第 36 卷 第 7 期 计 算 机 工 程 2010 年 4 月
Vol.36 No.7 Computer Engineering April 2010
·软件技术与数据库· 文章编号：1000—3428(2010)07—0030—03 文献标识码：A 中图分类号：TP311.12

XML 集成中关系数据的动态转换方法

冯敬娥[1,2]，黎建辉[1,2]
(1. 中国科学院计算机网络信息中心科学数据实验室，北京 100190；2. 中国科学院研究生院，北京 100190)

摘 要：在关系数据库向 XML 集成的过程中，静态转换方法对关系数据进行一次转换，无法满足实际应用中某些特定关系数据的转换需求。为实现动态转换，在静态转换系统通用映射模型的基础上提出动态转换系统，给出相应的动态转换算法。通过实例验证该算法的有效性和通用性。

关键词：关系数据库；动态转换；可扩展标记语言；XQuery 查询

Dynamic Translation Method for Relational Data in XML Integration

FENG Jing-e[1,2], LI Jian-hui[1,2]
(1. Scientific-data Lab, Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190;
2. Graduate University of Chinese Academy of Sciences, Beijing 100190)

【Abstract】During the process of relational database integrated to XML, relational data is translated for once by static transform method. This method can not meet the translation needs of special relational data in some applications. In order to realize dynamic translation, this paper proposes a dynamic translation system based on the universal mapping model of static translation system and gives the relevant dynamic translation algorihtm. The efficiency and generability of this algorithm are verified by examples.

【Key words】relational database; dynamic translation; eXtensible Markup Language(XML); XQuery

本文在分析静态转换系统[1]的基础上，提出动态转换系统，并借鉴相关技术给出基于通用映射模型 RtoX 的相应算法。

1 静态转换系统简介

文献[1]分别为关系模式和 XML Schema 建立通用模型 R 和 X，定义了描述关系模式和 XML Schema 之间映射关系的通用映射模型 RtoX，并提出基于该模型的静态转换算法。RtoX 映射模型的建模方法决定了其具有通用性，它对最终的映射关系建模，其中，怎样映射由用户定义，以满足各种转换需求。本文简要介绍 R, X 和 RtoX。

假设存在关系表名集合 T、列名集合 C 和原子数据类型集合 A。按照文献[1]思路，关系模式为五元组 $R=(T_R, C_R, F, PK, FK)$，分别代表表名集合、表与列的对应关系、列的类型和主外键约束。对 XML Schema 建模时，首先定义一些基本概念，然后将 XML Schema 的模型 X 形式定义为六元组 $X=(ns, E_S, A_S, ST, CT_S, r)$，分别代表命名空间、$ns$ 中元素声明集合、所有复杂元素类型定义中使用的属性集合、内置简单类型集合、ns 中所有元素使用的复杂类型定义的集合、ns 中根元素的声明。

RtoX 映射定义为七元组 $M(R, S, M_a, M_c, M_t, \nabla, \Delta)$。其中，$Ma$ 是关系列与属性间映射关系的集合；Mc 是关系列与简单类型元素间映射关系集合；Mt 是关系表与复杂类型元素间映射关系集合；∇，Δ 分别是对 Mt 中映射关系的来源关系表约束集合和映射间关联约束的集合。一个模式实例如下：

例 1

M_a={m_{aSM}(STUDENT.SNAME,STU_Type.姓名),
m_{aCG}(SELECT_COURSE.GRADE,SC_Type.成绩),
m_{aCT}(SELECT_COURESE.TIME,SC_Type.时间)}
M_c={m_{cSN}(STUDENT.SNO,STU_Type.学号),
m_{cST}(STUDENT.TEL,STU_Type.电话),
m_{cCN}(SELECT_COURESE.CNO,SC_Type.课号),
m_{cCM}(COURESE.CNAME,SC_Type.课名)}
M_t={m_{tS}(Student,学生), m_{tX}(SELECT_COURS.选课信息)}
$\nabla = \varnothing$
$\Delta = \varnothing$

2 动态转换系统

文献[1]在通用映射模型的基础上，给出关系数据到 XML 的静态转换算法，而在实际应用中需要进行动态转换。本文利用 RtoX 中提供的关系模式和 XML Schema 的映射信息，提出动态转换方法。用户可以输入 XQuery 语句来限定需要转换的关系数据，把此查询语句转换成对关系数据的查询语句，并把查询结果转换为 XML 数据。其中的关键技术是查询转换。动态转换系统结构见图 1。

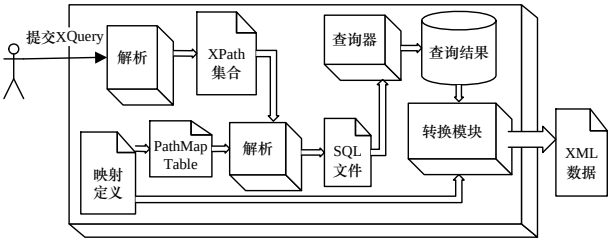

图 1 动态转换系统结构

已有的查询转换[2-4]都借鉴某种中间数据结构进行。

作者简介：冯敬娥(1983－)，女，硕士研究生，主研方向：XML 与关系数据库集成，数据共享技术；黎建辉，高级工程师、博士
收稿日期：2009-10-28 **E-mail：**fengjinge@sdb.cnic.cn

XRel[2]采用 XPathCore 图，silkRoute 采用 ForestView。XRel 在已存储 XML 数据的基础上进行讨论，SilkRoute 没利用 XPath 的信息，转换的 SQL 语句有多个 join 连接[5]。如果能利用已有通用映射模型 RtoX 进行动态转换，则可以继承其通用性优点，并利用 XPath 信息。

3 动态转换算法

3.1 PathMapTable的建立算法

中间数据结构 PathMapTable 是一个二维关系表，存储到每个 XML 节点的路径对应的关系模式信息，从映射文件提取信息到其中。

转换思路如下：从通用映射模型 RtoX 提取出 XQuery 查询路径对应的关系模式信息到 PathMapTable 后，当用户输入简单 XQuery 查询时，按 PathMapTable 表提供的信息，将查询解析为针对相应的关系数据库的 SQL 查询语句，对查询结果进行转换。

可以使用文献[1]的 R 模型对此表建模，本文简单介绍 XPath 和 RSInfo：

(1) XPath 表示目标 XML Schema 树中从根节点到某个节点的路径。

(2)RSInfo 表示这些路径中的节点表示的元素或属性对应的关系模式信息。令 $t \in TR$, $t.c \in CR(t)$，用正则式记为

RSInfo＝(/t|t.c)n

PathMapTable 建立算法中的函数、变量描述如下：

(1)insert(String tablename, String column1, String column2) 分别把 column1, column2 加入 tablename 表的第 1 列和第 2 列。add(String var, String add)让字符串 var 连接 add。

(2)add(path, e)方法将名字为 path 的字符型变量的值增加“/e”。path 变量保留到父节点的路径，RSInfo 列需要 table 变量来保存对应的关系模式。

对于任意给定的关系数据库 D，设 D 的关系模式为 $R=(T_R, C_R, F, PK, FK)$，目标 Schema 为 $S=(ns, E_S, A_S, ST, CT_S, r)$，映射实例为 $M=(R, S, M_a, M_c, M_t, \nabla, \Delta)$。

PathMapTable 的建立算法描述如下：

输入 映射实例 *M, S, R*
输出 PathMapTable 或出错返回 NULL
查找 m_t，使得 $m_t \in M \wedge m_t.e.n = S.r$
if $C_R(m.t) \varnothing R.C$，返回 NULL;else 令 path=“/mt.e.n”
while(m_t.t != ∅){
 add(table, m_t.t),
 ele=mt.e.n, add(path,ele), insert(PathMapTable, path,table)
 //复杂元素下必然有子节点，需要改变 path 值
 tc=mt.e.t，//依据复杂元素映射定义可知 $t_c \in CT_S$
 a= $t_c.A_c$;
 while(a ∈ A_c &&a!= ∅){
  在 Ma 中寻找符合 m.a=a 的映射 m_a
  if (m_a= ∅) {break:}
  else{
   insert(PathMapTable, path/a,table/m_a.c)
  }
 while(e' ∈ E_c){ //按元素声明的顺序遍历 tc.Etc，
 If(e'.t ∈ ST){
 在 MC 中寻找满足 m..e = e ' 的映射 mc
 If(mc= ∅){
  break;
 }
 else{insert(PathMapTable, path/ m_c.e , table/ m_c.c)}
 }//end if
 else{ // e'.t ∈ CT_S
  在 Mt 中寻找满足 m_t'.e=e' 的映射 m_t'
  while（m_t'!= ∅ ）{
   if(m_t' = ∅) break;
   else{m_t= m_t':}
  }
 }//end else
}//end while
}//end while

例 1 提取的 PathMapTable(XPath, RSInfo)结果如下：

学生 student
学生/名字 student/name
学生/学号 student/sno
学生/电话 student/tel
学生/选课信息 student/select_course
学生/选课信息/时间 student/time
学生/选课信息/成绩 student/grade
学生/选课信息/课号 student/cno
学生/选课信息/课名 student/cname

为了支持查询转换算法，定义 2 种操作，即节点匹配、最长匹配。节点匹配指只匹配 XPath 最后一个节点，例如“学号”匹配“学生/学号”。最长匹配指顺序匹配，结果为相同字符数最多的一个。

3.2 查询转换算法

由 PathMapTable 可知每个节点的路径和相应关系模式信息。由文献[1]可知，映射文件是用户自定义的，因此，只要再利用关系模式 R 的外键信息，就可以组装成所需 SQL 语句。由于 XQuery 语法结构较复杂，因此本文支持部分 XPath 语句。

沿用文献[1]对 X 的定义，XPath 的节点名称为 $NodeNames=\{a \vee e \mid a \in A_s.n, e \in E_s.n\}$，设 C 为一个数值常量或字符串常量，则 $F=\{f \mid f(@a)=C \wedge a \in A_s \vee f(e)=C \wedge e \in E_s\}$，XPath 节点数记为 n，则

$$XP=\{/\, r((m \vee m[f(@a)] \vee m[f(e)])/)^n \mid m \in Nodenames,\ n\text{为XPath节点数},\ f \in F\}$$

令 SQL 语句为一个三元组{*S, F, W*}，*S, F, W* 分别代表 select, from, where 语句，定义如下：

$$S=\{\text{“select } c_1, t_j \in T_R \ldots c_n\text{”} \mid c_i \in C_R \vee \text{“*”},\ i=1,2,\cdots,n\}$$

$$F=\{\text{“from } t_1, t_2, \cdots, t_n\text{”} \mid t \in T_R,\ i=1,2,\cdots,n\}$$

$$W=\{\text{“where } f(c_1) \text{ and } f(c_2) \text{ and}\cdots\text{and } f(c_n) \text{ and } t_1.c_i=t_2.c_j \text{ and } t_3.c_i=t_4.c_j \text{ and}\cdots\text{and } t_m.c_i=t_{m+1}.c_j\text{”} \mid c_i \in C_R \wedge t_j \in T_R \wedge \forall c_i(\exists t \in T_R c_i \in t.c) \wedge \forall t_i(\exists t_j \in F),\ m=1,2,\cdots,n\text{-}1\}$$

基于 PathMapTable 的查询转换算法描述如下：

输入 XP, PathMapTable
输出 SQL 语句三元组

(1)选择最后一个节点从 PathMapTable 中进行节点匹配，得到对应的 RSInfo 列，记为 RName，按照定义，为(/t|t.c)n。选择最后一个节点的信息，如果为 t，则令 S=select t.*；如果是 t.c，则令 S=select t.c. 将整个路径中的 t 提取放入 fromtable 中。

(2)整个 XPath 去掉[]进行最长匹配得到该查询的关系模式信息 RName。设路径第 i 个节点记为 Pi，如果 Pi 为 N[f(@a)]或 N[f(e)]，该节点的限制[f(@a)] 或[f(e)]记为 c_i。则整个 XPath 的限制 C={c_i, i=1,2,⋯,n}。为每个限制中的 a 或者 e 进行节点匹配得到 RName 为 R_i，路径最后一个节点为 r_i，令 $c_i=f(r_i)$；则 W + =where c_1 and c_2 and⋯ and c_n。

(3)将 Rname 的每个节点 t|t.c 的 t 提取放入集合 fromtable，第 i 个节点的 t 记为 t_i，令 F=from $t_1,t_2,\cdots, t_n$。

(4)对 fromtable 中每个元素都在关系模式 R 中提供的 FK 中查找，如果 t_i 和 t_j 有外键连接 $t_j.c_j=t_i.c_i$，则 W+=and $t_j.c_j=t_i.c_i$。

(5)返回 SQL 语句(S, F, W)。

例 2

XPath 输入：学生/选课信息[@成绩>90]/学号，即

```
Select student.sno
From student,select_course
Where select_course.grade>90 and
        student.sno=select_course.sno
```

如果 XQuery 是 FLWR 语句，则变量替换消掉 L，转换为 FWR 语句，记为 NXQ 语句。可定义为 $NXQ=\{F, W, R\}$，“NXQ.F”，“NXQ.W”，“NXQ.R”分别表示 for, where 和 return 语句，用 x, y 表示变量，<tag>表 XML 标签，使用 XP 定义，分别定义为 $F=\{\text{for } x \text{ in } xp \mid x \subseteq NodeName, xp \in XP\}$，$W=\{f \mid f \in F\}$，return 语句中含自定义标签的语句集合 $RExp=\{<tag>xp</tag> \mid xp \in XP\}$，则 $R=\{nxq \vee rExp \mid nxq \in NXQ, rExp \in RExp\}$。

通过对 NXQ 语句解析，得到 FWR 分别对应的 XP 信息。将 for 语句的路径对应的关系模式信息解析为 SQL 语句的 from 信息，where 语句解析为 SQL 中的 where 信息，return 语句如果是{<tag>XP</tag>}，则解析为 SQL 中的 return 信息，如果是 NXQ，则递归求解。

FLWR 语句到 SQL 语句的转换算法描述如下：

输入 FLWR 语句 NXQ, PathMapTable

输出 SQL 语句

(1)如果 F=for x in XP，则对 L,W,R 中的 x 都替换为 XP，如果 L=let y=XP，则在 W,R 中的 y 都替换为 XP。R 如果是 NXQ 语句，对 R 进行同样的变量替换，直到 R 为简单返回语句。这样消掉变量后，F.W,R 中均剩下 XP。

(2)对 NXQ.F 中的 XP 进行最长匹配，得到的 RName 中的每个 t 都存入 fromtable。

(3)如果 NXQ.W 为空，则转(4)，否则令 W=where，对 NXQ.W 中的 XP 去掉[]进行最长匹配，得到的 RName 中的每个 t 都存入 fromtable 集合中，分别对限制 f(a|e)中的属性或元素进行节点匹配，得到 RName 最后一个节点为 t.c 时，W+= f(t.c)。

(4)令 S=select，如果 NXQ.R 是简单返回语句，对 R 中的每个路径进行最长匹配，得到的 RName 的每个节点的 t 值都存入 fromtable，对每个最后节点记为 t.c，如果是 t，则记为 t.*，令 S+= t.c|t.*，转(5)，否则令 NXQ=NXQ.R，转(2)。

(5)令 F=from，对 fromtable 的第 i 个元素记为 t_i，F+=t_i, i=1,2,…,n，n 为 fromtable 的元素个数。

(6)对 fromtable 中每个元素都在关系模式 R 中提供的 FK 信息中查找，如果 t_i 和 t_j 有外键连接 $t_j.c_j=t_i.c_i$，则 W+=and $t_j.c_j=t_i.c_i$。

(7)返回 SQL 语句(S,F,W)。

例 3

输入 XQuery

```
<student>
  for $stu in $SelectCourseView/student
return
        for $sub $stu/courseinfo
        where $sub/[@grade>=90]
        return
<id>{data($sub/cid)}</id>
<courseinfo>
  <cid>{data($stu/cid)}</cid>
  <tiem>{data($sub/time)}</time>
</courseinfo>
</student>
```

例 3 的转换结果如下：

```
slect student.sno,select_course.cno,select_course.time
from student,select_course
where select_course.grade>=90 and
student.sno = select_course.sno
```

3.3 动态转换算法

基于通用映射模型的动态转换算法描述如下：

输入 关系数据库 R，通用映射模型 mt，XP 语句

输出 XML 数据

(1)执行 CREATE_PATHMAP(m_t)，将该映射的关系模式信息和 XPath 信息存入 PathMapTable(XPath, RSInfo)。

(2)检测 XQuery，若是 XP 格式，则转入 XPtoSQL 算法，如果是 FLWR 格式，则转入 FLWRtoSQL 算法。

(3)在关系数据库引擎上执行 SQL 语句，结果集记为 R0。

(4)将 R0 根据转换算法[5]进行转换，得到 XML 数据。

例 2 的转换结果如下：

```
<学生 姓名＝“卢蓉”>
<学号>2001</学号>
<电话></电话>
<选课信息 时间＝“” 成绩＝“92”><>
<课号>200</课号>
<课名></课名>
</学生>
<学生 姓名＝“王允文”>
<学号>2003</学号>
<电话></电话>
<选课信息 时间＝“” 成绩＝“90”><>
<课号>101</课号>
<课名></课名>
</学生>
```

4 结束语

本文系统通过受限的 XPath 和 XQuery 限定需要转换的关系数据源，使用一个中间数据结构 PathMapTable 来存储通用映射模型，推导出目标 Schema 下的路径对应的关系数据库的路径信息，并利用关系模式的外键将其转换成 SQL 语句。使用查询结果作为数据源进行转换，从而达到动态转换的目的。由于该系统基于通用映射模型，因此可以用于任何关系库向 XML 的集成，但它能支持的查询限定语句有一些语法限制，需要进一步扩展。

参考文献

[1] 史晔翎. 关系数据库面向应用需求 XML 模式的集成方法的研究和实现[D]. 北京: 中国科学院研究生院, 2008.

[2] Yoshikawa M, Amagasa T, Shimura T, et al. XRel: A Path-based Approach to Storage and Retrieval of XML Documents Using Relational Databases[C]//Proc. of ACM Conference on Internet Technology. New York, USA: ACM Press, 2001: 110-141.

[3] Andez M F, Kadiyska Y, Suciu D, et al. SilkRoute: A Framework for Publishing Relational Data in XML[J]. ACM Transactions on Database Systems, 2002, 27(4): 1-55.

[4] 甄玉钢, 刘璐莹, 康建初. 基于 XML 的异构数据库集成系统架构与开发[J]. 计算机工程, 2006, 32(2): 85-87.

[5] 姚文隽. 关系模式下从 XML 到 SQL 的查询转换及优化技术[D]. 南京: 东南大学, 2004.

编辑 陈 晖