跨域认证

教学内容

- 第　　Session
- 第二　Token
- 第三　JWT的使用

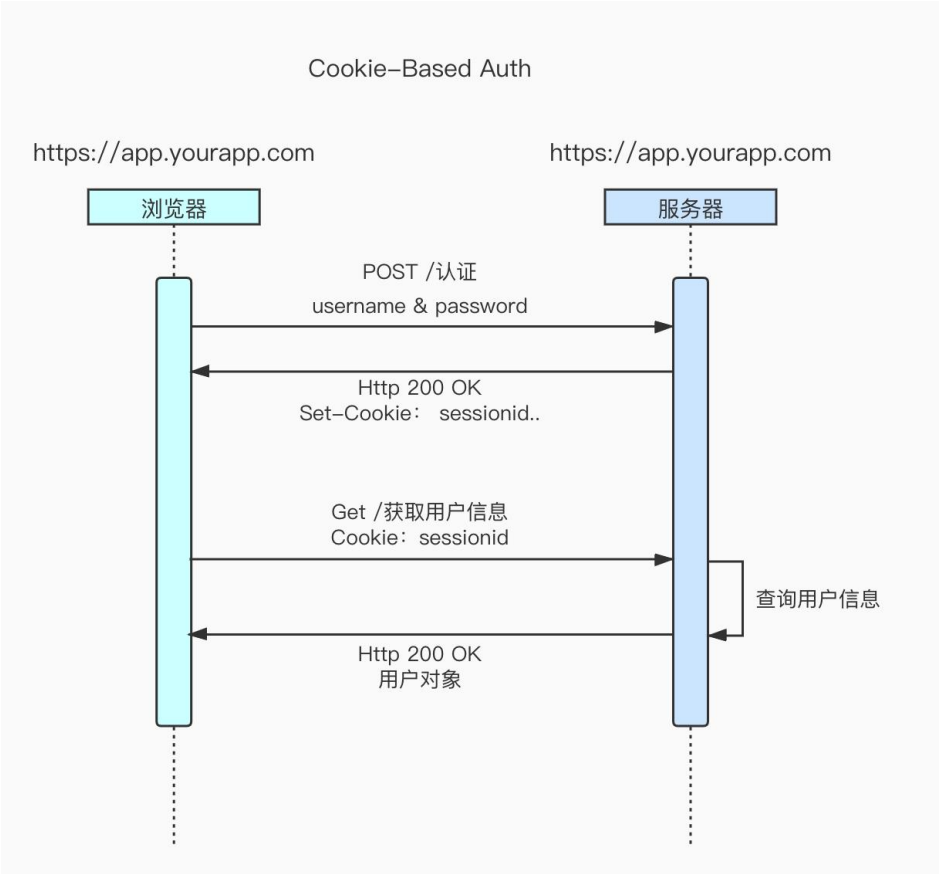
Session

互　　服务离不开用户　　流程是下　　样

- 用户向服务器发　用户名和密码
- 服务器　　后，在当前对　（session）　　保存相关数据，比如用户登录时　等
- 服务器向用户　回　个 session_id，写入用户的 Cookie
- 用户　后的每　次　求，　会　　Cookie，将 session_id 传回服务器
- 服务器收到 session_id，找到前期保存的数据，由此得知用户的　份

Session

- session　　流程



Session

session 的方式应用 常普 ，但也存在 些 ，扩展性不好，如果是服务器 ，或 是 域的服务导向架构，就 求 session 数据共享，每台服务器够 取 session， 对此种 有两种方案：

- 种 决方案是session 数据持久化，写入数据库或别的持久层 各种服务收到 求后， 向持久层 求数据 种方案的优点是架构清晰， 点是工程比 大
- 种方案是服务器不再保存 session 数据，所有数据 保存在客户端，每次求 发回服务器 Token 就是 种方案的 个代

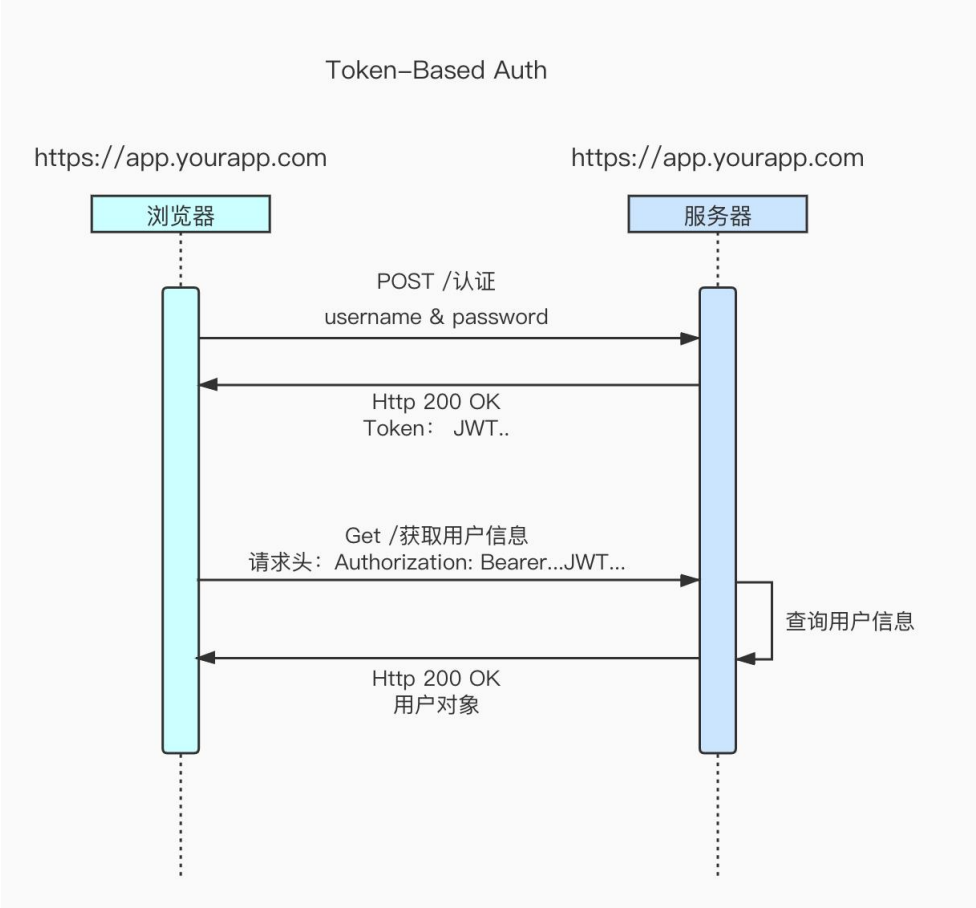
Token

Token 是在服务端产生的　串字符串,是客户端　　源接口（API）时所　　的源凭　，流程如下：

- 客户端使用用户名　密码　求登录，服务端收到　求，去　　用户名与密码
- 　　成功后，服务端会签发　个 token 并把　个 token 发　　客户端
- 客户端收到 token 以后，会把它存储　来，比如放在 cookie　或 localStorage
- 客户端每次向服务端　求　源的时候　　带着服务端签发的 token
- 服务端收到　求，然后去　　客户端　求　　带着的 token ，如果　　成功，就向客户端　回　求的数据

Token

- token　　流程



Token 的特点

- 基于 token 的用户 是 种服务端无状态的 方式，服务端不用存放 token 数据
- 用 析 token 的 算时 换取 session 的存储空 ，从 减 服务器的压力，减少 的查 数据库
- token 完全由应用 理，所以它可以 开同源策略

JWT

- JSON Web Token（简称 JWT）是　个token的具体实现方式，是目前最流的　域　　决方案
- JWT 的原理是，服务器　　以后，生成　个 JSON 对　，发回　用户，具体如下：

```
{
  "姓名": "张三",
  "角色": "管理员",
  "到期时间": "2018年7月1日0点0分"
}
```

- 用户与服务端　信的时候，　　发回　个 JSON 对　　服务器完全只　　个对　　定用户　份
- 为了　止用户　改数据，服务器在生成　个对　的时候，会加上签名

JWT

JWT 的由三个　分　成，依次如下：

- Header（头　）
- Payload（　　）
- Signature（签名）

三　分最　　合为完整的字符串，中　使用 . 分　，如下：

- Header.Payload.Signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Header

- Header　分是　个 JSON 对　，描　JWT 的元数据

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

- alg属性　示签名的算法（algorithm），　是 HMAC SHA256（写成 HS256）
- typ属性　示　个令牌（token）的　型（type），JWT 令牌　写为JWT
- 最后，将上　的 JSON 对　使用 Base64URL 算法　成字符串

Payload

- Payload　分也是　个 JSON 对　，用来存放实　　传　的数据　JWT　定了7个官方字段，供　用
  - iss (issuer)：签发人
  - exp (expiration time)：过期时间
  - sub (subject)：主题
  - aud (audience)：受众
  - nbf (Not Before)：生效时间
  - iat (Issued At)：签发时间
  - jti (JWT ID)：编号
- 注意，JWT　　是不加密的，任何人　可以　到，所以不　把秘密信息放在个　分
- 　个 JSON 对　也　使用 Base64URL 算法　成字符串

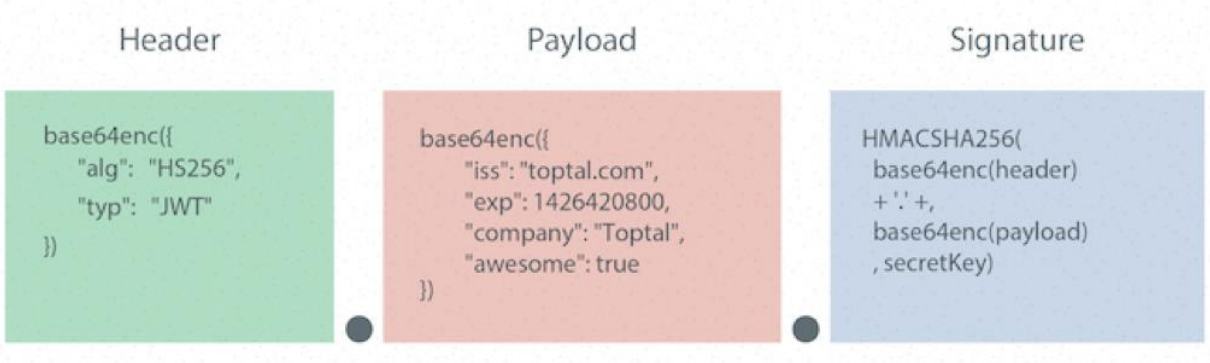
Signature

- Signature　分是对前两　分的签名，　止数据　改
- 　先，　指定　个密　（secret）　个密　只有服务器才知　，不　泄用户
- 然后，使用 Header　指定的签名算法（　是 HMAC SHA256），按照下的公式产生签名

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

JWT

- 算出签名以后，把 Header　Payload　Signature 三个　分拼成　个字符串，每个　分之　用"点"（`.`）分　，就可以　回　用户



JWT的特点

- 客户端收到服务器　回的 JWT，可以储存在 Cookie　　，也可以储存在 localStorage
- 客户端每次与服务器　信，　　带上　个 JWT，可以把它放在 Cookie　动发　，但是　样不　　域
- 更好的做法是放在 HTTP　求的头信息`Authorization`字段　　，单独发

JWT的实现

- 加入依

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

生成Token

```
//7天过期
private static long expire = 604800;
//32位秘钥
private static String secret = "abcdfghiabcdfghiabcdfghiabcdfghi";

//生成token
public static String generateToken(String username){
    Date now = new Date();
    Date expiration = new Date(now.getTime() + 1000 * expire);
    return Jwts.builder()
            .setHeaderParam("type","JWT")
            .setSubject(username)
            .setIssuedAt(now)
            .setExpiration(expiration)
            .signWith(SignatureAlgorithm.HS512,secret)
            .compact();
}
```

析Token

```
//解析token
public static Claims getClaimsByToken(String token){
    return Jwts.parser()
                .setSigningKey(secret)
                .parseClaimsJws(token)
                .getBody();

}
```