OAuth 2.0 文档

初见OAuth2.0

OAuth 2.0是目前最流行的授权机制，用来授权第三方应用，获取用户相关的数据。

一、情景再现：

现在的小区都有门禁系统，对于小区的住户来说，他们拥有永久的密码去进出小区。双十一过后，小区送快递的快递员多了起来。每天都有快递员来送货。必须找到一个办法，让快递员通过门禁系统，进入小区。

如果住户把自己的密码，告诉快递员，他就拥有了与我同样的权限，这样好像不太合适。万一我想取消他进入小区的权力，也很麻烦，我自己的密码也得跟着改了，还得通知其他的快递员。

有没有一种办法，让快递员能够自由进入小区，又不必知道小区居民的密码，而且他的唯一权限就是送货，其他需要密码的场合，他都没有权限呢?

二、授权机制的设计

于是，设计了一套授权机制。

第一步，门禁系统的密码输入器下面，增加一个按钮，叫做"获取授权"。快递员需要首先按这个按钮，去申请授权。

第二步，他按下按钮以后，屋主（也就是我）的手机就会跳出对话框：有人正在要求授权。系统还会显示该快递员的姓名、工号和所属的快递公司。

我确认请求属实，就点击按钮，告诉门禁系统，我同意给予他进入小区的授权。

第三步，门禁系统得到我的确认以后，向快递员显示一个进入小区的令牌（access token）。令牌就是类似密码的一串数字，该数据令牌会有一个有效期，从当前的时间向后，过了有效期之后，再进入小区就需要重新授权。

第四步，快递员向门禁系统输入令牌，进入小区。

有人可能会问，为什么不是远程为快递员开门，而要为他单独生成一个令牌？这是因为快递员可能每天都会来送货，第二天他还可以复用这个令牌。另外，有的小区有多重门禁，快递员可以使用同一个令牌通过它们。

三、互联网场景

我们把上面的例子搬到互联网，就是 OAuth 的设计了。

首先，居民小区就是储存用户数据的网络服务。比如微信登录，微信储存了我的好友信息，获取这些信息，就必须经过微信的"门禁系统"。

其次，快递员（或者说快递公司）就是第三方应用，想要穿过门禁系统，进入小区。

最后，我就是用户本人，同意授权第三方应用进入小区，获取我的数据。

简单说，OAuth 就是一种授权机制。数据的所有者告诉系统，同意授权第三方应用进入系统，获取这些数据。系统从而产生一个短期的进入令牌（token），用来代替密码，供第三方应用使用。

四、令牌与密码

令牌（access_token）与密码（password）的作用是一样的，都可以进入系统，但是有三点差异。

（1）令牌是短期的，到期会自动失效，用户自己无法修改。密码一般长期有效，用户不修改，就不会发生变化。

（2）令牌可以被数据所有者撤销，会立即失效。以上例而言，屋主可以随时取消快递员的令牌。密码一般不允许被他人撤销。

（3）令牌有权限范围（scope），比如只能进小区的二号门。对于网络服务来说，只读令牌就比读写令牌更安全。密码一般是完整权限。

上面这些设计，保证了令牌既可以让第三方应用获得权限，同时又随时可控，不会危及系统安全。这就是 OAuth 2.0 的优点。

注意，只要知道了令牌，就能进入系统。系统一般不会再次确认身份，所以**令牌必须保密，泄漏令牌与泄漏密码的后果是一样的。** 这也是为什么令牌的有效期，一般都设置得很短的原因。

五、互联网应用案例

情景再现：

简书，此网站为了方便用户登陆，他们支持很多个第三方登录，就拿微博登陆来说，用户在使用简书的网站的时候，为了方便登陆。他们会选择第三方登录，如果用户手机中有微博应用的话，弹出来一个授权按钮之后，用户点击授权，用户授权之后会生成一个授权码之类的字符串，简书服务器拿着这个串去访问用户的微博信息。例如微博昵称，微博的设置的地址，还可以访问微博绑定的手机号码，对于一些比较新的应用网站来时，通过支持第三方登录可以获取不少的用户量。

名词定义：

在详细讲解OAuth 2.0之前，需要了解几个专用名词。

（1） Third-party application：第三方应用程序，本文中又称"客户端"（client），即上一节例子中的"云冲印"。
（2）HTTP service：HTTP服务提供商，本文中简称"服务提供商"，即上一节例子中的Google。
（3）Resource Owner：资源所有者，本文中又称"用户"（user）。
（4）User Agent：用户代理，本文中就是指浏览器。
（5）Authorization server：认证服务器，即服务提供商专门用来处理认证的服务器。
（6）Resource server：资源服务器，即服务提供商存放用户生成的资源的服务器。它与认证服务器，可以是同一台服务器，也可以是不同的服务器。

实现思路：

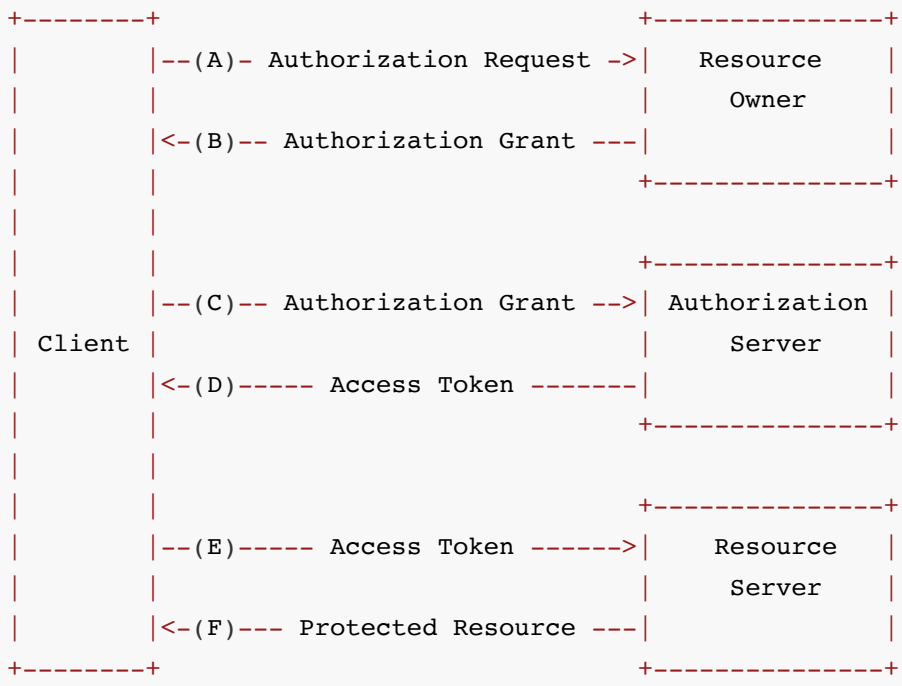
OAuth在 **客户端** 与 **服务提供商** 之间，设置了一个授权层（authorization layer）。"客户端"不能直接登录"服务提供商"，通过登录授权层，将用户与客户端区分开来。"客户端"登录授权层所用的令牌（token），与用户的密码不同。用户可以在登录的时候，可以指定授权层令牌的权限范围和有效期。

"客户端"登录授权层以后，"服务提供商"根据令牌的权限范围和有效期，向"客户端"开放用户储存的资料。

运行流程

OAuth 2.0的运行流程，摘自RFC 6749。

官网解释地址：https://tools.ietf.org/html/rfc6749#section-4.3.2 [page 10]



（A）用户打开客户端以后，客户端要求用户给予授权。
（B）用户同意给予客户端授权。
（C）客户端使用上一步获得的授权，向认证服务器申请令牌。
（D）认证服务器对客户端进行认证以后，确认无误，同意发放令牌。
（E）客户端使用令牌，向资源服务器申请获取资源。
（F）资源服务器确认令牌无误，同意向客户端开放资源。

六、OAuth授权的四种方式

官方文档关于OAuth的定义：https://tools.ietf.org/html/rfc6749#section-4.3.2

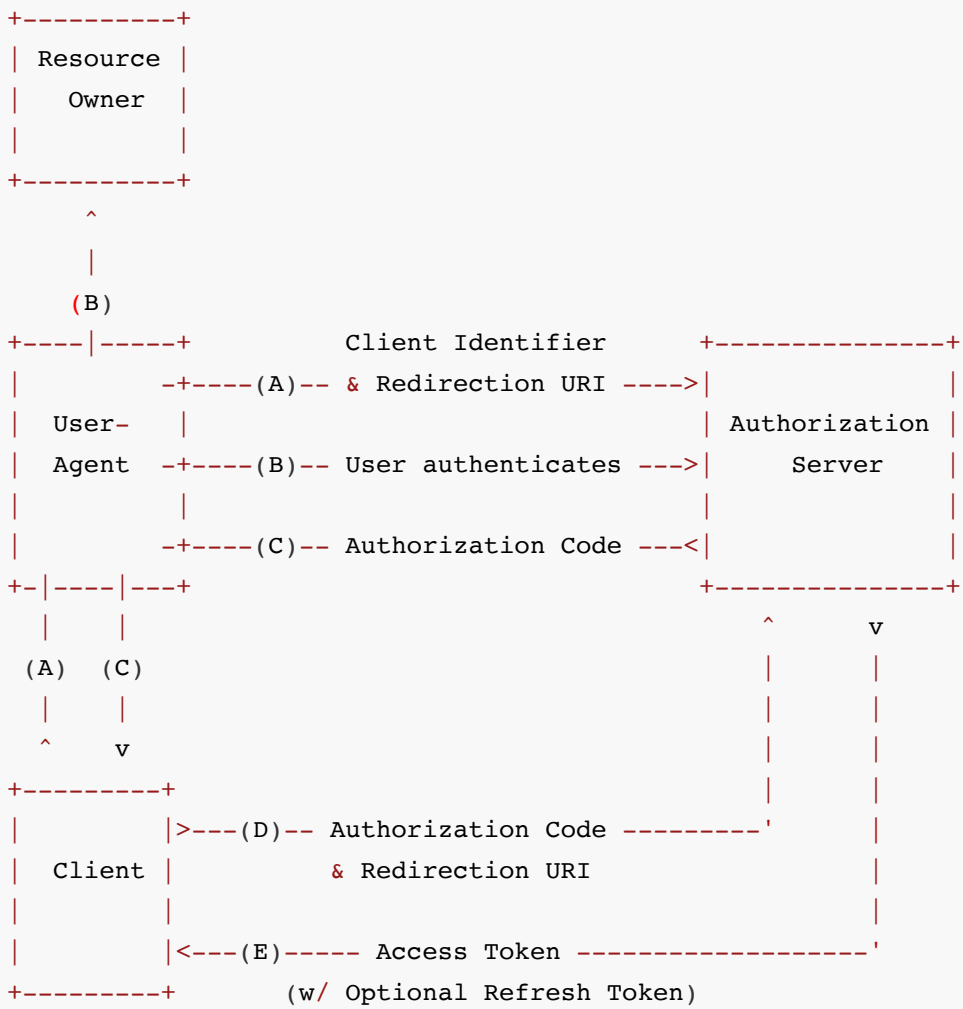
客户端必须得到用户的授权（authorization grant），才能获得令牌（access token）。OAuth 2.0定义了四种授权方式。

- 授权码模式（authorization code）
- 简化模式（implicit）
- 密码模式（resource owner password credentials）

- 客户端模式（client credentials）

授权码模式：

授权码模式（authorization code）是功能最完整、流程最严密的授权模式。它的特点就是通过客户端的后台服务器，与"服务提供商"的认证服务器进行互动。



步骤如下：

（A）用户访问客户端，后者将前者导向认证服务器。
（B）用户选择是否给予客户端授权。
（C）假设用户给予授权，认证服务器将用户导向客户端事先指定的"重定向URI"（redirection URI），同时附上一个授权码。
（D）客户端收到授权码，附上早先的"重定向URI"，向认证服务器申请令牌。这一步是在客户端的后台的服务器上完成的，对用户不可见。
（E）认证服务器核对了授权码和重定向URI，确认无误后，向客户端发送访问令牌（access token）和更新令牌（refresh token）。

A步骤中，客户端申请认证的URI，包含以下参数：

- response_type：表示授权类型，必选项，此处的值固定为"code"
- client_id：表示客户端的ID，必选项

- redirect_uri：表示重定向URI，可选项
- scope：表示申请的权限范围，可选项
- state：表示客户端的当前状态，可以指定任意值，认证服务器会原封不动地返回这个值。

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

B步骤中，用户主动授权。

C步骤中，服务器回应客户端的URI，包含以下参数：

- code：表示授权码，必选项。该码的有效期应该很短，通常设为10分钟，客户端只能使用该码一次，否则会被授权服务器拒绝。该码与客户端ID和重定向URI，是一一对应关系。
- state：如果客户端的请求中包含这个参数，认证服务器的回应也必须一模一样包含这个参数。

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

D步骤中，客户端向认证服务器申请令牌的HTTP请求，包含以下参数：

- grant_type：表示使用的授权模式，必选项，此处的值固定为"authorization_code"。
- code：表示上一步获得的授权码，必选项。
- redirect_uri：表示重定向URI，必选项，且必须与A步骤中的该参数值保持一致。
- client_id：表示客户端ID，必选项。

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

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

E步骤中，认证服务器发送的HTTP回复，包含以下参数：

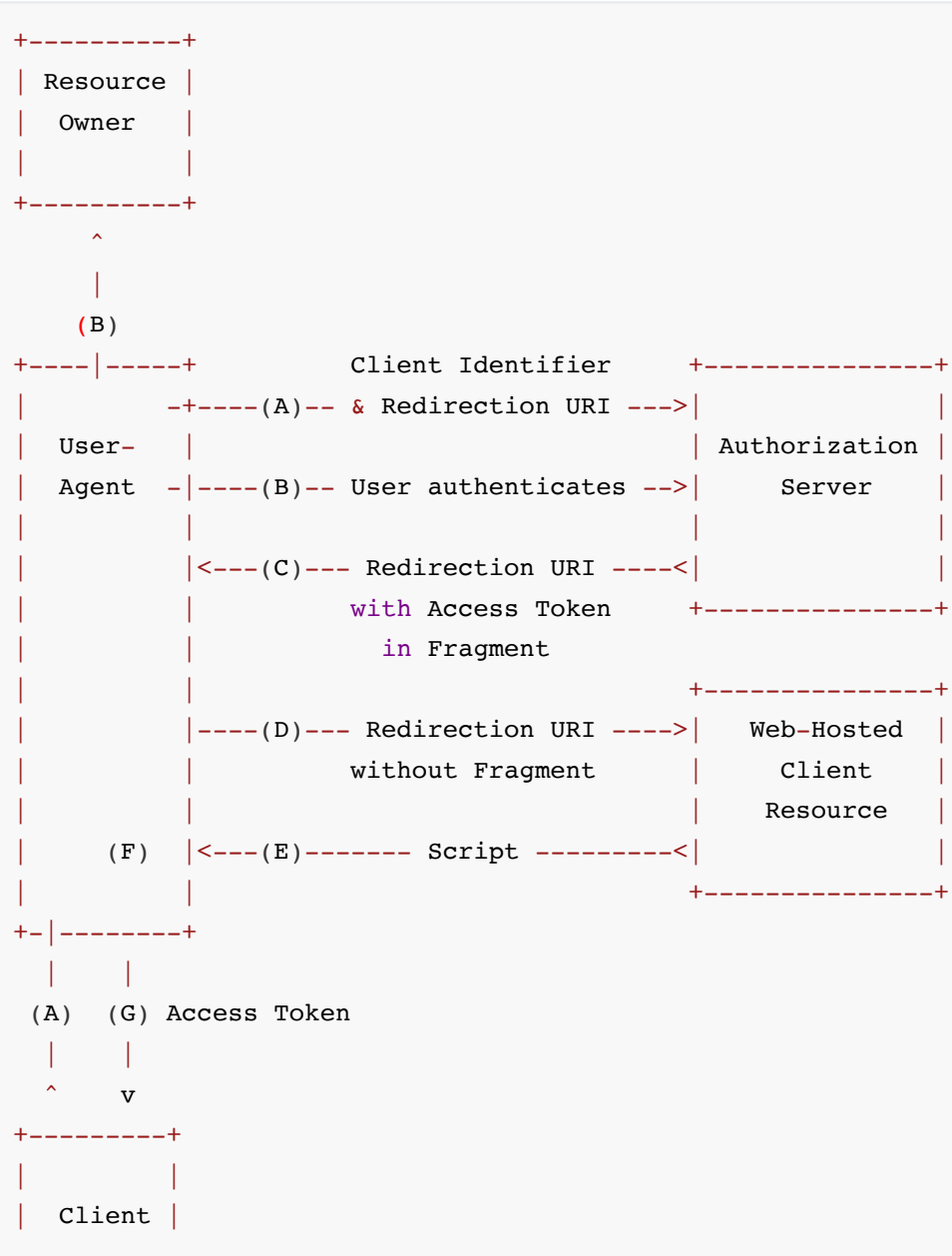
- access_token：表示访问令牌，必选项。
- token_type：表示令牌类型，该值大小写不敏感，必选项，可以是bearer类型或mac类型。
- expires_in：表示过期时间，单位为秒。如果省略该参数，必须其他方式设置过期时间。
- refresh_token：表示更新令牌，用来获取下一次的访问令牌，可选项。
- scope：表示权限范围，如果与客户端申请的范围一致，此项可省略。

```
HTTP/1.1 200 OK
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store
  Pragma: no-cache

  {
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"example",
    "expires_in":3600,
    "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
    "example_parameter":"example_value" # 可能还有其他参数
  }
```

简化模式（implicit grant type）：

不通过第三方应用程序的服务器，直接在浏览器中向认证服务器申请令牌，跳过了"授权码"这个步骤，因此得名。所有步骤在浏览器中完成，令牌对访问者是可见的，且客户端不需要认证。



```
     |          |
     +----------+
```

他的步骤如下：

（A）客户端将用户导向认证服务器。

（B）用户决定是否给于客户端授权。

（C）假设用户给予授权，认证服务器将用户导向客户端指定的"重定向URI"，并在URI的Hash部分包含了访问令牌。

（D）浏览器向资源服务器发出请求，其中不包括上一步收到的Hash值。

（E）资源服务器返回一个网页，其中包含的代码可以获取Hash值中的令牌。

（F）浏览器执行上一步获得的脚本，提取出令牌。

（G）浏览器将令牌发给客户端。

下面是上面这些步骤所需要的参数。

A步骤中，客户端发出的HTTP请求，包含以下参数：

- response_type：表示授权类型，此处的值固定为"token"，必选项。
- client_id：表示客户端的ID，必选项。
- redirect_uri：表示重定向的URI，可选项。
- scope：表示权限范围，可选项。
- state：表示客户端的当前状态，可以指定任意值，认证服务器会原封不动地返回这个值。

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

C步骤中，认证服务器回应客户端的URI，包含以下参数：

- access_token：表示访问令牌，必选项。
- token_type：表示令牌类型，该值大小写不敏感，必选项。
- expires_in：表示过期时间，单位为秒。如果省略该参数，必须其他方式设置过期时间。
- scope：表示权限范围，如果与客户端申请的范围一致，此项可省略。
- state：如果客户端的请求中包含这个参数，认证服务器的回应也必须一模一样包含这个参数。

```
HTTP/1.1 302 Found
Location: http://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
&state=xyz&token_type=example&expires_in=3600
```

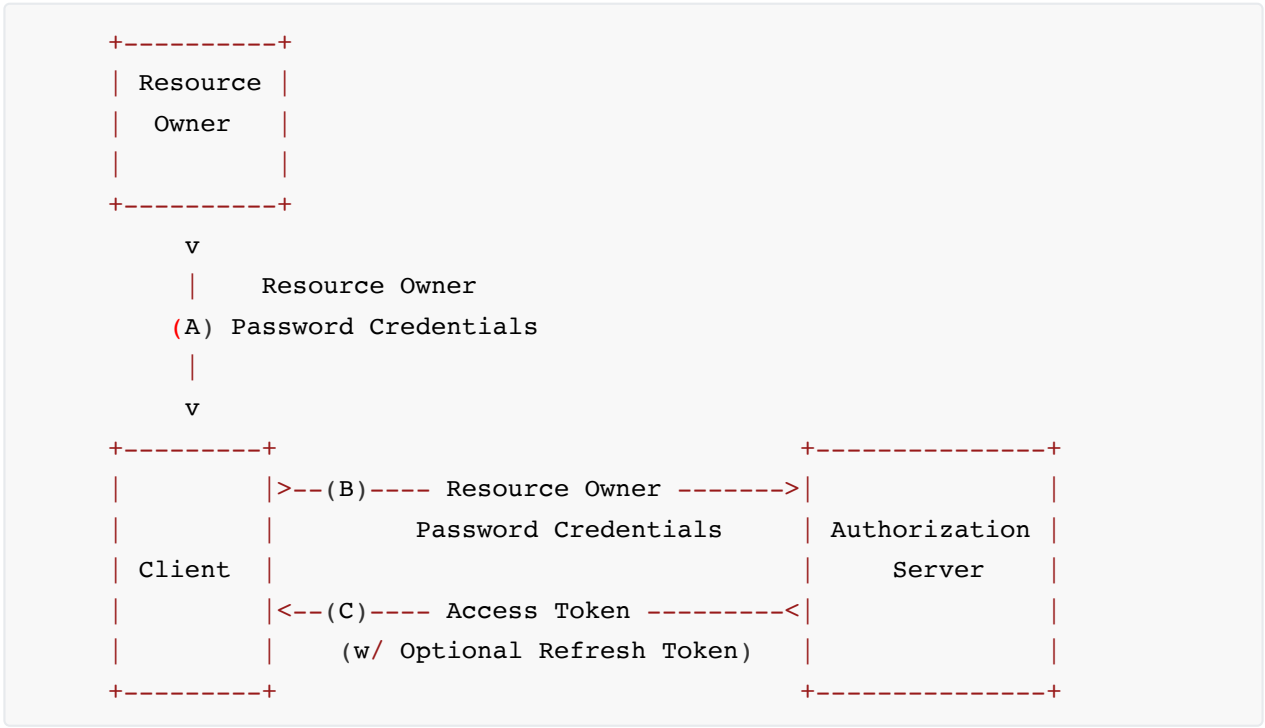
在上面的例子中，认证服务器用HTTP头信息的Location栏，指定浏览器重定向的网址。注意，在这个网址的Hash部分包含了令牌。

根据上面的D步骤，下一步浏览器会访问Location指定的网址，但是Hash部分不会发送。接下来的E步骤，服务提供商的资源服务器发送过来的代码，会提取出Hash中的令牌。

密码模式：

密码模式（Resource Owner Password Credentials Grant）中，用户向客户端提供自己的用户名和密码。客户端使用这些信息，向"服务商提供商"索要授权。

在这种模式中，用户必须把自己的密码给客户端，但是客户端不得储存密码。这通常用在用户对客户端高度信任的情况下，比如客户端是操作系统的一部分，或者由一个著名公司出品。而认证服务器只有在其他授权模式无法执行的情况下，才能考虑使用这种模式



步骤如下：

（A）用户向客户端提供用户名和密码。

（B）客户端将用户名和密码发给认证服务器，向后者请求令牌。

（C）认证服务器确认无误后，向客户端提供访问令牌。

B步骤中，客户端发出的HTTP请求，包含以下参数：

- grant_type：表示授权类型，此处的值固定为"password"，必选项。
- username：表示用户名，必选项。
- password：表示用户的密码，必选项。
- scope：表示权限范围，可选项。

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

     grant_type=password&username=johndoe&password=A3ddj3w
```

C步骤中，认证服务器向客户端发送访问令牌，下面是一个例子。

```
HTTP/1.1 200 OK
   Content-Type: application/json;charset=UTF-8
   Cache-Control: no-store
   Pragma: no-cache

   {
     "access_token":"2YotnFZFEjr1zCsicMWpAA",
     "token_type":"example",
     "expires_in":3600,
     "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
     "example_parameter":"example_value"
   }
```

客户端模式：

客户端模式（Client Credentials Grant）指客户端以自己的名义，而不是以用户的名义，向"服务提供商"进行认证。严格地说，客户端模式并不属于OAuth框架所要解决的问题。在这种模式中，用户直接向客户端注册，客户端以自己的名义要求"服务提供商"提供服务，其实不存在授权问题。

它的步骤如下：

（A）客户端向认证服务器进行身份认证，并要求一个访问令牌。

（B）认证服务器确认无误后，向客户端提供访问令牌。

A步骤中，客户端发出的HTTP请求，包含以下参数：

- grant*type：表示授权类型，此处的值固定为"client*credentials"，必选项。
- scope：表示权限范围，可选项。

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

   grant_type=client_credentials
```

认证服务器必须以某种方式，验证客户端身份。

B步骤中，认证服务器向客户端发送访问令牌，下面是一个例子。

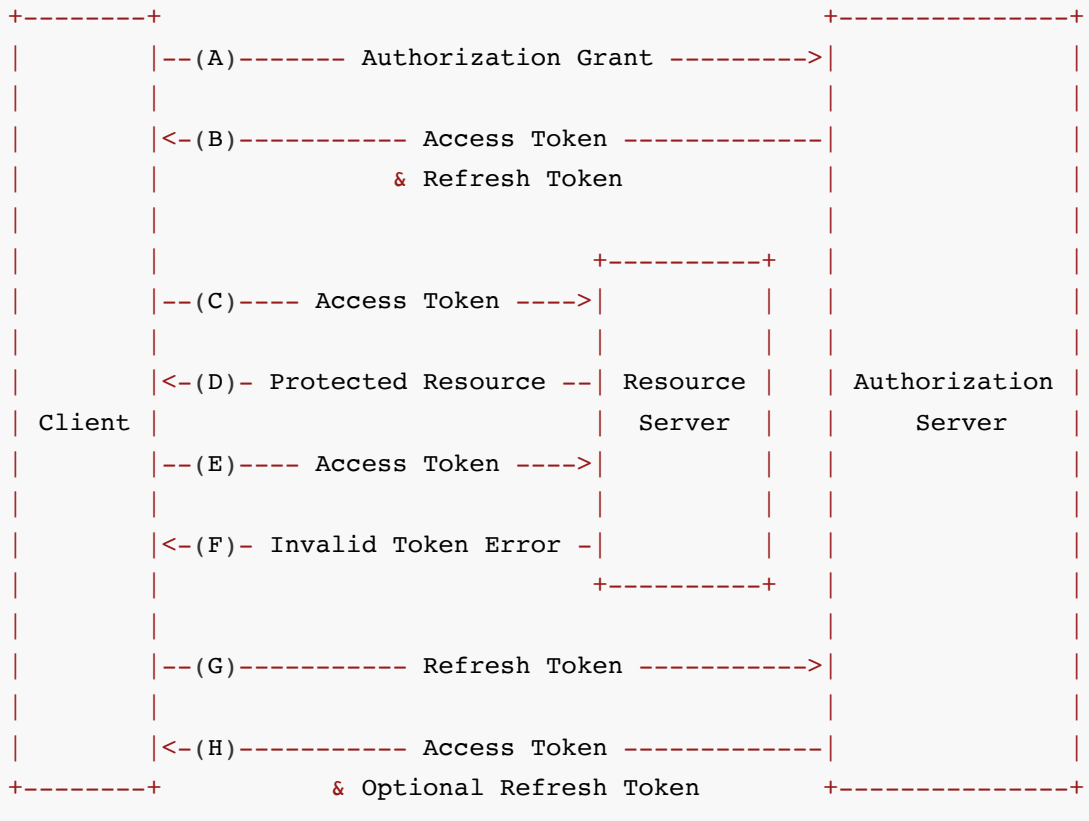
```
HTTP/1.1 200 OK
    Content-Type: application/json;charset=UTF-8
    Cache-Control: no-store
    Pragma: no-cache

    {
      "access_token":"2YotnFZFEjr1zCsicMWpAA",
      "token_type":"example",
      "expires_in":3600,
      "example_parameter":"example_value"
    }
```

七、刷新token

刷新令牌是用于获取访问令牌的凭据。 刷新令牌由授权服务器发布给客户端，并且当当前访问令牌用于获取新的访问令牌变得无效或过期，或获取其他访问令牌具有相同或更窄的范围（访问令牌可以具有更短生命周期，且权限少于资源授权所有者）。 发行刷新令牌是可选的，具体取决于授权服务器。 如果授权服务器发出刷新令牌，它包含访问令牌。

刷新令牌是一个字符串，代表授予的授权客户由资源所有者负责。 字符串通常对客户端。 令牌表示用于检索授权信息。 与访问令牌不同，刷新令牌是仅用于授权服务器，并且永远不会发送到资源服务器。



步骤如下： (A) 客户端发起认证授权请求。 (B) 授权服务器对客户端进行身份验证并验证 授权授予，如果有效，则颁发访问令牌和刷新令牌。 (C) 客户端通过之前的令牌，向服务器的保护资源发送请求， (D) 资源服务器验证访问令牌，如果有效，提供有效的服务。 (E) 步骤C和步骤D会重复发送请求直到access token过期，如果客户端知道access token过期。直接跳到步骤G。发送请求。

(F) 如果access token校验过期，那么资源服务返回一个错误：无效的token (G) 客户端通过与授权服务器进行身份验证并提供刷新令牌来请求新的访问令牌。 客户认证要求是基于客户端类型和授权服务器策略。

(H) 授权服务器对客户端进行身份验证并验证刷新令牌，如果有效，则颁发新的访问令牌（以及（可选）新的刷新令牌）