第四章 Spring Boot 和数据库的技术实践

1. 实战 Mybatis3-多模块实战(module)

MyBatis 是支持普通 SQL查询，存储过程和高级映射的优秀持久层框架。MyBatis 消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。MyBatis 使用简单的 XML或注解配置和原始映射，将接口和 Java 的POJOs（普通的 Java对象）映射成数据库中的记录。

特点：

- 简单易学：本身就很小且简单。没有任何第三方依赖，最简单安装、易于学习，易于使用，通过文档和源代码可以快速上手。
- 灵活：mybatis不会对应用程序或者数据库的原有设计强加任何影响。 sql写在xml里，便于统一管理和优化。通过 sql 语句可以满足操作数据库的所有需求。
- 解除 sql 与程序代码的耦合：通过提供DAO层，将业务逻辑和数据访问逻辑分离，使系统的设计更清晰，更易维护，更易单元测试。
- 提供xml标签，支持编写动态sql。

1.1 集成步骤

1.1.1 配置依赖

```
<!--mybatis -->
<dependency>
<groupId>org.mybatis.spring.boot</groupId>
<artifactId>mybatis-spring-boot-starter</artifactId>
<version>1.3.2</version>
</dependency>
```

1.1.2 基本配置

修改 application .properties加入如下代码

```
# mapper.xml文件位置,如果没有映射文件，请注释掉
mybatis.mapper-locations=classpath:mapper/*.xml
```

1.1.3 接口扫描

1. @Mapper 注解

   需要给每一个Mapper接口添加@Mapper 注解，才能被识别。

   ```
   @Mapper
   public interface SysUserMapper {
   }
   ```

2. @MapperScan 注解

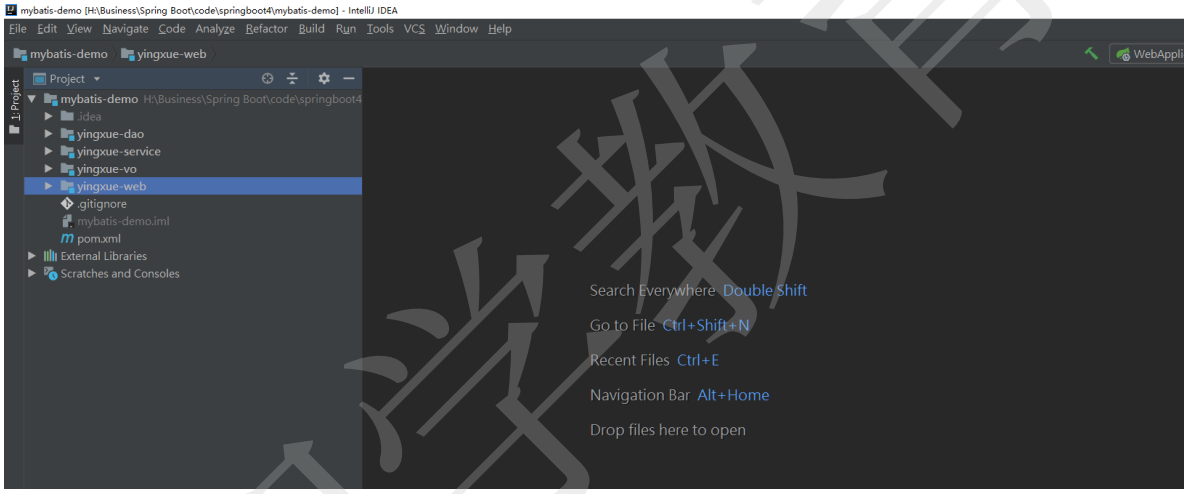
   我们也可以不加注解，而是在启动类上添加扫描包注解

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

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

/**
 * @ClassName: WebApplication
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@SpringBootApplication
@MapperScan("com.yingxue.lesson.mapper")
public class WebApplication {
    public static void main(String[] args) { SpringApplication.run(WebApplication.class,args); }
}
```

1.2 Spring Boot 整合实战 Mybatis3

1.2.1 创建项目

步骤和2-4课程一样先创建一个根项目，删除无用的.mvn目录、src目录、mvnw及mvnw.cmd文件，最终只留.gitignore和pom.xml、然后创建module



1.2.2 项目目录

- yingxue-service：服务层
- yingxue-web：控制层
- yingxue-dao：数据持久层
- yingxue-vo：接收前端&响应端数据封装

1.2.3 整理根项目pom

1. 修改 spring-boot-starter-parent 版本统一2.1.6.RELEASE
2. 删除 dependencies 标签及其中的依赖，因为 Spring Boot 提供的父工程已包含，并且父 pom 原则上都是通过 dependencyManagement 标签管理依赖包。

```xml
<dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-service</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-dao</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-web</artifactId>
```

```
                <version>${project.version}</version>
            </dependency>

            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-vo</artifactId>
                <version>${project.version}</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

3. 删除 build 标签及其中的所有内容，spring-boot-maven-plugin 插件作用是打一个可运行的包，多模块项目仅仅需要在入口类所在的模块添加打包插件，这里父模块不需要打包运行。而且该插件已被包含在 Spring Boot 提供的父工程中，这里删掉即可。
4. 声明管理项目所需jar包

```
    <properties>
        <java.version>1.8</java.version>
        <mysql.version>5.1.47</mysql.version>
        <druid.version>1.1.10</druid.version>
        <mybatis.version>1.3.2</mybatis.version>
        <swagger2-version>2.9.2</swagger2-version>
    </properties>
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>${mysql.version}</version>
            </dependency>
            <dependency>
                <groupId>com.alibaba</groupId>
                <artifactId>druid-spring-boot-starter</artifactId>
                <version>${druid.version}</version>
            </dependency>
            <dependency>
                <groupId>org.mybatis.spring.boot</groupId>
                <artifactId>mybatis-spring-boot-starter</artifactId>
                <version>${mybatis.version}</version>
            </dependency>
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger2</artifactId>
                <version>${swagger2-version}</version>
            </dependency>
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger-ui</artifactId>
                <version>${swagger2-version}</version>
            </dependency>
```

5. 整理后的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>
    <packaging>pom</packaging>
    <modules>
        <module>yingxue-web</module>
        <module>yingxue-service</module>
        <module>yingxue-dao</module>
        <module>yingxue-vo</module>
    </modules>
    <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>mybatis-demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>mybatis-demo</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <java.version>1.8</java.version>
        <mysql.version>5.1.47</mysql.version>
        <druid.version>1.1.10</druid.version>
        <mybatis.version>1.3.2</mybatis.version>
        <swagger2-version>2.9.2</swagger2-version>
    </properties>
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-service</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-dao</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-web</artifactId>
                <version>${project.version}</version>
            </dependency>

            <dependency>
                <groupId>${project.groupId}</groupId>
                <artifactId>yingxue-vo</artifactId>
                <version>${project.version}</version>
            </dependency>
            <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>${mysql.version}</version>
            </dependency>
            <dependency>
                <groupId>org.mybatis.spring.boot</groupId>
                <artifactId>mybatis-spring-boot-starter</artifactId>
                <version>${mybatis.version}</version>
            </dependency>
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger2</artifactId>
                <version>${swagger2-version}</version>
            </dependency>
            <dependency>
                <groupId>io.springfox</groupId>
                <artifactId>springfox-swagger-ui</artifactId>
                <version>${swagger2-version}</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
</project>
```

1.2.4 配置模块间的依赖关系

1. **yingxue-vo**

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

```
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <version>${parent.version}</version>
    <artifactId>yingxue-vo</artifactId>

</project>
```

2. **yingxue-dao**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <parent>
        <artifactId>mybatis-demo</artifactId>
        <groupId>com.yingxue.lesson</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <version>${parent.version}</version>
    <artifactId>yingxue-dao</artifactId>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>com.yingxue.lesson</groupId>
            <artifactId>yingxue-vo</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
        </dependency>
    </dependencies>

</project>
```

3. **yingxue-service**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <parent>
        <artifactId>mybatis-demo</artifactId>
        <groupId>com.yingxue.lesson</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <version>${parent.version}</version>
    <artifactId>yingxue-service</artifactId>

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

```
</project>
```

4. **yingxue-web**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <parent>
        <artifactId>mybatis-demo</artifactId>
        <groupId>com.yingxue.lesson</groupId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>yingxue-web</artifactId>
    <version>${parent.version}</version>

    <dependencies>
        <dependency>
            <groupId>com.yingxue.lesson</groupId>
            <artifactId>yingxue-service</artifactId>
        </dependency>
    </dependencies>

</project>
```

1.2.5 加入热部署devtools依赖

yingxue-web pom文件加入如下代码

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

在spring-boot-maven-plugin 插件加入一个 fork为 true的配置

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

1.2.6 构建数据脚本

```
CREATE DATABASE IF NOT EXISTS test_mybatis DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
use test_mybatis;
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(11) 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 NULL COMMENT '性别(1.男 2.女)',
  `deleted` tinyint(4) DEFAULT '0' COMMENT '是否删除(0未删除；1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 NULL 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;
```

1.2.7 逆向生成代码

1. yingxue-web resources文件下创建 generatorConfig.xml

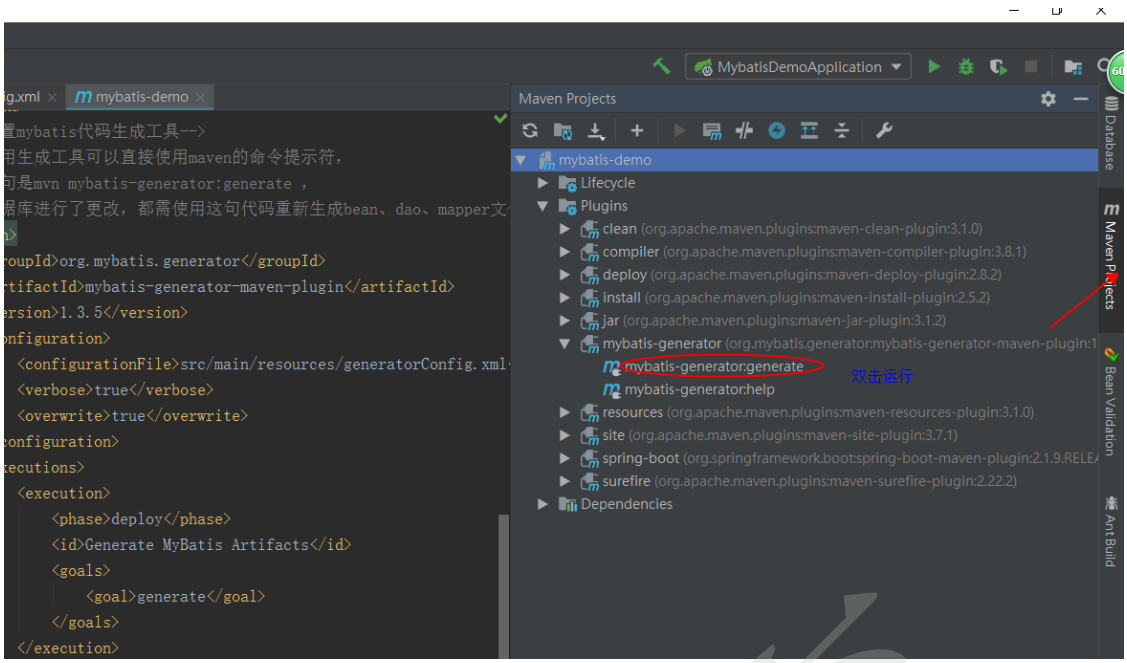
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE generatorConfiguration PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration
1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">
<generatorConfiguration>
    <!--classPathEntry:数据库的JDBC驱动,换成你自己的驱动位置  -->
    <classPathEntry 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/test_mybatis" 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H:\Business\Spring Boot\code\springboot4\mybatis-demo\yingxue-
dao\src\main\java  (要改成你自己实际的目录) -->
        <javaModelGenerator targetPackage="com.yingxue.lesson.entity"
targetProject="H:\Business\Spring Boot\code\springboot4\mybatis-demo\yingxue-dao\src\main\java">
            <property name="enableSubPackages" value="false"/>
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>
        <!-- 生成mapxml文件 H:\Business\Spring Boot\code\springboot4\mybatis-demo\yingxue-
dao\src\main\resources (要改成你自己实际的目录)  -->
        <sqlMapGenerator targetPackage="mapper" targetProject="H:\Business\Spring
Boot\code\springboot4\mybatis-demo\yingxue-dao\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="H:\Business\Spring Boot\code\springboot4\mybatis-demo\yingxue-dao\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>
```

2. yingxue-web 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>
```

3. 运行 mybatis-generator



1.2.8 配置项目

1.2.8.1 加入接口扫描

yingxue-web 创建项目启动类

```
package com.yingxue.lesson.web;

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

/**
 * @ClassName: WebApplication
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@SpringBootApplication(scanBasePackages = "com.yingxue.lesson")
@MapperScan("com.yingxue.lesson.mapper")
public class WebApplication {
    public static void main(String[] args) {
        SpringApplication.run(WebApplication.class,args);
    }
}
```

1.2.8.2 加入数据库配置

yingxue-web resouces文件下创建application .properties

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

```
server.port=8080
spring.application.name=company-frame
#数据库配置
spring.datasource.url=jdbc:mysql://localhost:3306/test_mybatis?serverTimezone=UTC&useUnico
de=true&characterEncoding=utf-8&useSSL=true
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

JDBC连接Mysql5 `com.mysql.jdbc.Driver`

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

1.2.8.3 加入 mybatis 配置

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

1.3 GET 获取用户信息接口

1.3.1 数据库插入一条数据

```
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, '0', NULL, NULL, '3', '2019-09-22 19:38:05', NULL);
```

1.3.2 新增接口方法

yingxue-service创建 UserService接口

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

import com.yingxue.lesson.entity.SysUser;
import com.yingxue.lesson.req.RegisterReqVO;
import com.yingxue.lesson.req.UpdateUserReqVO;

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

    SysUser getUserInfo(String id);

}
```

1.3.3 实现接口方法

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

import com.yingxue.lesson.entity.SysUser;
import com.yingxue.lesson.mapper.SysUserMapper;
import com.yingxue.lesson.req.RegisterReqVO;
import com.yingxue.lesson.req.UpdateUserReqVO;
import com.yingxue.lesson.service.UserService;
import com.yingxue.lesson.utils.PasswordUtils;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.Date;
```

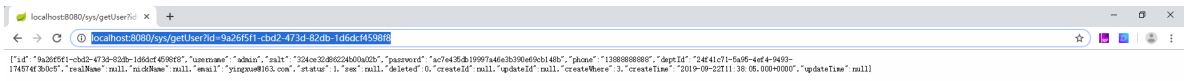
```
import java.util.UUID;

/**
 * @ClassName: UserServiceImpl
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private SysUserMapper sysUserMapper;
    @Override
    public SysUser getUserInfo(String id) {
        System.out.println("=====">+id);
        return sysUserMapper.selectByPrimaryKey(id);
    }

}
```

1.3.4 新增接口

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

import com.yingxue.lesson.entity.SysUser;
import com.yingxue.lesson.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName: UserCotroller
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/sys")
public class UserCotroller {
    @Autowired
    private UserService userService;
    @GetMapping("/getUser")
    public SysUser getUserInfo(@RequestParam String id){
        return userService.getUserInfo(id);
    }
}
```

1.3.4 启动项目

浏览器输入 http://localhost:8080/sys/getUser?id=9a26f5f1-cbd2-473d-82db-1d6dcf4598f8



测试成功

2.Spring Boot 集成 Druid 监控数据源

2.1 Druid 介绍

Druid 是阿里巴开源平台上的一个项目，整个项目由数据库连接池、插件框架和 SQL 解析器组成，该项目主要是为了扩展 JDBC 的一些限制，可以让程序员实现一些特殊的需求，比如统计SQL 信息、SQL 性能收集、SQL 注入检查、SQL 翻译等，程序员可以通过定制来实现自己需要的功能。Druid 首先是一个数据库连接池，但它不仅是一个数据库连接池，还包含了了一个 ProxyDriver，一系列列内置的 JDBC 组件库。在 Java 的世界中 Druid 是监控做的最好的数据库连接池，在功能、性能、扩展性方面，也有不错的表现。

2.2 Druid 可以做什么

- 替换其他 Java 连接池，Druid 提供了了一个高效、功能强大、可扩展性好的数据库连接池。
- 可以监控数据库访问性能，Druid 内置提供了一个功能强大的 StatFilter 插件，能够详细统计 SQL 的执 行性能，这对于线上分析数据库访问性能有很大帮助。
- SQL 执行日志，Druid 提供了不同的 LogFilter，能够支支持 Common-Logging、Log4j 和 JdkLog，可以 按需要选择相应的 LogFilter，监控应用的数据库访问情况。
- 扩展 JDBC，如果你要对 JDBC 层有编程的需求，可以通过 Druid 提供的 Filter 机制，很方便编写 JDBC 层的扩展插件。

2.3 Spring Boot 集成 Druid

非常令人高兴的是，阿里为 Druid 也提供了 Spring Boot Starter 的支持。官网这样解释：Druid Spring Boot Starter 用于帮助你在 Spring Boot 项目中轻松集成 Druid 数据库连接池和监控。Druid Spring Boot Starter 主要做了哪些事情呢？其实这个组件包很简单，主要提供了很多自动化的配置，按照 Spring Boot 的理念对很多内容进行了预配置，让我们在使用的时候更加的简单和方便。

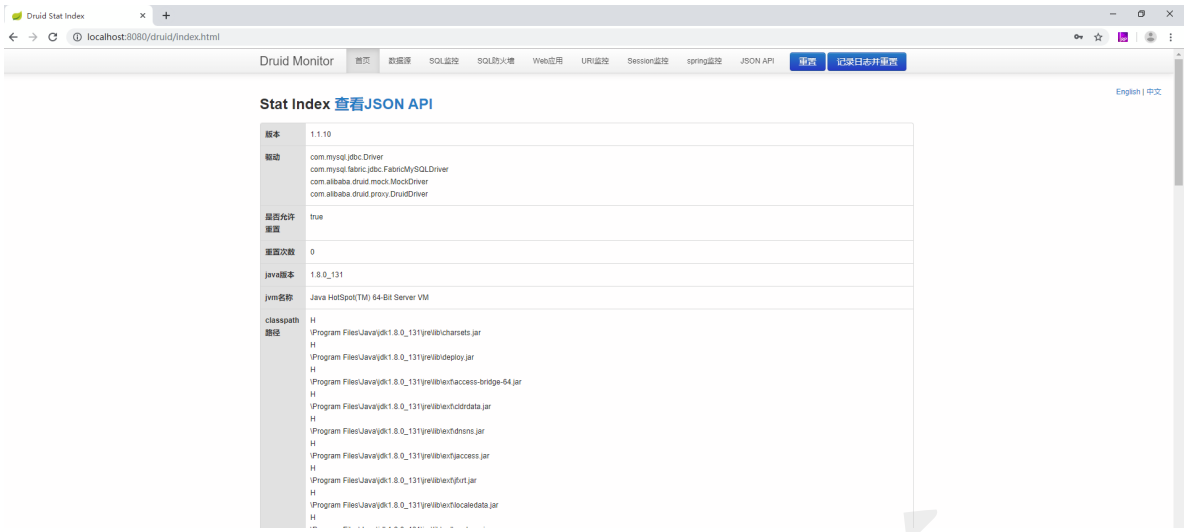
2.4 mybatis 使用连接池

yingxue-web application.properties 加入链接池配置

Druid Spring Boot Starter 配置属性的名称完全遵照 Druid，可以通过 Spring Boot 配置文件来配置 Druid 数据库连接池和监控，如果没有配置则使用默认值。

```
#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/test_mybatis?
useUnicode=true&characterEncoding=utf-8&useSSL=false
spring.datasource.druid.username=root
spring.datasource.druid.password=root
##################    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
# 监控后台账号和密码
spring.datasource.druid.stat-view-servlet.login-username=admin
spring.datasource.druid.stat-view-servlet.login-password=66666

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

配置完成后，直接启动项目访问地址：http://localhost:8080/druid，就会出现 Druid 监控后台的登录页面，输入账户和密码后，就会进入首页。



3. RESTful API 详解

31 什么是Restful API

RESTful API 从字面就可以知道，他是 rest 式的接口，所以就要先了解什么是 rest？

- rest 不是一个技术，也不是一个协议
- rest 指的是一组架构约束条件和原则，提供了一个新的架构设计思路，满足这些约束条件和原则的应用程序或设计就是 RESTful

在REST规则中，有两个基础概念：对象、行为

- 对象就是我们要操作的对象，例如添加用户的操作，那么对象就是user
- 行为有4种常用的：查看、新增、编辑、删除

rest 的提出者很巧妙的利用 http 现有方法来对应这4种行为：

- GET - 查看
- POST - 新增
- PUT - 编辑
- DELETE - 删除

3.2 为什么用RESTful API

在以前移动端没有盛行的时候，那时候前后端的融合还是比较紧密的没有前后端分离，后端处理业务逻辑，然后使用页面模板来输出显示给用户，以前用户主要就是用浏览器

访问，这样的结构没什么问题，但现在移动客户端越来越重要，显然不能使用这个结构，需要从新为移动客户端开发接口， 而 RESTful API 可以通过一套统一的接口为所有客户端提供 web 服务，实现前后端分离。

3.3 RESTful API是怎么定义的？

下面通过几个示例了解下Restful API的定义方式：

- 查看所有用户

  GET http://yingxue.com/users

- 新增一个用户

  POST http://yingxue.com/user

- 根据ID查看一个用户

  GET http:///yingxue.com/user/123456

- 更新用户

  PUT http:///yingxue.com/user/123456

- 删除用户

  DELETE http:///yingxue.com/user/123456

可以看到Restful API的风格非常简洁、统一、明确

例如查看操作，用普通方式的话，定义方式是任意的，如：

http:///yingxue.com/getUser

'getUser' 就是随意定义的，通过这个名字才可以看出是查看用户信息的意思，如果开发人员用了一个没有明确意义的名字，那就需要看文档或者代码才能知道含义了

而RESTful API 通过 GET 方法就知道是查看操作，通过 user 就知道查看的对象是什么

3.4. 实战 RESTful 之 GET 获取用户信息

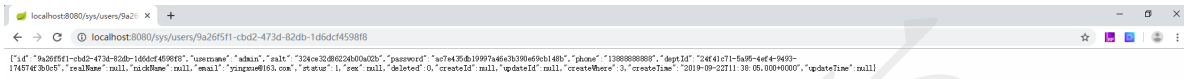
3.4.1 业务分析

一般获取信息接口都是用 id 来查询的，我们这里呢也不例外，也是用用户id来查询用户信息。

3.4.2 新增RESTful 风格获取用户信息接口

```
    @GetMapping("/users/{id}")
    public SysUser getUserDetail(@PathVariable("id") String id){
        return userService.getUserInfo(id);
    }
```

3.5 测试



4. 实战 RESTful 之 POST 用户注册接口

4.1 业务分析

在用户注册功能的时候，我们后端首先要思考，我们到底要收集用户什么信息，这个要结合现有产品设计的文档来定。因为用户注册就相当于起点了所以我们要根据现有业务来决定要收集用户什么信息，根据我们上一章节的用户表结构来看用户注册需要用户名、密码、手机号、邮箱、性别、昵称、来源。

4.2 实战编码

4.2.1 创建一个 RegisterReqVO

yingxue-vo 创建

用于接收前端表单数据

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

/**
 * @ClassName: RegisterReqVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
public class RegisterReqVO {
    private String username;
    private String password;
    private String phone;
    private String email;
    private Integer sex;
    private Integer createWhere;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
```

```
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Integer getSex() {
        return sex;
    }

    public void setSex(Integer sex) {
        this.sex = sex;
    }

    public Integer getCreateWhere() {
        return createWhere;
    }

    public void setCreateWhere(Integer createWhere) {
        this.createWhere = createWhere;
    }

    @Override
    public String toString() {
        return "RegisterReqVO{" +
                "username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", phone='" + phone + '\'' +
                ", email='" + email + '\'' +
                ", sex=" + sex +
                ", createWhere=" + createWhere +
                '}';
    }
}
```

4.2.2 创建注册接口方法

修改 UserService 加入如下方法

```
String register(RegisterReqVO vo);
```

4.2.3 实现注册接口业务

首先呢我们注册的时候要用户输入用户名密码，而密码呢我们不能明文的存储到 DB 所以我们需要一些加密/绝密工具。

yingxue-service 创建

1. PasswordUtils

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

import java.util.UUID;

/**
* @ClassName:       PasswordUtils
```

```
*                   密码工具类
* @Author:          小霍
* @UpdateUser:      小霍
* @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);
    }
}
```

2. PasswordEncoder

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

import java.security.MessageDigest;

/**
* @ClassName:       PasswordEncoder
*                   密码加密
* @Author:          小霍
* @UpdateUser:      小霍
* @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) {

    }
```

```
}
```

3. 注册接口业务实现

   修改 UserServiceImpl 加入如下方法

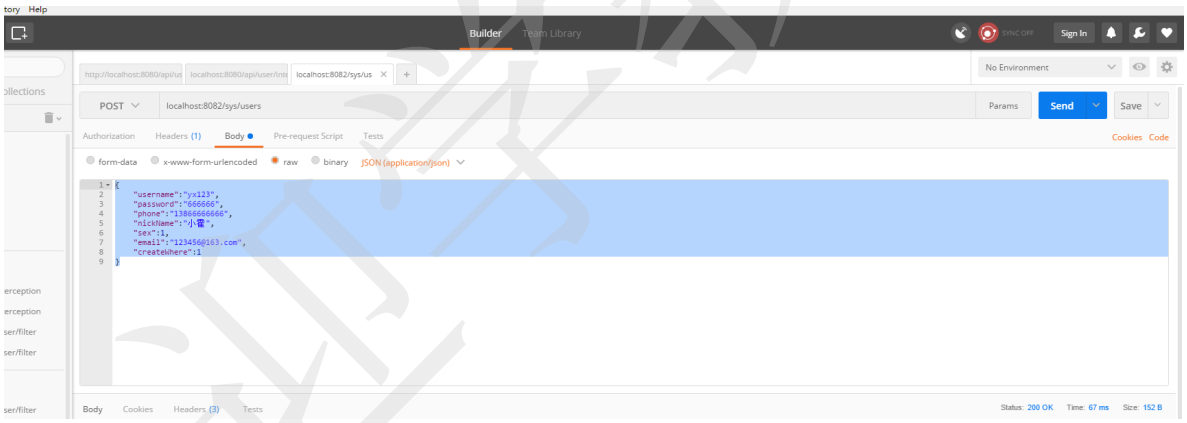
```
    @Override
    public String register(RegisterReqVO vo) {
        SysUser sysUser=new SysUser();
        BeanUtils.copyProperties(vo,sysUser);
        sysUser.setSalt(PasswordUtils.getSalt());
        String encode = PasswordUtils.encode(vo.getPassword(), sysUser.getSalt());
        sysUser.setPassword(encode);
        sysUser.setId(UUID.randomUUID().toString());
        sysUser.setCreateTime(new Date());
        int i = sysUserMapper.insertSelective(sysUser);
        if (i!=1){
            return "注册失败";
        }
        return "注册成功";
    }
```

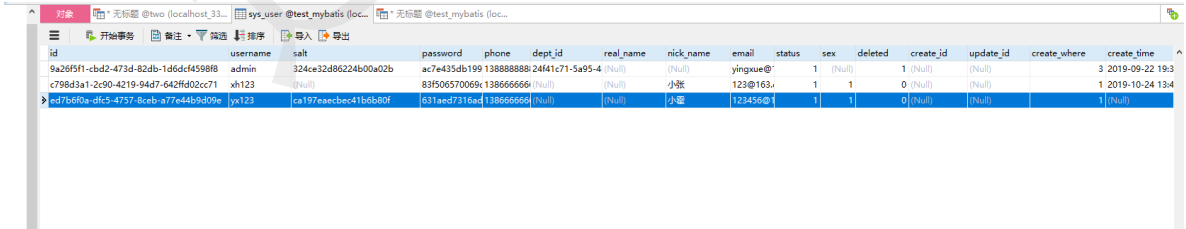
4.3 新增RESTful 风格注册接口

修改 UserController 加入如下方法

```
    @PostMapping("/users")
    public String register(@RequestBody RegisterReqVO vo){
        return userService.register(vo);
    }
```

4.4 测试





5. 实战 RESTful 之 PUT 修改用户信息接口

5.1 业务分析

更新用户信息，我们要思考让用户自己能改到什么信息，这里暂时不做限制所以用户所有信息都可以修改。

5.2 实战编码

5.2.1 创建一个 updateUserReqVO

用于接受前端表单数据

yingxue-vo 创建

```
public class UpdateUserReqVO {
    private String id;

    private String username;

    private String password;

    private String phone;

    private String deptId;

    private String realName;

    private String nickName;

    private String email;

    private Integer status;

    private Integer sex;

    private Integer createWhere;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getDeptId() {
        return deptId;
    }

    public void setDeptId(String deptId) {
        this.deptId = deptId;
    }

    public String getRealName() {
        return realName;
    }

    public void setRealName(String realName) {
        this.realName = realName;
    }

    public String getNickName() {
        return nickName;
    }
```

```
    public void setNickName(String nickName) {
        this.nickName = nickName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Integer getStatus() {
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }

    public Integer getSex() {
        return sex;
    }

    public void setSex(Integer sex) {
        this.sex = sex;
    }

    public Integer getCreateWhere() {
        return createWhere;
    }

    public void setCreateWhere(Integer createWhere) {
        this.createWhere = createWhere;
    }
}
```

5.2.2 创建更新用户接口方法

```
String updateUserInfo(UpdateUserReqVO vo);
```

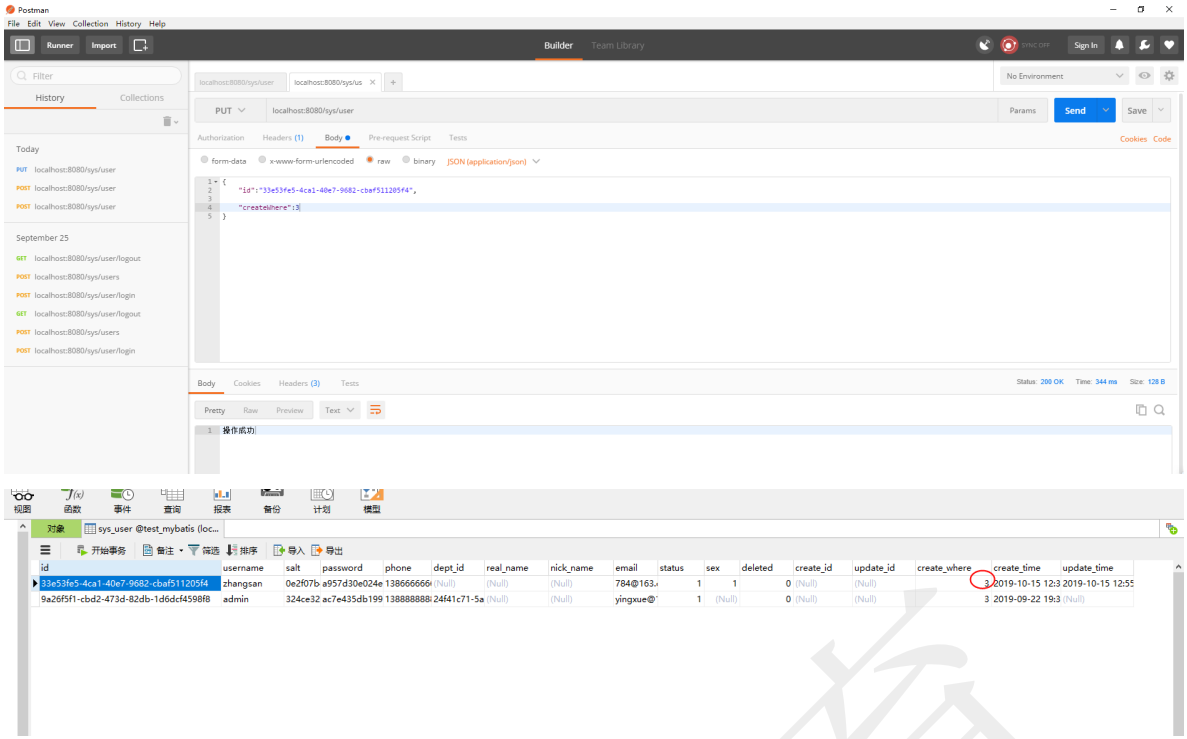
5.2.3 实现更新用户信息接口业务

```
 @Override
    public String updateUserInfo(UpdateUserReqVO vo) {
        SysUser sysUser=sysUserMapper.selectByPrimaryKey(vo.getId());
        if (null==sysUser){
            return "操作失败";
        }
        SysUser update=new SysUser();
        BeanUtils.copyProperties(vo,update);
        update.setUpdateTime(new Date());
        int count= sysUserMapper.updateByPrimaryKeySelective(update);
        if (count!=1){
            return "操作失败";
        }
        return "操作成功";
    }
```

5.3 新增 RESTful 风格更新用户信息接口

```
    @PutMapping("/users")
    public String updateUserInfo(@RequestBody UpdateUserReqVO vo){
        return userService.updateUserInfo(vo);
    }
```

5.4 测试



6. 实战 RESTful 之 DELETED 删除用户接口

6.1 业务分析

删除我们在企业里一般敏感数据都是做逻辑删除，就是用一个 deleted 字段来表明数据状态；而不是直接物理删除

6.2 实战编码

6.2.1 创建删除用户接口方法

```
String deleted(String userId);
```

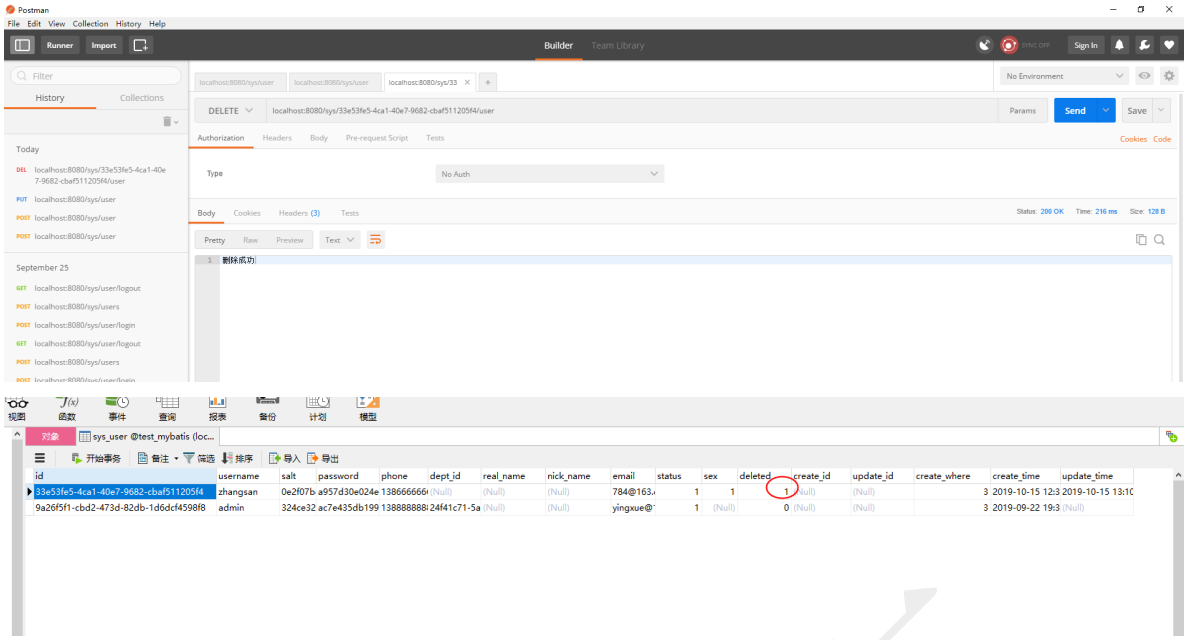
6.2.2 实现删除用户接口

```
    @Override
    public String deleted(String userId) {
        SysUser sysUser=new SysUser();
        sysUser.setId(userId);
        sysUser.setUpdateTime(new Date());
        sysUser.setDeleted(1);
        int count=sysUserMapper.updateByPrimaryKeySelective(sysUser);
        if (count==0){
            return "删除失败";
        }
        return "删除成功";
    }
```

6.2.3 新增RESTful 风格 删除接口

```
    @DeleteMapping("/users/{id}")
     public String deleted(@PathVariable("id") String id){
         return userService.deleted(id);
     }
```

6.3 测试



7. Spring boot 事务整合

spring boot 事务是怎么控制的呢？就是我们熟知的 @Transactional 注解了

7.1 业务分析

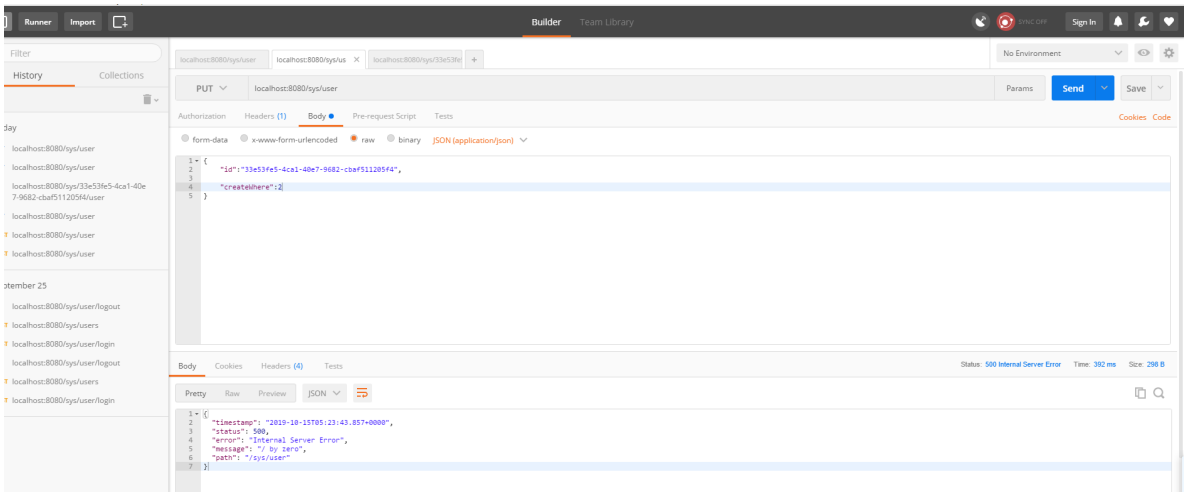
让代码更新完用户信息后抛出异常，看看数据会不会回滚

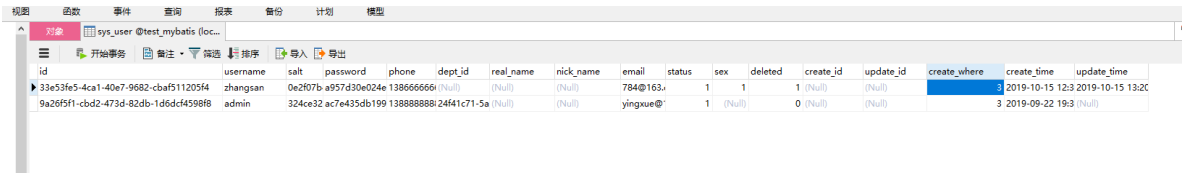
7.2 实战编码

修改更新用户信息接口让抛出一个异常

```
    @Override
    @Transactional(rollbackFor = Exception.class)
    public String updateUserInfo(UpdateUserReqVO vo) {
        SysUser sysUser=sysUserMapper.selectByPrimaryKey(vo.getId());
        if (null==sysUser){
            return "操作失败";
        }
        SysUser update=new SysUser();
        BeanUtils.copyProperties(vo,update);
        update.setUpdateTime(new Date());
        int count= sysUserMapper.updateByPrimaryKeySelective(update);
        if (count!=1){
            return "操作失败";
        }
        int i=1/0;
        return "操作成功";
    }
```

7.3 测试



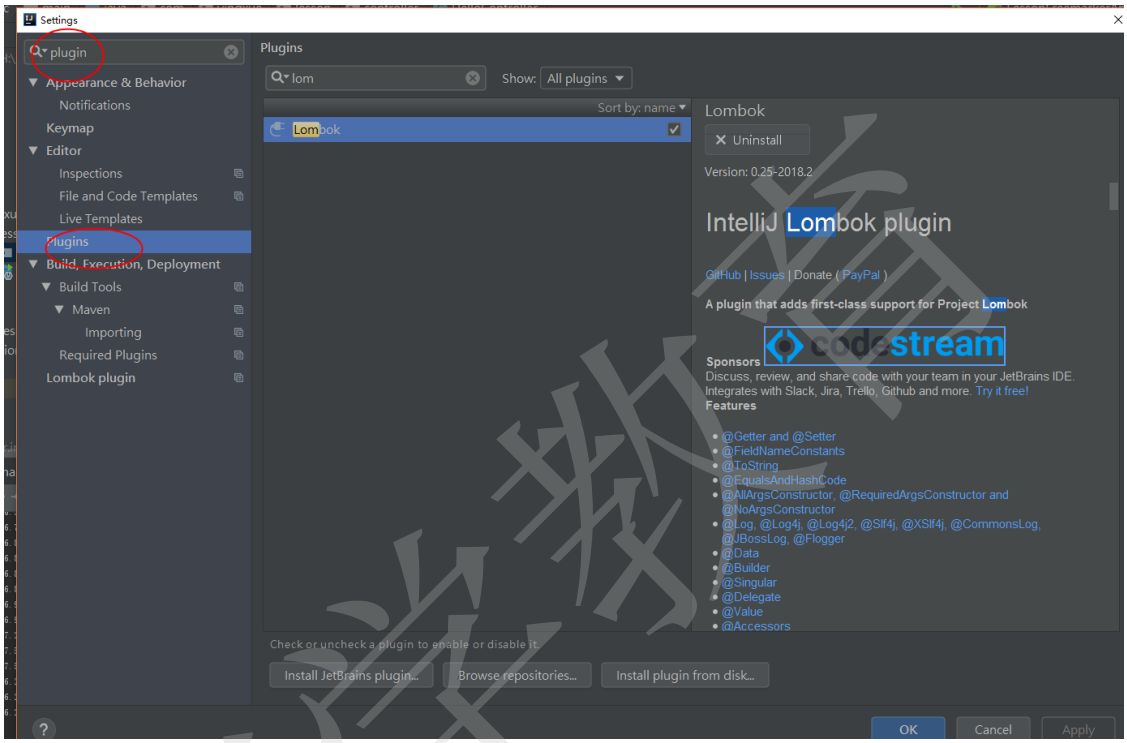


8. Spring Boot 优雅的集成 Lombok

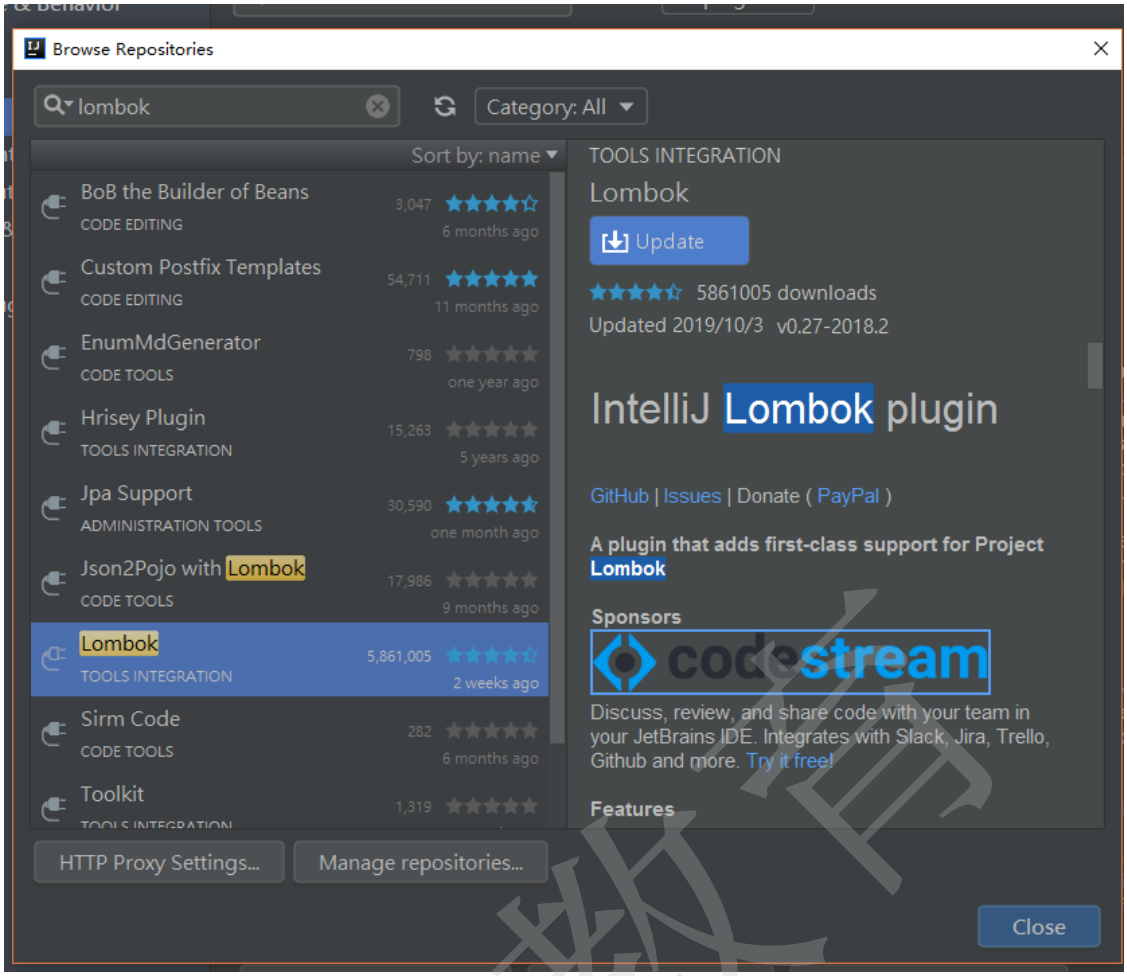
集成 lombok 要注意一下如果你所在的公司都没有人用过 lombok 插件的时候，假如你自己想使用的话一定要先和组员报备一声免得别的同事没有装插件照成编译错误，多多避免一些负面情绪

8.1 安装插件

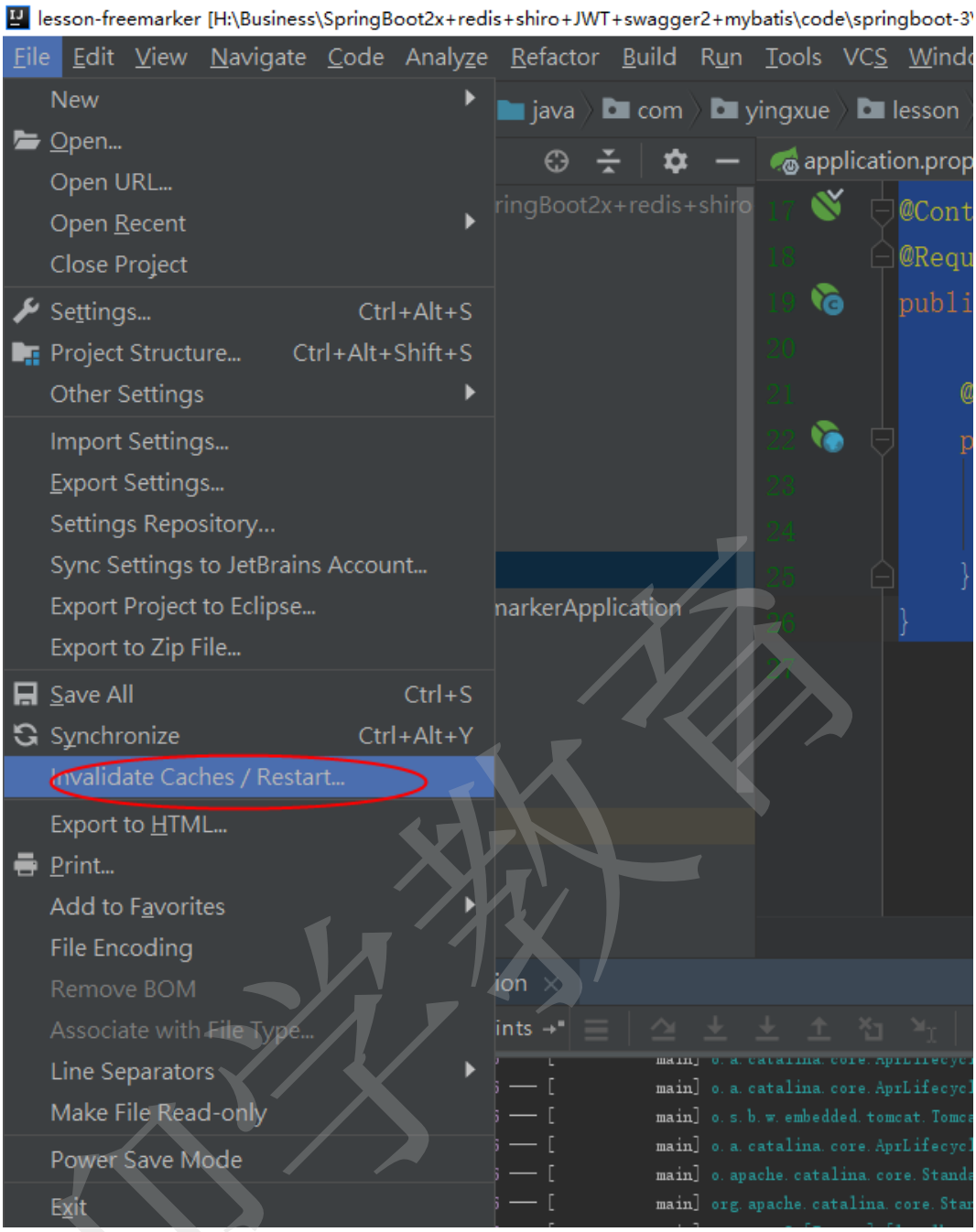
1. 打开 settings 搜索 plugin



2. 点击 Browse repositories 搜索 lombok install



3. 安装好后 重启 idea



8.2 实战操作

8.2.1 加入依赖

yingxue-vo pom 加入如下代码

```
        <!--lombok-->
         <dependency>
             <groupId>org.projectlombok</groupId>
             <artifactId>lombok</artifactId>
         </dependency>
```

8.3 测试

8.3.1 创建ResponseVO

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

import lombok.Data;

/**
 * @ClassName: ResponseVO
 * TODO:类文件简单描述
 * @Author: 小霍
```

```
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
public class ResponseVO {
    private String name;
    private String age;
    private String phone;
}
```

8.3.2 创建测试接口

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

import com.yingxue.lesson.vo.resp.ResponseVO;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName: LombokController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/test")
public class LombokController {

    @GetMapping("/lombok")
    public ResponseVO testLombok(){
        ResponseVO vo=new ResponseVO();
        vo.setPhone("13866666666");
        vo.setAge(30);
        vo.setName("zhangsan");
        return vo;
    }
}
```

9. Spring Boot 优雅的集成Swagger2

9.1 实战swagger2

9.1.1 引入swagger 依赖

yingxue-vo 加入如下代码

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

9.1.2 创建 swagger 配置类

yingxue-web 创建 SwaggerConfig

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

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;


/**
 * @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 {
    @Bean
    public Docket createRestApi() {

        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())//创建该Api的基本信息（这些基本信息会展现在文档页面中）
                .select()//函数返回一个ApiSelectorBuilder实例用来控制哪些接口暴露给Swagger ui来展现
                .apis(RequestHandlerSelectors.basePackage("com.yingxue.lesson.web.controller"))//指定需要
扫描的包路路径
                .paths(PathSelectors.any())
                .build()

                ;
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("spring boot2.x实战")
                .description("spring boot2.x 零基础到高级实战系列")
                .termsOfServiceUrl("")
                .version("1.0")
                .build();
    }

}
```

9.1.3 接收表单VO

yingxue-vo 创建

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

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

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

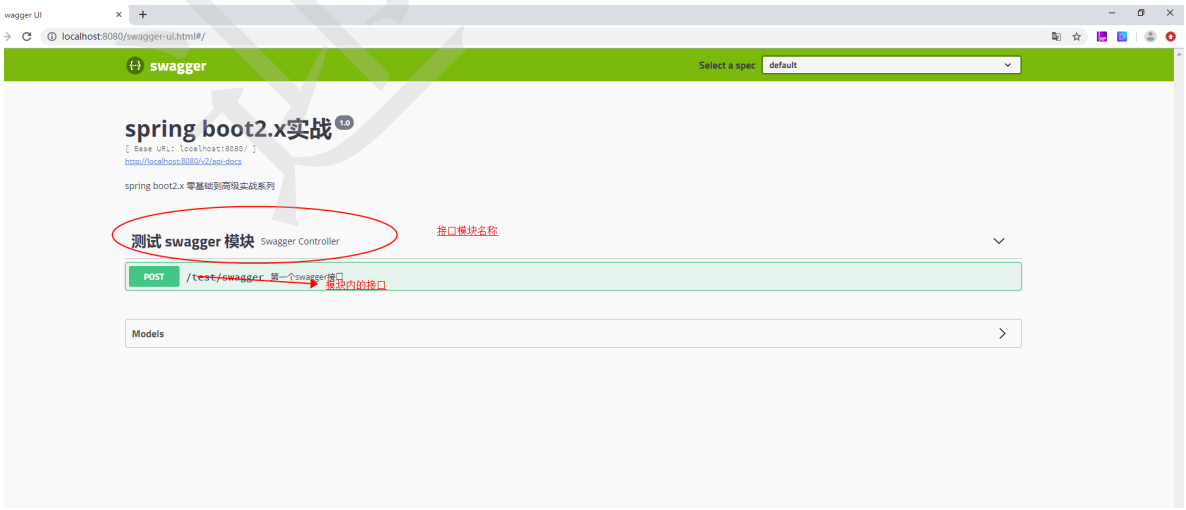
```
public class SwaggerReqVO {
    @ApiModelProperty(value = "名字")
    private String name;
    @ApiModelProperty(value = "年龄")
    private Integer age;
    @ApiModelProperty(value = "电话")
    private String phone;
}
```

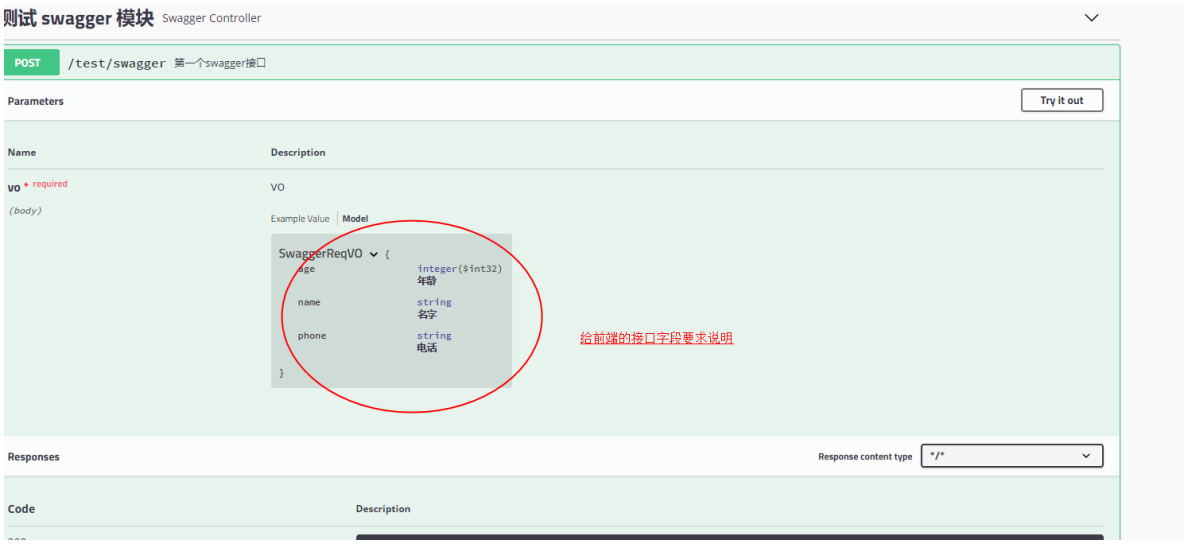
9.2.5 创建测试接口

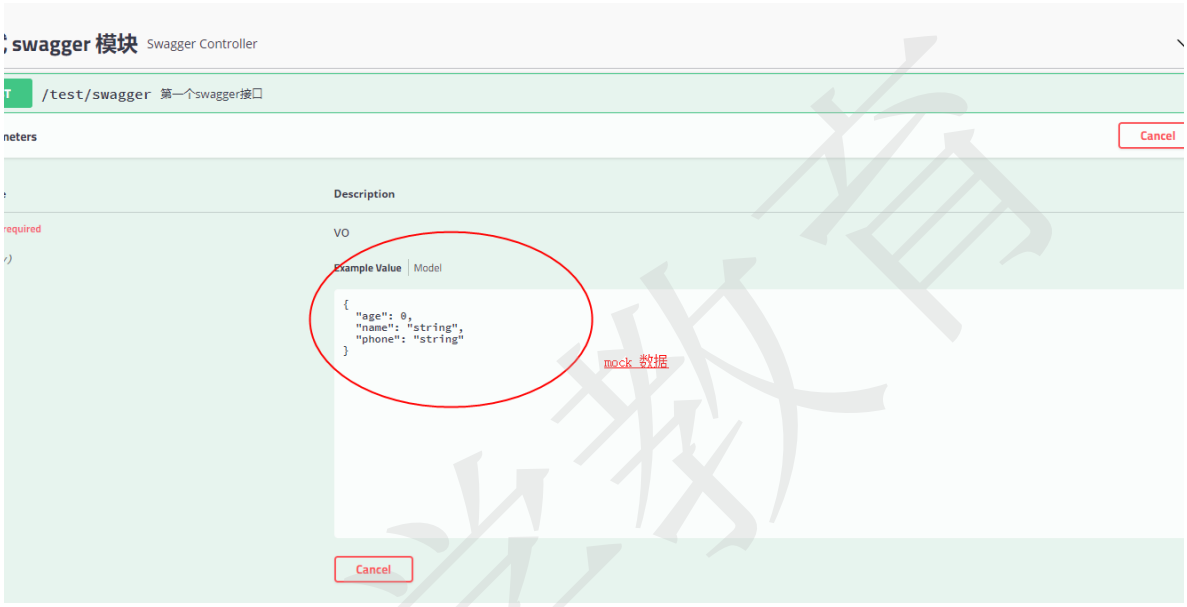
```
package com.yingxue.lesson.controller;

import com.yingxue.lesson.vo.req.SwaggerReqVO;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName: SwaggerController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/test")
@Api(tags = "测试 swagger 模块")
public class SwaggerController {

    @ApiOperation(value = "第一个swagger接口")
    @PostMapping("/swagger")
    public SwaggerReqVO testSwagger(@RequestBody SwaggerReqVO vo){
        return vo;
    }
}
```

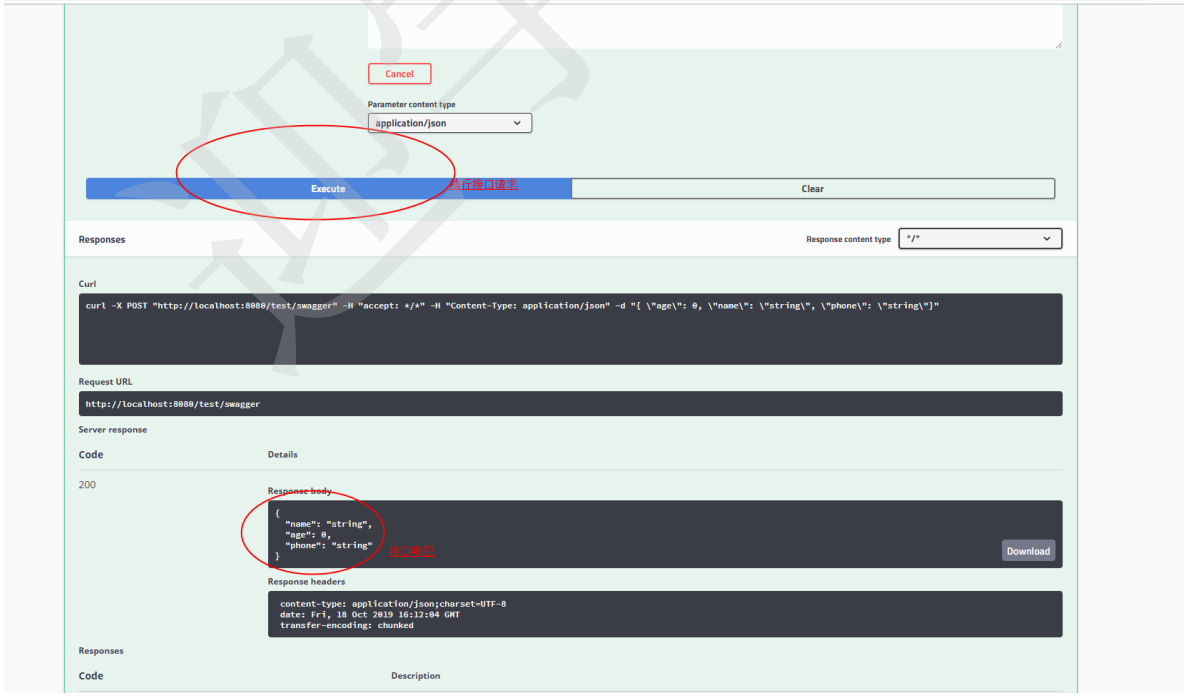
9.2.6 测试

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









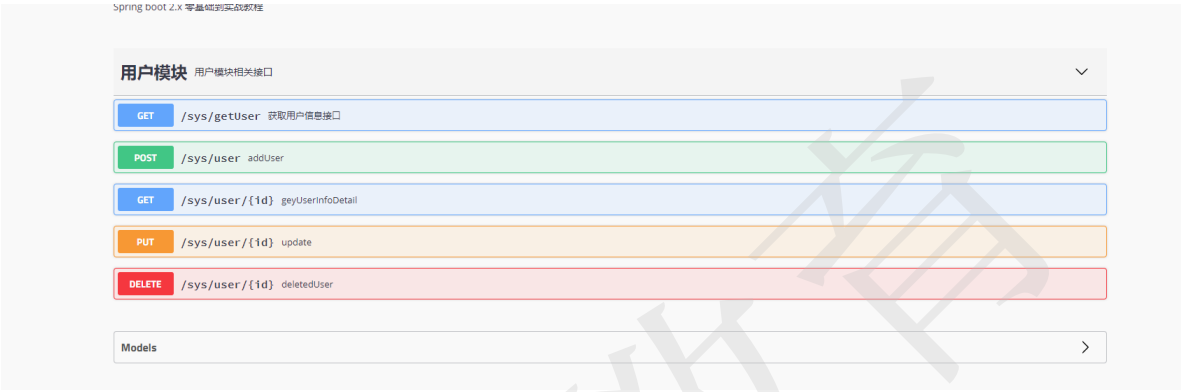
10. swagger2 常用注解使用及其说明

10.1 Api

@Api 用在类上，说明该类的作用。可以标记一个 Controller 类作为 Swagger 文档资源，使用方式代码如下所示。

```
/**
 * @ClassName: UserController
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@RestController
@RequestMapping("/sys")
@Api(tags = "用户模块",description = "用户模块相关接口")
public class UserController {

}
```

效果图如图 1 所示。



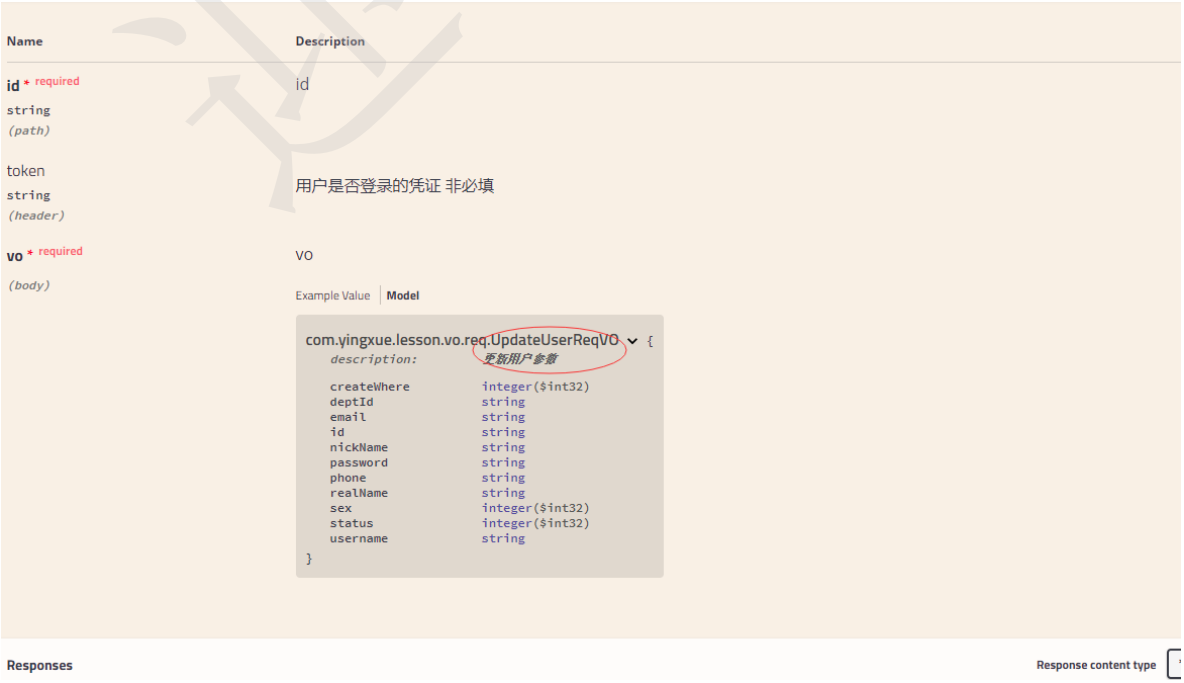
tags：接口说明，可以在页面中显示。可以配置多个，当配置多个的时候，在页面中会显示多个接口的信息。description:对tags 进行描述说明

10.2 ApiModel

@ApiModel 用在类上，表示对类进行说明，用于实体类中的参数接收说明。使用方式代码如下所示。

```
@Data
@ApiModel(value = "com.yingxue.lesson.vo.req.UpdateUserReqVO",description = "更新用户")
public class UpdateUserReqVO {
}
```

效果图如图 2 所示。



10.3 ApiModelProperty

@ApiModelProperty() 用于字段，表示对 model 属性的说明。使用方式代码如下所示。

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

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

/**
 * @ClassName: UpdateUserReqVO
 * TODO:类文件简单描述
 * @Author: 小霍
 * @UpdateUser: 小霍
 * @Version: 0.0.1
 */
@Data
@ApiModel(value = "com.yingxue.lesson.vo.req.UpdateUserReqVO",description = "更新用户参数")
public class UpdateUserReqVO {
    @ApiModelProperty(value = "用户id")
    private String id;
    @ApiModelProperty(value = "账号")
    private String username;
    @ApiModelProperty(value = "密码")
    private String password;
    @ApiModelProperty(value = "手机号")
    private String phone;
    @ApiModelProperty(value = "机构id")
    private String deptId;
    @ApiModelProperty(value = "真实名称")
    private String realName;
    @ApiModelProperty(value = "昵称")
    private String nickName;
    @ApiModelProperty(value = "邮箱")
    private String email;
    @ApiModelProperty(value = "状态")
    private Integer status;
    @ApiModelProperty(value = "性别")
    private Integer sex;
    @ApiModelProperty(value = "涞源")
    private Integer createWhere;


}
```

效果图如图 3 所示。

10.4 ApiParam

@ApiParam 用于 Controller 中方法的参数说明。使用方式代码如下所示。

```
@GetMapping("/getUser")
public SysUser getUserInfo(@ApiParam(value = "用户id",required = true) @RequestParam String id){
    return userService.getUserInfo(id);
}
```

- value：参数说明
- required：是否必填

效果图如图 4所示。

10.5 ApiOperation

@ApiOperation 用在 Controller 里的方法上，说明方法的作用，每一个接口的定义。使用方式代码如下所示。

- value：接口名称

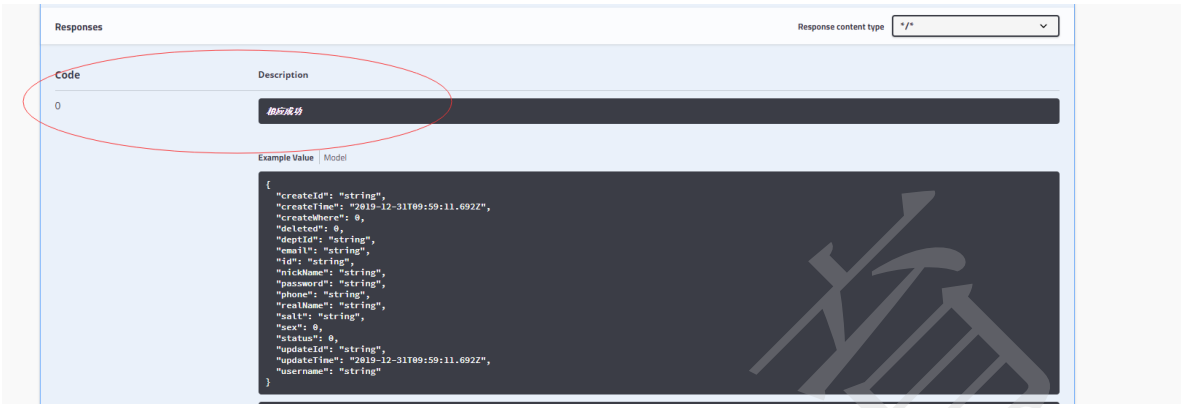
```
 @GetMapping("/getUser")
@ApiOperation(value = "获取用户信息接口")
public SysUser getUserInfo(@ApiParam(value = "用户id",required = true) @RequestParam String id){
    return userService.getUserInfo(id);
}
```

效果图如图 4所示。

10.6 ApiResponse 和 ApiResponses

@ApiResponse 用于方法上，说明接口响应的一些信息；@ApiResponses 组装了多个 @ApiResponse。使用方式代码如下所示。

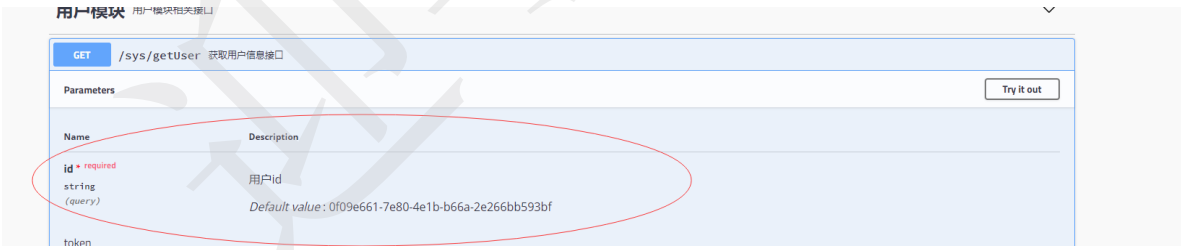
```
    @GetMapping("/getUser")
    @ApiOperation(value = "获取用户信息接口")
    @ApiResponses({ @ApiResponse(code = 0, message = "相应成功", response = SysUser.class) })
    public SysUser getUserInfo(@ApiParam(value = "用户id",required = true) @RequestParam String id){
        return userService.getUserInfo(id);
    }
```



10.7 ApiImplicitParam 和 ApiImplicitParams

用于方法上，为单独的请求参数进行说明。使用方式代码如下所示。

```
@GetMapping("/getUser")
    @ApiOperation(value = "获取用户信息接口")
    @ApiResponses({ @ApiResponse(code = 0, message = "相应成功", response = SysUser.class) })
    @ApiImplicitParams({
            @ApiImplicitParam(name = "id", value = "用户id", dataType = "string", paramType = "query",
required = true, defaultValue = "0f09e661-7e80-4e1b-b66a-2e266bb593bf")
    })
    public SysUser getUserInfo(@ApiParam(value = "用户id",required = true) @RequestParam String id){
        return userService.getUserInfo(id);
    }
```



11.swagger2 多环境安全配置

1. 领导要求生产环境关闭 swagger我们该怎么做？
    1. 在 配置文件新增开关

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

    2. 修改 SwaggerConfig 动态设置开关

    ```
    @Configuration
    @EnableSwagger2
    public class SwaggerConfig {
        @Value("${swagger2.enable}")
        private boolean enable;
        @Bean
        public Docket createRestApi() {
    ```

```
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())//创建该Api的基本信息（这些基本信息会展现在文档页面中）
                .select()//函数返回一个ApiSelectorBuilder实例用来控制哪些接口暴露给Swagger ui来展现
                .apis(RequestHandlerSelectors.basePackage("com.yingxue.lesson.controller"))//
指定需要扫描的包路路径
                .paths(PathSelectors.any())
                .build()
                .enable(enable)//设置开关
                ;
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("spring boot2.x实战")
                .description("spring boot2.x 零基础到高级实战系列")
                .termsOfServiceUrl("")
                .version("1.0")
                .build();
    }

}
```

12. swagger2 全局配置头部信息

1. 假如入某个接口需从header拿一些数据检验怎么办？

   swagger 可全局设置 header 入口

   修改 SwaggerConfig 配置类

```
@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();
        tokenPar.name("token").description("swagger测试用(模拟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());
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo())//创建该Api的基本信息（这些基本信息会展现在文档页面中）
                .select()//函数返回一个ApiSelectorBuilder实例用来控制哪些接口暴露给Swagger ui来展现
                .apis(RequestHandlerSelectors.basePackage("com.yingxue.lesson.controller"))//指定需
要扫描的包路路径
                .paths(PathSelectors.any())
                .build()
                .globalOperationParameters(pars)
                .enable(enable)

                ;
    }

    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("spring boot2.x实战")
                .description("spring boot2.x 零基础到高级实战系列")
                .termsOfServiceUrl("")
                .version("1.0")
                .build();
    }

}
```

测试

修改 testSwagger方法

```
    @ApiOperation(value = "第一个swagger接口")
    @PostMapping("/swagger")
    public SwaggerReqVO testSwagger(@RequestBody SwaggerReqVO vo, HttpServletRequest request){
        System.out.println(request.getHeader("token"));
        return vo;
    }
```

重启项目