Java异常处理技术及EJB中的应用

郭广军 羊四清 戴经国 贺文华

湖南人文科技学院计算机科学系，湖南 娄底 417000

2007-08-20

摘要：在企业级的大型软件开发中，严谨强大的Java异常处理机制为软件质量控制提供了有力的技术支持，提高了软件的可读性、可维护性、容错性和开发效率。深入分析了Java异常处理的基本机制及语法结构、总结了异常处理的基本原则，进一步探索了EJB中异常处理的方法，并给出了其关键技术、优化策略和应用实例。

关键词：Java；EJB；异常处理。

中国分类号：TP311.52 文献标识码：A

Java Exception Handling Technique and Application in EJB

GUO Guang-jun YANG Si-qing DAI Jing-guo HE Wen-hua

(Department of Computer Science of Hunan University of Humanities and Science and Technology, Loudi Hunan 417000, China)

Abstract: In the large-scale enterprise software development, the precise, strong exception handling mechanism of the Java provided powerful technique support for the software quality control, as well as, the readability, maintainability, fault-tolerance performance and development efficiency of the software are improved. In this paper, first the basic exception handling mechanism and syntax structure of the Java are thoroughly analyzed, and the basic principle of the exception handling is advanced, then the exception handling mechanism of the EJB is probed into, the key technique, optimization strategy and example of the exception handling in EJB are provided last.

Keywords: Java; EJB; Exception Handling

1 引言

传统的基于函数返回错误代码的错误处理方法存在明显的不足，如程序逻辑复杂，结构不清；可靠性差，维护不便；返回信息有限，不直观需译码；返回代码标准化困难，代码复用率低；在面向对象的应用系统中，有些如构造方法等没有返回值而无法报告程序错误。因此，在Java中新的错误检测和报告方法—异常处理机制应运而生。

异常是指中断程序正常执行流程的错误事件，如程序打开不存在的文件、装载不存在的类、网络连接中断、被零除、访问数组越界、系统资源耗尽等。在Java中，异常[1](Exception，例外)是特殊的运行错误对象，是异常类的一个对象，而每个异常类代表一种运行错误，在异常类中封装了该运行错误的信息和处理错误的方法等内容。Java异常处理机制的基本思想是由发现而不能处理错误的方法引发一个异常对象，然后由该方法的直接或间接调用者捕获并处理这个错误。其优越性有：在catch中传播与捕获错误信息，实现了错误代码与业务逻辑的分离，结构清晰；可对错误类型分组并标准化；方便了对错误的定位与维护；能有效防止由于异常而导致程序运行崩溃，可靠性高；强制程序员考虑程序的容错性、健壮性和安全性。

2 Java异常处理机制

2.1 异常类

2.1.1 系统定义的异常类

异常类用于处理异常，分为系统定义的异常类和用户自定义的异常类[1]。在java.lang包中提供的Throwable类是异常类层次结构的顶层类，Error类和Exception类是从Throwable类直接派生的两个知名子类。

Exception类：它代表了Java语言中异常的基本属性，除Java预定义的由Exception类派生的诸多异常类外，还支持用户扩展Exception类来实现自定义异常类。其构造函数主要有public Exception()和public Exception(String s)等；它从Throwable类继承了若干方法，常用的有：public String toString()方法，用来返回异常类信息；public void printStackTrace()方法，默认在当前标准输出设备上输出当前异常对象的堆栈使用轨迹。Exception类定义的是非致命性错误，允许用户编写代码来处理这类错误，并继续程序的执行。通常触发异常（Exception）的原因有打开的文件不存在；网络连接中断；操作数超过允许范围；想要加载的类文件不存在；试图通过空的引用型变量访问对象；数组下标越界等。

Error类：它定义的错误是致命性错误，如虚拟机错误、装载错误、动态连接错误，一般会导致程序停止执行，通常是由于Java系统或执行环境发生错误（Error）而导致的。由于这类异常主要与硬件、运行时系统有关，而不是由用户程序本身抛出的，因此用户程序不对这类异常进行处理。

需指出，除java.lang包中定义的异常处理之外，其他的Java包中也包括异常。实际上几乎每个Java包都定义了相应的异常类。此外，运行时异常RuntimeException类及其派生子类是Java程序员不用处理的异常。Java创建者认为运行时异常不应由程序来处理，而且程序也很难真正的对付运行时异常。

2.1.2 用户自定义异常类

热点专题

- 信心09,冬天来了,春天还会远吗?
- 低功耗技术，是鸡还是蛋?
- 华北计算机系统工程研究所(电子六所)总结表彰暨春节联欢会
- Powerwise高效能解决方案
- 2008Security China中国国际社会公共安全产品博览会
- 视频信号处理技术
- 2008嵌入式技术创新及...
- 2008飞思卡尔技术论坛
- Altera公司SOPC...
- 第十届高交会电子展
- 科技闪耀北京奥运
- ADLINK DAY—2008年量测与自动化技术国际高峰研讨会
- 中国电子学会Xilinx杯开放源码硬件创新大赛
- 赛灵思公司Virtex-5系列FPGA
- 3G知识
- IPTV
- 触摸屏技术
- RoHS

杂志精华

- 基于CC2430的无线传感器...
- 无线传感器网络应用系统综述
- 无线传感器网络在野外测量中的...
- 基于竞争的无线传感器网络
- 用于矿井环境监测的无线传感器...
- 具有自适应通信能力的无线传感...
- 基于传感器网络技术的深孔测径...
- 基于无线传感器网络的家庭安防...
- 基于ATmega128L与C...
- 无线传感器网络中移动节点设备...

用户自定义异常类是指扩展Exception类或其他某个已经存在的系统异常类或其他用户异常类而形成新的异常类。可以给新的异常类定义新的属性和方法，或重载父类的属性和方法，并使这些属性和方法能够体现该类所对应的错误信息。要特别注意的是：第一 一个方法被覆盖时，覆盖的方法必须扔出与被覆盖方法相同的异常或其异常类的子类；第二 若父类抛出多个异常，则覆盖方法只能抛出父类所抛出的异常的一个子集，或者说不能抛出新的异常。

2.2 基本机制与语法结构

2.2.1 基本机制

Java异常处理机制采用中断模式[2]，即引发并抛出异常后，中止正在执行的程序块，控制流转至异常处理器，待完成异常处理后，再返回调用点继续执行。异常处理的基本算法是：

Step1：抛出异常，即创建一个异常对象并将它交给运行时系统的过程；

Step2：捕获异常，即找到异常处理程序的过程：运行时系统从发生错误的方法开始回溯，在方法调用堆栈里向后搜索，直到找到能处理当前发生的异常的处理程序的方法；

Step3：处理异常，即通过方法调用来实现对异常的处理；

Step4：终止异常处理。若运行时系统在方法调用栈中遍历了所有的方法而未找到合适的异常处理程序，则显示缺省错误并终止执行运行时系统的异常处理。

2.2.2 语法结构

Java异常处理机制通过throws、throw、try、catch和finally 5个关键词来实现，分为三个基本部分[3]。

· throws：此关键字统一定制并明确标明了一个方法可能抛出的各种异常，这些异常是该方法定义的一部分。其实质是允许将异常处理递归交给调用它的上一级方法去处理，此时Java编译器会强制此方法的调用者必须将其放在调用方法的try、catch块中以抛出并捕获处理这些异常。

· throw：此语句用来抛出紧跟其后的一个异常对象给此方法的调用者，此异常对象可用new创建，或者是一个Throwable的实例句柄通过参数传到catch中。因为用户自定义的异常不能由系统自动抛出，所以必须借助于throw语句来抛出各种错误情况所对应的异常，且要求在程序中的合适位置定义好用户自定义的异常类。

· try-catch-finally：此语句是Java错误处理的基本结构，主要用来捕获和处理一个或多个异常。通常由try、catch、finally三个块组成。ⅰ）try块：将所有可能抛出异常的代码部分放入try块中；ⅱ）catch块：用紧跟在try块后面一个或多个catch子句来捕获异常，其的目标是处理异常，把变量设到合理的状态，并象没有出错一样继续运行。若一个子程序不处理这个异常，则可返回到上一级处理，如此不断的递归向上直到最外一级。ⅲ）finally块：finally是Java异常处理机制的精髓，使用finally可以维护对象的内部状态、清理非内存资源、将系统恢复到应该处于的状态。若没有finally，要实现其功能的代码是很费解的。finally块是可选块，若定义了finally块，则不论try块中有无异常产生，finally块都会被执行；甚至若在try或catch块中执行了return、break语句，finally块也会被执行，但要特别注意此时finally块后面的语句并不会被执行。只有在try或catch中执行了System.exit(0)操作，才不会执行finally块。另要特别指出的是：捕获异常时，catch语句是按其位置由前至后依次对被抛出的异常对象进行匹配捕获，若有多个catch语句，则异常类要按从子类到父类的顺序放置；在应用技巧中，还可通过在try块中由throw抛出“异常”，然后在catch块中捕获之，实现程序中业务逻辑控制流程的跳转。

2.3 异常处理的基本原则

对于非运行时异常必须捕获或声明，而对运行时异常则不必，可以交给Java运行时系统来处理；对于自定义的异常类，通常把它作为Exception类的子类，且类名常以Exception结尾，不要把自定义的异常类作为RuntimeException类或Error类的子类；在捕获或声明异常时，要选取合适类型的异常类，注意异常类的层次，根据不同的情况使用一般或特殊的异常类；根据具体的情况选择在何处处理异常，或者在方法内捕获并处理，或者用throws子句把它交给调用栈中上层的方法去处理；在catch语句中尽可能指定具体的异常类型，必要时使用多个catch；使用finally语句为异常处理提供统一的出口；若无法处理某个异常，则不捕获它；若捕获了一个异常，则要对它进行适当的处理；尽量在靠近异常被抛出的地方捕获异常；除非要向上层递归抛出异常，否则要在捕获异常的地方将其记录到日志中。

3 EJB中的异常处理

3.1 EJB异常处理

EJB（Enterprise JavaBean）是J2EE企业级应用开发的核心组件，EJB的分布式和事务属性使得其异常处理成为一个更重要的问题[4]。EJB中异常可分为三类[5]：ⅰ）JVM异常：由JVM抛出，是一种致命错误。ⅱ）系统异常：一般由JVM以RuntimeException的子类抛出，是一种非受查异常。ⅲ）应用程序异常：它是一种定制异常，由应用程序或第三方类库以Exception类或其子类抛出，是一种受查异常，通常由EJB方法的调用者来处理之。EJB容器拦截了EJB组件上的每一个方法调用，因此方法调用中发生的每一个异常也被EJB容器所拦截。EJB规范只处理应用程序异常和系统异常这两种类型的异常。

应用程序异常：是指在远程接口的方法说明中所声明的异常，它不是远程异常RemoteException，也不应继承RuntimeException或其子类。但是，在远程接口方法的throws子句中声明的非受查异常并不会被当作应用程序异常。应用程序异常是业务工作流中的例外，当这种类型的异常被抛出时，客户机可得到一个恢复选项。当发生应用程序异常时，默认情况下EJB容器不会自动回滚事务，只有被显式说明并通过调用关联的EJBContext对象的setRollbackOnly()方法才能回滚事务。实际上，对于应用程序异常EJB容器通常以它原本的状态传送给客户机而不会以任何方式包装或修改它。

系统异常：通常被定义为非受查异常，EJB方法不能从这种异常中恢复。当EJB容器拦截到非受查异常时，会自动回滚事务并执行必要的清理工作，然后把该非受查异常包装到RemoteException类或其子类中并抛给客户机。对于受查异常，若要使用EJB容器的内务处理，则必须将受查异常作为非受查异常抛出。因此，每当触发受查系统异常时，应该对其原始的异常以javax.ejb.EJBException类或其子类方式包装后抛出。由于EJBException本身是非受查异常，因此不需要在方法的throws子句中声明它。EJB容器会自动捕获EJBException或其子类，并把它包装到RemoteException中，然后抛给客户机。

需指出，虽然EJB规范规定系统异常由应用程序服务器记录，但记录格式会因应用程序服务器的不同而异。为确保异常记录格式的统一，方便对其进行统计分析，在代码中记录异常是一种好的策略。此外，在EJB1.0规范中要求把受查系统异常以RemoteException抛出，而EJB 1.1及以上规范则规定EJB实现类绝不应抛出RemoteException。

3.2 关键技术

3.2.1 日志机制

尽管用System.out.println()方法跟踪异常方便，但开销大，对I/O处理的同步控制将大大降低系统吞吐量。缺省时堆栈跟踪被输出至控制台，但在实际的应用系统中，通过浏览控制台来查看异常跟踪不太现实。因此，在大型应用系统的开发、测试和运行等生

命周期中，采用日志机制和恰当的编码策略，精确地记录各种类型的异常，可以降低系统开销，提高软件性能和质量。知名的日志实用程序有两种：Log4J[6]是Apache的Jakarta的一个开放源代码的项目，J2SE 1.4捆绑提供了日志处理包（java.util.logging）[7]，它们的使用方法请参考相关文献。

3.2.2 Decorators设计模式

在面向对象的程序设计中若用一个对象（the Decorators）包装另外一个对象，被称为Decorators设计模式。基于Decorators设计模式，通过包装原始的异常消息并在EJB组件中将其重新抛出，以方便对该异常的查询和访问。其次，由于Log4J只能记录String消息，所以要利用Decorators设计模式定义一个专门类负责把堆栈跟踪转换成String，以获取该堆栈跟踪的String表示。

3.3 EJB异常处理策略

3.3.1 常见EJB异常处理的不足

·抛出带有出错消息的异常：此种方法存在异常内容可能被“吞掉”的现象。

·记录到控制台并抛出一个异常：仅当控制台可用时调用者才能向后跟踪。

·包装原始的异常以保护其内容：可能导致重复记录，产品日志或控制台不能被交叉引用。

3.3.2 EJB异常处理的优化策略

·优化应用程序异常处理：由EJB开发者显式抛出，通常包装了含义清楚消息，不必将其记录到EJB层或客户机层，而以一种直观有意义的方式提供给最终用户，并带上其解决方案的途径。

实体Bean一般是哑类，通常应用程序异常主要来源于会话Bean。从实体Bean抛出的应用程序异常类型通常是CreateException、FinderException、RemoveException及它们的子类。当引用其它EJB远程接口时，实体Bean会遇到RemoteException，在查找其它EJB组件时会遇到NamingException，若使用BMP实体Bean，则会碰到SQLException。与这些类似的受查系统异常应该被捕获，并被包装起来，作为EJBException或它的一个子类抛出。在优化的EJB设计中，客户机一般不直接调用实体Bean上的方法，而通过会话Bean间接访问实体Bean。若会话Bean调用相同的实体Bean方法，则要避免对异常的重复记录。会话Bean和实体Bean处理系统异常的方式基本相似。可采用访问标识技术避免对同一异常的重复记录。

·优化系统异常处理：比应用程序异常处理更为复杂，它的发生不受EJB开发者的控制且异常信息不直观，需要对其原始异常进行包装，以清楚地表达系统异常的含义。

·优化Web层设计：通常把异常记录到Web层本身，则易于实现且代码简洁。这要求Web层必须是EJB层的唯一客户机，且Web层必须建立在业务委派（Business Delegate）、FrontController或拦截过滤器（Intercepting Filter）等模式和Struts或其它类似于MVC的框架基础上。

实际应用中，即使采用良好的异常处理策略，但由于编译器和运行时优化，会限制使用堆栈跟踪程序跟踪异常的能力。通常把大的方法调用分割为更小的、更易于管理的块，并提高代码复用率；并将异常类型按需要划分成细粒度的、具体的异常，在捕获异常时则捕获已规定好的具体类型的异常，避免捕获所有类型的异常。

3.4 EJB异常处理实例

例1：ejbCreate()方法中的FinderException异常处理。其代码如下：

```
public Object ejbCreate(RatepayingOrderValue value) throws CreateException {
try { if (value.getItemName() == null) {
throw new CreateException("不能创建报税单！"); }
String taxpId = value.getTaxpayerId();
Taxpayer taxp = taxpayerHome.fingByPrimaryKey(taxpId);
this.taxpayer = taxp;
} catch (FinderException fe) {
//作为应用程序异常，还是系统异常？
} catch (RemoteException re) {
//这是系统异常，并记录在日志中。
throw ExceptionLogUtil.createLoggableEJBException(re);
}
return null;
}
```

例1中报税单RatepayingOrder实体Bean的ejbCreate()方法试图获取纳税人Taxpayer实体Bean的一个远程引用，可能导致FinderException。此处，尽管可把FinderException当作应用程序异常或作系统异常，但是若把它当系统异常则更好，因为这可以提高EJB组件对客户机的透明性。同理，对于会话Bean或者实体Bean试图创建另一个会话Bean，可能导致的CreateException，或者会话Bean在它的某个容器回调方法中获得了一个FinderException等，都最好将其作为系统异常。此外，若考虑会话Bean在处理下报税单时，用户须具有一个简历，若没有，则会话Bean将触发ObjectNotFoundException异常，这时最好将其作为应用程序异常抛出，以告知用户其简历丢失。

例2：logon()方法的InvalidUserDataException应用程序异常处理。其代码如下：

```
public void logon(String user, String password) throws InvalidUserDataException
{     if (user == null || password ==null)
                throw new InvalidUserDataException();
       serviceImpl.logon(user, password);
}
```

以下是应用程序异常类InvalidUserDataException的定义。

```
public class InvalidUserDataException extends Exception
{  public InvalidUserDataException()
       {super(“用户名或密码不能为空！”); }
}
```

4 结论

在企业级的大型软件开发中，严谨强大的Java异常处理机制为软件质量控制提供了有力的技术支持，提高了软件的可读性、可维护性、容错性和开发效率。充分有效的利用Java异常处理机制、采用合适的异常处理策略，是提高EJB中异常处理的性能的有效途径。

参考文献：

[1] Bruce Eckel. Thinking in Java[M].Beijing: China Machine Press,2000.240-281.

[2] 赵化冰,唐英,唐文彬,芦东昕. Java异常处理[J]. 计算机应用, 2003,12:46-48.

[3] 张聪品,赵琛,糜宏斌. 异常处理机制研究[J].计算机应用研究,2005,4:86-89.

[4] [美]Chuck Cavaness Brian Keeton著,智慧东方工作室 译. EJB 2.0企业级应用程序开发[M].北京：机械工业出版社,2002,3.294-310.

[5] EJB异常处理的最佳做法. http://www.evget.com/view/article/viewArticle.asp?article=548

[6] Log4J框架. http://jakarta.apache.org/log4j/docs/index.html

[7] Java Logging API. http://java.sun.com/j2se/1.4/docs/api/java/util/logging/package-summary.html