### 认证与授权

参考：

<https://www.cnblogs.com/jinhengyu/p/10257792.html>

认证就是认证用户的身份，像飞机票用你的身份证验证你是不是你自己

授权就是系统成功验证你的身份后。最终会授予你访问资源的权限，

简单的说就是你认证成功后系统授予你哪些权限。

我们知道，HTTP是无状态的，所以，当我们需要获得用户是否在登录的状态时，我们需要检查用户的登录状态，一般来说，用户的登录成功后，服务器会发一个登录凭证（又被叫作Token），就像你去访问某个公司，在前台被认证过合法后，这个公司的前台会给你的一个访客卡一样，之后，你在这个公司内去到哪都用这个访客卡来开门，而不再校验你是哪一个人。在计算机的世界里，这个登录凭证的相关数据会放在两种地方，一个地方在用户端，以Cookie的方式（一般不会放在浏览器的Local Storage，因为这很容易出现登录凭证被XSS攻击），另一个地方是放在服务器端，又叫Session的方式（SessonID存于Cookie

Auth(认证)

认证的方式有很多种，常见的认证方式有：

### 常见的认证方式

#### http basic auth

这是传统的认证方式使用username和password来进行登录。它实现的原理是

1. 把username和password做成username:password这样即用表单分隔。得到一个字符串s
2. 使用base64编码，将字符串进行base64编码
3. 将base64编码后的字符串放到http的头[Authorization](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization) 字段中，形成 Authorization: Basic aGFvZW86Y29vbHNoZWxsCg，然后发送到服务端。
4. 服务端在http头中没有看到认证字段返回错误否则返回200

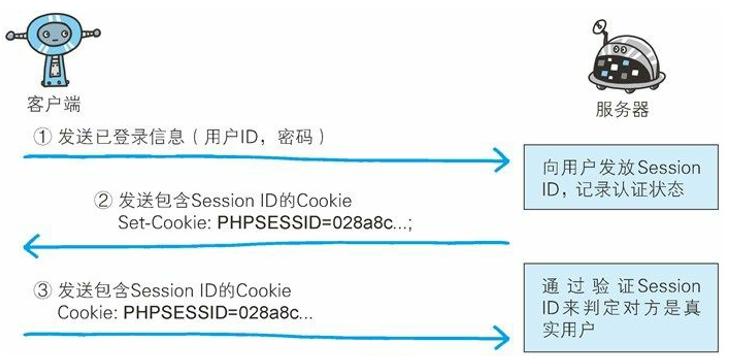
采用base64编码，放在请求中，很容易被破解不是很安全

#### HTTP 表单认证

这种认证方式也是我们常用的认证方式。

基于表单认证一般会使用Cookie 来管理 Session（会话）

基于表单认证本身是通过服务器端的 Web 应用，将客户端发送过来的用户 ID 和密码进行校验。但鉴于 HTTP 是无状态协议，之前已认证成功的用户状态无法通过协议层面保存下来。即无法实现状态管理，因此即使当该用户下一次继续访问，也无法区分他与其他的用户。于是我们会使用 Cookie 来管理 Session，以弥补 HTTP 协议中不存在的状态管理功能。



还有像

JWT – JSON Web Tokens认证和oauth2.0认证。

#### JWT

参考：<https://www.jianshu.com/p/576dbf44b2ae>

##### 什么是JWT

Json web token 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准（[(RFC 7519](https://link.jianshu.com?t=https:/tools.ietf.org/html/rfc7519)).该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。JWT的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

##### 基于token机制的认证流程

用户使用用户名密码来请求服务器

服务器进行验证用户的信息

服务器通过验证发送给用户一个token

客户端存储token，并在每次请求时附送上这个token值

服务端验证token值，并返回数据

##### jwt的结构

###### 头部(header)

承载两个部分信息，声明类型，声明加密算法

{

'typ': 'JWT',

'alg': 'HS256'

}

将头部进行base64加密构成第一部分

###### 载荷(payload)

存放有效信息包括三部分

* 标准中注册的声明
* 公共的声明
* 私有的声明

**标准中注册的声明** (建议但不强制使用) ：

* **iss**: jwt签发者
* **sub**: jwt所面向的用户
* **aud**: 接收jwt的一方
* **exp**: jwt的过期时间，这个过期时间必须要大于签发时间
* **nbf**: 定义在什么时间之前，该jwt都是不可用的.
* **iat**: jwt的签发时间
* **jti**: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

**公共的声明** ：  
公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息.但不建议添加敏感信息，因为该部分在客户端可解密.

**私有的声明** ：  
私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64是对称解密的，意味着该部分信息可以归类为明文信息。

定义一个payload:

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

然后将其进行base64加密，得到Jwt的第二部分。

###### 签证(signature)

wt的第三部分是一个签证信息，这个签证信息由三部分组成：

* header (base64后的)
* payload (base64后的)
* secret

这个部分需要base64加密后的header和base64加密后的payload使用.连接组成的字符串，然后通过header中声明的加密方式进行加盐secret组合加密，然后就构成了jwt的第三部分。

// javascript

var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);

var signature = HMACSHA256(encodedString, 'secret'); // TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

将这三部分用.连接成一个完整的字符串,构成了最终的jwt:

###### 如何应用

一般是在请求头里加入Authorization，并加上Bearer标注：

fetch('api/user/1', {

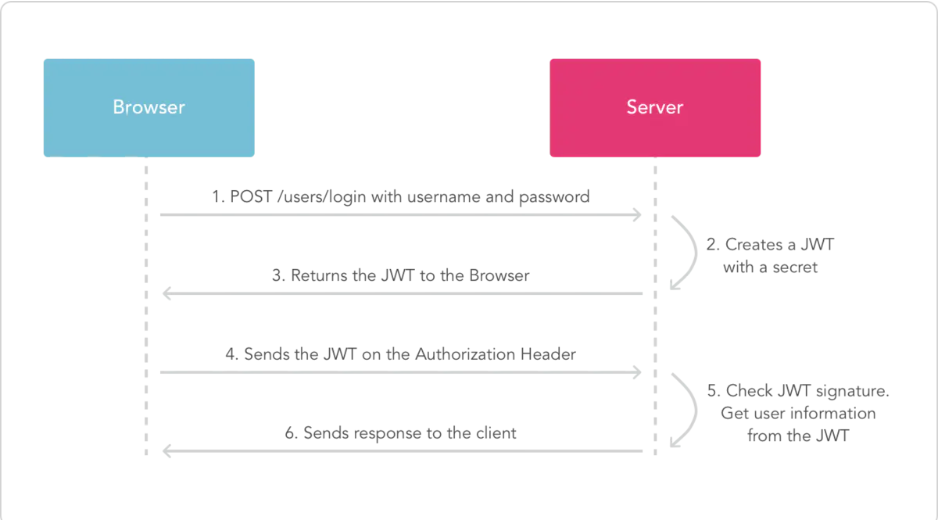
headers: {

'Authorization': 'Bearer ' + token

}

})

###### 认证流程



官网：<https://jwt.io/>

里面有各种语言的jwt,用composer安装

###### 用户登录成功，获取到token，再次访问登录接口会怎样？

#### OAuth2.0协议

oAuth2.0协议(开放授权)是一个开放标准，允许用户授权给第三方用户访问存储在服务器上的资源，并且用户不需要向第三方提供账号密码。

##### 应用场景：

原生app授权：app登录请求后台接口，为了安全认证，所有请求都带token信息，如果登录验证、请求后台数据。

前后端分离单页面应用（spa）：前后端分离框架，前端请求后台数据，需要进行oauth2安全认证，比如使用vue、react后者h5开发的app。

涉及角色：

* Resource Owner（资源拥有者：用户）
* Client （第三方接入平台：请求者）
* Resource Server （服务器资源：数据中心）
* Authorization Server （认证服务器）

##### 实现流程

（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。

##### 授权模式

参考：<https://www.jianshu.com/p/84a4b4a1e833>

<https://www.jianshu.com/p/84a4b4a1e833>

中的授权模式

 授权码模式（authorization code）

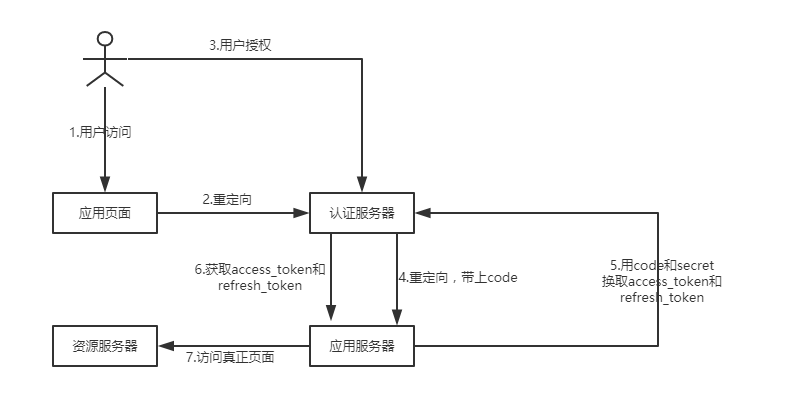
 简化模式（implicit） （适用于单页面应用）

 密码模式（resource owner password credentials）

 客户端模式（client credentials）

###### 授权码模式

流程：



用户访问客户端，重定向到认证服务器，

认证服务器向用户展示授权页面等待用户授权，

用户授权后认证服务器生成 code和client\_id并发送给服务器

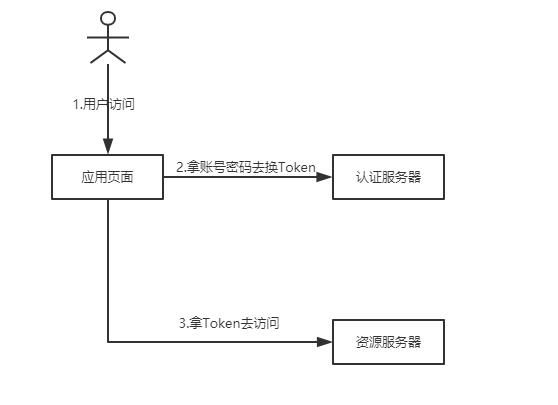
服务器拿到client\_id去后台查询相应的client\_secret

用code和client\_secret到认证服务器换区access\_token和refresh\_token

客户端发送访问令牌（access token）和更新令牌（refresh token）

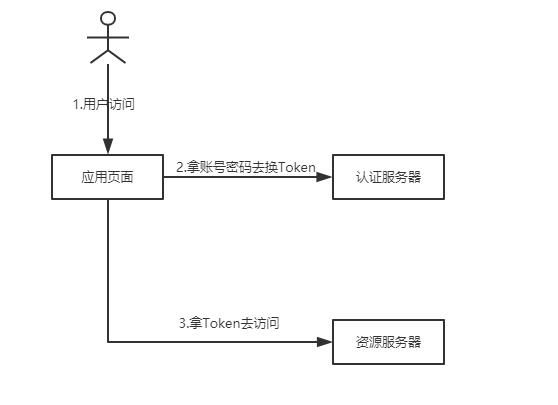
客户端携带令牌访问相关资源

###### 简化模式



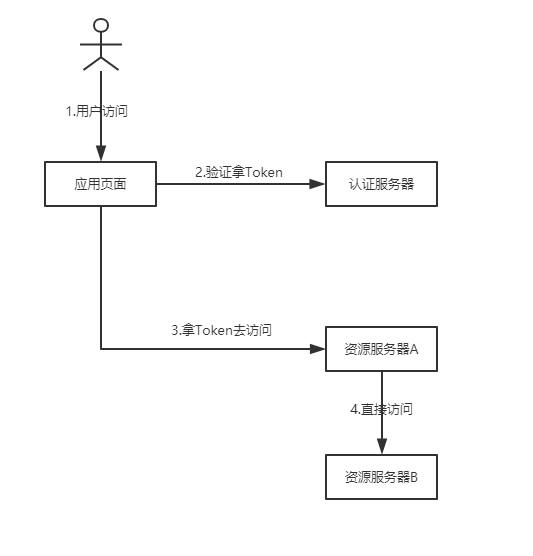
* 第一步：用户访问页面时，重定向到认证服务器。
* 第二步：认证服务器给用户一个认证页面，等待用户授权。
* 第三步：用户授权，认证服务器想应用页面返回Token
* 第四步：验证Token，访问真正的资源页面

###### 密码模式



* 第一步：用户访问用页面时，输入第三方认证所需要的信息(QQ/微信账号密码)
* 第二步：应用页面那种这个信息去认证服务器授权
* 第三步：认证服务器授权通过，拿到token，访问真正的资源页面

###### 客户端凭证模式



* 第一步：用户访问应用客户端
* 第二步：通过客户端定义的验证方法，拿到token，无需授权
* 第三步：访问资源服务器A
* 第四步：拿到一次token就可以畅通无阻的访问其他的资源页面