支持 StAX 的高效 XML 解析器的设计与实现*)

任 鑫 曹冬磊 金蓓弘

(中国科学院软件研究所软件工程技术中心 北京 100080)

摘 要 StAX是JCP提出的一种新的XML解析方式,它提供给用户更多的解析控制权。本文给出了用于XML语法分析的下推自动机模型的设计以及StAX解析器OnceStAXParser的实现。OnceStAXParser在经过了严格的XML兼容性测试和StAX API兼容性测试之后,还从多个方面进行了性能优化,包括自动机实现优化、有计划的预分配和延迟处理策略以及适度封装策略等。性能测试数据表明,OnceStAXParser的吞吐量比Sun SJSXP平均高5%,比BEA StAX RI平均高38%。

关键词 XML解析器,StAX,性能优化

Design and Implementation of an Efficient StAX Based XML Parser

REN Xin CAO Dong-Lei JIN Bei-Hong

(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract StAX is presented by JCP JSR-173 specification which supports XML pull parsing and gives more parsing control to users. This paper describes the design of pushdown automaton model for XML syntax analysis and illustrates the implementation of StAX parser OnceStAXParser. After passing the rigorous XML conformance tests and StAX API conformance tests, OnceStAXParser is optimized from many aspects such as the implementation of pushdown automaton, arranged pre-allocation and lazy processing, appropriate encapsulation strategy, etc. The performance test results from XML Test suite show that the throughput of OnceStAXParser is 5% more than that of SJSXP on the average, and is 38% more than that of BEA StAX RI on the average.

Keywords XML parser, StAX, Performance optimization

1 引言

作为一种数据表示和数据交换的标准,XML广泛应用于文档处理领域(例如XHTML、XSLT)、数据库领域(例如XML数据库)、Web服务领域(例如SOAP消息、WSDL等)。在某些应用场合,例如在Web服务领域,因为系统传递的消息是基于XML的,当系统的消息比较密集、并发程度比较高的时候,XML解析会成为整个系统性能的一个瓶颈。所以,提高XML解析效率具有非常重要的实际意义。

目前常用的解析XML的API是SAX(Simple API for XML processing)和DOM(Document Object Model)。尽管这两种方法都存在各自的优点,但也有很明显的缺点。

DOM解析器需要将整个文档解析一遍,并将解析结果以树型结构保存到内存中(这棵树被称为DOM树),然后才把程序的控制权交给用户。这导致了DOM方式不够灵活,需要占用大量的资源。解析XML文档所用的内存大小和XML文档大小一般要达到10:1的比例[1]。也有一些DOM解析器采用延迟节点展开技术,这样可以实现部分解析文档的功能,但这种实现方式也要占用大量的资源,开销很大,解析效率依然不高。

SAX解析器在解析过程中触发事件并激活用户预定义的回调(callback)方法。用户不能对解析过程加以控制,不能进行迭代的处理,使用上不够灵活。

与DOM、SAX相比,StAX(The Streaming API for XML)是以拉(PULL)方式解析XML的Java API,目前已经通过了JCP的审核,成为了JSR-173规范。StAX接口具有灵活小巧的特点,用户可以控制StAX的解析过程,比较适合于网络计算和移动环境的应用需求,有利于实现高效地解析XML文档。BEA公司提供了StAX的参考实现BEA StAX RI,但该实现并不完全符合StAX规范的要求,存在功能缺陷,它只通过了大部分的XML兼容性测试。SJSXP是Sun公司开发的基于StAX的高效XML解析器,符合XML 1.0和Namespace 1.0规范。Woodstox是一个开源的StAX XML解析器,目前的版本是2.0。Microsoft在.NET框架中提供的XmlTextReader,尽管不支持StAX接口,但也是一种拉方式的解析器。总之,XML拉式解析正在成为工业界关注的热点。

我们按照XML 1.0[2]和Namespace 1.0[3]规范,构造了支持StAX接口的XML解析器OnceStAXParser。它支持XML的良构性约束(well-formedness constraint, WFC)的验证,但不进行有效性约束(validity constraint)的验证。

本文给出了OnceStAXParser的设计和实现,按如下方式组织:第2节描述了实现XML语法分析的下推自动机模型,第3节给出了StAX的接口实现,第4节总结了我们在性能优化方面所采取的措施,第5节介绍了性能测试方法和测试结果,最后是全文小结。

*)本文工作受国家973项目(编号2002CB312005)、国家863项目(编号2001AA113010)的资助。任 鑫 硕士研究生,研究方向:分布式计算、软件工程;曹冬磊 博士研究生,研究方向:分布式计算、软件工程;金蓓弘 博士,副研究员,研究方向:分布式计算、软件工程。

2 XML 的语法分析

这一节在分析 XML 语言的基础上，构造了 XML 语言自动机，然后将它应用到 StAX 接口中，设计符合 StAX 规范要求的状态机模型。

2.1 XML 语言自动机设计

XML 语法采用 EBNF(Extended Backus Normal Form)表示，其中以大写字母开头的符号都是正则语言的开始符号，因此，可以构造确定的有穷状态自动机(deterministic finite automation，DFA)来识别它们。例如对于一个注释(Comment)，可以通过“<! --”来判断一个标记(markup)是注释，然后调用词法分析器中的 getComment 的方法来得到注释中的字符。图 1 给出了解析 Comment 的状态转移图。

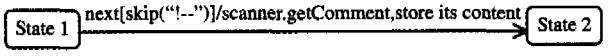

图 1 解析 Comment 的状态转移图

我们以这些大写字母开头的符号作为最基本的语法符号，对以小写字母开头的非终结符做推导，并做产生式的替换，消除左递归，尽量使产生式由大写字母开头的符号、终结符和有穷语言运算(并、乘、Kleene 闭包)组成，同时进行化简。最后，构造下推自动机，自顶向下对 XML 文档进行语法分析和处理。

与语法分析有关的 XML 语法产生式有下面几条：(数字对应 XML 规范中语法产生式的序号)

[1]document::=prolog element Misc*

[22]prolog::=XMLDecl? Tisc*(doctypedecl Misc*)?

[27]Misc::=Comment|PI|S

[39]element::=EmptyElem Tag|STagcontentETag

[43]content::=CharData? ((element|Reference|CDSect|PI|Comment)CharData?)*

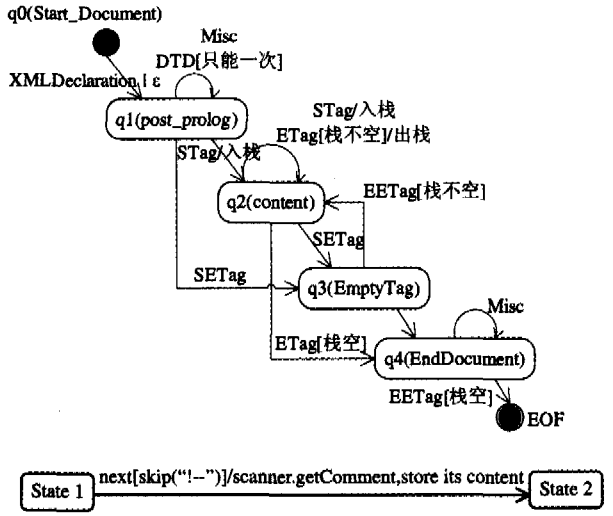

图 2 XML 语言自动机的状态转移图

其中，产生式 27 是一个正则表达式，所以可以构造有穷状态自动机来识别它们；考虑到 OnceStAXParser 是非验证型 XML 处理器，可以简化对某些产生式(例如产生式 45 elementdecl)的解析，于是可以将产生式 22 变换为正则表达式，从而也可以用有穷状态自动机来识别；对于产生式 39、43，根据泵引理(pumping lemma)[3]，它们是上下文无关文法，所以可以构造一个下推自动机来识别它们。为了区分 prolog 中的 Misc，将这里的 Misc 改称 End_Document。图 2 给出了语法分析得出的完整的 XML 语言自动机的状态转移图。为了简洁起见，图上没有标明异常情况。

2.2 XMLStreamReader 状态机设计

StAX 规范将 XML 语言中的每个标记和一种事件类型相对应[4]。用户每调用一次 XMLStreamReader 提供的 next 方法，XMLStreamReader 就向前解析一个事件，并返回事件类型。用户在下一次调用 next 方法之前，可以通过调用 XMLStreamReader 的 getXXX 方法得到当前事件的属性。按照上一小节得出的状态转移图，我们设计了符合 StAX 规范的解析 XML 语言的状态机(受用户调用 next 方法的驱动)，如图 3 所示。

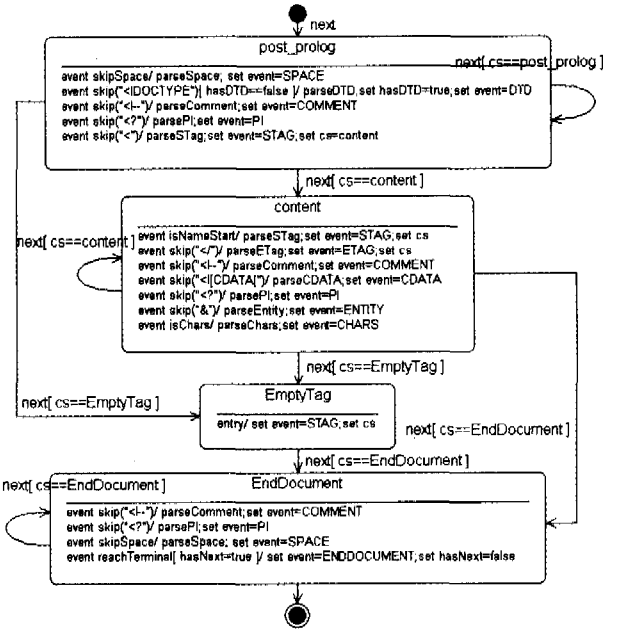

图 3 描述符合 StAX 规范的 XML 解析过程的状态图

2.3 良构性约束的实现

在语法分析器的设计过程中，除了要求满足语法定义以外，还要实现 XML 规范中定义的良构性约束。例如，WFC 要求 XML 文档必须符合产生式 document 的定义。语法解析自动机的设计可以保证这一点。WFC(Element Type Match)要求：ETag 中的元素名应该与 STag 中的元素名对应。那么，OnceStAXParser 在创建解析器对象的同时创建一个栈，用于在解析 STag 的时候记录元素名字；在解析 ETag 时，通过核对栈顶元素的元素名完成该项 WFC 的检查。WFC(Unique Att Spec)要求：一个属性名不应该在同一个元素或空元素中多次出现。那么，OnceStAXParser 在处理完 STag 中的所有属性后，对于属性列表统一进行该项约束的检查，判断是否有重名的属性。

为了增加对于名字空间的支持，我们创建了 NamespaceContextImpl 类对 XML 文档中出现的名字空间声明进行管理。Namespace1.0 规范也列出了一些良构性约束，我们在 XMLStreamReader 的名字空间版本的设计中增加了对这些约束的验证。关于名字空间主要的约束内容与解决方法如下：

• WFC(NCName 产生式)：XML 规范中的 Name 产生式更换为 NCName 产生式；在词法分析器中增加对 NCName 的分析方法，用于支持名字空间的 XMLStreamReader 对名字的

解析；

• WFC(Prefix Declared)：在解析 STag 的同时检查名字空间前缀(prefix)是否绑定到一个 URI；

• WFC(Namespace Scoping)：由 NamespaceContext 类进行控制，在处理一个元素的 STag 的时候，创建一个新的作用域，在处理这个元素的 ETag 的时候，退出这个作用域；

• WFC(Uniqueness of Attributes)：属性(包括名字空间)可以通过它的本地名(localname)和由前缀绑定的 URI 的值区分。在解析器处理完 STag 中的所有属性后，对于属性列表统一进行该项约束的检查，即首先处理所有的名字空间声明，然后确定所有属性前缀绑定的 URI，再判断是否有重名的属性。

3 接口实现

StAX 主要提供两大类接口：基于指针的 API，包括 XMLStreamReader、XMLStreamWriter；基于事件的 API，包括 XMLEventReader、XMLEventWriter、XMLEvent 等。

XMLStreamReaderImpl 是 XMLStreamReader 的实现类，图 4 给出了它与其它相关类的关系。AttrManager 是属性管理的核心类。EntityManager 是实体管理的核心类。DTDParser 是解析 DTD 的核心类。NamespaceContextImpl 负责实现名字空间管理。StringBuffer 类提供操纵变长的字符数组的功能。LocatorImpl 记录字符流的读取位置，包括行、列值。Configuration 类保存系统属性。Scanner 是词法分析的核心类。Scanner 通过字符解码器从输入流中读取 XML 文档内容，分析出 XML 规范中定义的基本符号(例如 Name、CharData 等)给语法分析部分，作为语法分析自动机的输入。在这部分主要包括以下两部分的工作：

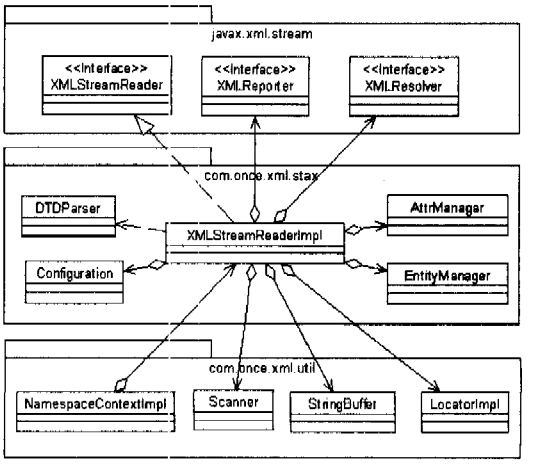

图 4 XMLStreamReaderImpl 类图

• 设计高效的解码算法，将输入流按照它的编码方式解码到创建好的缓冲区中；

• 构造有限自动机，对缓冲区中的字符进行分析，得到终结符。

图 5 给出了典型 XMLStreamReader 使用的实现流程。

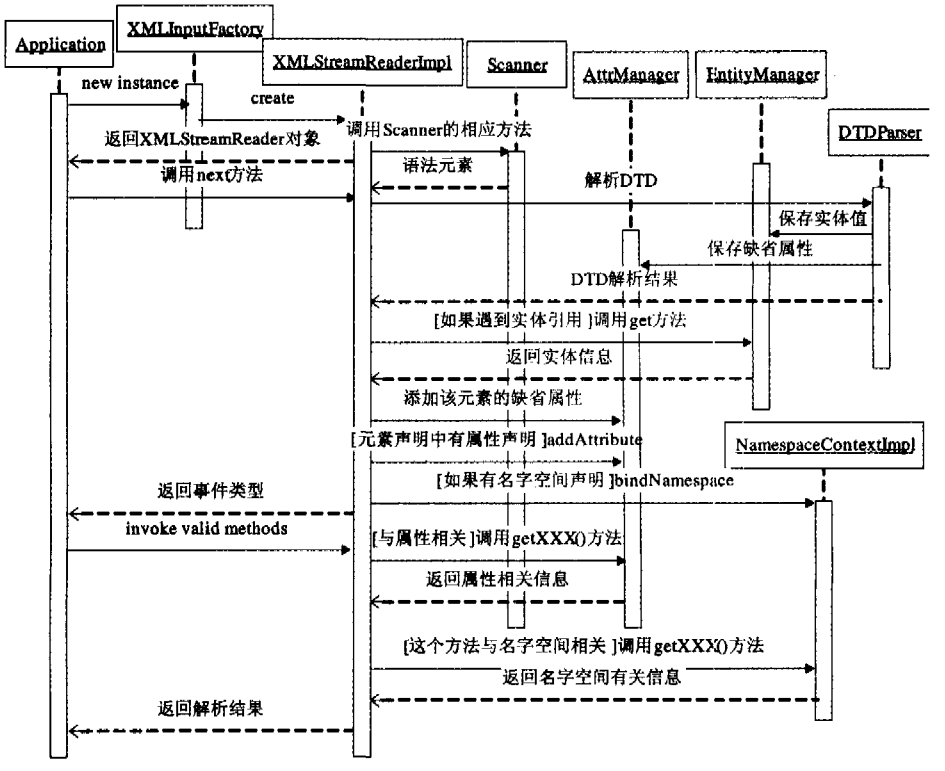

图 5 XMLStreamReader 使用的实现流程图

XMLEventReaderImpl 是 XMLEventReader 接口的实现类，它将一个 XMLStreamReaderImpl 对象作为成员变量，通过调用 XMLStreamReaderImpl 的方法完成解析，然后将解析结果封装成事件对象，返回给应用程序。

XMLStreamWriterImpl 是 XMLStreamWriter 接口的实现类，它的实现是利用字符编码器将用户输入写到输出流当中，其中如果遇到特殊字符，需要把它们转换成字符引用的方式。

XMLEventWriterImpl 是 XMLEventWriter 接口的实现类，它的实现是建立在 XMLStreamWriterImpl 基础上的，通过调用 XMLStreamWriterImpl 相应事件下的处理方法来完成其功能。

XMLEvent 接口定义了基本的事件类型，通过实现 XMLEvent 及其子接口来保存不同事件的信息。

为了验证 OnceStAXParser 的正确性，我们进行了严格的 XML 兼容性测试和 API 兼容性测试。XML 兼容性测试采用的是 W3C 的 XML Conformance Test Suite[6]，其中包含了 2000 多个测试文件。我们采用 javanet[7] 上开源的测试包对 OnceStAXParser 进行 StAX 接口的兼容性测试。OnceStAXParser 已经全部通过了这些测试。

4 性能优化

虽然通过减少自动机的状态可提高解析效率，为了进一步提高解析器的性能，还需要进行多方面的性能优化。

4.1 基于统计的自动机实现优化

XMLStreamReader 的解析主过程-next 方法接收来自词法分析识别出的标记，并通过判断标记的类型做相应的解析。在 next 方法中，对这些标记的判断是顺序执行的，所以可以通过优化这些判断的先后顺序来提高解析效率。

经过对大量常用的 XML 文件的测试，统计出各种标记(STag，ETag，Comment，CDATA，PI，Entity，Chars)出现的频率，如表 1 所示。根据上面的统计可以看出，字符数据(Chars)和 STag、ETag 出现频度占有绝对优势，所以我们优先判断这些标记的类型。

表 1 不同标记的出现频率

标记	STag	ETag	Comment	CDATA	PI	Entity	Chars
出现次数	190819	190819	2602	78	103	26	380838
比例	24.93%	24.93%	0.34%	0.01%	0.01%	0.00%	49.76%

4.2 有计划的预分配和延迟处理

在 Scanner 中，我们预先分配了一定大小的字符缓冲区来保存解码后的字符数据。如果缓冲区过小，会带来频繁调用填写缓冲区的操作，增加总的 I/O 操作；如果缓冲区过大，每次读取都会需要调用多次 I/O 操作才能填满缓冲区。所以，字符缓冲的大小会影响到整个解析器的性能。我们经过对一系列不同缓冲大小的性能测试之后，选取了性能最优的 8k 作为 Scanner 的字符缓冲的大小。测试数据如图 6。

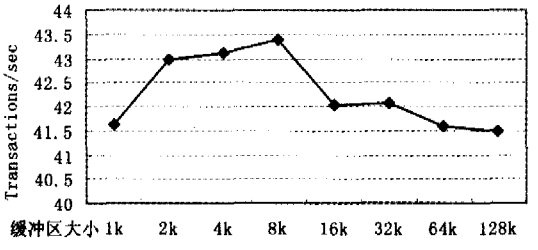

图 6 不同字符缓冲下的性能测试数据

按照 XML 语法定义，一些 XML 元素可以不出现在 XML 文档中，所以在解析过程中可以采取延迟创建对象的策略：只有当用到该对象的时候再创建。例如，只有当解析到 DTD 的时候，我们才创建解析 DTD 的 DTDParser。

4.3 对象重用与对象池技术

由于解析过程中需要创建和回收数量较多的字符数组，而 Java 虚拟机对于对象的创建和系统垃圾回收都需要较大的时间开销，因此，为了提高效率、减少对象的生成，在解析开始的时候，创建一个对象池，保存所有创建的字符数组对象，并在解析下一个事件前回收这些对象，以便循环使用。使用对象池和不使用对象池两种方式的测试数据如表 2(TEST1、TEST2、TEST3 的定义参考第 5 节性能测试部分)。

表 2 采取两种不同方式的性能比较结果

	TEST1	TEST2	TEST3
不使用池方式	41.84	83.753334	164.53667
使用池方式	43.36	85.8557143	171.23
性能提高比例	3.63%	2.51%	4.07%

4.4 针对面向对象语言的特点的性能提高

继承、封装和多态是面向对象语言的 3 大特征与理论基础。但对于面向对象思想不适当的过度使用，可能会比较严重地影响计算机程序的效率。因此，在对性能要求较高的软件项目中，核心部分的代码有时候必须在对象封装性与代码效率两者之间进行权衡，适当地舍弃部分继承、封装与多态性，而换取更高的性能。

在 OnceStAXParser 中，最多只有 1 层继承，避免了层数过多的继承关系；对类进行了适度封装，减少 get、set 方法的使用，减少方法调用的次数。同时，尽量避免使用多态，这样可以减少系统运行时的开销。

5 性能测试

我们利用 Sun 提供的 XML Test[11]，对 BEA StAX RI、SUN SJSXP、Woodstox 和 OnceStAXParser 进行了性能测试。测试环境采用台式机，CPU 为 Pentium4 2.8G，支持超线程(hyper-threading)，内存为 400Hz 1G，支持双通道(dual-channel)。操作系统是 Windows XP with Service Pack 2，Java 的运行环境是 J2SE 1.4.2_03。解析器的属性被设置为不支持验证。图 7 是测试结果(其中 Transaction 的定义参考 11；TEST1 只解析测试文档的 25%，TEST2 只解析测试文档的 50%，TEST3 解析测试文档的全部)。表 3 给出了性能比较结果。测试结果表明，OnceStAXParser 的吞吐量比 Woodstox 平均高 4.6%，比 Sun SJSXP 平均高 5%，比 BEA StAX RI 平均高 38%。

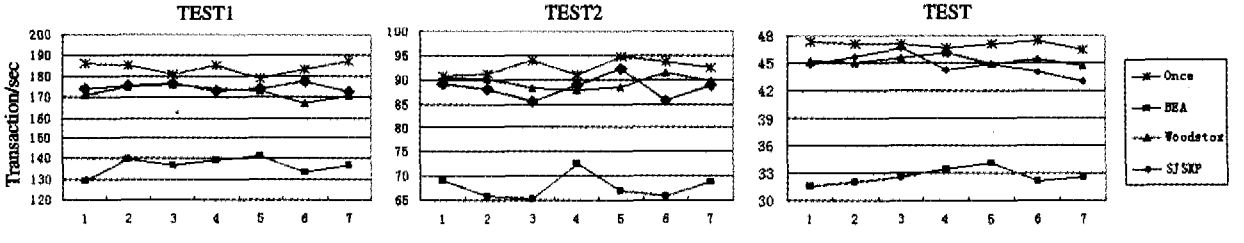

图 7 性能测试结果

(下转第 273 页)

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