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
> - https://www.cnblogs.com/mysticbinary/p/19050592

一、技术介绍

1.概述

- Token：是一个字符串，包含了用户信息和一些加密信息，常用于验证用户的身份和权限。Token的使用可以避免在每次请求时都需要进行用户身份验证而降低Web应用程序的性能，同时解决用户每次访问都需要登录的问题。
- 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）

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

- 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！

```
{
    "sub": "1234567890",
    "name": "john",
    "admin": "true"
}
```

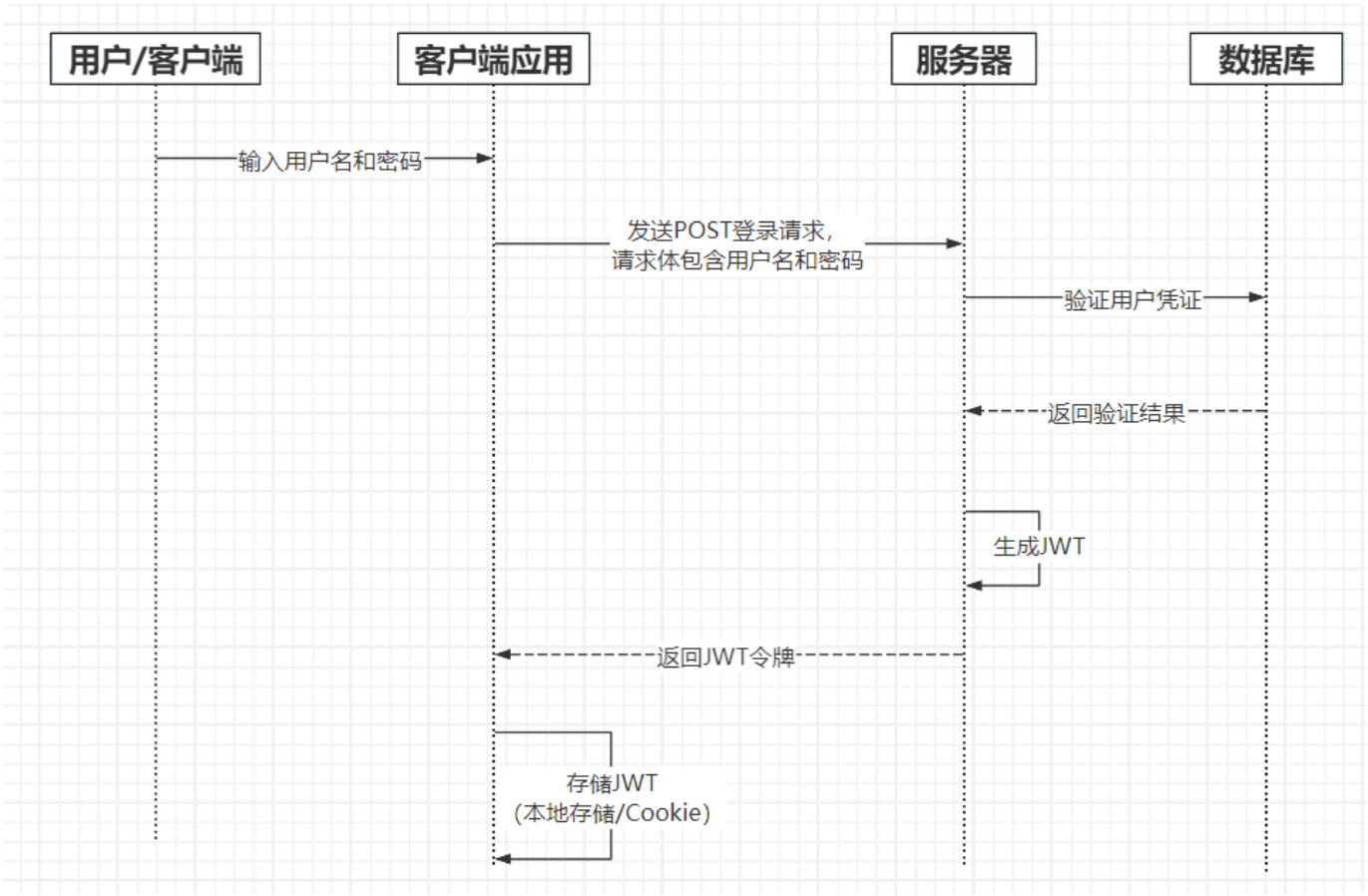
- Signature（签名：由头部、载荷和密钥共同生成，用于验证JWT的真实性和完整性，一般情况下也采用`Base64`编码）

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

4.JWT的工作时序图

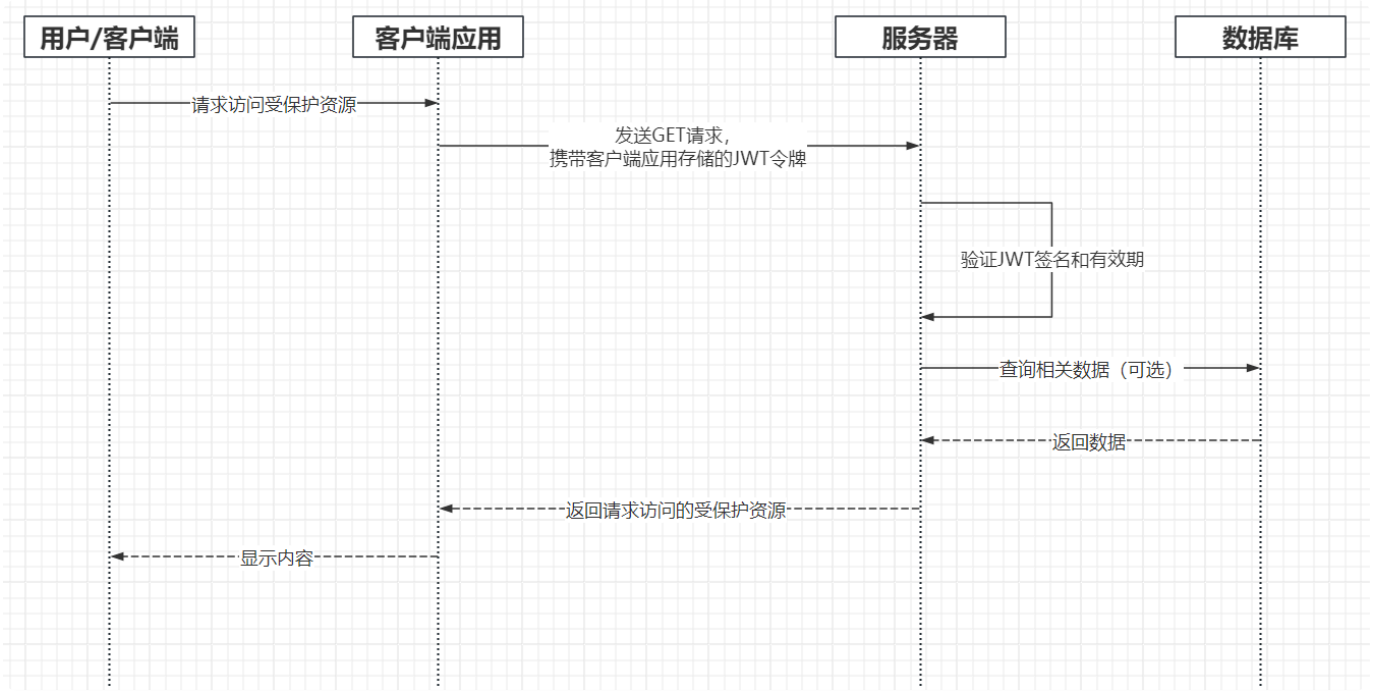
4.1 登录阶段

图片参考：https://www.cnblogs.com/mysticbinary/p/19050592



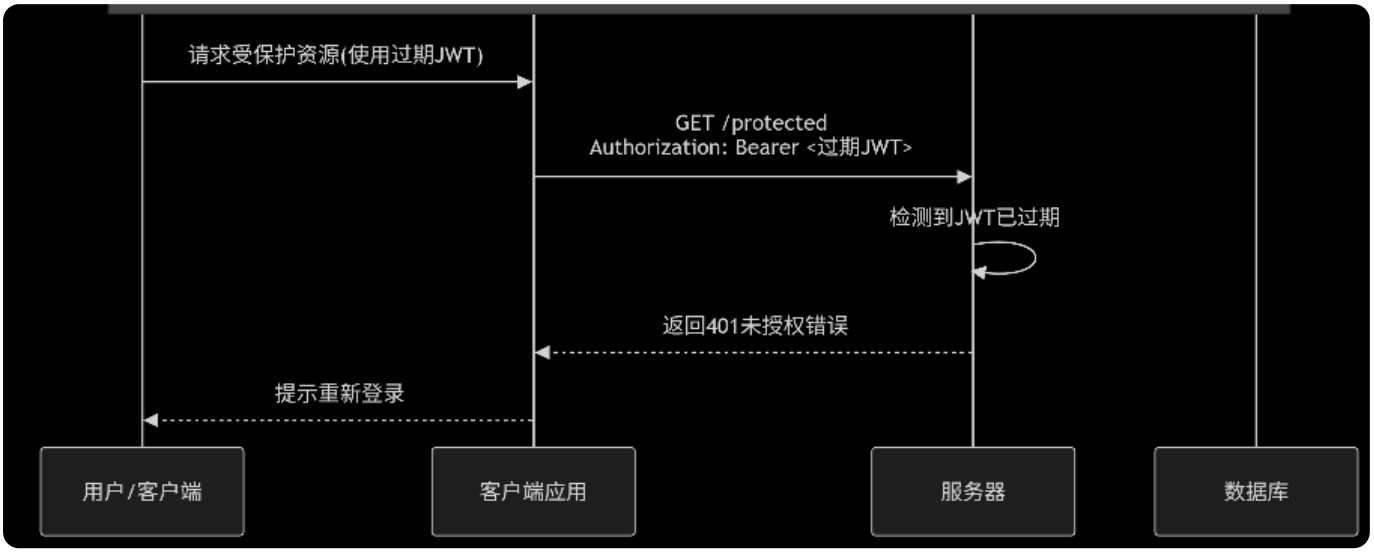
4.2 访问受保护资源

图片参考：https://www.cnblogs.com/mysticbinary/p/19050592



4.3 JWT令牌过期场景

图片来源：https://www.cnblogs.com/mysticbinary/p/19050592



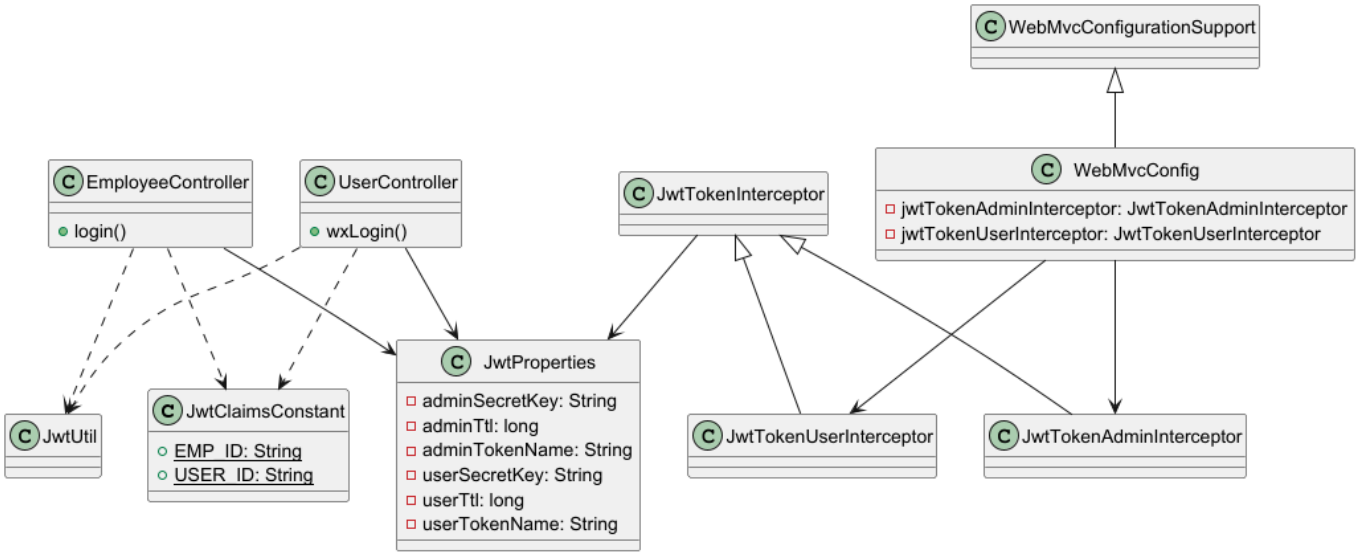
二、项目应用

涉及到的文件如下：

```
sky-take-out: pom.xml

sky-common:
    pom.xml
    src/main/java/com.sky:
        constant: JwtClaimsConstant
        properties: JwtProperties
        utils: JwtUtil

sky-server:
    pom.xml
    src/main/java/com.sky:
        config: WebMvcConfig
        controller:
            admin: EmployeeController
            user: UserController
        interceptor:
            JwtTokenInterceptor
            JwtTokenAdminInterceptor
            JwtTokenUserInterceptor
    src/main/resources: application.yml
```



1.导入JWT的`Maven`依赖坐标

1.1 `sky-take-out: pom.xml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>spring-boot-starter-parent</artifactId>
        <groupId>org.springframework.boot</groupId>
        <version>2.7.3</version>
    </parent>

    <groupId>com.sky</groupId>
    <artifactId>sky-take-out</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>

    <modules>
        <module>sky-common</module>
        <module>sky-pojo</module>
        <module>sky-server</module>
    </modules>

    <properties>
        <jjwt>0.9.1</jjwt>
        <jaxb-api>2.3.1</jaxb-api>
        <fastjson>1.2.76</fastjson>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt</artifactId>
                <version>${jjwt}</version>
            </dependency>
            <dependency>
                <groupId>javax.xml.bind</groupId>
                <artifactId>jaxb-api</artifactId>
```

```
                <version>${jaxb-api}</version>
            </dependency>
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>fastjson</artifactId>
                <version>${fastjson}</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

1.2 sky-common: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>sky-take-out</artifactId>
        <groupId>com.sky</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <modelVersion>4.0.0</modelVersion>
    <artifactId>sky-common</artifactId>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-json</artifactId>
        </dependency>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
        </dependency>
        <dependency>
```

```
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
        </dependency>
    </dependencies>
</project>
```

1.3 sky-server: pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <artifactId>sky-take-out</artifactId>
        <groupId>com.sky</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>

    <modelVersion>4.0.0</modelVersion>
    <artifactId>sky-server</artifactId>

    <dependencies>

        <dependency>
            <groupId>com.sky</groupId>
            <artifactId>sky-common</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.sky</groupId>
            <artifactId>sky-pojo</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>

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

```
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <scope>compile</scope>
        </dependency>

        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2.编写JWT工具类`JwtUtil`

```
/**
 * JWT工具类
 */
public class JwtUtil {

    /**
     * 生成jwt
```

```
     * 使用HS256算法, 私匙使用固定秘钥
     *
     * @param secretKey jwt秘钥
     * @param ttlMillis jwt过期时间(毫秒)
     * @param claims    设置的信息
     */
    public static String createJwt(String secretKey, long ttlMillis, Map<String, Object> claims) {

        // 指定签名的时候使用的签名算法是HS256
        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        // 生成jwt过期的时间点
        long expMillis = System.currentTimeMillis() + ttlMillis;
        Date exp = new Date(expMillis);

        // 设置jwt的body
        JwtBuilder builder = Jwts.builder()
                // 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builder的claims赋值，
                // 一旦写在标准声明赋值之后，就会覆盖掉那些标准声明
                .setClaims(claims)
                // 设置签名使用的签名算法和秘钥
                .signWith(signatureAlgorithm, secretKey.getBytes(StandardCharsets.UTF_8))
                // 设置过期时间
                .setExpiration(exp);

        return builder.compact();
    }

    /**
     * token解密
     *
     * @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sign就可以被伪造，如果对接多个客户端建议改造成多个
     * @param token     加密后的token
     * @return
     */
    public static Claims parseJwt(String secretKey, String token) {
        // 得到DefaultJwtParser
        Claims claims = Jwts.parser()
                // 设置签名的秘钥
```

```
47
    .setSigningKey(secretKey.getBytes(StandardCharsets.UTF_8))
48                  // 设置需要解析的jwt
49                  .parseClaimsJws(token).getBody();
50          return claims;
51      }
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;
```

```
    private String adminTokenName;

    // 用户端微信用户生成jwt令牌相关配置
    private String userSecretKey;
    private long userTtl;
    private String userTokenName;

}
```

4.2 application.yml

```
sky:
  jwt:
    # 设置jwt签名加密时使用的秘钥
    admin-secret-key: cszsh
    #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401!
    #Caution: 等项目上线后再改回来
    admin-ttl: 720000000
    # 设置前端传递过来的令牌名称
    admin-token-name: token
    user-secret-key: itzsh
    user-ttl: 720000000
    user-token-name: authentication
```

5.自定义拦截器

5.1 JwtTokenInterceptor

```
/**
 * jwt令牌校验的拦截器抽象父类（模板方法设计模式）
 */
@Slf4j
```

```java
public abstract class JwtTokenInterceptor implements HandlerInterceptor {

    @Autowired
    protected JwtProperties jwtProperties;

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        // 判断当前拦截到的是Controller的方法还是其他资源
        if (!(handler instanceof HandlerMethod)) {
            // 当前拦截到的不是动态方法，直接放行
            return true;
        }

        // 1.从请求头中获取令牌
        String token = getAndTokenFromHeader(request);

        // 2.校验令牌
        try {
            checkToken(token);
            // 3.通过，放行
            return true;
        } catch (Exception e) {
            // 4.不通过，响应401状态码
            response.setStatus(401);
            return false;
        }
    }

    // 钩子方法，由子类重写
    protected abstract String getAndTokenFromHeader(HttpServletRequest request);
    protected abstract void checkToken(String token);
}
```

5.2 `JwtTokenAdminInterceptor`

```java
/**
 * 管理端jwt令牌校验的拦截器
 */
```

```java
@Component
@Slf4j
public class JwtTokenAdminInterceptor extends JwtTokenInterceptor {

    // 从请求头中获取管理员令牌
    @Override
    protected String getAndTokenFromHeader(HttpServletRequest request)
{
        return request.getHeader(jwtProperties.getAdminTokenName());
    }

    // 校验管理员令牌，并将empId存入ThreadLocal中
    @Override
    protected void checkToken(String token) {
        Claims claims =
JwtUtil.parseJwt(jwtProperties.getAdminSecretKey(), token);
        Long empId =
Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());
        BaseContext.setCurrentId(empId);
    }
}
```

5.3 `JwtTokenUserInterceptor`

```java
/**
 * 用户端jwt令牌校验的拦截器
 */
@Component
@Slf4j
public class JwtTokenUserInterceptor extends JwtTokenInterceptor {

    // 从请求头中获取用户令牌
    @Override
    protected String getAndTokenFromHeader(HttpServletRequest request)
{
        return request.getHeader(jwtProperties.getUserTokenName());
    }

    // 校验用户令牌，并将userId存入ThreadLocal中
    @Override
    protected void checkToken(String token) {
```

```
        Claims claims = JwtUtil.parseJwt(jwtProperties.getUserSecretKey(), token);
        Long userId = Long.valueOf(claims.get(JwtClaimsConstant.USER_ID).toString());
        // BaseContext内部使用了ThreadLocal技术，它是线程隔离的
        // 不会与BaseContext.setCurrentId(empId);发生冲突
        BaseContext.setCurrentId(userId);
    }
}
```

6.在`WebMvcConfig`中将自定义拦截器添加到拦截器类中

```
/**
 * 配置类，注册web层相关组件
 */
@Configuration
@Slf4j
public class WebMvcConfig extends WebMvcConfigurationSupport {

    @Autowired
    private JwtTokenAdminInterceptor jwtTokenAdminInterceptor;
    @Autowired
    private JwtTokenUserInterceptor jwtTokenUserInterceptor;

    // 设置静态资源映射
    @Override
    protected void addResourceHandlers(ResourceHandlerRegistry registry) {
        log.info("开始设置静态资源映射...");

 registry.addResourceHandler("/doc.html").addResourceLocations("classpath:/META-INF/resources/");

 registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-INF/resources/webjars/");
    }


```

```
    // 注册自定义拦截器
    @Override
    protected void addInterceptors(InterceptorRegistry registry) {
        log.info("开始注册自定义拦截器...");
        // 管理端：除了登录请求，其他请求必须进行jwt令牌校验
        registry.addInterceptor(jwtTokenAdminInterceptor)
                .addPathPatterns("/admin/**")
                .excludePathPatterns("/admin/employee/login");
        // 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令牌校验
        registry.addInterceptor(jwtTokenUserInterceptor)
                .addPathPatterns("/user/**")
                .excludePathPatterns(new String[]{"/user/user/login","/user/shop/status"});
    }
}
```

7.编写登录相关接口

7.1 `EmployeeController`

```
/**
 * 员工管理模块
 */
@RestController
@RequestMapping("/admin/employee")
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;
    @Autowired
    private JwtProperties jwtProperties;

    // 员工登录
    @PostMapping("/login")
    public Result<EmployeeLoginVO> login(@RequestBody EmployeeLoginDTO employeeLoginDTO) {

```

```
        Employee employee = employeeService.login(employeeLoginDTO);

        // 登录成功后，生成jwt令牌
        Map<String, Object> claims = new HashMap<>();
        claims.put(JwtClaimsConstant.EMP_ID, employee.getId());
        String token = JwtUtil.createJwt(
                jwtProperties.getAdminSecretKey(),
                jwtProperties.getAdminTtl(),
                claims);

        // 构建EmployeeLoginVO对象
        EmployeeLoginVO employeeLoginVO = EmployeeLoginVO.builder()
                .id(employee.getId())
                .userName(employee.getUsername())
                .name(employee.getName())
                .token(token)
                .build();
        return Result.success(employeeLoginVO);
    }
}
```

7.2 UserController

```
/**
 * 用户端-用户接口
 */
@RestController
@RequestMapping("/user/user")
public class UserController {

    @Autowired
    private UserService userService;
    @Autowired
    private JwtProperties jwtProperties;

    // 微信用户登录
    @PostMapping("/login")
    public Result<UserLoginVO> wxLogin(@RequestBody UserLoginDTO
userLoginDTO) {

        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  }
```