# OAuth2.0系列教程

极客学院出版

# 前言

OAuth 2.0 是目前比较流行的做法，它率先被Google, Yahoo, Microsoft, Facebook等使用。之所以标注为2.0，是因为最初有一个1.0协议，但这个1.0协议被弄得太复杂，易用性差，所以没有得到普及。2.0是一个新的设计，协议简单清晰，但它并不兼容1.0，可以说与1.0没什么关系。

> 致谢
>
> 内容翻译：http://ifeve.com/oauth2-tutorial-all/
>
> 作者：Jakob Jenkov 译者：林浩 校对：郭蕾

| 更新日期 | 更新内容 |
| --- | --- |
| 2015-06-10 | OAuth 2.0系列教程 |

# 目录


前言 . . . . . 1

第 1 章　引言 . . . . . 3

第 2 章　综述 . . . . . 5

第 3 章　角色 . . . . . 8

第 4 章　客户端类型 . . . . . 10

第 5 章　授权 . . . . . 14

第 6 章　端点 . . . . . 19

第 7 章　请求和响应 . . . . . 22

第 8 章　授权码授权 . . . . . 24

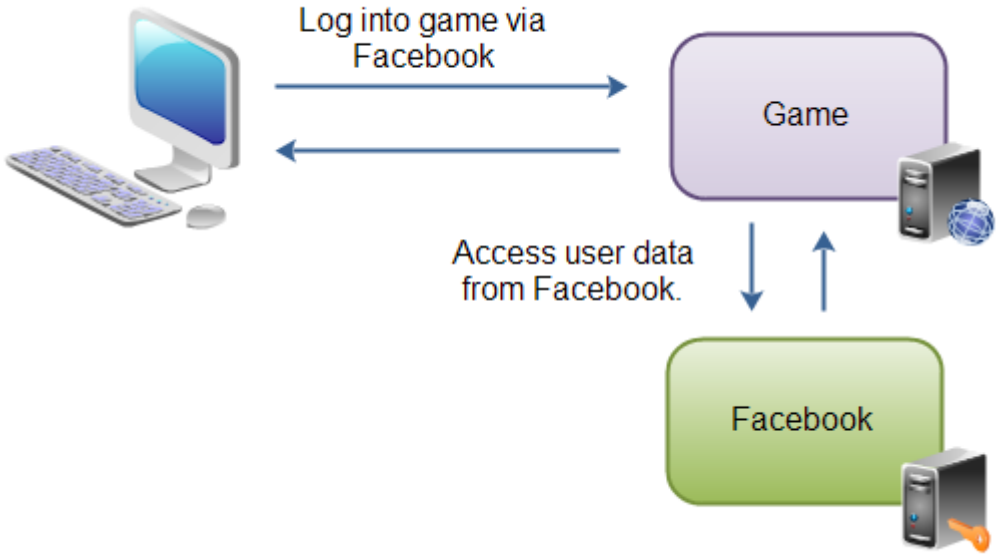
第 9 章　契约请求和响应 . . . . . 26

第 10 章　资源拥有者密钥证书授权请求和响应 . . . . . 28

第 11 章　客户端证书请求和响应 . . . . . 30

# 1 引言

OAuth 2.0是一个应用之间彼此访问数据的开源授权协议。比如，一个游戏应用可以访问Facebook的用户数据或者一个基于地理的应用可以访问Foursquare的用户数据等。下面是一张阐述该概念的图：




> OAuth 2.0怎么通过应用共享数据的例子

用户访问web游戏应用，该游戏应用要求用户通过Facebook登录。用户登录到了Facebook,再重定向会游戏应用， 游戏应用就可以访问用户在Facebook的数据了，并且该应用可以代表用户向Facebook调用函数(如发送状态更新)。

### OAuth 2.0实用案例

OAuth 2.0要么用来创建一个能够从其他应用读取用户信息的应用(如上面图表中的游戏应用)，要么创建一个使其他应用访问自己的用户数据的应用(如上面例子中的Facebook)。OAuth 2.0是OAuth 1.0的替代品，OAuth 1.0更加复杂。OAuth 1.0涉及到了证书等，而OAuth 2.0更简单，它不需要任何证书，仅仅就SSL/TLS。

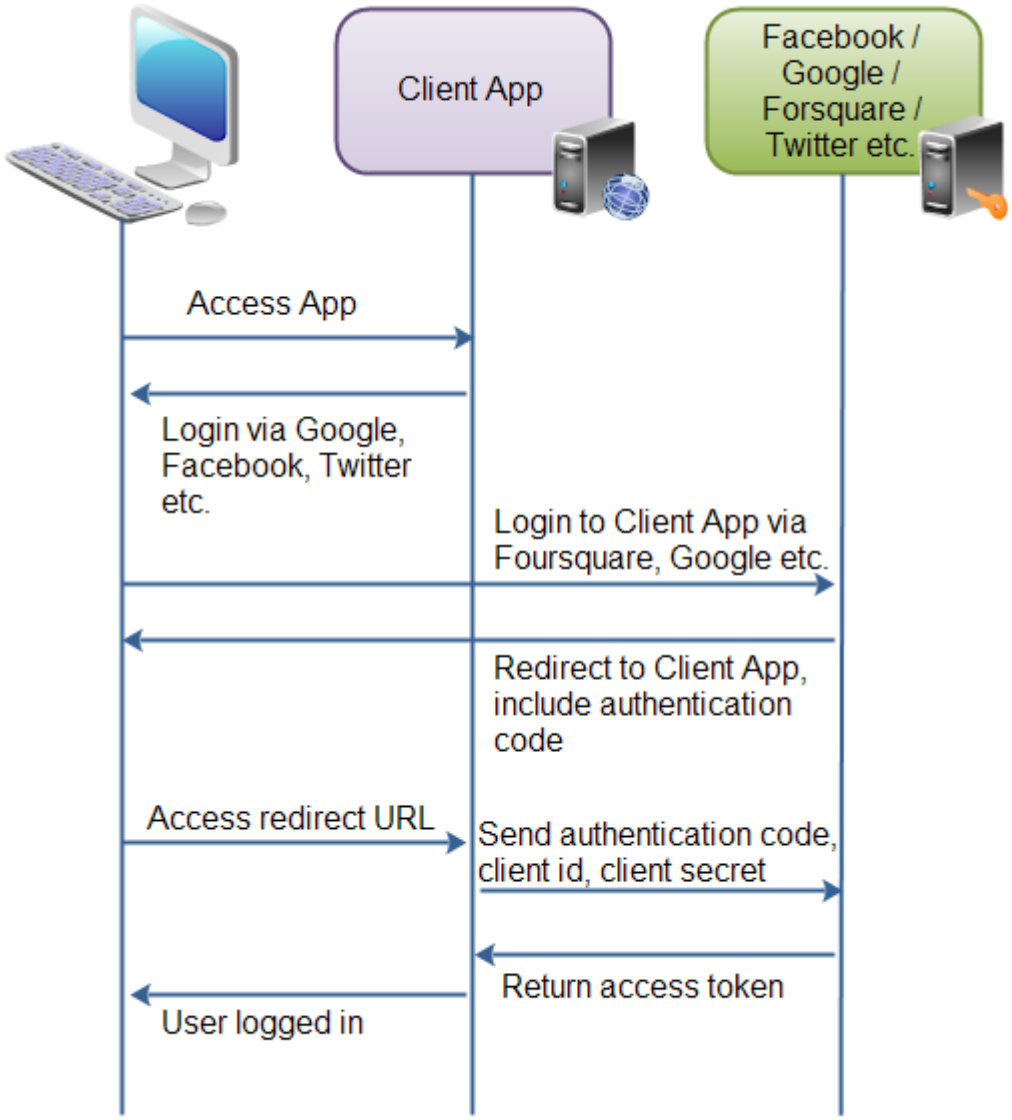
### OAuth 2.0规范

该指南的目标是提供一个OAuth 2.0的很容易理解的概述，但是不会描述规范的每一个细节。如果你想实现OAuth 2.0, 你将很有可能要全面学习该规范，你可以在这里找到该规范：http://tools.ietf.org/html/draft-ietf-oauth-v2-23

# 2

# 综述

如引言所说的，OAuth 2.0是一个能够使应用彼此访问数据的开放授权协议，这里我们将阐述该协议是怎么工作的以及规范中提到的概念。该图说明了整个授权过程：




> OAuth 2.0怎样被用来在应用间共享数据的例子

第一步，用户访问客户端web应用。应用中的按钮”通过Facebook登录”(或者其他的系统，如Google或Twitter)。

第二步，当用户点击了按钮后，会被重定向到授权的应用(如Facebook)。用户登录并确认授权应用中的数据给客户端应用。

第三步，授权应用将用户重定向到客户端应用提供的URI，提供这种重定向的URI通常是通过注册客户端应用程序与授权应用程序完成。在注册中，客户端应用的拥有者组注册该重定向URI，在注册过程中认证应用也会给客户端应用客户端标识和密码。在URI后追加一个认证码。该认证码代表了授权。

第四步，用户在客户端应用访问网页被定位到重定向的URI。在背后客户端应用连接授权应用，并且发送在重定向请求参数中接收到的客户端标识，客户端密码和认证码。授权应用将返回一个访问口令。

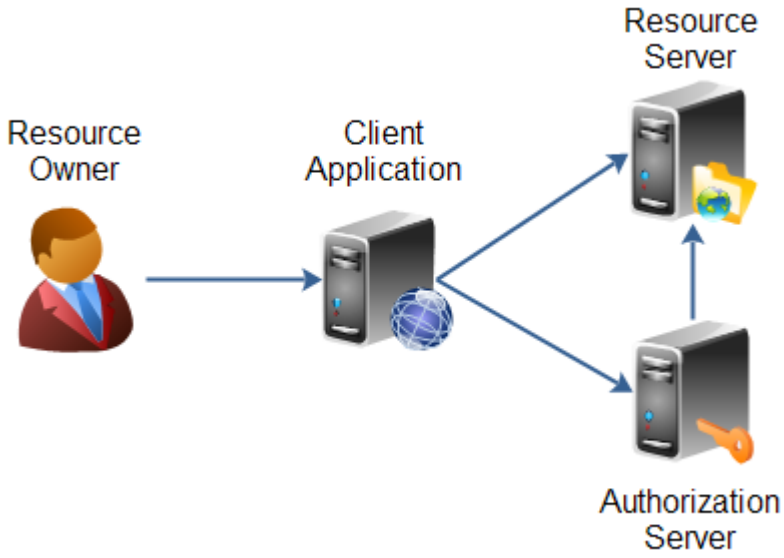
一旦客户端有了访问口令，该口令便可以被发送到Facebook、Google、Twitter等来访问登录用户的资源。

# 3

# 角色

OAuth 2.0为用户和应用定义了如下角色：

- 资源拥有者
- 资源服务器
- 客户端应用
- 授权服务器
- 这些角色在下图中表示为：




> OAuth 2.0规范中的角色定义

资源拥有者是指拥有共享数据的人或应用。比如Facebook或者Google的用户就是是资源拥有者，他们拥有的资源就是他们的数据。资源拥有者在上图中被描述为人，这也是最常见的情况。但资源拥有者也可以是一个应用。OAuth 2.0规范中包含这两种可能性。

资源服务器是指托管资源的服务器。比如，Facebook或Google就是资源服务器(或者有一个资源服务器)。

客户端应用是指请求访问存储在资源服务器的资源的应用。资源被资源拥有者所拥有。客户端应用可以是一个请求访问用户Facebook账号的第三方游戏。

授权服务器是指授权客户端应用能够访问资源拥有者所拥有的资源。授权服务器和资源服务器可以是同一个服务器，但不是必须的。如果这两个服务器是分开的，OAuth 2.0没有讨论这个两个服务器应该如何通信。这是由资源服务器和授权服务器开发者自己设计决定的。

# 4

# 客户端类型

OAuth 2.0客户端角色被细分为一系列类型和配置，本节将阐述这些类型和配置。

OAuth 2.0规范定义了两种客户端类型：

- 保密的
- 公有的

保密的客户端能够对外部保持客户端密码保密。该客户端密码是由授权服务器分配给客户端应用的。为了避免欺骗，该密码是授权服务器用来识别客户端的。例如一个保密的客户端可以是web应用，除了管理员，没有任何人能够访问服务器和看到该密码。

公有的客户端不能使客户端密码保密。比如移动手机应用或桌面应用会将密码嵌入在内部。这样的应用可能被破解，并且泄漏密码。这同于在用户的浏览器上运行的JavaScript应用。用户可以使用一个JavaScript调试器来寻找到应用程序，并查看客户端密码。

**客户端配置**

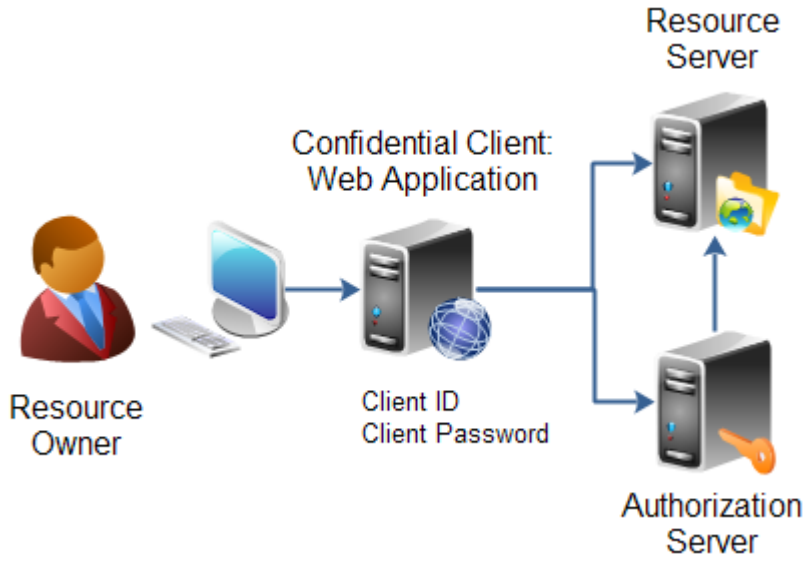
OAuth 2.0规范也提到了一系列客户端配置文件。这些配置文件是具体类型的应用程序，这可以是保密或公开的。这些配置文件有：

- web应用
- 用户代理
- 原生

### Web应用

Web应用是指运行在Web服务器内的应用。实际上，Web应用典型地由浏览器部分和服务端部分组成。如果Web应用需要访问资源服务器(如Facebook账号)，然后客户端密码被保存在服务器上。因此密码是保密的。

这里阐释了一个保密的客户端应用：

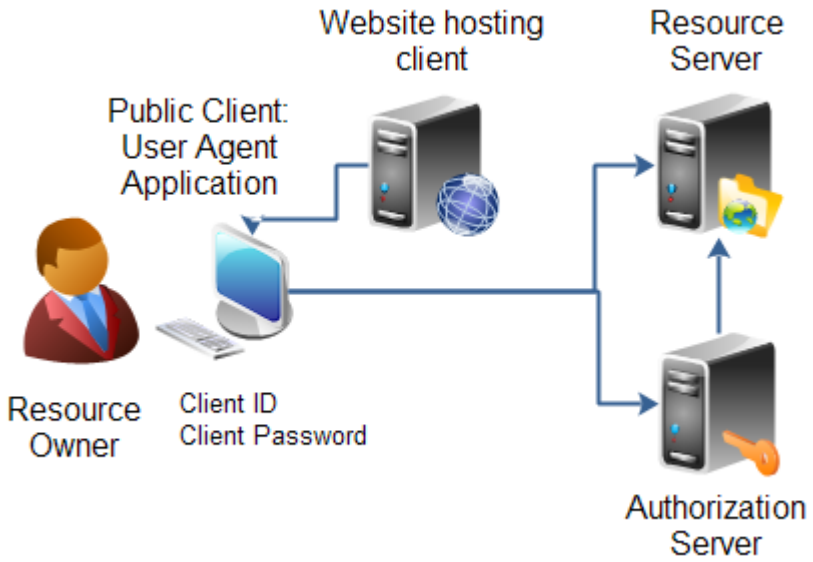



> 保密的客户端：web应用

**用户代理应用**

用户代理应用比如运行在浏览器上的的JavaScript应用。浏览器是用户代理。用户代理应用可以保存在web服务器上，但应用程序只运行一次下载的用户代理。一个例子就像一个JavaScript游戏只能运行在浏览器里。

这里阐释了一个客户端用户代理应用：

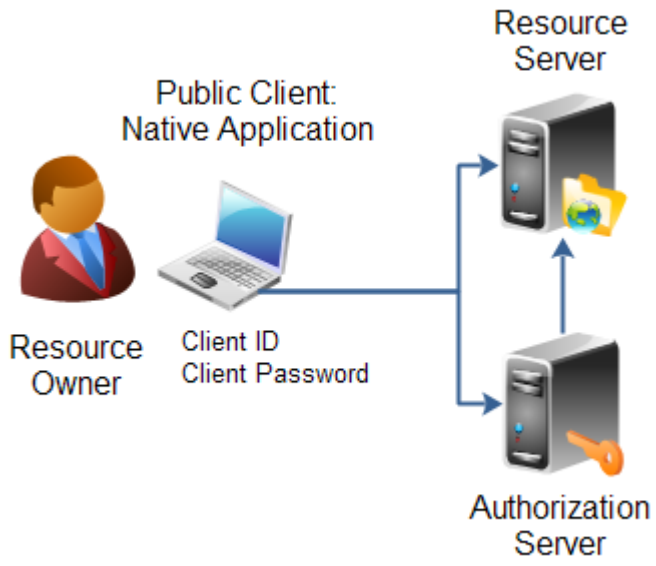



> 公有客户端：用户代理应用

**原生应用**

原生应用比如桌面应用或移动手机应用。原生应用典型地被安装在用户计算机或设备(手机，平板等)上。因此客户端密码也被存储在用户计算机或设备上。

这里阐释了客户端原生应用:




> 公有客户端：本地应用

**混合应用**

有些应用是这些配置的混合使用。比如本地应用也可以有服务器部分，来做一些工作(如数据存储)。OAuth2.0规范没有提及这种混合型。然而，在大多数情况下，混合型将能够使用这些配置文件的认证模型。

# 5

# 授权

当一个客户端应用想要访问拥有者托管在资源服务器的资源时，它必须先获得授权，本节将讲述客户端如何获取授权。

**客户端标识，客户端密钥和重定向URI**

在客户端应用能请求访问资源服务器的资源之前，客户端应用程序，必须先在资源服务器相关联的授权服务器中进行注册。

注册一个一次性的任务。一旦注册了，除非客户端注册被取消了，注册将持续有效。

注册后客户端应用将由授权服务器分配客户端标识和密钥。在授权服务器上，客户端标识和密钥是唯一标识客户端应用的。如果客户端应用注册了多个授权服务器(如Facebook, Twitter和Google等)，每一个授权服务器将发出唯一的标识给该客户端应用。

无论什么时候客户端应用，想要访问同样资源服务器上的资源，它都需要通过发送客户端标识和密钥到授权服务器来验证自己。

在注册过程中，客户端应用也注册了一个重定向URI，当资源拥有者授权给客户端应用时，该重定向URI会被使用。当资源拥有者成功的通过授权服务器授权给客户端应用时，资源拥有者被重定向回客户端应用，再跳转到该重定向URI。

- 授权批准

授权批准由资源服务器，及与其相关的授权服务器，给予客户端应用。

OAuth 2.0列举四种不同类型授权批准，每一种类型都有不同的安全特性。这些授权批准类型为：

- 授权码
- 契约
- 资源拥有者密钥证书
- 客户端证书
- 每种授权批准在下文都会提到。

**授权码**

用授权码来授权批准原理如下：资源拥有者(用户)访问客户端应用。客户端应用告诉用户通过授权服务器(如Facebook, Google和Twitter等)来登录到客户端应用。

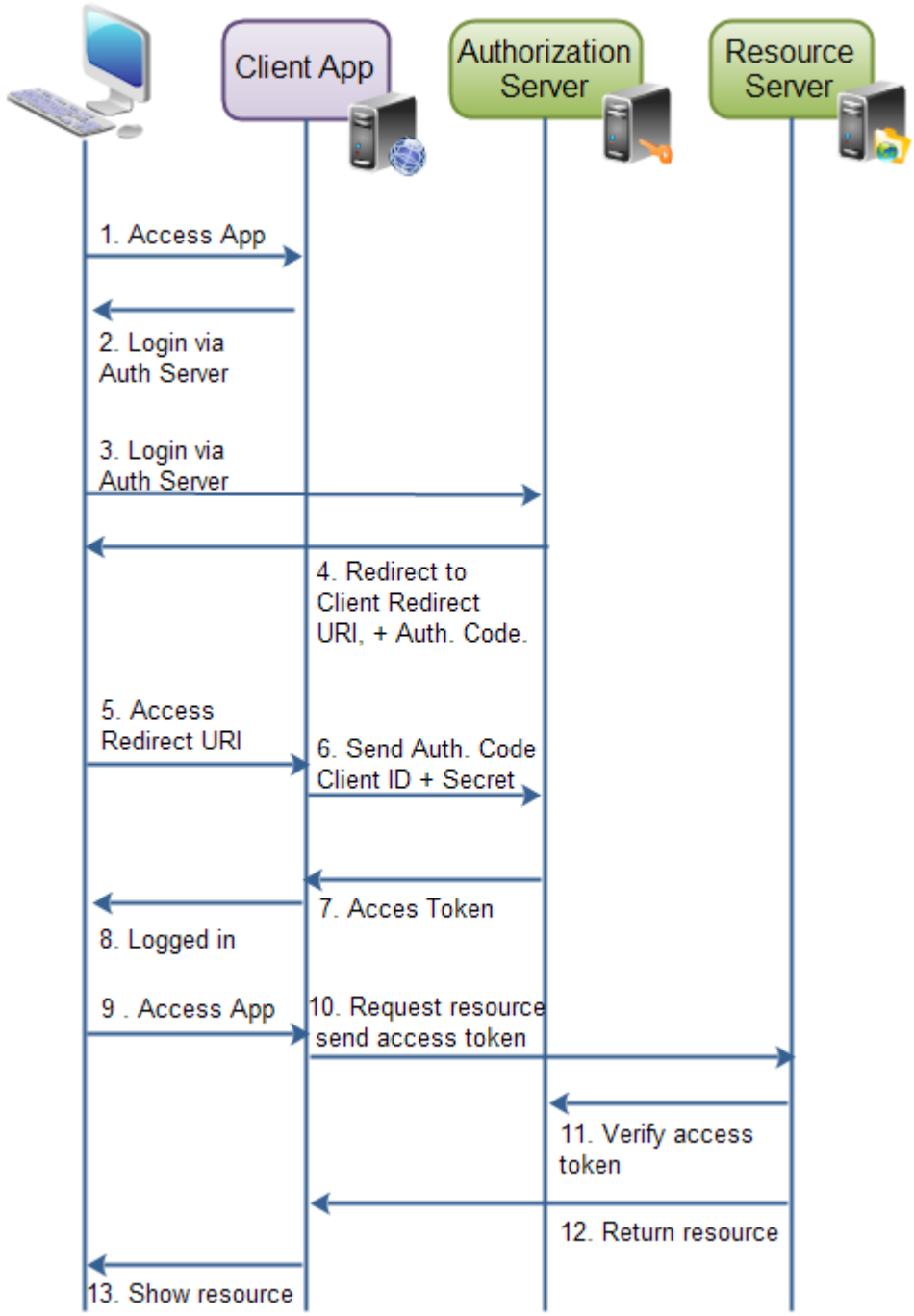
为了通过授权服务器登录，用户通过客户端应用被重定向到授权服务器。客户端应用发送它的客户端标识给授权服务器，那么授权服务器就知道是哪个应用尝试访问受保护的资源。当被重定向回客户端应用时，授权服务器发

送给用户特定的重定向URI, 即客户端已经提前与授权服务器注册。随着重定向，授权服务器发送一个代表授权的授权码。

当在客户端应用的重定向URI被访问时，客户端应用直接连接授权服务器。客户端应用发送授权码，客户端标识及密钥，如果客户端应用能接受这些值，那么授权服务器返回一个访问令牌。

现在客户端应用就可以用该访问令牌请求资源服务器的资源了。该访问令牌可作为客户端授权和授权访问资源。

下面是当用授权码授权客户端应用时的授权过程：



> 通过授权码授权

**契约**

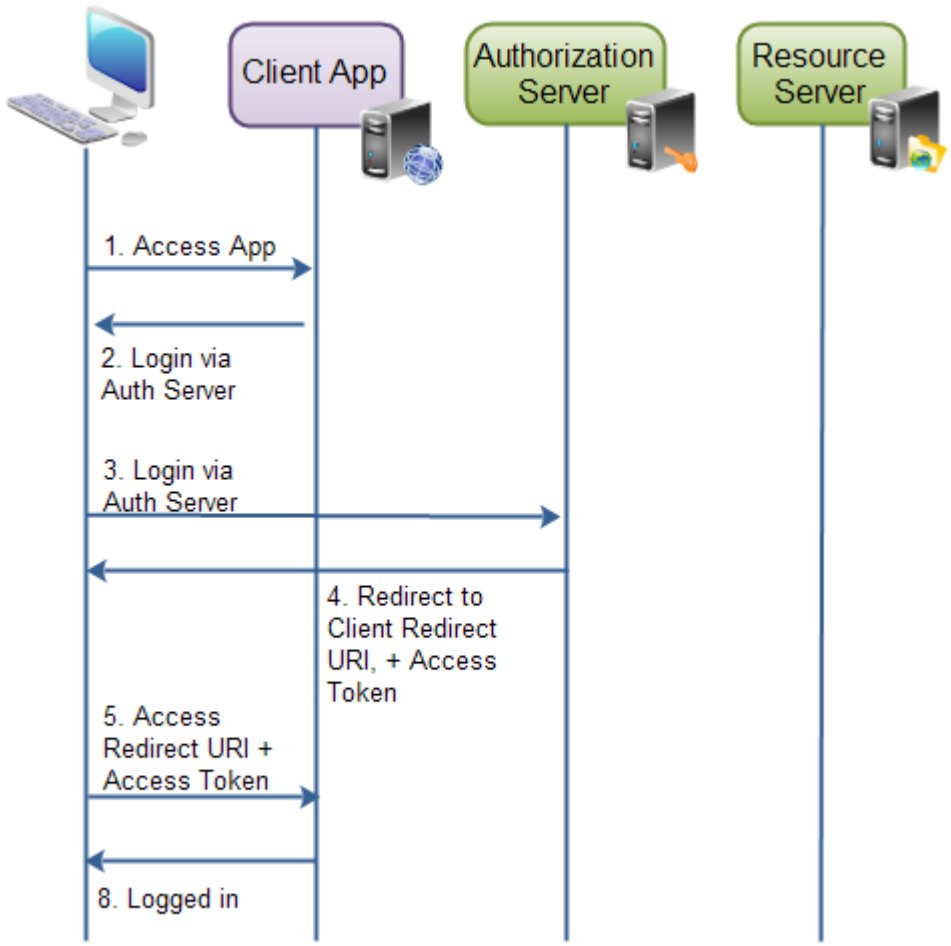
契约授权类似于授权码授权，除了用户完成授权后，访问令牌返回给客户端应用外。当用户代理被重定向到重定向URI时，访问令牌因此被返回。

当然这意味着访问令牌可以被用户代理访问，或者在契约授权过程中参与的原生应用。访问令牌在web服务器上不是安全存储的。

进一步说，客户端应用可以只发送它的客户端标识给授权服务器。如果客户端也发送它的密钥，那么客户端密钥将不得不保存在用户代理或原生应用里，那将使它很容易被破解。

契约授权大多数用在用户代理或原生应用中。用户代理或原生应用将收到来授权服务器的访问令牌。

下面是阐释契约授权的图：




> 契约授权

### 资源拥有者密钥证书

资源拥有者证书授权方法通过客户端应用访问资源拥有者证书来工作。比如，用户可以在客户端应用输入他的Twitter用户名及密钥(证书)。该客户端应用就可以用着用户名和密钥访问用户在Twitter的资源。

用资源拥有者密钥证书要求客户端应用很多信任。你不想在那些你怀疑会滥用证书的客户端应用中输入证书。

资源拥有者密钥证书通常被用在用户代理或原生应用中。

### 客户端证书

客户端证书授权对于客户端需要在资源服务器访问资源或调用函数的情形使用，与特定的资源拥有者无关(如用户)。比如，从Foursquare获取场地列表，这并没有必要通过某个Foursquare用户才能做。

# 6

# 端点

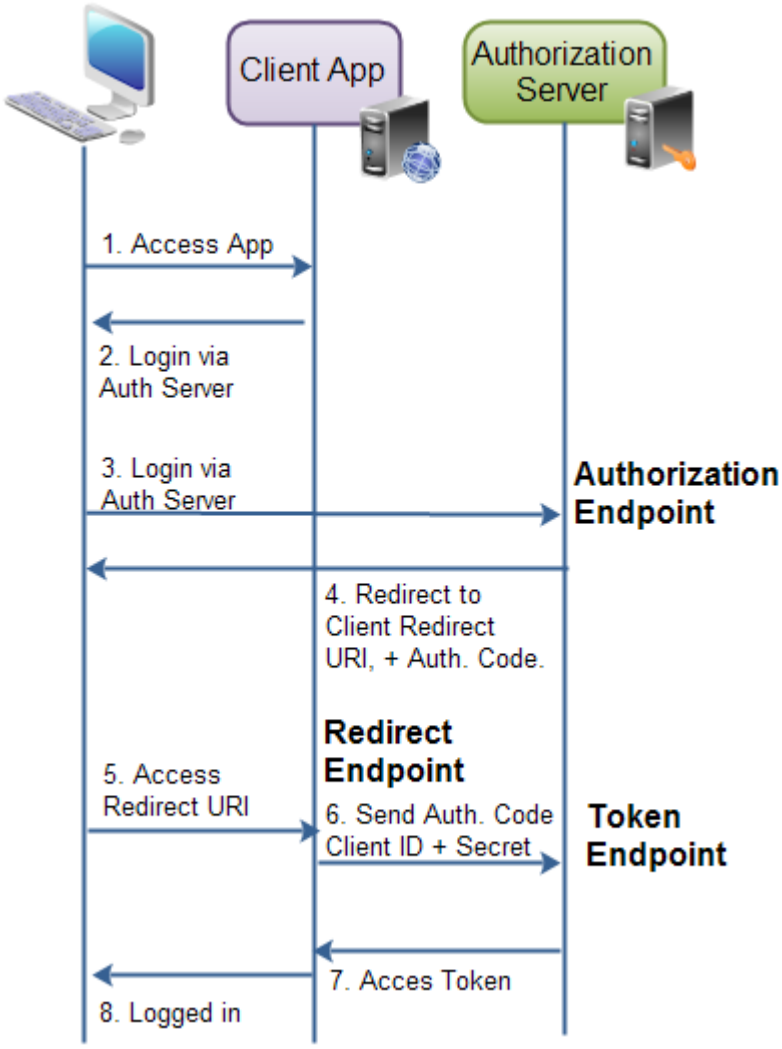
OAuth 2.0定义了一系列端点。端点典型的就是web服务器上的URI。比如，一个Java Servlet, JSP page, PHP page, ASP.NET网页等等。

这些端点定义有：

- 授权端点
- 令牌端点
- 重定向端点

授权端点和令牌端点都位于授权服务器上，重定向端点位于客户端应用上。每个端点都会在下面讲述。

这些端点在下图中阐释为：




OAuth 2.0端点

OAuth 2.0规范没有描述这些端点怎么被发现或记录。这取决于实现者来决定。大多数网站都有一个子网站开发人员来记录这些端点。

**授权端点**

授权端点是资源拥有者所登录的授权服务器，并授权给客户端应用的端点。

**令牌端点**

令牌端点是在授权服务器上为了一个访问令牌，客户端应用要交换授权码，客户端标识和客户端密钥的端点。

**重定向端点**

重定向端点是在授权端点授权以后，资源拥有者被重定向到客户端应用的端点。

# 7

# 请求和响应

当客户端应用请求授权和访问令牌时，它发送http请求到授权服务器，同它的授权和令牌端点。被发送来回的请求和响应取决于授权类型。记住，这四种授权类型：

- 授权码授权
- 契约授权
- 资源拥有者密钥证书授权
- 客户端证书授权

每一种授权类型的请求和响应的更多细节将在下文分开地阐释。

然而，下面讲述的信息大多只是一个总结。为了得到它们更多的细节描述，你可能不得不查询OAuth 2.0规范，或者你尝试集成的系统(如Facebook, Google, Twitter, Foursquare等)文档。

# 8

# 授权码授权

授权码授权总共由2个请求和2个响应组成。一个授权请求+响应，和一个令牌请求+响应。 授权请求 授权请求被发送到授权端点以获取一个授权码。这是请求中用到的参数：

| response_type | 必须。必须被设置到代码里 |
| --- | --- |
| client_id | 必须。当客户端被注册时，授权服务器要标识的客户端。 |
| redirect_uri | 可选。通过客户端注册的重定向URI。 |
| scope | 可选。请求可能的作用域。 |
| state | 可选(推荐的)。任何需要被传递到客户端请求的URI客户端的状态。 |

授权响应 授权响应包含了需要用来获取访问令牌的授权码。这是响应包括的参数：

| code | 必须。授权码 |
| --- | --- |
| state | 如果出现在请求中，必须包含。如果有的话，和客户端请求中发送的state参数一样。 |

**授权错误**如果授权期间发生错误，两种情况会发生。 第一种情形是，客户端没有被授权或识别。比如，请求中错误的重定向URI。这种情况下，授权服务器没有必要重定向资源拥有者到重定向URI，而是通知资源拥有者发生了错误。 第二种情形是，客户端被正确地授权了，但是其他某些事情失败了。这种情况下下面地错误响应会被发送到客户端，包括在重定向URI中：

| error | 必须。必须是预先定义的错误码之一。参见规范查查这些错误码及它们的含义。 |
| --- | --- |
| error_description | 可选。一段UTF-8编码的描述错误的文本。适用于开发者，而不是最终用户。 |
| error_uri | 可选。 一个指向包含人类可读的错误信息网页的URI。 |
| state | 必须。如果出现在授权请求期间，和请求中的state参数一样。 |

**令牌请求**一旦授权码被获取到了，客户端可以用它获取访问令牌。这是访问令牌请求参数：

| grant_type | 必须。必须被设置到授权码中。 |
| --- | --- |
| code | 必须。被授权服务器接收到的授权码。 |
| redirect_uri | 必须。如果请求URI包括在授权请求中，之后必须是相同的。 |

令牌响应 访问令牌请求的响应是包含访问令牌及一些更多信息的JSON字符串：

```
{ "access_token"  : "...",
  "token_type"    : "...",
  "expires_in"    : "...",
  "refresh_token" : "...",
}
```

access_type属性是授权服务器分配的访问令牌。 token_type是被授权服务器分配的令牌类型。 expires_in属性是指访问令牌过多少秒后，就不再有效。访问令牌过期值是可选的。 refresh_token属性包含令牌过期后刷新的令牌。刷新的令牌用于，一旦响应返回的不再有效时，包含一个新的访问令牌。

# 9

# 契约请求和响应

契约授权包含一个请求和一个响应。 契约授权请求 契约授权请求包含下面的参数：

| response_type | 必须。必须被设置在令牌中。 |
| --- | --- |
| client_id | 必须。当客户端被注册时，有授权服务器分配的客户端标识。 |
| redirect_uri | 可选。由客户端注册的重定向URI。 |
| scope | 可选。请求可能的作用域。 |
| state | 可选(推荐)。任何需要被传递到客户端请求的URI客户端的状态。 |

契约授权响应 契约授权包含下面的参数。注意，契约授权响应不是JSON：

| access_token | 必须。授权服务器分配的访问令牌。 |
| --- | --- |
| token_type | 必须。令牌类型。 |
| expires_in | 推荐。访问令牌过期的秒数。 |
| scope | 可选。访问令牌的作用域。 |
| state | 必须。i如果出现在授权请求期间，和请求中的state参数一样。 |

**契约授权错误**如果授权期间发生错误，两种情况会发生。 第一种情形是，客户端没有被授权或识别。比如，请求中错误的重定向URI。这种情况下，授权服务器没有必要重定向资源拥有者到重定向URI，而是通知资源拥有者发生了错误。 第二种情形是，客户端是好的，但是发生了其他事情。这种情况下下面地错误响应会被发送到客户端，包括在重定向URI中：

| error | 必须。必须是预先定义的错误码之一。参见规范查查这些错误码及它们的含义。 |
| --- | --- |
| error_description | 可选。一段UTF-8编码的描述错误的文本。适用于开发者，而不是最终用户。 |
| error_uri | 可选。 一个指向包含人类可读的错误信息网页的URI。 |
| state | 必须。如果出现在授权请求期间，和请求中的state参数一样。 |

10

# 资源拥有者密钥证书授权请求和响应

资源拥有者者密钥证书授权包含单个的请求+响应。 资源拥有者密钥证书授权请求 请求包含下面的参数：

| grant_type | 必须。必须设置到密码中。 |
| --- | --- |
| username | 必须。UTF-8编码的资源拥有者用户名。 |
| password | 必须。UTF-8编码的资源拥有者密码。 |
| scope | 可选。授权的作用域。 |

**资源拥有者密钥证书授权响应**响应是包含访问令牌的JSON结构数据。JSON结构像这样：

```
{ "access_token"  : "...",
  "token_type"    : "...",
  "expires_in"    : "...",
  "refresh_token" : "...",
}
```

access_type属性是授权服务器分配的访问令牌。 token_type是被授权服务器分配的令牌类型。 expires_in属性是指访问令牌过多少秒后，就不再有效。访问令牌过期值是可选的。 refresh_token属性包含令牌过期后刷新的令牌。刷新的令牌用于，一旦响应返回的不再有效时，包含一个新的访问令牌。

11

# 客户端证书请求和响应

客户端证书授权包含下面的参数：

| grant_type | 必须。必须设置到客户端证书中。 |
| --- | --- |
| scope | 可选。授权的作用域。 |

客户端授权响应： 客户端授权响应包含下面的参数：

```
{ "access_token" : "...",
  "token_type"   : "...",
  "expires_in"   : "...",
}
```

access_type属性是授权服务器分配的访问令牌。 token_type是被授权服务器分配的令牌类型。 expires_in属性是指访问令牌过多少秒后，就不再有效。访问令牌过期值是可选的。 新的令牌不应该被包含在这种授权类型的请求中。