高效 DOM 实现的技术研究*)

郭红艳　杨　波　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　DOM是目前为止唯一成为W3C正式标准的XML解析模型。本文充分考虑了DOM模型的特点,设计并实现了一个高性能的支持DOM的XML解析器:OnceDOMParser。为了提高DOM实现的性能,我们采用用户堆提高对象管理的效率,减少对象在JVM中创建的数量,并采取了数据的延迟装载策略。OnceDOMParser经过了严格的XML兼容性测试和DOM API兼容性测试,多方面的性能测试表明OnceDOMParser性能优越,其平均吞吐量比目前最流行的XML解析器Xerces高43.7%左右。

关键词　XML解析器,DOM,延迟加载

Research on High Performance DOM Implementation

GUO Hong-Yan　YANG Bo　JIN Bei-Hong

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

Abstract　DOM is the only XML processing model recommended by W3C. In this paper, we design and implement a high performance DOM compatible XML parser: OnceDOMParser, based on the thorough analysis of DOM model. For better efficiency, OnceDOMParser introduces user heap to manage the XML data, which reduces the creation of the JVM objects. Besides we adopt lazy load strategy for DOM data. OnceDOMParser has passed the rigorous XML conformance tests and DOM API tests. And various performance tests have proved its excellent performance. It gains about 43.7% higher throughput than that of Xerces2. 6. 2.

Keywords　XML parser, DOM, Lazy load

1　引言

XML(eXtensible Markup Language)[1,2]是一种广泛使用的标记语言,应用于Web数据传输、数据集成、文档存储等场合。DOM(Document Object Model)[3]是目前为止唯一成为W3C正式标准的XML解析模型。由于DOM有统一的规范,同时在工业界已有若干功能完善的实现者,用户可以在Java、C/C++、Perl、JavaScript等多种编程语言中使用DOM接口,因此,在XML文档处理中,DOM接口得以普遍应用。

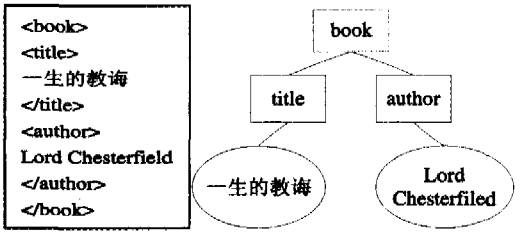

图1　DOM"语法树"示意图

DOM对XML文档的处理方式是把整个XML文档读到内存中,解析其内容,然后构造出一个特定的树状结构,该树包含了XML文档的各个语法单元以及它们之间的关系,相当于一个"语法树",如图1,然后DOM提供各种方法完成对这棵树的遍历、插入、删除、修剪等操作。DOM的优点是信息量丰富,XML文档内容全部都保留在内存中,而且可以随机访问,这在处理前后相互关联的元素时非常方便。

目前有两个广泛使用的DOM解析器实现:Crimson[4]和Xerces Java[5]。Crimson是Sun捐赠给Apache的,后被集成在JDK中。Crimson支持SAX和DOM,解析小XML文档比较快,而解析大的XML文档时性能不太理想,并且其实现不完全符合XML规范,且存在功能上缺陷,不能通过所有的兼容性测试。Xerces来自IBM捐赠给Apache的XML4J项目,已经从最初的1.0版本发展到现在的2.6.2版本,并且还在持续的开发和修改当中。Xerces的功能比较完整,支持最新版本的SAX和DOM,解析速度比较快,使用抽象接口来构建整个系统的设计,灵活性比较大,在工业界中得到了广泛应用。本文是以Xerces2.6.2 DOM解析器为性能比较对象。

我们设计和实现了支持DOM的XML解析器OnceDOMParser,本文给出了OnceDOMParser的设计和实现,详细介绍了高效DOM实现的具体技术,全文按如下方式组织:第2节描述了OnceDOMParser的体系结构,第3节给出了实现高效DOM的具体技术,包括用户堆和数据延迟加载(Lazy-Load)技术,第4节介绍了功能和性能测试方法,证明了第3节介绍的技术对性能的改善是非常有效的,最后是全文小节。

2　OnceDOMParser的体系结构

OnceDOMParser相当于对XML文档进行了一个转换,把一个XML文档从平面的文件格式转换为DOM模型中的文档(Document)对象,其主要功能可以分为语法解析、文档构建、文档访问三个部分[6]。

语法解析包括词法分析和语法分析,用于提取出XML文档中的各个语法单元。文档构建是指将XML文档中的语

*)本文工作受国家973项目(编号2002CB312005)、国家863项目(编号2001AA113010)的资助。**郭红艳**　硕士研究生,研究方向:分布式计算,软件工程。**杨　波**　硕士研究生,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。

法单元构造成特定的数据结构，即一棵语法树，该语法树实现了 DOM 规范中的 Document 接口，语法树中的各个节点实现了 DOM 规范中的相应接口，这样一棵语法树在 DOM 中被称为一个文档(Document)。DOM 文档在构建成功以后，包含了 XML 文档中的各种信息，例如各种标记信息以及字符数据等等，相当于是 XML 文档的一个有结构的复本，随后用户对 XML 数据的访问都发生在这个 DOM 文档中，包括 DOM 模型的读取、修改等操作。

OnceDOMParser 中有相应的组件来实现上述三个部分的功能：

- 语法解析器：负责分析 XML 文档的词法和语法，识别出语法单元；
- 文档构建器：负责构造 DOM 模型中的文档对象；
- DOM 节点模型：实现 DOM 规范中规定的各种数据结构。

我们重用了软件工程技术中心开发的 OnceStAXParser 中的 XMLStreamReader[7] 作为语法解析器。

StAX(The Streaming API for XML)[8] 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，有利于实现高效地解析 XML 文档。

XMLStreamReader 的工作过程是：分析 XML 文档的词法和语法，每当识别出一个语法单元的时候，就会处于一个事件状态之中，同时将这个语法单元的相关信息记录下来，以便报告给用户。例如，XMLStreamReader 解析到一个元素的开始标签时，处于 START_ELEMENT 状态，会记录下元素的名字、属性等相关信息，用户可以调用它的 getName()方法得到元素的名字，调用 getAttributeName(int index)和 getAttributeValue(int index)得到元素指定属性的名字和值。又例如，XMLStreamReader 解析到字符数据的时候处于 CHARACTERS 状态之中，用户可以调用它的 getText()方法得到字符数据的值。

由于 XMLStreamReader 以流方式解析 XML 文档，将 XML 文档中的各个语法单元依次报告给上层，因此 OnceDOMParser 可以直接使用 XMLStreamReader 来进行语法解析。

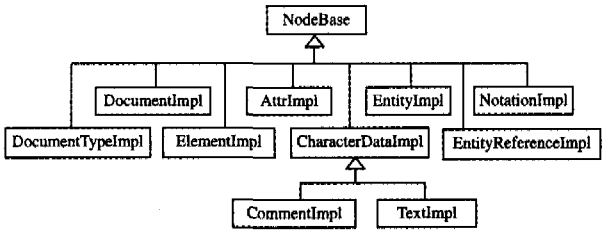

图 2　DOM 节点模型的类结构图

DOM 节点模型组件是 OnceDOMParser 的核心，实现了 DOM 规范中的各个接口，提供了 Element、Text 等各种节点类的实现，是 DOM 文档对象的主体部分。当 OnceDOMParser 解析完一个 XML 文档后，会在内存中建立一个 DOM 文档对象并保存下来，用户随后所有的访问都是发生在这个 DOM 文档对象中，包括节点的查找、添加、删除等等操作，因此，DOM 模型的实现对于 OnceDOMParser 的性能是至关重要的。分析 DOM 模型，可以提取出一个父类 NodeBase，然后各种节点类继承 NodeBase，分别实现 DOM 规范规定的接口，如图 2 所示。NodeBase 也以聚合的形式包含多个 NodeBase 对象，作为它自己的子节点。

类 DocumentBuilderImpl 是文档构建器组件中的关键类，它是 JAXP 中的 DocumentBuilder 抽象类的子类，实现了 JAXP 规范[9] 规定的接口。DocumentBuilderImpl 负责调用 XMLStreamReader 来进行语法解析，并将得到的语法单元转化成 DOM 节点模型中的文档对象保存。

3　用户堆和数据延迟加载技术

本节将介绍 DOM 节点模型和文档构建器组件的设计优化和实现，其中所采用的用户堆和数据延迟加载技术构成了高效 DOM 实现的核心技术。

3.1　用户堆

由上节可以看出，所有的 XML 文件信息都对应一个节点类，为了构建一棵 DOM 树，我们要生成大量的节点对象，在 JVM(Java Virtual Machine)中，每个对象除了本身的数据以外，还需要一些额外的管理空间来保存与 Java 对象相关的信息[10]，例如类型信息、虚函数表等，并且 JVM 在对象的创建和回收上都要耗费不少时间和资源，这些额外开销对于 Java 程序的性能有着不可忽略的影响。

为了尽量减少 Java 对象在 JVM 中的创建，我们在 DOM 节点模型组件中采取了“用户堆”的方式，并在使用节点对象的时候才创建节点对象，这样可以减少不必要的对象创建，提高程序性能。为了快速存取，“用户堆”使用多个数组来分别保存节点的各项信息。对于不同类型的节点，用户堆要保存不同的节点信息，例如对于一个 Element 节点，要保存节点名称(对于不支持名字空间的 DOM 解析器，只有节点名 name，对于支持名字空间的 DOM 解析器，节点名称包含 prefix、localname、uri)，父节点，兄弟节点，属性列表(包括属性名字空间声明和一般属性声明)，子节点列表，等等。而对于 Attr 节点，它没有属性列表，并且没有父节点和兄弟节点等，但它有节点名称和节点值、以及所属的元素节点。节点信息的多样性增加了用户堆设计的难度。为了节省内存开销，我们使用了最少的数组来压缩存储节点信息，例如，使用一个数组保存节点的前一个兄弟节点的索引，同时当节点类型为 Attr(没有兄弟节点)时，这个数组则保存它所属的元素节点索引。另外，由于 STag 中元素的属性总是紧跟着元素被解析，也就是被存进用户堆，因此，用户堆中紧随该 Element 存储的若干节点就是该元素的属性节点，所以我们用数组 fAttrCount 保存元素的属性个数，这样就方便查找所有属性，同时当节点类型为 Attr 时，该数组用来保存它是否是名字空间声明。

用户堆使用了二维数组，避免了过长的一维数组导致性能下降。在选择数组的初始长度时，一方面数组太小，需要频繁扩充数组，增加了数组复制的 I/O 操作；另一方面，数组太大会增加 JVM 的维护开销。经过了一系列测试，我们发现比较合适的一维数组的初始长度在 256 到 2048 之间，二维初始长度在 8 到 32。最后的实现选用了 512、32 分别作为一维和二维的初始长度。

在 Once DOM Parser 中 DOM 文档对象类的实现为 DocuementImpl，当用户访问某个 DOM 节点对象时，DOM 文档需要从用户堆中取出节点数据创建节点对象返回给用户(节点数据使用了延迟加载技术，下节会给出详细说明)，所以为了方便文档对象获取用户堆的信息，用户堆的核心数据结构以及对用户堆的管理用 DocuementImpl 的子类 LazyDocumentImpl 实现。

使用了用户堆之后，文档构建器 DocumentBuilderImpl 解

析 XML 文档的时候，就不需要在 JVM 的系统堆中生成大量的节点对象了，只需要将各个节点及其相关信息存放在 LazyDocumentImpl 中，当解析完成后，返回给用户的 DOM 文档对象实际上是 LazyDocumentImpl。

DocumentBuilderImpl 工作的时候，首先创建一个 XMLStreamReader 和一个空的 DOM 文档对象，接着调用 XMLStreamReader 的 next 方法，解析至下一个事件，然后根据事件类型调用 LazyDocumentImpl 的相应的 createXXX() 方法存储节点信息，在 LazyDocumentImpl 中再调用 XMLStreamReader 的 get 方法（例如 getName、getText 等）得到当前事件的 XML 信息存放到 DOM 文档对象的用户堆中；处理完一个事件以后，接着调用 XMLStreamReader 的 next 方法进入下一个事件的处理，直至 XML 文档结束，图 3 给出了 DocumentBuilderImpl 的实现序列图。

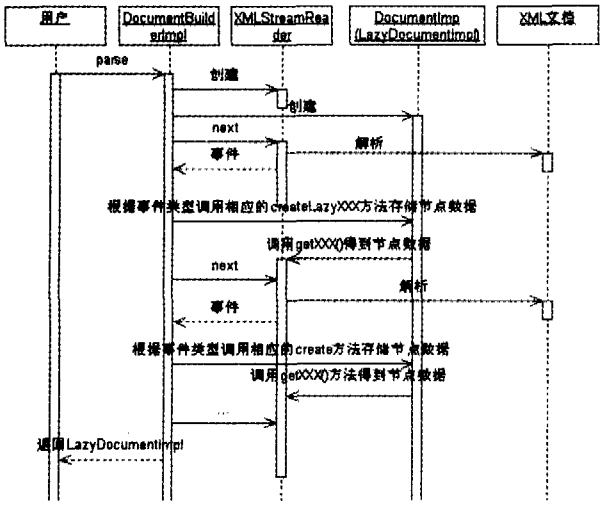

图 3　DocumentBuilderImpl 的工作过程

3.2　数据的延迟加载

DOM 的节点对象包含众多信息，如 ElementImpl 节点就包含 localname、prefix、uri、attrbutes、childNodeList 等信息，而用户对节点的所有信息进行访问的概率并不大，所以，我们采用了延迟装载的方式减少了不必要的信息复制，当用户访问节点信息时才从“用户堆”中复制出节点信息。具体地说，就是在用户访问 DOM 文档中节点的时候，虽然创建了相应的节点对象返回给用户，但这个节点对象并不从用户堆中复制出节点信息，它只保存了一个指向用户堆的索引 fNodeIndex，当用户访问到节点信息时才从用户堆中加载。我们将节点信息分为三类：基本信息（包括节点名称，值等）；子节点列表；元素属性节点集合。我们分别给这三种信息设置一个 loaded 标志：m_valueLoaded、m_childLoaded 和 m_attributeLoaded，用来表示该类数据是否已从用户堆中取出。当用户访问某一项数据时，才将该类数据从用户堆中取出，并将相应的 loaded 标志设置为 true。当用户再次访问到该类数据时，将直接从节点类中取出数据返回，而不是从用户堆中装载数据。图 4 给出了多次访问节点的子节点和基本信息的实现序列图。

除了对节点信息进行延迟装载外，延迟装载策略同样可以应用到查询遍历上。DOM 规范规定的 Element 接口有一个搜索方法 getElementsByTagName(String name)，该方法在

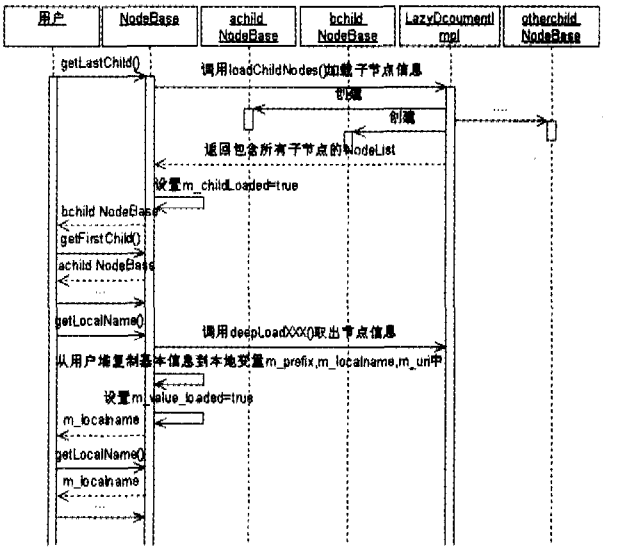

图 4　访问延迟加载的节点的过程

它的子节点树中查找所有名字为 name 的元素，然后将这些元素存放在一个 NodeList 中返回。在一个节点树中执行查找，需要遍历树中的每一个节点，耗时是非常大的。我们采取了延迟装载的方式执行对节点树的遍历，即用户首次调用 getElementsByTagName 方法的时候并不实际执行查找，而是返回一个特殊的 NodeList，记录下查找的开始点和查找的参数，等随后用户访问这个 NodeList 的时候再执行遍历。特别地，当用户访问 NodeList 的第 i 个节点的时候，只在节点树中查找到第 i 个匹配节点即可返回，不需要再进一步遍历。考虑到用户调用 getElementsByTagName 方法之后，一般都只会使用结果集的第一个节点，这种延迟策略将会大大减少不必要的遍历操作，可以大幅度提高用户访问 DOM 模型的速度。

4　功能和性能测试

OnceDOMParser 经过了严格的 XML 兼容性测试和 DOM API 兼容性测试。我们采用了 W3C 提供的 XML 兼容性测试软件包 XMLConf 软件包[11]进行 XML 兼容性测试，采用 DOM Test Suites (DOM TS)[12]进行了 DOM API 兼容性测试。OnceDOMParser 通过了上述总共 3888 个测试。

同时，我们进行了有关的实验，用以验证和评价用户堆和数据延迟加载技术。我们还利用 Sun 提供的测试 XML 处理性能的软件包 XML Test1.1[13]，对 OnceDOMParser 与目前最流行的 XML 解析器 Xerces 进行全面的性能测试。

4.1　节点的延迟生成和延迟加载

首先对节点延迟生成和延迟加载优化之前和优化之后的 OnceDOMParser 进行测试，优化之前的 OnceDOMParser 会将解析文档的过程中生成所有节点对象保存在 Document 中，优化之后的 OnceDOMParser 则在解析文档的过程中将节点信息保存在 Document 的用户堆中，直到用户访问节点时才生成所需节点。

测试程序解析一个约 100k 的 XML 文档，然后，首先利用 getDocumentElement() 得到文档根元素，接着，用 getLastChild()、getLocalName() 得到根元素的最后一个子节点以及该节点的名字。整个测试运行了 100 次。

图 5 给出了优化前后 OnceDOMParser 所花的时间。从图中看出，优化后所花费的时间仅占优化前的 60%，因此，节点的延迟生成和延迟装载可以有效地提高程序的性能。

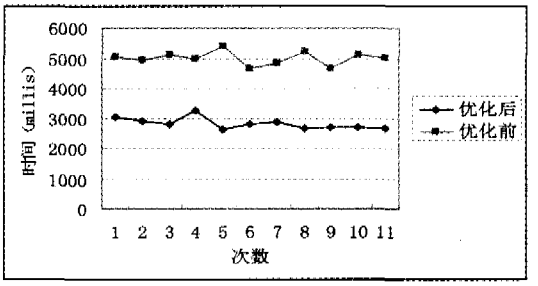

图5 关于节点延迟生成和延迟加载的测试结果

效地提高程序的性能。

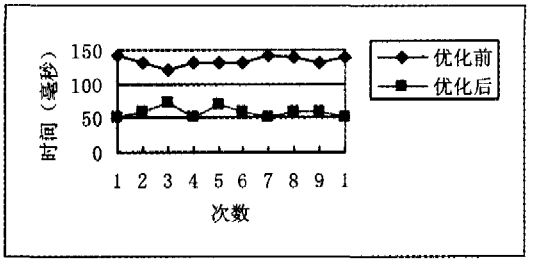

图6 BenchGetElements的结果

4.2 查询结果集合的延迟装载

针对查询结果集合优化前后的OnceDOMParser进行了测试。优化前，getElementsByTagName执行的时候一次性地在元素的子节点树中查找所有的元素；优化后，getElementsByTagName执行的时候会生成一个SearchResultNodeList返回给用户，然后用户访问SearchResultNodeList的时候再根据需要从节点树中查找出用户访问的节点。

测试程序解析一个约100k的XML文档，然后，通过调用Element的getElementsByTagName来获取指定的Target元素。执行getElementsByTagName 100次。两种情况所花费的时间如图6。从图中看出，优化后所花费的时间仅为优化前的一半左右，因此，查询结果集合的延迟装载同样可以有

4.3 系统吞吐量测试

系统吞吐量测试采用了Sun公司提供的XML Test1.1。系统吞吐量，在Sun XML Test中，是指每分钟执行XML事务的平均数，这里一个XML事务包括解析、访问、修改(包括内容修改和结构修改)、序列化四步。Sun XML Test1.1中给出了六个测试场景，test1、test2、test3分别是访问文档的25%、50%、100%内容，test4测试内容修改，test5测试结构修改，test6综合了内容和结构修改。当然六个测试都包括对文档的解析，并且后三个包含了对修改后的文档的序列化。表1是SUN XML Test1.1的测试结果。可以看出OnceDOMParsr平均比Xerces2.6.2快43.7%以上。

表1 性能测试结果比较

测试场景	平均性能提升	Once1st	Once2nd	Once3rd	Xerces1st	Xerces2nd	Xerces3rd
Test1	43%	276.52	271.08	272.87	192.95	188.65	191.35
Test2	45%	267.54	267.67	266.48	187.92	179.31	182.59
Test3	52%	257.26	258.27	262.866	173.10	168.42	170.16
Test4	38%	140.17	140.31	142.33	104.80	101.91	99.37
Test5	43%	142.16	140.01	141.08	98.38	100.17	97.24
Test6	40%	139.54	138.80	140.61	99.19	99.57	98.71

小结 本文在对DOM模型进行研究的基础上，构造了Java运行环境下的XML解析器OnceDOMParser。对其进行的性能测试结果表明OnceDOMParser性能卓越，解析效率要高于目前最流行的XML parser Xerces 43.7%。

下一步工作包括以下几方面：1)增加有效性验证检查；2)扩充OnceDOMParser功能，实现对HTML的支持等；3)针对Java语言本身对系统进行进一步的性能优化。

参考文献

1 W3C. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210, 1998
2 W3C. Extensible Markup Language (XML) 1.1. http://www.w3.org/TR/2004/REC-xml11-20040204, 2004
3 W3C. Document Object Model. http://www.w3.org/DOM/
4 The Apache XML Project: Crimson. http://xml.apache.org/crimson/
5 The Apache XML Project: Xerces Java 2. http://xml.apache.org/xerces2-j/
6 OnceDOMParser详细设计报告. 软件工程技术中心技术文档, 2005
7 OnceStAXParser详细设计报告. 软件工程技术中心技术文档, 2005
8 Java Community Process. JSR 173: Streaming API for XML. http://jcp.org/en/jsr/detail?id=173
9 Java API for XML Processing. http://java.sun.com/xml/jaxp/index.jsp
10 Venners B. Inside the Java Virtual Machine. (Second Edition). 2003
11 W3C. Extensible Markup Language (XML) Conformance Test Suites 20031210. http://www.w3.org/XML/Test/
12 W3C. DOM TS(DOM Conformance Test Suites). http://www.w3.org/DOM/Test/
13 SUN. XMLTest1, 1. http://java.sun.com/performance/reference/codesamples/

(上接第266页)

3 Ncube C, Dean J C. The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Products. Proceedings of ICCBSS, Orlando, Florida USA, 2002. 176~187
4 Ruhe G. Intelligent Support for Selection of COTS Products. In: Proc. of the Net. ObjectDays 2002, Erfurt, Springer 2003
5 岳超源. 决策理论与方法. 科学出版社, 2003. 170~246
6 Alves C, Castro J. CRE: A Systematic Method for COTS Components Selection. XV Brazilian Symposium on Software Engineering (SBES) Rio de Janeiro, Brazil, October 2001
7 Sheng Jinfang, Chen Songqiao, Wang Bin. COTS Evaluation and Selection Based on Requirements Decomposition, Chinese Journal of Electronics, 2005(1): 62~67
8 Fourer R, Gay D M, Kernighan B W. AMPL: A Modeling Language for Mathematical Programming, Duxbury Press, 2002
9 MINLP World, http://www.gamsworld.org/minlp/index.htm, 2005