JSON Web Token

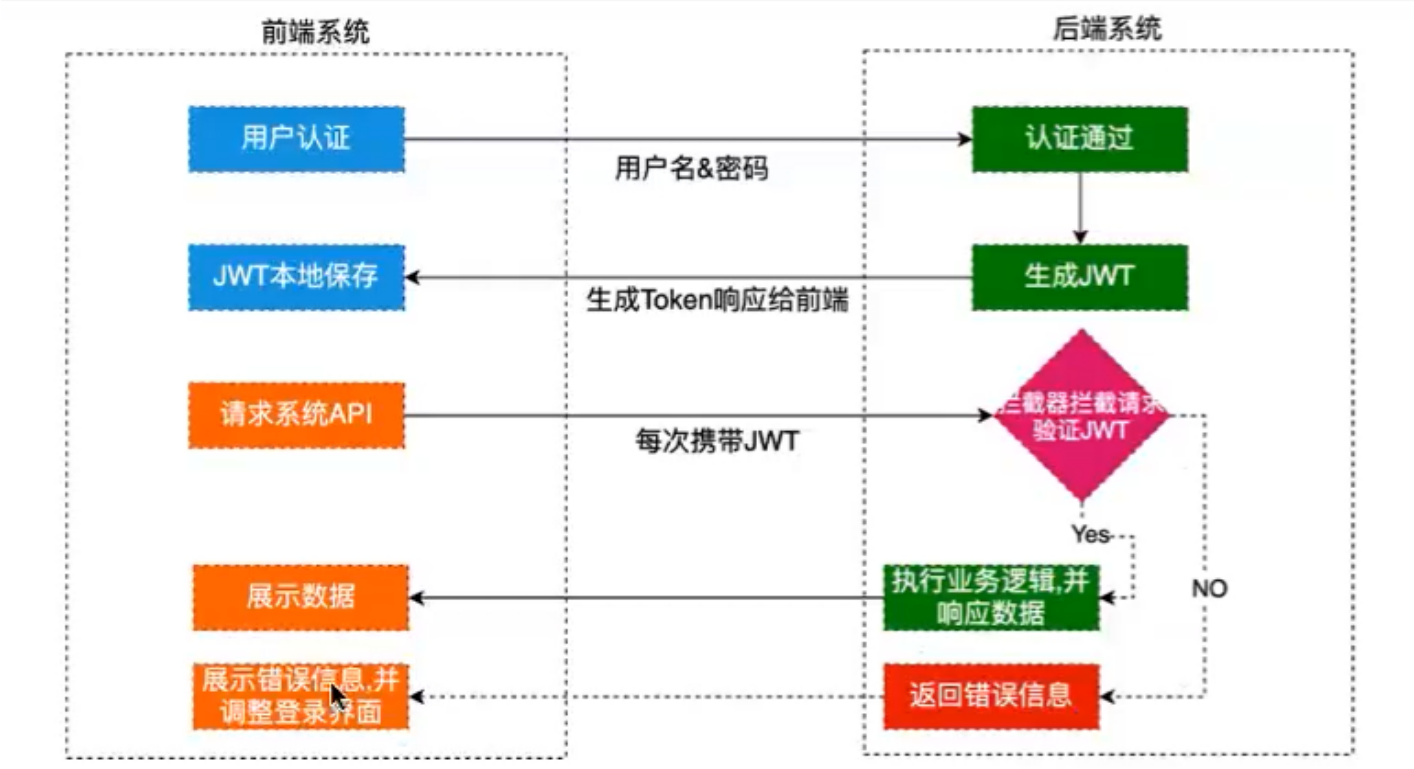
1. 什么是JWT?

- JSON Web Token (JWT)是一个开放标准(RFC 7519)，它定义了一种紧凑的、自包含的方式，用于作为JSON对象在各方之间安全地传输信息。该信息可以被验证和信任，因为**它是数字签名的。**

1.1 什么时候应该用JWT?

- **Authorization (授权)** : 这是使用JWT的最常见场景。一旦用户登录，后续每个请求都将包含JWT，允许用户访问该令牌允许的路由、服务和资源。**单点登录**是现在广泛使用的JWT的一个特性，因为它的开销很小，并且可以轻松地跨域使用。
- **Information Exchange (信息交换)** : 对于安全的在各方之间传输信息而言，JSON Web Tokens无疑是一种很好的方式。因为JWT可以被签名，例如，用公钥/私钥对，你可以确定发送人就是它们所说的那个人。另外，由于签名是使用头和有效负载计算的，您还可以**验证内容没有被篡改。**

1.2 认证流程



1. 首先，前端通过Web表单将自己的用户名和密码发送到后端的接口。这一过程一般是一个HTTP **POST请求**。建议的方式是通过SSL加密的传输(https协议) ，从而避免敏感信息被嗅探。

2. 后端核对用户名和密码成功后，将**用户的id等其他信息作为JWT Payload** (负载)，将其与头部分别进行Base64编码拼接后签名，形成一个JWT(Token)。形成的JWT就是一个形同11. zzz. xxx的字符串。**token head.payload.signature**
3. 后端将JWT字符串作为登录成功的返回结果**返回给前端**。 前端可以将返回的结果保存在localStorage或sessionStorage上， 退出登录时前端删除保存的JWT即可。
4. 前端在每次请求时将JWT**放入HTTP Header中的Authorization位**。 (解决XSS和XSRF问题)
5. 后端检查是否存在，如存在验证JWT的有效性。
   - 检查签名是否正确;
   - 检查Token是否过期;
   - 检查Token的接收方是否是自己(可选)
6. 验证通过后后端使用JWT中包含的用户信息进行其他逻辑操作，返回相应结果。

1.3 JWT优势在哪?

1. 简洁(Compact): 可以通过URL，POST参数或者在HTTP header发送，数据量小，传输速度快
2. 自包含(Self-contained):**负载中包含了所有用户所需要的信息**，避免了多次查询数据库
3. 因为Token是 以**JSON加密**的形式**保存在客户端**的，所以JWT是跨语言的，原则上任何web形式都支持。
4. 不需要在服务端保存会话信息，特别**适用于分布式微服务**。

1.4 JWT具体包含信息

1. **header**

标头通常由两部分组成: **令牌的类型**(即JWT) 和**所使用的签名算法**，例如HMAC、SHA256或RSA。 它会使用Base64 编码组成JWT 结构的第一部分。

注意:Base64是一 种编码，也就是说，它是可以被翻译回原来的样子来的。它并不是一种加密过程。

Java

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

2. **Payload**

令牌的第二部分是有效负载，其中包含声明。**声明是有关实体(通常是用户)和其他数据的声明**。同样的，它会使用Base64 编码组成JWT结构的第二部分

JSON

```
{
    "sub" : "HS256"
    "name" : "yjiewei"
    "admin" : "true"
}
```

3. Signature

header和payload都是结果Base64编码过的，中间用.隔开，第三部分就是前面两部分合起来做签名，密钥绝对自己保管好，签名值同样做Base64编码拼接在JWT后面。（签名并编码）

Apache

```
HMACSHA256 (base64Ur1Encode(header) + "." + base64Ur1Encode(payload) , secret);
```

2. 光说不练假把式

2.1 整合pom

HTML

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!--引入mybatis-->
    <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
        <version>2.1.0</version>
    </dependency>

    <!--引入jwt-->
    <dependency>
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>3.10.3</version>
    </dependency>
```

```
20
21      <!--引入mysql-->
22      <dependency>
23          <groupId>mysql</groupId>
24          <artifactId>mysql-connector-java</artifactId>
25          <version>8.0.16</version>
26      </dependency>
27
28      <!--引入druid-->
29      <dependency>
30          <groupId>com.alibaba</groupId>
31          <artifactId>druid</artifactId>
32          <version>1.2.0</version>
33      </dependency>
34
35      <dependency>
36          <groupId>org.projectlombok</groupId>
37          <artifactId>lombok</artifactId>
38          <optional>true</optional>
39      </dependency>
40      <dependency>
41          <groupId>org.springframework.boot</groupId>
42          <artifactId>spring-boot-starter-test</artifactId>
43          <scope>test</scope>
44      </dependency>
45  </dependencies>
```

2.2 测试JWT加密过程

TypeScript

```
1   public class Jwt {
2
3       /**
4        * 获取JWT令牌
5        */
6       @Test
7       public void getToken() {
8           Map<String, Object> map = new HashMap<>();
9
10          Calendar instance = Calendar.getInstance();
11          instance.add(Calendar.SECOND, 2000);
12
13          /**
```

```
        /**
         * header可以不写有默认值
         * payload 通常用来存放用户信息
         * signature 是前两个合起来的签名值
         */
        String token = JWT.create().withHeader(map) //header
                .withClaim("userId", 21)//payload
                .withClaim("username", "yjiewei")//payload
                .withExpiresAt(instance.getTime())//指定令牌的过期时间
                .sign(Algorithm.HMAC256("!RHO4$%*^fi$R")); //签名，密钥自己记住
        System.out.println(token);
    }

    /**
     * 令牌验证:根据令牌和签名解析数据
     * 常见异常:
     *   SignatureVerificationException 签名不一致异常
     *   TokenExpiredException          令牌过期异常
     *   AlgorithmMismatchException     算法不匹配异常
     *   InvalidClaimException          失效的payload异常
     */
    @Test
    public void tokenVerify() {
        // token值传入做验证
        String token = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE2MjkyMDg0NjgsInVzZXJJZCI6MjEsInVzZXJuYW1lIjoieWppZXdlaSJ9.e4auZWkykZ2Hu8Q20toaks-4e62gerPlDEPHvhunCnQ";
        /**
         * 用户Id: 21
         * 用户名: yjiewei
         * 过期时间: Tue Aug 17 21:54:28 CST 2021
         */
        JWTVerifier jwtVerifier = JWT.require(Algorithm.HMAC256("!RHO4$%*^fi$R")).build();
        DecodedJWT decodedJWT = jwtVerifier.verify(token); // 验证并获取解码后的token
        System.out.println("用户Id: " + decodedJWT.getClaim("userId").asInt());
        System.out.println("用户名: " + decodedJWT.getClaim("username").asString());
        System.out.println("过期时间: " + decodedJWT.getExpiresAt());
    }
}
```

3. 封装工具类

Java就是封装抽象封装抽象...所以这里也封装一个工具类。

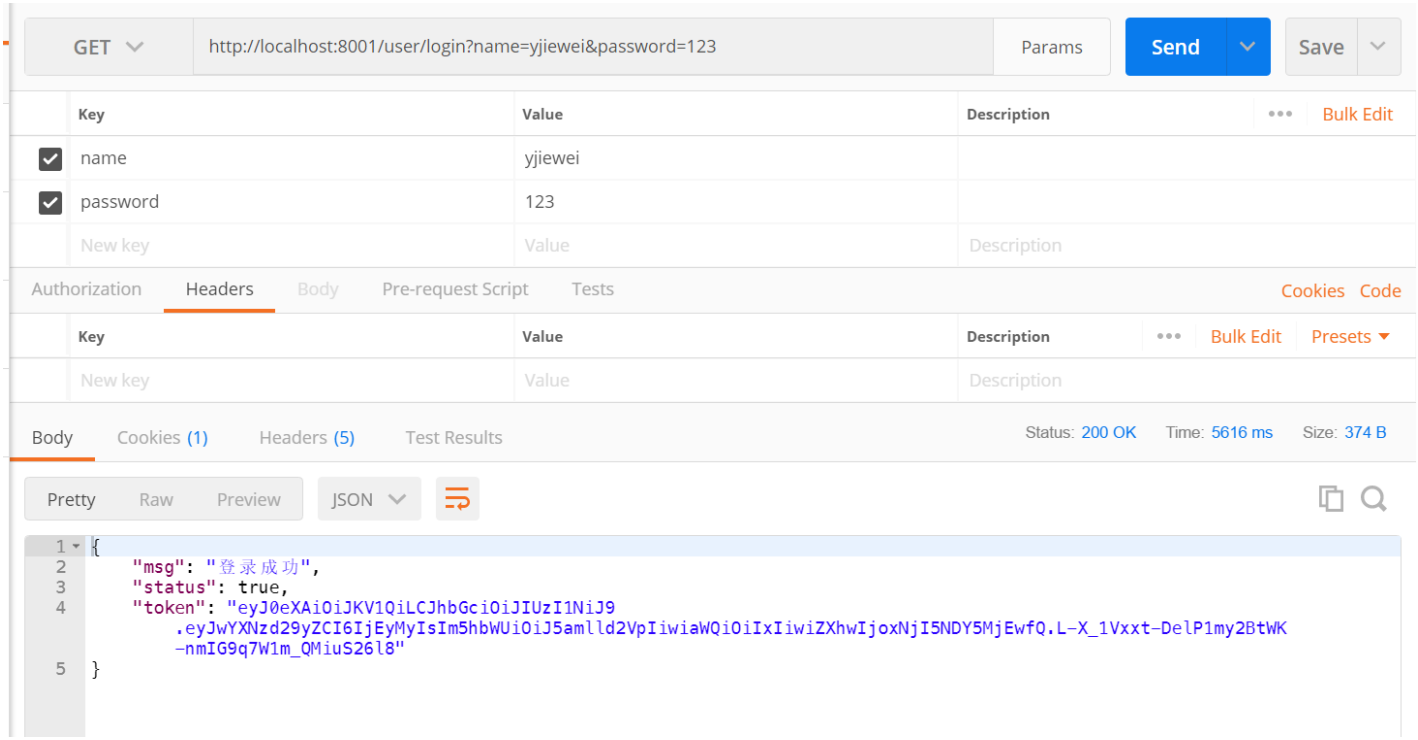
TypeScript

```
public class JWTUtil {

    /**
     * 密钥要自己保管好
     */
    private static String SECRET = "privatekey#^&^%!save";

    /**
     * 传入payload信息获取token
     * @param map payload
     * @return token
     */
    public static String getToken(Map<String, String> map) {
        JWTCreator.Builder builder = JWT.create();

        //payload
        map.forEach(builder::withClaim);

        Calendar instance = Calendar.getInstance();
        instance.add(Calendar.DATE, 3); //默认3天过期

        builder.withExpiresAt(instance.getTime());//指定令牌的过期时间
        return builder.sign(Algorithm.HMAC256(SECRET));
    }

    /**
     * 验证token
     */
    public static DecodedJWT verify(String token) {
        //如果有任何验证异常，此处都会抛出异常
        return JWT.require(Algorithm.HMAC256(SECRET)).build().verify(token);
    }

    /**
     * 获取token中的payload
     */
    public static Map<String, Claim> getPayloadFromToken(String token) {
        return
JWT.require(Algorithm.HMAC256(SECRET)).build().verify(token).getClaims();
    }
}
```

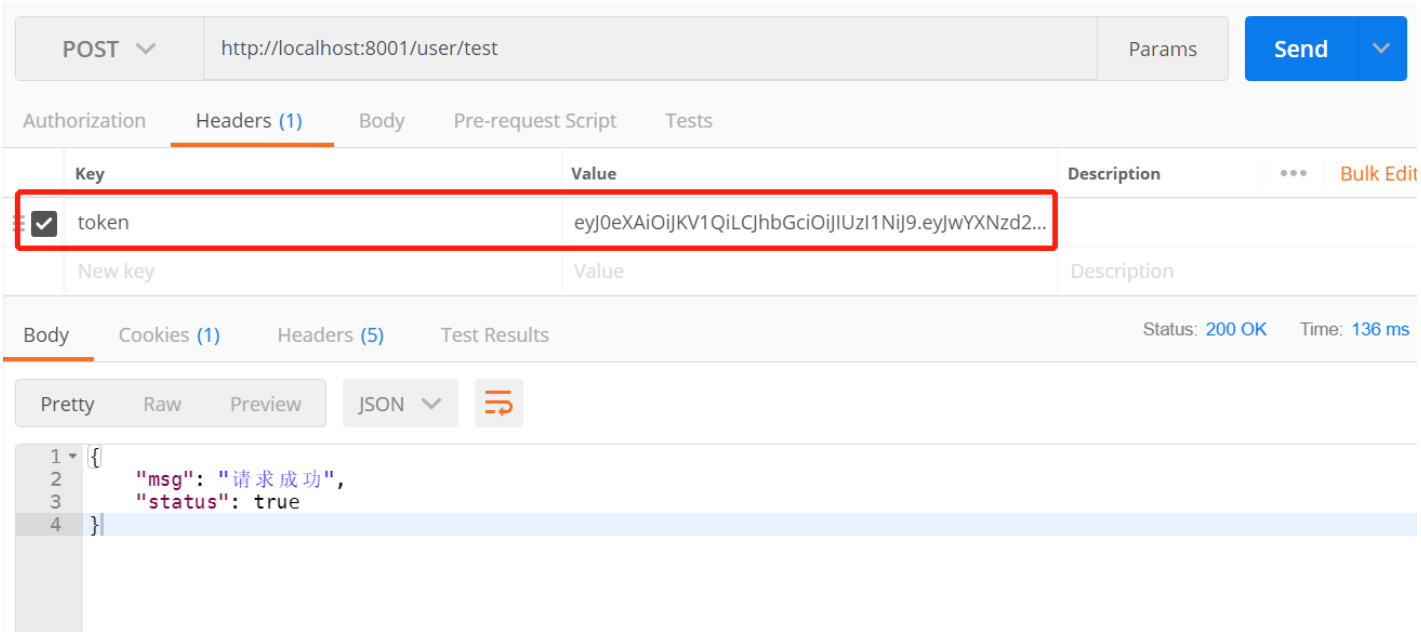
4. JWT 整合SpringBoot

> 我们要对每个请求都去验证实现单点登录，通过拦截器拦截请求对JWT做验证。

4.1 登录并获取token



4.2 在请求头中添加token



4.3 重要的代码片段

TypeScript

```
// 拦截器
@Slf4j
public class JWTInterceptor implements HandlerInterceptor {

    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {

        //获取请求头中的令牌
        String token = request.getHeader("token");
        log.info("当前token为: {}", token);

        Map<String, Object> map = new HashMap<>();
        try {
            JWTUtil.verify(token);
            return true;
        } catch (SignatureVerificationException e) {
            e.printStackTrace();
            map.put("msg", "签名不一致");
        } catch (TokenExpiredException e) {
            e.printStackTrace();
            map.put("msg", "令牌过期");
        } catch (AlgorithmMismatchException e) {
            e.printStackTrace();
            map.put("msg", "算法不匹配");
        } catch (InvalidClaimException e) {
            e.printStackTrace();
            map.put("msg", "失效的payload");
        } catch (Exception e) {
            e.printStackTrace();
            map.put("msg", "token无效");
        }

        map.put("status", false);

        //响应到前台: 将map转为json
        String json = new ObjectMapper().writeValueAsString(map);
        response.setContentType("application/json;charset=UTF-8");
        response.getWriter().println(json);
        return false;
    }
```

```
}
```

Java

```java
// 指定拦截路径
@Configuration
public class InterceptorConfig implements WebMvcConfigurer {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new JWTInterceptor())
                .addPathPatterns("/user/test")
                .excludePathPatterns("/user/login");
    }
}
```

TypeScript

```typescript
// 拦截请求并验证请求头中的token
@PostMapping("/user/test")
public Map<String, Object> test(HttpServletRequest request) {
    String token = request.getHeader("token");
    DecodedJWT verify = JWTUtil.verify(token);
    String id = verify.getClaim("id").asString(); // 我前面存的时候转字符串了
    String name = verify.getClaim("name").asString();
    log.info("用户id: {}", id);
    log.info("用户名: {}", name);

    //TODO:业务逻辑
    Map<String, Object> map = new HashMap<>();
    map.put("status", true);
    map.put("msg", "请求成功");
    return map;
}
```