# http含义

http协议意为超文本传输协议，超文本意味着传输的不止是文本，还可以传输视频，图片，音频等内容

# 为什么需要http

http是web上进行任何数据传输的基础，我们所需要了解的是它是服务器和客户端之间的协议，通常由用户代理（即客户端，如浏览器，当然还有爬虫，只不过我们暂时不需要考虑这种情况）向服务器发出请求（后续会描述请求是个什么样子），然后服务器处理请求并返回响应（响应就是经过服务器处理后返回给客户端的数据），客户端拿到响应数据后就会在浏览器上面展示。这个数据传输的过程需要规定一种协议来使两端都能解析明白以及传输安全，所以就有了http协议。（不过第一版http协议根本就不安全而且效率低，后续才摒弃了明文传输以及实现了多路复用，有兴趣的同学可以去了解一下http的弊端以及新一点的QUIC协议）

这里稍微补充一下浏览器的工作方式

浏览器是一种客户端，也是一种用户代理，用户代理，顾名思义就是能够代表用户操作的东西，假如你在浏览器上面点了某个新闻，那么浏览器就会察觉到你的行为并且根据你的行为向服务器发出相应的请求，这个时候服务器处理请求返回响应，给浏览器提供这个新闻的数据，于是你就成功的看见了这个新闻。了解一点前端的同学我用下面的话来讲可能更专业，浏览器最初会发送一个请求来请求基础的html文档来填充页面基础内容，后续会请求到css来渲染页面内容，如果用户有后续操作那就执行js脚本来实现用户之间的交互来实时更新页面内容。ok，讲解完毕。

# http请求解析（稍微认真看一下，后续会用到）

GET / HTTP/1.1　（第一行就是说明你请求类型，请求路径，请求协议）
Host: developer.mozilla.org　（后续的就是header，描述一些要给服务端的数据）
Accept-Language: zh

# http是无状态的

无状态：每一次请求都是独立的，每个请求包含了这次请求所需的所有信息，发送请求不会导致状态变更。

有状态：服务器会记录之前请求导致的状态变更，并且根据这些状态来处理后续请求。

使用有状态协议可以实现更复杂的逻辑，也可以根据用户的请求来制定个性化内容。使用无状态协议可以占用更小服务器开销，可以提高容错性。

# 前后端交互

不懂得运维的前端不是一个好后端，作为后端，做业务的时候需要与前端进行密切配合。通常由后端编写接口文档来方便前后端交互，至于接口文档怎么写，这边推荐上apifox来自动生成，比较方便。那么前后端交互具体怎么实现呢

### 发送客户端请求

首先由客户端，一般是web浏览器，发送客户端请求，一般长这样：

POST /contact_form.php HTTP/1.1
Host: developer.mozilla.org
Content-Length: 64
Content-Type: application/x-www-form-urlencoded

name=Joe%20User&request=Send%20me%20one%20of%20your%20catalogue

那么这一长串字母是什么意思，首先第一排叫请求行，有请求方法类型，目标url以及请求参数，请求参数可以有文档路径和使用的http协议版本，接下来的没一行都叫请求头，表示请求所携带的表头，为服务器提供所需的部分信息，具体有哪些类型的信息呢，为了防止抽象难以理解，列举了一些，看看下面

`Host`: 指定目标服务器的主机名。

- 例：`Host: www.example.com`

`User-Agent`: 标识客户端信息（如浏览器类型、版本）。

- 例：`User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)`

`Accept`: 指定客户端可接受的内容类型。

- 例：`Accept: text/html, application/json`

`Authorization`: 用于提供身份验证信息。

- 例：`Authorization: Bearer <token>`（这个后续会着重讲）

最后一行叫请求体，是可选数据块，包含更多数据，通常在post方法里面使用

还有一点要补充的是，param参数和post里面传来的其他参数是保存在请求的哪一部分里面呢，一般来说，在get方法里面param参数保存在请求行里面的url里面，在post方法里面也可以保存在请求体里面，post的其他参数如表单一般都在请求体里面

## 服务端响应

上例子

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 55743
Connection: keep-alive
Cache-Control: s-maxage=300, public, max-age=0
Content-Language: en-US
Date: Thu, 06 Dec 2018 17:37:18 GMT
ETag: "2e77ad1dc6ab0b53a2996dfd4653c1c3"
Server: meinheld/0.6.1
Strict-Transport-Security: max-age=63072000
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Vary: Accept-Encoding,Cookie
Age: 7

```
<!DOCTYPE html>… (包含一个网站自定义页面，帮助用户找到丢失的资源)
```

跟客户端请求结构差不多啊，无许多言。

# cookie session token

铺垫完了，终于到了主体部分了。。。

众所周知，没有匹配的钥匙就打不开对应的门，在互联网里面也是这样，需要验证身份或者权限来确定你是否有资格来访问这些资源，举个例子，教务在线没有登陆就只是游客身份，只能访问有限资源。而下面要讲的就可以是我们要用的钥匙

## cookie

我来简单把cookie整个大概喵两句，首先由客户端发送请求，服务端这边如果选择在响应头里面设置cookie，就像这样

HTTP/1.1 200 OK
Set-Cookie: session_id=abc123; Path=/; HttpOnly

客户端收到这种带有cookie的响应以后就会在设备端生成一个小型数据文件，用于存储cookie携带信息，之后每次客户端请求都会自动带上这个cookie，用于保存会话状态，其中就可以有登陆状态，用来识别是否是合法用户。大概长这样

GET /profile HTTP/1.1
Host: www.example.com
Cookie: session_id=abc123

（这段是拓展，时间充足就讲）：

所以为什么要有cookie，因为http是无状态的，有了cookie就可以实现http的有状态，保持需要记录的会话记录，比如登陆状态和浏览记录。根据这个会话记录就可以实现用户鉴权以及个性化推荐，购物车状态跟踪等操作。cookie分成第一方cookie和第三方cookie第一方就是在同一网站下的会话记录，第三方就是其他网站的会话记录，很简单理解的就是，第一方就是你在段视频平台下的个性化推荐，第三方就是你刚看在购物网站上看了点什么商品，可能有其他网站会推送它的广告。还可以分为会话cookie和持久性cookie，区别就是1保存时间，会话cookie保存到你关闭浏览器那一刻，持久性即使关闭了也还在，时间自己设置。使用cookie就是为了实现有状态，从而实现鉴权，个性化推荐，广告等。cookie的危害就是容易暴露隐私啊。。。所以有的网站会问你是否接受所有cookie，看情况自己选择吧。还可能会引发安全问题。这个就需要session和token了啊。。。

## session

简单点讲吧，session就是为了弥补cookie的不足而出现的，众所周知，cookie在http请求头里面，很容易遭到攻击，而且每次请求都得带上一长串cookie，有点没必要了。所以干脆在服务端建立关于用户会话的记录，可以理解为建立一个用户行为的表，服务端只在cookie里面传一个sessonid，相当于用户行为表的id，然后客户端每次只用在cookie里面传一个sessionid给服务端，服务端就根据id搜索该用户所有的会话记录，这样就弥补了cookie的两个不足了。说的有点抽象，cookie和session确实比较难以分清，回去多看看。

# jwt

# 什么是JWT认证?

## JWT 是基于令牌的身份验证

首先我简单解释一下什么是JWT认证。

当您想到登录时，您是否有这样的印象：应用程序维护用户的登录状态，以便它可以显示为该用户定制的页面?
这是**基于会话的身份验证**。

今天我们要学习的JWT就是**基于token的认证**。使用基于令牌的身份验证，应用程序**不会让用户保持登录状态**。
那么如何在不保持用户登录状态的情况下识别用户呢?
用户向您的应用程序发送 HTTP 请求，例如当他们想要登录或获取信息时。届时，预先发出的**JWT（=令牌）将包含在每个请求中并发送**。每次应用程序收到请求时，它都会验证 JWT 是否有效，并可以确认该请求是否来自注册用户。

## BASE64URL编码

URL安全意味着URL被正确识别。例如，`?`、`=`、`&`等在 URL 中具有特殊含义。如果 JSON 数据包含此类符号，则将无法正确识别 URL。为了防止这种情况发生，JWT 将 JSON 数据编码为 BASE64URL。

例如，假设我们有以下 JSON。

```
{"iss":"joe",
 "exp":1300819380,
 "http://example.com/is_root":true}
```

当这个 JSON 被 BASE64URL 编码时，它变成如下。

```
eyJpc3MiOiJqb2UiLCJleHAiOjEzMDA4MTkzODAsImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

顺便说一句，BASE64 编码和 BASE64URL 编码是不同的东西。 BASE64 编码在编码时使用两种类型的字符（`+`和`/`），但这些符号在 URL 中具有特殊含义。因此，在 BASE64URL 编码中，请使用`-`和`_`进行编码。

现在让我们看看如何创建 JWT。

首先，我们看一下完成的 JWT。
JWT 由三部分组成：标头、负载和签名。每个由 .（点）分隔。
该字符串是 JWT 本身。也称为令牌。
该 JWT 将由应用程序发出并传递给用户，用户将在每个请求中包含该 JWT。

现在让我们创建 JWT。
在这里，我们将按顺序创建标头+有效负载作为前半部分，并将签名作为后半部分。

# 步骤1

首先，让我们创建 JWT 的前半部分（标头 + 有效负载）。

顾名思义，JWT（JSON Web Token）是基于 JSON 创建的。
让我们准备标头和负载 JSON。

## 标头

header 是描述 JWT 的部分。
`alg`是必填字段，指示在后续步骤中添加签名时要使用的算法。
`typ`是令牌的类型，假设是`JWT` 。

标头

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

## 有效负载

有效负载是 JWT 的信息部分。
在下面的示例中， `sub`是唯一标识用户的标识符， `exp`是 JWT 的到期日期。您可以输入任何其他项目。

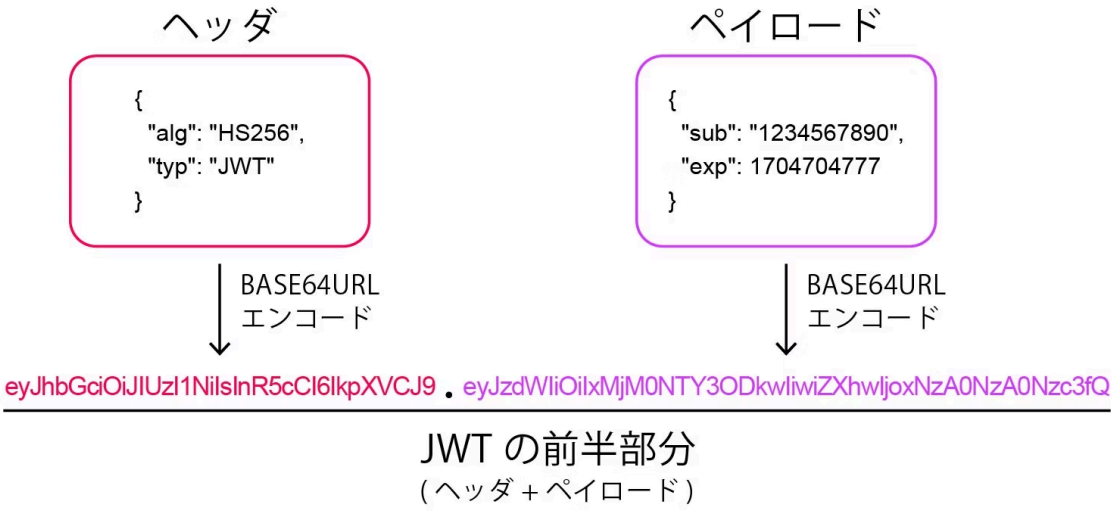
**请注意，您不应包含任何敏感信息，例如密码**。
有效负载将在下一步中进行编码，但它可以解码回原始 JSON，因此如果有人窃取 JWT，很容易看到内容。

有效负载

```
{
  "sub": "1234567890",
  "exp": 1704704777
}
```

# 步骤2

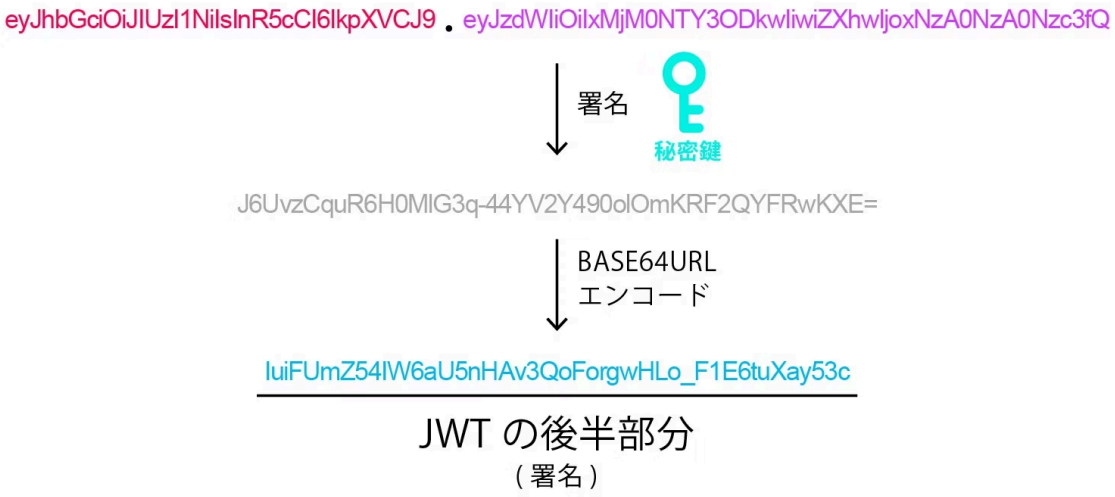
BASE64URL 对 STEP1 中准备的标头和有效负载 JSON 进行编码。用 .（点）连接两者。




现在我们有了 JWT 的前半部分。

## 步骤3

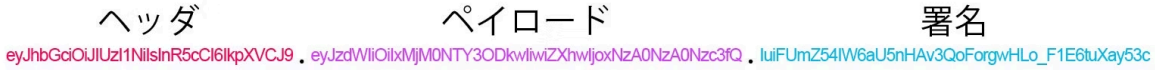
从这里，我们将创建 JWT 的后半部分（签名）。
使用标头`alg`中指定的算法和私钥对第 2 步中创建的 JWT（标头 + 有效负载）的前半部分进行签名。
进一步对该签名进行 BASE64URL 编码。



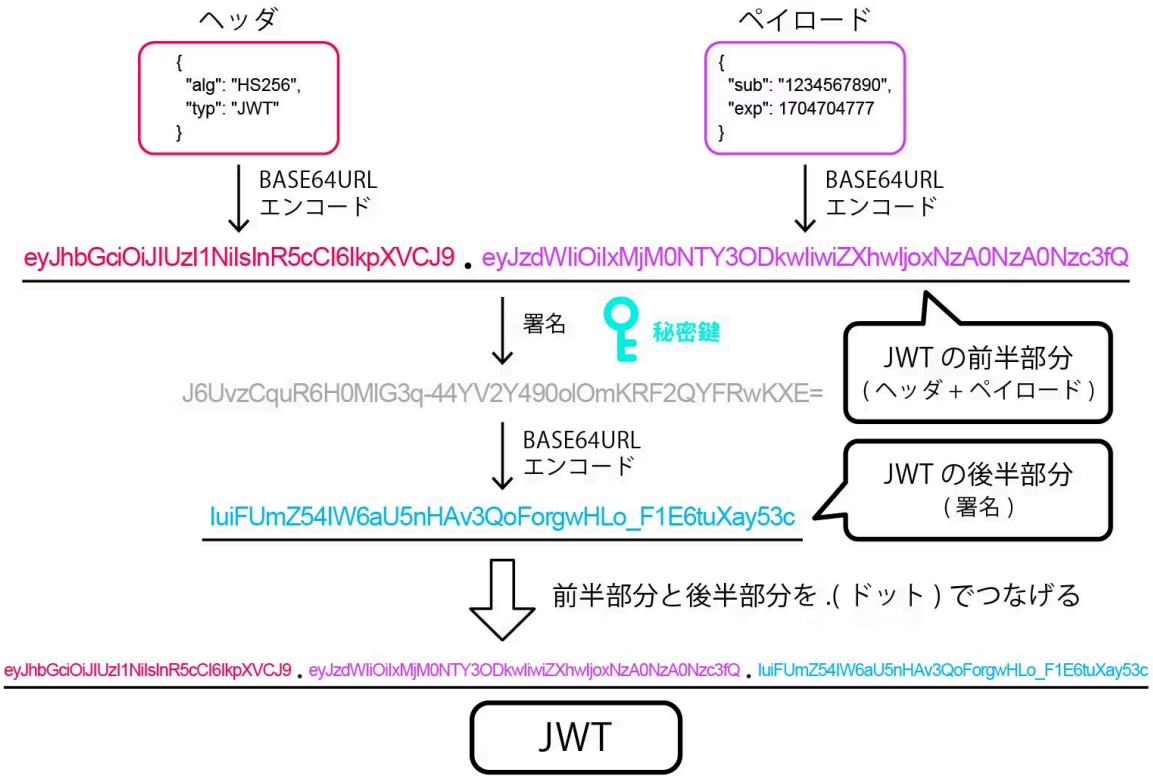

JWT 的后半部分现已完成。

## 步骤4

接下来，用 .（点）连接前半部分（标头 + 有效负载）和后半部分（签名）。
JWT 现已完成！




总结起来就是下图这样的。




# 4.JWT验证

我们如何验证这个 JWT 是否有效?
在此之前，我们先来解释一下JWT所扮演的角色。

## JWT是一种检测篡改的机制

**JWT的目的不是防止内容被查看或篡改，而是检测篡改行为。**

如果不知道私钥，就无法看到 JWT 的签名部分。但是，签名部分包含的信息与标头+有效负载相同，而标头+有效负载部分只是经过编码，因此可以轻松解码回原始 JSON。这意味着如果有人窃取此 JWT，他们可以轻松查看其内容甚至重写它。
换句话说，JWT**并不是一种防止内容被看到的机制**。
当然，您必须防止您的 JWT 被他人窃取，但为了避免即使被盗也造成任何损害，JWT 不得包含密码或其他信息。

万一发生某些内容被篡改的情况，**JWT 的作用就是检测这种篡改**。
**验证结果认为是真实的JWT，没有被篡改**。

## 如何检测篡改

那么如何验证JWT没有被篡改呢?
这里，我们将考虑使用“公共密钥方法”进行JWT签名的情况。使用同一个私钥来创建和验证签名。

接收 JWT 的应用程序使用用于创建签名的相同私钥来验证 JWT 的前半部分（标头 + 有效负载）和 JWT 的后半部分（签名）。
后半部分（签名）是前半部分（头部+负载）的签名，因此**可以使用创建签名时使用的私钥来验证两者是否匹配**。如果它们匹配，则它是有效的 JWT。

**现在，让我们考虑一下如果 JWT 被篡改会发生什么。**
例如，假设 JWT 的有效负载部分已被某人篡改。
应用程序使用私钥验证接收到的 JWT 的前半部分（标头 + 有效负载）和 JWT 的后半部分（签名）。然后，签名和被篡改的有效负载部分不匹配，因此可以检测到篡改。
另外，假设有人窃取 JWT，篡改有效负载，然后使用任意密钥添加签名（因为私钥未知）。同样，当您使用正确的私钥验证 JWT 时，您可以检测到签名是使用无效的私钥创建的。

# 第三方登陆

实际上，大多数网站提供的第三方登录都遵循OAuth协议，虽然大多数网站的细节处理都是不一致的，甚至会基于OAuth协议进行扩展，但大体上其流程是一定的。今天，我们就来看看基于OAuth2的第三方登陆功能是这样一个流程。

# OAuth2.0的基本流程

OAuth协议目前已经升级到了2.0，大部分的网站也是支持OAuth2.0的，因此让我们先看看OAuth2。


上图中所涉及到的对象分别为：

- Client 第三方应用，我们的应用就是一个Client
- Resource Owner 资源所有者，即用户
- Authorization Server 授权服务器，即提供第三方登录服务的服务器，如Github
- Resource Server 拥有资源信息的服务器，通常和授权服务器属于同一应用

根据上图的信息，我们可以知道OAuth2的基本流程为：

1. 第三方应用请求用户授权。
2. 用户同意授权，并返回一个凭证（code）
3. 第三方应用通过第二步的凭证（code）向授权服务器请求授权
4. 授权服务器验证凭证（code）通过后，同意授权，并返回一个资源访问的凭证（Access Token）。

5. 第三方应用通过第四步的凭证（Access Token）向资源服务器请求相关资源。
6. 资源服务器验证凭证（Access Token）通过后，将第三方应用请求的资源返回。

# gorm

众所周知，要多看官方文档，所以直接上链接。

https://gorm.io/zh_CN/docs/index.html

上课会根据文档讲讲怎么使用它，预习的同学可以先看看文档

# 作业：

## level 1：

天冷加衣

## level 2：

给之前作业的项目加上cookie，session，jwt随意一种鉴权。如果之前项目没做就自己实现登陆注册还有ping的接口，加上鉴权。

## level 3:

了解jwt原理，搓一个简单jwt（学姐上课讲一下简单的流程，可以仿照以及拓展），session不用搓，找资料了解其大致实现原理就可以

# 参考文献

https://juejin.cn/post/7340481613144293395