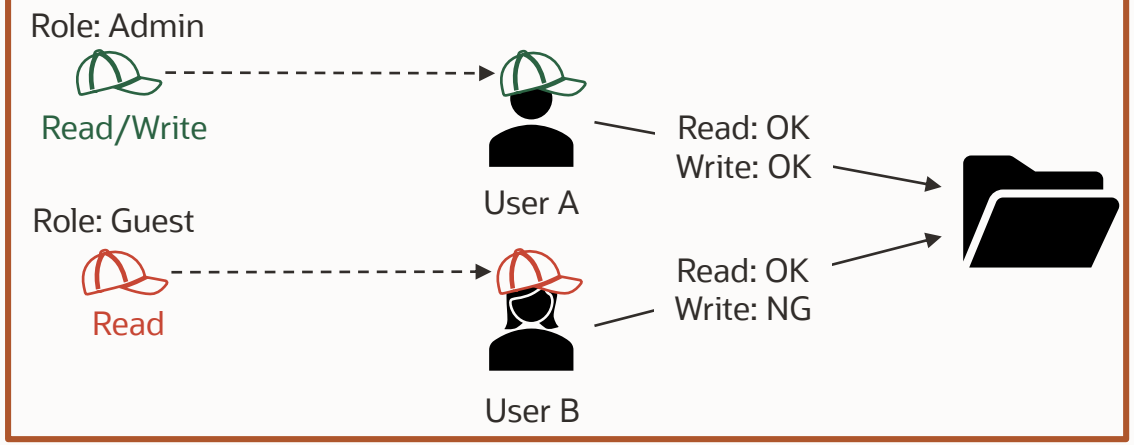
Directionary access control (DAC)



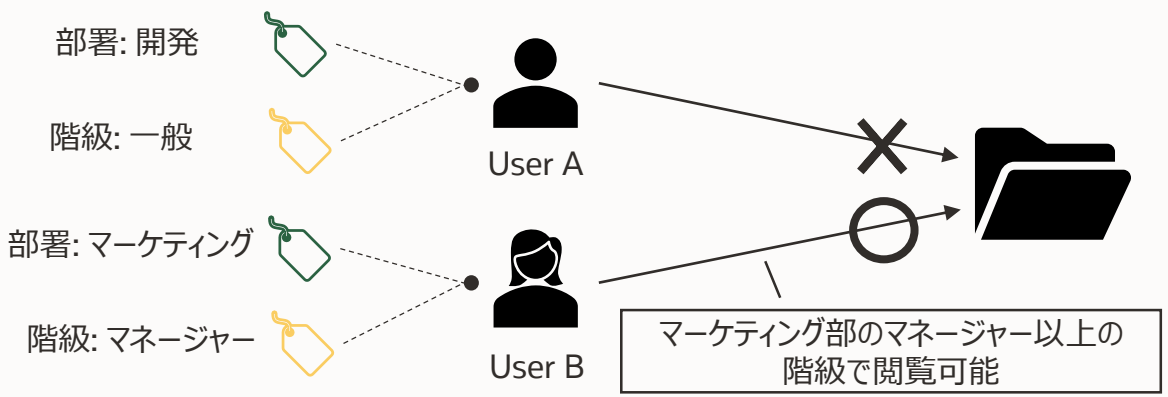
Mandatory access control (MAC)



Role-based access control (RBAC)



Attribute-based access control (ABAC)



Agenda

1. 認証・認可のおさらい
2. **Javaで実現するマイクロサービスの認証・認可**
3. JSON Web Token(JWT) とその周辺仕様
4. OAuth 2.0/OpenID Connect 1.0
5. MicroProfile – JWT Propagation & デモ
6. まとめ

Eclipse MicroProfile

https://microprofile.io

- マイクロサービス・アーキテクチャのためにEnterprise Javaを最適化することを目的としたコミュニティ仕様
- Java EE 7 のサブセットとしてスタートし、マイクロサービスの実装に有効な仕様を追加
- コミュニティドリブンで迅速なリリースを重視
- ベンダーに捉われず、移植性・相互運用性に焦点を当てた仕様
- 最新版: 4.0 (2021/07 時点)
  - 2021/07/07に4.1のリリースされるはず…

Open Tracing 2.0	Open API 2.0	REST Client 2.0	Config 2.0
Fault Tolerance 3.0	Metrics 3.0	JWT Propagation 1.2	Health 3.0
CDI 2.0	JSON-P 1.1	JAX-RS 2.1	JSON-B 1.0

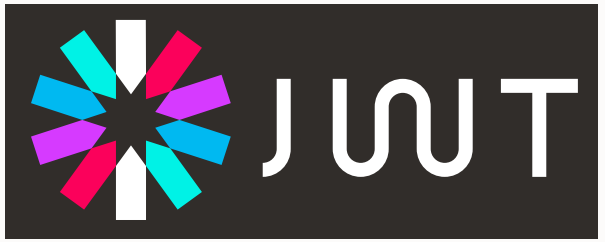
https://microprofile.io

Eclipse MicroProfile – JWT Propagation

This specification outlines a proposal for using OpenID Connect(OIDC) based JSON Web Tokens(JWT) for role based access control(RBAC) of microservice endpoints.

引用) https://download.eclipse.org/microprofile/microprofile-jwt-auth-1.2/microprofile-jwt-auth-spec-1.2.html#_introduction

本仕様は、OpenID Connect(OIDC)ベースのJSON Web Token(JWT)を用いて、マイクロサービスのエンドポイントのロールベースのアクセスコントロール(RBAC)を行うための提案をまとめたものです。



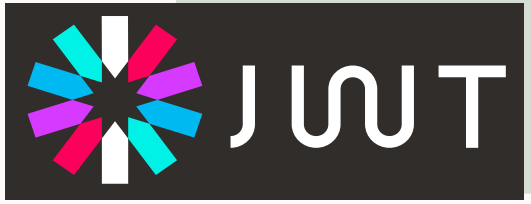
MicroProfile JWT Propagationを理解するためのロードマップ

JWT

OAuth 2.0

OpenID Connect 1.0

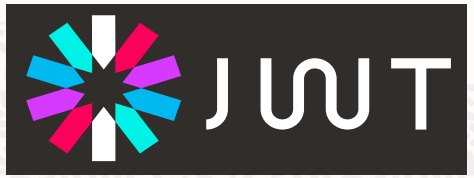
MP-JWT



Agenda

1. 認証・認可のおさらい
2. Javaで実現するマイクロサービスの認証・認可
3. **JSON Web Token(JWT) とその周辺仕様**
4. OAuth 2.0/OpenID Connect 1.0
5. MicroProfile – JWT Propagation & デモ
6. まとめ

JSON Web Token

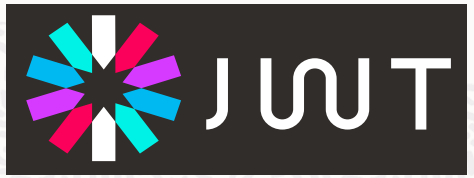


JSON Web Token

- jot(ジョット) と発音することが RFC 7519 で推奨されている
- スペースに制約のある環境を想定したコンパクトなクレームの表現方法
  - JSONをURL-safeな形でコンパクトにエンコードしたもの
  - コンパクトにするという目的で省略形が使われることが多い (e.g. audience → aud)
- JOSEと呼ばれる仕様群の一つ

JOSEと関連仕様

JOSE = JavaScript Object Signing and Encryption



JOSE

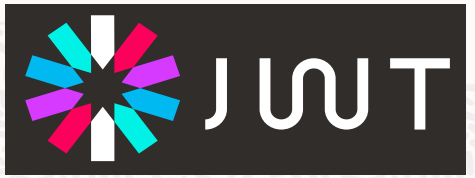
- 当事者間でクレームを安全に転送する方法を提供することを目的としたフレームワークで、この目的を満たすための仕様群を提供

関連仕様；

- JWS (JSON Web Signature) - RFC 7515
- JWE (JSON Web Encryption) - RFC 7516
- JWK (JSON Web Key) - RFC 7517
- JWA (JSON Web Algorithms) - RFC 7518
- JWT (JSON Web Token) - RFC 7519

JSON Web Signature(JWS)

RFC 7515



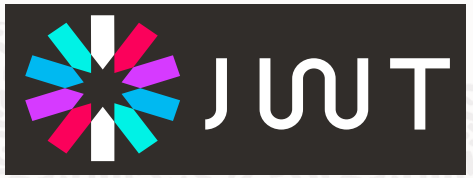
JSON Web Signature (JWS)は, JavaScript Object Notation (JSON)をベースとしたデータ構造を用いて, デジタル署名やMessage Authentication Codes (MACs)により保護されたコンテンツを表現するための手段である. 本仕様で使用する暗号アルゴリズムと識別子はJSON Web Algorithms (JWA) で述べられている. 関連する暗号化の機能は, JSON Web Encryption (JWE) で述べられている.

引用) https://tex2e.github.io/rfc-translater/html/rfc7515.html - RFC 7515 JSON Web Signature 日本語訳

- 各部は、Base64URLエンコードされて表現される
- RFC 7515 では、ペイロードがJSONであることは要求されていない (*an arbitrary sequence of octets*)

JSON Web Encryption(JWE)

RFC 7516



JSON Web Encryption（JWE）は、JSONベースのデータ構造を使用して暗号化されたコンテンツを表します。この仕様で使用する暗号化アルゴリズムと識別子は、別のJSON Web Algorithms（JWA）仕様と、その仕様で定義されているIANAレジストリで説明されています。関連するデジタル署名とメッセージ認証コード（MAC）機能については、別のJSON Web Signature（JWS）仕様で説明されています。

引用) https://tex2e.github.io/rfc-translater/html/rfc7516.html - RFC 7516 JSON Web Encryption 日本語訳

- 各部は、Base64URLエンコードされて表現される
- RFC 7516 では、暗号化の対象である平文がJSONであることは要求されていない

JSON Web Key(JWK)

RFC 7517

JSON Web Key（JWK）は、暗号鍵を表すJavaScript Object Notation（JSON）データ構造です。この仕様では、JWKのセットを表すJWK Set JSONデータ構造も定義されています。この仕様で使用する暗号化アルゴリズムと識別子は、別のJSON Web Algorithms（JWA）仕様と、その仕様で確立されたIANAレジストリで説明されています。

引用) https://tex2e.github.io/rfc-translater/html/rfc7517.html - RFC 7517 JSON Web Key 日本語訳

使用シーン；

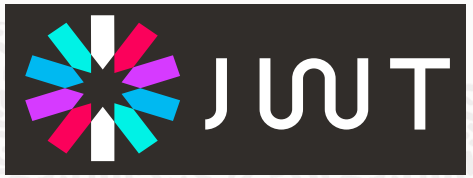
- JWSの署名を検証するための公開鍵表現
- JWEで非対称暗号アルゴリズムを用いた際の暗号化するための公開鍵表現

JSON Web Key Set (JWKS)

```
{
  “keys”: [
    {
      “kid”: “orange-1234”,
      “kty”: “RSA”,
      “n”: “sszb…”,
      “e”: “AQAB”
    },
    {
      “kid”: “orange-5678”,
      “kty”: “RSA”,
      …
    }
  ]
}
```

鍵のJSON表現 = JWK

JWS, JWEとJWT(+JWA)の関係

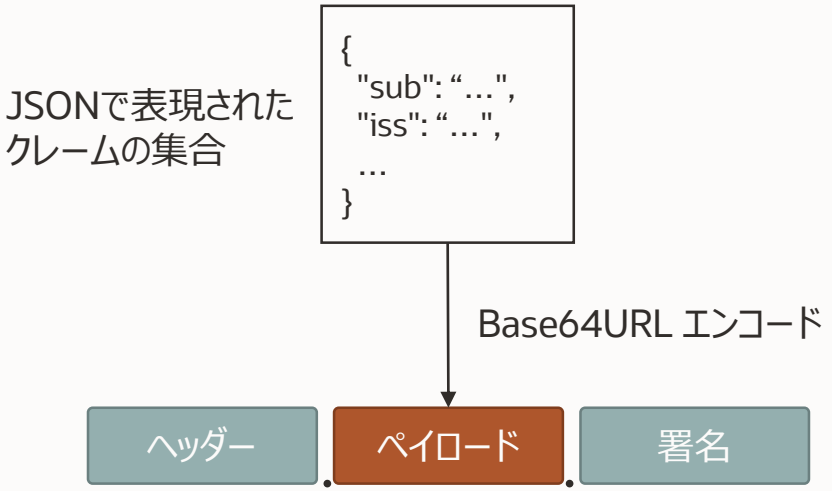


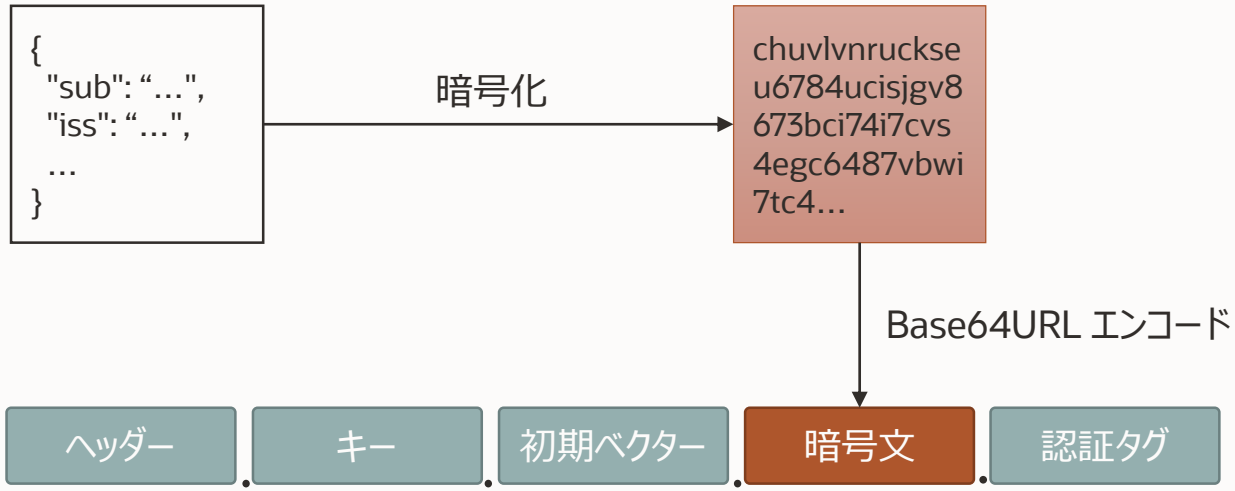
JWS形式のJWT

- JSONで表現されたクレームの集合をJWSの2番目のフィールドに埋め込んだもの
- 署名を作成する際に用いる暗号アルゴリズムがJWA(RFC 7518)に定義されている

JWE形式のJWT

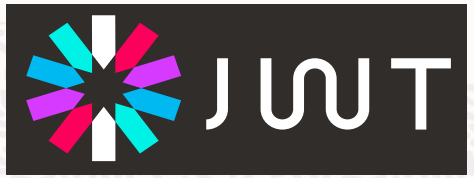
- JSONで表現されたクレームの集合を暗号化の後にJWEの4番目のフィールドに埋め込んだもの
- この時に用いる暗号化アルゴリズムがJWA(RFC 7518) に定義されている





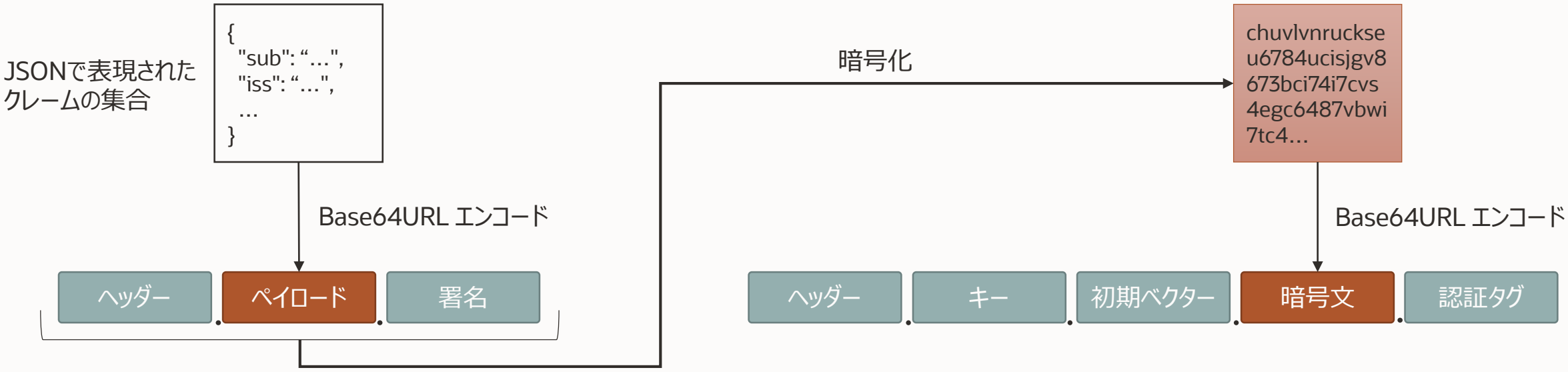
参考) https://qiita.com/TakahikoKawasaki/items/8f0e422c7edd2d220e06

Nested JWT



- 署名と暗号化の両方が行われたJWTの形式のこと
- 署名と暗号化の順番は問わないが、仕様によっては順番が定められている
  - e.g. IDトークン: 署名は必須で暗号化を実施する場合は、署名後に実施する

Nested JWT (JWSがJWEに含まれているパターン)



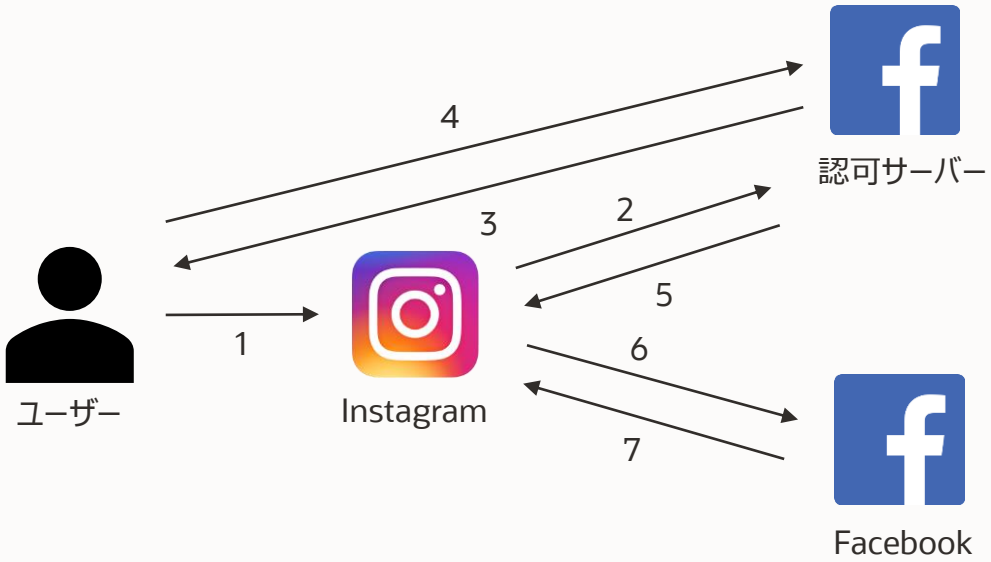
参考) https://qiita.com/TakahikoKawasaki/items/8f0e422c7edd2d220e06

Agenda

1. 認証・認可のおさらい
2. Javaで実現するマイクロサービスの認証・認可
3. JSON Web Token(JWT)とその周辺仕様
4. **OAuth 2.0/OpenID Connect 1.0**
5. MicroProfile – JWT Propagation & デモ
6. まとめ

OAuth 2.0 の概要

- 権限委譲／認可情報を扱うプロトコルで RFC 6749 に規定されている
  - 日本語訳 ⇒ http://openid-foundation-japan.github.io/rfc6749.ja.html
- 認証のためのプロトコルではない



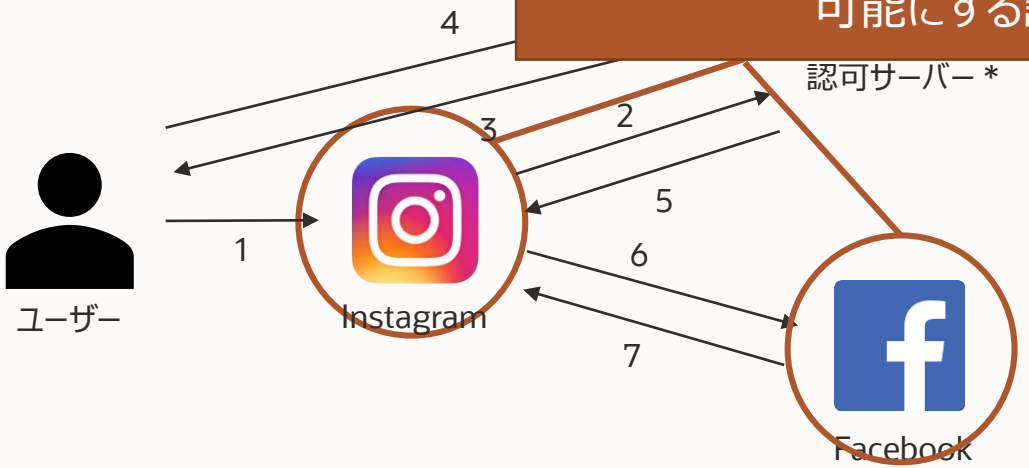
1. Instagramに画像の投稿を行う
2. Instagramが認可サーバー(Facebook)に対して、Facebookへのアクセス権を要求する
3. 認可サーバー(Facebook)は、「Facebookへのアクセス権をInstagramに委譲すること」についてユーザーの合意を求める
4. ユーザが合意する
5. 認可サーバー(Facebook)は権限が委譲された証明として、アクセストークンを発行し、Instagramへ渡す
6. Instagramは、発行されたアクセス権を持ってFacebookのAPI(投稿の自動連係)にアクセスする
7. アクセストークンの有効性を確認し、問題なければ当該APIのレスポンス結果を返却する

参考) OAuth、OAuth認証、OpenID Connectの違いを整理して、理解できる本

OAuth 2.0 の概要

- 権限委譲／認可情報を扱うプロトコルで RFC 6749 に規定されている
  - 日本語訳 ⇒ http://openid-foundation-japan.github.io/rfc6749.ja.html
- 認証のためのプロトコルではない

OAuth 2.0 は3rdパーティ・アプリケーションから HTTPサービスへの限定的なアクセスを 可能にする認可のフレームワーク



ook)に対して、Facebookへのア

5. 認可サーバー(Facebook)は、「Facebookへのアクセス権を Instagramに委譲すること」についてユーザーの合意を求める
4. ユーザが合意する
5. 認可サーバー(Facebook)は権限が委譲された証明として、アクセストークンを発行し、Instagramへ渡す
6. Instagramは、発行されたアクセス権を持ってFacebookのAPI(投稿の自動連係)にアクセスする
7. アクセストークンの有効性を確認し、問題なければ当該APIのレスポンス結果を返却する

参考) OAuth、OAuth認証、OpenID Connectの違いを整理して、理解できる本

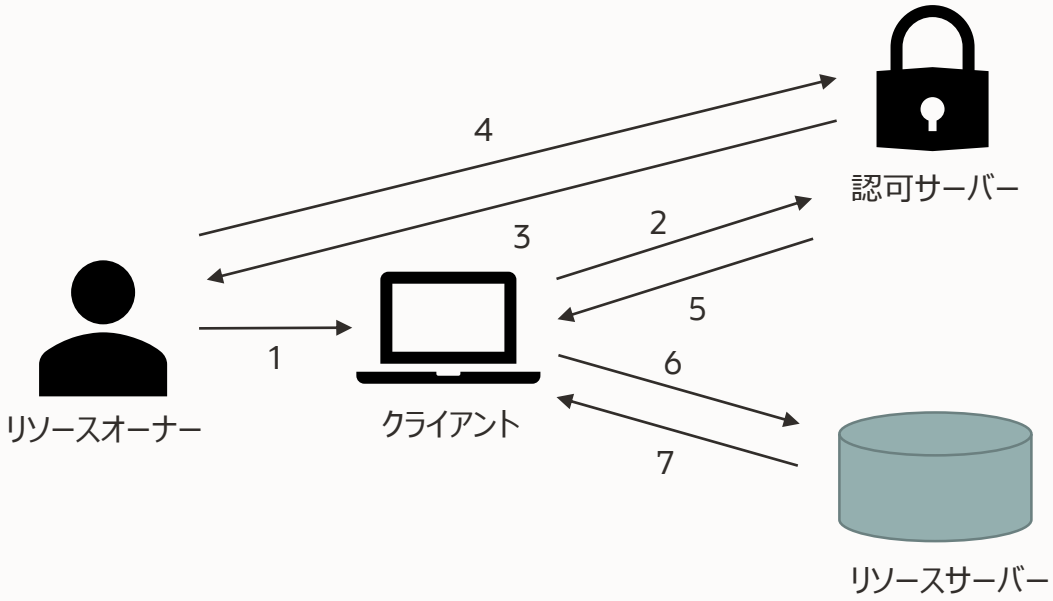
OAuth 2.0のアクター(登場人物)

リソースオーナー(Instagramのユーザー)
- リソースの所有者

リソースサーバー(Facebook API)
- Web API

クライアント(Instagram)
- リソースサーバーを利用するアプリケーション

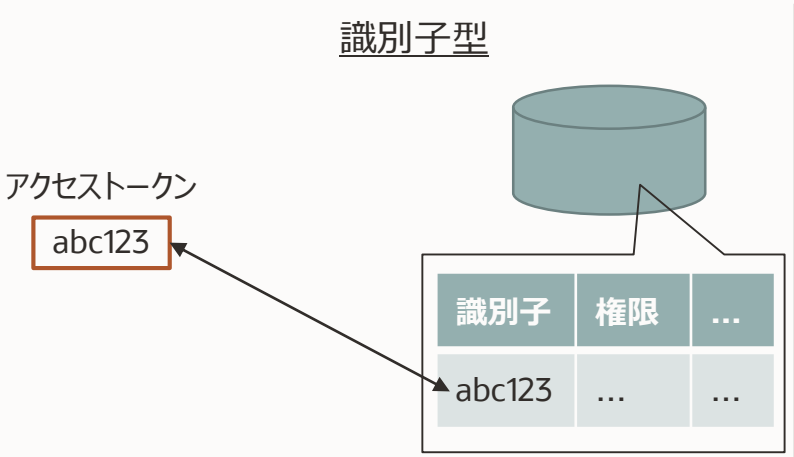
認可サーバー(Facebook)
- アクセストークンを発行するサーバー



参考) OAuth、OAuth認証、OpenID Connectの違いを整理して、理解できる本

アクセストークン

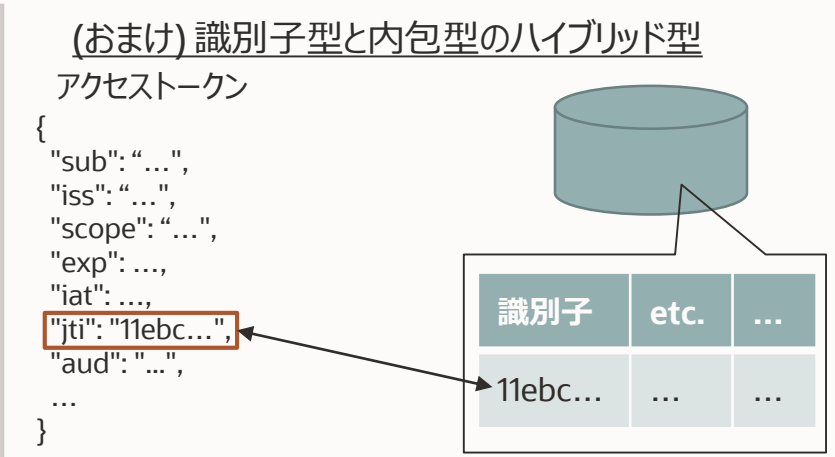
- 保護されたリソースにクライアントがアクセスするために必要な資格情報
  - 多くの場合はBearerトークンのため、ビルの入館証/電車の切符 etc. によく例えられる
    - e.g. 所持している**本人性は問わない**ため、その辺に落ちている入館カードで**ビルの特定エリア**に入れてしまう
- リソースサーバーに向けて発行される
- アクセストークンの実装方法は、RFC 6749 で定められていないが、大別すると以下の2種類
  - identifier (識別子型)
  - self-contain (内包型)



内包型

アクセストークン

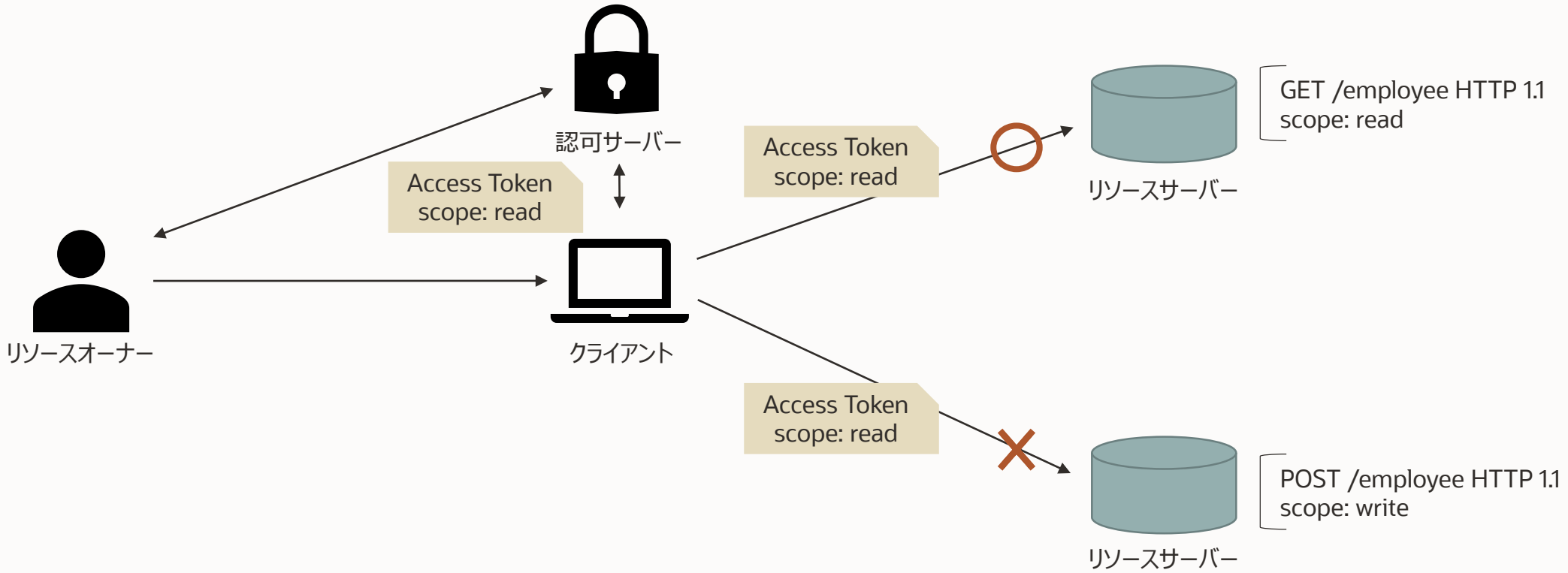
```
{
 "sub": "admin",
 "iss": "https://identity.oraclecloud.com/",
 "scope": "first_scope",
 "exp": 1623138587,
 "iat": 1623134987,
 "jti": "11ebc…",
 "aud": "http://localhost:7987/",
 …
}
```



参考) https://qiita.com/TakahikoKawasaki/items/970548727761f9e02bcd

スコープ

- アクセストークンに紐づくアクセス権を細かく制御するための仕様
- アクセストークンに紐づけたいスコープの指定は、クライアントから認可サーバへのリクエスト時に指定する (※後述)



クライアントタイプ

コンフィデンシャルクライアント

- クライアントシークレットを安全に保持できるクライアント
- サーバーサイドのWebアプリケーション

パブリッククライアント

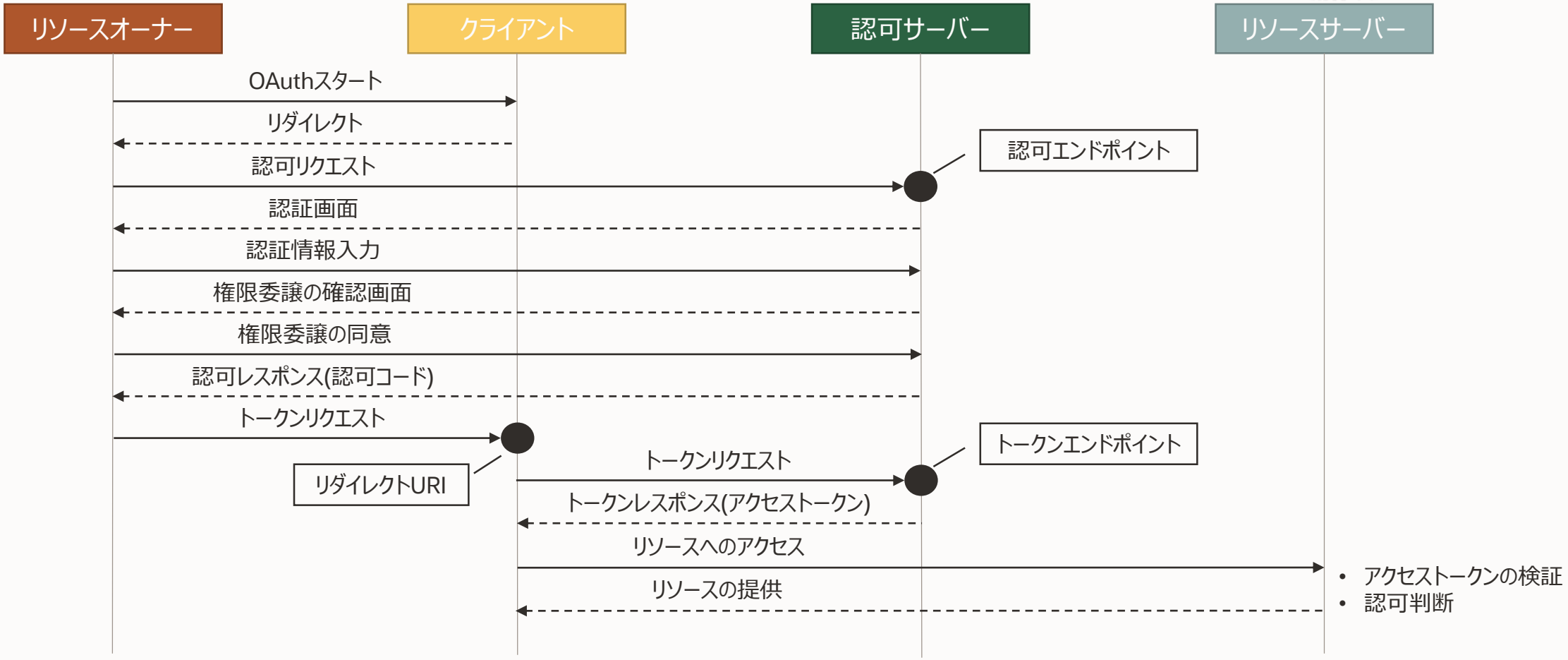
- クライアントシークレットを安全に保持できないクライアント
- ネイティブアプリやJavaScriptアプリケーション

グラントタイプ
アクセストークンの取得方法を定義

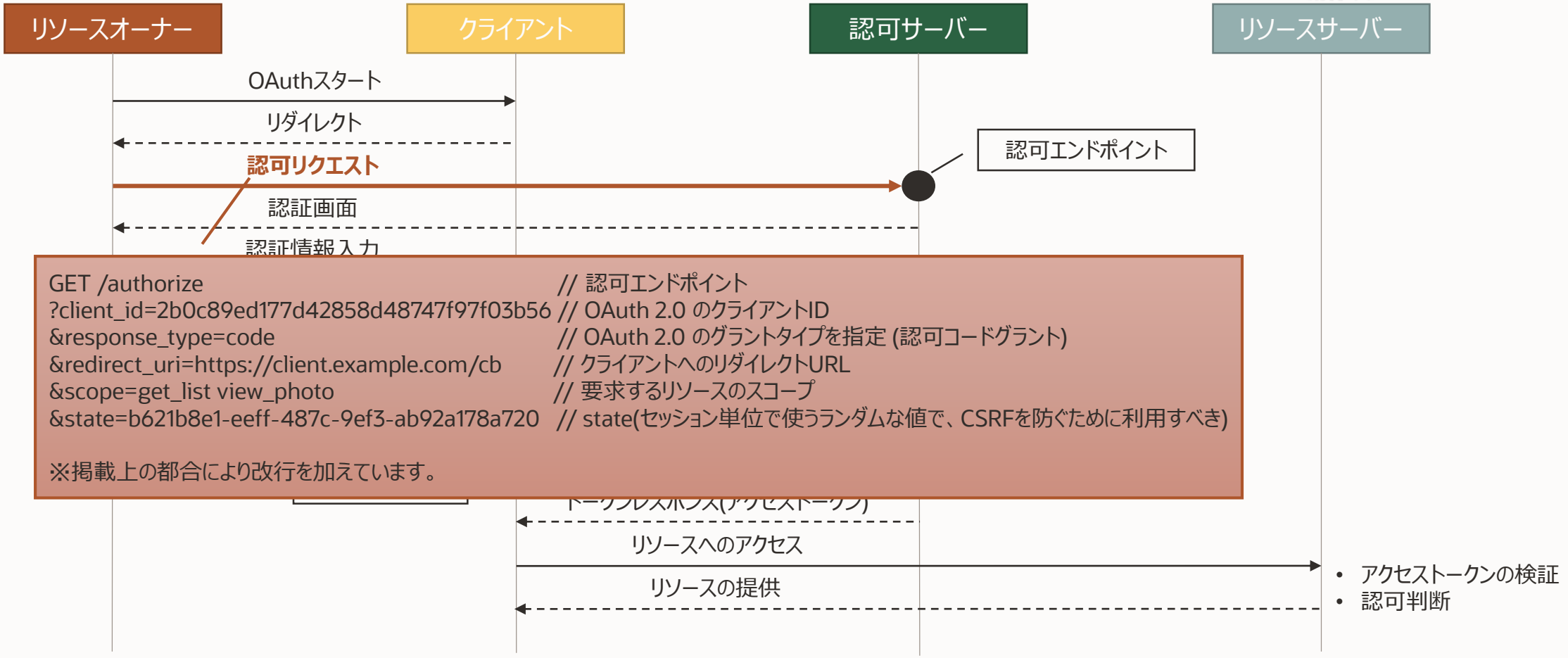
- 認可コードグラント
  - OAuth 2.0の基本フロー
  - 短命な認可コードを用いてアクセストークンと交換する
- インプリシットグラント(非推奨)
  - 認可コードを用いずに直接アクセストークンを受け取るフロー
  - SPAなどを想定したフローだが、現在は非推奨で認可コードグラント + PKCE[^1]が推奨されている
- リソースオーナーパスワードクレデンシャル(非推奨)
  - リソースオーナー(=エンドユーザー)のID/PWを受け取り、アクセストークンを発行するフロー
- クライアントクレデンシャルグラント
  - クライアント認証のみで、アクセストークンを発行するフロー
  - クライアントとリソースオーナーが同一の場合に使用される

[^1] PKCEの話やグラントタイプ全般に関する事は、OCHaCafe #5 避けては通れない認証・認可 をご参照ください

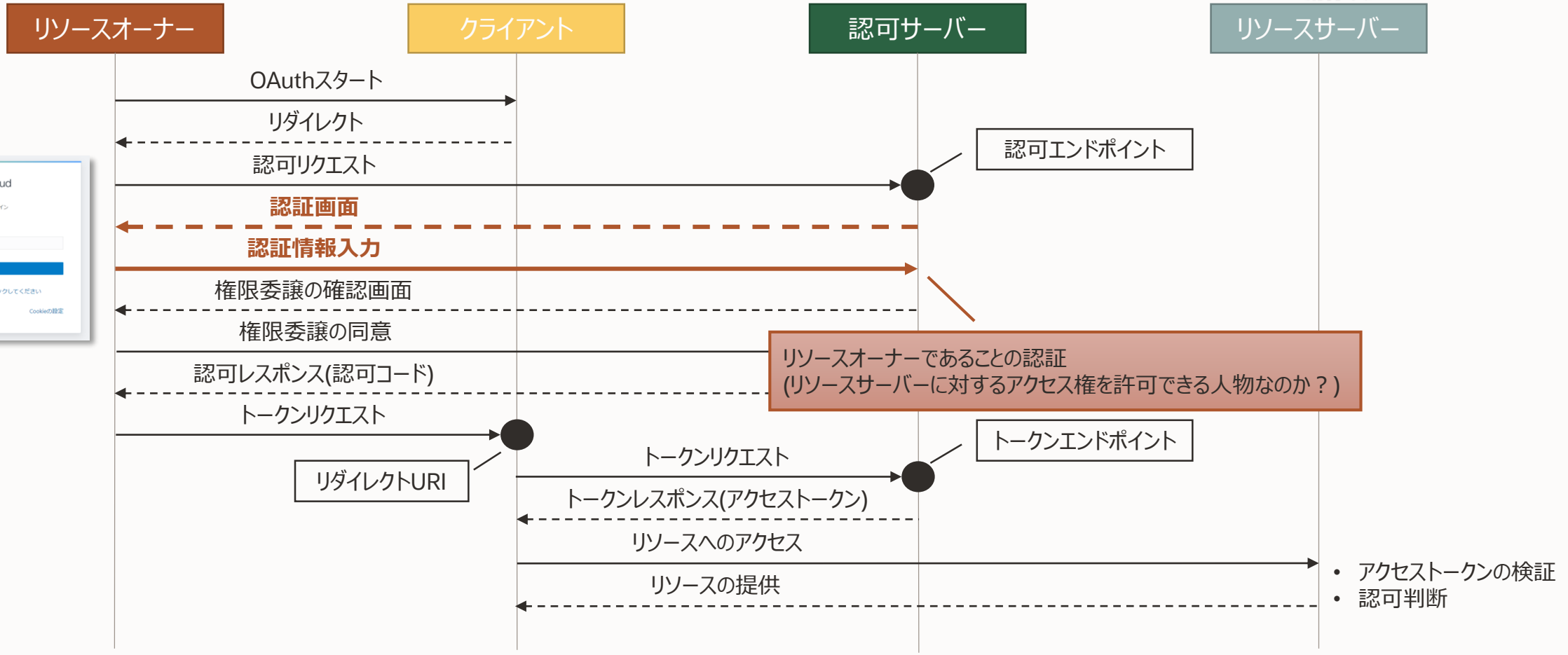
認可コードグラントによるアクセストークンの取得フロー



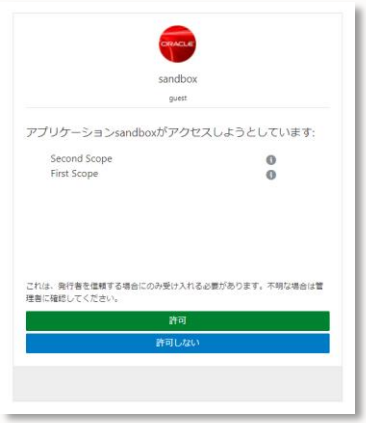
認可コードグラントによるアクセストークンの取得フロー

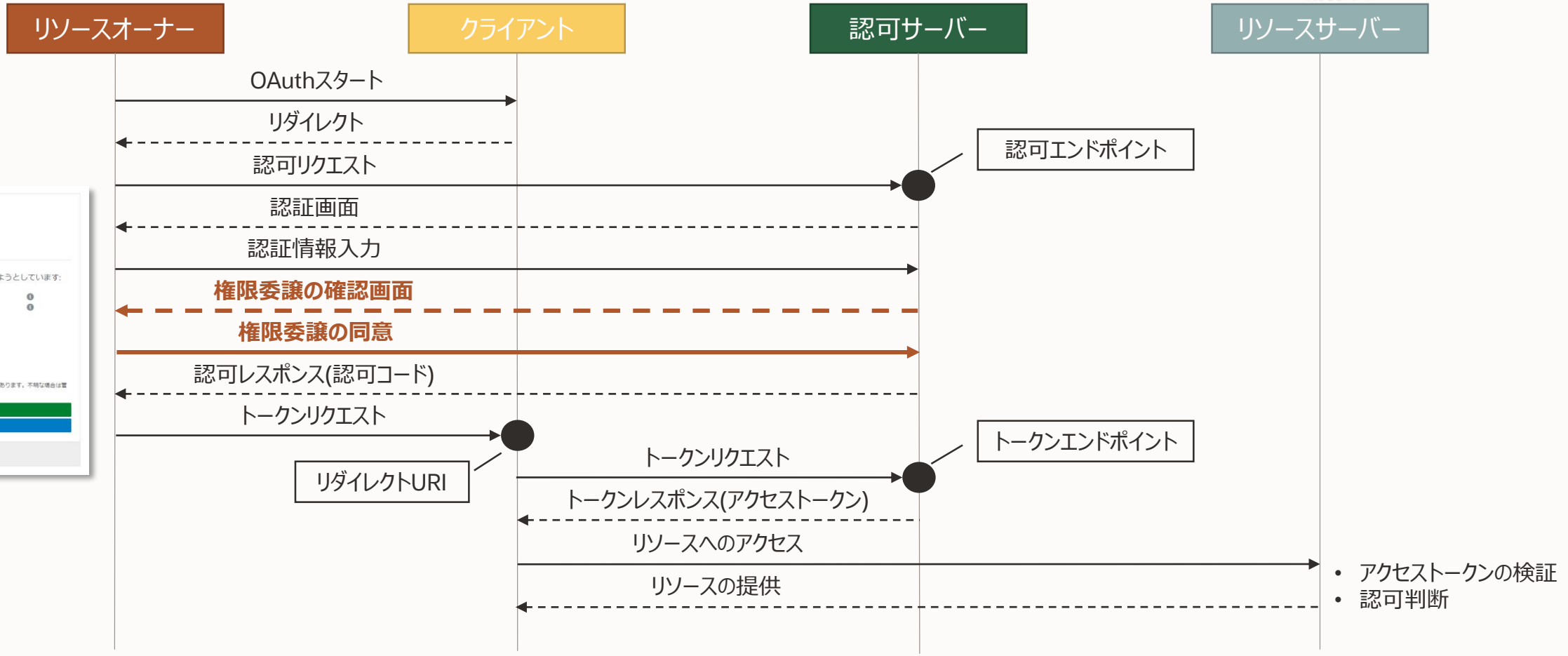


認可コードグラントによるアクセストークンの取得フロー

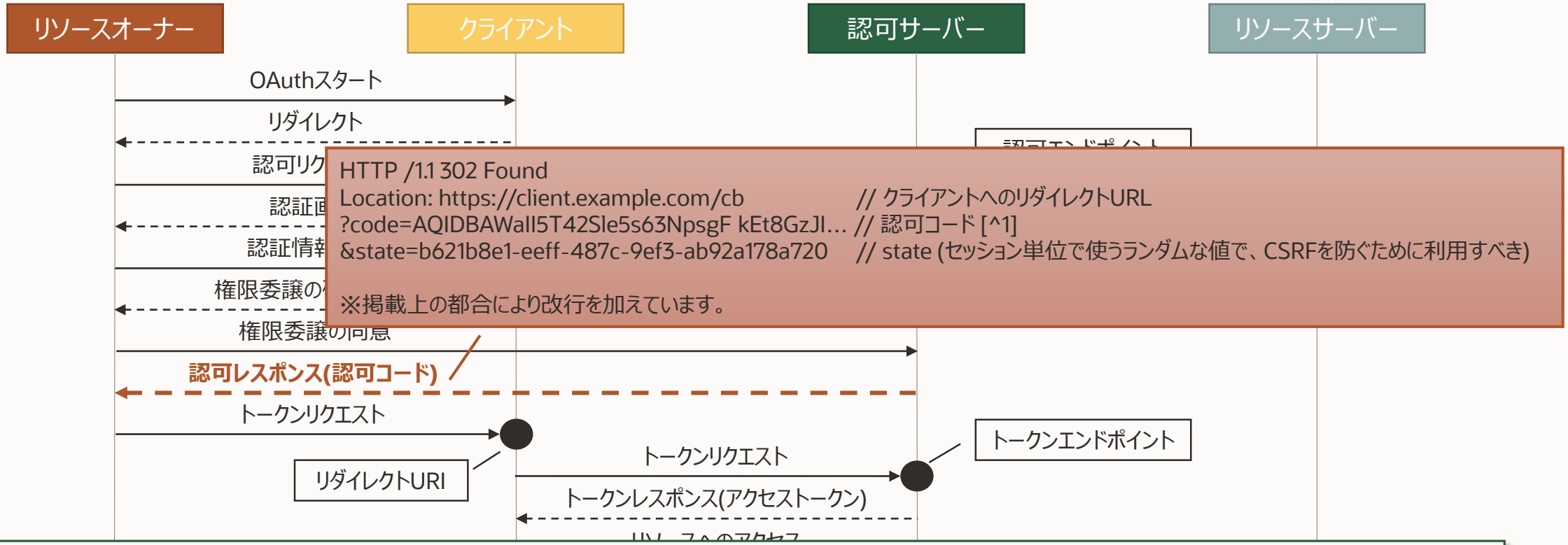


認可コードグラントによるアクセストークンの取得フロー



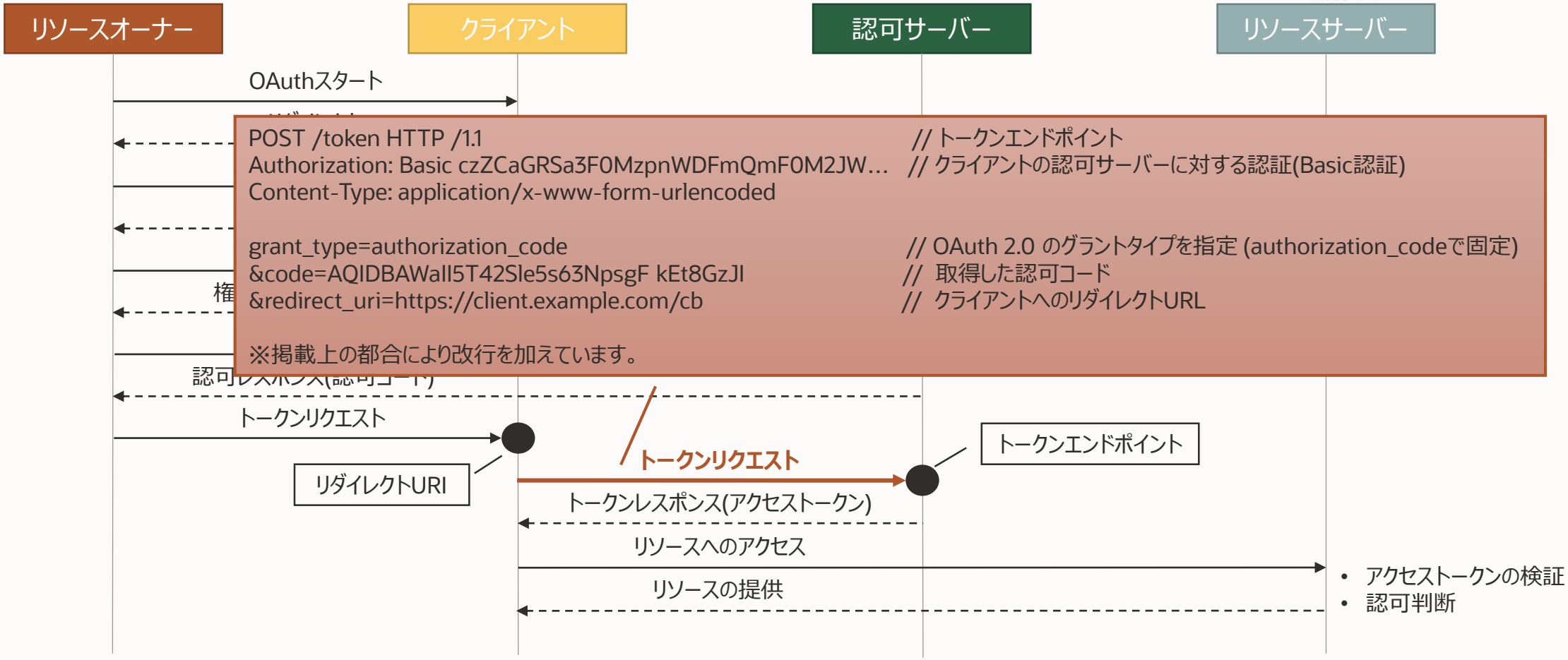


認可コードグラントによるアクセストークンの取得フロー

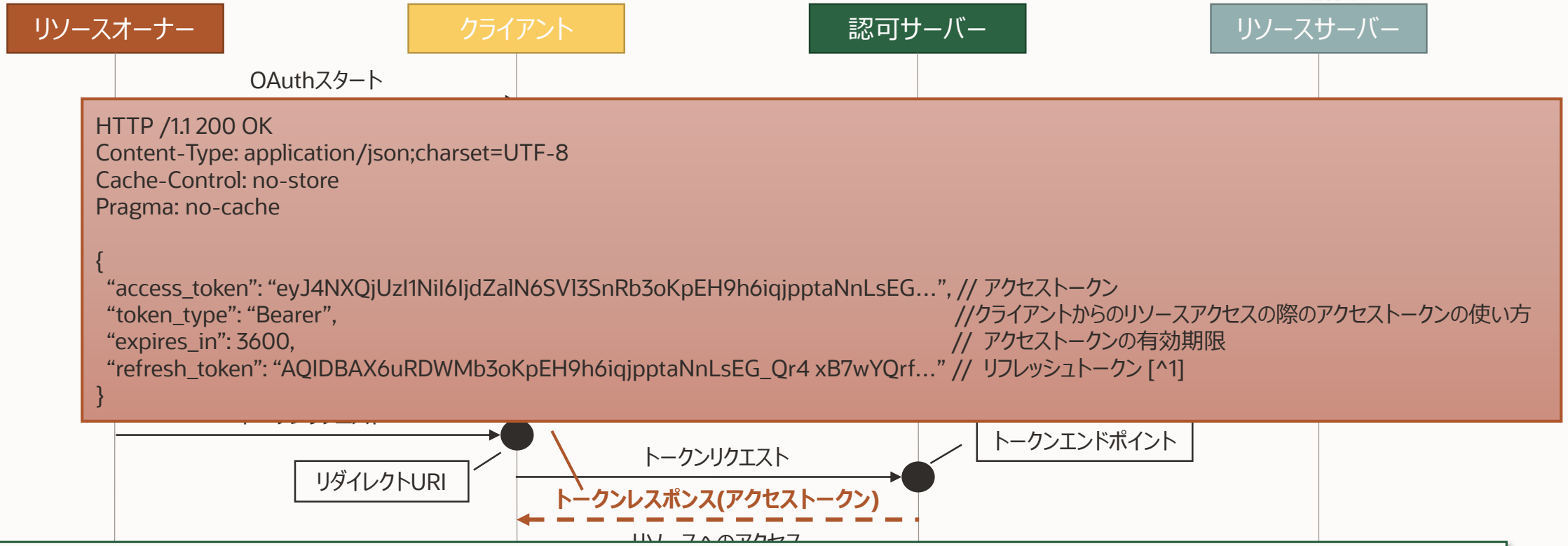


[^1] リソースオーナー(ブラウザ)を介さずにアクセストークンをクライアントが取得するために用いる一時コードのこと。
RFC 6749 では、有効期限を10分以内に設定することが推奨されている。

認可コードグラントによるアクセストークンの取得フロー

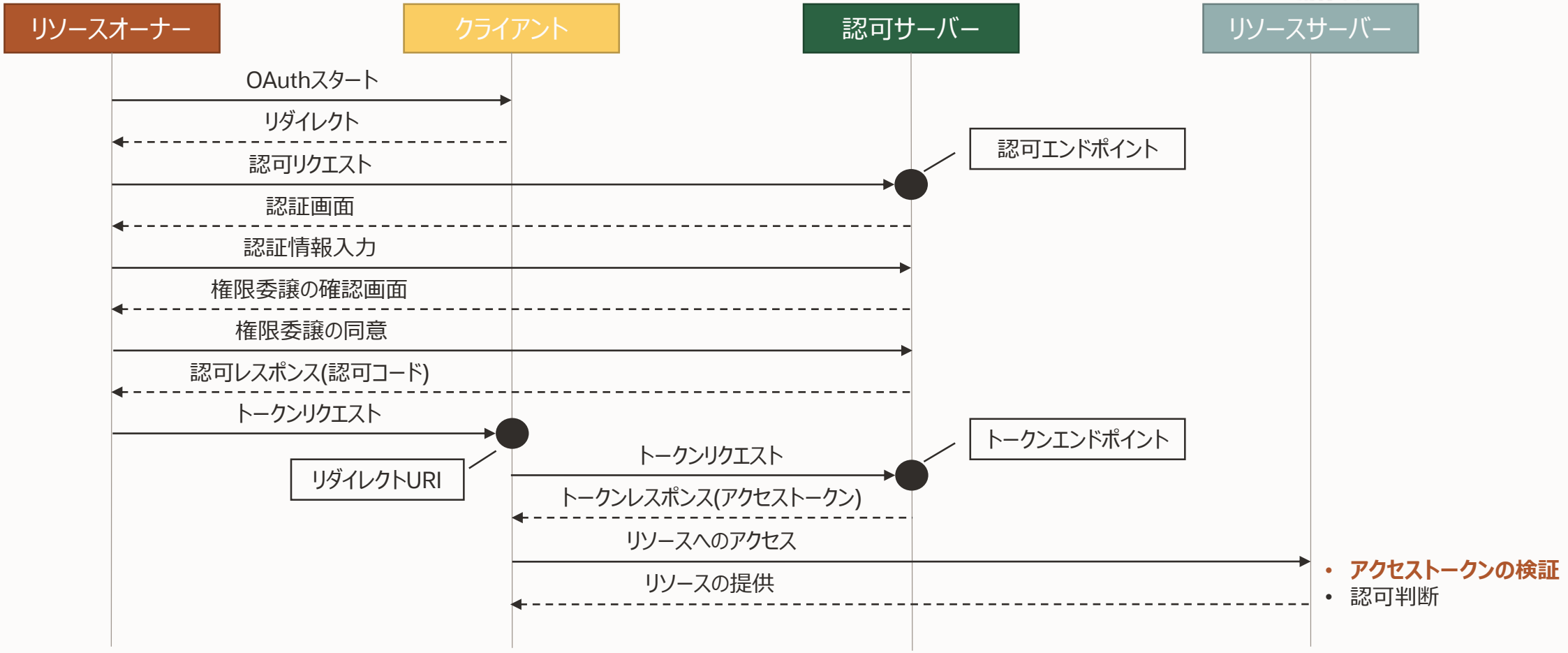


認可コードグラントによるアクセストークンの取得フロー



[^1] 現在発行されているアクセストークンが無効化、もしくは期限切れとなった際に新しいアクセストークンを取得するためのコード。
リフレッシュトークンの発行はオプションであり、認可サーバーの実装に委ねられる。

認可コードグラントによるアクセストークンの取得フロー

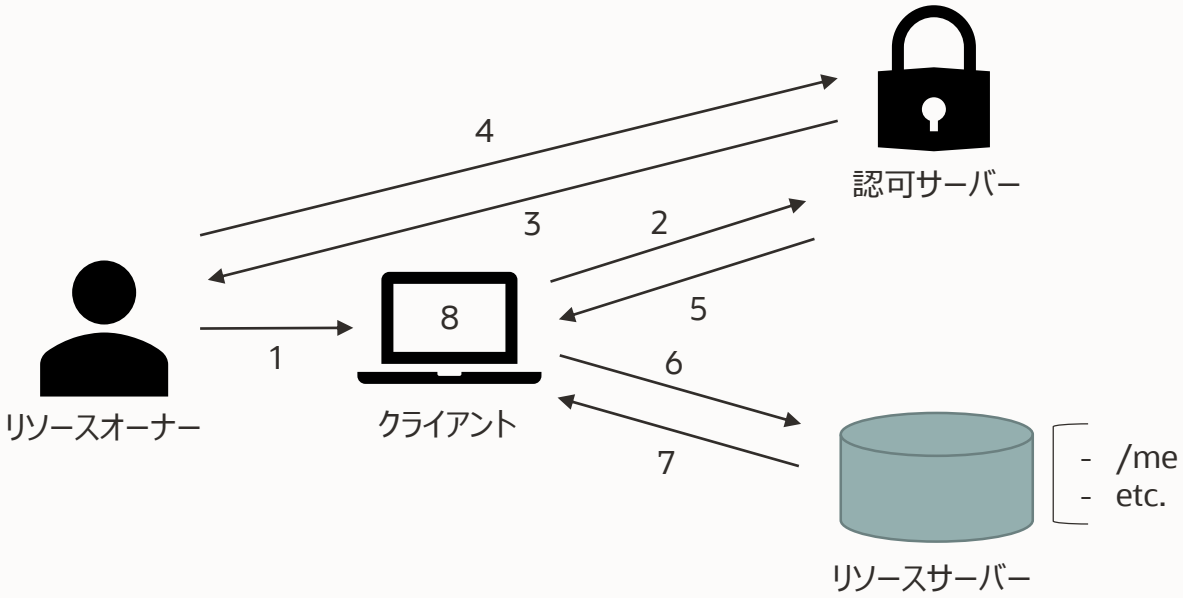


OAuth 認証？

よく言われている OAuth 認証ってなに？

OAuth認証 = OAuth + ユーザー情報を提供するAPI (Profile API)を用いてユーザーの認証を行うこと

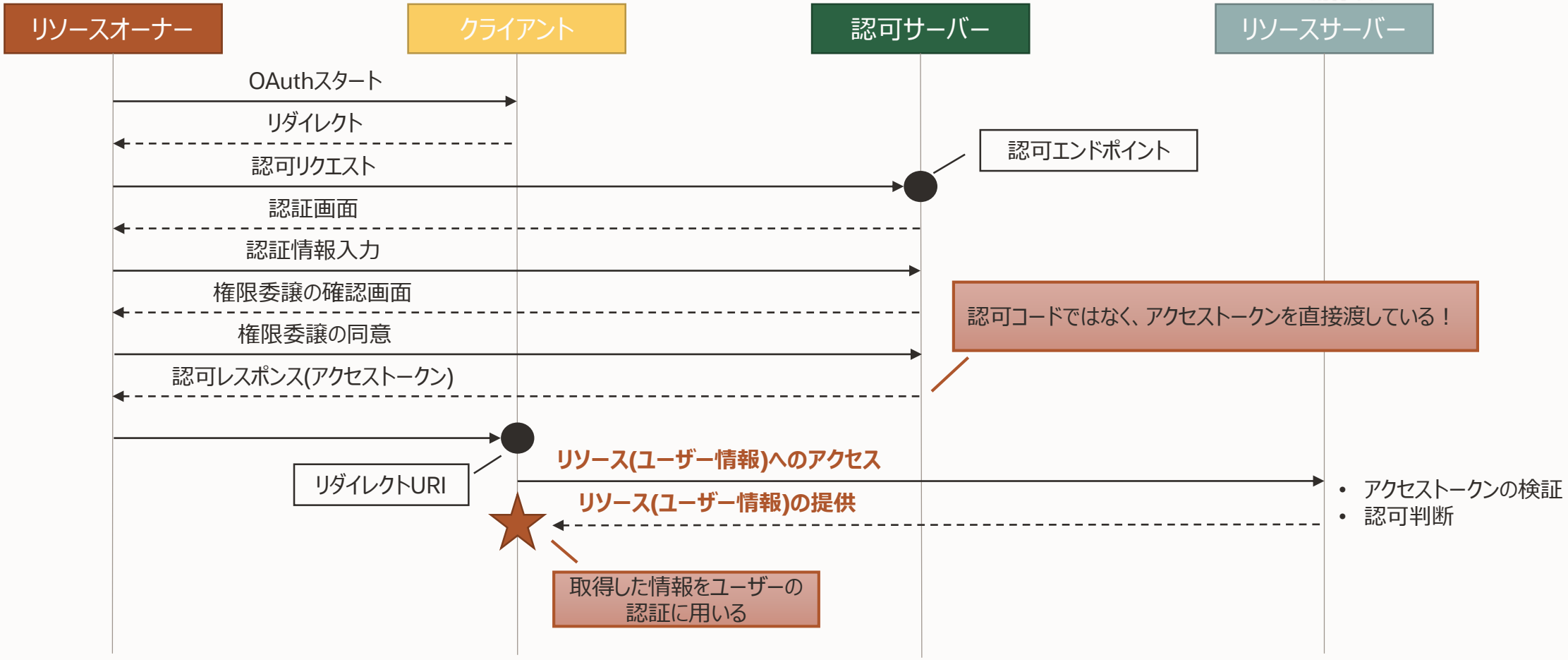
- OAuthで認証を行うというよりは、ユーザー情報を提供するAPIを叩くための認可処理にOAuthを使う



1. クライアントに対して、ログインの要求
2. クライアントが認可サーバーに対して、リソースサーバー(/me)へのアクセス権を要求する
3. 認可サーバーは、「リソースサーバー(/me)へのアクセス権をクライアントに委譲すること」についてリソースオーナーの合意を求める
4. リソースオーナーが合意する
5. 認可サーバーは権限が委譲された証明として、アクセストークンを発行し、クライアントへ渡す
6. クライアントは、発行されたアクセス権を持ってリソースサーバー(/me)にアクセスする
7. アクセストークンの有効性を確認し、問題なければ当該APIのレスポンス結果(ユーザー情報)を返却する
8. クライアントは、取得したユーザー情報を用いて認証を行う

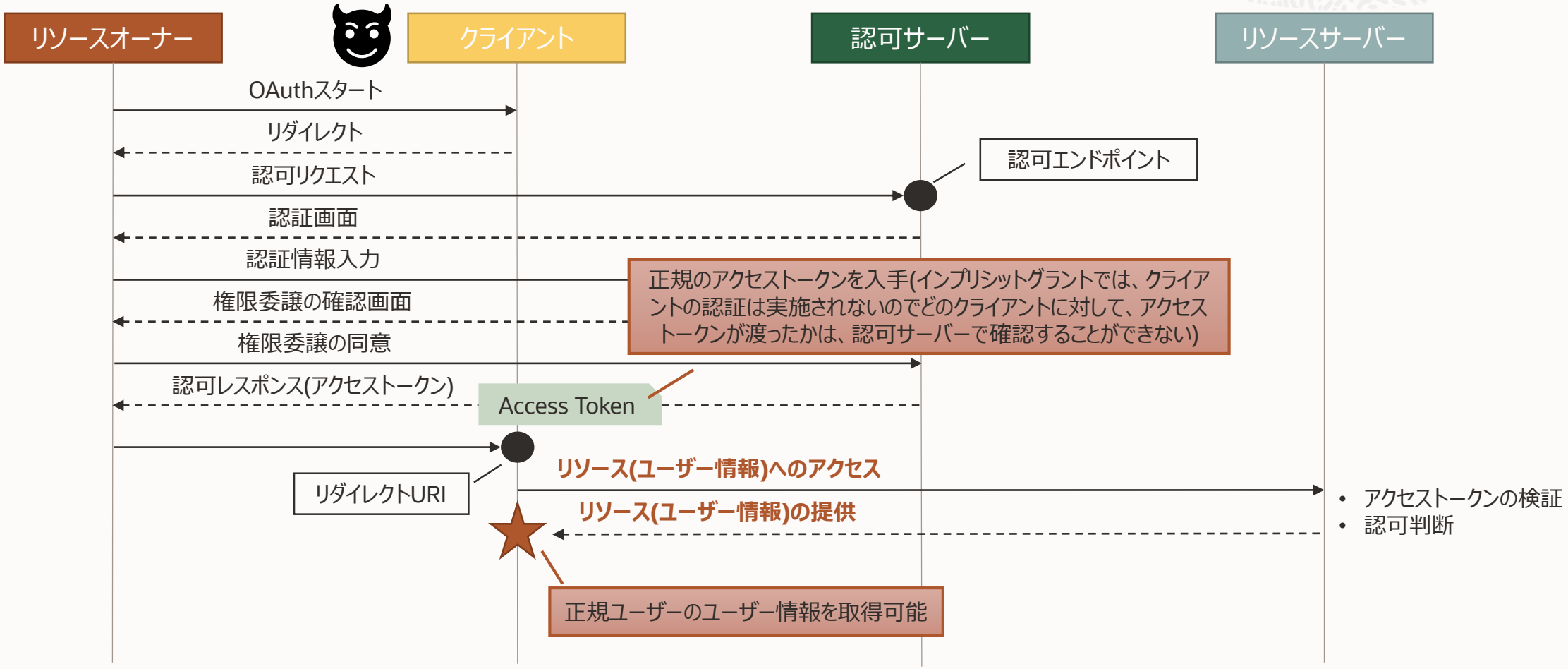
OAuth 認証のシーケンス

インプリシットグラントを用いたOAuth認証の例



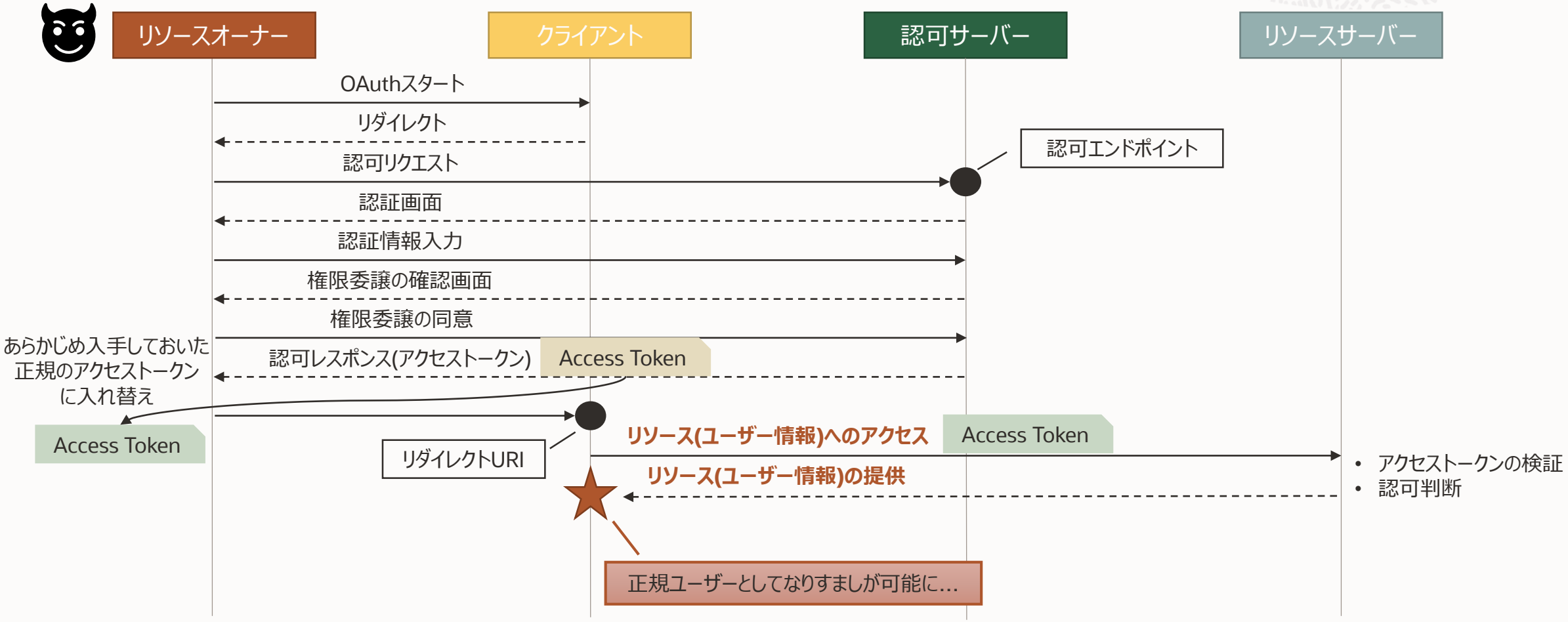
OAuth 認証の危険性

インプリシットグラントを用いたOAuth認証の例



OAuth 認証の危険性

インプリシットグラントを用いたOAuth認証の例



OAuth 2.0 まとめ

- (繰り返し) **権限委譲／認可情報を扱うプロトコル**で RFC 6749 に規定されている
- まずは、基本フローである認可コードグラントを理解する
- 誤った使い方／実装をすると、アプリケーションに脆弱性を埋め込むことになる(※参考情報あり)
- エンドユーザーの認証が目的ならOpenID Connectを使用する(※後述)

OAuth と OpenID Connect の関係

OpenID Connect 1.0 は, OAuth 2.0 プロトコルの上にシンプルなアイデンティティレイヤーを付与したものである. このプロトコルは Client が Authorization Server の認証結果に基づいて End-User のアイデンティティを検証可能にする. また同時に End-User の必要最低限のプロフィール情報を, 相互運用可能かつ RESTful な形で取得することも可能にする.

引用) http://openid-foundation-japan.github.io/openid-connect-core-1_0.ja.html

- 別なプロトコルではなく、OAuthを拡張し認証目的で使えるようにしたのがOpenID Connect
- リクエストのパラメータで、OAuthかOIDCとしての動作(レスポンス)が決まる

引用) OCHaCafe #5 避けては通れない認証・認可 - https://www.slideshare.net/oracle4engineer/ochacafe5

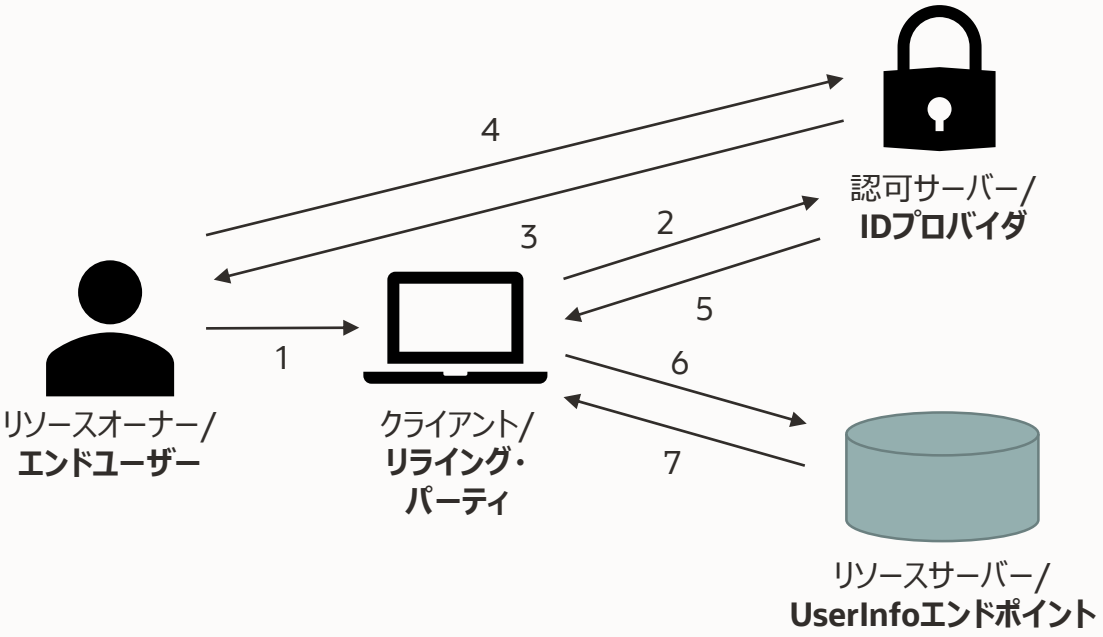
OpenID Connect 1.0 のアクター

エンドユーザー
- OIDCにはリソースの概念がないので、単にエンドユーザーと呼ぶ

リライング・パーティー(Relying Party)
- IDプロバイダが認証した結果を受け取るアプリケーションのこと

UserInfoエンドポイント
- ユーザーの情報を提供するAPIエンドポイント

IDプロバイダ(Identity Provider/OpenID Provider)
- IDトークンおよびアクセストークンを発行するサーバー



※OAuthアクター/**OpenID Connectアクター**

IDトークン

- エンドユーザーの認証に必要な項目(Claim)を含んだセキュリティトークン
  - 誰が？誰を？誰のために？などの認証イベントの情報を含むように標準化されている
- JWS形式のJWTやNested JWT(JWSをJWEが含んでいる形式)で表現される
- クライアント(Relying Party)に向けて発行される
  - (再掲) アクセストークンは、リソースサーバーに向けて発行される

JWT形式のアクセストークンのペイロード例

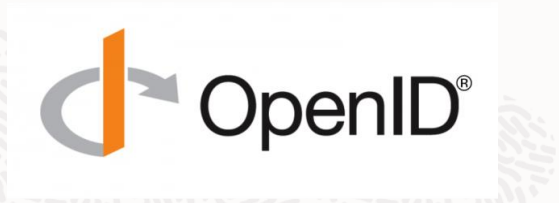
```
{
  "sub": "admin",
  "iss": "https://identity.oraclecloud.com/",
  "client_id": "1d6278e65db64a549041302e5420b43f",
  "sub_type": "user",
  "scope": "first_scope second_scope",
  "user_lang": "en",
  "exp": 1623138587,
  "iat": 1623134987,
  "jti": "11ebc825b7406d80a85da9ac9646b46f",
  "aud": "https://ochacafe4.cf/",
  … 略
}
```

発行先が違う

IDトークンのペイロード例

```
{
  "at_hash": "6UCttKdihAWGKaZBKp0sQQ",
  "sub": "admin",
  "iss": "https://identity.oraclecloud.com/",
  "client_id": "1d6278…",
  "exp": 1623163911,
  "iat": 1623135112,
  "jti": "11ebc8260159b282991fdb9557780118",
  "aud": [
    "https://identity.oraclecloud.com/",
    "1d6278…"
  ],
  … 略
}
```

ペイロード内の標準クレーム

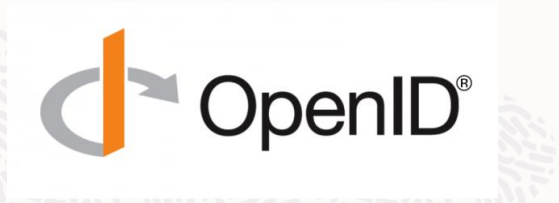


claim	description	
* iss	Issue Identifier: ID Tokenの発行者	誰が？(IdPが)
* sub	Subject Identifier: End-Userの識別子	誰を？(End-Userを)
* aud	Audience: ID Tokenの発行対象で、OAuth 2.0のclient_idを含む必要がある	誰のために？(Relying Partyのために)
* exp	Expiration Time: ID Tokenの有効期限	
* iat	Issuer At: JWTの発行時刻	
auth_time	End-Userの認証が発生した時刻。リクエストにmax_ageが含まれている場合は必須。	
nonce	ClientセッションとID Tokenを紐づける文字列。リプレイ攻撃防止用のパラメータ	
acr	Authentication Context Class Reference: 実施された認証処理が満たすAuthentication Context Classを示す	
amr	Authentication Method References: 認証方法を示す	
azp	Authorizated Party: ID Tokenの発行対象である認可された関係者(=OAuth 2.0のclient_id)	

*: 必須項目

参考) http://openid-foundation-japan.github.io/openid-connect-core-1_0.ja.html#IDToken

スコープ

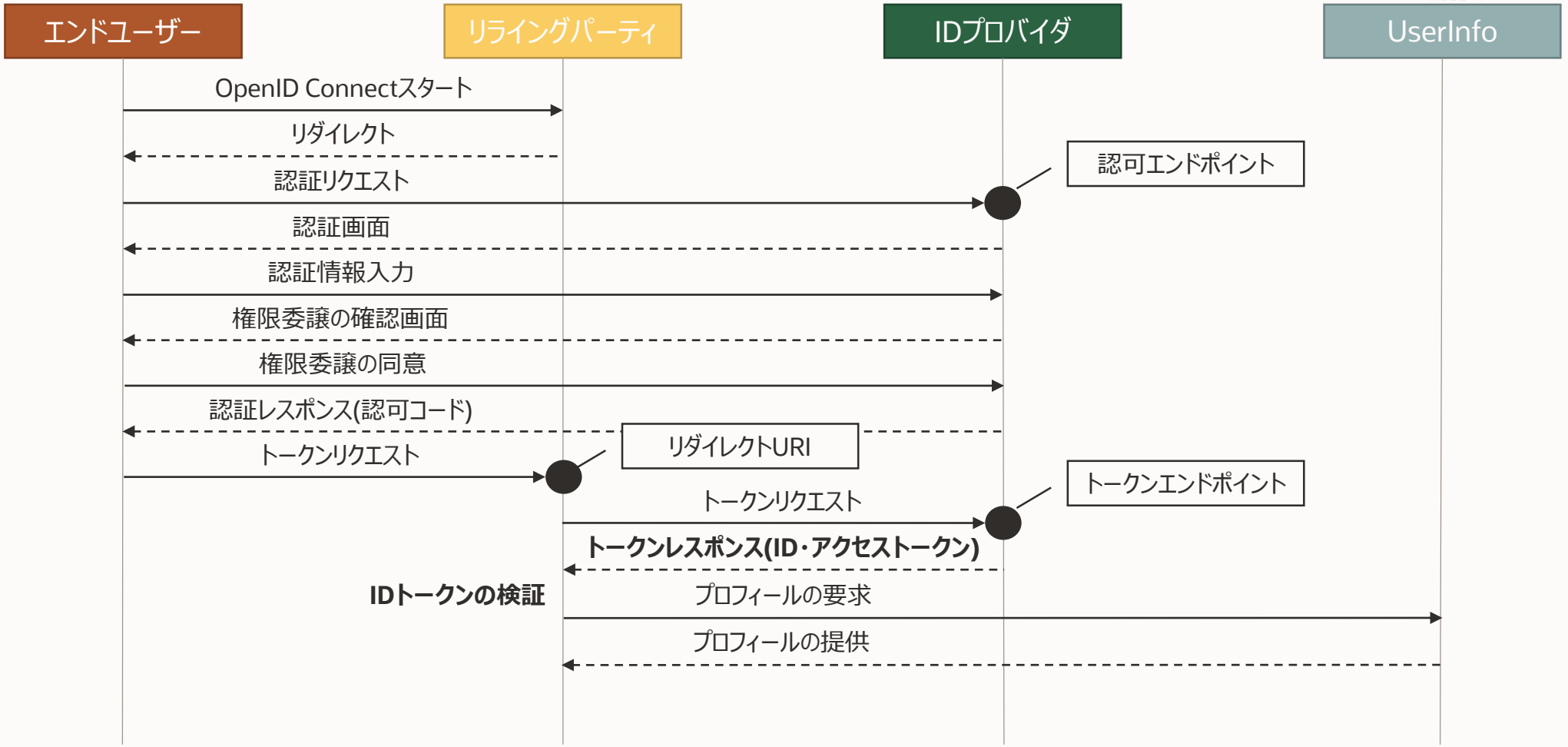


scope	description
* openid	OAuth or OIDCの判定に使用する。OIDCとしてのレスポンスを取得したい場合は指定必須。
profile	End-Userのデフォルトプロフィール Claim (name, gender etc.) へのアクセス要求に必要
email	email, email_verified Claim へのアクセス要求に必要
address	address Claim へのアクセス要求に必要
phone	phone_number, phone_number_verified Claim へのアクセス要求に必要

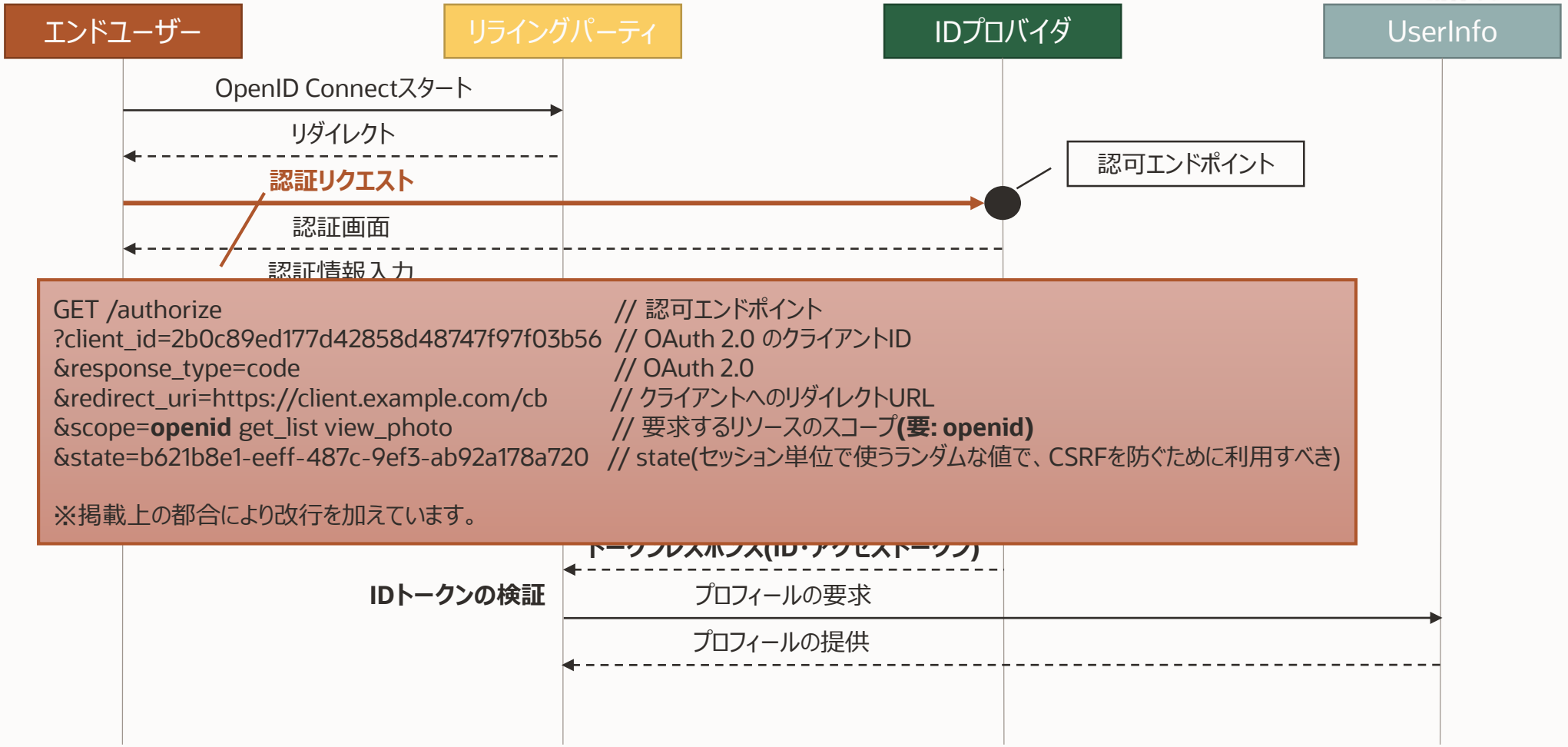
*: 必須項目

参考) http://openid-foundation-japan.github.io/openid-connect-core-1_0.ja.html#ScopeClaims

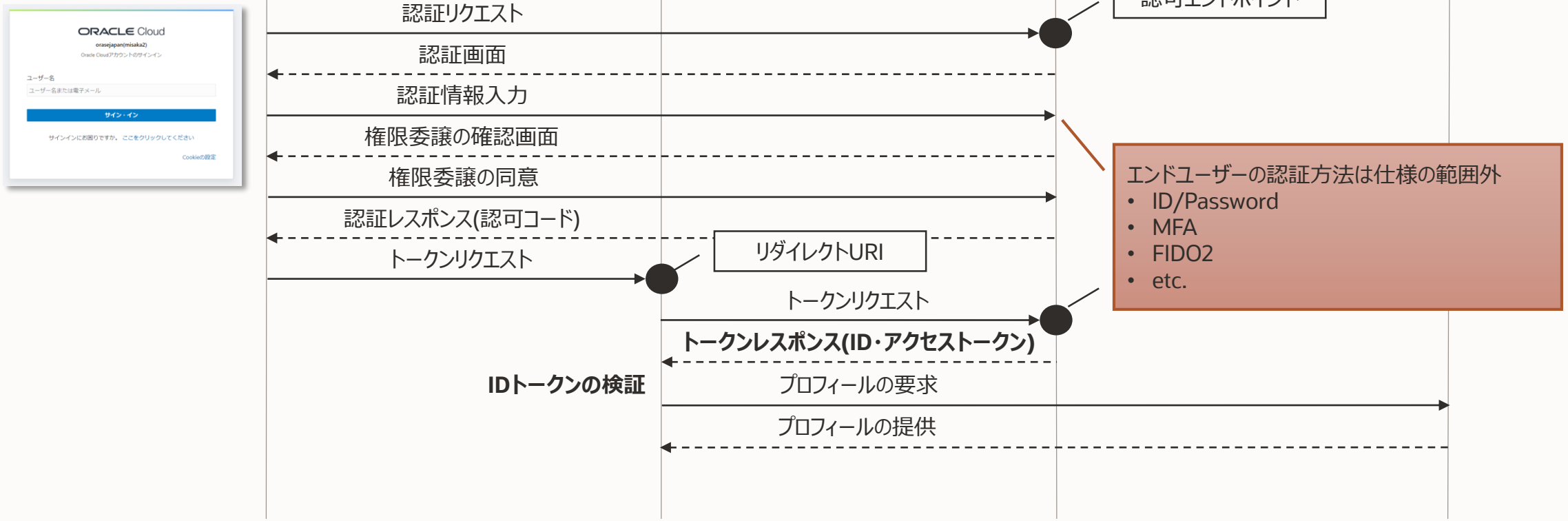
認可コードフローによるトークン(ID/アクセス)の取得フロー



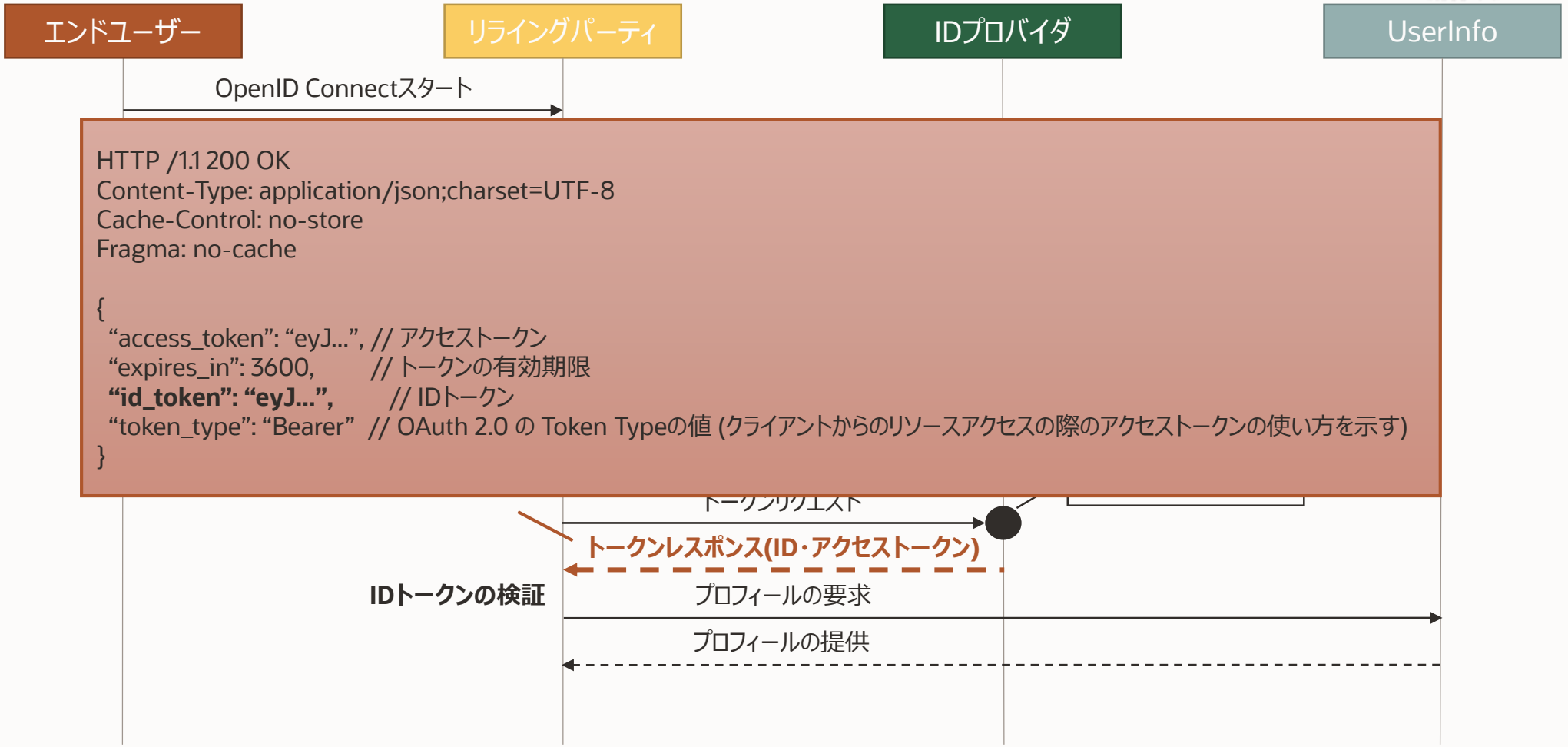
認可コードフローによるトークン(ID/アクセス)の取得フロー



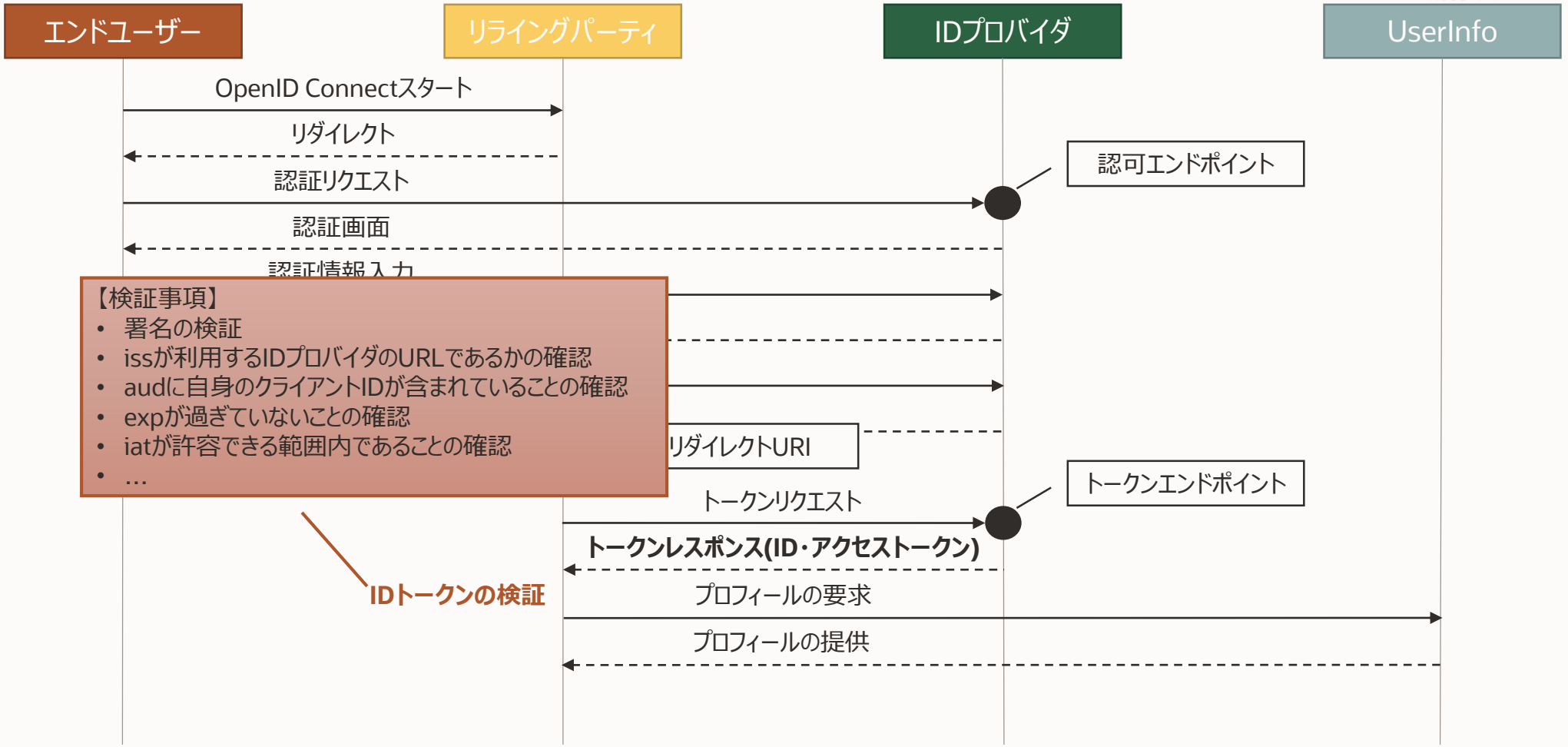
認可コードフローによるトークン(ID/アクセス)の取得フロー



認可コードフローによるトークン(ID/アクセス)の取得フロー



認可コードフローによるトークン(ID/アクセス)の取得フロー



OpenID Connect 1.0まとめ

- OAuthを拡張し、認証目的で使用できるようにした仕様
  - OpenID Connect Core 1.0 incorporating errata set 1
  - まずは、OAuthをしっかりと学習することが大切
- OAuthのJWTなアクセストークンとOIDCのIDトークンの違いをしっかりと理解する
  - トークンの発行先
    - アクセストークン → リソースサーバー
    - IDトークン → Relying Party(OAuthクライアント)
  - IDトークンはどんな情報を含むように標準化されているのか？
    - 誰が？誰を？誰のために？などといった認証イベントに関する情報

Agenda

1. 認証・認可のおさらい
2. Javaで実現するマイクロサービスの認証・認可
3. JSON Web Token(JWT)とその周辺仕様
4. OAuth 2.0/OpenID Connect 1.0
5. **MicroProfile – JWT Propagation & デモ**
6. まとめ

MP - JWT

MP - JWT

- https://download.eclipse.org/microprofile/microprofile-jwt-auth-1.2/microprofile-jwt-auth-spec-1.2.html
- マイクロサービス間で相互運用可能な認証・認可のために使われるJWTのフォーマットを標準的に定義したもの
  - OIDCのIDトークンの標準的なクレームに加え、Jakarta(Java) EEのRBACセキュリティ・モデルに組み込むために必要なクレームが追加されている

```
{
    "typ": "JWT",
   "alg": "RS256",
   "kid": "abc-1234567890"
}
{
      "iss": "https://server.example.com",
      "jti": "a-123",
      "exp": 1311281970,
      "iat": 1311280970,
      "sub": "24400320",
      "upn": "jdoe@server.example.com",
      "groups": ["red-group", "green-group", "admin-group", "admin"],
}
```

MP - JWT に含まれる標準的なクレーム (Header)

claim name	description
* alg	MP - JWTの署名に使用される暗号アルゴリズムを識別するためのもの (RS-256 or ES-256 or RSA-OAEP)
* enc	クレーム or 入れ子になったJWT(Nested JWT)を暗号化する場合の暗号アルゴリズムを指定する (A256GCM固定)
typ	トークンのフォーマットを指定するもの (JWT固定)
kid	検証キーがJWKの場合、どの鍵が使われたかを指定する

*: 必須項目

MP - JWT に含まれる標準的なクレーム (Payload)

claim name	description	
* iss	Issue Identifier: ID Tokenの発行者	**誰が？**
* iat	Issuer At: JWTの発行時刻	
* exp	Expiration Time: ID Tokenの有効期限	
* upn	User Principal Name: End-Userの識別子(Human Readable) [^1]	**(誰を？)**
sub	Subject Identifier: End-Userの識別子	**誰を？**
jti	JWT ID: JWTの識別子	
aud	Audience: ID Tokenの発行対象で、OAuth 2.0のclient_idを含む必要がある	**誰のために？**
groups	End-Userに割り当てられているグループ名のリスト[^2]	

*: 必須項目

[^1] 別クレームで代用可能 (e.g. sub, preferred_username)
[^2] 別クレームやカスタムマッパーを作成することで代用可能

MP-JWT検証の要件

1. JOSE “alg”ヘッダーが含まれていること
2. (暗号化されたトークンを期待する場合) JOSE “enc”ヘッダーが含まれていること
3. “iss”クレームが含まれていること、期待通りのissであること
4. “iat”クレームが含まれていること
5. “exp”クレームが含まれていること
6. “upn”, “preferred_username”, “sub”のうち少なくとも一つのクレームを含むこと

MP-JWTを用いてJAX-RSアプリケーションでRBACをするための準備

@LoginConfig(authMethod = “MP-JWT”)

```
import org.eclipse.microprofile.annotation.LoginConfig;

import javax.ws.rs.ApplicationPath;
import javax.ws.rs.core.Application;

@LoginConfig(authMethod = "MP-JWT", realmName = "TCK-MP-JWT")
@ApplicationPath("/")
public class TCKApplication extends Application {

}
```

application.yaml/microprofile-config.properties

```
# configure JWT handling
mp.jwt.verify.issuer: https://server.example.com
mp.jwt.verify.publickey.location: publicKey.pem
...
```

引用) https://download.eclipse.org/microprofile/microprofile-jwt-auth-1.2/microprofile-jwt-auth-spec-1.2.html#_marking_a_jax_rs_application_as_requiring_mp_jwt_access_control

MP-JWTとJakarta(Java) EE Container APIのマッピング

CDI

MP - JWT全体をインジェクションする

```
@Path("/endp")
@DenyAll
@ApplicationScoped
public class RolesEndpoint {
    @Inject
    private JsonWebToken jwt;
...
}
```

クレーム単位でインジェクションする

```
@ApplicationScoped
public class MyEndpoint {
    @Inject
    @Claim(value="exp", standard=Claims.iat)
    private Long timeClaim;
...
}
```

引用) https://download.eclipse.org/microprofile/microprofile-jwt-auth-1.2/microprofile-jwt-auth-spec-1.2.html#_injection_of_code_jsonwebtoken_code

MP-JWTとJAX-RS Container APIのマッピング

javax.ws.rs.core.SecurityContext

SecurityContext#getUserPrincipal

```
javax.ws.rs.core.SecurityContext.getUserPrincipal()
```

- org.eclipse.microprofile.jwt.JsonWebTokenのインスタンスが返却される

SecurityContext#isUserInRole(String)

```
javax.ws.rs.core.SecurityContext.isUserInRole()
```

- MP - JWTの”groups”に任意の文字列が含まれているかを確認する

MP-JWTを用いたRBAC

Common Security Annotation

javax.annotation.security.RolesAllowed

```
public class MyEndpoint {

    @RolesAllowed({"Admin", "Guest"})
    public String greet() { return "Hello, world"; }
}
```

javax.annotation.security.PermitAll

```
@PermitAll
public class MyEndpoint {

    public String greet() { return "Hello, world"; }
}
```

javax.annotation.security.DenyAll

```
@DenyAll
public class MyEndpoint {

    public String greet() { return "Hello, world"; }
}
```

Helidon

Javaベースのマイクロサービス・フレームワーク

Oracleがホストするオープンソースプロジェクト

- GitHubでソースコードを公開
  https://github.com/oracle/helidon

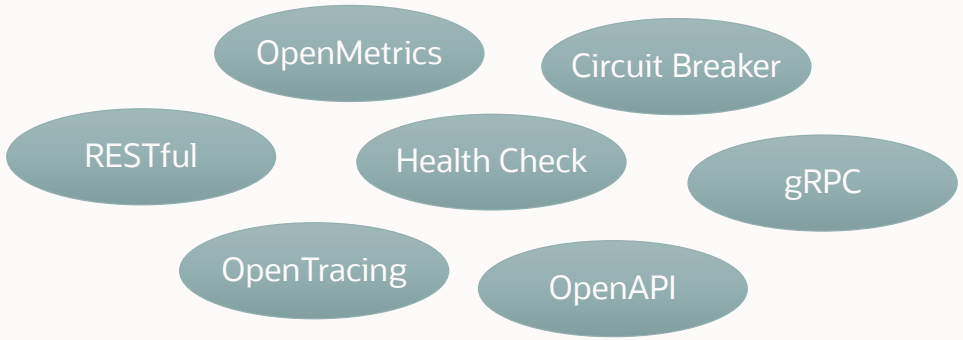
マイクロサービス・アプリケーションが必要とする機能を提供するJavaライブラリ群

JVM上の単体アプリケーションとして動作

標準的なツールで開発＆デプロイ可能

- Java SE, Maven, Docker, Kubernetes, etc.

二つのエディションを提供: SE, MP



helidon SE

- マイクロ・フレームワーク
- 超軽量フットプリント
- 関数型
- Reactive Web Server
- GraalVM native image

helidon MP

- Eclipse MicroProfile 3.3
- 軽量フットプリント
- 宣言型
- Java EEサブセット + マイクロサービス関連機能
- GraalVM polyglot

【宣伝】日本語ドキュメントもあります！！！
https://oracle-japan-oss-docs.github.io/helidon/docs/v2/#/about/01_overview

Oracle Identity Cloud Service (IDCS)

エンタープライズ向けのID・アクセス管理サービス

【提供サービス】

Single Sign On

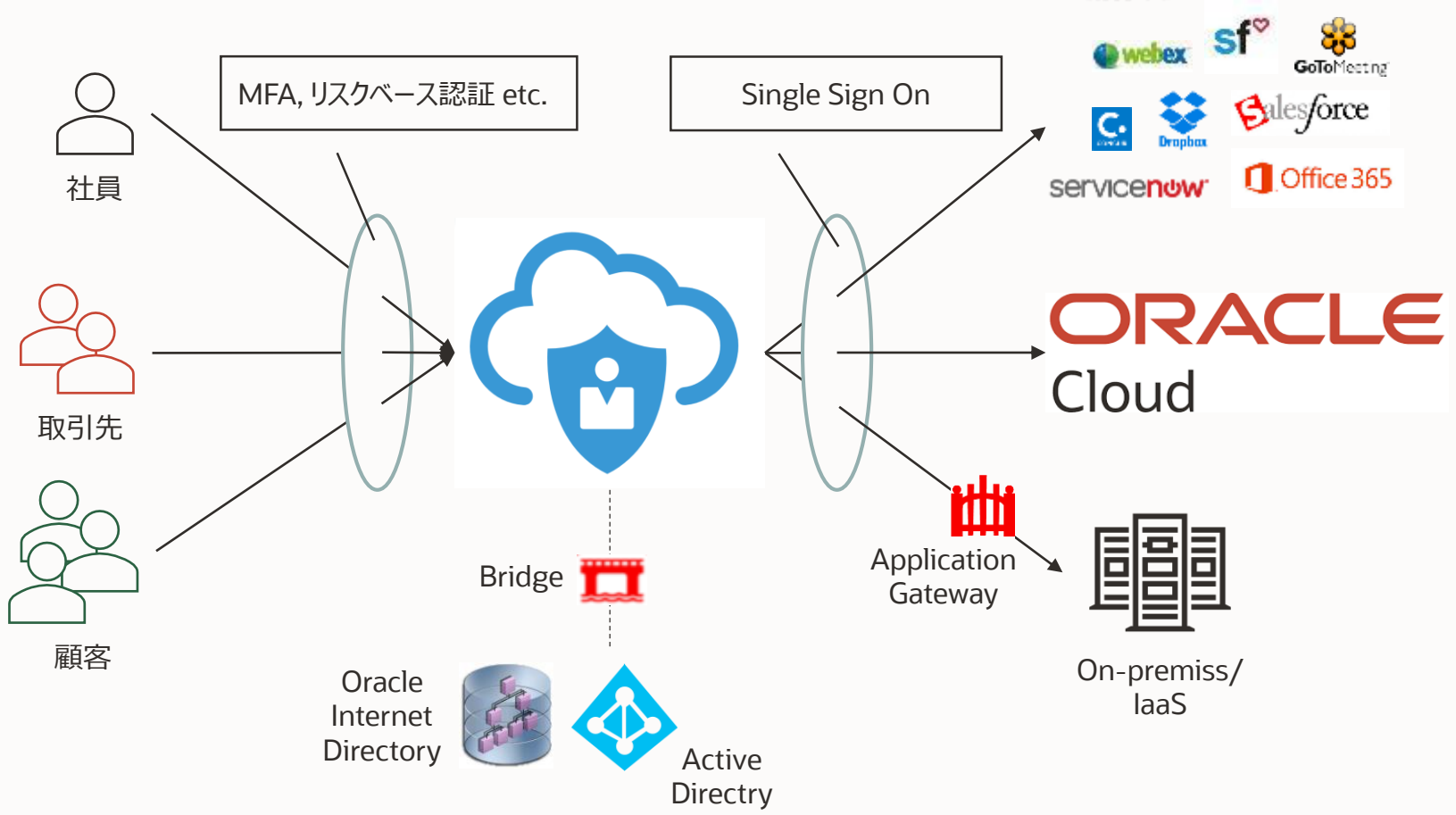
- フェデレーション標準(SAML/OIDC)に対応
- 外部IdPの認証の引継ぎ
- オンプレミス用のエージェント

認証強化

- MFA
- FIDO
- リスクベース認証
- etc.

ID管理

- セルフ・サービス機能
- AD/OIDからの同期エージェント



Helidon + IDCS を用いた実装例

必要な依存関係は二つだけ

```
<dependencies>
    <!-- OIDCに準拠し、認証を行うためのProvider -->
    <dependency>
        <groupId>io.helidon.microprofile</groupId>
        <artifactId>helidon-microprofile-oidc</artifactId>
    </dependency>
    <!-- IDCS用のカスタムマッパー[^1] -->
    <dependency>
        <groupId>io.helidon.security.providers</groupId>
        <artifactId>helidon-security-providers-idcs-mapper</artifactId>
    </dependency>
</dependencies>
```

[^1]: IDCSは、MP-JWTを用いてRBACを行うために必要なクレーム(groups)をJWTに含めないため、対応する項目を取得するためのカスタムマッパーが必要(Helidonから提供されています)

Helidon + IDCS を用いた実装例

application.yaml/microprofile-config.propertiesにOpenID Connectとカスタムマッパー用の設定を定義

```
security:
  providers:
    - oidc:
        client-id: 1d62...
        client-secret: 788a6824...
        identity-uri: https://idcs-f0017d...
        scope-audience: https://ochacafe.season4
        frontend-uri: http://138.xx.xx.xx
        audience: https://idcs-f0017d...
        server-type: idcs
        idcs-roles: true
        header-use: true
    - idcs-role-mapper:
        multitenant: false
        oidc-config:
          client-id: 1d62...
          client-secret: 788a6824...
          identity-uri: https://idcs-f0017d...
```

Helidon + IDCS を用いた実装例

ログイン用のエンドポイントに対して、@Authenticated を付ければ、対応するSecurity Providerが認証処理を実行

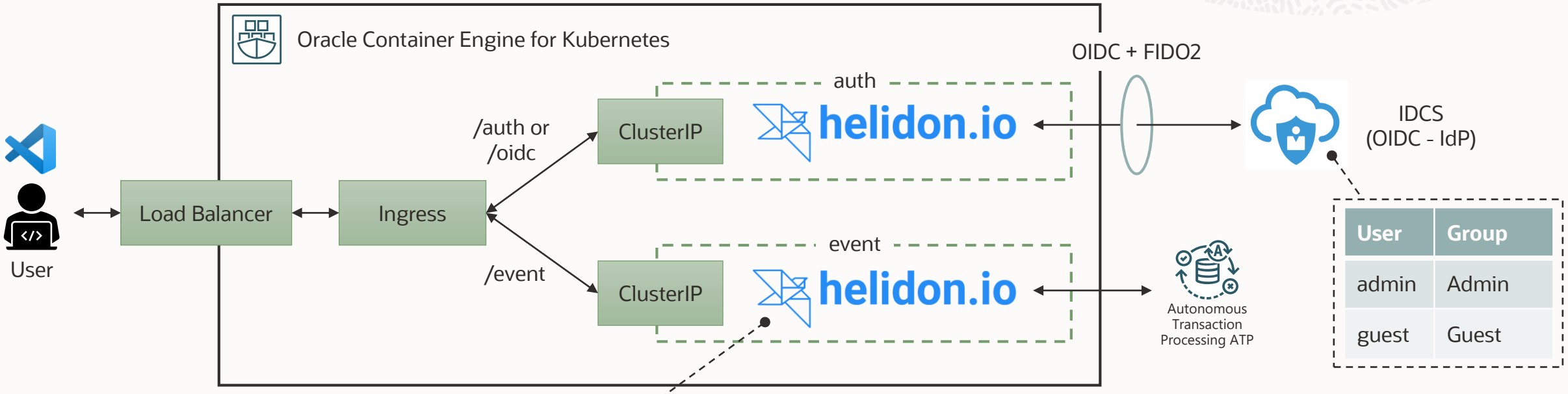
```
@Path("auth")
public class AuthResource {

    @GET
    @Path("login")
    @Authenticated
    @Produces(MediaType.APPLICATION_JSON)
    public JsonObject login(@Context SecurityContext securityContext, @Context ContainerRequestContext context) {
        return JSON.createObjectBuilder()
                .add("access_token", getAccessToken(context))
                .build();
    }
...
}
```

Helidon + IDCS を用いた実装例

RBAC制御が必要なエンドポイントに対して、@Authenticated, @RolesAllowed を付与する

```
@Path("event")
public class EventResource {
    private final EventService eventService;

    @Inject
    public EventResource(EventService eventService) {
        this.eventService = eventService;
    }

    @GET @Produces(MediaType.APPLICATION_JSON)
    @Authenticated @RolesAllowed({"Admin", "Guest"})
    public List<Event> getAllEvent() {
        return eventService.getAllEvent();
    }

    @POST @Consumes(MediaType.APPLICATION_JSON) @Produces(MediaType.APPLICATION_JSON)
    @Authenticated @RolesAllowed("Admin")
    public Event createEvent(CreateEventRequest createEventRequest) {
        return eventService.createEvent(createEventRequest);
    }

}
```

デモ



User	Group
admin	Admin
guest	Guest

Path	RolesAllowed
GET /event	Admin, Guest
GET /event/season/{season}	Admin, Guest
POST /event	Admin

API コール

Path	Result (Guest / Admin)
GET /event	**HTTP 200 / HTTP 200**
GET /event/season/{season}	**HTTP 200 / HTTP 200**
POST /event	**HTTP 403 / HTTP 200**

Agenda

1. 認証・認可のおさらい
2. Javaで実現するマイクロサービスの認証・認可
3. JSON Web Token(JWT) とその周辺仕様
4. OAuth 2.0/OpenID Connect 1.0
5. MicroProfile – JWT Propagation & デモ
6. **まとめ**

まとめ

- Eclipse MicroProfile - JWT Propagation
  - マイクロサービスのエンドポイント(JAX-RS)に対して、RBACを行うためにOIDCのIDトークンに追加するクレーム(upn, groups)を定義
- OAuth 2.0
  - 認可のプロトコル
  - ユーザーの認証が目的なら独自実装(OAuth認証)等はせずにOpenID Connectを使う
- OpenID Connect 1.0
  - 認可 + 認証 + Profile API
  - OAuthで使われるJWTなアクセストークンとIDトークンの違いを理解する

参考情報

Eclipse MicroProfile Interoperable JWT RBAC

- https://download.eclipse.org/microprofile/microprofile-jwt-auth-1.2/microprofile-jwt-auth-spec-1.2.html

Authentication and Authorization of End User in Microservice Architecture

- https://iopscience.iop.org/article/10.1088/1742-6596/910/1/012060

OAuth、OAuth認証、OpenID Connectの違いを整理して理解できる本

- https://authya.booth.pm/items/1550861

ID トークンが分かれば OpenID Connect が分かる

- https://qiita.com/TakahikoKawasaki/items/8f0e422c7edd2d220e06

OAuth認証は何か？なぜダメなのか – 2020冬

- https://ritou.hatenablog.com/entry/2020/12/01/000000

参考情報

JWS RFC 7515

- https://datatracker.ietf.org/doc/html/rfc7515

JWE RFC 7516

- https://datatracker.ietf.org/doc/html/rfc7516

JWK RFC 7517

- https://datatracker.ietf.org/doc/html/rfc7517

JWA RFC 7518

- https://datatracker.ietf.org/doc/html/rfc7518

JWT RFC 7519

- https://datatracker.ietf.org/doc/html/rfc7519

OpenID Connect Core 1.0 incorporating errata set 1 (日本語訳)

- http://openid-foundation-japan.github.io/openid-connect-core-1_0.ja.html

OAuth 2.0 RFC 6749 (日本語訳)

- https://openid-foundation-japan.github.io/rfc6749.ja.html

参考情報

Helidon

- https://helidon.io/docs/v2/#/about/01_overview (英語)
- https://oracle-japan-oss-docs.github.io/helidon/docs/v2/#/about/01_overview (日本語)

Oracle Identity Cloud Service(IDCS)

- https://docs.oracle.com/cd/E83857_01/paas/identity-cloud/index.html

OCHaCafe #5 避けては通れない認証・認可

- https://www.slideshare.net/oracle4engineer/ochacafe5

デモのソースコード

- https://github.com/oracle-japan/ochacafe-mp-jwt-demo

Thank you


ORACLE