

理解OAuth 2.0

[OAuth](#)是一个关于授权（authorization）的开放网络标准，在全世界得到广泛应用，目前的版本是2.0版。

本文对OAuth 2.0的设计思路和运行流程，做一个简明通俗的解释，主要参考材料为[RFC 6749](#)。



一、应用场景

为了理解OAuth的适用场合，让我举一个假设的例子。

有一个"云冲印"的网站，可以将用户储存在Google的照片，冲印出来。用户为了使用该服务，必须让"云冲印"读取自己储存在Google上的照片。



问题是只有得到用户的授权，Google才会同意"云冲印"读取这些照片。那么，"云冲印"怎样获得用户的授权呢？

传统方法是，用户将自己的Google用户名和密码，告诉"云冲印"，后者就可以读取用户的照片了。这样的做法有以下几个严重的缺点。

- (1) "云冲印"为了后续的服务，会保存用户的密码，这样很不安全。
- (2) Google不得不部署密码登录，而我们知道，单纯的密码登录并不安全。
- (3) "云冲印"拥有了获取用户储存在Google所有资料的权力，用户没法限制"云冲印"获得授权的范围和有效期。
- (4) 用户只有修改密码，才能收回赋予"云冲印"的权力。但是这样做，会使得其他所有获得用户授权的第三方应用程序全部失效。
- (5) 只要有一个第三方应用程序被破解，就会导致用户密码泄漏，以及所有被密码保护的数据泄漏。

OAuth就是为了解决上面这些问题而诞生的。

二、名词定义

在详细讲解OAuth 2.0之前，需要了解几个专用名词。它们对读懂后面的讲解，尤其是几张图，至关重要。

- (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。

知道了上面这些名词，就不难理解，OAuth的作用就是让"客户端"安全可控地获取"用户"的授权，与"服务商提供商"进行互动。

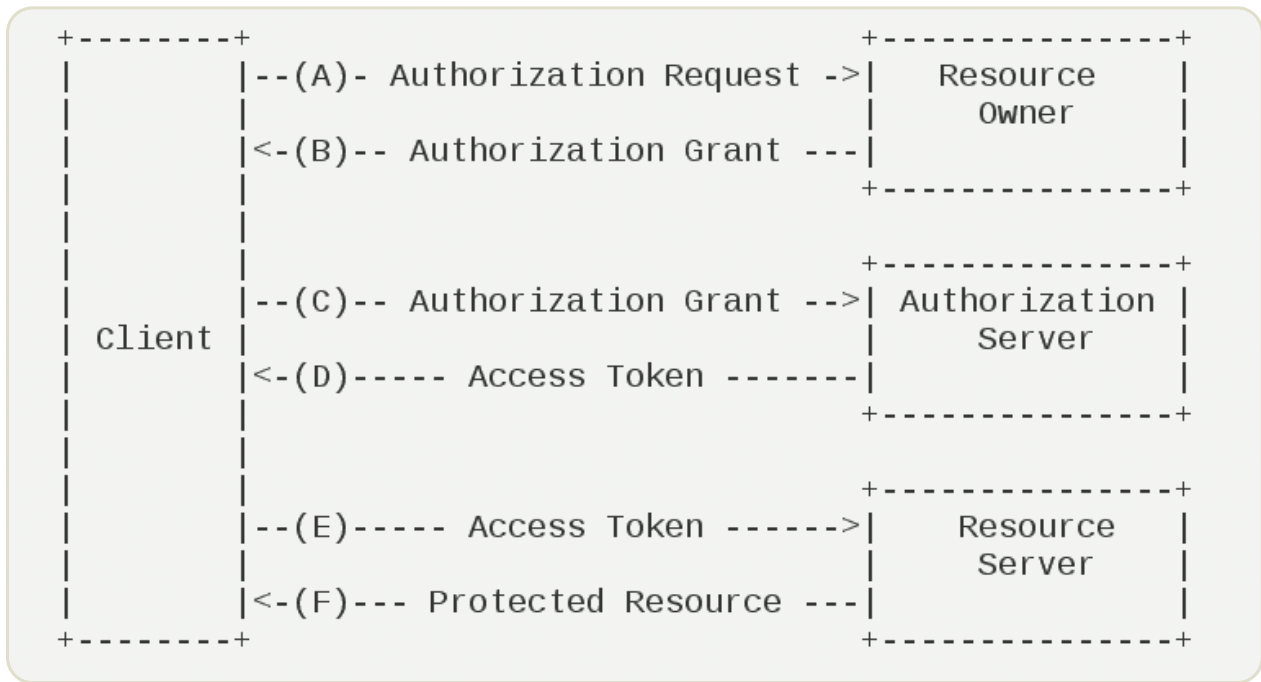
三、OAuth的思路

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”，只能登录授权层，以此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，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。

四、运行流程

OAuth 2.0的运行流程如下图，摘自RFC 6749。



- (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。

不难看出来，上面六个步骤之中，B是关键，即用户怎样才能给予客户端授权。有了这个授权以后，客户端就可以获取令牌，进而凭令牌获取资源。

下面一一讲解客户端获取授权的四种模式。

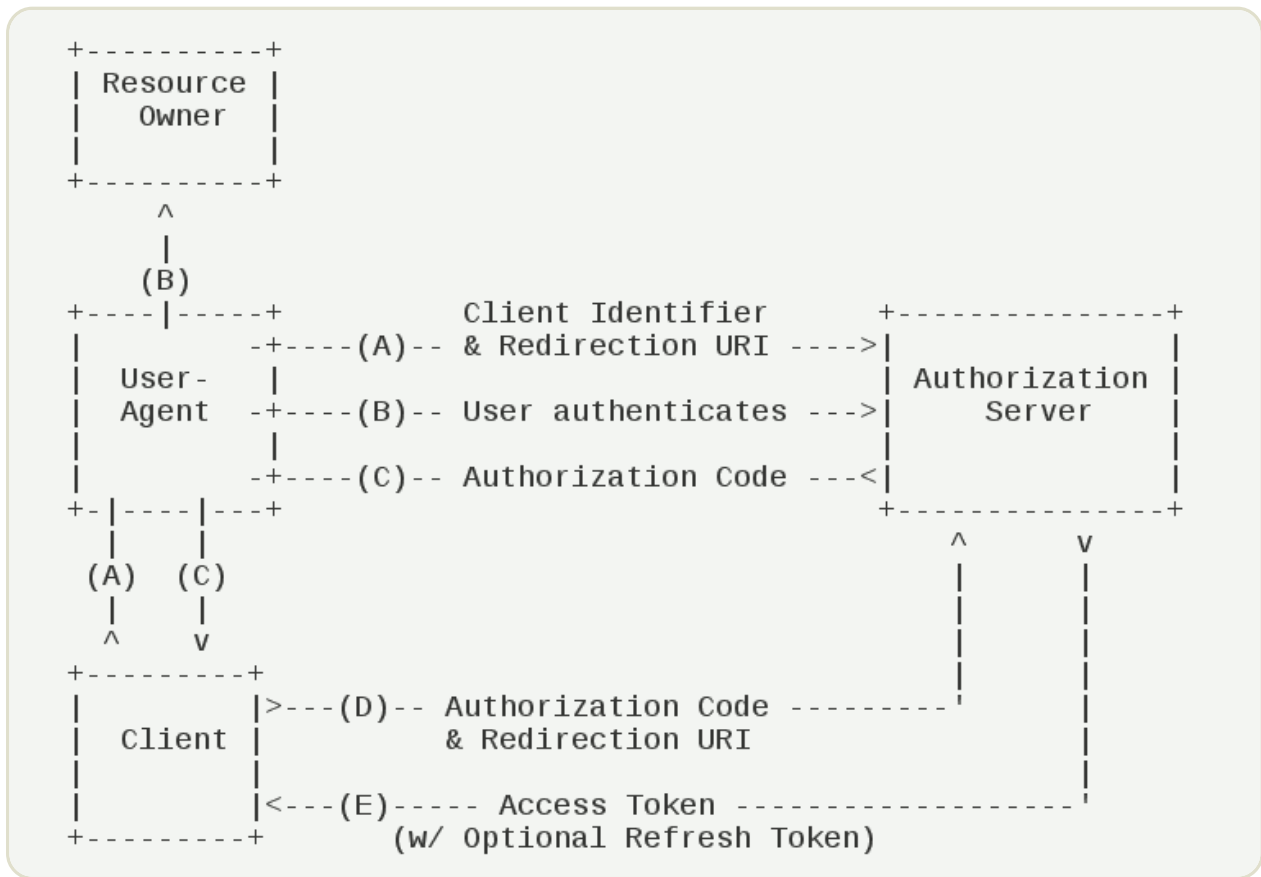
五、客户端的授权模式

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）

六、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。



它的步骤如下：

- (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**）。

下面是上面这些步骤所需要的参数。

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。

下面是一个例子。

```
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
```

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。

下面是一个例子。

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=Splxl0BeZQQYbYS6WxSbIA
    &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。

下面是一个例子。

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

grant_type=authorization_code&code=Splxl0BeZQQYbYS6WxSbIA
&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。

下面是一个例子。

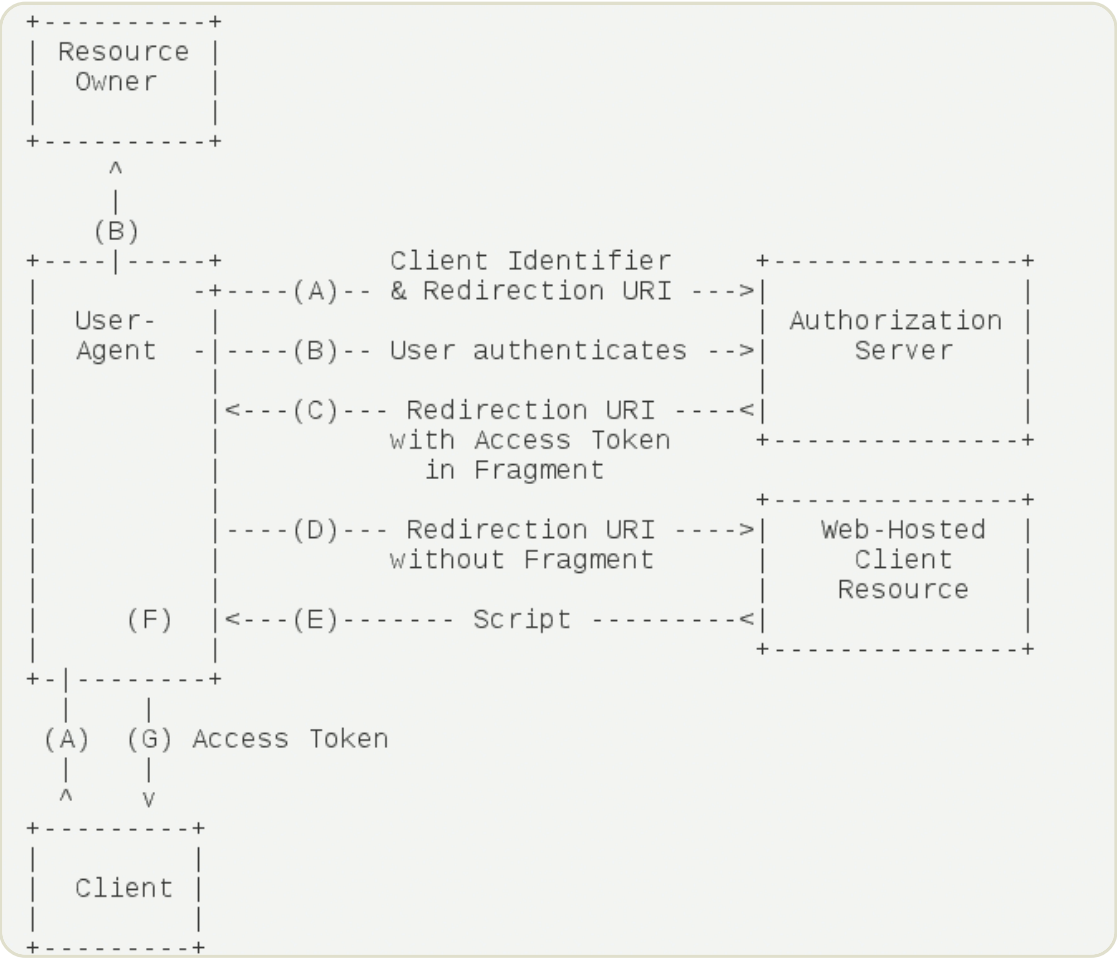
```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

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

从上面代码可以看到，相关参数使用JSON格式发送（Content-Type: application/json）。此外，HTTP头信息中明确指定不得缓存。

七、简化模式

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。



它的步骤如下：

- (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。

下面是一个例子。

```
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。

下面是一个例子。

```
HTTP/1.1 302 Found
Location: http://example.com/cb#access\_token=2YotnFZFEjr1zCsicMWpAA
    &state=xyz&token\_type=example&expires\_in=3600
```

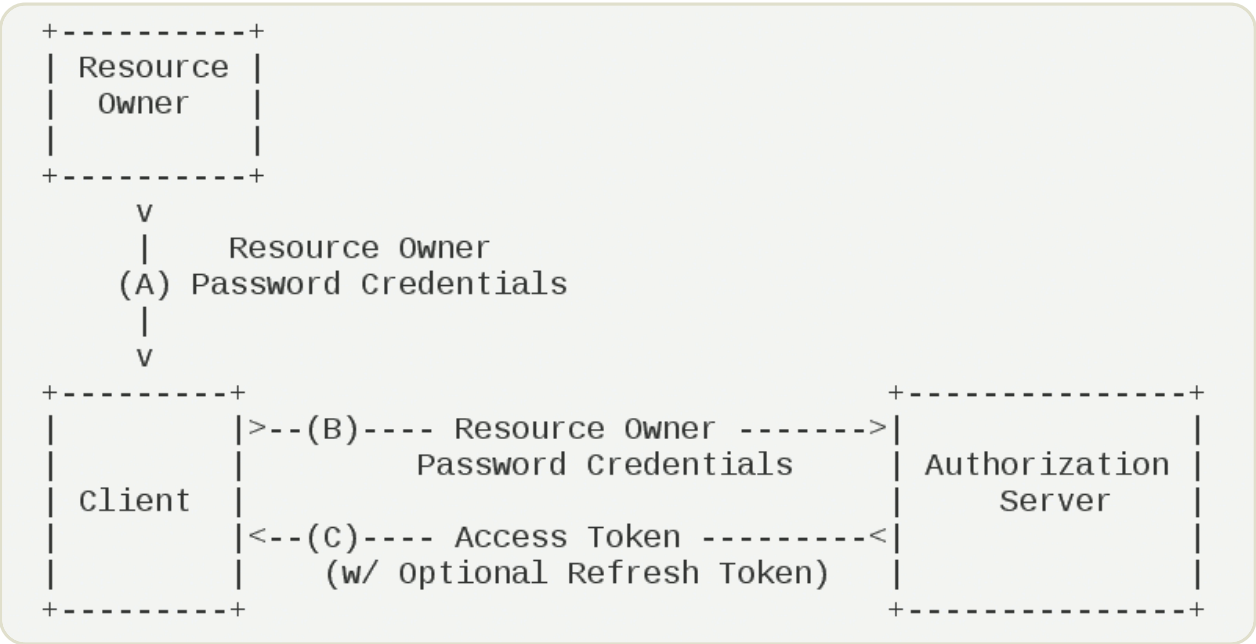
在上面的例子中, 认证服务器用HTTP头信息的Location栏, 指定浏览器重定向的网址。注意, 在这个网址的Hash部分包含了令牌。

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。

八、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。



它的步骤如下：

- (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。

下面是一个例子。

```
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

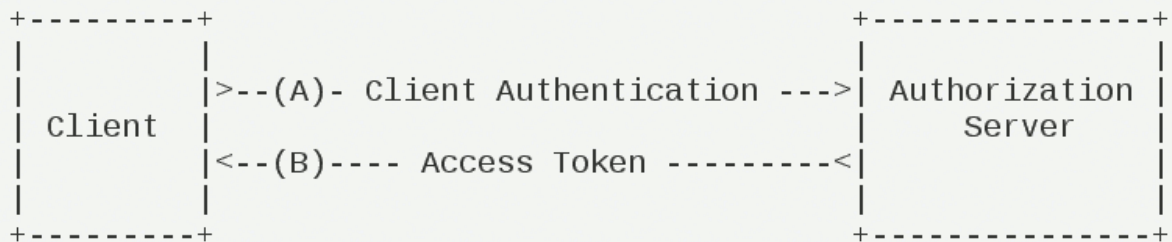
{
  "access_token": "2YotnFZFEjr1zCsicMWpAA",
  "token_type": "example",
  "expires_in": 3600,
  "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter": "example_value"
}
```

上面代码中，各个参数的含义参见《授权码模式》一节。

整个过程中，客户端不得保存用户的密码。

九、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：

- **granttype**：表示授权类型，此处的值固定为"clientcredentials"，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"
}
```

上面代码中，各个参数的含义参见《授权码模式》一节。

十、更新令牌

如果用户访问的时候，客户端的“访问令牌”已经过期，则需要使用“更新令牌”申请一个新的访问令牌。

客户端发出更新令牌的HTTP请求，包含以下参数：

- **granttype**：表示使用的授权模式，此处的值固定为“**refresh_token**”，必选项。
- **refresh_token**：表示早前收到的更新令牌，必选项。
- **scope**：表示申请的授权范围，不可以超出上一次申请的范围，如果省略该参数，则表示与上一次一致。

下面是一个例子。

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

grant_type=refresh_token&refresh_token=tGzv3J0kF0XG5Qx2TlKWIA
```