基于关系数据库的 OWL 本体存储与提取技术①

刘 侨, 王新房, 雷 鸣

(西安理工大学 自动化与信息工程学院, 西安 710048)

摘 要: WEB 本体语言(OWL)是一种用于对本体进行语义描述的语言, 不仅提供强大的语义表达能力, 而且能够表达机器可以理解的内容. 关系数据库在存储与管理大规模数据方面相应技术比较成熟. 针对该情况, 结合两者的优势, 提出一种基于关系数据库存储 OWL 本体的存储模式,主要针对本体类的存储模式的设计思想和关键接口实现技术进行叙述. 案例研究表明,所设计的类可以有效的实现关系数据库和 OWL 本体类之间的数据转换.

关键词: 关系数据库; Web 本体语言; 本体

Storage and Extraction of OWL Ontology Based on Relational Database

LIU Qiao, WANG Xin-Fang, LEI Ming

(Faculty of Automation and Information Engineering, Xi'an University of Technology, Xi'an 710048, China)

Abstract: OWL is a language that depicts semantics of ontology. It does not only provide the strong semantic expression, but presents content that computer can be understandable. In the area of storage and managing mega data , the relevant techniques of relational database are natured. In this paper, the advantages of both are fully taken. It puts forward a storage model of relational database storing OWL ontology. This paper mainly gives design ideas and key interface technologies of the OWL Class storage model. Case studies show that classes that we design can effectively achieve data conversion relationship between the database and the OWL ontology Class.

Key words: relational database; ontology web language(OWL); ontology

目前我们正处在万维网发展的新时代, 即第二代网络—语义网的研究、开发阶段. 而网络本体语言 OWL 是实现其功能的核心语言工具. 网络本体语言 OWL 是 W3C 开发的一种网络本体语言[1], 用于对本体进行语义描述, 基于 OWL 构建的本体就称为 OWL 本体. 传统的知识表示面对的是人, 而基于 OWL 本体的知识表示面对的是机器或程序, 它为我们驾驭多种信息和信息服务提供了一种强大的、实用的方法. OWL 本体中的语义功能可以使更多信息变得可用, 可以避免信息泛滥带来的负面影响.

近年来, 随着 OWL 语言的出现和推广[2], 越来越多的人使用这种语言构建了大量的本体. 如何将这些本体进行有效的组织存储管理, 就成为了语义本体知识系统各类技术中的最为关键的基础性技术[3]. 而关系数据库是当前万维网上绝大部分数据的存储方式, 因此本体和关系数据库模式之间的相互转换是语义网研究中的一项重要内容, 本文所做的主要工作如图 1 圈中所示, 有两方面的内容需要完成:

1)实现将 OWL 本体存入到关系数据库中, 以方便用户对其信息的查询以及本体信息的管理.

2)实现本体从关系数据库中的提取, 并重构为本体对象, 以供用户的推理、查询以及管理.

下面的章节将详述具体类的功能实现.

1 类存储提取方案设计

如何将 OWL 本体类存入关系数据库, 我们提出了一种用于在关系数据库中存贮 OWL 本体类的方法, 内容涉及到数据库设计、接口函数设计等关键技术.

① 收稿时间:2014-10-11;收到修改稿时间:2014-11-14

下面将详细介绍类存储模式的设计.

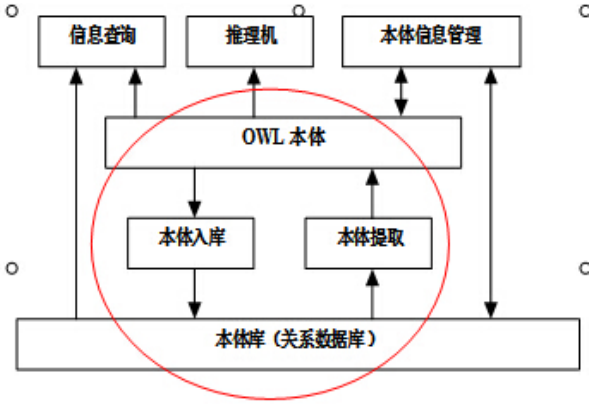

图 1　各模块之间的关系

1.1 本体类存储模式设计思路

A: 本体中无论类、属性(属性包括对象属性和数据类型属性)还是实例都是一种资源, OWL 语言使用 URI 来唯一标识资源. 因此建立类表(Class _Tab)去组织本体中的类资源和其 URI, 而其余所有表中的资源都用其 URI 来表示并在这些表之间建立起外键关联. 这样当本体中的资源发生变化时, 无需更改其他表, 提高了这种模式的可扩展性.

B: OWL 提供了 subClassOf, equivalentClass 等来描述类之间的关系, 所以建立类描述信息表(Class _D_ Tab)存放类的描述信息. 其中对于描述信息的存储我们采用两个属性列描述信息类型号(D_Id)和描述信息(ED)去组合的方式分别表示出不同的描述信息. 例如当 D_Id=1 时, ED 就等于等效类信息; 当 D_Id=2 时, ED 就等于父类信息; 以此类推.

C: 本体中类有一定的层次结构, 所以设计类树层次表(Class _Tree _Tab)专门存储其中子类父类、属性和子属性之间的结构关系.

类存储模式图如图 2 所示.

1.2 基于 OWL API 的本体操作

设计好本体的存储模式之后, 我们还需要考虑的一个问题就是如何对本体进行解析. 解析是本体存储的第一步, 这一步需要本体推理机的实现, OWL API 是一个 java API,借助 OWL API 可以方便地实现对 OWL 本体的操作. 下来我们将介绍要完成本体的存储和提取所要涉及到的关键技术. 分别将介绍 OWL API 中的接口和类、本体的装载、本体的信息的提取以及本体的重构.

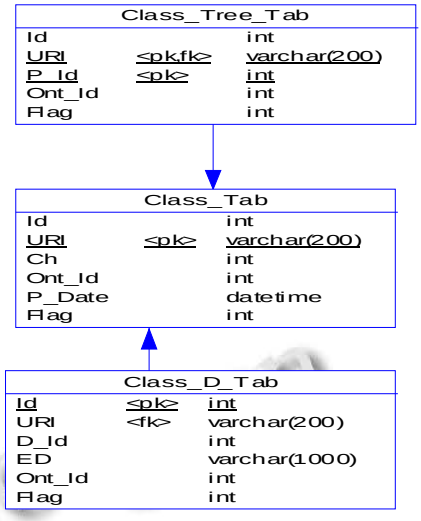

图 2　基于关系数据库的类存储模式图

A: OWL API 中的接口和类.

OWL API 提供了众多的 Java 接口, org.semanticweb.owlapi.model 包中提供了有用于操作本体的主要接口和类. 下面我们介绍其中最主要的接口或类, 如表 1 所示.

表 1　OWL API 中的接口和类

OWLOntologyManager 接口	该接口负责对本体的管理，比如本体的创建、装载和访问，或者说它负责管理本体和本体文档之间的映射。通过 OWLManager 类的 静态方法 createOWLOntologyManager 可以获得一个 OWLOntologyManager 接口对象。
OWLEntity 接口	OWL 中的实体的抽象表示。类、对象属性、数据属性和个体等都是该接口的子接口。实现该接口的类的实例均可由 IRI 唯一标识。
OWLOntology 接口	该接口是 OWL2 规范中本体的抽象表示。一个本体由一系列 OWLAxiom 和 OWLAnnotations 构成。一个 OWLOntology 对象可以有一个 IRI 对象来唯一标识它,也可以没有 IRI,如果有 IRI 的话，那么他也可以有一版本 IRI(Version IRI)唯一标识它的版本号。我们不能直接改变 OWLOntology 对象，必须通过 OWLOntologyManager 实现对本体的修改。
OWLDataFactory 接口	该接口负责实现实体(Entities)、类表达式(Class Expressions)和各类公理对象(Axioms)的创建。
OWLClass 接口	OWLClass 接口表示 OWL2[4]规范中的类(Class)，类是本体中最基本的实体。
IRI 类	IRI 类是 International Resource Identifiers 的缩写，本体中的每一个实体都需要一个唯一的 IRI 来唯一标识自己。IRI 类似于 URI，IRI 对象可以由一个字符串唯一的确定，通常这个字符串有一个前缀(比如: http://MyWeb)和一个局部名(比如学生)。前缀和局部名之间是字符“#”，比如(http://MyWeb#学生)。

B: 本体装载.

要将 OWL 文件中存放的本体信息转到关系数据库中, 因此, 装载本体和信息提取是一项基本操作. 根据上面我们讲述的 OWL API 中的接口和类, 我们可以编程实现本体的装载和信息提取.

C: 本体信息的提取.

装载本体文件后, 我们得到一个 OWLOntology 对象, 本体的类描述信息都包含在该对象中. 接下来我们叙述类以及类描述信息具体的提取流程.

1)调用本体对象的 getClassesInSignature 方法, 得到全部类的集合

2)针对类集合, 开始一个循环

3)从集合中取出一个类

a)提取类的 IRI 信息. 调用类对象的 getIRI()方法可得到类的 IRI 信息.

b)提取类的等效类信息. 调用类对象的 getEquivalentClasses 方法得到一个集合, 集合中的每一项都是一个类表达式, 与该类等效.

c)提取类的父类. 调用类对象的 getSuperClasses 方法得到一个集合, 集合中的每一项都是一个类表达式, 该类是它的子类.

d)提取类的成员信息. 调用类对象的 getIndividuals 方法得到一个集合, 集合中的每一项都是一个个体, 它是该类的实例.

e)提取类的目标键信息. 调用本体对象的 getHasKeyAxioms 方法得到一个集合, 集合中的每一项都是一个 OWLHasKeyAxiom 对象, 每个 OWLHasKeyAxiom 对象是由对象属性和数据属性构成的集合.

f)提取类的不相交类信息. 调用本体对象的 getDisjointClassesAxioms 方法得到一个集合, 集合中的每一项都是 OWLDisjointClassesAxiom 对象, 每个 OWLDisjointClassesAxiom 是类表达式的集合.

g)提取类的不相交并信息. 调用本体对象的 getDisjointUnionAxioms 方法得到一个集合, 集合的每一项都是一个 OWLDisjointUnionAxiom, 每个 OWLDisjointUnionAxiom 对象都是类表达式的集合.

4)类集合中是否有待处理的类, 如果有, 转到 3), 否则, 转到 5)

5)结束

对于对象属性描述信息、数据属性描述信息、个体信息及 SWRL 规则信息的提取理念上是相同的, 在此不再详述.

D: 本体的重构.

将本体信息存贮之后, 在后继的工作中我们需要从关系数据库中提取一个本体的相关信息, 重新构建本体, 以供用户后继使用. 具体本体重构步骤如下:

首先创建本体对象, 创建本体对象后, 通过 OWLDataFactory 接口提供的相关 getX 方法(这里 X 是返回的类型), 创建相关的 OWLAxiom 对象, 并通过 OWLOntologyManager 的 addAxiom 方法向本体中添加 OWLAxiom 对象,详细的过程将在第 3 节描述. 表 2 给出了常用的 OWLDataFactory 接口的 getX 方法.

表 2 OWLDataFactory 接口提供的常用 getX 方法

get 方法	返回的对象类型
getOWLClass	OWLClass
getOWLClassAssertionAxiom	OWLClassAssertionAxiom
getOWLEquivalentClassesAxiom	OWLEquivalentClassesAxiom
getOWLSubClassOfAxiom	OWLSubClassOfAxiom
getOWLHasKeyAxiom	OWLHasKeyAxiom
getOWLDisjointClassesAxiom	OWLDisjointClassesAxiom
getOWLDisjointUnionAxiom	OWLDisjointUnionAxiom

2 OWL本体存贮及提取的实现

OWL API 方便了我们对于 OWL 本体的操作, 但是实现本体的存储与提取其中还有一个关键的环节就是需要设计实现相关的 Java 类实现本体描述信息到描述字符串的转换, 将 OWL 本体的描述信息转换为可阅读的字符串形式进行存贮(描述字符串), 再设计实现相应的 Java 类, 实现描述字符串到本体描述信息的转换(Java 本体对象的重构). 下面我们将具体讲此技术的实现过程.

2.1 描述信息对象到描述字符串的转换

在使用 OWL API 将本体信息存入关系数据库时, 需要将用于描述类的相关 Java 对象转换为描述字符串, 这样才有利于存贮信息的阅读以及理解. 为此我们设计实现了类 OWLClassExpressionToDescription, 用于类表达式对象的转换. 下面将详细叙述该类的设计原理及实现.

OWL 2 提供了一系列类表达式, 包括类声明、布尔运算、对特性的全称和存在量词限定、特性值限定枚举等运算. OWLClassExpressionToDescription 类用

于类表达式对象到描述字符串的转换,它需要实现 OWLClassExpressionVisitor 接口, 接口中有 18 个 visit 成员函数需要实现, 这些重载的成员函数分别对不同的对象进行处理. 除 OWLObjectComplementOf、OWLObjectUnionOf 和 OWLObjectIntersectionOf 对象外, 其它 15 个对象都是基本对象, 可直接根据它们构建对应的描述字符串. OWLObjectComplementOf、OWLObjectUnionOf 和 OWLObjectIntersectionOf 三个对象是由基本对象通过“与”运算符、“或”运算符和“补”运算符构成的, 通过调用类表达式对象的 accept 方法再次调用对应的 visit 方法.

2.2 描述字符串到描述信息对象的转换

由于数据库中存放的都是各个 Java 对象的字符串描述, 因此要实现本体信息的重构, 关键是要解决从这些字符串描述获取对应 Java 对象. 对于类对象, 数据库中存贮的是它的 IRI 字符串, 且都可以通过调用 OWLDataFactory 对象的相应 get 方法来获取对应的 Java 对象:

(1) 调用 IRI 类的 create 方法创建 IRI 对象.

(2) 调用 OWLDataFactory 对象的 getOWLClass 获取 OWLClass 对象.

对于其它对象的获取, 我们设计实现两个类: MyChecker 和 MyParser.

(1) MyChecker 类

是供 MyParser 使用的辅助类, 虽说是辅助类, 但它的作用非常重要. 为了解析各类对象的字符串描述, 必须根据实际情况实现 OWL API 中的 OWLEntityChecker 接口. 实现 OWLEntityChecker 接口中的 getOWLClass(String name)成员函数, 这个成员函数根据 IRI 返回适当的 Java 对象.

(2) MyParser 类

设计实现一个 Java 类(MyParser)用于解析描述字符串, 得到对应的 Java 对象. 需要设计实现 4 个公用成员函数完成对类描述字符串的解析, 这 4 个公用成员函数的函数原型如下:

```
OWLClassExpression ParseOWLClassExpression (String desc);
OWLDisjointClassesAxiom ParseOWLDisjointClassesAxiom
(OWLClass clexp,OWLEntityChecker checker,String desc);
OWLDisjointUnionAxiom ParseOWLDisjointUnionAxiom
(OWLClass cl,OWLEntityChecker checker,String desc);
OWLLiteral ParseOWLLiteral(String desc);
```

其中, 返回值类型就是通过对描述字符串的解析而获取的 Java 对象.

3 测试

下面用一个实例验证从 Java 对象到描述字符串以及从描述字符串到 Java 对象转换之间的正确性. 我们使用 Protégé4.3[5]本体编辑工具编辑一个本体,本体 IRI 为: http://www.semanticweb.org/ontologies/Family, 将该本体保存在 Family.owl 文件中, 在该本体中, 从 OWL 语法出发, 我们尽可能设计一些可能出现的情况而无需考虑真实的物理含义, 表 4-1 中给出了本体中包含的类、对象属性、数据属性和个体. 所有实体的 IRI 均以 http://www.semanticweb.org/ontologies/Family 开始. 比如, http://www.semanticweb. org/ontologies /Family#FistClass, 表中实体的名称均为短名, 个体名称也一样. 图 3 为我们要测试的本体的类.

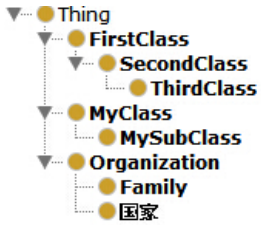

图 3　测试本体中的类

3.1 描述信息对象到描述字符串的转换测试

由于篇幅限制, 我们仅仅测试类的描述信息. 以 Protégé4.3 中显示的内容为参考, 用第三节设计的 4 个转换类得到相关对象的描述字符串, 与 Protégé4.3 中显示的内容进行对比. 我们以 SecondClass 类为例, 测试该类的父类描述信息的转换.

运行结果与 Protégé4.3 中显示的内容完全相同. 对于测试类的其他描述信息, 代码和上面的代码类似, 运行结果也与 Protégé4.3 中显示的内容完全相同.

3.2 描述字符串到描述信息对象的转换测试

还是以 Family.owl 文件为例, 我们用 3.2 节中设计的函数实现描述字符串到描述信息对象的转换, 实现信息的重构. 我们还是以 SecondClass 类为例, 测试该类的父类描述字符串的转换. 提取后的文件与源文件进行对比, 内容完全相同.

表 3 转换过程

Protégé4.3 中显示的该类的父描述信息	提取出的本体描述对象信息	转换后入库信息
SubClass Of (年龄 some integer[>= 40]) and (年龄 some integer[<= 50]) FirstClass Like some Self not (MyClass) 年龄 exactly 5 integer[>= 40 , <= 50] 年龄 max 3 integer[>= 40 , <= 50] 年龄 min 3 integer[>= 40 , <= 50] 年龄 only integer[>= 40 , <= 50] 年龄 some integer[>= 40 , <= 50] 年龄 value 40 身份证号 max 4 string 身份证号 min 2 string 身份证号 only string 身份证号 value "089x"	<http://www.semanticweb.org/ontologies/ ObjectIntersectionOf(DataSomeValuesFrom ObjectComplementOf(<http://www.semantic ObjectHasSelf(<http://www.semanticweb.o DataSomeValuesFrom(<http://www.semantic DataAllValuesFrom(<http://www.semanticw DataAllValuesFrom(<http://www.semanticw DataHasValue(<http://www.semanticweb.or DataHasValue(<http://www.semanticweb.or DataMinCardinality(3 <http://www.semant DataMinCardinality(2 <http://www.semant DataExactCardinality(5 <http://www.sema DataMaxCardinality(3 <http://www.semant DataMaxCardinality(4 <http://www.semant <http://www.semanticweb.org/ontologies/ <http://www.semanticweb.org/ontologies/ <http://www.semanticweb.org/ontologies/ DataExactCardinality(1 <http://www.sema	ED FirstClass (年龄 some integer[>= 40]) and (年龄 some integer[<= 50]) not MyClass Like some Self 年龄 some integer[>= 40,<= 50] 年龄 only integer[>= 40,<= 50] 身份证号 only string 年龄 value 40 身份证号 value "089x" 年龄 min 3 integer[>= 40,<= 50] 身份证号 min 2 string 年龄 exactly 5 integer[>= 40,<= 50] 年龄 max 3 integer[>= 40,<= 50] 身份证号 max 4 string

表 4 转换过程

数据库中存储的描述字符串信息	重构后的描述信息对象(截取其中部分代码)	源文件的描述信息对象
ED FirstClass (年龄 some integer[>= 40]) and (年龄 some integer[<= 50]) not MyClass Like some Self 年龄 some integer[>= 40,<= 50] 年龄 only integer[>= 40,<= 50] 身份证号 only string 年龄 value 40 身份证号 value "089x" 年龄 min 3 integer[>= 40,<= 50] 身份证号 min 2 string 年龄 exactly 5 integer[>= 40,<= 50] 年龄 max 3 integer[>= 40,<= 50] 身份证号 max 4 string	<rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/Family#FirstClass"/> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="http://www.semanticweb.org/ontologies/Family#身份证号码 "/> <owl:onDataRange rdf:resource="string"/> </owl:Restriction> </rdfs:subClassOf>	<rdfs:subClassOf rdf:resource="&Family;FirstClass"/> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="&Family;身身份证号码"/> <owl:allValuesFrom rdf:resource="&xsd;string"/ > </owl:Restriction> </rdfs:subClassOf>

4 结论

本文介绍了基于关系数据库的本体存储模式的设计与实现技术. 本次设计的类是基于 MyEclipse10 Java 平台, 借助 OWL API 对本体信息进行推理解析. 通过实验测试, 我们得出我们设计的类可以有效的实现本体描述信息与容易理解字符串之间的相互转换. 本文的讨论只限于本文所设计的存储模式, 在后续的工作中, 我们还应该针对存储模式的查询效率以及可扩展性上进行进一步的测试完善.

参考文献

1 Michel K. OWL Web Ontology Language Guide Recommendation. http: //www.w3.org/TR/2004/REC-owl-features-20040210/. [2004-02-10].

2 朱勤斯,虞慧群.一种基于语义网技术和本体的数据集成方法.华东理工大学学报,2009,34(1):199–215.

3 史一民,李冠宇,刘宁.语义网服务中本体服务综述.计算机工程与设计,2008,29(23):5976–5980.

4 http://wenku.baidu.com/view/631a6deb998fcc22bcd10d3c. html.

5 http://protege.stanford.edu/.