

异常处理机制研究^{*}

张聪品¹, 赵 琛², 糜宏斌²

(1. 河南师范大学 计算机科学系, 河南 新乡 453007; 2. 中国科学院 软件研究所, 北京 100080)

摘 要: 介绍了异常处理机制, 包括异常的抛出、捕获、传播, 并描述了异常的处理模式、传播机制、处理环境。不同应用领域的异常处理机制不同, 以 Java 语言和工作流管理系统为例, 分别介绍和讨论了程序设计语言层面和企业层面上的异常处理机制。

关键词: 异常处理机制; 软件设计; Java; 工作流管理系统

中图法分类号: TP314 文献标识码: A 文章编号: 1001-3695(2005)04-0086-04

Research on Exception Handling Machine

ZHANG Cong-pin¹, ZHAO Chen², MI Hong-bin²

(1. Dept. of Computer Science, Henan Normal University, Xinxiang Henan 453007, China; 2. Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: Introduce the exception handling machine. In order to understand it, introduce the exception of Java and workflow management system in detail.

Key words: Exception Handling; Software Design; Java; Workflow Management System

异常指正常情况之外的情形, 发生异常并不意味着一定发生错误。软件系统中出现异常非常频繁, 甚至操作系统, 因此我们认为非常健壮的系统也经常出现异常。在实际工作中, 即使发现并处理了大量的异常, 很多情况下系统的崩溃仍然是由于异常引起的, 分析数据也表明, 对异常不合适的处理会引起系统崩溃^[1]。为了增强系统的健壮性和可靠性, 许多程序设计语言引入了异常处理机制。研究表明^[1] 异常更多地发生在现实世界中, 因此, 现在对异常处理的研究已经渗透到了软件设计的许多方面。

1 软件设计问题

在 20 世纪 50 年代, 伴随着第一台电子计算机的问世, 经过训练的数学家和电子工程师开始以写软件作为职业。这个时期大多数软件是由使用该软件的个人或机构研制的, 软件带有强烈的个人色彩, 软件设计是在某个人的头脑中完成的一个隐藏的过程。到了 60、70 年代软件开始作为一种产品被广泛使用, 出现了软件作坊, 专职应别人的需求写软件, 软件开发的方法仍沿用早期的个体化软件开发方式, 但由于软件的数量急剧膨胀, 需求也日趋复杂, 失败的软件开发项目屡见不鲜, 即出现了所谓的软件危机。软件危机的原因, 一方面是与软件本身的特点有关; 另一方面是与软件开发和维护的方法不正确有关。

60 年代末提出了软件工程的概念, 按照工程化的原则和方法组织软件开发工作, 解决了软件开发和维护的方法问题, 但软件设计技术中的问题是软件工程无法解决的。

为了增强系统的健壮性, 避免程序错误的运行, 程序设计

语言的编译器会尽可能地发现程序中的问题。但有些问题(编程错误或系统错误)只有在运行期间才会发生, 必须在运行期间才能解决, 需要软件设计人员考虑到这些情况, 并传递相关的信息以便在运行过程中正确地处理这些问题。有些程序设计语言, 设置了一个是否正常的状态变量, 通过检查该变量的值发现异常; 还有一些语言需要每个函数返回自身运行是否正确信息, 正常情况和非正常情况用不同的返回值来表示。无论哪种方法, 都存在以下问题:

(1) 如果程序员假设程序在正常情况下运行而没有处理可能出现的非正常情形, 应用程序实际上充满了各种潜在的问题, 一旦发生, 可能会导致系统崩溃。

(2) 因为程序中充满了错误检测语句, 程序难以阅读。

(3) 由于程序可能遭遇各种各样的错误, 程序不可能包括所有的错误处理。

(4) 因为状态变量的值或者函数的返回值是在程序运行中进行检查的, 如果希望修改返回值和状态值将非常困难。

上述软件设计中对异常情况处理的技术问题是造成软件危机的原因之一, 为此, 一些程序设计语言提供了异常处理机制, 把异常处理的技术固化到程序设计语言中。异常机制包含两个方面, 即抛出异常和捕获异常。抛出一个异常意味着发出一个错误信号, 而捕获一个异常是指得到和处理抛出的错误信号。引入异常处理机制简化了错误控制代码, 我们不再需要检查一个特定的错误, 然后在程序的许多地方对其进行控制, 我们只需要在一个地方即异常控制器中处理问题。这样不仅减少了代码量, 而且将用于描述具体操作的代码与专门纠正错误的代码分隔开。

2 异常处理机制

在发生异常时, 可能不知道如何处理该异常, 但肯定不能

按照正常情况继续执行了。这时, 由运行的语言或系统抛出异常, 抛出异常意味着出现了不能按照正常的控制流处理的不正常的情况。抛出异常后, 异常传播机制将控制流转向异常处理器, 异常处理器是处理异常的指令集合。但实际处理问题时, 收到异常信息的异常处理器如果没有足够的信息处理该异常, 异常传播机制将把异常传播到上一层的异常处理器, 直到找到能处理该异常的异常处理器。

2.1 异常处理模式

处理模式规定控制流的转移方式, 文献[2] 中提供了异常处理机制可以采纳的处理模式:

(1) 无条件转移模式。在无条件转移模式中, 需在异常处理器前加上标号, 抛出异常后, 与执行程序设计语言中的 Goto 语句类似, 无条件中止正在执行的操作而转向标号位置开始执行。由于无条件转移机制本身难以维护和易产生错误的缺陷, 以及异常处理本身对健壮性和高效性的需求, 异常处理结构一般不采用无条件转移模式。

(2) 中断模式。抛出异常后, 该模式中止正在执行的块, 控制流转向异常处理器, 完成异常处理后, 控制流返回到调用点继续执行, 与没有发生异常程序正常执行完警戒区中的内容一样。大多数程序设计语言中的异常处理结构采用了这种模式。

(3) 重试模式。它是具有特殊处理语义的中断模式, 即失败后重新执行。在这种模式中必须指出重新开始执行的位置。一般情况下, 重新执行的位置是警戒区的开始, 其他位置没有什么实际意义。如果异常处理块中有多个异常处理器, 这些异常处理器的重试位置必须相同。由于这种模式可以用程序设计语言中的循环结构和中断模式结合起来表示, 所以异常处理结构不采用这种模式。

(4) 恢复模式。在恢复模式中, 抛出异常后, 控制流转移到异常处理器, 异常处理器负责纠正当前未能完成的操作, 然后回到抛出异常的位置继续执行。在这种模式中调用异常处理器的过程与调用常规程序一样, 不同之处在于执行完异常处理器后回到抛出异常的位置继续执行, 抛出异常的位置不固定。恢复模式的特点决定了该模式比较容易实现, 因此, 有些异常处理结构采用了恢复模式。

综上所述, 异常机制的处理模式一般采用中断模式或恢复模式, 但无论采用哪种模式, 都把异常处理代码与警戒区分开。警戒区代表一个特殊的代码区域, 在此区域有可能产生异常, 在警戒区的后面跟随异常处理器, 如果运行警戒区中代码时抛出异常, 则通过异常传播机制寻找相应的异常处理器。

2.2 异常传播机制

异常传播机制将控制流转向异常处理器, 传播模式有抛弃和恢复两种类型, 分别对应处理模式中的中断和恢复模式^[3]。抛弃传播意味着控制流将不能再回到抛出异常点, 抛出异常点到处理器之间的代码不再被执行; 恢复传播意味着控制流将回到抛出异常点, 但在处理器中可以改变为抛弃传播。抛弃传播和恢复传播模式能够共存于同一个异常处理机制中。在声明或抛出异常时异常与传播模式结合在一起, 按照不同的机制进行传播, 异常传播机制指寻找异常处理器的方式, 一般情况下, 采用动态传播机制。文献[4] 提供了一种通过词汇链寻找异常处理器的静态传播机制。

在动态传播机制中, 调用栈是被调用的方法序列, 传播机制沿调用栈从抛出异常的方法开始寻找处理异常的方法, 直到找到合适的异常处理器为止。如果找到异常处理器表, 表中有多个异常处理器, 传播机制没有指定具体的异常处理器, 按照一致性、最近、具体化三个指导原则从表中选择具体的处理器。一致性指选择与传播的异常相匹配的处理器; 最近原则指传播路径上先找到的处理器; 具体化原则指最接近最精确的处理器。软件设计人员可以设置上述指导原则的优先权, 通常一致性原则优先权最高, 最近原则次之, 具体化原则最低。所以很多情况下选择的异常处理器不一定是最合适的, 它是距离抛出异常的方法最近的适当处理器。

静态传播机制使用静态名字绑定, 编译时异常与相应的异常处理器已联系在一起。执行警戒区代码抛出异常后, 沿着词典定义的词汇寻找相应的异常处理器。

2.3 异常与运行环境

异常处理机制与运行环境有关, 并发环境下的异常处理机制比顺序环境下的异常处理机制复杂得多。异常处理机制与下面的执行属性密切相关:

(1) 执行状态。独立运行的状态信息, 包括局部和全局数据、当前的执行位置、活动记录。

(2) 线程。进程内部单一的一个顺序流, 多个线程可以并发执行, 改变执行状态。

(3) 互斥。访问共享资源时的一种机制, 顺序化并发执行的线程。

无论在顺序环境还是并发环境, 异常处理机制对控制流的改变直接影响执行状态, 并发环境下与线程、互斥密切相关。

2.4 异常的层次

面向对象的继承是类层次的一个重要方面, 在某个类下定义的子类将继承它的父类的特性, 并且能够具有它自己的特殊性质。异常的层次与面向对象中的类层次相似, 也具有继承性, 因此软件设计人员能够沿异常的层次, 在多个层次位置上处理异常。

通过继承来创建子异常类时, 子类不能有多个父类, 它只能继承一个父类^[5]。因在声明异常时, 将指定它的传播模式。从这个角度出发把异常分为抛弃模式异常、恢复模式异常、抛弃/恢复模式异常三种类型。一般情况下继承限于同类异常, 如果子类异常与父类异常的传播模式不同, 当抛出的子类异常被父类异常的处理器捕捉后, 传播机制会导致一系列问题。

3 面向对象语言 Java 中的异常处理机制

面向对象语言如今已经成为最流行最重要的语言。许多面向对象语言都提供了异常处理机制, Java 语言中的异常处理机制具有代表性。

3.1 异常与异常类

Java 中的异常又称为例外, 为了能够及时有效地处理程序中的例外情形, Java 中引入了异常和异常类。作为面向对象的语言, 异常与其他语言要素一样, 是面向对象规范的一部分, 是异常类的对象。异常在 Java 中被称为 Error 或 Exception, 是 Throwable 或 Error 的子类, 出现在程序中的原因是无法建立动态链接或运行时出错。

Java 中定义了很多异常类, 每个异常类都代表了一种运行错误, 类中包含了该运行错误的信息和处理错误的方法等内容。除此之外, 对某个应用程序所特有的运行错误, 则需要编程人员根据程序的特殊逻辑在用户程序中自己创建用户自定义的异常类和异常对象。这种用户自定义异常主要用来处理用户程序中特定的逻辑运行错误。每当 Java 程序运行过程中发现一个可识别的运行错误时, 即该错误有一个异常类与之对应时, 系统都会产生一个相应的该异常类的对象, 即产生一个异常。

3.2 异常处理机制

一旦一个异常对象产生了, 系统中就用相应的机制来处理它, 保证整个程序运行的安全性, 这就是 Java 的异常处理机制, Java 的异常处理机制采用中断模式^[6]。它由下面三部分组成(每一部分分别与 Java 关键字联系在一起):

(1) Throws。它后面跟随全部潜在的异常类型, 通知客户程序员该方法会抛出异常, 客户程序员可以方便地加以控制。当编写一个新的方法时, Java 程序员明确指出调用者可以处理的潜在错误状态, 所有这些潜在的错误状态都是方法定义中的一部分, 任何调用该方法中的代码必须处理这些错误状态, 这由 Java 编译器强制完成。

(2) Throw。它后面跟随异常对象, 表示抛出该异常对象给方法的调用者。用户自定义的异常不可能依靠系统自动抛出, 而必须借助于 Throw 语句来定义何种情况算是产生了此种异常对应的错误, 并应该抛出这个异常的新对象。由于系统不能识别和创建用户自定义的异常, 所以需要编程者在程序中的合适位置创建自定义异常的异常, 并利用 Throw 语句将这个新异常对象抛出。

(3) Try-Catch-Finally。它将所有可能抛出异常的代码部分放入 Try 块(警戒区)中; 将所有异常及其处理部分紧跟在 Try 块的 Catch 块中; Finally 块是可选的, 如果定义了 Finally 块, 不管有无异常产生, 它都会被执行。因此, 在把控制权交给其他程序之前, 用它处理任何必要的清理工作(如关闭文件和流, 释放系统资源等)。Finally 块的目的是把系统恢复到应该处于的状态。

在 Java 程序中, 异常对象是依靠以 Catch 语句为标志的异常处理语句来捕捉和处理的。Java 语言规定, 每个 Catch 语句块都应该与一个 Try 语句块相对应, 这个 Try 语句块用来启动 Java 的异常处理机制, 可能抛出异常的语句, 包括 Throw 语句, 调用可能抛出异常方法的方法调用语句, 都应该在这个 Try 语句块中。当异常发生时, 检查与当前方法相联系的 Catch 子句表。每个 Catch 子句包含其有效指令范围, 能够处理的异常类型, 以及处理异常的代码块地址。Try 块产生的异常对象被第一个 Catch 块所接收, 则程序的控制流就跳转到这个 Catch 语句块中, 语句块执行完毕后就退出当前的方法, Try 块中尚未执行的语句和其他的 Catch 块将被忽略; 如果 Try 块产生的异常与第一个 Catch 块不匹配, 系统将自动转到第二个 Catch 块进行匹配, 如果第二个仍不匹配, 就转向第三个……, 直到找到一个可以接收该异常对象的 Catch 块, 即完成控制流的跳转。如果所有的 Catch 块都不能与当前的异常对象匹配, 则说明当前方法不能处理这个异常对象, 程序流程将返回到调用该方法的上层。如果这个上层方法中定义了与所产生的异常对象相

匹配的 Catch 块, 控制流就跳转到这个 Catch 块中, 否则继续回溯到更上层的方法。如果所有的方法中都找不到合适的 Catch 块, 则由 Java 运行系统来处理这个异常对象。此时通常会终止程序的执行, 退出虚拟机返回到操作系统, 系统调用一个缺省的异常处理模块, 并在标准输出上打印相关异常信息。由于虚拟机从第一个匹配的 Catch 子句处继续执行, 所以 Catch 子句表中的顺序是很重要的。综上所述, 异常的处理主要包括捕捉异常、程序控制流跳转和异常处理语句块的定义。

4 workflow 管理系统中的异常机制

并行操作是以事务为单位进行的, 事务是逻辑工作单位, 是定义的一组操作序列, 具有原子性、一致性、隔离性和持续性。

4.1 并行处理的 CA 模型

在并行处理中, 同时执行多个线程。文献[7]把协作(CA)事务的集合作为并行处理系统的动态结构模型。CA 事务中并行的线程以通信和协作为基础, 通信模式有共享变量模式和消息传递模式, CA 事务中的线程有共同的目标, 每个线程仅完成总目标的一部分。CA 事务在一些对象集合上执行一组操作, 并发的多个线程合作完成这些操作, 接口部分指定了 CA 事务要操作的对象和操作这些对象的线程。

为了提高容错能力, CA 事务把会话、事务和异常处理结合在一起, 建立了统一结构参考模型, 会话保证协同操作活动(活动是任务的基本执行步骤)的封闭性, 控制线程之间的并行和通信; 事务保证访问共享变量时的一致性, 确保 CA 事务具有原子性、一致性、隔离性和持续性; 异常处理提供系统发生软硬件错误时的恢复机制, 一旦 CA 事务中某个线程抛出异常, 必须通知其他线程, 把线程的控制流转向它们各自的异常处理器, 同时把正在使用的外部变量恢复到正常状态。CA 事务内部不能创建和结束线程。

4.2 并行处理的异常机制

并行处理的异常机制比顺序执行环境下传统的异常处理机制复杂得多, 因为异常处理过程包含并行成分; 其次分布环境的不同位置可能同时抛出异常。但它与传统的异常处理机制也有许多相同之处, 首先需要定义警戒区, 在警戒区后面跟着相应的异常处理器, 如果警戒区代码执行时抛出异常, 改变控制流的模式也有无条件转移、中断、重试、恢复, 使用最多的仍然是中断模式。异常抛出后, 相应的处理器必须捕捉异常并处理它, 如果本地环境没有足够资源, 需要把异常传播到上面一层。并行处理的异常处理机制包括下面几个方面:

(1) 声明异常。CA 事务有两种类型异常: CA 事务内部能够处理, 在 CA 事务定义中声明并指定处理的线程; CA 事务内部无法处理, 需要传播到执行 CA 事务的上层环境中处理, 它在 CA 事务接口中声明。

(2) 异常处理和传播。如果 CA 事务内部能够处理抛出的异常, 则控制流按照中断模式转向相应的异常处理器, CA 事务的所有参与线程都要调用相应的处理器, 如果 CA 事务内部不能处理抛出的异常, 传播机制选择一种传播模式把异常传播到 CA 事务的上层环境。传播模式有两种: 预先定义或动态定义一个主线程, 由它负责向上层发送传播异常的信息; 由抛出异常的线程负责发送传播异常的消息。

(3) 控制流的转向。异常抛出后按照中断模式转移控制流, 控制流转移到相应的异常处理器。与传统机制类似, 首先在 CA 事务内寻找异常处理器, 如果找不到相应的异常处理器, 传播异常到上层寻找, 直到找到相应的异常处理器为止。

(4) CA 事务的外部资源。CA 事务的线程通过共享外部资源协作, 因此外部资源的状态可能被改变, 如果 CA 事务抛出异常, 应该把外部资源恢复到合理状态。

(5) 异常解析。处理 CA 事务同时抛出的多个异常时, 一种方案是给每个异常加权, 然后按照异常的优先权先后处理这些异常。该方案很简单, 因相关信息太少, 实现起来难度较大, 因此常采用异常图描述并发控制中的异常。异常图描述了异常的层次结构, 把抛出的所有异常的析取作为异常解析。

4.3 workflows 管理系统中异常处理机制的讨论

目前, 开发和运行企业过程的最好工具是 workflows 管理系统, workflows 是业务过程的一个计算机实现, 而 workflows 管理系统则是这一实现的软件环境。workflows 管理系统具有实时分布性, 实时分布系统一旦在某个节点发生错误, 将影响到整个计算机系统。目前大多数 workflows 管理系统都提供了异常处理策略、系统容错能力、故障恢复策略、数据保护及数据恢复方法。

许多研究人员^[8]综合了面向对象语言环境的异常处理思想和并发控制的事务概念在 workflows 系统中, 提出了引入 workflows 管理系统中的异常机制, 他们使用事务概念消除分布系统中异常对协作应用程序的影响, 用时间触发实时系统中的与时间相关的异常。这些思想主要包括以下几方面:

(1) 异常信号。一旦发生意外, 立即产生异常对象。异常对象属于一个或多个类, 并具有属性。不同类型的异常使用类标志符区别, 属性包含异常环境的详细信息。异常按层次组织。

(2) 异常处理。抛出异常后, 停止正常的控制流, 转向能够调用的异常处理器。异常处理与环境相关, 同一个异常在不同的环境下表现不同, 或者被处理或者被传播到上一层。当过程 C_0 抛出异常 X 时, C_0 首先调用 C_1 , 如果 C_1 不能处理异常 X, 则调用 C_2 , 若 C_2 也不能处理 X, 继续调用 C_3 …… , 直到调用 C_n , C_n 中提供异常 X 的处理器 H。另外, 可以把异常处理器与异常本身结合在一起^[8]。异常类的分层机制允许异常处理器处理相应异常的子类。

(3) 恢复控制流。异常处理器经常在过程 C_n 的环境下执行, 在执行异常处理器 H 时, 所有过程 $C_k(k < n)$ 都处于挂起状态, 当处理器 H 执行完毕后, 有下面几种方案恢复控制流:

指定恢复某个过程 C_k , 对于 $j < k$ 的所有过程 C_j 都被终止, 从 C_k 抛出异常后的语句继续执行; 在上述方案中, 不是从 C_k 抛出异常后的语句继续执行, 而是从过程 C_k 的起始端开始执行; 异常处理器 H 中并没有恢复语句, 所有的调用 C_{n-1} 都被终止, 控制流转向 C_n 。将被终止的过程, 对超过生命周期的对象执行清除操作。workflows 环境允许终端用户处理异常。

根据对 workflows 管理系统中异常处理机制的研究, 笔者提出了 workflows 管理系统数字银行项目中异常处理的一种设计方案^[9]。

5 结束语

现在, 异常处理机制已经引起人们的普遍关注, 对异常的研究已经渗透到了计算机的许多领域。本文介绍了异常处理机制, 描述了面向对象语言 Java 中的异常结构, 讨论了 workflows 管理系统中的异常处理机制。

参考文献:

[1] ewayne E Perry, *et al.* Current Trends in Exception Handling[J] . IEEE Trans. Software Engineering, 2000, 26(9) : 817-819.

[2] S Yemini, D M Berry. A Modular Verifiable Exception Handling Mechanism[J] . IEEE Trans. Software Engineering, 2000, 7(2) : 214-243.

[3] Peter A Buhr, W Y Russell Mok. Advanced Exception Handling Mechanisms[J] . IEEE Trans. Software Engineering, 26(9) : 820-836.

[4] J L Knudsen. Exception Handling: A Static Approach[J] . Software Practice and Experience, 1984, 14(5) : 429-449.

[5] T A Cargill. Does C ++ Really Need Multiple Inheritance[J] . Proc. Advanced Computing System Assoc, 1990, 12(4) : 315-323.

[6] Bruce Eckel. Thinking in Java[M] . Beijing: China Machine Press, 2000. 240-281.

[7] Alexander Romanovsky, Jie Xu, Brian Randell. Coordinated Exception Handling in Real-time Distributed Object Systems[J] . Computer Systems Science & Engineering, 1999, (4) : 197-207.

[8] Claus Hagen, *et al.* Exception Handling in Workflow Management Systems[J] . IEEE Trans, Software Engineering, 2000, 26(10) : 943-958.

[9] 张聪品, 糜宏斌. workflows 管理系统中的一种异常处理[J] . 计算机工程, 2003, 29(15) : 118-120.

作者简介:

张聪品, 女, 讲师, 硕士, 研究方向为异常的静态分析方法; 赵琛, 男, 副研究员, 硕士生导师, 博士, 研究方向为编译技术与软件测试; 糜宏斌, 男, 副研究员, 博士, 研究方向为数据库与网络工程。

(上接第 85 页)

在应用型 GIS, 如土地利用规划管理信息系统的开发中, 用户的需求常常不定, 并且会随着业务的变化而提出新的要求。将设计模式应用到此类 GIS 软件开发中, 能够得到较好的软件结构, 并提高开发效率。当然, 设计模式并不是万能的, 设计模式的本质是“封装变化”, 所以只有在 GIS 产品需求强调变化时, 它才有用武之地。在满足需求的基础上, 选择最简洁的设计方案要比套用尽可能多的设计模式更重要。如何在 GIS 软件开发中正确应用设计模式, 以提高软件开发的质量, 还有待进一步研究。

参考文献:

[1] rich Gamma, Richard Helm, Ralph Johoson, *et al.* Design Patterns:

Elements of Reusable Object-Oriented Software[M] . Massachusetts: Addison-Wesley Publishing Company Inc. , 1995.

[2] 阎宏. Java 与模式[M] . 北京: 电子工业出版社, 2002.

[3] Billy Hollis, Rockford Lhotka. VB. NET 程序设计教程[M] . 康博. 北京: 清华大学出版社, 2001.

[4] 张世博, 周树杰, 闵艳. Java 程序开发中的设计模式[J] . 微型电脑应用, 2002, 18(9) : 45-47.

[5] 吴小锋, 张新长, 张润朋. 基于 GeoMedia WebMap 的 WebGIS 研究与开发[J] . 计算机应用研究, 2002, 19(7) : 112-114.

[6] 王晓军. 用 GeoMedia 组件进行二次开发的实践[J] . 石油工业计算机应用, 2001, (3) : 20-23.

[7] 李满春, 陈刚, 姚志军, 等. 县级土地利用规划管理信息系统的分析与设计[J] . 国土资源遥感, 2003, (1) : 65-69.

作者简介:

曹志刚(1980-), 男, 硕士研究生, 研究方向为地理信息系统与开发。