用 JavaCC 和 JJTree 构造扩展模式文档解析器

周 健[1,2],孙丽艳[1]
(1.安徽财经大学,安徽 蚌埠 233010;
2.北京科技大学,北京 100083)

摘 要:XML正成为互联网中数据表示和数据交换的标准。扩展DTD支持面向XML的查询,但结构复杂,现有工具无法解析扩展DTD,获取元素之间的面向对象信息。利用JavaCC和JJTree设计解析器,包括构造语法和语义分析器,设计扩展DTD的语法规则和解析器的类结构,从而解析扩展DTD,利用解析后的扩展DTD的语法树获取面向对象信息,方便高效,不仅支持XML存储建立,而且支持面向对象XML查询方式,使得查询方式更为灵活多样。

关键词:JavaCC;JJTree;扩展DTD;解析
中图分类号:TP391 **文献标识码**:A **文章编号**:1673-629X(2008)09-0087-04

Designing Parser of Extended DTD by JavaCC and JJTree

ZHOU Jian[1,2],SUN Li-yan[1]
(1.Anhui University of Finance & Economics,Bengbu 233010,China;
2.Beijing University of Science and Technology,Beijing 100083,China)

Abstract:XML is emerging as the dominant standard for data representation and exchange over the Web.So the study of storage is becoming important more and more on object-oriented XML data.Extended DTD supports the object-oriented XML data, through parse the extended XML schema,can get the relationship among elements, which not only supports the new data model, but also supports the new way of query and the validity of language and XML document. In this paper,introduce the system design of compiler on extended DTD.

Key words:Java compiler compiler;JJTree;extended DTD;parser

0 引 言

XML(Extended Markup Language)[1]是由嵌套的标记元素构成的自描述标记语言,它正成为Internet上数据描述和交换的主要标准,针对XML数据的存储和查询技术已成为研究热点。众所周知,面向对象的方法具有很强的建模能力,例如继承、非单调继承、多态性、复杂数据结构等。如何将面向对象的特征引入到XML中,用以提高XML语言的建模能力是一个重要的研究方向。XML文档中结构信息由XML模式语言来进行描述,目前有很多XML模式语言,例如:DTD[2,3]、SOX[4]、XDR[5]、DSD[6]、XML Schema[7]。其中只有XML Schema和SOX支持继承,但它们都不支持多重继承和多态。因此有必要选择一种模式语言支持面向对象的XML文档。关于这方面的研究正在继续中,如把面向对象的观点应用到DTD中,称为扩展DTD[8],通过它建立一种基于规则并具有继承和多态特性的存储模型。虽然把面向对象观点扩充到DTD中简单而且易于理解,但现有解析工具不支持扩展DTD语法结构,扩展DTD也不支持XML格式,因此不能利用现有工具对扩展DTD进行解析,如果从扩展DTD中得到继承和多态的信息,就需要把扩展DTD转化成语法树,再从语法树遍历过程中得到所需要的继承和多态信息。如何能把扩展DTD转化为合理有效的语法树是人们所关心的问题。

收稿日期:2007-12-24
基金项目:安徽省科研计划项目资助(2007jq1084)
作者简介:周 健(1979-),男,安徽凤阳人,博士研究生,讲师,研究方向为XML数据库、网络安全。

1 扩展DTD

扩展DTD中语法包括扩展前和扩展后的语法语义,这里主要关注于扩展后的语法语义。在扩展DTD中扩展了五种语法语义[8,9]:

(1)元素继承'<! ELEMENT' element-name ['ISA' super-element-names-list]'('element-content-models')>'。通过使用该语法结构,定义元素的父亲元素,扩展元素不仅继承了基类元素的属性也继承了它的独占元素,如<! ELEMENT student ISA

person (addr, dept, takes)>,说明了元素 student 继承元素 person,它的独占元素是 addr, dept, takes,也继承了 person 的独占属性和嵌套元素;

(2)元素和属性阻断,'<! ELEMENT' element - name '(BLOCKED FROM' super - element - names - list ')>'和'<! ATTLIST' element - name attribute - name '(BLOCKED FROM' super - element - nameslist')>',通过该语法结构定义扩展类阻断从超类继承得到的元素或属性,如<! ELEMENT homephone (BLOCKED FROM person)>定义元素 teacher 阻断从 person 中继承得到的元素 homephone;

(3)元素或属性重命名,使用语法结构'ISA' {super - element {'WITHELEMENT (' old - element - name'AS'new - elementnam')'}* }+定义元素重命名机制,如<! ELEMENT TA ISA student WITHELEMENT (dept AS student - dept), teacher WITHELEMENT (dept AS teacher - dept) >中定义扩展元素 TA 从基类元素 student 继承得到的 dept 被重命名为 student - dept,从基类元素 teacher 中继承得到的元素 dept 被重命名为 teacher - dept,不会发生命名冲突;

(4)多态定义,一个元素定义不仅表示本身,而且代表它扩展的元素,如语句<! ELEMENT univ (person *, course *)>定义在 XML 文档中,一个 person 类不仅代表自己本身的实例,而且包括{student、teacher、TA}的实例,因为这些元素都是 person 类的扩展类;

(5)多态引用,使用'<! ATTLIST' elem attr 'IDREF' '[' targetElem']' ....'>'和'<! ATTLIST' elem attr 'IDREFS' '[' targetElem']' ....'>'定义元素的引用和被引用关系,例如<! ATTLIST teaches courses IDREFS course #IMPLIED>定义了元素 teaches 的属性,courses 引用了 course 元素,在本质上指的是同一类对象。扩展 DTD 能够支持元素继承层次、多重继承、重载、阻断、多态和冲突处理机制等。具体的扩展模式语言语法参见文献[8]。

2 JavaCC 和 JJTree

现有工具如 DTD parser[3,10],不支持扩展 DTD 的解析,且 DTD 不支持 XML 格式。因此获取扩展 DTD 的信息,必须构造扩展 DTD 解析器,通过编译器对扩展 DTD 文档进行分析形成语法树,从语法树中遍历提取信息。完整的解析器包含有词法、语法、语义分析器。如果手工建立词法、语法分析器,是一项非常繁杂的工作。幸运的是,有很多词法分析器和语法分析器的自动生成工具可以使用。现有工具如 JavaCC 和 JJTree[11~15]、YACC 和 Lex[16~18] 等。JavaCC (Java Compiler Compiler)是由 Sun 公司开发的编译器自动生成工具,有以下的显著特征:

(1)文法规则表达能力强、直观、易于调试;

(2) 词法、语法规则都定义在同一个文件中,易于阅读和维护;

(3)提供功能强大的建立分析树的预处理器——JJTree,提供树和节点类,易于将代码分成单离的解析和动作类;

(4)可以定制生成语法词法分析器的行为,是否对大小写敏感等;

(5)独特的语法(Syntactic)向前看和语义(Semantic)向前看规则,JavaCC 提供了"向前看"规则来解决移进冲突。

考虑到设计是基于 Java 平台的,因此选择 JavaCC 和 JJTree 解析扩展 DTD。

3 扩展 DTD 解析

解析过程如下:首先通过 JavaCC 和 JJTree 构造扩展 DTD 的解析器;然后将扩展 DTD 读入内存解析,形成格式良好的数据结构,也就是形成语法树结构;最后对语法树进行遍历,获取元素之间的多态和继承信息。如图 1 所示,前四步是构造扩展 DTD 解析器的构造过程,后四步是对扩展 DTD 解析和信息获取的过程。

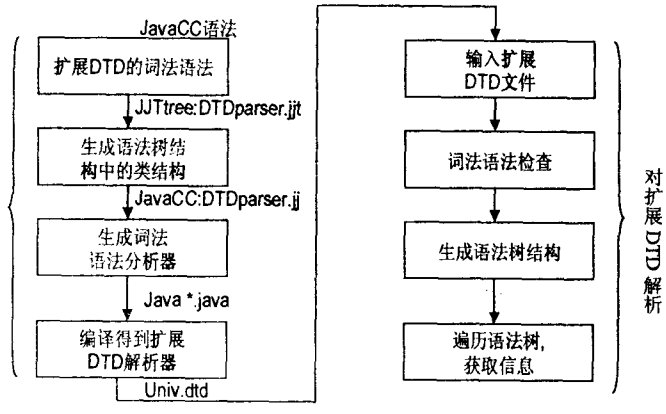

图 1 扩展 DTD 解析流程

使用正则表达式,在名为 DTDparser.jjt 的文件里使用 JavaCC 语法定义该语言词法规则、文法规则、语义动作说明(java 代码段)。要在 DTDparser.jjt 文件中定义语言,需要做以下 5 件事:

(1)定义解析环境,JavaCC .jj (执行完 JJTree 后获得)文件通过执行 JavaCC 将被转换为 .java 文件。JavaCC 将获取 DTDparser .jj 文件里 PARSER_BEGIN

与PARSER_END的中间部分并将之复制到Java结果文件中,可以将解析前后所有与解析器有关的动作置于该文件中。

(2)定义"空白",就是将空格、跳格、回车和换行作为分隔符处理,而不是将其忽略,在SKIP区域中定义这些字符。

(3)定义"标记(token)",就是定义该语言所识别的标记,标记是将对解析程序有意义的解析字符串的最小单位,扫描输入字符串以及判断是何标记的过程称作记号赋予器(tokenizer)。

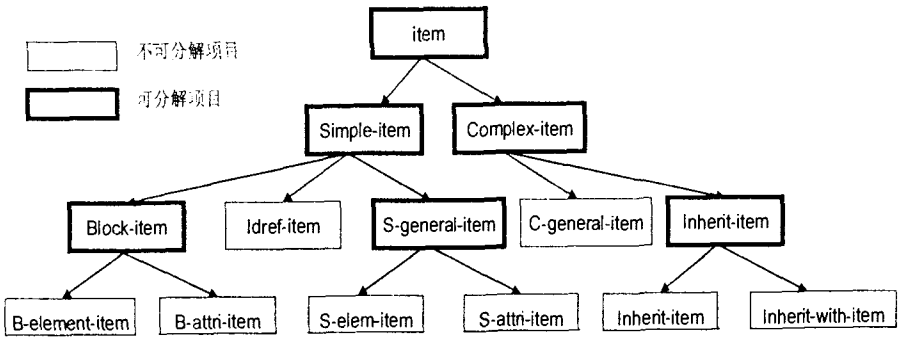

图2　扩展DTD语句分类

(4)按照标记定义语言本身的语法,按照标记来定义解析规则(产生式),当对DTDparser.jj文件运行JavaCC时,产生式将被转换为方法,编译并"运行"该程序以查看解析器运作是否正确。

(5)定义每个解析阶段中将发生的行为动作,动作是指为响应特定产生式而执行的Java代码。

由于篇幅原因,主要关注第4步骤中的定义语言本身语法。通过对扩展DTD的语法分析,扩展DTD语句是包含在"<"(LBRACKET)和">"(RBRACKET)中,两个标志符号内有一个DOCTYPE关键字,说明文件根元素(XMLHEAD),在"["(LFRACKET)和"]"(RFRACKET)中为元素或属性说明性语句,说明元素或属性的数据结构,每个说明语句都包含在"<"和">"中。因此把每个说明行语句看成一项,在语法树中用一个节点标志,在获取信息中,扫描分析每个节点及其子树,分析节点名称就可以获取信息。把每个尖括号内的内容看成一项(item),例如<! ELEMENT name (#PCDATA)>,对项目分类,使得对语法树的遍历简单和高效。项目可以从对元素的描述上,分成简单、复杂项目,简单项目分成一般简单项目、阻断项目、引用项目;复杂项目分成复杂项目、继承项目;继承项目分成一般继承项目和有重命名项目。其中阻断简单项目还要分成元素和属性两种,一般简单项目也是一样分成属性和元素说明项目。如图2所示。

针对8种元素或属性说明项目,构造语法树节点类:idrefItem(引用说明项目)、BelemItem(元素阻断说明项目)、BattriItem(属性阻断说明项目)、SelemItem(简单元素说明项目)、SattriItem(简单属性说明项目)、CgeneralItem(复杂元素说明项目,不含有继承关系)、InheritgenItem(一般继承项目)、InheritwithItem(重命名项目)。相应构造项目的节点类(ASTidrefItem, ASTBelemItem, ASTBattriItem, ASTSelemItem, ASTSattriItem, ASTInheritgenItem, ASTInheritwithItem, ASTCgeneralItem)和两个基本节点类(SimpleNode、Node)。

对元素或属性说明项目语句,需使用正则表达式描述语法。如IdrefItem={属于的元素名称、引用属性名称、被引用属性名称、数据类型说明};InheritgenItem={属性名称,& 父亲说明,& 包含的嵌套元素};括号内的这些项有些是不包含分割符的文本字符串(只需要一个标记和类节点表示),有些是包含有分割符的字符串(使用 & 标志,需要分成多个标记和类节点),如图3所示定义InheritgenItem元素说明语句,使用同样的方式构造其他元素或属性说明性语句。执行JJTree命令后,会生成一些节点类如ASTinclude(描述复杂元素数据类型)、ASTtype(描述简单元素数据类型)。

```
Token: {
<! ELEMENT: "! ELEMENT" > | < COMMA: "," > | <
LBRACKET: "<" > | < RBRACKET: ">" > |
<LPAREN: "(" > | < RPAREN: ")" > | <FATHER: "ISA" > |
<NAME: <LETTER> (<LETTER> | <DIGIT>) * > | <
#LETTER: ["_","a" - "z","A" - "Z"] > | < #DIGIT: ["0" -
"9"] > ……}
Include(): {{{
<LPAREN > (< NAME > < COMMA > ) * < NAME > <
RPAREN > |
InheritgenItem(): {{{
<LBRACKET > <! ELEMENT > <NAME > <FATHER > (<
NAME > ) + (Include())? <RBRACKET > |
```

图3　InheritgenItem元素说明语句

如果JavaCC没有发现文法错误,那么生成的扩展DTD解析器类结构如图4所示,DTDparser.java(语法分析器,也是主程序),DTDParserTokenManager.java(词法分析器)、Token.java(标记的信息)、DTDparserConstants.java(定义分析器中使用的常量)、SimpleCharStream.java(字符流类)、ParserException.java(可恢复异常,解析器发现问题时抛出)、TokenMgrError.java(无法恢复错误管理,记号管理器发现问题抛

出)、JJTDtdparserState. java(提供方法操作节点栈)、DtdparserTreeConstants. java(定义一些有用的符号常量)、Node. java(语法树节点接口类)、SimpleNode. java(语法树中所有节点的父类,提供了一些基本方法),虚框内为语法树中包含的各种节点类(部分)。在 DTDparser. java 中输入文件(univ. dtd),通过指针 jjtThis 返回语法树的头节点。

通过 JJTree 把扩展 DTD 解析成抽象语法树(AST:Abstract Syntax Tree),语法分析结果不再是符号序列,而是由一个对象构成的树型层次结构。在该语法树中,所需要的信息集中在语法树的第四层到第六层,在这里获取元素间关系,扫描语法树的标签,如果该标签满足需要则从该标签下获取镜像值(image),这里需要添加或修改 SimpleNode 部分的函数,如添加函数 getName(),以便获得镜像值。如果需要获取所有的元素名称,那么可以扫描项目的节点类的标记,获取标记名称为 SelemItem、CgeneralItem、InheritgenItem、InheritwithItem 的说明项目的节点的元素名称(element - name)就可以得到所有的元素。如果获取阻断信息,只要扫描标签符合 BelemItem 和 BattriItem 名称的,得到该标签下的子树,扫描该子树下节点标签为 name、resource 就可以得到所有阻断信息中的阻断名称和阻断来源。同样其他的信息,如继承、重命名等等都在扫描标签的基础上,得到符合标签的子树,对每个项目包含的镜像值进行获取。如图 5 所示。

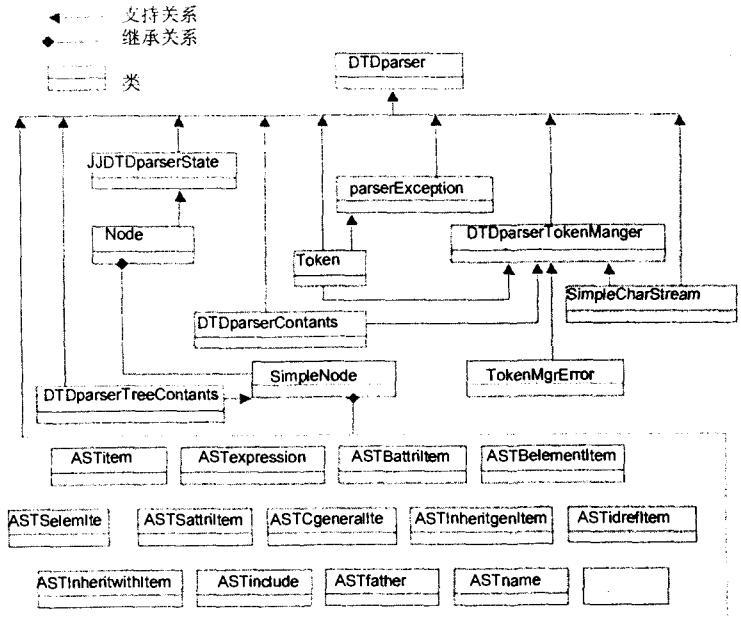

图 4 扩展 DTD 解析器的构造

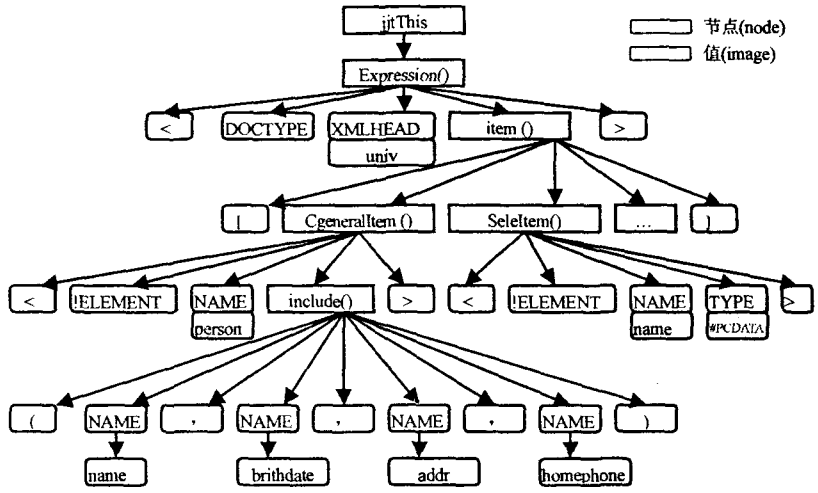

图 5 扩展模式文档的部分语法树结构

4 结束语

通过使用先进的语法分析器自动生成工具——JavaCC 和 JJTree,构造了扩展 DTD 的编译器。该编译器不仅可以对新模式语言——扩展 DTD 进行语法语义检测,而且通过该编译器,对扩展 DTD 中描述的内容形成格式良好的语法树。通过该语法树,只要编写一些简单的算法就可以获取模式文档中描述元素和属性的关系,这样就解决了现有工具不支持扩展 DTD,而且从扩展 DTD 获取继承和多态信息困难这一问题。

参考文献:

[1] Bray T, Paoli J. Extensible Markup Language (XML)1.0 (Third Edition)[EB/OL]. 2004 - 04. http://www.w3.org/TR/REC - xml/.

[2] W3Schools. Document type definition, DTD[EB/OL]. 2004 - 07. http://www.w3schools.com/dtd/default.asp.

[3] Wutka. Java DTD Parser[EB/OL]. 2004. http://www.wutka.com/dtdparser.html.

[4] Davidson A, Fuchs A. Simple OutLine XML—SOX[EB/OL]. 1999 - 07. http://www.w3c.org/TR/NOTE - SOX/.

[5] Sun Microsystems. RFC1014—XDR External Data Representation standard: External data Representation standard[EB/OL]. 1987 - 07. http://www.faqs.org/rfcs/rfc1014.html.

[6] Møller A. Document Structure Description. The DSD 2.0 specification[EB/OL]. 2004 - 07. http://www.brics.dk/DSD/.

[7] Sperberg - McQueen C M, Thompson H. XML Schema 1.1

(下转第 94 页)

不同类型服务的、标准的、可重用的策略。虽然服务进行分类还没有业界统一的标准,但通用描述、发现和集成(UDDI)注册表正在成为事实上的标准。通过对UDDI注册表中的服务进行分类,不但可以更好地为潜在的候选项分类,而且还能发现可以重用的现有服务,从而避免了功能的不必要重复。

第三步是确定服务粒度,服务的粒度是一项重要的设计要点。细粒度的接口为请求者应用程序提供了更多的灵活性,但也意味着交互的模式可能随着不同的服务请求者而不同,使对于服务提供者的支持更加困难。粗粒度接口保证服务请求者将以一致的方式使用服务,但却缺乏灵活性。针对本系统,对于外部的消耗推荐使用粗粒度的接口,而企业内部可以使用细粒度的接口。即"内部服务容器"中的Java应用容器组和Web服务容器组是细粒度的服务,而"外部服务容器"中是粗粒度的服务。

第四步是服务获取,定义了服务的组合之后,就要确定如何获取实际的服务,即内部构建、获取服务或预订外部供应商提供的服务。按照一般规则,那些关键性业务服务(即有益于业务流程、能为机构争取竞争优势地位的服务)最好是由内部提供。另外,创建一个服务之间关系的目录有助于构建服务组合。UDDI注册表是一个价值极高的工具,利用它可以构建一个服务和相关产物的完整在线目录。

通过服务发现、服务分类、确定服务粒度以及服务获取,在资源共享与集成基础上,涉及科技基础条件资源涵盖的各个领域,跨平台、跨领域地构建Web合成服务,为实现科技资源的共享提供更加强大的服务。

构建的Web合成服务通过门户(Web Portal)提供包括内容聚合、单点登陆、个性化定制和安全管理等服务的基础Web平台。客户端(Clients),即用户可以使用浏览器,支持WAP协议的手机或者其它的设备访问信息门户获取信息。

3 结束语

Web Services技术的主要目标就是在现有的各种异构平台的基础上构筑一个通用的与开发平台、语言无关的技术平台,各种不同平台之上的应用依靠这个技术平台来彼此连接和集成。文中结合工作流技术和Web Services技术,在此基础上建立了基于Web Services的Web服务合成应用。目前Web服务合成技术在国内仍处于起步阶段,由于它需要其他相关技术的支持,如复合服务执行及事务处理等技术的协作,所以如果要使Web服务合成发挥出其所拥有的潜能,仍然有许多的研究工作需要开展。

参考文献:

[1] 柴晓路,梁宇奇. Web Service技术、架构和应用[M]. 北京:电子工业出版社,2003:5-15.

[2] 王 强. 业务流程开发新纪元——BPEL4WS语言介绍[EB/OL]. 2003-03. http://www.ibm.com/developerworks/cn/webservices/ws-bpel/part1/index.html.

[3] 麻志毅,陈泓婕. 一种面向服务的体系结构参考模型[J]. 计算机学报,2006,29(7):1012-1013.

[4] 余名高,贾秀峰,林坤江,等. 基于Web服务的企业应用集成[J]. 计算机技术与发展,2007,17(5):56-57.

[5] 许科峰,高建民,陈富民,等. 基于Web Services的企业应用集成技术及实现[J]. 计算机应用,2004,24(3):155-156.

[6] BPEL4WS语言规范[EB/OL]. 2002-07. http://www.ibm.com/developerworks/library/ws-bpel/.

(上接第90页)

specification[EB/OL]. 2000-04. http://www.w3.org/XML/Schema.

[8] Wang G, Liu M. Extending XML Schema with Nonmonotonic Inheritance[C]//In Proceedings of 1st International Workshop on XML Schema and Data Management(ER Workshop XSDM'03). Chicago, Illinois, USA:[s.n.],2003.

[9] 张晓琳,王国仁. 用继承扩展XML-RL[J]. 小型微型计算机系统,2005,26(2):243-247.

[10] Bourret R. DTD parser vision2.0[EB/OL]. 2005-04-10. http://www.rpbourret.com/dtdparser/.

[11] 姚 砺,束永安. 用JavaCC构造编译器的方法[J]. 计算机工程,2003,29(9):39-41.

[12] 张 昱,付 雄. 含Xpath的表达式的解析与应用[J]. 小型微型计算机系统,2004,25(3)3:442-446.

[13] Sun o'reilly. Java Compiler Compiler - Javacc[EB/OL]. 2005-04-10. http://javacc.dev.java.net/.

[14] Katz H. JavaCC、解析树和XQuery语法[EB/OL]. 2002-12. http://bbs.xml.org.cn/dispbbs.asp? boardID=14&ID=8111.

[15] Sun o'reilly. The document of javacc[EB/OL]. 2005-04-10. http://javacc.dev.java.net/doc/CharStream.html.

[16] Bansal A. Yacc与Lex快速入门[EB/OL]. 2000-11. http://www-128.ibm.com/developerworks/cn/linux/sdk/lex/index.html.

[17] Johnson S C. Yacc:另一个编译器的编译器[EB/OL]. 2005-07-01. http://mhss.nease.net/unix/yacc.html.

[18] 李朝阳,潘 清. 应用YACC和Lex工具开发命令分析程序[J]. 软件世界,1997(2):44-45.