## 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

（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**;** }  
}

## 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：

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

