基于 SAX 模型的 XML 文档更新算法研究*

倪德明, 谭 帅, 潘志宇
(中山大学 计算机软件研究所, 广东 广州 510275)

摘 要: XML 作为交换资料的标准, 广泛应用于分布式系统中, 但是在使用 XML DOM 来处理大型 XML 文件时, 会占用过多的存储器资源并需要更多的 CPU 时间。提出一种基于 SAX 模型的 XML 文档更新算法, 利用 SAX 模型占用时间和空间少的优点进行文档的更新。形式化地定义了 XML 文档的更新脚本, 使用 X-Path 来表示所更新的文档节点, 在 XML 文档的 SAX 扫描过程中, 生成 XML 文档的节点的 X-Path, 从而判断文档节点是否需要更新, 然后通过将更新脚本应用到需更新文档实现文档更新。

关键词: SAX; XML diff; X-path
中图分类号: TP31 文献标识码: A 文章编号: 0529-6579 (2005) S2-0111-05

XML DOM 是常用的处理 XML 文件的标准方法, 使用 DOM 时, XML 资料以类树结构被装入内存中, 树上的每一个节点对应 DOM 文件中的一个元素。DOM 作为处理 XML 的传统方法, 具有处理模型简单直观, 可以随时修改和遍历 XML 文件的优点。但是这种方法需要读取整个 XML 文件并将它的文件结构解析出来存储到树结构中, 构造树结构的效率不高、速度缓慢, 并且会占用太多存储器资源[1]。

SAX (Simple API for XML, XML 简单编程接口) 是现在较流行的另一种方法, 它是基于事件的 API, 以流的方式来解析 XML 文件, 处理 XML 只需一遍扫描。SAX 的目的是提供一种更自然的方法来使用 XML, 解决 DOM 占用资源太多的缺点。

目前对于 XML 存储的大数据文件的更新, 大多数是采用 DOM 的方式实现的。采用这种方法存在一些缺点。一方面, DOM 需要占用太多的空间, 另一方面, 也需要更多的时间。特别是在分布式计算多源数据共享的情况下, 对 XML 文件的更新资料所占用的内存开销和网络带宽是用户无法承受的。[1]

本文定义了对 XML 文档的更新脚本, 使用 X-Path 来表示所要更新的文档节点。在 XML 文档的 SAX 扫描过程中, 使用一种文档堆栈结构来跟踪所扫描的 XML 文档节点, 并使用父元素堆栈[4] 和兄弟元素堆栈[5] 来跟踪父子关系和兄弟关系, 并生成文档节点的相应的 X—Path。然后通过将生成的 X-Path 与更新脚本中的 X-Path 相比较, 判断该节点是否需要更新, 从而对 XML 文档进行更新。采用 XML SAX 的方式, 实现对数据文件的动态更新, 在时间和空间上取得比较好的效果。

1 XML 文档更新脚本形式定义

1.1 X-Path 在更新脚本中的应用

XML 文档的结构是树形结构, X-Path 是对 XML 文档进行树查询的方式。在更新脚本中使用 X—Path 来表示节点, 并在 SAX 扫描中用 X-Path 来定位节点, 可以使得更新脚本的表示更加的通用, 简洁, 提高给外部统一的接口[3]。

更新算法是根据更新脚本中 XML 文档的更新操作而在 SAX 事件流中对文档结点进行更新的, 这就要求在更新脚本中, 更新操作所作用的文档结点不能够重复出现, 即在同一个更新脚本中, 不能对同一个 XML 文档结点进行两次不同的更新操作。

XML 的 SAX 模型是对 XML 文档节点进行扫描的, 是对需更新文档的节点进行的扫描。这样, 每当有节点被更新时, 被更新的节点就被写入另一个 XML 文档流中, 在对需更新的 XML 文档节点进行的扫描不能知道被更新节点。因此, 对于新更新的节点来说, XML 文档 SAX 处理模型不能处理被更新后的节点。

设 DOC 是需更新的 XML 文档, node1, node2, …, noden 是该 XML 文档中的所有文档节点, 函数 PathOfNode (node) 是计算某一文档节点 XPath 的函数, 存在更新脚本 UpdateScript {<path1, param>, <…>, …, <pathm, paramm>} 是对文档

* 收稿日期: 2005—09—07
作者简介: 倪德明 (1964 年生), 男, 博士, 副教授; 通讯联系人: 潘志宇; E-mail: zhiyupan@163.com

DOC 所要进行的更新操作，则：

任意 pathi（0＜i＜m），pathi ∈{PathOfNode（nodej），1＜j＜n}，并且在更新脚本中，对于任意的 pathi，不存在 pathi＝pathj（1＜j＜m，i≠j）。

1.2 更新脚本的形式定义

在文献[2]中，给出了对 XML 文档进行更新的更新语言的定义，该更新语言是针对 XML 文档的 DOM 模型进行定义的。本文结合该语言以及 SAX 模型的特点，来定义本算法的更新脚本。

XML 文档更新操作是对 XML 文档更新的基本操作。XML 文档更新编辑脚本是更新 XML 文档的基本操作系列，由一系列的 XML 文档更新基本操作组成。

下面将对 XML 文件更新的基本操作和更新编辑脚本定义如下：

定义：

更新编辑操作 EditOperation＝＜Target path，Process instruction（...）＞，其中，Target path 为要操作的 XML 文档的 Xpath 路径，Process instruction 为对该路径的操作。在本文中，使用 XML 文档格式来表示更新编辑操作，其格式如下：

```
＜EditOP：process instruction name＝"PI"＞
    ＜Target path＞X Path＜/Target path＞
    ＜element＞ update param＜/element＞
＜/EditOP：process instruction＞
```

该 XML 文档表示了更新命令的格式，name 为更新操作的名字，Target 是更新操作作用的目标路径，element 是更新操作的文档结点参数。

在更新编辑操作中，更新操作由 Insert，Update，Rename，Remove，等操作组成，分别定义如下：

Insert：将文档结点插入到 XML 文档中。在更新算法中，有两个插入操作 InsertBefore 和 InsertAfter，定义如下：

* InsertBefore（insertednode，path）表示插入结点 insertednode 到 path 所指的目标 XML 文档结点之前，作为目标结点的前序兄弟。

* InsertAfter（insertednode，path）表示插入结点 insertednode 到 path 所指的目标 XML 文档结点之后，作为目标结点的后序兄弟。

例如，有更新编辑操作，

```
＜EditOP：InsertAfter name＝"InsertAfter"＞
    ＜Target path＞ section[0] / subsection[0] /
      heading[0]＜/Target path＞
    ＜element＞ heading /"Title"＜/element＞
＜/EditOP：InsertAfter＞
```

表示生成一个结点＜heading＞ Title ＜/heading＞，然后插入该结点到第一个 section 的第一个 subsection 的第一个 heading 结点之后。如下列格式所示：

```
＜section＞                          ＜section＞
＜subsection＞                       ＜subsection＞
＜heading＞Abstract＜/heading＞      ＜heading＞Abstract＜/heading＞
＜/subsection＞                      ＜heading＞Title＜/heading＞
＜/section＞                         ＜/subsection＞
                                     ＜/section＞
```

插入前的 XML 文件　　插入后的 XML 文件

同理，对 XML 更新的其他操作定义如下：

Remove（path）：表示删除在更新操作的 Path 中所指 XML 文档位置的结点。

Update（newvalue，path）：将更新操作中的 Path 所指的 XML 文档位置的文档结点的值更新为 newvalue。

Rename（newname，path）：将 path 所指的一个属性或者元素结点名称重命名。Newname 为将新命名的结点或属性名称。

更新脚本：设 Oi 表示对目标 XML 文档的编辑操作，Oi＝EditOperation ＝＜path，Operation＞，编辑脚本就是一个编辑操作的系列，则编辑操作定义如下：

EditScript＝{O_1 O_2 ⋯O_i ⋯ O_n}

对于更新编辑脚本，使用 XML 的格式定义更新编辑脚本如下：

```
＜UpdateScript＞
    ＜XMLupdate：process instruction name＝""＞
        ＜target path＞ X Path    ＜/target path＞
        ＜element＞ update param＜/element＞
    ＜/XMLupdate：process instruction＞
    .....
＜/UpdateScript＞
```

2 XML 更新算法

当源 XML 文档变化时，生成相应的变化编辑脚本，传送到需要同步的目标 XML 文档，需要更新的目标 XML 文档收到变化编辑脚本后，采用本更新算法进行更新。根据得到的编辑脚本，基于 SAX 模型来完成从一个 SAX 流读出 XML 文档，将算法应用在该流上，生成另一 XML 流，实现对 XML 文档的更新（图 1）。

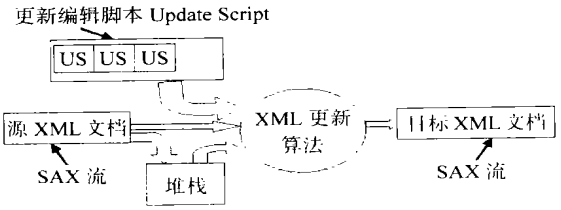

图 1　XML 文档更新算法示意图

Fig. 1　Illustration of the updating algorithm for XML document

2.1　文档节点 X-Path 生成

更新脚本中，使用 X-Path 表示所更新的文档节点，而在 XML 文档的 SAX 扫描中，只知道扫描节点在 XML 文档中的次序索引(DOI)[4]，如图 2 中，SAX 扫描中能够知道节点在文档中的次序索引，而 X-Path 是对文档节点的路径的表示。这样，就需要在 XML 文档的 SAX 扫描中，生成文档节点的 X-Path，并使用 X-Path 来判断该节点是否需要更新。

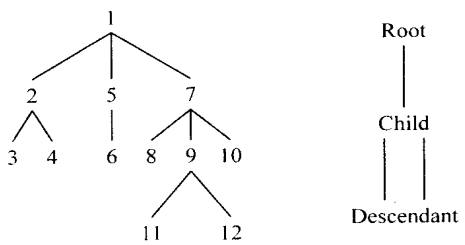

图 2　XML 文档 SAX 扫描中的节点次序及节点的 XPath 表示

Fig. 2　The ordering and xpath presentation of nodes in XML SAX scanning

XML 文档是一种树形结构，要生成文档的 X-Path 表示就需要知道文档节点在树结构中的位置，为此，使用父元素堆栈结构来跟踪父子关系[5]，使用兄弟元素结构来跟踪兄弟关系[5]，并用链表来保存 SAX 扫描过的文档节点的树形结构。通过查询该链表，就能够计算出节点的 X-Path。通过对 SAX 事件 startDocument、startElement 和 endElement 进行处理。在 startDocument 中，初始化文档次序计数器、父元素节点堆栈、兄弟元素节点堆栈。在 startElement 中，从父元素节点堆栈中得到正在处理的 XML 文档节点的父结点，生成正在处理节点的父亲和兄弟节点，把当前节点的节点次序、名字、父元素、兄弟元素添加到 XML 文档树的列表中，然后，将父元素和兄弟元素压入堆栈。在 endElement 中，就需要将父元素堆栈和兄弟元素堆栈的栈顶元素弹出。

2.2　更新算法流程

算法采用堆栈来记录在 XML 文档的扫描中遇到的文档结点，在扫描过程中，将遇到的新结点入堆栈[6]，处理完该结点后，将该结点出栈，该堆栈的内容为算法正在扫描的结点。XML 文档结点是由结点开始标示(<)和文档结束标示(/>)表示的，在 SAX 扫描中，当遇到文档开始标示时，可以知道开始扫描新结点，在后续的处理中，所处理的内容都是该结点，直到遇到该结点的结束标示，就可以知道对该结点的处理结束[4]。本算法中，利用 SAX 的这种特点，在 XML 文档的 SAX 流中，当遇到文档结点的开始标示时，将文档结点入栈，在遇到文档结点的结束标示时，将该结点的所有内容(包括子结点)出栈。这样在 SAX 流的处理过程中，堆栈中保存的总是当前处理的结点，于是便可以对 SAX 处理结点进行跟踪。

堆栈定义：

```
TYPE ElementType;
Typedef struct ElementStack {
        ElementType element;
        ElementStack *next;   }
```

算法名称：UpdateXML (SAXXML DOC, UpdateScript ES)

算法名称为 UpdateXML，DOC 为 XML 文档的 SAX 流，ES 为编辑脚本

输入参数：XML 文档编辑脚本 ES，待更新 XML 文档 DOC

输出结果：更新后的 XML 文档 (SAX 流) DOC1

算法功能：根据更新编辑脚本，更新 XML 文档 (对于插入，暂只给出插入后序兄弟的情况)，将更新后的 XML 文档写入另一 XML 文档中

子过程：

PushNodeToStack (Node)：将文档结点入堆栈；

POP ()：将堆栈中的栈顶元素弹出；

GetStackTop ()：得到栈顶元素的函数；

IsEmpty (ElementStack)：判断堆栈是否为空；

PathHash (string)：计算字符串 hash 值；

UpdatePath (node)：更新全局路径。

算法逻辑结构:

```
/*定义全局堆栈 GlobalStack 并设置其为 NULL; */
ElementStack GlobalStack=NULL;
/*文档结点是否被更新的标志 */
Int isUpdated = 0;
/*设定全局路径变量 P 为 NULL;  */
XPath P=NULL;
/*设定编辑脚本的 XPath 路径变量 ESPath 为 NULL;  */
XPath ESPath = Null;
/*第一个文档结点元素进堆栈 */
PushNodeToStack (FirstNode);
/*当文档结点堆栈不是空的时候, 一直进行处理 */
while (!IsEmpty (GlobalStack))
{
        e=GetStackTop ();
        /*如果栈顶元素是文档结点开始标记, 则将 e 入栈 */
        if (e 是开始元素)
        {
        /*将 e 加入全局 P 中, 入栈;  */
            PushNodeToStack (e);
            UpdatePath (e);
            /*在编辑缓冲区中, 查找该结点是否需要更新 */
            while (编辑脚本缓冲区不空时)
              {
               ESPath = 更新编辑操作中的 XPath;
               /*如果在 SAX 处理中, 当前处理的文档结点与更新操作作用的 XPath 一致, 就将
               更新脚本中的更新操作 Operation 应用到相应的路径 P 所指向的 XML 文档中。 */
              if (PathHash(GlobalStack)==PathHash(ESPath))
              {
                若 Operation 是 Insert, 则将编辑脚本中的元素结点写入到另一 XML 的 SAX 流中;
                若 Operation 是 Update, 则用编辑脚本中的元素更新 e 后, 再将其点写入到另一
                XML 的 SAX 流中;
                若 Operation 是 Remove, 则对文档节点不作任何处理, 返回下一个循环;
                若 Operation 是 Rename, 则更改节点的名称, 并将该节点下的子节点写入更改了名
                称的节点下;
                /*标示文档结点被更新 */
                IsUpdated = 1;
                Break;
        }
        Else
            {
                将文档节点写到另一个 XML 文档流中。
             }
    }
}
        /*如果栈顶元素是文档结点结束标记, 则将 e 出栈 */
```

```
    if (e 是结束元素) 出栈;
        /*如果该文档结点不需要更新，则将该结点直接写入另一 XML 文档中 */
    If (isUpdated) 将结点 e 写入另一 XML 文档;
}
```

3　算法复杂度分析

时间复杂度上，本算法主要是在扫描 SAX 的 XML 文档的过程中，对 XML 文档的动态更新，算法仅需要扫描一遍 XML 文档便能实现更新。设 XML 文档的结点数为 n，时间复杂度为 $O(n)$。

在空间复杂度上，采用将 XML 文档 SAX 流写到另一 XML 文档 SAX 流，占用较少内存。设 XML 文档的结点数为 n，处理过程中需存储正在处理的节点、父元素节点及兄弟元素节点，故空间复杂度为 $O(1)$。使用 DOM 方式进行 XML 文档更新时，无论 XML 文档的大小，都需要将整个 XML 文档读入内存，空间负杂度为 $O(n)$。

因此，在时间复杂度方面与使用 DOM 模型进行处理相当，而在空间复杂度方面，本算法有了显著的提高。

4　结　论

本文定义了对 XML 文档的更新脚本，并在更新脚本中使用 X-Path 来表示 XML 文档节点，在 XML 文档的 SAX 扫描过程中，采用文档节点堆栈来跟踪 SAX 扫描的节点，父元素节点堆栈来跟踪节点间的父子关系，兄弟节点堆栈来跟踪兄弟关系，并使用一个内部链表来存储 XML 文档的树形结构，使得在 XML 文档的 SAX 扫描过程能够计算出所扫描节点 X-Path，判断该节点是否需要更新，进而进行更新操作。使用 SAX 模型与比使用 DOM 模型相比，占用比较少的时间和空间，在一定程度上提高了算法的性能。

因为 SAX 的一遍扫描的特点，决定了算法中的更新脚本所使用的 X-Path 是受约束的。SAX 模型对文档节点只能进行一遍扫描，故更新脚本中也只能对每个节点进行一次更新，这样，更新脚本中的 XPath 就最多只能出现一次。

参考文献：

[1] BRAY T, PAOLI J, SPERBERG-MCQUEEN C M, et al. Extensible Markup Language(XML) 1.0(3rd edition)[EB/OL]. http://www.w3.org/TR/2004/REC-xml-20040204/#sec-well-formed, 2004-02.

[2] LAUX A, MARTIN L. XUpdate-XML update language[EB/OL]. http://www.xmldb.org/xupdate/xupdate-wd.html, 2000-09-14.

[3] CLARK J, DEROSE S. XML Path Language (XPath) Version 1.0[EB/OL]. http://www.w3.org/TR/xpath, 1999-10.

[4] KATZ H. Tip: SAX and document order-track sibling relationships [EB/OL]. http://www-900.cn.ibm.com/developerWorks/cn/xml/tips/x-tipsaxdo3/index-eng.shtml, 2003-02.

[5] KATZ H. Tip: SAX and document order-track parent-child relationships [EB/OL]. http://www-900.cn.ibm.com/developerWorks/cn/xml/tips/x-tipsaxdo2/index-eng.shtml, 2003-02.

[6] 严蔚敏, 吴伟民. 数据结构. 北京: 清华大学出版社, 1998.

Research of XML Document Updating Algorithm Based on SAX Model

NI De-ming, TAN Shuai, PAN Zhi-yu

(Software Research Institute, Sun Yat-sen University, Guangzhou 510275, China)

Abstract: The eXtensible Markup Language (XML) is becoming the new standard for data exchange over the Internet. But updating large XML document will take very much memory and time. This paper proposes an XML update algorithm based on SAX model, which can decrease memory and time take for updating XML document. Define the update script for XML document, and use X-Path to locate XML document node. While XML SAX scanning, it generates X-Path of document node and uses the X-Path to judge whether the node needs to be updated. The algorithm will apply update script to the destination XML document to updated XML document dynamically.

Key words: SAX; XML diff; X-path