

Gulimall

社交登陆&单点登陆



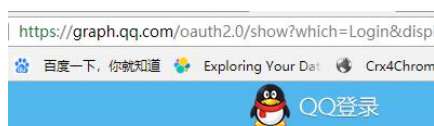
一、社交登陆



QQ、微博、github 等网站的用户量非常大，别的网站为了简化自我网站的登陆与注册逻辑，引入社交登陆功能：

步骤：

- 1)、用户点击 QQ 按钮
- 2)、引导跳转到 QQ 授权页



- 3)、用户主动点击授权，跳回之前网页。

1、OAuth2.0

- **OAuth:** OAuth（开放授权）是一个开放标准，允许用户授权第三方网站访问他们存储在另外的服务提供者上的信息，而不需要将用户名和密码提供给第三方网站或分享他们数据的所有内容。
- **OAuth2.0:** 对于用户相关的 OpenAPI（例如获取用户信息，动态同步，照片，日志，分享等），为了保护用户数据的安全和隐私，第三方网站访问用户数据前都需要显式的向用户征求授权。
- 官方版流程：

关于OAuth2.0协议的授权流程可以参考下面的流程图，其中Client指第三方应用，Resource Owner指用户，Authorization Server是我们的授权服务器，Resource Server是API服务器。



- (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。

2、微博登陆准备工作

1、进入微博开放平台

微博开放平台-新浪大数据应用平台

优惠: 我们将为您提供试用账号 品牌: 新浪舆情通 特色: 专属多维度的舆情监测 详情: 舆情
全网每日采集上亿+条数据,新浪微博授权全量数据,数据类型共涵盖全网大信息源.
www.yqt365.com 2019-03 ▼ V3 - 评价 广告

新浪微博开放平台-首页

微博开放平台为移动应用提供了便捷的合作模式,满足了多元化移动终端用户随时随地快速登录、分享信息的需求,助力实现移动Apps、健康设备、智能家居、车载等多类型终端的社...
<https://open.weibo.com/> ▼ - 百度快照 - 90%好评

微博登录

微博登录介绍 微博登录包括身份认证、用户关系以及内容传播。允许用户使

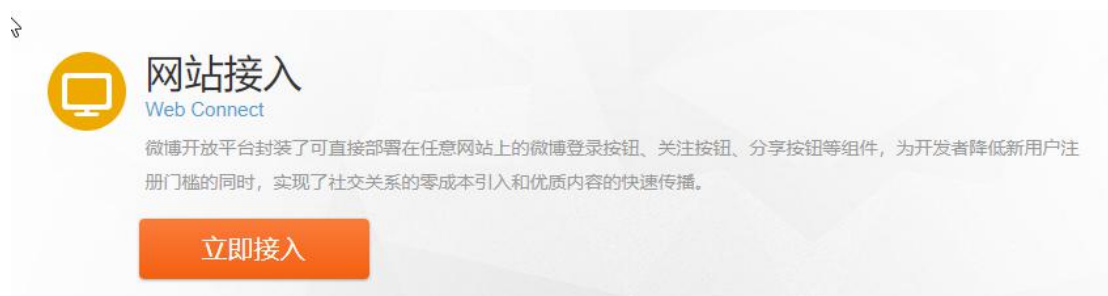
微博组件

新浪微博签名档,可以放置在你的博客、论坛,或是其它可以引用网上图片的位

2、登陆微博，进入微连接，选择网站接入



3、选择立即接入



4、创建自己的应用



5、我们可以在开发阶段进行测试了



记住自己的 app key 和 app secret 我们一会儿用

6、进入高级信息，填写授权回调页的地址



7、添加测试账号（选做）



8、进入文档，按照流程测试社交登陆



1、微博登录

基于OAuth2.0协议，使用微博 Open API 进行开发，即可用微博帐号登录你的网站，让你的网站降低新用户注册成本，快速获取大量用户。

[了解微博登录](#)

3、微博登陆测试

1、引导用户到如下地址

https://api.weibo.com/oauth2/authorize?client_id=YOUR_CLIENT_ID&response_type=code&redirect_uri=YOUR_REGISTERED_REDIRECT_URI



2、用户同意授权，页面跳转至 `xxx/?code=CODE`

<http://www.gulishop.com/success?code=fef987b3f9ad1169955840b467bfc661>

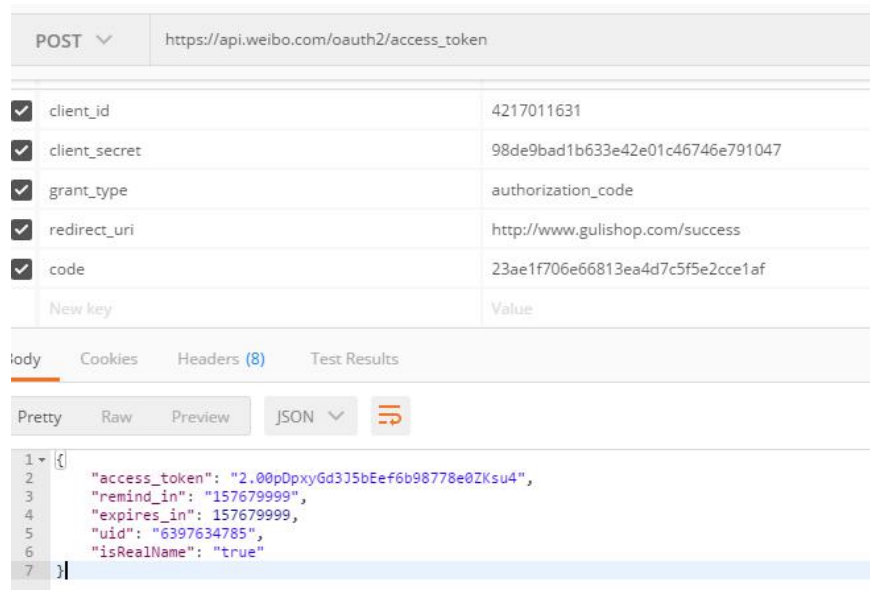
3、使用返回的 `code`，换取 `access token`

https://api.weibo.com/oauth2/access_token?client_id=YOUR_CLIENT_ID&client_secret=YOUR_CLIENT_SECRET&grant_type=authorization_code&redirect_uri=YOUR_REGISTERED_REDIRECT_URI&code=CODE

https://api.weibo.com/oauth2/access_token?client_id=4217011631&client_secret=98de9bad1b633e42e01c46746e791047&grant_type=authorization_code&redirect_uri=http://www.gulishop.com/success&code=fef987b3f9ad1169955840b467bfc661

注意，上面这个是 post 请求

```
{
  "access_token": "2.00pDpxyGd3J5bEef6b98778e0ZKsu4",
  "remind_in": "157679999",
  "expires_in": 157679999,
  "uid": "6397634785",
  "isRealName": "true"
}
```



4、使用 AccessToken 调用开发 API 获取用户信息



至此微博登陆调试完成。

Oauth2.0: 授权通过后, 使用 code 换取 access_token, 然后去访问任何开放 API

- 1)、code 用后即毁
- 2)、access_token 在几天内是一样的
- 3)、uid 永久固定

二、SSO (单点登陆)

Single Sign On 一处登陆、处处可用

0、前置概念:

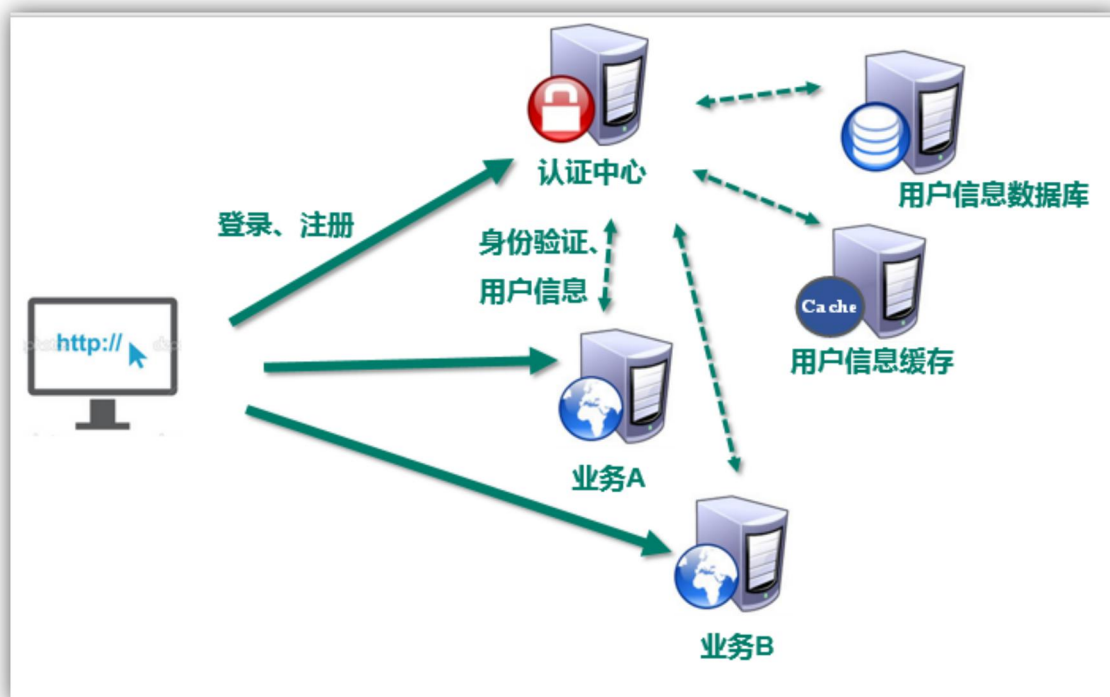
1)、单点登录业务介绍

早期单一服务器, 用户认证。



缺点: 单点性能压力, 无法扩展

分布式, SSO(single sign on)模式



解决：

用户身份信息独立管理，更好的分布式管理。
可以自己扩展安全策略
跨域不是问题

缺点：

认证服务器访问压力较大。

2）、几个基本概念

2.1 什么是跨域 Web SSO。

域名通过“.”号切分后，从右往左看，不包含“.”的是顶级域名，包含一个“.”的是一级域名，包含两个“.”的是二级域名，以此类推。

例如对网址 <http://www.cnblogs.com/baibaomen>，域名部分是 www.cnblogs.com。
用“.”拆分后从右往左看：

`cookie.setDomain(".cnblogs.com");` //最多设置到本域的一级域名这里

`cookie.setDomain(".baidu.com");` //最多设置到本域的一级域名这里

“com”不包含“.”，是顶级域名；“cnblogs.com”包含一个“.”，是一级域名；

www.cnblogs.com 包含两个“.”，是二级域名。

blog.cnblogs.com

news.cnblogs.com

跨域 Web SSO 指的是针对 Web 站点，各级域名不同都能处理的单点登录方案。

2.2 浏览器读写 cookie 的安全性限制：一级或顶级域名不同的网站，无法读到彼此写的 cookie。

所以 baidu.com 无法读到 cnblogs.com 写的 cookie。

一级域名相同，只是二级或更高级域名不同的站点，可以通过设置 domain 参数共享 cookie 读写。这种场景可以选择不跨域的 SSO 方案。

域名相同，只是 https 和 http 协议不同的 URL，默认 cookie 可以共享。知道这一点对处理 SSO 服务中心要登出

2.3 http 协议是无状态协议。浏览器访问服务器时，要让服务器知道你是谁，只有两种方式：

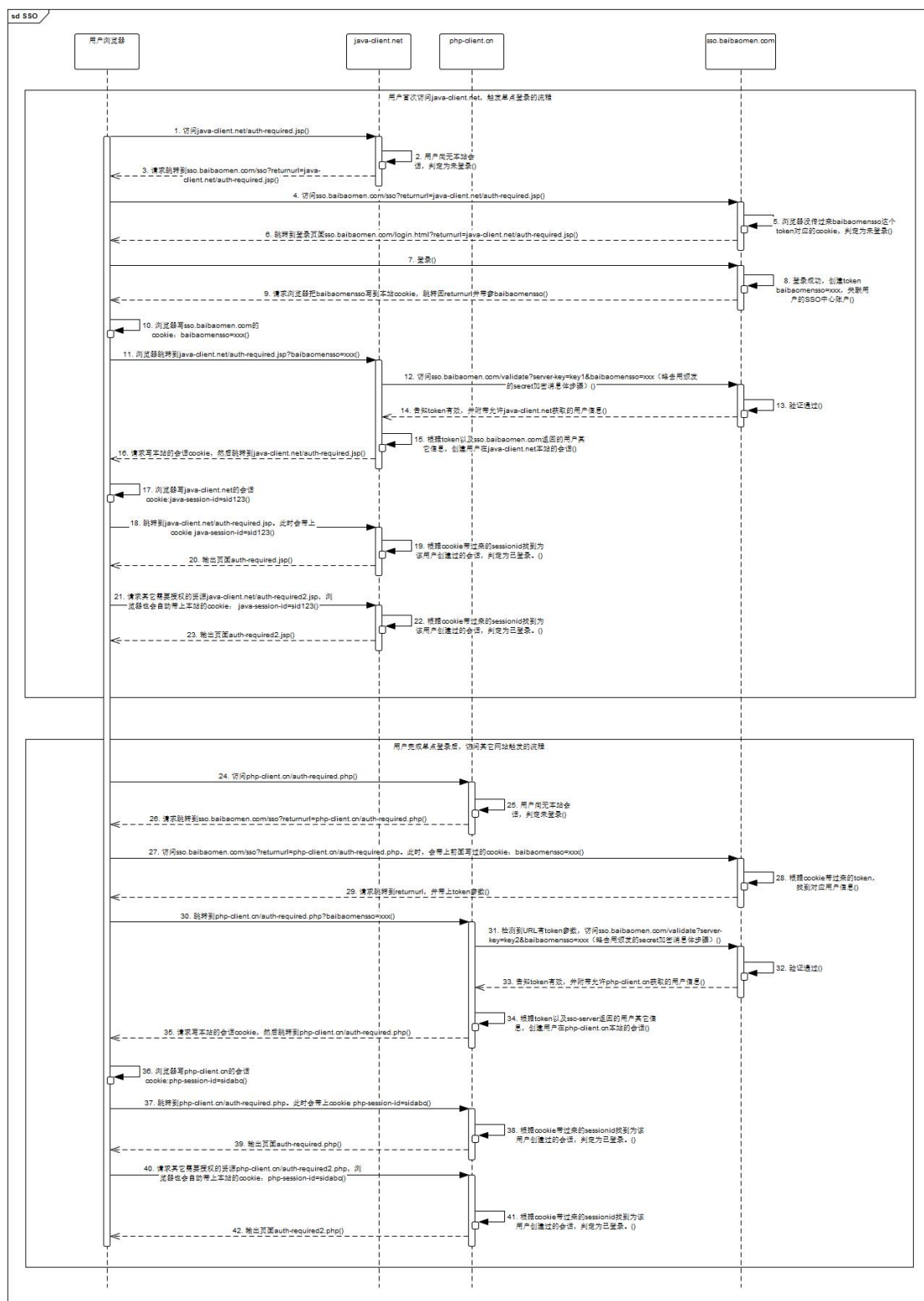
方式一：把“你是谁”写入 cookie。它会随每次 HTTP 请求带到服务端；

方式二：在 URL、表单数据中带上你的用户信息（也可能在 HTTP 头部）。这种方式依赖于从特定的网页入口进入，因为只有走特定的入口，才有机会拼装出相应的信息，提交到服务端。

大部分 SSO 需求都希望不依赖特定的网页入口（集成门户除外），所以后一种方式有局限性。适应性强的方式是第一种，即在浏览器通过 cookie 保存用户信息相关凭据，随每次请求传递到服务端。我们采用的方案是第一种。



1、Cookie 接入方式



2、Token 接入方式

类似社交登陆

3、有状态登录

为了保证客户端 `cookie` 的安全性，服务端需要记录每次会话的客户端信息，从而识别客户端身份，根据用户身份进行请求的处理，典型的设计如 `tomcat` 中的 `session`。

例如登录：用户登录后，我们把登录者的信息保存在服务端 `session` 中，并且给用户一个 `cookie` 值，记录对应的 `session`。然后下次请求，用户携带 `cookie` 值来，我们就能识别到对应 `session`，从而找到用户的信息。

缺点是什么？

- 服务端保存大量数据，增加服务端压力
- 服务端保存用户状态，无法进行水平扩展
- 客户端请求依赖服务端，多次请求必须访问同一台服务器

即使使用 `redis` 保存用户的信息，也会损耗服务器资源。

4、无状态登录

微服务集群中的每个服务，对外提供的都是 `Rest` 风格的接口。而 `Rest` 风格的一个最重要的规范就是：服务的无状态性，即：

- 服务端不保存任何客户端请求者信息
- 客户端的每次请求必须具备自描述信息，通过这些信息识别客户端身份

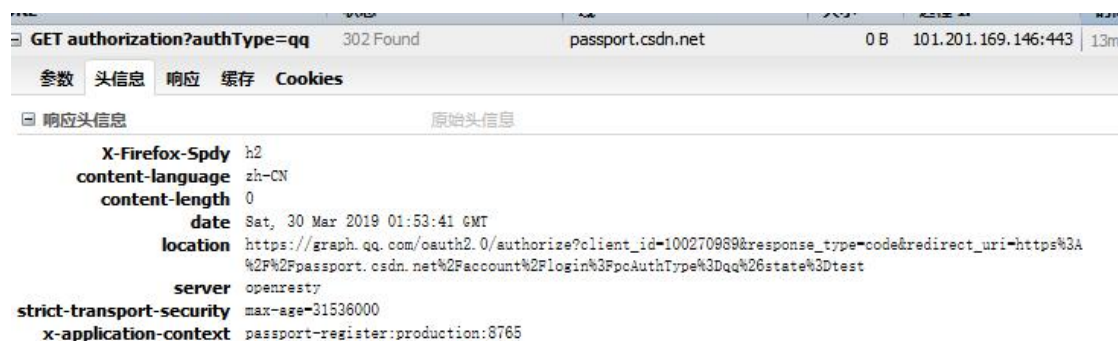
带来的好处是什么呢？

- 客户端请求不依赖服务端的信息，任何多次请求不需要必须访问到同一台服务
- 服务端的集群和状态对客户端透明
- 服务端可以任意的迁移和伸缩
- 减小服务端存储压力

5、集成社交登陆

1)、用户点击不同的社交登陆按钮，先来我们自己的服务器

<https://passport.csdn.net/v1/register/authorization?authType=qq/sina>



2)、命令浏览器重定向到用户授权页

用户确认授权

<https://graph.qq.com/oauth2.0/authorize>



3)、qq 返回的响应，会命令用户重定向到指定位置

4)、服务器的这个位置就可以收到我们的 code 码

收到 code 码，服务器自己用 code 交换 access_token 令牌，并获取到用户的信息。给浏览器只给用户的信息即可；

access_token=UUID

浏览器访问带 UUID_token 而不是 access_token；

三、JWT

1、简介

JWT，全称是 Json Web Token，是 JSON 风格轻量级的授权和身份认证规范，可实现无状态、分布式的 Web 应用授权；官网：<https://jwt.io>

GitHub 上 jwt 的 java 客户端：<https://github.com/jwtkt/jjwt>

我们最终可以利用 jwt 实现无状态登录

2、数据格式

JWT 包含三部分数据：

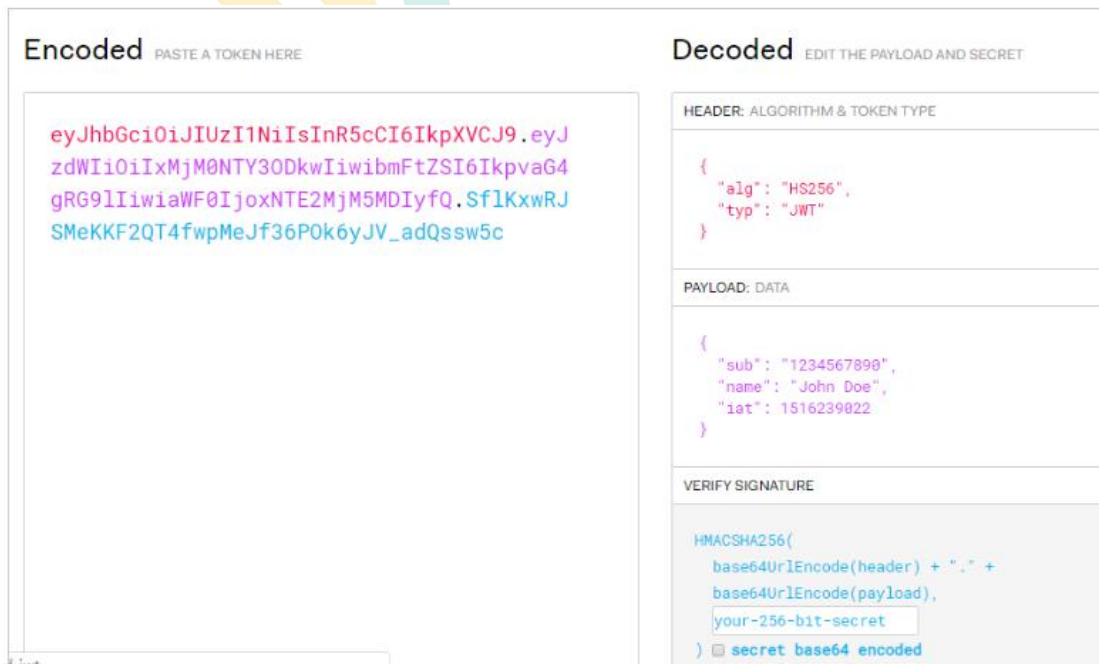
- **Header: 头部**，通常头部有两部分信息：

- token 类型：JWT
- 加密方式：base64 (HS256)

- **Payload: 载荷**，就是有效数据，一般包含下面信息：

- 用户身份信息（注意，这里因为采用 base64 编码，可解码，因此不要存放敏感信息）
 - 注册声明：如 token 的签发时间，过期时间，签发人等
- 这部分也会采用 base64 编码，得到第二部分数据

- **Signature: 签名**，是整个数据的认证信息。根据前两步的数据，再加上指定的密钥（secret）（不要泄漏，最好周期性更换），通过 base64 编码生成。用于验证整个数据完整和可靠性



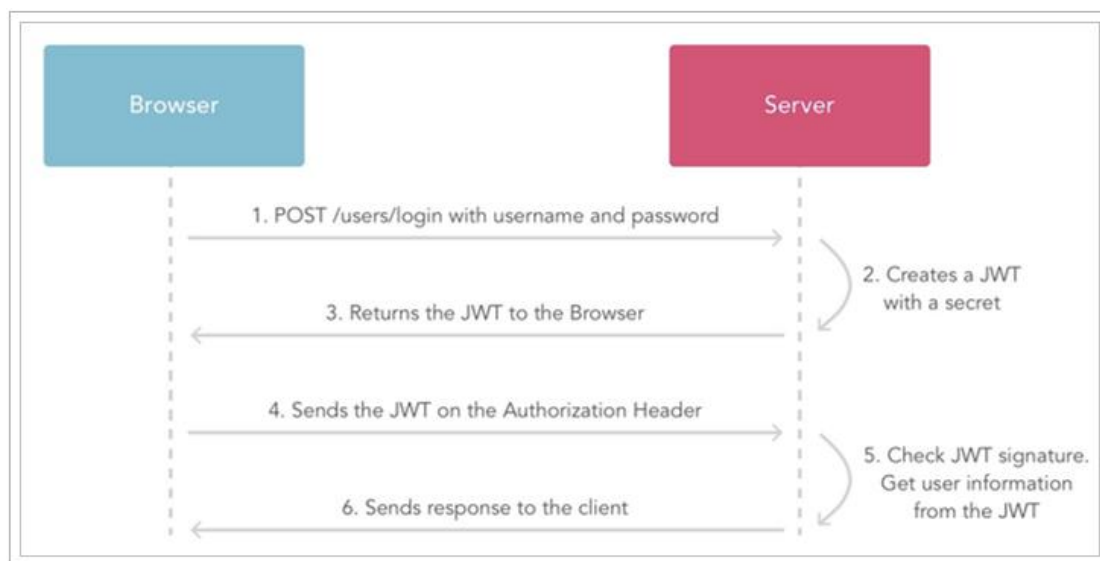
The image shows a web-based tool for encoding and decoding JWT tokens. It is divided into two main sections: 'Encoded' and 'Decoded'.

Encoded Section: Labeled 'PASTE A TOKEN HERE'. It contains a text area with a sample JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c`.

Decoded Section: Labeled 'EDIT THE PAYLOAD AND SECRET'. It displays the decoded components of the token:

- HEADER: ALGORITHM & TOKEN TYPE:** `{ "alg": "HS256", "typ": "JWT" }`
- PAYLOAD: DATA:** `{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }`
- VERIFY SIGNATURE:** Shows the signature verification process: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret)`. Below this, there is a checkbox labeled 'secret base64 encoded' which is currently checked.

3、交互流程

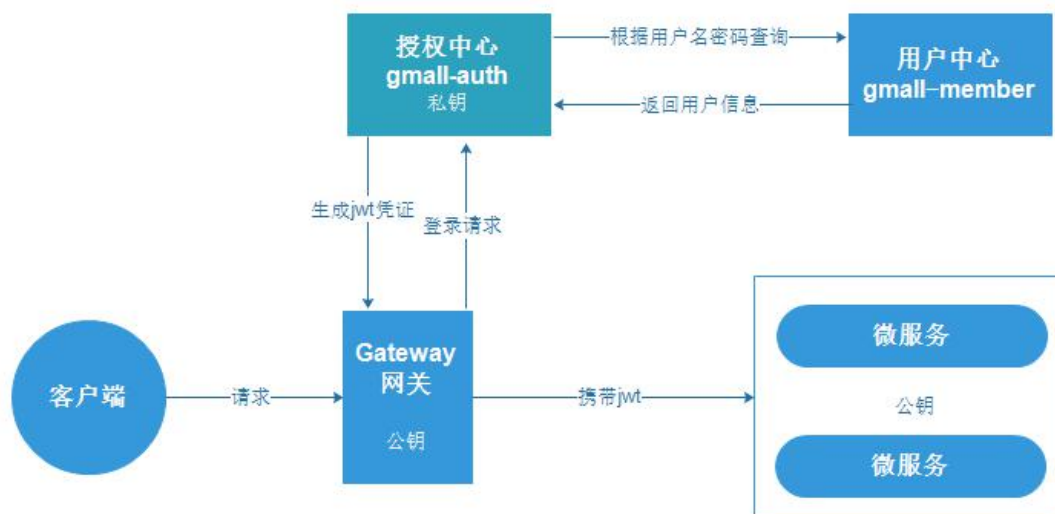


步骤：

- 1、用户登录
- 2、服务的认证，通过后根据 secret 生成 token
- 3、将生成的 token 返回给浏览器
- 4、用户每次请求携带 token
- 5、服务端利用秘钥解读 jwt 签名，判断签名有效后，从 Payload 中获取用户信息
- 6、处理请求，返回响应结果

因为 JWT 签发的 token 中已经包含了用户的身份信息，并且每次请求都会携带，这样服务的就无需保存用户信息，甚至无需去数据库查询，完全符合了 Rest 的无状态规范。

4、授权中心流程



5、JWT 优势

- 易于水平扩展
 - 在 cookie-session 方案中，cookie 内仅包含一个 session 标识符，而诸如用户信息、授权列表等都保存在服务端的 session 中。如果把 session 中的认证信息都保存在 JWT 中，在服务端就没有 session 存在的必要了。当服务端水平扩展的时候，就不用处理 session 复制（session replication）/ session 黏连（sticky session）或是引入外部 session 存储了[实际上 spring-session 和 hazelcast 能完美解决这个问题]。
- 防护 CSRF（跨站请求伪造）攻击
 - 访问某个网站会携带这个域名下的 cookie。所以可能导致攻击。但是我们可以把 jwt 放在请求头中发送。
 - Jwt 放在请求头中，就必须把 jwt 保存在 cookie 或者 localStorage 中。保存这里 js 就会读写，又会导致 xss 攻击。可以设置 cookie，httponly=true 来防止 xss
- 安全
 - 只是 base64 编码了，cookie+session 直接将数据保存在服务端，看都看不见，请问哪个更安全？
 -

6、使用 JWT 带来的问题

- 我们不建议使用 jwt+cookie 代替 session+cookie 机制，jwt 更适合 restful api
- jwt token 泄露了怎么办？
 - 这个问题可以不考虑，因为 session+cookie 同样泄露了 cookie 的 jsessionid 也会有这个问题
 - 我们可以遵循以下规范减少风险
 - ◆ 使用 https 加密应用
 - ◆ 返回 jwt 给客户端时设置 httpOnly=true 并且使用 cookie 而不是 LocalStorage 存储 jwt，防止 XSS 攻击和 CSRF 攻击
- secret 如果泄露会导致大面积风险
 - 定期更新
 - Secret 设计可以 and 用户关联起来，每个用户不一样。防止全用一个 secret
- 注销和修改密码
 - 传统的 session+cookie 方案用户点击注销，服务端清空 session 即可，因为状态保存在服务端。我们不害怕注销后的假登录
 - Jwt 会有问题。用户如果注销了或者修改密码了。恶意用户还使用之前非法盗取来的 token，可以在不重新登录的情况下继续使用
 - ◆ 可以按程度使用如下设计，减少一定的风险
 - 清空客户端的 cookie，这样用户访问时就不会携带 jwt，服务端就认为用户需要重新登录。这是一个典型的假注销，对于用户表现出退出的行为，实际上这个时候携带对应的 jwt 依旧可以访问系统。
 - 清空或修改服务端的用户对应的 secret，这样在用户注销后，jwt 本身不变，但是由于 secret 不存在或改变，则无法完成校验。这也是为什么将

secret 设计成和用户相关的原因

- 借助第三方存储，管理 jwt 的状态，可以以 jwt 为 key，去 redis 校验存在性。但这样，就把无状态的 jwt 硬生生变成了有状态了，违背了 jwt 的初衷。实际上这个方案和 session 都差不多了。
- 修改密码则略微有些不同，假设号被到了，修改密码（是用户密码，不是 jwt 的 secret）之后，盗号者在原 jwt 有效期之内依旧可以继续访问系统，所以仅仅清空 cookie 自然是不够的，这时，需要强制性的修改 secret

■ 续签问题

- ◆ 传统的 cookie 续签方案一般都是框架自带的，session 有效期 30 分钟，30 分钟内如果有访问，session 有效期被刷新至 30 分钟。而 jwt 本身的 payload 之中也有一个 exp 过期时间参数，来代表一个 jwt 的时效性，而 jwt 想延期这个 exp 就有点身不由己了，因为 payload 是参与签名的，一旦过期时间被修改，整个 jwt 串就变了，jwt 的特性天然不支持续签！
- ◆ 可如下解决，但都不是完美方案
 - 每次请求刷新 jwt：简单暴力，性能低下，浪费资源。
 - 只要快要过期的时候刷新 jwt：jwt 最后的几分钟，换新一下。但是如果用户连续操作了 27 分钟，只有最后的 3 分钟没有操作，导致未刷新 jwt，就很难受。
 - 完善 refreshToken：借鉴 oauth2 的设计，返回给客户端一个 refreshToken，允许客户端主动刷新 jwt。这样做，还不如用 oauth2
 - 使用 redis 记录独立的过期时间：jwt 作为 key，在 redis 中保存过期时间，每次使用在 redis 中续期，如果 redis 没有就认为过期。但是这样做，还不如用 session+cookie

■ 总结

- ◆ 在 Web 应用中，别再把 JWT 当做 session 使用，绝大多数情况下，传统的 cookie-session 机制工作得更好
- ◆ JWT 适合一次性的命令认证，颁发一个有效期极短的 JWT，即使暴露了危险也很小，由于每次操作都会生成新的 JWT，因此也没必要保存 JWT，真正实现无状态。