深入理解OAuth2.0及其应用

今天分享的部分内容取自

http://www.rfcreader.com/#rfc6749\_line163，以及阮一峰博客<http://www.ruanyifeng.com/blog/2014/05/oauth_2_0.html>

今天分享的技术主题是OAuth2.0，为什么选择这个主题，因为OAuth2.0几乎已经统治了整个世界。

如果你不清楚什么是OAuth2.0，那么看看你手机里的app，打开这些app的第一个界面很可能就是一个OAuth2.0的入口，比如使用微博登录，或者使用微信账号登录。

当然除了用户免注册登录这样的应用外，各种支付场景也是OAuth2.0大规模应用的场景，应用内支付会跳转到支付宝或者微信，用户授权支付成功再跳转回购物的app。

帮助大家梳理一下，我们来看看在我们身边OAuth2.0的那些大平台。

先看看国外

• App.net

• Azure

• Box.com

• Dropbox

• Facebook

• Foursquare

• Github

• Google

• Instagram

• Keycloak

• LinkedIn

• Mailchimp

• Salesforce

• Shopify

• Soundcloud

• Stripe

• Twitter

如果是有参与海外项目开发的开发者必然已经是非常熟悉他们了，那国内呢？

互联网拥有最多用户的四大应用 微信 qq 微博 还有支付宝。

现在我们来回答一个问题，作为开发者，我们需要详细了解这套规则吗？

答案是肯定的。

如果作为框架设计者，需要了解所有细节，建议大家去看一看详细的规则，<http://www.rfcreader.com/#rfc6749_line163>。

不过作为使用者，我们只要能理解即可。

那么如何来理解这数十万字的文档内容呢？

我们应该从几个关键的名词开始

Third-party application：第三方应用程序

HTTP service：HTTP服务提供商

Resource Owner：资源所有者

User Agent：用户代理

Authorization server：认证服务器

Resource server：资源服务器

OAuth在"客户端"与"服务提供商"之间，设置了一个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。

运行流程如下

（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。

好了，流程很清楚，但是为什么要这样做呢？

这个问题问得很好

我们可以想象一个场景，比如新浪微博，如果我做了一个app名叫《龙哥IT俱乐部》需要使用微博登录并获取微博相册中的头像 、昵称、以及粉丝关注信息，以及使登录用户自动关注我的官方微博。

如果没有OAuth机制，那么用户想要使用微博账号登录应用，就只能把用户名与密码提交给我的app，让我拿着用户名密码去向新浪验证这账号密码对不对，想一想，这明显不合理，因为用户会觉得很不安全。

而且如果把用户名和密码告诉第三方，那么第三方就等于拥有用户的所有资源的权限，不仅限于获取头像和粉丝数据，这也是很危险的行为，要知道拿到用户账号密码后，我甚至可以把用户的密码修改掉。

OAuth2.0让一切变的合理，我们来结合案例看看整个过程

那么在这个案例中，Third-party application就是《龙哥IT俱乐部》这个app，我的app作为第三方应用要从Resource server中获取资源，毫无疑问，新浪微博的服务器就是Resource server，而我需要获取的资源，比如用户的相册头像，用户的昵称，用户的关系网络等资源就保存在新浪的Resource server上，但是这些资源本身是属于用户的，用户就是Resource Owner（资源所有者），如果有一个女孩叫小芳，他使用自己的微博账号登录我的app，那么小芳就是资源所有者。

小芳通过《龙哥IT俱乐部》打开客户端以后，客户端要求用户给予授权，这时手机将《龙哥IT俱乐部》跳转到《新浪微博》的授权页面。

用户在新浪微博的授权页面中输入用户名与密码发送请求到微博的Authorization server进行身份验证授权。

微博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，并跳转回到《龙哥IT俱乐部》客户端

客户端使用令牌，向微博资源服务器申请获取小芳的头像，关系网络等资源。

微博的资源服务器确认令牌后向客户端开放资源。

整个过程既方便又安全，第三方应用没有用户账号信息，只有一个授权的令牌，用户可以只分享一部分权利给第三方应用，而且还可以随时取消这个授权，一旦授权取消，令牌随即变的无效。

这就好像是古代皇宫大内，如果皇帝要让某官员办事，他有两种做法，一种是对所有人宣布，他就代表我，他让大家干嘛大家就配合。

毫无疑问，这样做问题很大，因为一旦大家认可这个人以后，他可以去做很多不该做的事情，发展自己的势力，直到羽翼丰满后造反起事。

所以皇帝的做法是为办事的人发一个拥有特定权利的令牌，这样见到令牌的人就知道这个人有哪些权利，等事情办完后再将令牌收回，这样就不用担心权利失控了。

理解了这种授权的机制，那么我们再来看一下客户端的授权模式

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）

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：

1. 用户访问客户端，后者将前者导向认证服务器。
2. 用户选择是否给予客户端授权。
3. 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。
4. 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。
5. 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=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。

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=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。

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。

再来看看简化模式

（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。

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。

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：

granttype：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。

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"

}

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"refreshtoken"，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=tGzv3JOkF0XG5Qx2TlKWIA