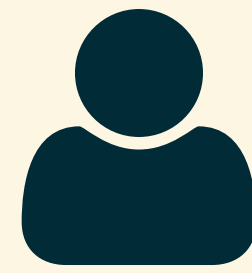


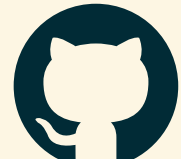


(?)

簡單易懂的 OAuth 2

鴨七

2013/11/26, Ruby Tuesday Taipei #27



- 鴨七（莊育承）
- **Rails** Developer at **KKBOX**, working at the **KKTIX** team
-  chitsaou,  @yorkxin,  blog.yorkxin.org

Today's Target

- 你會知道 OAuth 2 Protocol 怎麼跑
- 你會看得懂 99% 基於 OAuth 2 的第三方 API 文件
- 你會知道怎麼用 OAuth 2 鎖你的 API

Agenda

- OAuth 2 是甚麼？ 可以拿來幹嘛？
- OAuth 2 通訊協定怎麼跑
- 製造 OAuth 2 Provider 的方法
- 第一次用 Rails + Grape API 整合 OAuth 2 就上手

OAuth 2 是甚麼？ 好吃嗎？

OAuth 2.0

- 全名 “The OAuth 2.0 Authorization Framework”
- Authorization (*n.*) 授權

人

授權給 程式

說 「你可以去跟母站拿我的資料」



人

授權給 程式

說 「你可以去跟母站拿我的資料」



人



授權給 程式

說 「你可以去跟母站拿我的資料」



User

人



App

授權給 程式

說 「你可以去跟母站拿我的資料」



Website

User

人

App

授權給 程式

說 「你可以去跟母站拿我的資料」

Website

API

User

人

App

授權給 程式

說 「你可以去跟母站拿我的資料」

Website

API

Sounds Familiar?

You

人

心測程式

授權給 程式

說「你可以去跟母站拿我的資料」

Facebook

API

Sounds Familiar?

The Old-School Way...

- App 要你輸入帳號密碼，還聲明不會亂搞
- 你要信任 App 不會拿你的密碼亂搞
- 就算 App 很乖，被幹走還是沒救
- 不管你信不信，我反正是

OAuth 如何解決這個問題

- 網站提供官方界面，讓你可以授權程式使用資料
- 程式存的是 Token 而非帳號密碼
也用 Token 去打 API
- Token 可以規定存取範圍（scopes, 如：好友列表）
- Token 有期限而且可以隨時撤銷（不怕被幹走）

OAuth 2 裡面的角色

- **Resource Owner** - 資料擁有者，通常是人類 (User)
- **Client** - 即 App，要存取 User 的資料的程式
- **Authorization Server** - 總管一切授權事務
- **Resource Server** - Client 從這裡拿資料，即 API

You

心測程式

人透過○○授權給程式

說「你可以去跟母站拿我的資料」

Facebook

API

**Resource
Owner**

You

**Authorization
Server**



Client

心測程式

人透過○○授權給程式

說「你可以去跟母站拿我的資料」

Facebook

API

Resource Server

Resource Owner

Resource Owner = User

- 就是你網站的 **User**
- **API** 拉到的是跟 **User** 有關的資料
- 例如好友名單、信件內容
- 授權必須經由**本人**親自確認

Resource Owner = Client

- 這種情況沒有人的存在。
- 例：爬公開資料
- Twitter 稱為 App-Only Authorization
- Facebook 稱為 App Access Token

Client

Client

- User 授權給第三方程式，這個程式就是 Client
- 例：Facebook 上的遊戲、手機上的 App
- 官方程式也算（例如手機 App、桌面 App）

Client 必須事先註冊

- “Client Registration”
- Client ID
- Client Secret （視為密碼）
- Redirect URI ← 這個很重要



yacisphere

App ID: 567672586646825

App Secret: 4486 [REDACTED] .b70c (reset)

● This app is in Sandbox Mode (Only visible to Admins, Developers and Testers)

基本資料

Display Name: [?]

yacisphere

Namespace: [?]

yacisphere

聯絡信箱: [?]

ducksteven@gmail.com

App Domains: [?]

localhost x

沙盒模式: [?]

☒ 啟用

☐ 已關閉


請選擇你的應用程式如何跟 Facebook 結合

✓ 以 Facebook 登入網站

網站位址 (URL) : [?]

http://localhost:3000/users/auth/facebook/callback

Client ID



yacisphere
App ID: 567672586646825
App Secret: 1188...b70c (reset)
● This app is in Sandbox Mode (Only visible to Admins, Developers and Testers)

基本資料

Display Name: [?]

yacisphere

Namespace: [?]

yacisphere

聯絡信箱: [?]

ducksteven@gmail.com

App Domains: [?]

localhost x

沙盒模式: [?]

☒ 啟用 ☐ 已關閉


請選擇你的應用程式如何跟 Facebook 結合

✓ 以 Facebook 登入網站

網站位址 (URL) : [?]

http://localhost:3000/users/auth/facebook/callback

Client Secret

**yacisphere**
App ID: 587672586618825
App Secret: 4486 [redacted] .b70c (reset)
This app is in Sandbox Mode (Only visible to Admins, Developers and Testers)

基本資料

Display Name: [?]

yacisphere

Namespace: [?]

yacisphere

聯絡信箱: [?]

ducksteven@gmail.com

App Domains: [?]

localhost x

沙盒模式: [?]

☒ 啟用 ☐ 已關閉


請選擇你的應用程式如何跟 Facebook 結合

☒ 以 Facebook 登入網站

網站位址 (URL) : [?]

http://localhost:3000/users/auth/facebook/callback

Redirect URI

**yacisphere**
App ID: 567672586646825
App Secret: 4486 [REDACTED] .b70c (reset)
● This app is in Sandbox Mode (Only visible to Admins, Developers and Testers)

基本資料

Display Name: [?]

yacisphere

Namespace: [?]

yacisphere

聯絡信箱: [?]

ducksteven@gmail.com

App Domains: [?]

localhost x

沙盒模式: [?]

☒ 啟用 ☐ 已關閉

請選擇你的應用程式如何跟 Facebook 結合

✓ 以 Facebook 登入網站

網站位址 (URL) : [?]

http://localhost:3000/users/auth/facebook/callback

Redirect URI

- User 本人親自確認之後，返回結果到 Client
- 要事先指定，若不一致則不可 Redirect 過去
- 可以指定多組，但通常是一組，或是開頭一致
- 最好是 HTTPS（不強制，例如手機 App 就辦不到）

Redirect URI

- <https://kktix.com/users/auth/facebook/callback>
- <http://kktix.dev/users/auth/facebook/callback>
- kktix-app:oauth2/callback

Public v.s. Confidential Client

- 根據「**能不能保護資料**」來區分
- **Confidential** - Server-Side Application
- **Public** - 手機 App / 桌面程式 / JavaScript App /
Browser Extension

Client Authorization (認證)

- 出示 Client ID + Secret 向 Auth. Server 認證自己
- 也就是說 Client 要登入到 Auth. Server
- 只適用 Confidential Client

Client

- 有 ID / Secret 用於認證
- 用 Redirect URI 確保瀏覽器轉址到正確的 Client
- Public / Confidential 有各自適用的授權流程

Endpoints

Endpoints

- **Authorization Endpoint** - 用來給 **User** 本人確認授權
- **Token Endpoint** - 用來讓 **Client** 取得真正的 Token
- **Redirection Endpoint** - **Client** 用來收資料用

Authorization Endpoint

- 給 User 本人確認授權
- 是一個網頁
- 拿到的是“Grant”（授權狀）而不是 Token
- User 答覆之後，會轉回 Client 的 Redirect URI

Token Endpoint

- 給 Client 取得真正的 Token
- JSON API 機械化界面，無網頁

Redirection Endpoint

- 用來從瀏覽器接收 Auth. Server 來的資料
- 開在 Client 而不是 Auth. Server
- Client 註冊時要寫的 “Callback URL”
- Auth. Server 會驗證 Redirect URI 是否相符才轉址

SSL! SSL! SSL!

- Auth. Server 上面的 Endpoints 必須上 HTTPS
- Client 必須驗證 SSL Certificate 是否合格
- Client Redirection Endpoint 不強制，但能上最好上
網站 → 最好上
手機 / 桌面 App → 沒辦法可以不要上

Resource Server

Resource Server

- Client 必須出示 Token 才能進去拿資料
- Client 只要出示 Token 就能進去拿資料
- 可以用 Scope 限制 Token 能夠取用的資料範圍

Resource Server

- Password-Free API
- Login via *YourWebsite*[™]

OAuth 2 Protocol 怎麼跑

「簡單啊， 讀 Spec 啊」 *

憨 人
*代誌不是本人所想的這麼簡單

- **RFC 6749:** The OAuth 2.0 Authorization Framework
 - 77 pages (PDF)
- **RFC 6750:** The OAuth 2.0 Authorization Framework: Bearer Token Usage
 - 19 pages (PDF)

/me 已讀。

- 用去 3 個禮拜
- 寫了 13 篇筆記 →

	posted in OAuth rails ruby Rack Grape
SEP 30	各大網站 OAuth 2.0 實作差異 posted in OAuth
SEP 30	OAuth 2.0 筆記 (7) 安全性問題 posted in OAuth
SEP 30	OAuth 2.0 筆記 (6) Bearer Token 的使用方法 posted in OAuth
SEP 30	OAuth 2.0 筆記 (5) 核發與換發 Access Token posted in OAuth
SEP 30	OAuth 2.0 筆記 (4.4) Client Credentials Grant Flow 細節 posted in OAuth
SEP 30	OAuth 2.0 筆記 (4.3) Resource Owner Password Credentials Grant Flow 細節 posted in OAuth
SEP 30	OAuth 2.0 筆記 (4.2) Implicit Grant Flow 細節 posted in OAuth
SEP 30	OAuth 2.0 筆記 (4.1) Authorization Code Grant Flow 細節 posted in OAuth
SEP 30	OAuth 2.0 筆記 (3) Endpoints 的規格 posted in OAuth
SEP 30	OAuth 2.0 筆記 (2) Client 的註冊與認證 posted in OAuth
SEP 30	OAuth 2.0 筆記 (1) 世界觀 posted in OAuth

OAuth 2.0 Spec(s)

- **RFC 6749** - 定義與 **Token 核發** 有關的 Protocol
- **RFC 6750** - 定義 「**Bearer Token**」 怎麼用

RFC 6749: OAuth 2 Protocol

- 定義了 OAuth 2 的**角色**、互動方式
- 定義了**資料**有哪些、怎麼跑
- 定義了**核發**、**換發** Token 等的 Protocol
- 規畫了 4 種常見「適合使用 OAuth 2 的**流程**」

RFC 6750: Bearer Token

- RFC 6749 只規定授權方式， 沒規定 **API 怎麼擋**
- RFC 6750 定義一種叫做“**Bearer**”的 Token Type
- 各種 type 有不同擋法， Bearer 是**直接 == 檢查**
- 一句話解釋：**「直接拿來就能用， 不用簽來簽去」**

Parameters & Data

Client ID / Secret

- 用於 Client 認證 (Client 登入到 Auth. Server)
- 註冊的時候發給，亂碼就行
- HTTP Basic Auth 或包在 Form 裡面 (禁用 URL)

HTTP Basic Auth

ID = **abc**, Secret = **123**



concat with `:`

abc:123



base64()

YWJjOjEyMw==



Basic Auth Header

Authorization: Basic **YWJjOjEyMw==**

Token(s)

- **Access Token** - 打 API 用的
- **Refresh Token** - Access Token 過期可以換發新的

Access Token

- 向 Resource Server 要資料，用 Access Token
- 可以綁一組 Scope
- 可以設期限，可以撤銷 (Revoke)
- 授權流程的目標就是要拿到 Access Token

Refresh Token

- 換發 Access Token 用，只會傳到 Auth. Server
- 綁定一個 Access Token，隨 Access Token 一起核發
- 用過就失效

新的 Access Token 會綁新的 Refresh Token

Scopes

- 用來表示「可以存取哪些資料」的權限範圍
- 如「好友名單」、「相片」
- 可以 A cover B，當然 Set 是最簡單的做法

Scopes

- 申請授權時可以規定
「一定要給」或「沒給就預設值」
- 用空格連起來，例如 "friends_list△photos"
- 但很多網站用逗號 `，`

State

- 用來防止 **CSRF** 攻擊
- Client ↔ Browser ↔ Auth. Server 是很危險ㄉ
- State 傳到 Auth. Server 就要**原封不動返回**
- **Client 驗證之**
- 亂碼值就行，也可以用 **HMAC** 簽

Protocol: 如何取得授權

從 Client 的視角

取得授權的流程

1. Client 向 Res. Owner 取得 “Grant”（授權狀）

整個 Protocol 最難的在這裡

2. Client 用 “Grant” 向 Authorization Server 換 Token

3. Token 拿到了，可以去打 API

最常見的 Scenario

Resource Owner

有網站 Facebook，你希望在那上面的 User

透過 Facebook 的 Authorization Server

給你的網站權限，可以讀取那 User 的資料

Client

Token

Resource Server



大家都串過

※用 Library 串

“Authorization Code Grant Flow”

Auth Code Grant Flow

- Grant 是具體的字串，稱作 Code
- 需要經過 Browser
- 適用於「網站」這種 Client

Resource
Owner

Client

Authorization
Server

※抄自 spec

Resource
Owner

Browser

Client

Authorization
Server

※抄自 spec

Resource
Owner

Browser

Client

Redirection
Endpoint



※抄自 spec

Resource
Owner

Authorization
Server

(A) ID, Redirect URI, Scope, State

Browser

Authorization
Endpoint

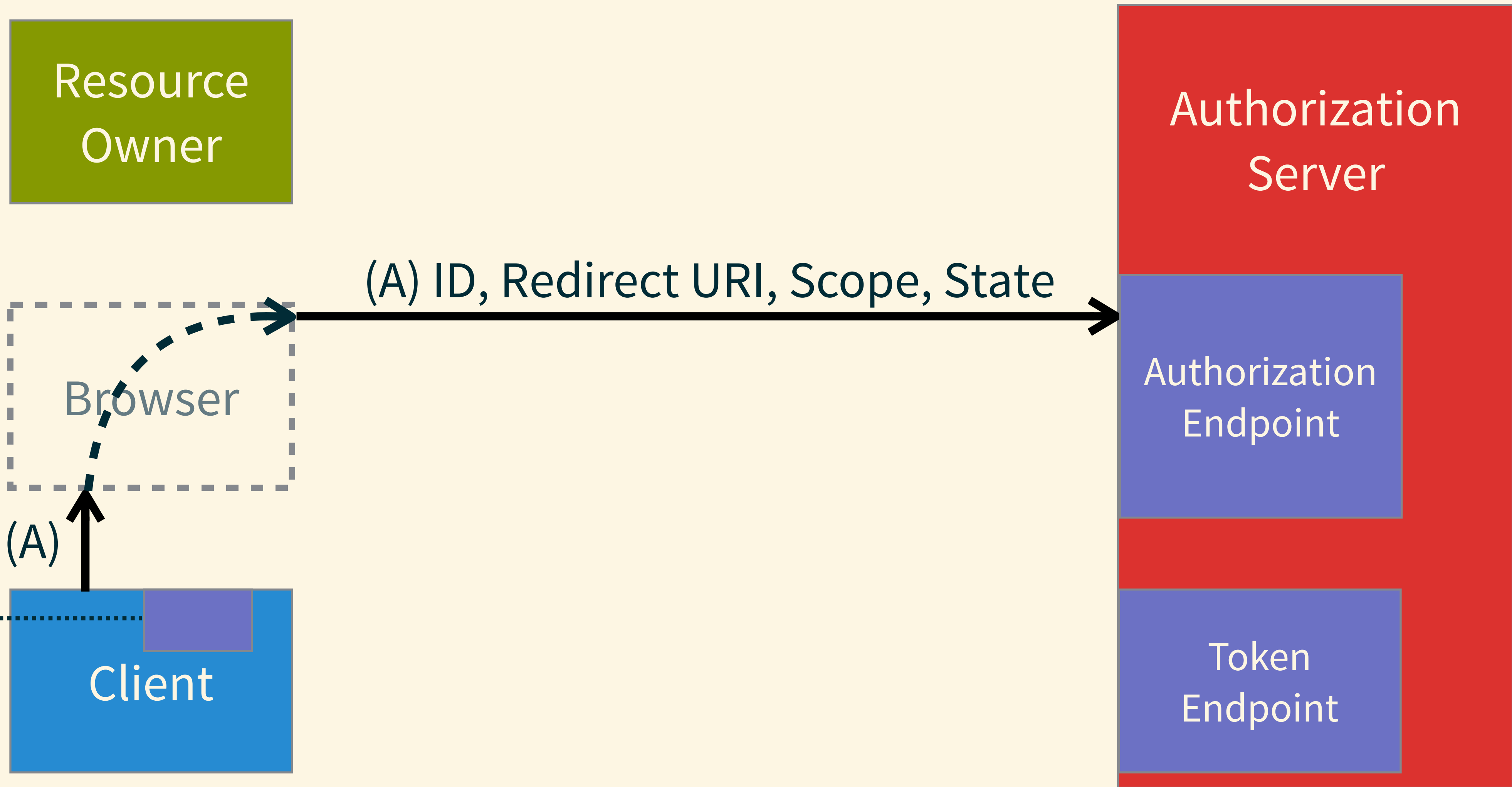
(A)

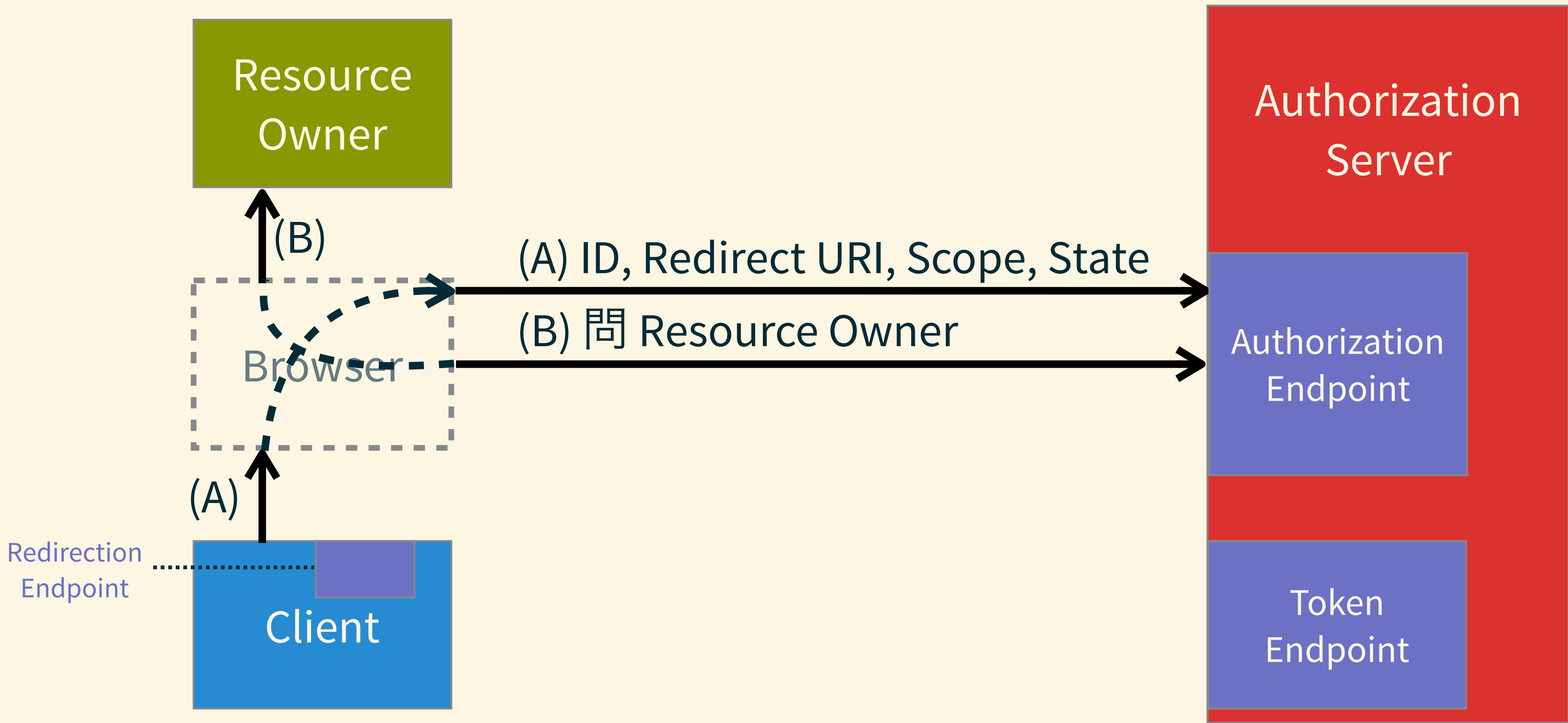
Redirection
Endpoint

Client

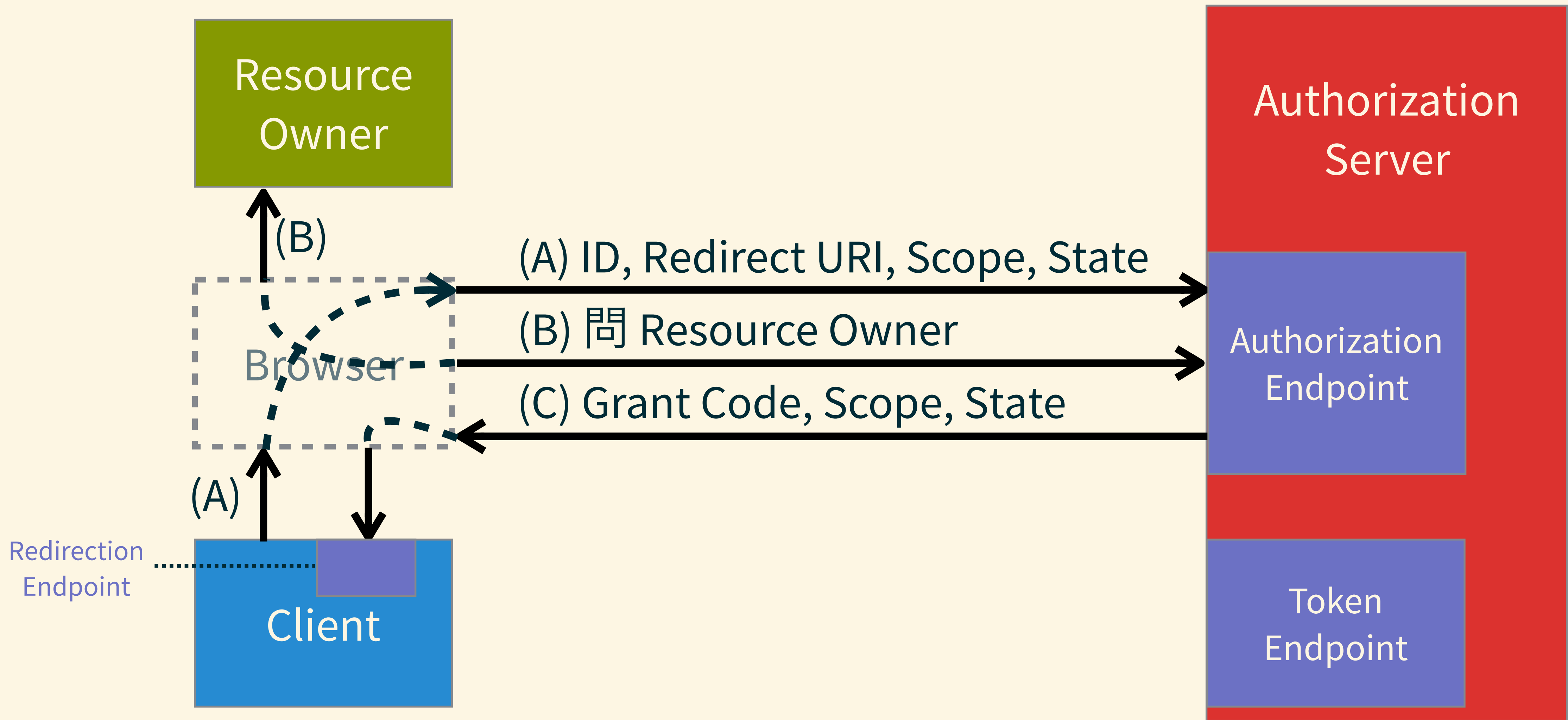
Token
Endpoint

※抄自 spec

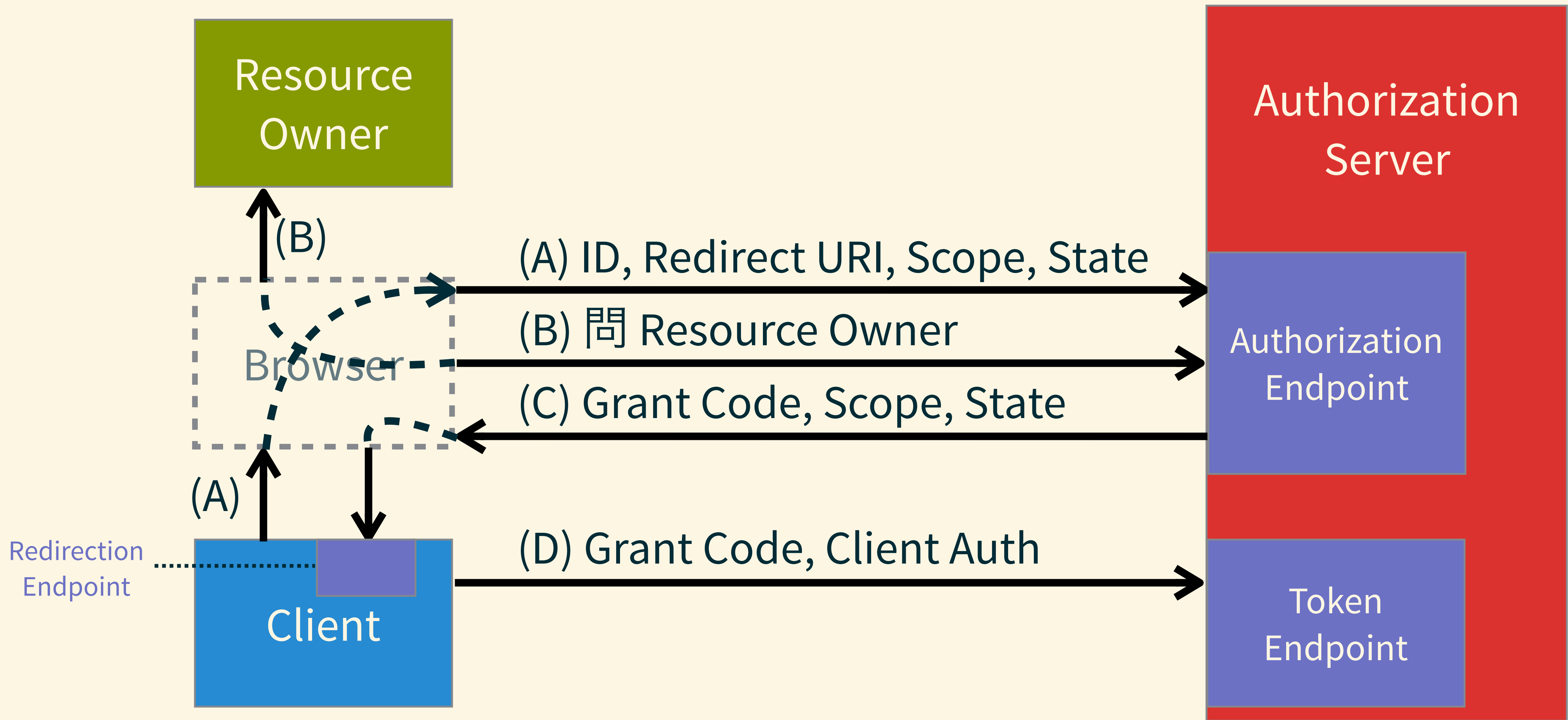




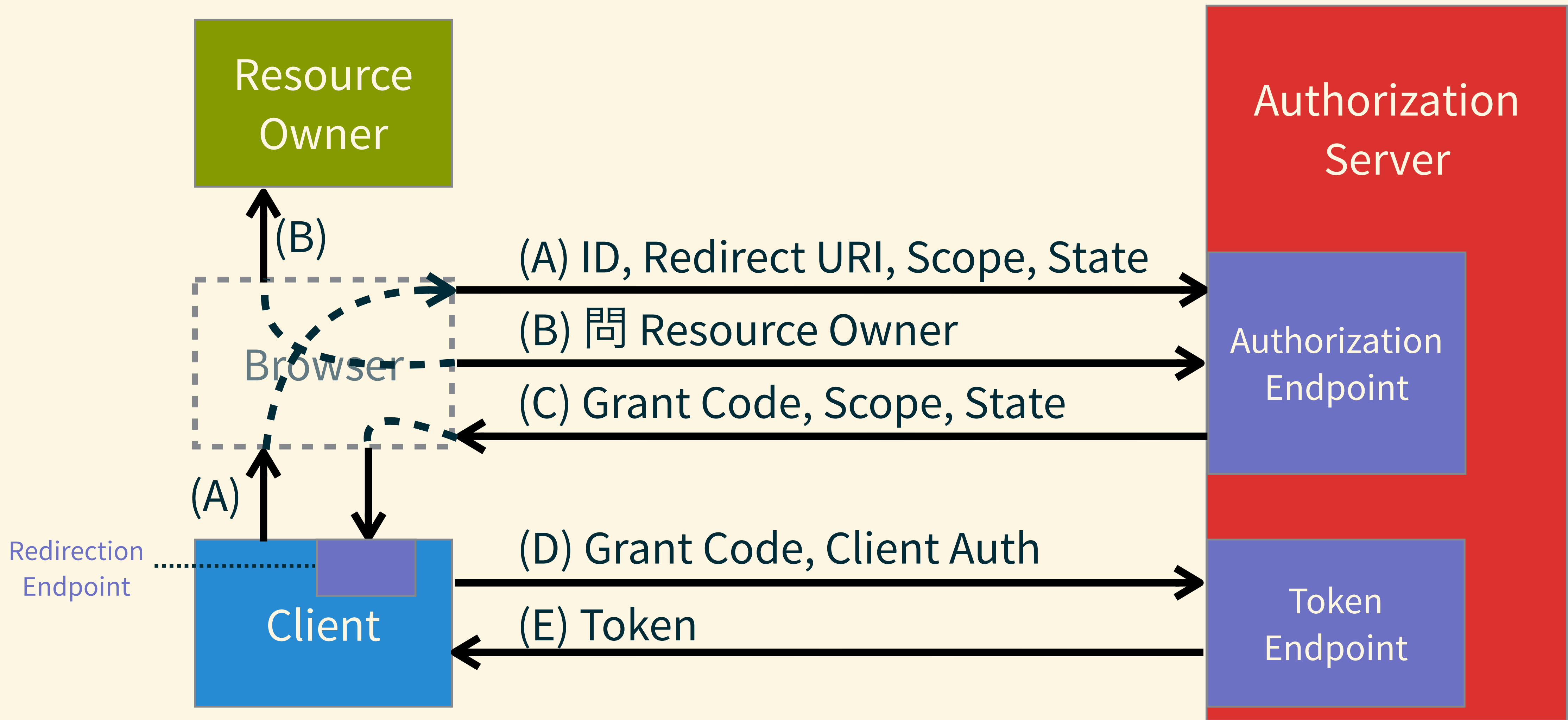
※抄自 spec



※抄自 spec



※抄自 spec

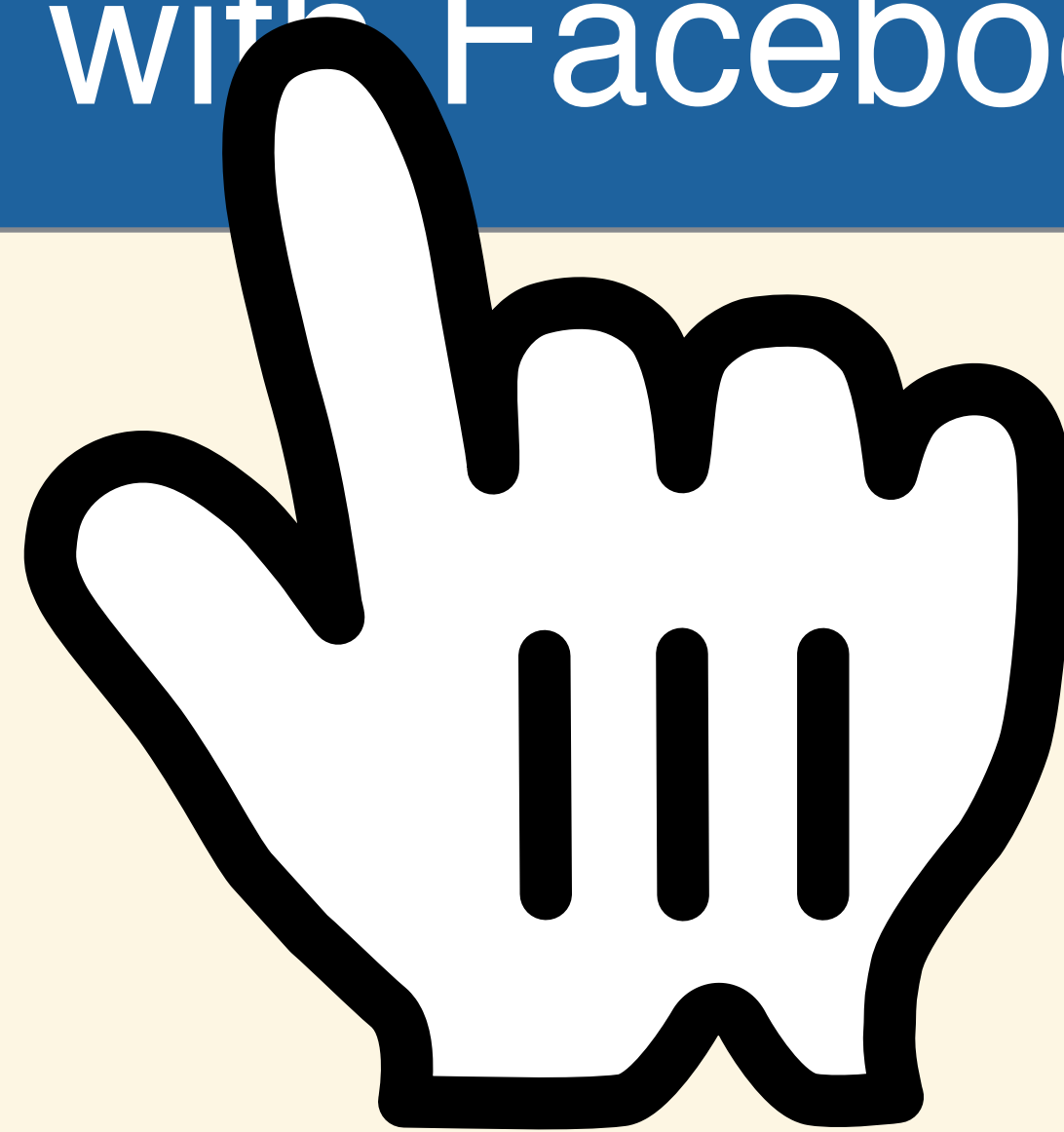


※抄自 spec

(A) 發出授權申請

Client 開一個網址給 User 點

Login with Facebook



(A) 發出授權申請

(GET Request)

```
https://graph.facebook.com/oauth/authorize?  
response_type=code&client_id=567672586646825&  
redirect_uri=http%3A%2F%2Flocalhost  
%3A3000%2Fusers%2Fauth%2Ffacebook  
%2Fcallback&state=446c63696b24b4b6687cdff62ea  
bce3a9c9d6333d7c807e6&scope=email
```

`https://graph.facebook.com/oauth/authorize?`

`response_type=code`

`&client_id=567672586646825`

`&redirect_uri=http://localhost:3000/users/
auth/facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=code`

`&client_id=567672586646825`

`&redirect_uri=http://localhost:3000/users/
auth/facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=code` 表示 Authorization Code Grant Flow

`&client_id=567672586646825`

`&redirect_uri=http://localhost:3000/users/
auth/facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=code` 表示 Authorization Code Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3000/users/auth/facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=code` 表示 Authorization Code Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3000/users/auth/facebook/callback` Redirect URI

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=code` 表示 Authorization Code Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3000/users/auth/facebook/callback` Redirect URI

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6` Client 的 State (防 CSRF)

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=code` 表示 Authorization Code Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3000/users/auth/facebook/callback` Redirect URI

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6` Client 的 State (防 CSRF)

`&scope=email` 想要請求的權限範圍 (Scopes)

Authorization Server 的動作

1. 根據 Client ID 找 Client
2. 確認 Redirect URI 跟事先註冊的相符
 - 若不正確則要噴錯誤，不可轉回該 URI
3. 確認想申請的 scope 正確（格式、內容等）
4. 沒問題就問 Resource Owner 要不要授權 (B)

(B) Auth. Server 問 Res. Owner

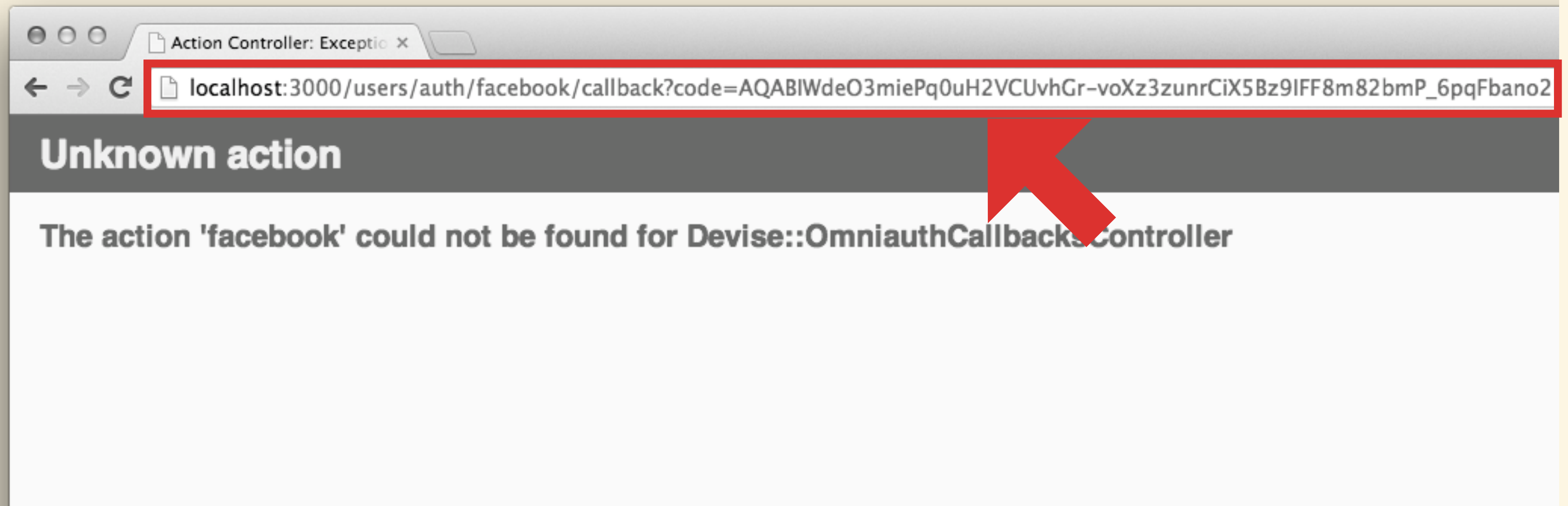


Authorization Server 的動作

- 若允許，則轉回 Client，附上 Grant Code
- 若不允許，則轉回 Client，並附上錯誤訊息
- 若剛剛有傳 state 過來，則原封不動附上

(C) Client 收到 Grant Code

(GET Request)



HTTP/1.1 302 Found

Location: [http://localhost:3000/users/auth/facebook/callback?
code=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP\(...\) &state=446c636
96b24b4b6687cdff62eabce3a9c9d6333d7c807e6](http://localhost:3000/users/auth/facebook/callback?code=AQABIWde03miePq0uH2VCUvhGr-voXz3zunrCiX5Bz9IFF8m82bmP(...) &state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6)

```
http://localhost:3000/users/auth/facebook/  
callback?
```

```
code=AQABIWde03miePq0uH2VCUvhGr-  
voXz3zunrCiX5Bz9IFF8m82bmP(...)
```

```
&state=446c63696b24b4b6687cdff62eabce3a9c9d63  
33d7c807e6
```

Redirection Endpoint

```
http://localhost:3000/users/auth/facebook/  
callback?
```

```
code=AQABIWde03miePq0uH2VCUvhGr-  
voXz3zunrCiX5Bz9IFF8m82bmP(...)
```

```
&state=446c63696b24b4b6687cdff62eabce3a9c9d63  
33d7c807e6
```


Redirection Endpoint

`http://localhost:3000/users/auth/facebook/
callback?`

授權狀 (Grant Code)

`code=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

Redirection Endpoint

`http://localhost:3000/users/auth/facebook/callback?`

授權狀 (Grant Code)

`code=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

原封不動的 State 值

Client 的動作

- 先檢查 **state** 跟 session 裡面存的有沒有一致
- 沒問題就去換 Token 了

(D) 拿 Code 換 Token

- Client 在後台做這件事
- POST 到 Auth. Server 的 Token Endpoint 換 Token
- Client Authentication
- `grant_type=code, code, redirect_uri, client_id` (僅 Public Client)

※ 事實上 Facebook 用 GET，不合標準，下圖請自行腦補

```
POST /oauth/access_token HTTP/1.1
```

```
Host: graph.facebook.com
```

```
Authorization: Basic YWJjOjEyMw==
```

```
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=authorization_code
```

```
&code=AQABIWde03miePq0uH2VCUvhGr-  
voXz3zunrCiX5Bz9IFF8m82bmP(...)
```

```
&redirect_uri=http%3A%2F%2Flocalhost  
%3A3000%2Fusers%2Fauth%2Ffacebook%2Fcallback
```

※ 事實上 Facebook 用 GET，不合標準，下圖請自行腦補

Token Endpoint

POST /oauth/access_token HTTP/1.1

Host: graph.facebook.com

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code

&code=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)

&redirect_uri=http%3A%2F%2Flocalhost
%3A3000%2Fusers%2Fauth%2Ffacebook%2Fcallback

※ 事實上 Facebook 用 GET，不合標準，下圖請自行腦補

Token Endpoint

POST /oauth/access_token HTTP/1.1

Host: graph.facebook.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code

&code=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)

&redirect_uri=http%3A%2F%2Flocalhost
%3A3000%2Fusers%2Fauth%2Ffacebook%2Fcallback

※ 事實上 Facebook 用 GET，不合標準，下圖請自行腦補

Token Endpoint

POST /oauth/access_token HTTP/1.1

Host: graph.facebook.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

表示「我拿到的授權狀是 Grant Code」

grant_type=authorization_code

&code=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)

&redirect_uri=http%3A%2F%2Flocalhost
%3A3000%2Fusers%2Fauth%2Ffacebook%2Fcallback

※ 事實上 Facebook 用 GET，不合標準，下圖請自行腦補

Token Endpoint

POST /oauth/access_token HTTP/1.1

Host: graph.facebook.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

表示「我拿到的授權狀是 Grant Code」

grant_type=authorization_code

&code=AQABIWde03miePq0uH2VCUvhGr-voXz3zunrCiX5Bz9IFF8m82bmP(...) Grant Code

&redirect_uri=http%3A%2F%2Flocalhost%3A3000%2Fusers%2Fauth%2Ffacebook%2Fcallback

※ 事實上 Facebook 用 GET，不合標準，下圖請自行腦補

Token Endpoint

POST /oauth/access_token HTTP/1.1

Host: graph.facebook.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

表示「我拿到的授權狀是 Grant Code」

grant_type=authorization_code

&code=AQABIWde03miePq0uH2VCUvhGr-voXz3zunrCiX5Bz9IFF8m82bmP(...) Grant Code

&redirect_uri=http%3A%2F%2Flocalhost%3A3000%2Fusers%2Fauth%2Ffacebook%2Fcallback 原 Redirect URI

Authorization Server 的動作

- 確認 Client Authentication (ID / Secret) 正確
- 找到該 Grant Code , 確認其 Redirect URI 一模一樣
- 沒問題的話就發 Token

(E) 發 Token

- Client 的 Token Request 的 Response
- 是 JSON Response
- 可以一併發 Refresh Token
- User 授權的 Scope 若與申請時不一致則要附上

HTTP/1.1 200 OK

Content-Type: application/**json**; charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

"access_token": "**2YotnFZFEjr1zCsicMWpAA**",

"token_type": "**Bearer**",

"expires_in": **3600**,

"refresh_token": "**tGzv3J0kF0XG5Qx2TlKWIA**",

"scopes": "**email**"

}

HTTP/1.1 200 OK

Content-Type: application/**json**; charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

Access Token

"access_token": "**2YotnFZFEjr1zCsicMWpAA**",

"token_type": "**Bearer**",

"expires_in": **3600**,

"refresh_token": "**tGzv3J0kF0XG5Qx2TlKWIA**",

"scopes": "**email**"

}

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json;charset=UTF-8
```

```
Cache-Control: no-store
```

```
Pragma: no-cache
```

```
{  
    "access_token": "2YotnFZFEjr1zCsicMWpAA",  
    "token_type": "Bearer" 表示這個 Token 是 Bearer Token  
    "expires_in": 3600,  
    "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",  
    "scopes": "email"  
}
```

HTTP/1.1 200 OK

Content-Type: application/**json**; charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

```
{  
    "access_token": "2YotnFZFEjr1zCsicMWpAA",  
    "token_type": "Bearer" 表示這個 Token 是 Bearer Token  
    "expires_in": 3600, 3600 秒之後過期  
    "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",  
    "scopes": "email"  
}
```



```
HTTP/1.1 200 OK
```

```
Content-Type: application/json;charset=UTF-8
```

```
Cache-Control: no-store
```

```
Pragma: no-cache
```

```
{  
    "access_token": "2YotnFZFEjr1zCsicMWpAA",  
    "token_type": "Bearer" 表示這個 Token 是 Bearer Token  
    "expires_in": 3600, 3600 秒之後過期  
    "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",  
    "scopes": "email"  Access Token 對應的 Refresh Token  
}
```

```
HTTP/1.1 200 OK
```

```
Content-Type: application/json;charset=UTF-8
```

```
Cache-Control: no-store
```

```
Pragma: no-cache
```

```
{  
  "access_token": "2YotnFZFEjr1zCsicMWpAA",  
  "token_type": "Bearer" 表示這個 Token 是 Bearer Token  
  "expires_in": 3600, 3600 秒之後過期  
  "refresh_token": "tGzv3J0kF0XG5Qx2TlKWIA",  
  "scopes": "email"  Access Token 對應的 Refresh Token  
}
```

若 scope 與申請時
不一致則要附上

手機 App / 桌面 App

- 視為 **Public** , Client **Authentication** 不可信
- Token Request 要附上 Client ID
確保 Token 發給正確的 Client
- App 聽固定 URI , 如
`kktix-app:oauth2-callback`

What about JavaScript App

- “User-Agent-Based Client”
- Public Client
- Redirect Endpoint 也沒辦法聽自訂 Protocol
- Auth. Code Grant Flow 不可用，怎麼辦？

“Implicit Grant Flow”

Implicit Grant Flow

- 專門給 **Public** Client 用
- Grant 不傳到 Client，**直接發 Token**
- 不經過 Token Endpoint
- 發的 Token 通常是**短效期**，降低被偷的風險
- **不給 Refresh Token**，過期重新跑申請流程

Resource
Owner

Authorization
Server

Client (JS)

※抄自 spec

Resource
Owner

Browser

Client (JS)

Authorization
Server

※抄自 spec

Resource
Owner

Browser

Client (JS)

Authorization
Server

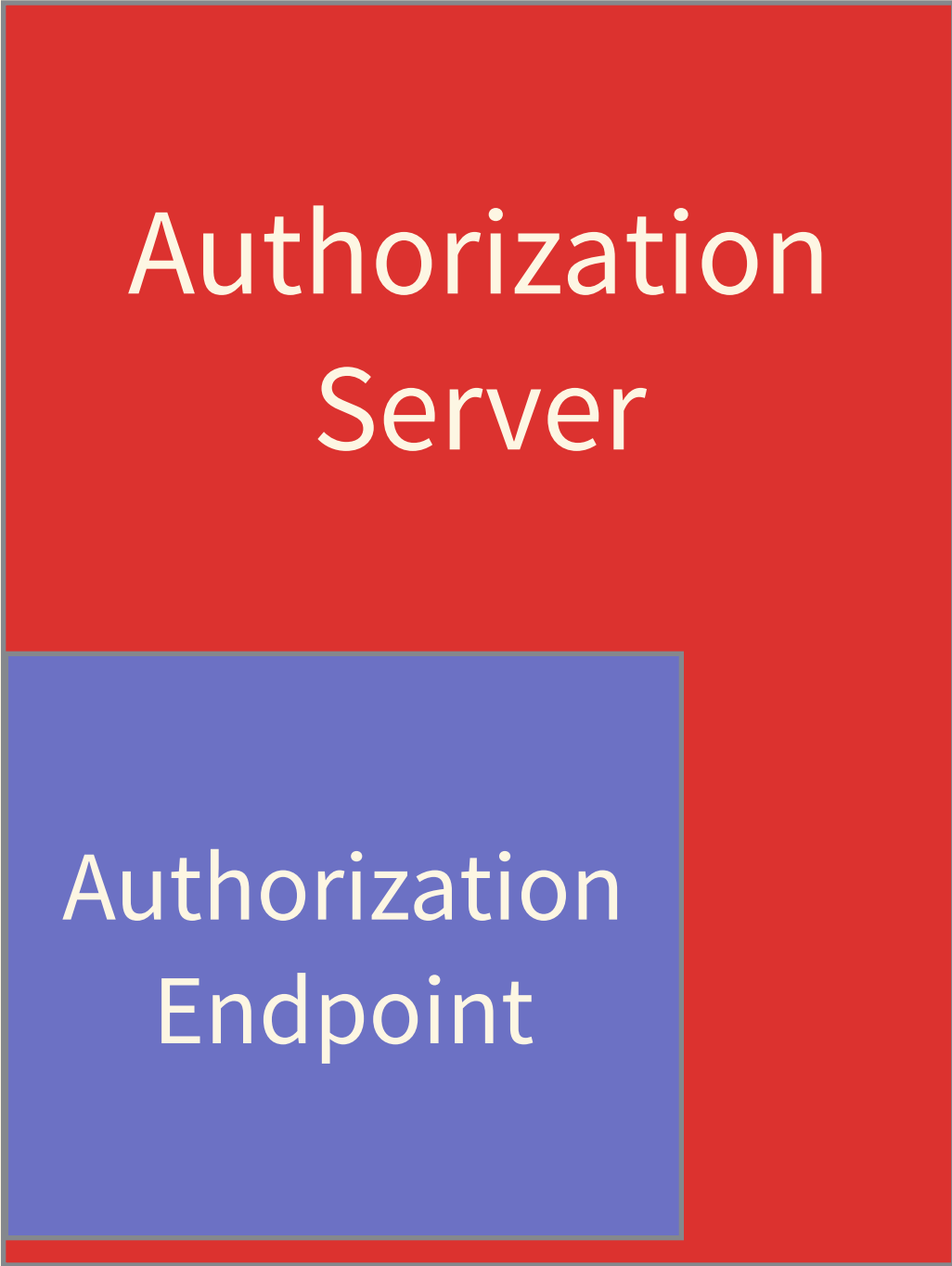
Client'
(Web Server)

※抄自 spec

Resource
Owner

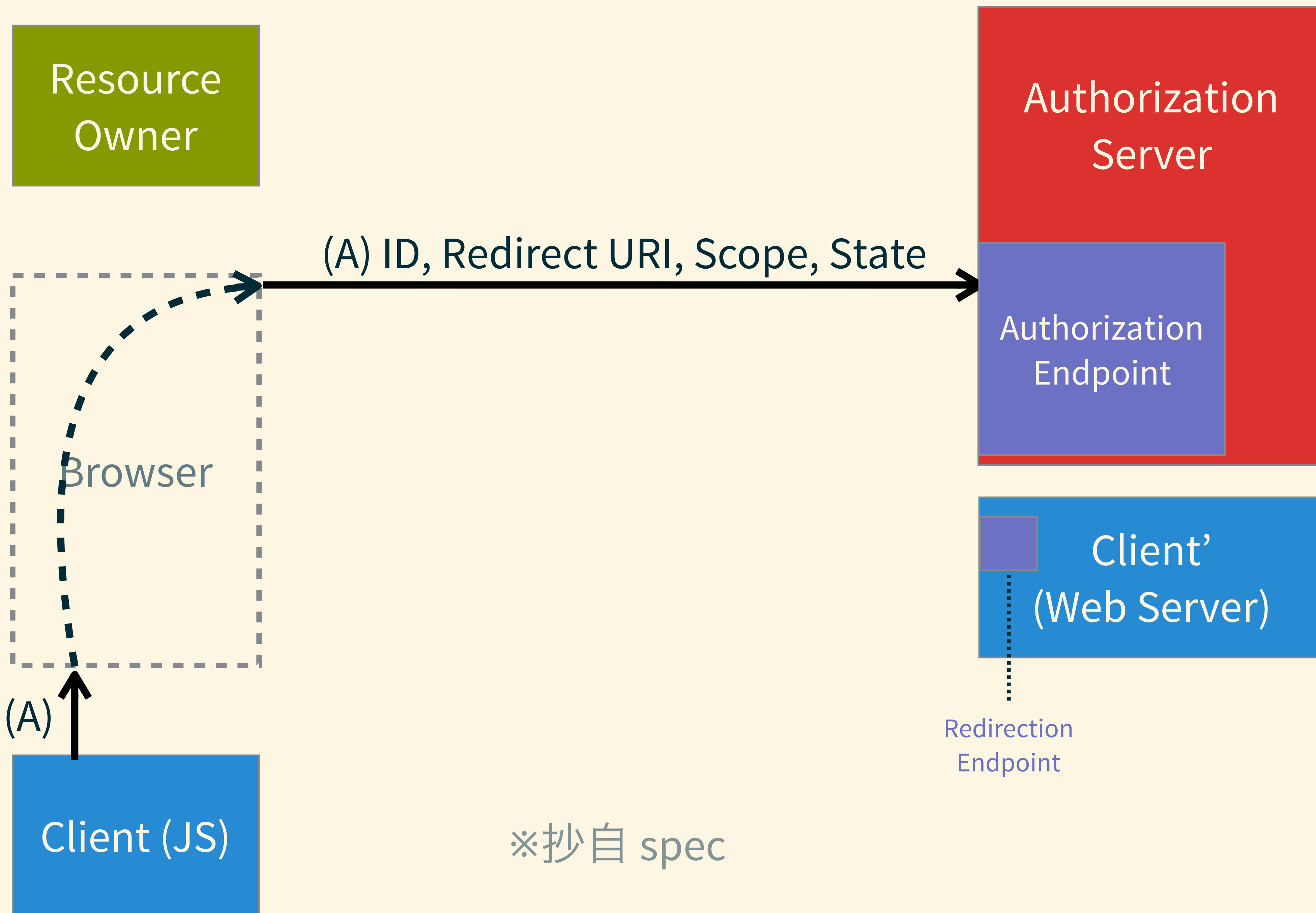
Browser

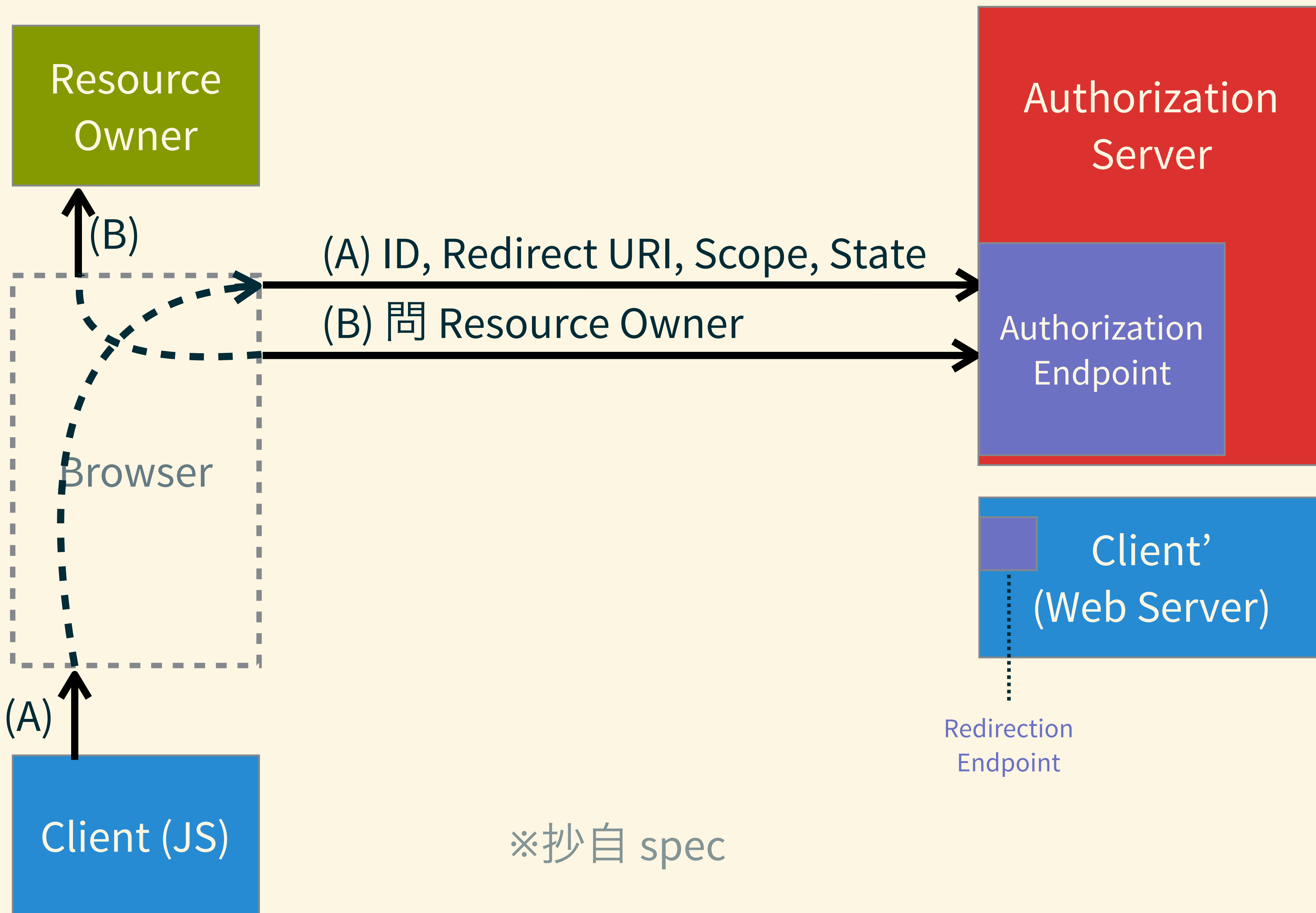
Client (JS)

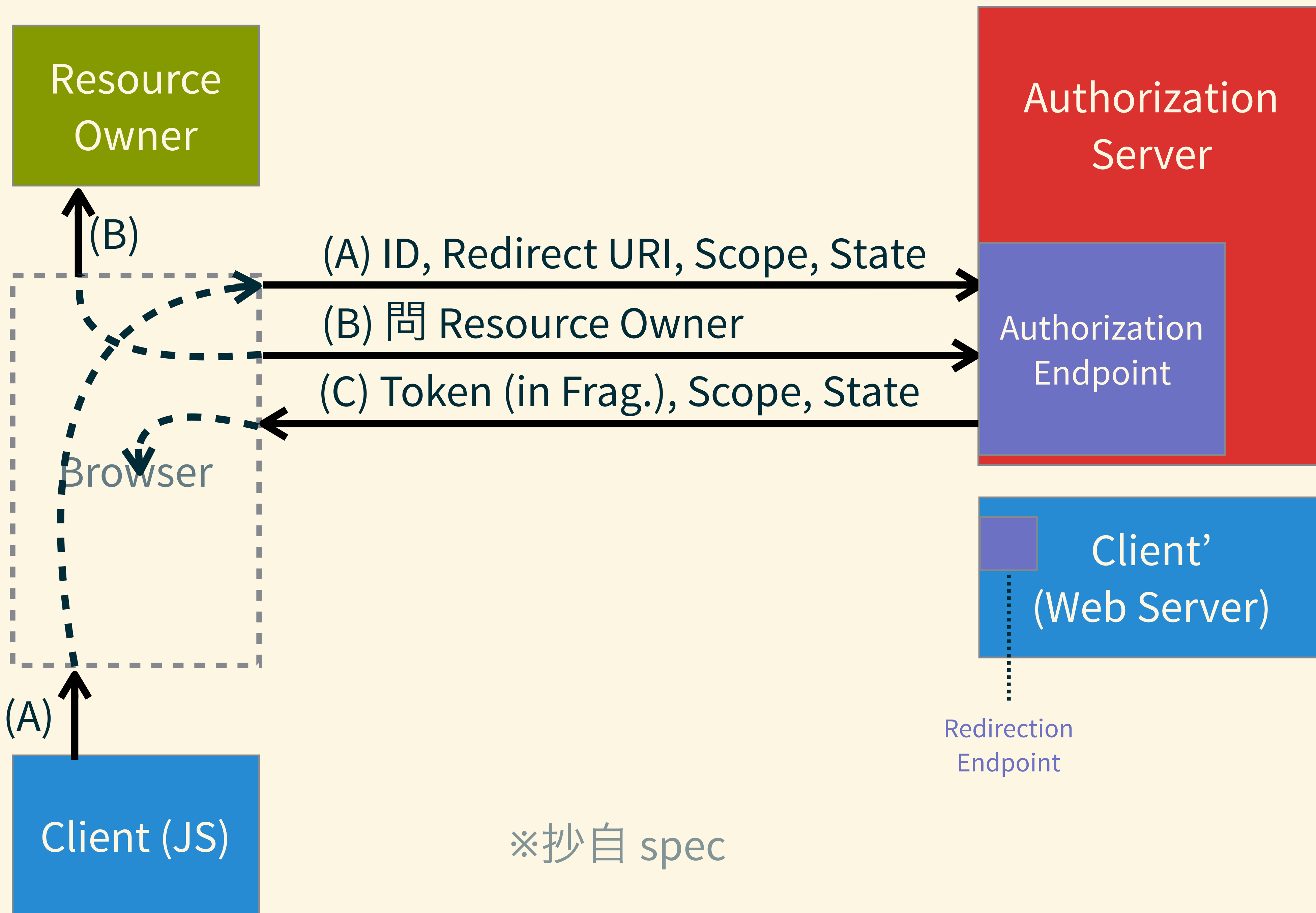


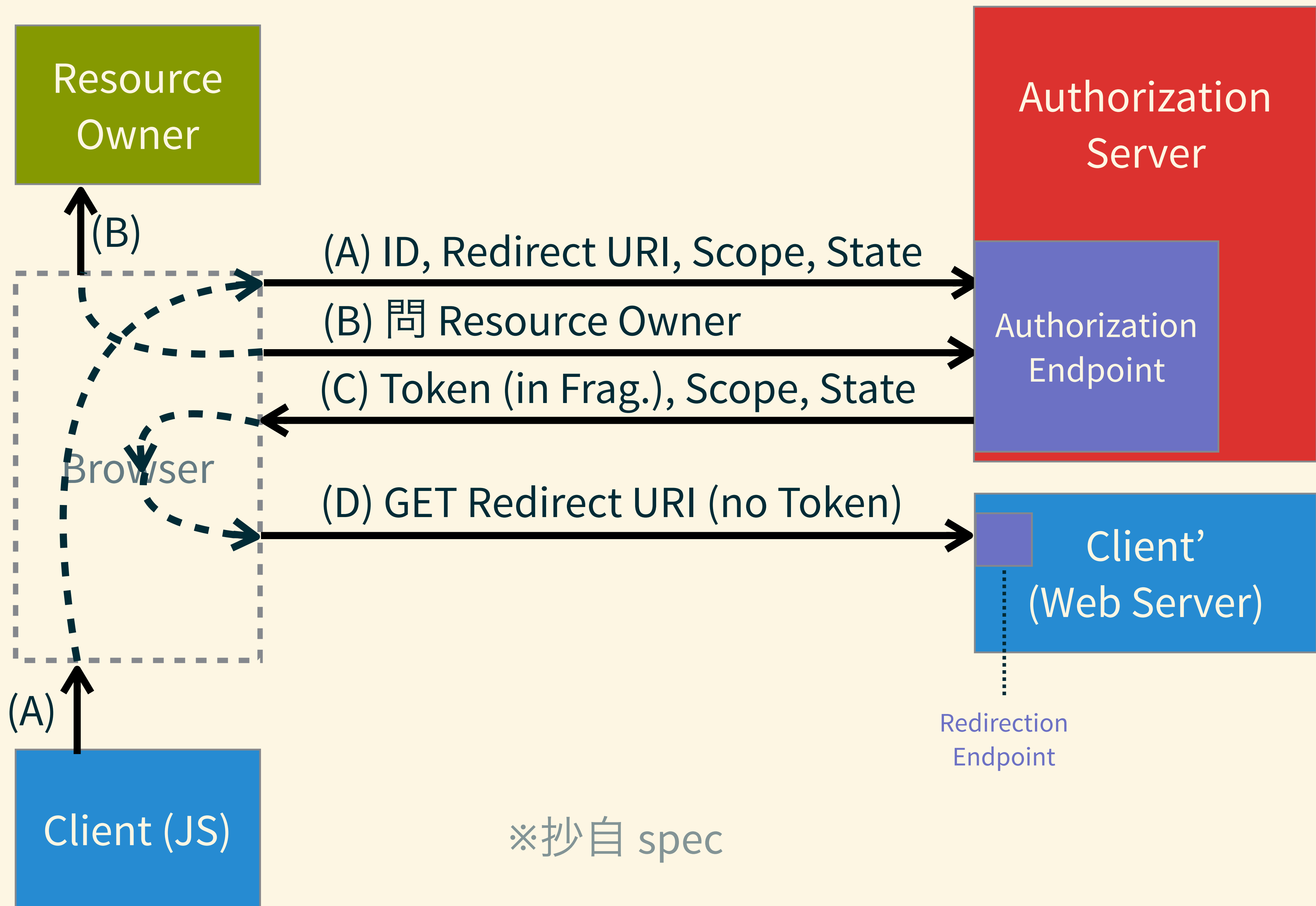
Redirection
Endpoint

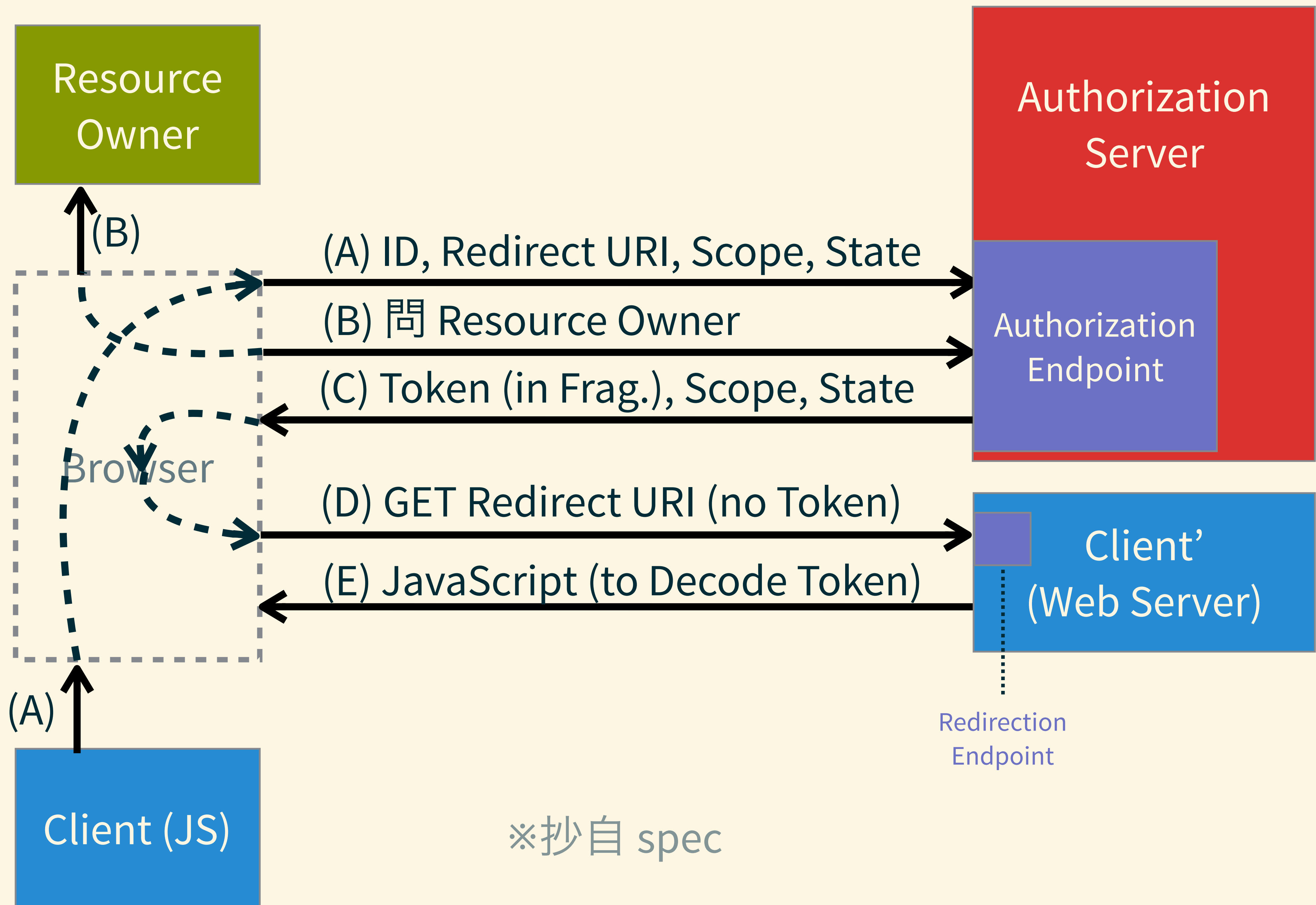
※抄自 spec

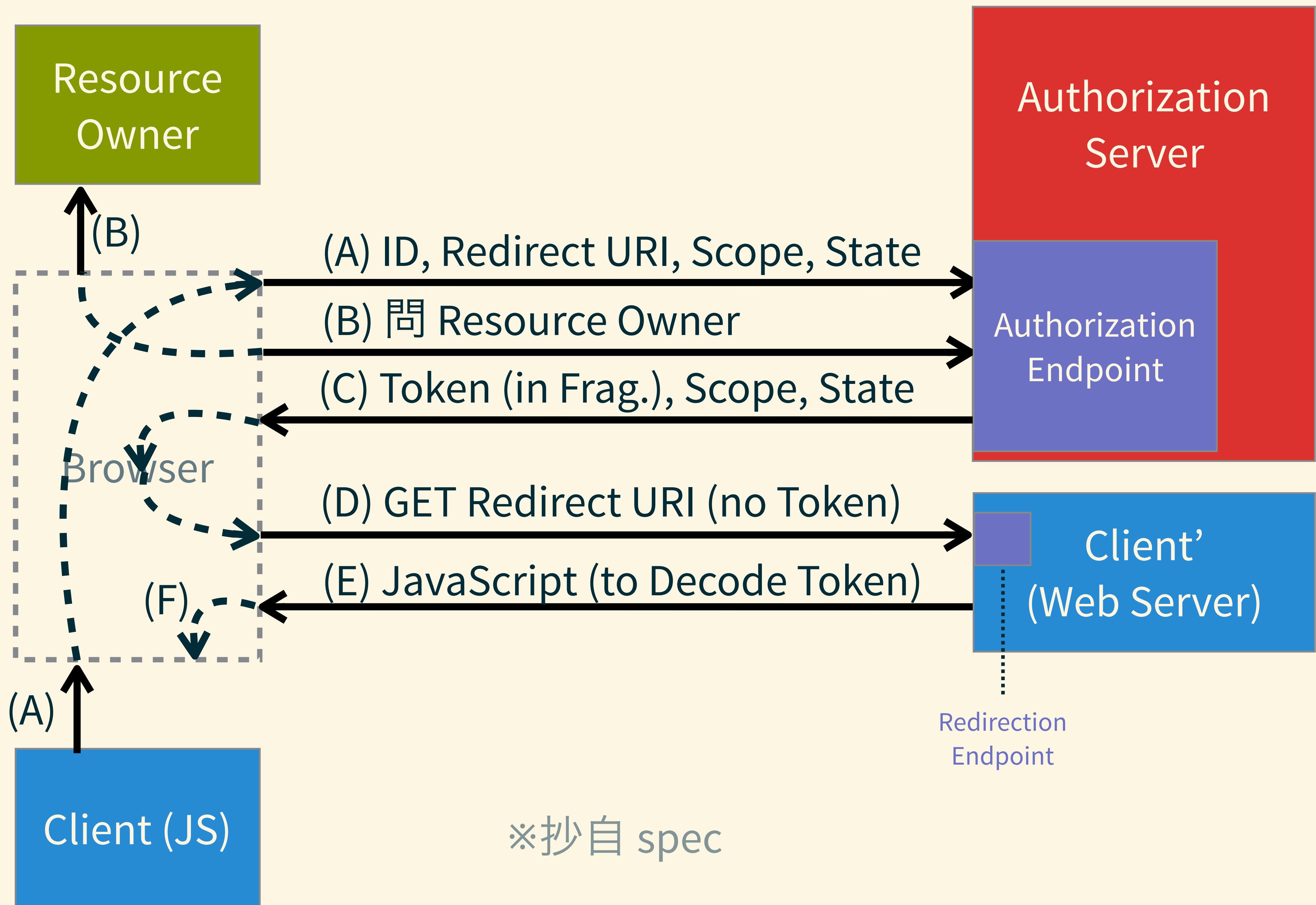


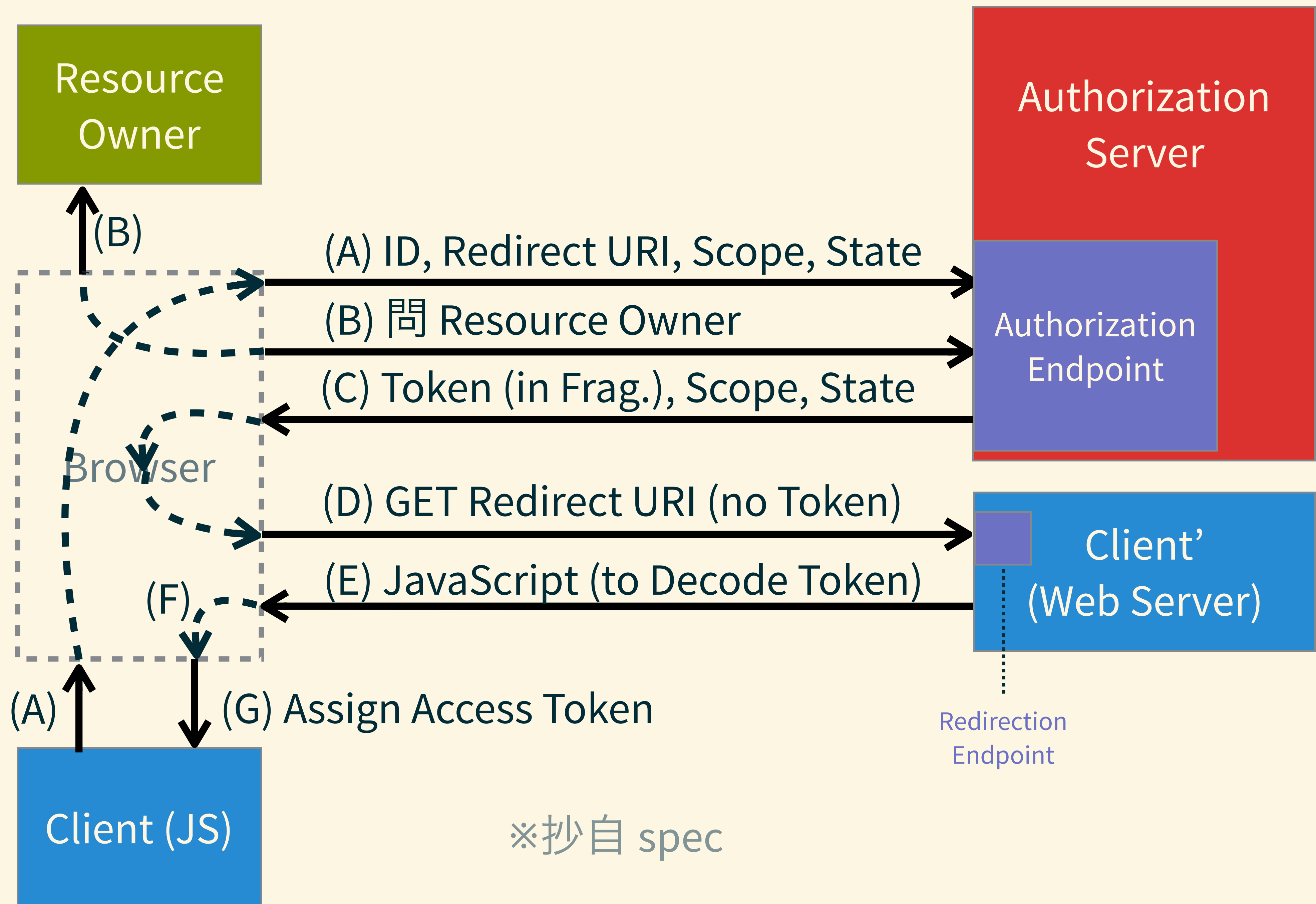


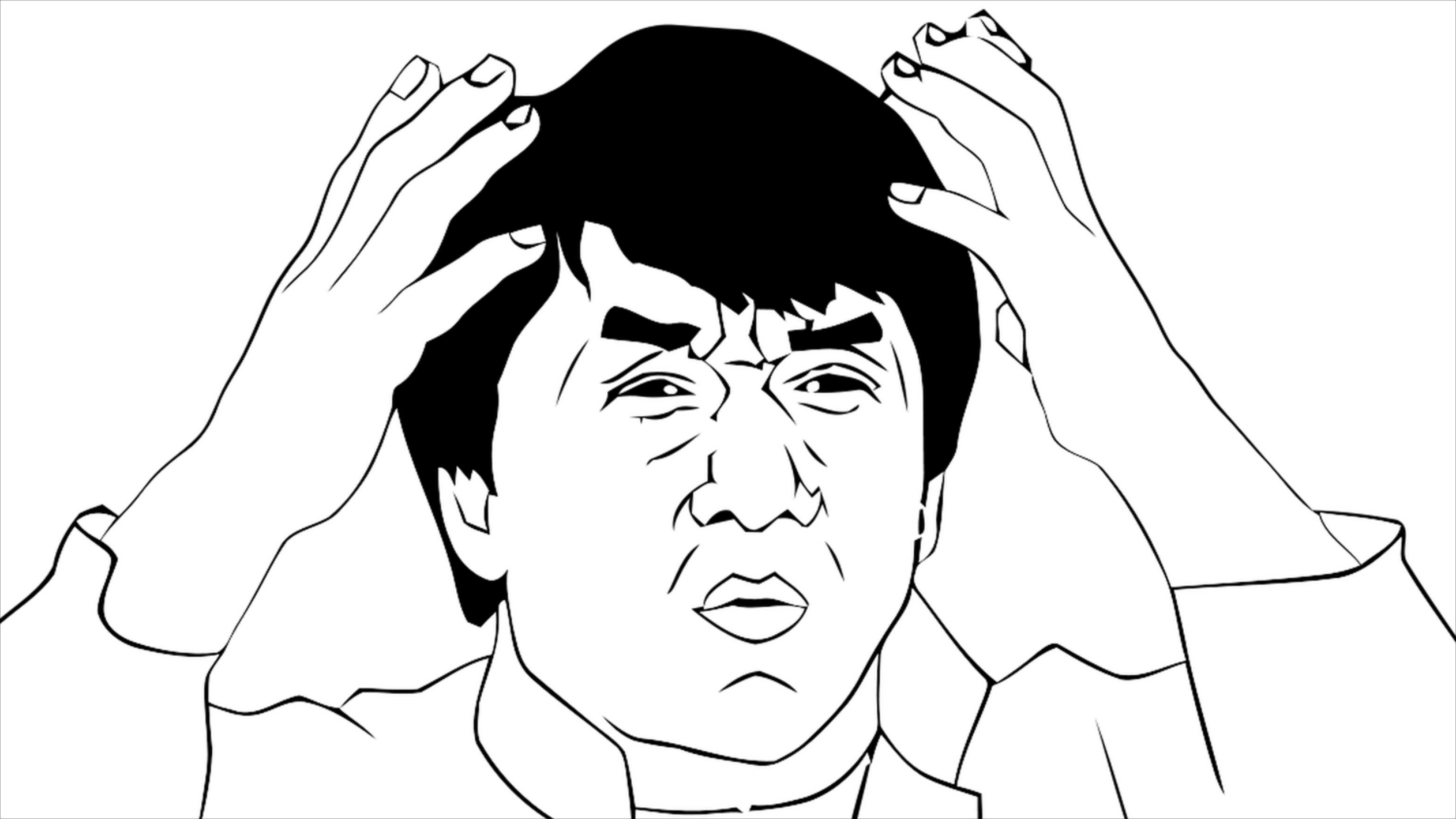












(C) ~ (G) 花生省魔術

- (C) Auth. Server 叫 Browser 轉回 Redirect URI
- `https://client.com/oauth2/callback#access_token=...`
- (D) Browser 會去 GET 該 URL，但不含 #（天性）
- Token 自動保留在 Browser 的 `Fragment Part` 裡面

(C) ~ (G) 花生省魔術

- (E) 再丟回一個 JavaScript 用來解出 # 裡面的 Token
- (F) Browser 自動 run 這個 JavaScript 解出 Token
- (G) 把解出來的 Token 丟給 JavaScript App

所以為什麼需要 Client' 分身

- JavaScript 不能開 Redirection Endpoint
所以用 Web Server 開
- 裡面就是一個 JavaScript code 用來解出 Token
並傳給 JavaScript App (真 Client)

`https://graph.facebook.com/oauth/authorize?`

`response_type=token`

`&client_id=567672586646825`

`&redirect_uri=http://localhost:3001/jsapp/
facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=token`

`&client_id=567672586646825`

`&redirect_uri=http://localhost:3001/jsapp/
facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=token` 表示 Implicit Grant Flow

`&client_id=567672586646825`

`&redirect_uri=http://localhost:3001/jsapp/
facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=token` 表示 Implicit Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3001/jsapp/
facebook/callback`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=token` 表示 Implicit Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3001/jsapp/
facebook/callback` Redirect URI

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=token` 表示 Implicit Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3001/jsapp/
facebook/callback` Redirect URI

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6` Client 的 State (防 CSRF)

`&scope=email`

Authorization Endpoint

`https://graph.facebook.com/oauth/authorize?`

`response_type=token` 表示 Implicit Grant Flow

`&client_id=567672586646825` Client ID

`&redirect_uri=http://localhost:3001/jsapp/facebook/callback` Redirect URI

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6` Client 的 State (防 CSRF)

`&scope=email` 想要請求的權限範圍 (Scopes)

HTTP/1.1 302 Found

Location: [http://localhost:3001/jsapp/facebook/callback#access_token=AQABIWde03miePq0uH2VCUvhGr\(...\) &state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6 &token_type=bearer&expires_in=3600](http://localhost:3001/jsapp/facebook/callback#access_token=AQABIWde03miePq0uH2VCUvhGr(...) &state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6 &token_type=bearer&expires_in=3600)

`http://localhost:3000/jsapp/facebook/
callback#`

`token=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&token_type=Bearer`

`&expires_in=3600`

Redirection Endpoint

`http://localhost:3000/jsapp/facebook/
callback#`

`token=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&token_type=Bearer`

`&expires_in=3600`

Redirection Endpoint

`http://localhost:3000/jsapp/facebook/
callback#`

`token=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)` Access Token

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

`&token_type=Bearer`

`&expires_in=3600`

Redirection Endpoint

`http://localhost:3000/jsapp/facebook/
callback#`

`token=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)` Access Token

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6` 原封不動的 State 值

`&token_type=Bearer`

`&expires_in=3600`

Redirection Endpoint

`http://localhost:3000/jsapp/facebook/
callback#`

`token=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)` Access Token

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6` 原封不動的 State 值

`&token_type=Bearer` 表示這個 Token 是 Bearer Token

`&expires_in=3600`

Redirection Endpoint

`http://localhost:3000/jsapp/facebook/
callback#`

`token=AQABIWde03miePq0uH2VCUvhGr-
voXz3zunrCiX5Bz9IFF8m82bmP(...)` Access Token

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6` 原封不動的 State 值

`&token_type=Bearer` 表示這個 Token 是 Bearer Token

`&expires_in=3600` 3600 秒之後過期

Redirection Endpoint 的內容

```
<script>  
var access_token = [extract token from frag];  
// store token in local storage or cookie  
</script>
```

Implicit Grant Flow Usage

- Facebook JavaScript SDK
- 手機 App、桌面 App 也適用

Facebook: **WebView 監聽固定 Redirect URI**

抓出 #access_token


特別注意事項

- 為降低 Token 被偷風險，通常會發短效期的 Token
- 因為可能偽造 Token 餵給 Client，所以必須驗證之
- Facebook: “Token Debug Endpoint”

What if 自產自用?

- 自己寫的 script 想要用 Token 去打 API
- Server App 已經存了帳號密碼，想改成 OAuth 2

**“Resource Owner Password
Credentials Grant Flow”**



Resource
Owner

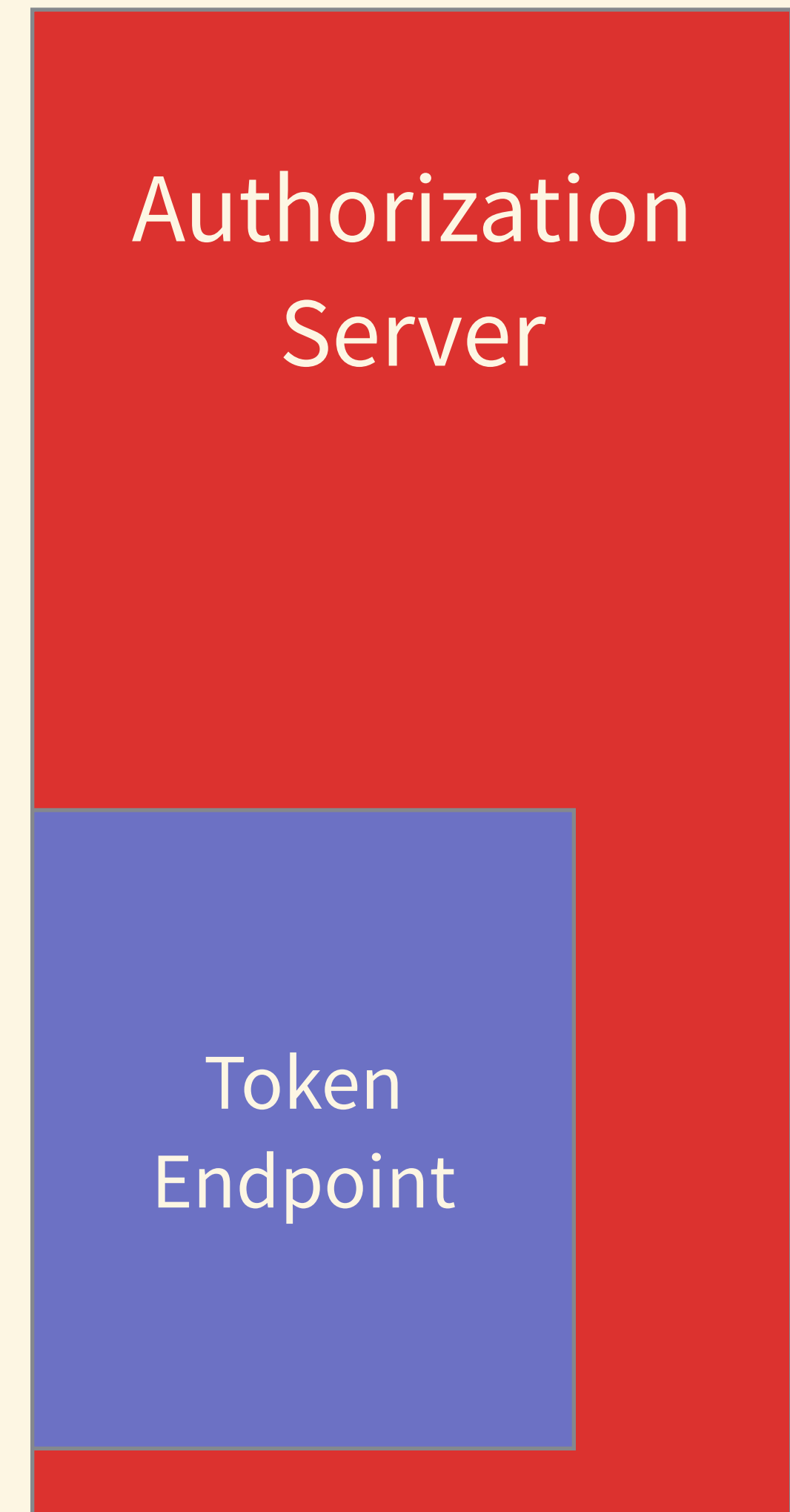
Client

Authorization
Server

※抄自 spec

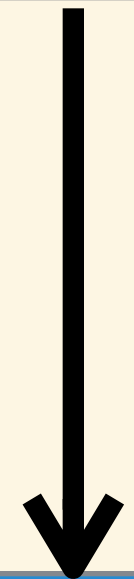
Resource
Owner

Client



※抄自 spec

Resource
Owner



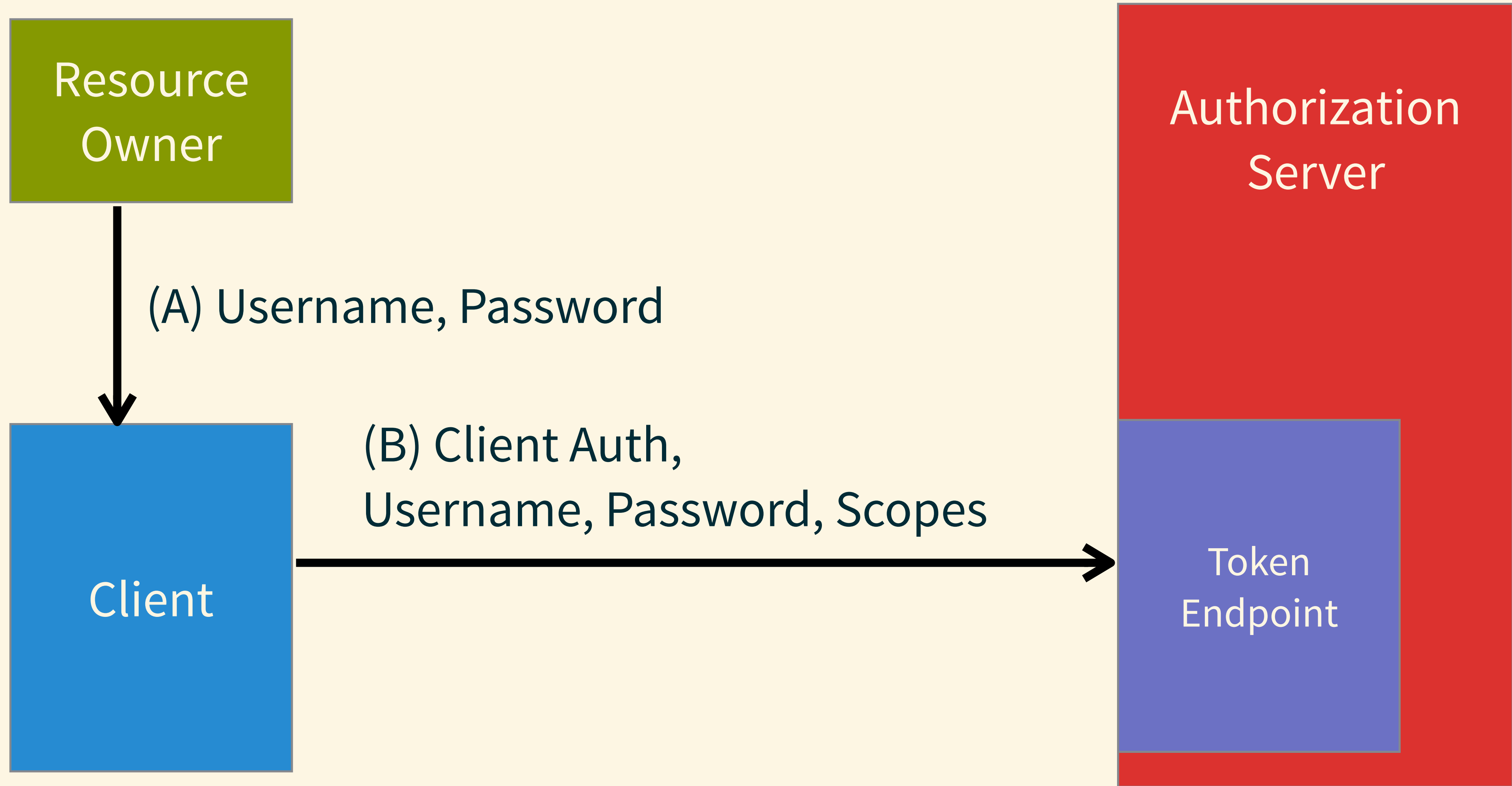
(A) Username, Password

Client

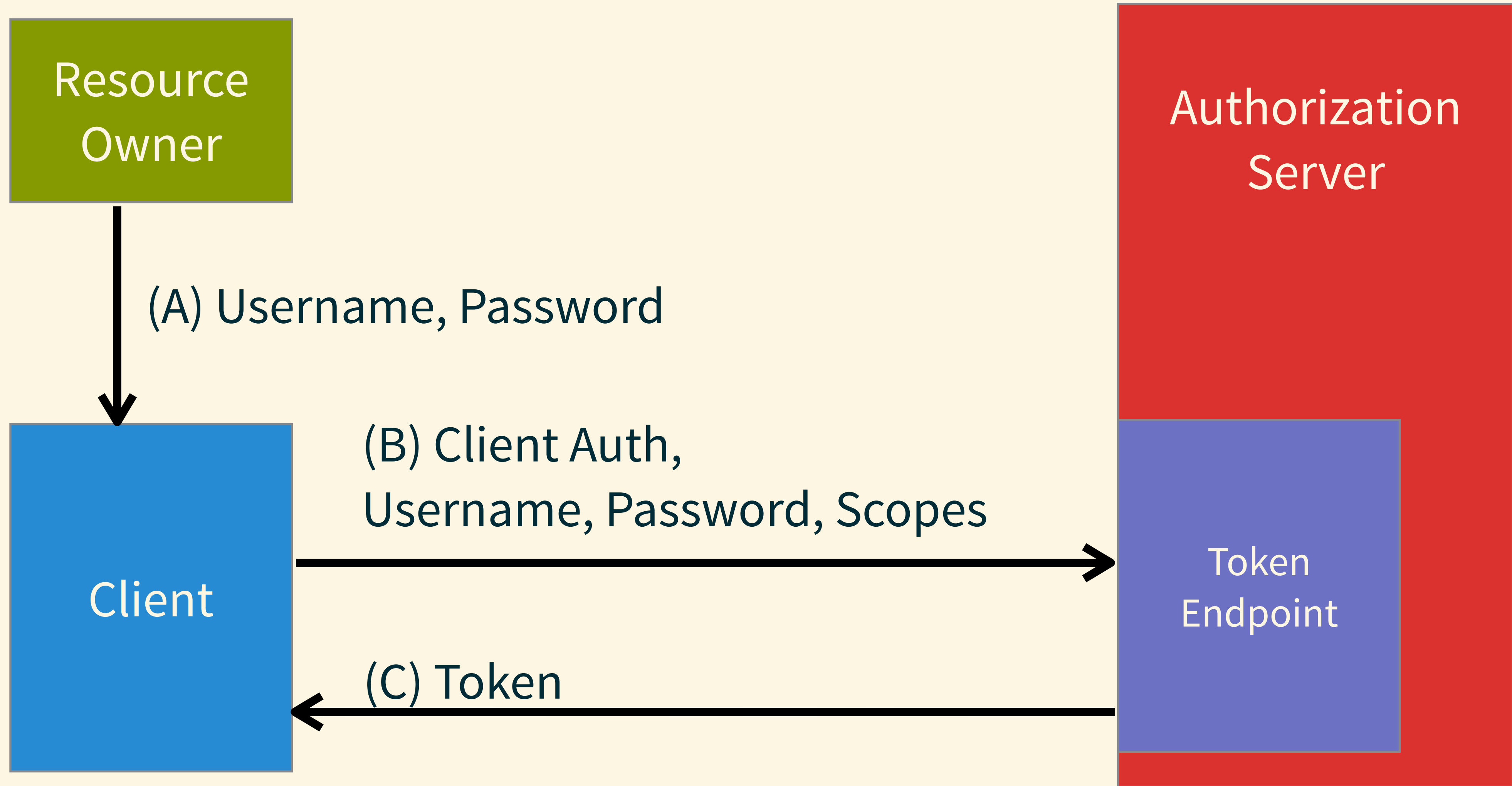
Authorization
Server

Token
Endpoint

※抄自 spec



※抄自 spec



※抄自 spec

Resource Owner Password Credentials Grant Flow

- 需要是 Resource Owner 高度信賴 Client
- 作業系統內建的應用程式（OS X 的 Twitter 整合）
- 官方應用程式（GitHub.app）
- 且其他別的流程都不適用

Resource Owner Password Credentials Grant Flow

- No “state” 因為不經過 Browser
- No “Redirect URI” 因為沒有 redirection
- 直接 POST 去 Token Endpoint 拿 Token

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=password

&username=chitsaou

&password=12345678

&scope=email

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=password

&username=chitsaou

&password=12345678

&scope=email

Token Endpoint

POST `/token` HTTP/1.1

Host: `oauth.example.com`

Client Authentication

Authorization: Basic `YWJjOjEyMw==`

Content-Type: `application/x-www-form-urlencoded`

`grant_type=password`

`&username=chitsaou`

`&password=12345678`

`&scope=email`

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Client Authentication

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=password

表示「我拿到的授權狀是
User 的 Password」

&username=chitsaou

&password=12345678

&scope=email

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Client Authentication

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=password

表示「我拿到的授權狀是
User 的 Password」

&username=chitsaou

&password=12345678

Credentials

&scope=email

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Client Authentication

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=password

表示「我拿到的授權狀是
User 的 Password」

&username=chitsaou

&password=12345678

Credentials

&scope=email

想要請求的權限範圍 (Scopes)

Password 的問題

- Token Endpoint 要防暴力破解
- Client 不准把帳密存起來

What if 不需要 User?

- 只存取公開資料
- Data-Mining Twitter Public Timeline

“Client Credentials Grant Flow”

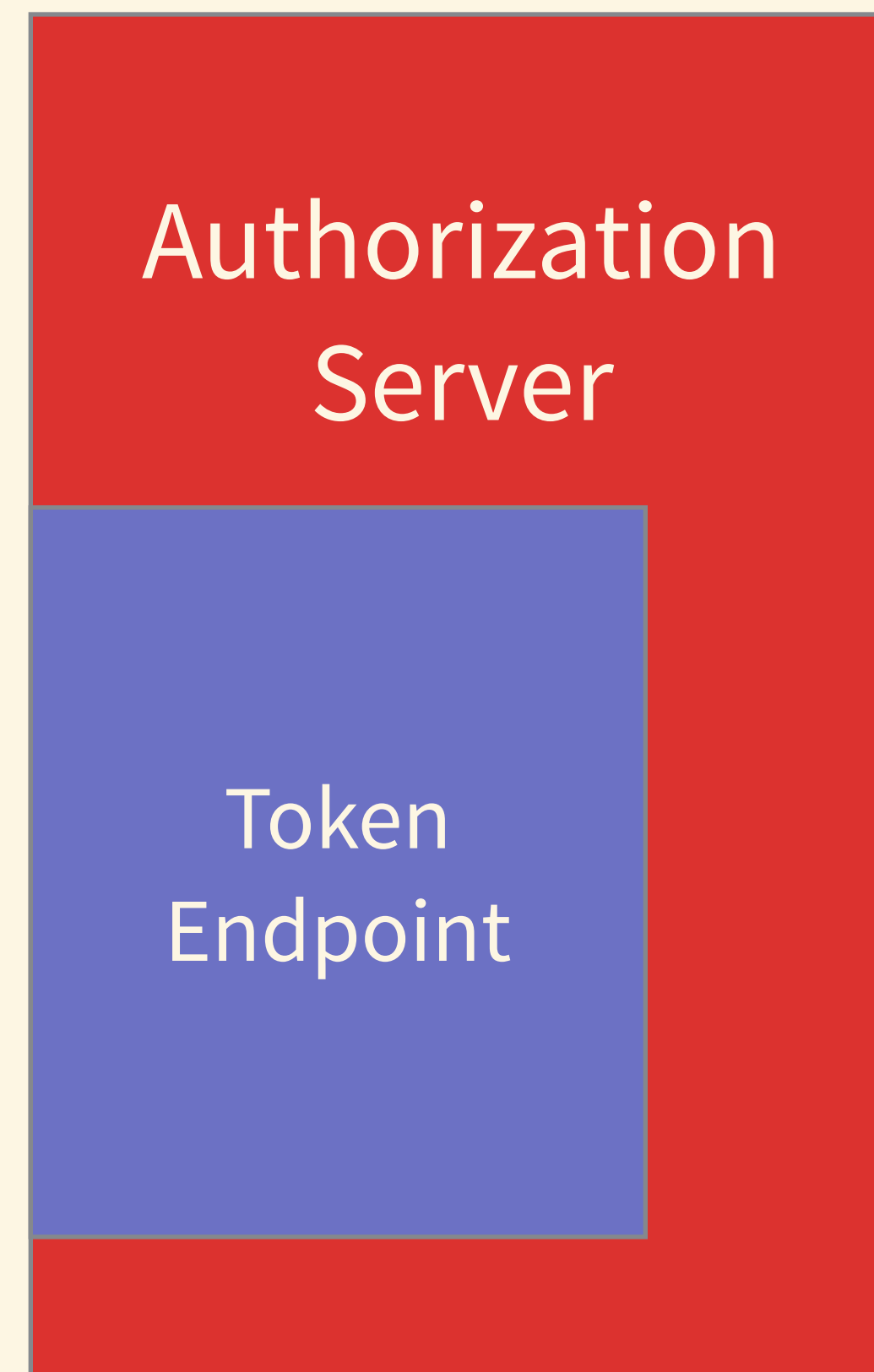
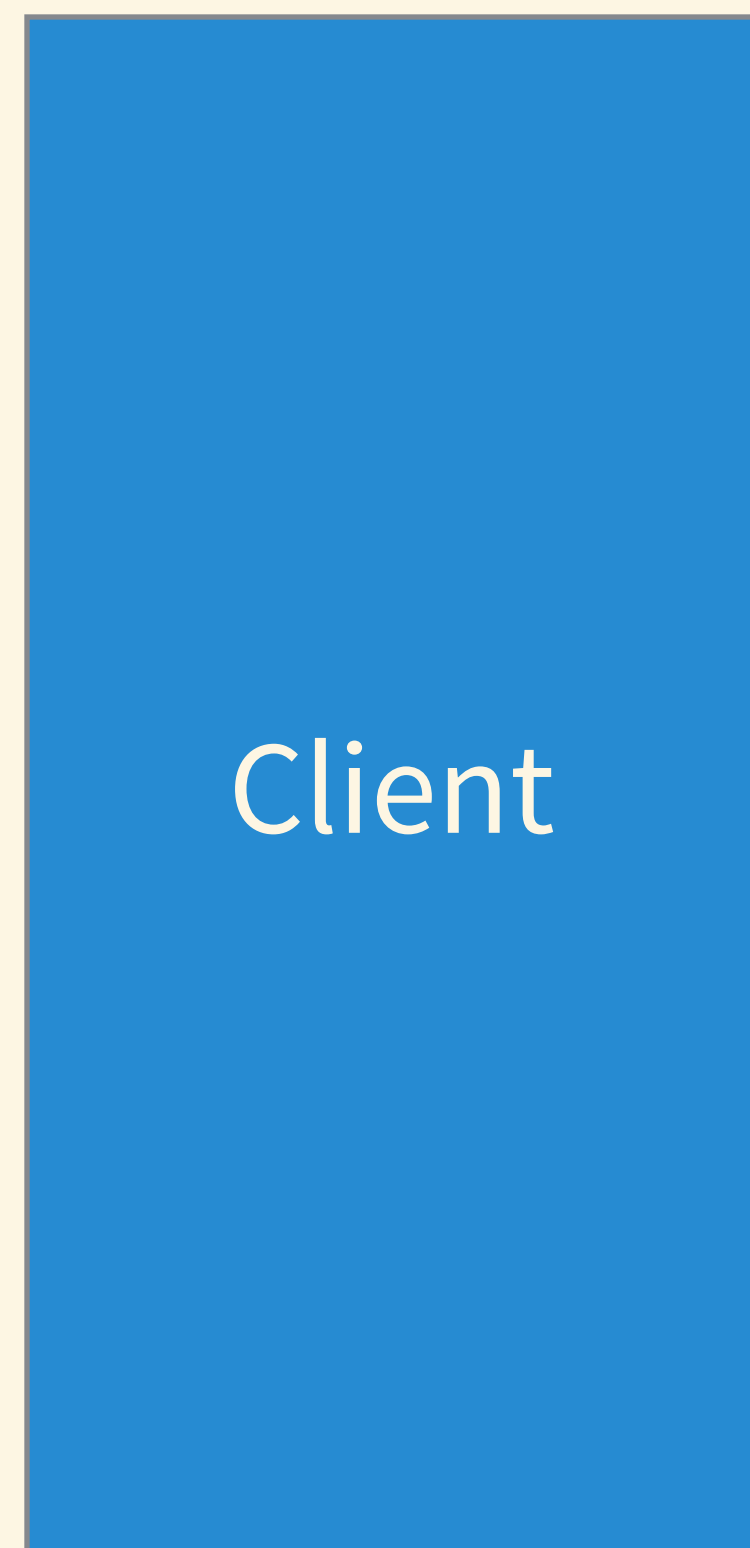


A diagram illustrating the interaction between a Client and an Authorization Server. The Client is represented by a blue rectangle on the left, and the Authorization Server is represented by a red rectangle on the right. The text 'Client' is centered within the blue rectangle, and 'Authorization Server' is centered within the red rectangle. Below the rectangles, the text '※抄自 spec' is displayed.

Client

Authorization
Server

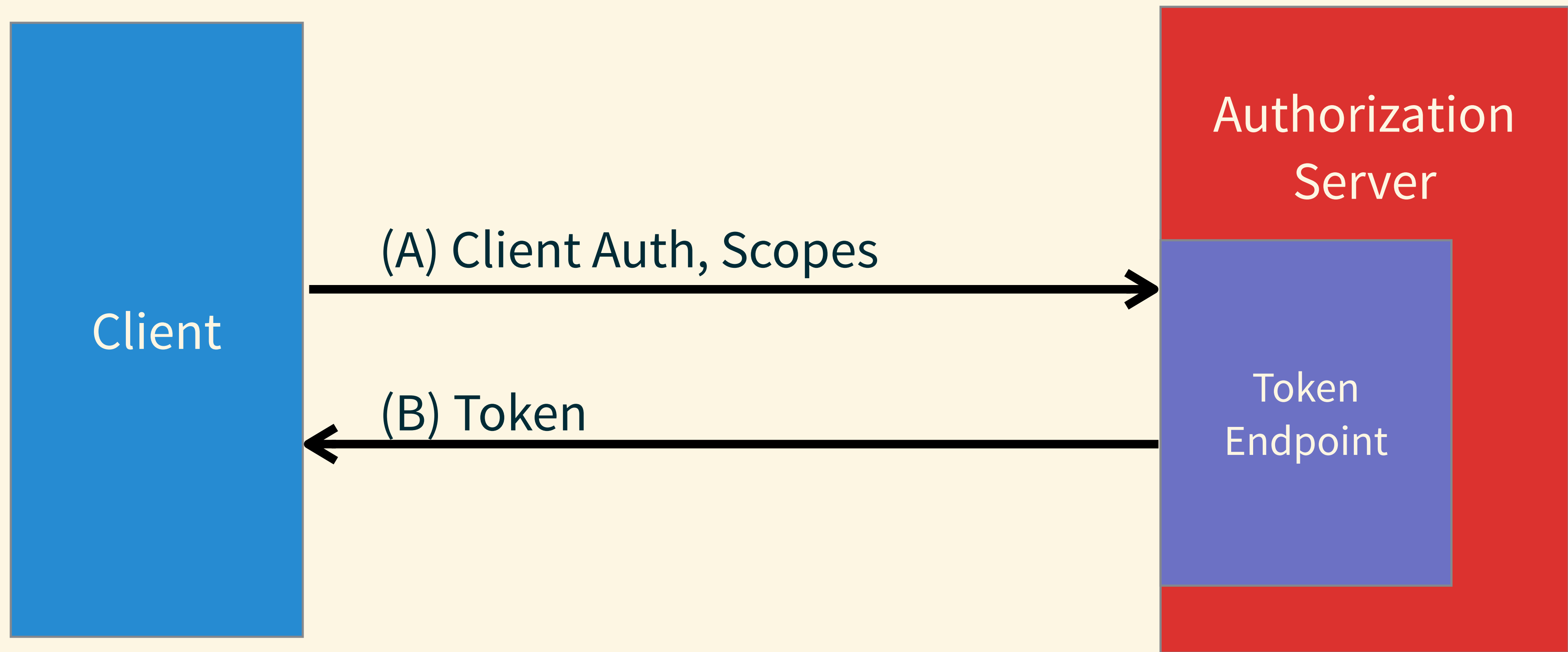
※抄自 spec



※抄自 spec



※抄自 spec



※抄自 spec

```
POST /token HTTP/1.1
```

```
Host: oauth.example.com
```

```
Authorization: Basic YWJjOjEyMw==
```

```
Content-Type: application/x-www-form-urlencoded
```

```
grant_type=client_credentials
```

```
&scope=search_timeline
```

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials

&scope=search_timeline

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Client Authentication

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials

&scope=search_timeline

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Client Authentication

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

表示「我 Client 要申請 Token 自用」

grant_type=client_credentials

&scope=search_timeline

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Client Authentication

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

表示「我 Client 要申請 Token 自用」

grant_type=client_credentials

&scope=search_timeline

想要請求的權限範圍 (Scopes)

4 built-in grant flows

- Authorization Code
- Implicit
- Resource Owner Password Credentials
- Client Credentials

發生錯誤時的回應方式

Errors on **Authorization** Endpoint

Errors on Authorization Endpoint

1. 認不出 Client、Redirect URI 不符
2. 參數傳錯 / Authorization Server 沒實作某些功能
3. Resource Owner 拒絕授權
4. Internal Server Error

Client 認不得、Redirect URI 不符

- Redirect URI 沒給 / 不正確 / 沒事先註冊
- 要視為攻擊，不可以 Redirect 到該 Redirect URI
- 要提示警告訊息給 Resource Owner

其他錯誤

- 要轉回 Redirect URI 向 Client 告知錯誤
- 在 URI 後面附上 error parameters
- Auth Code Grant Flow - 用 Query `?error=...`
- Implicit Grant Flow - 用 Fragment `#error=...`

Error Parameters

- `error` - Error Code , 必填, 下詳
- `error_description` - 簡單說明錯誤
- `error_uri` - 一個網址指到詳細說明網頁
- `state` - 傳回 Client 的 state 原值, 之前有給就必填
- 某些還會給 `error_subcode` (尤其中國的網站)

`http://localhost:3000/users/auth/facebook/
callback?`

`error=access_denied`

`&error_description=User Denies Authorization`

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

Redirection Endpoint

`http://localhost:3000/users/auth/facebook/
callback?`

`error=access_denied`

`&error_description=User Denies Authorization`

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

Redirection Endpoint

`http://localhost:3000/users/auth/facebook/callback?`

`error=access_denied` 錯誤碼

`&error_description=User Denies Authorization`

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6`

Redirection Endpoint

`http://localhost:3000/users/auth/facebook/callback?`

`error=access_denied` 錯誤碼

簡述錯誤是什麼

`&error_description=User Denies Authorization`

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6`

Redirection Endpoint

`http://localhost:3000/users/auth/facebook/callback?`

`error=access_denied` 錯誤碼

簡述錯誤是什麼

`&error_description=User Denies Authorization`

看詳細說明的 URL

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6`

Redirection Endpoint

`http://localhost:3000/users/auth/facebook/callback?`

`error=access_denied` 錯誤碼

簡述錯誤是什麼

`&error_description=User Denies Authorization`

看詳細說明的 URL

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d6333d7c807e6` 原封不動的 State 值

`http://localhost:3000/users/auth/facebook/
callback#` ← Implicit Flow: 換成 Fragment Parameter

`error=access_denied`

`&error_description=User Denies Authorization`

`&error_uri=https://doc.example.com/...`

`&state=446c63696b24b4b6687cdff62eabce3a9c9d63
33d7c807e6`

Client 造成的錯誤

error =	Meaning
invalid_request	沒給必要參數、給了不正確的參數、重複給參數、或其他原因導致無法解讀。
unauthorized_client	Client 不准用這種 Response Type 來取得 Authorization Code。
unsupported_response_type	Authorization Server 不支援使用這種 Response Type。
invalid_scope	所要求的 scope 不正確、不明、無法解讀。

授權失敗

error =

`access_denied`

Meaning

Resource Owner 或 Authorization Owner
拒絕授權的申請。

Authorization Server Internal Server Error

error =

Meaning

server_error

Authorization Server 遇到意外的情況而無法處理請求。

temporarily_unavailable

Authorization Server 超載或維修中，無法處理。

Authorization Server Internal Server Error

error =

Meaning

`server_error`

Authorization Server 遇到意外的情況而無法處理請求。

`temporarily_unavailable`

Authorization Server 超載或維修中，無法處理。

Q: 為什麼不能用 500 回?

A: 因為 500 不支援 Location 轉址，error 傳不回 Client

Errors on **Token** Endpoint

Errors on Token Endpoint

1. Auth. Server 認不得 Client
2. 參數傳錯 / Authorization Server 沒實作某些功能
3. 授權狀 (Grant) 不正確
4. Internal Server Error

回應方式

- 一律包進 JSON 回
- 通常回 400 Bad Request
- 若 Client 認證失敗 & 用 Authorization Header 認證
 - 塞 WWW-Authenticate header

Error Parameters (\cong Auth E.P.)

- `error`
- `error_description`
- `error_uri`
- 某些還會給 `error_subcode` (尤其中國的網站)
- No state (因為沒有 state 傳進來啊)

HTTP/1.1 401 Unauthorized

Content-Type: application/json; charset=UTF-8

WWW-Authenticate: Basic

Cache-Control: no-store

Pragma: no-cache

```
{  
  "error": "invalid_client",  
  "error_description": "Client Authentication  
Failed",  
  "error_uri": "https://doc.example.com/..."  
}
```


各有建議的 Response Code

HTTP/1.1 401 Unauthorized

Content-Type: application/json; charset=UTF-8

WWW-Authenticate: Basic

Cache-Control: no-store

Pragma: no-cache

```
{  
  "error": "invalid_client",  
  "error_description": "Client Authentication  
Failed",  
  "error_uri": "https://doc.example.com/..."  
}
```

各有建議的 Response Code

HTTP/1.1 401 Unauthorized

Content-Type: application/json; charset=UTF-8

WWW-Authenticate: Basic

Cache-Control: no-store

Pragma: no-cache

```
{
  "error": "invalid_client",
  "error_description": "Client Authentication
Failed",
  "error_uri": "https://doc.example.com/..."
}
```

Client 透過 Basic Auth
認證失敗則塞這個 header

各有建議的 Response Code

HTTP/1.1 401 Unauthorized

Content-Type: application/json; charset=UTF-8

WWW-Authenticate: Basic

Cache-Control: no-store

Pragma: no-cache

Client 透過 Basic Auth
認證失敗則塞這個 header

```
{  
    "error": "invalid_client",  
    "error_description": "Client Authentication  
Failed",  
    "error_uri": "https://doc.example.com/..."  
}
```

錯誤碼

各有建議的 Response Code

HTTP/1.1 401 Unauthorized

Content-Type: application/json; charset=UTF-8

WWW-Authenticate: Basic

Cache-Control: no-store

Pragma: no-cache

```
{  
  "error": "invalid_client",  
  "error_description": "Client Authentication  
Failed",  
  "error_uri": "https://doc.example.com/..."  
}
```

Client 透過 Basic Auth
認證失敗則塞這個 header

錯誤碼

簡述錯誤是什麼

Authorization Server 認不出 Client

error =

Meaning

invalid_client

Client 認證失敗

要回 401 Unauthorized

如果是用 Basic Auth 送出 Client Auth , 則要附上

WWW-Authenticate: Basic

授權狀不正確

error =

Meaning

`invalid_grant`

提出的 Grant 或是 Refresh Token 不正確、
過期、被撤銷，或 Redirection URI 不符，
或根本就不是給你這個 Client。

Client 造成的錯誤

error =	Meaning
invalid_request	沒給必要參數、給了不正確的參數、重複給參數、或其他原因導致無法解讀。
unauthorized_client	Client 不准用這種 Response Type 來取得 Authorization Code。
unsupported_grant_type	Authorization Server 不支援使用這種 Grant Type。
invalid_scope	所要求的 scope 不正確、不明、無法解讀。

Authorization Server Internal Server Error

error =

—

Meaning

—

Authorization Server Internal Server Error

error =

Meaning

—

—

Q: 為什麼沒有?

A: 因為是 Client 直接發 Request 到 Server, 直接噴 500 就行

拿到 Token 了，如何打 API

RFC 6750 “Bearer Token Usage”

HTTP/1.1 200 OK

Content-Type: application/json; charset=UTF-8

Cache-Control: no-store

Pragma: no-cache

{

 "access_token": "2YotnFZFEjr1zCsicMWpAA",

 "token_type": "Bearer",

 "expires_in": 3600,

 "refresh_token": "tGzuv3J0kF0XG5Qx2TlKWIA",

 "scopes": "email"

}

Bearer Token

- RFC 6750 定義，一種 OAuth 2.0 的 Token
- 用法最簡單：出示 Token 給 Resource Server
- Token 內容亂碼就行
- Spec 只規定出示 Token 的方式 & 錯誤訊息

出示 Token 的方法

- (in Header) `Authorization: Bearer 2YotnFZF...` **最推薦**
- (in Body) `&access_token=2YotnFZF...`
- (in URL) `?access_token=2YotnFZF...` **不推薦**

Authorization: Basic XXXXXXXXX 最推薦

- 必須支援
- 直接把 Token 字串放進 Header 就行了

```
GET /me.json HTTP/1.1  
Host: api.example.com  
Authorization: Bearer 2YotnFZF...
```

in Request Body

- 適用於 "**Form**" 之類的 request (POST, PATCH etc.)
- 不可以是 multi-part
- Content-Type: application/x-www-form-**urlencoded**

```
POST /api/articles HTTP/1.1
```

```
Host: api.example.com
```

```
Content-Type: application/x-www-form-urlencoded
```

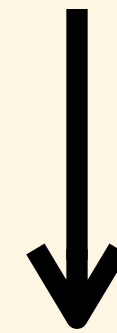
```
  title=Test%20Article  
&content=%28ry%0A%0D  
&access_token=2YotnFZF...
```


in URL Query Param

不推薦

- 不推薦，因為可能會被記在 log 裡
- 要確保 response 是 non-cacheable
- 有 Form 能用就盡量用 Form Body

```
GET /api/articles?access_token=2YotnFZF...  
HTTP/1.1  
Host: api.example.com
```



```
HTTP/1.1 200 OK  
Content-Type: application/json; charset=UTF-8  
Cache-Control: private (or no-store if not 200)
```

發生錯誤時的回應方式

Bearer Token Error

1. 參數傳錯
2. Token 不正確
3. Scope 權限不足

回應方式

- 一律用 **WWW-Authenticate** header 回錯誤
- 有出示 Token，才能附上其他 Error Parameters
- ※ 沒規定不能再回 JSON 回

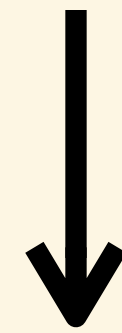
error =	Status	Meaning
invalid_request	400	沒給必要參數、給了不正確的參數 重複給參數、重複出示 Access Token 或其他原因導致無法解讀。
invalid_token	401	Access Token 過期、被撤回、無法解讀。 Client 可以重新申請一個 Token 再試。
insufficient_scope	403	Token 的 scope 權限不足。 Error 可以附上 scope= 來提示所需權限。

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="The API"
  error="invalid_token"
  error_description="Token is not usable."
```

```
HTTP/1.1 403 Forbidden
WWW-Authenticate: Bearer realm="The API",
  error="insufficient_scope",
  error_description="Scope not sufficient.",
  scope="friend_list photo"
```

完全沒出示 Access Token

```
GET /api/articles HTTP/1.1  
Host: api.example.com
```



```
HTTP/1.1 401 Unauthorized  
WWW-Authenticate: Bearer realm="The API"
```


Token 過期 → 換發

Token Refresh

Token Refresh

- 拿 Access Token 核發時的 Refresh Token 去換發
- 原本沒有發 Refresh Token，就不能換發
- POST 到 Token Endpoint

```
POST /token HTTP/1.1
```

```
Host: oauth.example.com
```

```
Authorization: Basic YWJjOjEyMw==
```

```
Content-Type: application/x-www-form-  
urlencoded
```

```
grant_type=refresh_token
```

```
&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA
```

```
&scope=search_timeline
```

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw==

Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token

&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA

&scope=search_timeline

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token

&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA

&scope=search_timeline

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

表示「我 Client 要換發新的 Token」

grant_type=refresh_token

&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA

&scope=search_timeline

Token Endpoint

POST /token HTTP/1.1

Host: oauth.example.com

Authorization: Basic YWJjOjEyMw== Client Authentication

Content-Type: application/x-www-form-urlencoded

表示「我 Client 要換發新的 Token」

grant_type=refresh_token

&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA

&scope=search_timeline

scope 可省略，不可大於
Access Token 申請過的權限

製造 OAuth 2 Provider 的方法

= 造 Authorization Server

Service	spec 相容	支援的 Grant Types	scope 分隔	Refresh Token	Client 認證
Facebook	✗	Auth Code, Implicit, Client Cred.	comma	△ (自製)	GET
GitHub	✗	Auth Code, Password (自製)	comma	✗	POST
Twitter	○	Client Cred.	(無 scope)	✗	Basic
Google	✗	Auth Code, Implicit	space	○	POST
Microsoft	✗	Auth Code, Implicit	? ? ?	○	POST
Dropbox	○	Auth Code, Implicit	(無 scope)	✗	Basic, POST
Amazon	○	Auth Code, Implicit	space	○	Basic, POST
Bitly	✗	Auth Code (半自製), Password	(無 scope)	✗	POST (Auth Code), Basic (Password)
新浪微博	✗	Auth Code	comma	✗	Basic, GET
豆瓣	✗	Auth Code, Implicit	comma	○	POST
BOX	✗	Auth Code	(無 scope)	○	POST
Basecamp	✗	Auth Code	(無 scope)	○	POST

別人怎麼做

- 不一定要支援所有 4 種 Grant Flow
- 不一定要有 Scope
- 不一定要有 Refresh Token

別人怎麼做（不好的方面）

- ✗ 切 Scope 很多人用 ` , `
- ✗ Client 認證不見得都支援 HTTP Basic Auth
- ✗ Bearer Token 不一定是用
“Bearer” Authorization Header

有 Spec 可以 Follow 就 Follow

- ✓ 切 Scope 用空格
- ✓ Client 認證至少要支援 Basic Auth
- ✓ Bearer Token 就用
“Bearer” Authorization Header

Step 1: 定義 Resource Owner

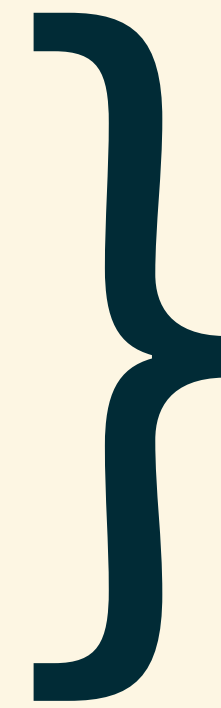
Resource Owner

- 就定義成網站上的 User 吧

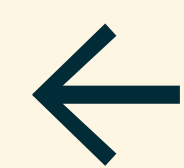
Step 2: 設計 Client 管理界面

Client Registration (CRUD)

- id
- secret (視為密碼)
- Redirect URI



反正不是給人記的
所以生亂碼就行



要求 Developer 事先指定

Step 3: 開 Endpoints

Authorization Endpoint

- 通常開在**主站**，方便直接讓 User 用 Cookie 登入
- 製作一個 **Dialog** 讓 User 決定要不要允許授權
- User 回答之後，**Redirect** 回 Client
- 會 Redirect 回 Client，所以要先確保 Client 正確

Authorization Endpoint

```
halt if !(client_id and redirect_uri matches)
```

```
error(invalid_scope) if !(valid_scope?)
```

```
error(unsupported_response_type)  
  if !(valid_response_type?)
```

```
# other errors (incomplete pseudo code)
```

Authorization Endpoint

```
result = ask_user_for_authorization!

if result == Authorized then
  if response_type == Code
    callback_params = generate_grant_code()
  else if response_type == Token
    callback_params = generate_token()

else if result == Denied then
  callback_params = generate_error_code(access_denied)
```

Authorization Endpoint

```
if response_type == Code
    redirect_to_client_with_query(callback_params)

else if response_type == Token
    redirect_to_client_with_fragment(callback_params)
```

Token Endpoint

- 通常開在 **API** subdomain
- 照 Spec 做 **JSON** Response 就行

Token Endpoint

`authorize_client!` or `error(invalid_client)`

`verify_redirect_uri!` or `error(invalid_request)`

`error(invalid_scope)` if `scope_given` and `!valid_scope?`

`# other errors`

`token = issue_token(expires_in: 30.days)`

`token_response_by_json(token)`

Step 4: 擋 API

“Resource Server Guard”

- 擋在 API 最外面的「保全」(Guard)
- 驗證每個 Request 都必須有正確的 Token

“Resource Server Guard”

```
fetch_token(from: header, form, query)
```

```
error(401) if !(token_given?)
```

```
error(400) if !(token_decodable?)
```

```
error(invalid_token) if expired? or revoked?
```

```
error(insufficient_scope) if scope_sufficient?
```

```
yield to API
```

第一次用 Rails + Grape API

整合 OAuth 2 就上手

第一次（略）就上手

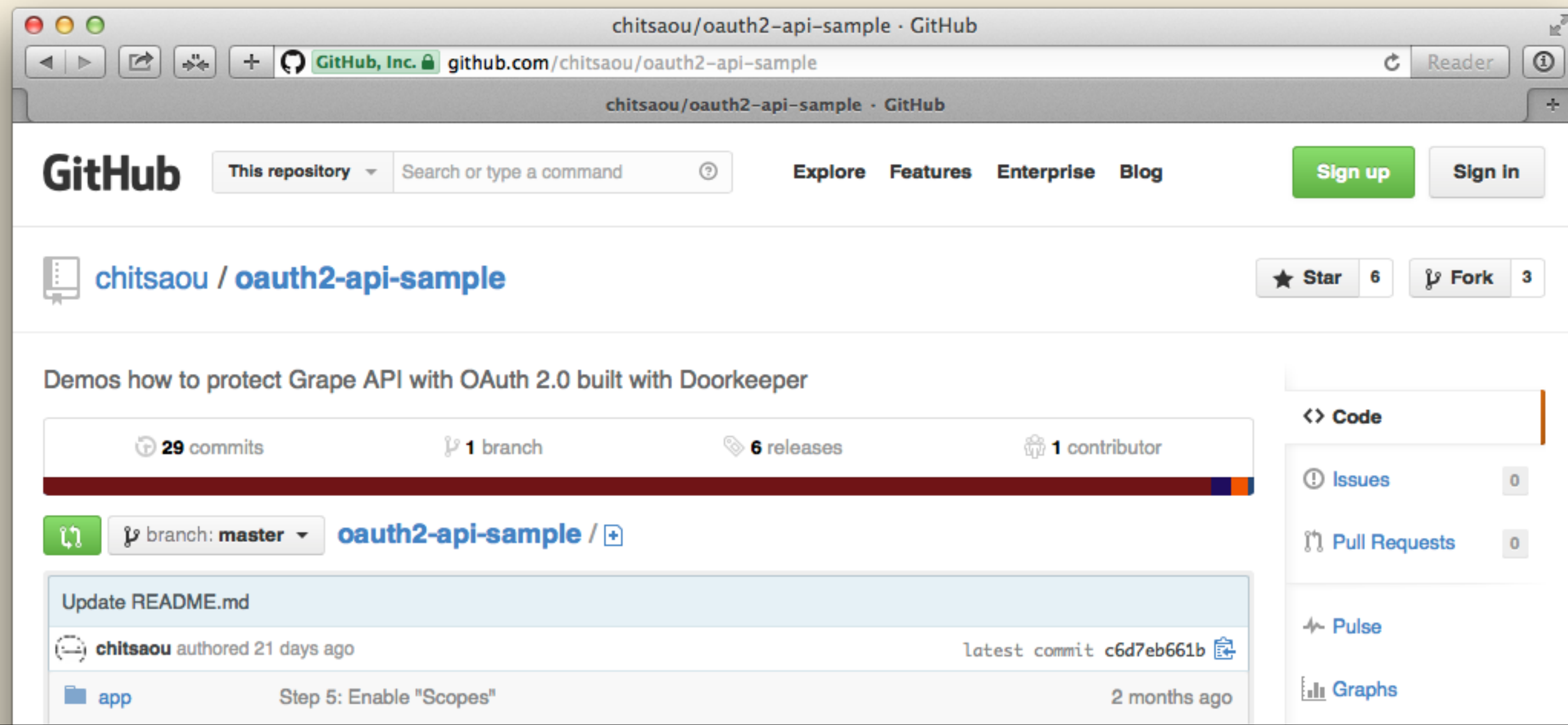
- 用 Devise 生 User (Resource Owner)
- 用 Doorkeeper 蓋 Authorization Server
- 用 Grape 蓋 API (Resource Server)
- 自己刻 Resource Server Guard 來鎖 API

Why 自己刻 Guard

- doorkeeper_for 是 Rails-Only
- Rack::OAuth2 完成度很高，但必須自己整合 ✓ Best Choice
- Warden::OAuth2 跟 Devise 卡在一起我不會解
- Grape::Middleware::Auth::OAuth2 根本沒做完

Source Available on GitHub

[GitHub.com/chitsaou/oauth2-sample-api](https://github.com/chitsaou/oauth2-sample-api)



Step 1: 定義 Resource Owner

定義 Resource Owner

```
$ rails g devise:install # 完
```

Step 1.5: 造 API

造 API

```
class SecretAPI < Grape::API
  namespace "secret"
  format :json
  get "hello" do
    {
      greeting: "Hi, #{current_user.email}"
    }
  end
end
```

Step 2: 設計 Client 管理界面

Step 3: 開 Endpoints

Step 2 + 3:
開 Authorization Server

開 Authorization Server

```
$ gem install doorkeeper  
$ rails g doorkeeper:install  
$ rails g doorkeeper:migration  
$ rake db:migrate
```

設定認證 Resource Owner 的方式

config/initializers/doorkeeper.rb

```
# 認證 Resource Owner 的方法, 直接接 Devise
resource_owner_authenticator do
  current_user ||
    warden.authenticate!(:scope => :user)
end
```

※ 抄官方文件

New Tables

- **oauth_application** - Clients Registration
- **oauth_access_grant** - Stores Auth Grant Codes
- **oauth_access_token** -
真正核發出去的 Access Tokens,
包含對應的 Refresh Token （預設關閉）

New Routes

Action(s)	Path	用途
new	/oauth/authorize	Authorization Endpoint
create	/oauth/authorize	User 許可授權時的 action
destroy	/oauth/authorize	User 拒絕授權時的 action
show	/oauth/authorize/:code	Local 測試用
update	/oauth/authorize	?
create	/oauth/token	Token Endpoint
show	/oauth/token/info	Token Debug Endpoint
resources	/oauth/applications	Clients 管理界面 (CRUD)
index	/oauth/authorized_applications	User 管理授權過的 Clients
destroy	/oauth/authorized_applications/:id	

Doorkeeper Built-In™

- ✓ Authorization Endpoint & Token Endpoint
- ✓ Token Debug Endpoint
(在 Implicit Flow 驗證 Token 的真實性)
- ✓ Client Registration Interface (CRUD)
- ✓ User 管理授權過的 Clients 的界面 (可 Revoke)


Let's Create a Client

<http://localhost:12345/auth/demo/callback>

New application

Name

Redirect uri Use um:ietf:wg:oauth:2.0:oob



Let's Get an Access Token

Application: Test App

Callback url:

`http://localhost:12345/auth/demo/callback`

Application Id:

`4a407c6a8d3c75e17a5560d0d0e4507c77b047940db6df882c86aaeac2c788d6`

Secret:

`c5ca88943d9f4e8b67ba7a13f75c553f1085d9821c6da8a254b6f96c0eb39679`

Link to authorization code:

[Authorize](#)



(A) 發出授權申請

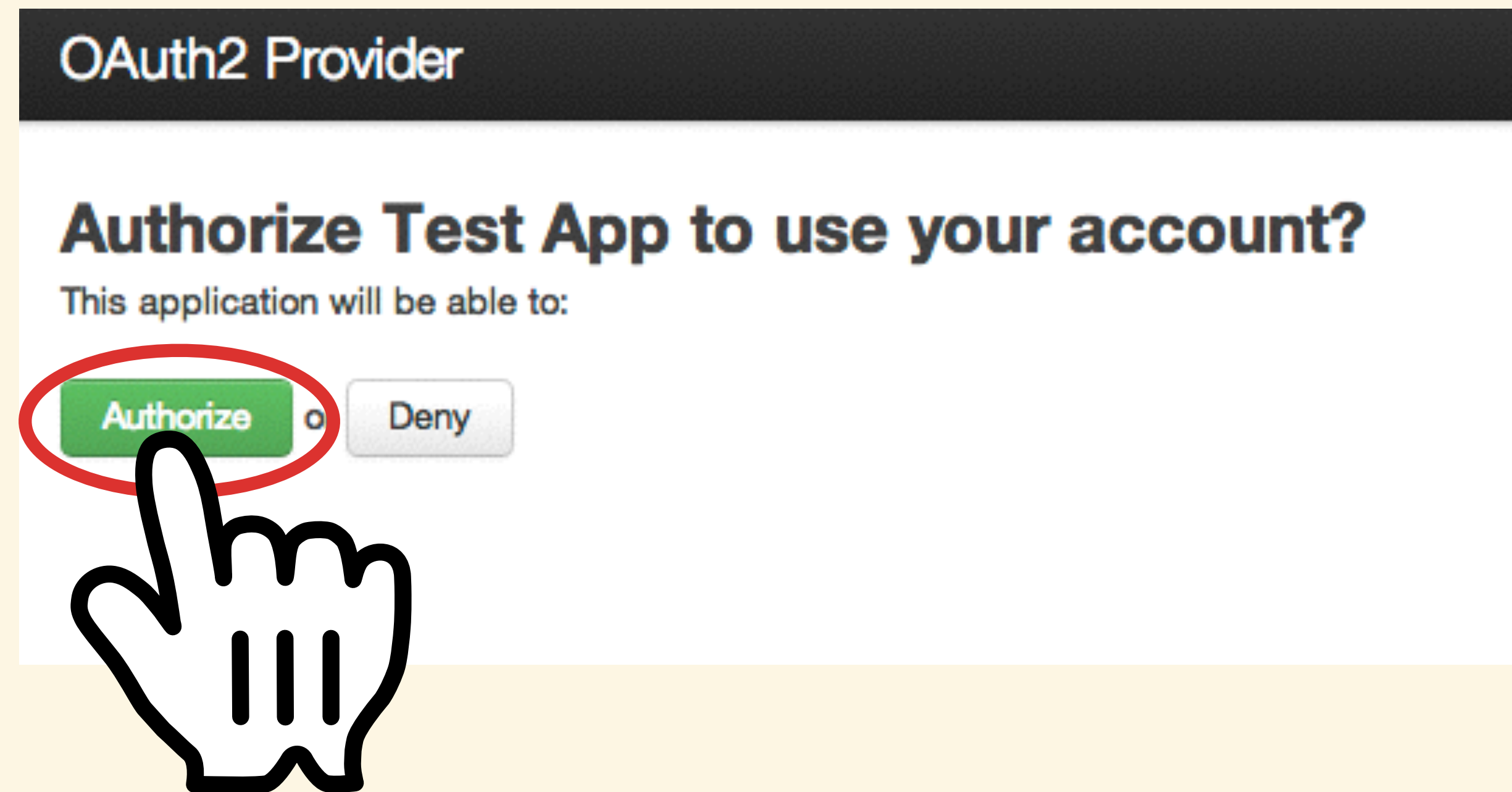
```
http://localhost:3000/oauth/authorize?
```

```
client_id=4a407c6a8d3c75e17a5560d0d0e4507c77b  
047940db6df882c86aaeac2c788d6
```

```
&redirect_uri=http://localhost:12345/auth/  
demo/callback
```

```
&response_type=code
```


(B) Auth. Server 問 User



(C) 授權狀下來了

```
http://localhost:12345/auth/demo/callback?
```

```
code=9e0ad73f94669d9743bb0c2e65c4784f723c11c7  
61852477a4d37d0cc9bb914d
```


(D) 拿 Code 換 Token

http://localhost:3000/oauth/token

POST

URL params

Headers (2)

Authorization

Basic NGE0MDdjNmE4ZDNjNzVIM

×

Add preset

Manage presets

Content-Type

application/x-www-form-urlencoded

×

Header

Value

form-data

x-www-form-urlencoded

raw

binary

redirect_uri

http://localhost:12345/auth/demo/c

×

code

9e0ad73f94669d9743bb0c2e65c47

×

client_id

4a407c6a8d3c75e17a5560d0d0e4

×

grant_type

authorization_code

×

Key

Value

Send

Preview

Add to collection

Reset

Body

Headers (10)

STATUS 200 OK

TIME 66 ms

Pretty

Raw

Preview

JSON

XML

Copy

1 {

2 "access_token": "4ead67fc8917761a7f0cd1f0cae30e905fc93e0d2543032f58395e5a55b9869e",

3 "token_type": "bearer",

4 "expires_in": 7200

5 }

Token 到手

```
4ead67fc8917761a7f0cd1f0cae30e905fc93e0d25430  
32f58395e5a55b9869e
```

Step 4: 擋 API

The Most Hard Part

api/concerns/api_guard.rb

```
module APISGuard  
  extend ActiveSupport::Concern  
  
end
```

Building Resource Server Guard

- Fetch Access Token via Rack::OAuth2
- Find Access Token from Model
- Validate if Token is not Expired && not Revoked
- Validate if Token has Sufficient Scopes
- If All Valid, Pass to Grape API

Fetch Access Token via Rack::OAuth2

- use Rack::OAuth2::Server::Resource::Bearer
- 事實上只負責把 `yield` 出來的 Token 存在某處
- 如果 Request 不帶 Token 就直接 Pass 到下個 stack
- 所以我只拿它來 Fetch Token

```
included do |base|  
  # OAuth2 Resource Server Authentication  
  use Rack::OAuth2::Server::Resource::Bearer,  
    'The API' do |request|  
    # Authenticator only fetches the raw token string  
  
    # Must yield access token to store it in the env  
    request.access_token  
  end  
end
```

(in helpers block)

```
def get_token_string  
  request.env[Rack::Auth2::Server::Resource::  
    ACCESS_TOKEN]  
end
```


Find Access Token Instance

- `Doorkeeper::AccessToken.authenticate`
 - `→ #find`
- returns `nil` if not found

(in helpers block)

```
def find_access_token(token_string)
  Doorkeeper::AccessToken.authenticate(token_string)
end
```

Validate the Token

- OAuth2::AccessTokenValidationService.validate
- 檢查有沒有 Expired、Revoked
- 可以傳 scopes 參數進去檢查 scope 有沒有齊

(in helpers block)

```
def validate_access_token(access_token, scopes)
  OAuth2::AccessTokenValidationService
    .validate(access_token, scopes: scopes)
end
```

```
def validate(token, scopes: [])
```

```
  if token.expired?  
    return EXPIRED
```

Token#expired?, Token#revoked?
是 Doorkeeper 內建的

```
  elsif token.revoked?  
    return REVOKED
```

```
  elsif !self.sufficient_scope?(token, scopes)  
    return INSUFFICIENT_SCOPE
```

```
  else  
    return VALID
```

```
  end
```

```
end
```

```
def sufficient_scope?(token, scopes)
  if scopes.blank?
    # no scope required => any token is valid
    return true
  else
    # scopes required
    #=> Check sufficiently by Set comparison
    required_scopes = Set.new(scopes)
    authorized_scopes = Set.new(token.scopes)

    return authorized_scopes >= required_scopes
  end
end
```

這是最簡單的集合比較
你可以依需求實作演算法

做 guard! 在 Grape 裡面擋住

```
class SecretAPI < Grape::API
  get "hello" do
    guard!
    {
      greeting: "Hi, #{current_user.email}"
    }
  end
end
```

(in helpers block)

```
def guard!(scopes: [])  
  token_string = get_token_string()  
  
  if token_string.blank?  
    raise MissingTokenError  
  
  elsif (  
    access_token = find_access_token(token_string)  
  ).nil?  
    raise TokenNotFoundError
```


(in helpers block)

```
def guard!(scopes: [])  
  token_string = get_token_string() 先抓出 Token String  
  
  if token_string.blank?  
    raise MissingTokenError  
  
  elsif (  
    access_token = find_access_token(token_string)  
  ).nil?  
    raise TokenNotFoundError
```

(in helpers block)

```
def guard!(scopes: [])  
  token_string = get_token_string() 先抓出 Token String  
  
  if token_string.blank?  
    raise MissingTokenError 抓到的是空字串，表示沒給 Token  
  
  elsif (  
    access_token = find_access_token(token_string)  
  ).nil?  
    raise TokenNotFoundError
```

(in helpers block)

```
def guard!(scopes: [])  
  token_string = get_token_string() 先抓出 Token String  
  
  if token_string.blank?  
    raise MissingTokenError 抓到的是空字串，表示沒給 Token  
  
  elsif (  
    access_token = find_access_token(token_string)  
  ).nil?  
    raise TokenNotFoundError 有給但找不到，是 Invalid Token
```

```
else
case validate_access_token(access_token, scopes)
when OAuth2::AccessTokenValidationService
  ::INSUFFICIENT_SCOPE
  raise InsufficientScopeError.new(scopes)
when OAuth2::AccessTokenValidationService::EXPIRED
  raise ExpiredError
when OAuth2::AccessTokenValidationService::REVOKED
  raise RevokedError
when OAuth2::AccessTokenValidationService::VALID
  @current_user = User.find(access_token
                             .resource_owner_id)
ennnd
```

```
else
case validate_access_token(access_token, scopes)
when OAuth2::AccessTokenValidationService
  ::INSUFFICIENT_SCOPE
  raise InsufficientScopeError.new(scopes)
when OAuth2::AccessTokenValidationService::EXPIRED
  raise ExpiredError
when OAuth2::AccessTokenValidationService::REVOKED
  raise RevokedError
when OAuth2::AccessTokenValidationService::VALID
  @current_user = User.find(access_token
                             .resource_owner_id)
ennnd
```

Scope 不符

```
else
  case validate_access_token(access_token, scopes)
  when OAuth2::AccessTokenValidationService
    ::INSUFFICIENT_SCOPE
    raise InsufficientScopeError.new(scopes)
  when OAuth2::AccessTokenValidationService::EXPIRED
    raise ExpiredError
  when OAuth2::AccessTokenValidationService::REVOKED
    raise RevokedError
  when OAuth2::AccessTokenValidationService::VALID
    @current_user = User.find(access_token
                              .resource_owner_id)

  ennnd
```

Scope 不符

過期了

```
else
case validate_access_token(access_token, scopes)
when OAuth2::AccessTokenValidationService
  ::INSUFFICIENT_SCOPE Scope 不符
  raise InsufficientScopeError.new(scopes)
when OAuth2::AccessTokenValidationService::EXPIRED 過期了
  raise ExpiredError
when OAuth2::AccessTokenValidationService::REVOKED 撤銷了
  raise RevokedError
when OAuth2::AccessTokenValidationService::VALID
  @current_user = User.find(access_token
                             .resource_owner_id)
ennnd
```

```
else
  case validate_access_token(access_token, scopes)
  when OAuth2::AccessTokenValidationService
    ::INSUFFICIENT_SCOPE
      raise InsufficientScopeError.new(scopes)
  when OAuth2::AccessTokenValidationService::EXPIRED
      raise ExpiredError
  when OAuth2::AccessTokenValidationService::REVOKED
      raise RevokedError
  when OAuth2::AccessTokenValidationService::VALID
    @current_user = User.find(access_token
                              .resource_owner_id)

ennnd
```

Scope 不符

過期了

撤銷了

都 OK 就設 current_user

最後是 Error Response

- Rack::OAuth2 裡面的直接拿來用
- 缺點: insufficient_scope 不會給 WWW-Authenticate
 - 因為照 RFC 2617 只有 401 需要回這個 Header

(in included block)

```
error_classes = [MissingTokenError,  
    TokenNotFoundError, ExpiredError,  
    RevokedError, InsufficientScopeError]  
  
rescue_from *error_classes,  
    oauth2_bearer_token_error_handler
```

```
def oauth2_bearer_token_error_handler
  Proc.new { |e|
    response = case e
      when MissingTokenError
        Rack::OAuth2::Server::Resource
          ::Bearer::Unauthorized.new
      when TokenNotFoundError
        Rack::OAuth2::Server::Resource
          ::Bearer::Unauthorized.new(
            :invalid_token, "Bad Access Token.")
      # etc. etc.
    end
    response.finish
  }
end
```

(in ClassMethods module)

guard_all!

```
class SecretAPI < Grape::API
  guard_all!
  get "hello" do
    {
      greeting: "Hi,#{current_user.email}"
    }
  end
end
```

(in ClassMethods module)

```
module ClassMethods
  def guard_all!(scopes: [])
    before do
      guard! scopes: scopes
    end
  end
end
```

Done★

```
$ curl -i http://localhost:3000/api/v1/sample/secret
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="The API"
Content-Type: application/json
Cache-Control: no-cache
{"error": "unauthorized"}
```

```
$ curl -i http://localhost:3000/api/v1/sample/secret \  
> -H "Authorization: Bearer XXXXXXXX" (亂填 Token)  
HTTP/1.1 401 Unauthorized  
WWW-Authenticate: Bearer realm="Protected by OAuth  
2.0", error="invalid_token", error_description="Token  
is expired. You can either do re-authorization or  
token refresh."  
Content-Type: application/json  
Cache-Control: no-cache  
{"error": "unauthorized"}
```



```
$ curl -i http://localhost:3000/api/v1/sample/secret \  
> -H "Authorization: Bearer  
4ead67fc8917761a7f0cd1f0cae30e905fc93e0d2543032f58395e  
5a55b9869e"  
HTTP/1.1 200 OK  
Content-Type: application/json  
  
{"greeting": "Hi, ducksteven@gmail.com"}
```

Final Notes

- Doorkeeper 預設沒有鎖開 Client 的權限，你記得要鎖
- Doorkeeper 不能指定只開哪些 Flows
我開了 PR 還沒 merge...
- 沒有修 insufficient_scope Error
缺 WWW-Authenticate 的問題

What about Omniauth?

- github.com/intridea/omniauth-oauth2
- 很簡單， 參數寫一寫就完成了

Conclusion

- OAuth 2 Spec 讀完才會知道他在幹嘛
- 讀完 Spec → 所有 API 所有 Library 你都看得懂
- 自己整合 Grape 和 OAuth 2 其實沒有很難
只要懂 Spec 就不難...

Final Words

- 依然建議你去讀 spec，因為我省略很多細節
- 尤其是 **Security** 的部份
- **Amazon 的 OAuth 2 Login** 文件我最推薦

References

- RFC 6749 (Spec) tools.ietf.org/html/rfc6749
- RFC 6750 (Spec) tools.ietf.org/html/rfc6750
- My Notes: blog.yorkxin.org/tags/OAuth
- Many OAuth2-based API Documents

Thank You!

Q&A Time