表单验证：

客户端和服务器端的验证？

主要讲服务器端验证：

表单数据传输？？？

@PostMapping(**"/register"**)  
**public** String register(@RequestParam String username,  
 @RequestParam String password,  
 @RequestParam String email,  
 @RequestParam Integer phone){  
 User user = **new** User();  
 user.setUsername(username);  
 user.setPassword(password);  
 user.setEmail(email);  
 user.setPhone(phone);  
 **userRepository**.save(user);  
 **return "redirect:/login"**;  
}

参数太多 写起来很麻烦，如何优化？

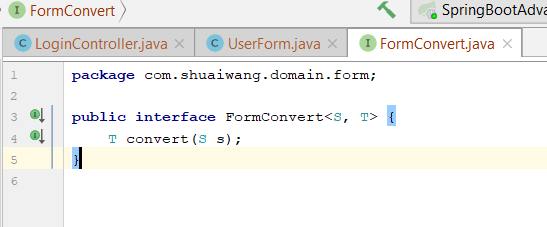
@PostMapping(**"/register"**)  
**public** String register(User user){  
 **userRepository**.save(user);  
 **return "redirect:/login"**;  
}

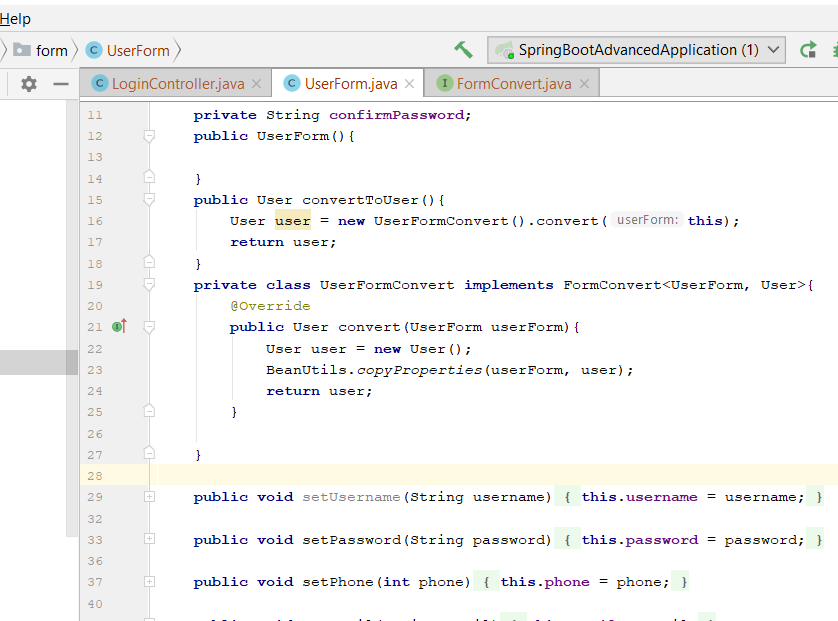
||

@PostMapping(**"/register"**)  
**public** String register(UserForm userForm){  
 User user = **new** User();  
 BeanUtils.*copyProperties*(userForm, user);  
 **userRepository**.save(user);  
 **return "redirect:/login"**;  
}

-----------------------\

@PostMapping(**"/register"**)  
**public** String register(UserForm userForm){  
 User user = userForm.convertToUser();  
 **userRepository**.save(user);  
 **return "redirect:/login"**;  
}



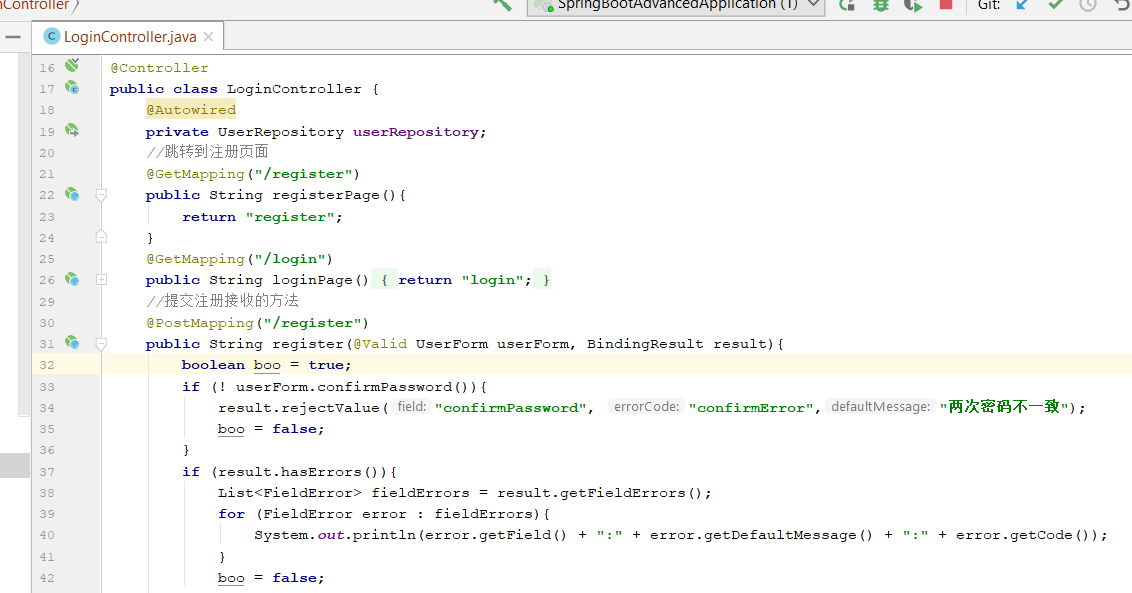


数据验证？？？

首先在模型层进行限制，@Blank,length,pattern, email等等



然后再controller层加入@Valid进行验证



BindingResult result=>esult.rejectValue(**"confirmPassword"**, **"confirmError"**,**"两次密码不一致"**);

=>List<FieldError> fieldErrors = result.getFieldErrors();  
**for** (FieldError error : fieldErrors){  
 System.***out***.println(error.getField() + **":"** + error.getDefaultMessage() + **":"** + error.getCode());  
}

错误处理：

如何把错误内容返回到前端？？？

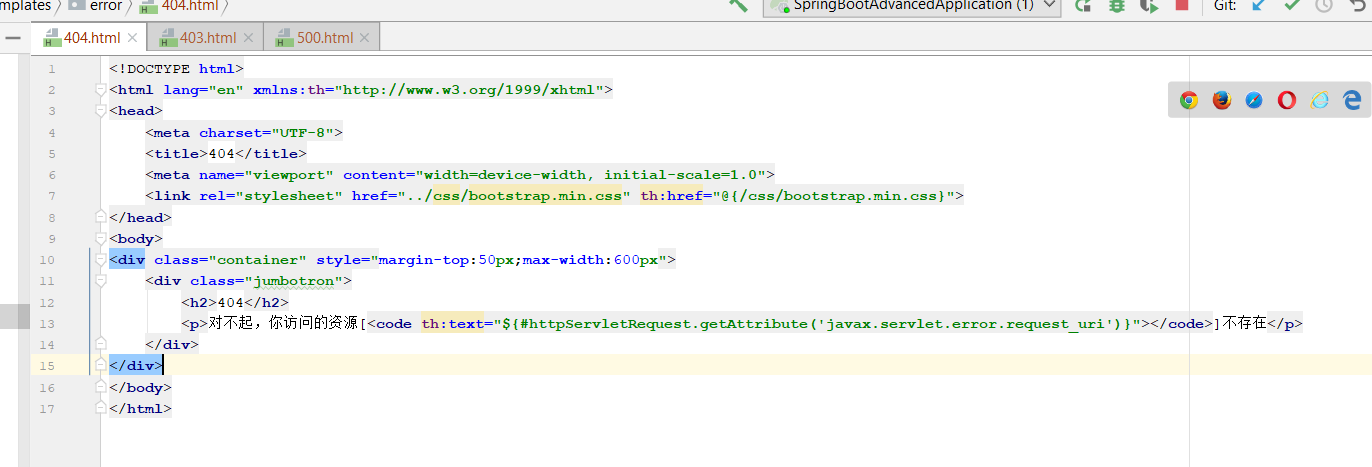
<!DOCTYPE **html**>  
<**html lang="en" xmlns:th="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**meta http-equiv="X-UA-Compatible" content="IE=edge"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1.0"**>  
 <**title**>register</**title**>  
 <**link rel="stylesheet" href="../static/css/bootstrap.min.css" th:href="@{/css/bootstrap.min.css}"**>  
</**head**>  
<**body**>  
<**div class="container" style="max-width**:600**px**;**margin-top**:50**px"**>  
 <**h3 class="page-header"**>注册</**h3**>  
 <**div class="well"**>  
 <**form action="/register" th:object="${userForm}" method="post"**>  
 <**div class="form-group"**>  
 <**label for="usernameId"**>用户名</**label**>  
 <**input type="text" name="username" id="usernameId" class="form-control" th:field="\*{username}"**>  
 <**p class="form-control-static text-danger" th:if="${#fields.hasErrors('username')}" th:errors="\*{username}"**>用户名不能为空</**p**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label for="passwordId"**>密码</**label**>  
 <**input type="password" name="password" id="passwordId" class="form-control" th:field="\*{password}"**>  
 <**p class="form-control-static text-danger" th:if="${#fields.hasErrors('password')}" th:errors="\*{password}"**></**p**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label for="confirmPasswordId"**>确认密码</**label**>  
 <**input type="password" name="confirmPassword" id="confirmPasswordId" class="form-control" th:field="\*{confirmPassword}"**>  
 <**p class="form-control-static text-danger" th:if="${#fields.hasErrors('confirmPassword')}" th:errors="\*{confirmPassword}"**></**p**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label for="emailId"**>邮箱</**label**>  
 <**input type="text" name="email" id="emailId" class="form-control" th:field="\*{email}"**>  
 <**p class="form-control-static text-danger" th:if="${#fields.hasErrors('email')}" th:errors="\*{email}"**>邮箱不能为空</**p**>  
 </**div**>  
 <**div class="form-group"**>  
 <**label for="phoneId"**>电话</**label**>  
 <**input type="text" name="phone" id="phoneId" class="form-control" th:field="\*{phone}"**>  
 <**p class="form-control-static text-danger" th:if="${#fields.hasErrors('phone')}" th:errors="\*{phone}"**>手机号不能为空</**p**>  
 </**div**>  
 <**p class="text-center"**>  
 <**button type="submit" class="btn btn-primary"**>注册</**button**>  
 </**p**>  
 </**form**>  
 </**div**>  
  
</**div**>  
*<!--<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>-->*<**script src="../static/js/jquery-3.2.1.min.js" th:href="@{/js/jquery-3.2.1.min.js}"**></**script**>  
<**script src="../static/js/bootstrap.min.js" th:href="@{/js/bootstrap.min.js}"**></**script**>  
</**body**>  
</**html**>

异常处理：

基于thymeleaf末班的异常处理

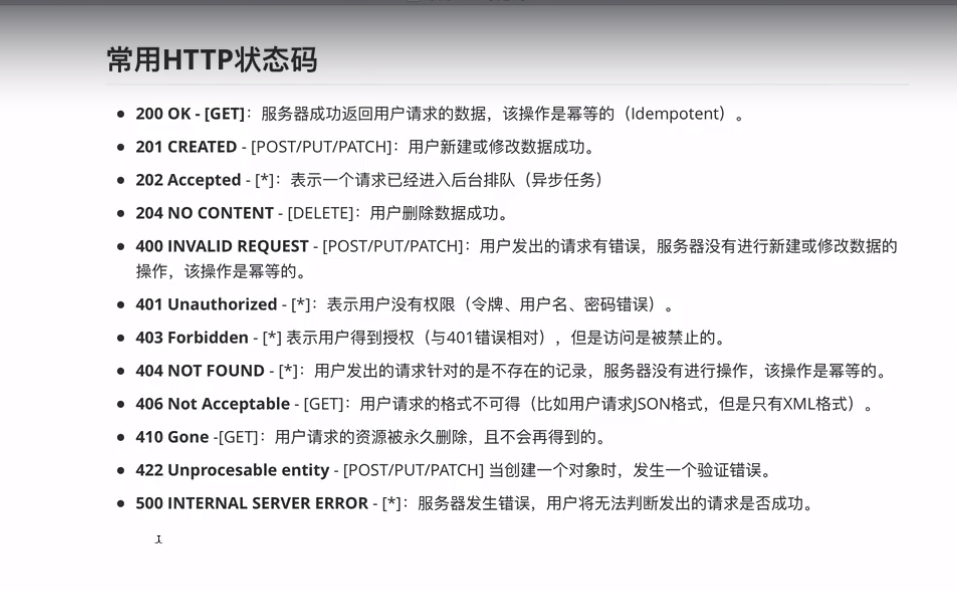
Restful API服务的异常处理？？？

在resource，templates文件夹下，建立error目录，加入404.html,即可customize 404错误页面



使用HTTP状态码进行异常处理

常用状态码：

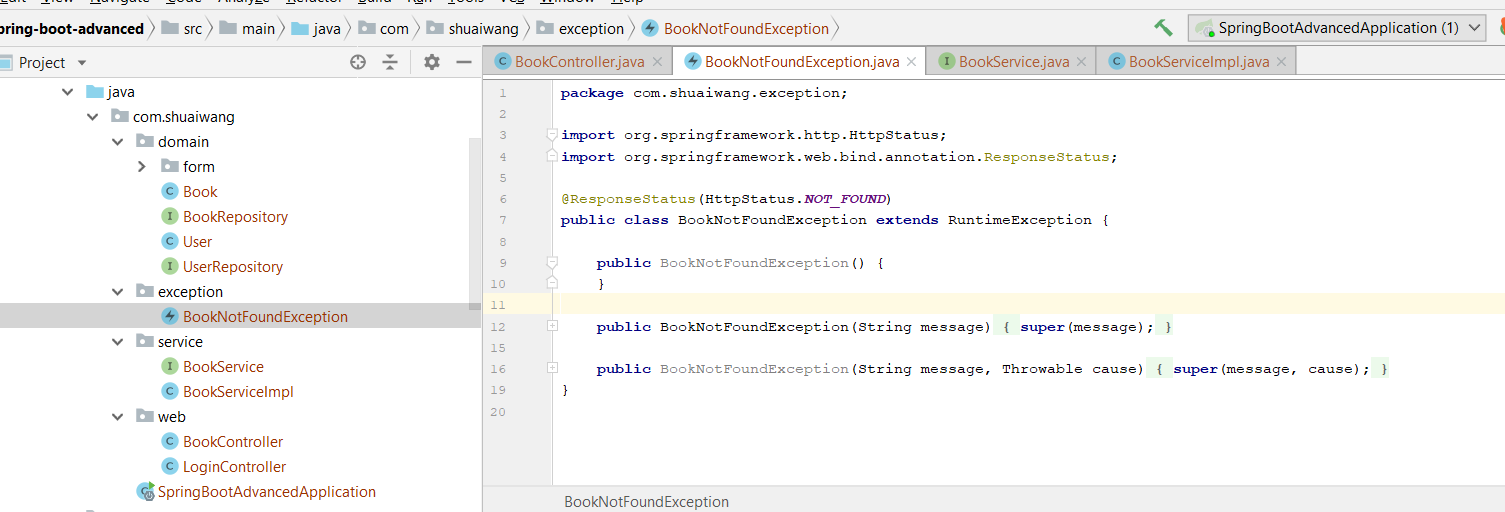


Spring boot 2.0以后，findById()会返回Optional<T>类型，可以通过isPresent()判断是否存在

在service层对Optional<T>进行判断如下：

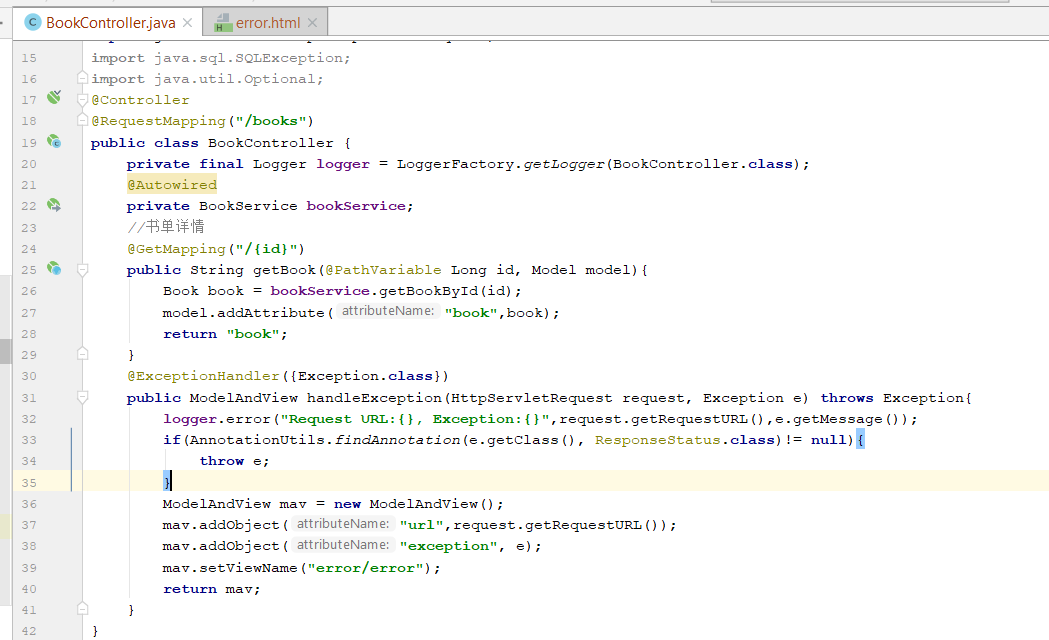
@Service  
**public class** BookServiceImpl **implements** BookService {  
 @Autowired  
 **private** BookRepository **bookRepository**;  
 @Override  
 **public** Book getBookById(Long id){  
 Optional<Book> o = **bookRepository**.findById(id);  
 **if** (! o.isPresent()){  
 **throw new** BookNotFoundException(**"书单不存在"**);  
 }  
 Book book = **bookRepository**.findById(id).get();  
 **return** book;  
 }  
}

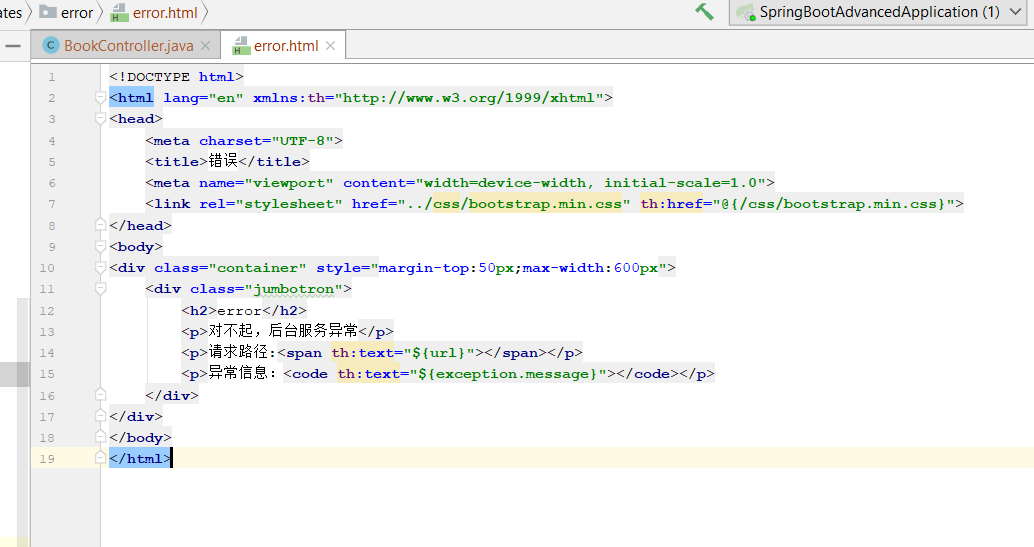
接着在com.shuaiwang包下新建exception文件夹，并使用@ResponseStatus



可以对产生的错误进行自定义状态码返回

在controller中处理异常？？？





浅显理解什么是ModelAndView???

Model是数据模型，例如book类，user类都是一个model，model具有属性，可以向表单注入model，即一个object，然后th:object=${book}=🡺接着th:name=”\*{book\_name}”这样使用，即向表单传入了一个object，就是一个model

View就是视图，可以是一个前端页面，比如一个表单，一个html文件等等。

ModelAndView mav = new ModelAndView();

Mav.addObject(url,”?”);

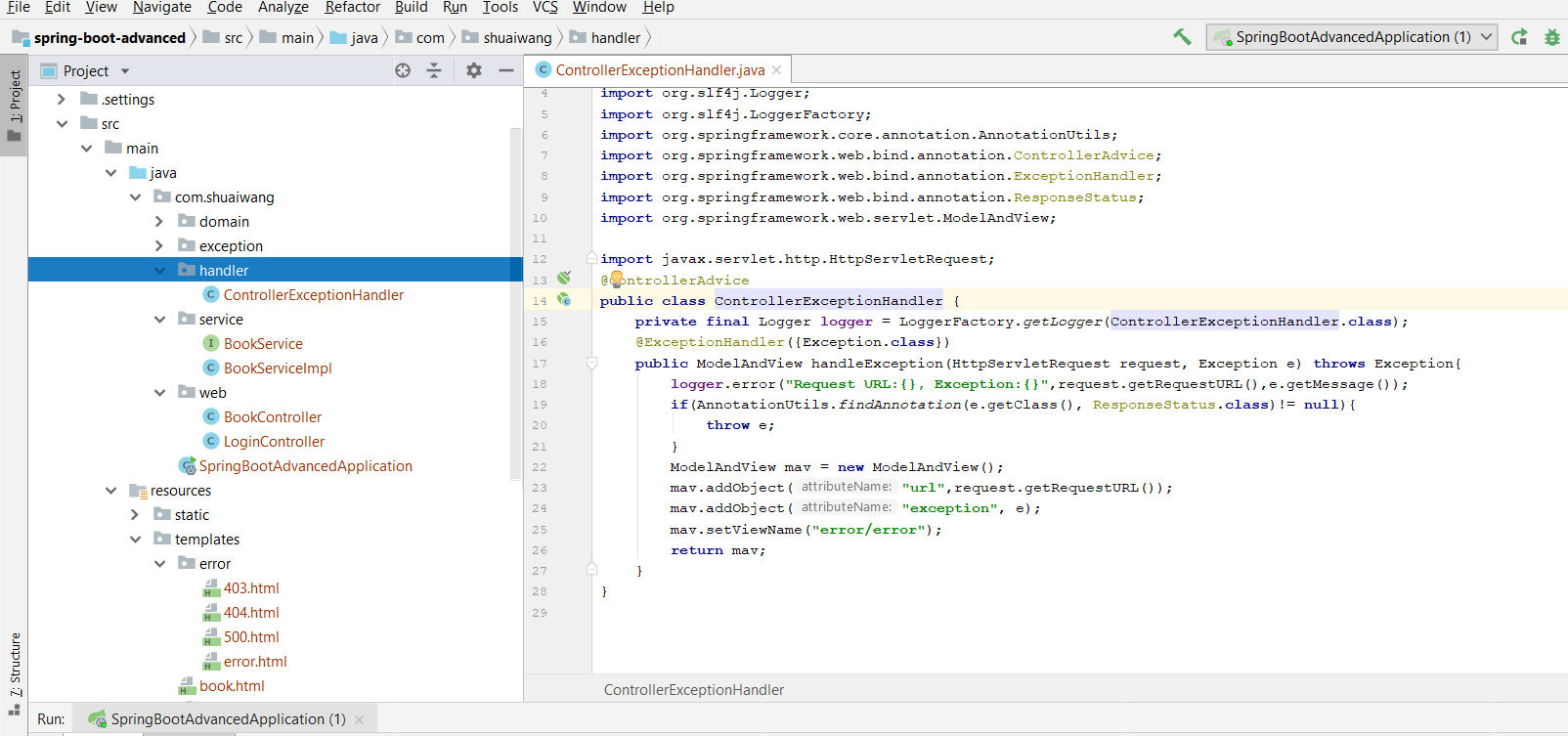
Mav.addObject(exception,”?”);

Mav.setViewName(“error/error”); => 设置返回的视图名称（error文件夹下的error.html文件）（包含url和exception两个object）

上面这种方法的exceptionhandler只能在单一controller中起作用

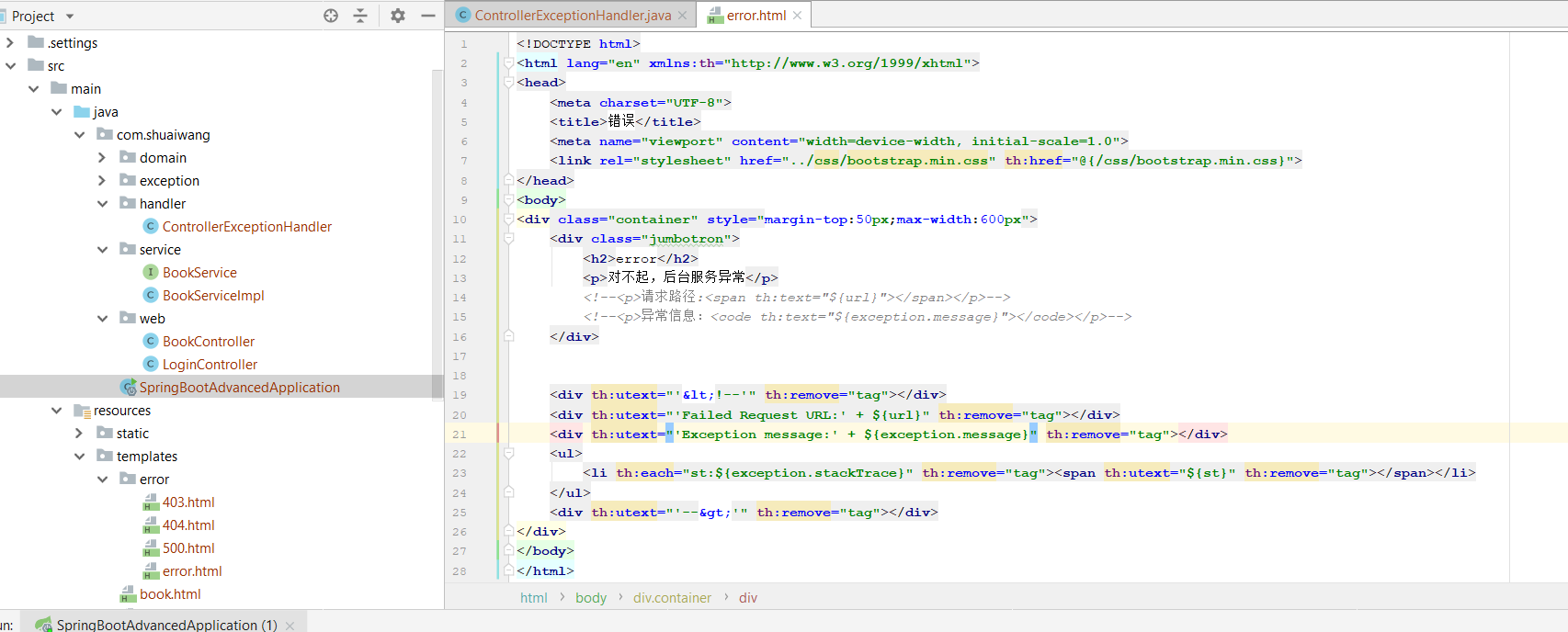
如何让异常处理全局统一起作用???

@ControllerAdvice 标记一个拦截器，所有的controller都会被他拦截，如果有异常，就会被catch到，进行全局统一处理。



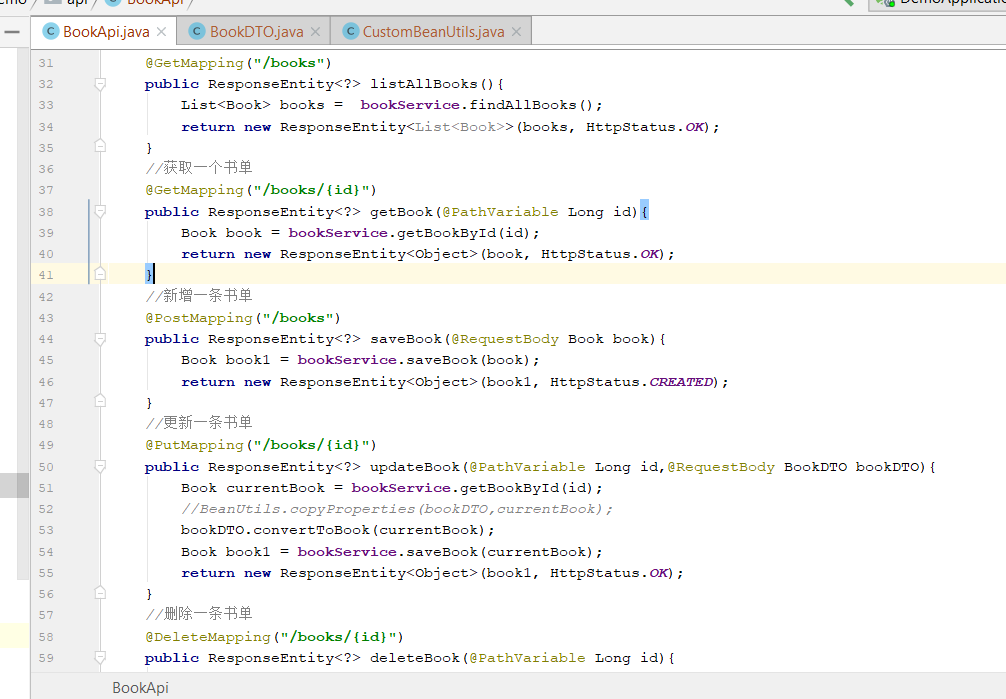
如何能再浏览器里隐藏堆栈信息，在源代码里显示？？？

使用th:utext=”” !!!!!!!!!!!!!!!!!!!!!!



Restful 服务的异常处理？？？

返回ResponseEntity!!!!!!!



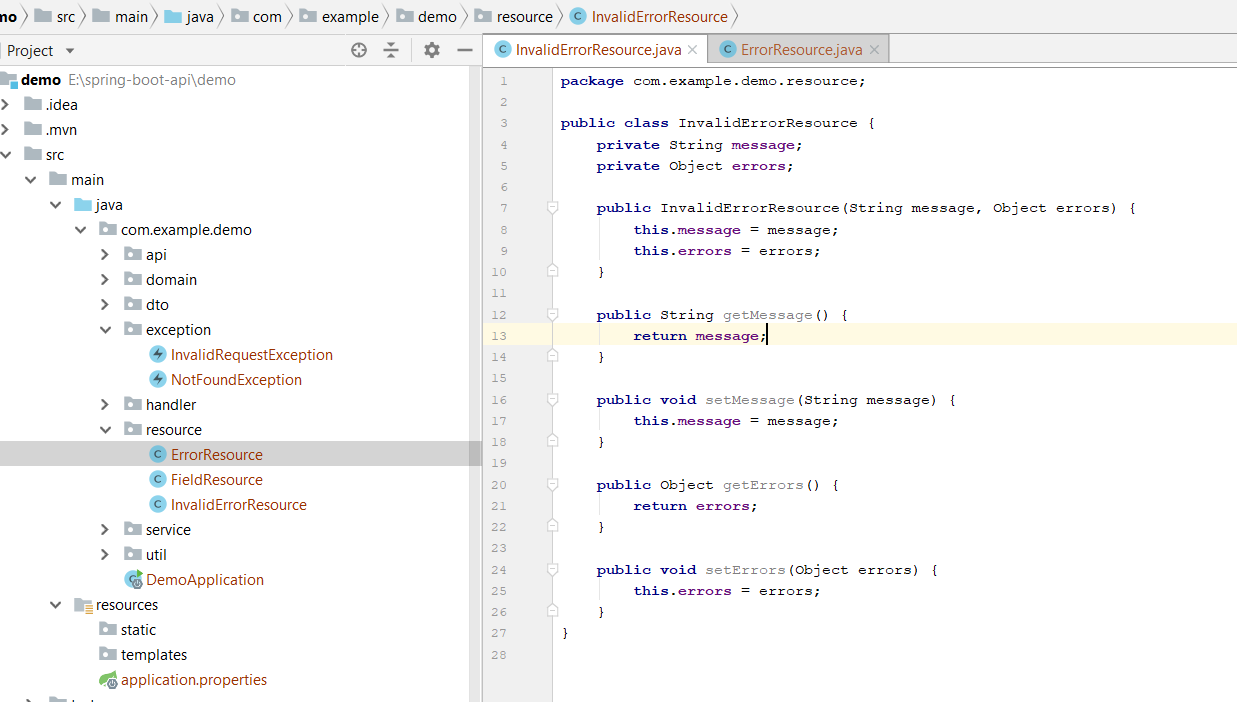
对bookDTO类型使用注解进行限制，[在book.api里使用@Valid](mailto:在book.api里使用@Valid)对请求输入BOOK参数进行参数检查，同时使用bindingResult.hasErrors(){

Throw new CustomizedException(“error message”);

}

Rest服务统一异常处理？？？

新建resource文件夹，定义各种类型异常的变量。



在handler文件夹下新建ApiExceptionHandler文件，使用

@RestControllerAdvice全局拦截异常，

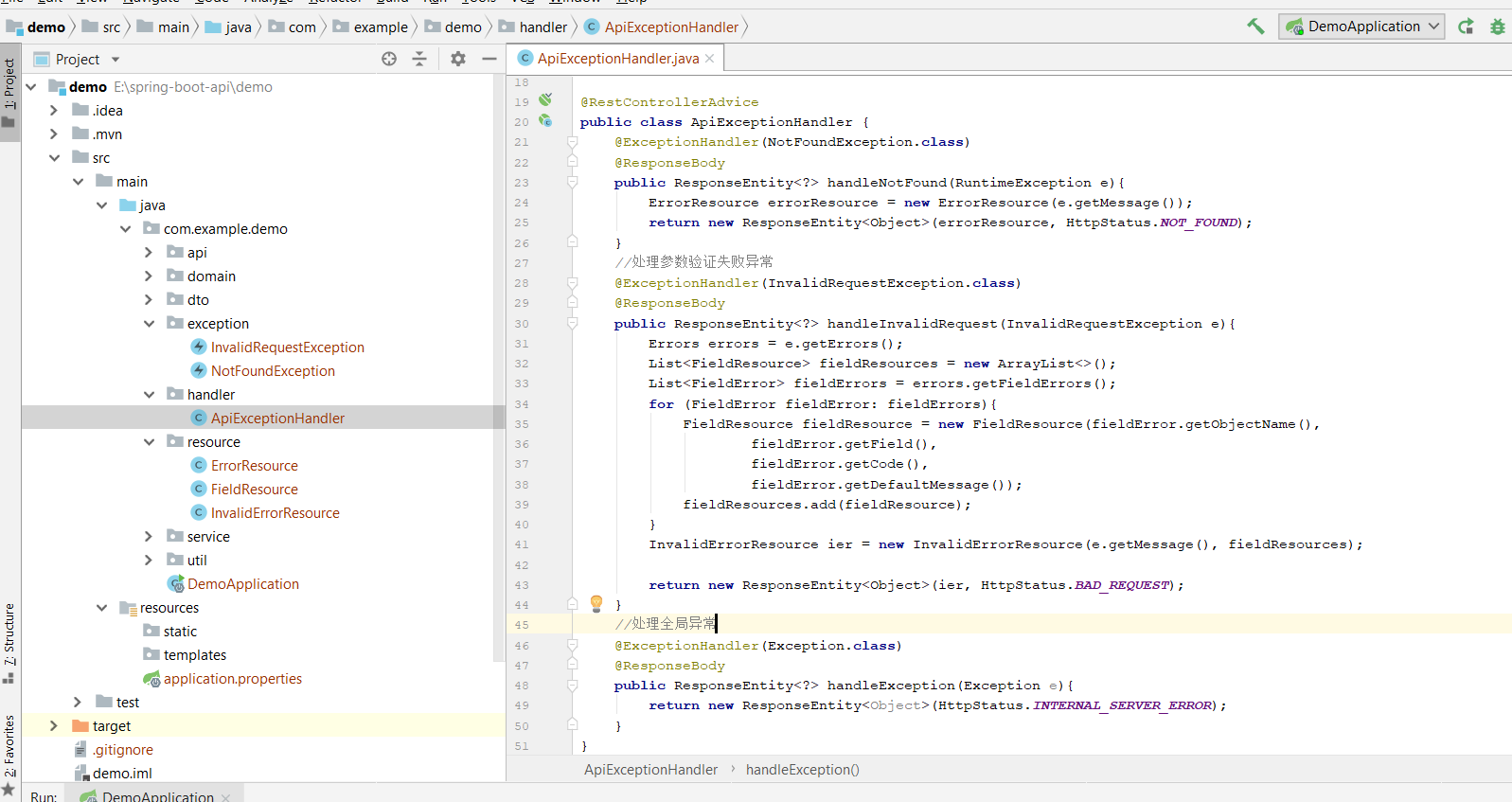
[使用@ExceptionHandler(Exception.class)](mailto:使用@ExceptionHandler(Exception.class))定义此方法拦截的异常，使用@ResponseBody返回Json格式，

使用ResponseEntity标记返回封装对象，Entity里需要放入对应异常（包括所有field）返回格式为Json格式，和对应的HttpStatus.code

404对应HttpStatus.NOT\_Found

参数验证失败对应HttpStatus.BAD\_REQUEST

500对应HttpStatus.INTERNAL\_SERVER\_ERROR



日志和AOP：控制台输出

控制台输出日志？？？

Spring boot提供了对常用日志的支持，如java util logging, log4j，log4j2，logback.默认使用logback记录日志。

Logback不需要任何配置

如何产生debug级别日志???

1.

Mvn package 首先打包，

然后使用java -jar \*\*\*.jar –debug运行输出debug级别日志。

2.

在yml或者property文件里使用debug:true打印debug级别的日志。



@RestController  
**public class** LogTestApi {  
 **private final** Logger **logger** = LoggerFactory.*getLogger*(**this**.getClass());  
 @GetMapping(**"/log"**)  
 **public** String log(){  
 **logger**.info(**"info --- log"**);  
 **logger**.warn(**"warn --- log"**);  
 **logger**.error(**"error --- log"**);  
 **logger**.debug(**"debug --- log"**);  
 **logger**.trace(**"trace --- log"**);  
 **return "logtest"**;  
 }  
}

日志级别配置：

Spring boot默认logback分为五种，

Trace<debug<info<warn<error

比如日志级别debug，不能输出trace级别

如何改变日志级别？？？

在properties文件中修改：

**logging.level.root**=**error  
logging.level.org.springframework.web**=**warn  
logging.level.com.example.demo**=**debug**

**使用spring.profiles.active=dev 即声明使用application-dev。Properties配置。**

文件输出日志

在application.properties里面写入

Logging.file.name=log/my.log

自定义配置日志：



如何自定义每一天生成一个日志？

加入logback-spring.xml在resource文件夹下！

*<?***xml version="1.0" encoding="UTF-8"** *?>*<**configuration**>  
 *<!--包含Spring boot对logback日志的默认配置-->* <**include resource="org/springframework/boot/logging/logback/defaults.xml"** />  
 <**property name="LOG\_FILE" value="${LOG\_FILE:-${LOG\_PATH:-${LOG\_TEMP:-${java.io.tmpdir:-/tmp}}}/spring.log}"**/>  
 <**include resource="org/springframework/boot/logging/logback/console-appender.xml"** />  
  
 *<!--重写了Spring Boot框架 org/springframework/boot/logging/logback/file-appender.xml 配置-->* <**appender name="TIME\_FILE"  
 class="ch.qos.logback.core.rolling.RollingFileAppender"**>  
 <**encoder**>  
 <**pattern**>${FILE\_LOG\_PATTERN}</**pattern**>  
 </**encoder**>  
 <**file**>${LOG\_FILE}</**file**>  
 <**rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy"**>  
 <**fileNamePattern**>${LOG\_FILE}.%d{yyyy-MM-dd}.%i</**fileNamePattern**>  
 *<!--保留历史日志一年的时间-->* <**maxHistory**>10</**maxHistory**>  
 *<!--  
 Spring Boot默认情况下，日志文件10M时，会切分日志文件,这样设置日志文件会在100M时切分日志  
 -->* <**timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP"**>  
 <**maxFileSize**>30KB</**maxFileSize**>  
 </**timeBasedFileNamingAndTriggeringPolicy**>  
 </**rollingPolicy**>  
 </**appender**>  
  
 <**root level="INFO"**>  
 <**appender-ref ref="CONSOLE"** />  
 <**appender-ref ref="TIME\_FILE"** />  
 </**root**>  
  
</**configuration**>  
*<!--  
 1、继承Spring boot logback设置（可以在appliaction.yml或者application.properties设置logging.\*属性）  
 2、重写了默认配置，设置日志文件大小在100MB时，按日期切分日志，切分后目录：  
  
 my.2017-08-01.0 80MB  
 my.2017-08-01.1 10MB  
 my.2017-08-02.0 56MB  
 my.2017-08-03.0 53MB  
 ......  
-->*

完全自定义配置logback???

Logback-custom.xml文件

AOP：切面编程

Aspect oriented programming

给程序定义一个切入点，在前后切入不同的执行内容

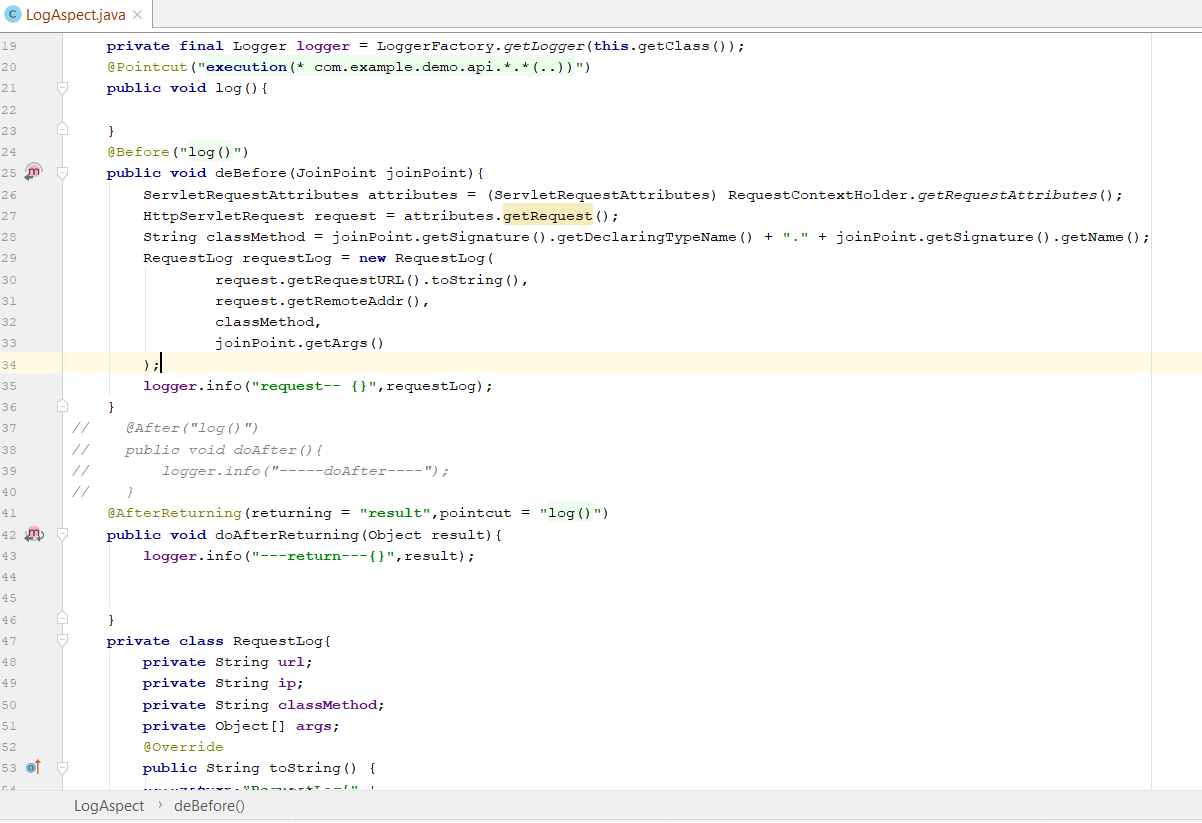
不会破坏原来的程序逻辑，因为只是定义方法前后的操作

AOP用于：日志记录，事务管理，安全检查，资源控制。

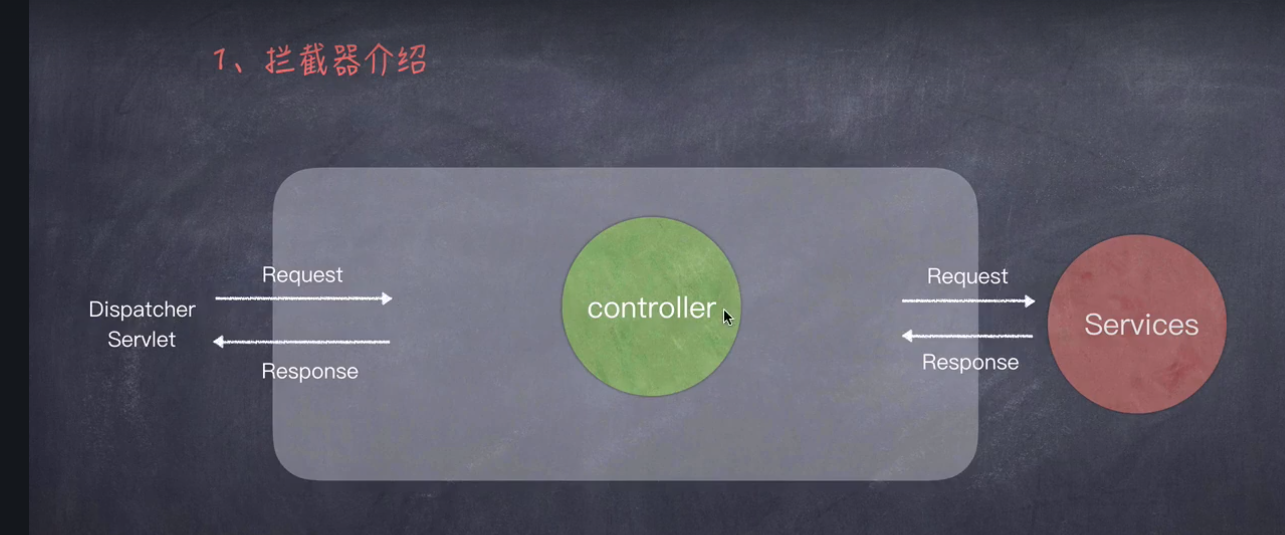
AOP统一记录日志：

请求：URL,IP,class,method,param,

返回内容：？



拦截器：



拦截器在dispatchservlet分发请求之后和返回响应之前进行额外的操作。

实现拦截器+注册拦截器！！！！！！

继承HandlerInterceptorAdapter，它是实现了HandleInterceptor的抽象类

实现拦截器：



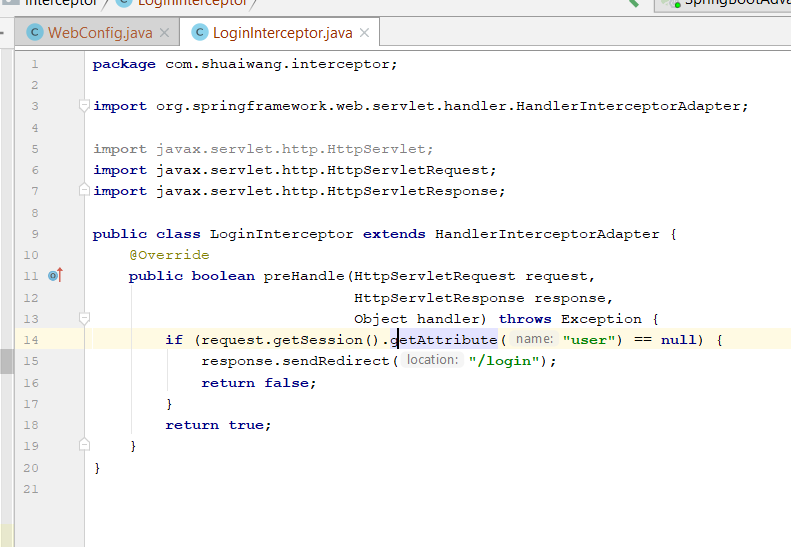


拦截器 登录实例：

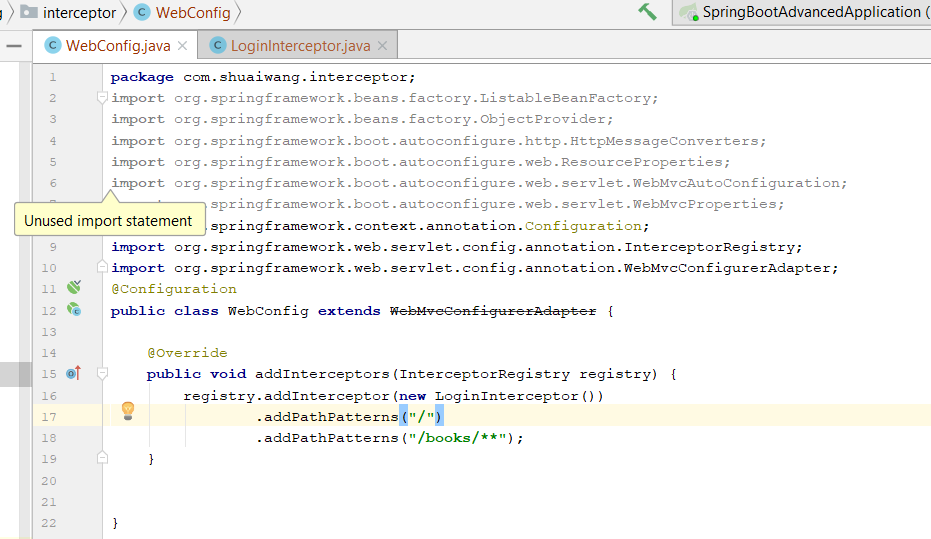
要求：

必须登录之后才能返回首页面和localhost：8080/books/1

实现拦截器



注册拦截器



里面写入拦截url。