用Xerces-J进行基于XML Schema的XML局部验证

张 昱，李 凡

（中国科学技术大学 计算机科学技术系，安徽 合肥 230027）
E-mail：yuzhang@ustc.edu.cn

摘 要：XML验证，特别是局部验证，是XML应用中的一个关键问题. 调研分析了Xerces-J2工具包中的基于XML Schema的XML解析、验证体系，它是一种延迟的整体验证. 利用Xerces-J2，设计实现了基于XML Schema的XML局部验证接口，包括类型检查和ID约束检查等.

关 键 词：XML Schema；局部验证；类型检查；ID约束检查
中图分类号：TP311 **文献标识码**：A **文 章 编 号**：1000-1220(2005)08-1369-05

Partial Validation of XML Document Based on XML Schema with Xerces-J

ZHANG Yu, LI Fan

(*Department of Computer Science, University of Science & Technology of China, Hefei* 230027, *China*)

Abstract: XML validation, especially partial validation, is one of the key problems in XML applications. The parsing and validating architecture based on XML Schema in Xerces-J2 toolkit is investigated and analyzed, which is a kind of deferred full validation. Using Xerces-J2 the XML partial validating interfaces based on XML Schema are designed and implemented, including type checking, identity constraints checking and so on.

Key words: XML Schema; partial validation; type checking; identity constraints checking

1 引 言

在XML的早期应用中，DTD作为XML文档定义发挥了很大的作用，但是DTD有自身的局限性：1）它有专门的非XML的语法规则，需要解析器专门处理；2）就XML的元素内容而言，DTD几乎没有数据类型的定义，等等. 2001年W3C发布了XML Schema建议[1-3]，其类型框架借鉴ISO11404的语言无关的类型标准以及一般OO语言和SQL的类型系统，类型系统相当丰富，包括复杂数据类型定义、ID约束(Identity Constraints)等等. 在XML Schema（以下简称Schema）建议推出后，一些组织逐步推出支持其解析和验证的工具包，最著名的是Microsoft的MSXML和Apache的Xerces系列[4]. 目前这些工具包只支持对XML文档的整体Schema验证和Schema组件的信息访问，并没有提供对XML的局部Schema验证. 而局部验证对XML应用是非常重要的：因为实际的XML数据往往是动态变化的，对这些XML数据频繁地进行整体验证，会大大地降低应用本身的性能和效率. 为此，笔者深入剖析Xerces-J2软件包，理解其类体系和Schema验证机制，设计并实现了基于Xerces-J2的XML局部Schema验证接口. 本文即是对这一工作的总结.

2 问题描述

若一个XML文档d满足所给的文档定义τ，则称d是有效的(valid)，记作$d\in sat(\tau)$. 对d的整体验证是指给定一个文档定义τ，从零开始检查$d\in sat(\tau)$是否成立. 整体验证是一种延迟(deferred)验证，目前可用的XML Schema验证工具都属此类.

一般地，对$d(d\in sat(\tau))$的更新操作包括：

1）将d中某结点用另一结点替换或修改结点的部分信息；

2）在某给定结点之后插入一个新结点；

3）将新结点作为某结点的第一个孩子插入；

4）删除某给定的结点（其子结点都删除）. 那么更新d后，对新产生文档d'的验证可以从对比d'与d的差异出发.

操作1)～3)需要检测新结点是否符合相应的文档定义；将新结点插入或者删除某结点后，需要检测父结点是否仍有效. 这里的结点可以是XML文档中的属性、文本或元素. 这样，对XML最基本的验证将集中在对XML结点的验证上. 由于Schema对XML数据不仅有简单、复杂类型的约束（图2中的＜xs:complexType＞），还有ID约束（如图2中的＜xs:key name="bookKey"＞），因此对XML结点的局部Schema验证将包括类型检查以及ID约束检查.

假设XML文档为d，Schema定义为τ，结点位置为p（为XPath表达式），则：

1）对d中p结点的类型检查定义为检查p结点的值v是否满足τ中相应的类型定义，接口形式可以表示为：check-

收稿日期：2004-02-18 基金项目：国家自然科学基金项目(60173049)资助. 作者简介：张 昱，女，1972年生，博士，副教授，研究方向为XML数据处理、程序设计语言理论和实现技术、软件体系结构；李凡，1981年生，硕士研究生.

Type(d,τ,p)或checkType(v,τ,p)；

2) 对p结点的ID约束检查是从τ中取得关于p的ID约束$\tau_{unique}(p)$、$\tau_{key}(p)$和$\tau_{keyref}(p)$，检查p结点的值v在d的上下文上是否满足这些约束，接口形式为checkIDCs(d,τ,p).

```
〈? xml version="1.0" ?〉
〈books xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="books-simple.xsd"〉
 〈book name="english" pagenum="217" version ="2.0"〉
   〈author〉John〈/author〉
   〈price currence="RMB"〉43〈/price〉
   〈ISBN〉2407763510〈/ISBN〉
   〈comments〉This is a good book.〈/comments〉
 〈/book〉
 〈book name="chinese" pagenum="321" version ="4.0"〉
   〈author〉nwu〈/author〉
   〈price currence="RMB"〉48〈/price〉
   〈ISBN〉2407763510〈/ISBN〉
 〈/book〉
〈/books〉
```

图1 一个XML文档例子books.xml

图1为一个书目信息的XML文档，图2为相应的Schema定义.对结点/books/book的类型检查将通过，但是ID约束检查不通过，因为〈books〉中包含2个具有相同〈ISBN〉文本"2407763510"的〈book〉元素，不符合图2中名为bookKey的key约束.

```
〈xs:schemaxmlns:xs="http://www.w3.org/2001/XML Schema"〉
 〈xs:element name="books"〉
  〈xs:complexType〉〈xs:sequence〉
   〈xs:element name="book" minOccurs="0" maxOccurs="unbounded"〉
    〈xs:complexType〉
     〈xs:sequence〉
      〈xs:element name="author" type="xs:string"/〉
      〈xs:element name = "price"〉
       〈xs:complexType〉〈xs:simpleContent〉
        〈xs:extension base="xs:decimal"〉
        〈xs:attribute name = "currence" type = "xs:string" /〉
        〈/xs:extension〉〈/xs:simpleContent〉
       〈/xs:complexType〉
      〈/xs:element〉
      〈xs:element name="ISBN" type="xs:string"/〉
      〈xs:element name="comments" type="xs:string" minOccurs="0"/〉
     〈/xs:sequence〉
     〈xs:attribute name="name" type="xs:string"/〉
     〈xs:attribute name="pagenum" type="xs:string"/〉
     〈xs:attribute name="version" type="xs:string"/〉
    〈/xs:complexType〉
   〈/xs:element〉
  〈/xs:sequence〉〈/xs:complexType〉
  〈xs:key name="bookKey"〉
   〈xs:selector xpath="book"/〉
   〈xs:field xpath="ISBN"/〉
  〈/xs:key〉
 〈/xs:element〉
〈/xs:schema〉
```

图2 books_simple.xsd的内容

3 Xerces-J2的Schema验证机制

XML的API规范主要有DOM(Document Object Model)和SAX(Simple API for XML)两类.前者以树结构(父/子结点)表示XML，该信息集为一个Document对象，其中每个信息项(结点)可以通过提供的方法来访问和修改.后者将XML文档视为事件流，如start Element、end Element等，SAX解析器生成这些事件并将它们分发到处理程序进行相应的处理；它不支持对XML数据的随意访问和修改.

3.1 整体验证方法

Xerces-J2的parsers包[1]中有SAXParser和DOMParser两类XML解析器，解析基于管道(pipeline).在解析器的配置中将DTD验证器作为管道的一部分，这意味着无论解析器是否设置报告验证错误，它仍然对文档进行验证.Xerces-J的Schema整体验证基于DOMParser.

```
    parser = new DOMParser();
parser.setProperty("http://apache.org/xml/properties/schema/
    external-noNamespaceSchemaLocation",
                                        "books-simple.xsd");
try {
  // 设置解析器报告验证错误
  parser.setFeature("http://xml.org/sax/features/validation", true);
  // 设置将schema验证器加入到管道
  parser.setFeature("http://apache.org/xml/features/validation/schema", true);
  parser.parse("books.xml"); // 对books.xml进行解析、验证
  doc = parser.getDocument(); // 取得DOM文档树
} catch (IOException ie){…
} catch (SAXException e){…
}
```

图3 Schema整体验证代码段

若要验证books.xml是否符合books_simple.xsd，可以用图3的代码段；由于books.xml根元素指定noNamespaceSchemaLocation属性，则第2条语句可以省略.Xerces使用特性体系，它允许用户使用setFeature( )方法启用或禁用表示某些能力的特性的URI.执行图3的代码后，将会报出如下错误：

[Error] books.xml: 13: 26: Duplicate key value [2407763510] declared for identity constraint of element "books".

值得注意的是，即使是用DOMParser解析，也可能出现与SAX有关的异常.这是因为DOMParser和SAXParser只是表示解析器最终产生了什么，而不表示将如何完成它.事实上，这两种解析器都使用SAX解析文档.这导致Xerces-J中Schema验证过程是不可逆的，除非从头扫描XML文件.

3.2 Xerces-J2的Schema验证体系

深入分析Xerces-J2源码，总结得到其Schema整体验证

[1]Xerces-J2主要收集在org.apache.xerces包中，简便起见，文中的各包名省略此前缀.本文采用2003年11月20日发布的Xerces-J 2.6.0版本.

体系(图 4). 图中右边的各个类负责扫描、解析 XML 文档. XMLDocumentFragmentScannerImpl 是核心的文档扫描类,它每读完一个片段,如由符号"〈"和"〉"包含的一条语句或文本串,就通过相应的 Dispatcher 实现类进行分发处理.

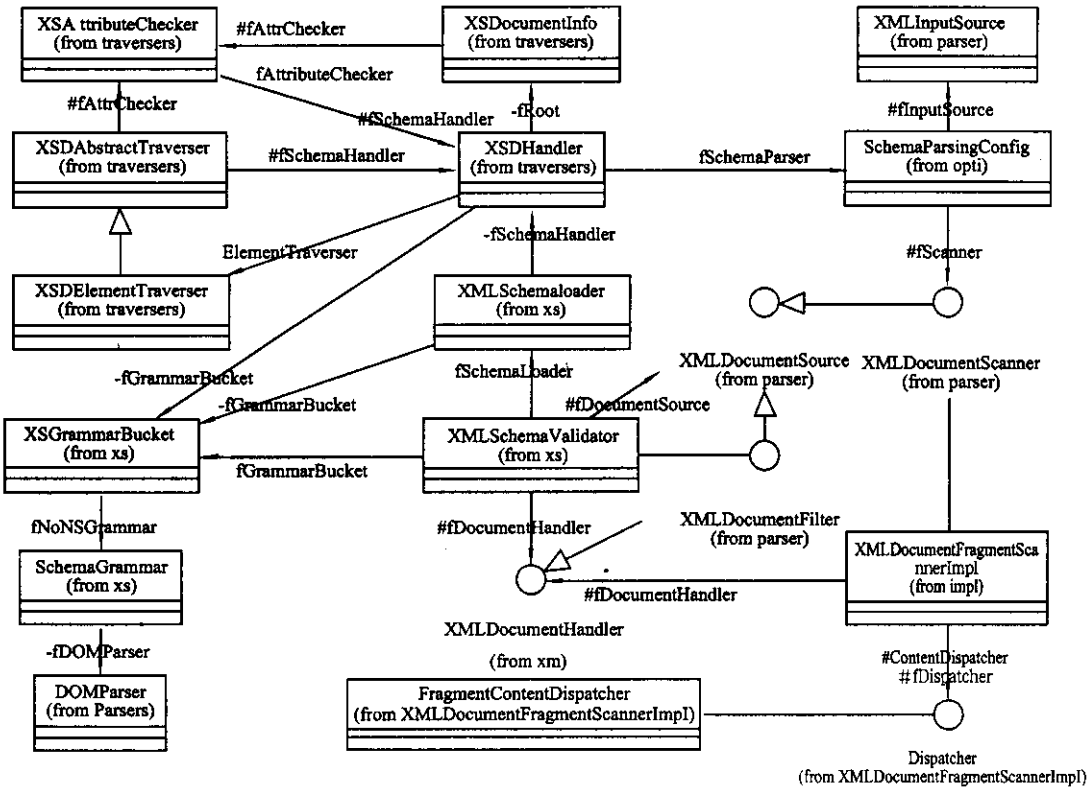

图 4　Xerces-J2 的 Schema 验证体系

XMLSchemaLoader 负责装载 Schema 文件并通过句柄 fSchemaHandler 的 parseSchema( )完成对它的解析,解析分两步:先通过 DOMParser 进行常规的 XML 解析得到一棵 DOM 树 schemaRoot;然后对 schemaRoot 进行 Schema 组件[2]的语法检查生成 SchemaGrammar 实例并存储到 fGrammarBucket 容器中. 这将用到图 4 左边的各种类.

XMLSchemaValidator 是验证的核心,它响应各 SAX 事件,根据 fGrammarBucket 中的语法定义完成对当前解析的文档片段的验证. 以元素验证为例,验证工作由 startElement( )和 endElement( )分担实施,前者负责检查当前元素的各个属性(属性组)的有效性,激活 ID 约束进行 XPath 的匹配;后者负责检查元素内容的有效性,检查 ID 约束,先处理 key 或 unique,后处理 keyref,检查根元素上附加的 schema 约束(Unique Particle Attribute 约束、particle 限制、元素声明的一致性等).

4　局部验证的设计与实现

我们设计 PartialSchemaValidate 类封装局部验证功能,除了支持类型检查和 ID 约束检查外,还需提供对 XML 文档和 Schema 文件的装载,按给定的 XPath 查找对应的 Schema 定义等.

4.1　Schema 的装载

利用 XMLSchemaLoader 类可以完成 Schema 的装载,所得的语法对象可以是 SchemaGrammar,也可以是 XSModel,其区别在于前者管理同一名称空间的语法,后者管理多个名称空间的语法. 装载的代码实现如下:

```
XSModel loadSchema(String file){ // file 为 Schema 的 URI
    XMLSchemaLoader schemaLoader = new XMLSchemaLoader
                                                    ();
    try{
            fGrammar = ( SchemaGrammar ) schemaLoader.
            loadGrammar(new XMLInputSource(null,file,null));
            fXSModel = fGrammar.toXSModel();
    }catch(IOException ioe){…}
    return fXSModel;
}
```

4.2　获取指定结点的 Schema 定义

Xerces-J 的 DOM 实现(dom 包)中,在属性和元素结点的实现类 AttrImpl、ElementNSImpl 中引入 type 域保存结点的类型定义. 当按图 3 解析时,这些 type 域为 xs. XSTypeDefinition 实现类的实例. 由于 type 是 protected 的且只有类型名的 public 访问方法,因此在 dom 包外难以利用这些已经得到的类型信息. 为此需要提供获取指定结点的 Schema 定义的方法,该方法的伪代码如下:

```
/* xpath 为寻址 XML 结点的 XPath 式,为简便起见,假设它的每一 step 不含. 若结点是元素则返回 XSElementDecl 实例,若结点是属性则返回 XSAttributeDecl 实例 */
XSObject getDecl(String xpath){
    1) 将 xpath 分解得到各个 steps;//可用 impl. xpath. XPath 类完成该解析
    2) 从语法对象 fGrammar 或 fXSModel 中取得该 xpath 关联的根元素的声明 fatherDecl
    3) i=1;
    4) while (i<steps. length)
        a) fatherType=(XSComplexTypeDecl)fatherDecl. get TypeDefinition(); // 取得元素的类型
        b) 取得 steps [i]的 stepname(去除前导符剩下的串)
        c) switch ( steps [i]的前导符 ){
          case '.': //自身
            break;
          case '@': //属性
            attrGroup = fatherType. getAttrGrp(); //取得属性组
            attrDecl = attrGroup. getAttributeUse(null,step name). getAttrDeclaration(); //取得属性声明
            return attrDecl;
          case '/': //子元素
            particle = (XSParticleDecl)fatherType. getParticle(); //获取 Particle 组件
            modelGrp = (XSModelGroupImpl)particle. getTerm();//获取模型组
            objList = (XSObjectListImpl)modelGrp. getParticles(); //获取模型组中的对象列表
            j=0;
            while( j<objList. getLength( ) )
              i) particle = (XSParticleDecl)objList. item(j);
              ii) elemDecl = (XSElementDecl)particle. getTerm();
              iii) if ( elemDecl. getName(). compareTo(step name)==0){ fatherDecl = elemDecl; break;}
              iv) j++
            if (j==objList. getLength( ) ) return null;//不存 在
            else if (j==objList. getLength()-1) return fatherDecl;
            break;
        }
        d) i++;
```

4.3 简单类型值的类型检查

Schema 中的类型有简单和复杂之分. 简单元素和属性在 Schema 中有类似的声明结构,在 Xerces-J2 中将它们统一处理. 通过 impl. dv. xs. XSSimpleTypeDecl 的 validate( )可以实施对简单类型值的验证,主要工作包括约束面的检查(checkFacets( ))以及 ID/IDREF/ENTITY 等附加规则的检查(checkExtraRules( )). 从而,对简单类型值的类型检查代码为:

```
boolean checkType(String value, XSSimpleTypeDecl type){
  try{
    type. validate(frag, null, null);
    return true;
  }catch(InvalidDatatypeValueException e){ return false; }
}
```

4.4 复杂类型值的类型检查

对于复杂类型的元素,Xerces-J2 会根据其内容类型,建立相应的内容模型 (CM, content model). CM 相关的类封装在 impl. xs. models 包中(图 5). CMBuilder 为 CM 构建类,其 getContentModel(typeDecl)方法根据传入的复杂类型声明创建并返回相应的 CM. CM 分四类:

1) 若是简单类型或空的,则 CM 为空;

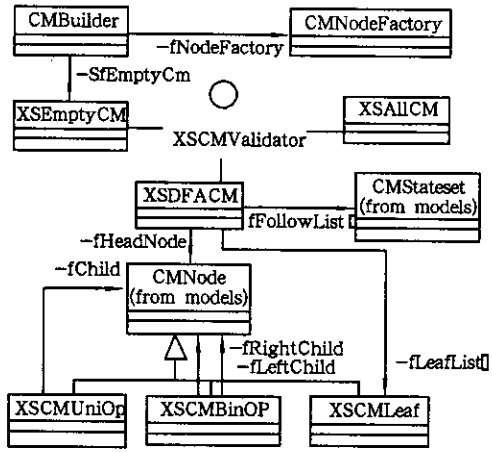

图 5 内容模型的类型

2) 若内容只有元素或是混合的且无 particle 组件(即元素声明、通配声明或模型组的 minOccurs 和 maxOccurs),则 CM 为 XSEmptyCM 实例,该实例全局唯一;

3) 若内容是 all 模型组,则 CM 为 XSAllCM 实例;

4) 否则 CM 为 XSDFACM 实例. 每类 CM 需实现 XSCMValidator 接口:startContentModel、checkUniqueParticleAttribution、oneTransition、whatCanGoHere、endContentModel.

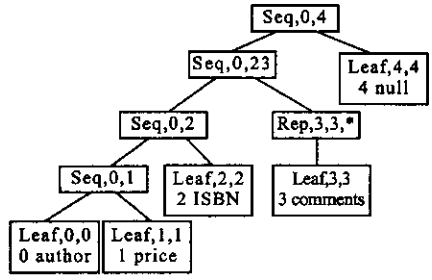

图 6 〈book〉的 CM 树

在三类 CM 中,XSDFACM 是核心且最为复杂. 它根据元素内容的 particle 树创建 CM 树并为该 CM 树建立 DFA 状态表. 不过一个 CM 树只反映元素内容当前一层的 particle 组件,至于子元素的 CM 则不予考虑. 图 6 为图 2 中 book 元素的 CM 树,结点矩形框中列出了结点的类型(Seq-顺序模型组、Choice-选择模型组、Any-通配、Rep-重复、Leaf-叶子)、对应的 DFA 初始状态、对应的 DFA 接受状态(若有多个通过空格分隔). 在叶子结点框中又额外标出了它在 DFA 中对应的状

态,以及它对应的子元素,如"0 author".

显然 Xerces-J 在 CM 树构造中,引入 Seq/Choice 模型组结点将模型组转化为二叉树,另外将+和?的重复次数处理统一成 *,即 a+——→aa*, a? ——→a|ε.

这样,基于 DFS 遍历和上述的 CM 模型实现对复杂类型值的检查,相应的算法如下:

```
boolean checkType(Node node, XSComplexTypeDecl type){
    return checkDFS(type, node, null, null);
}
boolean checkDFS(XSObject fCurType, Node fCurNode,
        XSCMValidator fCurCM, int[]fCurCMState) {
1) if (fCurCM! =null)调用fCurCM.oneTransition( )进行面临
                    fCurNode 的状态转换得到元素或通配声明decl
2) fCurCM=null; fCurCMState=null;
3) 若decl 是通配声明(XSWildcardDecl),则return true;
4) if (decl! =null && decl instanceof XSElementDecl)
      fCurType = ((XSElementDecl)decl).fType
5) fCurCM = fCurType.getContentModel(fCMBuilder);
   fCurCMState = (fCurCM! =null)? fCurCM.startContentModel
                                          ():null;
6) 取得fCurType 的属性组,对fCurNode 的各个属性进行简单类型
                                  检查,若检查不通过则return false;
7) 依次取得fCurNode 的各个子元素结点childNode
   a) 根据childNode 的tag 名,取得子元素的类型至childType
   b) if (! checkDFS(childType, childNode, fCurCM,
                                  fCurCMState)) return true;
8) 检查元素文本内容的有效性 //参见XMLSchemaValidator.
                                  processElementContent( )
9) 返回验证结果
}
```

4.5　值的 ID 约束检查

为进行ID约束检查,Xerces-J引入XMLSchemaValidator.ValueStoreBase 及子类(KeyValueStore、UniqueValueStore 和 KeyRefValueStore)管理一个 ID 约束的值存储,其值的添加通过addValue()进行,它检查值的唯一性. XMLSchemaValidator $ ValueStoreCache 负责管理解析中遇到的多个约束:当一个约束首次遇到时,其值存储将被存到 cache 中的局部哈希表 fIdentityConstraint2ValueStoreMap 内;一旦它被验证是有效的(即当走出作用域并通过 key 或 unique 约束检查),则将该值存储并入到 cache 中的全局哈希表 fGlobalIDConstraintMap 中;这样当遇到 keyref 约束时,只要在fGlobalIDConstraintMap 中检验key 引用的有效性即可. 这样 ID 约束检查的主要过程可设计为:

1) 在 3.4 中的checkDFS( )的6)步后激活相关的 ID 约束:

a) 调用 XSElementDecl 实例的getIDConstraints( )获取当前的 ID 约束数组 IDCs [];

b) 利用 Xalan 包的 XPathAPI 类提供的方法查找整个 XML 文档中与 ID 约束的Selector 有关的结点selectorNodes;

c) 依次为各 ID 约束 IDCs [] 建立值存储,将相关的selectorNodes 值添加到其中.

2) 在checkDFS( )的8)步后增加对ID 约束的检查,先检查key 和 unique,再检查 keyref.

5　结束语

由于 Xerces-J2 本身的验证体系庞大而复杂,并且类之间的耦合性太强,因此目前完成的局部验证工作比较粗糙,今后仍需要进一步优化上述验证算法,扩展算法中容许处理的输入范围,如支持更一般的 XPath 式.

References:

[1] Fallside D C. XML schema part 0: primer, W3C recommendation: REC-xmlschema-0-20010502, 2001 [EB/OL]. http://www.w3.org/TR/2001/REC-xmlschema-0/.

[2] Thompson H S, Beech D, Maloney M et al. XML schema part 1: structures, W3C recommendation: REC-xmlschema-1-20010502 2001 [EB/OL]. http://www.w3.org/TR/2001/REC-xmlschema-1/.

[3] Biron P V, Malhotra A. XML schema part 2: datatypes, W3C recommendation: REC-xmlschema-2-20010502, 2001 [EB/OL]. http://www.w3.org/TR/xmlschema-2/.

[4] http://xml.apache.org.