

基于领域本体和关系模型的 XML 语义集成方法

李华昱¹, 欧阳纯萍², 徐九韵¹

(1. 中国石油大学(华东) 计算机与通信工程学院, 山东 东营 257061; 2. 北京科技大学 计算机与通信工程学院, 北京 100083)

(lhyzj@upc.edu.cn)

摘要: 由于缺乏足够的语义信息, 不同模式的 XML 数据之间很难进行互操作。针对油气井工程中的 XML 数据集成需求, 借助领域全局本体, 提出一种模式无关的 XML 语义集成方法。该方法首先在 XML Path 路径与领域本体之间进行语义映射, 屏蔽其模式差异; 然后, 按照模型映射方法将 XML 存储为关系数据; 最后通过查询重写将 SPARQL 转换为 SQL 语句, 实现语义查询。该方法对 XML 模式进行语义标注, 利用关系数据库存储与查询 XML 数据, 能有效处理领域 XML 数据的语义集成。

关键词: 领域本体; XML 模式; 语义映射; 语义查询; 油井工程

中图分类号: TP391 **文献标志码:** A

Semantic integration method for XML data based on domain ontology and relational model

LI Hua-yu¹, OUYANG Chun-ping², XU Jiu-yun¹

(1. College of Computer and Communication Engineering, China University of Petroleum (East China), Dongying Shandong 257061, China;

2. College of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)

Abstract: Due to the lack of sufficient semantic information, it is difficult to interoperate among XML data files with different schemas. Facing the needs to integrate XML data in the field of well-engineering, this paper proposed a model-independent semantic integration method for XML data using domain global ontology. This method built semantic mapping between domain ontology and paths of XML schema to shield the pattern differences; then, XML data was stored as relational data using model-mapping method; finally, the SPARQL query was rewritten into SQL statements to realize semantic query. By semantic annotating of XML schema and using relational database to store and query XML data, this method solves the semantic integration for domain XML data effectively.

Key words: domain ontology; XML schema; semantic mapping; semantic query; well engineering

0 引言

作为信息表示和数据交换的一个重要标准, XML 是科学数据的一种重要来源和表现形式, 用户可以利用 XML 简单、易扩展的优点, 自由定制其组织结构。然而, 这种灵活性容易引起相同语义的 XML 数据具有不同的模式结构或相反的情况, 导致 XML 数据之间、XML 与其他数据格式之间很难进行互操作。此外, XML 模式缺乏足够的语义信息, 无法支持基于领域术语的语义查询。为此, 迫切需要进行语义集成, 为用户提供透明、统一和基于语义的数据访问接口。

目前, 利用语义 Web 和本体技术是解决 XML 语义集成的有效方法, 并产生了一些研究成果。Bohring 等人提出的 XML2OWL^[1] 集成框架可以自动地从 XML 模式或文档中抽取 OWL 本体模型, 也支持将 XML 数据转换为 OWL 实例数据。Ferdinand 等人^[2] 提出了将 XML 提升至 RDF 和将 XML 模式映射为 OWL 的两种独立方法。Lehti 等人^[3] 将 OWL 本体作为全局模式, 在 XML 模式与全局本体之间建立映射, 但并不对实例数据进行转换, 在进行查询时, 将针对于 OWL 本体的查询转换为 XQuery 后, 再对 XML 进行查询。Cruz 和 Xiao 等人提出的集成方案^[4-5] 首先对每个 XML 数据产生一

个资源描述框架(Resource Description on Framework, RDF) 本体, 然后再将局部本体合并生成全局本体, 并通过一个映射表记录全局本体和各局部本体之间的映射关系, 最后, 将基于全局本体的 RDF 查询转换为针对 XML 的 XQuery 查询。文献[6]提出了一种利用模式图模型进行语义映射的算法, 该算法通过计算概念间的匹配相似度对 XML 模式进行语义标注, 实现 XML 语义集成。

作为关系数据的重要补充, 油井工程领域中存在若干个 XML 模型, 如石油标记模型 PetroXML、测井标记模型 WellLogML、地球物理标记模型 GeophysicsML 和井场信息传输标准语言 WITSML^[7] 等。为了充分利用这些异构 XML 数据资源, 为决策支持提供全面的数据支撑, 本文提出了一种基于领域本体的 XML 语义集成方法。该方法采用 XML 模式无关的映射方法, 将 XML 的 Path 路径与领域全局本体进行映射, 屏蔽了 XML 模式差异; 然后按照节点模型映射方法将 XML 转换为关系数据进行存储, 一方面可以与现有的关系存储模式相匹配, 另一方面也可以利用关系数据库的成熟技术, 提高 XML 的存储和检索效率; 最后, 通过查询重写算法将 SPARQL 查询转换为 SQL 语句, 实现对转存后 XML 数据的语义查询。

收稿日期: 2011-06-16; **修回日期:** 2011-08-06。 **基金项目:** 国家自然科学基金资助项目(60373008); 教育部科学技术研究重点资助项目(108008); 中央高校基本科研业务费专项资金资助项目(10CX01003A)。

作者简介: 李华昱(1977-), 男, 山东寿光人, 讲师, 博士, CCF 会员, 主要研究方向: 语义 Web、数据集成; 欧阳纯萍(1979-), 女, 湖南衡阳人, 讲师, 博士, 主要研究方向: 语义 Web、数据集成; 徐九韵(1966-), 男, 山东临沂人, 教授, 博士, 主要研究方向: Web 服务、数据信息系统。

1 实现架构

该方法采用3层实现架构:用户层、语义集成层和包装器层,如图1所示。

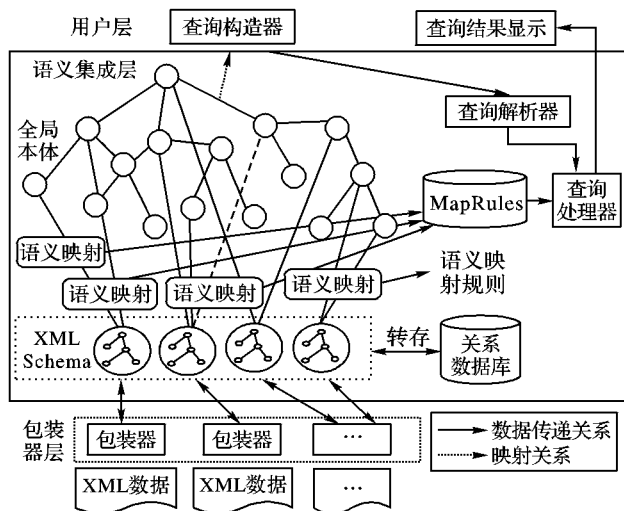


图1 整体结构

1) 用户层:提交语义查询需求和获取查询结果。其中,查询构造器负责将基于领域术语的查询请求转换为 SPARQL

语句并提交给语义集成层。

2) 语义集成层:是实现架构的核心,主要包括油气井工程领域全局本体(Well Engineering Ontology, WeOnto)、查询解析器、查询处理器、语义映射规则库和 XML-Relational 转存器等部件。WeOnto 描述了领域共享数据的抽象语义模型,是在领域数据专家指导下建立起来的。WeOnto 一方面为查询构造器提供语义查询对象标识,另一方面利用语义映射对 XML 模式进行语义标注,映射过程中产生的映射规则将被保存在语义映射规则库中。查询解析器负责解析 SPARQL 语句,并将解析后的信息提交给查询处理器,查询处理器利用该信息,通过查询重写算法,将 SPARQL 查询转换为 SQL 语句,从转存为关系数据的原 XML 数据中获取查询结果。

3) 包装器层:验证 XML 数据的有效性,获取 XML 的模式信息以及相关的元数据(如提交单位和日期等)。

2 XML 语义映射

XML 语义映射是语义集成的关键,其目的是在 WeOnto 与 XML 模式之间建立映射,以支持基于领域术语的语义查询。本文提出的语义映射方法是模式无关的,是将 XML 中的 Path 与 WeOnto 中的类或属性的层次结构进行映射,屏蔽了 XML 的模式差异,为用户提供统一和透明的全局语义查询视图。图2显示了 WeOnto 本体的部分结构。

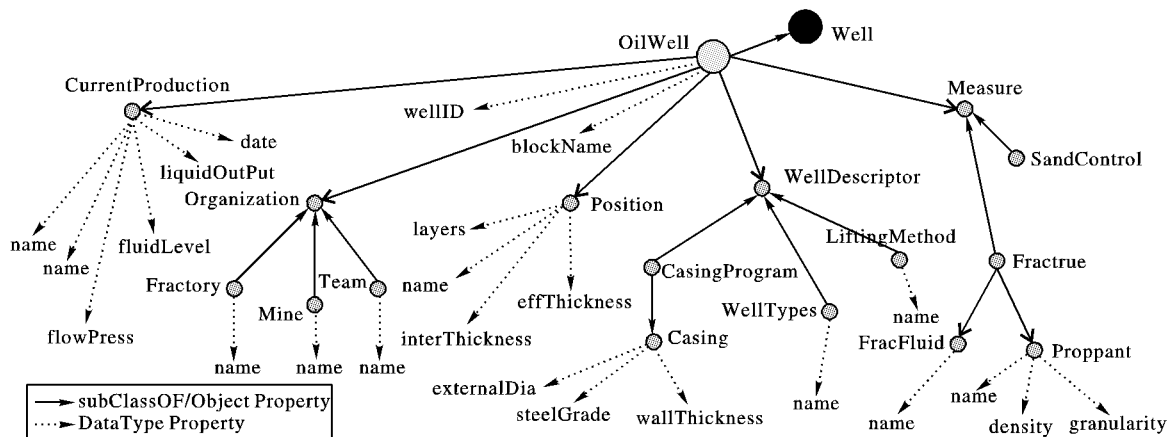


图2 WeOnto 全局本体(部分)

2.1 XML 模式表示

在油气井工程领域中,存在多种模式的 XML 数据,如果针对每一种模式建立一种映射方式,会导致映射关系复杂,无法进行有效管理。为此,采用 XML 的路径集合表示其数据模式,将 XML 中每一条 Path 路径映射为 WeOnto 中的一个查询路径(类或对象的层次结构),实现语义映射。

在 XML 模式中,Path 是由一个或多个定位步(Location Step)组成,定位步是由不同的节点构成,包括根节点、元素节点、属性节点和文本节点。Path 包括两种类型:绝对路径和相对路径,绝对路径是从文档树的根节点开始定位,而相对路径则是直接从某个定位步(即节点)开始定位。本文的 Path 指满足末端节点为文本节点或者属性节点的绝对路径,称为映射路径,记为 XPath。

参考文献[8],下面给出基于路径的 XML 模式定义。

定义1 基于路径的 XML 模式 XPathModel 是一个4元组,记为:XPathModel = <E, T, A, P>。其中:E表示所有元素;T表示所有文本元素,A表示所有属性元素,P表示映射路径 XPath。

图3为一个油井压裂措施的 XML 的模式结构,其

XPathModel 包含的两个 XPath 为:CYC#/QK#/@MC 和 CYC#/QK#/#HJ/CSLX#/@RQ。

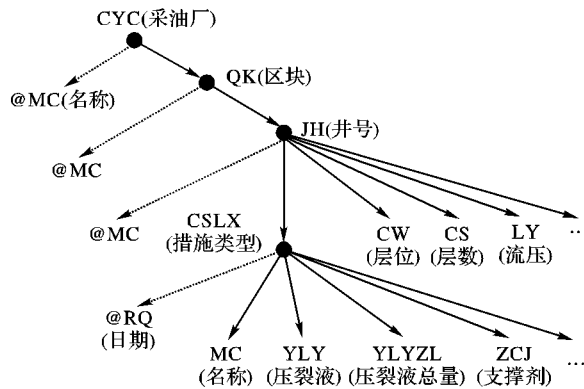


图3 压裂措施 XML 数据模型

2.2 XML 语义视图

采用 LAV(Local as Views)^[9] 映射方法对 XML 模式进行语义映射,将每一条 XPath 映射为 WeOnto 中的查询路径,由此,整个 XML 的 XPathModel 被映射为一系列语义视图,记为 X-SV(XML Semantic Views),X-SV 包括所有 XPath 路径的语

义映射信息。X-SV 定义如下:

定义2 $X\text{-SV}(P_1, P_2, \dots, P_n) \leftarrow \text{RDF}(X, Y)$ 。其中, P_i 是 XML 中的映射路径, 即 XPath; $\text{RDF}(X, Y) = \{ \text{RDF}_{P_i}(X, Y) \mid \forall \text{RDF}_{P_i} = \{ \text{rdf}_1, \text{rdf}_2, \dots, \text{rdf}_n \} \}$, RDF_{P_i} 为 rdf 三元组集合, RDF_{P_i} 以一系列的 rdf 三元组对映射路径 P_i 进行定义; $X = \{ X_1, X_2, \dots, X_n \}$ 为映射变量, X_i 对应与 WeOnto 中的类; Y 为 WeOnto 中的数据类型属性, 对应映射路径 P_i 中的末端节点。

定义3 给出了 RDF_{P_i} 的形式化定义:

定义3 RDF_{P_i} 是 rdf 三元组集合: $\text{RDF}_{P_i}(X, Y) = \{ \text{rdf}_1: \langle ? X_1, \text{rdf: type, WeOnto: StartClass} \rangle, \text{rdf}_2: \langle ? X_2, \text{rdfs: subclassOf} ? X_1 \rangle \mid \langle ? X_1, : \text{objAtt} ? X_2 \rangle, \text{rdf}_3: \langle ? X_2, \text{rdf: type, WeOnto: } X_2 \text{Class} \rangle, \dots, \text{rdf}_n: \langle ? X_{n-1}, \text{WeOnto: dataTypeAtt}, ? Y \rangle \}$ 。其中: $\text{rdf}_1: \langle ? X \text{ rdf: type WeOnto: StartClass} \rangle$ 表示从 WeOnto 的某个类开始定位; rdf_n 表示映射路径 P_i 的末端节点 Y 映射为类 X_{n-1} 的某个数据类型属性, 即 $Y \in \text{Range}(\text{dataTypeAttOf} X_{n-1})$; $\forall \text{rdf}_{2i} (i = 1, 2, \dots, k), 2i < n$, 表示类 X_{2i} 和 X_{2i-1} 的关系: X_{2i} 为 X_{2i-1} 子类或 X_{2i} 为 X_{2i-1} 的对象属性; $\forall \text{rdf}_{2i+1} (i = 1, 2, \dots, k), 2i < n$, 表示 X_{2i+1} 在 WeOnto 中的类标识。

X-SV 以语义查询的形式定义了一种映射方式, 表示要定位到 XPath 中末端节点需要在 WeOnto 中构造的查询语句表达式。如 FractureInfo.xml 中的 XPath: CYC#/@ MC 对应的 X-SV 为:

$\{ ? x \text{ rdf: type : OilWell. } ? x : \text{organization} ? y. ? y \text{ rdf: type : Organization. } ? y : \text{factory} ? z. ? z \text{ rdf: type : Factory. } ? z : \text{name} ? \text{MC} \}$

基于 X-SV 形式化定义, 以下给出 X-SV 的实例化表示结构。

定义4 X-SV 实例化结构记为 X-SVL(X-SV List):

$X\text{-SVL} = \{ \text{Node} \mid \text{Node} = (\text{pnode}, \text{node}, \text{cnode}). \text{Node}, \text{pnode}, \text{node}, \text{cnode} \in C \cup P \}$ 。C、P 表示 WeOnto 的类和属性。

1) X-SVL 是一个有序节点集合, 由若干个 Node 元素构成; Node 是一个三元组, 表示 WeOnto 中的类或数据类型属性, pnode、node 和 cnode 的有序关系表示 Node 在 WeOnto 的层次, pnode 为 node 的父类, cnode 为 node 的子类或数据类型属性。

2) Node 具有两种类型节点: NodeC 表示类节点, NodeDTP 表示数据类型属性节点 (Data Type Property, DTP)。

3) $\forall \text{Node}_i, \text{Node}_{i+1}, \text{Node}_i = (\text{pnode}_i, \text{node}_i, \text{cnode}_i), \text{Node}_{i+1} = (\text{pnode}_{i+1}, \text{node}_{i+1}, \text{cnode}_{i+1})$, 满足: $\text{cnode}_i = \text{Node}_{i+1}$ 。

如 XPath: CYC#/QK#/@ MC 的 X-SVL 为:

$\{ \text{NodeC}_{\text{OilWell}} (\# \text{Well}, \# \text{OilWell}, \text{NodeDTP}_{\text{blockName}}), \text{NodeDTP}_{\text{blockName}} (\text{NodeC}_{\text{OilWell}}, : \text{blockName}) \}$

3 XML 数据存储与解析

作为一种存储方式, XML 具有自描述、可移植等特点, 能够以树型或者层次结构描述数据; 但是, XML 也存在缺点, 如模式复杂多样、需要对其进行解析和文本转换, 从而导致数据访问速度较慢; 此外, 针对 XML 的查询语言要求用户了解 XML 文档的结构和语法格式, 这在一定程度上限制了 XML 数据的使用效率, 因此, 需要提供一种有效的 XML 数据管理方式。针对集成中的 XML 存储问题, 利用基于前缀编码的节

点模型映射方法, 将 XML 数据转存为关系模型, 采用 SQL 语言对转存后的 XML 数据进行查询, 借助关系数据库查询优化等技术, 提高存储和查询效率。

3.1 XML 关系存储模型

基于关系模式存储 XML 数据是将 XML 数据分解到若干个关系表中, 从而将 XML 的查询转化为一列针对关系模型的查询。然而, XML 模式与关系模式在语法、结构和约束等方面存在很多差异。因此, 将 XML 数据存储为关系模型需要经过解析和映射过程进行实现。

将 XML 数据映射为关系模式, 有两类映射方法: 模型映射 (Model Mapping) 和结构映射 (Structure Mapping)。模型映射将 XML 树结构映射为关系模式, 该方法对于所有 XML 文档都有固定对应的关系模式, 是 XML 模式无关的; 结构映射将 XML 模式映射为关系模式以表示其逻辑结构, 是模式相关的。

Yoshikawa 等人利用节点模型映射方法提出了一个 XML 的关系存储模式, 称为 XRel^[10]。XRel 通过区间编码表示 XML 文档层次结构, 该模式由 Element、Attribute、Text 和 Path 4 个关系表组成, 分别存储元素、属性、文本和路径信息。在 XRel 方法基础上, 提出一种基于前缀编码的模型映射方法, 记为 PreCode-XRel。本方法采用前缀编码对元素、文本和属性进行编码, 其关系模式包括如下 4 个表:

Element (nodeID, pathID, prefixID, parentID);
Attribute (nodeID, pathID, prefixID, parentID, value);
Text (nodeID, pathID, prefixID, parentID, value);
Path (pathID, pathExp)。

prefixID 字段表示元素 (文本或属性) 在文档树结构中的层次编码, 它将一个节点的双亲编码作为该节点的编码前缀。例如, 节点 node 的 prefixID 为 prefix (node), 则其孩子节点 nodeChild 的 prefixID 编码为 prefix (nodeChild) = prefix (node). n, n 是 nodeChild 在 node 所有孩子节点中具有相同路径的序号。通过 prefixID 可以快速判定 XML 数据中任意两个节点间的结构关系。

3.2 XML 数据解析

对 XML 数据按照关系模式进行存储, 首先需要对其进行解析。采用先序遍历算法对 XML 树结构和数据进行解析, 并将解析后的信息保存在以上 4 个关系表中; 然后, 利用 PrefixID-Coding 算法对所有元素进行前缀编码。整个解析过程包括以下步骤。

1) 读入 XML 文档, 利用先序遍历算法对其进行解析, 根据节点类型分别记录节点的先序遍历序号、父节点的先序遍历序号、节点路径、文本节点值和属性节点值等信息。

2) 将以上信息分别存储在 Element、Attribute、Text 和 Path 表中, 这些表存放在语义集成层中的关系数据库中。

3) 执行 PrefixID-Coding 算法, 在 Element、Attribute、Text 和 Path 表之间进行关系运算, 对各个节点进行前缀编码。图 4 显示了对 FractureInfo.xml 执行 PrefixID-Coding 算法后的各元素的前缀编码。

4 XML 语义查询

4.1 SPARQL 语句解析

SPARQL 解析的目的是将查询构造器构造的 SPARQL 语句中的 Select、Where 和 Filter 子句分别进行解析, 生成符合 X-SVL 定义的集合对象, 从而能够与 X-SV 语义视图进行匹

配,实现语义查询。生成的集合对象具有两种类型: MapColumn 和 FilterColumn,分别对应查询的数据项和筛选条件。该解析算法需要以下 4 个辅助关系表。

1) WhereInfo(CID, Property, Value):保存 Where 子句中出现的绑定变量、属性或类型标识符以及对应的值。

2) SelectTable(CID, Type):保存 Select 子句中出现的绑定变量及其类型(如类绑定变量和属性绑定变量)。

3) ClassTable(CID, Type):保存 WhereInfo 表中的类绑定变量及该变量在 WeOnto 中对应的类。如果某个绑定变量对应一个类层次关系,则只保存最底层的类。

4) ConditionTable(CID, Property, Compar, Value):保存 Filter 子句出现的属性绑定变量、变量对应的类名称、条件设置和数值。

SPARQL 解析算法如下所示。

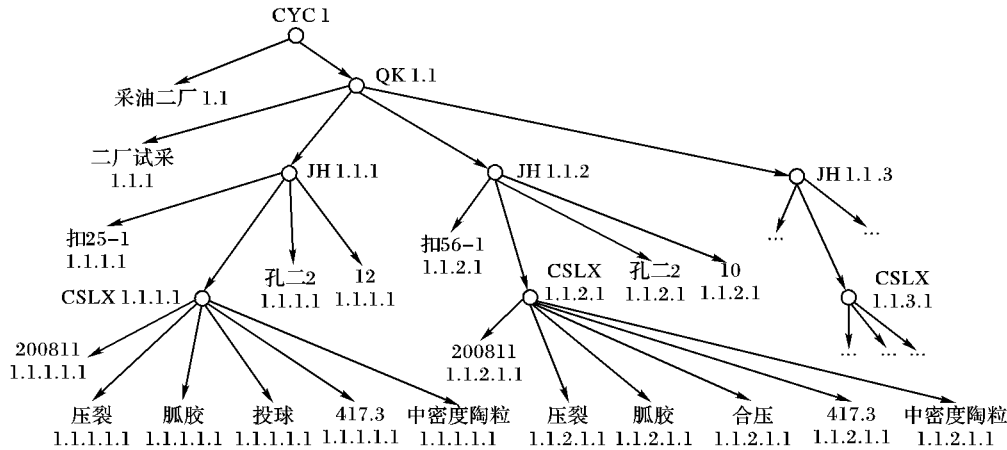


图4 压裂措施 XML 数据前缀编码

Input: SPARQL 语句 Q

Output: Q 对应的 MapColumn 和 FilterColumn 集合

/* 生成 MapColumn 集合实例 */

```

1) 对 Q 进行解析,将解析后的信息保存到上述 4 个表中
2) for ( int i = 0; i < SelectTable. rowCount(); i++)
3) { String binding = SelectTable. getValueAt( i, 0);
    //获取元组中的绑定变量
4) String mark = SelectTable. getValueAt( i, 1);
    //获取 binding 的类型标识
5) if mark. equals( "ClassType") { //判断是否是类绑定变量
6) OntClass ontClass = ClassTable. find( binding);
    //获取 binding 对应的类
7) Iterator iter = ontClass. listDTP();
    //获取 ontClass 的属性 DTP
8) While ( iter. hasNext())
    MapColumn mapc = new MapColumn( iter. next());
    //为每个 DTP 构造 MapColumn 实例
9) }
10) if mark. equals( "PropertyType") {
    //判断是否是属性绑定变量
11) OntClass ontClass = ClassTable. find( binding);
    //获取该属性对应的类
12) MapColumn mapc = new MapColumn( ontClass, binding); }
/* 生成 FilterColumn 集合实例 */
13) for ( int i = 0; i < conditionTable. rowCount(); i++)
14) { String cName = conditionTable. getValueAt( i, 0);
    //获取类名称
15) String dtpName = conditionTable. getValueAt( i, 1);
    //获取属性名称
16) String comName = conditionTable. getValueAt( i, 2);
    //获取比较类型
17) String value = conditionTable. getValueAt( i, 3);
    //获取比较数值常量
18) FilterColumn fc =
    new FilterColumn( cName, dtpName, comName, value); }

```

4.2 SPARQL 语义查询重写

查询重写是利用 X-SV 定义,将全局语义查询重写为转为关系模式的 XML 数据的 SQL 查询。基于 X-SVL 存储结

构,SPARQL 语义查询重写算法包括如下 4 个步骤:

1)按照 4.1 节的解析算法对查询 Q 进行解析,解析后 Q 的 MapColumn 实例集合记为 $Q_{\text{MapColumn}}$; FilterColumn 实例集合记为 $Q_{\text{FilterColumn}}$ 。

2)将 XML 数据的 X-SVL 集合记为 CollX-SVL,每一个元素记为 $X\text{-SVL}_i$ ($0 \leq i \leq n, n = |\text{CollX-SVL}|$),将 $X\text{-SVL}_i$ 与 $Q_{\text{MapColumn}}$ 和 $Q_{\text{FilterColumn}}$ 进行包含关系判断:如果 $Q_{\text{FilterColumn}} \subseteq X\text{-SVL}_i$ 并且 $Q_{\text{MapColumn}} \subseteq X\text{-SVL}_i$,则将 $X\text{-SVL}_i$ 放入集合 CollSatable 中。

3)依次从 CollSatable 集合中选取 $X\text{-SVL}_i$,按照以下步骤完成查询重写:

① $\forall X\text{-SVL}_i \in \text{CollSatable}, X\text{-SVL}_i$ 中存在某些 Node 元素与 $Q_{\text{FilterColumn}}$ 的实例具有相同的映射路径,从 $X\text{-SVL}_i$ 中获取这些 Node 对应的 XPath 对象,每个 XPath 记为 pathID_i ;将 $Q_{\text{FilterColumn}}$ 实例中设置的条件值记为 value_i 。然后,构造集合 CollPathID 和 CollPathValue 分别存储 pathID_i 和 value_i 元素,即:

$$\text{CollPathID} = \{\text{pathID}_1, \text{pathID}_2, \dots, \text{pathID}_m\}$$

$$\text{CollPathValue} = \{\text{value}_1, \text{value}_2, \dots, \text{value}_m\}$$

②分别在 Text 和 Attribute 两个关系表中对 pathID_i 路径定义的条件数值进行关系运算,返回满足条件的 xID (xID 是 XML 文档的编号)和符合这些条件的元素对应的前缀编码。首先对 Text 表进行关系运算,结果记为 $Q_{\text{Text}}(\text{xID}, \text{prefixID})$,公式如下:

$$Q_{\text{Text}}(\text{xID}, \text{prefixID}) \leftarrow$$

$$\pi_{\text{Text1.docID}, \text{Text1.prefixID}} \sigma_{\text{expression1}} \text{Text1} \times \text{Text2}$$

其中: $\text{expression1} = ((\text{Text1.pathID} = \text{pathID1 and Text1.value} = \text{value1}) \dots \text{or } (\text{Text1.pathID} = \text{pathIDm and Text1.value} = \text{valuem}))$ and $\text{Text1.prefixID like Text2.prefixID}$ group by Text1.docID, prefixID; Text₁ 和 Text₂ 是 Text 表的两个别名。

然后,对 Attribute 表进行同样的关系运算,结果记为 $Q_{\text{Attribute}}(\text{xID}, \text{prefixID})$:

$$Q_{\text{Attribute}}(\text{xID}, \text{prefixID}) \leftarrow$$

$$\pi_{\text{Attribute1.docID, Attribute1.prefixID}} \sigma_{\text{expression2}} \text{Attribute}_1 \times \text{Attribute}_2$$

其中: $\text{expression2} = ((\text{Attribute1.pathID} = \text{pathID1 and Attribute1.value} = \text{value1}) \cdots \text{or} ((\text{Attribute1.pathID} = \text{pathIDm and Attribute1.value} = \text{valuem})) \text{ and Attribute1.prefixID like Attribute2.prefixID}) \text{ group by Attribute1.docID, prefixID}$; Attribute_1 和 Attribute_2 是 Attribute 表的两个别名。

③分别对 Q_{Text} 和 $Q_{\text{Attribute}}$ 结果集进行筛选:如果某个 xID 对应的 prefixID 的记录数目等于选择条件的数目 m , $m = |\text{Coll}_{\text{PathValue}}|$ 或者 m 的倍数,则该 xID 文档中可以作为候选的查询文档,即文档中可能具有满足查询条件的记录,并且这些记录的前缀编码与 prefixID 具有包含或者被包含的关系。首先,从 Q_{Text} 和 $Q_{\text{Attribute}}$ 结果集中删除不满足上述条件的记录,然后,从每个 xID 对应的每 m 个具有包含或者被包含关系的前缀编码中选择其中任意一项(如保留前缀编码最长的一项),记为 $\text{prefixID}'$,该编码将用于4)中的关系运算。

4)对 Attribute 表和 Text 表分别进行如下关系运算:

$$Q_{\text{Text}}' \leftarrow \pi_{\text{pathID, value}} \sigma_{(\text{pathID in CollPathID}) \cdots (\text{xID} = Q_{\text{Text.xID}})} \text{Text} \times Q_{\text{Text}}$$

$$Q_{\text{Attribute}}' \leftarrow$$

$$\pi_{\text{pathID, value}} \sigma_{(\text{pathID in CollPathID}) \cdots (\text{xID} = Q_{\text{Attribute.xID}})} \text{Attribute} \times Q_{\text{Attribute}}$$

Q_{Text}' 结果集中的记录形式为 $\{\text{pathID}_i, \text{value}_i\}$, 表示 pathID_i 对应的 XPath 中的末端文本节点是一个查询项,数值为 value_i ; 同样, $Q_{\text{Attribute}}'$ 结果集中的记录形式为 $\{\text{pathID}_i, \text{value}_i\}$, 表示 pathID_i 对应的 XPath 中的末端属性节点是一个查询项,数值为 value_i 。最后,这些记录组成最终查询结果。

4.3 算法分析与比较

SPARQL 解析算法包括两个解析过程:SPARQL 语句解析和 WeOnto 模型解析。前者采用文本解析方法进行处理,即读入整个 SPARQL 语句,然后根据 Select、Where 及 Filter 子句中的符号标识将逐个将类、属性和常量保存到 4 个关系表中,由于应用中的 SPARQL 文件大小普遍不超过 128 KB,所以该过程具有很高的执行效率。在进行 WeOnto 解析时,首先利用 Jena 和 Pallet 读入整个 WeOnto 模型,然后再根据关系表中的相关记录确定 WeOnto 中的类或属性。该过程运行效率主要取决于 WeOnto 本体的规模,目前的 WeOnto 1.0 版本中包含 63 个实体类,157 个属性和 247 个约束匿名类,通过解析函数中加入的计时功能,首次将 WeOnto 读入内存到解析结束,共耗费 0.392 s(实验机器 CPU 为 Intel Core 2 2.10 GHz, 1 GB Memory),后续的解析耗时 0.093 ~ 0.130 s。SPARQL 语义查询重写算法本质上是数据库 SQL 查询处理,其执行效率取决于数据规模和关系运算的复杂性(选择、联结和投影)。不同数据库中的两张 Text 表或者 Attribute 表进行联结操作时,处理方法是将其中的一张表复制到对方的数据库中去,然后再进行联结操作。这种处理方法生成的查询计划简单,在数据量较小的情况下,性能非常好。但是,当单表数据记录达到 10 万 ~ 100 万条时,执行耗时较大,为此,可以通过提高数据库性能及网络带宽,提升系统集成及查询性能。

针对 SPARQL-SQL 解析与查询, Chebotko^[11] 和 Harris^[12] 提出了相关的算法,与本文算法相同,它们采用特定的关系模型对解析信息进行存储,然而,它们并不支持 SPARQL 中的 Filter 子句解析;文献[13]提出的 Sparql2SQL of Jena 方法采用 Jena APIs 访问本体和关系模型,由于 Jena 效率局限性,系统性能不高。此外,以上三种方法都是对关系数据进行查询处理,并不支持 XML 数据的集成与查询。本文算法下一步的研究重点是实现 Union 子句的支持。

4.4 实例分析

以 FractureInfo.xml 文档为例,以下从 XML 解析、语义映射和语义查询 3 个过程给出实现过程:

1)解析 FractureInfo.xml,将元素、属性、文本和路径分别存储在 Element、Attribute、Text 和 Path 关系表中。解析后的信息为:具有 18 条绝对路径(包括 14 条 XPath 路径)、58 个节点(包括属性节点和文本节点)、8 个属性节点和 22 个文本节点。然后,在 FractureInfo.xml 与 WeOnto 之间建立映射,即构造 X-SV 和 X-SVL。语义映射如表 1 所示。

FractureInfo.xml 对应的 X-SV 定义为:

$$\text{X-SV}_{\text{FractureInfo}}(P_1, P_2, \cdots, P_{14}) \leftarrow$$

$\{? x \text{ rdf:type :OilWell. ? x :organization ? y. ? y rdf:type :Organization. ? y:factory ? z. ? z rdf:type :Factory. ? z:name ? MC\}, \{? x \text{ rdf:type :OilWell. ? x:blockName ? QK\}, \cdots, \{? x \text{ rdf:type :OilWell. ? x:measure ? y. ? y rdf:type :Fracture. ? y:propAgent ? ZCJ\}$

2)执行 SPARQL 解析算法,生成 MapColumn 和 FilterColumn 实例。如查询 Q:“查询在 2009 年 1 月份实施了压裂措施,并且该措施采用的是投球工艺,当前月产油量大于 40 吨的油井井号和该井的流压”。对应的 $Q_{\text{MapColumn}}$ 和 $Q_{\text{FilterColumn}}$ 实例为:

$\text{MapColumn}_{\text{WellID}} = \{ \text{NodeC}_{\text{OilWell}} (\# \text{ Well}, \# \text{ OilWell}, \text{NodeDTP}_{\text{wellID}}), \text{NodeDTP}_{\text{wellID}} (\text{NodeC}_{\text{OilWell}}, : \text{wellID}) \}$

$\text{MapColumn}_{\text{FlowPress}} = \{ \text{NodeC}_{\text{OilWell}} (\# \text{ Well}, \# \text{ OilWell}, \text{NodeC}_{\text{Output}}), \text{NodeC}_{\text{Output}} (\text{NodeC}_{\text{OilWell}}, \# \text{ Output}, \text{NodeDTP}_{\text{FlowPress}}), \text{NodeDTP}_{\text{FlowPress}} (\text{NodeC}_{\text{Output}}, : \text{flowPress}) \}$

$\text{FilterColumn}_{\text{FinishDate}} = \{ \text{NodeC}_{\text{OilWell}} (\# \text{ Well}, \# \text{ OilWell}, \text{NodeC}_{\text{Fracture}}), \text{NodeC}_{\text{Fracture}} (\text{NodeC}_{\text{OilWell}}, \# \text{ Fracture}, \text{NodeDTP}_{\text{finishDate}}), \text{NodeDTP}_{\text{finishDate}} (\text{NodeC}_{\text{Fracture}}, : \text{finishDate}, = '200901') \}$

$\text{FilterColumn}_{\text{MeasName}} = \{ \text{NodeC}_{\text{OilWell}} (\# \text{ Well}, \# \text{ OilWell}, \text{NodeC}_{\text{Fracture}}), \text{NodeC}_{\text{Fracture}} (\text{NodeC}_{\text{OilWell}}, \# \text{ Fracture}, \text{NodeDTP}_{\text{measName}}), \text{NodeDTP}_{\text{measName}} (\text{NodeC}_{\text{Fracture}}, : \text{measName}, = '压裂') \}$

$\text{FilterColumn}_{\text{FractType}} = \{ \text{NodeC}_{\text{OilWell}} (\# \text{ Well}, \# \text{ OilWell}, \text{NodeC}_{\text{Fracture}}), \text{NodeC}_{\text{Fracture}} (\text{NodeC}_{\text{OilWell}}, \# \text{ Fracture}, \text{NodeDTP}_{\text{fractType}}), \text{NodeDTP}_{\text{fractType}} (\text{NodeC}_{\text{Fracture}}, : \text{fractType}, = '投球') \}$

$\text{FilterColumn}_{\text{OilOutput}} = \{ \text{NodeC}_{\text{OilWell}} (\# \text{ Well}, \# \text{ OilWell}, \text{NodeC}_{\text{Output}}), \text{NodeC}_{\text{Output}} (\text{NodeC}_{\text{OilWell}}, \# \text{ Output}, \text{NodeDTP}_{\text{OilOutput}}), \text{NodeDTP}_{\text{OilOutput}} (\text{NodeC}_{\text{Output}}, : \text{oilOutput} > 40) \}$

3)将所有 X-SVL_i 与 $Q_{\text{MapColumn}}$ 和 $Q_{\text{FilterColumn}}$ 匹配,获取满足匹配条件的所有 X-SVL_i 。FractureInfo.xml 有 4 条 XPath 的 X-SVL 与 $Q_{\text{FilterColumn}}$ 实例相同,这 4 条路径为 P_{10} 、 P_{13} 、 P_{14} 和 P_{16} ;此外,FractureInfo.xml 文档中的 P_6 和 P_{11} 对应的 X-SVL 与 $Q_{\text{MapColumn}}$ 实例相同。因此,在第 4)步对 Q 进行查询重写。

4)构造 $\text{Coll}_{\text{PathID}}$ 和 $\text{Coll}_{\text{PathValue}}$ 集合,存储与 $Q_{\text{FilterColumn}}$ 实例匹配的 P_{10} 、 P_{13} 、 P_{14} 和 P_{16} 映射路径标识以及对应的选择条件,即:

$$\text{Coll}_{\text{PathID}} = \{10, 13, 14, 16\}$$

$$\text{Coll}_{\text{PathValue}} = \{> 40, = '压裂', = '200901', = '投球'\}$$

利用该信息,对 Text 表和 Attribute 表分别进行以下 SQL 运算,根据返回的 XML 文档编号 xID 和前缀编码 prefixID 判断哪些 XML 文档中存在满足条件的数据。SQL 语句如下:

Select t1.xID, t1.prefixID

From Text t1, Text t2

Where t1.pathID = t2.pathID And ((t1.pathID = 10 And Cast(t1.

```

value as float) > 40) Or (t1.pathID = 13 And t1.value =
'Fracture') Or (t1.pathID = 14 And t1.value = '200901') Or
(t1.pathID = 16 And t1.value = 'TouQiu')) And t1.prefixID
like t2.prefixID Order by t1.prefixID, t1.pathID
Select a1.xID, a1.prefixID
From Attribute a1, Attribute a2
Where a1.pathID = a2.pathID And ((a1.pathID = 10 And Cast
(a1.value as float) > 40) Or (a1.pathID = 13 And a1.
value = 'Fracture') Or (a1.pathID = 14 And a1.value =
'200901') Or (a1.pathID = 16 And a1.value = 'TouQiu')) And
a1.prefixID like a2.prefixID Order by a1.prefixID, a1.pathID
返回的结果记录如表 2 所示。

```

表 1 FractureInfo 映射信息表

PID	XPath	WeOnto 映射路径
P2	CYC#/@MC	OilWell. Organization. Fractory. name
P4	CYC#/QK#/@MC	OilWell. blockName
P6	CYC#/QK#/#HJ/@MC	OilWell. wellID
P7	CYC#/QK#/#HJ/NY	OilWell. OutPut. date
P8	CYC#/QK#/#HJ/CW	OilWell. Position. name
P9	CYC#/QK#/#HJ/CS	OilWell. Position. layers
P10	CYC#/QK#/#HJ/YCYL	OilWell. OutPut. oilOutPut
P11	CYC#/QK#/#HJ/LY	OilWell. OutPut. fluidPress
P13	CYC#/QK#/#HJ/CSLX#/@MC	OilWell. Measure. meaName
P14	CYC#/QK#/#HJ/CSLX#/@RQ	OilWell. Measure. finishDate
P15	CYC#/QK#/#HJ/CSLX#/YLY	OilWell. Measure. Fracture. FractFluid. name
P16	CYC#/QK#/#HJ/CSLX#/GYLX	OilWell. Measure. Fracture. fractType
P17	CYC#/QK#/#HJ/CSLX#/YLYZL	OilWell. Measure. Fracture. fluidTotal
P18	CYC#/QK#/#HJ/CSLX#/ZCJ	OilWell. Measure. Fracture. propAgent

表 2 FractureInfo. xml 查询信息

xID	prefixID	pathID
2009011201	1.1.1.1	10
2009011201	1.1.1.1.1	13
2009011201	1.1.1.1.1	14
2009011201	1.1.1.1.1	16
2009011201	1.1.2.1.1	13
2009011201	1.1.2.1.1	14
2009011201	1.1.2.1.1	16
2009011201	1.1.3.1.1	13
2009011201	1.1.3.1.1	14
2009011203	1.1.2.1.1	13
2009011203	1.1.2.1.1	16

表 2 表明,只有 xID 为“2009011201”、prefixID 为 1.1.1.1 或 1.1.1.1.1 对应 pathID 列的数值组成的集合等于 Coll_{pathID},即在“2009 年 1 月 12 日”提交的编号为“01”的 FractureInfo. xml 文档中具有满足条件的记录,这些记录的前缀编码与 1.1.1.1 和 1.1.1.1.1 具有“Like”关系,将‘prefixID’赋值为“1.1.1.1”。

5) 将 Q_{MapColumn} 实例与 X-SVL 进行匹配,返回的 XPath 编号为 6 和 11;基于第 4) 步的结果,分别对 Text 表和 Attribute 表构造如下 SQL 语句,获取最终查询结果:

```

Select pathID, value
From Text
Where (pathID = 6 Or pathID = 11)
And (prefixID like '1.1.1.1%' Or prefixID like '1.1.1.1.1%')
Select pathID, value
From Attribute

```

```

Where (pathID = 6 Or pathID = 11)
And (prefixID like '1.1.1.1%' Or prefixID like '1.1.1.1.1%')

```

两个 SQL 语句各返回一条查询结果,分别为 {6, 扣 25-1} 和 {11, 10.3}, 即满足查询条件的井号是“扣 25-1”,其 2009 年 1 月的流压为“10.3 Mpa”。

5 系统实现及应用

本方法系统架构基于 Java 技术实现。采用 JavaApplet 构造基于 Web 的用户查询界面,借助 Jena API 解析 WeOnto 本体并实现 XML 语义映射;利用 Dom4j 对 XML 进行解析并将解析后的数据存储在 SQL Server 2000 数据库中。

该系统已经在中石油大港油田进行了实际应用。通过对该油井日常生产、工艺措施和井下设备 3 种类型的 XML 数据进行语义集成,提供基于领域术语的语义查询功能,取得了良好的应用效果。

6 结语

针对油气井工程领域中的 XML 数据资源,从 XML 语义映射、XML 数据存储和语义查询 3 个方面进行了重点研究,提出了一种基于领域本体的 XML 语义集成方法。该方法在领域全局本体与 XML 的 Path 路径之间建立语义映射,提供统一、透明和基于语义的数据访问视图。为了对不同模式的 XML 数据进行有效存储,采用基于节点模型的映射方法,将 XML 数据转换为关系模型进行存储,并在此基础上提出了一种 SPARQL-SQL 语义查询重写算法。比较目前的集成方案,该方法结合关系数据库的存储、查询技术和本体提供的语义描述能力,为用户提供更加有效的异构 XML 集成与应用服务。下一步,将对语义视图定义的自动化和查询的语义保持方面进行研究。

参考文献:

- [1] BOHRING H, AUER S. Mapping XML to OWL ontologies[C]// Proceedings of the 13 Leipziger Informatik-Tage (LIT 2005), LNI72. Heidelberg: Springer, 2005: 147 - 156.
- [2] FERDINAND M, ZIRPINS C, TRASTOUR D. Lifting XML schema to OWL[C]// Proceedings of the International Conference on Web Engineering, LNCS 3140. Heidelberg: Springer, 2004: 776 - 777.
- [3] LEHTI P, FANKHAUSER P. XML data integration with OWL: experiences and challenges[C]// Proceedings of the 2004 International Symposium on Applications and the Internet. Washington, DC: IEEE Computer Society, 2004: 160 - 167.
- [4] CRUZ I F, XIAO H, HSU F. An ontology-based framework for XML semantic integration[C]// IDEAS'04: Proceedings of the International Database Engineering and Applications Symposium. Washington, DC: IEEE Computer Society, 2004: 217 - 226.
- [5] XIAO H. Query processing for heterogeneous data integration using ontologies[D]. Chicago: University of Illinois, 2006.
- [6] 王渊,陈玉. 一种 XML Schema 和 Ontology 间的语义映射算法[J]. 小型微型计算机系统, 2005, 26(11): 1988 - 1990.
- [7] KIRKMAN M A, SYMMONDS M E, HARBINSON S W, et al. Wellsite information transfer standard markup language[EB/OL]. [2011-02-20]. http://en.wikipedia.org/wiki/Wellsite_information_transfer_standard_markup_language.

(下转第 3267 页)

明显优于从网页整体分析链接优先级的爬行策略。这主要由于通过网页分块可先过滤掉部分噪声,其次在计算 CL 优先级时考虑了块优先级,避免了在只考虑页面内容优先级和锚文本优先级时,锚文本过短、信息量不足以明确刻画 CL 主题的缺点。

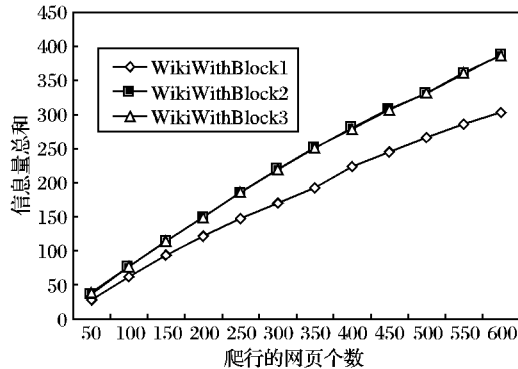


图3 实验2 信息量总和比较

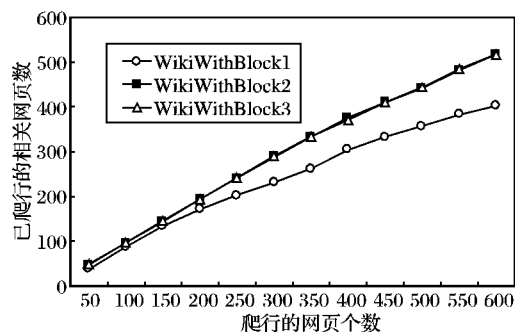


图4 实验2 查准率比较

由图3、4可知, N 取2、3时WikiWithBlock N 在信息量总和和查准率上明显优于 N 取1。这是因为 $N=1$ 时,只从分类树中提取了主题概念向下1层的概念,所以主题向量空间的大小 $|VS_1|=19$,而 $N=2$ 时 $|VS_2|=151$,使用了更多的概念来描述主题,这样当词语向 VS_i 中映射时,就会被映射到更加细化、与词语本身在实际中的意义更加相符的概念上,而 $N=1$ 时对主题的描述过于泛化,导致收集到的网页的信息量总和很少,查准率很低;而 $N=3$ 的 $|VS_3|=251$,比 $N=2$ 对主题的描述更加具体,但却在信息量总和和查准率上没有明显的改变,这是由于在主题分类子树中,一些在网页中常出现的概念,例如“CBA 球员”处在第二层,且相应的分枝中没有第三层分类,所以词语大多也只映射到第二层,导致 $N=3$ 与 $N=2$ 相比性能相近。这取决于分类树中属于不同主题的概念是否完整和如何组织,而另一方面则表明对主题描述的具体程度只需维持在一定的水平。

4 结语

本文基于 Wikipedia,对主题进行描述和计算主题相关性,并在爬行过程中引入网页分块,提出了一种基于 Wikipedia 和网页分块的主题爬行策略,有效改善了传统主题爬行策略中基于关键词集和从网页整体分析链接优先级的爬行策略的不足。实验表明,该策略在主题爬行过程中获得了良好的效果和明显的性能提升,并且表明了对主题描述的详细程度会对主题爬行的性能产生明显的影响。如何在爬行过程中自适应地学习主题向量空间中每个概念对主题描述的贡献度,即优化主题描述,是下一步的研究工作。

参考文献:

- [1] AGGARWAL C C, AL-GARAWI F, YU P S. Intelligent crawling on the world wide Web with arbitrary predicates[C]// Proceedings of the 10th International Conference on World Wide Web. New York: ACM, 2001:96-105.
- [2] DAVISON B D. Topical locality in the Web[C]// Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York: ACM, 2000: 272-279.
- [3] MENCZER F, PANT G, SRINIVASAN P. Topical Web crawlers: evaluating adaptive algorithms[J]. ACM Transactions on Internet Technology, 2004, 4(4): 378-419.
- [4] ZHENG HAI-TAO, KANG B Y, KIM H G. An ontology-based approach to learnable focused crawling [J]. Information Sciences, 2008, 178(23): 4512-4522.
- [5] SU CHANG, GAO YANG, YANG JIANMEI, *et al.* An efficient adaptive focused crawler based on ontology learning[C]// Proceedings of the 5th International Conference on Hybrid Intelligent Systems. Washington, DC: IEEE, 2005: 73-78.
- [6] 赵佳鹤,王秀坤,刘亚欣.基于语义分析的主题信息采集系统的设计与实现[J]. 计算机应用, 2007, 27(2): 406-408.
- [7] Wikipedia [EB/OL]. [2011-02-16]. <http://wikipedia.jaylee.cn/>.
- [8] STRUBE M, PONZETTO S P. WikiRelate! computing semantic relatedness using Wikipedia[C]// Proceedings of the National Conference on Artificial Intelligence. Cambridge: AAAI Press, 2006: 1419-1424.
- [9] 陈竹敏. 面向垂直搜索引擎的主题爬行技术研究[D]. 济南: 山东大学, 2008.
- [10] 中文维基百科资源[EB/OL]. [2010-11-09]. <http://dumps.wikimedia.org/zhwiki/>.
- [11] HERSOVICI M, JACOVI M, PELLEGG D, *et al.* The shark-search algorithm: an application: tailored Web site mapping[C]// Proceedings of the 7th World Wide Web Conference. Amsterdam: Elsevier Science, 1998: 317-326.

(上接第3263页)

- [8] AN Y, BORGIDA A, MYLOPOULOS J. Constructing complex semantic mappings between XML data and ontologies [EB/OL]. [2011-02-20]. <http://disi.unitn.it/~p2p/RelatedWork/Matching/iswc05.constructing.pdf>.
- [9] DUSCHKA O M, GENESERETH M R. Answering recursive queries using views [C]// Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. New York: ACM, 1997:109-116.
- [10] YOSHIKAWA M, AMAGASA T, SHIMURA T, *et al.* XRel: a path-based approach to storage and retrieval of XML documents using relational databases [J]. ACM Transactions on Internet Technology, 2001, 1(1):110-141.
- [11] CHEBOTKO A, LU S, JAMIL H M, *et al.* Semantics preserving SPARQL-to-SQL query translation for optional graph patterns. TR-DB-052006-CLJF [R]. Detroit: Wayne State University, Department of Computer Science, 2006.
- [12] HARRIS S. SPARQL query processing with conventional relational database systems[C]// Proceedings of Web Information Systems Engineering 2005. Berlin: Springer-Verlag, 2005:235-244.
- [13] Sparql2sql: A query engine for SPARQL over Jena triple stores [EB/OL]. [2011-02-14]. <http://jena.sourceforge.net/>