异常传播网络静态建模方法及特性分析

邱 翔 张 莉

(北京航空航天大学 计算机学院, 北京 100191)

摘 要: 异常传播现象已经成为软件开发者在异常处理过程中必须要考虑的主要问题之一. 以往研究者更多关注于异常传播路径以及异常抛出的根源,而忽略异常传播结构的网络化特征. 通过定义异常传播过程中方法之间的异常传递关系对异常传播结构进行网络化建模,提出了异常传播网络模型. 通过对 5 个开源软件包的异常传播网络进行度分布、传播异常类型层次分布、聚集系数和平均最短路径等网络特征的分析发现:异常传播网络的出度和入度基本服从幂律分布;异常传播过程几乎都与"不要传播抽象异常"原则相违背;由于异常传播过程的"弱边界性",异常传播网络并没有传统软件的高内聚特征. 上述分析成果可以为软件开发者在设计异常处理过程时提供指导和建议.

关 键 词: 异常传播; 异常传播路径; 网络化建模

中图分类号: TP 311

文献标识码: A **文 章 编 号**: 1001-5965(2012)04-0535-06

Static modeling method and characteristic analysis of exception propagation network

Qiu Xiang Zhang Li

(School of Computer Science and Technology, Beijing University of Aeronautics and Astronautics, Beijing 100191, China)

Abstract: In area of the exception handling, exception propagation is of catholic concern. Previous researchers focused more on exception propagation paths and the source of a thrown exception but ignored the networked characteristic of exception propagation. An exception propagation networked model was proposed based on the definition of exception transitivity relationship between two methods. A case study was presented for the analysis of main characteristics of complex network, which includes the degree distribution, the exception hierarchy distribution, the clustering coefficient and the average shortest path. The results show that the in-degree and out-degree distribution are basically subject to a power law distribution; Most exception propagation processes violate the rule of "Do not spread the abstract exception"; The exception propagation network appears the low cohesion feature due to the "Weak Border" characteristic. These empirical findings can provide the guidance for software developers in the design process for exception handling.

Key words: exception propagation; exception propagation path; networked modeling

软件系统功能的多样化和复杂化使软件鲁棒性问题已经成为业界关注的主要问题之一. 异常处理作为提高系统鲁棒性的重要手段,它将系统在异常情况下的正常控制流与处理该异常的控制流进行分离. 但遗憾的是,软件系统的复杂性使异常控制流的分析越发的困难. 在复杂的软件系统中,异常处理机制与编程语言紧密耦合[1]. 因此,本文关注的是 Java 语言的异常处理机制. 对于 Java 语言而言,一旦抛出异常,程序会搜索匹配该异常的第 1 个 catch 块进行异常处理. 由抛出异常

收稿日期: 2010-12-08; **网络出版时间**: 2012-04-20 09:14

网络出版地址: www.cnki.net/kcms/detail/11.2625.V.20120420.0914.003.html

基金项目: 国家自然科学基金资助项目(90818017)

作者简介: 邱 翔(1984 -),男,四川金阳人,博士生,qiuxiang008@126.com.

的方法到最终确定包含该异常类型处理块的方法所途经的调用链就是该异常类型的异常传播路径. 由于异常传播的全局性[2]、弱边界性[3],增加了软件维护过程的难度和成本.

以往研究者对异常传播现象进行分析时,关注的是对单一传播路径的抽取和异常抛出源点的确定,而在异常传播过程中存在多条异常传播路径都包含的方法节点,本文称为纽带节点,这类方法节点让软件系统中的异常传播路径不再是孤立的链式结构. 异常传播路径通过这些纽带节点进行关联,出现了异常传播的网络化现象[2].

随着系统科学的发展,复杂网络已作为理解和抽象复杂系统的有效手段. 通过对软件元素及其之间的关系进行网络化描述,并由此分析软件拓扑结构的特性及其演化规律,为软件的开发和维护过程提供理论支持[4-6]. 以方法调用关系为基础的软件网络图[4]具有高聚集系数和平均距离很小的小世界特性,其度分布基本符合幂律分布. 本文所讨论的异常传播网络的基础同样是方法调用依赖关系. 因此,本文也将关注它的小世界特性及其度分布的分布特征.

本文关注的是 Java 中的可检测异常(Compiled-Exception)[1],又称编译期异常,该类异常可以通过对代码或编译中间结果的静态分析完成对其传播路径的搜索. 如无特殊说明,下文所提及的异常传播路径指的是针对可检测异常的异常传播路径. 在以往研究工作的基础之上,首先对异常传播路径进行提取,然后通过定义异常传播路径中方法之间的异常传递关系对异常传播网络化特征进行统一描述,提出了异常传播网络模型. 最后基于5个开源软件包对度分布、传播异常类型的层次分布、聚集系数以及平均最短路径等指标进行分析.

1 相关研究

现有的关于软件静态结构的复杂网络建模研究都是借鉴传统软件工程的建模技术,主要是针对结构化软件和面向对象软件. 通过复杂网络理论,编程元素及其之间的关系可以用于构造软件系统的基本拓扑结构. 例如结构化软件的子过程或函数以及它们之间的调用关系[4];面向对象软件的类以及类之间的关系[4-5].

随着软件鲁棒性问题在业界重视程度的提高,异常传播这一研究领域也受到国内外研究者越来越多的关注. 其中比较著名的研究是文献[6]针对面向对象软件提出的 Jex 异常传播分析工具. 文献[7-8]提出了基于控制流的异常传播分析方法,该方法主要关注异常抛出和捕获的程序上下文环境. 文献[2]针对异常传播过程中的"重抛出"问题,对 Java 服务应用 Tomcat 进行"重抛出"的识别和数据可达性分析.

上述研究者都没有对异常传播结构的网络化特征进行建模和分析,而更多的关注于对异常传播路径的提取以及异常抛出源点的识别.

2 异常传播网络

2.1 异常传播结构

以 Java 为例,异常传播结构的生成主要有以下3种情况产生[8]:

1) 方法中 try 块抛出异常且该方法的 catch 块捕获了异常,程序继续执行;

2) 方法中 try 块抛出异常,该方法也找到了匹配的处理程序,但在 catch 块又重新抛出了异常或又引发其他异常,异常将向外层的 try 块(如果有 try 块的嵌套)传播或沿方法调用序列传播;

3) 方法中 try 块抛出异常但该方法中没有找到匹配的处理程序,异常将向外层的 try 块(如果有 try 块的嵌套)传播或沿方法调用序列传播.

由此可知,面向方法调用的软件依赖图构成异常传播结构的基础,方法之间的调用关系构成了异常传播的路径.

本文以往的工作[9]在方法层次对软件依赖关系扩展,将方法和异常类型通过抛出关系和捕获关系进行关联,将软件依赖关系和异常类型层次结构进行有机地结合,提出了软件扩展依赖图的概念. 以软件扩展依赖图为基础,提出了一种基于深度优先搜索的异常传播路径提取算法,将异常传播结构描述成与某种被处理异常相关的方法调用链,即异常传播路径.

2.2 异常传播网络模型

采用文献[9]提出的抽取算法对异常传播结构进行抽取,可以获取较为完整的异常传播路径. 同时也可以得到与异常传播路径相关的异常抛出源点及与传播过程相关的异常类型. 本文在已有的工作基础之上提出一种异常传播网络模型,定义如下:

异常传播网络 $EPG = \langle V, E \rangle$,EPG 为有向无权图,$V$ 为节点集,对于任意 $v_i \in V$ 代表类或接口中的方法;E 为节点之间边的集合,对于任意有序对 $\langle v_i, v_j \rangle \in V$ 当且仅当 v_i 与 v_j 之间存在异常传递关系,即 v_i 抛出异常 E 且 v_j 是异常 E 的接收方法.

图1为 tomcat-coyote.jar 的异常传播网络图.

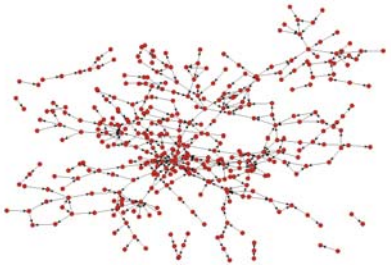


图 1　异常传播网络模型

3　异常传播网络的特性分析

本文以开源软件 DependceFinder(http://dep-find.sourceforge.net)对编译后的中间结果 jar 包进行解析,生成描述方法调用关系的 XML 文件;开源框架 ASM(http://asm.ow2.org)可以实现对异常类型抛出和捕获语义的识别,从而完成对异常传播结构的抽取.

本文在对异常传播网络构建过程中,仅对 jar 包内的.class 文件进行分析,并不包括第三方工具包,因此会对传播路径提取的完整性造成一定的影响.

本文将针对 5 个开源软件进行异常传播路径的提取,选取的开源软件如下:

1) ant.jar:流程脚本引擎,用于自动化调用程序完成项目的编译、打包、测试等;

2) httpclient-4.0.1.jar:支持 HTTP 协议的客户端编程工具包;

3) tomcat-coyote.jar:为 tomcat 的容器提供了对底层 socket 连接数据的封装;

4) log4j-1.2.16.jar:基于 java 的日志工具包;

5) jsse.jar:java 安全套接字扩展包.

软件包统计数据如表 1 所示.

表 1　软件包统计数据

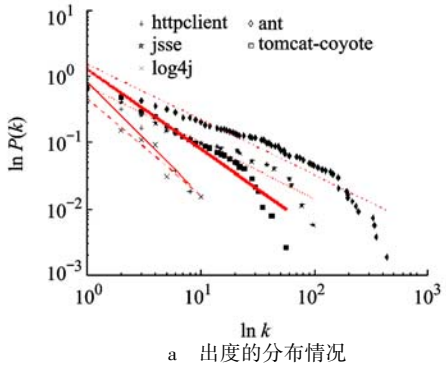
名称	规格/KB	异常传播路径数	节点数	边数	C_{rand}	C	d_{rand}	d
ant.jar	1 478	1 491	535	8140	0.028 69	0.037 4	2.607 26	2.92
httpclient-4.0.1.jar	284	44	54	67	0.019 38	0.021 5	3.978 52	1.41
tomcat-coyote.jar	768	556	376	1351	0.009 76	0.052 5	4.584 73	2.34
log4j-1.2.16.jar	470	50	65	58	0.015 00	0.0	2.674 11	1.28
jsse.jar	576	214	173	863	0.023 52	0.029 1	3.331 61	4.54

注:异常传播路径数—通过异常传播路径抽取算法抽取后的异常传播路径总数;节点数,边数—通过将异常传播路径网络化后所包含的节点数和边数;C_{rand},C—具有相同规模的随机网络和异常传播网络的聚集系数;d_{rand},d—具有相同规模的随机网络和异常传播网络的平均最短路径.

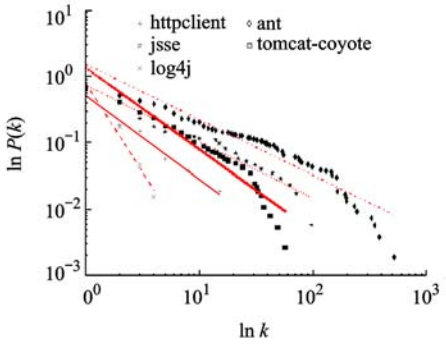
3.1　度

度作为复杂网络中节点的基本属性之一,体现了节点在复杂网络中的重要程度. 节点 v_i 的度定义为与该节点 v_i 连接的其他节点的数目. 作为有向网络,度分为出度(out-degree)和入度(in-degree). 在异常传播网络中,节点的出度表示该方法抛出异常过程的数目;而入度则表示该方法接收异常过程的数目,可能是对该异常类型的再次抛出或者处理. 度的概念反映的是方法接收或抛出异常过程的数目,而忽略具体的异常类型. 网络中节点度的分布情况可以用分布函数 $P(k)$ 表示. 本文采用度的补累积分布 CCDF[10](Complementary Cumulative Distribution Function)表示度的分布情况 $P(k)$,即 $P(k)$ 表示节点的度大于等于 k(节点的出度或入度)的概率.

图 2 描述的是双对数坐标系下 5 个软件包异常传播网络度的 CCDF 分布. 由图可知:①异常传播网络的出度和入度分布在尾部都出现下坠现象,其主要原因是异常传播网络中存在极少的具有大量异常抛出过程数目或异常接收过程数目的方法节点,如 tomcat-coyote.jar 中入度的最大值为



a　出度的分布情况



b　入度的分布情况

图 2　双对数坐标系下异常传播网络度分布

57,而此类节点只有 1 个,相似情况也发生在出度分布中;②5 个软件包的异常传播网络的规模差异较大,但其出度和入度分布却十分相似;表 2 给出了度分布的线性相关系数. 虽然异常传播网络分布尾部都出现下坠现象,但是其分布总体线性相关系数(R^2)较高,说明传播网络的出度和入度基本满足幂律分布.

表 2 度分布线性相关系数

符号	软件包	R^2	
		出度	入度
+	httpclient	0.97	0.99
★	jsse	0.93	0.93
×	log4j	0.95	0.98
◆	ant	0.90	0.90
■	tomcat-coyote	0.90	0.90

存在极少的具有大量出度或入度的节点是异常传播网络度分布尾部出现下坠现象的根源,而异常传播网络更多地关注同时具有高出度和高入度的方法节点,如 tomcat-coyote. jar 中 org. apache. jk. config. WebXml2Jk. generate 方法入度为 30,出度为 31,这类节点是促使传播结构网络化的纽带节点,又称之为集散(hub)节点. 文献[11]的研究表明:降低入度 hub 之间的相互连接、减少入度 hub 的后继节点数量可以降低大范围变更的频繁发生,提高软件稳定性. 针对这些集散节点的代码变更将直接造成整个传播网络结构的变化. 因此,这些节点将是软件设计初期、后期程序理解以及维护的重点.

3.2 异常类型层次结构

在面向对象语言中,一个异常被表示为一个类的实例,这就为异常传播路径的提取带来了挑战. 由于异常的分类来源于类的层次结构,因此,在对异常类型进行捕获时可以发生如下情况:对异常类型 A 的捕获语句可以实现对 A 的一系列子类型的捕获,而异常的抛出也存在这种情况. 而在算法实现过程中,异常类型层次结构是判断方法所抛出的异常类型是否被捕获的关键因素.

本文将异常传播路径中抛出或处理的异常类型信息与异常传播网络拓扑特征进行综合分析以发现软件开发人员在设计异常处理机制时对异常类型层次使用的倾向性. 其中异常类型层次结构的第 0 层为 Java 的顶层异常类型,即 java. lang. Th-rowable;第 1 层代表继承于 java. lang. Throwable 的异常类型,如 java. lang. Exception;第 2 ~ 4 层以此类推.

3.2.1 异常类型的抛出

在异常传播结构提取时,是以对该异常的处理方法为搜索起点,配合异常类型层次结构的分析过程,实现对异常类型抛出关系的识别,如图 3 所示. 本文对抛出异常类型层次分布的描述以度分布为基础,对于图中的每个点的含义为:出度为 k 的节点集合的平均抛出异常类型层次,主要描述了节点异常抛出数目与抛出异常类型层次之间的对应关系. 从图中可以看出,ant. jar,tomcat-coyote. jar,jsse. jar 抛出的异常类型层次比较稳定. ant. jar,tomcat-coyote. jar 大部分集中在第 2 层,以 java. io. IOException(继承 java. lang. Exception)为主;jsse. jar 大部分集中在第 3 层,以 java. net. SocketException(继承 java. io. IOException)为主;在异常传播过程中存在“不要传播抽象异常”的原则[12],而这 5 个软件包都违背了上述原则,在其异常传播过程中所抛出的异常类型主要集中在 IOException 或 SocketException 等抽象异常类型.

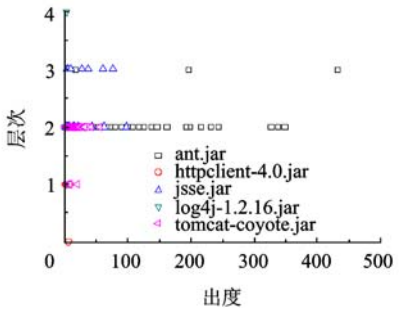


图 3 异常传播网络中抛出异常类型层次分布情况

3.2.2 异常类型的捕获

与异常类型的抛出相比较,异常类型的捕获总是在含有 catch 块的方法节点中,因此,对于每一条传播路径,只有一个方法节点捕获相关异常类型,如图 4 所示. 本文对捕获异常类型层次分布的描述与抛出异常类型的层次描述类似,对于图中的每个点的含义为:节点入度为 k 的节点集合的平均捕获异常类型层次. 图中可见,与异常的抛出相对应,对异常的捕获主要集中高层异常类型 java. lang. Exception 或 java. io. IOException. 其中一个原因是 java. lang. Exception 或 java. io. IOException 作为异常类型层次结构的较高层次,在

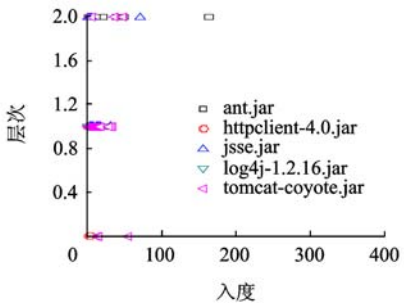


图 4 异常传播网络中捕获异常类型层次分布情况

catch 块中可以处理所有子异常类型,为开发者提供了便捷,但为后续的软件维护增加了异常语义分析的难度. 另外,自定义异常的广泛应用也是异常类型层次多样化的一个原因. 此外,异常类型的捕获节点都集中在入度较少的节点中,同时也印证了 4.1 节中入度较高的节点是中转节点的结论.

3.3　网络平均距离与聚集系数

网络中 2 个节点 i 和 j 之间的距离 d_{ij} 定义为连接这 2 个节点的最短路径上的边数. 网络中的平均最短路径长度 L 定义为任何 2 个节点之间距离的平均值,即

$$L = \frac{2}{N(N-1)} \sum_{i>j} d_{ij}$$

式中 N 为网络中的节点数[13]. 对于平均距离较大的网络,信息在网络节点中传播开来需要较长的时间. 而对于平均距离较小的网络,网络的连接会相对较强,不容易被破坏,信息的传递也会更迅速、更稳定和更加准确可靠[11]. 本文对任意 2 个节点之间的最短路径采用 BFS(广度优先算法)进行可达节点的搜索,但对于异常传播网络而言,并不是强连通图,因此,对于 2 个节点之间的不可达现象将不在平均最短路径的计算范围内.

聚集系数一般和平均最短路径同时提到,它们是反映复杂网络“小世界”特性的 2 个重要性质[13]. 所谓网络的聚集特性,意指网络中你朋友的朋友可能彼此也是朋友. 假设网络中的一个节点 v_i 有 k_i 条边与其他节点相连,这 k_i 个节点成为节点 v_i 的邻居. 这 k_i 个节点的实际边数为 E_i 和总的可能的边数 $k_i(k_i-1)/2$ 之比定义为节点 v_i 的聚集系数 C_i,即 $C_i = 2E_i/k_i(k_i-1)$. 整个网络的聚集系数 C 为网络中所有节点聚集系数 C_i 平均值[13].

本文对表 1 中的软件包进行平均路径长度和聚集系数进行分析可知:①除了 tomcat-coyote. jar 的聚集系数大于随机网络,ant. jar,httpclient-4. 0. 1. jar 和 jsse. jar 的聚集系数与随机网络持平,而 log4j-1. 2. 16. jar 的聚集系数为 0. 对于小世界网络,其聚集系数 $C \gg C_{\text{rand}}$,以上开源软件包的异常传播网络都不具备“小世界”的特性,该现象主要是与异常传播的“弱边界性”相关,在复杂软件系统中,异常的产生和处理过程并不一定在一个局部化区域内,异常传播频繁跨越组件的边界[2],降低了方法节点之间的紧密程度;②ant. jar,tomcat-coyote. jar 和 jsse. jar 的平均最短路径都在两跳以上,而 httpclient-4. 0. 1. jar 和 log4j-1. 2. 16. jar 的平均最短路径小于两跳. 异常传播网络的平均最短路径的主要影响因素是异常传播路径的跳数. 除 tomcat-coyote. jar,ant. jar 和 jsse. jar 外,其余 2 个软件包的异常传播路径中长度为 2 跳的比例都在 90% 以上,能够遵循 Shenoy 提出的第 3 条规律:异常捕获的代码位置尽可能地靠近异常抛出的位置[14],而 tomcat-coyote. jar,ant. jar 和 jsse. jar 的多跳数传播路径的比例(tomcat-coyote. jar-34. 2% ,ant. jar-74. 9% 和 jsse. jar-52. 8%)明显大于其余 2 个软件包[9].

4　结　论

在软件工程领域,复杂网络的广泛应用使研究者对软件结构有了更深一步的了解. 传统软件网络化建模聚焦于软件的各个层次,通过实例化软件各个层次的元素以及元素之间的关系,形成网络化结构. 本文在以往工作的基础之上,通过定义异常在方法之间的传递关系,首次将单一、分割的异常传播路径进行网络化建模.

本文选取 5 个不同类型的软件包所抽取的异常传播网络进行特征分析. 由分析可知,异常传播网络的出度和入度基本符合幂律分布;由异常类型层次的分析结果可知,异常传播过程大多数违反了“不要传播抽象异常”的原则;此外,由于异常传播的“弱边界性”,软件异常传播网络的聚集系数都很低,使得无法体现“小世界”特性.

通过度分布、传播异常类型的层次分布、聚集系数以及平均最短路径等指标对其进行分析可知:①异常传播网络以网络化的形式描述了软件系统的异常处理架构,根据其度分布的特点,实施对集散节点数量的有效控制,能够有效地降低软件变更和维护的成本;②虽然在异常传播过程中抽象异常仍然是软件设计者在异常抛出和处理过程中的首选,但是将抽象异常实例化能够有效协助追踪异常传播的源头;③异常传播网络结构的聚集系数较低,没有呈现软件系统的高内聚特征,因此需要软件设计者注重异常处理结构设计时的边界性设计,以提高系统结构的稳定性和鲁棒性. 例如可以利用系统中各个模块中确定的边界为异常处理提供一个清晰的说明,因为这样可以清晰地理解异常传播路径[12, 15].

在构建异常传播网络过程中,异常传播路径抽取的完整性将决定网络的规模,由于本文考虑的抽取对象仅局限于软件包本身的. class 文件而不包含第三方文件,会对传播路径的完整性造成一定的影响. 因此,进一步提高异常传播结构的完

整性是下一步工作的重点.

参考文献（References）

[1] Wikipedia. Exception handling[EB/OL]. 2011[2011-03-31]. http://en. wikipedia. org/wiki/Exception_handling#Checked_exceptions

[2] Fu C, Ryder B G. Exception-propagation analysis: revealing exception handling architecture in Java server applications[C]// Proc 29th international Conference on Software Engineering. Washington DC: IEEE Computer Society, 2007: 230 – 239

[3] Van Dooren M, Steegmans E. Combining the robustness of checked exceptions with the flexibility of unchecked exceptions using anchored exception declarations[C]//Proc OOPSLA'05. New York: ACM Press, 2005: 455 – 471

[4] Myers C R. Software systems as complex networks: structure function and evolvability of software collaboration graphs[J]. Physical Review E, 2003, 68(42): 46116-1 – 46116-15

[5] Ma Yutao, He Keqing, Du Dehui, et al. A complexity metrics set for large-scale object-oriented software systems[C]//Proceedings of the Sixth IEEE International Conference on Computer and Information Technology. Washington DC: IEEE Computer Society, 2006: 189 – 195

[6] Robillard M, Murphy G. Static analysis to support the evolution of exception structure in object-oriented systems[J]. ACM Transactions on Software Engineering and Methodology, 2003, 2(12): 191 – 221

[7] Mao Chengying, Lu Yansheng. Improving the robustness and reliability of object-oriented programs through exception analysis and testing[C]//Proc 10th IEEE International Conference on Engineering of Complex Computer Systems. Washington DC: IEEE Computer Society, 2005: 432 – 439

[8] 姜淑娟,徐宝文,史亮,等. 一种基于异常传播分析的依赖性分析方法[J]. 软件学报, 2007, 18(4): 832 – 841
Jiang Shujuan, Xu Baowen, Shi Liang, et al. An approach to analyzing dependence based on exception propagation analysis[J]. Journal of Software, 2007, 18(4): 832 – 841(in Chinese)

[9] Qiu Xiang, Zhang Li, Lian Xiaoli. Static analysis for Java exception propagation structure[C]//2010 International Conference on Progress in Informatics and Computing. Washington DC: IEEE Computer Society, 2010: 1041 – 1046

[10] Siganos G, Faloutsos M, Faloutsos P, et al. Power laws and the AS-level intemet topology[J]. IEEE/ACM Trans on Networking, 2003, 11(4): 514 – 524

[11] 钱冠群. 基于复杂网络的软件静态结构分析[D]. 北京: 北京航空航天大学计算机学院, 2009
Qian Guanqun. Software static structure analysis based on complex network method[D]. Beijing: School of Computer Science and Engineering, Beijing University of Aeronautics and Astronautics, 2009(in Chinese)

[12] Robillard M, Murphy G. Designing robust Java programs with exceptions[J]. ACM SIGSOFT Software Engineering Notes, 2000, 6(25): 2 – 10

[13] 何克清. 软件网络[M]. 北京: 科学出版社, 2007
He Keqing. Software network[M]. Beijing: Science Press, 2007(in Chinese)

[14] Shenoy S. Best practices in EJB exception handling[EB/OL]. New York: IBM, 2002[2011-03-31]. http://www. ibm. com/developerworks/java/library/j-ejbexcept/index. html

[15] Malayeri D, ldrich J. Practical eception secifications[J]. Advanced Topics in Exception Handling Techniques, LNCS, 2006, 4119: 200 – 220

[16] Cacho N, Dantans F, Garcia A, et al. Exception flows made explicit: an exploratory study[C]//Proc XXIII Brazilian Symposium on Software Engineering. Washington DC: IEEE Computer Society, 2009: 43 – 53

（编　辑：文丽芳）