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";
}
```

FormConvert

LoginController.java | UserForm.java | FormConvert.java

```java
package com.shuaiwang.domain.form;

public interface FormConvert<S, T> {
    T convert(S s);
}
```

Help

form > UserForm

SpringBootAdvancedApplication (1)

LoginController.java | UserForm.java | FormConvert.java

```java
    private String confirmPassword;
    public UserForm(){

    }
    public User convertToUser(){
        User user = new UserFormConvert().convert( userForm: this);
        return user;
    }
    private class UserFormConvert implements FormConvert<UserForm, User>{
        @Override
        public User convert(UserForm userForm){
            User user = new User();
            BeanUtils.copyProperties(userForm, user);
            return user;
        }

    }

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

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

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

数据验证？？？

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

LoginController.java × UserForm.java × FormConvert.java ×

```java
import org.hibernate.validator.constraints.Length;
import org.springframework.beans.BeanUtils;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.Pattern;

public class UserForm {
    public static final String PHONE_REG = "^((13[0-9])|(15[^4])|(18[0,2,3,5-9])|(17[0-8])|(147))\\d{8}$";
    @NotBlank
    private String username;
    @NotBlank
    @Length(min=6, message = "密码至少需要六位")
    private String password;
    @Pattern(regexp=PHONE_REG, message = "请输入正确手机号")
    private String phone;
    @Email
    private String email;
    @NotBlank
    private String confirmPassword;
    public UserForm(){

    }
    public boolean confirmPassword(){
        if (this.password.equals(this.confirmPassword)){
```

然后再 controller 层加入@Valid 进行验证

LoginController.java ×

```java
@Controller
public class LoginController {
    @Autowired
    private UserRepository userRepository;
    //跳转到注册页面
    @GetMapping("/register")
    public String registerPage(){
        return "register";
    }
    @GetMapping("/login")
    public String loginPage() { return "login"; }
    //提交注册接收的方法
    @PostMapping("/register")
    public String register(@Valid UserForm userForm, BindingResult result){
        boolean boo = true;
        if (! userForm.confirmPassword()){
            result.rejectValue( field: "confirmPassword",  errorCode: "confirmError", defaultMessage: "两次密码不一致");
            boo = false;
        }
        if (result.hasErrors()){
            List<FieldError> fieldErrors = result.getFieldErrors();
            for (FieldError error : fieldErrors){
                System.out.println(error.getField() + ":" + error.getDefaultMessage() + ":" + error.getCode());
            }
            boo = false;
```

```
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:600px;margin-top:50px">
    <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

```
mplates > error > 404.html >
404.html  403.html  500.html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>404</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../css/bootstrap.min.css" th:href="@{/css/bootstrap.min.css}">
</head>
<body>
<div class="container" style="margin-top:50px;max-width:600px">
    <div class="jumbotron">
        <h2>404</h2>
        <p>对不起，你访问的资源[<code th:text="${#httpServletRequest.getAttribute('javax.servlet.error.request_uri')}"></code>]不存在</p>
    </div>
</div>
</body>
</html>
```

使用 HTTP 状态码进行异常处理

常用状态码：

常用HTTP状态码

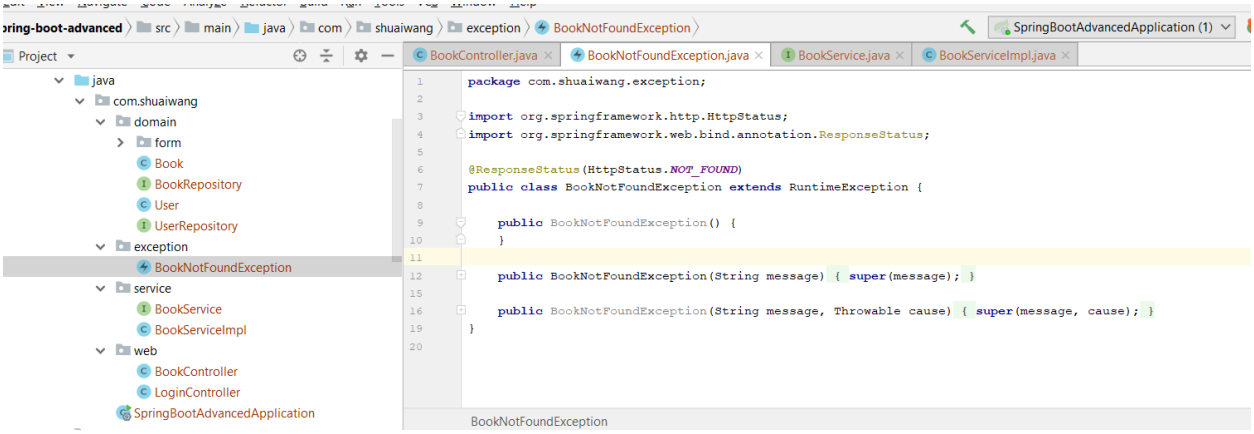
- **200 OK - [GET]**：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
- **201 CREATED** - [POST/PUT/PATCH]：用户新建或修改数据成功。
- **202 Accepted** - [*]：表示一个请求已经进入后台排队（异步任务）
- **204 NO CONTENT** - [DELETE]：用户删除数据成功。
- **400 INVALID REQUEST** - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。
- **401 Unauthorized** - [*]：表示用户没有权限（令牌、用户名、密码错误）。
- **403 Forbidden** - [*] 表示用户得到授权（与401错误相对），但是访问是被禁止的。
- **404 NOT FOUND** - [*]：用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的。
- **406 Not Acceptable** - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
- **410 Gone** -[GET]：用户请求的资源被永久删除，且不会再得到的。
- **422 Unprocesable entity** - [POST/PUT/PATCH] 当创建一个对象时，发生一个验证错误。
- **500 INTERNAL SERVER ERROR** - [*]：服务器发生错误，用户将无法判断发出的请求是否成功。

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？？？

BookController.java × error.html ×

```java
import java.sql.SQLException;
import java.util.Optional;
@Controller
@RequestMapping("/books")
public class BookController {
    private final Logger logger = LoggerFactory.getLogger(BookController.class);
    @Autowired
    private BookService bookService;
    //书单详情
    @GetMapping("/{id}")
    public String getBook(@PathVariable Long id, Model model){
        Book book = bookService.getBookById(id);
        model.addAttribute( attributeName: "book",book);
        return "book";
    }
    @ExceptionHandler({Exception.class})
    public ModelAndView handleException(HttpServletRequest request, Exception e) throws Exception{
        logger.error("Request URL:{}, Exception:{}",request.getRequestURL(),e.getMessage());
        if(AnnotationUtils.findAnnotation(e.getClass(), ResponseStatus.class)!= null){
            throw e;
        }
        ModelAndView mav = new ModelAndView();
        mav.addObject( attributeName: "url",request.getRequestURL());
        mav.addObject( attributeName: "exception", e);
        mav.setViewName("error/error");
        return mav;
    }
}
```

tes 〉 error 〉 error.html 〉 SpringBootAdvancedApplication (1)

BookController.java × error.html ×

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>错误</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="../css/bootstrap.min.css" th:href="@{/css/bootstrap.min.css}">
</head>
<body>
<div class="container" style="margin-top:50px;max-width:600px">
    <div class="jumbotron">
        <h2>error</h2>
        <p>对不起，后台服务异常</p>
        <p>请求路径:<span th:text="${url}"></span></p>
        <p>异常信息: <code th:text="${exception.message}"></code></p>
    </div>
</div>
</body>
</html>
```

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。

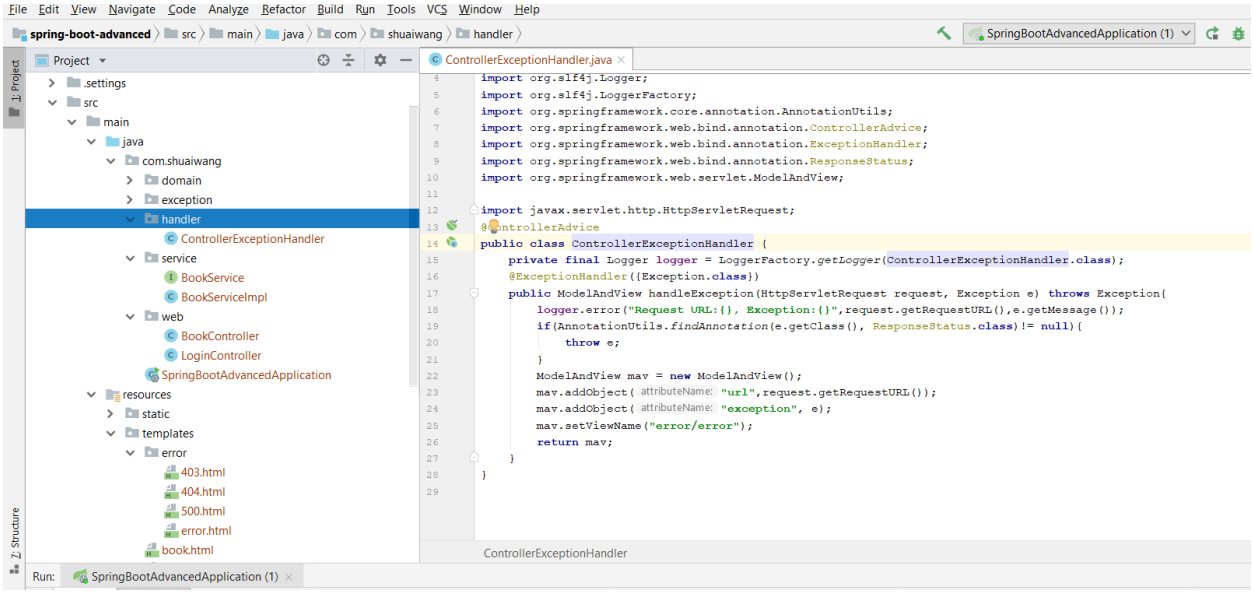
```
ModelAndView mav = new ModelAndView();
Mav.addObject(url,"?");
Mav.addObject(exception,"?");
Mav.setViewName("error/error");   => 设置返回的视图名称
```

（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

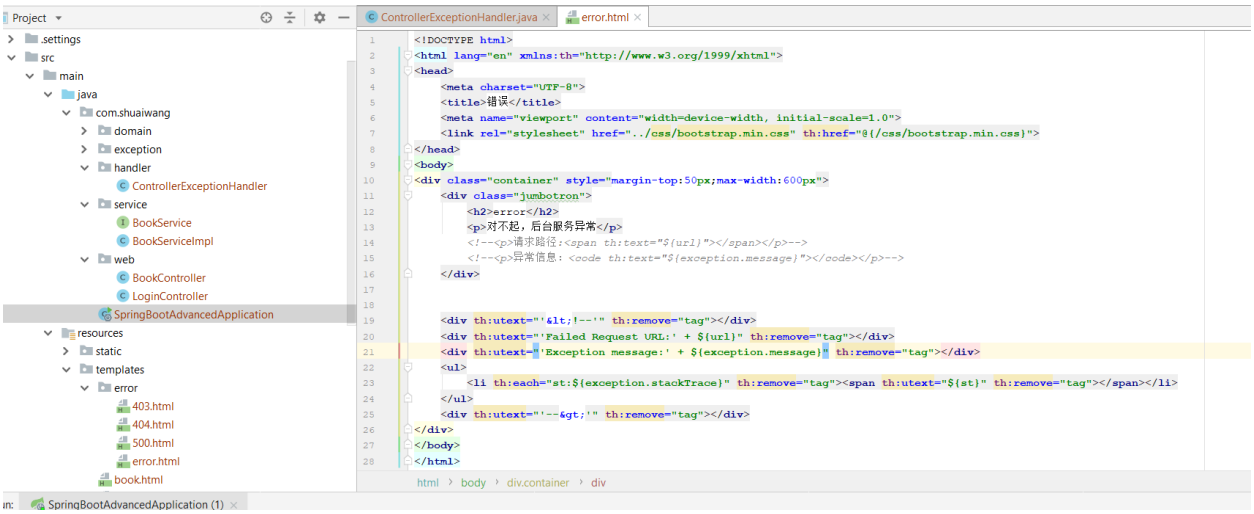
如何让异常处理全局统一起作用???

@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!!!!!!!

```java
@GetMapping("/books")
public ResponseEntity<?> listAllBooks(){
    List<Book> books =  bookService.findAllBooks();
    return new ResponseEntity<List<Book>>(books, HttpStatus.OK);
}
//获取一个书单
@GetMapping("/books/{id}")
public ResponseEntity<?> getBook(@PathVariable Long id){
    Book book = bookService.getBookById(id);
    return new ResponseEntity<Object>(book, HttpStatus.OK);
}
//新增一条书单
@PostMapping("/books")
public ResponseEntity<?> saveBook(@RequestBody Book book){
    Book book1 = bookService.saveBook(book);
    return new ResponseEntity<Object>(book1, HttpStatus.CREATED);
}
//更新一条书单
@PutMapping("/books/{id}")
public ResponseEntity<?> updateBook(@PathVariable Long id,@RequestBody BookDTO bookDTO){
    Book currentBook = bookService.getBookById(id);
    //BeanUtils.copyProperties(bookDTO,currentBook);
    bookDTO.convertToBook(currentBook);
    Book book1 = bookService.saveBook(currentBook);
    return new ResponseEntity<Object>(book1, HttpStatus.OK);
}
//删除一条书单
@DeleteMapping("/books/{id}")
public ResponseEntity<?> deleteBook(@PathVariable Long id){
```

对 bookDTO 类型使用注解进行限制，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

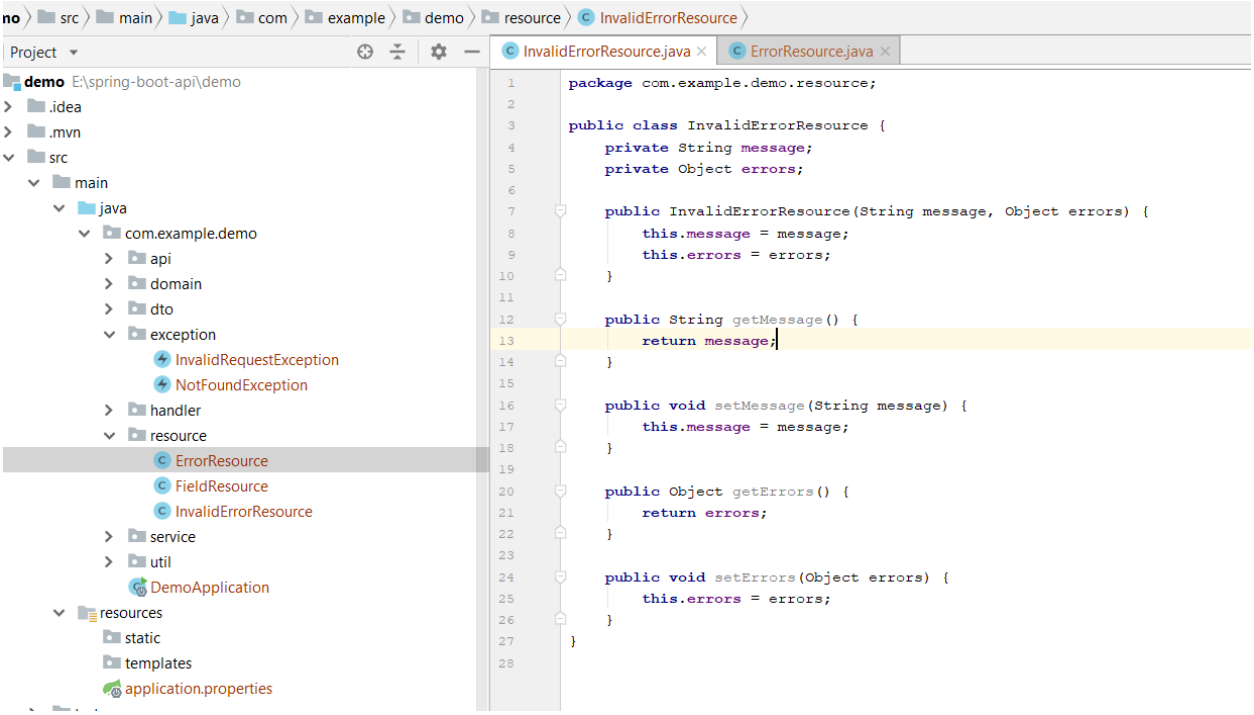
bindingResult.hasErrors(){

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，

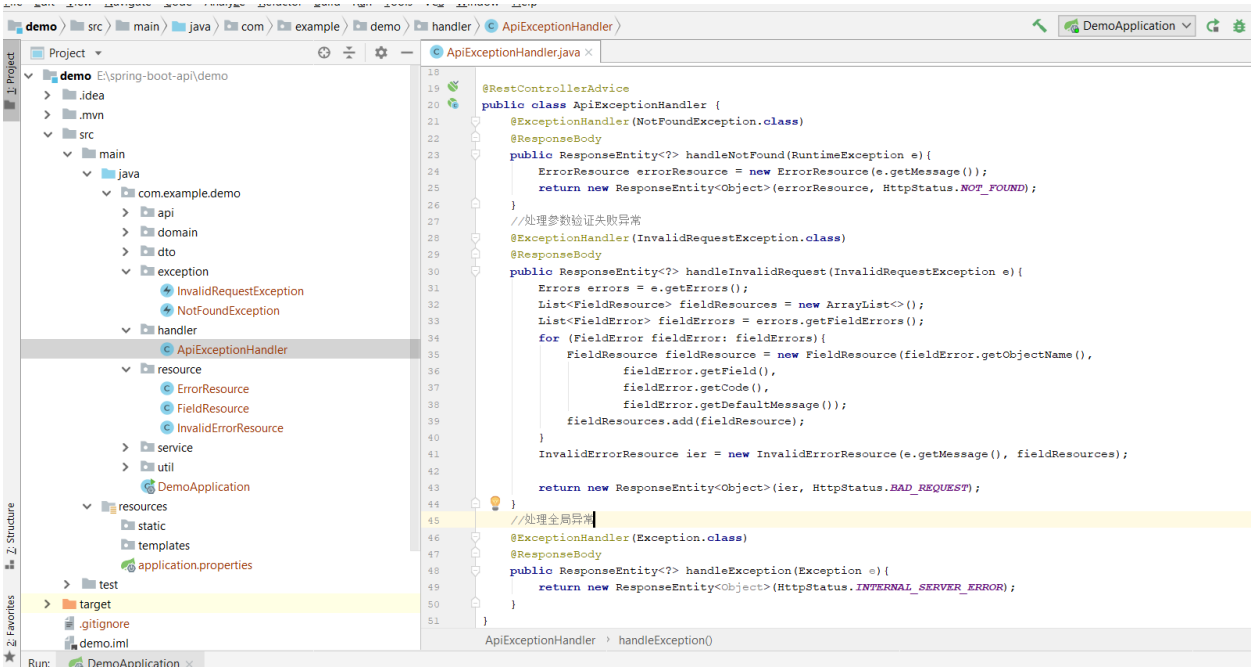
使用@ExceptionHandler(Exception.class)定义此方法拦截的异常，使用@ResponseBody 返回 Json 格式，

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。

2、日志级别

日志级别从低到高：

**`TRACE` < `DEBUG` < `INFO` < `WARN` < `ERROR`

```java
@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：

4、自定义配置

1、在 `application.properties` 或 `application.yml` 中配置：

配置属性	书名
`logging.config`	指定日志配置文件的位置
`logging.file`	指定日志文件，可以是相对路径，也可以是绝对路径
`logging.path`	指定日志文件存放目录
`logging.pattern.console`	指定在控制台输出的日志格式
`logging.pattern.file`	指定在日志文件保存的日志格式
`logging.pattern.level`	指定日志的级别
`logging.exception-conversion-word`	log异常时使用哪个格式转换器(`base.xml` 中定义了三个 `conversionRule`)
`logging.register-shutdown-hook=false`	系统启动时

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
<?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：？

LogAspect.java

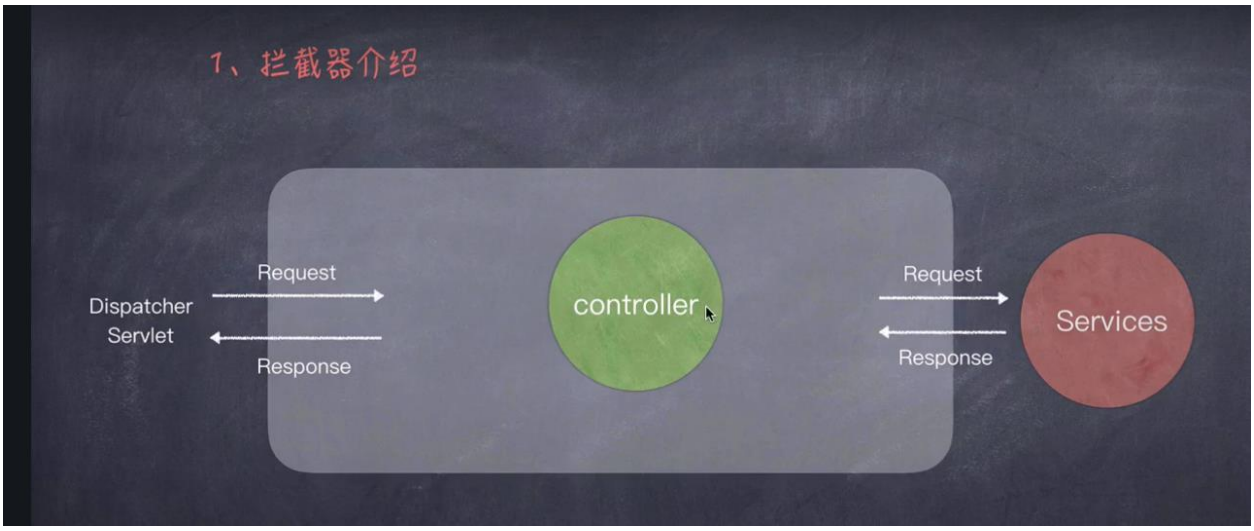
```java
    private final Logger logger = LoggerFactory.getLogger(this.getClass());
    @Pointcut("execution(* com.example.demo.api.*.*(..))")
    public void log(){

    }
    @Before("log()")
    public void deBefore(JoinPoint joinPoint){
        ServletRequestAttributes attributes = (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        HttpServletRequest request = attributes.getRequest();
        String classMethod = joinPoint.getSignature().getDeclaringTypeName() + "." + joinPoint.getSignature().getName();
        RequestLog requestLog = new RequestLog(
                request.getRequestURL().toString(),
                request.getRemoteAddr(),
                classMethod,
                joinPoint.getArgs()
        );
        logger.info("request-- {}",requestLog);
    }
//    @After("log()")
//    public void doAfter(){
//        logger.info("-----doAfter----");
//    }
    @AfterReturning(returning = "result",pointcut = "log()")
    public void doAfterReturning(Object result){
        logger.info("---return---{}",result);


    }
    private class RequestLog{
        private String url;
        private String ip;
        private String classMethod;
        private Object[] args;
        @Override
        public String toString() {
```

LogAspect › deBefore()

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Interceptor.md

拦截器Interceptor

1、实现拦截器

继承`HandlerInterceptorAdapter`， 它是一个实现了`HandlerInterceptor`的抽象类

```
public class LoginInterceptor extends HandlerInterceptorAdapter {

    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {
        // 在controller方法调用前打印信息
        System.out.println("拦截器");
        // 继续调用下一个拦截器
        return true;
    }
}
```

`HandlerInterceptor`接口

2、注册拦截器

```
@Configuration
public class WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LoginInterceptor())
                .addPathPatterns("/**").excludePathPatterns("/","/login");
    }
}
```

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

```java
package com.shuaiwang.interceptor;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LoginInterceptor extends HandlerInterceptorAdapter {
    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {
        if (request.getSession().getAttribute( name: "user") == null) {
            response.sendRedirect( location: "/login");
            return false;
        }
        return true;
    }
}
```

注册拦截器

```java
package com.shuaiwang.interceptor;
import org.springframework.beans.factory.ListableBeanFactory;
import org.springframework.beans.factory.ObjectProvider;
import org.springframework.boot.autoconfigure.http.HttpMessageConverters;
import org.springframework.boot.autoconfigure.web.ResourceProperties;
import org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration;
import org.springframework.boot.autoconfigure.web.servlet.WebMvcProperties;
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 WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LoginInterceptor())
                .addPathPatterns("/")
                .addPathPatterns("/books/**");
    }


}
```

里面写入拦截 url。