## 1 表单验证

（1）将传入的参数封装在一个类中，在类中对变量进行条件限制

（2）标签如：@GeneratedValue、@NotBlank(message = "这个字段必传")、@Min(value = 18, message = "未成年少女禁止入门")、@Length()等

（3）在控制类中传入的类对象参数前面添加@Valid

（4）返回错误的信息，在控制层方法添加一个参数：BindingResult bindingResult

判断如下：注释的内容是其他测试输出的内容，请按自己需要的返回就行了

if(bindingResult.hasErrors()){  
 *logger*.info(bindingResult.getFieldError().getDefaultMessage())**;**// return 12/0;  
// return ResultUtils.error(1,bindingResult.getFieldError().getDefaultMessage());  
 return null**;** }

## 2 AOP的使用

AOP是spring的两大特性之一，所以一定要会使用。

添加日志输入请求信息：

1. 创建一个切面类：添加@Aspect、@Component（引入容器，spring的另一个特性）
2. 为了减少路径的重复，创建一个切面方法（路径为每个在GirlController下的方法）

如：@Pointcut("execution(public \* com.imooc.controller.GirlController.\*(..))")  
public void log(){  
}

(3) 定义日志输出对象

//使用log打印日志  
private final static Logger *logger* = LoggerFactory.*getLogger*(HttpAspect.class)**;**

(4) 添加访问方法前先执行的方法，如：

@Before("log()")  
public void doBefore(JoinPoint joinPoint){  
 ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*()**;** HttpServletRequest request = attributes.getRequest()**;** //url  
 *logger*.info("url={}"**,**request.getRequestURL())**;** //ip  
 *logger*.info("ip={}"**,**request.getRemoteAddr())**;** //method  
 *logger*.info("method={}"**,**request.getMethod())**;** //类方法  
 *logger*.info("class\_method={}"**,**joinPoint.getSignature().getDeclaringTypeName()+"."+joinPoint.getSignature().getName())**;** //参数  
 *logger*.info("args={}"**,**joinPoint.getArgs())**;**}

控制台输出结果如下：

2019-01-22 10:19:41.015 INFO 1412 --- [nio-8080-exec-2] com.imooc.aspect.HttpAspect : url=http://127.0.0.1:8080/girls/getAge/14

2019-01-22 10:19:41.015 INFO 1412 --- [nio-8080-exec-2] com.imooc.aspect.HttpAspect : ip=127.0.0.1

2019-01-22 10:19:41.015 INFO 1412 --- [nio-8080-exec-2] com.imooc.aspect.HttpAspect : method=GET

2019-01-22 10:19:41.015 INFO 1412 --- [nio-8080-exec-2] com.imooc.aspect.HttpAspect : class\_method=com.imooc.controller.GirlController.getAge

2019-01-22 10:19:41.015 INFO 1412 --- [nio-8080-exec-2] com.imooc.aspect.HttpAspect : args=14

（5） 添加一个将结果输出在控制台的方法：返回的结果设置为Object，包含所有引用数据类型，如果参数写的是object而不是object.toString()，（toString()方法要在该类重写toString方法，输出才好看。），结果为null不会报错，反之则会在控制台报错。

@AfterReturning(returning = "object"**,**pointcut = "log()")  
public void returnResult(Object object){  
 *logger*.info("response={}"**,**object.toString())**;**}

## 3 统一异常处理（WEB和Ajax）

（1）创建自定义异常类，如：注意在springboot中创建的话最好继承RuntimeException

package com.imooc.exception**;***/\*\*  
 \** ***@ClassName*** *GirlException  
 \** ***@Description*** *springboot只会对RuntimeException进行回滚，所以在自定义类的时候要继承RuntimeException  
 \** ***@Author*** *zhoup  
 \** ***@Date*** *2019/1/21 17:16  
 \** ***@Version*** *1.0  
 \*\*/*public class GirlException extends RuntimeException {  
  
 private Integer code**;** public GirlException(Integer code**,**String message) {  
 super(message)**;** this.code = code**;** }  
  
 public Integer getCode() {  
 return code**;** }  
  
 public void setCode(Integer code) {  
 this.code = code**;** }  
}

（2）创建异常处理类

package com.imooc.handle**;**import com.imooc.domain.Result**;**import com.imooc.enums.ResultEnums**;**import com.imooc.exception.GirlException**;**import com.imooc.utils.ResultUtils**;**import org.slf4j.Logger**;**import org.slf4j.LoggerFactory**;**import org.springframework.web.bind.annotation.ControllerAdvice**;**import org.springframework.web.bind.annotation.ExceptionHandler**;**import org.springframework.web.bind.annotation.ResponseBody**;***/\*\*  
 \** ***@ClassName*** *ExceptionHandle  
 \** ***@Description*** *\** ***@Author*** *zhoup  
 \** ***@Date*** *2019/1/21 17:24  
 \** ***@Version*** *1.0  
 \*\*/*@ControllerAdvice  
public class ExceptionHandle {  
 private final static Logger *logger* = LoggerFactory.*getLogger*(ExceptionHandle.class)**;** @ExceptionHandler(value = Exception.class)//处理的都是Exception的子类，RuntimeException也是它的子类  
 @ResponseBody//异常返回给页面  
 public Result handle(Exception e){  
 if(e instanceof GirlException){  
 GirlException girlException = (GirlException) e**;** return ResultUtils.*error*(girlException.getCode()**,**girlException.getMessage())**;** }else{  
 *logger*.error("【系统异常】={}"**,**e)**;** return ResultUtils.*error*(ResultEnums.*UNKNOW\_ERROR*.getCode()**,**ResultEnums.*UNKNOW\_ERROR*.getMessage())**;** }  
 }  
}

页面输出效果：

{

"code": 100,

"message": "我猜你还在读小学吧",

"data": null

}

{

"code": -1,

"message": "未知错误",

"data": null

}

补充结果输出工具类和异常输出枚举类：

package com.imooc.utils**;**import com.imooc.domain.Result**;***/\*\*  
 \** ***@ClassName*** *ResultYtils  
 \** ***@Description*** *\** ***@Author*** *zhoup  
 \** ***@Date*** *2019/1/21 16:44  
 \** ***@Version*** *1.0  
 \*\*/*public class ResultUtils {  
  
 public static Result success(Object data){  
 Result result = new Result()**;** result.setCode(**0**)**;** result.setMessage("成功")**;** result.setData(data)**;** return result**;** }  
  
 public static Result success(){  
 return *success*(null)**;** }  
  
 public static Result error(Integer code**,**String message){  
 Result result = new Result()**;** result.setCode(code)**;** result.setMessage(message)**;** result.setData(null)**;** return result**;** }  
}

package com.imooc.enums**;**public enum ResultEnums {  
 *UNKNOW\_ERROR*(-**1,**"未知错误")**,** *SUCCESS*(**0,**"成功")**,** *PRIMARY\_ERROR*(**100,**"我猜你还在读小学吧")**,** *MIDDLE\_ERROR*(**101,**"我猜你还在读初中吧")  
 **;** private Integer code**;** private String message**;** ResultEnums(Integer code**,** String message) {  
 this.code = code**;** this.message = message**;** }  
  
 public Integer getCode() {  
 return code**;** }  
  
 public String getMessage() {  
 return message**;** }  
}

补充一个ajax和web统一处理的类

package com.zpr.exception**;**import com.zpr.pojo.IMoocJSONResult**;**import org.springframework.web.bind.annotation.ControllerAdvice**;**import org.springframework.web.bind.annotation.ExceptionHandler**;**import org.springframework.web.bind.annotation.RestControllerAdvice**;**import org.springframework.web.servlet.ModelAndView**;**import javax.servlet.http.HttpServletRequest**;**import javax.servlet.http.HttpServletResponse**;***/\*\*  
 \** ***@ClassName*** *CommonExceptionHandler  
 \** ***@Description*** *\** ***@Author*** *zhoup  
 \** ***@Date*** *2019/1/29 8:36  
 \** ***@Version*** *1.0  
 \*\*/*@RestControllerAdvice//没有改为rest那么ajax异常之后跳转到没有结果输出的错误error  
public class CommonExceptionHandler{  
 public static final String *COMMON\_ERROR\_VIEW* = "error"**;** @ExceptionHandler(value = Exception.class)  
 public Object errorHandler(HttpServletRequest request**,** HttpServletResponse response**,**Exception e)throws Exception{  
 e.printStackTrace()**;** if(*isAjax*(request)){//判断是否Ajax异常  
 return IMoocJSONResult.*errorException*(e.getMessage())**;** }else{  
 ModelAndView mav = new ModelAndView()**;** mav.addObject("exception"**,**e)**;** mav.addObject("url"**,**request.getRequestURL())**;** mav.setViewName(*COMMON\_ERROR\_VIEW*)**;** return mav**;** }  
  
  
 }  
  
 public static boolean isAjax(HttpServletRequest request) {  
 return (request.getHeader("X-Requested-With")!=null&&"XMLHttpRequest".equals(request.getHeader("X-Requested-With").toString()))**;** }  
}

## 4 单元测试

贴一个控制层的异常，觉得这个比较容易忘，参数比较复杂：控制层不是像service层一样直接注入service类就可以调用和判断，我们还需要判断路径访问和请求结果、返回内容等问题。

package com.imooc.controller**;**import org.junit.Test**;**import org.junit.runner.RunWith**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc**;**import org.springframework.boot.test.context.SpringBootTest**;**import org.springframework.test.context.junit4.SpringRunner**;**import org.springframework.test.web.servlet.MockMvc**;**import org.springframework.test.web.servlet.request.MockMvcRequestBuilders**;**import org.springframework.test.web.servlet.result.MockMvcResultMatchers**;**import static org.junit.Assert.\***;**@RunWith(SpringRunner.class)  
@SpringBootTest  
@AutoConfigureMockMvc  
public class GirlControllerTest {  
  
 @Autowired  
 private MockMvc mvc**;** @Test  
 public void girlList() throws Exception {  
 mvc.perform(MockMvcRequestBuilders.*get*("/girls"))  
 .andExpect(MockMvcResultMatchers.*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*content*().string("[{\"id\":1,\"cupSize\":\"A\",\"age\":12,\"money\":0.0},{\"id\":4,\"cupSize\":\"A\",\"age\":12,\"money\":0.0},{\"id\":5,\"cupSize\":\"A\",\"age\":35,\"money\":0.0},{\"id\":6,\"cupSize\":\"A\",\"age\":23,\"money\":0.0},{\"id\":7,\"cupSize\":\"C\",\"age\":23,\"money\":0.0},{\"id\":8,\"cupSize\":\"B\",\"age\":23,\"money\":0.0},{\"id\":9,\"cupSize\":\"A\",\"age\":23,\"money\":0.0},{\"id\":10,\"cupSize\":\"C\",\"age\":23,\"money\":0.0},{\"id\":11,\"cupSize\":\"C\",\"age\":14,\"money\":0.0},{\"id\":12,\"cupSize\":\"C\",\"age\":23,\"money\":150.0},{\"id\":13,\"cupSize\":\"C\",\"age\":23,\"money\":150.0},{\"id\":14,\"cupSize\":\"C\",\"age\":4,\"money\":150.0}]"))**;** }  
}

service的方法的单元测试：直接生成测试方法比较方便（IDEA），在类或者方法的地方右键Go to (Test),自己添加测试的类。

package com.imooc.service**;**import com.imooc.domain.Girl**;**import org.junit.Assert**;**import org.junit.Test**;**import org.junit.runner.RunWith**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.boot.test.context.SpringBootTest**;**import org.springframework.test.context.junit4.SpringRunner**;**import static org.junit.Assert.\***;**@RunWith(SpringRunner.class)  
@SpringBootTest  
public class GirlServiceTest {  
 @Autowired  
 private GirlService girlService**;** @Test  
 public void findOne(){  
 Girl girl = girlService.findOne(**10**)**;** Assert.*assertEquals*(new Integer(**23**)**,**girl.getAge())**;** }  
}

当进行maven打包时，会自动运行我们创建的单元测试：mvn clean package;

当然在IDEA中不用使用命令行,直接找到maven视图就可以了,IDEA真的是超级强大,当然你也可以在打包时不选择测试: mvn clean package -Dmaven.test.skip=true(目测这个只能用命令行)

## 5 Json格式返回数据注解

当我们将json数据返回页面时，有时候我们可以对返回的数据进行处理，可以在pojo类中对字段进行加注解，（在类上面即整个类通用，单项则针对单项起作用），如密码项可以忽略，时间格式规范以及字段为空时忽略：常用的三项：

@JsonIgnore

@JsonFormat(pattern="yyyy-MM-dd")

@JsonInclude(Include.*NON\_NULL*)

未加限制：

{

name: "zola",

age: 17,

birthday: "2019-01-23 02:27:52 下午"

}

加了注解限制：(这里的格式是使用了工具类返回格式)

{

status: 200,

msg: "OK",

data: {

name: "zola",

age: 17,

birthday: "2019-01-23 02:28:42 下午"

},

ok: null

}

## 6 配置devtools热部署工具（IDEA）

热部署：热部署

devtools可以实现页面热部署（即页面修改后会立即生效，

这个可以直接在application.properties文件中配置spring.thymeleaf.cache=false来实现）

实现类文件热部署（类文件修改后不会立即生效），实现对属性文件的热部署。

即devtools会监听classpath下的文件变动，并且会立即重启应用（发生在保存时机），

注意：因为其采用的虚拟机机制，该项重启是很快的

（1）base classloader （Base类加载器）：加载不改变的Class，例如：第三方提供的jar包。

（2）restart classloader（Restart类加载器）：加载正在开发的Class。

为什么重启很快，因为重启的时候只是加载了在开发的Class，没有重新加载第三方的jar包。

（1）配置maven

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <!-- optional=true, 依赖不会传递, 该项目依赖devtools;  
 之后依赖boot项目的项目如果想要使用devtools, 需要重新引入 -->  
 <optional>true</optional>  
</dependency>

（2）配置application.properties文件

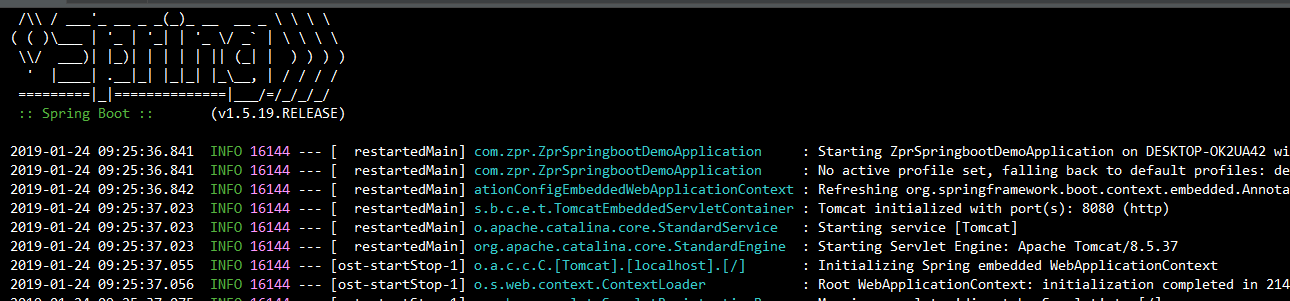
#关闭缓存, 即时刷新  
#spring.freemarker.cache=false  
#spring.thymeleaf.cache=true  
  
#热部署生效  
spring.devtools.restart.enabled=true  
#设置重启的目录,添加那个目录的文件需要restart  
spring.devtools.restart.additional-paths=src/main/java  
# 为mybatis设置，生产环境可删除  
restart.include.mapper=/mapper-[\\w-\\.]+jar  
restart.include.pagehelper=/pagehelper-[\\w-\\.]+jar  
#排除那个目录的文件不需要restart  
#spring.devtools.restart.exclude=static/\*\*,public/\*\*  
#classpath目录下的WEB-INF文件夹内容修改不重启  
#spring.devtools.restart.exclude=WEB-INF/\*\*

（3）如果你是eclipse或者sts，应该就可以了，然后IDEA需要再配置一下

1） “File” -> “Settings” -> “Build,Execution,Deplyment” -> “Compiler”，选中打勾 “Build project automatically” 。

　　2） 组合键：“Shift+Ctrl+Alt+/” ，选择 “Registry” ，选中打勾 “compiler.automake.allow.when.app.running” 。

（4）修改自己的文件然后控制台会显示重新启动



## 7 从配置文件读取实体类数据

这里有个问题没解决，有思路的麻烦可以给我解答一下，就是Date方式的读取

（1） 添加maven

<!--配置类与资源文件对应的依赖-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-configuration-processor</artifactId>  
 <optional>true</optional>  
</dependency>

（2） User.properties配置文件（放在resources路径下就可以）

com.zpr.autouser.name = 菲菲  
com.zpr.autouser.password = 345  
com.zpr.autouser.age = 34  
#com.zpr.autouser.birthday = 2016-11-11

（3）在实体类上添加注解（我看到还有别的添加注解的方式，所以不仅仅这种方式配置）

@Component //神奇，不知道为什么@Configuration没有用，表明与配置文件对应  
@ConfigurationProperties(prefix = "com.zpr.autouser") //这里是配置文件的参数前面对应的前缀  
@PropertySource(value = "classpath:user.properties") //配置文件路径

（4）创建一个控制层方法

@RequestMapping("/getAutoUser")  
public IMoocJSONResult getAutoUser(){  
 User bean = new User()**;** BeanUtils.*copyProperties*(user**,**bean)**;** return IMoocJSONResult.*ok*(bean)**;**}

返回数据：

{

"status": 200,

"msg": "OK",

"data": {

"name": "菲菲",

"age": 34,

"birthday": null

},

"ok": null

}

## 8 整合模板引擎（freemaker和thymeleaf）

敲黑板：前端页面的要放在templates（动态模板）路径下，两种模板又分别放在自己的路径下，好区分；

### （1）添加maven

<!--添加freemaker模板-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-freemarker</artifactId>  
</dependency>

### （2）添加配置（application.properties）

# freemarker 静态资源配置  
#  
############################################################  
#设定ftl文件路径  
spring.freemarker.template-loader-path=classpath:/templates  
# 关闭缓存, 即时刷新, 上线生产环境需要改为true  
spring.freemarker.cache=false  
spring.freemarker.charset=UTF-8  
spring.freemarker.check-template-location=true  
spring.freemarker.content-type=text/html  
spring.freemarker.expose-request-attributes=true  
spring.freemarker.expose-session-attributes=true  
spring.freemarker.request-context-attribute=request  
spring.freemarker.suffix=.ftl

### （3）添加index.ftl文件

<!DOCTYPE html>  
<html>  
<head lang="en">  
 <meta charset="UTF-8" />  
 <title></title>  
</head>  
<body>  
FreeMarker模板引擎  
<h1>${user.name}</h1>  
<h1>${user.password}</h1>  
<h1>${user.age}</h1>  
</body>  
</html>

### （4）控制层访问路径方法

@Controller  
@RequestMapping("ftl")  
public class FreemarkerController {  
  
 @Autowired  
 private User user**;** @RequestMapping("/index")  
 public String index(ModelMap map) {  
 map.addAttribute("user"**,** user)**;** return "freemarker/index"**;** }  
}

### （5）访问结果

贴图片得先保存，我这么懒就直接贴页面代码

<!DOCTYPE html>

<html>

<head lang="en">

<meta charset="UTF-8" />

<title></title>

</head>

<body>

FreeMarker模板引擎

<h1>菲菲</h1>

<h1>345</h1>

<h1>34</h1>

</body>

</html>

## （6）thymleaf的配置和freemaker的配置大同小异

### Maven

<!--添加thymeleaf模板-->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>

### 配置

############################################################  
#  
# thymeleaf 静态资源配置  
#  
############################################################  
spring.thymeleaf.prefix=classpath:/templates/  
spring.thymeleaf.suffix=.html  
spring.thymeleaf.mode=HTML5  
spring.thymeleaf.encoding=UTF-8  
spring.thymeleaf.content-type=text/html  
# 关闭缓存, 即时刷新, 上线生产环境需要改为true  
spring.thymeleaf.cache=false

#设定静态文件路径，js,css等  
spring.mvc.static-path-pattern=/static/\*\*

### 控制层方法：

@RequestMapping("/index")  
 public String index(ModelMap map) {  
 map.addAttribute("name"**,** "thymeleaf-imooc")**;** return "thymeleaf/index"**;** }

### 页面：

<!DOCTYPE html>  
<html>  
<head lang="en">  
 <meta charset="UTF-8" />  
 <title></title>  
</head>  
<body>  
Thymeleaf模板引擎  
<h1 th:text="${name}">hello world~~~~~~~</h1>  
</body>  
</html>

### 显示页面代码：

<!DOCTYPE html>

<html>

<head lang="en">

<meta charset="UTF-8" />

<title></title>

</head>

<body>

Thymeleaf模板引擎

<h1>thymeleaf-imooc</h1>

</body>

</html>

## 9 整合MyBatis和实现分页

这个是MyBatis插件作者的guthub，可以去看看：

https://github.com/abel533/Mapper/wiki

### （1）添加maven

<!--mybatis-->  
<dependency>  
 <groupId>org.mybatis.spring.boot</groupId>  
 <artifactId>mybatis-spring-boot-starter</artifactId>  
 <version>1.3.1</version>  
</dependency>  
<!--mapper-->  
<dependency>  
 <groupId>tk.mybatis</groupId>  
 <artifactId>mapper-spring-boot-starter</artifactId>  
 <version>1.2.4</version>  
</dependency>  
<!--pagehelper-->  
<dependency>  
 <groupId>com.github.pagehelper</groupId>  
 <artifactId>pagehelper-spring-boot-starter</artifactId>  
 <version>1.2.3</version>  
</dependency>

<dependency>  
 <groupId>com.alibaba</groupId>  
 <artifactId>druid</artifactId>  
 <version>1.1.0</version>  
</dependency>  
  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.39</version>  
</dependency>

### （2）配置文件（数据库要根据自己的）

############################################################  
#  
# 配置数据源相关 使用阿里巴巴的 druid 数据源  
#  
############################################################  
spring.datasource.url=jdbc:mysql://localhost:3307/springboot1?useUnicode=true&characterEncoding=UTF-8&useSSL=false&serverTimezone=UTC&nullNamePatternMatchesAll=true  
spring.datasource.username=root  
spring.datasource.password=123456  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.druid.initial-size=**1**spring.datasource.druid.min-idle=**1**spring.datasource.druid.max-active=**20**spring.datasource.druid.test-on-borrow=true  
spring.datasource.druid.stat-view-servlet.allow=true  
  
  
############################################################  
#  
# mybatis 配置  
#  
############################################################  
# mybatis 配置  
mybatis.type-aliases-package=com.zpr.pojo  
mybatis.mapper-locations=classpath:mapper/\*.xml  
# 通用 Mapper 配置  
mapper.mappers=com.zpr.utils.MyMapper  
mapper.not-empty=false  
mapper.identity=MYSQL  
# 分页插件配置  
pagehelper.helperDialect=mysql  
pagehelper.reasonable=true  
pagehelper.supportMethodsArguments=true  
pagehelper.params=count=countSql

### （3）生成逆向工程

添加generatorConfig.xml文件，用来从数据库生成对应的实体类等的，跟pom.xml在同一个路径

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE generatorConfiguration  
 PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  
 "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  
  
<generatorConfiguration>  
 <context id="MysqlContext" targetRuntime="MyBatis3Simple" defaultModelType="flat">  
 <property name="beginningDelimiter" value="`"/>  
 <property name="endingDelimiter" value="`"/>  
  
 <plugin type="tk.mybatis.mapper.generator.MapperPlugin">  
 <property name="mappers" value="com.zpr.utils.MyMapper"/>  
 </plugin>  
  
 <jdbcConnection driverClass="com.mysql.cj.jdbc.Driver"  
 connectionURL="jdbc:mysql://localhost:3307/springboot1?useUnicode=true**&amp;**characterEncoding=UTF-8**&amp;**useSSL=false**&amp;**serverTimezone=UTC**&amp;**nullNamePatternMatchesAll=true"  
 userId="root"  
 password="123456">  
 </jdbcConnection>  
  
 <!-- 对于生成的pojo所在包 -->  
 <javaModelGenerator targetPackage="com.zpr.pojo" targetProject="src/main/java"/>  
  
 <!-- 对于生成的mapper所在目录 -->  
 <sqlMapGenerator targetPackage="mapper" targetProject="src/main/resources"/>  
  
 <!-- 配置mapper对应的java映射 -->  
 <javaClientGenerator targetPackage="com.zpr.mapper" targetProject="src/main/java"  
 type="XMLMAPPER"/>  
  
  
 <table tableName="sys\_user"></table>  
   
 </context>  
</generatorConfiguration>

在utils包下添加GeneratorDisplay.java类

package com.zpr.utils**;**import java.io.File**;**import java.util.ArrayList**;**import java.util.List**;**import org.mybatis.generator.api.MyBatisGenerator**;**import org.mybatis.generator.config.Configuration**;**import org.mybatis.generator.config.xml.ConfigurationParser**;**import org.mybatis.generator.internal.DefaultShellCallback**;**public class GeneratorDisplay {  
  
 public void generator() throws Exception{  
  
 List<String> warnings = new ArrayList<String>()**;** boolean overwrite = true**;** //指定 逆向工程配置文件  
 File configFile = new File("generatorConfig.xml")**;** ConfigurationParser cp = new ConfigurationParser(warnings)**;** Configuration config = cp.parseConfiguration(configFile)**;** DefaultShellCallback callback = new DefaultShellCallback(overwrite)**;** MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config**,** callback**,** warnings)**;** myBatisGenerator.generate(null)**;** }   
   
 public static void main(String[] args) throws Exception {  
 try {  
 GeneratorDisplay generatorSqlmap = new GeneratorDisplay()**;** generatorSqlmap.generator()**;** } catch (Exception e) {  
 e.printStackTrace()**;** }  
   
 }  
}

还有一个MyMapper，用来给Mapper接口们继承的

/\*  
 \* The MIT License (MIT)  
 \*  
 \* Copyright (c) 2014-2016 abel533@gmail.com  
 \*  
 \* Permission is hereby granted, free of charge, to any person obtaining a copy  
 \* of this software and associated documentation files (the "Software"), to deal  
 \* in the Software without restriction, including without limitation the rights  
 \* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell  
 \* copies of the Software, and to permit persons to whom the Software is  
 \* furnished to do so, subject to the following conditions:  
 \*  
 \* The above copyright notice and this permission notice shall be included in  
 \* all copies or substantial portions of the Software.  
 \*  
 \* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR  
 \* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,  
 \* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE  
 \* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER  
 \* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,  
 \* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN  
 \* THE SOFTWARE.  
 \*/  
  
package com.zpr.utils**;**import tk.mybatis.mapper.common.Mapper**;**import tk.mybatis.mapper.common.MySqlMapper**;***/\*\*  
 \* 继承自己的MyMapper  
 \*  
 \** ***@author*** *liuzh  
 \** ***@since*** *2015-09-06 21:53  
 \*/*public interface MyMapper<**T**> extends Mapper<**T**>**,** MySqlMapper<**T**> {  
 //*TODO* //*FIXME 特别注意，该接口不能被扫描到，否则会出错*}

然后在GeneratorDisplay.java类右键运行生成相应的实体类和Mapper接口

### （4）在应用入口类添加注解，就是我们启动的那个类

//扫描mybatis mapper的包路径  
@MapperScan(basePackages = "com.zpr.mapper")  
//扫描需要的所有包，包含一些自用的工具类包所在的路径  
@ComponentScan(basePackages = {"com.zpr"**,**"org.n3r.idworker"})

这里我后续的操作是需要在Mapper接口类上面添加@Component,不然注入的时候总是划红线，虽然不影响，强迫症。

其他是MyBatis的CRUD自己看代码吧。

## 10 定时任务

### （1）在应用类添加注解

@EnableScheduling

### （2）定时处理类

package com.zpr.task**;**import java.text.SimpleDateFormat**;**import java.util.Date**;**import org.springframework.scheduling.annotation.Scheduled**;**import org.springframework.stereotype.Component**;**@Component  
public class TestTask {  
  
 private static final SimpleDateFormat *dateFormat* = new SimpleDateFormat("HH:mm:ss")**;** // 定义每过3秒执行任务  
// @Scheduled(fixedRate = 3000)  
// @Scheduled(cron = "3-45 \* \* \* \* ?")  
 public void reportCurrentTime() {  
 System.*out*.println("现在时间：" + *dateFormat*.format(new Date()))**;** }  
}

启动类会自动开始定时执行，这个是cron时间表达式转换的工具网址: <http://cron.qqe2.com/>（记得年是不能用的，具体用法百度吧）

## 11 启动异步任务

（1）在入口类添加注解

//启动异步任务  
@EnableAsync

（2）创建异步方法类

package com.zpr.task**;**import org.springframework.scheduling.annotation.Async**;**import org.springframework.scheduling.annotation.AsyncResult**;**import org.springframework.stereotype.Component**;**import java.util.concurrent.Future**;**@Component  
public class AsyncTask {  
   
 @Async  
 public Future<Boolean> doTask11() throws Exception {  
 long start = System.*currentTimeMillis*()**;** Thread.*sleep*(**1000**)**;** long end = System.*currentTimeMillis*()**;** System.*out*.println("任务1耗时:" + (end - start) + "毫秒")**;** return new AsyncResult<>(true)**;** }  
   
 @Async  
 public Future<Boolean> doTask22() throws Exception {  
 long start = System.*currentTimeMillis*()**;** Thread.*sleep*(**700**)**;** long end = System.*currentTimeMillis*()**;** System.*out*.println("任务2耗时:" + (end - start) + "毫秒")**;** return new AsyncResult<>(true)**;** }  
   
 @Async  
 public Future<Boolean> doTask33() throws Exception {  
 long start = System.*currentTimeMillis*()**;** Thread.*sleep*(**600**)**;** long end = System.*currentTimeMillis*()**;** System.*out*.println("任务3耗时:" + (end - start) + "毫秒")**;** return new AsyncResult<>(true)**;** }  
}

（3）调用控制类：异步启动时，时间取最长的那个，但是由于程序也需要消耗一些时间，所以会比最大的时间再大一点

package com.zpr.controller**;**import com.zpr.task.AsyncTask**;**import org.springframework.beans.factory.annotation.Autowired**;**import org.springframework.web.bind.annotation.RequestMapping**;**import org.springframework.web.bind.annotation.RestController**;**import java.util.concurrent.Future**;**@RestController  
@RequestMapping("tasks")  
public class DoTask {  
   
 @Autowired  
 private AsyncTask asyncTask**;** @RequestMapping("test1")  
 public String test1() throws Exception {  
   
 long start = System.*currentTimeMillis*()**;** Future<Boolean> a = asyncTask.doTask11()**;** Future<Boolean> b = asyncTask.doTask22()**;** Future<Boolean> c = asyncTask.doTask33()**;** while (!a.isDone() || !b.isDone() || !c.isDone()) {  
 if (a.isDone() && b.isDone() && c.isDone()) {  
 break**;** }  
 }  
   
 long end = System.*currentTimeMillis*()**;** String times = "任务全部完成，总耗时：" + (end - start) + "毫秒"**;** System.*out*.println(times)**;** return times**;** }  
}

## 12 拦截器

### （1）创建一个类继承WebMvcConfigurerAdapter

package com.zpr.config**;**import com.zpr.controller.interceptor.OneInterceptor**;**import com.zpr.controller.interceptor.TwoInterceptor**;**import org.springframework.context.annotation.Configuration**;**import org.springframework.web.servlet.config.annotation.InterceptorRegistry**;**import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter**;**@Configuration  
public class WebMvcConfigurer extends WebMvcConfigurerAdapter {  
  
 @Override  
 public void addInterceptors(InterceptorRegistry registry) {  
 */\*\*  
 \* 拦截器按照顺序执行  
 \*/* registry.addInterceptor(new OneInterceptor()).addPathPatterns("/one/\*\*")**;  
//** registry.addInterceptor(new TwoInterceptor()).addPathPatterns("/two/\*\*")  
// .addPathPatterns("/one/\*\*")**;** super.addInterceptors(registry)**;** }  
  
}

### （2）创建一个拦截器类

package com.zpr.controller.interceptor**;**import java.io.IOException**;**import java.io.OutputStream**;**import java.io.UnsupportedEncodingException**;**import javax.servlet.http.HttpServletRequest**;**import javax.servlet.http.HttpServletResponse**;**import com.zpr.pojo.IMoocJSONResult**;**import com.zpr.utils.JsonUtils**;**import org.slf4j.Logger**;**import org.slf4j.LoggerFactory**;**import org.springframework.web.servlet.HandlerInterceptor**;**import org.springframework.web.servlet.ModelAndView**;**public class OneInterceptor implements HandlerInterceptor {  
 final static Logger *log* = LoggerFactory.*getLogger*(TwoInterceptor.class)**;** */\*\*  
 \* 在请求处理之前进行调用（Controller方法调用之前）  
 \*/* @Override  
 public boolean preHandle(HttpServletRequest request**,** HttpServletResponse response**,** Object object) throws Exception {  
   
// System.out.println("被one拦截，放行...");  
 *log*.error("被one拦截，放行...")**;** return true**;** /\*if (true) {  
 returnErrorResponse(response, IMoocJSONResult.errorMsg("被one拦截..."));  
 }  
   
 return false;\*/  
 }  
   
 */\*\*  
 \* 请求处理之后进行调用，但是在视图被渲染之前（Controller方法调用之后）  
 \*/* @Override  
 public void postHandle(HttpServletRequest request**,** HttpServletResponse response**,** Object object**,** ModelAndView mv)  
 throws Exception {  
 // *TODO Auto-generated method stub* }  
   
 */\*\*  
 \* 在整个请求结束之后被调用，也就是在DispatcherServlet 渲染了对应的视图之后执行  
 \* （主要是用于进行资源清理工作）  
 \*/* @Override  
 public void afterCompletion(HttpServletRequest request**,** HttpServletResponse response**,** Object object**,** Exception ex)  
 throws Exception {  
 // *TODO Auto-generated method stub* }  
   
 public void returnErrorResponse(HttpServletResponse response**,** IMoocJSONResult result)  
 throws IOException**,** UnsupportedEncodingException {  
 OutputStream out=null**;** try{  
 response.setCharacterEncoding("utf-8")**;** response.setContentType("text/json")**;** out = response.getOutputStream()**;** out.write(JsonUtils.*objectToJson*(result).getBytes("utf-8"))**;** out.flush()**;** } finally{  
 if(out!=null){  
 out.close()**;** }  
 }  
 }  
}

### （3）创建一个控制类方法测试

@Controller  
@RequestMapping("one")  
public class OneController {  
  
 @RequestMapping("/index")  
 public String index(ModelMap map) {  
 map.addAttribute("name"**,** "imooc22")**;** return "thymeleaf/index"**;** }

}

### （4）日志显示

2019-02-01 08:32:50.497 ERROR 5860 --- [nio-8080-exec-1] c.z.c.interceptor.TwoInterceptor : 被one拦截，放行...

这里我没有拦截，你们可以试着把另一个注释去掉就可以看到拦截效果了。