微重启技术及适毁性软件设计研究*)

朱 岩 王慧强

(哈尔滨工程大学计算机科学与技术学院 哈尔滨 150001)

摘 要 微重启是一种新型的针对大型分布式应用软件系统的低损耗、快速恢复技术，本文详细介绍了微重启的技术原理和实施策略，并通过一个应用实例介绍了微重启技术的应用过程，其中重点描述了微重启树的优化过程。而适毁性软件设计思想的实质是使系统具有安全高效的"可微重启性"。本文分析并总结了满足适毁性的应用系统特性，根据微重启的技术特点给出了应用改进方向，最后对技术存在的问题和发展前景进行了总结。

关键词 面向恢复计算，递归恢复，微重启，适毁性软件，组件级更新

Research on Microreboot and Design of Crash-only Software

ZHU Yan WANG Hui-Qiang

(College Computer Science & Technology, Harbin Engineering University, Harbin 150001)

Abstract Microreboot is a new technology for fast and cheap recovery applied for large-scale distributed application system. This paper introduces its principles and deploy strategies in detail, and especially describes the evolution of the r-map by an instance. Then the paper introduces a new design named crash-only software which makes application system more safely and effectively "microrebootable"and the properties of the design are summarized. Finally, the paper gives improved application directions of the technology, and a summary of the problems and further development on microreboot.

Keywords ROC, Recursive recovery, Microreboot, Crash-only software, Component-level rejuvenation

1 引言

Shimon Perres 定律指出，如果一个问题无法彻底解决，那么不应把它看作一个问题，而应该把它当作一个事实。面对事实，只能考虑如何处理它，从而使它的负面影响降到最小。面对硬件故障，软件失效，人为操作失误等事实，系统安全领域的专家将此定律视为提高系统可靠性的法宝。由分别来自美国斯坦福大学和加州大学伯克利分校的科研人员组成的联合开发小组提出了一种新方针，他们将研究的重点从传统的单纯的提高系统性能，避免攻击或容错转向到如何从已发生故障中快速恢复，从而保证系统的高可用性，称之为面向恢复计算 ROC (recovery-oriented computing)[1]。

微重启正是在 ROC 项目框架内提出的，针对大型分布式应用软件系统发生故障时的快速恢复技术。首先应该指出，微重启技术以及在其基础上发展起来的适毁性软件设计思想针对的是诸如以 Internet 为平台的大规模分布式应用软件系统，此类系统多由大量相互关联的短运行任务而非长运行操作组成，因此在固定时间域内由重启造成的任务的丢失及对整体的影响相对较小；并且研究表明，此类系统故障多具有间歇性和暂时性特点并可通过重启完全解决[2,3]。微重启技术有别与传统的重启方式(宏重启)，它采用递归恢复[4~6]的方法，即将系统划分为多个故障隔离的组件子集，首先重启可能引起故障的最小子集但不影响系统其他部分的正常运行，如果不起作用，再依照故障传播路径递近地重启更大范围子集，直到故障最终解决或者需要其他恢复策略的执行。微重启(递归重启)可以有效避免系统因全面重启而造成的数据丢失和事务进程的中断，并且极大地缩短了因全面重启而引起的冗长恢复时间；通过快速地解决局部故障以避免整体宕机，从而提高了应用系统的可用性。

2 微重启

研究人员选择重启作为系统的恢复技术有以下三点原因[2]：(1)重启本身容易理解和执行。(2)重启实现了对过时或溢出资源的可靠性回收，使系统回到可信状态。(3)重启使系统回到初始时最易理解和测试分析的状态。

图 1[2]是一个基本的可递归恢复系统的框架。

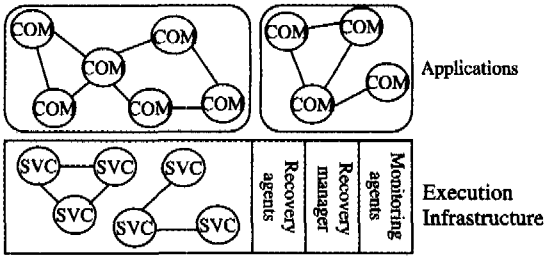

图 1 基本可递归恢复框架

执行平台(execution infrastructure)可以是操作系统、应用服务器或者 JVM，他们为应用程序提供一系列服务(svc)，应用程序(applications)本身是由大量相互关联的组件构成(com)。系统中可独立恢复的最小单元(com 和 svc)叫做一个恢复单元(r-unit)。检测代理(monitoring agent)负责实时监测系统的运行状态，查找系统故障并将相关信息传送给恢复管理器(recov-

*)基金项目：国防预研重点资助项目(413150702)和武备预研基金资助项目(51416060104CB0101)。朱 岩 硕士研究生，研究方向为计算机网络、信息安全；王慧强 教授，博导，研究方向为可靠性理论、计算机网络。

ery manager),恢复管理器基于以上信息和相关历史记录决定具体恢复路径,从而触发恢复代理(recovery agent)执行微重启操作。

2.1 故障检测

检测代理在三个层面上对系统的运行情况进行监测:平台层,应用程序层,端到端。平台层以组件的基本行为为参照,例如操作系统内核时时监视一个进程的内存使用率以及 I/O 操作,JVM 检查应用程序同步原语的使用等;应用程序层综合使用程序计数器和行为监视器进行检测,如果组件长时间内没有运行即可推断其故障的发生,又如行为监视器在一段时间内没有接收到来自集群中相关节点发出的周期性心跳信号即可推测出其已经失效;端到端检测使用终端用户可视界面验证系统的存活性,例如执行一个 SQL 请求,验证当前状态下后台数据库的可用性。

检测代理将所有系统运行状态的显著变化信息递交给恢复管理器,这些信息不仅包括发生故障的组件信息,还包括系统中新组件的部署及无用组件的剥离信息。

2.2 递归恢复图的建立

恢复管理器动态的保存系统的故障传播信息,当它接收到从检测代理发送过来的故障信息时相应的计算最优的递归恢复路径。首先我们需建立一个故障描述图,图中将部署到系统的组件作为节点,直接的故障传播路径作为组件间的边。研究人员发明了一种技术描述系统的故障传播信息,称之为自动故障路径推断 AFPI(automatic failure-path inference)[7,8]。此过程包括两个阶段:在发生入侵阶段,恢复管理器对系统执行故障注入操作,监测系统对注入故障的反应,建立基本的故障描述草图;在无入侵发生阶段,恢复管理器被动的观测正常操作情况下系统发生故障以及故障的传播情况,并在第一阶段的基础上对描述图进行优化。

故障描述图中的一组节点可能会表现出一种循环的故障依赖关系,为了保证组件恢复的独立性,这样的一组节点必须被当作故障描述图中的一个独立单元。恢复管理器将计算故障描述图中的所有连通分支并将这样的循环子图分割成为一个个独立单元,最终得到的递归恢复图应该是非循环的。

2.3 递归恢复过程

假设恢复图中有两个恢复单元 A,B,由一条由 A 指向 B 的有向边连接,我们称 A 为 B 的直接上游节点,B 为 A 的直接下游节点[2]。在恢复过程中,假设要恢复系统中一个已发生故障的节点 r,实际上是恢复可能被 r 传播而发生故障的所有 r 的下游节点。我们将待恢复节点 r 作为一个对象,它有两个可调用的方法 pre-recovery()和 post-recovery(),恢复过程如图 2 所示。

```
recover(r)
    invoke r.pre-recovery()
    for each r-unit r_i  immediately
        downstream from r
    invoke recover(r_i)
    invoke r.post-recovery()
```

图 2 节点 r 的一般恢复过程

节点 r 调用 pre-recovery()方法递归的恢复它的所有下游节点,调用 post-recovery()方法执行重启操作。如果在 recover (r)完成后,检测代理发现系统故障仍然存在,那么恢复管理器推断节点 r 的故障可能是由其上游节点发生故障而传播过来的,这时恢复代理将在与 r 关联的所有直接上游节点上调用 recover()方法,这种递归恢复过程不断地重复执行直到故障的消除或者必要的人为干预的介入。

3 应用实例研究

微重启技术首次被应用到一个成熟的软件系统中—Mercury 地面接收站,此应用系统的主要任务是在科学实验卫星通过地面接收站上空时接收卫星传回的科学数据,对卫星运行轨道进行计算和预测等。Mercury 是一个基于 Internet 平台的软状态系统。软件组件间的控制和互操作逻辑是通过由 XML 命令语言编写的消息包来完成的,所有消息包通过一个基于 TCP/IP 协议的消息总线进行传递。由于 Mercury 的软状态特性,系统组件自身不需要保持持续的状态信息,因此可安全重启。

3.1 重启树优化过程

研究人员在不改变程序任何源代码的前提下将系统划分为多个具有良好边界的组件子系统,特别的是,其递归恢复图呈现出简单的树型结构,如图 3 中树Ⅰ所示:

其中圆圈为组件节点,其各自完成的功能本文不做具体介绍;方框为恢复代理,可以看作直观概念上的重启按钮,按下按钮其管辖的节点集将重启。具体优化过程如下。

3.1.1 节点层的降低 图 3 是原始微重启树的第一步优化,根据微重启的工作原理,在树Ⅰ中,发生在任何组件的故障都将导致树中唯一的重启按钮被按下,即导致系统的整体重启,则 $MTTR_I = \max(MTTRci)$,其中 ci 为节点集的第 i 个子节点。而树Ⅱ中每个组件配置一个恢复代理,即每个组件可在不影响其他组件的情况下独立重启,整个系统的 MTTR 为单一组件的恢复时间,又经过实际测试表明,在树Ⅰ中,整体重启导致的资源争夺实际增大了 $\max(MTTRci)$,而这种情况在树Ⅱ中是不会出现的,因此我们最后得出结论 $MTTR_{II} < MTTR_I$。

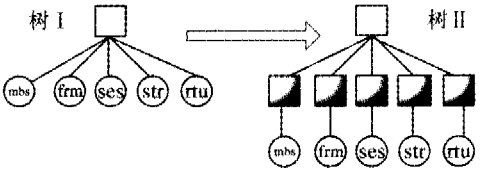

图 3 节点层的降低

3.1.2 节点的分解 研究人员经过观测发现树中的 frm 组件具有频繁失效及恢复时间长的缺点,即高 MTTR 低 MTTF,因此降低了系统的可用性。但是幸运的是 frm 本身包括两个组件:pbc 和 fed,pbc 较稳定,但是重启恢复的时间较长,而 fed 不稳定但是恢复时间相对较短。将组件分离后的重启树如图 4 中的树Ⅲ所示,这种分离需要系统配置的改变但是不需要修改任何程序源代码。

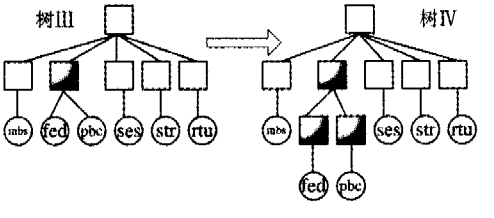

图 4 节点分解及子节点层的降低

根据第一步优化的经验,使子组件具有可独立重启能力。从优化后的树Ⅳ可以看出,先前发生在 frm 中的故障现在可通

过单独重启 fed,pbc 或者同时重启 fed 和 pbc 进行修复。因为 $MTTR_{fed} \ll MTTR_{pbc}$ 且 $MTTF_{fed} \ll MTTF_{pbc}$,因此大部分故障将由快速微重启 fed 进行修复,从而取代了树Ⅱ中故障只能通过缓慢重启 frm 进行恢复的情况,降低了整体的 MTTR。

由于现在软件工程技术的现实,fed 和 pbc 没办法实现完全分离。研究人员发现若干次 fed 故障的发生会最终导致 pbc 的失效,这时需要递归到 fed 恢复代理的父节点,同时重启 fed 和 pbc 两个组件,这是树Ⅳ中设置两组件的结合恢复点的第一点原因。如果 frm 分解后直接将两子节点设置成递归树的根节点的第一层可独立重启节点,那么当出现上述关联故障时,fed 重启后检测代理发现故障仍然存在,即递归到根节点从而导致树的所有组件节点的重启,反而提高了系统 MTTR,这是设置子组件结合恢复点的第二个原因。

3.1.3 *节点的合并* 研究发现,微重启树中的 ses 和 str 两组件虽然可独立构造,但是却有一种功能上的依赖关系,即其中任意组件重启后都需要等待对等组件重启,然后以初始化状态重新同步工作。这种组件间的功能依赖导致了故障依赖的产生,恢复代理重启其中某一故障组件后被告知关联故障存在于对等组件,因此还需要重启另一组件。根据递归恢复图的构建原则,将有故障依赖关系的两个组件的恢复代理合并成一个节点,任何一个组件发生故障都会导致两组件同时重启,这时系统的 MTTR 由原来的 $MTTR_{ses}+MTTR_{str}$ 转化为 $\max(MTTR_{ses},MTTR_{str})$,进一步降低了整体的 MTTR。优化过程如图 5。

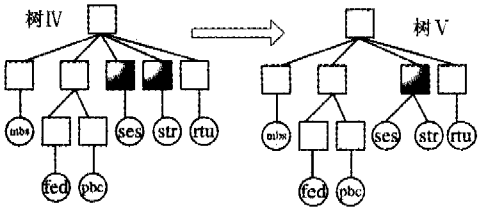

图 5 节点合并

3.1.4 *节点层的提升* 在节点的分解中已经提到,fed 和 pbc 实际并未完全分离,系统中同样存在这样一种特殊故障,在 pbc 中出现却需要同时重启 pbc 和 fed 才能最终修复。这时恢复管理器经常做出误判,只单独重启 pbc,当发现故障仍然存在时再递归到上层恢复代理节点,这样增加了 MTTR,因此将 pbc 提升到上一层,如图 6 中树Ⅵ所示,这样只要故障出现在 pbc 中即重启整个子树,同时重启 fed 和 pbc,此种优化进一步降低了系统 MTTR。

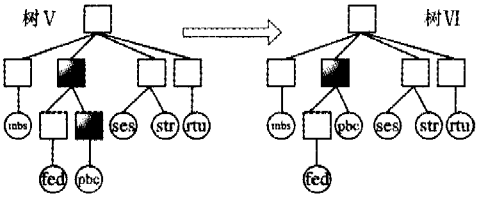

图 6 节点层的提升

3.2 小结

如上描述可以看出,微重启树的优化过程是以降低系统 MTTR 为根本目的的,不同的系统应该根据其具体运行情况和组件的配置情况做出相应的优化。

通过微重启技术的引入,系统整体的 MTTR 由原来的十分钟减少到两分钟,保证了系统的高可用性要求。从此例也可以看出,关键系统 MTTR 带来的损失是非线性的[1,9],两分钟的 MTTR 可能对系统任务没有大的影响,而十分钟的 MTTR 也许会错过卫星短暂通过地面接收站上空时其数据接收,轨道运算等关键性任务。因此,对于有高可用性要求的应用系统来说-如军用系统、在线电子商务系统等,减少 MTTR 比单纯提高 MTTF 更有价值。

4 适毁性软件设计

适毁性软件设计是对微重启广泛有效应用的一种扩展,以使其更加安全高效的应用到多种分布式应用系统中(并非只有软状态系统)。适毁性软件基本设计思想:可安全失效,可快速恢复[3]。

以 Internet 为平台的大型企业级应用系统一般包含大量分布式的异构组件,组件间进行快速持续的事务型状态信息改变一例如用户会话状态信息。由于此类系统结构和计算任务的复杂性,使用传统的故障处理模型分类是不切实际的,因而研究人员提出了新的方法,即强制性失效故障组件一即使不知道故障产生的根源,随即通过微重启进行快速恢复。这样做的好处是将系统所有随机故障的处理模型转化为一个简单的等式:故障=失效,恢复=重启,恢复管理器只需知道如何处理这种单一故障,因而最大限度地降低了故障分析和处理的时间。

4.1 满足适毁性的系统特性[2,3,10,11]

组件细粒度设计:故障组件的恢复时间主要取决于组件运行的底层平台对其重启及初始化的时间,因此应该依据程序逻辑和启动时间将组件设计的尽可能地小,开发者可以充分的使用已有的较成熟的基于组件的设计框架(J2EE[12,13]、.NET 等),将系统分割成良好边界的细粒度组件。

状态信息隔离:为了保证微重启过程中组件进程的持续性以及减少数据的丢失,使用专用状态存储器来保存重要的状态信息,他们被部署在应用程序之外。数据/进程恢复的分离不仅保证了微重启的安全性,也使应用程序的恢复变得更加简单,提高了系统整体的健壮性,因为这时程序员只需要专注于系统功能逻辑的设计,而将繁重的数据管理工作转交给专门的数据库管理员。

组件边界:所有的组件必须是松耦合连接的,同时他们具有良好定义的强制性故障隔离边界,使组件故障及组件的重启过程对系统的影响降到最低。

组件间通信:组件间使用超时响应信号进行交互,如果一个请求在规定的时间域内未被响应,那么发出请求组件将假设被呼叫组件已经发生故障并将此信息传送给恢复管理器,恢复代理将对其执行微重启操作。相似的,如果一个组件调用了一个正在进行微重启的组件,它将接收到一个 t 秒后重试的异常信号,请求将会在 t 秒后重新发出(如果这一请求的存活时间在 t 时间域内),这种做法保证了故障组件重新快速融入到系统中。重启/重试机制透明的恢复暂时未被响应的请求,相对于终端用户隐藏了系统已发生的故障以及其恢复过程。

资源租借:在一个频繁进行微重启的系统中,资源的使用应该使用租借机制而不是持续性分配,以确保系统资源的可用性。如果一个请求的计算被挂起并且无法更新,其 CPU 的使用应该被中止;一定形式的持续状态应该使用一段较长的资源租借期,到期后其状态信息将被删除或存档,与之相关的资源将被释放。租借机制有效地减少了系统挂起和拥塞现象的发生。

5 应用改进方向

由于微重启技术部署、执行成本的低廉以及快速恢复故障的特点，研究人员倾向于将其部署在故障恢复的最前线一即使不知道故障产生的根源以及此故障是否可最终通过微重启操作而被解决，如果不起作用，再执行其他的恢复操作，而微重启执行过程的损耗基本可以忽略。根据微重启技术的特点进一步提出以下两点应用改进方向：

(1)组件级更新：软件系统普遍存在内存泄漏，未释放锁定的文件，文件描述符泄漏等缺陷，这些缺陷在系统运行过程中不断累计会导致软件性能逐渐下降，甚至会使整个系统失效，这种现象称为软件老化。传统软件更新[14]采用成本可控的有计划性停机重启来消除软件老化现象，避免了突发性系统失效带来的损失，但是系统整体停机的损失不可忽视。基于微重启技术的组件级更新，通过计划性的组件微重启解决软件老化现象，但是与传统的系统整体重启相比，其损耗相对较小。

(2)组件级失效转移：一般的大型电子商务公司为了满足大量的数据处理和用户请求的需要，都在服务器端采用多节点的集群形式，并且为了提高整体可用性，普遍采用节点失效转移的故障处理策略。本文提出组件级的失效转移策略，即设定某些特定组件的失效转移，例如一些关键任务或者高 MTTR 的组件，当这些组件进行微重启时，对他们的应用请求都将转移到集群中未发生故障的其他节点上，这样进一步减少了微重启对系统可用性的影响。

结束语 微重启技术主要针对的是大型分布式应用系统，这是由此类系统的组件化设计框架和系统频发故障特点决定的。而适毁性软件设计思想的实质是使应用系统具有安全高效的"可微重启性"。对于大型应用系统来说，可用性直接关系到终端用户满意度指标，包括用户请求的响应时间和响应能力等，微重启通过局部故障的快速解决提高了用户可感知的系统可用性，因此具有非常重要的现实意义。但是，微重启技术部署和实施过程还有许多细节需要研究和完善，例如对于不同应用系统，组件的粒度设计和递归恢复图的优化是一难点；专用状态存储器中组件状态信息的抽象；微重启的部署和执行对系统性能的影响等等。这些都需要后续更进一步的研究。

现今，基于组件构架的分布式应用系统在人们的现实生活中以及科研、军事领域都得到了越来越广泛的应用一例如企业电子商务系统、军用指挥系统等，可以预见微重启技术将有更加广阔的发展前景，基于微重启的自动、快速恢复技术的研究对系统可靠性研究领域将具有不可估量的巨大作用。

参 考 文 献

1 Patterson D, Brown A, Broadwell P, Candea G, et al. Recovery oriented computing (ROC): motivation, definition, techniques, and case studies, Technical Report UCB/CSD-02-1175, UC Berkeley, Berkeley, CA, March 2002
2 Candea G, Cutler J, Fox A. Improving availability with recursive microreboots: a soft-state system case study. Performance Evaluation Journal, March 2004, 56: 1～3
3 Candea G, Fox A. Crash-only software. In: Proc. 9th Workshop on Hot topics in Operating Systems, May 2003
4 Candea G, Fox A. Recursive Restartability: Turning the Reboot Sledgehammer into a Scalple. 8th Workshop on Hot Topic in Operating System, Schloss Elmau, Germany, May 2001
5 Candea G, Fox A. Designing For High Availability and Measurability. 1st Workshop on Evaluation and Architecting System Dependability, Göteborg Sweden, July 2001
6 Candea G, Cutler J, Fox A, et al. Reducing Recovery Time in a Small Recursive Restartable System. International Conference on Dependable System and Network, Washington, D. C, June 2002
7 Candea G, Kiciman E, Zhang S, Keyani P, Fox A. JAGR: An Autonomous Self-Recovering Application Server 5th International Workshop on Active Middleware Service, Seattle, WA, June 2003
8 Candea G, Delgado M, Chen M, Fox A. Automatic Failure-Path Inference: A Generic Introspection Technique for Internet Applications. 3th IEEE Workshop on Internet Applications, San Jose, CA, June 2003
9 Fox A, Patterson D. When Does Fast Recovery Trump High Reliability? 2th Workshop on Evaluating and Architecting Systems for Dependability, San Jose, CA, October 2002
10 Candea G, Kawamoto S, Fujiki Y, Friedman G, Fox A. Micro reboot: A Technique for Cheap Recovery. In Proc. 6th Symposium on Operation System Design and Implementation, 2004
11 Candea G, Kiciman E, Kawamoto S, Fox A. Autonomous Recovery in Comoponentized Internet Application. Cluster Computing Journal, Kluwer Academic Publishers(to appear 2005)
12 窦蔷，袁臻，刘冬梅. 基于构件的中间件技术 J2EE. 计算机科学，2004，31(6)：13～17
13 韦华颖，詹剑锋，王沁. 分布式构件技术综述. 计算机应用研究，2004，21(10)：12～15
14 Huang Y, Kintala C, Kolettis N, Funton N D. Software Rejuvenation: Analysis, Module and Application. In: Proc. 25th IEEE Int'l Symp, On Fault Tolerant Computing, IEEE Computer Society Pree, Los Alamitos, CA, 1995. 381～390

(上接第 131 页)

表 3 性能比较结果

性能提高百分比	OnceStAXParser 比 BEA RI	OnceStAXParser 比 Woodstox	OnceStAXParser 比 SJSXP
TEST1	34.64%	6.71%	5.38%
TEST2	36.58%	3.37%	4.71%
TEST3	44.38%	3.82%	5.10%
平均值	38.54%	4.63%	5.06%

小结 本文在对 XML 语法进行研究的基础上，设计了符合 StAX 规范的解析 XML 的自动机模型，构造了 Java 运行环境下的 XML 解析器。经过 XML 兼容性测试和 StAX API 兼容性测试表明，OnceStAXParser 完全符合 XML 规范和 StAX 规范要求。对其进行的性能测试结果表明，OnceStAXParser 的解析效率要明显高于同类产品。

今后工作将在以下方面展开：①增加对有效性验证的检查；②利用延迟解析技术，进一步提高性能；③将 OnceStAXParser 应用到不同的应用领域，比如 SOAP 引擎中。

参 考 文 献

1 Brownell D. SAX2. O'Reilly & Associates Inc, ISBN: 0-596-00237-8, 2002
2 W3C. Extensible Markup Language(XML) 1.0(Third Edition). http://www.w3.org/TR/2004/REC-xml-20040204, 2004
3 W3C. Namespaces in XML. http://www.w3.org/TR/1999/REC-xml-names-19990114, 1999
4 Hopcroft J E, Motwani R, Ullman J D. Introduction to Automata Theory, Languages, and Computation 2nd Ed. Addison-Wesley Publishing Company, 2001
5 Java Community Process. Streaming API for XML JSR-173 Specification(Final v1.0). http://jcp.org/en/jsr/detail?id=173, 2003
6 W3C. Extensible Markup Language(XML) Conformance Test Suites 20031210. http://www.w3.org/XML/Test/, 2003
7 TatuSaloranta. StAX conformance test suit. http://www.cowtowncoder.com/, 2004
8 Sun Microsystems. XML Test(1.0). http://java.sun.com/performance/reference/codesamples, 2004
9 Jack Shirazi. Java Performance Tuning Second Edition. O'Reilly & Associates Inc, ISBN: 0-596-00377-3, 2003
10 Li Quanzhong, Michelle K, Edward S, et al. XVM: A Bridge between XML Data and Its Behavior. In: Proceedings International WWW Conference 2004, 2004. 155～163
11 Bloch J. Effective Java: Programming Language Guide. Addison Wesley, 2001
12 陈火旺，等. 程序设计语言编译原理. 第 3 版. 北京：国防工业出版社，2000
13 蒋宗礼，姜守旭. 形式语言与自动机理论. 北京：清华大学出版社，2003