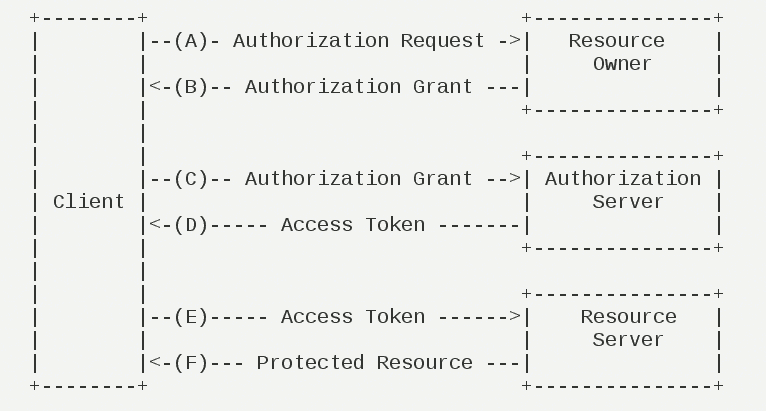
###### OAuth 2.0的运行流程如下图（摘自RFC 6749）：



图一

###### 名词解释：

（1） **Third-party application**：第三方应用程序，本文中又称"客户端"（client），图一左边的Client。

（2）**HTTP service**：HTTP服务提供商，本文中简称"服务提供商"。比如google。

（3）**Resource Owner**：资源所有者，本文中又称"用户"（user）。比如在google存储了图片的用户小明。

（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。比如google的图片存储服务器

###### 如何工作

让我们描述下这样一个场景，“某App 希望获得 Tom在 google上相关的数据”。

OAuth 2 在整个流程中有四种角色：

* 资源拥有者(Resource Owner) - 这里是 Tom
* 资源服务器(Resource Server) - 这里是google图片存储服务器
* 授权服务器(Authorization Server) - 这里当然还是google，因为google有相关数据
* 客户端(Client) - 这里是某App

参看图一，当 Tom 试图登录 google，打开了某App， 某App 将他重定向到 google的授权服务器，当 Tom 登录成功，并且许可自己的Email和个人信息被 某App 获取。这两个资源被定义成一个 Scope（权限范围） ，一旦准许， 某App 的开发者就可以申请访问权限范围中定义的这两个资源（Tom的Email和个人信息）。Tom 允许了权限请求，再次通过重定向返回 某App ，重定向返回时携带了一个 Access Token（访问令牌） ，接下来 某App 就可以通过这个 Access Token 从 google直接获取相关的授权资源（也就是Email和个人信息），而无需重新做 Tom 相关的鉴权。而且每当 Tom 登录了 某App ，都可以通过之前获得的 Access Token ，直接获取相关授权资源。

###### 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）

如上例，“某App 希望获得 Tom在 google上相关的数据”，Tom可以采用密码模式对某App进行授权。

###### token存储

引入微服务时，需要找一个 Token 存储的方式，来保证授权服务器可以被水平扩展。所有访问微服务资源的请求都在Http Header中携带 Token ，被访问的服务接下来再去请求授权服务器验证 Token 的有效性，目前这种方式，我们需要两次或者更多次的请求，但这是为了安全性也没什么其他办法。但扩展 Token 存储会很大影响我们系统的可扩展性，因此需要我们引入 JWT。

简短来说，响应一个用户请求时，将用户信息和授权范围序列化后放入一个JSON字符串，然后使用Base64进行编码，最终用 Token 进行签名，得到一个 JSON Web Token ，我们可以像使用 Access Token 一样的直接使用它。当这些JWT在授权服务器使用RSA私钥进行签名，假设其他所有的资源服务器都将持有一个RSA公钥。当资源服务器接收到这个在Http Header中存有 Token 的请求，资源服务器就可以拿到这个 Token ，并验证它是否使用正确的私钥签名（是否经过授权服务器签名）。验证通过，反序列化后就拿到OAuth 2的验证信息，使用JWT可以简单的传输 Token ，并保证 Token 很难被伪造。 Access Token 字符串中包含用户信息和权限范围，我们所需的全部信息都有了，所以不需要维护 Token 存储，资源服务器也不必要求 Token 检查。