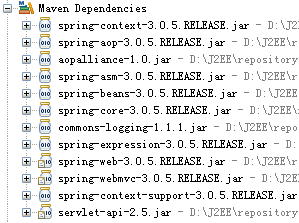
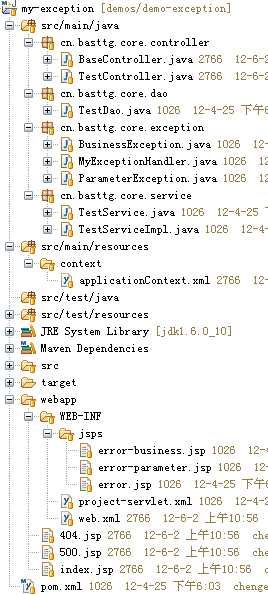
**[使用Spring MVC统一异常处理实战](http://cgs1999.iteye.com/blog/1547197)**

**博客分类：**

* [J2EE](http://cgs1999.iteye.com/category/222314)
* [Spring](http://cgs1999.iteye.com/category/222260)
* [Spring MVC](http://cgs1999.iteye.com/category/222262)
* [案例文档](http://cgs1999.iteye.com/category/233938)

[Spring](http://www.iteye.com/blogs/tag/Spring)[MVC](http://www.iteye.com/blogs/tag/MVC)[异常处理](http://www.iteye.com/blogs/tag/%E5%BC%82%E5%B8%B8%E5%A4%84%E7%90%86)

**1 描述**   
在J2EE项目的开发中，不管是对底层的数据库操作过程，还是业务层的处理过程，还是控制层的处理过程，都不可避免会遇到各种可预知的、不可预知的异常需要处理。每个过程都单独处理异常，系统的代码耦合度高，工作量大且不好统一，维护的工作量也很大。   
那么，能不能将所有类型的异常处理从各处理过程解耦出来，这样既保证了相关处理过程的功能较单一，也实现了异常信息的统一处理和维护？答案是肯定的。下面将介绍使用Spring MVC统一处理异常的解决和实现过程。   
  
**2 分析**   
Spring MVC处理异常有3种方式：   
（1）使用Spring MVC提供的简单异常处理器SimpleMappingExceptionResolver；   
（2）实现Spring的异常处理接口HandlerExceptionResolver 自定义自己的异常处理器；   
（3）使用@ExceptionHandler注解实现异常处理；   
  
**3 实战**   
**3.1 引言**   
为了验证Spring MVC的3种异常处理方式的实际效果，我们需要开发一个测试项目，从Dao层、Service层、Controller层分别抛出不同的异常，然后分别集成3种方式进行异常处理，从而比较3种方式的优缺点。   
  
**3.2 实战项目**   
**3.2.1 项目结构**   
  
  
**3.2.2 Dao层代码**

Java代码  [收藏代码](javascript:void())

1. @Repository("testDao")
2. public class TestDao {
3. public void exception(Integer id) throws Exception {
4. switch(id) {
5. case 1:
6. throw new BusinessException("12", "dao12");
7. case 2:
8. throw new BusinessException("22", "dao22");
9. case 3:
10. throw new BusinessException("32", "dao32");
11. case 4:
12. throw new BusinessException("42", "dao42");
13. case 5:
14. throw new BusinessException("52", "dao52");
15. default:
16. throw new ParameterException("Dao Parameter Error");
17. }
18. }
19. }

**3.2.3 Service层代码**

Java代码  [收藏代码](javascript:void())

1. public interface TestService {
2. public void exception(Integer id) throws Exception;
4. public void dao(Integer id) throws Exception;
5. }
7. @Service("testService")
8. public class TestServiceImpl implements TestService {
9. @Resource
10. private TestDao testDao;
12. public void exception(Integer id) throws Exception {
13. switch(id) {
14. case 1:
15. throw new BusinessException("11", "service11");
16. case 2:
17. throw new BusinessException("21", "service21");
18. case 3:
19. throw new BusinessException("31", "service31");
20. case 4:
21. throw new BusinessException("41", "service41");
22. case 5:
23. throw new BusinessException("51", "service51");
24. default:
25. throw new ParameterException("Service Parameter Error");
26. }
27. }
29. @Override
30. public void dao(Integer id) throws Exception {
31. testDao.exception(id);
32. }
33. }

**3.2.4 Controller层代码**

Java代码  [收藏代码](javascript:void())

1. @Controller
2. public class TestController {
3. @Resource
4. private TestService testService;
6. @RequestMapping(value = "/controller.do", method = RequestMethod.GET)
7. public void controller(HttpServletResponse response, Integer id) throws Exception {
8. switch(id) {
9. case 1:
10. throw new BusinessException("10", "controller10");
11. case 2:
12. throw new BusinessException("20", "controller20");
13. case 3:
14. throw new BusinessException("30", "controller30");
15. case 4:
16. throw new BusinessException("40", "controller40");
17. case 5:
18. throw new BusinessException("50", "controller50");
19. default:
20. throw new ParameterException("Controller Parameter Error");
21. }
22. }
24. @RequestMapping(value = "/service.do", method = RequestMethod.GET)
25. public void service(HttpServletResponse response, Integer id) throws Exception {
26. testService.exception(id);
27. }
29. @RequestMapping(value = "/dao.do", method = RequestMethod.GET)
30. public void dao(HttpServletResponse response, Integer id) throws Exception {
31. testService.dao(id);
32. }
33. }

**3.2.5 JSP页面代码**

Java代码  [收藏代码](javascript:void())

1. <%@ page contentType="text/html; charset=UTF-8"%>
2. <html>
3. <head>
4. <title>Maven Demo</title>
5. </head>
6. <body>
7. <h1>所有的演示例子</h1>
8. <h3>[url=./dao.do?id=1]Dao正常错误[/url]</h3>
9. <h3>[url=./dao.do?id=10]Dao参数错误[/url]</h3>
10. <h3>[url=./dao.do?id=]Dao未知错误[/url]</h3>

13. <h3>[url=./service.do?id=1]Service正常错误[/url]</h3>
14. <h3>[url=./service.do?id=10]Service参数错误[/url]</h3>
15. <h3>[url=./service.do?id=]Service未知错误[/url]</h3>

18. <h3>[url=./controller.do?id=1]Controller正常错误[/url]</h3>
19. <h3>[url=./controller.do?id=10]Controller参数错误[/url]</h3>
20. <h3>[url=./controller.do?id=]Controller未知错误[/url]</h3>

23. <h3>[url=./404.do?id=1]404错误[/url]</h3>
24. </body>
25. </html>

**3.3 集成异常处理**   
**3.3.1 使用SimpleMappingExceptionResolver实现异常处理**   
1、在Spring的配置文件applicationContext.xml中增加以下内容：

Xml代码  [收藏代码](javascript:void())

1. <bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
2. <!-- 定义默认的异常处理页面，当该异常类型的注册时使用 -->
3. <property name="defaultErrorView" value="error"></property>
4. <!-- 定义异常处理页面用来获取异常信息的变量名，默认名为exception -->
5. <property name="exceptionAttribute" value="ex"></property>
6. <!-- 定义需要特殊处理的异常，用类名或完全路径名作为key，异常也页名作为值 -->
7. <property name="exceptionMappings">
8. <props>
9. <prop key="cn.basttg.core.exception.BusinessException">error-business</prop>
10. <prop key="cn.basttg.core.exception.ParameterException">error-parameter</prop>
12. <!-- 这里还可以继续扩展对不同异常类型的处理 -->
13. </props>
14. </property>
15. </bean>

2、启动测试项目，经验证，Dao层、Service层、Controller层抛出的异常（业务异常BusinessException、参数 异常ParameterException和其它的异常Exception）都能准确显示定义的异常处理页面，达到了统一异常处理的目标。   
  
3、从上面的集成过程可知，使用SimpleMappingExceptionResolver进行异常处理，具有集成简单、有良好的扩展性、对已有代码没有入侵性等优点，但该方法仅能获取到异常信息，若在出现异常时，对需要获取除异常以外的数据的情况不适用。   
  
**3.3.2 实现HandlerExceptionResolver 接口自定义异常处理器**   
1、增加HandlerExceptionResolver 接口的实现类MyExceptionHandler，代码如下：

Java代码  [收藏代码](javascript:void())

1. public class MyExceptionHandler implements HandlerExceptionResolver {
3. public ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,
4. Exception ex) {
5. Map<String, Object> model = new HashMap<String, Object>();
6. model.put("ex", ex);
8. // 根据不同错误转向不同页面
9. if(ex instanceof BusinessException) {
10. return new ModelAndView("error-business", model);
11. }else if(ex instanceof ParameterException) {
12. return new ModelAndView("error-parameter", model);
13. } else {
14. return new ModelAndView("error", model);
15. }
16. }
17. }

2、在Spring的配置文件applicationContext.xml中增加以下内容：

Xml代码  [收藏代码](javascript:void())

1. <bean id="exceptionHandler" class="cn.basttg.core.exception.MyExceptionHandler"/>

3、启动测试项目，经验证，Dao层、Service层、Controller层抛出的异常（业务异常BusinessException、参数 异常ParameterException和其它的异常Exception）都能准确显示定义的异常处理页面，达到了统一异常处理的目标。   
  
4、从上面的集成过程可知，使用实现HandlerExceptionResolver接口的异常处理器进行异常处理，具有集成简单、有良好的扩展性、对已有代码没有入侵性等优点，同时，在异常处理时能获取导致出现异常的对象，有利于提供更详细的异常处理信息。   
  
**3.3.3 使用@ExceptionHandler注解实现异常处理**   
1、增加BaseController类，并在类中使用@ExceptionHandler注解声明异常处理，代码如下：

Java代码  [收藏代码](javascript:void())

1. public class BaseController {
2. /\*\* 基于@ExceptionHandler异常处理 \*/
3. @ExceptionHandler
4. public String exp(HttpServletRequest request, Exception ex) {
6. request.setAttribute("ex", ex);
8. // 根据不同错误转向不同页面
9. if(ex instanceof BusinessException) {
10. return "error-business";
11. }else if(ex instanceof ParameterException) {
12. return "error-parameter";
13. } else {
14. return "error";
15. }
16. }
17. }

2、修改代码，使所有需要异常处理的Controller都继承该类，如下所示，修改后的TestController类继承于BaseController：

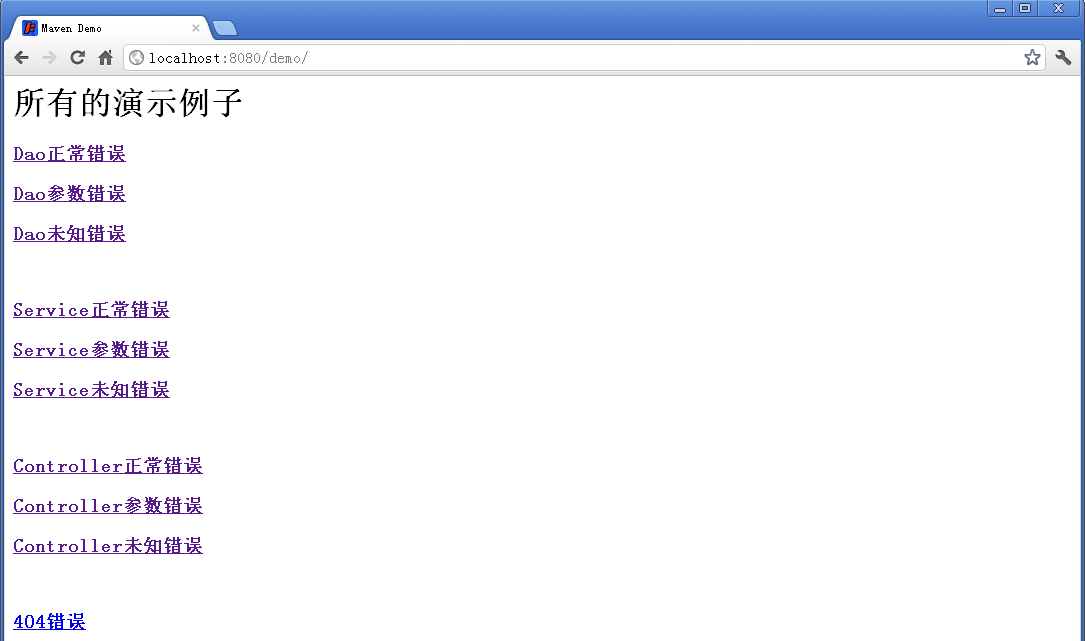
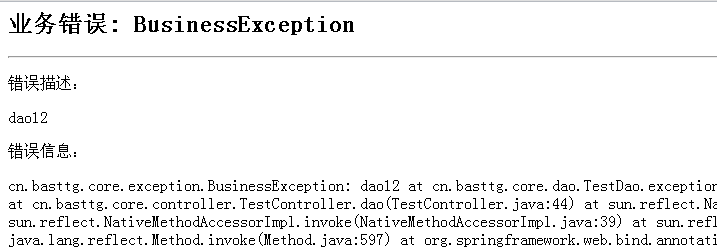
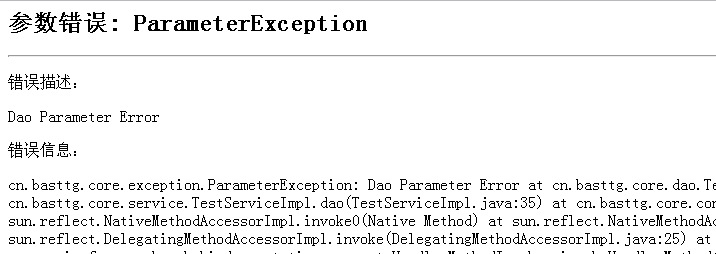
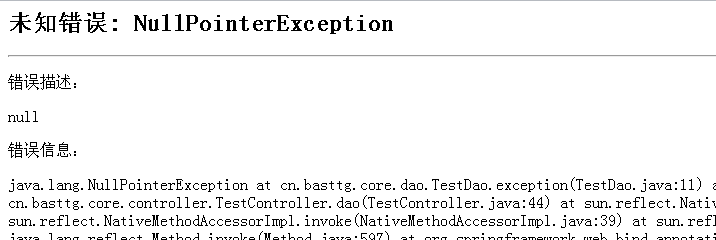
Java代码  [收藏代码](javascript:void())

1. public class TestController extends BaseController

3、启动测试项目，经验证，Dao层、Service层、Controller层抛出的异常（业务异常BusinessException、参数 异常ParameterException和其它的异常Exception）都能准确显示定义的异常处理页面，达到了统一异常处理的目标。   
  
4、从上面的集成过程可知，使用@ExceptionHandler注解实现异常处理，具有集成简单、有扩展性好（只需要将要异常处理的 Controller类继承于BaseController即可）、不需要附加Spring配置等优点，但该方法对已有代码存在入侵性（需要修改已有代 码，使相关类继承于BaseController），在异常处理时不能获取除异常以外的数据。   
  
**3.4 未捕获异常的处理**   
对于Unchecked Exception而言，由于代码不强制捕获，往往被忽略，如果运行期产生了Unchecked Exception，而代码中又没有进行相应的捕获和处理，则我们可能不得不面对尴尬的404、500……等服务器内部错误提示页面。   
我们需要一个全面而有效的异常处理机制。目前大多数服务器也都支持在Web.xml中通过<error-page> (Websphere/Weblogic)或者<error-code>(Tomcat)节点配置特定异常情况的显示页面。修改 web.xml文件，增加以下内容：

Xml代码  [收藏代码](javascript:void())

1. <!-- 出错页面定义 -->
2. <error-page>
3. <exception-type>java.lang.Throwable</exception-type>
4. <location>/500.jsp</location>
5. </error-page>
6. <error-page>
7. <error-code>500</error-code>
8. <location>/500.jsp</location>
9. </error-page>
10. <error-page>
11. <error-code>404</error-code>
12. <location>/404.jsp</location>
13. </error-page>
15. <!-- 这里可继续增加服务器错误号的处理及对应显示的页面 -->

**4 解决结果**   
1、运行测试项目显示的首页，如下图所示：   
  
  
2、业务错误显示的页面，如下图所示：   
  
  
3、参数错误显示的页面，如下图所示：   
  
  
4、未知错误显示的页面，如下图所示：   
  
  
5、服务器内部错误页面，如下图所示：   
  
  
**5 总结**   
综合上述可知，Spring MVC集成异常处理3种方式都可以达到统一异常处理的目标。从3种方式的优缺点比较，若只需要简单的集成异常处理，推荐使用 SimpleMappingExceptionResolver即可；若需要集成的异常处理能够更具个性化，提供给用户更详细的异常信息，推荐自定义实现 HandlerExceptionResolver接口的方式；若不喜欢Spring配置文件或要实现“零配置”，且能接受对原有代码的适当入侵，则建议 使用@ExceptionHandler注解方式。   
  
**6 源代码**   
源代码项目如下所示，为Maven项目，若需运行，请自行获取相关的依赖包。   
[点击这里获取源代码](http://dl.iteye.com/topics/download/6685e62d-b02d-3217-8c45-c4e1e8959928)   
  
**7 参考资料**   
[1] Spring MVC统一处理异常的方法   
http://hi.baidu.com/99999999hao/blog/item/25da70174bfbf642f919b8c3.html   
[2] SpringMVC 异常处理初探   
http://exceptioneye.iteye.com/blog/1306150   
[3] Spring3 MVC 深入研究   
http://elf8848.iteye.com/blog/875830   
[4] Spring MVC异常处理   
http://blog.csdn.net/rj042/article/details/7380442