一种关系数据库模式到本体映射的失效检测方法

唐富年 姚 莉 漆学田 肖清滔
(国防科技大学 C⁴ISR 技术国防科技重点实验室 长沙 410073)

摘 要 从关系数据库到本体的语义映射对于实现关系数据库和本体之间的互操作具有十分重要的作用。然而,关系数据库模式和本体的演化都会造成这些语义映射的失效,只有准确定位受影响的映射集合才能进行进一步的映射维护工作。提出了一种映射失效检测方法,首先通过对关系数据库模式(本体)的新旧版本进行比较来求取变化元素集合和合理演化路径,然后基于该变化元素集合生成查询测试集,并且借助相应的映射将查询测试集中的查询重写。最后,根据重写后的查询能否返回合法实例,就可以判定查询重写所使用的映射是否失效。

关键词 关系数据库,本体,语义映射,映射维护
中图法分类号 TP182 **文献标识码** A

Detecting Invalid Mappings between Relational Database and Ontology

TANG Fu-nian YAO Li QI Xue-tian XIAO Qing-tao
(C⁴ISR Technology National Defense Science and Technology Key LAB, National University of Defense Technology, Changsha 410073, China)

Abstract The semantic mappings between relational database and ontology are critical to the implementation of interoperability between them. However, these mappings would be invalid if schema (or ontology) evolved, and it is a pre-requisite work to find the affected mappings while some change occurs. In this paper, by comparing the new version and old version of the schema (ontology), rational evolution paths between the old version and new version were discovered, and a set of changed elements were detected. Based on the set of changed elements, we can design a query testing set, and queries in the testing set can be reformulated in virtual of corresponding mappings. Finally, if the reformulated queries can not return a valid result set, the mappings used in reformulating queries are proved to be invalid.

Keywords Relational database, Ontology, Semantic mapping, Mapping maintenance

1 引言

近年来,有关如何在语义 Web 的框架之中使用关系数据库的问题逐渐成为研究热点。语义 Web 的应用通常可以通过关系模式到本体的语义映射来实现对关系数据库的访问[1]。映射是关系数据库和本体之间的桥梁,反映了两种模型之间的语义关联,对本体和关系数据库的互操作有着极为重要的作用。

然而,在 Web 这样的动态环境中,关系数据库模式和本体都是不断演化的,这通常会使关系数据库和本体之间的语义关联发生改变,造成语义映射的失效。语义映射一旦失效,则合法的关系数据库实例就无法转换为合法的本体实例[2],进而导致对数据的语义查询出现故障。

映射失效的检测问题是进行映射维护的关键。尽管目前已经出现了一些针对映射失效检测问题的原型系统,如 MAVERIC[3]等,但是这些系统都主要针对模式映射,无法直接应用于关系数据库和本体之间的语义映射。鉴于这一现状,本文从数据库模式演化和本体演化两个角度出发研究语义映射的失效检测问题。

2 关系数据库模式到本体的语义映射

本体是语义 Web 的基本单元,通过进行关系数据库模式和本体之间的语义映射,能够准确地描述关系数据库的语义。而且,进行关系数据库到本体的映射是语义 Web 环境下集成和访问关系数据库的重要途径之一。

2.1 关系数据库到本体的映射方式

目前关系数据库到本体的映射有很多种,总体来说可以归类为两种方式。

(1)从关系数据库直接生成本体,同时建立数据库元素和本体实体之间的映射。这种情况下生成的本体通常和数据库模式非常相似,而且本体的类和属性一般直接来自于数据库的表和列,映射的形式一般也比较简单。这种映射的生成过程通常可以实现自动化,因而不用担心映射维护的问题。但是这种映射方式还必须进行本体间的映射才能实现异构数据库的集成。支持这种映射方式的工具主要有 RDB2ONT[4]和 DB2OWL[5]等。

(2)从关系数据库模式到已有本体的映射。这种映射方式如图 1 所示,生成的映射结果含有较为丰富的语义信息,通

到稿日期:2009-04-20 返修日期:2009-07-07 本文受国家自然科学基金(70971134,90716015)资助。
唐富年(1981—),男,博士生,主要研究方向为本体技术与信息集成,E-mail:tfn1981@sina.com;姚 莉(1965—),女,教授,博士生导师;漆学田(1980—),男,博士生;肖清滔(1985—),男,硕士生。

常面向异构关系数据库的集成。但是映射发现过程通常只能以手工或者半自动化方式进行，映射维护的任务也非常艰巨。支持这种映射方式的主要有 MAPONTO[6]，R2O[7]等。本文的研究以基于本体的异构关系数据库集成为应用背景，研究这种复杂语义映射的失效检测问题。

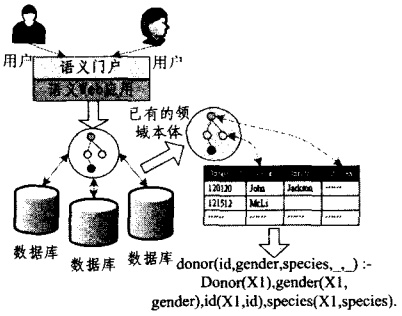

图 1　将关系数据库映射到一个已存在的本体上

2.2　关系数据库到本体的映射系统

关系数据库模式中包含了大量的关系数据库语义信息，是对具体业务领域的建模。关系数据库模式由一组关系模式组成，通常可以将关系模式做如下定义。

定义 1　关系模式 $\mathcal{R}$ 由一组属性 $\mathcal{A}$(attribute)和约束(Σ)构成，可以写作 $\mathcal{R}:=(\mathcal{A},\Sigma)$。

本体是概念模型明确的规范说明，它体现的是领域中共享的知识，反映了相关领域中公认的概念集。结合文献[8]中本体模型的定义和本文的具体问题，将本体模型做如下定义。

定义 2　本体模型可以定义为一个十元组：

$OntoModel:=\langle C,P,I,Rp,Hc,Hp,domain,range,\min card,\max card\rangle$

其中，C：本体的类集合。

P：本体的属性集合，可以区分为数据类型属性(DP)和对象属性(OP)。

I：本体的实例集合。

Rp：$Rp\subseteq P$，是一系列关系属性，包括对称属性、传递属性、函数属性和反函数属性等。

$Hc\subseteq C\times C$：为概念继承层次结构。如果$(c_1,c_2)\in Hc$，则 c_1 为 c_2 的子概念，而 c_2 为 c_1 的父概念。

$Hp\subseteq P\times P$：为无环的属性继承层次结构。如果$(p_1,p_2)\in Hp$，则 p_1 为 p_2 的子属性，而 p_2 为 p_1 的父属性。

函数 $domain:P\rightarrow 2^C$，给定了属性的定义域。

函数 $range:P\rightarrow 2^C\cup DR$，给定了属性的值域。其中，对象属性的值域是概念集合，而数据类型属性的值域为数值类型集合 DR。

函数 $\min card:C\times P\rightarrow N_0$，描述了每个概念-属性对的最小基数。

函数 $\max card:C\times P\rightarrow(N_0\cup\{\infty\})$，描述了每个概念-属性对的最大基数。

关系数据库模式和本体两种不同的语义模型，通过映射发现过程可以找到两种模型之间的语义关联，实现它们之间的互操作。映射是连接关系数据库和本体的桥梁，但是映射本身含有的语义信息非常有限，映射系统的大部分语义仍然蕴含于本体和关系数据库模式中。离开这两个参与映射的模型所构建的上下文环境，就无法理解映射所含有的语义。在此，将关系数据库模式、本体和它们之间的语义映射所构成的映射系统定义如下。

定义 3　关系数据库到本体的映射系统为一个三元组$\langle\mathcal{R},\mathcal{O},\mathcal{M}\rangle$，其中，$\mathcal{R}$ 为一个关系模式集合，$\mathcal{O}$ 为一个基于 *OntoModel* 本体模型的领域本体，$\mathcal{M}$ 为关系模式到本体的语义映射集合。

在动态环境下，映射系统中某一点上的变化都可能对系统中的其他要素产生影响，进行失效映射的检测和维护时必须将映射系统作为整体来考查。Yuan An 等人使用 Horn 子句描述关系数据库到本体的语义映射，并且对基于这种映射进行的查询处理做了初步的论述[6]。本文的重点不在于找出一种新的映射表达形式，而在于如何在动态、开放式环境下检测这种语义映射是否失效。

2.3　MAPONTO 与映射维护

加拿大 Toronto 大学 2005 年开发的 MAPONTO 系统是一种通过人机交互半自动生成映射的工具。当进行关系数据库模式和本体的映射时，系统以 OWL 本体文件和 SQL DDL 文件为输入，用户可以手工生成一组从关系数据库模式中的元素到本体实体的对应(correspondence)，然后系统会以一阶逻辑公式的形式输出语义映射表达式。利用 MAPONTO 对其实验数据集中 univ-cs 本体和 UTCSDB 数据库模式进行映射，可以得到如下的公式：

$$\begin{aligned}\text{Student}(&name,position,email,phone,supervisor)\rightarrow\mathcal{O}:\\&\text{Student}(x_1)\wedge\text{Professor}(x_2)\wedge\text{University}(x_3)\\&\wedge\text{advisor}(x_1,x_2)\wedge\text{affiliatedOf}(x_1,x_3)\wedge\text{personName}(x_1,name)\wedge\text{orgName}(x_3,position)\wedge\\&\text{personName}(x_2,supervisor)\wedge\text{emailAddress}(x_1,email)\wedge\text{telephoneNumber}(x_1,phone)\end{aligned}$$

与其他映射工具相比，MAPONTO 产生的映射表达式含有非常丰富的语义信息，是一种“集成电路”式的语义映射。这种映射“集成”了多个关系模式元素和本体实体之间的语义关联，但是不便于维护和重用[1]。Y. An 等人随后意识到这一问题，提出了一种“双向工程”的映射维护算法[9]，从 action 轴和 effect 轴两个方向来刻画 CM(本体)和数据库模式的变化情况，并且给出了进行映射维护的基本方法和思路。但是，在文献[2,9]中并没有对映射失效检测方法进行说明。

3　失效语义映射的检测方法

3.1　映射的失效检测与维护问题

在基于本体的数据集成和数据交换系统中，借助关系数据库和本体之间的语义映射可以实现：

(1)关系数据库实例到本体实例的转换；

(2)本体查询到 SQL 查询的转换。

为了在语义 Web 上发布关系数据库中的数据，通常可以将关系数据库的内容“dump”到与之映射的本体上[5]。因此可以从实例转换的角度来定义映射的有效性。

定义 4　假定存在关系数据库到本体的映射 $m\in\mathcal{M}$，关系模式 $\mathcal{R}$ 上的查询 Q_s 的结果集为 $\mathcal{I}$，本体中类 C 上的查询 Q_o 的结果集为 $\mathcal{J}$，如果对于关系模式的任意合法实例 $I\in\mathcal{I}$，均存在 C 的合法实例 $J\in\mathcal{J}$，则认为映射 m 是有效的。

关系数据库模式到本体的语义映射所含有的语义分别来自于关系数据库模式和本体，通常映射失效问题主要是由关系数据库模式和本体的演化造成的，因此可以通过捕捉关系

数据库模式的变化和本体的变化来缩小失效映射的搜索范围。本文假设在初始状态下关系数据库和本体之间的语义映射均有效，只有当数据库模式和本体发生演化时才进行映射失效检测和映射维护。

此外，语义映射的失效检测还应该和映射维护方法紧密相关。增量维护方式[10]是比较传统的映射维护方法，在每次发生变化时都需要进行映射的维护。文献[11]认为增量维护的方法效率较低，而且不能保证映射维护的结果满足实际需求，因此提出通过映射组合的方法来实现映射维护。基于这种映射组合的方法来维护语义映射，并不需要知道模式演化和本体演化过程中究竟是哪些元素实际发生了变化，只需要知道当前版本的模式和本体与原先版本的模式和本体究竟存在哪些不同即可。这样，就可以摆脱映射维护过程对数据库日志文件和本体演化日志文件的依赖，即使日志文件损坏或者没有日志文件的支持，也同样能够进行映射的维护。

此外，根据定义 4 可知，考查语义映射失效与否应当从实例的角度入手，需要设计相应的查询测试集合对确定出来的失效映射集合进行验证。

3.2 模式演化条件下的变化检测

数据库模式的演化问题很常见，著名的 Wikipedia 曾经在 4 年半的时间里，数据库模式发生了 170 多次变更[12]。在数据库模式演化过程中，表、列和约束等元素均可能发生变化。可能进行的变更操作有数据库模式元素的添加、删除、合并、分割、移动、拷贝、重命名等。根据本文给出的关系模式定义，可以把表划分为列和约束的集合。这样，任何一个数据库模式变更操作都可以写成一组属性列和约束的添加和删除操作。在 MAPONTO 的映射过程中仅考虑了主键约束和外键约束，本文在维护过程中也仅考虑这两种约束。具体来说，对数据库模式所做的任何操作都可以分解为表 1 所列的原子操作的集合。

表 1　数据库模式变更的原子操作

操作	符号	操作	符号
添加表名	AddTName	删除表名	RemoveTName
添加属性列	AddAtt	删除属性列	RemoveAtt
添加主键约束	AddCPK	删除主键约束	RemoveCPK
添加外键约束	AddCFK	删除外键约束	RemoveCFK

需要说明的是，表 1 中的原子操作并不是实际对关系数据库进行的操作，因为映射维护时关注的是变化的元素，并不需要知道实际发生了哪些变更操作。在进行模式变化检测时，可以通过比较初始状态的模式和当前模式之间的差别，找出已经发生变化的元素，构建一条“合理”的原子操作序列，则该序列就可以看作是关系数据库模式的一条合理演化路径。可以通过下述算法找到模式中发生改变的元素集合。

Algorithm　确定关系模式变化的元素集合

Input：*oldSchema*，*newSchema*，

Output：*addSet*，*removeSet*

1)从 *oldSchema* 中任取一个表 $Table_i$，在 *newSchema* 中查找与之名称相同的表。如果能找到与之名称匹配的 $Table_j'$，则将 $Table_i$ 放入队列 *oldSet* 中，将 $Table_j'$ 放入队列 *newSet* 中；否则将 $Table_i$ 放入集合 *removeSet* 中。

2)对 *oldSchema* 中的所有表进行上一步的操作之后，如果 *newSchema* 中还有表，则将所有表添加到 *addSet* 中。

3)如果 *removeSet* 和 *addSet* 均不为空，对 *removeSet* 中的每一个表都在 *addSet* 中查找与其最相似的表，即如果两个表中有 n 个原子是相同的，则它们的相似度就记为 n。找出 n 值最大的表，然后如果 $\exists e \in Table_i$ 且 $e \in Table_j'$，则将 e 从集合 *removeSet* 中删除。如果 $\exists e \in Table_i'$ 且 $e \in Table_j'$，则将 e 从 *addSet* 中删除。

4)将 *oldSet* 和 *newSet* 中对应位置上的 $Table_n$ 和 $Table_n'$ 分解为模式树(如图 2 所示)进行比较。如果 $\exists e \in Table_n$ 且 $e \notin Table_n'$，则将 e 放入集合 *removeSet*；如果 $\exists e \in Table_n'$ 且 $e \notin Table_n$，则将 e 放入 *addSet* 中。

5)算法终止，得到最终的 *addSet* 和 *removeSet*。

在比较模式的各个原子节点时，除了节点名称之外，还要比较其他特征。例如，在比较属性列时，只有当列名和所属表名都相同时才认为是相同的；在比较约束时，只有当键名、所属表名、存在参照关系的表的名称均相同时才认为是相同的。

在此通过一个例子进行说明。假设模式演化需求为：将 Student 表 name 列上的主键约束去掉，插入 studentID 列并将其作为 Student 表的主键，则通过上述算法可以得到旧模式剩余元素集合 $removeSet = \{CPK(name)\}$ 和新模式剩余元素集合 $addSet = \{CPK(studentID), studentID)\}$。要想从 *removeSet* 出发得到 *addSet*，只需要对 *removeSet* 中的元素进行删除操作，对 *addSet* 中的元素进行添加操作，即

$$SchemaCh = RemoveCPK(name) \circ AddAtt(studentID) \circ AddCPK(studentID)$$

变化元素集合 $changSet = removeSet \cup addSet$。不过，该集合中出现的元素并不一定是实际施加了变更操作的元素。有了 *removeSet* 和 *addSet*，还可以进一步确定出从旧模式到新模式的映射。将该模式映射和已有的语义映射进行适当组合，就能够得到新的语义映射。本文得到该变化元素集合的最终目的是确定出受影响的映射以及映射受影响的位置，为进一步的映射维护提供依据。图 2 所示为将关系模式分解为模式树。

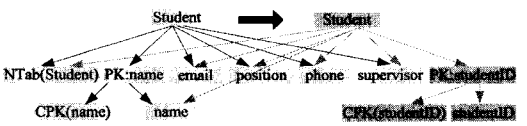

图 2　将关系模式分解为模式树

3.3 本体演化条件下的变化检测

与关系数据库模式相比，本体是一种语义更为丰富的模型，它的演化要比数据库模式的演化更为复杂[13]。因此在映射维护过程中，还需要考查本体演化对映射的影响。

从关系数据库到已有本体进行映射时，本体独立于关系数据库模式的领域本体，它的演化与数据库模式的演化也相互独立。对于本体的演化，本文也采用版本比较的方式进行变化检测，这样做的好处是在维护过程中通过变化元素集合可以很容易得到从旧本体到新本体的映射，进而通过映射组合实现对失效映射的维护。根据本文所依据的本体模型可以得到表 2 所列的本体原子变化操作集。

表 2　本体演化原子操作集

操作	添加	删除
概念	AddConcept	RemoveConcept
概念继承关系	AddSubConcept	RemoveSubConcept
属性	AddProperty	RemoveProperty
属性继承关系	AddSubProperty	RemoveSubProperty
实例	AddInstance	RemoveInstance
属性定义域	AddDomain	RemoveDomain
属性值域	AddRange	RemoveRange

MAPONTO 映射表达式中并不含有关于本体基数约束的信息，因此可以认为在本体演化的过程中对基数约束的变更对这种映射不产生影响。类和类的继承关系在本体中处于核心地位，据此可以得到图 3 所示的以类为中心的结构，该结构是本体图模型的一个子图。如果将本体模型分解为若干个形如图 3 所示的子图结构，那么本体新旧版本的比较就可以转换为对这些子图的比较。

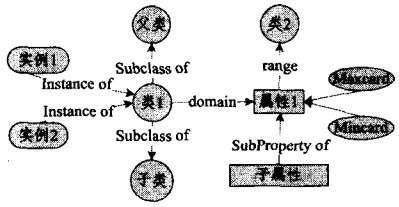

图 3　以类 1 为中心的本体模型子图

基于本体子图进行变化元素比较的算法流程如下。

Algorithm　确定本体变化元素集合

Input：*oldOntoModel*，*newOntoModel*

Output：*addSet*，*removeSet*

1）从 *oldOntoModel* 中任取一个类 C_i，在 *newOntoModel* 中查找与之名称相同的类。如果能够找到与之名称匹配的 C_j'，则将 C_j' 标记为 *match*；否则，将 C_i 标记为 *remove*。

2）对 *oldOntoModel* 中的所有类进行上一步的操作之后，将 *oldOntoModel* 中所有标记为 *remove* 的 C_i 添加到 *removeSet* 中，剩余元素添加到队列 *oldset* 中；将 *newOntoModel* 中没有被标记为 *match* 的类 C_j' 添加到 *addSet* 中，剩余元素添加到队列 *newSet* 中。

3）如果 *removeSet* 和 *addSet* 均不为空，则比较 *removeSet* 中的类和 *addSet* 中的类的相似度：比较两个集合中类的父类、属性列表、实例列表和子类列表。若有 n 个元素相同，则相似度为 n。为 *removeSet* 中的类 C_i 找出 n 值最大的 C_j'，分别添加到 *oldset* 和 *newSet* 中。

4）对于 *oldSet* 和 *newSet* 中对应位置上的类 C_n 和 C_n'，比较以二者为中心的子图 G_n 和 G_n'。如果 $\exists e \in G_n$ 且 $e \notin G_n'$，则将 e 放入集合 *removeSet*；如果 $\exists e \in G_n'$ 且 $e \notin G_n$，则将 e 放入 *addSet* 中。

5）算法终止，得到最终的 *addSet* 和 *removeSet*。

和前面给出的模式比较算法一样，上述算法终止后得到的 *addSet* 和 *removeSet* 之和即为本体演化过程中发生变化的元素。通过对 *removeSet* 中的元素进行删除操作，对 *addSet* 中的元素进行添加操作，即可得到一条合理的本体演化路径。

3.4　失效映射集合的确定

映射表达式中的元素均来自于关系数据库模式和本体，因此对于 $\forall m \in \mathcal{M}$，$\forall e \in changeSet$，如果有 $\forall e \in m$，则 m 可能为失效映射，所有可能失效的映射构成集合 $\mathcal{M}^*$。值得注意的是，对于 $\forall e \in addSet$，在维护过程中都需要通过人机交互为 e 添加新的对应。

由于在前一阶段得到的变化元素集合并不一定是实际发生变化的元素，因此还需要从实例的角度来验证 $\mathcal{M}^*$ 中的映射是否确实失效。在映射失效的判定过程中，测试查询集合的设计很关键，直接关系到失效判定的准确性。若已经得到了变化元素集合，则可以针对变化元素集合来设计测试查询集合。

当模式发生演化时，可以按照如下原则来设计查询：

Γ_1：当非外键列发生变化，则设计针对该列的查询，过滤条件加于变化的列上；

Γ_2：当外键发生变化，则设计针对具有参考关系的两个表的连接查询，过滤条件加于外键上；

Γ_3：当表名或主键发生改变，则进行整个表的查询，过滤条件加于主键上。

这样就可以得到一个 SQL 查询集合 $Q_s=\{Q_1,\cdots,Q_n\}$。根据原语义映射中的对应关系，可以将该查询集合重写为等价的本体查询集合 $Q_s'=\{Q_1',\cdots,Q_n'\}$。如果 Q_i' 无法在本体中查询到合法实例，则证明该映射已经失效。

当本体发生演化时，可以按照如下原则来设计本体查询：

Γ_4：当属性发生变化，则过滤条件加于变化属性上，否则过滤条件可以随意指定。

这样就可以得到一个 SQL 查询集合 $Q_o=\{q_1,\cdots,q_n\}$。根据原映射中的语义对应关系，可以将该查询集合重写为等价的 SQL 查询集合 $Q_o'=\{q_1',\cdots,q_n'\}$。如果 q_i' 无法在关系数据库中查询到合法实例，则证明该映射已经失效。

4　实验与总结

映射失效检测应当能够自动化。本文在实验过程中借助 JADE3.6 平台实现了两个 Agent，分别对模式演化和本体演化进行自动探测，进行版本比较并生成变化元素集合。实验采用 MAPONTO 实验数据集中的部分数据，如表 3 所列。

表 3　实验数据集和模拟操作

数据库模式	操作	本体	操作
3sdb2	添加/删除表	3sdb2	
VLDB	合并/分解表 添加/删除列	Conference	添加/删除概念
DBLP	添加/删除主键	Bibliographic	添加/删除属性
UTCS	添加/删除外键	univ-cs	

实验中使用 SPARQL 作为本体查询语言，SPARQL 查询和 SQL 查询的转换参见文献[14]。通过版本比较的方法无需对日志文件进行分析，仅需关注数据库模式与本体的初始版本和当前版本的差别，构建一条合理的演化路径，而无需关注演化过程中的实际操作。

作为一项探索性研究，本文提出的失效映射方法还有很多缺陷和不足。主要表现在两个方面：

(1)在模式和本体的变化检测过程中，即使新版本相对于旧版本变化很小，也需要进行完全的检测。如果关系数据库模式和本体中的元素数目较多，则检测效率相对较低。

(2)通过变化检测虽然缩小了失效映射的搜索范围，但是却有可能遗漏掉某些已经失效的映射。

映射的维护是一个迭代的过程，映射失效检测仅仅是其中一个重要的环节。在后续的工作中，将围绕上述两点不足进行改进，并且在此基础上探索这种复杂语义映射的维护问题。

参 考 文 献

[1] Konstantinou N, Spanos D, Mitrou N. Ontology and Database Mapping: A Survey of Current Implementations and Future Directions[J]. Journal of Web Engineering, 2008, 7(1): 1-24

[2] An Y, Topaloglou T. Maintaining Semantic Mappings Between Database Schemas and Ontologies[C]// Semantic Web, Ontologies and Databases. Berlin/Heidelberg: Springer, 2008: 138-152

[3] Mccann R, Alshebli B, Le Q, et al. Mapping Maintenance for Data Integration Systems[C]// Proceedings of the 31st VLDB Conference. Trondheim, Norway, 2005

[4] Trinh Q, Barker K, Alhajj R. Rdb2ont: A tool for generating owl ontologies from relational database systems[C]// the Advanced

International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW06). 2006
[5] Ghawi R, Cullot N. Database-to-Ontology Mapping Generation for Semantic Interoperability[C]// Very Large DataBase '07. Vienna, Austria, 2007
[6] An Y. Discovering and Using Semantics for Database Schemas [D]. University of Toronto, 2007
[7] Barrasa J, Corcho O, Gómez-pérez A. R2O, an Extensible and Semantically Based Database-to-ontology Mapping Language [C]//2nd Workshop on Semantic Web and Databases. 2004
[8] Stojanovic L. Methods and Tools for Ontology Evolution[D]. University of Karlsruhe, 2004
[9] An Y, Hu X, Song I. Round-Trip Engineering for Maintaining Conceptual-Relational Mappings[C]//CAiSE'08. 2008
[10] Velegrakis Y, Miller R J, Popa L. Mapping Adaptation Under Evolving Schemas[C]// Proceedings of the 29th VLDB Conference. Berlin, Germany, 2003
[11] Yu C, Popa L. Semantic Adaptation of Schema Mappings when Schema Evolve[C]// Proceeding of the 31st VLDB Conference. 2005
[12] Almeida R B, Mozafari B, Cho J. On the evolution of wikipedia [C]// International Conference on Weblogs and Social Media (ICWSM'2007). Colorado, USA, 2007
[13] Noy N F, Klein M. Ontology Evolution: Not the Same as Schema Evolution[J]. Knowledge and Information System, 2004, 6(4): 428-440
[14] Sparql2sql[EB/OL]. http://jena.sourceforge.net/sparql2sql

(上接第 158 页)

客户端参考词库则提取了搜狗互联网词库(SogouW)[12]中被标识为 N(名词)、V(动词)、ADJ(形容词)和 ADV(副词)的任意一种类型,且词频在 1 亿 3 千万以上的 3014 条双字词汇。

试验从客户端参考词库中随机抽取词汇构造查询进行学习,以能正确返回记录的主键为成功标记。学习完成后开始自动提取数据。

碰撞处理:试验使用的是新闻数据库,由于数据库内容的原因,出现了一个请求可以返回多条数据库记录的情况,对于这种情况也按照返回一个正确结果计量,在匹配重复时按照主键最大的一条数据进行判断。

考虑到服务器端程序对结果可能产生的影响,针对在服务器端使用模糊匹配、精确匹配两种方式处理客户端提交数据的情形,分别进行了实验。

在服务器端使用模糊匹配时的结果如图 2 所示。

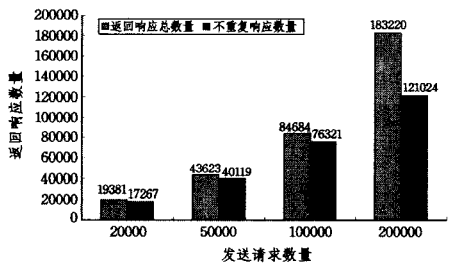

图 2 服务器端使用模糊匹配时算法返回结果数量

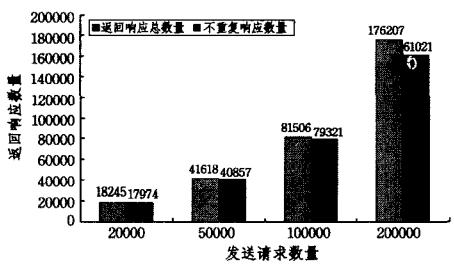

图 3 服务器端使用精确匹配时算法返回结果数量

可以看到,在客户端使用模糊匹配时,算法有较好的抽取效率,但是随着发送请求数量的增加,返回结果重复的情况也随之增加,这是因为模糊匹配可能导致多个请求被映射到同一条 Wdb 元组。

在服务器端使用精确匹配时的结果如图 3 所示。

当在服务器端使用精确匹配时,返回结果数量较使用模糊匹配时有所下降,但是算法的有效性增大,当客户端发起 20 万条抽取请求时,程序的有效性由使用模糊匹配的 60.5% 增加到了 80.5%。因此,该方式具有一定的实用价值。

结束语 随着 Deep Web 的迅速发展,针对 Deep Web 的数据抽取越来越成为一个研究的热点。每个 Deep Web 数据源都是一个完全自治的系统,因此给基于 Deep Web 的数据抽取带来了一定的困难。本文给出了一种用于 Deep Web 数据抽取的在本体支持下的机器学习算法,通过对 Deep Web 查询接口中的各个属性根据类型进行分类,进而采用不同的构造策略,在领域本体库的支持下可以自动地构造相应的请求字符串来完成 Deep Web 中数据的抽取。实验结果表明,本文提出的算法有较高的效率和准确率,可以以较小的代价获得较高质量的数据。

参 考 文 献

[1] Chang K C-C, He B, Li C, et al. Structured databases on the web: Observations and implications[J]. SIGMOD Record, 2004, 33(3): 61-67
[2] 刘伟,孟小峰,孟卫一. Deep Web 数据集成研究综述[J]. 计算机学报,2007,30(9):1475-1489
[3] Zheng Z, He B, Chan K C-C. Understanding Web query interfaces: Best-effort parsing with hidden syntax[C]// Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. 2004: 107-118
[4] 刘伟,孟小峰,凌妍妍. 一种基于图模型的 Web 数据库采样方法[J]. 软件学报,2008,19(2):179-193
[5] 马安香,张斌,高克宁,等. 基于结果模式的 Deep Web 数据抽取[J]. 计算机研究与发展,2009,46(2):280-288
[6] 陈鹏,刘烈宏. 深度 Web 资源搜索关键技术[J]. 北京航空航天大学学报,2009,35(1)
[7] Liu W, Meng X, Meng W. Vision-based Web Data Records Extraction[C]// Proceedings of the 9th International Workshop in Web and Databases. New York: ACM, 2006: 20-25
[8] Zhai Y, Liu B. Web Data Extraction Based on Partial Tree Alignment[C]// Proceedings of the 14th international Conference on World Wide Web. New York: ACM, 2005: 76-85
[9] 梅家驹,竺一鸣,高蕴琦,等. 同义词词林[M]. 上海:上海辞书出版社,1983
[10] 哈尔滨工业大学信息检索研究中心[OL]. http://ir.hit.edu.cn/. 2009
[11] 搜狗全网数据库(SogouCA)精简版[OL]. http://www.sogou.com/labs/dl/ca.html 2007
[12] 搜狗互联网词库(SogouW)[OL]. http://www.sogou.com/labs/dl/w.html 2007