反模式检测研究综述*

盛津芳，胡培培，王 斌
（中南大学 信息科学与工程学院，长沙 410083）

摘 要：反模式是一类在软件设计和实现过程中重复出现的不良解决方案，这种方案会妨碍软件的维护和演化。反模式检测不仅能够帮助识别软件缺陷，避免未来的工作中出现同样的错误，还能够对软件重构提供指导，对于软件的质量保证有着重要意义。从静态反模式检测和动态反模式检测入手，详细分析了目前反模式检测领域的相关方法，对相关检测工具进行了调查分析，总结了各种检测方法的应用以及存在的不足。

关键词：反模式；静态反模式检测；性能反模式检测；缺陷检测工具
中图分类号：TP301 **文献标志码**：A **文章编号**：1001-3695(2013)12-3525-04
doi:10.3969/j.issn.1001-3695.2013.12.002

Survey of research on anti-pattern detection

SHENG Jin-fang, HU Pei-pei, WANG Bin
(*School of Information Science & Engineering, Central South University, Changsha* 410083, *China*)

Abstract: Anti-patterns are poor solutions to recurring implementation and design problems that impede the maintenance and evolution of software. Anti-pattern detection can not only help identify software defects, avoid trapping into the same problems, but also provide guidance for software refactoring, which has great meaning for improvement of softwarequality. Starting with static anti-pattern detection and dynamic anti-pattern detection, this paper analysed the current relevant methods in the field of anti-pattern detection in detail, and then made a survey of the related detection tools. At last it concluded the application of various detection methods as well as their shortcomings.

Key words: anti-pattern; static anti-pattern detection; performance anti-pattern detection; defect detection tools

反模式(anti-pattern)描述了对某个问题必然产生不良后果的常见解决方案[1]。文献[2]对胖球(blob)、面条代码(spaghetti code)这两种反模式对程序可理解性的影响进行了实证研究，其结果表明，反模式的存在还会带来系统理解上的困难。虽然通过目前的软件质量管理工具、软件测试技术可以发现软件功能或编码的错误，但是无法解决由于不良设计导致系统可维护性、可扩展性、可重用性差及性能低下等问题。即使能够检测出系统设计上可能存在的问题，但是对如何进行改善也往往是未知的。反模式正是衔接这种问题和解决方案之间的桥梁。反模式不仅描述了带来负面效果的解决方案，同时提供了对应的重构方案，因此基于反模式的缺陷检测有助于更好地理解反模式，从而在系统的设计或实现过程中避免类似的错误，进一步为系统重构提供指导。

1 静态检测技术

1.1 手动检测

手动检测是最原始的检测方式，整个检测过程没有相关自动化工具的支持，主要体现为对相关文档或代码的阅读分析[3~7]。通过预先定义一系列规则，分析人员根据这些规则来判断软件中是否存在缺陷。文献[3]将关注点从软件系统的实现转向抽象的设计如类图、序列图、状态图等；文献[4,5]主要关注需求的缺陷检测，其中需求是以 SCR 状态机形式表示的；文献[6]也关注需求的缺陷检测，但其需求是以自然语言的形式描述的；文献[7]将检测的重点放在用户接口上。

手动检测过程中，分析人员可结合实际情况及自身对设计的理解来进行分析判断，避免了反模式检测的不确定性。不足之处是该方法需要对整个系统进行分析，随着系统规模的扩大，系统的复杂性也随之增加，这将是一个非常耗时的过程。显然这种方法不适用于大规模系统。

1.2 基于抽象语法树的检测

基于抽象语法树的反模式检测方法首先通过对源代码进行词法和语法分析，产生反模式的中间表示形式即抽象语法树(abstract syntax tree, AST)。文献[8]提出了一种使用抽象语法树检测代码克隆的方法，通过比较语法树结构的相似度来检测克隆的表达式、语句以及数据声明。文献[9,10]从软件演化的角度出发，跟踪代码在软件的多个版本中的历史信息，识别具有相同起源的克隆，但是由于现有的遗留系统很少会保留这些信息，该方法的适用性不强。

代码味道用于识别面向对象系统中的问题类[11]。基于 AST 的方法适合对类中的代码味道进行识别，但对类之间的代码味道识别效果不好。

1.3 基于度量的检测

基于度量的检测方法利用软件度量对反模式的表现症状进行表示，通过对度量设置相应的阈值，并以规则的形式来表示，从而完成反模式的检测。下面介绍几种代表性的方法。

a) Ciupke[12] 提出一种针对遗留系统进行分析的方法。该

收稿日期：2013-03-20；**修回日期**：2013-05-09 **基金项目**：国家自然科学基金资助项目(61240039)

作者简介：盛津芳(1971-)，女，副教授，硕导，主要研究方向为软件工程(jfsheng@csu.edu.cn)；胡培培(1989-)，女，硕士，主要研究方向为软件工程；王斌(1973-)，男，副教授，博导，主要研究方向为软件工程.

方法通过对遗留系统的源码进行分析得到系统的实际设计模型,将频繁出现的缺陷问题以谓词演算的形式表示,再通过系统设计模型与实际设计模型进行比较来定位软件中的缺陷问题。这种检测方法适用于遗留系统,采用的大多为简单的度量,如一个类不能包含超过六个对象、继承树的深度不能大于6等,对于复杂的反模式则难以检测到。

b)检测策略(detection strategy)[13]是 Marinescu 提出的一种检测设计缺陷的方法。所谓检测策略,即针对具体存在的问题选择合适的度量进行表示,组成一组反模式描述规则,然后根据反模式规则指定相应的解析模板,即针对具体问题设置合适的阈值。文中对九种设计缺陷定义了相应的检测规则。该方法的优势在于能根据异常的度量值清楚地知道导致系统缺陷的原因,而不需要对这些异常度量值进行分析推理来找出问题的原因。

c)DCPP matrix[14]是一种利用设计变化传播概率矩阵来检测反模式的方法,这种方法适用于检测 shotgun surgery bad smell(霰弹式修改)及 divergent change bad smell(发散式变化)两种软件缺陷。霰弹式修改指的是一个变化将会引发多个类的修改,会对系统造成波动影响。发散式变化指的是某个类受到多种变化的影响,这是一种定量的检测方法,它通过分析软件一个组件的变化对其他组件的影响来达到检测反模式的目的。这种方法的优势是可以根据矩阵来了解软件的波动影响,在此基础上也能评估软件的复杂性与可维护性。能否鉴别软件开发中一个产品是否容易受到其他变化的影响,这点对于大型系统来说是非常重要的。

d)DECOR[15]是 Moha 等人提出的一种检测代码味道的方法。该方法从描述反模式的一些经典文献出发,通过对自然语言描述的反模式进行领域分析,提取相关的关键词,从而制定描述设计缺陷或反模式的规则。这种规则用 BNF 文法加以形式化,并以规则卡的形式表示,缺陷的检测则通过特定的元模型解析器来生成相应的检测算法。该方法能够检测四种反模式:blob(胖球)、funcational decomposition(功能分解)、spaghetti code(面条代码)、Swiss army knife(瑞士军刀)及 15 种代码味道。该方法依据自然语言描述的反模式来完成反模式的形式化描述,这个过程不依赖于特定的表示语言或解析模型,在不同的平台下只需实现检测算法就能完成反模式检测,但采用的固定的规则模板限制了反模式的描述能力和可扩展性。

e)Khomh 等人[16]提出了基于贝叶斯信念网络(BBN)的反模式检测方法。该方法利用文献[15]中的规则卡构建贝叶斯信念网络,通过贝叶斯分类方法计算一个类是否是反模式的概率。这样质量分析人员可以根据概率的大小来优先验证那些有可能是反模式的类。这种方法借助于历史检测数据,能够处理检测过程中的检测结果的不确定性以及边缘值的判定问题,而且可以根据检测结果来不断校正数据模型,从而提高检测的准确率。但是规则卡的表达能力有限,在有较多复合规则的情况下,构建 BBN 时产生较多的中间节点,BBN 的调整会带来额外的开销;此外,不完整的训练数据集也会影响检测结果的准确率,影响了该方法的实用性。

1.4 可视化检测

Dhambri 和 Sahraoui 提出了一种基于系统 3D 图的检测方法[17,18]。该方法通过逆向工程工具 PADL 及度量提取工具 POM 获取系统的结构关系及度量属性的值,使用软件可视化框架 VERSO 产生大规模面向对象系统的 3D 表示图,如图 1 所示。每个小盒子的颜色、高度、角度等分别表示软件的不同度量属性,然后结合图形及预定义的检测算法完成反模式的检测。该方法目前能够检测的反模式包括 blob、misplaced class(遗失类)、functional decomposition、Swiss army knife、divergent change、shotgun surgery。

ABS(antipatterns identification using B-splines)是 Oliveto 等人[19]提出的一种基于 B 样条曲线的反模式表示方法。如图 2 所示,图中两条曲线分别表示构造的反模式曲线及类的实际曲线,通过计算两条曲线之间的相似度来判断该类是否是反模式。

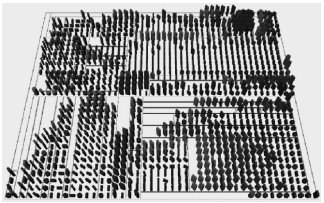

图 1 面向对象系统 3D 表示图示例

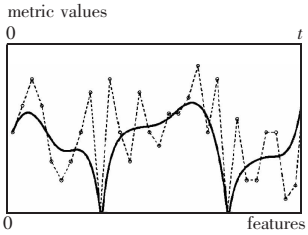

图 2 类/反模式的 B 样条曲线表示图

这两种方法是自动检测和手动检测方法的折中,它们结合了系统上下文信息和检测人员的经验知识,解决了完全的自动化检测方法难以结合外部信息的难题,也相对提升了完全手动验证的效率低下问题。

1.5 基于本体描述的检测

SPARSE[20]是 Settas 等人提出的一种基于 OWL 本体的反模式检测方法。该方法采用本体描述反模式的根源、症状、后果及这些因素与其他反模式之间的关系。语义关系被翻译成产生式规则,根据系统的表现症状并结合反模式知识库的信息便能检索软件系统中存在的反模式。这种方法能够对 Web 上出现的 31 种反模式进行表示。

这种方法适用于根据反模式的表现症状来确定造成这种情况的反模式。因为各种反模式之间是有关联的,一个反模式的存在通常会伴随着其他反模式的存在。一个反模式可能表现出与其他反模式相同的症状。该方法的不足之处在于随着反模式种类的增多,而使反模式本体增多,有可能会造成反模式本体描述产生冲突、重复或不一致的现象。

2 动态检测技术

静态检测技术依赖于源代码或设计文档,但在对应用系统进行维护和优化时并不能保证得到源码或所需的文档。而且随着构件技术的发展,许多软件系统通过组装已有的商用构件进行开发,这意味着获得系统完整的源代码或设计文档更为困难。

动态反模式检测主要依赖系统运行时获取的信息,由于这类反模式通常给系统性能带来不良的影响,也被称为性能反模式。目前的反模式检测研究大都集中在静态检测,对动态反模式检测的研究则甚少,主要有以下几种方法。

a)Mos 等人[21]研发了一种面向 JEE 应用的性能监测工具 COMPAS(component performance assurance solutions)。该工具拦截 EJB 之间的调用并监测每个方法调用的响应时间,生成 UML 模型,用户针对模型输入不同的负载来测量不同场景下系统的性能,通过该方法可以检测系统中存在的性能瓶颈。类似的方法有 Guo 等人[22]研发的工具 JPManager 及 Meyerhöfer 等人[23]提出的针对 J2EE 系统的性能监控方法 TestEJB,它们

用于帮助开发人员识别系统中的性能瓶颈。

b)Parsons 等人[24]针对企业级系统提出了一种 EJB 性能反模式的检测方法及性能诊断工具 PAD(performance antipattern detection)。该方法是一种基于运行时系统的检测方法,它采集系统的构件部署信息及系统运行时刻的信息,并利用数据挖掘算法找出这些信息中存在的关联关系和模式。这些关联关系作为事实被加载到规则引擎,反模式也被定义为一系列规则,通过推理引擎完成规则的推理,即完成反模式的检测。该方法主要是针对基于构件的企业级系统的反模式检测,能够检测由于系统部署和设计引起的反模式;此外,该方法结合了数据挖掘技术,能够识别出系统中潜在的性能反模式和性能缺陷。

c)Crasso 等人[25]开发了一个提高 Java 企业级应用系统性能的工具 JEETuningExpert。该方法也是在系统运行时进行性能反模式检测,通过 Java 反射机制拦截 EJB 之间的调用关系并跟踪调用的堆栈信息,这些信息以日志的形式保存在 XML 文件中。通过预先分析构件之间的交互行为预定义基于 Drools 的检测规则,接着对 XML 文件进行解析提取出对应的信息与预定义的规则进行匹配,满足规则的事实表明该事实是一个反模式。该方法的重点在于为开发人员提供检测之后的重构指导策略,目前该方法仅支持四种性能反模式的检测。

d)北京大学张磊、兰灵等人提出了一种基于运行时软件体系结构的 JEE 反模式检测工具[26,27]。该方法通过监测系统运行时体系结构来采集与应用系统相关的信息,其中包括静态信息如主机、构件等的属性信息,动态信息如构件方法调用、主机内存消耗等。采用 MOF 模型对反模式进行描述,反模式的检测逻辑则根据反模式的关注点和表现形式来制定。通过执行反模式检测逻辑所对应的 QVT 程序找出存在的反模式。这种方法的优势在于将反模式描述与反模式检测进行分离,适合于检测与构件调用相关的反模式,对于与构件内部逻辑相关的反模式则无法检测。

3　相关工具介绍

Analyst4J 是一个 Java 代码静态分析引擎,使用 C&K 度量集对输入的 Java 文件进行度量,支持对 blob、functional decomposition、spaghetti code 这三种反模式及多种代码味道的识别[27]。Aspect[29]、LCLINT[30]、extended static checker[31]使用程序验证技术来识别代码味道,这些工具需要工程师在代码中增加注释来协助验证系统的正确性。SemmleCode[32]是一个代码分析工具,通过 SemmleCode 工程师可以使用声明式查询语言 QL 对源代码进行查询从而检测出代码味道。PMD 能够检测 Java 源码中违反设计原则的地方,也允许使用 Java 或 XPath 来定义新的代码味道的检测规则,但是该类工具需要用户对 Java 或 XPath 比较熟悉[33]。

Dependency Finder[34]是对 Java 程序编译后的字节码文件进行分析的工具集,它的核心是一个依赖分析工具,通过它可以提取组件之间的依赖图并进行分析。VizzAnalyzer[35]是一个软件质量分析工具,通过阅读软件代码、设计需求及文档进行软件的质量分析。CROCOPAT 对面向对象系统进行结构化分析,从而发现系统中的设计模式及编码上的问题[36]。模型监测器 Bogor[37]、MOPS[38]则使用模型检查技术检查 C 语言系统中违背安全属性的代码问题。

目前也有一些与性能检测相关的工具。SABER[39]检测 J2EE 应用系统编码中的错误,并提供信息来解释编码为什么存在缺陷。文献[40,41]介绍了两种性能管理工具 JProfiler 及 Borland,能够检测出某些编程上的失误,如内存泄漏等。Smalllint 针对 SmallTalk 代码来检测编码问题[42]。FINDBUGS 能检测 Java 系统的正确性以及与性能相关的缺陷[43]。

4　结束语

反模式检测的目的是为了及时对系统进行重构,改善软件的质量。目前反模式检测技术主要分为静态检测和动态检测两类。静态检测的信息源于系统设计文档、源代码、版本库等;动态检测则主要是利用运行时系统的信息(构件调用序列、资源消耗等)进行分析。

静态检测技术在实际中应用较多的属于手动检测和基于度量的检测。手动检测的检测结果可靠,但是效率较为低下,适合分析较小规模的系统。基于软件度量的检测方法较为成熟,这种方法容易理解且实现简单,但有一定的局限性,即检测结果是否准确在很大程度上依赖于度量属性的选取和阈值的设置。基于抽象语法树的检测能够识别代码克隆,但构造树的代价较高,寻找树相似的匹配算法复杂度高,不适用于较大型的系统。可视化检测是一种较直观的检测方法,但检测的过程中可能会融入太多的主观性。基于本体描述的检测的本质更偏向于反模式的推理,即根据反模式的表现症状来推断它最有可能属于的反模式种类。

动态检测技术作为对静态检测方法的补充,能够识别系统中可能存在的性能反模式。目前在动态反模式检测方面的研究甚少,大多面向 J2EE 应用系统。这些方法中一些能够检测出系统中可能存在的潜在的性能问题和性能反模式,一些能够检测出预定义的性能反模式。但动态检测是基于运行时系统的,它自身的数据采集过程会带来性能上的消耗。

通过上述对反模式检测技术的分析,可知目前的检测方法存在如下不足:

a)反模式描述的能力不足。目前对反模式描述的研究大都就简避繁,仅仅针对一些常见的且容易被描述的反模式,并且每种反模式描述方法也仅适用于特定的几种反模式。

b)反模式检测的高漏报率和误报率。阈值的设置通常是基于经验性的,难以提供经过实证的阈值,导致了检测结果的漏报和误报,妨碍了检测结果的适用性和可靠性。

c)检测过程难以结合系统的上下文信息。反模式的判定与系统所处的上下文有关,也与专家的经验知识相关,未充分考虑这些因素的影响会造成检测结果的漏报和误报。

d)开放式的系统带来了检测上的困难。目前基于构件的系统大多由商业构件组装而成,难以获得系统的源代码或设计文档,而且构件也可能基于不同的编程语言,这也给反模式的检测带来了困难。

e)检测工具的效果难以进行评估和对比。反模式的检测结果受组织机构的特点、人员经验及所熟悉的技术类型等的影响,可能产生不一致的检测结果。而且系统中真正存在的反模式数目是未知的,因此难以对检测方法的有效性做出评价。

参考文献:

[1] BROWN W J,MALVEAU R C,McCORMICK H W,*et al*. 反模式危机中软件、架构和项目的重构[M]. 宋锐,等译. 北京:人民邮电出

版社,2008.
[2] ABBES M, KHOMH F, GUEHENEUC Y G, *et al*. An empirical study of the impact of two antipatterns, blob and spaghetti codeon program comprehension[C]//Proc of the 15th European Conference on Software Maintenance and Reengineering. Washington DC: IEEE Computer Society, 2011: 181-190.
[3] MOHA N. Detection and correction of design defects in object-oriented designs[C]//Proc of the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications. New York: ACM Press, 2007: 949-950.
[4] FUSARO P, LANUBIL F, VISAGGIO G. A replicated experiment to assess requirements inspection techniques[J]. Empirical Software Engineering, 1997, 2(1): 39-57.
[5] PORTER A A, Jr VOTTA L G, BASILI V R. Comparing detection methods for software requirements inspections: a replicated experiment[J]. IEEE Trans on Software Engineering, 1995, 21(6): 563-575.
[6] BASILI V R, GREEN S, LAITENBERGER O, *et al*. The empirical investigation of perspective-based reading[R]. College Park: University of Marland, 1995.
[7] ZHANG Zhi-jun, BASILI V, SHNEIDERMAN B. An empirical study of perspective-based usability inspection[C]//Proc of Annual Meeting of Human Factors and Ergonomics Society. [S. l.]: SAGE Publications, 1998: 1346-1350.
[8] BAXTER I D, YAHIN A, MOURA L, *et al*. Clone detection using abstract syntax trees[C]//Proc of International Conference on Software Maintenance. Washington DC: IEEE Computer Society, 1998: 368-377.
[9] GODFRE M W, ZOU Li-jie. Using origin analysis to detect merging and splitting of source code entities[J]. IEEE Trans on Software Engineering, 2005, 31(2): 166-181.
[10] KIM M, NOTKIN D. Using a clone genealogy extractor for understanding and supporting evolution of code clones[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(4): 1-5.
[11] LI Wei, SHATNAWI R. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution[J]. Journal of Systems and Software, 2007, 80(7): 1120-1128.
[12] CIUPKE O. Automatic detection of design problems in object-oriented reengineering[C]//Proc of Technology of Object-Oriented Languages and Systems. Washington DC: IEEE Computer Society, 1999: 18-32.
[13] MARINESCU R. Detection strategies: metrics-based rules for detecting design flaws[C]//Proc of the 20th IEEE International Conference on Software Maintenance. Piscataway: IEEE Press, 2004: 350-359.
[14] RAO A A, REDDY K N. Detecting bad smells in object oriented design using design change propagation probability matrix[C]//Proc of International Multi Conference of Engineers and Computer Scientists. 2008.
[15] MOHA N, GUEHENEUC Y G, DUCHIEN L, *et al*. DECOR: a method for the specification and detection of code and design smells[J]. IEEE Trans on Software Engineering, 2010, 36(1): 20-36.
[16] KHOMH F, VAUCHER S, GUEHENEUC Y G, *et al*. A Bayesian approach for the detection of code and design smells[C]//Proc of the 9th International Conference on Quality Software. Piscataway: IEEE Press, 2009: 305-314.
[17] LANGELIER G, SAHRAOUI H, POULIN P. Visualization-based analysis of quality for large-scale software systems[C]//Proc of the 20th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM Press, 2005: 214-223.
[18] DHAMBRI K, SAHRAOUI H, POULIN P. Visual detection of design anomalies[C]//Proc of the 12th European Conference on Software Maintenance and Reengineering. Washington DC: IEEE Computer Society, 2008: 279-283.
[19] OLIVETO R, KHOMH F, ANTONIOL G, *et al*. Numerical signatures of antipatterns: an approach based on B-splines[C]//Proc of the 14th Conference on Software Maintenance and Reengineering. Washington DC: IEEE Computer Society, 2010: 248-251.
[20] SETTAS D L, MEDITSKOS G, STAMELOS I G, *et al*. SPARSE: a symptom-based antipattern retrieval knowledge-based system using semantic Web technologies[J]. Expert Systems with Applications, 2011, 38(6): 7633-7646.
[21] MOS A, MURPHY J. COMPAS: adaptive performance monitoring of component based systems[C]//Proc of the 26th International Conference on Software Engineering. 2004: 35-40.
[22] GUO Jiang, LIAO Yue-hong, PAIVIZ B. A performance validation tool for J2EE applications[C]//Proc of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems. Washington DC: IEEE Computer Society, 2006: 10-396.
[23] MEYERHOFER M, NEUMANN C. TestEJB: a measurement framework for EJBs[C]//Proc of the 7th International Symposium on Component-based Software Engineering. Berlin: Springer-Verlag, 2004: 294-301.
[24] PARSONS T, MURPHY J. A framework for automatically detecting and assessing performance antipatterns in component based systems using run-time analysis[C]//Proc of the 9th International Workshop on Component Oriented Programming. 2004.
[25] CRASSO M, ZUNINO A, MORENO L, *et al*. JEETuningExpert: a software assistant for improving Java enterprise edition application performance[J]. Expert Systems with Applications, 2009, 36(9): 11718-11729.
[26] 张磊,王玮琥,宋晖,等. 一种基于运行时软件体系结构的 JEE 反模式检测工具[C]//全国软件与应用学术会议论文集. 2011: 221-225.
[27] 兰灵,黄罡,王玮琥,等. 基于反模式的中间件应用系统性能优化[J]. 软件学报, 2008, 19(9): 2167-2180.
[28] Analyst4j[EB/OL]. http://www. codeswat. com/.
[29] JACKSON D. Aspect: detecting bugs with abstract dependences[J]. ACM Trans on Software Engineering and Methodology, 1995, 4(2): 109-145.
[30] EVANS D. Static detection of dynamic memory errors[J]. ACM SIGPLAN Notices, 1996, 31(5): 44-53.
[31] DETLEFS D L. An overview of the extended static checking system[C]//Proc of the 1st Workshop on Formal Methods in Software Practice. 1996: 1-9.
[32] VERBAERE M, HAJIYEV E, De MOOR O. Improve software quality with SemmleCode: an eclipse plugin for semantic code search[C]//Proc of the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications. New York: ACM Press, 2007: 880-881.
[33] PMD[EB/OL]. http://pmd. sourceforge. net/.
[34] Dependency Finder[EB/OL]. http://dep_nd. sourceforge. net.
[35] ARiSA. yzer. php. VizzAnalyzer[EB/OL]. http://www. arisa. se/vizz_ana/yzer. php.
[36] BEYER D, NOACK A, LEWERENTZ C. Efficient relational calculation for software analysis[J]. IEEE Trans on Software Engineering, 2005, 31(2): 137-149.
[37] DWYER M B, HATCLIFF J. Bogor: an extensible and highly-modular software model checking framework[J]. ACM SIGSOFT Software Engineering Notes, 2003, 28(5): 267-276.
[38] CHEN Hao, WAGNER D. MOPS: an infrastructure for examining security properties of software[C]//Proc of the 9th ACM Conference on Computer and Communications Security. New York: ACM Press, 2002: 235-244.
[39] REIMER D, SCHONBERG E, SRINIVAS K, *et al*. SABER: smart analysis based error reduction[J]. ACM SIGSOFT Software Engineering Notes, 2004, 29(4): 243-251.
[40] IBM. WebSphere studio profiling tool[EB/OL]. http://www. javaperformancetuning. com/tools/webspereprofiler.
[41] Java Performance Tuning. Borland optimizeit enterprise suite[EB/OL]. http://www. javaperformancetuning. com/tools/optimizeit.
[42] BRANT J. Smalllint[DB/OL]. http://st-www. cs. uiuc. edu/users/brant/Refactory/Lint. html.
[43] HOVEMEYER D, PUGH W. Finding bugs is easy[J]. ACM Sigplan Notices, 2004, 39(12): 92-106.