Spring MVC 统一异常处理实战

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

```
📸 my-exception [demos/demo-exception]
🚊 👍 src/main/java
  😑 🖶 cn. basttg. core. controller
     🗓 🖟 🚹 BaseController. java 2766 12-6-2
     🖮 🖟 TestController. java 2766 12-6-2
  🚊 🚠 cn. basttg. core. dao
     🗓 🖟 TestDao.java 1026 12-4-25 下午6
  🖮 🯭 cn. basttg. core. exception
     🖮 🖟 BusinessException. java 1026 12-
     🗓 🖟 🚹 MyExceptionHandler. java 1026 - 12
     🗓 🖟 ParameterException.java 1026 12
  🖃 🖶 cn. basttg. core. service
     🗓 🖟 🊹 TestService. java 1026 - 12-4-25 -
     🖮 🖟 TestServiceImpl.java 1026 12-4-
🚊 👍 src/main/resources
  in text
     🌁 src/test/java

<u>→</u> # src/test/resources

→ Marian JRE System Library [jdk1.6.0_10]

표 ៉ Maven Dependencies
🗓 🛅 src
표 🗁 target
🖃 🚰 webapp
  😑 🗁 WEB-INF
     🖃 🗁 jsps
        ---🔝 error-parameter.jsp 1026 - 12-
        📑 500. jsp 2766 12-6-2 上午10:56 che
     📑 index.jsp 2766 12-6-2 上午10:56 🤞
  🌇 pom.xml 1026 12-4-25 下午6:03 chenge
🖃 🕍 Maven Dependencies

    im ing context=3.0.5. RELEASE. jar = D:\J2.

  i om spring-aop-3.0.5. RELEASE. jar - D:\J2EE\r
  🖮 🚾 spring-asm-3.0.5. RELEASE. jar - D:\J2EE\r
  i 🚾 spring-beans-3.0.5. RELEASE. jar - D:\J2EE
  i spring-core-3.0.5. RELEASE.jar - D:\J2EE\:
  i 🚾 spring-expression-3.0.5. RELEASE.jar - D:
  료 \overline a spring-web-3.0.5. RELEASE. jar - D:\J2EE\r
  🕕 🚮 spring-webmvc-3.0.5. RELEASE, jar - D:\J2E
  ɨ 🚾 spring-context-support-3.0.5. RELEASE. jar
```

3.2.2 Dao 层代码

Java 代码 ☆

@Repository("testDao")

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

3.2.3 Service 层代码

Java 代码 ☆

```
1. public interface TestService {
      public void exception(Integer id) throws Exception;
2.
3.
4.
      public void dao(Integer id) throws Exception;
5. }
6.
7. @Service("testService")
8. public class TestServiceImpl implements TestService {
9.
      @Resource
10.
       private TestDao testDao;
11.
       public void exception(Integer id) throws Exceptio
12.
 n {
           switch(id) {
13.
           case 1:
14.
15.
               throw new BusinessException("11", "service11
 ");
16.
           case 2:
```

```
throw new BusinessException("21", "service21
17.
  ");
18.
            case 3:
                throw new BusinessException("31", "service31
19.
  ");
            case 4:
20.
                throw new BusinessException("41", "service41
21.
  ");
            case 5:
22.
                throw new BusinessException("51", "service51
23.
  ");
            default:
24.
25.
                throw new ParameterException("Service Parame
  ter Error");
26.
            }
27.
        }
28.
       @Override
29.
30.
        public void dao(Integer id) throws Exception {
31.
            testDao.exception(id);
32.
        }
33. }
```

3.2.4 Controller 层代码

Java 代码 ☆

```
1. @Controller
2. public class TestController {
3.
      @Resource
4.
      private TestService testService;
5.
      @RequestMapping(value = "/controller.do", method = Re
6.
  questMethod.GET)
      public void controller(HttpServletResponse respons
7.
  e, Integer id) throws Exception {
          switch(id) {
8.
9.
          case 1:
               throw new BusinessException("10", "controlle
10.
  r10");
           case 2:
11.
               throw new BusinessException("20", "controlle
12.
  r20");
13.
           case 3:
               throw new BusinessException("30", "controlle
14.
  r30");
```

```
15.
            case 4:
                throw new BusinessException("40", "controlle
16.
  r40");
17.
            case 5:
                throw new BusinessException("50", "controlle
18.
  r50");
19.
            default:
20.
                throw new ParameterException("Controller Par
  ameter Error");
21.
            }
22.
        }
23.
24.
        @RequestMapping(value = "/service.do", method = Requ
  estMethod.GET)
        public void service(HttpServletResponse response, In
25.
  teger id) throws Exception {
26.
            testService.exception(id);
27.
        }
28.
       @RequestMapping(value = "/dao.do", method = RequestM
  ethod.GET)
        public void dao(HttpServletResponse response, Intege
30.
  r id) throws Exception {
            testService.dao(id);
31.
32.
        }
33. }
```

3.2.5 JSP 页面代码

Java 代码 ☆

```
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>
11.
12.
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>
16.
17.
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]</h
3>
21.
22.
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:

Xm1 代码 ☆

```
1. <bean class="org.springframework.web.servlet.handler.Simp
  leMappingExceptionResolver">
     <!-- 定义默认的异常处理页面, 当该异常类型的注册时使用 -
2.
  ->
     roperty name="defaultErrorView" value="error">
3.
  perty>
     <!-- 定义异常处理页面用来获取异常信息的变量名,默认名为 ex
  ception -->
     roperty name="exceptionAttribute" value="ex">
5.
  ertv>
     <!-- 定义需要特殊处理的异常,用类名或完全路径名作为 key,
6.
  异常也页名作为值 -->
     roperty name="exceptionMappings">
7.
8.
         ops>
9.
              key="cn.basttg.core.exception.BusinessE
  xception">error-business</prop>
                  key="cn.basttg.core.exception.Paramete
10.
  rException">error-parameter</prop>
11.
             <!-- 这里还可以继续扩展对不同异常类型的处理 -
12.
  ->
13.
          </props>
     14.
```

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 ☆

```
1. public class MyExceptionHandler implements HandlerExcepti
  onResolver {
2.
3.
      public ModelAndView resolveException(HttpServletReque
  st request, HttpServletResponse response, Object handle
  r,
4.
               Exception ex) {
          Map<String, Object> model = new HashMap<String, O</pre>
  bject>();
6.
          model.put("ex", ex);
7.
          // 根据不同错误转向不同页面
8.
9.
           if(ex instanceof BusinessException) {
10.
                return new ModelAndView("error-business", mo
  del);
11.
            }else if(ex instanceof ParameterException) {
12.
                return new ModelAndView("error-parameter", m
  odel);
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 ☆

- 1. <bean id="exceptionHandler" class="cn.basttg.core.excepti
 on.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、对己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 ☆

```
1. public class BaseController {
      /** 基于@ExceptionHandler 异常处理 */
2.
3.
      @ExceptionHandler
      public String exp(HttpServletRequest request, Excepti
4.
  on ex) {
5.
6.
          request.setAttribute("ex", ex);
7.
8.
          // 根据不同错误转向不同页面
9.
          if(ex instanceof BusinessException) {
                return "error-business";
10.
           }else if(ex instanceof ParameterException) {
11.
12.
                return "error-parameter";
13.
            } else {
               return "error";
14.
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 ☆

- 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 ☆

```
1. <!-- 出错页面定义 -->
2. <error-page>
      <exception-type>java.lang.Throwable/exception-typ
3.
  e>
4.
      <location>/500.jsp</location>
5. </error-page>
6. <error-page>
7.
      <error-code>500
      <location>/500.jsp</location>
8.
9. </error-page>
10. <error-page>
11.
      <error-code>404
       <location>/404.jsp</location>
13. </error-page>
14.
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:

业务错误: BusinessException

错误描述:

dao12

错误信息:

on.basttg.core.exception.BusinessException: dao12 at on.basttg.core.dao.TestDao.exceptat on.basttg.core.controller.TestController.dao(TestController.java:44) at sun.reflectsun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39) at sun.java.lang.reflect.Method.invoke(Method.java:597) at org.springframework.web.bind.annotation

3、参数错误显示的页面,如下图所示:

参数错误: ParameterException

错误描述:

Dao Parameter Error

错误信息:

cn. basttg. core. exception. ParameterException: Dao Parameter Error at cn. basttg. core. da cn. basttg. core. service. TestServiceImpl. dao(TestServiceImpl. java:35) at cn. basttg. core sun. reflect. NativeMethodAccessorImpl. invokeO(Native Method) at sun. reflect. NativeMethodAccessorImpl. invoke(DelegatingMethodAccessorImpl. java:25)

4、未知错误显示的页面,如下图所示:

未知错误: NullPointerException

错误描述:

null

错误信息:

java.lang.NullPointerException at cn.basttg.core.dao.TestDao.exception(TestDao.java:1 cn.basttg.core.controller.TestController.dao(TestController.java:44) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39) at sun.java.lang.reflect.Method.invoke(Method.java:597) at org.coringframework web bind.appo

5、服务器内部错误页面,如下图所示:

404错误!

错误描述:

您访问的页面不存在! 请与系统管理员联系!

错误信息:

您访问的页面不存在!

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。