异常处理——一种提高软件健壮性的方法*)

姜淑娟[1,2] 徐宝文[1,3]

(东南大学计算机科学与工程系 南京210096)[1] (中国矿业大学计算机科学与技术学院 徐州 221008)[2]

(江苏省计算机信息处理技术重点实验室开放基金 苏州大学 苏州215006)[3]

Exception Handling——An Approach to Improving Software Robustness

JIANG Shu-Juan[1,2] XU Bao-Wen[1,3]

(Department of Computer Science & Engineering, Southeast University, Nanjing 210096)[1]

(School of Computer Science & Technology, China University of Mining & Technology, Xuzhou 221008)[2]

(Jiangsu Province Key Laboratory of Computer Information Processing, Soochow University, Suzhou 215006)[3]

Abstract Exception handling is a technique that tests and handles exception events. Unlike the traditional methods that usually deal with exceptions at later design and implementation phases and easily result in many problems, we emphasis that sufficient attention should be paid to software exception handling during the development of the software requirements definition. By enforcing this policy through all phases of software development, the level of robustness can be improved considerably. In this paper, the concepts of exception handling are firstly introduced, then the methods of exception handling are discussed, all kinds of exception handling methods and tools are also compared. The current problems and future directions are analyzed at the end of the paper.

Keywords Exception handling, Control flow, Robustness, Program analysis, Error detecting and handling

1 引言

随着软件系统功能的不断完善和加强,软件的越来越复杂,对软件健壮性的要求也越来越高。人们已提出了多种提高软件健壮性的方法[11,13,14],异常处理是其中一种比较有效的方法。但目前异常处理还没有引起人们足够的重视[7],程序设计人员经常把异常处理推迟到设计阶段的后期或是编码阶段来考虑[8]。然而,一个很小的异常就有可能造成整个系统的崩溃。如果在软件生命期的每一个阶段都适当地考虑和处理可能出现的各种异常,那么软件的健壮性就能够得到提高,从而也就可以降低软件的费用。文[7]指出,可能出错的系统和不可能出错的系统的主要区别在于当系统出错时它能否被修复。

本文介绍了异常处理的基本概念;讨论了三种具有代表性的异常处理的方法;介绍了几种常见的异常处理工具;讨论了异常处理技术目前存在的问题,并给出了本文的结论。

2 基本概念

异常处理的概念是由 John Goodenough 在1975年首次提出的[1]。异常有多种不同的定义[1,2,3,5],目前为大多数人所接受的定义是:异常是在程序的执行过程中检测到的不正常的事件,它使程序不能继续沿着正常的路径执行。这些不正常的事件可能是由错误引起的(例如除以0),也可能是由软件设计人员自己定义的某一需要特殊处理的事件引发的。异常的来源有硬件错误、软件错误和状态错误等。异常通常分为预定义异常(也叫系统定义异常)和用户定义异常两类,预定义异常是隐式定义的并由硬件和操作系统来检测;用户定义异常是在应用级上检测和定义的。

异常处理是当一个程序在执行过程中引发一个异常时负责使程序的正常控制流改变到一个异常的控制流(即转到一个异常处理程序)的机制。一个异常处理机制可能是语言本身固有的,也可能是通过函数库的调用而增加到语言上的功能。

异常处理程序是用于响应异常的一段程序。一般情况下,异常处理程序与正常程序是分开的,在程序正常执行的情况下这部分代码不会被执行。每个异常处理程序都与一个实体相关联,它用来响应该实体内所产生的异常。实体可以是一段程序代码,也可以是一个过程、一个语句、一个对象等等。事实上,实体通常称为保护区或保护块,发生异常的点则称为异常引发点。

2.1 异常处理模型

当一个异常被引发并且在执行了相应的处理程序之后,系统应继续正常控制流的执行,那么正常的控制流应从什么地方开始呢?一般而言,至少有如下两种异常处理模型:

(1) 继续模型(Resumption Model):程序从实体中的异常引发点往后继续执行,通常在发生异常的情况下,继续从该点往后执行已不可能,因此继续模型较少使用。

(2) 终止模型(Termination Model):废弃异常引发点后的一段代码,中止该程序块的执行。在终止模型中至少可以分为四种不同的终止方式。

·返回(Return):即在异常处理程序执行后把控制流转到

*)本文得到国家自然科学基金(60073012)、江苏省科技攻关项目(BE2001025)、江苏省自然科学基金(BK2001004)、江苏省"三三三"人才基金、武汉大学软件工程国家重点实验室开放基金和江苏省计算机信息处理技术重点实验室(苏州大学)基金的资助。姜淑娟 副教授,在职博士生,主要从事程序设计语言、软件工程等方面的教学与研究工作。徐宝文 教授,博士生导师,主要从事程序设计语言、软件工程、知识与信息获取技术等方面的教学与科研工作。

紧接在保护区之后的语句，保护区中异常引发点后的代码废弃。

·严格终止(Strict Termination)：即在异常处理程序执行后中止程序的执行并把控制流转到操作环境下。

·重试(Retry)：即在异常处理程序执行后中止产生异常的那段程序，重新执行它，即再提供一次正常执行的机会。

·传播(Propagation)：即将异常交给上层模块中的异常处理程序处理，分为显式传播和隐式传播。前者也叫重新引发，要求任何被传播的异常必须在处理程序中显式地引发。CLU使用这种方式。隐式传播也叫自动传播，如果在调用者内部没有发现已引发的异常的处理程序，则该异常就自动地传播到上一级的调用者，直到找到一个处理程序。Ada使用这种方式。

尽量采用显式传播的方式传播异常；采用终止模型的处理方式更容易构造可靠的系统，虽然继续模型更灵活，但使用起来比较困难。

2.2 异常处理程序的绑定方式

当程序运行过程中引发一个异常时，需要激活一个处理程序来处理异常。绑定处理程序有三种不同的方式：

(1) 静态方式：一个处理程序被静态地绑定到一个保护区上，该处理程序用来处理在该保护区的执行期间所引发的所有异常。处理程序的绑定独立于程序的控制流，当异常引发时用不着查找处理程序。

(2)动态方式：处理程序的绑定依赖于程序的控制流。当某个异常引发时，就要实时地查找处理程序来处理异常，在编译时不能确定由哪个处理程序来处理异常。异常处理程序是在程序的执行过程中动态地确定的，通常是通过执行一个语句使一个处理程序对应于某个特定的异常。如：在PL/I，通过一个语句ON来动态地绑定一个处理程序到一个特定的异常，这种绑定关系一直持续到下一个与该特定异常相连的ON语句出现或引发该异常的块退出为止。

(3)半动态方式：是前面两种方式的混合。本地的处理程序可以用静态绑定的方法绑定到引发异常的保护区上，如果没有一个处理程序绑定到所引发的异常上，就可通过动态的方法查找一个合适的处理程序。首先，在那些连接到本保护区的处理程序中动态查找，如果没找到，则异常处理机制就向引发该异常的方法的调用者发出一个异常的信号。然后，就在该调用者的保护区相连的处理程序中查找，若还找不到合适的处理程序，则沿调用链继续向上查找，直到找到一个合适的处理程序为止。

异常处理程序的绑定方式与异常传播联系很密切。如果处理程序的绑定方式是采用静态绑定的方式，则没有异常的传播，因为绑定在编译时就完成了；只有采用半动态绑定方式或动态绑定方式时，才有异常的传播。

2.3 清理活动

不管代码是正常地执行还是被一个异常打断，程序都应保持一个一致的状态。在这种情况下，就要求程序在中止之前，做一些清理活动。清理活动可以通过撤消一些已做的动作来还原程序到一个有效的状态，释放那些不用的资源。清理活动可以通过异常处理机制的特定设计来支持，如Java语言中的finally结构，就可以放置清理代码。

3 异常处理方法

传统的方法经常是把异常处理推迟到设计阶段的后期或是编码阶段进行[8]，这样容易引起一定的问题[9]。我们主张把异常处理放在软件生命期的早期阶段[8,10~12,14]，这就是说在进行需求分析时就要考虑到异常处理的问题，并且在以后的每一个时期的开发过程中都要作为一个重要的方面来考虑。de Lemos 和 Romanovsky 在文[12]中提出了一种方法：在软件的生命期的每一个阶段都定义一个异常类以及它的相应的异常处理程序，随着软件开发的进展，这些异常类及其相应的处理程序被一步一步地具体化，如图1所示。

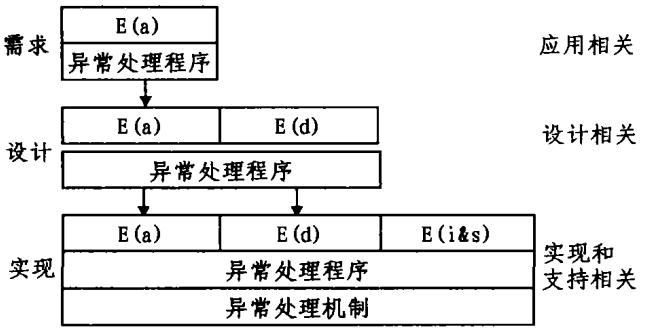

图1 软件生命期中不同阶段的异常处理

异常处理的方法有很多。例如，Cui 和 Gannon 在文[3]中提出的面向数据的异常处理的方法、Sinha 和 Harrold 提出的控制流分析法和控制依赖分析法[16]、Choi 等提出的要素控制流法[17]、Maxion 和 Olszewski 提出的鱼骨图分析法[11]，下面介绍其中几种典型的方法。

3.1 鱼骨图(fishbone diagam)方法

鱼骨图又叫原因—效果图，是由 Ishikawa 于1982年首先提出的[6]。它是一种利用图形结构的方式来帮助程序设计人员记忆在软件设计过程中应该需要注意的各个方面的内容的技术，鱼骨图广泛应用于质量控制[11]。鱼骨图同样可以应用于异常处理的分析过程中，用它可以帮助设计人员预测异常发生的条件。Maxion 在文[11]中给出了一种异常条件的分类方法，如图2所示。在图2中描述了在软件系统中可能会遇到的异常，它的形状像一个鱼骨，在鱼头的地方是应该避免的“异常失败”，每根鱼刺代表一种可能引发异常失败的问题类型，在每根鱼刺上标有该类问题的举例。在图2中异常错误被分为八个方面，即计算问题、硬件问题、输入输出与文件问题、库函数问题、数据输入问题、函数/过程调用的返回值问题、外部用户/客户问题、空指针与内存问题。如果取每类问题的英文单词的第一字母，就是所谓的异常 CHILDREN 问题。

了解了图2所示的各种异常的鱼骨图结构，软件的设计人员在设计期间可以利用此图来考虑和处理在软件运行期间可能出现的各种异常。

3.2 控制流分析法(Control-Flow Analysis)

存在于程序中的控制流关系可以用控制流图(Control-Flow Graph，即 CFG)来表示，在 CFG 中结点表示语句，边表示语句之间的控制流，控制流图又分为过程内的控制流图(Intraprocedural CFG)和过程之间的控制流图(Interprocedural CFG)。Sinha 和 Harrold 在文[16]中给出了一种描述 Java 语言的过程内的和过程间的控制流的方法。

用该方法构造的 CFG 与别的 CFG 的区别在于：该 CFG 包含了表示 throw 语句、catch 语句和 finally 语句的结点；也包含这三种语句引起的正常和异常的控制流所表示的边。其中：

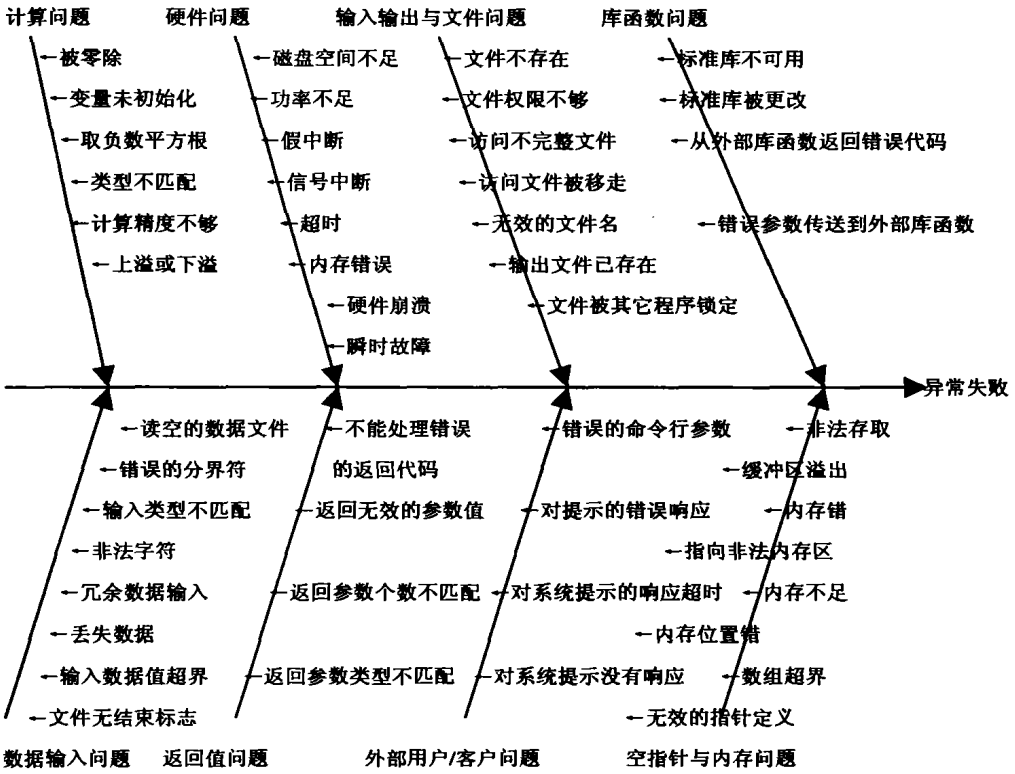

图2 各种异常的鱼骨图结构

(1)一个 throw 语句可有一个后继结点也可有多个后继结点,这些后继结点是由 throw 语句所能引起的异常类型来决定的,通过对该异常的类型进行推断来决定这些异常的潜在的类型,为每一种 throw 语句所能引起的异常类型生成一条边,作为 throw 结点的输出边。有关类型推断的详细内容请参见文[19~21]。

(2)该 CFG 包含异常退出结点,一个异常退出结点表示该 throw 语句所引发的异常向外传播了,一个方法中的异常退出结点数与该方法所直接引发的并且没有处理的异常数是一样多的。

(3)在该过程内的 CFG 中,对于那些处理直接引发异常的处理程序的 catch 语句结点有输入边,而对那些仅仅处理间接引发异常的处理程序的 catch 语句结点没有输入边,然而在描述过程间的 CFG 时,这些结点就有输入边。

(4)对 finally 语句生成一个单独的 finally 块,对于不管是以正常方式到达 finally 块的语句还是以异常方式到达 finally 块的语句的点都对 finally 块有一个输入边[8,10~12,14]。

一个程序 P 的过程间的控制流图(ICFG)是由该程序的每一个过程(或方法)的 CFG 以及用来连接这些 CFG 的调用边和返回边来组成的。在每一个调用点,用一个调用边连到被调用方法的入口结点,用一个返回边把被调用方法的退出结点连到相应的返回结点。在 ICFG 中用异常返回边来表示过程间的异常控制流,异常返回边连结两个过程,它一端是一个被调用方法的异常退出结点,另一端是一个 catch 结点,或是一个 finally 块的结点,或是一个调用该方法的过程的异常退出结点。Sinha 和 Harrold 在文[16]中给出了 CFG 的构造算法和 ICFG 的构造算法。

该方法准确地表示了在 throw 语句中能引起的异常类型,也准确地表示了穿过方法传播的异常的类型,这种表示对理解异常处理的结构提供了有用的信息。

3.3 要素控制流分析法

要素控制流图是由 Choi 等在1999年提出的一种表示程序中的过程内的控制流的方法[17]。这种方法特别适用于那些含有大量 PEI(Potential Exception-throwing Instruction)即潜在的异常抛出指令,如数组的存与取、变量的读与写、方法的调用等)的程序。它既可以表示显式引发的异常所引起的控制流变化情况也可以表示隐式引发的异常所引起的控制流变化情况。对于显式引发的异常,该方法生成的边与前面所讲的控制流方法相似;对于隐式引发的异常,该方法并没有从每一个 PEI 生成边,而是合并这些 PEI 到一个基本块(单入口和单出口),为该基本块中的能隐式引发异常的每一种类型生成一条要素边,把这些要素边连到异常处理块或退出块。该方法大大地简化了传统的 CFG,在文[17]中 Choi 等也给出了适用于 FCFG 的数据流分析算法。

4 异常处理工具

目前,异常处理的工具和环境有很多[18,22,23],如针对 Ada 语言的工具 ADAPT[24],使用该工具可以分析异常传播的路径,能够发现不可达的异常处理程序,还能够发现定义过但并没有使用的异常等;针对 Java 语言的异常处理工具 Jex;还有针对在不可获取软件源代码的情况下异常处理的工具 Xep 等。下面介绍几个常见的工具。

4.1 Xept 工具

Xept 是一个用于异常处理的软件工具。用这个工具可以根据代码对库函数的调用信息来进行异常检测和错误恢复的工作,这对那些程序开发人员不可能获得软件的源代码而仅仅可以存取到模块的接口的软件特别有用[18]。Xept 技术主要包括:

(1)对选定函数的接口和异常的规范用类似于 C 语言的简单语言来描述。

(2)根据该规范可以生成截获代码,该截获代码既可以调用相应的实际函数也可以把异常同应用所定义的异常处理程序或库所提供的异常处理程序联结起来。

(3)对象代码调用截获的函数而不是普通的函数。

Xept 虽然是为异常处理开发的,但它的方法也可应用到别的地方,如用该方法可以扩展已有函数或根据环境来修改函数;在 Xept 中,一个函数的规范准确地描述了它接口的形式、异常的类型以及如何处理异常,使用这种规范可以自动地生成不同形式的代码,如可以生成函数调用的路径和返回值的截获代码,这对进行功能测试很有好处;Xept 还提供了一个方便的方法把异常处理的代码从正常的应用代码中分离出来。

4.2 Jex 工具

Robillard 和 Murphy 在文[22]中给出了一个从 Java 源程序中获取异常信息的工具 Jex。用户给出 Java 的源文件和用户调用的包的列表,Jex 就可以获取该程序的包括 Java API 所引发的异常在内的异常结构的信息,这些信息可以帮助程序开发人员推断出穿过模块的异常流,并确定捕获到可能隐式引发异常的位置,并确定合适的异常处理程序放置的位置。对 Java 的每一个方法,Jex 能生成可能从这个方法引发异常的准确的异常类型。如果一个方法引发了一个异常,则引发异常的方法和类的名字就可以知道,如果一异常是由 runtime environment 引发的,则用 environment 来标识,所有生成的信息放在一个与分析的类相对应的 Jex 文件中。通过对生成的信息进行分析,开发人员可以为捕获不到的异常或不可预测的异常设计异常处理程序来提高软件的健壮性。

结论 异常处理是一种用来检测异常并对其进行处理的技术,在理论和应用两个方面的研究都取得了可喜的进展,但异常处理技术也还存在一些困难和问题,主要体现在以下几个方面:(1)异常处理还没有引起软件设计人员的足够重视,许多软件系统的设计并没有在设计的初始阶段就把异常处理考虑进去,而是到了设计阶段的后期或是到编码阶段才考虑到异常处理,这样势必会影响到软件的健壮性;(2)实际可用的异常处理的工具较少,特别是针对面向对象程序、并发程序、实时系统的工具就更加稀少,显然不能满足软件开发方法的需要;(3)异常处理技术比较单一,基本上还是采用像控制流、控制依赖和数据流等比较传统的方法,这对面向对象的程序和基于构件的程序来说就不太适用;(4)基于构件的软件开发方法的兴起,也为测试和维护这些使用该方法开发的系统引入了新的问题,构件的提供者独立地开发和测试构件,而构件的使用者在没有源代码的情况下独立在分析和测试这些构件的应用,这就为异常处理提出了新的跳战和机遇。

尽管异常处理技术存在这些问题,但它的作用是巨大的,人们对异常处理的理论和方法的研究也从未间断过,并且随着软件开发方法的发展而不断地得到改进和扩充。

异常处理今后的发展应考虑下面几个方面:①研究更有效、更准确的实用方法,特别是针对面向对象程序和基于构件的程序的异常处理的方法;②开发用于进行异常处理的有效工具,特别是针对基于构件的程序和实时程序的异常处理的工具;③研究软件生命期的每一个阶段的异常处理的工具,例如用于需求分析阶段的异常处理的工具,用于总体设计阶段的异常处理的工具,而不是仅仅研究用于编码阶段的异常处理的工具等。

致谢 在本文的写作过程中周毓明博士提出了许多有益的建议,在此表示感谢。

参 考 文 献

1 Goodenough J. Exception Handling: Issues and a Proposed Notation. Communications of the ACM, 1975, 18: 683～696
2 Gehani N H. Exceptional C or C with exceptions, Softw. Pract. Exper., 1992, 22: 827～848
3 Cui Q, Gannon J. Data-Oriented Exception Handling. IEEE Transactions on Software Engineering 1992, SE-18: 393～401
4 D rew S, Gough K. Exception Handling: Expecting the Unexpected, Computer Languages, 1994, 8: 69～87
5 Lang J, Stewart D. A Study of the Applicability of Existing Exception-Handling Techniques to Component -Based Real-Time Software Technology. ACM Transactions on Programming Languages and Systems, 1998, 20: 274～301
6 Ishikawa, Kaoru. Guide to Quality Control, Asian Productivity Organization, Tokyo, 1982
7 Howell C, Vecellio G. Experiences with Error Handling in Critical Systems, ECOOP Workshop 2000: Advances in Exception Handling Techniques, LNCS 2022, Springer-Verlag, Berlin, Heidelberg, 2001. 181～188
8 de Lemos R, Romanovsky A. Exception Handling in the Software Lifecycle. International Journal of Computer Systems Science and Engineering, 2001, 16(2): 167～181
9 Miller R, Tripathi A. Issues with Exception Handling in Object-oriented Systems. In: Proc. of 11th European Conf. on Object-Oriented Programming, Lecture Notes in Computer Science 1241, 1997. 85～103
10 van Lamsweerde A, Letier E. Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering, 2000, 26: 978～1005
11 Maxion R A, Olszewski R T. Improving Software Robustness with Dependability Cases. In: 28th Intl. Symposium on Fault-Tolerant Computing, 1998. 346～355
12 de Lemos R, Romanovsky A. Exception Handling in a Cooperative Object-Oriented Approach. In: Proc. of the 2nd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing, France, May 1999
13 Garcia A F, Beder D, Rubira C M F. An Exception Handling Software Architecture for Developing Fault-Tolerant Software. In: Proc. of the 5th IEEE HASE, USA, 2000. 311～320
14 Garcia A F, Rubira C M F. An Architectural-Based Reflective Approach to Incorporating Exception Handling into Dependable Software. In: A. Romanovsky, et al. eds. Exception Handling, LNCS 2022, Springer-Verlag, Berlin, Heidelberg, 2001. 189～206
15 Alleman G B. Exception Handling in CORBA Environments, galleman@niwotridge.com 8 March 2000, Revised, Oct. 2000
16 Sinha S, Harrold M J. Analysis and Testing of Programs with Exception-Handling Constructs. IEEE Transactions on Software Engineering, 2000, 26
17 Choi J-D, Grove D, Hind M, Sarkar V. Efficient and Precise Modeling of Exceptions for the Analysis of Java Programs. In: Proc. of'99 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering, 1999. 21～31
18 Vo K-P, Wang Y-M, Chung P E, Huang Y. Xept: A Software Instrumentation Method for Exception Handling, Eighth International Symposium on Software Reliability Engineering, 1997. 60～69
19 Chatterjee R, Ryder B G, Landi W A. Complexity of Points-To Analysis of Java in the Presence of Exceptions. IEEE Transactions on Software Engineering, 2001, 27
20 Palsberg J, Schwartzbach M. Object-Oriented Type Inference, Conference on Object-Oriented Programming Systems, Languages, and Applications, 1991. 146～161
21 Plevyak J, Chien A. Precise Concrete Type Inference for Object-Oriented Languages, Conference on Object-Oriented Programming Systems, Languages, and Applications, 1994. 324～340
22 Robillard M P, Murphy G C. Analyzing Exception Flow in java programs. In: Proc. of the 7th Annual ACM SIGSOFT Symposium on the Foundations of Software Engineering, Sep. 1999
23 Romanovsky A, Sanden B. Except for Exception Handling …. http://www.cs.ncl.ac.uk/research/trs /papers/735.pdf
24 Brennan P T. Observation on Program-Wide Ada Exception Propagation. TRIADA, 1993. 189～195