请求驱动的校务管理系统异常处理机制①

李惠欢

(中山大学 网络与信息技术中心, 广州 510275)

摘 要: 异常处理机制是所有 WEB 应用设计中的关键组成部分, Spring MVC 框架提供了异常处理接口和多种解析器. 通过源码分析法深入探讨了各种解析器的工作原理, 总结得出它们的时间复杂度并进行了应用方式对照. 结合我校在校务管理系统基础框架上异常处理的应用需求情况, 将异常划分为业务异常和系统异常两大类, 设计并实施一个包含 AJAX 异常处理和 ExceptionHandler 统一异常处理的异常机制.

关键词: Spring MVC; 异常处理; 异常解析器; AJAX; ExceptionHandler

Exception Handling Mechanism in School Management System Based on Request

LI Hui-Huan

(The Network and Information Technology Center, Sun Yat-sen University, Guangzhou 510275, China)

Abstract: The exception handling mechanism is one of the key parts in any web application design. Spring MVC framework provides an exception handling interface and several exception resolvers. Through analyzing of the Spring Framework's source code, it describes the way which these exception resolvers work in, deduces their time complexity and lists their applicable scenarios. According to the application requirement of exception handling in framework of eCampus System in SYSU, it divides exception into two categories: business exception and system exception and designs and implements an exception handling including exception handling applying for AJAX and global exception handling of ExceptionHandler.

Key words: spring MVC; exception handling; exception resolver; AJAX; ExceptionHandler

异常处理用于处理软件中出现的异常状况, 即超出程序正常执行流程的某些特殊情况. 利用异常处理机制, 通知外界程序不能正常执行, 可以对用户在程序中的非法输入进行控制和提示, 防止程序崩溃. 通过异常处理机制, 开发者不需要使用返回值表示程序执行出错以及发生了什么错误, 并可以中断程序的继续执行.

Spring MVC 在异常处理机制上, 除了提供统一接口外, 它还提供了几种附带的异常处理方式. 在实际应用中可以单独采用某种方式或者组合使用, 也可以自定义特定系统特定场景的专用异常解析器. 在校务管理系统基础框架的异常处理设计中, 既使用了 ExceptionHandler 异常解析器实现异常的统一通知和日志记录, 也结合 JQuery 框架扩展定义了专用于处理 AJAX 响应异常的解析器. 此设计已经应用在教务管理系统、人事管理系统和权限管理系统上.

1 句柄异常解析器接口

HandlerExceptionResolver 接口是 Spring MVC 框架的异常处理核心接口, 用于处理控制器执行期间出现的未被处理异常. 接口中只定义了一个解析异常方法, 所有实现类都需要重写此方法以完成异常解析及处理. 接口定义如下[1]:

```
public interface HandlerExceptionResolver {
     ModelAndView resolveException( HttpServletRequest request,
HttpServletResponse response, Object handler, Exception ex);
}
```

关键抽象类 AbstractHandlerExceptionResolver 实现了 Ordered 接口和 HandlerExceptionResolver 接口. 所有继承自该抽象类的异常解析器, 用户可以利用参

① 收稿时间:2015-12-17;收到修改稿时间:2016-02-25 [doi:10.15888/j.cnki.csa.005334]

数 order 设定解析器的优先级别. 方法级异常解析器抽象类 AbstractHandlerMethodExceptionResolver, 是在对象成员方法的级别上实现异常处理. 上述各接口和抽象类的类关系如图 1 所示.

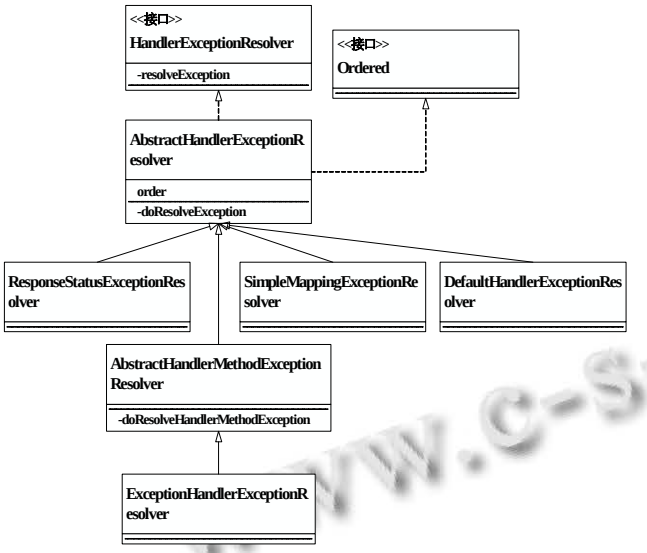

图 1　Spring MVC 异常处理相关的类图

2　四个异常解析器

基于句柄异常解析器接口, Spring WEB MVC 框架提供有四个异常解析器的实现类, 用于处理应用中抛出的异常.

2.1 简单映射异常解析器

SimpleMappingExceptionResolver 是 Spring 早在 2003 年就已经提供的一个简单便捷的异常解析器, 它根据配置文件进行异常解析[2]. 使用者可以配置异常类名与视图名的映射关系, 可以为所有未处理异常设定默认错误视图, 可以设定错误视图中引用异常所使用的名字, 可以记录异常消息, 可以为特定的已被解析错误视图设置 HTTP 状态码, 还可以设定默认的 HTTP 响应状态码、除外异常等参数[3]. 简单映射异常解析器的处理过程见图 2.

在是否除外异常中, 算法时间复杂度与配置中定义的除外异常数量n_1有关, 记为$O(n_1)$. 判定异常映射表有否定义时, 时间复杂度与配置中定义的异常映射表的异常数量n_2以及发生异常与定义异常的继承深度d_1有关, 记为$O(n_2 * d_1)$. 在有否定义 HTTP 状态码的判定中, 则与配置中定义的状态码数量n_3相关, 为$O(n_3)$. 因而算法时间复杂度为$O(n_1 + n_2 * d_1 + n_3)$, 由此可见解析器的性能与配置密切相关.

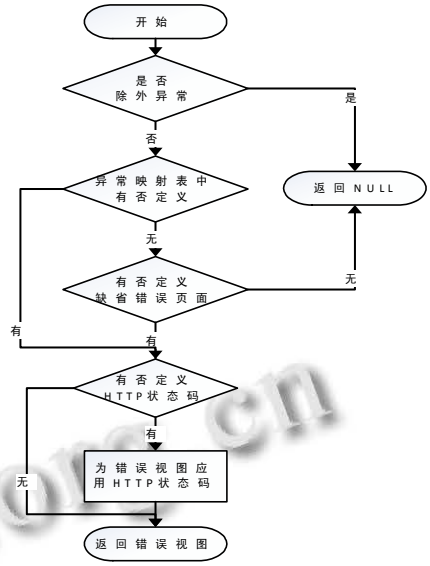

图 2　简单映射异常解析器的异常处理流程图

在是否除外异常中, 算法时间复杂度与配置中定义的除外异常数量 n_1 有关, 记为 $O(n_1)$. 判定异常映射表有否定义时, 时间复杂度与配置中定义的异常映射表的异常数量 n_2 以及发生异常与定义异常的继承深度 d_1 有关, 记为 $O(n_2*d_1)$. 在有否定义 HTTP 状态码的判定中, 则与配置中定义的状态码数量 n_3 相关, 为 $O(n_3)$. 因而算法时间复杂度为 $O(n_1+n_2*d_1+n_3)$, 由此可见解析器的性能与配置密切相关.

2.2 默认句柄异常解析器

作为 HandlerExceptionResolver 接口的默认实现, 默认句柄异常解析器用于解析标准的 Spring 异常(如绑定异常、类型不匹配异常、方法参数无效异常等十多种), 解析器将匹配并把这些异常转换成相应的 HTTP 状态码. 通过分析源码, 默认句柄异常解析器的时间复杂度为 O(1).

2.3 响应状态异常解析器

ResponseStatusExceptionResolver 处理由响应状态注解定义的未被捕获异常, 其异常处理方法在该类及其接口、注解、父类中寻找响应状态注解, 从而获得注解的属性设置, 再通过 HttpServletResponse 的 sendError 方法发出响应错误消息[2]. @ResponseStatus 响应状态注解用于标记特定方法或异常类的返回状态码和原因, 当被注解的方法调用返回时, 或者当抛出被注解异常时, 状态码被应用于返回的 HTTP 响应中.

解析器性能主要取决于查找响应状态注解所需的时间. 假设当前在某个类 C_i 中查找响应状态注解, 其

父类为 S_i. 类 C_i 中定义的注解数量为 N_i, 其中非 Java 原生注解的数量为 $Q_i(Q_i \leqslant N_i)$, 这些非 Java 原生注解记为集合 $P_i=\{p_{ij}|1 \leqslant j \leqslant O_i\}$. 假设类 C_i 实现的接口数量为 M_i, 这些接口集合记为 $K_i=\{k_{ij}|1 \leqslant j \leqslant M_i\}$. 则查找响应状态注解的时间复杂度为公式(1). 可见, 响应状态异常解析器的性能与未被捕获异常及其接口、注解、父类中的注解及其继承深度密切相关.

$$T(C_i)=\begin{cases} O(N_i)+\sum_{j=1}^{M_i}T(k_{ij})+\sum_{j=1}^{Q_i}T(p_{ij})+O(1), S_i = null 或 S_i = Object \\ O(N_i)+\sum_{j=1}^{M_i}T(k_{ij})+\sum_{j=1}^{Q_i}T(p_{ij})+T(S_i), S_i \neq null 且 S_i = Object \end{cases} \tag{1}$$

2.4 异常句柄异常解析器

ExceptionHandlerExceptionResolver 为方法级解析器, 继承自抽象句柄方法异常解析器类, 用于处理在控制器 Controller 及控制器增强 ControllerAdvice 中以异常句柄注解定义的方法.

异常句柄注解@ExceptionHandler 可用于句柄类及句柄方法的异常处理. 使用异常句柄注解的句柄方法可以应用灵活属性设置, 可以使用异常、请求对象、响应对象、会话对象等作为参数, 支持句柄方法多种返回值类型, 还能结合响应状态注解组合使用[4].

异常句柄异常解析器处理过程如下:

1)在异常句柄缓存中寻找异常抛出所在控制器的异常句柄方法解析器实例, 如果没有则实例化一个.

2)查看该实例是否含有能处理抛出异常的方法, 有则获得相应的句柄方法对象.

3)如果控制器中找不到合适的处理方法, 则从异常句柄增强缓存中寻找异常处理方法. 如果找到则获取句柄方法对象, 否则返回 NULL.

4)向句柄方法注入用于处理异常句柄注解所定义参数及调用方法返回值的句柄方法参数解析器集合、句柄方法返回值集合.

5)试图使用 invokeAndHandle 方法调用异常处理方法, 并对异常句柄注解所定义参数及异常处理方法返回值进行处理, 返回包含视图名、参数信息的模型视图容器.

6)处理模型视图容器结果, 生成最终模型视图对象.

由此可见, 解析器首先对控制器内部及其父类中定义的异常句柄注解优先处理, 因而局部定义的异常处理方法将优先于全局定义的异常处理方法. 局部定义的异常处理方式, 根据异常继承深度进行就近选择; 而全局定义的异常处理方法, 则使用排序比较器 OrderComparator 进行控制器增强排序, 将调用优先级较高的控制器增强中定义的相应处理方法.

此解析器的性能由在异常抛出的控制器内查找合适的处理方法、在控制器增强中查找合适的处理方法以及异常处理方法调用三部分决定. 在控制器中查找时, 如果忽略控制器在异常处理缓存中不存在需要实例化的情况, 其性能取决于在控制器中定义的异常句柄注解的数量 n, T(C)=O(n). 而在控制器增强中查找时, 除了需查找异常句柄注解外, 还需要检测该控制器增强是否支持异常句柄注解, 这与其可指定的异常类型、定义的注解数量和基础包数量都有关, 记为 $T(f(b_i))$. 假设整个应用中控制器增强数量为 p, 其中某个控制器增强 $a_i(1 \leqslant i \leqslant p)$所定义的异常句柄注解数为 q_i, 则时间复杂度可以表示为 $T(A)=\sum_{i=1}^{p}\left(O(q_i)+T(f(b_i))\right)$. 假设自定义的异常处理方法为 f(n), 遍历该方法返回值的时间复杂度为 T(r), 从代码可知, 调用异常处理方法的时间复杂度为 T(H)=2×T(r)+ T(f(n)).

2.5 几种异常解析器的比较和应用场景

经过上述分析和实际应用情况, 总结出这 4 种异常解析器在工作原理、应用场景和性能的比较.

表 1 几种异常解析器情况对照表

	异常句柄异常解析器	响应状态异常解析器	默认句柄异常解析器	简单映射异常解析器
解析范围	全局/控制器内部	全局	全局	全局
解析方式	查找 @ExceptionHandler 注解	查找 @ResponseStatus 注解	映射匹配	映射匹配
定制性	很强	偏弱	不能定制	一般
可否日志记录	自编码	无	无	参数配置
注册方式	配置 <annotation-driven/>	配置 <annotation-driven/>	MVC 名字空间等方式	在上下文中配置 bean
易用性	一般, 需要自编码	容易	容易	一般
性能	主要视乎自编码程序的性能	毫秒级	毫秒级	毫秒级(不记录日志的情况下)

性能测试环境为 JDK7+Tomcat7+Spring 及 Spring MVC 4.0, 测试先后定义各异常解析器为首个进行异常解析的 4 个场景, 利用 loadrunner 模拟虚拟用户数为 100、200、500 和 1000 情况下的压力测试结果. 在不进行日志记录和业务操作的情况下, 用户请求异常的响应时间均在毫秒级内. 而且从上节时间复杂度的分析可见, 除默认句柄异常解析器外, 其他解析器的性能均取决于解析器的配置、所定义注解以及处理方法的性能.

2.6 异常解析器的应用场景

默认句柄异常解析器用于处理 Spring 内部异常, 对所有应用 Spring 的系统而言都是不可或缺的组成部分, 多种注册方式, 静默配置让用户更省心. 简单映射异常解析器适用于小系统、不需要对异常进行严格区别化处理的全局异常处理的情景, 映射式配置简单易用, 还可以进行异常日志记录. 对于自定义的异常可以加入响应状态注解, 并配置响应状态异常解析器, 用于截获其他高优先级的异常解析器无进行异常处理的情景. 在简单映射异常解析器不能满足需求的时候, 可使用异常句柄异常解析器. 根据实际需要, 针对某些特定异常实现控制器层的或者全局化的异常处理定制, 局部处理优于全局定义.

另外, 在控制器增强中定义的全局异常处理方法, 应做好全局规划, 对某类异常应避免重复定义, 同一控制器增强中定义的异常处理方法具有一定随机性. 多个解析器组合使用的时候必须定义好异常解析器的优先级别, 避免出现漏处理或被其他解析器优先处理等“意外”情况. 如果这些解析器不能满足需要, 需自定义控制器内部的局部解析器, 可以通过自定义 AbstractHandlerMethodExceptionResolver 抽象类的实现类或者写 ExceptionHandlerExceptionResolver 的子类两种方式实现.

3 校务系统异常处理的实现

3.1 设计考虑

校务管理系统是 B/S 结构的应用管理系统, 涵盖本科教务、人事管理、权限管理等多个子系统, 采用 Java+Tomcat+Oracle 的技术路线, 基于统一基础框架、统一 UI、统一数据库进行设计开发. 基础框架涵盖前端 UI 组件、认证及权限控制组件、监控与日志组件、缓存组件、文件 I/O 组件、异常处理组件等, 层次结构如图 3 示. 集成了多个开源框架, 包括 JQuery、Spring、Spring MVC、MyBatis 等. 异常处理是基础框架中不可或缺的重要组成部分.

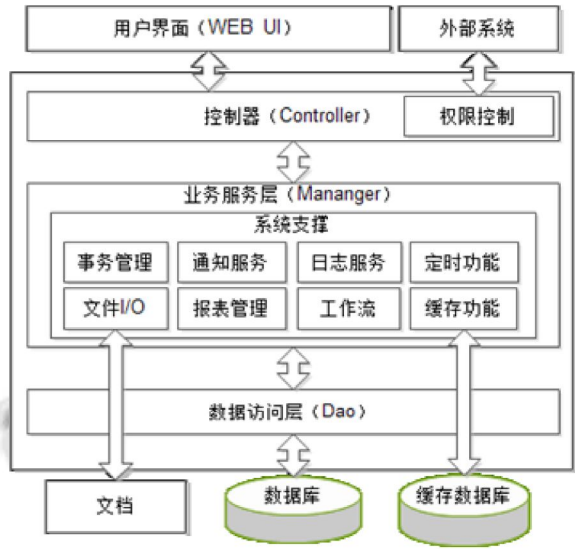

图 3 基础框架层次结构图

考虑以下三个异常场景: 在电话号码输入框输入非数字, 可直接在 UI 层处理; 在用户名输入框输入未注册的用户名, 可通过 Ajax 异步查询的方式将问题反馈给用户; 批量导入数据时, 由于磁盘空间满导致请求失败并反馈信息给用户. 由此可见, 异常可能出现在系统的各个层次结构上, 异常的捕捉处理可分为后台服务器处理以及 UI 界面处理两部分. 下面主要考虑普通用户请求、Ajax 异步请求时发生的异常, 直接在 UI 层处理的异常相对简单.

基础框架中将异常划分为业务异常和系统异常两类(见图 4). 业务异常只要指由于违背业务规则而引起的异常情况, 如用户不存在、业务操作不在允许时间范围、用户输入校验不通过等, 这类异常往往只需用户调整操作或调整输入内容则可以避免和纠正的情况. 系统异常是指业务异常以外的不可预测的应用级异常, 如数据库链接不上、IO 读写错误等, 这类异常则需要管理员介入处理才能解决.

这两种异常均继承自 Java 的运行时异常 RuntimeException(Java 虚拟机运行过程中抛出的异常). 相对于 Checked Exception 而言, 它并不需要在方法中定义 throws 字句, 而且程序设计者可以不处理此类异常. 因此在各业务系统开发过程中, 程序员可以不考虑其他模块抛出的业务异常和系统异常. 而且各业务系统可以通过继承这两个类, 创建本系统或本模块适用的

特殊异常类, 特别是定义反馈给用户的提示消息.

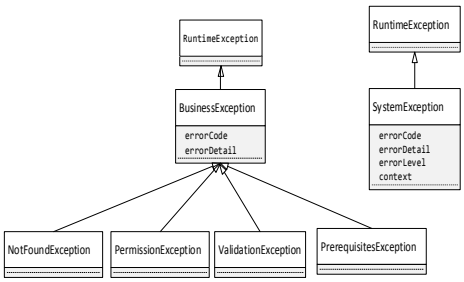

图 4 基础框架的异常类图

业务异常定义了异常编号 errorCode 属性, 用于快速定位异常发生的位置; 定义了异常详细描述信息 errorDetail, 目的是为客服人员和用户提供更多的异常参考依据, 在 errorDetail 中可以描述异常发生的可能原因、排查方式以及相应的解决方法. 系统异常在此基础上定义了异常级别 errorLevel, 根据严重程度划分成严重、错误、警告、信息、调试五个等级; 以及上下文信息 context, 可用于保存异常发生时的关键性信息, 方便排查原因, 例如保存异常发生时的用户 ID、读取文件路径等信息.

校务管理系统的异常处理设计原则为: 业务代码应尽力处理所有的业务类异常, 抛出的异常(系统类异常和未处理的业务异常)则应用 Spring MVC 处理机制进行解析处理, 做到不把内部错误暴露给用户, 增强系统健壮性和用户体验度. 当抛出这两类异常时, 系统框架都会记录异常详情到日志文件中, 并将提示信息反馈给用户, 对于系统异常还会即时给管理员发送邮件或短信.

3.2 后台及 UI 层的实现

在后台, 中央调度处理器将用户请求或 Ajax 异步请求转发给相应的控制器进行处理, 当捕捉到异常发生时则交给异常解析器进行处理, 如图 5 所示. 中央调度处理器在初始化时, 将依据配置文件的设置, 寻找所有的句柄异常解析器接口的实现类, 并应用排序比较器进行排序, 从而初始化异常解析器集合.

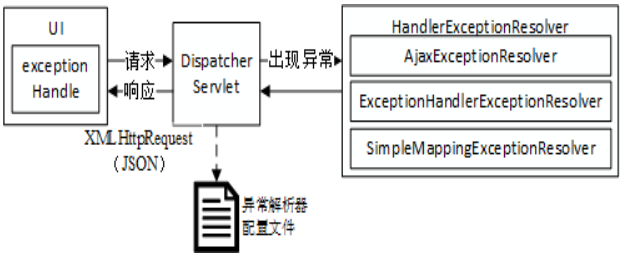

图 5 系统异常处理示意图

在基础框架配置文件中定义异常解析器配置实例如下:

```
<!-- AJAX 异常处理 -->
<bean class="AjaxExceptionResolver" p:order="0">
        <property name="ajaxErrorView" value="error/exception" />
</bean>
<!-- 配置注解驱动 -->
<mvc:annotation-driven />
<!-- 统一异常处理 -->
<bean class="SimpleMappingExceptionResolver" p:order="3">
<!-- 定义默认的异常处理页面 -->
<property name="defaultErrorView" value="error/exception" />
<!-- 定义异常处理页面用来获取异常信息的变量名, 默认为 exception -->
<property name="exceptionAttribute" value="exception" / >
<!-- 设置日志输出级别, 不定义则默认不输出警告等错误日志信息 -->
<property name="warnLogCategory" value="SimpleExceptionResolver" />
</bean>
```

<annotation-driven/>配置中定义的异常句柄异常解析器、响应状态异常解析器和默认句柄异常解析器, 它们优先级分别是 0、1 和 2. 对于相同优先级的情况, 将按照配置文件中配置的先后次序排放优先级. 所以上述实例中的优先级依次为: Ajax 异常解析器、异常句柄异常解析器、响应状态异常解析器、默认句柄异常解析器和简单映射异常解析器. 可见, 实例中应用产生的大部分业务异常和系统异常应由 Ajax 异常解析器和异常句柄异常解析器解析完毕.

在 UI 界面处理上, 利用 XMLHttpRequest 对象实现异常信息的传递. 通过分析后台返回的 JSON 格式的 XMLHttpRequest 实例, UI 可以判定请求在后台是否发生了异常, 如果是则调用 UI 层统一异常处理函数 exceptionHandle 处理或由相应页面做个性化处理.

3.3 AJAX 异常解析器

基础框架中定义的 Ajax 异常解析器是全局的异步请求异常解析, 继承自抽象句柄异常解析器类. 其处理过程大致如下:

1) 检测是否为 Ajax 异步请求, 如果不是则返回 null(不处理);

2) 若是 Ajax 异步请求, 则记录日志并给管理员发送消息, 然后以 JSON 格式通过 response 送出错误信息;

3) 在前端页面/控件上需分析返回信息, 并反馈提示信息给用户(如图 6 示).

由于目前使用的浏览器厂家和版本的多样性, 对于是否为异步请求的检测, 实际上并不能完全正确判定所有的 Ajax 请求. 为了避免给用户返回一些非友好的错误信息和页面, 可以使用 JQuery 提供的全局 Ajax 事件处理器进行过滤. JQuery 的全局 Ajax 事件处理器, 可用于处理页面上每个 Ajax 请求, 在任一 ajax 请求的开始执行、发送前、完成后、出错时、成功时, 均可

注册相应的回调函数. 因此在校务管理系统框架上定义了对 Ajax 出错时的统一处理方法, 如下:

```
$(document).ajaxError(function(event, xhr, options, exc) {
        exceptionHandle(xhr);//具体的处理函数
});
```

图 6 Ajax 异常解析器处理效果图

3.4 ExceptionHandler 统一异常处理

Ajax 异常解析器不能解析的异常, 将由异常句柄异常解析器接力解析. 当控制器内部无定义局部适用的异常解析方法时, 异常句柄异常解析器将查找是否在控制器增强中定义了全局适用的处理方法. 因此, 对于非 Ajax 异步请求产生的异常情况, 可以定义适用于业务异常及系统异常的全局解析方式. 对于业务异常, 系统将记录异常信息到日志文件中, 并送出错误提示信息; 而对于系统异常, 系统除记录日志外还会给管理员发送即时消息. 业务异常解析方法如下.

```
@ControllerAdvice(basePackages = "cn.edu.sysu.framework.controller")
public class GlobalExceptionHandler {
    public static Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);
    /** 处理业务异常 */
    @ExceptionHandler(BusinessException.class)
    @ResponseBody
    public String handleBusinessException(HttpServletResponse response, Exception ex) {
        BusinessException e = (BusinessException) ex;
        logger.error("business exception:", ex);
        return JSONUtil.errorMsg(e);//封装成 json 格式
    }
}
```

4 结语

校务管理系统基础框架的异常处理机制, 将异常划分为业务异常和系统异常两大类, 在后台服务器处理时, 通过 Ajax 异常解析器、ExceptionHandler 异常解析器等, 对捕捉到的两大类异常进行差异化处理, 并最终在 UI 界面上将错误信息反馈给用户. 通过源码分析法分析 Spring MVC 所提供的四个异常解析器的时间复杂度, 参考此分析结果确定了系统异常解析器的配置方式, 并注意漏处理、重复设置、优先级设定不恰当等情况. 通过扩展 HandlerExceptionResolver 接口, 定义 Ajax 异常解析器实现对 Ajax 异步请求出现异常的情况进行个性化解析.

这里只涉及到服务器中间件上应用出现的 Java 异常以及前端页面代码出现的异常, 对于数据库服务器上存储过程和触发器等产生的异常, 需要综合考虑其他异常处理机制, 有待完善.

参考文献

1 Pivotal Software, Inc.Spring Framework 4.1.2.RELEASE API. 2014. http://docs.spring.io/spring/docs/current/ javadoc-api/.

2 Chapman P. Exception Handling in Spring MVC. Nov.01, 2013. http://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc.

3 Walls C.耿渊,张卫滨译.Spring 实践.北京:人民邮电出版社,2013.

4 Johnson R, Hoeller J, Donald K, et a1.Spring Framework Reference Documentation. 2014. http://docs.spring.io/spring/docs/4.1.1.RELEASE/spring-framework-reference/htmlsingle/#mvc-exceptionhandlers.

5 管华,应时,贾向阳,等.面向服务软件异常处理研究综述.计算机科学,2013(40):1–8.

6 毛莉娜.基于 AOP 的 Java 异常处理框架和工具的分析与设计.广东技术师范学院学报(自然科学),2014,(7):27–30.

7 陈红跃,张宏军,陈刚.Java 异常处理策略研究.计算机技术与发展,2012,(22):9–12.

8 刘淑华.J2EE 项目中一种新的错误处理方法.计算机应用与软件,2013,(30):143–145.

9 张语涵,刘淑华,周永鑫.Java Web 应用中错误和异常处理方法研究.现代计算机,2013(8):61–65.

10 王新雨,须文波,柴志雷.Java 虚拟机中异常机制实时性的研究及实现.计算机工程与应用,2008,44(34):84–85.

11 刘怀愚,朱昌杰,李璟.时间复杂度的几种计算方法.电脑知识与技术,2011,(7):4636–4638.

12 舒礼莲.基于 Spring MVC 的 Web 应用开发.计算机与现代化,2013,(11):167–168,173.