参看：https://blog.csdn.net/eson\_15/article/details/51731567

# 统一异常处理

## 概念

springmvc在处理请求过程中出现异常信息交由异常处理器进行处理，自定义异常处理器可以实现一个系统的异常处理逻辑。

一般项目中都需要作异常处理，基于系统架构的设计考虑，使用统一的异常处理方法。

## 异常处理思路

系统中异常包括两类：预期异常和运行时异常RuntimeException，

前者通过捕获异常从而获取异常信息，在代码手动处理异常可以try/catch捕获，可以向上抛出。后者主要通过规范代码开发、测试通过手段减少运行时异常的发生。

系统的dao、service、controller出现都通过throws Exception向上抛出，最后由springmvc前端控制器交由异常处理器进行异常处理，如下图：

http请求

DispatcherServlet

Handler(Controller)

service

mapper

左边的箭头就是请求处理的过程

有异常向上抛出

有异常向上抛出

有异常向上抛出

DispatcherServlet统一try/catch捕获异常调用异常处理器（全系统只有一个）

异常处理器需要实现HandlerExceptionResolver接口

要求dao、service、controller遇到异常全部向上抛出异常，方法向 上抛出异常throws Exception

# 自定义异常（统一异常处理解决方案）



## 自定义异常类

为了区别不同的异常,通常根据异常类型进行区分，这里我们创建一个自定义系统异常。

系统自定义的异常类是CustomException ，在controller方法中、service方法中手动抛出此类异常。针对系统自定义的CustomException异常，就可以直接从异常类中获取异常信息，将异常处理在错误页面展示。针对非CustomException异常，对这类重新构造成一个CustomException，异常信息为“未知错误”，此类错误需要在系统测试阶段去排除。

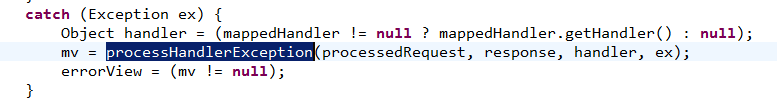
如果controller、service、dao抛出此类异常说明是系统预期处理的异常信息。



*/\*\*  
 \* <p>Description: 系统自定义的异常类型，实际开发中可能要定义多种异常类型</p>  
 \*/***public class** CustomException **extends** Exception {  
   
 *//异常信息* **private** String **message**;  
   
 **public** CustomException(String message){  
 **super**(message);  
 **this**.**message** = message;  
 }  
  
 **public** String getMessage() {  
 **return message**;  
 }  
  
 **public void** setMessage(String message) {  
 **this**.**message** = message;  
 }  
  
}

## 自定义异常处理器

前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常，进行异常处理。



*/\*\*  
 \* <p>Description: 自定义异常处理器</p>  
 \*/***public class** CustomExceptionResolver **implements** HandlerExceptionResolver {  
  
 *//前端控制器DispatcherServlet在进行HandlerMapping、调用HandlerAdapter执行Handler过程中，如果遇到异常就会执行此方法  
 //handler最终要执行的Handler，它的真实身份是HandlerMethod  
 //Exception ex就是接收到异常信息* @Override  
 **public** ModelAndView resolveException(HttpServletRequest request,  
 HttpServletResponse response, Object handler, Exception ex) {  
 *//输出异常* ex.printStackTrace();  
   
 *//统一异常处理代码  
 //针对系统自定义的CustomException异常，就可以直接从异常类中获取异常信息，将异常处理在错误页面展示  
 //异常信息* String message = **null**;  
 CustomException customException = **null**;  
 *//如果ex是系统 自定义的异常，直接取出异常信息* **if**(ex **instanceof** CustomException){  
 customException = (CustomException)ex;  
 }**else**{  
 *//针对非CustomException异常，对这类重新构造成一个CustomException，异常信息为“未知错误”* customException = **new** CustomException(**"未知错误"**);  
 }  
   
 *//错误 信息* message = customException.getMessage();  
   
 request.setAttribute(**"message"**, message);  
  
   
 **try** {  
 *//转向到错误 页面* request.getRequestDispatcher(**"/WEB-INF/jsp/error.jsp"**).forward(request, response);  
 } **catch** (ServletException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
   
 **return new** ModelAndView();  
 }  
}

## 异常处理器配置

在springmvc.xml中添加：

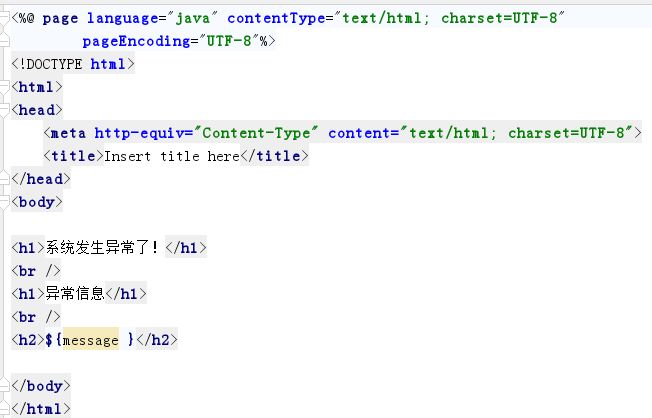


<!-- 配置全局异常处理器 -->

<bean

id=*"customHandleException"* class=*"cn.itcast.ssm.exception.CustomHandleException"*/>

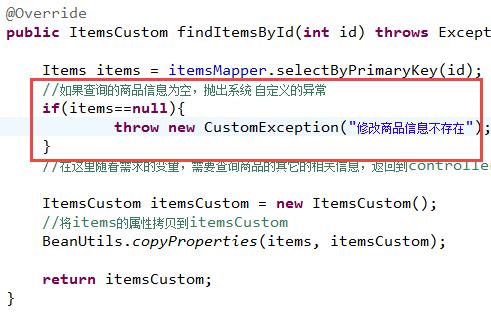
## 错误页面



<%@ **page language**="**java**" **contentType**="**text/html; charset=UTF-8**"  
 **pageEncoding**="**UTF-8**"%>  
<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=UTF-8"**>  
 <**title**>Insert title here</**title**>  
</**head**>  
<**body**>  
  
<**h1**>系统发生异常了！</**h1**>  
<**br** />  
<**h1**>异常信息</**h1**>  
<**br** />  
<**h2**>**${**message **}**</**h2**>  
  
</**body**>  
</**html**>

## 异常测试

可以在controller方法、service方法、dao实现类中抛出异常，要求dao、service、controller遇到异常全部向上抛出异常，方法向 上抛出异常throws Exception；修改ItemController方法“queryItemList”，抛出异常：



# 异常处理

## 异常处理

Spring MVC 通过 HandlerExceptionResolver 处理程序的异常，包括 Handler 映射、数据绑定以及目标方法执行时发生的异常。

SpringMVC 提供的 HandlerExceptionResolver 的实现类



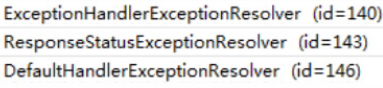
## HandlerExceptionResolver

DispatcherServlet 默认装配的 HandlerExceptionResolver：

没有使用 <mvc:annotation-driven/> 配置：



使用了 <mvc:annotation-driven/> 配置– ：



主要处理 Handler 中用 **@ExceptionHandler**注解定义的方法。

第一种方式：

@RequestMapping("/testExceptionHandlerExceptionResolver")

**public** String testExceptionHandlerExceptionResolver(@RequestParam("i") **int** i){

System.*out*.println("result: " + (10 / i));

**return** "success";

}

/\*\*

\* 1. 在 **@ExceptionHandler** 方法的入参中可以加入 Exception 类型的参数, 该参数即对应发生的异常对象

\* 2. **@ExceptionHandler** 方法的入参中不能传入 Map. 若希望把异常信息传导页面上, 需要使用 ModelAndView 作为返回值

\* 3. **@ExceptionHandler** 方法标记的异常有优先级的问题.

\* 4. **@ControllerAdvice**: 如果在当前 Handler 中找不到 **@ExceptionHandler** 方法来出来当前方法出现的异常,

\* 则将去 **@ControllerAdvice** 标记的类中查找 **@ExceptionHandler** 标记的方法来处理异常.

\*/

@ExceptionHandler({ArithmeticException.**class**})

**public** ModelAndView handleArithmeticException(Exception ex){

System.*out*.println("出异常了: " + ex);

ModelAndView mv = **new** ModelAndView("error");

mv.addObject("exception", ex);

**return** mv;

}

@ExceptionHandler 注解定义的方法优先级问题：

例如发生的是NullPointerException，但是声明的异常有RuntimeException 和 Exception，此候会根据异常的最近继承关系找到继承深度最浅的那个 @ExceptionHandler注解方法，即标记了 RuntimeException 的方法

ExceptionHandlerMethodResolver 内部若找不到@ExceptionHandler 注解的话，会找

**@ControllerAdvice** 中的**@ExceptionHandler** 注解方法

第二种方式（推荐）:

@ControllerAdvice

**public** **class** SpringMVCTestExceptionHandler {

@ExceptionHandler({ArithmeticException.**class**})

**public** ModelAndView handleArithmeticException(Exception ex){

System.*out*.println("----> 出异常了: " + ex);

ModelAndView mv = **new** ModelAndView("error");

mv.addObject("exception", ex);

**return** mv;

}

}

## ResponseStatusExceptionResolver

在异常及异常父类中找到 @ResponseStatus 注解，然后使用这个注解的属性进行处理。定义一个 @ResponseStatus 注解修饰的异常类

@ResponseStatus(value=HttpStatus.*FORBIDDEN*, reason="用户名密码不匹配!")

**public** **class** UserNameNotMatchPasswordException **extends** RuntimeException{

**private** **static** **final** **long** *serialVersionUID* = 1L;

}

若在处理器方法中抛出了上述异常：

若ExceptionHandlerExceptionResolver 不解析述异常。由于触发的异常 UnauthorizedException 带有@ResponseStatus注解。因此会被ResponseStatusExceptionResolver 解析到。最后响应HttpStatus.UNAUTHORIZED 代码给客户

端。HttpStatus.UNAUTHORIZED 代表响应码401，无权限。关于其他的响应码请参考 HttpStatus 枚举类型源码。

//@ResponseStatus(reason="测试",value=HttpStatus.NOT\_FOUND)

@RequestMapping("/testResponseStatusExceptionResolver")

**public** String testResponseStatusExceptionResolver(@RequestParam("i") **int** i){

**if**(i == 13){

**throw** **new** UserNameNotMatchPasswordException();

}

System.*out*.println("testResponseStatusExceptionResolver...");

**return** "success";

}

## DefaultHandlerExceptionResolver

对一些特殊的异常进行处理，比如NoSuchRequestHandlingMethodException、

HttpRequestMethodNotSupportedException、HttpMediaTypeNotSupportedException、HttpMediaTypeNotAcceptableException等。

@RequestMapping(value="/testDefaultHandlerExceptionResolver",method=RequestMethod.*POST*)

**public** String testDefaultHandlerExceptionResolver(){

System.*out*.println("testDefaultHandlerExceptionResolver...");

**return** "success";

}

## SimpleMappingExceptionResolver

如果希望对所有异常进行统一处理，可以使用

SimpleMappingExceptionResolver，它将异常类名映射为视图名，即发生异常时使用对应的视图报告异常

<!-- 配置使用 SimpleMappingExceptionResolver 来映射异常 -->

<bean class=*"org.springframework.web.servlet.handler.SimpleMappingExceptionResolver"*>

<property name=*"exceptionAttribute"* value=*"ex"*></property>

<property name=*"exceptionMappings"*>

<props>

<prop key=*"java.lang.ArrayIndexOutOfBoundsException"*>error</prop>

</props>

</property>

</bean>

@RequestMapping("/testSimpleMappingExceptionResolver")

**public** String testSimpleMappingExceptionResolver(@RequestParam("i") **int** i){

String [] vals = **new** String[10];

System.*out*.println(vals[i]);

**return** "success";

}

<h4>error page</h4>

${requestScope.ex }