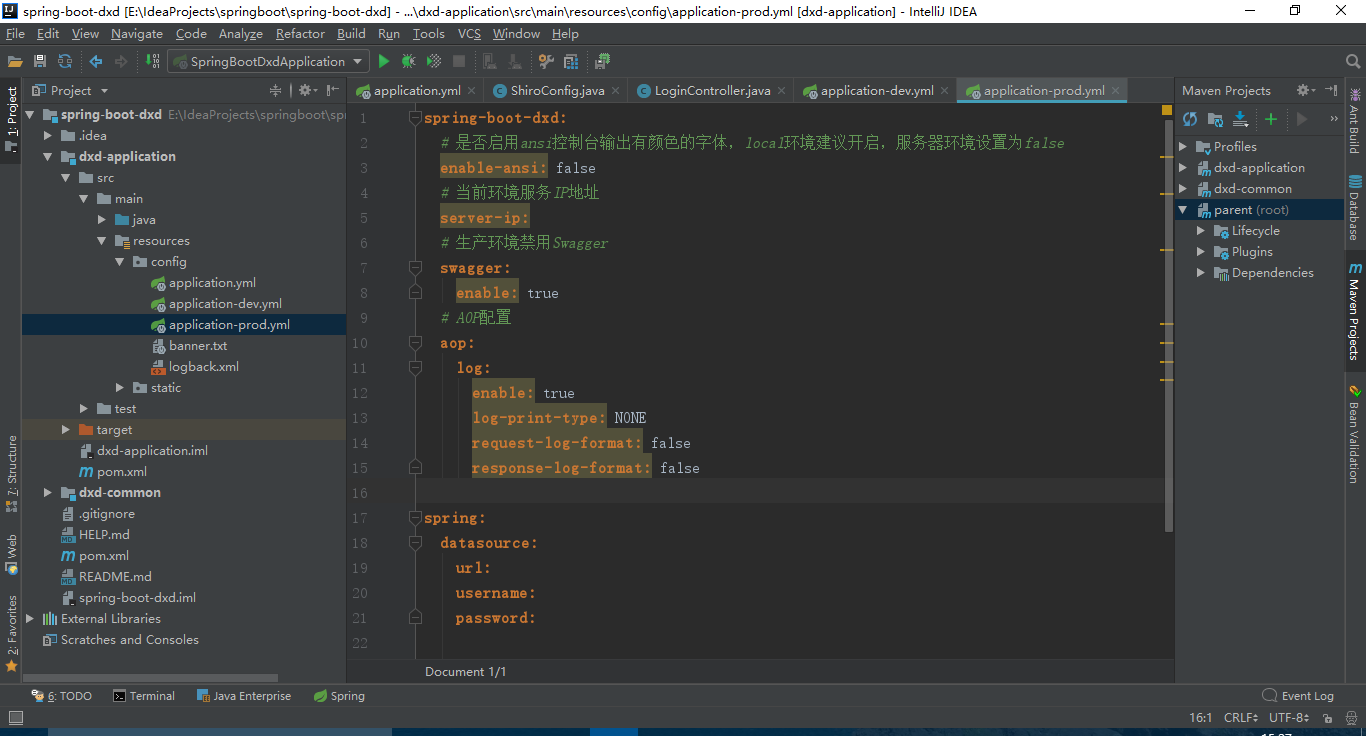
## 项目架构

### maven多模块

|  |  |
| --- | --- |
| **dxd-application** | 应用启动类及相关配置文件 |
| **dxd-common** | 应用通用模块，系统框架相关 |
| **dxd-business** | 业务处理模块，具体业务开发 |

### dxd-application说明

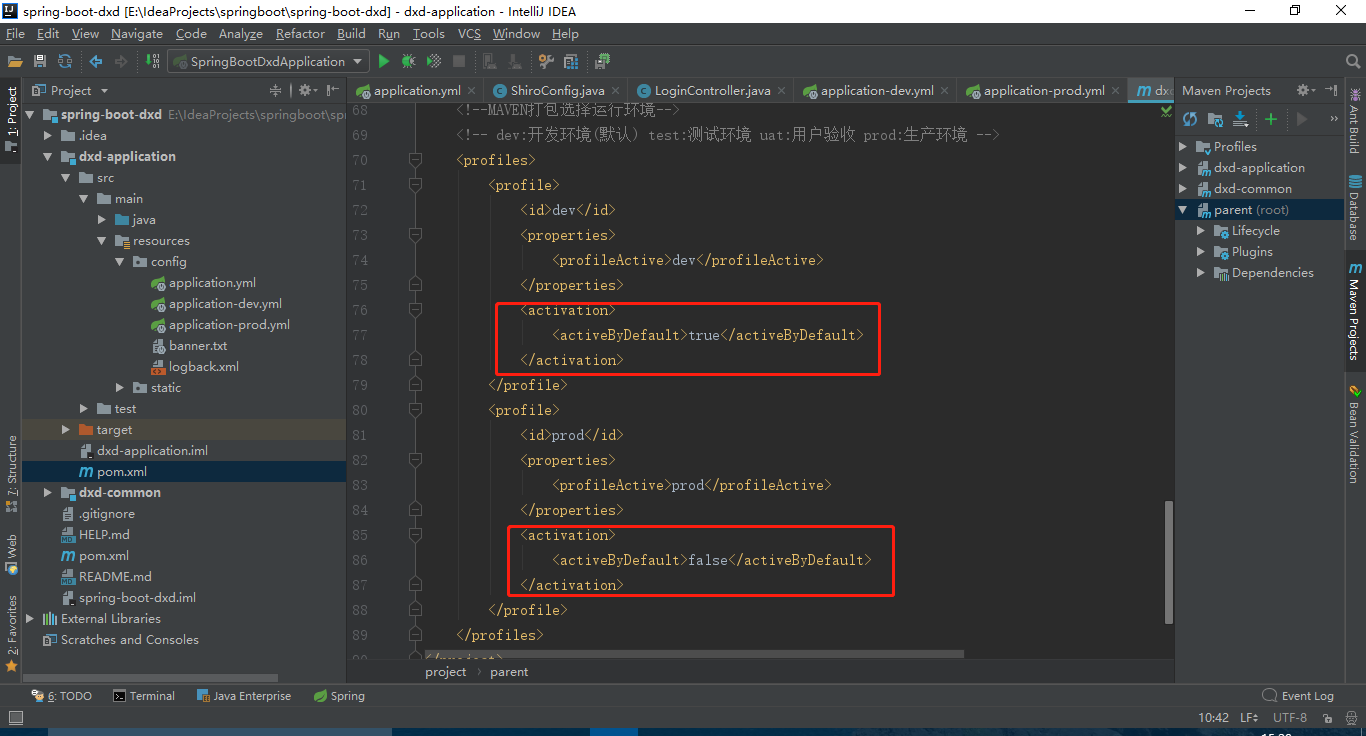


多环境配置文件:

application-dev.yml 开发环境

application-prod.yml 生产环境

可在pom.xml中修改默认激活的运行环境。



### dxd-common模块说明

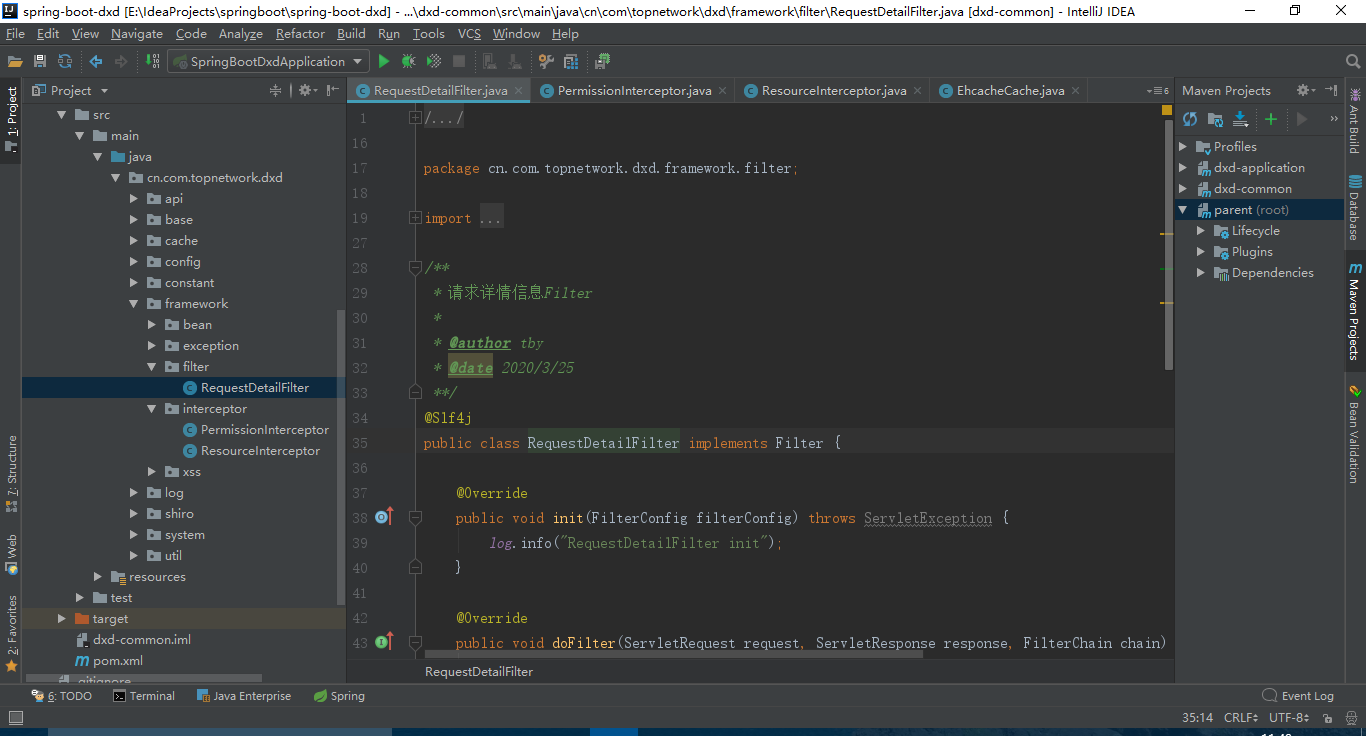
|  |  |
| --- | --- |
| **api** | api响应码 |
| **base** | 公共接口或抽象类 |
| **cache** | 缓存模块 |
| **config** | 配置模块 |
| **constant** | 常量类模块 |
| **framework** | 框架核心模块，包含过滤器，拦截器，异常处理等 |
| **log** | AOP日志模块 |
| **shiro** | jwt+shiro权限框架模块 |
| **system** | 数据库中sys表相关的操作模块，loginController存在这里 |
| **util** | 工具类模块 |

## 2.自定义filter和Interceptor

### 2.1 自定义filter

自定义filter类，实现Filter接口。

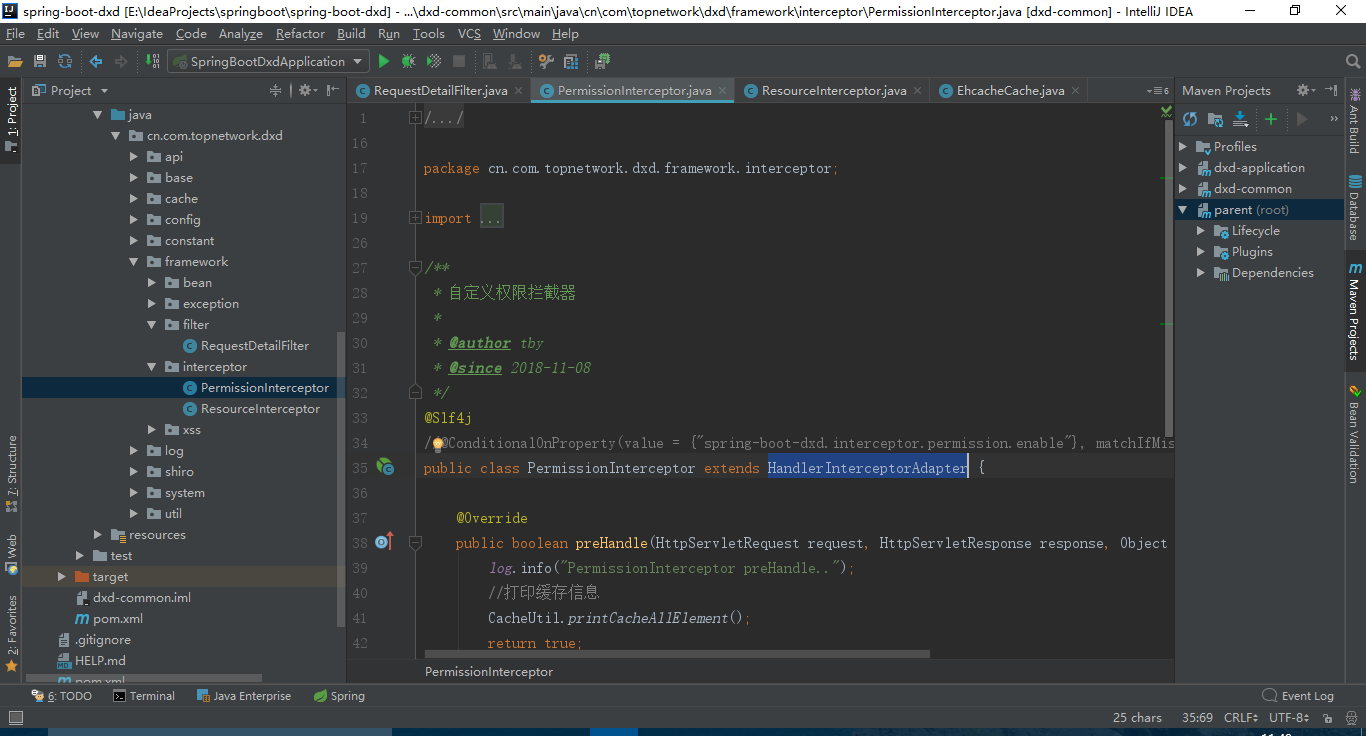
参考：cn.com.topnetwork.dxd.framework.filter包下类



### 2.2 自定义interceptor

自定义interceptor类，继承HandlerInterceptorAdapter。

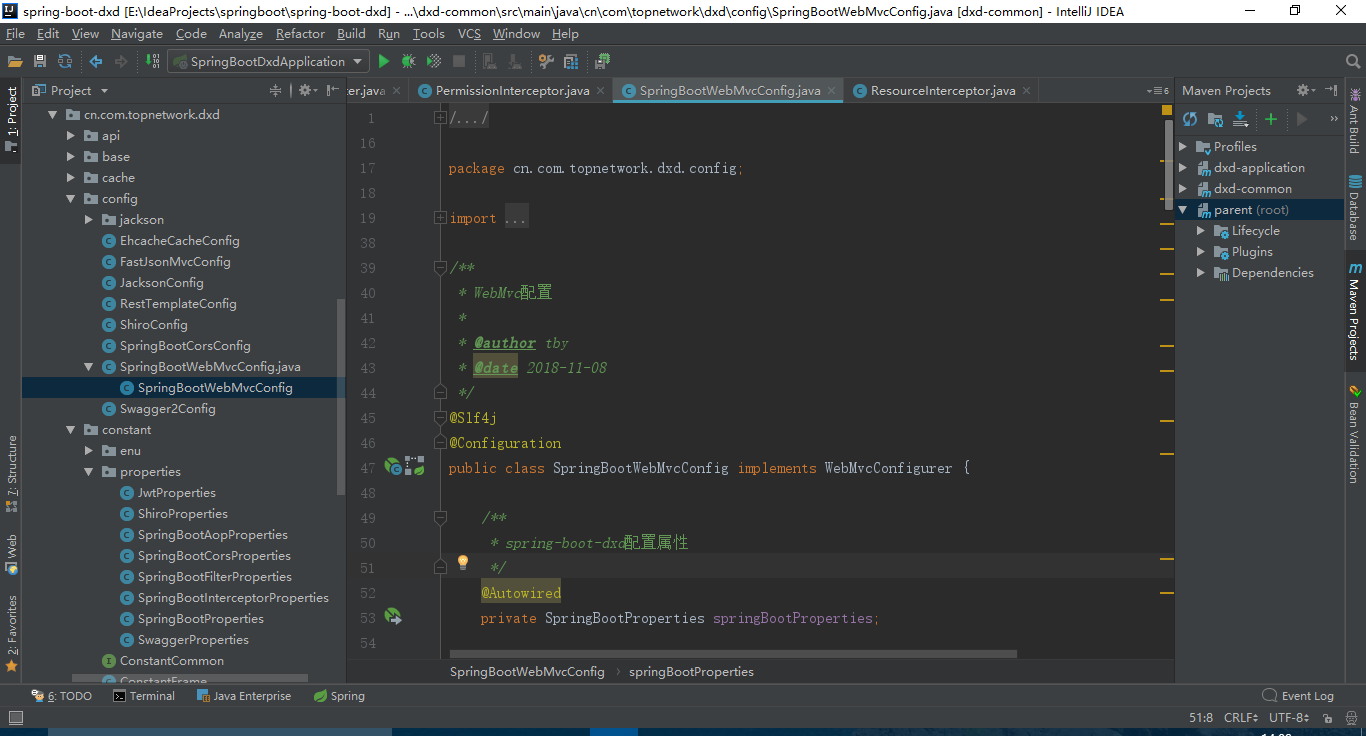
参考：cn.com.topnetwork.dxd.framework.interceptor包下类

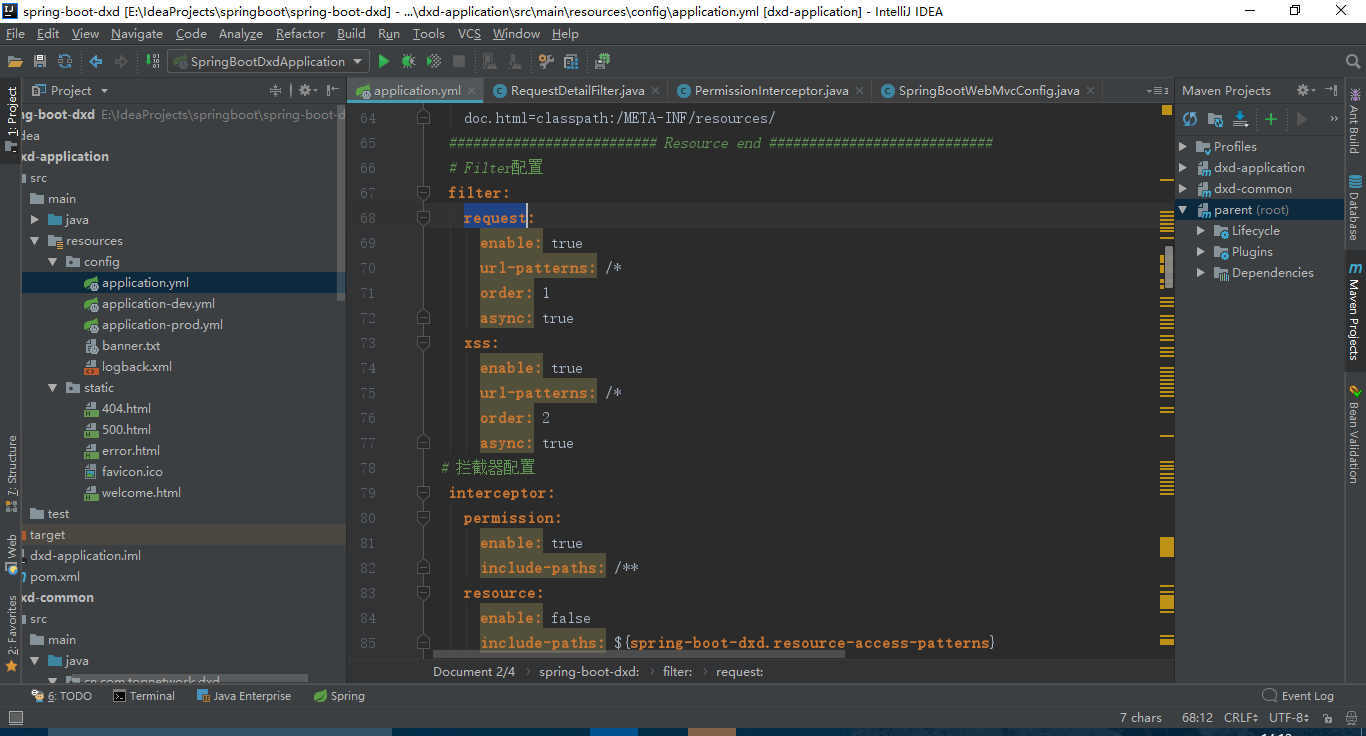


### 2.3 配置类配置

在配置类SpringBootWebMvcConfig中添加自定义的filter和interceptor类。同时将配置信息配置在application.yml文件中。

参考：cn.com.topnetwork.dxd.config.SpringBootWebMvcConfig





## 3.自定义缓存

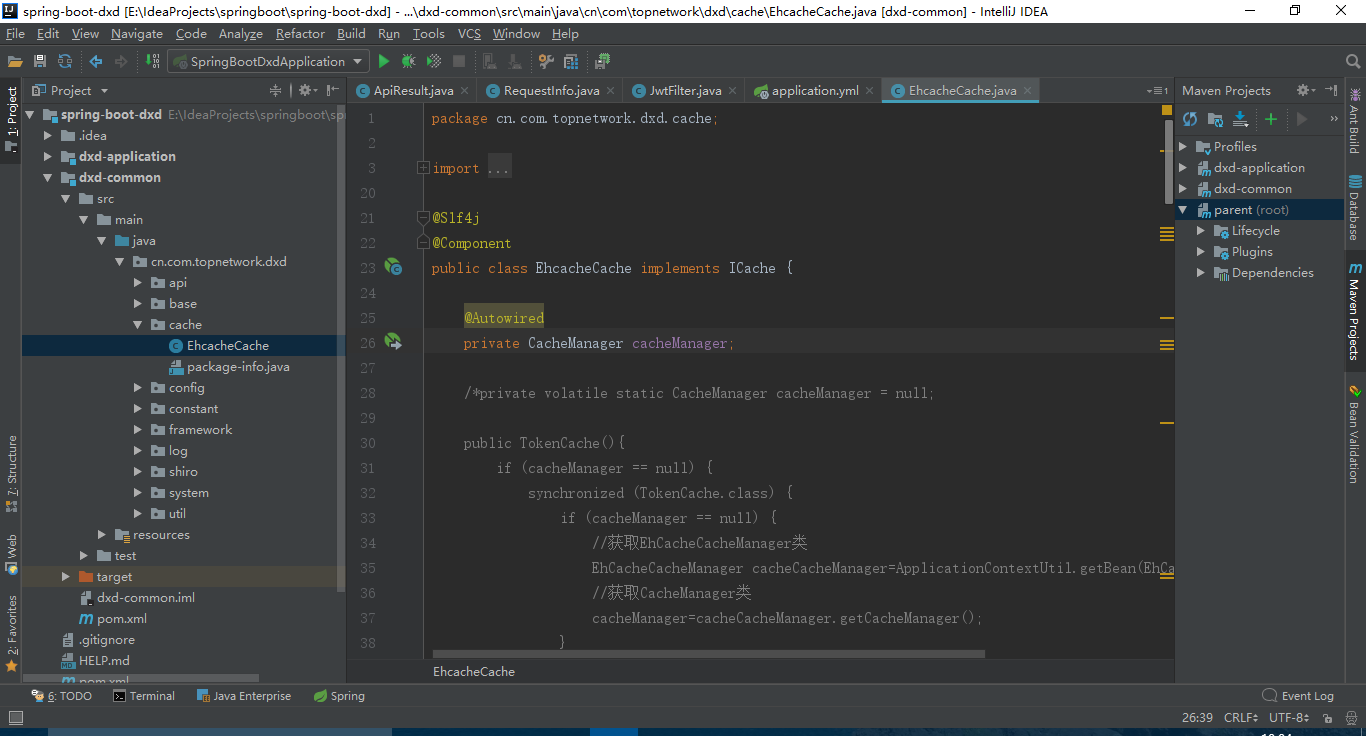
默认集成Ehcache缓存，可自定义扩展。扩展步骤：

### 3.1 自定义缓存类

自定义缓存类，实现Icache接口。该类由spring容器管理，添加@component注解。

参考：cn.com.topnetwork.dxd.cache.EhcacheCache

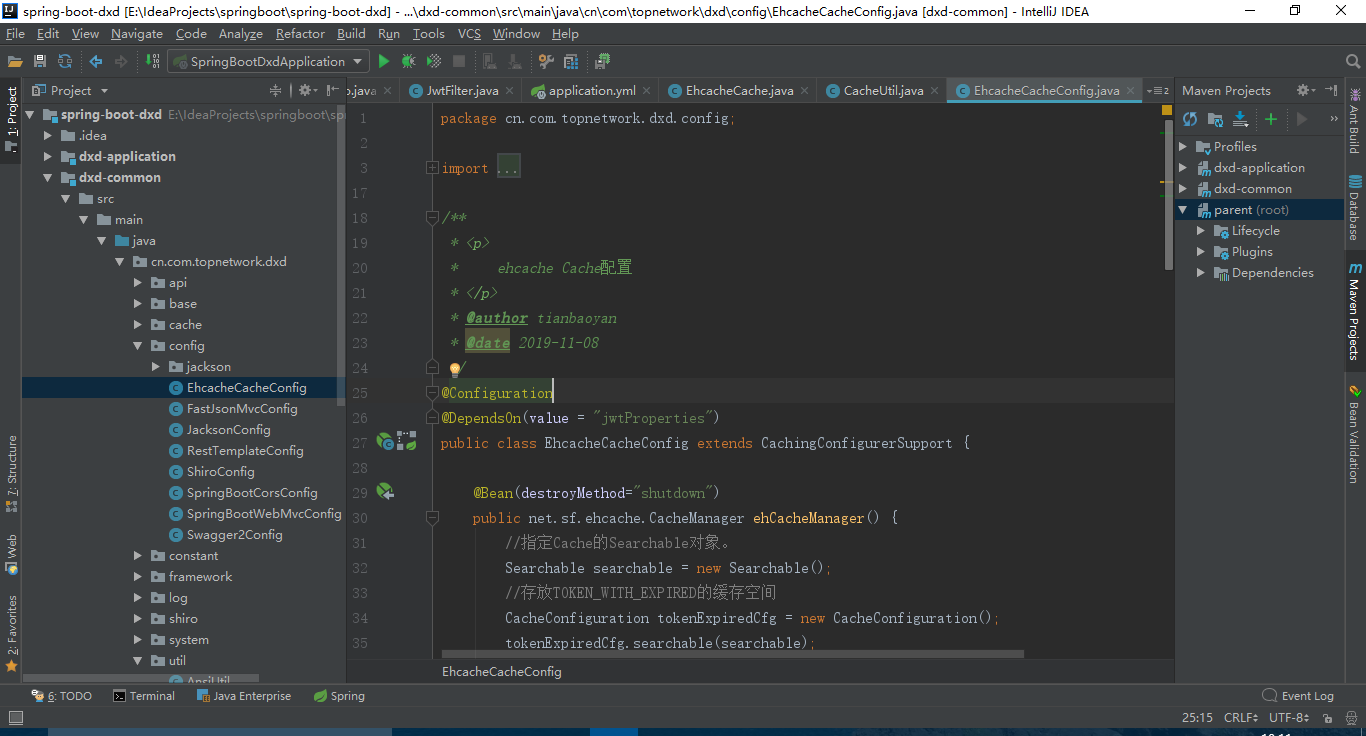
操作工具类：cn.com.topnetwork.dxd.util.CacheUtil



### 3.2 配置类

自定义配置类,继承CachingConfigurerSupport

参考：cn.com.topnetwork.dxd.config.EhcacheCacheConfig

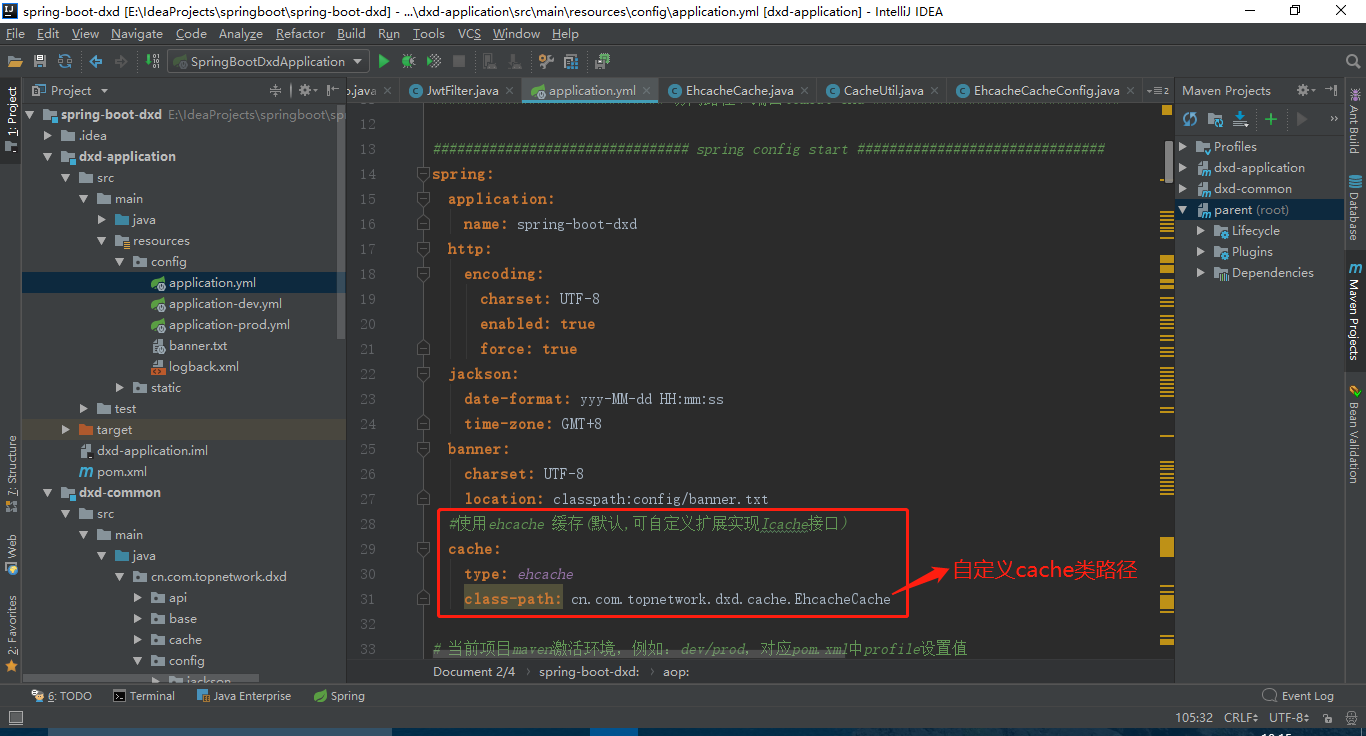


### 3.3 配置文件配置

application.yml中修改cache相关配置

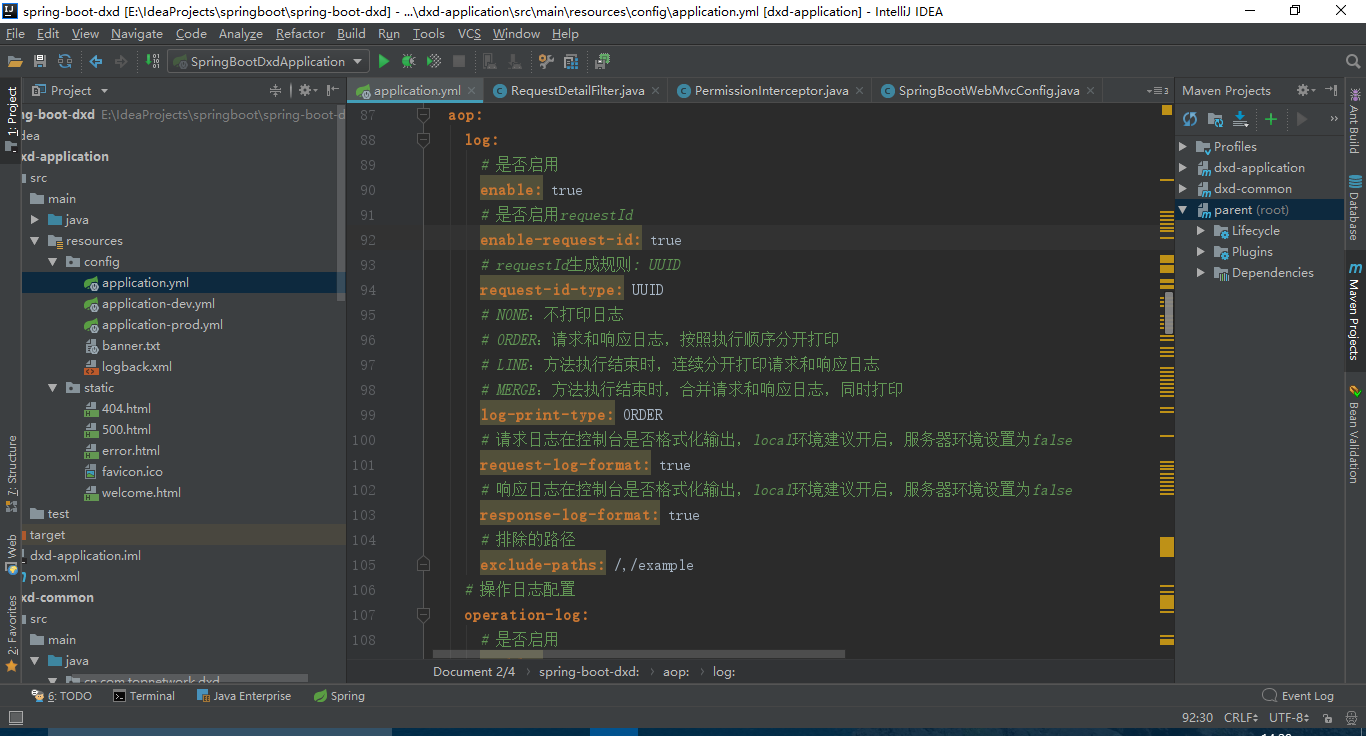
cache.type>>缓存类型

cache.class-path>>自定义缓存类路径



## Aop日志配置

### 配置文件



**log：日志Aop配置，目标：LogAop.java，打印请求响应信息。如果禁用，将不用打印日志，也不会记录操作和登录日志**

aop：切面配置，对应SpringBootAopProperties.java属性配置类

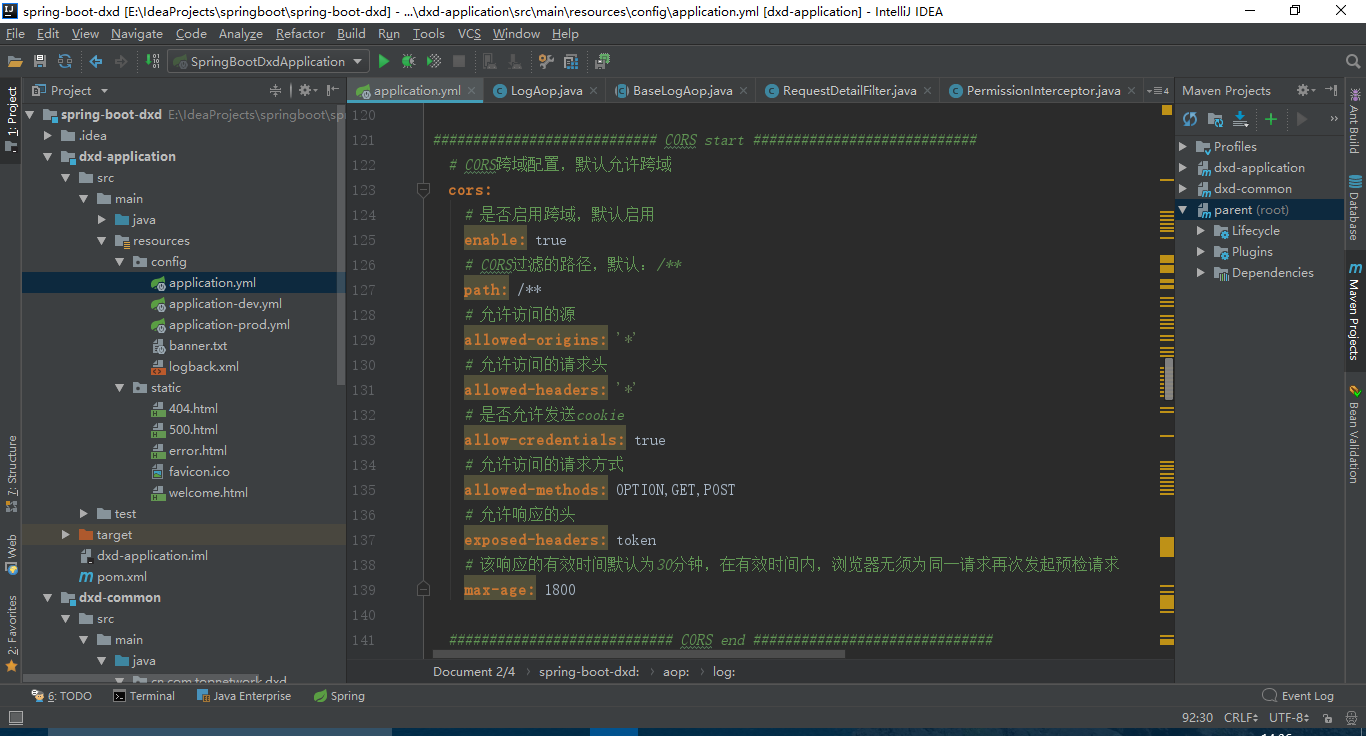
enable：启用禁用，如果为false，将不启用aop切面日志

operation-log：操作日志配置，默认记录@OperationLog注解的方法，可自定义排除路径，也可使用@OperationLogIgnore注解标注在类或方法上排除，操作记录会保存到sys\_operation\_log

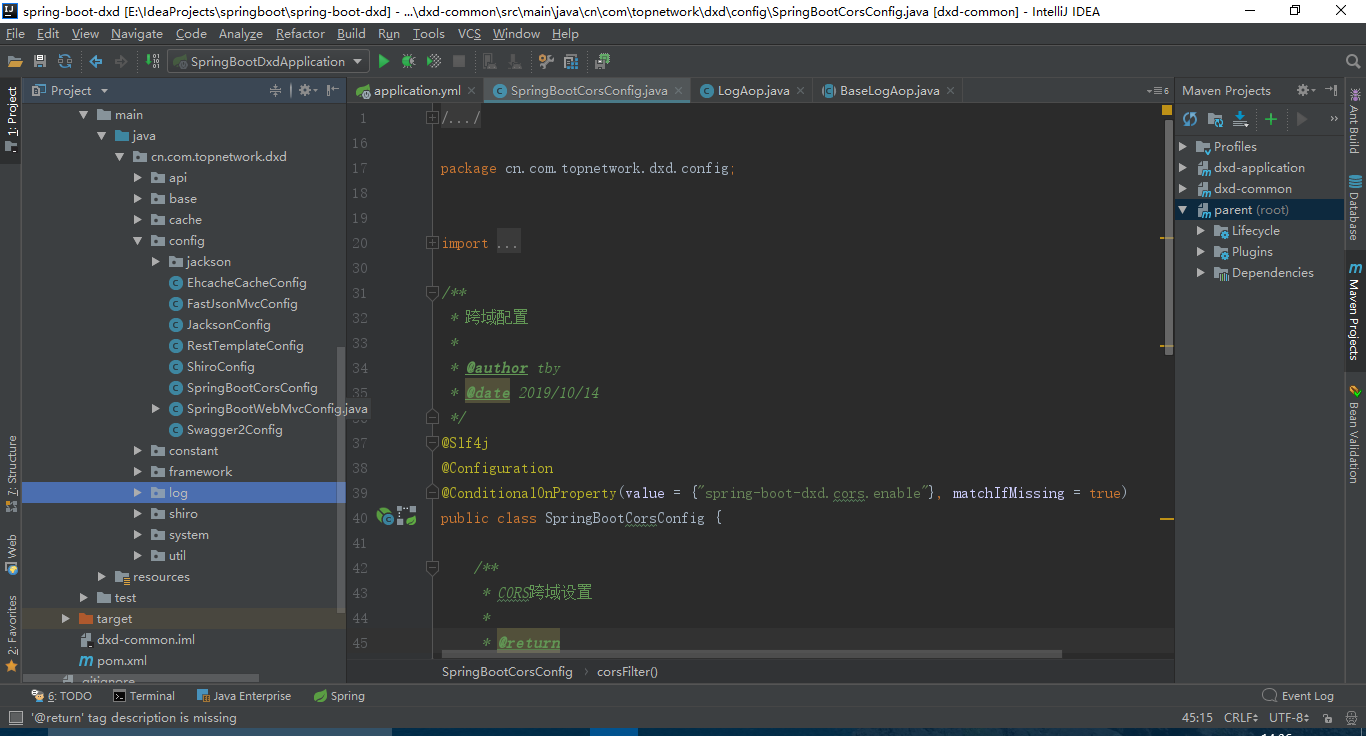
login-log：登录日志配置，只会保存登录登出日志，需指定登录登出地址，登录日志保存到sys\_login\_log

## Cors跨域配置

### 配置文件

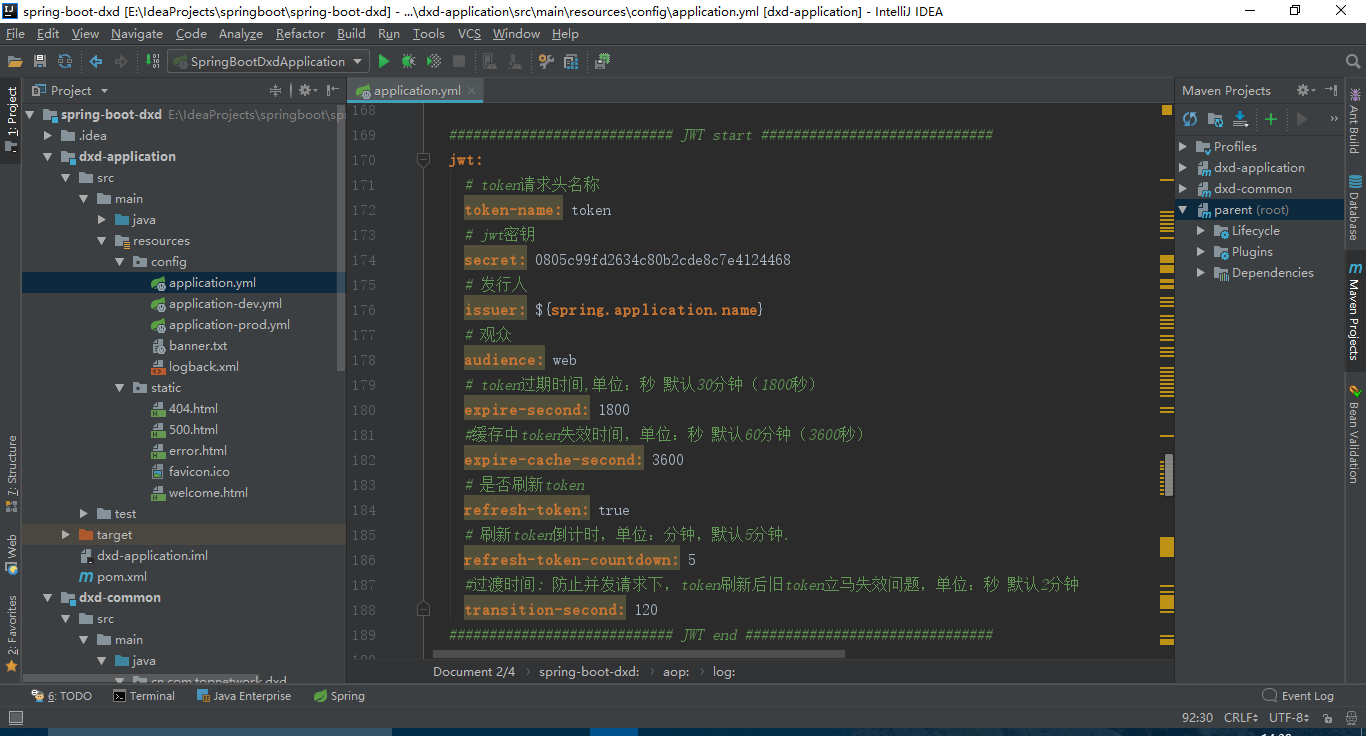


### 配置类



## JwtToken

### 6.1配置文件



### 6.2主要配置参数说明

|  |  |
| --- | --- |
| spring-boot-dxd.jwt | |
| expire-second | 生成token字符串中的有效时间，默认30分钟 |
| expire-cache-second | 缓存中token的有效时间，默认60分钟 |
| refresh-token-countdown | 刷新token倒计时，默认5分钟 |
| transition-second | 失效token的过渡时间，主要防止刷新token时存在并发请求，后续token重复刷新问题，默认2分钟。 |

### 6.3根据默认参数时间点流程

(1)0~25分钟之内访问后台应用，token不变化。

(2)25分钟之后,使用原token访问后台应用，token会产生变化。

25~60分钟之内,访问过后台应用，则会刷新token。

25~60分钟之内,没有访问过后台应用，则原token会过期，需要重新登陆获取token。

假设第30分钟访问：①第30分钟产生新token

②30~32分钟，旧token仍有效(处理并发请求，重复刷新问题)

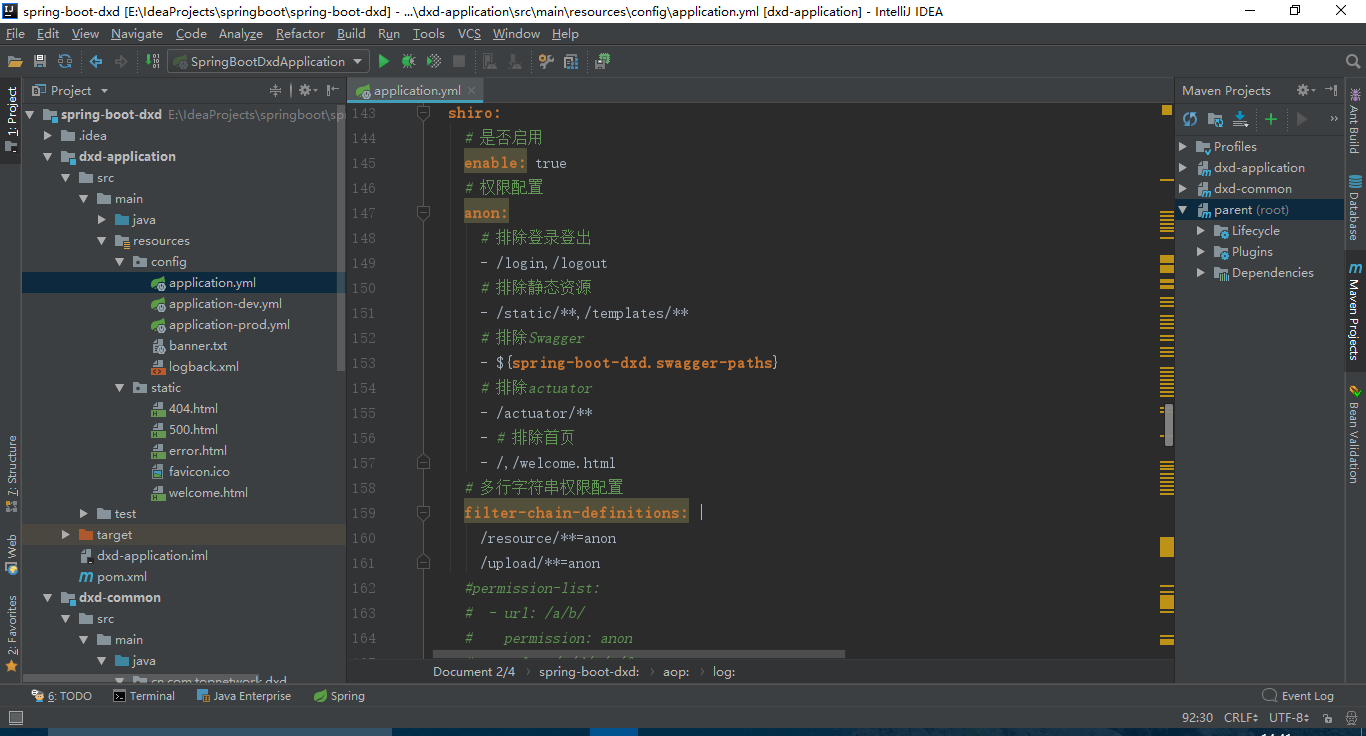
③32分钟之后，旧token无效，只能使用新token访问

1. 产生新token会存放在response header 中，前端可通过判断response header中是否携带token值，判断token是否进行过刷新。

**注意：支持用户并发访问,即同一个用户可同时产生多个token并发访问系统。Token缓存中按照>>用户名+token值 生成缓存Key值。**

## Shiro权限控制

### 7.1配置文件



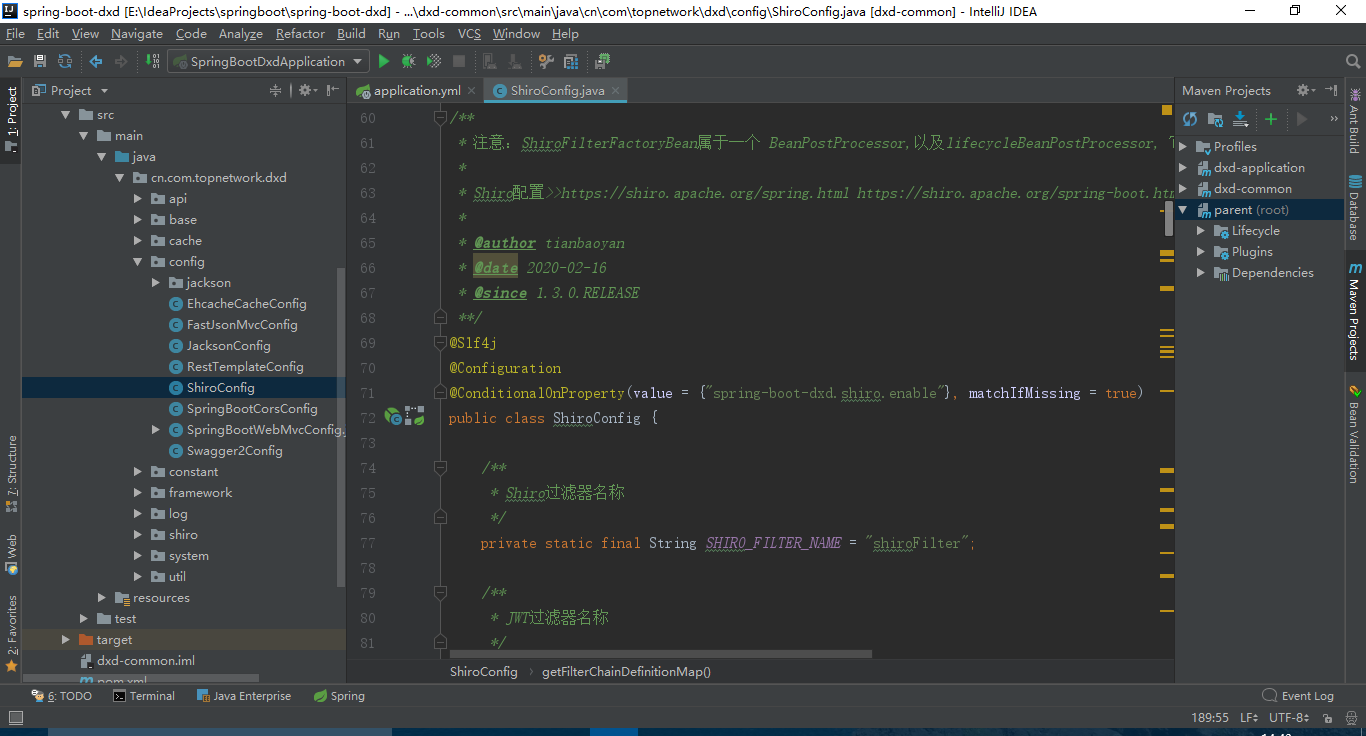
**注意: enable：设置为false时，不会加载shiro配置，后台不会拦截访问请求。**

shiro：Shiro权限配置，对应ShiroProperties.java属性配置类，目标配置类：ShiroConfig

anon：配置不需要权限校验的路径，接受集合数组对象，每行可写单个或多个路径，使用逗号隔开

filter-chain-definitions：路径权限配置，值为anon时：不需要校验，等同于上面的anon配置，也可以指定其它权限过滤器名称

### 7.2 配置类



### 7.3 主要类及配置说明

Ream 主要包含两个，DbReam和JwtReam

DbReam>>前端调用/login 接口获取token时，根据传入的用户名+密码从数据库中进行认证，认证通过，则产生token,同时按照[用户名+token]为key值 存放在后端token缓存中,同时将用户登陆信息(包含权限相关)按照[用户名]为key值存放在后端user缓存中,然后将token存放在response header中返回给前端。

JwtReam>>前端对后端的接口调用，都需要在request header中携带token值方可操作。[原理通过过滤器，详细参考类JwtFilter]。通过[用户名+token](其中用户名通过token获取)从后端token缓存中进行校验认证，认证通过则可调用接口。同时从user缓存中，获取登陆用户信息（主要是权限信息），从而可以通过shiro对操作接口的权限控制（注解或编码方式）。

**注意：如果用户更新了权限信息，则需要重新登陆获取token方可生效。因为只有调用/login接口，才是从数据库获取权限信息，然后存放缓存，后续权限信息从缓存中获取。**

### 7.4 shiro权限注解

可在方法上添加@RequiresRoles,@RequiresPermissions等注解实现对接口的权限控制。

更多参考：<https://www.w3cschool.cn/shiro/>

## Swagger/knife4j

Swagger2 路径：

[http://[ip:port/context-path]/swagger-ui.html](http://47.105.159.10:8888/api/swagger-ui.html)

Knife4j路径：

[http://[ip:port/context-path]/doc.html#/home](http://47.105.159.10:8888/api/doc.html" \l "/home)

用户名/密码： admin/admin

## 与前端集成

### token获取

token获取后端接口：[context-path]/login

建议集成在用户登陆操作方法中。前端登陆执行后,开启新线程调用token获取接口，获取到token后，将token存放在前端缓存中。

### 接口操作

建议编写rest请求工具类，（resttemplate）工具类主要包含以下要点：

1. 统一从前端缓存中获取当前用户的token值，将其放入 requrest header中。
2. 统一使用该工具类调用后端接口
3. 统一处理response请求，判断其header中是否存在token属性，如果存在，说明该token已经自动延期刷新，然后将新token值替换到用户的前端缓存中。

### token失效(手动)

token失效后端接口：[context-path]/logout

建议集成在用户退出操作方法中。前端登出执行后,开启新线程调用token失效接口，则该token会从后端框架缓存中删除，从而失效。