**1、什么是OAuth2.0**

OAuth 2.0 是一个开放的标准协议，允许用户让第三方应用访问该用户在某一网站上存储的私密的资源（如照片，视频，联系人列表），而无需将用户名和密码提供给第三方应用。OAuth允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的网站（例如，视频编辑网站)在特定的时段（例如，接下来的2小时内）内访问特定的资源（例如，仅仅是某一相册中的视频）。这样，OAuth让用户可以授权第三方网站访问他们存储在另外服务提供者的某些特定信息，而非所有内容。例如：一个游戏可以获取Facebook中的用户信息，首先用户进入游戏的web应用，该应用要求用户通过Facebook账户登录，并定向到Facebook的登录界面，用户登录Facebook后，会重定向到之前的游戏应用。此时该应用就获取到了用户在Facebook的用户数据以及授权信息。

**2、OAuth2.0的用例**

OAuth 2.0既可以用于在某个应用内访问其它应用的用户信息，又可以提供用户授权服务供其它应用调用。

**3、OAuth2.0的名词定义**

**·Third-party Application**：第三方应用程序，又称"客户端"，会请求访问存放在资源服务器上的资源

**·HTTP Service**：HTTP服务提供商，例如：Facebook。

**·Resource Owner：**资源所有者，例如：Facebook上的一个用户就是一个资源所有者，他们所拥有的资源就是用户数据。

**·User Agent：**用户代理，例如：浏览器。

**·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。对客户端应用进行授权，授权通过后客户端应用才可以访问资源服务器上的资源。

**·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。

**4、OAuth2.0的处理过程**

首先用户访问客户端应用，在这个应用中会有一个“通过Facebook登录”的按钮。

第二步，当用户点击这个按钮时，用户被重定向到认证服务器（Facebook）。然后用户开始登录，登录成功后会被询问客户端应用是否可以使用他的用户信息。用户点击确认按钮。

第三步，认证服务器将用户重定向到客户端应用提供的URL。这个重定向URL一般会在认证服务器中进行注册，注册是由客户端应用的所有者进行的。注册完成后，认证服务器会生成一个client id和client password。重定向后的URL会有一个code参数，该参数是此次认证的一个标识。

第四步，重定向完成后，用户会进入重定向后的页面，同时客户端应用会在后台与认证服务器进行通讯，发送client id,client password和上一步获取到的code参数到认证服务器，认证服务器返回access token给客户端应用。

一旦客户端应用拿到了access token，就可以用这个token去访问Facebook提供的相关资源。

**5、ClientId、ClientPassword、Redirect Url**

客户端应用需要在认证服务器上注册，注册完成后，认证服务器会生成这个应用的client id和client password。client\_id和client\_password在同一个认证服务器中是唯一的，不会重复。客户端应用可以在多个认证服务器中进行注册（如分别在Facebook和Google中注册），不同的认证服务器会为客户端应用生成不同的client\_id和client\_password。

当客户端应用需要访问资源服务器上的资源时，首选要通过认证服务器进行认证，认证时要发送对应的client\_id和client\_password到认证服务器。

客户端应用在认证服务器进行注册时，需要填写一个重定向URL。当资源所有者对客户端应用进行授权成功后，资源所有者（可简单理解为系统用户）会被重定向到重定向URL所指定的页面。

**6、OAuth2.0的客户端类型**

OAuth 2.0规范定义了2种客户端类型：

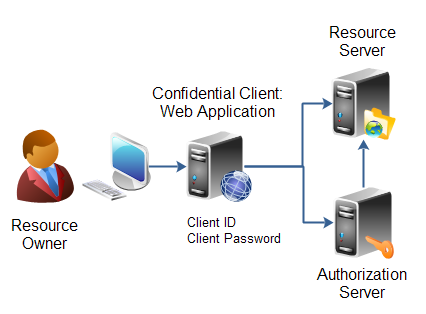
·私密型：私密型客户端会保存client password。认证服务器会给每一个客户端应用生成一个client password，认证服务器通过该client password来识别该客户端应用是一个注册过的应用，而不是其它的欺诈程序。一个web应用可以是私密性客户端，只有系统管理员可以登录这个应用的服务器和查看client password。

·公开型：公开型客户端不会保存client password。例如移动APP或桌面程序，如果client password被保存在此类应用中，就可以通过破解手段拿到client password，这是非常不安全的。

**客户端表现形式：**

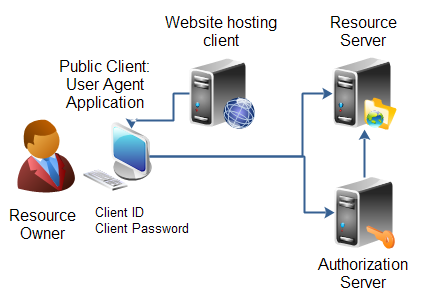
·Web Application （web应用）

web程序运行在web服务器上。web应用做认证时用到的client password是保存在服务器上的，因此是私密的。下面是一个web应用的示例图：

****

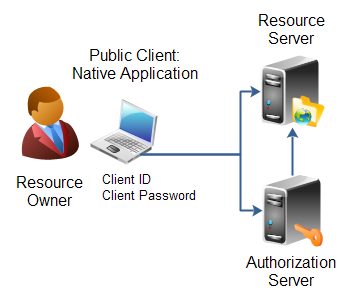
·User Agent （富web客户端）

富web客户端应用是指由javascript构建的web应用，浏览器是客户端代理。这类应用的特点是，程序是存放在web服务器上的，但是运行时，浏览器下载javascript程序到本地，直接在浏览器中执行，例如那些用javascript开发的网页版游戏。下面是一个富web客户端的示例图：



·Native （原生应用）

这里指没有后端服务器的应用，一次所有的数据和配置只能存放在客户端程序中，原生应用包括移动APP和桌面程序。原生应用直接安装在用户的设备上（电脑或手机、平板），client password会保存在用户的设备里。下面是一个原生应用的示例图：



·Hybrid（混合应用）

这类应用通常是将原生应用和web应用的开发技术混合在一起，也会有对应的后端服务器。OAuth 2.0规范中并没有提及此类应用，此类应用可以灵活选用上述三种认证类型中的任何一种。

**7、OAuth2.0的授权模式**

OAuth 2.0规范列举了4中认证授权方式，每种方式都有不同的安全特点：

·Authorization Code（授权码模式）

·Implicit（简化模式）

·Resource Owner Password Credentials（用户密码模式）

·Client Credentials（客户端模式）

**8、授权码模式**

认证过程如下：

1）资源所有者（用户）进入客户端应用。

2) 客户端应用让用户通过认证服务器进行登录。

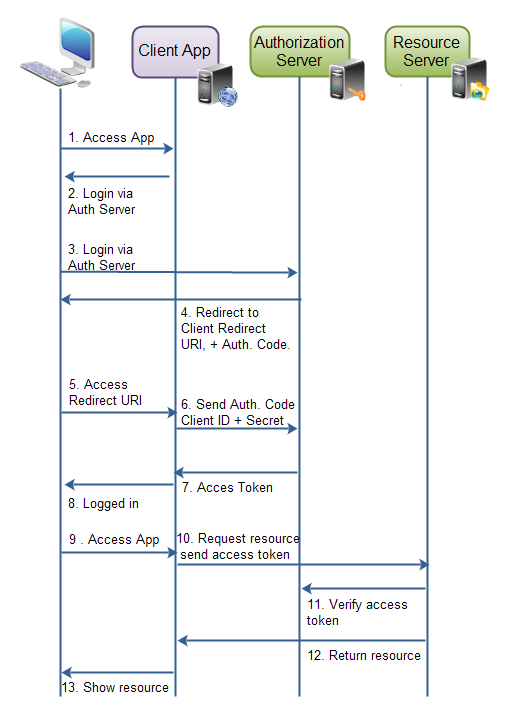
3) 登录之前，客户端应用会把用户重定向到认证服务器的登录页面，同时把client id, redirect url发送到认证服务器，这样认证服务器就知道是哪一个客户端应用在请求认证授权，以及认证授权后重定向到客户端应用的哪个地址。

4) 用户在认证服务器上进行登录，登录成功后，会提示用户是否要对客户端应用进行授权，用户选择同意后，会被重定向回客户端应用。

5) 当重定向回客户端应用时，使用的是客户端应用在认证服务器上注册时填写的重定向URL，同时认证服务器会发送一个代表此次认证过程的一个授权码。

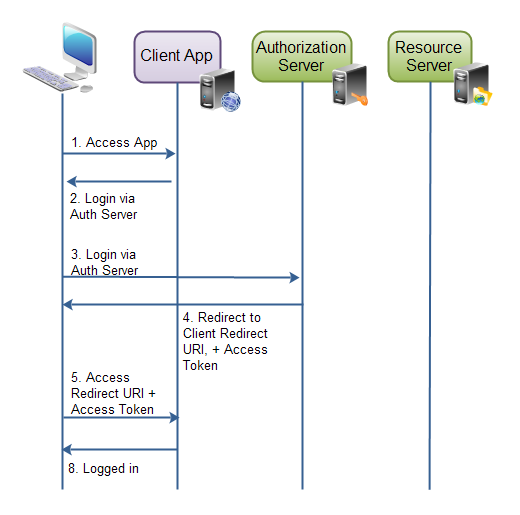
6） 当成功重定向到客户端应用后，客户端应用会在后台与认证服务器进行交互，将上个步骤中获取到的授权码，连同client id,client password发送给认证服务器。

7） 认证服务器对收到的数据进行校验，通过后发送access token给客户端应用。这时客户端应用就可以用接收到的access token去资源服务器访问相关资源。



**9、简化模式**

简化模式与授权码模式类似，区别仅在于当用户成功登录之后，重定向到客户端应用时，access token会直接返回给客户端应用。这意味着access token在客户端应用中是可见的。而Authorization Code（授权码模式），access token是在web服务器中的，对客户端来说不可见。这是这两种模式的最大区别。 并且，客户端应用只发送client id到认证服务器。如果连同client password一起发送的话，client password需要存储在客户端应用中，这会是一个安全隐患，很容易通过破解手段拿到存放在客户端应用程序中的client password。



**10、用户密码模式**

密码模式允许客户端应用直接使用用户的用户名和密码。例如用户可以直接在客户端应用中录入Twitter的用户名和密码。只有在充分信任客户端应用的情况下，才能使用密码模式。（因为用户名和密码是在客户端应用中录入的，因此客户端应用可以获取并保存用户的用户名和密码）。

**11、客户端模式**

客户端模式用于访问跟用户无关的资源，因此不需要用户授权。

**12、OAuth2.0节点**

OAuth 2.0定义了节点集合。一个节点一般是web服务器上的一个URL。具体包括：

·认证节点：用户进行登录操作的地址。

·Token节点：认证服务器提供的，让客户端应用获取access token的地址。

·重定向节点：用户成功登录后，会被重定向到此地址。

认证节点和Token节点在认证服务器上，重定向节点在客户端应用上。

**13、OAuth2.0请求和响应**

当客户端应用请求access token时，会发送http请求到认证服务器。不同的认证授权类型会有不同的请求和响应内容。

**授权码模式**有2个请求和2个响应：

·认证请求＋响应

·access token请求＋响应。

1）授权请求发送到认证服务器，然后会获取到一个授权码。

|  |  |  |
| --- | --- | --- |
| response\_type | 必选项 | 授权类型,固定值为 "code" |
| client\_id | 必选项 | 客户端应用在认证服务器注册时生成的client id |
| redirect\_uri | 可选项 | 客户端应用在认证服务器注册时填写的重定向URL地址 |
| scope | 可选项 | 请求的权限范围 |
| state | 可选项(建议提供) | 客户端应用的请求URL中的参数，可以是任意值，认证服务器会原封不动地返回这个值 |

2）授权响应含有授权码，这个授权码在后续获取access token时需要提供。

|  |  |  |
| --- | --- | --- |
| code | 必选项 | 认证服务器返回的授权码, 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，否则会被认证服务器拒绝。该码与客户端ID和重定向URI，是一一对应关系。 |
| state | 必选项 | 如果客户端应用的请求中有这个参数，既为这个参数的值. |

3) 客户端应用获取到授权码后，可以用此授权码去获取access token。请求参数如下：

|  |  |  |
| --- | --- | --- |
| grant\_type | 必选项 | 授权模式,固定值为 "authorization\_code" |
| client\_id | 必选项 | 客户端Id |
| redirect\_uri | 必选项 | 如果授权请求中包含，则该参数值必须保持一致。 |
| code | 必选项 | 从认证服务器中获得的授权码 |
| client\_secret | 必选项 | 客户端密码 |

4) access token的响应内容是json格式的：

{ "access\_token" : "...",

"token\_type" : "...",

"expires\_in" : "...",

"refresh\_token" : "...",

}

access\_token : 访问令牌,

token\_type : 令牌类型，可以是bearer类型或mac类型，一般是bearer。

expires\_in : 以秒为单位的令牌失效时间,

refresh\_token : 当访问令牌失效时，可以用更新令牌获取一个新的访问令牌

此外，HTTP头信息中明确指定不得缓存。

**简化模式**只有1个请求和1个响应：

1）授权请求发送到认证服务器，然后会获取到一个访问令牌。

|  |  |  |
| --- | --- | --- |
| response\_type | 必选项 | 授权类型,固定值为 "token" |
| client\_id | 必选项 | 客户端应用在认证服务器注册时生成的client id |
| redirect\_uri | 可选项 | 客户端应用在认证服务器注册时填写的重定向URL地址 |
| scope | 可选项 | 请求的权限范围 |
| state | 可选项(建议提供) | 客户端应用的请求URL中的参数，可以是任意值，认证服务器会原封不动地返回这个值 |

2）授权响应包含以下参数，注意该响应的格式不是json的。

|  |  |  |
| --- | --- | --- |
| access\_token | 必选项 | 由认证服务器产生的访问令牌 |
| token\_type | 必选项 | 令牌类型，该值大小写不敏感 |
| expires\_in | 可选项(建议提供) | 访问令牌的过期时间，单位为秒 |
| scope | 可选项 | 访问令牌的权限范围 |
| state | 必选项 | 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。 |

**用户密码模式**只有1个请求和1个响应：

1）授权请求发送到认证服务器，然后会获取到一个访问令牌。

|  |  |  |
| --- | --- | --- |
| grant\_type | 必选项 | 授权类型,固定值为 "password" |
| username | 必选项 | UTF-8编码的用户名 |
| password | 必选项 | UTF-8编码的密码 |
| scope | 可选项 | 请求的权限范围 |

2) access token的响应内容是json格式的：

{ "access\_token" : "...",

"token\_type" : "...",

"expires\_in" : "...",

"refresh\_token" : "...",

}

access\_token : 访问令牌,

token\_type : 令牌类型，

expires\_in : 以秒为单位的令牌失效时间,

refresh\_token : 当访问令牌失效时，可以用更新令牌获取一个新的访问令牌

**客户端模式**只有1个请求和1个响应：

1）授权请求发送到认证服务器，然后会获取到一个访问令牌。

|  |  |  |
| --- | --- | --- |
| grant\_type | 必选项 | 授权类型,固定值为 "client\_credentials" |
| scope | 可选项 | 请求的权限范围 |

2) access token的响应内容是json格式的：

{ "access\_token" : "...",

"token\_type" : "...",

"expires\_in" : "...",

}

access\_token : 访问令牌,

token\_type : 令牌类型，

expires\_in : 以秒为单位的令牌失效时间,

此种授权类型没有refresh\_token

**14、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：

|  |  |  |
| --- | --- | --- |
| grant\_type | 必选项 | 授权类型,固定值为"refresh\_token" |
| refresh\_token | 必选项 | 早前收到的更新令牌 |
| scope | 可选项 | 请求的权限范围，不可以超出上一次申请的范围 |