第35卷 第8期 北京工业大学学报 Vol.35 No.8
2009年 8月 JOURNAL OF BEIJING UNIVERSITY OF TECHNOLOGY Aug. 2009

基于XQA查询代数的去除相关性方法

廖湖声，汤 林
（北京工业大学 计算机学院，北京 100124）

摘 要：介绍了一种用于支持复杂XML数据查询优化的XML查询代数XQA以及用于实现XQuery语言的中间语言FXQL，进而提出了一种基于XQA代数的去除相关性方法(de-correlation)，通过查询重写引入连接运算的方法，使得人们有可能更多地利用高性能的连接算法来提高查询效率．在扩展了广义表结构的XDM数据模型的基础上，这种查询重写通过FXQL语言的程序变换方法来实现，易于与各种函数式语言编译优化技术相互结合．实验结果证明了该方法的有效性．

关键词：可扩展标记语言；查询语言；查询代数；去除相关性
中图分类号：TP 311 **文献标志码**：A **文章编号**：0254-0037(2009)08-1108-07

随着计算机网络技术的发展，XML语言已经成为事实上的数据交换和数据共享标准，作为W3C推荐标准[1]的XML数据查询语言XQuery也得到普遍的重视．鉴于XML半结构化数据的特征以及XQuery语言的函数式特征，使得XQuery语言的优化实现面临不少新的挑战．因此，研究人员提出了种种查询代数和优化方法[2]．人们提出了采用树作为数据模型的XAT[3]及优化方法[4]和TAX[5]代数，采用广义表和元组作为数据模型的代数[6]及针对半结构化数据的查询代数SAL、基于SAL的查询重写和代价估计的优化方法[7]；BizQuery系统也应用了基于完备集的查询重写的优化方法[8]；Natix系统使用了针对XPath的完备的查询代数[9]，能提高XPath的查询效率；文献[10]提出了一种NAL查询代数和基于NAL的针对嵌套查询的优化方法；OrientX系统采用基于模式树匹配的OrientXA查询代数[11]．

为了提高XQuery语言的执行效率，作者发展了一种XML查询代数XQA(XML query algebra)以及一种描述查询计算的中间语言FXQL(functional XML query language)，用于支持复杂XML数据查询的优化．本文提出了一种基于XQA代数的去除相关性方法(de-correlation)，针对XML数据处理中的嵌套查询请求，通过查询重写引入连接运算，使得人们有可能利用高性能的连接算法来提高查询效率．

1 FXQL语言与XQA查询代数

1.1 FXQL语言

鉴于XQuery语言是一种函数式语言，为了有效地描述查询计算、实现各种查询优化，作者在λ演算的基础上设计一种简易的、函数式的中间语言，用于描述XQuery语言的语义，有效地支持逻辑优化和物理优化的实现．FXQL语言的抽象语法为

exp→const	表示常数
exp→id	表示变量名
exp→if exp then exp else exp	表示条件表达式
exp→id (exp, ⋯, exp)	表示函数调用，Id为函数名．
exp→exp where id = exp, ⋯, id = exp, id(id, ⋯, id) = exp, ⋯, id(id, ⋯, id) = exp	

收稿日期：2008-01-14．
基金项目：北京市自然科学基金资助项目(4082003)．
作者简介：廖湖声(1954—)，男，广东大埔人，教授．

表示带绑定的 FXQL 表达式. 其中，“id = exp”和“id(id, …, id) = exp”分别表示变量绑定和函数定义.

在 XQuery 形式语义和 XQuery 数据模型中，XDM 数据模型由原子值(如 double、boolean 等等类型)、XML 节点和扁平的数据序列组成. XQuery 语言提供的所有运算都作用于 XML 数据模型实例，扁平化运算隐含在各种序列运算中. 为了挖掘 XQuery 序列运算中的优化机会，FXQL 语言采用广义表(list)作为数据模型，其元素可以是原子值和 XML 节点(元素节点、属性节点等)，其结构的产生式为

list	→	[]	空表
list	→	[elem, elem, …, elem]	有多个元素的广义表
elem	→	Atom	元素为原子值
elem	→	Node	元素为 XML 节点
elem	→	list	元素为广义表

在这种数据模型的基础上，各种 XQuery 序列运算将表示为广义表运算和扁平化运算的组合，使得人们在查询优化中有可能将 2 种运算分解后适当地重新组合，进而完成查询程序的优化. 在广义表数据模型之上，利用广义表的连接 + +、取表首 hd(s)、取表尾 tl(s)等运算，可以表示各种 XML 查询计算. 然而，FXQL 语言并没有直接地区分 XDM 数据序列的相关运算，难以实现相关的查询优化.

1.2 XQA 查询代数

为了有效地描述各种查询优化方法，作者设计了一种查询代数 XQA，面向广义表表示的 XDM 序列运算提供了一组查询算子. 这些查询算子都可以定义为 FXQL 语言标准函数，实现中则可以采用专用的优化算法. 以下为 XQA 查询代数中的主要查询算子.

```
Flat: List → List                表示输入和输出都是广义表
Flat(s) = if empty(s) then[]
          else if list(hd(s)) then hd(s) + + Flat(tl(s))
          else [hd(s). Flat(tl(s))]
```

其中，Flat 为算子，用于完成广义表的扁平化运算；s 为广义表，empty(s)求 s 是否为空表；list(s)为求 s 是否是广义表；[h.t]表示由表头 h 和表尾 t 构造广义表.

```
Foreach: List→(Elem→Elem)→List
Foreach(s, f) = if empty(s) then []
                else [f(hd(s)). Foreach(tl(s), f)]
```

其中，Foreach 为算子，作用于一个广义表和一个函数，得到加工后的广义表；该函数的输入是广义表元素，输出也是广义表元素. Foreach(s, f)对广义表 s 中最外层的每一项应用 f 操作. 例如：Foreach([1, 2, 3], f) where f($ i) = $ i + 1 的求值结果为[2, 3, 4].

```
Filter: List→(Elem→Bool)→List
Filter(s, f) = if   empty(s) then     []
               else if f(hd(s)) then  [hd(s). Filter(tl(s), f)]
               else   Filter(s[2..], f)
```

其中，Filter 为算子，用于完成选择运算，按照函数参数 f 指定的条件，对广义表 f 的元素进行筛选后，返回由符合条件的元素组成的广义表.

为了支持高性能连接运算，XQA 中提供了连接算子 Join.

```
Join: List→List→(Elem→Elem→Boolean)→List
```

Join(s1, s2, f)对广义表 s1、s2 的项做笛卡尔乘积，得到的二元组做 f 条件判断，满足条件 f 的保留. 例如：Join([1, 3, 5], [2, 4, 6], f) where f($ x, $ y) = $ x < $ y 的求值结果是是[[1, 2], [1, 4], [1, 6], [3, 4], [3, 6], [5, 6]]. 实现中可以针对 Join 运算采用高效的连接算法.

为了减少广义表的中间表示，XQA 把 Foreach 算子与 Join 算子结合，得到算子 ForJoin：

```
ForJoin: List→List→(Elem→Elem→Boolean)→(Elem→Elem→Elem)→List
```

运算 ForJoin(s1, s2, f1, f2) 等价于 Foreach(Join(s1, s2, f1), f2),但可以采用高效算法实现. ForGJoin 算子的作用类似 ForJoin 算子.

ForGJoin:List→List→(Elem→Elem→Boolean)→(Elem→Elem→Elem)→List

但是求值后的结果按照 s1 中的每一项进行分组,组成子表,例如:表达式

ForGJoin([1,3,5],[2,4,6],f1,f2)

where f1(x,y)=x<y,

f2(x,y)=x*y

的计算结果为[[1*2,1*4,1*6],[3*4,3*6],[5*6]],即[[2,4,6],[12,18],[30]].

针对多广义表的处理,XQA 引入了算子 MForeach,用于针对两个表中的相应元素作指定的处理.

MForeach:List→List→(Elem→Elem→Elem)→List

MForeach(s1,s2,f)对两个广义表中的对应的项做 f 操作,例如:MForeach([1,2,3],[4,5,6],f) where f(x, y)=x*y 的求值结果是[4,10,18].

对于 XQuery 中各种轴操作,统一记为 Step(QName, e) ,Step 是一个具体的轴操作,如 Child、Descendant 等,QName 为节点名称,e 为一个广义表元素,可以是 XML 节点,也可以是广义表.

Step:String→Elem→List

各种节点构造统一记为 Constructor(QName, e),Constructor 为具体的构造类型,如 Element、Attribute 等,QName 为要构造的节点名称,e 是广义表元素.

Constructor:String→Elem→Node

2 基于 XQA 的去除相关性方法

2.1 双重循环的优化

在数据库查询优化技术中,经常采用去除相关性的优化方法,也就是采用连接运算代替嵌套查询中多重循环计算,从而扩大了高效连接算法的应用范围. 文献[10]提出了一种去除嵌套循环的方法,主要针对选择条件是比较运算和集合运算的情况. 本文的去除相关性方法对选择条件的使用没有限制.

XQuery 语言经常使用 FLWOR 表达式来描述 XML 节点序列的循环处理. 对于具备连接条件的、面向 2 个数据源的 FLWOR 表达式,可以直接翻译成如下形式的 FXQL 表达式

Flat(Foreach(s1, f1))

where f1(a)=Foreach(Filter(s2,f2),f3)

where f2(b)=p(a,b),

f3(b)=g(a,b)

其中,函数 p 为任意连接条件;函数 g 用于任意投影计算. 对符合这种模式的程序模块,可以利用 Join 算子进行优化,用连接操作代替多重循环,也就是通过等效的程序变换,将上述模式的表达式改写为

Flat(Foreach(Join(s1,s2,f2),f1)

where f2(a,b)=p(a,b),

f1(ab) =g(nth(ab,1), nth(ab,2))

其中 nth(s, n)为取广义表 s 中的第 n 项. 随后,进一步合并 Foreach 和 Join 算子,得

Flat(ForJoin(s1,s2,f1,f2))

where f1(a,b)=p(a,b),

f2(a,b)=g(a,b)

从而,去除了 2 个 Foreach 算子和 Filter 算子表示的多重循环以及选择运算,进而有可能利用高效的连接算法提高执行效率.

本节介绍的双重循环的优化规则记为优化规则 1.

2.2 通用的去除相关性方法

在描述比较复杂的查询计算时，FLWOR表达式经常需要嵌套使用；而且许多情况下，内部FLWOR的计算结果需要参与某种运算后，再参与外部FLWOR的运算．这种XQuery程序翻译成FXQL语言后，表现为

```
Foreach(s1, f1)
where    f1(a) = func(a,   Flat(Foreach(Filter(s2, f2), f3))
                  where    f2(b) = p(a, b),
                           f3(b) = g(a, b)
```

其中，s1为外部循环处理广义表；s2为内部循环处理广义表；在内部循环中，以函数p为任意选择条件、以g为任意映射运算；func为组织内部循环计算结果参与外部循环的任意计算．

为了尽可能利用连接运算，提出一种通用的优化规则，将上述间接的多重循环变换成等价的直接循环，进而利用优化规则1，将多重循环变换成连接运算，引入算子MForeach将上述模式的程序改写为

```
MForeach(s1, s, f1)
where s = Foreach(s1, f4)
          where f4(a) = Foreach(Filter(s2, f2), f3)
                        where f2(b) = p(a, b),
                              f3(b) = g(a, b),
        f1(a, ab) = func(a, Flat(ab))
```

随后，利用ForGJoin连接运算算子，代替多重循环和选择运算，变换为

```
MForeach(s1, s, f1))
where f1(a, ab) = func(a, Flat(ab)),
      s = ForGJoin(s1, s2, f2, f3)
          where f2(a, b) = p(a, b),
                f3(a, b) = g(a, b)
```

也就是，对ForGJoin连接运算结果进行分组后，再参与MForeach描述的映射运算，从而有可能利用高效的连接算法获得查询效率的提高．由于函数func代表了外部循环中组织内部循环计算结果的任意函数，这种方法适用于广义表表示的数据集合之间的所有连接运算．

本节介绍的通用的去除相关性方法记为优化规则2．

3 查询代数的实现与查询优化的应用

3.1 核心XQuery的翻译

查询代数和去除相关性的方法已经实现在一个XQuery语言查询引擎中．在系统实现中，XQuery程序经过语法分析、静态类型检查之后，被翻译成核心XQuery语言的表达式；随后，被翻译成FXQL中间语言表示的查询代数，进入查询优化处理阶段．在查询优化处理中，为了实现上述优化方法，系统首先进行程序分析，找出符合上述多重循环模式的程序模块，按照上述优化规则进行程序变换，从而实现了基于XQA代数的查询重写，去除了相关性．

本文仅介绍从核心XQuery到查询代数XQA的常用翻译规则；其中XQA代数采用FXQL语言描述，XQA算子作为FXQL标准函数，翻译结果表现为这些查询算子函数的复合运算．

规则T　XQExp→Exp　　将XQuery表达式翻译为FXQL表达式

翻译规则

```
T[const]        →   const                                   常数
```

T[$var] → $var 变量
T[exp0, exp1] → T[exp0] + + T[exp1] 表的连接
T[if (exp0) then exp1 else exp2] → if T[exp0] then T[exp1]else T[exp2] 条件式
T[let $x: = exp0 return exp] → T[exp] where $x = T[exp0] 局部环境
T[f(exp0, …, expn)] → f(T[exp0], …, T[expn]) 函数调用
T for $x in exp0 return exp] FLWOR 式
→ Flat(Foreach (T[exp0], f)) where f($x) = T[exp]
T[for $x in exp0 where exp1 return exp]
→ Flat(Foreach (Filter(T[exp0], f), g)) where f($x) = T[exp1], g($x) = T[exp]
T[for $x in exp0, $y in exp1where exp2 return exp]
→ Flat(Foreach(T[exp0], f))
where g($x) = Foreach(Filter(T[exp1], g), h) where g($y) = T[exp2], h($y) = T[exp]

按照上述翻译规则,XQuery 语言中的各种运算被翻译为 FXQL 中的标准函数调用,let 子句中的变量绑定被翻译成局部环境中的绑定,而核心的 FLWOR 表达式被翻译成 Flat、Foreach 和 Filter 等算子的组合应用,而且通过函数抽象定义了 2 种局部函数. 按照类似的方法,XQuery 语言中的自定义函数也被翻译为 FXQL 语言的自定义函数.

3.2 应用举例

假设有两个 XML 文档 Samp1.xml 和 Samp2.xml.
考虑以下 XQuery 程序的处理

```
for $a in doc("Samp1.xml")/a,
    $b in doc("Samp2.xml")/b
      where $a > $b
      return $a * $b
```

应用上述翻译规则,该程序被翻译为 FXQL 语言表示的查询代数

```
Flat(Foreach(Descendant-or-self("a", doc ("Samp1.xml")), f1))
where
    f1( $a) = Foreach(Filter(Descendant-or-self("b", doc("Samp2.xml")), f2), f3)
              where f2( $b) = $a > $b,
                    f3( $b) = $a * $b
```

此表达式符合优化规则 1,被变换为

```
Flat(ForJoin(Descendant-or-self("a", doc ("Samp1.xml")),
             Descendant-or-self("b", doc ("Samp2.xml")), f1, f2))
where f1( $a, $b) = $a > $b,
      f2( $a, $b) = $a * $b
```

考虑另一个 XQuery 程序例

```
for $a in doc("Samp1.xml")/descendant-or-self::a
return func( $a, for $b in doc("Samp2.xml")/descendant-or-self::b
                 where $a > $b
                 return $a * $b)
```

其中,func 为用户定义的函数. 应用上述翻译规则,翻译为 FXQL 表示的查询代数

```
Flat(Foreach(Descendant-or-self("a", doc ("Samp1.xml"), f1))
where f1( $a) = func ( $a,
                  Flat(Foreach(Filter(
```

```
                                    Descendant-or-self("b", doc("Samp2.xml")), f2), f3))
                where f2( $ b) = $ a > $ b,
                      f3( $ b) = $ a * $ b
```

其中，最外层 Flat 算子的参数符合通用的去除相关性方法的优化规则，优化

```
Flat(MForeach(Descendant-or-self("a", doc ("Samp1.xml"), $ s, f1))
where $ s = ForGJoin(Descendant-or-self("a", doc ("Samp1.xml"),
                     Descendant-or-self("b", doc("Samp2.xml")), f2, f3)
          where f2( $ a, $ b) = $ a > $ b,
                f3( $ a, $ b) = $ a * $ b),
     f1( $ a, $ ab) = func( $ a, Flat( $ ab))
```

3.3 实验结果

作者在 XQuery 查询引擎中实现了上述查询优化和一个通用的连接算法，并根据从 W3C 的 XQuery Use Case 中选择和改写了部分案例，统计了运行时间，对比优化前与优化后运行时间，结果如表 1.

表 1　实验结果

Table 1　Testing Results

查询编号	优化规则	未优化执行时间/ms	优化后执行时间/ms	优化效果/%
1	1	485	390	19.6
2	1	641	563	12.2
3	1	797	750	5.9
4	1	266	234	12.0
5	1	220	203	7.7
6	2	572 125	371 187	35.1
7	2	359	281	21.7
8	2	578	516	10.7

注：优化效果 = (未优化执行时间 - 优化后执行时间)/未优化执行时间 × 100%

由结果可以看出，总体上查询时间减少，达到了一定的优化效果. 如果在执行引擎中能区别连接条件，分别采用更高效的连接算法，或者采用并行算法，预计能达到更好的优化效果.

4 结束语

探讨了基于 XML 查询代数的查询优化技术，介绍了一种基于广义表数据模型的 XML 查询代数 XQA 和一种基于 λ 演算的中间语言. 提出了一种用于去除相关性的查询优化方法，并且介绍了该方法在 XQuery 语言查询引擎中的实现策略和应用案例. 该方法面向广义表形式的数据模型，通用性较强，而且采用 FXQL 中间语言程序变换的形式实现，有利于查询代数优化方法和函数式语言编译优化方法的综合利用.

XQuery 语言的实现技术涉及 XML 数据查询优化和函数式语言的优化实现技术，需要利用数据库理论中较成熟的查询优化技术，并且研究 XML 数据查询的特点及其优化实现技术；同时，鉴于 XQuery 语言是一种函数式语言，其应用的扩展更多地依靠自定义函数来描述业务逻辑；该语言的优化实现也需要参考函数式语言实现技术的发展.

参考文献：

[1] SCOTT B, DON C, FERNÁNDEZ M, et al. XQuery 1.0: An XML Query Language[EB/OL]. MIT/CSAIL, USA: W3C, [2008-01-14]. http://www.w3.org/TR/xquery/.

[2] 孟小峰, 王宇, 王小锋. XML查询优化研究[J]. 软件学报, 2005, 17(10): 2069-2086.
MENG Xiao-Feng, WANG Yu, WANG Xiao-Feng. Research on XML query optimization[J]. Journal of Software, 2005, 17(10): 2069-2086. (in Chinese)

[3] ZHANG X, RUNDENSTEINER E. XAT: XML algebra for rainbow system[R/OL]. Worcester, USA: Worcester Polytechnic Institute, [2008-01-14]. http://davis.wpi.edu/dsrg/Old/TECH-REPS/tech-wpi-cs-02-24-xat.ps.

[4] WANG S, RUNDENSTEINER E, MANI M. Optimization of nested xquery expressions with orderby clauses[C]// Proceedings of the 21st international Conference on Data Engineering Workshops. Amsterdam, Netherlands: Elsevier Science Publishers B. V., 2007: 303-325.

[5] JAGADISH H, LAKSHMANAN L, SRIVASTAVA D, et al. TAX: a tree algebra for XML[C]// Revised Papers From the 8th international Workshop on Database Programming Languages, Lecture Notes In Computer Science. London, UK: Springer-Verlag, 2001: 149-164.

[6] FERNÁNDEZ M, SIMÉON J, SUCIU Dan, et al. A data model and algebra for XML query[R/OL]. NJ, USA: AT&T Labs, [2008-01-14]. http://homepages.inf.ed.ac.uk/wadler/papers/xquery-algebra/xquery-algebra.html.

[7] BEERI C, TZABAN Y. SAL: an algebra for semistructured data and XML[C]// Proceeding of the 2nd ACM SIGMOD Workshop on the Web and Databases. Philadelphia, Pennsylvania, USA: ACM Press, 1999: 37-42.

[8] GRINEV M, KUZNETSOV S. Towards an exhaustive set of rewriting rules for XQuery optimization: BizQuery experience [C]// Proceedings of the 6th East European Conference on Advances in Databases and information Systems, Lecture Notes In Computer Science. London, UK: Springer-Verlag, 2002: 340-345.

[9] BRANTNER P, HELMER P, KANNE C, et al. Full-fledged algebraic XPath processing in natix[C]// Proceedings of the 21st International Conference on Data Engineering. Washington, DC, USA: IEEE Computer Society, 2005: 705-716.

[10] MAY N, HELMER S, MOERKOTTE G. Nested queries and quantifiers in an ordered context[C]// Proceedings of the 20th International Conference on Data Engineering. Washington, DC, USA: IEEE Computer Society, 2004: 239-250.

[11] 孟小峰, 罗道锋, 蒋瑜, 等. OreintXA:一种有效的XQuery查询代数[J]. 软件学报, 2004, 15(10): 1648-1660.
MENG Xiao-feng, LUO Dao-feng, JIANG Yu, et al. OrientXA: an effective xquery algebra[J]. Journal of Software, 2004, 15(10): 1648-1660. (in Chinese)

A Decorrelation Method Based on XQA Query Algebra

LIAO Hu-sheng, TANG Lin

(College of Computer Science and Technology, Beijing University of Technology, Beijing 100124, China)

Abstract: XQA (an XML Query Algebra), as well as a intermediate functional language named FXQL (a Functional XML Query Language) for implementation of XQuery language, is introduced to support optimizing for complex XML query. Furthermore, a decorrelation method base on XQA is put forward, which helps to use efficient Join algorithms to speed up query execution by rewriting the nested XML query with Join operator. Based on an extended XDM model with list structure, the query rewriting is implemented by a program transformation on FXQL program and easy to be combined with the various optimizations for functional language. Results of experiments demonstrate the effect of this optimization.

Key words: XML; query language; auery algebra; decorrelation

(责任编辑 张士瑛)