# *doubleball项目流程*

一、新建一个Maven工程，设置groupId为com.wjx ，artifactId为doubleball 。

二、设置pom.xml

|  |
| --- |
| ？？？？？？ |

三、在主资源目录resources下添加资源配置文件application.properties，内容如下

|  |
| --- |
| **spring.datasource.url**= **jdbc:mysql://127.0.0.1:3306/doubleball?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&autoReconnect=true****spring.datasource.username**= **root spring.datasource.password**= **123456 spring.datasource.driver-class-name**= **com.mysql.jdbc.Driver**  **server.port**: **8080** |

四、添加启动类

在项目目录中新建一个启动类，在里面写main方法(App类与Controller包同级别)

APP.java

|  |
| --- |
| **package** com.doubleball;  **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.SpringBootApplication; @SpringBootApplication **public class** APP {   **public static void** main(String[] args) {  SpringApplication.*run*(APP.**class**,args);  } } |

五、添加全局异常捕获

在项目下面新建一个包com.doubleball.exception，新建一个异常拦截类GlobalExceptionHandler进行拦截Controller抛出的异常。

先建一个结果类：

|  |
| --- |
| **package** com.doubleball.exception;  **import** com.alibaba.fastjson.JSON; **import** java.io.Serializable; **import** java.util.HashMap; **import** java.util.Map; **public class** JsonResult **implements** Serializable {  *// Serializable接口是启用其序列化功能的接口。实现java.io.Serializable 接口的类是可序列化的。没有实现此接口的类将不能使它们的任意状态被序列化或逆序列化。* **private int code**; *//返回码 非0即失败* **private** String **message**; *//消息提示* **private** Map<String, Object> **data**; *//返回的数据* **public int** getCode() {  **return code**;  }   **public void** setCode(**int** code) {  **this**.**code** = code;  }   **public** String getMessage() {  **return message**;  }   **public void** setMessage(String message) {  **this**.**message** = message;  }   **public** Map<String, Object> getData() {  **return data**;  }   **public void** setData(Map<String, Object> data) {  **this**.**data** = data;  }   **public** JsonResult(){};   **public** JsonResult(**int** code, String message, Map<String, Object> data) {  **this**.**code** = code;  **this**.**message** = message;  **this**.**data** = data;  }   **public static** String success(Map<String, Object> data) {  **return** JSON.*toJSONString*(**new** JsonResult(0, **"解析成功"**, data));  }   **public static** String success() {   **return** *success*(**new** HashMap<String, Object>(0));  }   **public static** String failed(**int** code, String msg) {  **return** JSON.*toJSONString*(**new** JsonResult(code, msg, **new** HashMap<String, Object>(0)));  }   **public static** String failed(String msg) {   **return** *failed*(-1, msg);  }   **public static** String failed() {   **return** *failed*(**"解析失败"**);  } } |

再建一个异常汇总的类：

|  |
| --- |
| **package** com.doubleball.exception;  **import** org.slf4j.Logger; **import** org.slf4j.LoggerFactory; **import** org.springframework.beans.ConversionNotSupportedException; **import** org.springframework.beans.TypeMismatchException; **import** org.springframework.http.converter.HttpMessageNotReadableException; **import** org.springframework.http.converter.HttpMessageNotWritableException; **import** org.springframework.web.HttpMediaTypeNotAcceptableException; **import** org.springframework.web.HttpRequestMethodNotSupportedException; **import** org.springframework.web.bind.MissingServletRequestParameterException; **import** org.springframework.web.bind.annotation.ControllerAdvice; **import** org.springframework.web.bind.annotation.ExceptionHandler; **import** org.springframework.web.bind.annotation.ResponseBody; **import** java.io.IOException; @ControllerAdvice *// 异常集中处理，更好的使业务逻辑与异常处理剥离开* @ResponseBody *// 不加上这个，不起效果* **public class** GlobalExceptionAop {  **private static final** String ***logExceptionFormat*** = **"Capture Exception By GlobalExceptionAop: Code: %s Detail: %s"**;  **private static** Logger *log* = LoggerFactory.*getLogger*(GlobalExceptionAop.**class**);  *// @ExceptionHandler(RuntimeException.class) // public String runtimeException(){ // return "出现异常了，这里在捕获全局异常，相当于手写AOP捕获异常。"; // }  //运行时异常* @ExceptionHandler(RuntimeException.**class**) *//统一处理某一类异常，从而能够减少代码重复率和复杂度* **public** String runtimeExceptionHandler(RuntimeException ex) {  **return** resultFormat(1, ex);  }  *//空指针异常* @ExceptionHandler(NullPointerException.**class**)  **public** String nullPointerExceptionHandler(NullPointerException ex) {  **return** resultFormat(2, ex);  }  *//类型转换异常* @ExceptionHandler(ClassCastException.**class**)  **public** String classCastExceptionHandler(ClassCastException ex) {  **return** resultFormat(3, ex);  }  *//IO异常* @ExceptionHandler(IOException.**class**)  **public** String iOExceptionHandler(IOException ex) {  **return** resultFormat(4, ex);  }  *//未知方法异常* @ExceptionHandler(NoSuchMethodException.**class**)  **public** String noSuchMethodExceptionHandler(NoSuchMethodException ex) {  **return** resultFormat(5, ex);  }  *//数组越界异常* @ExceptionHandler(IndexOutOfBoundsException.**class**)  **public** String indexOutOfBoundsExceptionHandler(IndexOutOfBoundsException ex) {  **return** resultFormat(6, ex);  }  *//400错误* @ExceptionHandler({HttpMessageNotReadableException.**class**})  **public** String requestNotReadable(HttpMessageNotReadableException ex) {  System.***out***.println(**"400..requestNotReadable"**);  **return** resultFormat(7, ex);  }  *//400错误* @ExceptionHandler({TypeMismatchException.**class**})  **public** String requestTypeMismatch(TypeMismatchException ex) {  System.***out***.println(**"400..TypeMismatchException"**);  **return** resultFormat(8, ex);  }  *//400错误* @ExceptionHandler({MissingServletRequestParameterException.**class**})  **public** String requestMissingServletRequest(MissingServletRequestParameterException ex) {  System.***out***.println(**"400..MissingServletRequest"**);  **return** resultFormat(9, ex);  }  *//405错误* @ExceptionHandler({HttpRequestMethodNotSupportedException.**class**})  **public** String request405(HttpRequestMethodNotSupportedException ex) {  **return** resultFormat(10, ex);  }  *//406错误* @ExceptionHandler({HttpMediaTypeNotAcceptableException.**class**})  **public** String request406(HttpMediaTypeNotAcceptableException ex) {  System.***out***.println(**"406..."**);  **return** resultFormat(11, ex);  }  *//500错误* @ExceptionHandler({ConversionNotSupportedException.**class**, HttpMessageNotWritableException.**class**})  **public** String server500(RuntimeException ex) {  System.***out***.println(**"500..."**);  **return** resultFormat(12, ex);  }  *//栈溢出* @ExceptionHandler({StackOverflowError.**class**})  **public** String requestStackOverflow(StackOverflowError ex) {  **return** resultFormat(13, ex);  }  *//其他错误* @ExceptionHandler({Exception.**class**})  **public** String exception(Exception ex) {  **return** resultFormat(14, ex);  }   **private** <T **extends** Throwable> String resultFormat(Integer code, T ex) {  ex.printStackTrace();  *log*.error(String.*format*(***logExceptionFormat***, code, ex.getMessage()));  **return** JsonResult.*failed*(code, ex.getMessage());  } } |

六、在数据库中新建一张双色球信息记录表doubleballstatistics，结构如下：

|  |
| --- |
| CREATE TABLE `doubleballstatistics` (  `id` int(10) unsigned NOT NULL AUTO\_INCREMENT COMMENT '主键',  `issue` int(10) unsigned NOT NULL COMMENT '期限id',  `red\_one` int(5) NOT NULL COMMENT '红球1',  `red\_two` int(5) NOT NULL COMMENT '红球2',  `red\_three` int(5) NOT NULL COMMENT '红球3',  `red\_four` int(5) NOT NULL COMMENT '红球4',  `red\_five` int(5) NOT NULL COMMENT '红球5',  `red\_six` int(5) NOT NULL COMMENT '红球6',  `blue` int(5) NOT NULL COMMENT '蓝球',  `draw\_prize\_date` varchar(20) NOT NULL COMMENT '开奖日期',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=2592 DEFAULT CHARSET=utf8 ROW\_FORMAT=COMPACT COMMENT='双色球信息记录表' |



在数据库中新建一张计算双色球统计信息结果doubleballcalculate，结构如下：

|  |
| --- |
| CREATE TABLE `doubleballcalculate` (  `id` int(10) unsigned NOT NULL AUTO\_INCREMENT COMMENT '主键',  `red\_one` int(10) NOT NULL COMMENT '红球1',  `red\_two` int(10) NOT NULL COMMENT '红球2',  `red\_three` int(10) NOT NULL COMMENT '红球3',  `red\_four` int(10) NOT NULL COMMENT '红球4',  `red\_five` int(10) NOT NULL COMMENT '红球5',  `red\_six` int(10) NOT NULL COMMENT '红球6',  `red\_seven` int(10) NOT NULL COMMENT '红球7',  `red\_eight` int(10) NOT NULL COMMENT '红球8',  `red\_nine` int(10) NOT NULL COMMENT '红球9',  `red\_ten` int(10) NOT NULL COMMENT '红球10',  `red\_eleven` int(10) NOT NULL COMMENT '红球11',  `red\_twelve` int(10) NOT NULL COMMENT '红球12',  `red\_thirteen` int(10) NOT NULL COMMENT '红球13',  `red\_fourteen` int(10) NOT NULL COMMENT '红球14',  `red\_fifteen` int(10) NOT NULL COMMENT '红球15',  `red\_sixteen` int(10) NOT NULL COMMENT '红球16',  `red\_seventeen` int(10) NOT NULL COMMENT '红球17',  `red\_eighteen` int(10) NOT NULL COMMENT '红球18',  `red\_nineteen` int(10) NOT NULL COMMENT '红球19',  `red\_twenty` int(10) NOT NULL COMMENT '红球20',  `red\_twenty\_one` int(10) NOT NULL COMMENT '红球21',  `red\_twenty\_two` int(10) NOT NULL COMMENT '红球22',  `red\_twenty\_three` int(10) NOT NULL COMMENT '红球23',  `red\_twenty\_four` int(10) NOT NULL COMMENT '红球24',  `red\_twenty\_five` int(10) NOT NULL COMMENT '红球25',  `red\_twenty\_six` int(10) NOT NULL COMMENT '红球26',  `red\_twenty\_seven` int(10) NOT NULL COMMENT '红球27',  `red\_twenty\_eight` int(10) NOT NULL COMMENT '红球28',  `red\_twenty\_nine` int(10) NOT NULL COMMENT '红球29',  `red\_thirty` int(10) NOT NULL COMMENT '红球30',  `red\_thirty\_one` int(10) NOT NULL COMMENT '红球31',  `red\_thirty\_two` int(10) NOT NULL COMMENT '红球32',  `red\_thirty\_three` int(10) NOT NULL COMMENT '红球33',  `blue\_one` int(10) NOT NULL COMMENT '蓝球1',  `blue\_two` int(10) NOT NULL COMMENT '蓝球2',  `blue\_three` int(10) NOT NULL COMMENT '蓝球3',  `blue\_four` int(10) NOT NULL COMMENT '蓝球4',  `blue\_five` int(10) NOT NULL COMMENT '蓝球5',  `blue\_six` int(10) NOT NULL COMMENT '蓝球6',  `blue\_seven` int(10) NOT NULL COMMENT '蓝球7',  `blue\_eight` int(10) NOT NULL COMMENT '蓝球8',  `blue\_nine` int(10) NOT NULL COMMENT '蓝球9',  `blue\_ten` int(10) NOT NULL COMMENT '蓝球10',  `blue\_eleven` int(10) NOT NULL COMMENT '蓝球11',  `blue\_twelve` int(10) NOT NULL COMMENT '蓝球12',  `blue\_thirteen` int(10) NOT NULL COMMENT '蓝球13',  `blue\_fourteen` int(10) NOT NULL COMMENT '蓝球14',  `blue\_fifteen` int(10) NOT NULL COMMENT '蓝球15',  `blue\_sixteen` int(10) NOT NULL COMMENT '蓝球16',  `calculate\_date` datetime DEFAULT NULL COMMENT '计算时间',  PRIMARY KEY (`id`)  ) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8 ROW\_FORMAT=COMPACT COMMENT='计算双色球统计信息结果' |



七、在com.doubleball.entity包下新建一个类DoubleballStatistics，名称要和数据库中的表名对应，字母大小写不用管，但是名称要一致，否则代码操作数据库时会找不到库，代码如下：

|  |
| --- |
| **package** com.doubleball.entity;  */\*\*  \** ***@author*** *wjx  \** ***@version*** *1.0  \** ***@date*** *2020/8/13 上午9:14  \*/* **public class** DoubleballStatistics {   **private** Integer **id**;  **private** Integer **issue**;  **private** Integer **red\_one**;  **private** Integer **red\_two**;  **private** Integer **red\_three**;  **private** Integer **red\_four**;  **private** Integer **red\_five**;  **private** Integer **red\_six**;  **private** Integer **blue**;  **private** String **draw\_prize\_date**;   **public** Integer getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   **public** Integer getIssue() {  **return issue**;  }   **public void** setIssue(**int** issue) {  **this**.**issue** = issue;  }   **public** Integer getRed\_one() {  **return red\_one**;  }   **public void** setRed\_one(**int** red\_one) {  **this**.**red\_one** = red\_one;  }   **public** Integer getRed\_two() {  **return red\_two**;  }   **public void** setRed\_two(**int** red\_two) {  **this**.**red\_two** = red\_two;  }   **public** Integer getRed\_three() {  **return red\_three**;  }   **public void** setRed\_three(**int** red\_three) {  **this**.**red\_three** = red\_three;  }   **public** Integer getRed\_four() {  **return red\_four**;  }   **public void** setRed\_four(**int** red\_four) {  **this**.**red\_four** = red\_four;  }   **public** Integer getRed\_five() {  **return red\_five**;  }   **public void** setRed\_five(**int** red\_five) {  **this**.**red\_five** = red\_five;  }   **public** Integer getRed\_six() {  **return red\_six**;  }   **public void** setRed\_six(**int** red\_six) {  **this**.**red\_six** = red\_six;  }   **public** Integer getBlue() {  **return blue**;  }   **public void** setBlue(**int** blue) {  **this**.**blue** = blue;  }   **public** String getDraw\_prize\_date() {  **return draw\_prize\_date**;  }   **public void** setDraw\_prize\_date(String draw\_prize\_date) {  **this**.**draw\_prize\_date** = draw\_prize\_date;  }  } |

八、在com.doubleball.mapper包下新建一个接口类DoubleballStatisticsMapper，代码如下：注意使用Mapper注解让系统自己扫包到这里

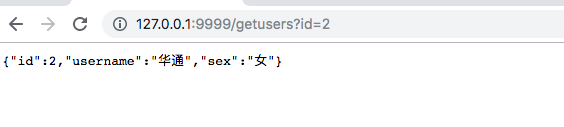
|  |
| --- |
| **package** com.doubleball.mapper;  **import** com.doubleball.entity.DoubleballStatistics; **import** org.apache.ibatis.annotations.\*; @Mapper **public interface** DoubleballStatisticsMapper {   @Select(**"select \* from DoubleballStatistics where id=#{id}"**)  DoubleballStatistics getBallById(@Param(**"id"**) Integer id);   @Insert(**"insert into DoubleballStatistics(issue,red\_one,red\_two,red\_three,red\_four,red\_five,red\_six,blue,draw\_prize\_date) values(#{issue},#{red\_one},#{red\_two},#{red\_three},#{red\_four},#{red\_five},#{red\_six},#{blue},#{draw\_prize\_date})"**)  **void** insertBall(@Param(**"issue"**) Integer issue,@Param(**"red\_one"**) Integer red\_one,@Param(**"red\_two"**) Integer red\_two,  @Param(**"red\_three"**) Integer red\_three,@Param(**"red\_four"**) Integer red\_four,@Param(**"red\_five"**) Integer red\_five,  @Param(**"red\_six"**) Integer red\_six,@Param(**"blue"**) Integer blue,@Param(**"draw\_prize\_date"**) String draw\_prize\_date);   @Update(**"update DoubleballStatistics set issue=#{issue},red\_one=#{red\_one},red\_two=#{red\_two},red\_three=#{red\_three},red\_four=#{red\_four},red\_five=#{red\_five},red\_six=#{red\_six},blue=#{blue},draw\_prize\_date=#{draw\_prize\_date} where id=#{id}"**)  **int** updateBall(@Param(**"id"**) Integer id,@Param(**"issue"**) Integer issue,@Param(**"red\_one"**) Integer red\_one,@Param(**"red\_two"**) Integer red\_two,  @Param(**"red\_three"**) Integer red\_three,@Param(**"red\_four"**) Integer red\_four,@Param(**"red\_five"**) Integer red\_five,  @Param(**"red\_six"**) Integer red\_six,@Param(**"blue"**) Integer blue,@Param(**"draw\_prize\_date"**) String draw\_prize\_date);   @Delete(**"delete from DoubleballStatistics where id=#{id}"**)  **void** deleteBll(@Param(**"id"**) Integer id);  } |

十、然后我们新建一个DoubleballController类进行访问数据，代码如下：这里用RestController注解返回JSON

|  |
| --- |
| @RestController  public class UserController {  @Autowired  **private** UserMapper **userMapper**;  @RequestMapping(**"/getuser"**)  **public** Object getUser(Integer id){  **return userMapper**.getUserById(id);  }   @RequestMapping(**"/insert"**)  **public** Object insert(String username,String sex){  **userMapper**.insertUser(username,sex);  **return "sucess"**;  }  } |

|  |
| --- |
|  |

然后在浏览器上输入信息 <http://127.0.0.1:9999/getusers?id=2> 回车即可



如果我们按照正常实际生产的方式来进行获取数据，需要新建Service层和实现层，实现的代码如下：

新建一个包com.wjxcms.service，在该包下新建一个接口UserService

|  |
| --- |
|  |

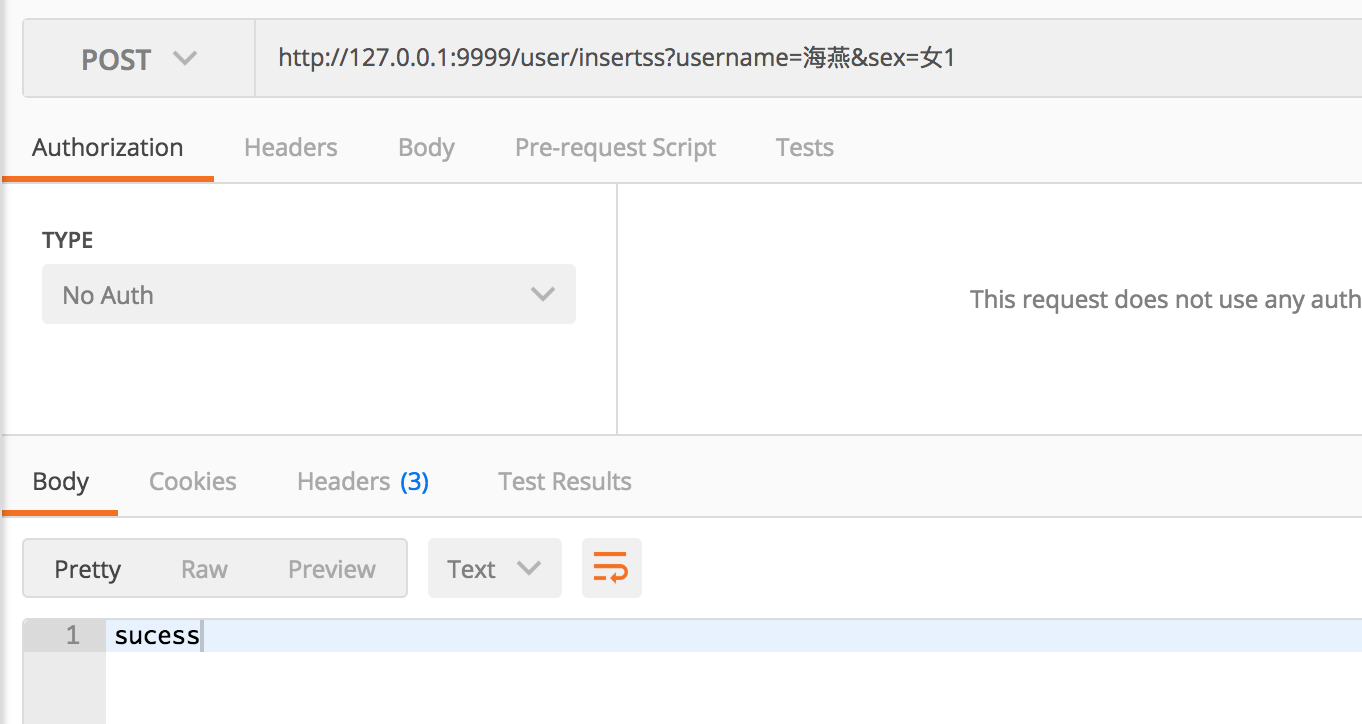
新建一个包com.wjxcms.service.impl，在该包下新建一个类UserServiceImpl。这里需要Service注解，也是让系统自动扫包。

|  |
| --- |
| **package** com.wjxcms.service.impl;  import com.wjxcms.mapper.UserMapper;  import com.wjxcms.service.UserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  @Service  **public class** UserServiceImpl **implements** UserService{  @Autowired  **private** UserMapper **userMapper**;  @Override  **public void** insertUser(String name,String sex){  **userMapper**.insertUser(name,sex);  System.***out***.println(**"执行 userMapper.insertUser(name,sex) "**);  }  } |

最后我们新建一个Controller进行访问数据，UserController代码如下：这里我们就用RestController注解返回JSON

|  |
| --- |
| @RestController  @RequestMapping(**"/user"**)  **public class** HelloCmsController {  @Autowired  **private** UserMapper **userMapper**;  @Autowired  **private** UserService **userService**;  @PostMapping(**"/insertss"**)  **public** Object insertUserss(String username,String sex){  **userService**.insertUser(username,sex);  System.***out***.println(**"执行 userService.insertUser(username,sex) "**);  **return "sucess"**;  }  } |

**最后将项目启动，什么都不用做变动，直接通过postman访问：http://127.0.0.1:9999/user/insertss?username=海燕&sex=女1，浏览器返回结果**



到这里就集成完毕了我们的MyBatis，我们这里就不讲深入了，这里只要学习过SSM框架的都很简单了，后面我们会集成一个非常牛X的框架，Mybatis-Plus，到时候你们会发现完全不用自己写SQL了

## SpringBoot事务管理

什么是事务？？？

**是指多个SQL作为单个逻辑工作单元执行的一系列操作，要么完全地执行，要么完全地不执行。**

**事物的四大特性（ACID）：**

**原子性(Atomicity)：**事务是一个原子操作，由一系列动作组成。事务的原子性确保动作要么全部完成，要么完全不起作用。对于其数据修改，要么全都执行，要么全都不执行。

**一致性(Consistency)：**一旦事务完成（不管成功还是失败），系统必须确保它所建模的业务处于一致的状态，而不会是部分完成部分失败。在现实中的数据不应该被破坏。事务在完成时，必须使所有的数据都保持一致状态。

**隔离性(Isolation)**：可能有许多事务会同时处理相同的数据，因此每个事务都应该与其他事务隔离开来，防止数据损坏。由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。

**持久性(Durability)**：一旦事务完成，无论发生什么系统错误，它的结果都不应该受到影响，这样就能从任何系统崩溃中恢复过来。通常情况下，事务的结果被写到持久化存储器中。事务完成之后，它对于系统的影响是永久性的。

比如说下面一系列操作（这里必须三条指令全部都执行完才能称为事务成功，只要有一条不成功，事物都不算成功，而且这里比如第三条信息失败了，第一条和第二条必须回滚）。

|  |
| --- |
| ①拿到一条数据  ②修改信息  ③更新回去 |

在SPringBoot中处理事务很简单，只需要一个注解，其他的SpringBoot自动帮我们实现。这里我们做一个简单的事务测试的例子，就是先将一个User的信息更新掉，然后再删除，我们在这两个过程中间制造一个异常看一下效果。

在UserMapper中新增两个SQL，一个更新，一个删除

|  |
| --- |
| **package** com.wjxcms.mapper;  @Mapper  **public interface** UserMapper {  @Update(**"update user set username=#{name} where id=#{id}"**)  **int** update(@Param(**"name"**) String name,@Param(**"id"**) Integer id);  @Delete(**"delete from user where id=#{id}"**)  **void** delete(@Param(**"id"**) Integer id);  } |

然后在Service接口中写一个事务测试方法

|  |
| --- |
| **package** com.wjxcms.service;  **public interface** UserService {  *// 执行我们的事务*  **public void** tranfor();  } |

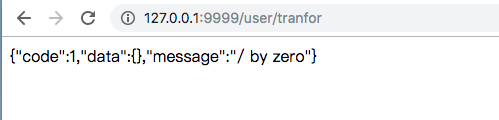
然后在UserServiceImpl中实现tranfor()方法，别忘了加上@Transactional

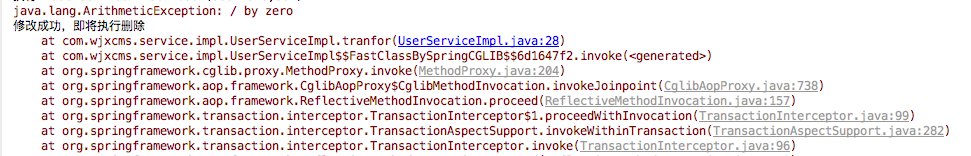
|  |
| --- |
| **package** com.wjxcms.service.impl;  @Service  @Transactional  **public class** UserServiceImpl **implements** UserService{  @Autowired  **private** UserMapper **userMapper**;  @Override  **public void** tranfor() {  *// 我们要做的操作，我们将某个用户的NAME更新成待删除，然后去删除，但是中间我们制造一个异常出来*  *//更新一条数据*  **userMapper**.update(**"更新语句"**,2) ;  *//抛出一个异常*  System.***out***.println(**"修改成功，即将执行删除"**);  **int** a=2/0 ;  System.***out***.println(**"异常已经发生"**);  *//删除一条数据*  **userMapper**.delete(2);  }  } |

注解返回JSON

|  |
| --- |
| @RestController  @RequestMapping(**"/user"**)  **public class** HelloCmsController {  @Autowired  **private** UserMapper **userMapper**;  @Autowired  **private** UserService **userService**;  @PostMapping(**"/insertss"**)  **public** Object insertUserss(String username,String sex){  **userService**.tranfor ();  **return "sucess"**;  }  } |

然后运行查看效果，发现Update根本不会成功，因为下面出现错误了，自动回滚，是不是很方便。这里顺便提一下数据缓存的操作，一般使用redis做数据缓存，这里画一下数据缓存的示例图。





**SpringBoot多数据源**

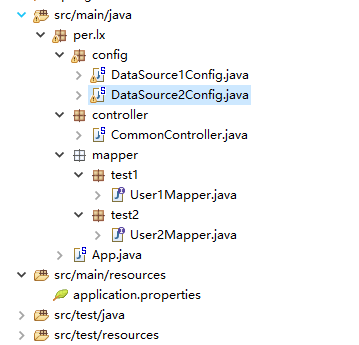
我们这里集成一下Mybatis的多数据源情况。

什么是多数据源：多数据源指的是如果一个web项目用到了多个数据库，那么就需要连接多个数据库操作，这就是多数据源。

多数据源的好处：我要用到A数据库(mysql)中的某些数据进行数据验证，并且要将数据插入B数据库中。这样做的好处是，不需要改变A中数据库的结构，就算是连接更多数据库的话，都不需要考虑其中结构问题，但是这就需要一个web项目跟多个数据库进行连接操作的技术，而这个技术正是我今天介绍的多数据源配置。

**项目结构**

在src/main/java中新建package为config，并且在其下新建两个文件，分别为DataSource1Config，DataSource2Config。结构如下所示：



我们只需在配置文件中配置一下我们的两个数据源，然后分别写一个配置类加载进去就可以了。

看配置文件（代表两个数据源：用spring.datasource.test\*区别）：

|  |
| --- |
| **spring.datasource.test1.url**=**jdbc:mysql://127.0.0.1:3306/double\_ball?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull**  **spring.datasource.test1.username**=**root**  **spring.datasource.test1.password**=**wjx123456**  **spring.datasource.test1.driverclassname**=**com.mysql.jdbc.Driver**  **spring.datasource.test2.url**=**jdbc:mysql://127.0.0.1:3306/double\_ball1?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull**  **spring.datasource.test2.username**=**root**  **spring.datasource.test2.password**=**wjx123456**  **spring.datasource.test2.driverclassname**=**com.mysql.jdbc.Driver** |

然后为每个配置源配置一个加载类：

配置源1的加载类：

|  |
| --- |
| **package** com.wjxsb.config;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.mybatis.spring.SqlSessionFactoryBean;  **import** org.mybatis.spring.SqlSessionTemplate;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.beans.factory.annotation.Qualifier;  **import** org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;  **import** org.springframework.boot.context.properties.ConfigurationProperties;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.context.annotation.Primary;  **import** org.springframework.jdbc.datasource.DataSourceTransactionManager;  **import** javax.sql.DataSource;  @Configuration  @MapperScan(basePackages = **"com.wjxsb.mapper.test1"**, sqlSessionTemplateRef = **"test1SqlSessionTemplate"**)  **public class** DataSource1Config {  @Bean(name = **"test1DataSource"**)  @ConfigurationProperties(prefix = **"spring.datasource.test1"**)  *//primary代表主库*  @Primary  **public** DataSource testDataSource() {  **return** DataSourceBuilder.*create*().build();  }    @Bean(name = **"test1SqlSessionFactory"**)  @Primary  **public** SqlSessionFactory testSqlSessionFactory(@Qualifier(**"test1DataSource"**) DataSource dataSource)  **throws** Exception {  SqlSessionFactoryBean bean = **new** SqlSessionFactoryBean();  bean.setDataSource(dataSource);  **return** bean.getObject();  }  @Bean(name = **"test1TransactionManager"**)  @Primary  **public** DataSourceTransactionManager testTransactionManager(@Qualifier(**"test1DataSource"**) DataSource dataSource) {  **return new** DataSourceTransactionManager(dataSource);  }  @Bean(name = **"test1SqlSessionTemplate"**)  @Primary  **public** SqlSessionTemplate testSqlSessionTemplate(  @Qualifier(**"test1SqlSessionFactory"**) SqlSessionFactory sqlSessionFactory) **throws** Exception {  **return new** SqlSessionTemplate(sqlSessionFactory);  }  } |

配置源2的加载类：

|  |
| --- |
| **package** com.wjxsb.config;  **import** org.apache.ibatis.session.SqlSessionFactory;  **import** org.mybatis.spring.SqlSessionFactoryBean;  **import** org.mybatis.spring.SqlSessionTemplate;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.beans.factory.annotation.Qualifier;  **import** org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;  **import** org.springframework.boot.context.properties.ConfigurationProperties;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.jdbc.datasource.DataSourceTransactionManager;  **import** javax.sql.DataSource;  @Configuration  @MapperScan(basePackages = **"com.wjxsb.mapper.test2"**, sqlSessionTemplateRef = **"test2SqlSessionTemplate"**)  **public class** DataSource2Config {  @Bean(name = **"test2DataSource"**)  @ConfigurationProperties(prefix = **"spring.datasource.test2"**)  **public** DataSource testDataSource() {  **return** DataSourceBuilder.*create*().build();  }  @Bean(name = **"test2SqlSessionFactory"**)  **public** SqlSessionFactory testSqlSessionFactory(@Qualifier(**"test2DataSource"**) DataSource dataSource)  **throws** Exception {  SqlSessionFactoryBean bean = **new** SqlSessionFactoryBean();  bean.setDataSource(dataSource);  **return** bean.getObject();  }  @Bean(name = **"test2TransactionManager"**)  **public** DataSourceTransactionManager testTransactionManager(@Qualifier(**"test2DataSource"**) DataSource dataSource) {  **return new** DataSourceTransactionManager(dataSource);  }  @Bean(name = **"test2SqlSessionTemplate"**)  **public** SqlSessionTemplate testSqlSessionTemplate(  @Qualifier(**"test2SqlSessionFactory"**) SqlSessionFactory sqlSessionFactory) **throws** Exception {  **return new** SqlSessionTemplate(sqlSessionFactory);  }  } |

建立一个实体类User：

|  |
| --- |
| **package** com.wjxsb.entity;  **public class** User {  **private int id**;  **private** String **username**;  **private** String **sex**;  **public int** getId() {  **return id**;  }  **public void** setId(**int** id) {  **this**.**id** = id;  }  **public** String getUsername() {  **return username**;  }  **public void** setUsername(String username) {  **this**.**username** = username;  }  **public** String getSex() {  **return sex**;  }  **public void** setSex(String sex) {  **this**.**sex** = sex;  }  @Override  **public String toString() {**  **return "User{" +"id=" + id+" , username='" + username+'\''+", sex='" + sex+'\'' +'}' ;**  **}** } |

分别在com.wjxsb.mapper.test1和com.wjxsb.mapper.test2包下建立两个接口mapper：

|  |
| --- |
| **package** com.wjxsb.mapper.test1;  **import** com.wjxsb.entity.User;  **import** org.apache.ibatis.annotations.Mapper;  **import** org.apache.ibatis.annotations.Param;  **import** org.apache.ibatis.annotations.Select;  @Mapper  **public interface** User1Mapper {  @Select(**"select \* from user where id=#{id}"**)  User getUserById(@Param(**"id"**) Integer id) ;  } |

|  |
| --- |
| **package** com.wjxsb.mapper.test2;  **import** com.wjxsb.entity.User;  **import** org.apache.ibatis.annotations.Param;  **import** org.apache.ibatis.annotations.Select;  **public interface** User2Mapper {  @Select(**"select \* from user where id=#{id}"**)  User getUserById(@Param(**"id"**) Integer id) ;  } |

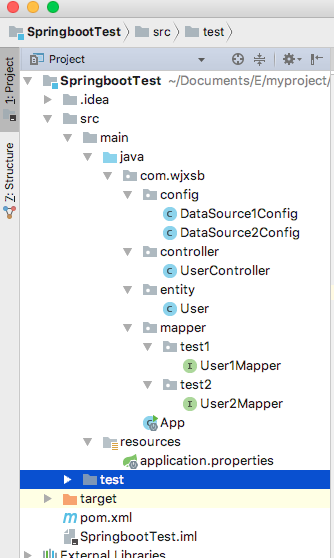
建立一个controller：

|  |
| --- |
| **package** com.wjxsb.controller;  **import** com.wjxsb.mapper.test1.User1Mapper;  **import** com.wjxsb.mapper.test2.User2Mapper;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public class** UserController {  @Autowired  **private** User1Mapper **user1Mapper**;  @Autowired  **private** User2Mapper **user2Mapper**;  @RequestMapping(**"/getUser1"**)  **public** Object getUser1(){  **return user1Mapper**.getUserById(1);  }  @RequestMapping(**"/getUser2"**)  **public** Object getUser2(){  **return user2Mapper**.getUserById(1);  } } |

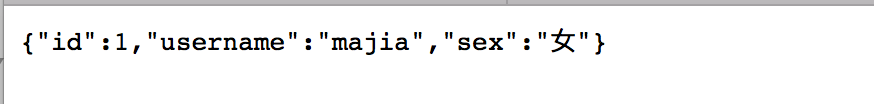
建立一个APP启动类：

|  |
| --- |
| **package** com.wjxsb;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public class** App {  **public static void** main(String[] args){  SpringApplication.*run*(App.**class**,args);  }  } |

整体结构如下：



运行APP后在浏览器输入<http://127.0.0.1:9999/getUser1>或者<http://127.0.0.1:9999/getUser2>，可以看到不同请求的结果：



这样的话就配置完成，可以调用不用的Mapper，实际上是存储在不同的库中。

这里大家会发现如果使用事务并且制造一个异常的话，会很明显的出现几种不同的效果，我们让mapper1和mapper2在不同顺序下面插入数据并且在中间抛出异常，发现只有一个插入成功了，另外一个没成功，为什么呢？就是因为@Primary的原因，这样就设置了主数据库。这里会发现设置了Primary的数据库只要出错都不会插入成功，都会事务回滚，大家注意了！！！！！

## SpringBoot集成Mybatis-Plus+代码生成

大家思考一个问题：平时业务代码不复杂的时候我们写什么代码写的最多，就是我们的SQL语句啊，配置那么多的Mapper.xml，还要配置什么resultMap这些东西，还要去管理paramtype。就会很容易出现错误，比如什么String can not parse to Integer之类的错误。

今天我们要学习一个超级牛X的框架，Mybatis-Plus，有了这个框架，我们可以做什么？不用写一句SQL，全部在JAVA代码中查找完毕，对于一些只做增删改查的管理系统，可以说业务层都可以免掉了。这样的框架是不是无比的强大，已经迫不及待的想要开始学习了，我们现在就正式开始。Mybatis-Plus官网：http://baomidou.oschina.io/mybatis-plus-doc/。

首先MP是什么：是一个 [Mybatis](http://www.mybatis.org/mybatis-3/) 的增强工具，在 Mybatis 的基础上只做增强不做改变，为简化开发、提高效率而生。

MP特性：无侵入，依赖少，损耗小，防SQL注入，通用CRUD，多种主键策略，代码生成，内置分页插件。。。。

**通用CRUD**：集成BaseMapper就可以使用MP封装的CRUD

多种主键策略：IdType.AUTO（自动），IdType.INPUT（用户输入），IdType.ID\_WORKER（自动），IdType.UUID（自动）。

配置方法，主键ID上加上注解：@TableId(value = "ID", type = IdType.AUTO)，一般情况下推荐大家使用自动增长主键。

**内置分页插件**：Page内置分页插件。

**代码生成**：MP自带代码生成工具，可以从Controller层直接生成到mapper层，包括实体类，让我们只关心请求地址和业务处理。

**集成Mybaits-Plus**

在pom.xml中导入相关jar包

|  |
| --- |
| *<!-- MP 核心库 -->*  <**dependency**>  <**groupId**>com.baomidou</**groupId**>  <**artifactId**>mybatis-plus</**artifactId**>  <**version**>3.2.0</**version**>  </**dependency**>  *<!-- 还需要加上如下 -->*  <**dependency**>  <**groupId**>com.baomidou</**groupId**>  <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  <**version**>3.2.0</**version**>  </**dependency**>  <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-autoconfigure</**artifactId**>  <**version**>1.3.2</**version**>  </**dependency**> |

到这里，Mybatis-Plus已经全部集成到系统里面去了，然后我们就来改造我们之前的UserMapper，这里也可以不是说改造，就是删除掉里面的内容然后继承至Mybatis-Plus相关内容就可以了，代码如下：

|  |
| --- |
| **package** com.wjxsb.mapper;  **import** com.baomidou.mybatisplus.core.mapper.BaseMapper;  **import** com.wjxsb.entity.User;  **import** org.apache.ibatis.annotations.Mapper;  **@Mapper**  **public interface UserMapper extends BaseMapper<User> {**  **}** |

User实体类代码如下：

|  |
| --- |
| **package** com.wjxsb.entity;  **public class** User {  **private int id**;  **private** String **username**;  **private** String **sex**;  **public int** getId() {  **return id**;  }  **public void** setId(**int** id) {  **this**.**id** = id;  }  **public** String getUsername() {  **return username**;  }  **public void** setUsername(String username) {  **this**.**username** = username;  }  **public** String getSex() {  **return sex**;  }  **public void** setSex(String sex) {  **this**.**sex** = sex;  }  @Override  **public** String toString() {  **return "User{"** +  **"id="** + **id** +  **", username='"** + **username** + **'\''** +  **", sex='"** + **sex** + **'\''** +  **'}'**;  }  } |

备注：

|  |
| --- |
| public static void main(String[] args) {  System.out.println( " \\\" " );  System.out.println( " \\\' " );  System.out.println( " } " );  System.out.println( '}' ); // 括号中的字符串不能写成类似 (' } ')，否则会提示“未结束的字符文字”  输出为：  \"  \'  }  } |

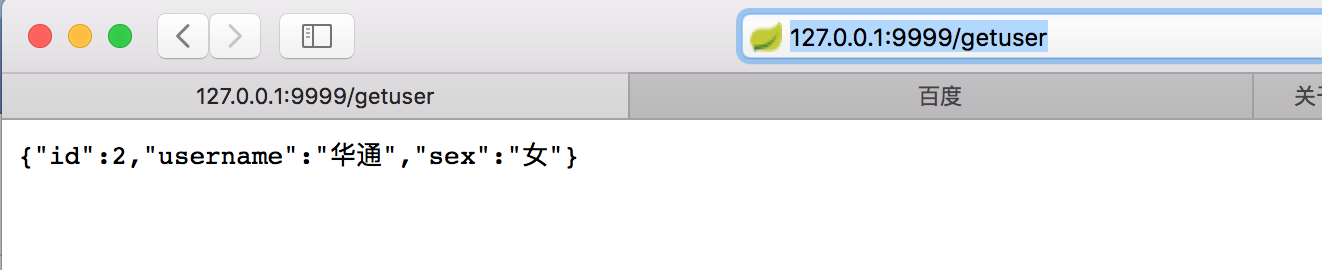
然后在controller文件中编写方法

|  |
| --- |
| **package** com.wjxsb.controller;  **import** com.wjxsb.mapper.UserMapper;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public class** SampleTest {  @Autowired  **private** UserMapper **userMapper**;  @RequestMapping(**"/getuser"**)  **public** Object test(){  **return userMapper**.selectById(2);  }  } |

然后编写启动类

|  |
| --- |
| **package** com.wjxsb;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  **public class** APP {  **public static void** main(String[] args){  SpringApplication.*run*(APP.**class**,args);  }  } |

然后启动启动类，在浏览器输入请求信息：



也可以添加几个接口服务，和接口实现，如下：

|  |
| --- |
| **package** com.wjxsb.service; **public interface** UserService {  **public void** tranfor(String username,Integer id); } |

|  |
| --- |
| **package** com.wjxsb.service.impl;  **import** com.wjxsb.entity.User; **import** com.wjxsb.mapper.UserMapper; **import** com.wjxsb.service.UserService; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.stereotype.Service; @Service **public class** UserServiceImpl **implements** UserService {   @Autowired  **private** UserMapper **userMapper**;   @Override  **public void** tranfor(String username, Integer id) {  User selectById = **userMapper**.selectById(id);  selectById.setUsername(username);  **userMapper**.updateById(selectById);  } } |

然后在controller文件中编写方法：

|  |
| --- |
| **package** com.wjxsb.controller;  **import** com.wjxsb.entity.User; **import** com.wjxsb.mapper.UserMapper; **import** com.wjxsb.service.UserService; **import** com.wjxsb.service.impl.UserServiceImpl; **import** org.springframework.beans.factory.annotation.Autowired; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RestController; @RestController **public class** UserController {  @Autowired  **private** UserMapper **userMapper**;   @Autowired  **private** UserService **userService**;   @Autowired  **private** UserServiceImpl **userServiceimpl**;   @RequestMapping(**"/getuser2"**)  **public** Object getuser(Integer id){  **return userMapper**.selectById(id);  }   @RequestMapping(**"/insert"**)  **public** Object insertUser(String username,String sex){  User user=**new** User();  user.setUsername(username);  user.setSex(sex);  **userMapper**.insert(user);  **return "success"**;  }   @RequestMapping(**"/update"**)  **public** String updateUser(String username,Integer id){  **userService**.tranfor(username,id);  **return "success"** ;  } } |

当我们需要对数据库操作时，常规的做法是建立dao，service，controller，mapper.xml文件，一系列繁琐的操作让我们总在做无用功，以前可以用逆向工程生成，现在mybatis-plus也给我们提供了模板，而且功能更加的灵活和强大，我们只要配置好基本的参数，就可以自动生成了。

AutoGenerator 是 MyBatis-Plus 的代码生成器，通过 AutoGenerator 可以快速生成 Entity、Mapper、Mapper XML、Service、Controller 等各个模块的代码，极大的提升了开发效率。

要使用代码生成工具，需要引入如下包：

|  |
| --- |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>2.1.8.RELEASE</**version**>  *<!--<relativePath/>-->* </**parent**>  <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter</**artifactId**>  <**version**>2.1.8.RELEASE</**version**>  </**dependency**>    <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**>  <**scope**>test</**scope**>  </**dependency**>  *<!--mybatis-plus自动的维护了mybatis以及mybatis-spring的依赖， 在springboot中这三者不能同时的出现，避免版本的冲突，表示：跳进过这个坑-->* <**dependency**>  <**groupId**>com.baomidou</**groupId**>  <**artifactId**>mybatis-plus-boot-starter</**artifactId**>  <**version**>3.2.0</**version**>  </**dependency**>   <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.21</**version**>  </**dependency**>   *<!-- 引入Druid依赖，阿里巴巴所提供的数据源 -->* <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>druid</**artifactId**>  <**version**>1.1.20</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.projectlombok</**groupId**>  <**artifactId**>lombok</**artifactId**>  <**version**>1.18.10</**version**>  <**scope**>provided</**scope**>  </**dependency**>   <**dependency**>  <**groupId**>com.baomidou</**groupId**>  <**artifactId**>mybatis-plus-generator</**artifactId**>  <**version**>3.2.0</**version**>  </**dependency**>  *<!--freemarker的模板引擎-->* <**dependency**>  <**groupId**>org.freemarker</**groupId**>  <**artifactId**>freemarker</**artifactId**>  <**version**>2.3.29</**version**>  </**dependency**>  </**dependencies**> |

新建一个mp包，然后新建一个文件，加入代码生成器：

|  |
| --- |
| **package** com.wjxsbmp.mybatisplus; **import** com.baomidou.mybatisplus.core.toolkit.StringPool; **import** com.baomidou.mybatisplus.generator.AutoGenerator; **import** com.baomidou.mybatisplus.generator.InjectionConfig; **import** com.baomidou.mybatisplus.generator.config.\*; **import** com.baomidou.mybatisplus.generator.config.po.TableInfo; **import** com.baomidou.mybatisplus.generator.config.rules.NamingStrategy; **import** com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;  **import** java.util.ArrayList; **import** java.util.List;  **public class** CodeGenerator {  *//作者* **private static final** String ***author*** = **"wjx"**;   *//数据库 // private static final String url = "jdbc:mysql://192.168.3.172:3306/poc?useUnicode=true&useSSL=false&characterEncoding=utf8";* **private static final** String ***url*** = **"jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull"**;   **private static final** String ***driverName*** = **"com.mysql.jdbc.Driver"**;  **private static final** String ***userName*** = **"root"**;  **private static final** String ***password*** = **"wjx123456"**;  **private static final** String ***table\_name*** = **"user"**;   *//表对应的基类名* **private static final** String ***entityName*** = **"User"**;   *//如UserController,UserService* **private static final** String ***name\_prex*** = **"User"**;   *//生成在该目录下* **private static final** String ***package\_name*** = **"com.wjxsbmp.mybatisplus"**;    **public static void** main(String[] args) {  *// 代码生成器* AutoGenerator mpg = **new** AutoGenerator();   *// 全局配置* GlobalConfig gc = **new** GlobalConfig();  String projectPath = System.*getProperty*(**"user.dir"**); *// 得到工程的根目录，例如/Users/finup/Documents/E/myproject/test/springboottest/SpringbootTest3*  gc.setOutputDir(projectPath + **"/src/main/java"**);  gc.setAuthor(***author***);  gc.setOpen(**false**);  gc.setEntityName(***entityName***);  gc.setMapperName(***name\_prex*** + **"Mapper"**);  gc.setControllerName(***name\_prex*** + **"Controller"**);  gc.setServiceName(***name\_prex*** + **"Service"**);  gc.setServiceImplName(***name\_prex*** + **"ServiceImpl"**);  mpg.setGlobalConfig(gc);   *// 数据源配置* DataSourceConfig dsc = **new** DataSourceConfig();  dsc.setUrl(***url***);  *// dsc.setSchemaName("public");* dsc.setDriverName(***driverName***);  dsc.setUsername(***userName***);  dsc.setPassword(***password***);  mpg.setDataSource(dsc);   *// 包配置* PackageConfig pc = **new** PackageConfig();  *//pc.setModuleName(scanner("模块名"));* pc.setParent(***package\_name***);  mpg.setPackageInfo(pc);   *// 自定义配置* InjectionConfig cfg = **new** InjectionConfig() {  @Override  **public void** initMap() {  *// to do nothing* }  };  List<FileOutConfig> focList = **new** ArrayList<>();  focList.add(**new** FileOutConfig(**"/templates/mapper.xml.ftl"**) {  @Override  **public** String outputFile(TableInfo tableInfo) {  *// 自定义输入文件名称* **return** projectPath + **"/src/main/resources/mapper/"** */\*+ pc.getModuleName()\*/* + **"/"** + tableInfo.getEntityName() + **"Mapper"** + StringPool.***DOT\_XML***;  }  });  cfg.setFileOutConfigList(focList);  mpg.setCfg(cfg);  mpg.setTemplate(**new** TemplateConfig().setXml(**null**));   *// 策略配置* StrategyConfig strategy = **new** StrategyConfig();  strategy.setNaming(NamingStrategy.***underline\_to\_camel***);  strategy.setColumnNaming(NamingStrategy.***underline\_to\_camel***); *// strategy.setSuperEntityClass("com.baomidou.ant.common.BaseEntity");* strategy.setEntityLombokModel(**true**);  strategy.setRestControllerStyle(**true**); *// strategy.setSuperControllerClass("com.baomidou.ant.common.BaseController");* strategy.setInclude(***table\_name***);  strategy.setSuperEntityColumns(**"id"**);  strategy.setControllerMappingHyphenStyle(**true**);  strategy.setTablePrefix(pc.getModuleName() + **"\_"**);  mpg.setStrategy(strategy);  mpg.setTemplateEngine(**new** FreemarkerTemplateEngine());  mpg.execute();  }  } |

或者下面这个也可以

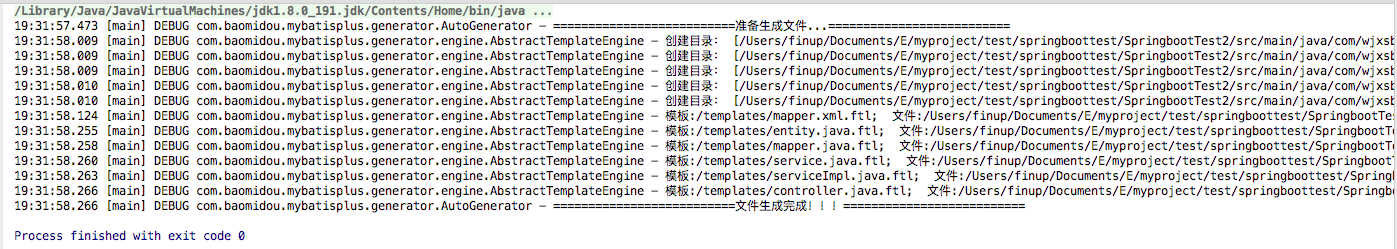
|  |
| --- |
| package com.wjxmp.generator;  import com.baomidou.mybatisplus.annotation.DbType;  import com.baomidou.mybatisplus.annotation.IdType;  import com.baomidou.mybatisplus.core.toolkit.StringPool;  import com.baomidou.mybatisplus.generator.AutoGenerator;  import com.baomidou.mybatisplus.generator.InjectionConfig;  import com.baomidou.mybatisplus.generator.config.\*;  import com.baomidou.mybatisplus.generator.config.po.TableInfo;  import com.baomidou.mybatisplus.generator.config.rules.NamingStrategy;  import com.baomidou.mybatisplus.generator.engine.FreemarkerTemplateEngine;  import java.util.ArrayList;  import java.util.List;  public class MysqlGenerator {  public static void main(String[] args) {  // Generator(new String[]{"x", "xx"});  Generator(new String[]{"user","role","permission"});  }  public static void Generator(String[] tableName) {  String projectPath = System.getProperty("user.dir"); *// 项目路径*  //============================== 全局配置  GlobalConfig gc = new GlobalConfig();  gc.setOutputDir(projectPath + "/src/main/java") *// 输出文件路径*  .setActiveRecord(true)// 是否支持 AR  .setAuthor("wjx") //设置作者名字  .setFileOverride(true) //文件覆盖(全新文件)  .setIdType(IdType.AUTO)//主键策略  .setBaseResultMap(true) //SQL 映射文件  .setBaseColumnList(true)//SQL 片段  .setOpen(false);  //============================== 数据源配置  DataSourceConfig dsc = new DataSourceConfig();  *// dsc.setDbType(DbType.MARIADB)* dsc.setDbType(DbType.***MYSQL***)  // .setUrl("jdbc:mariadb://localhost:3306/test")  .setUrl("jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull")  // .setDriverName("org.mariadb.jdbc.Driver")  .setDriverName("com.mysql.jdbc.Driver")  .setUsername("root")  //.setSchemaName("public")  .setPassword("wjx123456");  //============================== 包配置  PackageConfig pc = new PackageConfig();  pc.setParent("com.wjxmp")//配置父包路径  .setModuleName("")//配置业务包路径，如果配置的话，会在com.wjxmp.base下生成mapper等目录  .setMapper("mapper")  .setEntity("entity")  .setService("service")  .setController("controller");  //.setServiceImpl("service.impl"); 会自动生成 impl，可以不设定  //============================== 自定义配置  InjectionConfig cfg = new InjectionConfig() {  @Override  public void initMap() {  // to do nothing  }  };  List<FileOutConfig> focList = new ArrayList<>();  focList.add(new FileOutConfig("/templates/mapper.xml.ftl") {  @Override  public String outputFile(TableInfo tableInfo) {  // 自定义输入文件名称  return projectPath + "/src/main/resources/mapper/" + pc.getModuleName()  + "/" + tableInfo.getEntityName() + "Mapper" + StringPool.DOT\_XML;  }  });  cfg.setFileOutConfigList(focList);  //============================== 策略配置  StrategyConfig strategy = new StrategyConfig();  strategy.setNaming(NamingStrategy.underline\_to\_camel)//设置命名规则 underline\_to\_camel 底线变驼峰  .setColumnNaming(NamingStrategy.underline\_to\_camel)//设置设置列命名 underline\_to\_camel 底线变驼峰  //.setSuperEntityClass("com.maoxs.pojo")//设置继承类  //.setSuperControllerClass("com.maoxs.controller")//设置继承类  .setEntityLombokModel(true)//是否加入lombok  // .setInclude(tableName)//设置表名  .setInclude("user","role","permission")//设置表名  //.setSuperEntityColumns("id") //设置超级超级列  .setControllerMappingHyphenStyle(true)//设置controller映射联字符  .setTablePrefix(pc.getModuleName() + "\_");//表的前缀  //============================== 生成配置  AutoGenerator mpg = new AutoGenerator();  mpg.setCfg(cfg)  .setTemplate(new TemplateConfig().setXml(null))  .setGlobalConfig(gc)  .setDataSource(dsc)  .setPackageInfo(pc)  .setStrategy(strategy)  // 选择 freemarker 引擎需要指定如下加，注意 pom 依赖必须有！  .setTemplateEngine(new FreemarkerTemplateEngine());  mpg.execute();  }  } |

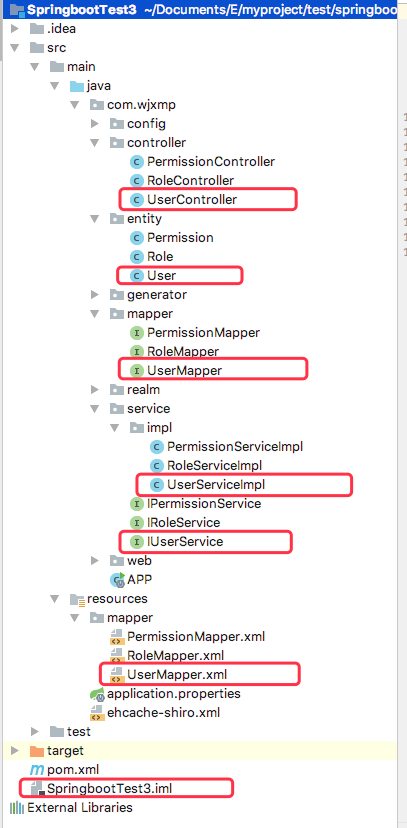
**在数据库中增加一个数据表user**

|  |
| --- |
| CREATE TABLE `user` (  `id` bigint(19) NOT NULL AUTO\_INCREMENT COMMENT '用户id',  `username` varchar(10) DEFAULT NULL COMMENT '用户姓名',  `sex` varchar(10) DEFAULT NULL COMMENT '性别',  `roleid` int(11) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='用户表'; |

在/src/main/java/resources下面添加配置文件application.properties，配置一下数据库的信息

|  |
| --- |
| spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull spring.datasource.username=root spring.datasource.password=wjx123456 spring.datasource.driver-class-name=com.mysql.jdbc.Driver  # 配置服务器端口，默认是8080，可以不用配置server.port=9999 |

运行代码生成器，会生成如下图标红处所示的文件或者包  




UserController内容如下：

|  |
| --- |
| **package** com.wjxsbmp.mybatisplus.controller; **import** org.springframework.web.bind.annotation.RequestMapping; **import** org.springframework.web.bind.annotation.RestController; @RestController @RequestMapping(**"/user"**) **public class** UserController {  } |

User内容如下：

|  |
| --- |
| **package com.wjxmp.entity;**  **import com.baomidou.mybatisplus.annotation.IdType;**  **import com.baomidou.mybatisplus.extension.activerecord.Model;**  **import com.baomidou.mybatisplus.annotation.TableId;**  **import java.io.Serializable;**  **import lombok.Data;**  **import lombok.EqualsAndHashCode;**  **import lombok.experimental.Accessors;**  **@Data**  **@EqualsAndHashCode(callSuper = false)**  **@Accessors(chain = true)**  **public class User extends Model<User> {**  **private static final long serialVersionUID = 1L;**  **/\*\***  **\* 用户id**  **\*/**  **@TableId(value = "id", type = IdType.AUTO)**  **private Integer id;**  **/\*\***  **\* 用户姓名**  **\*/**  **private String username;**  **/\*\***  **\* 性别**  **\*/**  **private String sex;**  **private Integer roleid;**  **@Override**  **protected Serializable pkVal() {**  **return this.id;**  **}**  **}** |

UserMapper内容如下：

|  |
| --- |
| **package com.wjxmp.mapper;**  **import com.wjxmp.entity.User;**  **import com.baomidou.mybatisplus.core.mapper.BaseMapper;**  **public interface UserMapper extends BaseMapper<User> {**  **}** |

备注：要用UserMapper，需要加上配置信息@Mapper

|  |
| --- |
| **package com.wjxmp.mapper;**  **import com.wjxmp.entity.User;**  **import com.baomidou.mybatisplus.core.mapper.BaseMapper;**  **import org.apache.ibatis.annotations.Mapper;**  **@Mapper**  **public interface UserMapper extends BaseMapper<User> {**  **}** |

UserMapper.xml内容如下：

|  |
| --- |
| *<?xml version="1.0" encoding="UTF-8"?>*  *<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">*  *<mapper namespace="com.wjxmp.mapper.UserMapper">*  *<!-- 通用查询映射结果 -->*  *<resultMap id="BaseResultMap" type="com.wjxmp.entity.User">*  *<id column="id" property="id" />*  *<result column="username" property="username" />*  *<result column="sex" property="sex" />*  *<result column="roleid" property="roleid" />*  *</resultMap>*  *<!-- 通用查询结果列 -->*  *<sql id="Base\_Column\_List">*  *id, username, sex, roleid*  *</sql>*  *</mapper>* |

IUserService内容如下

|  |
| --- |
| *package com.wjxmp.service;*  *import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;*  *import com.baomidou.mybatisplus.core.metadata.IPage;*  *import com.baomidou.mybatisplus.extension.plugins.pagination.Page;*  *import com.wjxmp.entity.User;*  *import com.baomidou.mybatisplus.extension.service.IService;*  *public interface IUserService extends IService<User> {*  *}* |

UserServiceImpl内容如下

|  |
| --- |
| *package com.wjxmp.service.impl;*  *import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;*  *import com.baomidou.mybatisplus.core.metadata.IPage;*  *import com.baomidou.mybatisplus.extension.plugins.pagination.Page;*  *import com.wjxmp.entity.User;*  *import com.wjxmp.mapper.UserMapper;*  *import com.wjxmp.service.IUserService;*  *import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;*  *import org.springframework.stereotype.Service;*  *@Service*  *public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements IUserService {*  *}* |

如果想用到分页功能，需要新建一个com.majiaxueyuan.config用来存储配置信息类的包，新建一个配置Mybatis-Plus的类MybatisPlusConfig

在SpringBoot中使用配置文件时，务必在类上加上Configuration的注解方便系统启动时加载。

|  |
| --- |
| package com.wjxmp.config;  import com.baomidou.mybatisplus.autoconfigure.SpringBootVFS;  import com.baomidou.mybatisplus.core.MybatisConfiguration;  import com.baomidou.mybatisplus.core.MybatisXMLLanguageDriver;  import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;  import com.baomidou.mybatisplus.extension.spring.MybatisSqlSessionFactoryBean;  import org.apache.ibatis.mapping.DatabaseIdProvider;  import org.apache.ibatis.plugin.Interceptor;  import org.mybatis.spring.boot.autoconfigure.MybatisProperties;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.core.io.DefaultResourceLoader;  import org.springframework.core.io.ResourceLoader;  import org.springframework.util.ObjectUtils;  import org.springframework.util.StringUtils;  import javax.sql.DataSource;  @Configuration  public class MybatisPlusConfig {  @Autowired  private DataSource dataSource;  @Autowired  private MybatisProperties properties;  @Autowired  private ResourceLoader resourceLoader = new DefaultResourceLoader();  @Autowired(required = false)  private Interceptor[] interceptors;  @Autowired(required = false)  private DatabaseIdProvider databaseIdProvider;  /\*\*      \*   mybatis-plus分页插件      \*/  @Bean  public PaginationInterceptor paginationInterceptor() {  PaginationInterceptor page = new PaginationInterceptor();  page.setDialectType("mysql");  return page;  }  /\*\*      \* 这里全部使用mybatis-autoconfigure 已经自动加载的资源。不手动指定      \* 配置文件和mybatis-boot的配置文件同步      \* @return      \*/  @Bean  public MybatisSqlSessionFactoryBean mybatisSqlSessionFactoryBean() {  MybatisSqlSessionFactoryBean mybatisPlus = new MybatisSqlSessionFactoryBean();  mybatisPlus.setDataSource(dataSource);  mybatisPlus.setVfs(SpringBootVFS.class);  if (StringUtils.hasText(this.properties.getConfigLocation())) {  mybatisPlus.setConfigLocation(this.resourceLoader.getResource(this.properties.getConfigLocation()));  }  mybatisPlus.setConfiguration(properties.getConfiguration());  if (!ObjectUtils.isEmpty(this.interceptors)) {  mybatisPlus.setPlugins(this.interceptors);  }  MybatisConfiguration mc = new MybatisConfiguration();  mc.setDefaultScriptingLanguage(MybatisXMLLanguageDriver.class);  mybatisPlus.setConfiguration(mc);  if (this.databaseIdProvider != null) {  mybatisPlus.setDatabaseIdProvider(this.databaseIdProvider);  }  if (StringUtils.hasLength(this.properties.getTypeAliasesPackage())) {  mybatisPlus.setTypeAliasesPackage(this.properties.getTypeAliasesPackage());  }  if (StringUtils.hasLength(this.properties.getTypeHandlersPackage())) {  mybatisPlus.setTypeHandlersPackage(this.properties.getTypeHandlersPackage());  }  if (!ObjectUtils.isEmpty(this.properties.resolveMapperLocations())) {  mybatisPlus.setMapperLocations(this.properties.resolveMapperLocations());  }  return mybatisPlus;  }  } |

然后在IUserService中添加接口方法如下

|  |
| --- |
| package com.wjxmp.service;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.baomidou.mybatisplus.extension.service.IService;  public interface IUserService extends IService<User> {  IPage<User> getAllUsers(Page<User> page, QueryWrapper queryWrapper);  } |

在UserServiceImpl中实现接口方法

|  |
| --- |
| package com.wjxmp.service.impl;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IUserService;  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  import org.springframework.stereotype.Service;  @Service  public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements IUserService {  @Override  public IPage<User> getAllUsers(Page<User> page, QueryWrapper queryWrapper) {  return baseMapper.selectPage(page, queryWrapper);  }  } |

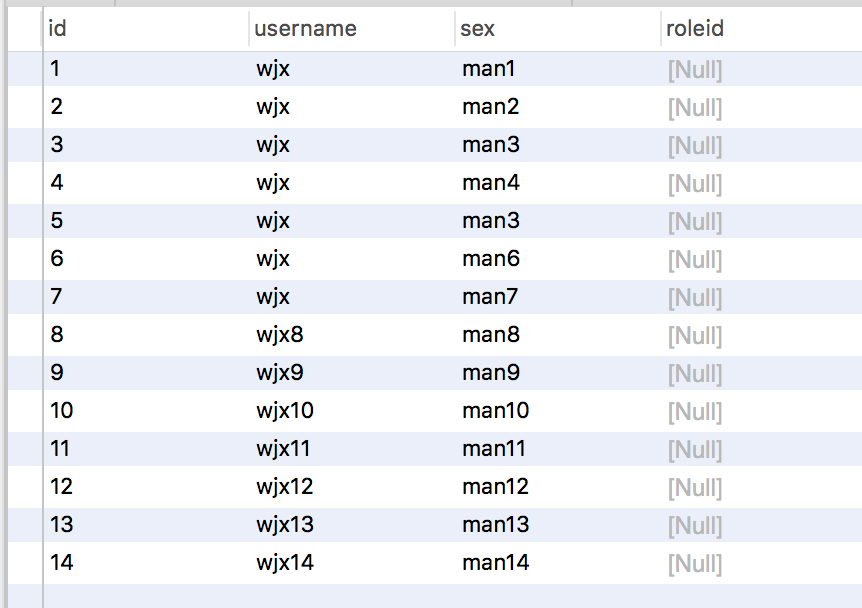
然后在UserController中添加请求方法

|  |
| --- |
| package com.wjxmp.web;  import com.baomidou.mybatisplus.core.conditions.Wrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IPermissionService;  import com.wjxmp.service.IRoleService;  import com.wjxmp.service.IUserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/user")  public class WebUserController {  @Autowired  private IUserService iUserService;  @Autowired  private UserMapper userMapper;  @RequestMapping("/getuser")  public Object getUser(Integer pgno,Integer pgsize){  // QueryWrapper<User> wrapper = new QueryWrapper<User>();  // wrapper.eq("username","wjx");  // return userMapper.selectList(wrapper);  Page<User> page = new Page<User>(pgno, pgsize);  QueryWrapper<User> wrapper = new QueryWrapper<User>();  wrapper.eq("username","wjx");  IPage<User> iPage =iUserService.getAllUsers(page, wrapper);  return iPage;  }  } |

添加启动类

|  |
| --- |
| package com.wjxmp;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  /\*\*  \* @author wangjunxiong  \* @date 2019/9/22 上午10:00  \*/  @SpringBootApplication  public class APP {  public static void main(String[] args){  SpringApplication.run(APP.class,args);  }  } |

在数据库user表中添加数据，类似如下



然后启动启动类，在浏览器中输入请求信息，如下所示：

http://127.0.0.1:9999/user/getuser?pgno=2&pgsize=3



【--🡪--🡪--🡪--🡪--🡪--🡪--🡪--🡪--🡪以下暂时没有用到--🡪--🡪--🡪--🡪--🡪--🡪--🡪--🡪--🡪

然后我们需要对实体类进行一个简单的修饰，并且告诉MP该表的主键是什么，然后主键的生成策略是什么。

|  |
| --- |
| @TableName("user")  public class User {  @TableId(value = "id", type = IdType.AUTO)  private Integer id;  private String username;             。。。。。。 |

然后我们这里来执行一下条件查询，这里我们要用到Mybatis-Plus中的包装（Wrapper）去构建我们的条件查询

下面举几个Wrapper语句的示例，

|  |
| --- |
| Wrapper<T> wrapper = new EntityWrapper<>();           构建一个实体类的包装工具  wrapper.eq("username", "LIAOXIANG");                          做条件判断  wrapper.between("id", 0, 100);                                         做范围判断  wrapper.groupBy("username");                                        分组  wrapper.isNotNull("username");                                        不为空判断  wrapper.orderBy("id", false);                                             排序，从小打大为true，反之false  。。。。。。。。。。  还有很多API，我这里就不一一给大家做讲解了，我们今天就用几个来演示一下，大家下去自己研究研究更多的功能 |

**代码生成工具**

使用我们的代码生成工具，需要导入一个模版引擎：

|  |
| --- |
| <dependency>  <groupId>org.apache.velocity</groupId>  <artifactId>velocity-engine-core</artifactId>  <version>2.0</version>  </dependency> |

代码我直接发给大家，然后大家下去跟着我自己练习一下就可以了。  
【<----<----<----<----<----<----<----以上暂时没有用到<----<----<----<----<----<----<----】

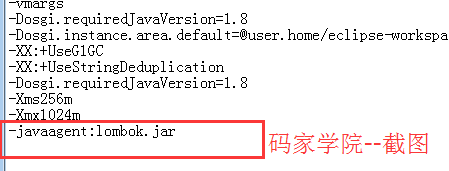
## SpringBoot集成lombok工具(生成setget方法，构造函数，打印日志)

平时要去打印日志

Logger log = LoggerFactory.getLogger(Class.class);

log.info();

平时我们写的一些重复代码，比如每个实体类的setter，getter方法，给每个类写上Logger获取的方法，这样写的话太繁琐，我们就可以使用lombok的工具去简化这个配置的操作。首先先要在Eclipse下安装lombok，在控制台中，使用java -jar lombok.jar为eclipse安装lombok插件，安装成功后需要重启eclipse，在eclipse目录下可以看到lombok.jar这个jar包，然后我们打开eclipse.ini配置文件查看到如下代码就正确



然后我们需要在pom.xml中引入lombok的jar包了

|  |
| --- |
| *<!--lombok 最新依赖-->* <**dependency**>  <**groupId**>org.projectlombok</**groupId**>  <**artifactId**>lombok</**artifactId**>  <**version**>1.18.10</**version**>  <**scope**>provided</**scope**> </**dependency**> |

到这里lombok就集成完毕了，我说过，我们要做最牛X的SpringBoot的教程，下面我们就看看这个工具的牛X之处

这里是一些常用的注解，还有更多的请大家参见官网：[http://projectlombok.org/features/index.](http://projectlombok.org/features/index.html)

|  |
| --- |
| @Data   ：注解在类上；提供类所有属性的 getter 和 setter 方法，此外还提供了equals、canEqual、hashCode、toString 方法  @Setter：注解在属性上；为属性提供 setting 方法  @Getter：注解在属性上；为属性提供 getting 方法  @Log4j ：注解在类上；为类提供一个属性名为log 的 log4j 日志对象**(应用时用@Log4j2)**  @SneakyThrows：无需在签名处显式抛出异常  @Slf4j: 注解在类上；为类提供一个属性名为log 的 Slf4j 日志对像  @NoArgsConstructor：注解在类上；为类提供一个无参的构造方法  @AllArgsConstructor：注解在类上；为类提供一个全参的构造方法 |

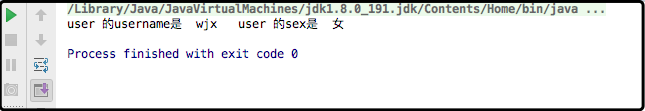
新建一个类User，内容如下：

|  |
| --- |
| **import** lombok.Getter; **import** lombok.Setter;  @Setter @Getter **public class** User {  String **username**;  String **sex**; } |

新建一个类LombokTest，内容如下：

|  |
| --- |
| **public class** LombokTest {  **public static void** main(String[] args) {  User user=**new** User(); // 此时默认是无参函数  user.setUsername(**"wjx"**);  user.setSex(**"女"**);  System.***out***.println(**"user 的username是 "**+user.getUsername()+**" user 的sex是 "**+user.getSex());  } } |

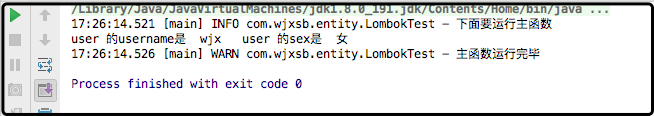
运行上面的主函数，得到如下结果：



**如果在类LombokTest上边添加注解@Slf4j**，内容如下：

|  |
| --- |
| @Slf4j **public class** LombokTest {  **public static void** main(String[] args) {  ***log***.info(**"下面要运行主函数"**);  User user=**new** User();  user.setUsername(**"wjx"**);  user.setSex(**"女"**);  System.***out***.println(**"user 的username是 "**+user.getUsername()+**" user 的sex是 "**+user.getSex());  ***log***.warn(**"主函数运行完毕"**);  } } |

运行上面的主函数，得到如下结果：



**如果在类LombokTest上将注解@Slf4j换成@Log4j2**，内容如下：

|  |
| --- |
| **import** lombok.extern.log4j.**Log4j2**;  **@Log4j2** **public class** LombokTest {  **public static void** main(String[] args) {  ***log***.info(**"下面要运行主函数"**);  User user=**new** User();  user.setUsername(**"wjx"**);  user.setSex(**"女"**);  System.***out***.println(**"user 的username是 "**+user.getUsername()+**" user 的sex是 "**+user.getSex());  ***log***.warn(**"主函数运行完毕"**);  } } |

**注意**：此处不能使用**@Log4j**，而是要使用**@Log4j2**。否则会在使用log.info()等时提示错误。

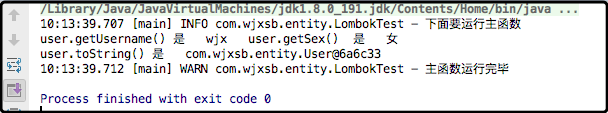
如**果**要在类User上添加注释@NoArgsConstructor，User就包含了无参构造函数

|  |
| --- |
| @Setter @Getter @NoArgsConstructor **public class** User {  String **username**;  String **sex**; } |

**那在类LombokTes**t中代码如下，内容如下：

|  |
| --- |
| **import** lombok.extern.log4j.Log4j2; @Log4j2 **public class** LombokTest {  **public static void** main(String[] args) {  ***log***.info(**"下面要运行主函数"**);  User user=**new** User();  user.setUsername(**"wjx"**);  user.setSex(**"女"**);  System.***out***.println(**"user.getUsername() 是 "**+user.getUsername()+**" user.getSex() 是 "**+user.getSex());  System.***out***.println(**"user.toString() 是 "**+user.toString());  ***log***.warn(**"主函数运行完毕"**);  } } |

**运行主函数，结果如下**



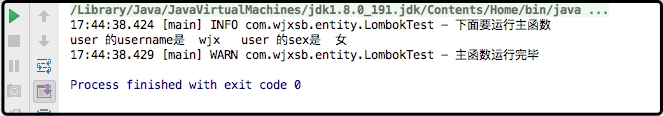
如**果**要在类User上添加注释@AllArgsConstructor，User就包含了有参构造函数

|  |
| --- |
| @Setter @Getter @AllArgsConstructor **public class** User {  String **username**;  String **sex**; } |

**那在类LombokTes**t中需要将 “User user=new User()”修改掉，内容如下：

|  |
| --- |
| **import** lombok.extern.log4j.Log4j2; @Log4j2 **public class** LombokTest {  **public static void** main(String[] args) {  ***log***.info(**"下面要运行主函数"**);  User user=**new** User(**"wjx"**,**"女"**);System.***out***.println(**"user 的username是 "**+user.getUsername()+**" user 的sex是 "**+user.getSex());  ***log***.warn(**"主函数运行完毕"**);  } } |

**运行主函数，结果如下**



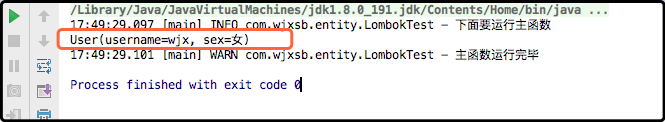
如**果**要在类User上所有的注释都删掉，换成加注释@Data，User就包含了无参构造函数

|  |
| --- |
| **import** lombok.Data; @Data **public class** User {  String **username**;  String **sex**; } |

**那在类LombokTes**t中需要将 “User user=**new** User(**"wjx"**,**"女"**)”修改掉，内容如下：

|  |
| --- |
| **import** lombok.extern.log4j.Log4j2; @Log4j2 **public class** LombokTest {  **public static void** main(String[] args) {  ***log***.info(**"下面要运行主函数"**);  User user=**new** User();  user.setUsername(**"wjx"**);  user.setSex(**"女"**);  System.***out***.println(user.toString());  ***log***.warn(**"主函数运行完毕"**);  } } |

**运行主函数，结果如下**



## SpringBoot集成Shiro安全框架

Apache Shiro是Java的一个安全框架。目前，使用Apache Shiro的人越来越多，因为它相当简单，对比Spring Security，可能没有Spring Security做的功能强大，但是在实际工作时可能并不需要那么复杂的东西，所以使用小而简单的Shiro就足够了。对于它俩到底哪个好，这个不必纠结，能更简单的解决项目问题就好了。

Shiro有哪些功能：身份认证/登录，授权，会话管理，加密，**Web支持。。。。主要用来控制登录，角色权限管理。**

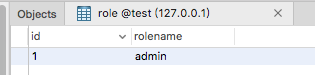
**记住这么一点，Shiro不会去维护用户、维护权限；这些需要我们自己去设计/提供；通过Realm让开发人员自己注入。**

**首先引入shiro相关jar包**

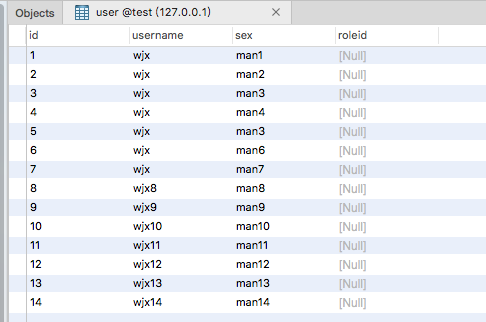
|  |
| --- |
| <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-spring</artifactId>  <version>1.4.0</version>  </dependency>  <dependency>  <groupId>org.apache.shiro</groupId>  <artifactId>shiro-ehcache</artifactId>  <version>1.4.0</version>  </dependency> |

新建一个表，并添加一条记录

|  |
| --- |
| CREATE TABLE `role` (  `id` int(11) NOT NULL,  `rolename` varchar(255) DEFAULT NULL,  PRIMARY KEY (`id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |



在user表中新增一列roleid



## 新建一个permission表：

CREATE TABLE `permission` (

`id` int(11) NOT NULL,

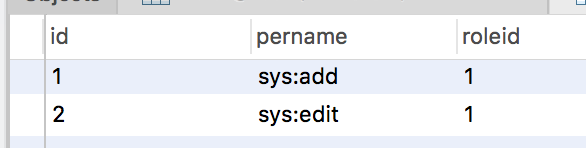
`pername` varchar(255) DEFAULT NULL,

`roleid` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8

## 添加如下记录



## 然后在src/main/resources下新建一个关于shiro的缓存配置文件ehcache-shiro.xml

|  |
| --- |
| **<?xml version="1.0" encoding="UTF-8"?>**  <ehcache updateCheck="false" name="cacheManagerConfigFile">  <defaultCache            maxElementsInMemory="10000"            eternal="false"             timeToIdleSeconds="120"            timeToLiveSeconds="120"            overflowToDisk="false"            diskPersistent="false"            diskExpiryThreadIntervalSeconds="120"            memoryStoreEvictionPolicy="LRU"/>   <cache name="shiro-activeSessionCache"     eternal="false"          maxElementsInMemory="10000"          overflowToDisk="false"           timeToIdleSeconds="0"          timeToLiveSeconds="0"          statistics="true"/>  </ehcache> |

如果该缓存配置文件不行，用下面配置了authorizationCache和authenticationCache的缓存文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <ehcache updateCheck="false" name="cacheManagerConfigFile">  <defaultCache  maxElementsInMemory="10000"  eternal="false"  timeToIdleSeconds="120"  timeToLiveSeconds="120"  overflowToDisk="false"  diskPersistent="false"  diskExpiryThreadIntervalSeconds="120"  memoryStoreEvictionPolicy="LRU"/>   <cache name="shiro-activeSessionCache"  eternal="false"  maxElementsInMemory="10000"  overflowToDisk="false"  timeToIdleSeconds="0"  timeToLiveSeconds="0"  statistics="true"/>  <cache name="authorizationCache"  maxElementsInMemory="2000"  eternal="false"  timeToIdleSeconds="3600"  timeToLiveSeconds="0"  overflowToDisk="false"  statistics="true"/>  <cache name="authenticationCache"  maxElementsInMemory="2000"  eternal="false"  timeToIdleSeconds="3600"  timeToLiveSeconds="0"  overflowToDisk="false"  statistics="true"/>  </ehcache> |

## 先在com.majiaxueyuan.realm目录下建立一个类UserRealm做登录权限控制，如下所示：

|  |
| --- |
| package com.wjxmp.realm;  import org.apache.shiro.authc.AuthenticationException;  import org.apache.shiro.authc.AuthenticationInfo;  import org.apache.shiro.authc.AuthenticationToken;  import org.apache.shiro.authz.AuthorizationInfo;  import org.apache.shiro.realm.AuthorizingRealm;  import org.apache.shiro.subject.PrincipalCollection;  public class UserRealm extends AuthorizingRealm {  // 授权：控制角色权限，控制权限  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {  return null;  }  // 认证：控制登录  @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authenticationToken) throws AuthenticationException {  return null;  }  } |

然后写ShiroConfiguration配置类：然后新建一个Shiro的配置文件在com.majiaxueyuan.config包下并注解@Configuration，这个配置只不过把最初的spring-shiro.xml形式的配置文件转成了java文件，这个大家自己拿下去看下就明白意思了，内容和xml形式完全一致

|  |
| --- |
| package com.wjxmp.config;  import com.wjxmp.realm.UserRealm;  import org.apache.shiro.cache.ehcache.EhCacheManager;  import org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator;  import org.apache.shiro.session.mgt.eis.MemorySessionDAO;  import org.apache.shiro.spring.LifecycleBeanPostProcessor;  import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;  import org.apache.shiro.spring.web.ShiroFilterFactoryBean;  import org.apache.shiro.web.mgt.DefaultWebSecurityManager;  import org.apache.shiro.web.servlet.SimpleCookie;  import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;  import org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import javax.servlet.Filter;  import java.util.HashMap;  import java.util.LinkedHashMap;  import java.util.Map;  @Configuration  public class ShiroConfiguration {  /\*\*  \* ShiroFilterFactoryBean 处理拦截资源文件问题。  \* 注意：单独一个ShiroFilterFactoryBean配置是会报错的，因为在  \* 初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager  \* Filter Chain定义说明 1、一个URL可以配置多个Filter，使用逗号分隔 2、当设置多个过滤器时，全部验证通过，才视为通过  \* 3、部分过滤器可指定参数，如perms，roles  \*/  @Bean  public ShiroFilterFactoryBean shirFilter(org.apache.shiro.mgt.SecurityManager securityManager) {  ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();  // 必须设置 SecurityManager  shiroFilterFactoryBean.setSecurityManager(securityManager);  // 拦截器  Map<String, String> filterChainDefinitionMap = new LinkedHashMap<String, String>();  //配置静态资源允许访问，语句后面必须用anon，表示所有url都可以匿名访问，直接访问  filterChainDefinitionMap.put("/user/login","anon"); //这个是login配置页面，是登录页面的跳转  filterChainDefinitionMap.put("/user/loginAction","anon"); //这个是实际登录的跳转  filterChainDefinitionMap.put("/getuser","anon");  filterChainDefinitionMap.put("/web/index","anon");  filterChainDefinitionMap.put("/js/\*\*","anon");  filterChainDefinitionMap.put("/css/\*\*","anon");  filterChainDefinitionMap.put("/index","anon");  //配置退出过滤器  filterChainDefinitionMap.put("/login/logout", "logout");  //登录后要跳转的链接  //shiroFilterFactoryBean.setSuccessUrl("/Welcome");//如果在jsp中实现了成功登陆后的跳转 是不是这块也可以不进行配置  // <!—authc：所有url都必须认证通过才可以访问；anon：所有url都可以匿名访问，直接访问-->  filterChainDefinitionMap.put("/\*\*", "authc");  // 如果不设置默认页面，当未登录去访问其他链接，则会自动寻找Web工程根目录下的"/login.jsp"页面  shiroFilterFactoryBean.setLoginUrl("/user/login");  // 未授权界面，就是没有权限的话，会跳转到下面的403页面;  shiroFilterFactoryBean.setUnauthorizedUrl("/403");  Map<String, Filter> filters=new HashMap<String,Filter>();  shiroFilterFactoryBean.setFilters(filters);  shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);  return shiroFilterFactoryBean;  }  // 该方法是为了拿到缓存的配置文件  @Bean  public EhCacheManager getEhCacheManager() {  EhCacheManager em = new EhCacheManager();  em.setCacheManagerConfigFile("classpath:ehcache-shiro.xml");  return em;  }  // 开启Controller中的shiro注解  @Bean  public DefaultAdvisorAutoProxyCreator getDefaultAdvisorAutoProxyCreator() {  DefaultAdvisorAutoProxyCreator daap = new DefaultAdvisorAutoProxyCreator();  daap.setProxyTargetClass(true);  return daap;  }  /\*\*  \* 配置org.apache.shiro.web.session.mgt.DefaultWebSessionManager  \* @return  \*/  @Bean  public DefaultWebSessionManager getDefaultWebSessionManager(){  DefaultWebSessionManager defaultWebSessionManager=new DefaultWebSessionManager();  defaultWebSessionManager.setSessionDAO(getMemorySessionDAO());  defaultWebSessionManager.setGlobalSessionTimeout(4200000);  defaultWebSessionManager.setSessionValidationSchedulerEnabled(true);  defaultWebSessionManager.setSessionIdCookieEnabled(true);  defaultWebSessionManager.setSessionIdCookie(getSimpleCookie());  return defaultWebSessionManager;  }  /\*\*  \* 配置org.apache.shiro.session.mgt.eis.MemorySessionDAO  \* @return  \*/  @Bean  public MemorySessionDAO getMemorySessionDAO(){  MemorySessionDAO memorySessionDAO=new MemorySessionDAO();  memorySessionDAO.setSessionIdGenerator(javaUuidSessionIdGenerator());  return memorySessionDAO;  }  @Bean  public JavaUuidSessionIdGenerator javaUuidSessionIdGenerator(){  return new JavaUuidSessionIdGenerator();  }  /\*\*  \* session自定义cookie名  \* @return  \*/  @Bean  public SimpleCookie getSimpleCookie(){  SimpleCookie simpleCookie=new SimpleCookie();  simpleCookie.setName("security.session.id");  simpleCookie.setPath("/");  return simpleCookie;  }  @Bean  public LifecycleBeanPostProcessor getLifecycleBeanPostProcessor(){  return new LifecycleBeanPostProcessor();  }  @Bean(name = "securityManager")  public DefaultWebSecurityManager getDefaultWebSecurityManager(UserRealm userRealm) {  DefaultWebSecurityManager dwsm = new DefaultWebSecurityManager();  dwsm.setRealm(userRealm);  //  <!-- 用户授权/认证信息Cache, 采用EhCache 缓存 -->  dwsm.setCacheManager(getEhCacheManager());  dwsm.setSessionManager(getDefaultWebSessionManager());  return dwsm;  }  @Bean  public UserRealm userRealm(EhCacheManager cacheManager) {  UserRealm userRealm = new UserRealm();  userRealm.setCacheManager(cacheManager);  return userRealm;  }  /\*\*  \* 开启shrio注解支持  \* @param userRealm  \* @return  \*/  @Bean  public AuthorizationAttributeSourceAdvisor getAuthorizationAttributeSourceAdvisor(UserRealm userRealm){  AuthorizationAttributeSourceAdvisor aasa=new AuthorizationAttributeSourceAdvisor();  aasa.setSecurityManager(getDefaultWebSecurityManager(userRealm));  return aasa;  }  } |

新建一个实体类User、Role、Permission。User 内容如下：

|  |
| --- |
| package com.wjxmp.entity;  import com.baomidou.mybatisplus.annotation.IdType;  import com.baomidou.mybatisplus.extension.activerecord.Model;  import com.baomidou.mybatisplus.annotation.TableId;  import java.io.Serializable;  import lombok.Data;  import lombok.EqualsAndHashCode;  import lombok.experimental.Accessors;  @Data  @EqualsAndHashCode(callSuper = false)  @Accessors(chain = true)  public class User extends Model<User> {  private static final long serialVersionUID = 1L;  @TableId(value = "id", type = IdType.AUTO)  private Integer id;  private String username;  private String sex;  private Integer roleid;  @Override  protected Serializable pkVal() {  return this.id;  }  } |

Role内容如下：

|  |
| --- |
| package com.wjxmp.entity;  import com.baomidou.mybatisplus.annotation.IdType;  import com.baomidou.mybatisplus.extension.activerecord.Model;  import com.baomidou.mybatisplus.annotation.TableId;  import java.io.Serializable;  import lombok.Data;  import lombok.EqualsAndHashCode;  import lombok.experimental.Accessors;  @Data  @EqualsAndHashCode(callSuper = false)  @Accessors(chain = true)  public class Role extends Model<Role> {  private static final long serialVersionUID = 1L;  @TableId(value = "id", type = IdType.AUTO)  private Integer id;  private String rolename;  @Override  protected Serializable pkVal() {  return this.id;  }  } |

Permission内容如下

|  |
| --- |
| package com.wjxmp.entity;  import com.baomidou.mybatisplus.annotation.IdType;  import com.baomidou.mybatisplus.extension.activerecord.Model;  import com.baomidou.mybatisplus.annotation.TableId;  import java.io.Serializable;  import lombok.Data;  import lombok.EqualsAndHashCode;  import lombok.experimental.Accessors;  @Data  @EqualsAndHashCode(callSuper = false)  @Accessors(chain = true)  public class Permission extends Model<Permission> {  private static final long serialVersionUID = 1L;  @TableId(value = "id", type = IdType.AUTO)  private Integer id;  private String pername;  private Integer roleid;  @Override  protected Serializable pkVal() {  return this.id;  }  } |

新建三个mapper接口UserMapper、RoleMapper、PermissionMapper

|  |
| --- |
| package com.wjxmp.mapper;  import com.wjxmp.entity.User;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  @Mapper  public interface UserMapper extends BaseMapper<User> {  } |

|  |
| --- |
| package com.wjxmp.mapper;  import com.wjxmp.entity.Role;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  @Mapper  public interface RoleMapper extends BaseMapper<Role> {  } |

|  |
| --- |
| package com.wjxmp.mapper;  import com.wjxmp.entity.Permission;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  @Mapper  public interface PermissionMapper extends BaseMapper<Permission> {  } |

新建三个controller接口UserController、RoleController、PermissionController

|  |
| --- |
| package com.wjxmp.controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/user")  public class UserController {  } |

|  |
| --- |
| package com.wjxmp.controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/role")  public class RoleController {  } |

|  |
| --- |
| package com.wjxmp.controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/permission")  public class PermissionController {  } |

新建三个service接口IUserService、IRoleService、IPermissionService

|  |
| --- |
| package com.wjxmp.service;  import com.wjxmp.entity.User;  import com.baomidou.mybatisplus.extension.service.IService;  public interface IUserService extends IService<User> {  } |

|  |
| --- |
| package com.wjxmp.service;  import com.wjxmp.entity.Role;  import com.baomidou.mybatisplus.extension.service.IService;  public interface IRoleService extends IService<Role> {  } |

|  |
| --- |
| package com.wjxmp.service;  import com.wjxmp.entity.Permission;  import com.baomidou.mybatisplus.extension.service.IService;  public interface IPermissionService extends IService<Permission> {  } |

新建三个service接口的实现类UserServiceImpl、RoleServiceImpl、PermissionServiceImpl

|  |
| --- |
| package com.wjxmp.service.impl;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IUserService;  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  import org.springframework.stereotype.Service;  @Service  public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements IUserService {  } |

|  |
| --- |
| package com.wjxmp.service.impl;  import com.wjxmp.entity.Role;  import com.wjxmp.mapper.RoleMapper;  import com.wjxmp.service.IRoleService;  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  import org.springframework.stereotype.Service;  @Service  public class RoleServiceImpl extends ServiceImpl<RoleMapper, Role> implements IRoleService {  } |

|  |
| --- |
| package com.wjxmp.service.impl;  import com.wjxmp.entity.Permission;  import com.wjxmp.mapper.PermissionMapper;  import com.wjxmp.service.IPermissionService;  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  import org.springframework.stereotype.Service;  @Service  public class PermissionServiceImpl extends ServiceImpl<PermissionMapper, Permission> implements IPermissionService {  } |

在web目录下新建一个WebUserController类，里面新增一个方法login和logout

|  |
| --- |
| package com.wjxmp.web;  import com.baomidou.mybatisplus.core.conditions.Wrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IPermissionService;  import com.wjxmp.service.IRoleService;  import com.wjxmp.service.IUserService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/user")  public class WebUserController {  @Autowired  private IUserService iUserService;  @Autowired  private IRoleService iRoleService;  @Autowired  private IPermissionService iPermissionService;  @Autowired  private UserMapper userMapper;  @RequestMapping("/login")  public String login(){  return "loginPage" ;  }  @RequestMapping("/logout")  public String logout(String username){  Subject subject = SecurityUtils.getSubject();  subject.logout();  return "已注销";  }  } |

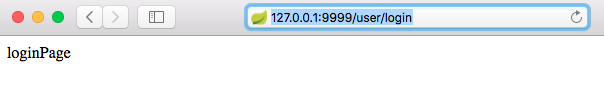
然后在Shiro的配置类ShiroConfiguration中添加login和logout方法的访问方式(其实已经添加了)

|  |
| --- |
| //配置静态资源允许访问，语句后面必须用anon，表示所有url都可以匿名访问，直接访问  filterChainDefinitionMap.put("/user/login","anon"); //这个是login配置页面，是登录页面的跳转  //配置退出过滤器  filterChainDefinitionMap.put("/login/logout", "logout");  // 如果不设置默认页面，当未登录去访问其他链接，则会自动寻找Web工程根目录下的"/login.jsp"页面  shiroFilterFactoryBean.setLoginUrl("/user/login"); |

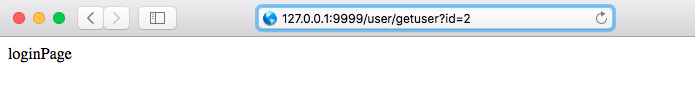
建立一个APP启动类：

|  |
| --- |
| **package** com.wjxsb;  **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.SpringBootApplication; @SpringBootApplication **public class** App {  **public static void** main(String[] args){  SpringApplication.*run*(App.**class**,args);  } } |

启动启动类，然后在浏览器上执行http://127.0.0.1:9999/user/login，结果如下



其他未设置可以匿名访问的方法，在执行时会跳转到loginPage页面。



在web目录下的类WebUserController 中新增一个loginAction方法

|  |
| --- |
| package com.wjxmp.web;  import com.baomidou.mybatisplus.core.conditions.Wrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IPermissionService;  import com.wjxmp.service.IRoleService;  import com.wjxmp.service.IUserService;  import org.apache.shiro.SecurityUtils;  import org.apache.shiro.authc.AuthenticationToken;  import org.apache.shiro.authc.UsernamePasswordToken;  import org.apache.shiro.subject.Subject;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/user")  public class WebUserController {  @Autowired  private IUserService iUserService;  @Autowired  private IRoleService iRoleService;  @Autowired  private IPermissionService iPermissionService;  @Autowired  private UserMapper userMapper;  @RequestMapping("/login")  public String login(){  return "loginPage" ;  }  @RequestMapping("/logout")  public String logout(String username){  Subject subject = SecurityUtils.getSubject();  subject.logout();  return "已注销";  }  @RequestMapping("/loginAction")  public String loginAction(String username){  Subject subject=SecurityUtils.getSubject();  AuthenticationToken token = new UsernamePasswordToken(username,"");  // 上面的语句也可以用右面的语句实现 UsernamePasswordToken token1 = new UsernamePasswordToken(username,"") ;  try {  // 如果执行subject.login不抛出异常，则证明登录成功  subject.login(token);  return "sucess !" ;  } catch (Exception e){  // 如果有异常，则证明登录账号或密码错误，此时需要在UserRealm类中实现控制登录代码  return "Failed" ;  }  }  } |

在Shiro的配置类ShiroConfiguration中添加loginAction方法的访问方式

|  |
| --- |
| filterChainDefinitionMap.put("/user/login","anon"); // 这个是login配置页面，是登录页面的跳转  filterChainDefinitionMap.put("/user/loginAction","anon"); // 这个是实际登录的跳转 |

在UserRealm中的认证方法doGetAuthenticationInfo中添加方法体如下

|  |
| --- |
| package com.wjxmp.realm;  import com.wjxmp.entity.User;  import com.wjxmp.service.IUserService;  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.realm.AuthorizingRealm;  import org.apache.shiro.subject.PrincipalCollection;  import org.springframework.beans.factory.annotation.Autowired;  public class UserRealm extends AuthorizingRealm {  @Autowired  private IUserService userService;  // 授权：控制角色权限，控制权限  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {  return null;  }  // 认证：控制登录  @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken at) throws AuthenticationException {  AuthenticationInfo info;  String username = (String) at.getPrincipal(); // 实际上就是获得username  User user = userService.selectUserByUsername(username);  if(user == null){  throw new UnknownAccountException(); //没找到账号  }  if (user!=null){  // 这是方法的原型 SimpleAuthenticationInfo(principal,credentials,realmName) principal字段传user对象或者是username都行，credentials这里是指从数据库中获取的password，realmName即当前realm的名称  info = new SimpleAuthenticationInfo(user.getUsername(),"","UserRealm");  return info;  }  return null;  }  } |

在接口 IUserService中添加方法如下

|  |
| --- |
| package com.wjxmp.service;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.baomidou.mybatisplus.extension.service.IService;  public interface IUserService extends IService<User> {  User selectUserByUsername(String username);  } |

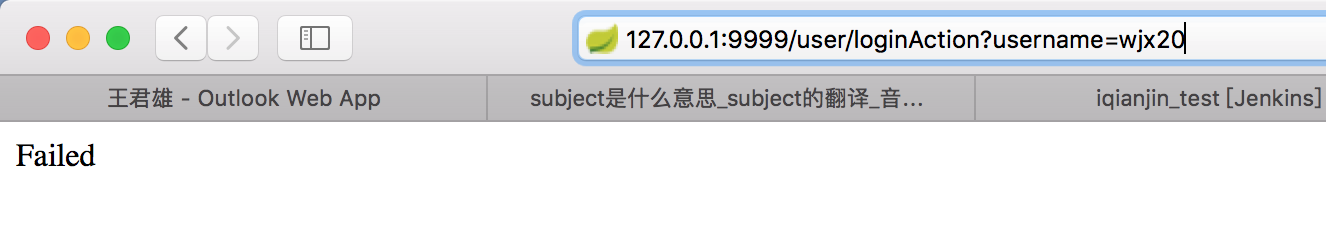
在接口 IUserService的实现中添加实现方法如下

|  |
| --- |
| package com.wjxmp.service.impl;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IUserService;  import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import java.util.List;  @Service  public class UserServiceImpl extends ServiceImpl<UserMapper, User> implements IUserService {  @Autowired  private UserMapper userMapper;  @Override  public User selectUserByUsername(String username) {  QueryWrapper<User> wrapper = new QueryWrapper<User>();  wrapper.eq("username",username);  List<User> selectList = userMapper.selectList(wrapper);  if (selectList.size()>0){  return selectList.get(0);  }  return null;  }  } |

启动启动类，先在浏览器上执行http://127.0.0.1:9999/user/getuser?id=2，因为没有登录，所以会跳到loginPage页面



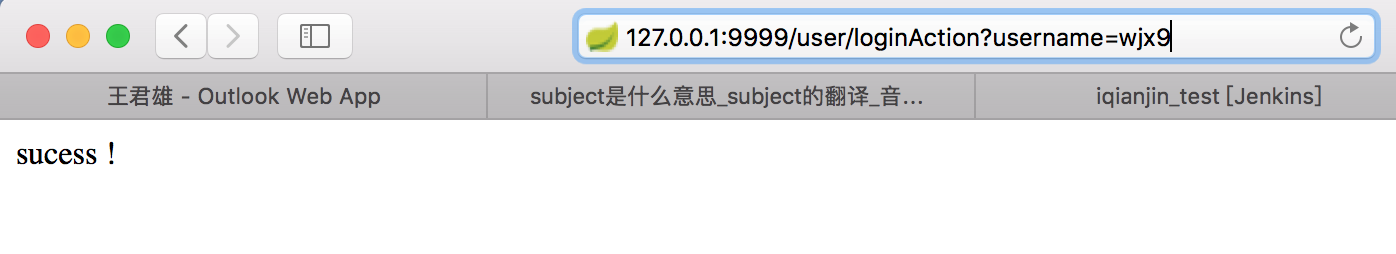
然后再在浏览器上执行http://127.0.0.1:9999/user/login Action?username=wjx20，因为数据库中没有wjx20，所以会执行失败，相当于没有登录。结果如下



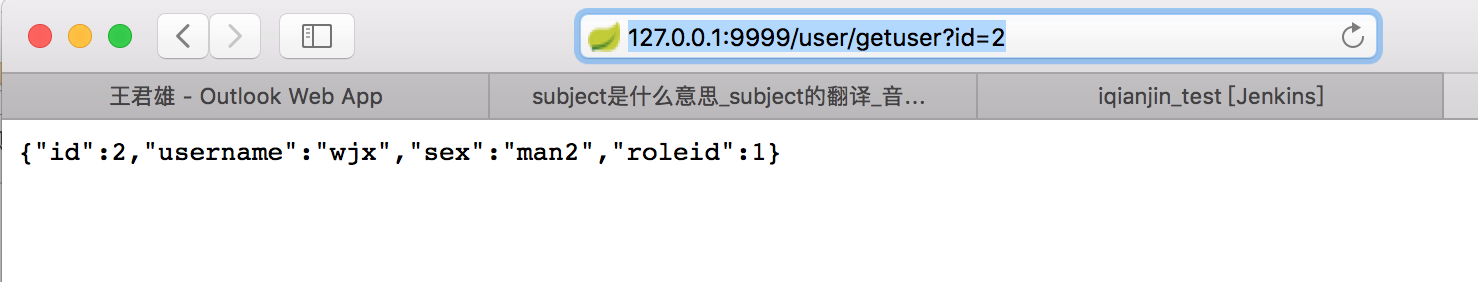
在浏览器上执行http://127.0.0.1:9999/user/getuser?id=2，因为没有登录，所以会跳到loginPage页面



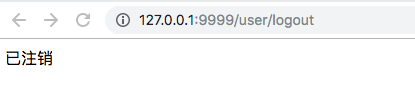
然后再在浏览器上执行http://127.0.0.1:9999/user/login Action?username=wjx9，因为数据库中有wjx9则会执行成功，相当于登录成功。结果如下



此时再次执行http://127.0.0.1:9999/user/getuser?id=2，因为前面已经登录，所以会正常执行



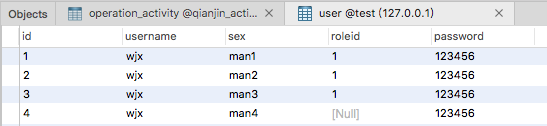
再在浏览器上执行http://127.0.0.1:9999/user/logout，相当于登录注销。结果如下



在浏览器上执行http://127.0.0.1:9999/user/getuser?id=2，因为没有登录，所以会跳到loginPage页面



但是在实际开发中，登录也需要验证密码，所以需要在user表先添加一个password字段。



然后需要在实体类User中添加密码信息

|  |
| --- |
| package com.wjxmp.entity;  import com.baomidou.mybatisplus.annotation.IdType;  import com.baomidou.mybatisplus.extension.activerecord.Model;  import com.baomidou.mybatisplus.annotation.TableId;  import java.io.Serializable;  import lombok.Data;  import lombok.EqualsAndHashCode;  import lombok.experimental.Accessors;  /\*\*  \* <p>  \* 用户表  \* </p>  \*  \* @author wjx  \* @since 2019-09-25  \*/  @Data  @EqualsAndHashCode(callSuper = false)  @Accessors(chain = true)  public class User extends Model<User> {  private static final long serialVersionUID = 1L;  @TableId(value = "id", type = IdType.AUTO)  private Integer id;  private String username;  private String sex;  private Integer roleid;    private String password;  @Override  public String toString() {  return "User{" +"id=" + id+" , username='" + username+'\''+", sex='" + sex+'\'' +", roleid=" + roleid+'}' ;  }  @Override  protected Serializable pkVal() {  return this.id;  }  // public static void main(String[] args) {  // System.out.println(" \\ \" 是 "+"\'"); // 输出 \ " 是 '  // System.out.println(" \\ \' 是 "+'\''); // 输出 \ ' 是 '  // System.out.println(" \"}\" 是 "+"}"); // 输出 "}" 是 }  // System.out.println(" \'}\' 是 "+'}'); // 输出 '}' 是 }  // System.out.println(" ')' 是 "+')'); // 输出 '}' 是 }  // }  } |

然后需要在loginAction方法中加上密码信息

|  |
| --- |
| package com.wjxmp.web;  import com.baomidou.mybatisplus.core.conditions.Wrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IPermissionService;  import com.wjxmp.service.IRoleService;  import com.wjxmp.service.IUserService;  import org.apache.shiro.SecurityUtils;  import org.apache.shiro.authc.AuthenticationToken;  import org.apache.shiro.authc.UsernamePasswordToken;  import org.apache.shiro.subject.Subject;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/user")  public class WebUserController {  @Autowired  private IUserService iUserService;  @Autowired  private IRoleService iRoleService;  @Autowired  private IPermissionService iPermissionService;  @Autowired  private UserMapper userMapper;  @RequestMapping("/login")  public String login(){  return "loginPage" ;  }  @RequestMapping("/logout")  public String logout(String username){  Subject subject = SecurityUtils.getSubject();  subject.logout();  return "已注销";  }  @RequestMapping("/loginAction")  public String loginAction(String username,String password){  Subject subject=SecurityUtils.getSubject();  String md5Hash = new Md5Hash(password,"123").toString(); // 原型是 Md5Hash(Object source, Object salt)  AuthenticationToken token = new UsernamePasswordToken(username,md5Hash);  // 上面的语句也可以用右面的语句实现 UsernamePasswordToken token1 = new UsernamePasswordToken(username,"") ;  try {  // 如果执行subject.login不抛出异常，则证明登录成功  subject.login(token);  return "sucess !" ;  } catch (Exception e){  // 如果有异常，则证明登录账号或密码错误，此时需要在UserRealm类中实现控制登录代码  return "Failed" ;  }  }  } |

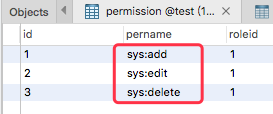
然后在UserRealm的方法doGetAuthenticationInfo中添加密码的信息

|  |
| --- |
| package com.wjxmp.realm;  import com.wjxmp.entity.User;  import com.wjxmp.service.IUserService;  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.realm.AuthorizingRealm;  import org.apache.shiro.subject.PrincipalCollection;  import org.springframework.beans.factory.annotation.Autowired;  public class UserRealm extends AuthorizingRealm {  @Autowired  private IUserService userService;  // 授权：控制角色权限，控制权限  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principalCollection) {  return null;  }  // 认证：控制登录  // @Override  // protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken at) throws AuthenticationException {  // AuthenticationInfo info;  // String username = (String) at.getPrincipal(); // 实际上就是获得username  // User user = userService.selectUserByUsername(username);  // if (user!=null){  // info = new SimpleAuthenticationInfo(username,"","anything"); // 这是方法的原型 SimpleAuthenticationInfo(principal,credentials,realmName) realmName写什么都行，没有影响  // return info;  // }  // return null;  // }  @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken at) throws AuthenticationException {  AuthenticationInfo info;  String username = (String) at.getPrincipal(); // 实际上就是获得username  User user = userService.selectUserByUsername(username);  if (user!=null){  // 这是方法的原型 SimpleAuthenticationInfo(principal,credentials,realmName) principal字段传user对象或者是username都行，credentials这里是指从数据库中获取的password，realmName即当前realm的名称  // info = new SimpleAuthenticationInfo(user.getUsername(),"","UserRealm");  info = new SimpleAuthenticationInfo(user.getUsername(),user.getPassword(),"UserRealm");  return info;  }  return null;  }  } |

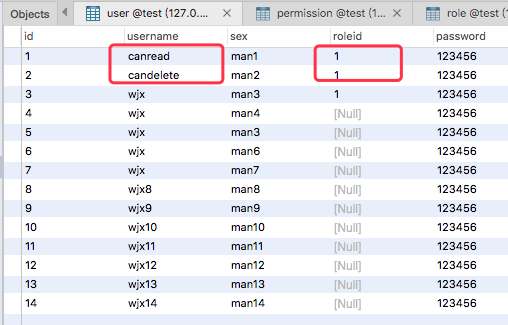
执行时，loginAction方法中的AuthenticationToken信息传递到UserRealm中，然后doGetAuthenticationInfo方法通过userService.selectUserByUsername(username)获取user信息，然后通过user获取用户密码。然后开始对比。这块对比逻辑是先对比username，但是username肯定是相等的，所以真正对比的是password。从这里传入的password（这里是从数据库获取的已加密的密码）和token（filter中登录时生成的）中的password(比如经过加密的md5Hash)做对比，如果相同就允许登录，不相同就抛出异常。如果验证成功，最终这里返回的信息authenticationInfo 的值与传入的第一个字段的值相同。

登录之后，我们进行权限的设置。

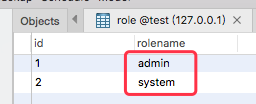
先在permission表增加一条记录



然后在user表修改记录



在role表增加一条记录



先在类UserRealm中实现授权方法如下所示

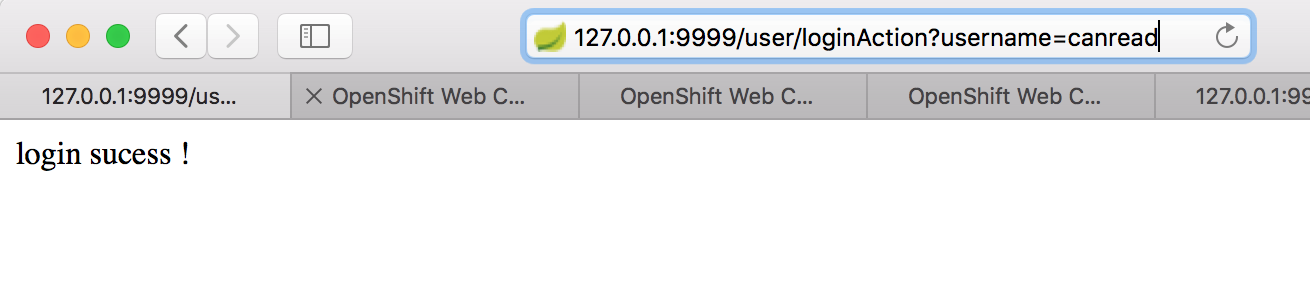
|  |
| --- |
| package com.wjxmp.realm;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.wjxmp.entity.Permission;  import com.wjxmp.entity.Role;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.PermissionMapper;  import com.wjxmp.mapper.RoleMapper;  import com.wjxmp.service.IUserService;  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.authc.UnknownAccountException;  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.ArrayList;  import java.util.List;  public class UserRealm extends AuthorizingRealm {  @Autowired  private IUserService userService;  @Autowired  private RoleMapper roleMapper;  @Autowired  private PermissionMapper permissionMapper;  // 授权：控制角色权限，控制权限  @Override  protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection pc) {  SimpleAuthorizationInfo info = new SimpleAuthorizationInfo() ;  String username = (String) pc.getPrimaryPrincipal();  // 将当前用户的角色和权限复制进来  User user = userService.selectUserByUsername(username);  Integer roleid = user.getRoleid();  Role role = roleMapper.selectById(roleid) ;  info.addRole(role.getRolename());  QueryWrapper<Permission> wrapper = new QueryWrapper<>();  wrapper.eq("roleid",role.getId()) ;  List<Permission> selectList = permissionMapper.selectList(wrapper);  List<String> perlist = new ArrayList<>();  selectList.forEach(per -> {  perlist.add(per.getPername());  } );  info.addStringPermissions(perlist);  return info;  }  // 认证：控制登录  // @Override  // protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken at) throws AuthenticationException {  // AuthenticationInfo info;  // String username = (String) at.getPrincipal(); // 实际上就是获得username  // User user = userService.selectUserByUsername(username);  // if (user!=null){  // info = new SimpleAuthenticationInfo(username,"","anything"); // 这是方法的原型 SimpleAuthenticationInfo(principal,credentials,realmName) realmName写什么都行，没有影响  // return info;  // }  // return null;  // }  @Override  protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken at) throws AuthenticationException {  AuthenticationInfo info;  String username = (String) at.getPrincipal(); // 实际上就是获得username  User user = userService.selectUserByUsername(username);  if(user == null){  throw new UnknownAccountException(); //没找到账号  }  if (user!=null){  // 这是方法的原型 SimpleAuthenticationInfo(principal,credentials,realmName) principal字段传user对象或者是username都行，credentials这里是指从数据库中获取的password，realmName即当前realm的名称  // info = new SimpleAuthenticationInfo(user.getUsername(),user.getPassword(),"UserRealm");  info = new SimpleAuthenticationInfo(user.getUsername(),"","UserRealm");  return info;  }  return null;  }  } |

然后在controller类中方法加上角色和权限的注解，如下所示

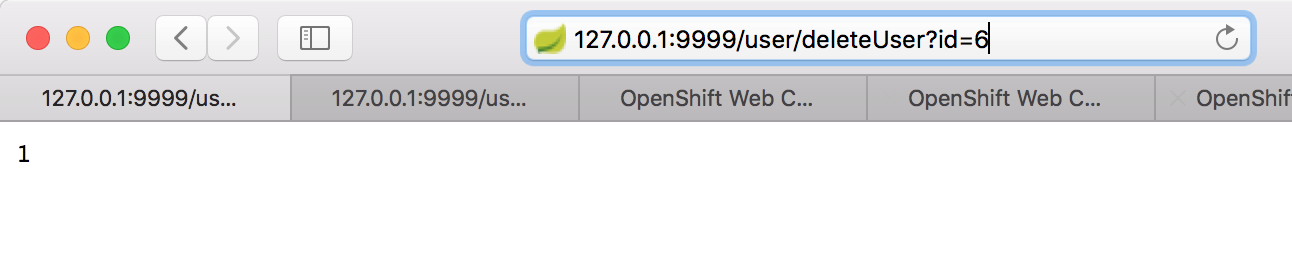
|  |
| --- |
| package com.wjxmp.web;  import com.baomidou.mybatisplus.core.conditions.Wrapper;  import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;  import com.baomidou.mybatisplus.core.mapper.BaseMapper;  import com.baomidou.mybatisplus.core.metadata.IPage;  import com.baomidou.mybatisplus.extension.plugins.pagination.Page;  import com.wjxmp.entity.User;  import com.wjxmp.mapper.UserMapper;  import com.wjxmp.service.IPermissionService;  import com.wjxmp.service.IRoleService;  import com.wjxmp.service.IUserService;  import org.apache.shiro.SecurityUtils;  import org.apache.shiro.authc.AuthenticationToken;  import org.apache.shiro.authc.UsernamePasswordToken;  import org.apache.shiro.authz.annotation.RequiresPermissions;  import org.apache.shiro.authz.annotation.RequiresRoles;  import org.apache.shiro.crypto.hash.Md5Hash;  import org.apache.shiro.subject.Subject;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  @RestController  @RequestMapping("/user")  public class WebUserController {  @Autowired  private IUserService iUserService;  @Autowired  private IRoleService iRoleService;  @Autowired  private IPermissionService iPermissionService;  @Autowired  private UserMapper userMapper;  @RequestMapping("/login")  public String login(){  return "loginPage" ;  }  // @RequestMapping("/logout")  // public String logout(User loginUser){  // Subject subject = SecurityUtils.getSubject();  // subject.logout();  // return "已注销";  // }  @RequestMapping("/logout")  public String logout(String username){  Subject subject = SecurityUtils.getSubject();  subject.logout();  return "login 已注销";  }  @RequestMapping("/loginAction")  public String loginAction(String username){  Subject subject=SecurityUtils.getSubject();  AuthenticationToken token = new UsernamePasswordToken(username,"");  // 上面的语句也可以用右面的语句实现 UsernamePasswordToken token1 = new UsernamePasswordToken(username,"") ;  try {  // 如果执行subject.login不抛出异常，则证明登录成功  subject.login(token);  return "login sucess !" ;  } catch (Exception e){  // 如果有异常，则证明登录账号或密码错误，此时需要在UserRealm类中实现控制登录代码  return "login Failed" ;  }  }  // @RequestMapping("/loginAction")  // public String loginAction(String username,String password){  // Subject subject=SecurityUtils.getSubject();  // AuthenticationToken token = new UsernamePasswordToken(username,password);  //// String md5Hash = new Md5Hash(password,"123").toString(); // 原型是 Md5Hash(Object source, Object salt)  //// AuthenticationToken token = new UsernamePasswordToken(username,md5Hash);  // // 上面的语句也可以用右面的语句实现 UsernamePasswordToken token1 = new UsernamePasswordToken(username,"") ;  // try {  // // 如果执行subject.login不抛出异常，则证明登录成功  // subject.login(token);  // return "login sucess !" ;  // } catch (Exception e){  // // 如果有异常，则证明登录账号或密码错误，此时需要在UserRealm类中实现控制登录代码  // return "login Failed" ;  // }  // }  // @RequestMapping("/getuser")  @RequestMapping(value = "getusers",method = RequestMethod.GET)  public Object getUser(Integer pgno,Integer pgsize){  // QueryWrapper<User> wrapper = new QueryWrapper<User>();  // wrapper.eq("username","wjx");  // return userMapper.selectList(wrapper);  Page<User> page = new Page<User>(pgno, pgsize);  QueryWrapper<User> wrapper = new QueryWrapper<User>();  wrapper.eq("username","wjx");  IPage<User> iPage =iUserService.getAllUsers(page, wrapper);  return iPage;  }  **@RequiresRoles("admin")**  @RequestMapping("getuser")  public Object getUser(Integer id){  return userMapper.selectById(id);  }  **@RequiresPermissions("sys:delete")**  @RequestMapping("/deleteuser")  public Object deleteUser(Integer id){  return userMapper.deleteById(id);  }  @RequestMapping("getexc")  public String getSum(){  int a=2/0;  return "ss";  }  @RequestMapping("getSum")  public Integer getSum(Integer a,Integer b){  return a+b ;  }  } |

然后启动，用用户candelete登录。

在浏览器上登录 http://127.0.0.1:9999/user/loginAction?username=canread



然后删除用户 http://127.0.0.1:9999/user/deleteUser?id=6



备注：在注释掉@RequiresPermissions("sys:delete")后，应该不能删除用户，但是测试的时候发现是可以。需要后续查找原因。

## SpringBoot使用AOP统一处理请求日志

AOP为Aspect Oriented Programming的缩写，意为：面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。在日常开发当中经常用来记录日志，方法跟踪、事务，权限等。

切面方法说明：

@Aspect -- 作用是把当前类标识为一个切面供容器读取

@Pointcut -- (切入点):就是带有通知的连接点，在程序中主要体现为书写切入点表达式

@Before -- 标识一个前置增强方法，相当于BeforeAdvice的功能

@AfterReturning -- 后置增强，相当于AfterReturningAdvice，方法退出时执行

@AfterThrowing -- 异常抛出增强，相当于ThrowsAdvice

@After -- final增强，不管是抛出异常或者正常退出都会执行

@Around -- 环绕增强，相当于MethodInterceptor

今天我们使用AOP去对我们的所有请求进行一个统一处理。

首先在pom.xml中引入我们需要的aop的jar包

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjrt</artifactId>  <version>1.7.2</version>  </dependency>  <dependency>  <groupId>org.aspectj</groupId>  <artifactId>aspectjweaver</artifactId>  <version>1.9.4</version>  </dependency> |

然后我们新建一个Aspect去对我们的所有Controller进行一个日志拦截。在SPringBoot中我们使用AOP也是很简单的，只需要在类上加上一个@Aspect的注解就好了，然后通过@Component注解到我们的Spring容器中去。

我们新建一个包com.wjxmp.log专门用来放我们的日志记录，代码如下：

|  |
| --- |
| package com.wjxmp.log;  import lombok.extern.slf4j.Slf4j;  import org.aopalliance.intercept.Joinpoint;  import org.aspectj.lang.ProceedingJoinPoint;  import org.aspectj.lang.annotation.\*;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.stereotype.Component;  import org.springframework.web.context.request.RequestContextHolder;  import org.springframework.web.context.request.ServletRequestAttributes;  import javax.servlet.http.HttpServletRequest;  import java.util.Enumeration;  import org.aspectj.lang.JoinPoint;  @Aspect  @Component  //@Slf4j  public class LogAspect {  private Logger log= LoggerFactory.getLogger(getClass());  **// private final** Logger **LOG** = LoggerFactory.*getLogger*(**this**.getClass());  //只针对 com.wjxmp.web.HtmlPageController 类切面  // @Pointcut("execution(public \* com.wjxmp.web.HtmlPageController.\*(..))")  // com.wjxmp.web 包中所有的类的所有方法切面  // @Pointcut("execution(public \* com.wjxmp.web.\*.\*(..))")  //统一切点,对com.wjxmp.web及其子包中所有的类的所有方法切面  @Pointcut("execution(public \* com.wjxmp.web..\*.\*(..))")  public void pointcut() {  log.info(">>>>>>>>>> 执行切入点方法 pointcut() >>>>>>>>>>>>");  }  //前置通知  @Before("pointcut()")  public void beforeMethod(JoinPoint joinPoint) throws Throwable {  log.info(">>>>>>>>>>>> 调用了前置增强方法 @Before(\"pointcut()\") beforeMethod(JoinPoint joinPoint)  >>>>>>>>>>>>");  // 接收到请求，记录请求内容。在Spring API中提供了一个非常便捷的工具类RequestContextHolder，能够在Controller中获取request对象和response对象，使用方法如下  ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();  HttpServletRequest request = attributes.getRequest();  // 记录下请求内容  log.info("URL : " + request.getRequestURL().toString());  log.info("HTTP\_METHOD : " + request.getMethod());  log.info("IP : " + request.getRemoteAddr());  Enumeration<String> enu = request.getParameterNames();  while (enu.hasMoreElements()) {  String name = (String) enu.nextElement();  log.info("name:{},value:{}", name, request.getParameter(name));  }  }  //@After: 后置通知  @After("pointcut()")  public void afterMethod(JoinPoint joinPoint){  log.info(">>>>>>>>>>>> 调用了final增强后置通知 @After(\"pointcut()\") afterMethod(JoinPoint joinPoint)  >>>>>>>>>>>>");  }  //@AfterRunning: 返回通知，rsult为返回内容  @AfterReturning(value = "pointcut()",returning = "result")  public void afterReturningMethod(Object result) throws Throwable {  log.info(">>>>>>>>>>>> 调用了后置增强方法 @AfterReturning(value = \"pointcut()\",returning = \"result\") afterReturningMethod(Object result)  >>>>>>>>>>>>");  // 处理完请求，返回内容  log.info("RESPONSE : " + result);  }  //另一种格式的@AfterRunning: 返回通知，rsult为返回内容  @AfterReturning(value = "pointcut()", returning="result")  public void afterReturningMethod(JoinPoint joinPoint, Object result) {  log.info(">>>>>>>>>>>> 调用了后置增强方法 @AfterReturning(value = \"pointcut()\", returning=\"result\") afterReturningMethod(JoinPoint joinPoint, Object result)  >>>>>>>>>>>>");  System.out.println("获取切入点的方法名称:" + joinPoint.getSignature().getName());  if (result instanceof String) {  result = ((String) result).toUpperCase();  }  System.out.println("目标方法的返回值result:" + result);  System.out.println("执行了后置通知......");  }  //@AfterThrowing: 异常通知  @AfterThrowing(value="pointcut()",throwing="e")  public void afterReturningMethod(JoinPoint joinPoint, Exception e){  log.info(">>>>>>>>>>>> 调用了异常抛出增强方法 @AfterThrowing(value=\"pointcut()\",throwing=\"e\") afterReturningMethod(JoinPoint joinPoint, Exception e) >>>>>>>>>>>>");  }  //@Around：环绕通知  @Around("pointcut()")  public Object Around(ProceedingJoinPoint pjp) throws Throwable {  log.info(">>>>>>>>> 开始执行环绕增强方法 @Around(\"pointcut()\") Around(ProceedingJoinPoint pjp)  >>>>>>>>>");  Object object = pjp.proceed();  log.info(">>>>>>>>> @Around(\"pointcut()\") Around(ProceedingJoinPoint pjp) 执行环绕增强方法之后--返回值：" +object);  return object;  }  } |

然后我们通过浏览器随便请求一次，可以看到控制台输出，这里我们就可以在实际生产环境中存储日志。



根据输出日志我们总结出aop切面的执行步骤：

|  |
| --- |
| 根据上面的切面设置，请求一个方法时：  1、先进行环绕增强方法@Around  2、如果在@Around里面有Object object = proceedingJoinPoint.proceed()，就会进行前置增强方法@Before，然后返回到@Around方法  3、再执行final增强方法@After  4、再执行后置增强方法@AfterReturning  如果有多个@AfterReturning方法，顺序执行 |

## SpringBoot定时任务@Scheduled

代码做定时任务：要么愚蠢一点，开个线程去做，线程里面一直在休眠去做。要不使用一些定时任务的框架去做。

这个到了后面就比较简单了，我们就简单的写一个例子给大家展示。

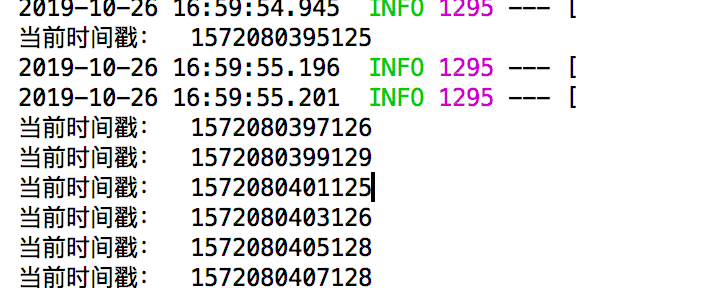
在service中新建一个类TimerTest，代码如下：

|  |
| --- |
| import org.springframework.scheduling.annotation.Scheduled;  import org.springframework.stereotype.Component;  @Component  public class TimerTest {  @Scheduled(fixedRate = 2000) // 2000是以毫秒为单位  public void showTime(){  System.out.println("当前时间戳： "+System.currentTimeMillis());  }  } |

另外我们需要在主函数启动类上开启定时器

|  |
| --- |
| import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.scheduling.annotation.EnableScheduling;  @SpringBootApplication  @EnableScheduling  public class APP {  public static void main(String[] args){  SpringApplication.run(APP.class,args);  }  } |

然后运行主函数，在启动日志可以看到如下信息



## SpringBoot异步调用Async

分布式情况下，定时任务的重复消费该如何进行？可以使用分布式调度平台，例如XXL-JOB 。

这个和定时器差不多，启动加上@EnableAsync ，只需要在我们需要异步的方法上面加上@Async注解。

在service中新建一个类AsyncTest，代码如下：

|  |
| --- |
| import org.springframework.scheduling.annotation.Async;  import org.springframework.stereotype.Component;  @Component  public class AsyncTest {  @Async  public void asyncOut(){  System.out.println("异步方法ID是： "+Thread.currentThread().getId());  }  } |

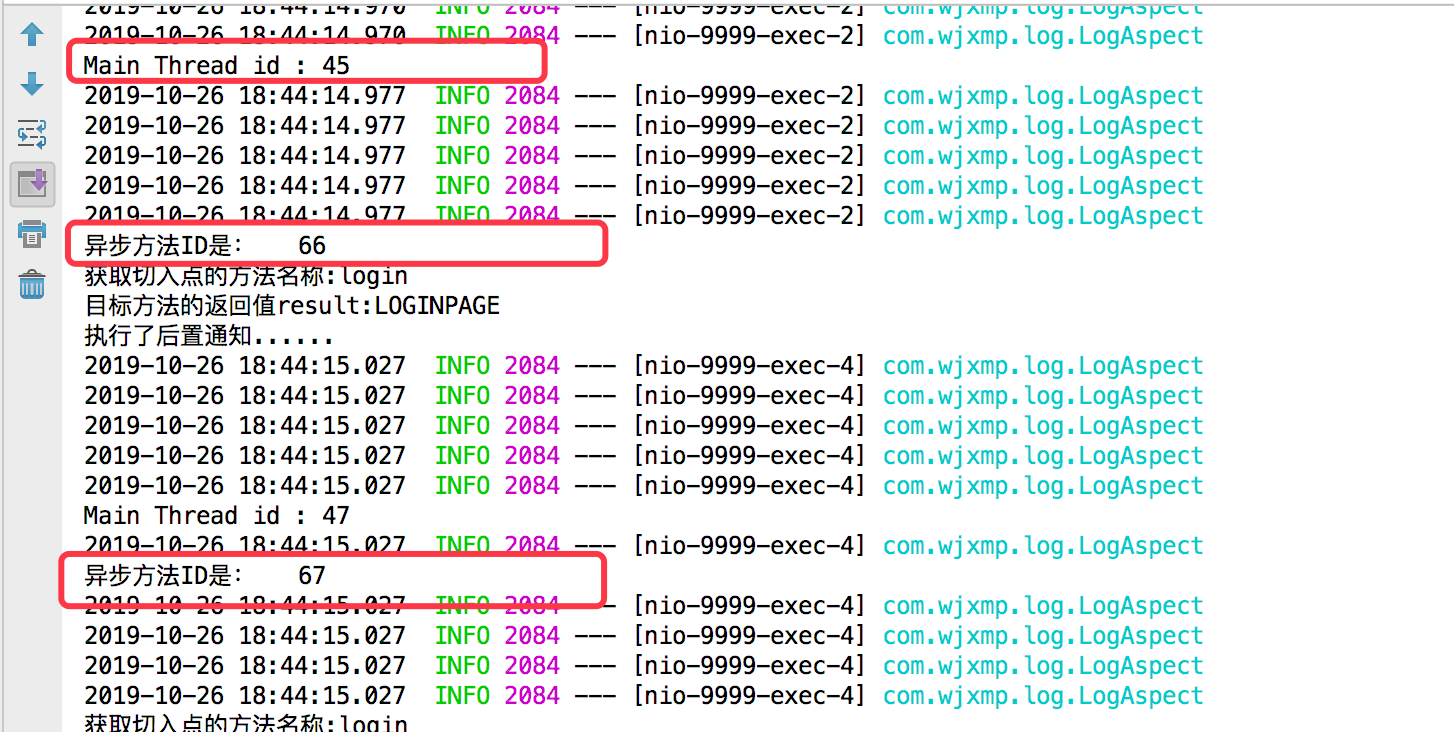
在某个接口方法中加入上面方法的调用

|  |
| --- |
| @Autowired  private AsyncTest asyncTest;  @RequestMapping("/login")  public String login(){  System.out.println("Main Thread id : "+Thread.currentThread().getId());  asyncTest.asyncOut();  return "loginPage" ;  } |

另外我们需要在主函数启动类上开启定时器

|  |
| --- |
| import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.scheduling.annotation.EnableScheduling;  @SpringBootApplication  @EnableAsync  public class APP {  public static void main(String[] args){  SpringApplication.run(APP.class,args);  }  } |

然后运行主函数，在启动日志可以看到如下信息



## SpringBoot自定义参数获取

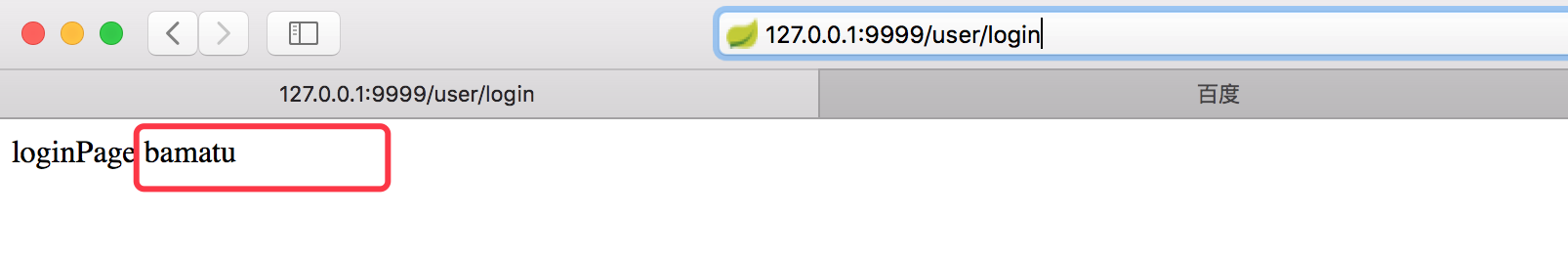
在配置文件中我们可以去在配置文件中自定义一些参数，比如：

|  |
| --- |
| name=bamatu |

我们在代码中获取到我们的这个配置文件

|  |
| --- |
| @Value("${name}")  private String name;  @RequestMapping("/login")  public String login(){  System.out.println("Main Thread id : "+Thread.currentThread().getId());  asyncTest.asyncOut();  return "loginPage"+" "+name ;  } |

然后运行主函数，在启动日志可以看到如下信息



## SpringBoot启动端口+访问路径

## 只需要在配置文件中添加访问端口和访问路径就OK，一般添加在配置文件开头。

备注：springboot 2.0之前，配置为 server.context-path=/\*\*\*

springboot 2.0之后，配置为 server.servlet.context-path=/\*\*\*

|  |
| --- |
| **server.port=9999**  **server.context-path=/springboot 或者**  **server.servlet.context-path=/springboot** |

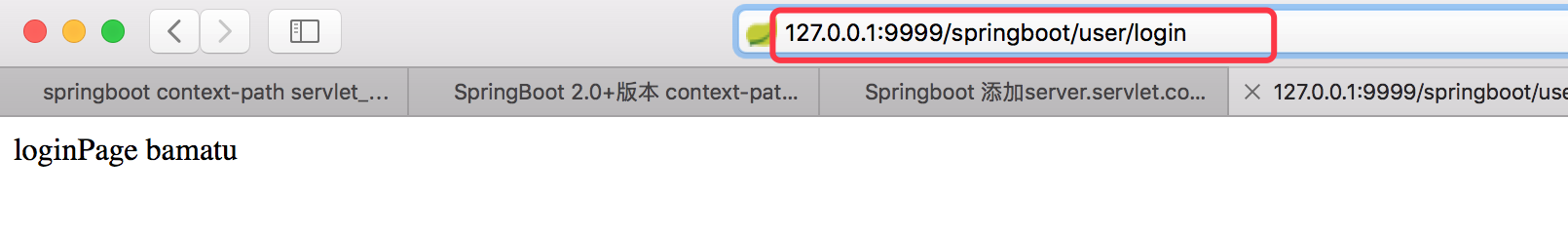
现在在浏览器上，在“域名/IP:端口”的后面加上/springboot，然后再加上其他路径才能访问到项目。

然后运行主函数，在浏览器上输入请求后<http://127.0.0.1:9999/user/login看到如下信息>

输入原来未加访问路径的地址，提示如下



输入加上访问路径的地址，提示如下



## SpringBoot配置文件yml

## 下面讲一下SpringBoot中另外一种格式的配置文件，名为application.yml的配置文件，这种配置文件更方便我们使用，有提示功能，而且SpringBoot也是默认去读取这个格式的配置文件，我们这里改变一下配置文件的风格。

下面是我们整套课程完整的配置文件：

|  |
| --- |
| spring.datasource.url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull  spring.datasource.username=root  spring.datasource.password=wjx123456  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  # 配置服务器端口，默认是8080，可以不用配置  server.port=9999  # 配置访问路径  server.servlet.context-path=/springboot  name=bamatu  spring.thymeleaf.suffix=.html  spring.thymeleaf.mode=HTML5  spring.thymeleaf.encoding=UTF-8  spring.thymeleaf.servlet.content-type=text/html |

下面我们把它改造成application.yml的风格，非常有层次感

|  |
| --- |
| spring:  datasource:  url: jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull  username: root  password: wjx123456  driver-class-name: com.mysql.jdbc.Driver  thymeleaf:  suffix: .html  mode: HTML5  encoding: UTF-8  servlet:  content-type: text/html  # 配置服务器端口，默认是8080，可以不用配置  server:  port: 9999  servlet:  context-path: /springboot  name: bamatu |



如果没有高亮，请在eclipse中安装SpringToolsSuits。然后重启eclipse，选择打开方式就可以高亮了。

备注：需要将同一开头的属性统一在一起，不能同时存在两种相同的开头的属性，否则启动会失败，此处与application.properties格式的属性文件不同。

**SpringBoot多环境区分**

**我们可以先创建一个配置文件application.yml，然后在该源配置文件上面增加下面的一个配置**

|  |
| --- |
| spring:   profiles:     active: wjx |

**然后再创建另一个配置文件application-wjx.yml即可， 这样系统就会优先去扫描配置文件是application-wjx.yml的配置文件。**

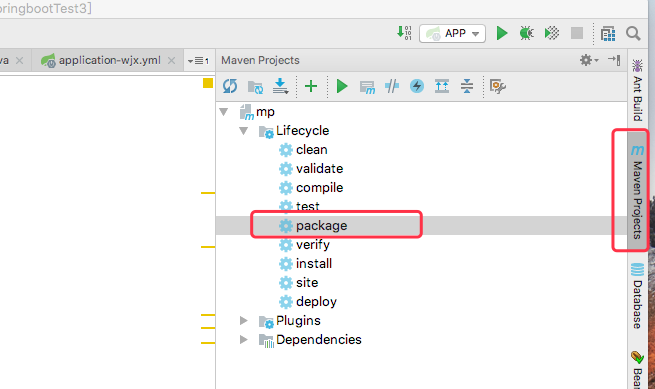
**如果在application-wjx.yml里面没有要请求的信息（包括属性、数据库配置等等），则会去application.yml里面寻找。如果在application.yml还不能找到，则会报错。**

## SpringBoot打包发布

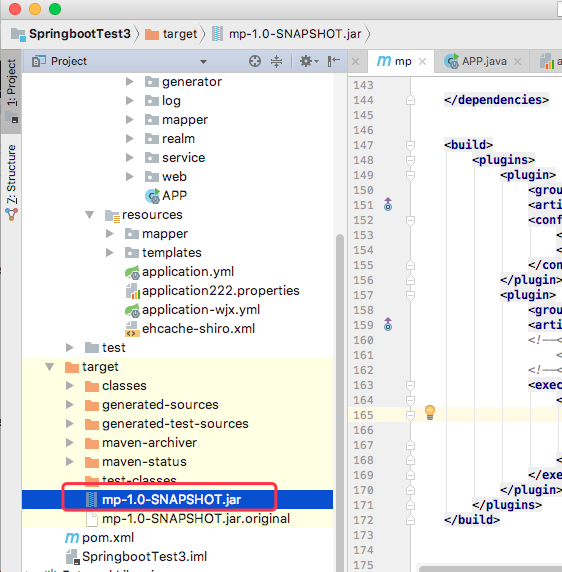
需要在pom文件中添加如下代码

|  |
| --- |
| **<build>**  **<plugins>**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-compiler-plugin</artifactId>**  **<configuration>**  **<source>1.8</source>**  **<target>1.8</target>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **<configuration> // 有时候不需要这个配置**  **<maimClass>com.majiaxueyuan.App</maimClass> // 有时候不需要这个配置**  **</configuration> // 有时候不需要这个配置**  **<executions>**  **<execution>**  **<goals>**  **<goal>repackage</goal>**  **</goals>**  **</execution>**  **</executions>**  **</plugin>**  **</plugins>**  **</build>** |

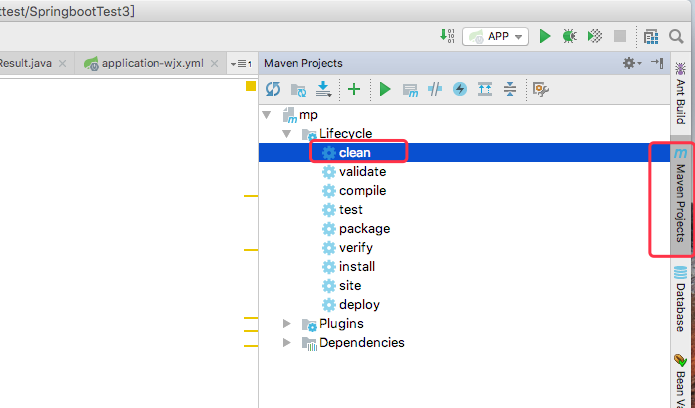
打包时，使用ideal的Maven Project的package，双击即可



会在左侧target目录下面生成jar包



删除包时，使用ideal的Maven Project的clean，双击即可



到这里，我们全宇宙最精辟的SpringBoot教程就结束了，我相信你已经在这里学习到了怎么去使用，怎么快捷的使用，怎么采用当下最牛X的框架去使用SpringBoot，后续我们将退出更多的中高级课程。包含多线程，高并发，高可用，分布式，微服务。感谢你们选择码家学院。

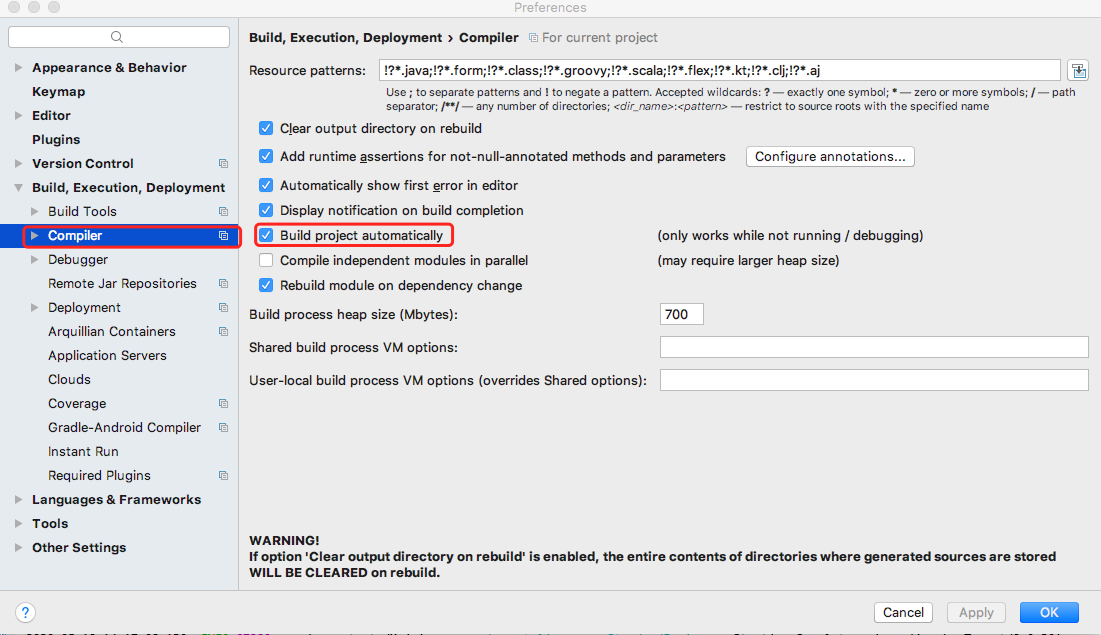
**热部署**

在pom文件中设置如下

|  |
| --- |
| *<?***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 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**groupId**>com.wjxspringboot</**groupId**>  <**artifactId**>test5</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**>   *<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-parent -->* <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>2.1.12.RELEASE</**version**>  *<!--<type>pom</type>-->* </**parent**>  <**dependencies**>   *<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  *<!--<version>2.3.0.RELEASE</version>-->* </**dependency**>   *<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-devtools -->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-devtools</**artifactId**>  *<!--<version>2.2.1.RELEASE</version>-->* </**dependency**>  </**dependencies**>  </**project**> |

版本信息中，设置如上时可以启用服务，即在2.1.12.RELEASE版本下可以，版本不统一时，可能会启动失败。

然后在 Preferences --> Compiler --> Build project automatically



勾选Build project automatically 。

启动服务后，当修改Java文件后，容器会重新加载本项目的Java类，不需要将项目停止再启动。

**QQ：438944209**

**网址：www.majiaxueyuan.com**

**SSM**：Spring+SpringMVC+MyBatis框架集由Spring、MyBatis两个开源框架整合而成（SpringMVC是Spring中的部分内容）。常作为数据源较简单的web项目的框架。

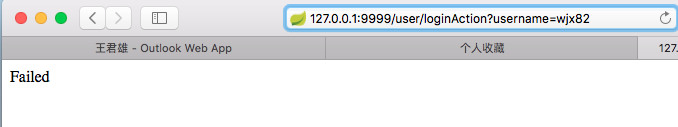
**SSH**：是 struts+spring+hibernate的一个集成框架，是目前比较流行的一种Web应用程序开源框架。

**CURD**： 它代表创建（Create）、更新（Update）、读取（Retrieve）和删除（Delete）操作。

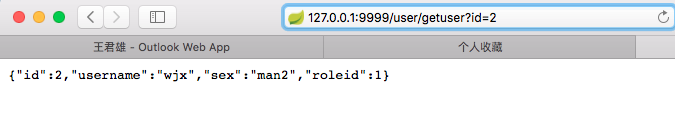
问题:

1、项目是SpringbootTest3

执行127.0.0.1:9999/user/loginAction?username=wjx82失败，但是后面的是登录后可进行的操作。应该是登录失败后，那些登录后才能进行的操作不应该被执行



http://127.0.0.1:9999/user/getuser?id=2



2、json转换

|  |
| --- |
| package com.iqianjin.lego.core.util;  import com.google.gson.Gson;  import com.google.gson.GsonBuilder;  import java.lang.reflect.Type;  import java.util.Date;  import org.joda.time.DateTime;  public class JsonUtils {  private static final Gson GSON = (new GsonBuilder()).registerTypeAdapter(Date.class, new JsonDateSerializer()).registerTypeAdapter(DateTime.class, new JsonJodaDateTimeSerializer()).create();  public JsonUtils() {  }  public static String toJson(Object obj) {  return GSON.toJson(obj);  }  public static <T> T parseObject(String json, Class<T> classOfT) {  return GSON.fromJson(json, classOfT);  }  public static <T> T parseObject(String json, Type typeOfT) {  return GSON.fromJson(json, typeOfT);  }  } |

|  |
| --- |
| package com.iqianjin.operation.db.activity.dto.memberupgrade;  public class MemberUpgradeAssignmentConfig {  private Integer floorMemberLevel;  private Integer awardId;  private String awardName;  private Integer awardType;  private Integer firstUpgradeType;  public Integer getFloorMemberLevel() {  return floorMemberLevel;  }  public void setFloorMemberLevel(Integer floorMemberLevel) {  this.floorMemberLevel = floorMemberLevel;  }  public Integer getAwardId() {  return awardId;  }  public void setAwardId(Integer awardId) {  this.awardId = awardId;  }  public String getAwardName() {  return awardName;  }  public void setAwardName(String awardName) {  this.awardName = awardName;  }  public Integer getAwardType() {  return awardType;  }  public void setAwardType(Integer awardType) {  this.awardType = awardType;  }  public Integer getFirstUpgradeType() {  return firstUpgradeType;  }  public void setFirstUpgradeType(Integer firstUpgradeType) {  this.firstUpgradeType = firstUpgradeType;  }  } |

|  |
| --- |
| activity.getConfig() =  {"floorMemberLevel":3,"firstUpgradeType":1,"awardId":4158,"awardName":"随机红包2411","awardType":0} |

|  |
| --- |
| MemberUpgradeAssignmentConfig upgradeAssignmentConfig = JsonUtils.*parseObject*(activity.getConfig(), MemberUpgradeAssignmentConfig.**class**); |

3、结果集

|  |
| --- |
| //  // Source code recreated from a .class file by IntelliJ IDEA  // (powered by Fernflower decompiler)  //  package com.iqianjin.lego.contracts;  public class Result<T> {  private int code;  private String message;  private T bean;  public Result() {  }  public Result(int code, String message) {  this.code = code;  this.message = message;  }  public Result(int code, String message, T bean) {  this.code = code;  this.message = message;  this.bean = bean;  }  public boolean isSuccess() {  return this.code == ResultCode.SUCCESS.getCode();  }  public int getCode() {  return this.code;  }  public Result setCode(int code) {  this.code = code;  return this;  }  public String getMessage() {  return this.message;  }  public Result setMessage(String message) {  this.message = message;  return this;  }  public T getBean() {  return this.bean;  }  public Result setBean(T bean) {  this.bean = bean;  return this;  }  public static Result ok() {  return new Result(ResultCode.SUCCESS.getCode(), ResultCode.SUCCESS.getMessage());  }  public static <T> Result ok(T bean) {  return new Result(ResultCode.SUCCESS.getCode(), ResultCode.SUCCESS.getMessage(), bean);  }  public static Result failure(int code, String message) {  return new Result(code, message);  }  public static <T> Result failure(int code, String message, T bean) {  return new Result(code, message, bean);  }  public static Result notLogin() {  return new Result(ResultCode.USER\_NOT\_LOGIN.getCode(), ResultCode.USER\_NOT\_LOGIN.getMessage());  }  public static Result notLogin(String message) {  return new Result(ResultCode.USER\_NOT\_LOGIN.getCode(), message);  }  public static Result invalidParam() {  return new Result(ResultCode.INVALID\_PARAM.getCode(), ResultCode.INVALID\_PARAM.getMessage());  }  public static Result invalidParam(String message) {  return new Result(ResultCode.INVALID\_PARAM.getCode(), message);  }  public static Result systemError() {  return new Result(ResultCode.INTERNAL\_SERVER\_ERROR.getCode(), ResultCode.INTERNAL\_SERVER\_ERROR.getMessage());  }  public static Result systemError(String message) {  return new Result(ResultCode.INTERNAL\_SERVER\_ERROR.getCode(), message);  }  } |

4、enum实例

|  |
| --- |
| package com.iqianjin.operation.common.constant;  public enum ActivityStatus {  NOT\_START(1, "未开始"), START(2, "已开始"), END(3, "已结束"), OFF(4, "下线"), PAUSE(5, "暂停");  private int code;  private String desc;  ActivityStatus(int code, String desc) {  this.code = code;  this.desc = desc;  }  public int getCode() {  return code;  }  public String getDesc() {  return desc;  }  } |

枚举调用

|  |
| --- |
| public ActivityStatus parseActivityStatus(OperationActivity activity, Date now) {  if (activity==null) {  throw new IllegalArgumentException("活动不存在");  }  //1, "已发布", 2, "未发布", 3, "临时下线"  Integer state = activity.getState();  ActivityStatus status;  if (state == OperationActivityState.NOT\_RELEASE.getCode()) {  status = ActivityStatus.NOT\_START;  } else if (state == OperationActivityState.PAUSE.getCode() ||  state == OperationActivityState.DELETED.getCode()) {  //都算作已下线  status = ActivityStatus.OFF;  } else {  Date startTime = activity.getStartTime();  Date endTime = activity.getEndTime();  Date offlineTime = activity.getOfflineTime();  if (now.before(startTime)) {  //未开始  status = ActivityStatus.NOT\_START;  } else if (now.equals(offlineTime) || now.after(offlineTime)) {  //已下线  status = ActivityStatus.OFF;  } else if (now.equals(endTime) || now.after(endTime)) {  //已结束  status = ActivityStatus.END;  } else {  //进行中  status = ActivityStatus.START;  }  }  return status;  } |

5、LoggerFactory用法

|  |
| --- |
| package com.iqianjin.operation.api.web.controller;  import com.iqianjin.lego.auth.SecurityContextHolder;  import com.iqianjin.lego.auth.annotation.RequiredAuth;  import com.iqianjin.lego.contracts.Result;  import com.iqianjin.operation.api.service.InvestAcquireCodeService;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  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;  import javax.annotation.Resource;  @RestController  @RequestMapping("/operation/api/acquire-code")  public class InvestAcquireCodeController {  private final Logger logger = LoggerFactory.getLogger(this.getClass());  @Resource  private InvestAcquireCodeService investAcquireCodeService;  @RequiredAuth  @GetMapping("/{activityId}/code-quantity")  public Result queryCodeQuantity(@PathVariable("activityId") Integer activityId) {  Long userId = SecurityContextHolder.getContext().getUserId();  try {  return investAcquireCodeService.getCodeQuantity(activityId, userId);  } catch (Exception e) {  logger.error("投资得码-查询用户码数量异常userId:{},activityId:{}", userId, activityId, e);  return Result.systemError();  }  }  } |

6、

|  |
| --- |
| package com.iqianjin.operation.api.web.controller;  import com.iqianjin.lego.auth.SecurityContextHolder;  import com.iqianjin.lego.auth.annotation.OptionalAuth;  import com.iqianjin.lego.auth.annotation.RequiredAuth;  import com.iqianjin.lego.contracts.Result;  import com.iqianjin.operation.api.service.ExchangeService;  import org.apache.commons.lang3.StringUtils;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.web.bind.annotation.\*;  import javax.annotation.Resource;  /\*\*  \* 满兑组件接口  \* wiki: http://wiki.puhuitech.cn/pages/viewpage.action?pageId=38983730  \* @author Ricky Fung  \*/  @RestController  @RequestMapping("/operation/api/exchange-award")  public class ExchangeController {  private final Logger logger = LoggerFactory.getLogger(this.getClass());  @Resource  private ExchangeService exchangeService;  /\*\*  \* 首页  \* @param activityId  \* @return  \*/  @OptionalAuth  @GetMapping("/{activityId}/homepage")  public Result getHomepage(@PathVariable(name = "activityId") Integer activityId) {  Long userId = SecurityContextHolder.getContext().getUserId();  logger.info("满兑组件-查询首页信息开始, 用户userId:{}, activityId:{}", userId, activityId);  return exchangeService.getHomepage(activityId, userId);  }  /\*\*  \* 提交兑换（不需要填收货地址 引导用户去"我的奖品页"填地址）  \* @param activityId  \* @param awardId  \* @param quantity  \* @return  \*/  @RequiredAuth  @PostMapping("/{activityId}/submit")  public Result submitExchange(@PathVariable(name = "activityId") String activityId,  @RequestParam(name = "sectionId", required = false) String sectionId,  @RequestParam(name = "awardId", required = false) String awardId,  @RequestParam(name = "quantity", required = false) String quantity) {  Long userId = SecurityContextHolder.getContext().getUserId();  logger.info("满兑组件-提交兑换申请开始, 用户userId:{}, activityId:{}, sectionId:{}, awardId:{}, quantity:{}",  userId, activityId, sectionId, awardId, quantity);  if (StringUtils.isAnyEmpty(activityId, sectionId, awardId, quantity)) {  return Result.invalidParam("参数不能为空");  }  try {  int actId = Integer.parseInt(activityId);  return exchangeService.submit(actId, userId,  Integer.parseInt(sectionId), Integer.parseInt(awardId), Integer.parseInt(quantity));  } catch (NumberFormatException e) {  return Result.invalidParam("参数格式不正确");  } catch (Exception e) {  logger.error(String.format("满兑组件-提交兑换申请异常, activityId:%s, userId:%s, awardId:%s, quantity:%s",  activityId, userId, awardId, quantity), e);  }  return Result.systemError();  }  } |

7、

|  |
| --- |
| if (StringUtils.isAnyEmpty(activityId, sectionId, awardId, quantity)) {  return Result.invalidParam("参数不能为空");  } |

8、

|  |
| --- |
| **package** com.iqianjin.operation.redis.model.ererydayrank;  **import** java.util.Date; **public class** SeqConfig {  **private** Integer **seq**;  **private** Date **startTime**;  **private** Date **endTime**;  **private** String **config**;  **private** Integer **status**;   **public** Integer getSeq() {  **return seq**;  }   **public void** setSeq(Integer seq) {  **this**.**seq** = seq;  }   **public** Date getStartTime() {  **return startTime**;  }   **public void** setStartTime(Date startTime) {  **this**.**startTime** = startTime;  }   **public** Date getEndTime() {  **return endTime**;  }   **public void** setEndTime(Date endTime) {  **this**.**endTime** = endTime;  }   **public** String getConfig() {  **return config**;  }   **public void** setConfig(String config) {  **this**.**config** = config;  }   **public** Integer getStatus() {  **return status**;  }   **public void** setStatus(Integer status) {  **this**.**status** = status;  } } |

|  |
| --- |
| **package** com.alibaba.fastjson;  **public abstract class JSON implements JSONStreamAware, JSONAware {**  **public static <T> List<T> parseArray(String text, Class<T> clazz) {**  **if (text == null) {**  **return null;**  **} else {**  **DefaultJSONParser parser = new DefaultJSONParser(text, ParserConfig.getGlobalInstance());**  **JSONLexer lexer = parser.lexer;**  **int token = lexer.token();**  **ArrayList list;**  **if (token == 8) {**  **lexer.nextToken();**  **list = null;**  **} else if (token == 20 && lexer.isBlankInput()) {**  **list = null;**  **} else {**  **list = new ArrayList();**  **parser.parseArray(clazz, list);**  **parser.handleResovleTask(list);**  **}**  **parser.close();**  **return list;**  **}**  **}**  **}** |

|  |
| --- |
| **Value =**  **"[{\"seq\": 1,\"startTime\":\"2019-11-13 00:00:00\",\"endTime\":\"2019-11-13 23:59:59\"},{\"seq\": 2,\"startTime\":\"2019-11-14 00:00:00\",\"endTime\":\"2019-11-14 23:59:59\"}]"** |

|  |
| --- |
| **public** List<SeqConfig> getSeqConfigList(Integer activityId) {  String key = EveryDayRankConstant.*getSeqSummaryKey*(activityId);  String value = **stringRedisTemplate**.opsForValue().get(key);  List<SeqConfig> seqConfigs = JSONArray.*parseArray*(value, SeqConfig.**class**);  **return** seqConfigs; } |

9、

|  |
| --- |
|  |

10、

|  |
| --- |
|  |

11、

|  |
| --- |
|  |

12、

|  |
| --- |
|  |

13、

|  |
| --- |
|  |

14、

|  |
| --- |
|  |

15、

|  |
| --- |
|  |

16、

|  |
| --- |
|  |

17、

|  |
| --- |
|  |

18、

|  |
| --- |
|  |

19、

|  |
| --- |
|  |

20、

|  |
| --- |
|  |

21、

|  |
| --- |
|  |

22、

|  |
| --- |
|  |

23、

|  |
| --- |
|  |

24、

|  |
| --- |
|  |

25、

|  |
| --- |
|  |

26、

|  |
| --- |
|  |

27、

|  |
| --- |
|  |

28、

|  |
| --- |
|  |

29、

|  |
| --- |
|  |

30、

|  |
| --- |
|  |

31、

|  |
| --- |
|  |