(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103488639 B
(45)授权公告日 2016.12.07

(21)申请号 201210192018.7

(22)申请日 2012.06.11

(65)同一申请的已公布的文献号
申请公布号 CN 103488639 A

(43)申请公布日 2014.01.01

(73)专利权人 北京大学
地址 100871 北京市海淀区颐和园路5号

(72)发明人 郭少松 包小源 陈薇 王腾蛟 杨冬青

(74)专利代理机构 北京君尚知识产权代理事务所(普通合伙) 11200
代理人 余长江

(51)Int.Cl.
G06F 17/30(2006.01)

(56)对比文件
US 2002120598 A1,2002.08.29,
CN 101010674 A,2007.08.01,
CN 1584884 A,2005.02.23,

审查员 王亮

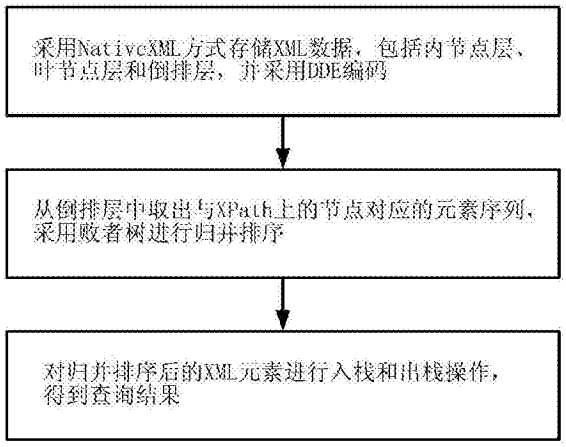
权利要求书1页 说明书9页 附图4页

(54)发明名称

一种XML数据的查询方法

(57)摘要

本发明提供一种XML数据的查询方法，其步骤包括：1)采用Native XML方式存储XML数据，其存储结构包括：内节点层，存储XML树的节点，XML元素采用DDE编码方式进行编码；叶节点层，存储XML树叶节点的文本数据；倒排层，存储内节点层的倒排索引；2)根据输入的XPath查询语句，从所述倒排层中取出与所述XPath的节点对应的元素序列，并采用败者树进行归并排序；3)对归并排序后的XML元素按顺序进行入栈和出栈操作，从缓冲区得到查询结果。本发明能够处理带有关键字“OR”和通配符“*”的XPath，并具有很高的效率。



CN 103488639 B

1.一种XML数据的查询方法，其步骤包括：

1)采用Native XML方式存储XML数据，其存储结构包括：内节点层，存储按照文档序排列的XML树的节点，其中XML元素采用DDE编码方式进行编码；叶节点层，存储XML树叶节点的文本数据；倒排层，存储内节点层的倒排索引，每个索引项是标签名字相同的元素按照文档序排列成的序列；

2)根据输入的XPath查询语句，从所述倒排层中取出与所述XPath的节点对应的元素序列，并采用败者树进行归并排序；所述采用败者树进行归并排序，是对两个元素的DDE编码进行比较，得到所述两个元素的前后关系，并设定在前的元素为胜者，在后的元素为败者；当XPath中出现通配符“*”时，引申出三种新轴：祖父子轴、绝对祖先后代轴、特殊祖先后代轴，采用所述三种新轴对含有通配符“*”的XPath进行等值改写；

3)对归并排序后的XML元素按顺序进行入栈和出栈操作，并从缓冲区得到查询结果。

2.如权利要求1所述的方法，其特征在于，所述内节点层中每条记录的信息包括：由节点名字映射成的整数标识符、DDE编码和节点类型。

3.如权利要求1所述的方法，其特征在于，所述倒排层中每个元素的信息包括：元素类型、该元素在内节点层的地址和DDE编码。

4.如权利要求1所述的方法，其特征在于，所述内节点层通过指针指向所述叶节点层。

5.如权利要求1所述的方法，其特征在于：所述XPath中每个节点q有两个数据结构：元素序列T_q和栈S_q；T_q是XML文档中与q名字匹配的所有元素，并按照文档序排列；S_q用于存储与q名字匹配的元素，并进行入栈和出栈操作。

6.如权利要求1所述的方法，其特征在于，在入栈操作时，栈中只保留新元素的祖先，栈中的所有元素都是祖先后代关系。

7.如权利要求6所述的方法，其特征在于，若元素e要入栈S_E，在XPath上节点E的父节点是A，则元素e入栈的判断条件是：

a)S_A中有未出链的元素；所述出链是指将不是e的祖先的记录从连接栈中所有元素的链表中删除；

b)e是S_A中未出链且最靠近栈顶的元素的孩子；

c)e的类型与XPath上E的类型相同。

一种XML数据的查询方法

技术领域

[0001] 本发明属于数据库技术领域，涉及半结构化数据XML的存储与查询方法，具体涉及一种能有效支持XML查询语言XPath的XML数据查询方法。

背景技术

[0002] 由于越来越多的应用系统采用XML作为标准格式来发布和交换数据，XML数据的规模急剧扩大，在IDC(互联网数据中心)最近发布的一份报告显示，500家受访企业的IT部门中有29%正在大量使用XML文档和XML数据库。如何有效管理XML数据成为迫切需要解决的问题。

[0003] 快速准确查找XPath在XML数据库中的所有匹配元素，是XML查询处理的核心操作。例如，一个XPath表达式：book[title='XML']//author[fn='Jane'AND ln='Doe']，这个表达式匹配的节点author需要满足：1)有一个子节点fn，它的内容是'Jane'；2)有一个子节点ln，它的内容是'Doe'；3)是book节点的后代，且book节点有一个内容为'XML'的title子节点。

[0004] XML模式匹配方法中较典型的有DB2开发的针对XML数据流的TurboXPath算法和2002年学术界提出的TwigStack算法。

[0005] TwigStack算法中，XPath上的每个节点q都对应着一个Tq和Sq。Tq代表元素序列，q是XPath上的标签名字，Tq是XML文档中与q名字匹配的所有元素，而且Tq中的元素按照文档序排列。Sq代表元素栈，存储与q名字匹配的元素，当算法正在处理的元素已经越过栈中元素的闭标签时，栈中元素要出栈。算法只对Tq中元素操作，跳过无关的XML元素，所以算法的IO效率很高。但是TwigStack算法不能处理两种情况：第一是有通配符“*”的XPath，例如//a/*[b]/c，因为TwigStack算法使用的是区间编码，即使有元素a与元素b和c的层次相差2，但也无法确定元素b和c是否有相同的父亲；第二是TwigStack算法只能处理twig之间是'AND'关系的XPath，例如//a[bAND c]/d，却不能处理有关键字'OR'的XPath，例如//a[b OR c]/d。

[0006] TurboXPath算法是DB2采用的对XML流(XML stream)的查询匹配算法，既没有索引，也没有编码，XML流中的XML元素按照文档序排列，可以方便地处理有关键字'OR'的XPath。TurboXpath功能较健全，但是对于数据库中的XML数据，TurboXPath算法从头到尾扫描XML文档，IO代价很大，尤其对于那些较大的XML文档。

发明内容

[0007] 本发明的目的是针对现有技术中的问题，提供一种新的XML数据的查询方法，能够处理带有关键字“OR”和通配符“*”的XPath，并具有很高的效率。

[0008] 为实现上述目的，本发明采用如下技术方案：

[0009] 一种XML数据的查询方法，其步骤包括：

[0010] 1)采用Native XML方式存储XML数据，其存储结构包括：内节点层，存储按照文档

序排列的XML树的节点，其中XML元素采用DDE编码方式进行编码；叶节点层，存储XML树叶节点的文本数据；倒排层，存储内节点层的倒排索引，每个索引项是标签名字相同的元素按照文档序排列成的序列；

[0011] 2)根据输入的XPath查询语句，从所述倒排层中取出与所述XPath的节点对应的元素序列，并采用败者树进行归并排序；

[0012] 3)对归并排序后的XML元素按顺序进行入栈和出栈操作，并从缓冲区得到查询结果。

[0013] 进一步地，所述内节点层中每条记录的信息包括：由节点名字映射成的整数标识符、DDE编码和节点类型。

[0014] 进一步地，所述倒排层中每个元素的信息包括：元素类型、该元素在内节点层的地址和DDE编码。

[0015] 进一步地，所述内节点层通过指针指向所述叶节点层。

[0016] 进一步地，所述采用败者树进行归并排序，是对两个元素的DDE编码进行比较，得到所述两个元素的前后关系，并设定在前的元素为胜者，在后的元素为败者。

[0017] 进一步地，所述XPath中每个节点q有两个数据结构：元素序列Tq和栈Sq；Tq是XML文档中与q名字匹配的所有元素，并按照文档序排列；Sq用于存储与q名字匹配的元素，并进行入栈和出栈操作。

[0018] 进一步地，在入栈操作时，栈中只保留新元素的祖先，栈中的所有元素都是祖先后代关系。

[0019] 进一步地，若元素e要入栈SE，在XPath上节点E的父节点是A，则元素e入栈的判断条件是：

[0020] a)SA中有未出链的元素；所述出链是指将不是e的祖先的记录从连接栈中所有元素的链表中删除；

[0021] b)e是SA中未出链且最靠近栈顶的元素的孩子；

[0022] c)e的类型与XPath上E的类型相同。

[0023] 进一步地，当XPath中出现通配符“*”时，引申出三种新轴：祖父子轴、绝对祖先后代轴、特殊祖先后代轴，并采用所述三种新轴对含有通配符“*”的XPath进行等值改写。

[0024] 本发明的XML数据查询方法，解决了TwigStack方法不能支持带有关键字“OR”和通配符“*”的XPath问题；对于数据库中XML数据的查询处理，具有与TwigStack方法同样的IO效率，且比TurboXPath方法的效率更高。目前，越来越多的应用系统采用XML作为标准格式来发布和交换数据，XML数据的规模急剧扩大，金融、医疗、电子政务、新闻等领域已经采用各自制定的XML标准来实现不同部门、不同企业之间的数据交换，本发明方法可广泛应用于这些领域，高效地实现对XML数据的有效查询和管理。

附图说明

[0025] 图1是本发明实施例的XML数据查询方法的步骤流程图。

[0026] 图2是本发明实施例的Native XML存储方式示意图。

[0027] 图3是图2中内节点层的示意图。

[0028] 图4是本发明实施例中对//a[//c]/b的查询流程图。

[0029] 图5是本发明实施例中对//a[//c]/b进行查询的的入栈出栈操作示意图。

[0030] 图6是本发明实施例中对//a/*[c]/b进行查询的的入栈出栈操作示意图。

具体实施方式

[0031] 下面通过具体实施例，并配合附图，对本发明做详细的说明。

[0032] 图1是本发明的XML数据查询方法的流程图，具体步骤包括：

[0033] 1)采用Native XML方式存储数据库中的XML数据。

[0034] 本发明的XML数据查询方法属于整体小枝连接方法，与早期结构化连接相比，整体小枝连接技术可以避免大量无效的中间结果。本发明方法的基础是Native XML存储，对XML元素采用DDE编码方式。Native存储机制维持了XML元素的文档序，通过一个元素的开标签物理地址就可取出以该元素为根的子文档。DDE编码用来对XML元素的常见结构关系(祖先后代，父子，兄弟等)进行判断。

[0035] 本发明的存储设计分为三层：内节点层、叶节点层和倒排层，如图2所示。

[0036] a)内节点层

[0037] 把XML树的节点按照文档序排列，存储在内节点层。该层的每条记录是一个XML树节点，每条记录的信息包括由节点名字映射成的整数标识符tagID(方便存储，方便比较)、DDE编码、节点类型(元素、属性、文本)等。图3是一个简单的内节点层的实例，其中，(a)为XML树；(b)为与(a)对应的顺序存储，以“/”开头的为闭标签记录；“Database”和“25.00”两个叶节点在这里只是指针，实际内容存储在叶节点层。

[0038] XML编码是用来判断XML元素之间的结构关系的。TwigStack算法不能处理带有通配符“*”的XPath，因为它采用的区间编码不能判断兄弟轴。本发明采用DDE编码，DDE编码比区间编码的好处有：

[0039] 区间编码能判定的轴DDE都能判定，而且DDE还能判定兄弟轴，区间编码却不能；

[0040] DDE编码能很好地支持XML文档的更新，即当XML文档变化时，原有的编码不需更改，区间编码做不到。

[0041] b)叶节点层

[0042] 每条记录存储一个叶节点的文本数据，存储XML树叶节点的文本数据。内节点层有指针指向这里，通过这些指针找到文本数据所在的物理页。

[0043] c)倒排层

[0044] 倒排层类似于IR系统中的倒排索引。倒排层中所有标签名字相同的元素组成一个序列，并按照文档序排列。序列中每个元素的信息有：它在内节点层的地址、元素类型、DDE编码等。根据倒排层存储的信息就可以完成对XPath的查询匹配，根据查询到的元素地址再去内节点层获取元素开闭标签之间的子文档。在图2所示的倒排层中，E1、E2、E3是表示XML元素信息。

[0045] 2)根据输入的XPath查询语句，从倒排层中取出与XPath上的节点对应的元素序列，并采用败者树进行归并排序。

[0046] 对于XPath上的每一个节点q，有两个数据结构：元素序列T_q和栈S_q。T_q是XML文档中与q名字匹配的所有元素，而且T_q中的元素按照文档序排列。S_q在算法进行过程中存储与q名字匹配的元素，一个新元素入栈，那些非其祖先的元素就会出栈。

[0047] 本发明的方法可以称为“TurboStack”方法。假设XPath有n个节点：q_1，q_2，……，q_n，与每个节点对应的T_{qi}($1 \leqq i \leqq n$)从倒排层获取，T_{qi}中的XML元素已经是有序了。TurboStack方法的输入是按照文档序排列的XML元素，因此需要对T_{q1}，T_{q2}，……，T_{qn}这n个元素序列进行归并排序，把n个序列合并成一个按照文档序排列的序列，作为算法的输入。

[0048] 可以依靠DDE编码并采用败者树来进行归并排序：对于两个元素的编码dde1和dde2，用这两个编码进行比较，可以判断出两个元素的前后关系，并设定在前的元素为胜者，在后的元素为败者。

[0049] 3)对于归并排序后的元素，按顺序进行入栈和出栈操作，从缓冲区得到查询结果。

[0050] 下面所示为本发明整体方法的执行流程，记为算法1，对其中的函数说明如下：constructStack(q)为节点q建立栈，GetStream(q)从底层存储中获取节点q的元素序列，MultiMergeSort(XPath)对XPath上所有节点的T_q进行归并排序，getPopElement(e)从栈中选取不是e的祖先的元素，match(e)判断e是否能入栈。

Algorithm 1. TurboStack

Input：XPath

Output：outputBuffer

[0051]

```
1. FOREACH q IN XPath
2.     Sq = constructStack(q)
3.     Tq = GetStream(q)
4. elementsList = MultiMergeSort( XPath )
5. outputBuffer = NULL;
6. FOREACH e IN elementsList
7.     WHILE( E = getPopElement(e) != NULL)
8.         PopStack(E, SE, outputBuffer)
9.     IF (match(e) == true)
10.        PushStack(e, Se, outputBuffer)
11. RETURN outputBuffer
```

[0052] 下面对入栈和出栈进行具体说明。

[0053] 3.1)入栈

[0054] 对所有T_q进行归并排序，使元素按照文档序排列后，就可以依次准备入栈。元素e要入栈，相当于在XML文档里遇到了e的开标签。栈中e的祖先仍然留在栈里，原因在于按照XML树结构，作为e的祖先结点的闭标签这时一定没有扫描到。而那些先于e入栈却并非e的祖先的元素，它们的闭标签已经过了，应该出栈。

[0055] 每次入栈都只保留新元素的祖先，栈中的所有元素都是祖先后代关系。把所有栈中的元素用链表连接起来，称这个链表为Last Push List，简写为LPL。新元素e入栈后，置于LPL的头部位置。一个新元素e在入栈之前，从LPL头开始依次检查每一个记录E_i，用DDE编

码将E_i与e比较，若E_i不是e的祖先，从LPL中删除E_i，称作出链，否则停止比较。出链标志着该元素已经出栈，但并没有真正从栈中移除。

[0056] 若e要入栈S_E，在XPath上E的上层节点是A，算法1中match(e)的判断条件是：

[0057] 1.S_A中还有未出链的元素；

[0058] 2.若A和E之间的轴是父子轴，S_A中未出链且最靠近栈顶的元素是a1，那么e必须是a1的孩子；

[0059] 3.e的类型与XPath上E的类型相同(E的类型可能是元素节点或者属性节点)。

[0060] 若E是XPath的根节点，则只要满足第三个条件即可入栈。

[0061] 入栈的新记录包含四个信息：

[0062] 1.元素信息(tagID,DDE,type)；

[0063] 2.指针P_{LPL}，指向LPL下一个记录；

[0064] 3.指针P_{stack}，指向S_A中还在LPL上且最靠近栈顶的记录。

[0065] 4.匹配状态位status，起初为false。若该记录也满足XPath对它的结构要求，则置为true。

[0066] 若E是XPath的叶节点，则status状态位初始化为true。

[0067] 若E是XPath的输出节点，把e放在结果缓冲区outputBuffer里。

[0068] 下面所示为入栈的执行流程，记为算法2，对其中的函数说明如下：

[0069] Push(e,S_E)把元素e放入S_E栈顶，Lappend_head(e,LPL)把元素e置于链表LPL的头部位置，Lappend(e,outputBuffer)把元素e放入输出缓冲区outputBuffer中。

Algorithm 2. PushStack

Input: e, S_E, outputBuffer

output: outputBuffer

1. Push(e, S_E)

2. Lappend_head(e , LPL) //作为头结点

[0070]

3. IF node E is the output node in XPath

4. Lappend (e, outputBuffer)

5. IF node E is the leaf node in XPath

6. e->status = true

7. RETURN outputBuffer

[0071] 3.2)出栈

[0072] 栈里的记录出链时，把这个记录的P_{LPL}置为NULL，但是该记录并不出栈。

[0073] 在XPath上，除了叶节点，其他节点都有子节点，子节点的栈称为父节点的子栈。父节点栈里的元素出链时，子栈的元素出栈。

[0074] 节点A有两个子节点B和C，A与B是父子轴，A与C是祖先后代轴。假设现在栈S_A里有两个记录a1和a2，a2在栈顶，现在要把a2出链。记录a2并不出栈，出栈的是S_B和S_C里的记录。

[0075] 首先要判断记录a2是否满足XPath对查询节点A在结构上的要求，即下文的算法3

的函数matchStructure(e)的过程：

[0076] a)S_B中记录$b_1,b_2,\cdots\cdots,b_n$的P_{stack}指针指向a2，S_C中记录$c_1,c_2,\cdots\cdots,c_m$ P_{stack}指针也指向a2。这些记录已经出链，且它们的匹配状态位status在出链的时候已经求出。

[0077] b)若b和c是AND关系，则

[0078] a2->status=(b_1->status||.....||b_n->status)&&(c_1->status||.....||c_m->status)；

[0079] 若b和c之间是OR关系，则

[0080] a2->status=(b_1->status||.....||b_n->status)||(c_1->status||.....||c_m->status)。

[0081] 若n=0或m=0，即S_B或S_C中没有指向a2的记录，那么S_B或S_C的状态位作为false看待。

[0082] S_B中$b_1,b_2,\cdots\cdots,b_n$全部出栈，因为它们不可能是a1的孩子；S_C中$c_1,c_2,\cdots\cdots,c_m$状态位status为false的出栈，剩下的记录的P_{stack}指针都指向a1，因为它们也是a1的后代。

[0083] 此时若a2->status=false，那么output buffer里属于a2后代的记录从缓冲区里删除。

[0084] 当XPath的根节点root的T_{root}为空且S_{root}中的记录也都出链，算法停止，outputBuffer里的元素就是查询结果。

[0085] 下面框中所示为出栈的执行流程，记为算法3，对其中的函数说明如下：IsEmpty(LPL,S_E)判断S_E是否还有未出链的元素，若没有返回true；Delete_Stacks(S_E,e)把S_E的子栈中e的后代元素删除；Delete_InterResult(outputBuffer,e)从输出缓冲区中删除e的后代元素；stack_top(S_E)返回栈中未出链且最靠近栈顶的元素；childStack(S_E)返回S_E的所有子栈；descendants(S_C,e)返回栈S_C中属于e的后代的元素；PC(E,C)判断E和C是否为父子关系；AD(E,C)判断E和C是否为祖先后代关系。

Algorithm3. PopStack

Input: e, S_E, outputBuffer

output: outputBuffer

[0086]

```
1. e->status = matchStructure(e)
2. IF   e->status = false
3.      Delete_InterResult(outputBuffer,e)
4. List_delete( LPL,e)
5. IF   IsEmpty(LPL, S_E ) == true
```

```
6.      Delete_Stacks(c, e)
7. ELSE
8.      e' = stack_top(SE)
9.   FOREACH SC <- childStack(SE )
10.      FOREACH ce<-descendants(SC,e)
11.         IF (AD(E,C) == true)
12.             ce->Pstack = e'
13.         ELSE IF (PC(E,C) == true)
14.             Stack_delete(SC ,ce) //ce 出栈
15. RETURN outputBuffer
```

[0087]

[0088] 3.3)带有通配符“*”的XPath

[0089] XPath的常见轴有祖先后代(AD)轴、父子(PC)轴等，若XPath中出现通配符“*”，则引申出三种新轴：

[0090] a)祖父子(grand parent-child，即GPC)轴，例如a/*/c，a和c是隔了两层的祖父子关系；a/*/*/c，a和c是隔了三层的祖父子关系。本发明使用/^{n}表示GPC轴，n是整数，表示隔了几层。

[0091] b)绝对祖先后代(absolute ancestor/descendant，即AAD)轴，例如a/*//c或者a//*//c，a和c是至少隔了两层的绝对祖先后代关系。使用//^{n}表示AAD轴，n是整数，表示至少隔几层。

[0092] c)特殊祖先后代(special ancestor/descendant，即SAD)轴，例如a//*/c，a和c是至少隔了两层的特殊祖先后代关系。使用///^{n}表示SAD轴，n是整数，表示至少隔几层。

[0093] 为什么要区分AAD和SAD?例如a//*[//d]//c，a和d、c之间是AAD轴，d和c之间没有关系；对于a//*[d]/c，a和d、c之间是SAD轴，而且d和c是兄弟关系；对于a//*[d]//c，a和d之间是SAD轴，a和c之间是AAD轴，d和c之间没有关系。

[0094] GPC，AAD和SAD都是AD的特例，使用DDE编码能很容易地判定GPC，AAD和SAD，因为DDE编码有层次信息。

[0095] 用GPC、SAD和AAD三种轴对有通配符“*”的XPath进行等值改写。

[0096] 当XPath中出现“*”且它是分叉节点，例如a/*[d]/c//e，改写为a[/^{2}d]/^{2}c//e，因为d和c还必须满足兄弟关系，在处理三种新轴时要特别注意这种情况。

[0097] 新元素入栈，若要入的栈是S_b，在XPath上b的父节点是a，a和b是GPC，SAD和AAD三种轴中的一种。入栈要满足的条件是：

[0098] a)S_a中必须有未出链的元素；

[0099] b)S_a中存在某元素与新元素满足轴所要求的层次关系。

[0100] c)新元素类型符合b的要求。

[0101] 若a与b、c都是/^{n}或者///^{n}轴，且n相等，那么b和c需要满足兄弟关系。假设现在元素a1出链，b_1，b_2，……，b_n和c_1，c_2，……，c_m是a1的后代，计算a1是否匹配，首先要兄弟配对，例

如(b_1,c_1,c_2),(b_2,b_3,c_3,c_4)……括号中的元素都是兄弟,则:

[0102] 若b和c之间是AND关系,则a1->status=[b_1->status&&(c_1->status||c_2->status)]|[(b_2->status||b_3->status)&&(c_3->status||c_4->status)]||……;

[0103] 若b和c之间是OR关系,则a1->status=(b_1->status||c_1->status||c_2->status)||(b_2->status||b_3->status||c_3->status||c_4->status)||.....。

[0104] 若无法配对,则a1->status=false。

[0105] 对于栈S_b和S_c中的记录是继续留栈,还是应该删除,GPC轴参照PC轴处理,SAD轴、AAD轴则参照AD轴处理。

[0106] 图4是以实例//a[//c]/b的查询流程图,其中:(a)对所有的元素序列归并排序;(b)依序处理每一个元素;(c)把匹配结果放入缓冲区。图5是图4所示实例的入栈出栈操作示意图。栈中每个记录的三部分是:左边是元素信息;右上边是匹配状态位status,F表示false,T表示true;右下边是指针P_{stack}。图下部展示的是LPL的变化。对//a[//c]/b进行查询的步骤具体说明如下:

[0107] 第一步,从倒排层中把节点a、c和b的元素序列T_a、T_c和T_b取出来。

[0108] 第二步,利用败者树对T_a、T_c和T_b进行归并排序,得到一个元素序列:第一个a、第二个a、c、第一个b、第二个b。

[0109] 第三步,这5个元素按顺序入栈和出栈操作:

[0110] 1)前3个元素都属于祖先后代关系,它们依次入栈,因为c是叶节点,所以c的status为true。

[0111] 2)第一个b元素要入栈,检查LPL,c的闭标签已过,c出链。因为b是叶节点,所以b的status为true。因为b是输出节点,第一个b放入输出缓冲区。

[0112] 3)第二个b要入栈,检查LPL,这时第一个b和第二个a的闭标签已过,它们出链。第二个a出链时,它的status变为true,因为它的子节点b和c的status都为true。第二个a出链使得第一个b元素出栈,因为它不是第一个a的子元素,但是c元素没有出栈,因为它是第一个a的后代,把c的Pstack指向第一个a。因为b是输出节点,第二个b元素也放入输出缓冲区。

[0113] 4)最后,所有元素处理完,LPL中的元素也要出链。当第一个a出链时,它的status变为true,因为它的子节点b和c的status都为true,然后Sb和Sc两个栈全部清空,因为Sa栈中没有元素了。

[0114] 第四步,最后检查output buffer里有两个结果。

[0115] 图6是对//a/*[c]/b的查询实例,要求通配符“*”有两个子节点c和b,c和b是兄弟关系。查询步骤如下:

[0116] 第一步,对XPath进行等值改写,改写后是//a[/2c]/2b,即a有两个孙子节点b和c,且b和c必须是兄弟。

[0117] 第二步,从倒排层中把节点a、c和b的元素序列Ta、Tc和Tb取出来。

[0118] 第三步,利用败者树对Ta、Tc和Tb进行归并排序,得到一个元素序列:第一个a、第二个a、c、第一个b、第二个b。

[0119] 第四步,这5个元素按顺序入栈和出栈操作:

[0120] 1)前3个元素都属于祖先后代关系,它们依次入栈,因为c是第一个a的孙子元素,所以c的Pstack指向第一个a元素。因为c是叶节点,所以c的status为true。

[0121] 2)第一个b元素要入栈，检查LPL，c的闭标签已过，c出链。第一个b是第一个a的孙子元素，所以b的Pstack指向第一个a元素。因为b是叶节点，所以b的status为true。因为b是输出节点，第一个b放入输出缓冲区。

[0122] 3)第二个b要入栈，检查LPL，这时第一个b和第二个a的闭标签已过，它们出链。第二个a出链时，它的statu仍然为false，因为它没有孙子元素c和b。现在栈Sa中只有第一个a元素还是有效的元素，但是第二个b元素并不是它的孙子元素，所以第二个b元素不满足入栈条件。

[0123] 4)最后，所有元素处理完，LPL中的元素也要出链。当第一个a元素出链时，它的status变为true，因为它的孙子元素b和c的status都为true，且b和c是兄弟关系。然后Sb和Sc两个栈全部清空，因为Sa栈中没有元素了。

[0124] 第五步，最后检查output buffer里有一个结果。

[0125] 以上实施例仅用以说明本发明的技术方案而非对其进行限制，本领域的普通技术人员可以对本发明的技术方案进行修改或者等同替换，而不脱离本发明的精神和范围，本发明的保护范围应以权利要求所述为准。

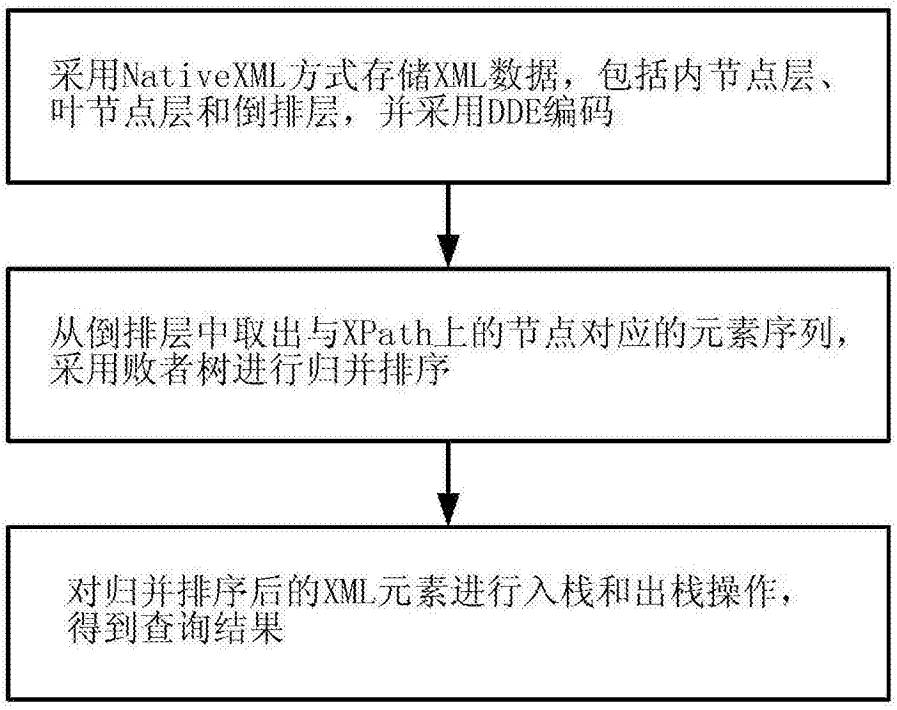

采用NativeXML方式存储XML数据，包括内节点层、叶节点层和倒排层，并采用DDE编码
从倒排层中取出与XPath上的节点对应的元素序列，采用败者树进行归并排序
对归并排序后的XML元素进行入栈和出栈操作，得到查询结果

图1

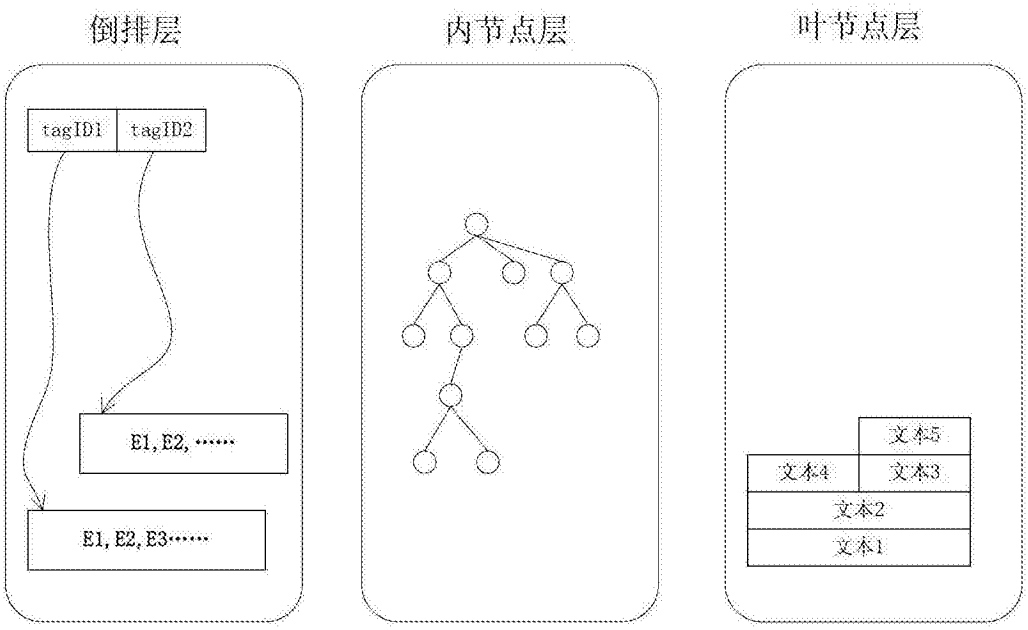
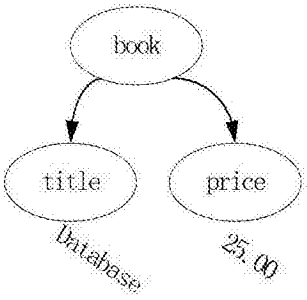

倒排层
内节点层
叶节点层
tagID1
tagID2
E1,E2,……
E1,E2,E3……
文本5
文本4
文本3
文本2
文本1

图2



(a)

book	title	Database	/Database	/title	price	25.00	/25.00	/price	/book

(b)

图3

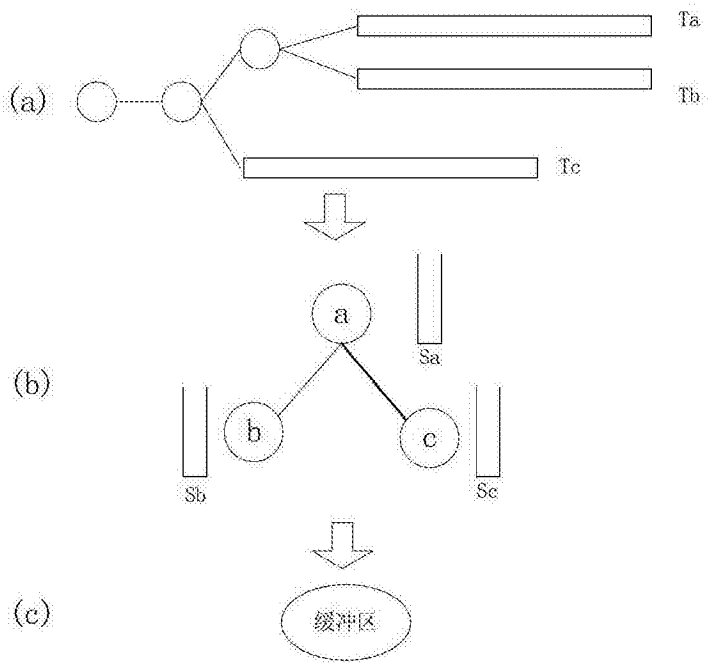


图4

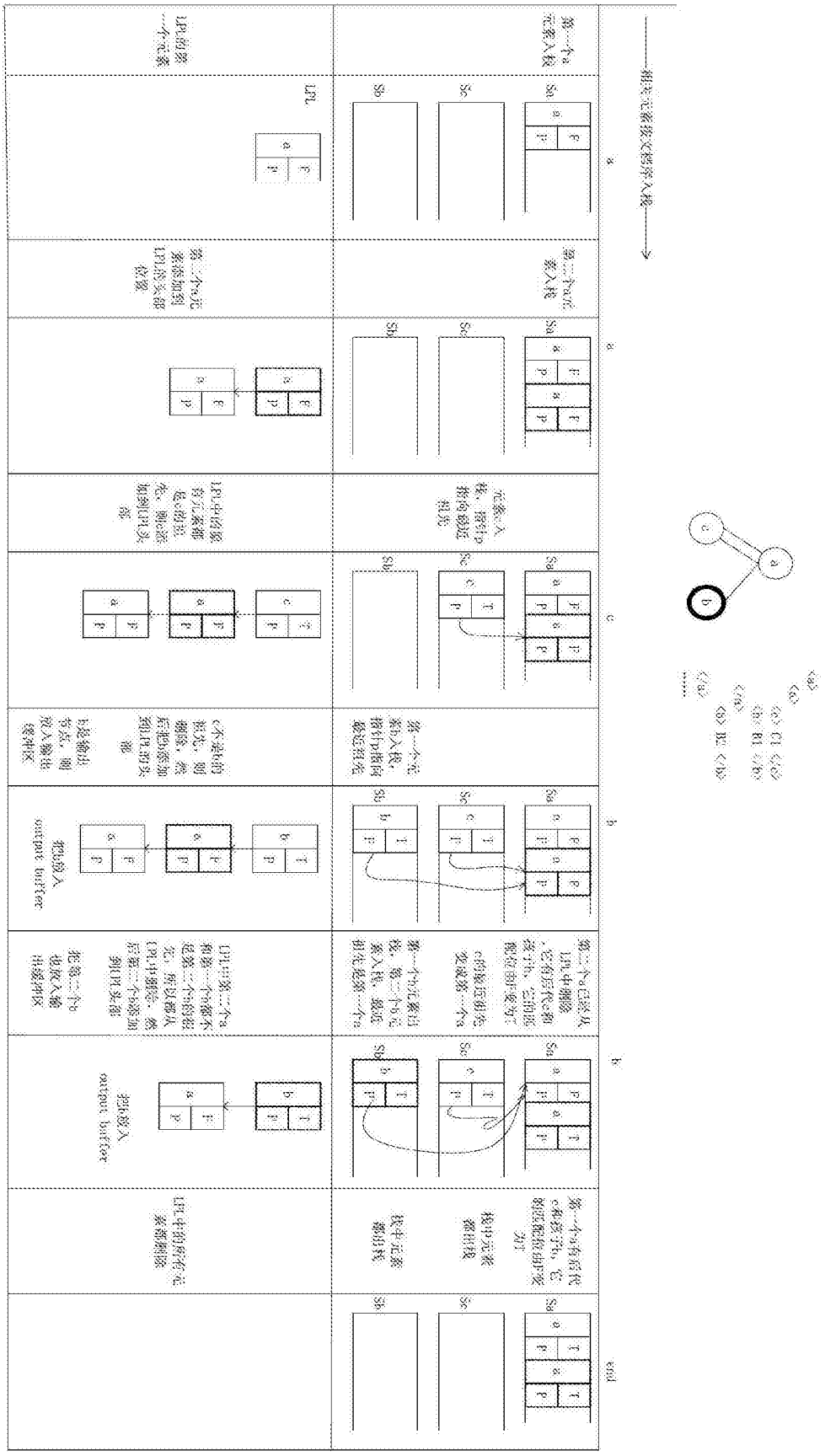


图5

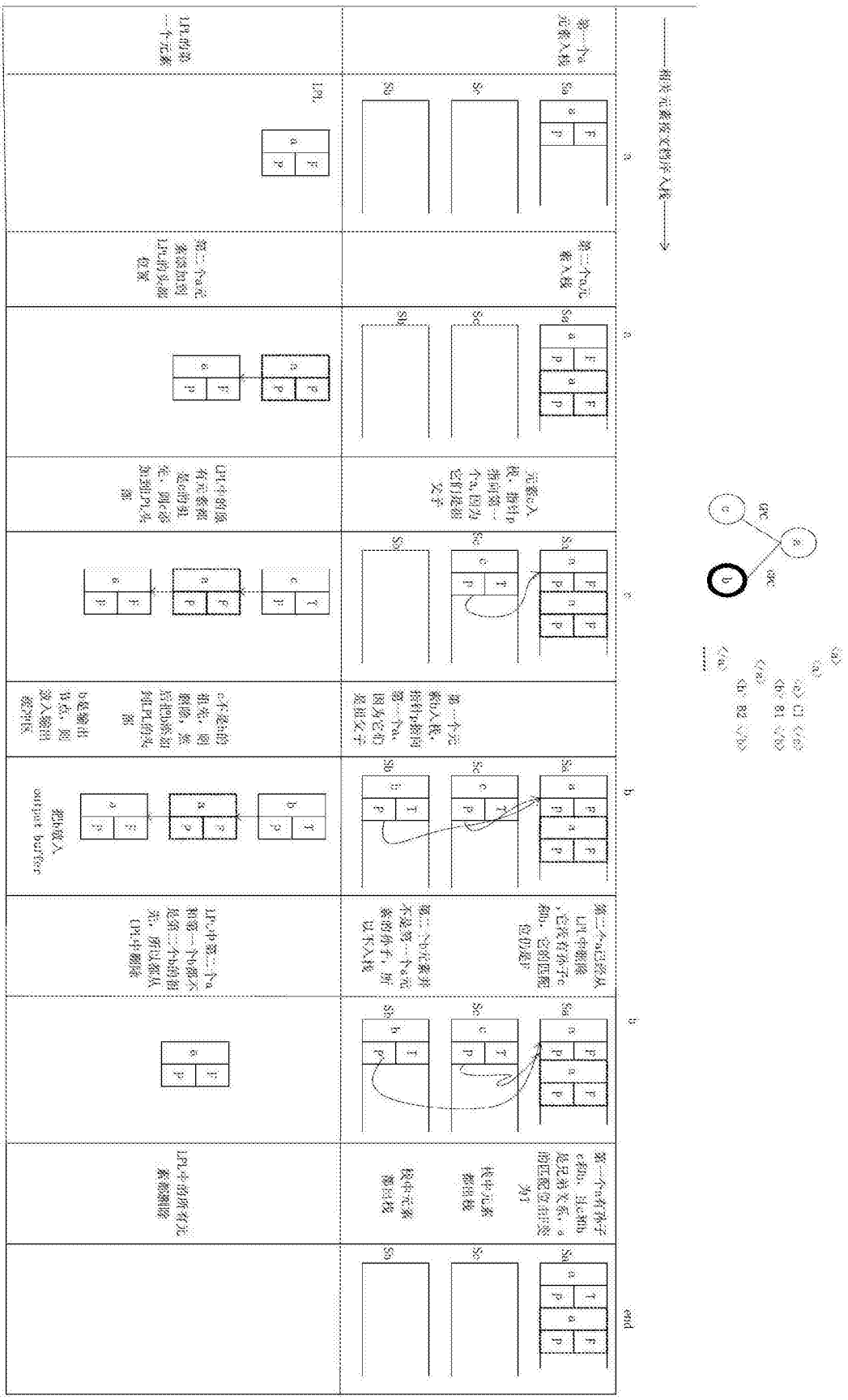


图6