ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.10, October 2008, pp.2728−2738
DOI: 10.3724/SP.J.1001.2008.02728

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

高性能的 XML 解析器 OnceXMLParser*

金蓓弘[1+], 曹冬磊[1,2], 任 鑫[1,2], 余 双[1,2], 戴蓓洁[1,2]

[1](中国科学院 软件研究所,北京 100190)

[2](中国科学院 研究生院,北京 100049)

A High Performance XML Parser OnceXMLParser

JIN Bei-Hong[1+], CAO Dong-Lei[1,2], REN Xin[1,2], YU Shuang[1,2], DAI Bei-Jie[1,2]

[1](Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: Beihong@iscas.ac.cn

JIN BH, Cao DL, Ren X, Yu S, Dai BJ. A high performance XML parser OnceXMLParser. *Journal of Software*, 2008,19(10):2728−2738. http://www.jos.org.cn/1000-9825/19/2728.htm

Abstract: An XML (extensible markup language) parser is the fundamental software for analyzing and processing XML documents. The implementation of a high performance full-validating XML parser is studied in this paper. This research develops a OnceXMLParser which supports three kinds of parsing models. It passes the rigorous XML conformance testing and API (application programming interface) conformance testing. OnceXMLParser adopts light-weighted architecture and is optimized on many aspects including efficient lexical analysis, statistical automaton implementation, reasonable resource allocation strategies and some fine tunings on language level. Performance testing results show that OnceXMLParser has outstanding parsing efficiency.

Key words: extensible markup language; parser; performance tuning

摘 要: XML(extensible markup language)解析器是分析、处理 XML 文档的基础软件.研究高性能验证型 XML 解析器的实现.开发了支持 3 种解析模型的 XML 解析器 OnceXMLParser,该解析器通过了严格的 XML 兼容性测试和 API 兼容性测试.OnceXMLParser 具有轻量级体系结构并进行了多方面的性能优化,包括高效的词法分析、基于统计分析的自动机实现、合理的资源分配策略以及语言层次上的优化.性能测试结果表明,OnceXMLParser 具有出色的解析性能.

关键词: XML(extensible markup language);解析器;性能优化

中图法分类号: TP314 **文献标识码**: A

XML[1]是 W3C 于 1998 年推出的一种可扩展标记语言,2004 年升级到 1.1 版[2].短短几年,XML 文档已被广泛用于电子商务、B2B 通信、企业信息交换/集成、Web 服务等应用中,成为网络环境下进行数据交换的基本方式之一.同时,XML 作为一种元语言,可用于描述一大类层次化标记语言.例如,Web 服务描述语言 WSDL(Web

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2004AA112010 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2002CB312005 (国家重点基础研究发展计划(973))

Received 2007-07-30; Accepted 2008-02-25

services description language)、资源描述框架 RDF(resource description framework)、数学标记语言 MathML 等.这些语言的应用直接依赖于对 XML 文档的解析.XML 解析器是分析、处理 XML 文档的基础软件.

一方面,由于 XML 文档结构严谨,并采用文本方式存储,所以在一些应用中,XML 文档会比较冗长;另一方面,在另一些应用,例如 Web 服务引擎中,基于 XML 文档的数据交换非常频繁.在这些应用中,XML 文档的解析会成为整个系统性能的一个瓶颈.所以,有必要构造高效的 XML 解析器,以提高处理 XML 文档的性能.

本文研究高性能验证型 XML 解析器,并构造了 Java XML 解析器 OnceXMLParser.第 1 节介绍 XML 及其解析模型.第 2 节叙述相关工作.第 3 节给出 OnceXMLParser 实现中的关键策略.第 4 节总结在 OnceXMLParser 中采用的优化策略.第 5 节给出 OnceXMLParser 与其他 XML 解析器的性能比较.最后是全文总结.

1 XML 及其解析模型

一个 XML 文档,从物理上看,是由一个或多个实体组成;从逻辑上看,是包含了一个或多个元素(参见文献[1]中产生式第[39]条);从字符流来看,是由字符数据和标记组成,其中标记包括元素标签、实体引用、字符引用、文档类型声明、处理指令(processing instruction,简称 PI)等 12 种.

XML 规范采用了 Extended Backus Naur Form(EBNF)描述语言的产生式,并定义了有关的良构性约束(well-formedness constraint,简称 WFC)和有效性约束(validity constraint,简称 VC).符合这些产生式并满足良构性约束的 XML 文档被称为良构的 XML 文档,而有效的 XML 文档是指一个满足有效性约束的良构的 XML 文档.验证型解析器是指同时检查 XML 文档的 WFC 和 VC 的 XML 解析器,仅检查 XML 文档 WFC 的 XML 解析器称为非验证型解析器.

为了测试 XML 处理软件与 XML 规范的兼容性,W3C 提供了 XML 兼容性测试软件包 XMLConf 软件包[3].XMLConf 中包含了来自多个组织、公司或个人(例如 Oasis,IBM,SUN,James Clark 等)的两千多个测试用例,为 XML 解析器的开发者提供了相对完备的 XML 兼容性测试依据.

到目前为止,对 XML 文档的解析或者采用基于树型结构的解析模型或者采用基于流的解析模型.Document Object Model(DOM)[4]属于典型的基于树型结构的模型,它也是目前唯一成为 W3C 正式标准的 XML 解析模型.利用 DOM 接口可以对整个 XML 文档进行随机动态的访问和修改,但由于它必须在内存中加载整个文档并构造完整的树结构,所以,内存开销比较大.基于流的解析模型是把 XML 文档看成输入/输出流,该类模型又可根据对事件的处理方式分成“推”和“拉”两种.Simple API for XML(SAX)[5]属于基于流的“推”模型,即,要求事件的消费者事先实现 SAX 所定义的事件处理过程并注册,然后,XML 解析器才会在解析 XML 文档时去处理这些事件.虽然 SAX 和 DOM 都必须一次解析完整个 XML 文档,但由于 SAX 无需把整个文档一次加载到内存中,所以,SAX 对内存要求较低,适合于处理大型的 XML 文档.但由于 SAX 不能提供元信息,如事件状态、元素父子关系等,所以,SAX 对复杂应用逻辑有点力不从心,不过,对于单遍读取应用,SAX 的处理效率相当高.2003 年,Streaming API for XML(StAX)[6]正式成为了 JCP 规范,它属于基于流的“拉”模型,即应用程序能够控制解析的过程,并按应用的要求有选择地返回事件以便进行处理,也可以随时停止处理.StAX 既提供底层的、基于指针的 API,也提供高级的、基于事件的 API,这给分析和处理 XML 文档带来极大的灵活性.SAX,DOM,StAX 接口都已形成了相应的接口兼容性测试包.例如,用于测试 XML 解析器是否符合 SAX2 规范、StAX 规范、DOM 规范的软件包分别有 sourceforge 提供的 SAX2Unit[7],Tatu Saloranta 提供的 StaxTest[8],W3C 的 DOM TS[9].

SUN 提供的 XML Test 1.1[10]可以用于测试 XML 处理性能.该测试包设计了一个独立的多线程程序,用来模拟能够并行处理多个 XML 文档的多线程服务器工作环境,它抽取了最常见的 XML 文档处理流程,通过考察 XML 文档的解析、访问、修改、序列化等操作,来度量系统处理 XML 文档的吞吐量.它可以测试 Java 环境下采用 DOM 和 SAX 接口解析 XML 文档的性能,还提供了针对.Net 环境的 PullParser 的测试程序,我们经过修改,使得它能够测试 Java 环境下采用 StAX 接口解析 XML 文档的性能.

2 相关工作

目前,已有各种各样的 XML 解析器支持不同的编程语言,包括 Java,C/C++,Perl,JavaScript 等,但它们都只支持一种或两种解析模型.

在 Java XML 解析器中,Apache 软件基金会提供的 Xerces[11]是目前使用比较广泛的 XML 解析器,也是当前 JDK 缺省的 XML 解析器.它支持 W3C 的 XML1.0/1.1 标准,支持 DOM 以及 SAX2,并具有验证功能.Crimson1.1[12]是较早版本 JDK 缺省的 XML 解析器,支持 XML1.0 规范、SAX2、DOM Level 2 核心规范和 JAXP1.1.

Crimson 的功能并不完善,例如,它不能识别 UTF-8 编码的字节序标记(byte order mark),不能识别元素的属性值中出现的非法的字符引用,它只能通过 XML 兼容性测试中的部分测试用例.虽然 Xerces 在某种极端情况下,例如读入 10M 左右大小的注释或 PI 时,会出现死循环现象,但相对其他 SAX 和 DOM 解析器,Xerces 的功能还是比较完善的.所以,我们把 Xerces2.9.0 作为我们 SAX 和 DOM 解析器的性能对比参照软件.

目前支持 StAX 接口的 XML 解析器包括 BEA 的 StAX 参考实现(RI)[13],Sun Java Streaming XML Parser (SJSXP)[14],Woodstox[15],Oracle 的 StAX Pull Parser[16],Codehaus 的 StAX 参考实现(RI)[17]等.

经过我们的测试,BEA StAX RI 在实现方面存在较多漏洞,例如,它在解析字符数据、字符引用和实体引用方面存在不少缺陷,不能识别出“]]>”为非法的字符数据,不能正确解析缺省属性值中的实体引用,如果在字符数据中存在字符引用,那么 BEA RI 在所报告的字符事件中不能获得正确的文本内容.Codehaus 的 StAX RI 不支持非名字空间模式,不支持对外部实体的解析,与 XML 规范的不一致处较多.SJSXP 在功能上也存在缺陷,例如,它不能正确解析良构文档类型定义(document type definition,简称 DTD)中的外部参数实体,不能处理符合 XML 规范的[#x10000,#x10FFFF]段的合法字符,也不能解析代理对(surrogate pair),不解析外部 DTD 中的缺省属性声明等.而 Oracle StAX Pull Parser 是基于 Oracle XDK 中的 SAX Parser 的,解析效率不高.Woodstox 的功能和性能均强于上述解析器,但它在词法分析方面也存在重大缺陷,例如,它不能全面支持 UTF-8 解码,特别是对代理对的处理不够完备,对基本字符、名字的识别上存在漏洞.Woodstox 是上述几种解析器中唯一一个实现了有效性验证功能的解析器,但在其验证实现上同样存在重大缺陷.它对于不符合 VC 的良构文档,并没有按照 XML 规范要求返回一个错误,而是会抛出一个异常,并终止解析过程.由于 Woodstox 的功能比较完备且性能测试结果最好,我们最终选择了 Woodstox3.2.0 作为 StAX 解析器的性能对比参照软件.

自动生成是构造 XML 解析器的另一种途径,XML 文法可以看成是一种扩展的上下文无关文法(extended context free grammar,简称 ECFG),已有一些研究讨论并提出了为 ECFG 自动生成解析器所需的算法[18,19],在系统实现上,Piccolo[20]是利用解析器生成工具 JFlex 和 BYACC/J 生成的一个验证型 SAX 解析器,仅支持 SAX 接口,包括 SAX1,SAX2.0.1.Piccolo 的功能并不完善,例如,它不支持 XML1.1 规范,不能正确处理实体引用中的相对路径.它的有效性验证功能也不完善,例如,对于 VC(proper conditional section/pe nesting)等规则都没有进行验证.同时,它的解析性能也远不如手工开发的 XML 解析器,如 Xerces.

文献[21]针对 Web 服务环境设计了一个自适应的 XML 解析器 Deltaser,它通过复用已有的 XML 文档片段的解析结果来提高解析速度,该技术并不适用于通用 XML 解析器的实现.文献[22]描述了一个针对嵌入式设备解析小型 XML 文档需求而设计的 XML 解析器 BNFParser.我们注意到,文献[21,22]均没有给出相应解析器与 XML 规范兼容性以及与 API 规范兼容性的说明.

与已有的 XML 解析器相比,我们开发的 OnceXMLParser 具有以下特点:

- 支持 XML1.0/1.1 和 XML 的名字空间[23].
- 同时支持 3 种解析模型,即提供 SAX2,DOM Level 2 Core,StAX 接口支持.
- 同时提供验证版、标准版和轻量版,其中,验证版提供验证型解析器功能,标准版提供非验证型解析器功能,轻量版仅用于解析良构的 XML 文档,即,用于被解析 XML 文档已知是良构的情况.
- 通过了严格的 XML 兼容性测试和 API 兼容性测试.
- 解析性能出色.

3 系统实现

与处理常规语言一样,XML 解析器需要完成词法分析和语法分析,然后,利用语法制导执行相关的语义动作,实现 DOM,SAX,StAX 接口功能.

按照 XML 规范,产生式中以大写字母开头的符号都是正则语言的开始,所以,一方面,我们可以构造确定的有限状态自动机(deterministic finite automaton,简称 DFA)来识别它们,另一方面,我们把这些符号也作为 XML 文法终结符的一部分.然后,进行文法的化简,包括进行产生式的替换、消除左递归、去掉单一产生式组,最后,构造下推自动机[24],自顶向下地对 XML 文档进行语法分析和处理.

我们构造的下推自动机 $M=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$,其中:状态集合 $Q=\{q0,q1,q2,q3,q4\}$,$q0$ 是起始状态,$q1$ 是解析文档声明 XMLDecl/DTD 之后的状态,$q2$ 是解析完根元素开始标记、开始解析其内容的状态,$q3$ 为根元素解析完毕、开始解析 Misc 直到文档结束的状态,$q4$ 是解析结束状态.输入符号集合 Σ={XMLDecl,Misc,doctypedecl,EmptyElemTag,*S*Tag,Content_item*,*E*Tag,$}(注:'$'表示文档结束).Z_0 是栈的开始符号,栈符号表 Γ={*S*Tag,DTD,Z_0}(*S* 表示栈顶符号),初始状态 $q_0=q0$,终结状态 $F=\{q4\}$,而状态转移函数 δ,$\delta: Q\times\Sigma\cup\{\varepsilon\}\times\Gamma\rightarrow 2^{Q\times\Gamma^*}$,是下列转移的集合:

1. ($q0$,XMLDecl,Z_0)={($q1$,Z_0)},表示解析到文档声明 XMLDecl 后进行的状态转移;
2. ($q0$,doctypedecl,Z_0)={($q1$,DTDZ_0)},表示解析到 DTD 后,将 DTD 入栈,并进行状态转移;
3. ($q0$,Misc,Z_0)={($q1$,Z_0)},表示解析到 Misc 后进行的状态转移;
4. ($q0$,*S*Tag,Z_0)={($q2$,*S*TagZ_0)},表示解析到根元素的开始标记 *S*Tag 后,将 *S*Tag 入栈,并进行状态转移;
5. ($q0$,EmptyElemTag,Z_0)={($q3$,Z_0)},表示解析到根元素为空元素时进行的状态转移;
6. ($q1$,Misc,Z_0)={($q1$,Z_0)},表示未解析到 DTD 情况下,解析到 Misc 后进行的状态转移;
7. ($q1$,Misc,DTD)={($q1$,DTD)},表示解析完 DTD 情况下,解析到 Misc 后进行的状态转移;
8. ($q1$,doctypedecl,Z_0)={($q1$,DTDZ_0)},表示解析到 DTD 后,将 DTD 入栈,并进行状态转移;
9. ($q1$,*S*Tag,Z_0)={($q2$,*S*TagZ_0)},表示无 DTD 情况下,解析到根元素的开始标记 *S*Tag 后,将 *S*Tag 入栈,并进行状态转移;
10. ($q1$,*S*Tag,DTD)={($q2$,*S*Tag)},表示存在 DTD 情况下,解析到根元素的开始标记 *S*Tag 后,将 DTD 出栈,将 *S*Tag 入栈,并进行状态转移;
11. ($q1$,EmptyElemTag,Z_0)={($q3$,Z_0)},表示无 DTD 情况下,解析到根元素为空元素时进行的状态转移;
12. ($q1$,EmptyElemTag,DTD)={($q3$,ε)},表示存在 DTD 情况下,解析到根元素为空元素时进行的状态转移;
13. ($q2$,*S*Tag,*S*)={($q2$,*S*Tag*S*)},表示将解析到的子元素开始标记 *S*Tag 入栈;
14. ($q2$,Content_item,*S*)={($q2$,*S*)},表示解析到无需入栈的元素内容;
15. ($q2$,*E*Tag,*S*Tag)={($q2$,ε)},表示解析到元素的结束标记 *E*Tag 时,将 *S*Tag 出栈;
16. ($q2$,ε,Z_0)={($q3$,Z_0)},表示栈内只有 Z_0 时进行的状态转移;
17. ($q3$,Misc,Z_0)={($q3$,Z_0)},表示循环解析 Misc;
18. ($q3$,$,$Z_0$)={($q4$,$\varepsilon$)}表示检测到文档结束,*M* 停机.

相应的状态转移图如图 1 所示.该下推自动机实现了 XML 语法分析的基本的主解析流程.

考虑到在识别出产生式后需要根据接口的要求执行语义动作,所以,一种可行的方法是提取 DOM,SAX,StAX 的事件接口,形成处理所有可能出现的事件的统一接口,解析器实现这些接口并注册到语法分析器,然后由这个公共的语法分析器分析 XML 文档,调用上述接口中的相应函数用于处理事件或构造 DOM 树.Xerces 采用了类似的思想定义了一套内部的接口 Xerces Native Interface (XNI),Xerces 的语法解析器解析 XML 文档产生

---

* 符号 Content_item 定义为 Content_item::=EmptyElemTag|Reference|CDSect|P I|Comment|CharData.

的结果被封装成符合 XNI 定义的事件.然后,将 XNI 提供的事件转化为符合 SAX 接口的事件或建立一棵符合 DOM 接口规范的 DOM 树.虽然这种设计结构灵活,代码重用性高,伸缩性强,但是,复杂的实现使得运行效率和解析性能在一定程度上有所降低.仔细分析不同解析模型所执行的语义动作,可以看到不同模型下所执行的语义动作基本不相同.SAX 解析器是在识别出有关记号(token)后直接调用事先注册的回调函数.StAX 解析器能够识别所有事件,可供用户选择其感兴趣的事件,因此,它需要维护内部的状态,以便在用户请求时获得有关信息.而 DOM 解析器利用 StAX 解析器提供的功能可以比较自然地构造 DOM 树.所以,我们基于上述基本的解析流程,分别为 SAX,StAX 建立了主解析流程,DOM 采用与 StAX 相同的主解析流程,主解析流程之外的语法分析组件,包括 DTD 解析组件、实体信息处理组件、名字空间声明管理组件、有效性验证组件等,则为共用.轻量级体系结构构成了 OnceXMLParser 实现高效解析的基础.

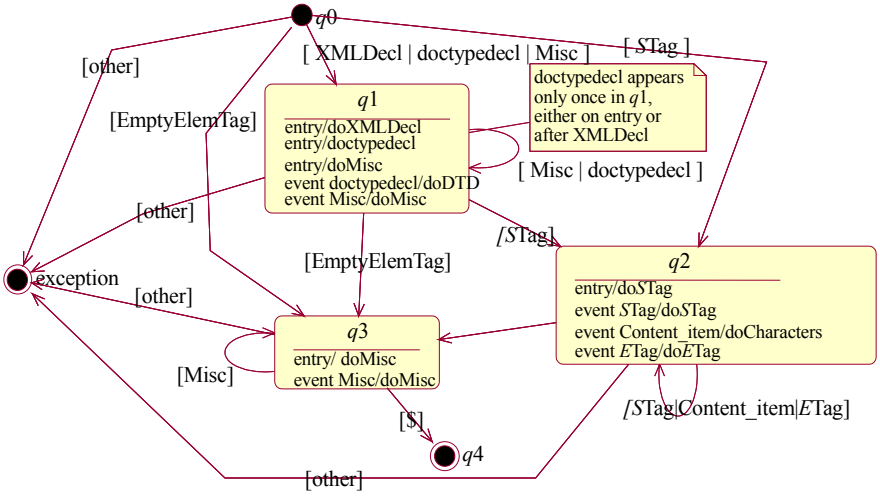


Fig.1 The main parsing process

图 1 基本的主解析流程

语法解析过程中还要处理 XML 规范定义的 WFC 以及名字空间规范定义的 Namespace Constraint(NSC).例如,Element Type Match(WFC)表示 *S*tag,*E*Tag 中元素类型必须匹配,我们通过一个元素栈即可完成该项检查;PEs in Internal Subset(WFC)是说明内部 DTD 的参数实体引用只能出现在标记可以出现的地方,而不能出现在标记内部,我们在解析产生式第[29]条的时候进行相应的检查.这条规则还说明,上述规则不能应用到外部参数实体或外部子集中.这意味着,在外部子集和外部参数实体中,参数实体引用可以出现在标记定义(参见文献[1]中产生式第[29]条)的内部,即可以在元素定义、属性列表定义、实体定义、记法(notation)定义、PI 的内部.由于规范没有明确说明参数实体引用可以具体出现的位置,所以,我们的实现中把产生式第[29]条中的非终结符或若干连续非终结符的连接(不计空格)都可以用参数实体引用来替换,增加了使用的灵活性.

在共用的解析器组件中,DTD 解析组件对 DTD 部分进行解析,并记录 DTD 定义的有效性约束条件.XML 定义了 5 大类由名字识别的实体,即内部一般/参数实体、外部一般/参数实体、未析实体以及两类特殊的实体即文档实体和外部 DTD 子集.在解析实体定义和引用过程中,实体信息处理组件负责记录实体的有关信息,并根据上下文及时设置和更换当前实体,使得实体引用能够被正确地替换.名字空间声明管理组件负责保存(名字空间前缀,URI 引用)对,提供相应的查询机制,管理名字空间作用域.有效性验证组件用于验证 XML 文档是否符合 DTD 中声明的有效性约束条件,并且负责给出错误信息.它的主要功能是实现元素验证和属性验证.元素验证部分要求 XML 文档中所有元素的子元素必须要符合 DTD 中的定义.考虑到 DTD 中对子元素的定义符合正则文法,我们构造了 NFA(non-deterministic finite automaton,非确定的有限自动机)[25]进行元素验证,即,DTD 解析组件会根据 DTD 中给出的子元素定义构造一个优化的 NFA,存储通过解析子元素定义获得的状态转移函数.

有效性验证组件在解析 XML 文档的过程中启动这个 NFA,如果子元素解析过程停机在 NFA 的结束状态,那么说明子元素满足有效性约束;否则,意味着元素验证失败,将报告错误信息.另一方面,由于 DTD 可以定义多种类型的属性,而关于属性验证的规则较多,所以属性验证需要的开销也比较大.因此,有效性验证组件提供了轻量级的集合处理类 OSet,以加速属性验证中频繁出现的各种唯一性检查操作,如对 ID 值、元素名、记法名等的唯一性检查.同时,还对验证流程进行了优化,例如,对元素属性类型的判断次序,根据统计规律进行了调整,以提高解析性能.

考虑到在实际应用中,常常有解析良构 XML 文档的需求,所以,我们对提供标准功能的 OnceXMLParser 进行裁减,并有针对性地进行了一些优化,形成了轻量版 OnceXMLParser.轻量版解析器省略了对 WFC/VC 的检查,跳过了对名字的合法性检查,也简化了对若干产生式的解析.例如,在对元素内容声明和元素内容的解析中,可以省略检查元素内容不能交织出现选择和顺序成分.我们针对基于流的“拉”解析模型的特点,在轻量版的 StAX 解析器中,对元素属性采取延迟解析的技术,即对于属性的解析延迟到用户调用 getAttributeValue()方法的时候,或者,如果用户不访问属性,那么,StAX 解析器就可以直接跳过属性部分,从而省略解析属性带来的开销.该方法并不适用于标准版及验证版的 StAX 解析器,因为不论用户是否访问属性,解析器也必须检查属性值中是否存在异常(例如存在非法字符),并报告异常.

OnceXMLParser 采用了第 2 节中提到的 XML 兼容性测试包和接口兼容性测试包进行功能测试.我们通过扩展 JUnit 框架,构造了 OnceXMLParser 的自动化测试工具,利用该工具完成并通过了上述所有测试.其中,验证版进行了 10 678 项兼容性测试,标准版进行了 9 459 项兼容性测试项.轻量版的功能测试是上述功能测试的子集,总共通过了 3 457 项测试.

4 优化策略

为了获得满意的解析性能,系统从多个方面进行了优化,除了上述的采用轻量级体系结构之外,系统还从下面 4 个维度进行了优化.

4.1 高效的词法处理

OnceXMLParser 的词法分析器 Scanner 完成了两部分工作:一是读取 XML 文档,按照文档的编码方式进行解码,并将解码后的字符数据送入预先分配的字符缓冲区,针对程序语言的词法分析器没有这部分工作;二是构造确定的有限自动机,读取终结符,送入语法分析器.这是 OnceXMLParser 逐一读取 XML 文档字符的唯一阶段,消耗掉相当可观的时间.

XML 规范要求至少支持 UTF-8 和 UTF-16 两种编码方式,经过我们的测试,Java 语言中提供的 UTF-8 解码器的效率无法满足高性能解析器的要求,所以,我们独立实现了一个专门用于 UTF-8 编码方式的解码器 UTF8Reader,用来提高 XML 解析器对原始 XML 文档数据的读取速度.表 1 给出了使用 UTF8Reader 和使用 Java 提供的 UTF-8 解码器读取 3 个大小分别为 1M,2M 和 10M 的 XML 文档各 1 000 次所花费的时间(单位:ms),从中可以看到,使用 UTF8Reader 可以提高读取性能.

Table 1 Performance testing results with/without UTF8Reader

表 1 使用 UTF8Reader 前后的性能比较

Size of documents (M)	UTF8Reader	UTF-8 decoder (provided by Java)	Improvement (%)
1	41 502	53 046	27.8
2	83 351	105 273	26.3
10	412 024	525 203	27.5

Scanner 中设计了两个缓冲区,一个扫描缓冲区和一个字符缓冲区,扫描缓冲区用于解码扫描,该缓冲区一般取页面大小的整数倍.字符缓冲区用于存放解码结果.该字符缓冲太小会导致更多次数的填充字符缓冲调用以及保存位于缓冲区结尾的不完整的标记.而过大的字符缓冲,会增加在 Scanner 中进行规范化(normalization)操作的开销.为此,我们测试了不同缓冲区大小下 Scanner 读取 XML 文档中记号的时间,图 2 给出了其中一组数

据,其中,*X* 轴表示扫描缓冲大小,*Y* 轴表示以毫秒计的读取时间.图上的每条线表示在相同字符缓冲下的测试结果.测试结果显示,字符缓冲大小为 4K 和 8K 时 Scanner 性能明显优于其他缓冲区设置,进一步分析发现,Scanner 在 8K 字符缓冲设置情况下,读取记号所需时间的均方差仅是 4K 字符缓冲的 68.8%,显示出比较稳定的性能.考虑到合理利用内存空间和避免额外的 I/O 操作,我们最终选取了(8K,8K)为扫描缓冲区和字符缓冲区的大小.

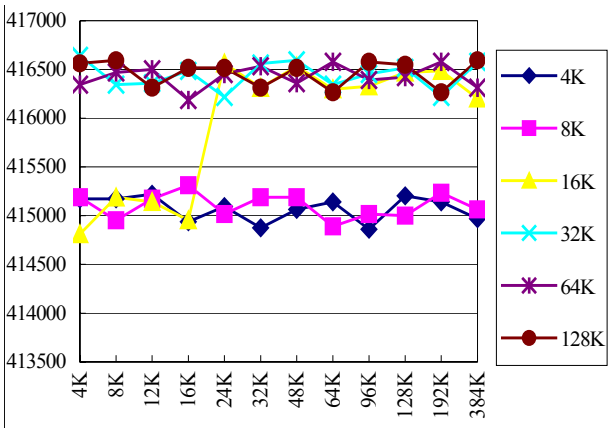


Fig.2 Performance testing results of Scanner for reading tokens

图 2 Scanner 读取记号时间测试结果

4.2 优化的自动机实现

在自动机的实现过程中,子解析过程中各个解析状态的循环内部对于下一字符的判断顺序还可以加以优化.由于我们是针对 XML 文档中的标记逐条进行语法解析的,所以对各种标记的识别效率将直接影响到解析器的效率.但是,由于不同的标记在 XML 文档中出现的频率差异很大,所以,我们在 OnceXMLParser 的语法分析器的实现上对这一部分作了重点优化.我们对来自 W3C 的一些典型的 XML 文档进行了统计,统计出各种标记/字符数据出现的频率,表 2~表 4 给出了标记/字符数据/属性类型的统计结果.

Table 2 The frequency of markups/characters in XML documents

表 2 XML 文档中标记/字符数据出现频度统计

Token/Character	*S*Tag	*E*Tag	Comment	CDATA	PI	Entity	CharData
Count	22 091	22 091	40 196	19	26	547	54 146
Percentage (%)	15.88	15.88	28.89	0.01	0.02	0.39	38.92

Table 3 The frequency of markups in DTD

表 3 DTD 中标记频度统计

Tokens	ElementDecl	AttListDecl	EntityDecl	Comment	PI
Count	7 845	141 450	28 884	40 056	0
Percentage (%)	3.59	64.82	13.24	18.35	0.00

Table 4 The frequency of attribute types

表 4 属性类型频率统计

Type	CDATA	ID	IDREF	IDREFS	NMTOKEN	ENUMERATION	Others
Count	102 744	7 708	100	191	15 232	12 815	32
Percentage (%)	74.1	5.55	0.07	0.1	10.97	9.2	0.01

根据上面的统计可以看出,CharData,Comment,*S*Tag 和 *E*Tag 的出现频度占有绝对优势.所以,在语法解析流程中应当将这些标记的识别放在优先的位置,也就是说,将对 CharData,Comment,*S*tag,*E*Tag 的判断放在其他标记的判断之前.而在 DTD 部分的解析中,可按照 AttListDecl,Common,EntityDecl,ElementDecl,PI 的次序进行判断.在属性的有效性验证过程中,也是先识别高频属性类型,然后处理低频类型,即按照 CDATA,

NMTOKEN,ENUMERATION 等的次序进行验证.在自动机的实现过程中,我们通过测试发现识别 XML 规范中的 Name 产生式的开销较大,这一方面是由于通常的 XML 文档中名字字符串就占有很大比重;另一方面是因为 XML 规范严格地定义了组成名字的字符集,解析器需要对组成名字的每一个字符进行合法性检查.为减少对 Name 产生式的识别操作,我们在 OnceXMLParser 验证版中设计实现了基于自动机的名字预测算法.我们首先在 DTD 解析组件解析 DTD 部分时,将已经通过合法性检查的元素名都保存在名字池中,随后,实现元素验证功能的自动机在解析 XML 文档正文时,将根据文档有效性约束和 DTD 中定义的元素出现规则,预测出下一个最可能出现的元素名.同时,如果下一元素名恰好是被预测到的元素名,则无需对此元素名进行合法性验证.通过上述的分析预测算法,OnceXMLParser 验证版对于相同的名字尽量只进行一次合法性检查,从而提高了解析效率.

4.3 合理的资源分配

按照 DOM 模型,在构造 DOM 树的时候要生成大量的节点对象、集合对象和字符串对象.JVM 在管理这些对象时,不仅需要一些额外的管理空间来保存与 Java 对象相关的信息,例如类型信息、虚函数表等,而且还要负责对象的自动回收.为了减少 JVM 管理对象的数量和开销,我们引入用户堆,由解析器自己管理字符串对象、集合对象和节点对象.

DOM 的节点对象包含众多信息,如 ElementImpl 节点就包含 localname,prefix,uri,attributes,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,故我们采用了延迟装载节点对象的方式,仅当用户访问有关节点信息时才从有关的用户堆中复制出节点信息,减少了不必要的信息复制.

另外,延迟策略也同样应用到了查询遍历上.DOM Element 接口的方法 getElementsByTagName(String name)是在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.考虑到用户调用 getElementsByTagName 方法之后,一般都只会使用结果集的第 1 个节点,所以,也采取了延迟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 个节点时,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.这种延迟策略可以减少不必要的遍历和数据复制操作,提高用户访问 DOM 模型的速度.

实验数据表明[26],上述资源分配策略均对 DOM 解析性能提高有所帮助.

4.4 Java层面的性能调整

继承、封装和多态是面向对象语言的三大基本特征.但对于面向对象思想的过度使用,反而有可能引起可读性和可维护性的下降,并有可能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特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,换取更高的性能.

在 OnceXMLParser 中,最多只有 1 层继承,主要用于控制对名字空间的支持,避免了层数过多的继承关系.我们对类进行了适度封装,虽然由于 Java 缺乏 C++语言所提供的 inline 关键字,不能从语言级支持方法的内联展开,但还是可以通过手工调整,减少简单方法特别是 get,set 方法的使用,避免小粒度的方法被重复多次调用.例如,我们使用性能调试工具 JProfiler 检查程序“热点”时发现,Scanner 的 getNCName()方法运行的开销比较大.getNCName()中调用了 XMLChar 类的字符判断方法 boolean isNCName(int c)进行字符的合法性判断:

```
public boolean isNCName(int c){return c<0x10000 &&(CHARS[c] & MASK_NCNAME) !=0;}
```

为此,我们将 CHARS[]数组更改为 static final 类型,然后将 isNCName 的方法体直接嵌入到上层的调用者 getNCName()的代码内部.这样虽然降低了封装性,但因为 CHARS[]是 final 类型,无法被外部用户修改,所以仍然能够保证代码安全性.上述代码重构虽然牺牲了部分代码的可读性,但却提高了性能.

5 性能比较

我们利用修改后的 Sun XML Test1.1,对 OnceXMLParser 进行性能测试.表 5~表 7 中的数据来自中国软件评测中心 2007 年 2 月出具的 XML 解析软件专题研究测试报告(编号 ZT280702037),测试所使用的是 CPU 为 3GHZ、内存为 2G 的台式 PC 机 DELL GX280,所使用的操作系统是 Windows 2003 Server,Java 的运行环境是 JDK 1.4.Test1~Test6 的定义请参见文献[10].表 5~表 7 中,O 表示 OnceXMLParser2.0,X 表示 Xerces2.9.0,W 表示 Woodstox3.2.0,括号内为 3 个参数 Well-formed,Validation,Namespace 的取值.例如,表 5 中,O(T,F,F)列就表示 OnceXMLParser 在进行良构性检查、不进行有效性验证、不支持名字空间情况下 SAX 接口的事务吞吐量.这样,表 5~表 7 中 O(T,T,*)列给出了 OnceXMLParser 验证版性能,O(T,F,*)给出了 OnceXMLParser 标准版性能.每次测试报告的是 XML Test1.1 定义的事务吞吐量.

Table 5 Performance testing results of OnceXMLParser and Xerces 2.9.0 on SAX

表 5 OnceXMLParser 与 Xerces2.9.0 在 SAX 接口上的比较

Transactions	X(T,F,F)	O(T,F,F)	X(T,F,T)	O(T,F,T,)	X(T,T,F,)	O(T,T,F)	X(T,T,T)	O(T,T,T)
SAX-Test-1	159.85	200.54	157.34	197.21	84.17	132.91	88.43	124.38
SAX-Test-2	81.39	110.21	75.73	103.20	47.68	70.60	41.59	63.87
SAX-Test-3	41.75	50.89	37.70	52.04	23.70	34.57	19.62	32.89

Table 6 Performance testing results of OnceXMLParser and Woodstox 3.2.0 on StAX

表 6 OnceXMLParser 与 Woodstox3.2.0 在 StAX 接口上的比较

Transactions	W(T,F,F)	O(T,F,F)	W(T,F,T)	O(T,F,T)	W(T,T,F)	O(T,T,F)	W(T,T,T)	O(T,T,T)
StAX-Test-1	183.30	223.04	184.48	216.32	112.33	152.59	103.10	132.41
StAX-Test-2	92.90	114.06	92.52	108.42	56.70	71.32	54.67	64.33
StAX-Test-3	46.35	57.73	47.06	54.57	28.65	39.04	28.14	34.77

Table 7 Performance testing results of OnceXMLParser and Xerces 2.9.0 on DOM

表 7 OnceXMLParser 与 Xerces2.9.0 在 DOM 接口上的比较

Transactions	X(T,F,F)	O(T,F,F)	X(T,F,T)	O(T,F,T)	X(T,T,F)	O(T,T,F)	X(T,T,T)	O(T,T,T)
DOM-Test-1	239.78	371.69	223.62	315.69	167.53	227.22	146.58	200.16
DOM-Test-2	229.40	363.78	212.67	307.43	151.01	226.19	145.95	201.38
DOM-Test-3	210.13	336.53	196.81	301.74	118.13	218.53	123.05	190.17
DOM-Test-4	225.93	351.01	211.93	305.10	121.18	226.23	142.83	201.44
DOM-Test-5	199.54	338.58	188.31	289.94	148.37	226.23	139.13	192.45
DOM-Test-6	198.48	333.64	185.18	285.53	150.79	209.19	135.78	189.78

概括而言,验证版 OnceXMLParser 在 SAX 上,平均比 Xerces2.9.0 快 52.27%;在 StAX 上,平均比 Woodstox3.2.0 快 27.92%;在 DOM 上,平均比 Xerces2.9.0 快 48.18%.标准版 OnceXMLParser 在 SAX 上,平均比 Xerces2.9.0 快 30.39%;在 StAX 上,平均比 Woodstox3.2.0 快 19.90%;在 DOM 上,平均比 Xerces2.9.0 快 54.84%.

我们还进行了 OnceXMLParser 轻量版与标准版的性能对比,表 8~表 10 给出了相关的测试数据.测试所使用的是 CPU 为 Pentium4 2.8GHZ、内存为 1G 的台式 PC 机 DELL GX280,所使用的操作系统是 Windows XP(Service Pack 2),Java 的运行环境是 J2SE 1.4.2_03.

Table 8 Performance testing results of light and standard OnceXMLParser on SAX

表 8 OnceXMLParser 轻量版与标准版在 SAX 接口上的比较

Transcations	O(T,F,F)	O(T,F,T,)	O(F,F,F)	O(F,F,T)
SAX-Test-1	209.44	197.98	219.46	202.11
SAX-Test-2	106.21	101.59	111.93	102.32
SAX-Test-3	53.24	51.20	55.96	52.21

Table 9 Performance testing results of light and standard OnceXMLParser on StAX

表 9 OnceXMLParser 轻量版与标准版在 StAX 接口上的比较

Transactions	O(T,F,F)	O(T,F,T)	O(F,F,F)	O(F,F,T)
StAX-Test-1	223.79	205.38	303.04	286.74
StAX-Test-2	110.91	104.43	155.98	144.85
StAX-Test-3	55.23	52.54	77.71	73.50

从表 8~表 10 的测试结果来看,轻量版 OnceXMLParser 的解析速度比标准版更快,SAX 接口平均提高 3.34%,StAX 接口平均提高 39.15%,DOM 接口平均提高 7.73%,达到了再次提高 XML 解析速度的目的.

Table 10 Performance testing results of light and standard OnceXMLParser on DOM

表 10 OnceXMLParser 轻量版与标准版在 DOM 接口上的比较

Transactions	O(T,F,F)	O(T,F,T)	O(F,F,F)	O(F,F,T)
DOM-Test-1	352.41	288.69	368.25	331.33
DOM-Test-2	345.29	283.17	354.93	322.32
DOM-Test-3	327.67	271.64	335.90	301.94
DOM-Test-4	341.47	281.54	350.20	318.60
DOM-Test-5	322.07	270.63	329.66	303.39
DOM-Test-6	318.92	269.16	324.16	299.90

6 结束语

XML 解析器是分析、处理 XML 文档的基础软件.本文研究了提高 XML 解析器性能的多项技术,给出了支持 DOM,SAX 和 StAX 规范的 XML 解析器 OnceXMLParser 的设计和实现.OnceXMLParser 通过了严格的 XML 兼容性测试和 API 兼容性测试.同时,性能测试结果表明,OnceXMLParser 具有出色的解析性能.

下一步的工作包括在 OnceXMLParser 中增加对 DOM Core 3 以及 XML 模式的支持.

致谢 感谢中国科学院软件研究所软件工程技术中心 XML 解析器小组成员杨波、汪剑超、张国栋、郭红艳、田四化在 OnceXMLParser 中所做的工作.

References:

[1] World Wide Web Consortium, Extensible Markup Language (XML) 1.0. 1998. http://www.w3.org/TR/1998/REC-xml-19980210

[2] World Wide Web Consortium, Extensible Markup Language (XML) 1.1. 2004. http://www.w3.org/TR/2004/REC-xml11-20040204

[3] World Wide Web Consortium, Extensible Markup Language (XML) Conformance Test Suites 20031210. 2003. http://www.w3.org/XML/Test/

[4] World Wide Web Consortium, Document Object Model. 2000. http://www.w3.org/DOM/

[5] David Brownell. SAX2. Sebastopol: O'Reilly & Associates, Inc., 2002.

[6] Java Community Process, Streaming API for XML JSR-173 Specification (Final v1.0). 2003. http://jcp.org/en/jsr/detail?id=173

[7] SAX2Unit. 2001. http://sourceforge.net/project/showfiles.php?group_id=8114&package_id=32032

[8] Tatu Saloranta, StaxTest. 2004. http://www.cowtowncoder.com/

[9] World Wide Web Consortium. DOM Conformance Test Suites. 2004. http://www.w3.org/DOM/Test/

[10] Sun Microsystems, XML Test v1.1. 2004. http://java.sun.com/performance/reference/codesamples

[11] Xerces2. 2007. http://xml.apache.org/xerces2-j/

[12] Crimson. 2004. http://xml.apache.org/crimson/

[13] BEA RI. 2004. http://dev2dev.bea.com/xml/stax.html

[14] SJSXP. 2007. https://sjsxp.dev.java.net/

[15] Woodstox. 2006. http://woodstox.codehaus.org/

[16] Oracle StAX Pull Parser. 2005. http://www.oracle.com/technology/tech/xml/xdk/staxpreview.html

[17] Codehaus StAX RI. 2006. http://stax.codehaus.org/

[18] Bruggemann-Klein A, Wood D. On predictive parsing and extended context-free grammars. In: Champarnaud J-M, Maurel D, eds. Proc. of the 7th Int'l Conf. on Implementation and Application of Automata. Tours: Springer-Verlag, 2002. 239–247.

[19] Albert J, Giammarresi D, Wood D. Normal form algorithms for extended context-free grammars. Theoretical Computer Science, 2001,267(1-2):35–47.

[20] Piccolo. http://piccolo.sourceforge.net/

[21] Takase T, Miyashita H, Suzumura T, Tatsubori M. An adaptive, fast, and safe XML parser based on byte sequences memorization. In: Ellis A, ed. Proc. of the 14th Int'l Conf. on World Wide Web. Chiba: ACM Press, 2005. 692–701.

[22] Zhou YM, Qu MB. A run-time adaptive and code-size efficient XML parser. In: Proc. of the 30th Int'l Computer Software and Applications Conf. Washington: IEEE Computer Society, 2006. 18−21. http://conferences.computer.org/compsac/2006/

[23] World Wide Web Consortium. Namespaces in XML. 1999. http://www.w3.org/TR/1999/REC-xml-names-19990114/

[24] Jiang ZL, Jiang SX. Formal Language and Theory of Automaton. Beijing: Tsinghua University Press, 2003 (in Chinese).

[25] Aho AV, Sethi R, Ullman JD. Compilers: Principles, Techniques, and Tools. Reading: Addison-Wesley, 1986.

[26] Yang B. The design and implementation of OnceDOMParser [MS. Thesis]. Beijing: Institute of Software, the Chinese Academy of Sciences, 2005 (in Chinese with English abstract).

附中文参考文献:

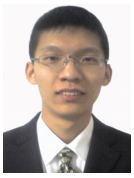
[24] 蒋宗礼,姜守旭.形式语言与自动机理论.北京:清华大学出版社,2003.

[26] 杨波.DOM 解析器 OnceDOMParser 的设计与实现[硕士学位论文].北京:中国科学院软件研究所,2005.

金蓓弘(1967－),女,浙江杭州人,博士,研究员,CCF 高级会员,主要研究领域为分布式计算,软件工程技术.

曹冬磊(1980－),男,博士生,主要研究领域为分布式计算.

任鑫(1981－),男,硕士,主要研究领域为分布式计算.



余双(1982－),男,硕士,主要研究领域为分布式计算.

戴蓓洁(1983－),女,硕士,主要研究领域为分布式计算.