**http://blog.csdn.net/zhang6560329/article/details/27095197**

**Jena的简单理解和一个例子**

转载 2014年05月26日 17:10:48

* 标签：
* [web service](http://so.csdn.net/so/search/s.do?q=web%20service&t=blog) /
* [sparql](http://so.csdn.net/so/search/s.do?q=sparql&t=blog) /
* [rdf](http://so.csdn.net/so/search/s.do?q=rdf&t=blog) /
* [jena](http://so.csdn.net/so/search/s.do?q=jena&t=blog)
* 3497

本文简单介绍Jena（Jena 2.4），使用Protégé 3.1（不是最新版本）创建一个简单的生物（Creature）本体，然后参照Jena文档中的一个例子对本体进行简单的处理，输出本体中的Class、Property等信息。

本文内容安排如下：

Ø         介绍Jena

Ø         运行Jena

Ø         Jena Ontology API

Ø         例子

Ø         参考资料

一、介绍Jena

Jena由 HP Labs（http://www.hpl.hp.com）开发的Java开发工具包, 用于Semantic Web(语义网)中的应用程序开发；Jana是开源的，在下载的文档中有Jena的完整代码。Jena框架主要包括:

a)       以RDF/XML、三元组形式读写RDF

资源描述框架是(RDF)是描述资源的一项标准(在技术上是W3C的推荐标准)，Jena文档中有一部分呢详细介绍了RDF和Jena RDF API，其内容包括对Jena RDF包的介绍、RDF模型的创建、读写、查询等操作，以及RDF容器等的讨论。

b)       RDFS，OWL，DAML+OIL等本体的操作

Jena框架包含一个本体子系统（Ontology Subsystem），它提供的API允许处理基于RDF的本体数据，也就是说，它支持OWL，DAML+OIL和RDFS。本体API与推理子系统结合可以从特定本体中提取信息，Jena 2还提供文档管理器（OntDocumentManager）以支持对导入本体的文档管理。

c)       利用数据库保存数据

    Jena 2允许将数据存储到硬盘中，或者是OWL文件，或者是关系数据库中。本文处理的本体就是OWL文件读入的。

d)       查询模型

Jena 2提供了ARQ查询引擎，它实现SPARQL查询语言和RDQL，从而支持对模型的查询。另外，查询引擎与关系数据库相关联，这使得查询存储在关系数据库中的本体时能够达到更高的效率。

e)       基于规则的推理

Jena 2支持基于规则的简单推理，其推理机制支持将推理器(inference reasoners)导入Jena，创建模型时将推理器与模型关联以实现推理。

Protégé是一个开源的本体编辑器（目前的版本是Protégé 3.2），用户可以在GUI环境下创建本体或者知识库。有一种说法是：Jena对应用程序就像Protégé对我们——我们使用Protégé操作本体，应用程序则是使用Jena来做同样的工作。当然这些应用程序还是得由我们来编写。

二、运行Jena

可以在Jena的主页（http://jena.sourceforge.net/downloads.html）下载Jena的最新版本，目前是Jena2.4版本。Jena是Java API，所以需要Java运行环境。本文使用的是jdk1.5.0\_04和Eclipse3.2。

将下载的Jena-2.4.zip解压到任意路径，解压之后生成Jena2.4文件夹，将Jena2.4\lib下的jar文件全部加入CLASSPATH，这样就可以在任意的Java编辑器中调用Jena API了。在解压目录下有一个test.bat文件，用于配置的测试。在控制台运行此程序，如果你的配置正确，测试将顺利完成。

如果使用Eclipse，则可以通过修改工程的Java创建路径的方法导入Jena jar文件。在Eclipse下创建Java工程，右健单击工程名字，选择“属性/Properties”，在打开的对话框中选择“Java创建路径/Java Build Path”，在右边标签中选择“库/Libraries”,之后选择“添加外部文件/Add Extenal JARs”，找到Jena2.4\lib目录下的所有jar文件并将其添加到工程。这样就可以运行Jean文档中的例子了。

三、Jena Ontology API

Jena2.4的Ontology API包含在ontology包（com.hp.hpl.jena.ontology）中，可以在目录\\Jena-2.4\src\com\hp\hpl\jena\ontology下查看所有程序的代码，Jena本体部分的说明网页是\\Jena-2.4\doc\ontology\index.html,本部分内容以及程序的编写主要参考这两个文档。

在语义网上有很多表示本体信息的本体语言，其中表达能力最强的是OWL，OWL按复杂程度分为 OWL Full、OWL DL和OWL Lite三个版本。其他的本体语言还有RDFS、DAML+OIL。Jena Ontology API为语义网应用程序开发者提供了一组独立于具体语言的一致编程接口。

Jena提供的接口本质上都是Java程序，也就是.java文件经过javac之后生成的.class文件。显然，class文件并不能提示本体创建使用的语言。为了区别于其他的表示方法，每种本体语言都有一个自己的框架（profile），它列出了这种语言使用的类（概念）和属性的构建方式和URI。因此，在DAML框架里，对象属性（）的URI是daml:ObjectProperty，而在OWL框架里却是owl:ObjectProperty。RDFS并没有定义对象属性，所以在RDFS框架里，对象属性的URI是null。

在Jena中，这种框架通过参数的设置在创建时与本体模型（Ontology Model）绑定在一起。本体模型继承自Jena中的Model类。Model允许访问RDF数据集合中的陈述（Statements），OntModel对此进行了扩展，以便支持本体中的各种数据对象：类（classes）、属性（properties）、实例（个体individuals）。

本部分简单介绍要用到的几个java类或者接口。

1．本体模型OntModel

本体模型（OntModel）是对Jena RDF模型的扩展（继承自RDF模型），提供了处理本体数据的功能。使用Jena处理本体首先就是要建立一个本体模型，之后就能够通过本体模型中所定义的方法操作模型，比如导入子模型（）、获取模型中本体的信息、操作本体属性以及将本体的表示输出到磁盘文件等等。Jena通过model包中的ModelFactory创建本体模型，ModelFactory是Jena提供用来创建各种模型的类，在类中定义了具体实现模型的成员数据以及创建模型的二十多种方法。一个最简单的创建本体模型的语句如下：

OntModel ontModel = ModelFactory.createOntologyModel();

该语句不含参数，应用默认设置创建一个本体模型ontModel，也就是说：它使用OWL语言、基于内存，支持RDFS推理。可以通过创建时应用模型类别（OntModelSpec）参数创建不同的模型，以实现不同语言不同类型不同推理层次的本体操作。例如，下面的语句创建了一个使用DAML语言内存本体模型。直观地讲，内存模型就是只在程序运行时存在的模型，它没有将数据写回磁盘文件或者数据库表。

OntModel ontModel = ModelFactory.createOntologyModel( OntModelSpec.DAML\_MEM );

更多类型设置可以参照OntModelSpec类中的数据成员的说明。

我们所使用的本体是从OWL文件获得的，也就是说，是从磁盘读取的。读取的方法是调用Jena OntoModel提供的Read方法。例如

ontModel.read("file:D:/temp/Creatrue/Creature.owl");

就是读取位于D盘相应目录下的Creature.owl文件以建立本体模型。Read方法也有很多重载，上面调用的方法以文件的绝对路径作为参数。其他的方法声明如下

read( String url );

read( Reader reader, String base );

read( InputStream reader, String base );

read( String url, String lang );

read( Reader reader, String base, String Lang );

read( InputStream reader, String base, String Lang );

2．文档管理器Document manager

本体文档管理器（OntDocumentManager）是用来帮助管理本体文档的类，它包含了导入本体文档创建本体模型、帮助缓存下载网络上的本体等功能。每个本体模型都有一个相关联的文档管理器。在创建本体模型时，可以创建独立的文档管理器并作为参数传递给模型工厂（ModelFactory）。文档管理器有非常多的配置选项，基本可以满足应用的需求。首先，每个文档管理器的参数都可以通过Java代码来设置（注：OntDocumentManager有五种重载的构造函数）。另外，文档管理器也可以在创建的时候从一个RDF格式的策略文件读取相应设定值。

下面的例子创建一个文档管理器并将它与以创建的本体模型关联。

OntModel m = ModelFactory.createOntologyModel();

OntDocumentManager dm = m.getDocumentManager();

    3．接口OntClass

这个接口中定义了本体种与概念（也就是类Class）相关的操作，通过OntModel中的

listClasses()

便可以返回模型中的所有概念组成的迭代器（Iterator），然后调用OntClass的各种方法具体进行具体操作。OntoClass对概念之间的各种关系都有相应的定义方法，典型的有添加子类、添加约束、创建互斥概念、迭代返回某种类型的概念以及相关的逻辑判断等等。

第四部分的例子主要是应用这个类的方法。

4．基本本体类型OntResource

所有本体API中用于表示本体的类继承自OntResource，这样就可以在OntResource中放置所有类公用的功能，并可以为一般的方法设置通用的返回值。Java接口OntResource扩展了Jena的RDF资源接口，所以任何可以接受资源或者RDFNode的方法都可以接受OntResource，并且也就可以接受任何其他本体值。虽然这个类涵盖了涉及本体的所有类，在例子中并没有使用它。从Jena Java Doc可以获得它的详细信息。

四、例子

本文实现了一个简单的例子。通过Protégé 3.1创建一个Creature本体，并将其存储到OWL文件Creature.owl，然后使用和Jena通过读取该文件创建本体模型，实践Jena中的一些Ontology API。本文不涉及如何使用Protégé 3.1创建本体，您可以到Protégé的主页或者是http://www.chengtao.name/modules/wordpress/寻找详细的资料，后者虽然只是一个博客，但确实提供了很多我们需要的资源，比如一个完整的Protégé使用教程。本文例子中对Jena的使用主要参照Jena文档中\\Jena-2.4\src-examples\jena\examples\ontology下面的describeClass。它读取food.owl和wine.owl两个文件建立本体模型，并显示模型中概念以及相关关系的详细信息。DescribeClass.java主要负责查询详细信息并显示出来，Main.java只是创建本体模型并调用DescribeClass.java的功能。

本体类层次图

例子的主要代码如下列出。

// 创建使用OWL语言的内存模型

OntModel ontModel = ModelFactory.createOntologyModel(OntModelSpec.OWL\_MEM);

ontModel.read("file:./Creature.owl"); // 读取当前路径下的文件，加载模型

// 定义一个类作为模型中Animal类的等等价类，并添加注释

OntClass cls = ontModel.createClass(":DongwuClass");

cls.addComment("the EquivalentClass of Animal...", "EN");

// 通过完整的URI取得模型中的Animal类

OntClass oc = ontModel.

getOntClass("http://www.owl-ontologies.com/marine.owl#Animal");

oc.addEquivalentClass(cls); // 将先前定义的类添加为Animal的等价类

// 迭代显示模型中的类，在迭代过程中完成各种操作

for (Iterator i = ontModel.listClasses(); i.hasNext();) {

OntClass c = (OntClass) i.next(); // 返回类型强制转换

if (!c.isAnon()) { // 如果不是匿名类，则打印类的名字

System.out.print("Class");

// 获取类的URI并输出，在输出时对URI做了简化（将命名空间前缀省略）

System.out.println(c.getModel().getGraph().

getPrefixMapping().shortForm(c.getURI()));

// 处理Animal类

if (c.getLocalName().equals("Animal")) { // 如果当前类是Animal

       System.out.println("  URI@" + c.getURI()); // 输出它的完整URI                 // 取得它的的等价类并打印

       System.out.print("  Animal's EquivalentClass is "+

c.getEquivalentClass());

       // 输出等价类的注释

System.out.println(" [comments:" +

c.getEquivalentClass().getComment("EN")+"]");

}// 处理Animal结束

// 迭代显示当前类的直接父类

for (Iterator it = c.listSuperClasses(); it.hasNext();)

{

       OntClass sp = (OntClass) it.next();

       String str = c.getModel().getGraph()

                            .getPrefixMapping().shortForm(c.getURI()) // 获取URI

                            + "'s superClass is " ;

       String strSP = sp.getURI();

       try{ // 另一种简化处理URI的方法

              str = str + ":" + strSP.substring(strSP.indexOf('#')+1);

              System.out.println("  Class" +str);

       }catch( Exception e ){}

} // super class ends

// 迭代显示当前类的直接子类

for (Iterator it = c.listSubClasses(); it.hasNext();){

System.out.print("  Class");

OntClass sb = (OntClass) it.next();

System.out.println(c.getModel().getGraph().getPrefixMapping()

                                          .shortForm(c.getURI())+ "'s suberClass is "

                                          + sb.getModel().getGraph().getPrefixMapping()

                                          .shortForm(sb.getURI()));

}// suber class ends

// 迭代显示与当前类相关的所有属性

for(Iterator ipp = c.listDeclaredProperties(); ipp.hasNext();){

       OntProperty p = (OntProperty)ipp.next();

       System.out.println("  associated property: " + p.getLocalName());

       }// property ends

}// anonymity ends

else // 是匿名类

{}

}// for ends

部分输出结果如下，说明部分不在输出中。

Class:Animal

  URI@http://www.owl-ontologies.com/marine.owl#Animal  // 完整的URI

// 等价类及其注释

  Animal's EquivalentClass is :DongwuClass [comments:the EquivalentClass of Animal...]  Class:Animal's superClass is :Creature // 直接父类

  Class:Animal's suberClass is :MixAnimal // 直接子类

  Class:Animal's suberClass is :GrassAnimal

  Class:Animal's suberClass is :MeatAnimal

  associated property: eat // 关联属性

  associated property: beEated

  associated property: mainEat

后注：该例子没有涉及Jena的推理功能，没有实现属性的过滤（只返回以当前类为主体的属性），不能输出间接父类（父类的父类）或间接子类，等等。因此，有待对Jena进一步熟悉之后作更多的改进。

五、参考资料

[1]          \\Jena-2.4\doc\ontology Jena 2 Ontology API文档

[2]          \\Jena-2.4\doc\javadoc Jena 2 Java文档

[3]          \\Jena-2.4\src-examples\jena\examples\ontology\describeClass

[4]          http://www.chengtao.name/modules/wordpress 维基博客

[5]          iMarine [http://iMarine.blog.163.com](http://imarine.blog.163.com/)