

JWT

参考视频或文章：

- 【【极简入门】15分钟学会JWT的使用】https://www.bilibili.com/video/BV1cK4y197EM?vd_source=b7f14ba5e783353d06a99352d23ebca9
- https://blog.csdn.net/Tangzx_/article/details/135889397?fromshare=blogdetail&sharetype=blogdetail&sharerId=135889397&sharerefer=PC&sharesource=2401_83600210&sharefrom=from_link

一、技术介绍

1. 概述

- **JWT(JSON Web Token)**: 通过数字签名的方式，以JSON对象为载体，在不同的服务终端之间安全的传输信息。
- **作用**: 授权认证。用户一旦登录，后续每个请求都将携带JWT，系统每次处理用户的请求之前，都要先进行JWT安全校验，只有校验通过才能继续处理请求。

2. JWT的优势

- **无状态**: 因为JWT的验证基于密钥，所以它不需要在服务端存储用户信息。这使得JWT可以作为一种无状态的身份认证机制。
- **跨语言和跨平台支持**: 可以在多种语言和平台之间使用。
- **安全性高**: 由于JWT的载荷可以进行加密处理，因此JWT能够保证数据的安全传输。同时，JWT的签名机制也能够保证数据的完整性和真实性。

3. JWT的组成

JWT由三部分组成，用 . 拼接起来：

- Header (头部，一般情况下采用 `Base64` 编码)

```
1  {
2      "typ": "JWT", // 令牌类型
3      "alg": "HS256" // 加密算法
4 }
```

- Payload (载荷：也称为声明信息，包含了一些有关实体的信息以及其他元数据，一般情况下采用 [Base64](#) 编码)

⚠ Warning

Payload中不能存放敏感或重要信息！

```
1  {
2      "sub": "1234567890",
3      "name": "john",
4      "admin": "true"
5 }
```

- Signature (签名：由头部、载荷和密钥共同生成，用于验证JWT的真实性和完整性，一般情况下也采用 [Base64](#) 编码)

```
1 HMACSHA256(
2     base64UrlEncode(header) + '.' + base64UrlEncode(payload),
3     secret
4 )
```

二、项目应用

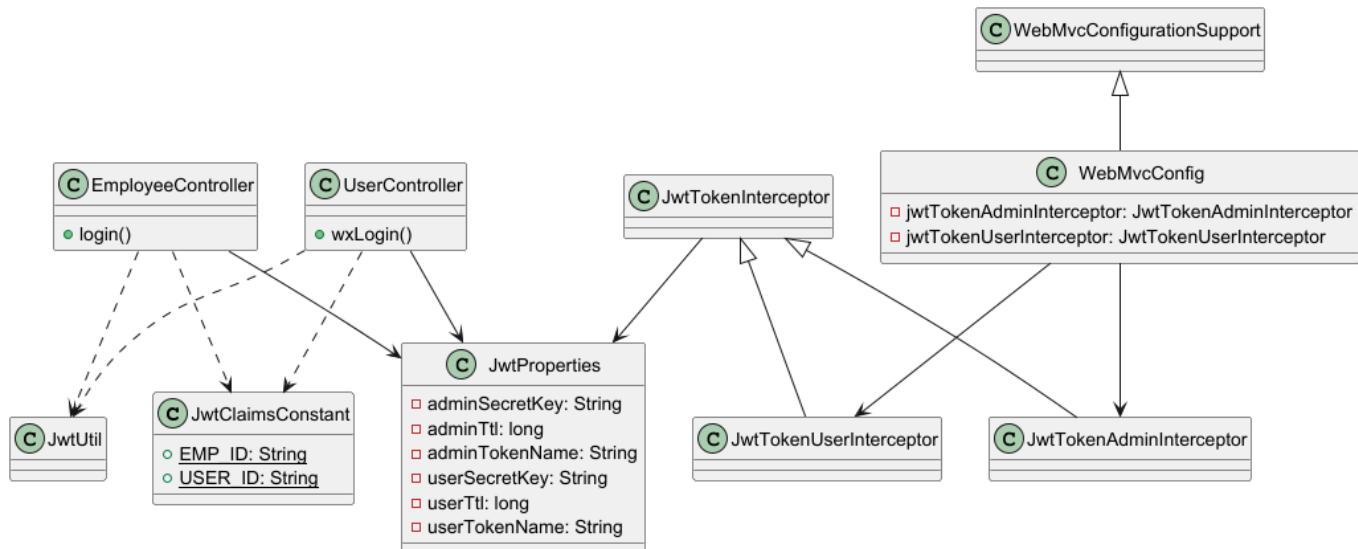
涉及到的文件如下：

```
1 sky-take-out: pom.xml
2
3 sky-common:
4     pom.xml
5     src/main/java/com.sky:
```

```

6     constant: JwtClaimsConstant
7     properties: JwtProperties
8     utils: JwtUtil
9
10    sky-server:
11        pom.xml
12        src/main/java/com.sky:
13            config: WebMvcConfig
14            controller:
15                admin: EmployeeController
16                user: UserController
17            interceptor:
18                JwtTokenInterceptor
19                JwtTokenAdminInterceptor
20                JwtTokenUserInterceptor
21        src/main/resources: application.yml

```



1. 导入JWT的 Maven 依赖坐标

1.1 sky-take-out: pom.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5             http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>

```

```
6   <parent>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <groupId>org.springframework.boot</groupId>
9     <version>2.7.3</version>
10    </parent>
11
12  <groupId>com.sky</groupId>
13  <artifactId>sky-take-out</artifactId>
14  <packaging>pom</packaging>
15  <version>1.0-SNAPSHOT</version>
16
17  <modules>
18    <module>sky-common</module>
19    <module>sky-pojo</module>
20    <module>sky-server</module>
21  </modules>
22
23  <properties>
24    <jjwt>0.9.1</jjwt>
25    <jaxb-api>2.3.1</jaxb-api>
26    <fastjson>1.2.76</fastjson>
27  </properties>
28
29  <dependencyManagement>
30    <dependencies>
31      <dependency>
32        <groupId>io.jsonwebtoken</groupId>
33        <artifactId>jjwt</artifactId>
34        <version>${jjwt}</version>
35      </dependency>
36      <dependency>
37        <groupId>javax.xml.bind</groupId>
38        <artifactId>jaxb-api</artifactId>
39        <version>${jaxb-api}</version>
40      </dependency>
41      <dependency>
42        <groupId>com.alibaba</groupId>
43        <artifactId>fastjson</artifactId>
44        <version>${fastjson}</version>
45      </dependency>
46    </dependencies>
47  </dependencyManagement>
48 </project>
```

1.2 sky-common: pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5             http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>sky-take-out</artifactId>
8         <groupId>com.sky</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11
12    <modelVersion>4.0.0</modelVersion>
13    <artifactId>sky-common</artifactId>
14
15    <dependencies>
16        <dependency>
17            <groupId>org.springframework.boot</groupId>
18            <artifactId>spring-boot-starter-json</artifactId>
19        </dependency>
20        <dependency>
21            <groupId>io.jsonwebtoken</groupId>
22            <artifactId>jjwt</artifactId>
23        </dependency>
24        <dependency>
25            <groupId>javax.xml.bind</groupId>
26            <artifactId>jaxb-api</artifactId>
27        </dependency>
28        <dependency>
29            <groupId>com.alibaba</groupId>
30            <artifactId>fastjson</artifactId>
31        </dependency>
32    </dependencies>
33 </project>
```

1.3 sky-server: pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5             http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>sky-take-out</artifactId>
8         <groupId>com.sky</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11
12    <modelVersion>4.0.0</modelVersion>
13    <artifactId>sky-server</artifactId>
14
15    <dependencies>
16
17        <dependency>
18            <groupId>com.sky</groupId>
19            <artifactId>sky-common</artifactId>
20            <version>1.0-SNAPSHOT</version>
21        </dependency>
22        <dependency>
23            <groupId>com.sky</groupId>
24            <artifactId>sky-pojo</artifactId>
25            <version>1.0-SNAPSHOT</version>
26        </dependency>
27
28        <dependency>
29            <groupId>org.springframework.boot</groupId>
30            <artifactId>spring-boot-starter</artifactId>
31        </dependency>
32        <dependency>
33            <groupId>org.springframework.boot</groupId>
34            <artifactId>spring-boot-starter-test</artifactId>
35            <scope>test</scope>
36        </dependency>
37        <dependency>
38            <groupId>org.springframework.boot</groupId>
39            <artifactId>spring-boot-starter-web</artifactId>
40            <scope>compile</scope>
41        </dependency>
```

```
42 <dependency>
43     <groupId>com.alibaba</groupId>
44     <artifactId>fastjson</artifactId>
45 </dependency>
46 <dependency>
47     <groupId>javax.xml.bind</groupId>
48     <artifactId>jaxb-api</artifactId>
49 </dependency>
50 </dependencies>
51
52 <build>
53     <plugins>
54         <plugin>
55             <groupId>org.springframework.boot</groupId>
56             <artifactId>spring-boot-maven-plugin</artifactId>
57         </plugin>
58     </plugins>
59 </build>
60 </project>
```

2. 编写JWT工具类 `JwtUtil`

```
1 /**
2  * JWT工具类
3 */
4 public class JwtUtil {
5
6     /**
7      * 生成jwt
8      * 使用HS256算法，私匙使用固定秘钥
9      *
10     * @param secretKey jwt秘钥
11     * @param ttlMillis jwt过期时间(毫秒)
12     * @param claims    设置的信息
13     */
14     public static String createJwt(String secretKey, long ttlMillis,
15         Map<String, Object> claims) {
```

```
16     // 指定签名的时候使用的签名算法是HS256
17     SignatureAlgorithm signatureAlgorithm =
18         SignatureAlgorithm.HS256;
19
20     // 生成jwt过期的时间点
21     long expMillis = System.currentTimeMillis() + ttlMillis;
22     Date exp = new Date(expMillis);
23
24     // 设置jwt的body
25     JwtBuilder builder = Jwts.builder()
26         // 如果有私有声明，一定要先设置这个自己创建的私有声明，这个是给
27         // builder的claims赋值，
28         // 一旦写在标准声明赋值之后，就会覆盖掉那些标准声明
29         .setClaims(claims)
30         // 设置签名使用的签名算法和秘钥
31         .signWith(signatureAlgorithm,
32             secretKey.getBytes(StandardCharsets.UTF_8))
33         // 设置过期时间
34         .setExpiration(exp);
35
36     /**
37      * token解密
38      *
39      * @param secretKey jwt秘钥：此秘钥一定要保存在服务端，不能泄露出去，否则
40      sign就可以被伪造，如果对接多个客户端建议改造成多个
41      * @param token      加密后的token
42      * @return
43      */
44     public static Claims parseJwt(String secretKey, String token) {
45         // 得到DefaultJwtParser
46         Claims claims = Jwts.parser()
47             // 设置签名的秘钥
48
49             .setSigningKey(secretKey.getBytes(StandardCharsets.UTF_8))
50             // 设置需要解析的jwt
51             .parseClaimsJws(token).getBody();
52
53     }
```

3. 编写JWT的Claims相关常量类 `JwtClaimsConstant`

```
1  /**
2   * JWT的Claims相关常量类
3   */
4  public class JwtClaimsConstant {
5      public static final String EMP_ID = "empId";
6      public static final String USER_ID = "userId";
7  }
```

4. 配置JWT相关属性类

4.1 `JwtProperties`

```
1 @Component
2 @ConfigurationProperties(prefix = "sky.jwt")
3 @Data
4 public class JwtProperties {
5
6     // 管理端员工生成jwt令牌相关配置
7     private String adminSecretKey;
8     private long adminTtl;
9     private String adminTokenName;
10
11    // 用户端微信用户生成jwt令牌相关配置
12    private String userSecretKey;
13    private long userTtl;
14    private String userTokenName;
15
16 }
```

4.2 application.yml

```
1 sky:
2   jwt:
3     # 设置jwt签名加密时使用的秘钥
4     admin-secret-key: cszsh
5     #Caution: 设置jwt过期时间，尽量设得长一点，防止过没多久运行项目令牌就过期报
6     401!
7       #Caution: 等项目上线后再改回来
8     admin-ttl: 7200000000
9     # 设置前端传递过来的令牌名称
10    admin-token-name: token
11    user-secret-key: itzsh
12    user-ttl: 7200000000
13    user-token-name: authentication
```

5.自定义拦截器

5.1 JwtTokenInterceptor

```
1 /**
2  * jwt令牌校验的拦截器抽象父类（模板方法设计模式）
3  */
4 @Slf4j
5 public abstract class JwtTokenInterceptor implements HandlerInterceptor {
6
7     @Autowired
8     protected JwtProperties jwtProperties;
9
10    @Override
11    public boolean preHandle(HttpServletRequest request,
12                             HttpServletResponse response, Object handler) throws Exception {
13        // 判断当前拦截到的是Controller的方法还是其他资源
14    }
15}
```

```

13     if (!(handler instanceof HandlerMethod)) {
14         // 当前拦截到的不是动态方法，直接放行
15         return true;
16     }
17
18     // 1.从请求头中获取令牌
19     String token = getAndTokenFromHeader(request);
20
21     // 2.校验令牌
22     try {
23         checkToken(token);
24         // 3.通过，放行
25         return true;
26     } catch (Exception e) {
27         // 4.不通过，响应401状态码
28         response.setStatus(401);
29         return false;
30     }
31 }
32
33 // 钩子方法，由子类重写
34 protected abstract String getAndTokenFromHeader(HttpServletRequest
request);
35     protected abstract void checkToken(String token);
36 }
```

5.2 JwtTokenAdminInterceptor

```

1 /**
2  * 管理端jwt令牌校验的拦截器
3 */
4 @Component
5 @Slf4j
6 public class JwtTokenAdminInterceptor extends JwtTokenInterceptor {
7
8     // 从请求头中获取管理员令牌
9     @Override
10    protected String getAndTokenFromHeader(HttpServletRequest request)
11    {
12        return request.getHeader(jwtProperties.getAdminTokenName());
13    }
14 }
```

```
13
14     // 校验管理员令牌，并将empId存入ThreadLocal中
15     @Override
16     protected void checkToken(String token) {
17         Claims claims =
18             JwtUtil.parseJwt(jwtProperties.getAdminSecretKey(), token);
19         Long empId =
20             Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());
21         BaseContext.setCurrentId(empId);
22     }
23 }
```

5.3 JwtTokenUserInterceptor

```
1 /**
2  * 用户端jwt令牌校验的拦截器
3  */
4 @Component
5 @Slf4j
6 public class JwtTokenUserInterceptor extends JwtTokenInterceptor {
7
8     // 从请求头中获取用户令牌
9     @Override
10    protected String getAndTokenFromHeader(HttpServletRequest request)
11    {
12        return request.getHeader(jwtProperties.getUserTokenName());
13    }
14
15     // 校验用户令牌，并将userId存入ThreadLocal中
16     @Override
17     protected void checkToken(String token) {
18         Claims claims =
19             JwtUtil.parseJwt(jwtProperties.getUserSecretKey(), token);
20         Long userId =
21             Long.valueOf(claims.get(JwtClaimsConstant.USER_ID).toString());
22         // BaseContext内部使用了ThreadLocal技术，它是线程隔离的
23         // 不会与BaseContext.setCurrentId(empId);发生冲突
24         BaseContext.setCurrentId(userId);
25     }
26 }
```

6. 在 `WebMvcConfig` 中将自定义拦截器添加到拦截器类中

```
1  /**
2  * 配置类，注册web层相关组件
3  */
4  @Configuration
5  @Slf4j
6  public class WebMvcConfig extends WebMvcConfigurationSupport {
7
8      @Autowired
9      private JwtTokenAdminInterceptor jwtTokenAdminInterceptor;
10     @Autowired
11     private JwtTokenUserInterceptor jwtTokenUserInterceptor;
12
13     // 设置静态资源映射
14     @Override
15     protected void addResourceHandlers(ResourceHandlerRegistry
16         registry) {
17         log.info("开始设置静态资源映射...");
18
19         registry.addResourceHandler("/doc.html").addResourceLocations("classpa
20         th:/META-INF/resources/");
21
22         registry.addResourceHandler("/webjars/**").addResourceLocations("class
23         path:/META-INF/resources/webjars/");
24     }
25
26
27     // 注册自定义拦截器
28     @Override
29     protected void addInterceptors(InterceptorRegistry registry) {
30         log.info("开始注册自定义拦截器...");
31         // 管理端：除了登录请求，其他请求必须进行jwt令牌校验
32         registry.addInterceptor(jwtTokenAdminInterceptor)
33             .addPathPatterns("/admin/**")
34             .excludePathPatterns("/admin/employee/login");
35
36         // 用户端：除了登录和查看店铺营业状态这两个请求，其他请求必须进行jwt令牌
37         // 校验
```

```
31     registry.addInterceptor(jwtTokenUserInterceptor)
32             .addPathPatterns("/user/**")
33             .excludePathPatterns(new String[]{"/user/user/login",
34         "/user/shop/status"}));
35 }
```

7. 编写登录相关接口

7.1 EmployeeController

```
1 /**
2  * 员工管理模块
3 */
4 @RestController
5 @RequestMapping("/admin/employee")
6 public class EmployeeController {
7
8     @Autowired
9     private EmployeeService employeeService;
10    @Autowired
11    private JwtProperties jwtProperties;
12
13    // 员工登录
14    @PostMapping("/login")
15    public Result<EmployeeLoginVO> login(@RequestBody EmployeeLoginDTO
employeeLoginDTO) {
16
17        Employee employee = employeeService.login(employeeLoginDTO);
18
19        // 登录成功后，生成jwt令牌
20        Map<String, Object> claims = new HashMap<>();
21        claims.put(JwtClaimsConstant.EMP_ID, employee.getId());
22        String token = JwtUtil.createJwt(
23                jwtProperties.getAdminSecretKey(),
24                jwtProperties.getAdminTtl(),
25                claims);
26    }
```

```
27     // 构建EmployeeLoginVO对象
28     EmployeeLoginVO employeeLoginVO = EmployeeLoginVO.builder()
29         .id(employee.getId())
30         .userName(employee.getUsername())
31         .name(employee.getName())
32         .token(token)
33         .build();
34     return Result.success(employeeLoginVO);
35 }
36 }
```

7.2 UserController

```
1 /**
2  * 用户端-用户接口
3 */
4 @RestController
5 @RequestMapping("/user/user")
6 public class UserController {
7
8     @Autowired
9     private UserService userService;
10    @Autowired
11    private JwtProperties jwtProperties;
12
13    // 微信用户登录
14    @PostMapping("/login")
15    public Result<UserLoginVO> wxLogin(@RequestBody UserLoginDTO
userLoginDTO) {
16
17        User user = userService.wxLogin(userLoginDTO);
18
19        // 登录成功后，生成jwt令牌
20        Map<String, Object> claims = new HashMap<>();
21        claims.put(JwtClaimsConstant.USER_ID, user.getId());
22        String token = JwtUtil.createJwt(
23            jwtProperties.getUserSecretKey(),
24            jwtProperties.getUserTtl(),
25            claims
26        );
27    }
```

```
28 // 构建UserLoginVO对象
29 UserLoginVO userLoginVO = UserLoginVO.builder()
30     .id(user.getId())
31     .openid(user.getOpenid())
32     .token(token)
33     .build();
34 return Result.success(userLoginVO);
35 }
36 }
```