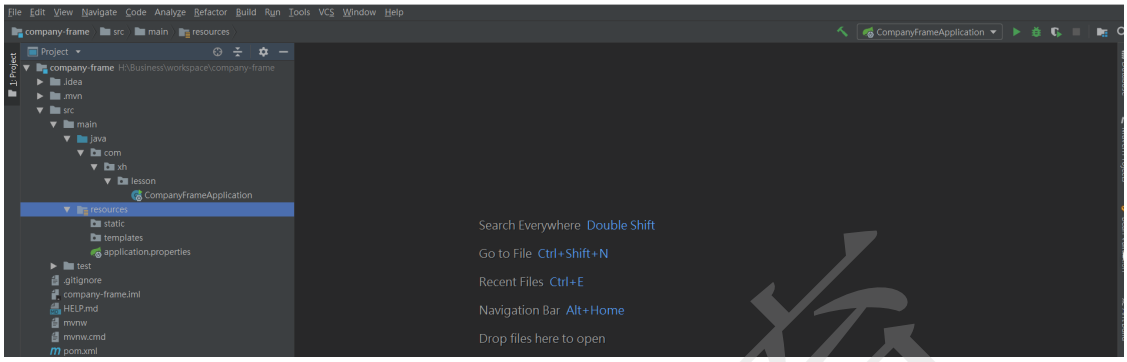
手把手 Spring Boot+redis+JWT+shiro+mybatis 3+swagger2 前后端分离实战脚手架

1.搭建项目

1.1 快速创建 Spring Boot 项目

- 目录如下



- pom 文件如下

```
<?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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.7.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.xh.lesson</groupId>
    <artifactId>company-frame</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>company-frame</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>

    </properties>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

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

</project>
```

1.2 加入依赖

1.2.1 数据库相关

```
        <!--数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--数据源-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.10</version>
        </dependency>
```

1.2.2 redis 相关

```
        <!--redis 依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
```

1.2.3 swagger2 相关

```
        <!--swagger2 依赖-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>
```

1.2.4 shiro 相关

```
        <!--shiro 依赖-->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.1</version>
        </dependency>
```

1.2.5 分页相关

```
        <!--分页依赖-->
        <dependency>
            <groupId>com.github.jsqlparser</groupId>
            <artifactId>jsqlparser</artifactId>
            <version>1.0</version>
        </dependency>
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper-spring-boot-starter</artifactId>
            <version>1.2.5</version>
        </dependency>
```

1.2.6 fastJson&lombok

```
        <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <!--fastJson-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.49</version>
        </dependency>
```

1.2.7 JWT 相关

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

JJWT是一个提供端到端的JWT创建和验证的Java库。永远免费和开源(Apache License，版本2.0)，JJWT很容易使用和理解

1.2.8 aop 相关

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

1.2.9 热部署相关

```
      <!--热部署工具包-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
        <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

完整的 pom

```
<?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 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.6.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.yingxue.lesson</groupId>
    <artifactId>company-frame</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```
    <name>company-frame</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

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

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <!--数据库驱动-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.47</version>
        </dependency>
        <!--数据源-->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.10</version>
        </dependency>
        <!--redis 依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
        <!--swagger2 依赖-->
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.9.2</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.9.2</version>
        </dependency>
        <!--shiro 依赖-->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.4.1</version>
        </dependency>
        <!--分页依赖-->
        <dependency>
            <groupId>com.github.jsqlparser</groupId>
            <artifactId>jsqlparser</artifactId>
            <version>1.0</version>
        </dependency>
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper-spring-boot-starter</artifactId>
            <version>1.2.5</version>
        </dependency>
        <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
        <!--fastJson-->
        <dependency>
```

```
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.49</version>
        </dependency>
        <!--JWT-->
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
        <!--Aop 支持-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>
        <!--热部署工具包-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-devtools</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <fork>true</fork>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

1.3 数据脚本

1.3.1创建 company_frame 数据库

```
CREATE DATABASE IF NOT EXISTS company_frame DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
```

1.3.2 创建 sys_user 用户表结构

```
DROP TABLE IF EXISTS `sys_user`;
CREATE TABLE `sys_user` (
  `id` varchar(64) NOT NULL COMMENT '用户id',
  `username` varchar(50) NOT NULL COMMENT '账户名称',
  `salt` varchar(20) DEFAULT NULL COMMENT '加密盐值',
  `password` varchar(200) NOT NULL COMMENT '用户密码密文',
  `phone` varchar(20) DEFAULT NULL COMMENT '手机号码',
  `dept_id` varchar(64) DEFAULT NULL COMMENT '部门id',
  `real_name` varchar(60) DEFAULT NULL COMMENT '真实名称',
  `nick_name` varchar(60) DEFAULT NULL COMMENT '昵称',
  `email` varchar(50) DEFAULT NULL COMMENT '邮箱(唯一)',
  `status` tinyint(4) DEFAULT '1' COMMENT '账户状态(1.正常 2.锁定 )',
  `sex` tinyint(4) DEFAULT '1' COMMENT '性别(1.男 2.女)',
  `deleted` tinyint(4) DEFAULT '1' COMMENT '是否删除(1未删除；0已删除)',
  `create_id` varchar(64) DEFAULT NULL COMMENT '创建人',
  `update_id` varchar(64) DEFAULT NULL COMMENT '更新人',
  `create_where` tinyint(4) DEFAULT '1' COMMENT '创建来源(1.web 2.android 3.ios )',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `update_time` datetime DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1.4 配置数据源

注：application.properties、 application.yml 两个任选一个(两者功能都是一样的只是格式语法不一样而已)

注意

`com.mysql.jdbc.Driver` 是 mysql-connector-java 5中的， `com.mysql.cj.jdbc.Driver` 是 mysql-connector-java 6中的

- JDBC连接Mysql6 `com.mysql.cj.jdbc.Driver` ，需要指定时区serverTimezone:
  - 例子

    `url=jdbc:mysql://localhost:3306/company_frame?serverTimezone=UTC&useUnicode=true&characterEncoding=utf-8&useSSL=false`

1.4.1 application.properties

```
server.port=8080
spring.application.name=company-frame
#数据库配置
spring.datasource.type=com.alibaba.druid.pool.DruidDataSource
spring.datasource.druid.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.druid.url=jdbc:mysql://localhost:3306/company_frame?
useUnicode=true&characterEncoding=utf-8&useSSL=false
spring.datasource.druid.username=root
spring.datasource.druid.password=root
```

1.4.2 application.yml

```
server:
  port: 8080

spring:
  datasource:
    type: com.alibaba.druid.pool.DruidDataSource
    druid:
      url: jdbc:mysql://localhost:3306/company_frame?useUnicode=true&characterEncoding=utf-
8&useSSL=false
      username: root
      password: root
      driver-class-name: com.mysql.jdbc.Driver
```

1.5 druid 连接池&数据监控配置

druid官方文档

1.5.1 修改application.properties 加入

```
##################    连接池配置    ################
#连接池建立时创建的初始化连接数
spring.datasource.druid.initial-size=5
#连接池中最大的活跃连接数
spring.datasource.druid.max-active=20
#连接池中最小的活跃连接数
spring.datasource.druid.min-idle=5
# 配置获取连接等待超时的时间
spring.datasource.druid.max-wait=60000
# 打开PSCache，并且指定每个连接上PSCache的大小
spring.datasource.druid.pool-prepared-statements=true
spring.datasource.druid.max-pool-prepared-statement-per-connection-size=20
spring.datasource.druid.validation-query=SELECT 1 FROM DUAL
spring.datasource.druid.validation-query-timeout=30000
#是否在获得连接后检测其可用性
spring.datasource.druid.test-on-borrow=false
#是否在连接放回连接池后检测其可用性
spring.datasource.druid.test-on-return=false
#是否在连接空闲一段时间后检测其可用性
spring.datasource.druid.test-while-idle=true
# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
spring.datasource.druid.time-between-eviction-runs-millis=60000
# 配置一个连接在池中最小生存的时间，单位是毫秒
spring.datasource.druid.min-evictable-idle-time-millis=300000
# 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
spring.datasource.druid.filters=stat,wall
```

1.5.2 修改application.yml 加入

```
##################     连接池配置     ################
      #连接池建立时创建的初始化连接数
      initial-size: 5
      #连接池中最大的活跃连接数
      max-active: 20
      #连接池中最小的活跃连接数
      min-idle: 5
      # 配置获取连接等待超时的时间
      max-wait: 60000
      max-pool-prepared-statement-per-connection-size: 20
      # 打开PSCache，并且指定每个连接上PSCache的大小
      pool-prepared-statements: true
      validation-query: SELECT 1 FROM DUAL
      query-timeout: 30000
      #是否在获得连接后检测其可用性
      test-on-borrow: false
      #是否在连接放回连接池后检测其可用性
      test-on-return: false
      #是否在连接空闲一段时间后检测其可用性
      test-while-idle: true
      # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
      time-between-eviction-runs-millis: 60000
      # 配置一个连接在池中最小生存的时间，单位是毫秒
      min-evictable-idle-time-millis: 300000
      # 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙
      filter: stat,wall
```

1.5.3 测试

- 启动应用程序
- 浏览器输入：http://localhost:8080/druid/sql.html

1.6 日志配置

spring boot默认使用logback作为日志系统，可以结合slf4j日志框架使用，springboot已经为当前使用的日志框架的控制台输出和文件输出做好了配置，

我们只需要在配置文件里加入以下配置即可。

1.6.1修改 application.properties

```
#logging配置
logging.file=${logging.path}/${spring.application.name}.log
logging.path=logs
logging.level.com.yingxue.lesson=debug
```

1.6.2修改 application.yml

```
#logging配置
logging:
  file: ${logging.path}/${spring.application.name}.log
  path: logs
  level:
    com.yingxue.lesson: debug
```

1.7 generatorConfig 集成

1.7.1 新建 generatorConfig.xml

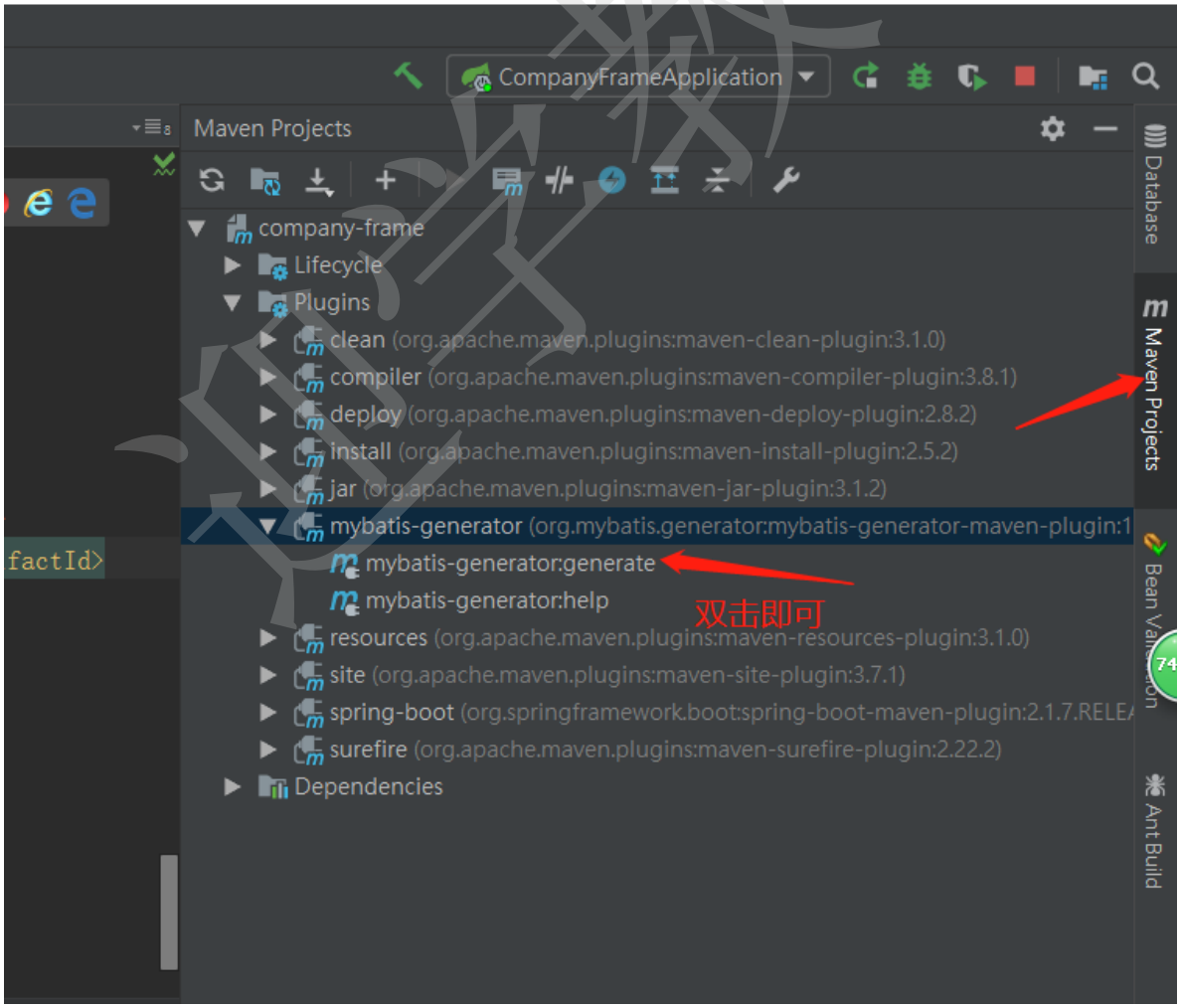
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <!--classPathEntry:数据库的JDBC驱动,换成你自己的驱动位置  -->
    <classPathEntry location="F:\mvnrepository\mysql\mysql-connector-java\5.1.28\mysql-connector-java-
5.1.28.jar" />

    <!-- 一个数据库一个context -->
    <!--defaultModelType="flat" 大数据字段，不分表 -->
    <context id="MysqlTables" targetRuntime="MyBatis3" defaultModelType="flat">
        <property name="autoDelimitKeywords" value="true"/>
        <property name="beginningDelimiter" value="`"/>
        <property name="endingDelimiter" value="`"/>
        <property name="javaFileEncoding" value="utf-8"/>
        <plugin type="org.mybatis.generator.plugins.SerializablePlugin"/>

        <plugin type="org.mybatis.generator.plugins.ToStringPlugin"/>

        <!-- 注释 -->
        <commentGenerator>
            <property name="suppressAllComments" value="true"/><!-- 是否取消注释 -->
            <property name="suppressDate" value="true"/> <!-- 是否生成注释代时间戳-->
        </commentGenerator>

        <!-- jdbc连接 -->
        <jdbcConnection driverClass="com.mysql.jdbc.Driver"
connectionURL="jdbc:mysql://localhost:3306/company_frame" userId="root"
                        password="root"/>
        <!-- 类型转换 -->
        <javaTypeResolver>
            <!-- 是否使用bigDecimal， false可自动转化以下类型（Long, Integer, Short, etc.） -->
            <property name="forceBigDecimals" value="false"/>
        </javaTypeResolver>

        <!-- 生成实体类地址 -->
        <javaModelGenerator targetPackage="com.yingxue.lesson.entity" targetProject="src/main/java">
            <property name="enableSubPackages" value="false"/>
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>
        <!-- 生成mapxml文件 -->
        <sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources">
            <property name="enableSubPackages" value="false"/>
        </sqlMapGenerator>
        <!-- 生成mapxml对应client，也就是接口dao -->
        <javaClientGenerator targetPackage="com.yingxue.lesson.mapper" targetProject="src/main/java"
type="XMLMAPPER">
            <property name="enableSubPackages" value="false"/>
        </javaClientGenerator>


        <table tableName="sys_user" domainObjectName="SysUser"
               enableCountByExample="false"
               enableUpdateByExample="false"
               enableDeleteByExample="false"
               enableSelectByExample="false"
               selectByExampleQueryId="true">
            <columnOverride column="sex" javaType="java.lang.Integer"/>
            <columnOverride column="status" javaType="java.lang.Integer"/>
            <columnOverride column="create_where" javaType="java.lang.Integer"/>
            <columnOverride column="deleted" javaType="java.lang.Integer"/>
        </table>

    </context>
</generatorConfiguration>
```

1.7.2 pom 添加相应 plugin

```
            <!--配置mybatis代码生成工具-->
```

```
<!--使用生成工具可以直接使用maven的命令提示符，
生成语句是mvn mybatis-generator:generate ，
一旦数据库进行了更改，都需使用这句代码重新生成bean、dao、mapper文件-->
<plugin>
    <groupId>org.mybatis.generator</groupId>
    <artifactId>mybatis-generator-maven-plugin</artifactId>
    <version>1.3.5</version>
    <configuration>
        <configurationFile>src/main/resources/generatorConfig.xml</configurationFile>
        <verbose>true</verbose>
        <overwrite>true</overwrite>
    </configuration>
    <executions>
        <execution>
            <phase>deploy</phase>
            <id>Generate MyBatis Artifacts</id>
            <goals>
                <goal>generate</goal>
            </goals>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>org.mybatis.generator</groupId>
            <artifactId>mybatis-generator-core</artifactId>
            <version>1.3.5</version>
        </dependency>
    </dependencies>
</plugin>
```

1.7.3 运行 mybatis-generator



1.7.4 配置 mybatis

- 修改 CompanyFrameApplication

```
package com.yingxue.lesson;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@MapperScan("com.yingxue.lesson.mapper")
public class CompanyFrameApplication {

    public static void main(String[] args) {
        SpringApplication.run(CompanyFrameApplication.class, args);
    }

}
```

- 修改 application.properties

```
#加入以下配置 对应生成 mapper.xml 的路径
mybatis.mapper-locations=classpath:mapper/*.xml
```

- 修改 application.yml

```
#加入以下配置 对应生成 mapper.xml 的路径
mybatis:
  mapper-locations: classpath:mapper/*.xml
```

1.8 集成swagger2

1.8.1 设置开关

注：往往线上一般是要把 swagger 接口入口给关闭而开发测试会打开，所以我们可以在配置文件上设置一个开关，多环境打包时候给出相应的值就可以了

- 修改 application.properties

```
#swagger 开关
swagger2.enable=true
```

- 修改 application.yml

```
#swagger 开关
swagger2:
  enable: true
```

1.8.2 创建 SwaggerConfig

```
package com.yingxue.lesson.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.ParameterBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Parameter;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;
import java.util.List;

/**
```

```
 * @ClassName: SwaggerConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Value("${swagger2.enable}")
    private boolean enable;

    @Bean
    public Docket createRestApi() {
        /**
         * 这是为了我们在用 swagger 测试接口的时候添加头部信息
         */
        List<Parameter> pars = new ArrayList<Parameter>();
        ParameterBuilder tokenPar = new ParameterBuilder();
        ParameterBuilder refreshTokenPar = new ParameterBuilder();
        tokenPar.name("authorization").description("swagger测试用(模拟authorization传入)非必填
header").modelRef(new ModelRef("string")).parameterType("header").required(false);
        refreshTokenPar.name("refresh_token").description("swagger测试用(模拟刷新token传入)非必填
header").modelRef(new ModelRef("string")).parameterType("header").required(false);
        /**
         * 多个的时候 就直接添加到 pars 就可以了
         */
        pars.add(tokenPar.build());
        pars.add(refreshTokenPar.build());
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())
                .select()
                .apis(RequestHandlerSelectors.basePackage("com.yingxue.lesson.controller"))
                .paths(PathSelectors.any())
                .build()
                .globalOperationParameters(pars)
                .enable(enable)
                ;
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("迎学教育")
                .description("迎学教育-spring boot 实战系列")
                .termsOfServiceUrl("")
                .version("1.0")
                .build();
    }

}
```

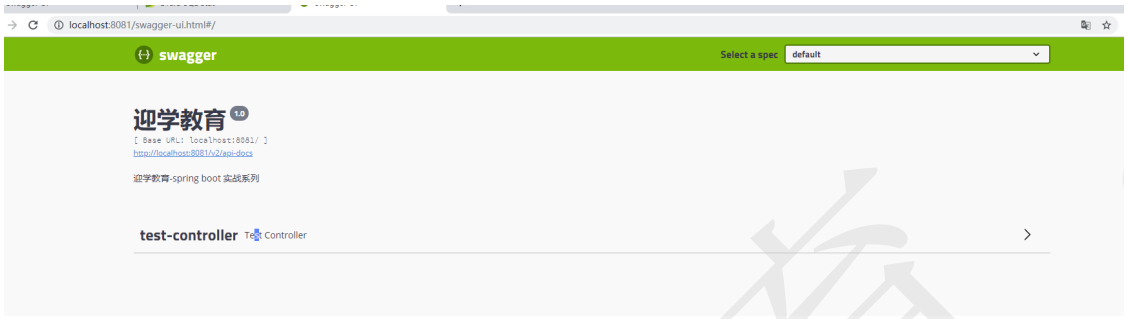
1.8.3 测试

- 创建 TestController

```
package com.yingxue.lesson.controller;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName: TestController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
```

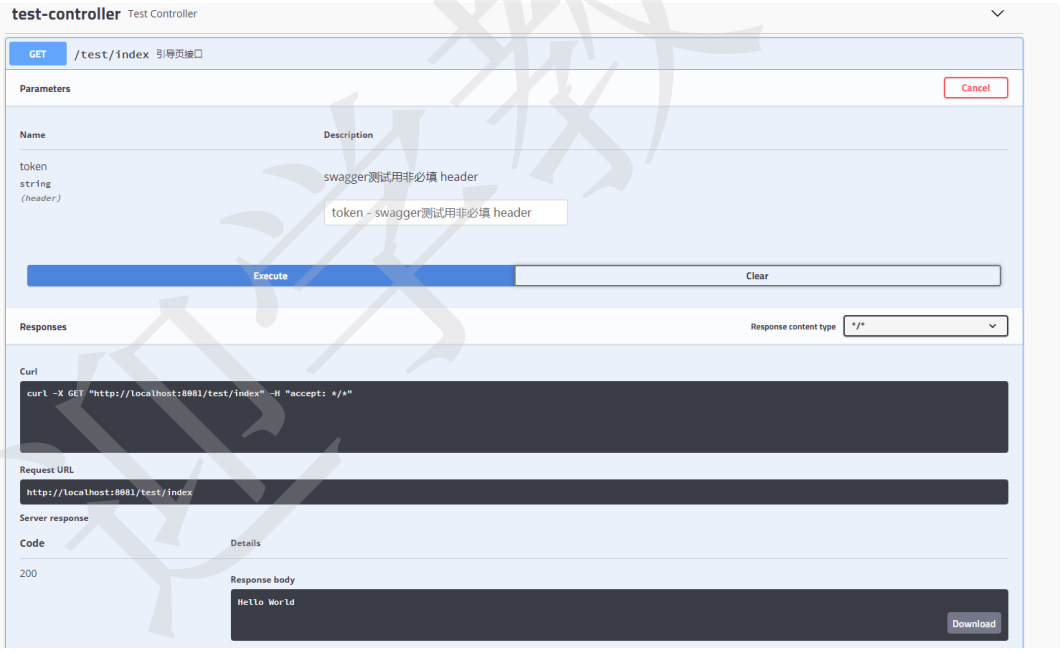
```
@RestController
@Api(value = "测试接口")
@RequestMapping("/test")
public class TestController {

    @GetMapping("/index")
    @ApiOperation(value = "引导页接口")
    public String testResult(){
        return "Hello World";
    }
}
```

- 浏览器输入 http://localhost:8080/swagger-ui.html



注意： swagger UI 的地址格式 http://host:port/swagger-ui.html

- 测试接口步骤
  - Try it out
  - Execute



1.9 集成 redisTemplate

1. 加入 redis 相关配置
    1. application.properties

```
# Redis 服务器地址
spring.redis.host=localhost
# Redis 服务器连接端口
spring.redis.port=6379
# 连接池最大连接数（使用负值表示没有限制） 默认 8
spring.redis.lettuce.pool.max-active=100
# 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
spring.redis.lettuce.pool.max-wait=PT10S
# 连接池中的最大空闲连接 默认 8
spring.redis.lettuce.pool.max-idle=30
# 连接池中的最小空闲连接 默认 0
spring.redis.lettuce.pool.min-idle=1
```

```
#链接超时时间
spring.redis.timeout=PT10S
```

2. application.yml

```
#redis配置开始
 redis:
   # Redis服务器地址
   host: localhost
   # Redis服务器连接端口
   port: 6379
   #Redis服务器连接密码（默认为空）
   #password:
   lettuce:
     pool:
       # 连接池最大连接数（使用负值表示没有限制）
       max-active: 1024
       # 连接池最大阻塞等待时间（使用负值表示没有限制）10秒
       max-wait: PT10S
       # 连接池中的最大空闲连接
       max-idle: 200
       # 连接池中的最小空闲连接
       min-idle: 0
   # 连接超时时间（10秒）
   timeout: PT10S
```

2. 创建一个序列化解析工具 类

```
package com.yingxue.lesson.serializer;

import com.alibaba.fastjson.JSON;
import org.springframework.data.redis.serializer.RedisSerializer;
import org.springframework.util.Assert;

import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

/**
 * @ClassName: MyStringRedisSerializer
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class MyStringRedisSerializer implements RedisSerializer<Object> {
    private final Charset charset;

    public MyStringRedisSerializer() {
        this(StandardCharsets.UTF_8);
    }

    public MyStringRedisSerializer(Charset charset) {
        Assert.notNull(charset, "Charset must not be null!");
        this.charset = charset;
    }

    @Override
    public String deserialize(byte[] bytes) {
        return (bytes == null ? null : new String(bytes, charset));
    }

    @Override
    public byte[] serialize(Object object) {
        if (object == null) {
            return new byte[0];
        }
        if(object instanceof String){
            return object.toString().getBytes(charset);
        }else {
            String string = JSON.toJSONString(object);
```

```
            return string.getBytes(charset);
        }

    }



}
```

3. 创建一个 redis 配置类 RedisConfig

```
package com.yingxue.lesson.config;

import com.yingxue.lesson.serializer.MyStringRedisSerializer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.StringRedisSerializer;

/**
 * @ClassName: RedisConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
redisConnectionFactory) {
        RedisTemplate<String, Object> template = new RedisTemplate();
        template.setConnectionFactory(redisConnectionFactory);
        StringRedisSerializer stringRedisSerializer=new StringRedisSerializer();
        MyStringRedisSerializer myStringRedisSerializer=new MyStringRedisSerializer();
        template.setKeySerializer(stringRedisSerializer);
        template.setValueSerializer(myStringRedisSerializer);
        template.setHashKeySerializer(stringRedisSerializer);
        template.setHashValueSerializer(myStringRedisSerializer);
        return template;
    }
}
```

4. 创建 redisTemplate 工具类

```
package com.yingxue.lesson.service;

import com.yingxue.lesson.exception.BusinessException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * @ClassName: RedisService
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class RedisService {
    @Autowired
    private RedisTemplate<String,Object> redisTemplate;
    /** -------------------key相关操作--------------------- */
```

```
/**
 * 是否存在key
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @return       java.lang.Boolean
 * @throws
 */
public Boolean hasKey(String key) {
    if (null==key){
        return false;
    }
    return redisTemplate.hasKey(key);
}

/**
 * 删除key
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @return       Boolean  成功返回true 失败返回false
 * @throws
 */
public Boolean delete(String key) {
    if (null==key){
        return false;
    }
    return redisTemplate.delete(key);
}

/**
 * 批量删除key
 * @Author:      小霍
 * @CreateDate:  2019/8/27 20:27
 * @UpdateUser:
 * @UpdateDate:  2019/8/27 20:27
 * @Version:     0.0.1
 * @param keys
 * @return       Long 返回成功删除key的数量
 * @throws
 */
public Long delete(Collection<String> keys) {
    return redisTemplate.delete(keys);
}

/**
 * 设置过期时间
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
• * @param timeout
• * @param unit
 * @return       java.lang.Boolean
 * @throws
 */
public Boolean expire(String key, long timeout, TimeUnit unit) {
    if (null==key||null==unit){
        return false;
    }
    return redisTemplate.expire(key, timeout, unit);
}

/**
 * 查找匹配的key
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param pattern
 * @return       java.util.Set<java.lang.String>
```

```
     * @throws
     */
    public Set<String> keys(String pattern) {
        if (null==pattern){
            return null;
        }
        return redisTemplate.keys(pattern);
    }


    /**
     * 移除 key 的过期时间，key 将持久保持
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean persist(String key) {
        if (null==key){
            return false;
        }
        return redisTemplate.persist(key);
    }

    /**
     * 返回 key 的剩余的过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param unit
     * @return       java.lang.Long 当 key 不存在时，返回 -2 。当 key 存在但没有设置剩余生存时间时，返回 -1 。否则，以秒为单位，返回 key的剩余生存时间
     * @throws
     */
    public Long getExpire(String key, TimeUnit unit) {
        if(null==key||null==unit){
            throw new BusinessException(4001004,"key or TomeUnit 不能为空");
        }
        return redisTemplate.getExpire(key, unit);
    }

    //*************String相关数据类型**************************
    /**
     * 设置指定 key 的值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param value
     * @return       void
     * @throws
     */
    public void set(String key, Object value) {

        if(null==key||null==value){
            return;
        }
        redisTemplate.opsForValue().set(key, value);
    }
    /**
     * 设置key 的值 并设置过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param value
     * * @param time
     * * @param unit
     * @return       void
     * @throws
     */
```

```
    public void set(String key,Object value,long time,TimeUnit unit){

        if(null==key||null==value||null==unit){
            return;
        }
        redisTemplate.opsForValue().set(key,value,time,unit);
    }
    /**
     * 设置key 的值 并设置过期时间
     * key存在 不做操作返回false
     * key不存在设置值返回true
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param value
    • * @param time
    • * @param unit
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean setifAbsen(String key,Object value,long time,TimeUnit unit){

        if(null==key||null==value||null==unit){
            throw new BusinessException(4001004,"kkey、value、unit都不能为空");
        }
        return redisTemplate.opsForValue().setIfAbsent(key,value,time,unit);
    }
    /**
     * 获取指定Key的Value。如果与该Key关联的Value不是string类型，Redis将抛出异常，
     * 因为GET命令只能用于获取string Value，如果该Key不存在，返回null
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Object
     * @throws
     */
    public Object get(String key){

        if(null==key){
            return null;
        }
        return  redisTemplate.opsForValue().get(key);
    }
    /**
     * 很明显先get再set就说先获取key值对应的value然后再set 新的value 值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param value
     * @return       java.lang.Object
     * @throws
     */
    public Object getSet(String key,Object value){

        if(null==key){
            return null;
        }
        return redisTemplate.opsForValue().getAndSet(key,value);
    }
    /**
     * 通过批量的key获取批量的value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param keys
     * @return       java.util.List<java.lang.Object>
     * @throws
     */
    public List<Object> mget(Collection<String> keys){

        if(null==keys){
```

```
            return Collections.emptyList();
        }
        return redisTemplate.opsForValue().multiGet(keys);
    }
    /**
     *  将指定Key的Value原子性的增加increment。如果该Key不存在，其初始值为0，在incrby之后其值为increment。
     *  如果Value的值不能转换为整型值，如Hi，该操作将执行失败并抛出相应异常。操作成功则返回增加后的value值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param increment
     * @return       long
     * @throws
     */
    public long incrby(String key,long increment){
        if(null==key){
            throw new BusinessException(4001004,"key不能为空");
        }
        return redisTemplate.opsForValue().increment(key,increment);
    }
    /**
     *
     * 将指定Key的Value原子性的减少decrement。如果该Key不存在，其初始值为0，
     * 在decrby之后其值为-decrement。如果Value的值不能转换为整型值，
     * 如Hi，该操作将执行失败并抛出相应的异常。操作成功则返回减少后的value值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param decrement
     * @return       java.lang.Long
     * @throws
     */
    public Long decrby(String key,long decrement){
        if(null==key){
            throw new BusinessException(4001004,"key不能为空");
        }
        return redisTemplate.opsForValue().decrement(key,decrement);
    }
    /**
     *  如果该Key已经存在，APPEND命令将参数Value的数据追加到已存在Value的末尾。如果该Key不存在，
     *  APPEND命令将会创建一个新的Key/Value。返回追加后Value的字符串长度。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param value
     * @return       java.lang.Integer
     * @throws
     */
    public Integer append(String key,String value){
        if(key==null){
            throw new BusinessException(4001004,"key不能为空");
        }
        return redisTemplate.opsForValue().append(key,value);
    }
//******************hash数据类型*********************
    /**
     * 通过key 和 field 获取指定的 value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param field
     * @return       java.lang.Object
     * @throws
     */
    public Object hget(String key, Object field) {
        if(null==key||null==field){
            return null;
        }
        return redisTemplate.opsForHash().get(key,field);
    }
```

```
/**
 * 为指定的Key设定Field/Value对，如果Key不存在，该命令将创建新Key以用于存储参数中的Field/Value对，
 * 如果参数中的Field在该Key中已经存在，则用新值覆盖其原有值。
 * 返回1表示新的Field被设置了新值，0表示Field已经存在，用新值覆盖原有值。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @param field
 * @param value
 * @return
 * @throws
 */
public void hset(String key, Object field, Object value) {
    if(null==key||null==field){
        return;
    }
    redisTemplate.opsForHash().put(key,field,value);
}

/**
 * 判断指定Key中的指定Field是否存在，返回true表示存在，false表示参数中的Field或Key不存在。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @param field
 * @return       java.lang.Boolean
 * @throws
 */
public Boolean hexists(String key, Object field) {
    if(null==key||null==field){
        return false;
    }
    return redisTemplate.opsForHash().hasKey(key,field);
}

/**
 * 从指定Key的Hashes Value中删除参数中指定的多个字段，如果不存在的字段将被忽略，
 * 返回实际删除的Field数量。如果Key不存在，则将其视为空Hashes，并返回0。
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @param fields
 * @return       java.lang.Long
 * @throws
 */
public Long hdel(String key, Object... fields) {
    if(null==key||null==fields||fields.length==0){
        return 0L;
    }
    return redisTemplate.opsForHash().delete(key,fields);
}


/**
 * 通过key获取所有的field和value
 * @Author:      小霍
 * @UpdateUser:
 * @Version:     0.0.1
 * @param key
 * @return       java.util.Map<java.lang.Object,java.lang.Object>
 * @throws
 */
public Map<Object, Object> hgetall(String key) {
    if(key==null){
        return null;
    }
    return redisTemplate.opsForHash().entries(key);
}

/**
```

```
     * 逐对依次设置参数中给出的Field/Value对。如果其中某个Field已经存在，则用新值覆盖原有值。
     * 如果Key不存在，则创建新Key，同时设定参数中的Field/Value。
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param hash
     * @return
     * @throws
     */
    public void hmset(String key, Map<String, Object> hash) {

        if(null==key||null==hash){
            return;
        }
        redisTemplate.opsForHash().putAll(key,hash);
    }

    /**
     * 获取和参数中指定Fields关联的一组Values，其返回顺序等同于Fields的请求顺序。
     * 如果请求的Field不存在，其值对应的value为null。
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param fields
     * @return        java.util.List<java.lang.Object>
     * @throws
     */
    public List<Object> hmget(String key, Collection<Object> fields) {

        if(null==key||null==fields){
            return null;
        }

        return redisTemplate.opsForHash().multiGet(key,fields);
    }

    /**
     * 对应key的字段自增相应的值
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * * @param field
     * * @param increment
     * @return        java.lang.Long
     * @throws
     */
    public Long hIncrBy(String key,Object field,long increment){
        if (null==key||null==field){
            throw new BusinessException(4001004,"key or field 不能为空");
        }
        return redisTemplate.opsForHash().increment(key,field,increment);

    }
    //***************List数据类型***************
    /**
     * 向列表左边添加元素。如果该Key不存在，该命令将在插入之前创建一个与该Key关联的空链表，之后再将数据从链表的头部插入。
     * 如果该键的Value不是链表类型，该命令将将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param strs 可以使一个string 也可以使string数组
     * @return        java.lang.Long 返回操作的value个数
     * @throws
     */
    public Long lpush(String key, Object... strs) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().leftPushAll(key,strs);
```

```
    }

    /**
     * 向列表右边添加元素。如果该Key不存在，该命令将在插入之前创建一个与该Key关联的空链表，之后再将数据从链表的尾部插入。
     * 如果该键的Value不是链表类型，该命令将将会抛出相关异常。操作成功则返回插入后链表中元素的数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param strs 可以使一个string 也可以使string数组
     * @return       java.lang.Long 返回操作的value个数
     * @throws
     */
    public Long rpush(String key, Object... strs) {
        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().rightPushAll(key,strs);
    }
    /**
     * 返回并弹出指定Key关联的链表中的第一个元素，即头部元素。如果该Key不存在，
     * 返回nil。LPOP命令执行两步操作：第一步是将列表左边的元素从列表中移除，第二步是返回被移除的元素值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Object
     * @throws
     */
    public Object lpop(String key) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().leftPop(key);
    }

    /**
     * 返回并弹出指定Key关联的链表中的最后一个元素，即头部元素。如果该Key不存在，返回nil。
     * RPOP命令执行两步操作：第一步是将列表右边的元素从列表中移除，第二步是返回被移除的元素值。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Object
     * @throws
     */
    public Object rpop(String key) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().rightPop(key);
    }

    /**
     *该命令的参数start和end都是0-based。即0表示链表头部(leftmost)的第一个元素。
     * 其中start的值也可以为负值，-1将表示链表中的最后一个元素，即尾部元素，-2表示倒数第二个并以此类推。
     * 该命令在获取元素时，start和end位置上的元素也会被取出。如果start的值大于链表中元素的数量，
     * 空链表将会被返回。如果end的值大于元素的数量，该命令则获取从start(包括start)开始，链表中剩余的所有元素。
     * 注：Redis的列表起始索引为0。显然，LRANGE numbers 0 -1 可以获取列表中的所有元素。返回指定范围内元素的列表。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param start
     * @param end
     * @return       java.util.List<java.lang.Object>
     * @throws
     */
    public List<Object> lrange(String key, long start, long end) {
        if(null==key){
            return null;
        }
```

```
        return redisTemplate.opsForList().range(key,start,end);
    }

    /**
     * 让列表只保留指定区间内的元素，不在指定区间之内的元素都将被删除。
     * 下标 0 表示列表的第一个元素，以 1 表示列表的第二个元素，以此类推。
     * 你也可以使用负数下标，以 -1 表示列表的最后一个元素， -2 表示列表的倒数第二个元素，以此类推。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param start
     * @param end
     * @return
     * @throws
     */
    public void ltrim(String key, long start, long end) {
        if(null==key){
            return;
        }
        redisTemplate.opsForList().trim(key,start,end);
    }

    /**
     * 该命令将返回链表中指定位置(index)的元素，index是0-based，表示从头部位置开始第index的元素，
     * 如果index为-1，表示尾部元素。如果与该Key关联的不是链表，该命令将返回相关的错误信息。 如果超出index返回这返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param index
     * @return       java.lang.Object
     * @throws
     */
    public Object lindex(String key, long index) {
        if(null==key){
            return null;
        }
        return redisTemplate.opsForList().index(key,index);
    }

    /**
     * 返回指定Key关联的链表中元素的数量，如果该Key不存在，则返回0。如果与该Key关联的Value的类型不是链表，则抛出相关的异常。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Long
     * @throws
     */
    public Long llen(String key) {

        if(null==key){
            return 0L;
        }
        return redisTemplate.opsForList().size(key);
    }
    //***************Set数据类型**************
    /**
     * 如果在插入的过程用，参数中有的成员在Set中已经存在，该成员将被忽略，而其它成员仍将会被正常插入。
     * 如果执行该命令之前，该Key并不存在，该命令将会创建一个新的Set，此后再将参数中的成员陆续插入。返回实际插入的成员数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param members 可以是一个String 也可以是一个String数组
     * @return       java.lang.Long 添加成功的个数
     * @throws
     */
    public Long sadd(String key, Object... members) {
        if (null==key){
```

```
            return 0L;
        }
        return redisTemplate.opsForSet().add(key, members);

    }

    /**
     * 返回Set中成员的数量，如果该Key并不存在，返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Long
     * @throws
     */
    public Long scard(String key) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().size(key);

    }

    /**
     * 判断参数中指定成员是否已经存在于与Key相关联的Set集合中。返回true表示已经存在，false表示不存在，或该Key本身并不存在。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param member
     * @return       java.lang.Boolean
     * @throws
     */
    public Boolean sismember(String key, Object member) {
        if (null==key){
            return false;
        }
        return redisTemplate.opsForSet().isMember(key,member);

    }

    /**
     * 和SPOP一样，随机的返回Set中的一个成员，不同的是该命令并不会删除返回的成员。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.String
     * @throws
     */
    public Object srandmember(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().randomMember(key);

    }
    /**
     * 和SPOP一样，随机的返回Set中的一个成员，不同的是该命令并不会删除返回的成员。
     * 还可以传递count参数来一次随机获得多个元素，根据count的正负不同，具体表现也不同。
     * 当count 为正数时，SRANDMEMBER 会随机从集合里获得count个不重复的元素。
     * 如果count的值大于集合中的元素个数，则SRANDMEMBER 会返回集合中的全部元素。
     * 当count为负数时，SRANDMEMBER 会随机从集合里获得|count|个的元素，如果|count|大与集合中的元素，
     * 就会返回全部元素不够的以重复元素补齐，如果key不存在则返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param count
     * @return       java.util.List<java.lang.String>
     * @throws
     */
    public List<Object> srandmember(String key,int count) {
```

```
        if(null==key){
            return null;
        }
        return redisTemplate.opsForSet().randomMembers(key,count);

    }

    /**
     * 通过key随机删除一个set中的value并返回该值
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.String
     * @throws
     */
    public Object spop(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().pop(key);

    }

    /**
     * 通过key获取set中所有的value
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> smembers(String key) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForSet().members(key);

    }
    /**
     * 从与Key关联的Set中删除参数中指定的成员，不存在的参数成员将被忽略，
     * 如果该Key并不存在，将视为空Set处理。返回从Set中实际移除的成员数量，如果没有则返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * * @param members
     * @return       java.lang.Long
     * @throws
     */
    public Long srem(String key, Object... members) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForSet().remove(key,members);

    }

    /**
     * 将元素value从一个集合移到另一个集合
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param srckey
     * * @param dstkey
     * * @param member
     * @return       java.lang.Long
     * @throws
     */
    public Boolean smove(String srckey, String dstkey, Object member) {
        if (null==srckey||null==dstkey){
            return false;
        }
```

```
        return redisTemplate.opsForSet().move(srckey,member,dstkey);

    }


    /**
     * 获取两个集合的并集
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param otherKeys
     * @return       java.util.Set<java.lang.Object> 返回两个集合合并值
     * @throws
     */
    public Set<Object> sUnion(String key, String otherKeys) {
        if (null==key||otherKeys==null){
            return null;
        }
        return redisTemplate.opsForSet().union(key, otherKeys);
    }
    //**********Sorted Set 数据类型*******************
    /**
     *添加参数中指定的所有成员及其分数到指定key的Sorted Set中，在该命令中我们可以指定多组score/member作为参数。
     * 如果在添加时参数中的某一成员已经存在，该命令将更新此成员的分数为新值，同时再将该成员基于新值重新排序。
     * 如果键不存在，该命令将为该键创建一个新的Sorted Set Value，并将score/member对插入其中。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param score
    • * @param member
     * @return       java.lang.Long
     * @throws
     */
    public Boolean zadd(String key, double score, Object member) {
        if (null==key){
            return false;
        }
        return redisTemplate.opsForZSet().add(key,member,score);

    }


    /**
     * 该命令将移除参数中指定的成员，其中不存在的成员将被忽略。
     * 如果与该Key关联的Value不是Sorted Set，相应的错误信息将被返回。 如果操作成功则返回实际被删除的成员数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
    • * @param members 可以使一个string 也可以是一个string数组
     * @return       java.lang.Long
     * @throws
     */
    public Long zrem(String key, Object... members) {
        if(null==key||null==members){
            return 0L;
        }
        return redisTemplate.opsForZSet().remove(key,members);

    }

    /**
     * 返回Sorted Set中的成员数量，如果该Key不存在，返回0。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @return       java.lang.Long
     * @throws
     */
    public Long zcard(String key) {
        if (null==key){
```

```
            return 0L;
        }
        return redisTemplate.opsForZSet().size(key);
    }

    /**
     * 该命令将为指定Key中的指定成员增加指定的分数。如果成员不存在，该命令将添加该成员并假设其初始分数为0，
     * 此后再将其分数加上increment。如果Key不存在，该命令将创建该Key及其关联的Sorted Set，
     * 并包含参数指定的成员，其分数为increment参数。如果与该Key关联的不是Sorted Set类型，
     * 相关的错误信息将被返回。如果不报错则以串形式表示的新分数。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param score
     * @param member
     * @return       java.lang.Double
     * @throws
     */
    public Double zincrby(String key, double score, Object member) {
        if (null==key){
            throw new BusinessException(4001004,"key 不能为空");
        }
        return redisTemplate.opsForZSet().incrementScore(key,member,score);
    }

    /**
     * 该命令用于获取分数(score)在min和max之间的成员数量。
     * （min=<score<=max）如果加上了“(”着表明是开区间例如zcount key (min max 则 表示（min<score=<max）
     * 同理zcount key min (max 则表明(min=<score<max) 返回指定返回数量。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param min
     * @param max
     * @return       java.lang.Long
     * @throws
     */
    public Long zcount(String key, double min, double max) {
        if (null==key){
            return 0L;
        }
        return redisTemplate.opsForZSet().count(key, min, max);

    }

    /**
     * Sorted Set中的成员都是按照分数从低到高的顺序存储，该命令将返回参数中指定成员的位置值，
     * 其中0表示第一个成员，它是Sorted Set中分数最低的成员。 如果该成员存在，则返回它的位置索引值。否则返回nil。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param member
     * @return       java.lang.Long
     * @throws
     */
    public Long zrank(String key, Object member) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().rank(key,member);

    }

    /**
     * 如果该成员存在，以字符串的形式返回其分数，否则返回null
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param member
     * @return       java.lang.Double
```

```
     * @throws
     */
    public Double zscore(String key, Object member) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().score(key,member);
    }

    /**
     * 该命令返回顺序在参数start和stop指定范围内的成员，这里start和stop参数都是0-based，即0表示第一个成员，-1表示最后一个成员。如果start大于该Sorted
     * Set中的最大索引值，或start > stop，此时一个空集合将被返回。如果stop大于最大索引值，
     * 该命令将返回从start到集合的最后一个成员。如果命令中带有可选参数WITHSCORES选项，
     * 该命令在返回的结果中将包含每个成员的分数值，如value1,score1,value2,score2...。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param min
     * @param max
     * @return       java.util.Set<java.lang.String> 指定区间内的有序集成员的列表。
     * @throws
     */
    public Set<Object> zrange(String key, long min, long max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().range(key, min, max);

    }
    /**
     * 该命令的功能和ZRANGE基本相同，唯一的差别在于该命令是通过反向排序获取指定位置的成员，
     * 即从高到低的顺序。如果成员具有相同的分数，则按降序字典顺序排序。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param start
     * @param end
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zReverseRange(String key, long start, long end) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().reverseRange(key, start, end);

    }

    /**
     * 该命令将返回分数在min和max之间的所有成员，即满足表达式min <= score <= max的成员，
     * 其中返回的成员是按照其分数从低到高的顺序返回，如果成员具有相同的分数，
     * 则按成员的字典顺序返回。可选参数LIMIT用于限制返回成员的数量范围。
     * 可选参数offset表示从符合条件的第offset个成员开始返回，同时返回count个成员。
     * 可选参数WITHSCORES的含义参照ZRANGE中该选项的说明。*最后需要说明的是参数中min和max的规则可参照命令ZCOUNT。
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param key
     * @param max
     * @param min
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zrangebyscore(String key, double min, double max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().rangeByScore(key, min, max);

    }
```

```
    /**
     * 该命令除了排序方式是基于从高到低的分数排序之外，其它功能和参数含义均与ZRANGEBYSCORE相同。
     * 需要注意的是该命令中的min和max参数的顺序和ZRANGEBYSCORE命令是相反的。
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param key
     * @param max
     * @param min
     * @return       java.util.Set<java.lang.String>
     * @throws
     */
    public Set<Object> zrevrangeByScore(String key, double min, double max) {
        if (null==key){
            return null;
        }
        return redisTemplate.opsForZSet().reverseRangeByScore(key, min, max);
    }
}
```

2. 前后端分离数据封装 DataResult

目前市面上公司开发模式普遍采用了前后端分离，而前后端交互一般会以 json 的形式交互，既然涉及到多方交互那就需要一些约定好的交互格式，然而每个人的想法有可能是不一样的千人前面所以定义的格式字段就可能不一样，如果我们后端不统一前端会"炸的" 笑脸~~~ 所以我们需要封装一个统一的返回格式。

2.1 创建 DataResult 类

```
package com.yingxue.lesson.utils;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: DataResult
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class DataResult <T> {
    /**
     * 请求响应code，0为成功 其他为失败
     */
    @ApiModelProperty(value = "请求响应code，0为成功 其他为失败", name = "code")
    private int code = 0;

    /**
     * 响应异常码详细信息
     */
    @ApiModelProperty(value = "响应异常码详细信息", name = "msg")
    private String msg;

    /**
     * 响应内容 ， code 0 时为 返回 数据
     */
    @ApiModelProperty(value = "需要返回的数据", name = "data")
    private T data;
}
```

2.2 加入相应的构造方法

```
public DataResult(int code, T data) {
```

```
        this.code = code;
        this.data = data;
        this.msg=null;
    }

    public DataResult(int code, String msg, T data) {
        this.code = code;
        this.msg = msg;
        this.data = data;
    }

    public DataResult(int code, String msg) {
        this.code = code;
        this.msg = msg;
        this.data=null;
    }
```

2.3 封装统一的相应 code 工具类

2.3.1 创建 ResponseCodeInterface

```
package com.yingxue.lesson.exception.code;

/**
 * @ClassName: ResponseCodeInterface
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public interface ResponseCodeInterface {
    int getCode();

    String getMsg();
}
```

2.3.2 创建 BaseResponseCode 枚举类

```
package com.yingxue.lesson.exception.code;

/**
 * @ClassName: BaseResponseCode
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public enum  BaseResponseCode implements ResponseCodeInterface{
    /**
     * 这个要和前段约定好
     *code=0：服务器已成功处理了请求。 通常，这表示服务器提供了请求的网页。
     *code=4010001：（授权异常） 请求要求身份验证。 客户端需要跳转到登录页面重新登录
     *code=4010002：(凭证过期) 客户端请求刷新凭证接口
     *code=4030001：没有权限禁止访问
     *code=400xxxx：系统主动抛出的业务异常
     *code=5000001：系统异常
     *
     */
    SUCCESS(0,"操作成功")
    ;
    /**
     * 错误码
     */
    private final int code;
    /**
     * 错误消息
     */
    private final String msg;

    BaseResponseCode(int code, String msg) {
```

```
        this.code = code;
        this.msg = msg;
    }
    @Override
    public int getCode() {
        return code;
    }

    @Override
    public String getMsg() {
        return msg;
    }
}
```

2.4 进一步封装 DataResult

为了 DataResult 可以提供给我们更友好的服务，我们对 DataResult 进行封装。

```
public DataResult() {
        this.code=BaseResponseCode.SUCCESS.getCode();
        this.msg=BaseResponseCode.SUCCESS.getMsg();
        this.data=null;
    }

    public DataResult(T data) {
        this.data = data;
        this.code=BaseResponseCode.SUCCESS.getCode();
        this.msg=BaseResponseCode.SUCCESS.getMsg();
    }

    public DataResult(ResponseCodeInterface responseCodeInterface) {
        this.data = null;
        this.code = responseCodeInterface.getCode();
        this.msg = responseCodeInterface.getMsg();
    }

    public DataResult(ResponseCodeInterface responseCodeInterface, T data) {
        this.data = data;
        this.code = responseCodeInterface.getCode();
        this.msg = responseCodeInterface.getMsg();
    }
    /**
     * 操作成功 data为null
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.xh.lesson.utils.DataResult<T>
     * @throws
     */
    public static DataResult success(){
        return new DataResult();
    }
    /**
     * 操作成功 data 不为null
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param data
     * @return       com.xh.lesson.utils.DataResult<T>
     * @throws
     */
    public static <T>DataResult success(T data){
        return new DataResult(data);
    }
    /**
     * 自定义 返回操作 data 可控
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param code
     * @param msg
```

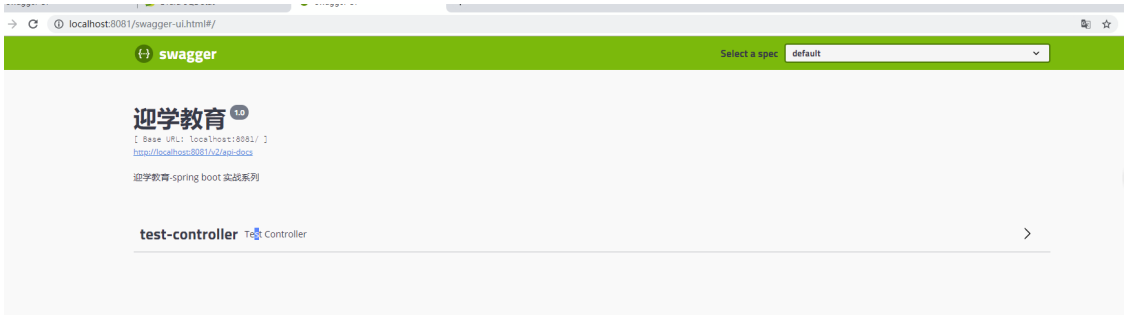
```
     * @param data
     * @return       com.xh.lesson.utils.DataResult
     * @throws
     */
    public static <T>DataResult getResult(int code,String msg,T data){
        return new DataResult(code,msg,data);
    }
    /**
     *  自定义返回  data为null
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param code
     * @param msg
     * @return       com.xh.lesson.utils.DataResult
     * @throws
     */
    public static DataResult getResult(int code,String msg){
        return new DataResult(code,msg);
    }
    /**
     * 自定义返回 入参一般是异常code枚举 data为空
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param responseCode
     * @return       com.xh.lesson.utils.DataResult
     * @throws
     */
    public static DataResult getResult(BaseResponseCode responseCode){
        return new DataResult(responseCode);
    }
    /**
     * 自定义返回 入参一般是异常code枚举 data 可控
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param responseCode
     * @param data
     * @return       com.xh.lesson.utils.DataResult
     * @throws
     */
    public static <T>DataResult getResult(BaseResponseCode responseCode, T data){

        return new DataResult(responseCode,data);
    }
```

2.5 创建 测试 DataResult 接口

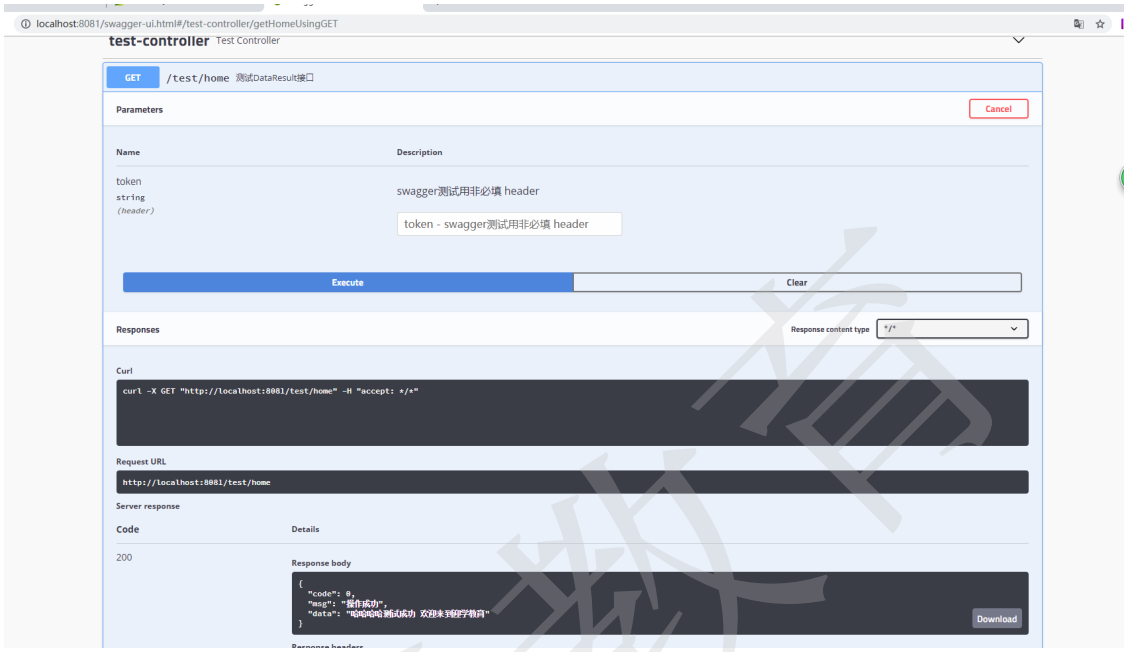
```
    @GetMapping("/home")
    @ApiOperation(value = "测试DataResult接口")
    public DataResult<String> getHome(){
        DataResult<String> result=DataResult.success("哈哈哈哈测试成功 欢迎来到迎学教育");
        return result;
    }
```

2.6 打开 swagger UI

- 浏览器输入：http://localhost:8080/swagger-ui.html



- 找到我们创建的 测试接口测试



3. 全局异常统一处理

因为目前市面上的企业开发模式大都是前后端分离的，就是前端只干前端的活，后端干后端的活，分工明确相互配合。既然要配合就需要一些统一的约定，特别是一些异常信息。因为一些原因系统异常、返回的格式往往不是我们和前后端约定好的格式；这个就需要我们对系统中的异常进行监控，引导到相应的封装方法，对异常信息进行包装返回统一的格式给前端。

在我们们学习这节课前，处理的方法是不是在 Controller 层进行 try-catch 然后再 catch 处理异常信息封装格式，但是，Controller 层每个方法体都写一些模板化的 try-catch 的代码，很难看也难维护，特别是还需要对 Service 层的不同异常进行不同处理的时候封装起来更麻烦。

这节课讲解使用 @RestControllerAdvice+ @ExceptionHandler 进行全局的 Controller 层异常处理，这样就可以避免在 Controller 层进行 try-catch 了，我们只需大胆的抛出异常。剩下的交给 @ControllerAdvice + @ExceptionHandler 处理就可以了。

3.1 解决 spring boot devtool 热部署后出现访问 404 问题

DevTools的检测时间和idea的编译所需时间存在差异。在idea还没完成编译工作前，DevTools就开始进行重启和加载，导致@RequestMapping没有被全部正常处理。其他方法没试，就直接用了看起来最简单的方法：牺牲一点时间，去加长devtools的轮询时间，增大等待时间。

解决方案如下：

spring.devtools.restart.poll-interval=3000ms spring.devtools.restart.quiet-period=2999ms

3.2基本使用示例

3.2.1 @RestControllerAdvice

创建一个 RestExceptionHandler 类 加上 @ControllerAdvice 注解

```
package com.yingxue.lesson.exception.handler;

import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.RestControllerAdvice;

/**
 * @ClassName: RestExceptionHandler
 * 公共异常处理类
```

```
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestControllerAdvice
@Slf4j
public class RestExceptionHandler {
}
```

3.2.2 @ExceptionHandler

声明一个异常处理的方法

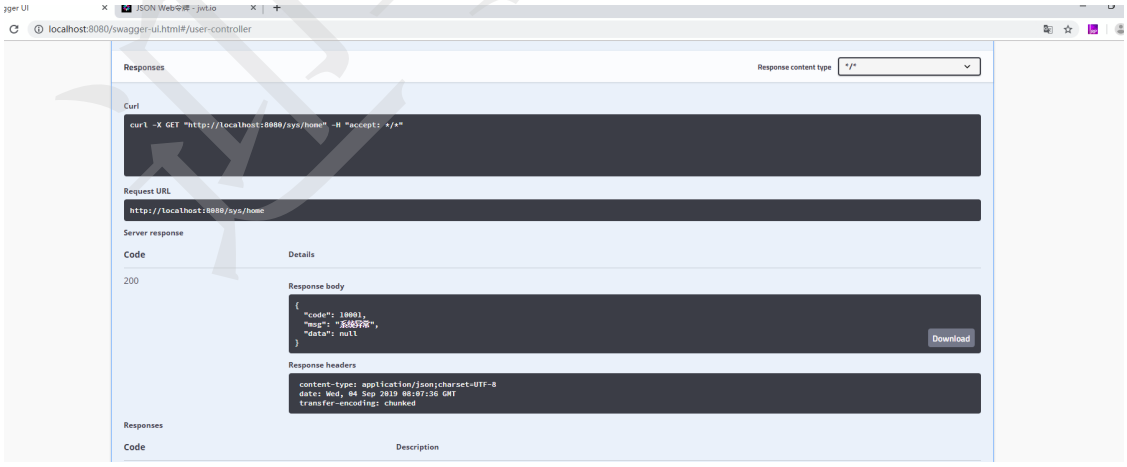
```
/**
     * 系统繁忙，请稍候再试
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param e
     * @return       com.yingxue.lesson.utils.DataResult<T>
     * @throws
     */
    @ExceptionHandler(Exception.class)
    public <T> DataResult<T> handleException(Exception e){
        log.error("Exception,exception:{}", e);
        return DataResult.getResult(BaseResponseCode.SYSTEM_BUSY);
    }
```

3.2.3 测试

- 修改 getHome() 方法

```
    @GetMapping("/home")
    @ApiOperation(value = "测试DataResult接口")
    public DataResult<String> getHome(){
        int i=1/0;
        DataResult<String> result=DataResult.success("哈哈哈哈测试成功 欢迎来到迎学教育");
        return result;
    }
```

- 打开浏览器到swagger UI 测试



3.3 实战中异常封装

在现实实战中，往往我们会在复杂的带有数据库事务的业务中，经常会遇到一些不规则的信息，这个就需要我们后端根据相应的业务抛出相应的运行时异常，进行数据库事务回滚，并希望该异常信息能被返回显示给用户。

3.3.1 自定义运行时异常 BusinessException

- 修改 BusinessException 类加入如下方法

```
    /**
     * 构造函数
     * @param code 异常码
     */
    public BusinessException(ResponseCodeInterface code) {
        this(code.getCode(), code.getMsg());
    }
```

- 修改 RestExceptionHandler 加入 封装自定义异常的方法

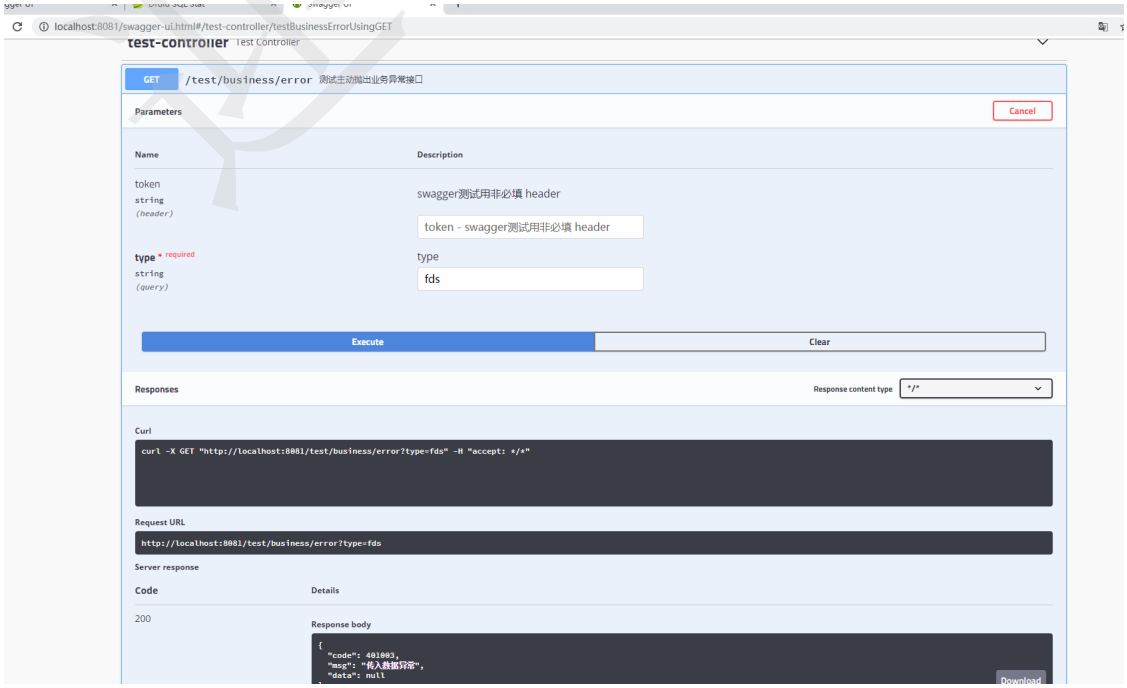
```
/**
     * 自定义全局异常处理
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param e
     * @return       com.yingxue.lesson.utils.DataResult<T>
     * @throws
     */
    @ExceptionHandler(value = BusinessException.class)
    <T> DataResult<T> businessExceptionHandler(BusinessException e) {
        log.error("BusinessException,exception:{}", e);
        return new DataResult<>(e.getMessageCode(),e.getDetailMessage());
    }
```

- 新增测试主动抛出业务异常接口

```
    @GetMapping("/business/error")
    @ApiOperation(value = "测试主动抛出业务异常接口")
    public DataResult<String> testBusinessError(@RequestParam String type){
        if(!type.equals("1")||type.equals("2")||type.equals("3")){
            throw new BusinessException(BaseResponseCode.DATA_ERROR);
        }
        DataResult<String> result=new DataResult(0,type);
        return result;
    }
```

3.3.2 测试

- 打开浏览器进入swagger ui 找到 测试主动抛出业务异常接口
- 执行测试



4. Hibernate Validator 详解

4.1.Hibernate Validator 简介

平时项目中，难免需要对参数 进行一些参数正确性的校验，这些校验出现在业务代码中，让我们的业务代码显得臃肿，而且，频繁的编写这类参数校验代码很无聊。鉴于此，觉得 Hibernate Validator 框架刚好解决了这些问题，可以很优雅的方式实现参数的校验，让业务代码和校验逻辑分开,不再编写重复的校验逻辑。Hibernate Validator 提供了 JSR 303 规范中所有内置约束的实现，除此之外还有一些附加的约束。**Bean Validation** 为 JavaBean 验证定义了相应的元数据模型和API。缺省的元数据是 Java Annotations，通过使用 XML 可以对原有的元数据信息进行覆盖和扩展。Bean Validation 是一个运行时的数据验证框架，在验证之后验证的错误信息会被马上返回。

4.2 Hibernate Validator 的作用

- 验证逻辑与业务逻辑之间进行了分离，降低了程序耦合度；
- 统一且规范的验证方式，无需你再次编写重复的验证代码；
- 你将更专注于你的业务，将这些繁琐的事情统统丢在一边。

4.3 Hibernate Validator 的使用

项目中，主要用于接口api 的入参校验和 封装工具类 在代码中校验两种使用方式。

4.4 常用的注解

- @NotEmpty 用在集合类上面
- @NotBlank 用在String上面
- @NotNull 用在基本类型上
- @Valid:启用校验

4.5 测试

4.5.1 创建 TestReqVO

```
package com.yingxue.lesson.vo.req;

import lombok.Data;

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;
import java.util.List;

/**
 * @ClassName: TestReqVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class TestReqVO {

    @ApiModelProperty(value = "名称")
    @NotBlank(message = "名称不能为空")
    private String name;

    @NotNull(message = "age 不能为空")
    @ApiModelProperty(value = "年龄")
    private Integer age;

    @NotEmpty(message = "id 集合不能为空")
    @ApiModelProperty(value = "id集合")
    private List<String> ids;
}
```

4.5.2 新增测试接口

修改 TestController.java 加入如下方法

```
    @PostMapping("/valid/error")
    @ApiOperation(value = "测试Validator抛出业务异常接口")
    public DataResult testValid(@RequestBody @Valid TestReqVO vo){
        DataResult result=DataResult.success();
        return result;
    }
```

4.6全局捕获校验抛出异常

修改 RestExceptionHandler.java 加入如下代码

```
 /**
     * 处理validation 框架异常
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param e
     * @return       com.yingxue.lesson.utils.DataResult<T>
     * @throws
     */
    @ExceptionHandler(MethodArgumentNotValidException.class)
    <T> DataResult<T> methodArgumentNotValidExceptionHandler(MethodArgumentNotValidException e) {

        log.error("methodArgumentNotValidExceptionHandler bindingResult.allErrors():{},exception:{}",
e.getBindingResult().getAllErrors(), e);
        List<ObjectError> errors = e.getBindingResult().getAllErrors();
        return createValidExceptionResp(errors);
    }
    private <T> DataResult<T> createValidExceptionResp(List<ObjectError> errors) {
        String[] msgs = new String[errors.size()];
        int i = 0;
        for (ObjectError error : errors) {
            msgs[i] = error.getDefaultMessage();
            log.info("msg={}",msgs[i]);
            i++;
        }
        return DataResult.getResult(BaseResponseCode.METHOD_IDENTITY_ERROR.getCode(), msgs[0]);
    }
```

5. 10分钟搞懂：JWT(Json Web Token)

5.1 JWT 简介

基于token的用户认证原理：让用户输入账号和密码，认证通过后获得一个token（令牌），在token有效期里用户可以带着token访问特定资源。

开始token并没有一个统一标准，大家都各自使用自己的方案。后来出现了JWT(Json Web Token)这个标准。

JWT本质上是一个对JSON对象加密后的字符串。当服务器认证用户通过后，一个包含用户信息的json对象被加密后返回给用户，json对象：

```
{
    "expire": "2019-11-29 20:19:00",
    "permissions": [
        "sys:user:list",
        "sys:dept:list",
        "sys:role:list"
    ],
    "role": [
        "dev"
    ],
    "userName": "dev123"
}
```

之后，用户访问服务器时，都要返回这个json对象。服务器只靠这个对象就可以识别用户身份，不需要再去查数据库。为了防止用户篡改数据，服务器在生成对象时将添加一个签名。

服务器不保存任何会话数据，也就是说，服务器变得无状态，从而更容易扩展。

5.2 JWT 怎么用

以浏览器接收到服务器发过来的jwt后，可以存储在Cookie 或 localStorage 中。之后，浏览器每次与服务器通信时都会带上JWT。可以将JWT放在Cookie中，会自动发送（不跨域），或将JWT放在HTTP请求头的授权字段中。

```
Authorization: Bearer <token>
```

也可放在url中，或POST请求的数据体中。

5.3 JWT 结构

jwt有3个组成部分，每部分通过点号来分割 header.payload.signature

- 头部（header) 是一个 JSON 对象
- 载荷（payload) 是一个 JSON 对象，用来存放实际需要传递的数据
- 签名（signature) 对header和payload使用密钥进行签名，防止数据篡改。

5.3.1 头部 header

Jwt的头部是一个JSON,然后使用Base64URL编码，承载两部分信息：

- 声明类型typ，表示这个令牌（token）的类型（type），JWT令牌统一写为JWT
- 声明加密的算法alg，通常直接使用HMACSHA256，就是HS256了，也可以使用RSA,支持很多算法(HS256、HS384、HS512、RS256、RS384、RS512、ES256、ES384、ES512、PS256、PS384)

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

- Base64URL 编码后(Base64编码后可能出现字符+和/，在URL中不能直接作为参数，Base64URL就是把字符+和/分别变成-和_。JWT有可能放在url中，所以要用Base64URL编码。)

  ```
  eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
  ```

5.3.2 载荷 payload

payload也是一个JSON字符串，是承载消息具体内容的地方，也需要使用Base64URL编码，就是存储我们要保存到客户端的信息，一般都是包含用户的基本信息，权限信息，时间戳等信息。

JWT指定了一些官方字段（claims）备用:

- iss: 签发人
- exp: 过期时间
- iat: 签发时间
- nbf: 生效时间
- jti: 编号
- sub: 主题
- aud: 受众

除了官方字段，在这个部分还可以添加私有字段，例如：

```
{
  "sub": "1dfaafa7-fddf-46f2-b3d8-11bfe9ac7230",
  "jwt-roles-key_": [
    "普通用户角色"
  ],
  "iss": "yingxue.com",
  "jwt-permissions-key": [
    "sys:user:list",
    "sys:dept:list",
    "sys:role:list",
    "sys:permission:list",
    "sys:log:list"
  ],
  "jwt-user-name-key": "dev123",
  "exp": 1575005723,
  "iat": 1574998523
}
```

Base64URL编码的后：

```
eyJzdWIiOiIxZGZhYWZhNy1mZGRmLTQ2ZjItYjNkOC0xMWJmZTlhYzcyMzAiLCJqd3Qtcm9sZXMta2V5XyI6WyLmma7pgJrnlKjmiLfo
p5LoibIiXSwiaXNzIjoieWluZ3h1ZS5jb20iLCJqd3QtcGVybWlzc2lvbnMta2V5IjpbInN5czp1c2VyOmxpc3QiLCJzeXM6ZGVwdDps
aXN0Iiwic3lzOnJvbGU6bGlzdCIsInN5czpwZXJtaXNzaW9uOmxpc3QiLCJzeXM6bG9nOmxpc3QiXSwiandOLXVzZXItbmFtZS1rZXki
OiJkZXYxMjMiLCJleHAiOjE1NzUwMDU3MjMsImlhdCI6MTU3NDk5ODUyM30
```

5.3.3 签名 Signature

Signature部分是对前两部分的防篡改签名。将Header和Payload用Base64URL编码后，再用点(.)连接起来。然后使用签名算法和密钥对这个字符串进行签名：

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

首先，需要指定一个密码（secret）。该密码保存在服务器中，**并且不能向用户公开。**然后，使用标头中指定的签名算法根据以下公式生成签名。signature = HMACSHA256(header + "." + payload, secret); 在计算出签名哈希后，JWT头，有效载荷和签名哈希的三个部分组合成一个字符串，每个部分用"."分隔，就构成整个JWT对象。 以上三部分都是在服务器定义，当用户登陆成功后，根据用户信息，按照jwt规则生成token返回给客户端。

签名信息：

```
qYWHdAbYZlP6akHTrDm-MkIWia8mPW-TO75eu8r0-Vk
```

组合在一起

3部分组合在一起，构成了完整的jwt：

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxZGZhYWZhNy1mZGRmLTQ2ZjItYjNkOC0xMWJmZTlhYzcyMzAiLCJqd3Qtcm9sZXMta2V5XyI
6WyLmma7pgJrnlKjmiLfop5LoibIiXSwiaXNzIjoieWluZ3h1ZS5jb20iLCJqd3QtcGVybWlzc2lvbnMta2V5IjpbInN5czp1c2VyOmx
pc3QiLCJzeXM6ZGVwdDpsaXN0Iiwic3lzOnJvbGU6bGlzdCIsInN5czpwZXJtaXNzaW9uOmxpc3QiLCJzeXM6bG9nOmxpc3QiXSwiand
OLXVzZXItbmFtZS1rZXkiOiJkZXYxMjMiLCJleHAiOjE1NzUwMDU3MjMsImlhdCI6MTU3NDk5ODUyM30.qYWHdAbYZlP6akHTrDm-
MkIWia8mPW-TO75eu8r0-Vk
```

5.4 使用要点

- JWT默认是不加密的，但也可以加密，不加密时不宜在jwt中存放敏感信息
- 不要泄露签名密钥(secret)
- jwt签发后无法撤回，有效期不宜太长
- JWT 泄露会被人冒用身份，为防止盗用，JWT应尽量使用 https 协议传输

5.5 实战流程(JWT 使用姿势)

大家有没有发现，现在的网站通常第一次登录验证通过后，在后续的操作都不需要用户名密码，那后端怎么确定这次访问的用户是合法用户呢？其实当第一次登录后，服务器生成一个Token 便将此 Token 返回给客户端，以后客户端只需带上这个 Token 前来请求数据即可，无需再次带上用户名和密码。

那么我们后端该怎么实现上述的业务呢？

1. **有状态 token**

   所谓的有状态就是 把生成的 token 保存在服务器端。

   实现步骤：

   - 当用户登录进来后端生成一个随机数 token(我通常用uuid) 然后把 token 做key userId 做为 value 存入 reids 并且设置失效时间。
   - 编写一个拦截器，设置要拦截的 api(即是受保护的api)和开放的api(用户登录、注册等接口)。
   - 去 header 或者 cookie 拿 token，如果 token 为空或者 token 已经失效(拿 token 去 redis 检测是否失效)则告知客户端引导到登录页面。
2. **无状态 token**

   所谓的无状态 token 就是服务器不保存 token 信息，当用户登陆成功后，返回 token 给客户端，客户端保存起来每次请求都会带过来。其实我们用 token 的作用就是拿到用户ID 只有拿到了 ID 才能区别是哪个用户访问，那么 JWT 刚刚好满足要求，JWT 是签发给客户端而且 用户 ID 直接存在 JWT 里面，客户端每次请求过来的时候我们直接解析 JWT 拿到用户 ID，这样就达到了识别用户的效果。

   但是在使用 JWT 的时候都会遇到下列的烦恼？

   无法作废已颁布的令牌。所有的认证信息都在 JWT 中，由于在服务端没有状态，即使你知道了某个 JWT 被盗取了，你也没有办法将其作废。在 JWT 过期之前（你绝对应该设置过期时间），你无能为力。

   不易应对数据过期。与上一条类似，JWT 有点类似缓存，由于无法作废已颁布的令牌，在其过期前，你只能忍受“过期”的数据。

   我的使用姿势：

- 用户登录进来，会生产两个 token (一个过期时间比较短的 access_token ,一个过期时间比较长的 refresh_token)，创建一个拦截器拦截用户请求。
- 当要更新jwt携带的数据时候，直接用refresh_token 刷新 access_token,而老的access_token 用redis 标记起来并设置过期时间(过期时间为该令牌剩余的过期时间)
- 当要作废令牌的时候，直接把这个令牌在redis 标记起来，并且设置过期时间(过期时间为该令牌剩余的过期时间)。

6. JWT 工具类封装

我们在日常开发中会多次去验证客户端传入的 token，所以我们要把验证的方法抽出来，封装成一个工具类，每次直接用工具类调用就可以了

- 首先创建一个 JwtTokenUtil

```
package com.yingxue.lesson.utils;

import lombok.extern.slf4j.Slf4j;

/**
 * @ClassName: JwtTokenUtil
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Slf4j
public class JwtTokenUtil {
}
```

- 加入 JWT 相关配置参数(.properties 和.yml 配置其中一个即可)
  - application.properties

```
#JWT 密钥
jwt.secretKey=78944878877848fg)
jwt.accessTokenExpireTime=PT2H
jwt.refreshTokenExpireTime=PT8H
jwt.refreshTokenExpireAppTime=P30D
jwt.issuer=yingxue.com
```

  - application.yml

```
#JWT 密钥
jwt:
 secretKey: 78944878877848fg)
 accessTokenExpireTime: PT2H
 refreshTokenExpireTime: PT8H
 refreshTokenExpireAppTime=P30D
 issuer: yingxue.com
```

- 创建配置读取类

```
package com.yingxue.lesson.utils;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;

import java.time.Duration;

/**
 * @ClassName: TokenSettings
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
@ConfigurationProperties(prefix = "jwt")
```

```
@Data
public class TokenSettings {
    private String secretKey;
    private Duration accessTokenExpireTime;
    private Duration refreshTokenExpireTime;
    private Duration refreshTokenExpireAppTime;
    private String  issuer;
}
```

- 创建初始化配置代理类

```
package com.yingxue.lesson.utils;

import org.springframework.stereotype.Component;

/**
 * @ClassName: InitializerUtil
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Component
public class InitializerUtil {
    private TokenSettings tokenSettings;

    public InitializerUtil(TokenSettings tokenSettings) {
        JwtTokenUtil.setTokenSettings(tokenSettings);
    }
}
```

- 修改JwtTokenUtil加入签发 token 方法

```
@Slf4j
public class JwtTokenUtil {

    private static String secretKey;
    private static Duration accessTokenExpireTime;
    private static Duration refreshTokenExpireTime;
    private static Duration refreshTokenExpireAppTime;
    private static String  issuer;

    public static void setTokenSettings(TokenSettings tokenSettings){
        secretKey=tokenSettings.getSecretKey();
        accessTokenExpireTime=tokenSettings.getAccessTokenExpireTime();
        refreshTokenExpireTime=tokenSettings.getRefreshTokenExpireTime();
        refreshTokenExpireAppTime=tokenSettings.getRefreshTokenExpireAppTime();
        issuer=tokenSettings.getIssuer();
    }
     /**
     * 生成 access_token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param subject
     * @param claims
     * @return       java.lang.String
     * @throws
     */
    public static String getAccessToken(String subject, Map<String,Object> claims){

        return generateToken(issuer,subject,claims,accessTokenExpireTime.toMillis(),secretKey);
    }
     /**
     * 签发token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param issuer 签发人
```

```
     * @param subject 代表这个JWT的主体，即它的所有人 一般是用户id
     * @param claims 存储在JWT里面的信息 一般放些用户的权限/角色信息
     * @param ttlMillis 有效时间(毫秒)
     * @return       java.lang.String
     * @throws
     */
    public static String generateToken(String issuer, String subject,Map<String, Object> claims,
long ttlMillis,String secret) {

        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);

        byte[] signingKey = DatatypeConverter.parseBase64Binary(secret);

        JwtBuilder builder = Jwts.builder();
        if(null!=claims){
            builder.setClaims(claims);
        }
        if (!StringUtils.isEmpty(subject)) {
            builder.setSubject(subject);
        }
        if (!StringUtils.isEmpty(issuer)) {
            builder.setIssuer(issuer);
        }
        builder.setIssuedAt(now);
        if (ttlMillis >= 0) {
            long expMillis = nowMillis + ttlMillis;
            Date exp = new Date(expMillis);
            builder.setExpiration(exp);
        }
        builder.signWith(signatureAlgorithm, signingKey);
        return builder.compact();
    }


}
```

```
// 上面我们已经有生成 access_token 的方法，下面加入生成 refresh_token 的方法(PC 端过期时间短一些)

   /**
     * 生产 PC refresh_token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param subject
    • * @param claims
     * @return       java.lang.String
     * @throws
     */
    public static String getRefreshToken(String subject,Map<String,Object> claims){
        return generateToken(issuer,subject,claims,refreshTokenExpireTime.toMillis(),secretKey);
    }
```

- 上面我们已经有生成 access_token 的方法，下面加入生成 refresh_token 的方法(APP 端过期时间长一些)

```
 /**
     * 生产 App端 refresh_token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param subject
    • * @param claims
     * @return       java.lang.String
     * @throws
     */
    public static String getRefreshAppToken(String subject,Map<String,Object> claims){
        return
generateToken(issuer,subject,claims,refreshTokenExpireAppTime.toMillis(),secretKey);
    }
```

- 加入解析令牌的方法

```
/**
     * 从令牌中获取数据声明
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       io.jsonwebtoken.Claims
     * @throws
     */
    public static Claims getClaimsFromToken(String token) {
        Claims claims;
        try {
            claims =
Jwts.parser().setSigningKey(DatatypeConverter.parseBase64Binary(secretKey)).parseClaimsJws(token).
getBody();
        } catch (Exception e) {
            claims = null;
        }
        return claims;
    }
```

- 加入获取 userId 方法

```
/**
     * 获取用户id
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       java.lang.String
     * @throws
     */
    public static String getUserId(String token){
        String userId=null;
        try {
            Claims claims = getClaimsFromToken(token);
            userId = claims.getSubject();
        } catch (Exception e) {
            log.error("eror={}",e);
        }
        return userId;
    }
```

- 加入获取 username 方法
  - 创建一个专门存放常量的类

```
package com.yingxue.lesson.constants;

/**
 * @ClassName: Constant
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class Constant {
}
```

```
      //Contants 加入 用户名 key 常量
      /**
       * 用户名称 key
       */
      public static final String JWT_USER_NAME="jwt-user-name-key";
```

```
 * 获取用户名称
       */
```

```
```

/** * 获取用户名 * @Author: 小霍 * @UpdateUser: * @Version: 0.0.1 * @param token * @return java.lang.String * @throws */ public static String getUserName(String token){

```
    String username=null;
    try {
        Claims claims = getClaimsFromToken(token);
        username = (String) claims .get(Constant.JWT_USER_NAME);
    } catch (Exception e) {
        log.error("eror={}",e);
    }
    return username;
}
```

```
```

- 验证 token 是否过期方法

```
/**
     * 验证token 是否过期(true:已过期 false:未过期)
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
    • * @param secretKey
     * @return       java.lang.Boolean
     * @throws
     */
    public static Boolean isTokenExpired(String token) {

        try {
            Claims claims = getClaimsFromToken(token);
            Date expiration = claims.getExpiration();
            return expiration.before(new Date());
        } catch (Exception e) {
            log.error("error={}",e);
            return true;
        }
    }
```

- 验证令牌方法

```
    /**
     * 校验令牌(true：验证通过 false：验证失败)
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       java.lang.Boolean
     * @throws
     */
    public static Boolean validateToken(String token) {
        Claims claimsFromToken = getClaimsFromToken(token);
        return (null!=claimsFromToken && !isTokenExpired(token));
    }
```

- 刷新 令牌方法

```
/**
     * 刷新token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param refreshToken
     * @param claims 主动去刷新的时候 改变JWT payload 内的信息
     * @return       java.lang.String
     * @throws
     */
    public static String refreshToken(String refreshToken,Map<String, Object> claims) {
        String refreshedToken;
        try {
            Claims parserclaims = getClaimsFromToken(refreshToken);
            /**
             * 刷新token的时候如果为空说明原先的 用户信息不变 所以就引用上个token里的内容
             */
            if(null==claims){
                claims=parserclaims;
            }
            refreshedToken =
generateToken(parserclaims.getIssuer(),parserclaims.getSubject(),claims,accessTokenExpireTime.toMil
lis(),secretKey);
        } catch (Exception e) {
            refreshedToken = null;
            log.error("error={}",e);
        }
        return refreshedToken;
    }
```

- 获取token的剩余过期时间

```
/**
     * 获取token的剩余过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * * @param secretKey
     * @return       long
     * @throws
     */
    public static long getRemainingTime(String token){
        long result=0;
        try {
            long nowMillis = System.currentTimeMillis();
            result= getClaimsFromToken(token).getExpiration().getTime()-nowMillis;
        } catch (Exception e) {
            log.error("error={}",e);
        }
        return result;
    }
```

这样一个完整的 JWT 工具类就创建完成啦

```java
package com.yingxue.lesson.utils;

import com.yingxue.lesson.constants.Constant;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.JwtBuilder;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import lombok.extern.slf4j.Slf4j;
import org.springframework.util.StringUtils;

import javax.xml.bind.DatatypeConverter;
import java.time.Duration;
import java.util.Date;
import java.util.Map;

/**
 * @ClassName: JwtTokenUtil
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Slf4j
public class JwtTokenUtil {
    private static String secretKey;
    private static Duration accessTokenExpireTime;
    private static Duration refreshTokenExpireTime;
    private static Duration refreshTokenExpireAppTime;
    private static String  issuer;

    public static void setTokenSettings(TokenSettings tokenSettings){
        secretKey=tokenSettings.getSecretKey();
        accessTokenExpireTime=tokenSettings.getAccessTokenExpireTime();
        refreshTokenExpireTime=tokenSettings.getRefreshTokenExpireTime();
        refreshTokenExpireAppTime=tokenSettings.getRefreshTokenExpireAppTime();
        issuer=tokenSettings.getIssuer();
    }
    /**
     * 生成 access_token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param subject
     * @param claims
     * @return       java.lang.String
     * @throws
     */
    public static String getAccessToken(String subject, Map<String,Object> claims){

        return generateToken(issuer,subject,claims,accessTokenExpireTime.toMillis(),secretKey);
    }
    /**
     * 生产 App端 refresh_token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param subject
     * @param claims
     * @return       java.lang.String
     * @throws
     */
    public static String getRefreshAppToken(String subject,Map<String,Object> claims){
        return generateToken(issuer,subject,claims,refreshTokenExpireAppTime.toMillis(),secretKey);
    }

    /**
     * 生产 PC refresh_token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param subject
     * @param claims
     * @return       java.lang.String
     * @throws
```

```
     */
    public static String getRefreshToken(String subject,Map<String,Object> claims){
        return generateToken(issuer,subject,claims,refreshTokenExpireTime.toMillis(),secretKey);
    }
    /**
     * 签发token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param issuer 签发人
     * @param subject 代表这个JWT的主体，即它的所有人 一般是用户id
     * @param claims 存储在JWT里面的信息 一般放些用户的权限/角色信息
     * @param ttlMillis 有效时间(毫秒)
     * @return       java.lang.String
     * @throws
     */
    public static String generateToken(String issuer, String subject,Map<String, Object> claims, long
ttlMillis,String secret) {

        SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

        long nowMillis = System.currentTimeMillis();
        Date now = new Date(nowMillis);

        byte[] signingKey = DatatypeConverter.parseBase64Binary(secret);

        JwtBuilder builder = Jwts.builder();
        if(null!=claims){
            builder.setClaims(claims);
        }
        if (!StringUtils.isEmpty(subject)) {
            builder.setSubject(subject);
        }
        if (!StringUtils.isEmpty(issuer)) {
            builder.setIssuer(issuer);
        }
        builder.setIssuedAt(now);
        if (ttlMillis >= 0) {
            long expMillis = nowMillis + ttlMillis;
            Date exp = new Date(expMillis);
            builder.setExpiration(exp);
        }
        builder.signWith(signatureAlgorithm, signingKey);
        return builder.compact();
    }
    /**
     * 获取用户id
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       java.lang.String
     * @throws
     */
    public static String getUserId(String token){
        String userId=null;
        try {
            Claims claims = getClaimsFromToken(token);
            userId = claims.getSubject();
        } catch (Exception e) {
            log.error("eror={}",e);
        }
        return userId;
    }
    /**
     * 获取用户名
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       java.lang.String
     * @throws
     */
    public static String getUserName(String token){
```

```
        String username=null;
        try {
            Claims claims = getClaimsFromToken(token);
            username = (String) claims .get(Constant.JWT_USER_NAME);
        } catch (Exception e) {
            log.error("eror={}",e);
        }
        return username;
    }
    /**
     * 从令牌中获取数据声明
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       io.jsonwebtoken.Claims
     * @throws
     */
    public static Claims getClaimsFromToken(String token) {
        Claims claims;
        try {
            claims =
Jwts.parser().setSigningKey(DatatypeConverter.parseBase64Binary(secretKey)).parseClaimsJws(token).getBod
y();
        } catch (Exception e) {
            claims = null;
        }
        return claims;
    }
    /**
     * 校验令牌(true：验证通过 false：验证失败)
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @return       java.lang.Boolean
     * @throws
     */
    public static Boolean validateToken(String token) {
        Claims claimsFromToken = getClaimsFromToken(token);
        return (null!=claimsFromToken && !isTokenExpired(token));
    }
    /**
     * 验证token 是否过期(true:已过期 false:未过期)
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * * @param secretKey
     * @return       java.lang.Boolean
     * @throws
     */
    public static Boolean isTokenExpired(String token) {

        try {
            Claims claims = getClaimsFromToken(token);
            Date expiration = claims.getExpiration();
            return expiration.before(new Date());
        } catch (Exception e) {
            log.error("error={}",e);
            return true;
        }
    }
    /**
     * 刷新token
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param refreshToken
     * @param claims 主动去刷新的时候 改变JWT payload 内的信息
     * @return       java.lang.String
     * @throws
     */
```

```
    public static String refreshToken(String refreshToken,Map<String, Object> claims) {
        String refreshedToken;
        try {
            Claims parserclaims = getClaimsFromToken(refreshToken);
            /**
             * 刷新token的时候如果为空说明原先的 用户信息不变 所以就引用上个token里的内容
             */
            if(null==claims){
                claims=parserclaims;
            }
            refreshedToken =
generateToken(parserclaims.getIssuer(),parserclaims.getSubject(),claims,accessTokenExpireTime.toMillis()
,secretKey);
        } catch (Exception e) {
            refreshedToken = null;
            log.error("error={}",e);
        }
        return refreshedToken;
    }
    /**
     * 获取token的剩余过期时间
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param token
     * @param secretKey
     * @return       long
     * @throws
     */
    public static long getRemainingTime(String token){
        long result=0;
        try {
            long nowMillis = System.currentTimeMillis();
            result= getClaimsFromToken(token).getExpiration().getTime()-nowMillis;
        } catch (Exception e) {
            log.error("error={}",e);
        }
        return result;
    }
}
```

7. SpringBoot+shiro+jwt 实现用户认证签发 token

首先用户登录进来，我们先验证用户名/密码，验证通过后我们会生成两个 token(access_token、refresh_token 他们的唯一区别是一个过期时间短一个过期时间长)然后把一些必要的参数封装成 LoginRespVO 响应回客户端。token 主要包含 用户id、用户登录名、用户所拥有的角色(这里我们先写 mock 数据)、用户所拥有的权限(这里我们先写 mock 数据)、和签发单位标识。

代码说明：

- LoginReqVO - 接收客户端表单提交数据
- LoginRespVO - 响应客户端数据
- UserController.java – 控制层
- UserService.java & UserServiceImpl.java – 服务层
- UserMapper.java & UserMapper.xml – 数据访问层
- PasswordEncoder & PasswordUtils - 密码校验工具类

插入数据脚本

```
INSERT INTO `sys_user` (`id`, `username`, `salt`, `password`, `phone`, `dept_id`, `real_name`,
`nick_name`, `email`, `status`, `sex`, `deleted`, `create_id`, `update_id`, `create_where`,
`create_time`, `update_time`) VALUES ('9a26f5f1-cbd2-473d-82db-1d6dcf4598f8', 'admin',
'324ce32d86224b00a02b', 'ac7e435db19997a46e3b390e69cb148b', '13888888888', '24f41c71-5a95-4ef4-9493-
174574f3b0c5', NULL, NULL, 'yingxue@163.com', '1', NULL, '1', NULL, NULL, '3', '2019-09-22 19:38:05',
NULL);
INSERT INTO `sys_user` (`id`, `username`, `salt`, `password`, `phone`, `dept_id`, `real_name`,
`nick_name`, `email`, `status`, `sex`, `deleted`, `create_id`, `update_id`, `create_where`,
`create_time`, `update_time`) VALUES ('9a26f5f1-cbd2-473d-82db-1d6dcf4598f4', 'dev123',
'324ce32d86224b00a02b', 'ac7e435db19997a46e3b390e69cb148b', '13666666666', '24f41c71-5a95-4ef4-9493-
174574f3b0c5', NULL, NULL, 'yingxue@163.com', '1', NULL, '1', NULL, NULL, '3', '2019-09-22 19:38:05',
NULL);
```

7.1 LoginReqVO

```
package com.yingxue.lesson.vo.req;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: LoginReqVO
 * TODO:接收客户端请求进来数据
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class LoginReqVO {
    @ApiModelProperty(value = "账号")
    private String username;
    @ApiModelProperty(value = "用户密码")
    private String password;
    @ApiModelProperty(value = "登录类型(1:pc;2:App)")
    @NotBlank(message = "登录类型不能为空")
    private String type;
}
```

7.2 创建 LoginRespVO

```
package com.yingxue.lesson.vo.resp;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: LoginRespVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class LoginRespVO {
    @ApiModelProperty(value = "token")
    private String accessToken;
    @ApiModelProperty(value = "刷新token")
    private String refreshToken;
    @ApiModelProperty(value = "用户名")
    private String username;
    @ApiModelProperty(value = "用户id")
    private String id;
    @ApiModelProperty(value = "电话")
    private String phone;

}
```

7.3 创建 UserService 接口

```
package com.yingxue.lesson.service;

import com.yingxue.lesson.vo.req.LoginReqVO;
import com.yingxue.lesson.vo.resp.LoginRespVO;


/**
 * @ClassName: UserService
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public interface UserService {

    /**
     * 用户登录接口
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param vo
     * @return       com.yingxue.lesson.vo.resp.LoginRespVO
     * @throws
     */
    LoginRespVO login(LoginReqVO vo);

}
```

7.4 创建 UserService实现类

```

/**
 * @ClassName: UserServiceImpl
 * TODO:类文件简单描述
 * @Author: 小霍
 * @CreateDate: 2019/9/7 22:56
 * @UpdateUser: 小霍
 * @UpdateDate: 2019/9/7 22:56
 * @Version: 0.0.1
 */
@Service
@Slf4j
public class UserServiceImpl implements UserService {



}
```

7.5 实现登录接口业务

- 修改 Contants 加入几个认证的常量

```
 /**
     * 权限key
     */
    public static final String JWT_PERMISSIONS_KEY="jwt-permissions-key_";


    /**
     * 角色key
     */
    public static final String JWT_ROLES_KEY="jwt-roles-key_";
```

- 实现获取角色业务

  先用 mock 数据后续讲到权限管理业务的时候直接从 DB 读取

修改 UserServiceImpl.java 加入如下代码

```
/**
     * 获取用户的角色
     * 这里先用伪代码代替
     * 后面我们讲到权限管理系统后 再从 DB 读取
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param userId
     * @return
     * @throws
     */
    private List<String> getRolesByUserId(String userId){
        List<String> list=new ArrayList<>();
        if("9a26f5f1-cbd2-473d-82db-1d6dcf4598f8".equals(userId)){
            list.add("admin");
        }else{
            list.add("test");
        }
        return  list;
    }
```

- 实现获取菜单权限业务

  先用 mock 数据后续讲到权限管理业务的时候直接从 DB 读取

  修改 UserServiceImpl.java 加入如下代码

```
/**
     * 获取用户的权限
     * 这里先用伪代码代替
     * 后面我们讲到权限管理系统后 再从 DB 读取
     * @Author:       小霍
     * @UpdateUser:
     * @Version:      0.0.1
     * @param userId
     * @return
     * @throws
     */
    private List<String>getPermissionsByUserId(String userId){
        List<String> list=new ArrayList<>();
        if("9a26f5f1-cbd2-473d-82db-1d6dcf4598f8".equals(userId)){
            list.add("sys:user:add");
            list.add("sys:user:list");
            list.add("sys:user:update");
            list.add("sys:user:detail");
        }else{
            list.add("sys:user:list");
        }
        return  list;
    }
```

- 实现具体登录业务

  首先拿到用户名先去 DB 查找有没有该用户？

  - 存在？
    - 存在：则验证密码是否正确？
      - 正确：生成业务 token 和刷新 token 响应客户端。
      - 不正确：抛出相应异常。
    - 不存在：则抛出运行时异常，响应客户端。
  - SysUserMapper.xml 创建 getUserInfoByName 方法

```
    <!-- @Description:     根据用户名获取用户信息-->
    <!-- @Author:          小霍-->
    <select id="getUserInfoByName" resultMap="BaseResultMap">
      SELECT <include refid="Base_Column_List"></include>
      FROM sys_user
      WHERE username=#{username}
      AND deleted=1
    </select>
```

- SysUsermapper.java 创建相同的名称方法

```
SysUser getUserInfoByName(String username);
```

- 实现登录业务

  新增密码校验工具类 PasswordEncoder、PasswordUtils

```
package com.yingxue.lesson.utils;

import java.security.MessageDigest;

/**
* @ClassName:       PasswordEncoder
*                   密码加密
* @Author:          小霍
* @CreateDate:      2019/9/7 13:45
* @UpdateUser:      小霍
* @UpdateDate:      2019/9/7 13:45
* @Version:         0.0.1
*/
public class PasswordEncoder {

    private final static String[] hexDigits = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
"a", "b", "c", "d",
            "e", "f" };

    private final static String MD5 = "MD5";
    private final static String SHA = "SHA";

    private Object salt;
    private String algorithm;

    public PasswordEncoder(Object salt) {
        this(salt, MD5);
    }

    public PasswordEncoder(Object salt, String algorithm) {
        this.salt = salt;
        this.algorithm = algorithm;
    }

    /**
     * 密码加密
     * @param rawPass
     * @return
     */
    public String encode(String rawPass) {
        String result = null;
        try {
            MessageDigest md = MessageDigest.getInstance(algorithm);
            // 加密后的字符串
            result = byteArrayToHexString(md.digest(mergePasswordAndSalt(rawPass).getBytes("utf-
8")));
        } catch (Exception ex) {
        }
        return result;
    }

    /**
     * 密码匹配验证
     * @param encPass 密文
```

```
     * @param rawPass 明文
     * @return
     */
    public boolean matches(String encPass, String rawPass) {
        String pass1 = "" + encPass;
        String pass2 = encode(rawPass);

        return pass1.equals(pass2);
    }

    private String mergePasswordAndSalt(String password) {
        if (password == null) {
            password = "";
        }

        if ((salt == null) || "".equals(salt)) {
            return password;
        } else {
            return password + "{" + salt.toString() + "}";
        }
    }

    /**
     * 转换字节数组为16进制字串
     *
     * @param b
     *            字节数组
     * @return 16进制字串
     */
    private String byteArrayToHexString(byte[] b) {
        StringBuffer resultSb = new StringBuffer();
        for (int i = 0; i < b.length; i++) {
            resultSb.append(byteToHexString(b[i]));
        }
        return resultSb.toString();
    }

    /**
     * 将字节转换为16进制
     * @param b
     * @return
     */
    private static String byteToHexString(byte b) {
        int n = b;
        if (n < 0)
            n = 256 + n;
        int d1 = n / 16;
        int d2 = n % 16;
        return hexDigits[d1] + hexDigits[d2];
    }

    public static void main(String[] args) {

    }


}
```

```
package com.yingxue.lesson.utils;

import java.util.UUID;

/**
* @ClassName:       PasswordUtils
*                   密码工具类
* @Author:          小霍
* @CreateDate:      2019/9/7 13:44
* @UpdateUser:      小霍
* @UpdateDate:      2019/9/7 13:44
* @Version:         0.0.1
*/
public class PasswordUtils {
```

```
    /**
     * 匹配密码
     * @param salt 盐
     * @param rawPass 明文
     * @param encPass 密文
     * @return
     */
    public static boolean matches(String salt, String rawPass, String encPass) {
        return new PasswordEncoder(salt).matches(encPass, rawPass);
    }

    /**
     * 明文密码加密
     * @param rawPass 明文
     * @param salt
     * @return
     */
    public static String encode(String rawPass, String salt) {
        return new PasswordEncoder(salt).encode(rawPass);
    }

    /**
     * 获取加密盐
     * @return
     */
    public static String getSalt() {
        return UUID.randomUUID().toString().replaceAll("-", "").substring(0, 20);
    }
}
```

修改 UserServiceImpl.java 加入如下代码

```
@Override
    public LoginRespVO login(LoginReqVO vo) {
        SysUser sysUser=sysUserMapper.getUserInfoByName(vo.getUsername());
        if (null==sysUser){
            throw new BusinessException(BaseResponseCode.NOT_ACCOUNT);
        }
        if (sysUser.getStatus()==2){
            throw new BusinessException(BaseResponseCode.USER_LOCK);
        }
        if(!PasswordUtils.matches(sysUser.getSalt(),vo.getPassword(),sysUser.getPassword())){
            throw new BusinessException(BaseResponseCode.PASSWORD_ERROR);
        }
        LoginRespVO respVO=new LoginRespVO();
        BeanUtils.copyProperties(sysUser,respVO);
        Map<String,Object> claims=new HashMap<>();
        claims.put(Constant.JWT_PERMISSIONS_KEY,getPermissionsByUserId(sysUser.getId()));
        claims.put(Constant.JWT_ROLES_KEY,getRolesByUserId(sysUser.getId()));
        claims.put(Constant.JWT_USER_NAME,sysUser.getUsername());
        String access_token=JwtTokenUtil.getAccessToken(sysUser.getId(),claims);
        String refresh_token;
        if(vo.getType().equals("1")){
            refresh_token=JwtTokenUtil.getRefreshToken(sysUser.getId(),claims);
        }else {
            refresh_token=JwtTokenUtil.getRefreshAppToken(sysUser.getId(),claims);
        }
        respVO.setAccessToken(access_token);
        respVO.setRefreshToken(refresh_token);
        return respVO;
    }
```

7.6 实现登出业务

- 创建 登录接口

```
/**
     * 退出登录
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param accessToken
    • * @param refreshToken
     * @return       void
     * @throws
     */
    void logout(String accessToken,String refreshToken);
```

- Contants 加入几个静态常量

```
/**
     * refresh_token 主动退出后加入黑名单 key
     */
    public static final String JWT_REFRESH_TOKEN_BLACKLIST="jwt-refresh-token-blacklist_";
    /**
     * access_token 主动退出后加入黑名单 key
     */
    public static final String JWT_ACCESS_TOKEN_BLACKLIST="jwt-access-token-blacklist_";
```

- 实现登出的具体业务

  首先要调用 subject.logout(); 这个主要是清空 shiro 的一些缓存信息，然后就是我们系统具体业务了 把 access_token 加入黑名单、refresh_token 加入黑名单。

```
@Override
    public void logout(String accessToken, String refreshToken) {
        if(StringUtils.isEmpty(accessToken)||StringUtils.isEmpty(refreshToken)){
            throw new BusinessException(BaseResponseCode.DATA_ERROR);
        }
        Subject subject = SecurityUtils.getSubject();
        log.info("subject.getPrincipals()={}",subject.getPrincipals());
        if (subject.isAuthenticated()) {
            subject.logout();
        }
        String userId=JwtTokenUtil.getUserId(accessToken);
        /**
         * 把token 加入黑名单 禁止再登录
         */

 redisService.set(Constant.JWT_ACCESS_TOKEN_BLACKLIST+accessToken,userId,JwtTokenUtil.getRemaining
Time(accessToken),TimeUnit.MILLISECONDS);
        /**
         * 把 refreshToken 加入黑名单 禁止再拿来刷新token
         */

 redisService.set(Constant.JWT_REFRESH_TOKEN_BLACKLIST+refreshToken,userId,JwtTokenUtil.getRemaini
ngTime(refreshToken),TimeUnit.MILLISECONDS);


    }
```

7.7 创建业务接口

- 创建 登录接口

```
package com.yingxue.lesson.controller;



import com.yingxue.lesson.constants.Constant;
import com.yingxue.lesson.exception.code.BaseResponseCode;
import com.yingxue.lesson.service.UserService;
import com.yingxue.lesson.utils.DataResult;
import com.yingxue.lesson.vo.req.LoginReqVO;
import com.yingxue.lesson.vo.resp.LoginRespVO;
import io.swagger.annotations.Api;
```

```
import io.swagger.annotations.ApiOperation;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;



/**
 * @ClassName: UserController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@Api(tags = "组织模块-用户管理")
@RequestMapping("/api")
public class UserController {
    @Autowired
    private UserService userService;


    @PostMapping("/user/login")
    @ApiOperation(value = "用户登录接口")
    public DataResult<LoginRespVO> login(@RequestBody LoginReqVO vo){
        DataResult<LoginRespVO> result=DataResult.success();
        result.setData(userService.login(vo));
        return result;
    }
}
```

- 静态 Constant 类新增两个静态常量

```
/**
     * 正常token
     */
    public static final String ACCESS_TOKEN="authorization";
    /**
     * 刷新token
     */
    public static final String REFRESH_TOKEN="refresh_token";
```

- 创建 用户登出接口

```
    @GetMapping("/user/logout")
    @ApiOperation(value = "用户登出接口")
    public DataResult logout(HttpServletRequest request){
        try {
            String accessToken=request.getHeader(Constant.ACCESS_TOKEN);
            String refreshToken=request.getHeader(Constant.REFRESH_TOKEN);
            userService.logout(accessToken,refreshToken);
        } catch (Exception e) {
            log.error("logout error{}",e);
        }
        return DataResult.success();
    }
```

- 创建引导客户端去登录接口

```
    @GetMapping("/user/unLogin")
    @ApiOperation(value = "引导客户端去登录")
    public DataResult unLogin(){
        DataResult result= DataResult.getResult(BaseResponseCode.TOKEN_ERROR);
        return result;
    }
```

8. mybatis-使用 pagehelper 实现分页封装

8.1 概述

在第一节课的时候我们已经把 pagehelper 插件的依赖引入项目了，接下来我们主要讲解怎么去封装使用 pagehelper 分页。

8.2 配置 pagehelper

在application.properties中添加pagehelper配置

```
pagehelper.helperDialect=mysql
pagehelper.reasonable=true
```

`helperDialect`：分页插件会自动检测当前的数据库链接，自动选择合适的分页方式。你也可以配置helperDialect属性来指定分页插件使用哪种方言。

`reasonable`：分页合理化参数，默认值为false。当该参数设置为 true 时，pageNum<=0 时会查询第一页， pageNum>pages（超过总数时），会查询最后一页。默认false 时，直接根据参数进行查询。

8.3 使用方法

```
        //Mapper接口方式的调用，推荐这种使用方式。
        PageHelper.startPage(vo.getPageNum(),vo.getPageSize());
        List<SysRole> sysRoles =sysRoleMapper.selectAll(vo);
```

8.4 实战封装

8.4.1 创建响应 VO

```
package com.yingxue.lesson.vo.resp;

import io.swagger.annotations.ApiModelProperty;
import io.swagger.annotations.ApiParam;
import lombok.Data;

import java.util.List;

/**
 * @ClassName: PageVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class PageVO <T>{

    /**
     * 总记录数
     */
    @ApiModelProperty(value = "总记录数")
    private Long totalRows;

    /**
     * 总页数
     */
    @ApiModelProperty(value = "总页数")
    private Integer totalPages;

    /**
     * 当前第几页
     */
```

```
    @ApiModelProperty(value = "当前第几页")
    private Integer pageNum;
    /**
     * 每页记录数
     */
    @ApiModelProperty(value = "每页记录数")
    private Integer pageSize;
    /**
     * 当前页记录数
     */
    @ApiModelProperty(value = "当前页记录数")
    private Integer curPageSize;
    /**
     * 数据列表
     */
    @ApiModelProperty(value = "数据列表")
    private List<T> list;
}
```

8.4.2 创建 分页工具类

```
package com.yingxue.lesson.utils;

import com.github.pagehelper.Page;
import com.yingxue.lesson.vo.resp.PageVO;

import java.util.List;

/**
 * @ClassName: PageUtil
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class PageUtil {
    private PageUtil(){}
    public static <T> PageVO<T> getPageVO(List<T> list){
        PageVO<T> result=new PageVO<>();
        if(list instanceof Page){
            Page<T> page= (Page<T>) list;
            result.setTotalRows(page.getTotal());
            result.setTotalPages(page.getPages());
            result.setPageNum(page.getPageNum());
            result.setCurPageSize(page.getPageSize());
            result.setPageSize(page.size());
            result.setList(page.getResult());
        }
        return result;
    }
}
```

8.5 代码实现

8.5.1 创建接收分页数据条件 VO

```
package com.yingxue.lesson.vo.req;

import io.swagger.annotations.ApiModelProperty;
import lombok.Data;

/**
 * @ClassName: UserPageReqVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class UserPageReqVO {
```

```
    @ApiModelProperty(value = "当前第几页")
    private Integer pageNum=1;

    @ApiModelProperty(value = "当前页数量")
    private Integer pageSize=10;

}
```

8.5.2 创建服务层业务接口

1. 接口

```
/**
     * 分页查询用户信息
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param vo
     * @return       com.yingxue.lesson.vo.resp.PageVO<com.yingxue.lesson.entity.SysUser>
     * @throws
     */
    PageVO<SysUser> pageInfo(UserPageReqVO vo);
```

2. dao 层接口

```
List<SysUser> selectAll(UserPageReqVO vo);
```

3. 动态 sql 语句

```
 <!-- @Description:     查询所有用户-->
  <!-- @Author:          小霍-->
  <select id="selectAll" resultMap="BaseResultMap"
parameterType="com.yingxue.lesson.vo.req.UserPageReqVO">
    select <include refid="Base_Column_List"></include>
    from sys_user
    <where>
      deleted=1
    </where>
  </select>
```

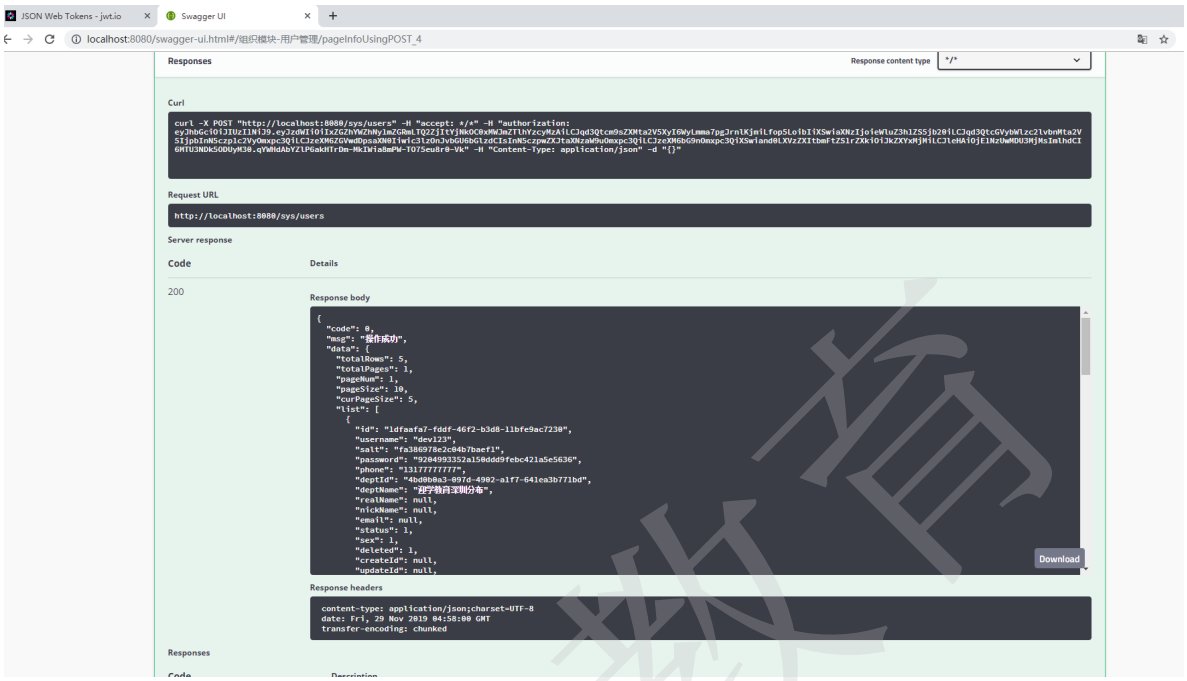
4. 分页接口实现类

```
 /**
     * 分页查询用户信息
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param vo
     * @return       com.yingxue.lesson.vo.resp.PageVO<com.yingxue.lesson.entity.SysUser>
     * @throws
     */
    @Override
    public PageVO<SysUser> pageInfo(UserPageReqVO vo) {
        PageHelper.startPage(vo.getPageNum(),vo.getPageSize());
        List<SysUser> sysUsers = sysUserMapper.selectAll(vo);
        return PageUtils.getPageVO(sysUsers);
    }
```

5. 控制层接口

```
    @PostMapping("/users")
    @ApiOperation(value = "分页获取用户列表接口")
    public DataResult<PageVO<SysUser>> pageInfo(@RequestBody UserPageReqVO vo){
        DataResult<PageVO<SysUser>> result=DataResult.success();
        result.setData(userService.pageInfo(vo));
        return result;
    }
```

8.6 测试结果



9. Spring Boot+Shiro+JWT+redis 前后端分离脚手架-自定义AccessControlFilter token认证

9.1 自定义UsernamePasswordToken

```
package com.yingxue.lesson.shiro;

import org.apache.shiro.authc.UsernamePasswordToken;

/**
 * @ClassName: CustomPasswordToken
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class CustomPasswordToken extends UsernamePasswordToken {
    private String token;

    public CustomPasswordToken(String token) {
        this.token = token;
    }

    @Override
    public Object getPrincipal() {
        return token;
    }
}
```

9.2 自定义 token 过滤器AccessControlFilter

这个类主要是拦截需求认证的请求，首先验证客户端 header 是否携带了 token ，如果没有携带直接响应客户端，引导客户端到登录界面进行登录操作，如果客户端 header 已经携带有 token 放开进入 shiro SecurityManager 验证

- 自定义 CustomAccessControlFilter

```
package com.yingxue.lesson.shiro;

import com.alibaba.fastjson.JSON;
import com.yingxue.lesson.constants.Constant;
import com.yingxue.lesson.exception.BusinessException;
import com.yingxue.lesson.exception.code.BaseResponseCode;
import com.yingxue.lesson.utils.DataResult;
import lombok.extern.slf4j.Slf4j;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.web.filter.AccessControlFilter;
import org.springframework.util.StringUtils;

import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.IOException;
import java.io.OutputStream;

/**
 * @ClassName: CustomAccessControlerFilter
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Slf4j
public class CustomAccessControlerFilter extends AccessControlFilter {
    @Override
    protected boolean isAccessAllowed(ServletRequest servletRequest, ServletResponse
servletResponse, Object o) throws Exception {
        return false;
    }

    @Override
    protected boolean onAccessDenied(ServletRequest servletRequest, ServletResponse
servletResponse) throws Exception {
        HttpServletRequest request= (HttpServletRequest) servletRequest;
        log.info(request.getMethod());
        log.info(request.getRequestURL().toString());
        //判断客户端是否携带accessToken
        try {
            String accessToken=request.getHeader(Constant.ACCESS_TOKEN);
            if(StringUtils.isEmpty(accessToken)){
                throw new BusinessException(BaseResponseCode.TOKEN_NOT_NULL);
            }
            CustomUsernamePasswordToken customUsernamePasswordToken=new
CustomUsernamePasswordToken(accessToken);
            getSubject(servletRequest,servletResponse).login(customUsernamePasswordToken);
        } catch (BusinessException e) {
            customRsponse(e.getCode(),e.getDefaultMessage(),servletResponse);
            return false;
        } catch (AuthenticationException e) {
            if(e.getCause() instanceof BusinessException){
                BusinessException exception= (BusinessException) e.getCause();
                customRsponse(exception.getCode(),exception.getDefaultMessage(),servletResponse);
            }else {

 customRsponse(BaseResponseCode.SHIRO_AUTHENTICATION_ERROR.getCode(),BaseResponseCode.SHIRO_AUTHENT
ICATION_ERROR.getMsg(),servletResponse);
            }
           return false;
        }
        return true;
    }

    /**
     * 自定义错误响应
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param code
    • * @param msg
    • * @param response
     * @return       void
```

```
     * @throws
     */
    private void customRsponse(int code, String msg, ServletResponse response){
        // 自定义异常的类，用户返回给客户端相应的JSON格式的信息
        try {
            DataResult result=DataResult.getResult(code,msg);
            response.setContentType("application/json; charset=utf-8");
            response.setCharacterEncoding("UTF-8");

            String userJson = JSON.toJSONString(result);
            OutputStream out = response.getOutputStream();
            out.write(userJson.getBytes("UTF-8"));
            out.flush();
        } catch (IOException e) {
            log.error("eror={}",e);
        }
    }
}
```

9.3 自定义 Shiro CredentialsMatcher

因为客户端首次登录后，后续的操作用户可以不在输入用户名密码，直接拿 token 凭证来验证用户，所以我们得改造一下 shiro 验证器，把它改造成验证 token 是否有效的业务逻辑。

首先要创建 几个静态常量 Contants

```
/**
 * 主动去刷新 token key(适用场景 比如修改了用户的角色/权限去刷新token)
 */
public static final String JWT_REFRESH_KEY="jwt-refresh-key_";
/**
 * 标记新的access_token
 */
public static final String JWT_REFRESH_IDENTIFICATION="jwt-refresj-identification_";

/**
 * access_token 主动退出后加入黑名单 key
 */
public static final String JWT_ACCESS_TOKEN_BLACKLIST="jwt-access-token-blacklist_";

/**
 * 标记用户是否已经被锁定
 */
public static final String ACCOUNT_LOCK_KEY="account-lock-key_";
/**
 * 标记用户是否已经删除
 */
public static final String DELETED_USER_KEY="deleted-user-key_";
```

创建自定义 CustomHashedCredentialsMatcher 类继承 HashedCredentialsMatcher从写 doCredentialsMatch 方法。

主要业务逻辑：

第一步：判断用户是否被锁定。

- 否：下一步。
- 是：引导到登录界面。

第二步：判断用户是否被删除。

- 否：下一步。
- 是：引导到登录界面。

第三步：判断用户是否是否主动退出(用户主动退出后端会把 Contants.JWT_ACCESS_TOKEN_BLACKLIST+access_token 作为 key 存入 redis 并且设置过期时间为 access_token 剩余的过期时间)

- 否：下一步。
- 是：引导到登录界面。

第四步：判断access_token 是否通过校验(校验是否过期)

- 否：引导到登录界面。

- 是：下一步。

第五步：判断用户是否需要刷新(因为后台修改了用户所拥有的角色/菜单权限的时候需要把相关联用户都用redis标记起来(过期时间为access_token 生成的过期时间)，需要刷新access_token重新分配角色)但是呢需要排除重新登录的用户所以呢还要比较这个accessToken和Constant.JWT_REFRESH_KEY+userId两者的剩余过期时间。

- 满足条件：下一步。
- 不满足：放行 处理相关业务。

```
package com.yingxue.lesson.shiro;

import com.yingxue.lesson.constants.Constant;
import com.yingxue.lesson.exception.BusinessException;
import com.yingxue.lesson.exception.code.BaseResponseCode;
import com.yingxue.lesson.service.RedisService;
import com.yingxue.lesson.utils.JwtTokenUtil;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.credential.HashedCredentialsMatcher;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.concurrent.TimeUnit;
import java.util.prefs.BackingStoreException;

/**
 * @ClassName: CustomHashedCredentialsMatcher
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class CustomHashedCredentialsMatcher extends HashedCredentialsMatcher {
    @Autowired
    private RedisService redisService;
    @Override
    public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {
        CustomUsernamePasswordToken customUsernamePasswordToken= (CustomUsernamePasswordToken) token;
        String accessToken= (String) customUsernamePasswordToken.getPrincipal();
        String userId= JwtTokenUtil.getUserId(accessToken);
        /**
         * 判断用户是否被锁定
         */
        if(redisService.hasKey(Constant.ACCOUNT_LOCK_KEY+userId)){
            throw new BusinessException(BaseResponseCode.ACCOUNT_LOCK);
        }
        /**
         * 判断用户是否被删除
         */
        if(redisService.hasKey(Constant.DELETED_USER_KEY+userId)){
            throw new BusinessException(BaseResponseCode.ACCOUNT_HAS_DELETED_ERROR);
        }

        /**
         * 判断token 是否主动登出
         */
        if(redisService.hasKey(Constant.JWT_ACCESS_TOKEN_BLACKLIST+accessToken)){
            throw new BusinessException(BaseResponseCode.TOKEN_ERROR);
        }
        /**
         * 判断token是否通过校验
         */
        if(!JwtTokenUtil.validateToken(accessToken)){
            throw new BusinessException(BaseResponseCode.TOKEN_PAST_DUE);
        }
        /**
         * 判断这个登录用户是否要主动去刷新
         *
         * 如果 key=Constant.JWT_REFRESH_KEY+userId大于accessToken说明是在 accessToken不是重新生成的
         * 这样就要判断它是否刷新过了/或者是否是新生成的token
         */
//
if(redisService.hasKey(Constant.JWT_REFRESH_KEY+userId)&&redisService.getExpire(Constant.JWT_REFRESH_KEY
+userId, TimeUnit.MILLISECONDS)>JwtTokenUtil.getRemainingTime(accessToken)){
//            /**
```

```
//             * 是否存在刷新的标识
//             */
//            if(!redisService.hasKey(Constant.JWT_REFRESH_IDENTIFICATION+accessToken)){
//                throw new BusinessException(BaseResponseCode.TOKEN_PAST_DUE);
//            }
//        }
        if(redisService.hasKey(Constant.JWT_REFRESH_KEY+userId)){
            /**
             * 通过剩余的过期时间比较如果token的剩余过期时间大与标记key的剩余过期时间
             * 就说明这个tokne是在这个标记key之后生成的
             */
            if(redisService.getExpire(Constant.JWT_REFRESH_KEY+userId,
TimeUnit.MILLISECONDS)>JwtTokenUtil.getRemainingTime(accessToken)){
                throw new BusinessException(BaseResponseCode.TOKEN_PAST_DUE);
            }
        }
        return true;
    }
}
```

10 Spring Boot+Shiro+JWT+redis 前后端分离脚手架-自定义 Realm

主要是继承 AuthorizingRealm 实现两个比较关键的方法 doGetAuthorizationInfo(主要用于用户授权，就是设置用户所拥有的角色/权限)、doGetAuthenticationInfo(主要用户用户的认证，以前是验证用户名密码这里我们会改造成验证 token 一般来说客户端只需登录一次后续的访问用 token来维护登录的状态，所以我们这里改造成严正 token)

```
package com.yingxue.lesson.shiro;

import com.yingxue.lesson.constants.Constant;
import com.yingxue.lesson.exception.BusinessException;
import com.yingxue.lesson.exception.code.BaseResponseCode;
import com.yingxue.lesson.service.PermissionService;
import com.yingxue.lesson.service.RedisService;
import com.yingxue.lesson.service.RoleService;
import com.yingxue.lesson.service.UserService;
import com.yingxue.lesson.utils.JwtTokenUtil;
import io.jsonwebtoken.Claims;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.Collection;
import java.util.List;
import java.util.Set;
import java.util.concurrent.TimeUnit;

/**
 * @ClassName: CustomRealm
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class CustomRealm extends AuthorizingRealm {
    @Autowired
    private RoleService roleService;
    @Autowired
    private PermissionService permissionService;
    @Autowired
    private RedisService redisService;
    @Override
    public boolean supports(AuthenticationToken token) {
        return token instanceof CustomUsernamePasswordToken;
    }
```

```
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {
        String accessToken= (String) principalCollection.getPrimaryPrincipal();
        SimpleAuthorizationInfo info=new SimpleAuthorizationInfo();
        String userId=JwtTokenUtil.getUserId(accessToken);
        /**
         * 通过剩余的过期时间比较如果token的剩余过期时间大与标记key的剩余过期时间
         * 就说明这个tokne是在这个标记key之后生成的
         */

 if(redisService.hasKey(Constant.JWT_REFRESH_KEY+userId)&&redisService.getExpire(Constant.JWT_REFRESH_KE
Y+userId, TimeUnit.MILLISECONDS)>JwtTokenUtil.getRemainingTime(accessToken)){
            //List<String> roleNames = roleService.getRoleNames(userId);
            List<String> roleNames = getRolesByUserId(userId);
            if(roleNames!=null&&!roleNames.isEmpty()){
                info.addRoles(roleNames);
            }
            //Set<String> permissions=permissionService.getPermissionsByUserId(userId);
            List<String> permissions=getPermissionsByUserId(userId);
            if(permissions!=null){
                info.addStringPermissions(permissions);
            }

        }else {
            Claims claims= JwtTokenUtil.getClaimsFromToken(accessToken);
            /**
             * 返回该用户的角色信息给授权期
             */
            if(claims.get(Constant.JWT_ROLES_KEY)!=null){
                info.addRoles((Collection<String>) claims.get(Constant.JWT_ROLES_KEY));
            }

            /**
             * 返回该用户的权限信息给授权器
             */
            if(claims.get(Constant.JWT_PERMISSIONS_KEY)!=null){
                info.addStringPermissions((Collection<String>)
claims.get(Constant.JWT_PERMISSIONS_KEY));
            }
        }


        return info;
    }

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws
AuthenticationException {
        CustomUsernamePasswordToken customUsernamePasswordToken= (CustomUsernamePasswordToken)
authenticationToken;
        SimpleAuthenticationInfo info=new
SimpleAuthenticationInfo(customUsernamePasswordToken.getPrincipal(),customUsernamePasswordToken.getCrede
ntials(),CustomRealm.class.getName());
        return info;
    }

     /**
     * 获取用户的角色
     * 这里先用伪代码代替
     * 后面我们讲到权限管理系统后 再从 DB 读取
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param userId
     * @return
     * @throws
     */
    private List<String> getRolesByUserId(String userId){
        List<String> list=new ArrayList<>();
        if("9a26f5f1-cbd2-473d-82db-1d6dcf4598f8".equals(userId)){
            list.add("admin");
        }else{
            list.add("test");
        }
```

```
        return  list;
    }

    /**
     * 获取用户的权限
     * 这里先用伪代码代替
     * 后面我们讲到权限管理系统后 再从 DB 读取
     * @Author:       小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param userId
     * @return
     * @throws
     */
    private List<String>getPermissionsByUserId(String userId){
        List<String> list=new ArrayList<>();
        if("9a26f5f1-cbd2-473d-82db-1d6dcf4598f8".equals(userId)){
            list.add("sys:user:add");
            list.add("sys:user:list");
            list.add("sys:user:update");
            list.add("sys:user:detail");
        }else{
            list.add("sys:user:list");
        }
        return  list;
    }
}
```

11 Spring Boot+Shiro+JWT+redis 前后端分离脚手架-shiro 核心配置

shiro 的配置主要有 Realm、securityManager、shiroFilterFactoryBean 三个关键的配置。

- 创建 ShiroConfig 配置类

```
/**
 * @ClassName: ShiroConfig
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Configuration
public class ShiroConfig {



}
```

- 注入自定义 CredentialsMatcher

  修改 ShiroConfig.java 加入如下代码

```
/**
     * 自定义密码 校验
     * @Author:       小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.yingxue.lesson.shiro.CustomHashedCredentialsMatcher
     * @throws
     */
    @Bean
    public CustomHashedCredentialsMatcher customHashedCredentialsMatcher(){
        return new CustomHashedCredentialsMatcher();
    }
```

- 注入自定义 Realm

  修改 ShiroConfig.java 加入如下代码

```
/**
     * 自定义域
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.yingxue.lesson.shiro.CustomRealm
     * @throws
     */
    @Bean
    public CustomRealm customRealm(){
        CustomRealm customRealm=new CustomRealm();
        customRealm.setCredentialsMatcher(customHashedCredentialsMatcher());
        return customRealm;
    }
```

- 注入 securityManager

  修改 ShiroConfig.java 加入如下代码

```
/**
     * 安全管理
     * @Author:      小霍
     * @Version:     0.0.1
     * @param
     * @return       org.apache.shiro.mgt.SecurityManager(包不要引错)
     * @throws
     */
    @Bean
    public SecurityManager securityManager(){
        DefaultWebSecurityManager securityManager=new DefaultWebSecurityManager();
        securityManager.setRealm(customRealm());
        return securityManager;
    }
```

- 注入 shiro 过滤器

  这里主要是配置一些要方行的 url 要拦截认证的 url

  修改 ShiroConfig.java 加入如下代码

```
/**
     * shiro过滤器，配置拦截哪些请求
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param securityManager
     * @return       org.apache.shiro.spring.web.ShiroFilterFactoryBean
     * @throws
     */
    @Bean
    public ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager){
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        //自定义拦截器限制并发人数,参考博客：
        LinkedHashMap<String, Filter> filtersMap = new LinkedHashMap<>();
        //用来校验token
        filtersMap.put("token", new CustomAccessControlFilter());
        shiroFilterFactoryBean.setFilters(filtersMap);
        Map<String, String> filterChainDefinitionMap = new LinkedHashMap<>();
        // 配置不会被拦截的链接 顺序判断
        filterChainDefinitionMap.put("/api/user/login", "anon");
        //放开swagger-ui地址
        filterChainDefinitionMap.put("/swagger/**", "anon");
        filterChainDefinitionMap.put("/v2/api-docs", "anon");
        filterChainDefinitionMap.put("/swagger-ui.html", "anon");
        filterChainDefinitionMap.put("/swagger-resources/**", "anon");
        filterChainDefinitionMap.put("/webjars/**", "anon");
        filterChainDefinitionMap.put("/druid/**", "anon");
        filterChainDefinitionMap.put("/favicon.ico", "anon");
        filterChainDefinitionMap.put("/captcha.jpg", "anon");
        filterChainDefinitionMap.put("/","anon");
```

```
        filterChainDefinitionMap.put("/csrf","anon");
        filterChainDefinitionMap.put("/**","token,authc");
        //配置shiro默认登录界面地址，前后端分离中登录界面跳转应由前端路由控制，后台仅返回json数据
        shiroFilterFactoryBean.setLoginUrl("/api/user/unLogin");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
        return shiroFilterFactoryBean;
    }
```

- 开启 aop 注解支持

  修改 ShiroConfig.java 加入如下代码

```
    /**
     * 开启shiro aop注解支持.
     * 使用代理方式;所以需要开启代码支持;
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param securityManager
     * @return
org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor
     * @throws
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor(SecurityManager
securityManager) {
        AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new
AuthorizationAttributeSourceAdvisor();
        authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
        return authorizationAttributeSourceAdvisor;
    }
    @Bean
    @ConditionalOnMissingBean
    public DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator() {
        DefaultAdvisorAutoProxyCreator defaultAdvisorAutoProxyCreator = new
DefaultAdvisorAutoProxyCreator();
        defaultAdvisorAutoProxyCreator.setProxyTargetClass(true);
        return defaultAdvisorAutoProxyCreator;
    }
```

12. Spring Boot+Shiro+JWT+redis 前后端分离脚手架-redis 缓存授权信息

12.1 自定义一个缓存工具类

RedisCache<K, V> 实现 shiro Cache<K, V> 缓存接口，并重写 Cache<K, V> get、put、remove、clear、size、keys、values等方法，这些方法都是 shiro 在对缓存的一些操作，就是当 shiro 操作缓存的时候都会调用相应的方法，我们只需重写这些相应的方法就可以把 shiro 的缓存信息存入到 redis 了。这就是一个优秀的开源框架所具备的扩展性，它提供了一个cacheManager 缓存管理器 我们只需重新这个管理器即可。

```
package com.yingxue.lesson.shiro;

import com.alibaba.fastjson.JSON;
import com.yingxue.lesson.constants.Constant;
import com.yingxue.lesson.service.RedisService;
import com.yingxue.lesson.utils.JwtTokenUtil;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.collections.CollectionUtils;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.cache.Cache;
import org.apache.shiro.cache.CacheException;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.*;
import java.util.concurrent.TimeUnit;

/**
 * @ClassName: RedisCache
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
```

```
 */
@Slf4j
public class RedisCache<K,V> implements Cache<K,V> {
    private String cacheKey;
    private long expire = 24;
    private RedisService redisService;
    public RedisCache(RedisService redisService){
        this.cacheKey= Constant.IDENTIFY_CACHE_KEY;
        this.redisService=redisService;
    }
    @Override
    public V get(K key) throws CacheException {
        log.info("Shiro从缓存中获取数据KEY值[{}]",key);
        if (key == null) {
            return null;
        }
        try {
            String redisCacheKey = getRedisCacheKey(key);
            Object rawValue = redisService.get(redisCacheKey);
            if (rawValue == null) {
                return null;
            }
            SimpleAuthorizationInfo simpleAuthenticationInfo=
JSON.parseObject(rawValue.toString(),SimpleAuthorizationInfo.class);
            V value = (V) simpleAuthenticationInfo;
            return value;
        } catch (Exception e) {
            throw new CacheException(e);
        }
    }

    @Override
    public V put(K key, V value) throws CacheException {
        log.info("put key [{}]",key);
        if (key == null) {
            log.warn("Saving a null key is meaningless, return value directly without call Redis.");
            return value;
        }
        try {
            String redisCacheKey = getRedisCacheKey(key);
            redisService.set(redisCacheKey, value != null ? value : null, expire, TimeUnit.HOURS);
            return value;
        } catch (Exception e) {
            throw new CacheException(e);
        }
    }

    @Override
    public V remove(K key) throws CacheException {
        log.info("remove key [{}]",key);
        if (key == null) {
            return null;
        }
        try {
            String redisCacheKey = getRedisCacheKey(key);
            Object rawValue = redisService.get(redisCacheKey);
            V previous = (V) rawValue;
            redisService.delete(redisCacheKey);
            return previous;
        } catch (Exception e) {
            throw new CacheException(e);
        }
    }

    @Override
    public void clear() throws CacheException {
        log.debug("clear cache");
        Set<String> keys = null;
        try {
            keys = redisService.keys(this.cacheKey + "*");
        } catch (Exception e) {
            log.error("get keys error", e);
        }
        if (keys == null || keys.size() == 0) {
```

```
            return;
        }
        for (String key: keys) {
            redisService.delete(key);
        }
    }

    @Override
    public int size() {
        int result = 0;
        try {
            result = redisService.keys(this.cacheKey + "*").size();
        } catch (Exception e) {
            log.error("get keys error", e);
        }
        return result;
    }

    @SuppressWarnings("unchecked")
    @Override
    public Set<K> keys() {
        Set<String> keys = null;
        try {
            keys = redisService.keys(this.cacheKey + "*");
        } catch (Exception e) {
            log.error("get keys error", e);
            return Collections.emptySet();
        }
        if (CollectionUtils.isEmpty(keys)) {
            return Collections.emptySet();
        }
        Set<K> convertedKeys = new HashSet<>();
        for (String key:keys) {
            try {
                convertedKeys.add((K) key);
            } catch (Exception e) {
                log.error("deserialize keys error", e);
            }
        }
        return convertedKeys;
    }

    @Override
    public Collection<V> values() {
        Set<String> keys = null;
        try {
            keys = redisService.keys(this.cacheKey + "*");
        } catch (Exception e) {
            log.error("get values error", e);
            return Collections.emptySet();
        }
        if (CollectionUtils.isEmpty(keys)) {
            return Collections.emptySet();
        }
        List<V> values = new ArrayList<V>(keys.size());
        for (String key : keys) {
            V value = null;
            try {
                value = (V) redisService.get(key);
            } catch (Exception e) {
                log.error("deserialize values= error", e);
            }
            if (value != null) {
                values.add(value);
            }
        }
        return Collections.unmodifiableList(values);
    }
    private String getRedisCacheKey(K key) {
        if(null==key){
            return null;
        }else {
            return this.cacheKey+ JwtTokenUtil.getUserId(key.toString());
        }
```

```
    }
}
```

12.2 创建一个缓存管理器

RedisCacheManager 实现 CacheManager接口的 getCache 方法

```
package com.yingxue.lesson.shiro;

import com.yingxue.lesson.service.RedisService;
import org.apache.shiro.cache.Cache;
import org.apache.shiro.cache.CacheException;
import org.apache.shiro.cache.CacheManager;
import org.springframework.beans.factory.annotation.Autowired;

/**
 * @ClassName: RedisCacheManager
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class RedisCacheManager implements CacheManager {
    @Autowired
    private RedisService redisService;
    @Override
    public <K, V> Cache<K, V> getCache(String s) throws CacheException {
        return new RedisCache<>(redisService);
    }
}
```

12.3 shiro 配置 redis 缓存

修改 ShiroConfig 配置类

- 注入自定义换成管理器

```
/**
     * 缓存管理器
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.xh.lesson.shiro.RedisCacheManager
     * @throws
     */
    @Bean
    public RedisCacheManager redisCacheManager(){
        return new RedisCacheManager();
    }
```

- 配置redis 缓存管理器

注： 在自定义的Realm 加上

```
customRealm.setCacheManager(redisCacheManager());
```

```
    /**
     * 自定义域
     * @Author:      小霍
     * @UpdateUser:
     * @Version:     0.0.1
     * @param
     * @return       com.xh.lesson.shiro.CustomRealm
     * @throws
     */
    @Bean
    public CustomRealm customRealm(){
        CustomRealm customRealm=new CustomRealm();
        customRealm.setCredentialsMatcher(customHashedCredentialsMatcher());
```

```
        customRealm.setCacheManager(redisCacheManager());
        return customRealm;
    }
```