Jena 简介

一般来说，我们在 Protege 这样的编辑器里构建了本体，就会想在应用程序里使用它，这就需要一些开发接口。用程序操作本体是很必要的，因为在很多情况下，我们要自动生成本体，靠人手通过 Protege 创建所有本体是不现实的。Jena 是 HP 公司开发的这样一套 API，似乎 HP 公司在本体这方面走得很靠前，其他大公司还在观望吗？

可以这样说，Jena 对应用程序就像 Protege 对我们，我们使用 Protege 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。其实 Protege 本身也是在 Jena 的基础上开发的，你看如果 Protege 的 console 里报异常的话，多半会和 Jena 有关。最近出了一个 Protege OWL API，相当于对 Jena 的包装，据说使用起来更方便，这个 API 就是 Protege 的 OWL Plugin 所使用的，相信作者经过 OWL Plugin 的开发以后，说这些话是有一定依据的。

题目是说用 Jena 处理 OWL，其实 Jena 当然不只能处理 OWL，就像 Protege 除了能处理 OWL 外还能处理 RDF(S)一样。Jena 最基本的使用是处理 RDF(S)，但毕竟 OWL 已经成为 W3C 的推荐标准，所以对它的支持也是大势所趋。

好了，现在来点实际的，怎样用 Jena 读我们用 Protege 创建的 OWL 本体呢，假设你有一个 OWL 本体文件（.owl），里面定义了动物类（http://www.zoo.com/ont/Animal，注意这并不是一个实际存在的 URL，不要试图去访问它），并且它有一些实例，现在看如下代码：

```
OntModel m = ModelFactory.createOntologyModel();
File myFile = ...;
m.read(new FileInputStream(myFile), "");
ResIterator iter = m.listSubjectsWithProperty(RDF.type, m.getResource("http://
www.zoo.com/ont/Animal"));
while (iter.hasNext()) {
    Resource animal = (Resource) iter.next();
    System.out.println(animal.getLocalName());
}
```

和操作 RDF(S)不同，com.hp.hpl.jena.ontology.OntModel 是专门处理本体（Ontology）的，它是 com.hp.hpl.jena.rdf.model.Model 的子接口，具有 Model 的全部功能，同时还有一些 Model 没有的功能，例如 listClasses()、listObjectProperties()，因为只有在本体里才有“类”和“属性”的概念。

上面的代码很简单，从 ModelFactory 创建一个 OntModel，从指定文件把模型读到内存里。再下面的代码是一个例子，作用是取出模型中所有 Animal 的实例（Individual，也叫个体），并打印它们的名称。要从 OntModel 里取实例，也可以用 listIndividuals()方法，只不过你得在得到的实例中判断它们是不是 Animal 的实例，我觉得不如用上面这种简易查询的方式来得方便。

Jena 里扩展了很多 Iterator，比如 ResIterator、StmtIterator 和 NodeIterator 等等，刚开始用会觉得很别扭，好象还不如都用 java 标准的 Iterator，不知道 Jena 的设计者是怎么考虑的。要熟练掌握还是得对整个 Jena 的 API 有全局掌握才好。

在循环里，我们得到的每个元素都是一个 Resource，因为本体里的任何东西都是资源，不论你想得到 Subject、Property 还是 Object，在 Jena 里实际得到的都是资源（Resource），在 Jena 里，Property 是 Resource 的子接口，而 Jena 并没有 Subject 或 Object 接口。（注：在 OWL 本体中，Subject->Property->Object 组成一个三元组，例如：张小刚->父亲->张大刚；或者：绵羊多利->rdf:type->动物，rdf:type 是一个特殊的属性，表示前者是后者的实例）

暂时先写到这，关于在本体中引入其他本体和使用推理，下次继续。

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。

G:\jade\lib\jade.jar;G:\jade\lib\iiop.jar;G:\jade\lib\commons-codec\commons-codec-1.3.jar;G:\jade\lib\javaTools.jar;G:\jade\lib\http.jar;G:\jena\lib\antlr-2.7.5.jar;G:\jena\lib\arq.jar;G:\jena\lib\arq-extra.jar;G:\jena\lib\commons-logging-1.1.jar;G:\jena\lib\concurrent.jar;G:\jena\lib\icu4j_3_4.jar;G:\jena\lib\iri.jar;G:\jena\lib\jena.jar;G:\jena\lib\jenatest.jar;G:\jena\lib\json.jar;G:\jena\lib\junit.jar;G:\jena\lib\alog4j-1.2.12.jar;G:\jena\lib\lucene-core-2.0.0.jar;G:\jena\lib\stax-api-1.0.jar;G:\jena\lib\wstx-asl-3.0.0.jar;G:\jena\lib\xercesImpl.jar;G:\jena\lib\xml-apis.jar

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();

(续)——见“对 Jena 的简单理解和一个例子_2”

RDF 越来越被认为是表示和处理半结构化数据的一种极好选择。本文中，Web 开发人员 Philip McCarthy 向您展示了如何使用 Jena Semantic Web Toolkit，以便在 Java 应用程序中使用 RDF 数据模型。

“资源描述框架（Resource Description Framework，RDF）”最近成为 W3C 推荐标准，与 XML 和 SOAP 等 Web 标准并排。RDF 可以应用于处理特殊输入数据（如 CRM）的领域，已经广泛用于社会网络和自助出版软件（如 LiveJournal 和 TypePad）。

Java 程序员将越来越多地得益于具有使用 RDF 模型的技能。在本文中，我将带您体验惠普实验室的开放源代码 Jena Semantic Web Framework（请参阅 参考资料）的一些功能。您将了解如何创建和填充 RDF 模型，如何将它们持久存储到数据库中，以及如何使用 RDQL 查询语言以程序方式查询这些模型。最后，我将说明如何使用 Jena 的推理能力从本体推断模型知识。

本文假设您已经就图形、三元组和模式等概念方面对 RDF 比较熟悉，并对 Java 编程有基本的了解。

创建简单的 RDF 模型

我们从基本操作开始：从头创建模型并向其添加 RDF 语句。本节，我将说明如何创建描述一组虚构家庭成员之间关系的模型，如图 1 中所示：

图 1. 虚拟家庭树

将使用来自“关系”词汇表（请参阅 参考资料）的属性 siblingOf 、 spouseOf 、 parentOf 和 childOf 来描述不同的关系类型。为简单起见，家庭成员用来自虚构名称空间的 URI（ http://family/ ）进行标识。词汇表 URI 通常以 Jena 代码形式使用，所以将它们声明为 Java 常量会非常有用，减少了错误输入。

Schemagen

当您通过 Jena 的 API 来使用模型时，为模型词汇表中的每个属性定义常量非常有用。如果有词汇表的 RDF、DAML 或 OWL 表示，Jena 的 Schemagen 工具可以自动生成这些常量，使您的工作更加容易。

Schemagen 在命令行中运行，使用的参数包括模式或本体文件的位置、要输出的类的名称和 Java 包。然后可以导出生成的 Java 类，其 Property 常量用于访问模型。

还可以使用 Ant 将 Schemagen 作为构建处理的一部分来运行，保持 Java 常量类与正在变化的词

Jena 的 ModelFactory 类是创建不同类型模型的首选方式。在这种情况下，您想要空的、内存模型，所以要调用的方法是 ModelFactory.createDefaultModel() 。这种方法返回 Model 实例，您将使用它创建表示家庭中每个成员的 Resource 。创建了资源后，可以编写关于这些资源的语句并添加到模型中。

汇表保持同步。

在 Jena 中，语句的主题永远是 Resource ，谓词由 Property 表示，对象是另一个 Resource 或常量值。常量在 Jena 中通过 Literal 类型表示。所有这些类型共享公共接口 RDFNode 。将需要四个不同的 Property 实例表示家庭树中的关系。这些实例使用 Model.createProperty() 创建。

将语句添加到模型中的最简单方法是通过调用 Resource.addProperty() 。此方法以 Resource 作为主题在模型中创建语句。该方法使用两个参数，表示语句谓词的 Property 和语句的对象。 addProperty() 方法被过载：一个过载使用 RDFNode 作为对象，所以可以使用 Resource 或 Literal 。还有有益过载，它们使用由 Java 原语或 String 表示的常量。在示例中，语句的对象是表示其他家庭成员的 Resource 。

通过使用三元组的主题、谓词和对象调用 Model.createStatement() ，还可以直接在模型上创建语句。注意以此种方式创建 Statement 不将其添加到模型中。如果想将其添加到模型中，请使用创建的 Statement 调用 Model.add() ，如清单 1 所示：

清单 1. 创建模型来表示虚构的家庭

```
// URI declarations
                        String familyUri = "http://family/";
                        String relationshipUri = "http://purl.org/voc
ab/relationship/";
                        // Create an empty Model
                        Model model = ModelFactory.createDefaultModel
();
                        // Create a Resource for each family member,
identified by their
                        URI Resource adam = model.createResource(fami
lyUri+"adam");
                        Resource beth = model.createResource(familyUr
```

```
i+"beth");
                        Resource chuck = model.createResource(familyU
ri+"chuck");
                        Resource dotty = model.createResource(familyU
ri+"dotty");
                        // and so on for other family members
                        // Create properties for the different types
of relationship to represent
                        Property childOf = model.createProperty(relat
ionshipUri,"childOf");
                        Property parentOf = model.createProperty(rela
tionshipUri,"parentOf");
                        Property siblingOf = model.createProperty(rel
ationshipUri,"siblingOf");
                        Property spouseOf = model.createProperty(rela
tionshipUri,"spouseOf");
                        // Add properties to adam describing relation
ships to other family members
                        adam.addProperty(siblingOf,beth);
                        adam.addProperty(spouseOf,dotty);
                        adam.addProperty(parentOf,edward);
                        // Can also create statements directly .. .
                        Statement statement = model.createStatement(a
dam,parentOf,fran);
                        // but remember to add the created statement
to the model
                        model.add(statement);
```

整个代码示例 FamilyModel.java 还说明了语句批量如何一次添加到模型中，或者作为一个数组或者作为 java.util.List 。

构建了家庭模型后，我们看一下如何使用 Jena 的查询 API 从模型中提取信息。

查询 RDF 模型

程序化地查询 Jena 模型主要通过 list() 方法在 Model 和 Resource 接口中执行。可以使用这些方法获得满足特定条件的主题、对象和 Statement 。它们还返回 java.util.Iterator 的特殊化，其具有返回特定对象类型的其他方法。

我们返回 清单 1 的家庭模型，看一下可以查询它的不同方法，如清单 2 所示：

清单 2. 查询家庭模型

```
// List everyone in the model who has a child:
                        ResIterator parents = model.listSubjectsWithP
roperty(parentOf);
                        // Because subjects of statements are Resourc
es, the method returned a ResIterator
                        while (parents.hasNext()) {
                        // ResIterator has a typed nextResource() met
hod
                        Resource person = parents.nextResource();
                        // Print the URI of the resource
                        System.out.println(person.getURI()); }
                        // Can also find all the parents by getting t
he objects of all "childOf" statements
                        // Objects of statements could be Resources o
r literals, so the Iterator returned
                        // contains RDFNodes
```

```
                        NodeIterator moreParents = model.listObjectsO
fProperty(childOf);
                        // To find all the siblings of a specific per
son, the model itself can be queried
                        NodeIterator siblings = model.listObjectsOfPr
operty(edward, siblingOf);
                        // But it's more elegant to ask the Resource
directly
                        // This method yields an iterator over Statem
ents
                        StmtIterator moreSiblings = edward.listProper
ties(siblingOf);
```

最通用的查询方法是 Model.listStatements(Resource s, Property p, RDFNode o)，下面说明的便利方法都是以其为基础。所有这些参数都可以保留为 null，在这种情况下，它们作为通配符，与任何数据都匹配。清单 3 中显示了 Model.listStatements() 的一些使用示例：

清单 3. 使用选择器查询模型

```
// Find the exact statement "adam is a spouse of dotty"
                        model.listStatements(adam,spouseOf,dotty);
                        // Find all statements with adam as the subje
ct and dotty as the object
                        model.listStatements(adam,null,dotty);
                        // Find any statements made about adam
                        model.listStatements(adam,null,null);
                        // Find any statement with the siblingOf prop
erty
                        model.listStatements(null,siblingOf,null);
```

导入和持久化模型

不是所有的应用程序都从空模型开始。更常见的是，在开始时从现有数据填充模型。在这种情况下，使用内存模型的缺点是每次启动应用程序时都要从头重新填充模型。另外，每次关闭应用程序时，对内存模型进行的更改都将丢失。

一种解决方案是使用 Model.write() 序列化模型到文件系统，然后在开始时使用 Model.read() 将其取消序列化。不过，Jena 还提供了持久化模型，它们会被持续而透明地持久存储到后备存储器。Jena 可以在文件系统中或在关系数据库中持久化它的模型。当前支持的数据库引擎是 PostgreSQL、Oracle 和 MySQL。

WordNet

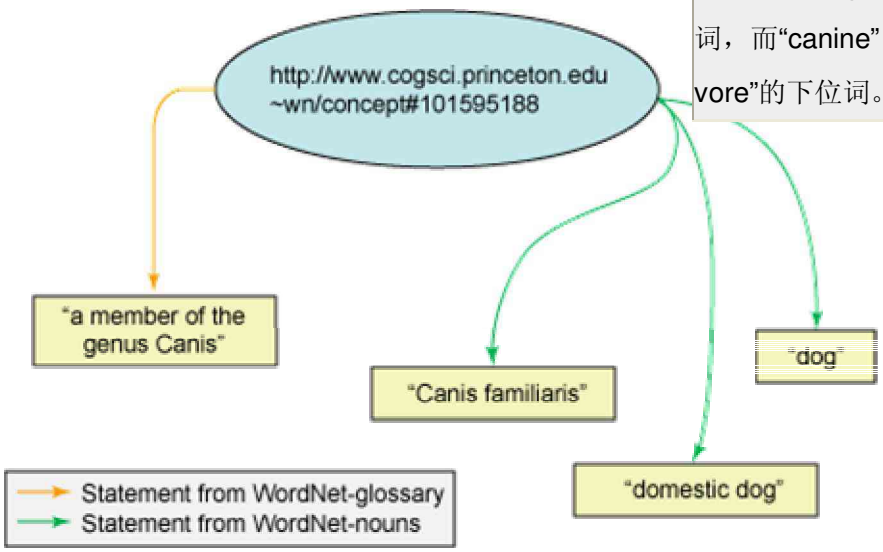
WordNet 是“英文语言的词汇数据库”。我使用的是 Sergey Melnik 和 Stefan Decker 的 RDF 表示。它具有四个单独的模型，本文示例中将使用其中三个模型。

WordNet-nouns 模型包含 WordNet 表示的所有“词汇概念”和用于表示每个概念的“单词形式”。例如，它包含由单词形式“domestic dog”、“dog”和“Canis familiaris”表示的词汇概念。

第二个模型是 WordNet-glossary。它提供模型中每个词汇概念的

为了说明如何导入和持久化模型，我将 WordNet 1.6 数据库的 RDF 表示导入到 MySQL 中。因为我使用的 WordNet 表示采用多个单独 RDF 文档的形式，将这些文档导入到一个 Jena 模型中会合并它们的语句。 图 2 说明了 Nouns 和 Glossary 模型合并后 WordNet 模型的片段的结构：

图 2. 合并的 WordNet nouns 和 glossary 模型的结构



简短定义。"dog"的词汇概念具有词汇条目"a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times."

WordNet-hyponyms 是第三个模型。它定义模型中概念的层次结构。概念"dog"是概念"canine"下位词，而"canine" 本身是概念"carnivore"的下位词。

创建数据库后台模型的第一步是说明 MySQL 驱动类，并创建 DBConnection 实例。 DBConnection 构造函数使用用户的 ID 和密码登录到数据库。它还使用包含 Jena 使用的 MySQL 数据库名称的数据库 URL 参数，格式为 "jdbc:mysql://localhost/dbname" 。Jena 可以在一个数据库内创建多个模型。 DBConnection 的最后一个参数是数据库类型，对于 MySQL，该参数为 "MySQL" 。

然后 DBConnection 实例可以与 Jena 的 ModelFactory 一起使用来创建数据库后台模型。

创建了模型后，可以从文件系统中读入 WordNet RDF 文档。不同的 Model.read() 方法可以从 Reader 、 InputStream 或 URL 填充模型。可以通过 Notation3 、N-Triples 或默认情况下通过 RDF/XML 语法解析模型。WordNet 作为 RDF/XML 进行序列化，所以不需要指定语法。读取模型时，可以提供基准 URI。基准 URI 用于将模型中的任何相对

URI 转换成绝对 URI。因为 WordNet 文档不包含任何相对 URI，所以此参数可以指定为 null 。

清单 4 显示了将 WordNet RDF/XML 文件导入到 MySQL 持久化模型的完整过程：

清单 4. 导入和持久化 WordNet 模型

```
// Instantiate the MySQL driver
                        Class.forName("com.mysql.jdbc.Driver");
                        // Create a database connection object
                        DBConnection connection = new DBConnection(DB
_URL, DB_USER, DB_PASSWORD, DB_TYPE);
                        // Get a ModelMaker for database-backed model
s
                        ModelMaker maker = ModelFactory.createModelRD
BMaker(connection);
                        // Create a new model named "wordnet."Setting
 the second parameter to "true" causes an
                        // AlreadyExistsException to be thrown if the
 db already has a model with this name
                        Model wordnetModel = maker.createModel("wordn
et",true);
                        // Start a database transaction.Without one,
each statement will be auto-committed
                        // as it is added, which slows down the model
 import significantly.
                        model.begin();
                        // For each wordnet model .. .
                        InputStream in = this.getClass().getClassLoad
er().getResourceAsStream(filename);
                        model.read(in,null);
                        // Commit the database transaction model.
                        commit();
```

由于已经填充了 wordnet 模型，以后可以通过调用 ModelMaker.openModel("wordnet",true)；来访问该模型。

仅使用 Jena 的 API 查询像 WordNet 这样巨大的模型将有一定的限制性，因为要执行的每类查询都将需要专门编写多行的代码。幸运的是，Jena 以 RDQL 形式提供了一种表达通用查询的机制。

RDF 数据查询语言（RDQL）

RDQL 是 RDF 的查询语言。虽然 RDQL 还不是正是的标准，但已由 RDF 框架广泛执行。RDQL 允许简明地表达复杂的查询，查询引擎执行访问数据模型的繁重工作。RDQL 的语法表面上类似 SQL 的语法，它的一些概念对已经使用过关系数据库查询的人来说将比较熟悉。在 Jena Web 站点中可以找到极好的 RDQL 指南，但几个简单的示例会对说明基础知识大有帮助。

使用 jena.rdfquery 工具可以在命令行上对 Jena 模型执行 RDQL 查询。RDFQuery 从文本文件中获取 RDQL 查询，然后对指定的模型运行该查询。对数据库后台模型运行查询需要相当多的参数。清单 5 中显示了运行下列示例需要的完整命令行：

清单 5. 从命令行运行 RDQL 查询

```
$java jena.rdfquery --data jdbc:mysql://localhost/jena --user dbuser
--password dbpass
                        --driver com.mysql.jdbc.Driver --dbType MySQL
 --dbName wordnet --query example_query.rdql
```

正如您看到的，这些参数中的大多数参数都提供了创建与 MySQL 的连接所需的详细信息。其中重要的部分是 --query example_query.rdql ，它是 RDQL 文件的位置。还要注意运行 jena.rdfquery 需要 Jena 的 lib 目录中的所有 JAR 文件。

清单 6 显示了您将检查的第一个查询：

清单 6. 查找“domestic dog”的 WordNet 词汇条目的 RDQL 查询

```
SELECT
                    ?definition
                    WHERE
                    (?concept, <wn:wordForm>, "domestic dog"),
                    (?concept, <wn:glossaryEntry>, ?definition)
                    USING
                    wn FOR <http://www.cogsci.princeton.edu/~wn/s
chema/>
```

SELECT 部分声明查询要输出的变量 — 在本例中，是名为 definition 的变量。 WHERE 子句引入第二个变量 concept 并定义与图形匹配的三元组。查询在具有 WHERE 子句中的所有三元组的图形中查找语句。所以，在英语中， WHERE 子句的意思为“查找具有 'domestic dog' 作为单词形式的概念，并查找这些概念的词汇条目”，如图 3 所示。 USING 子句提供一种便利，用于声明名称空间的前缀。

图 3. 清单 6 中的 WHERE 子句匹配的图形

运行查询的结果为：

```
definition
```

```
                    ========================================
================================
                    "a member of the genus Canis (probably descen
ded from the common wolf) that has
                    been domesticated by man since prehistoric ti
mes; occurs in many breeds; "the
                    dog barked all night""
```

所以这种情况仅有一个结果。清单 7 中显示的下个查询的意思为“查找单词 'bear' 表示的概念，并查找这些概念的词汇条目”。

清单 7. 查找“bear”的 WordNet 词汇条目的 RDQL 查询

```
SELECT
                    ?definition
                    WHERE
                    (?concept, <wn:wordForm>, "bear"),
                    (?concept, <wn:glossaryEntry>, ?definition)
                    USING
                    wn FOR <http://www.cogsci.princeton.edu/~wn/s
chema/>
```

此查询返回 15 个结果，因为此单词形式表示多个不同的概念。结果为：

```
definition
                    ========================================
================================
                    "massive plantigrade carnivorous or omnivorou
s mammals with long shaggy coats
                    and strong claws"
                    "an investor with a pessimistic market outloo
```

k"

"have on one's person; "He wore a red ribbon"; "bear a scar""

"give birth (to a newborn); "My wife had twins yesterday!""

清单 8 中显示了另一个示例，查找其他两个单词的上位词（母词）：

清单 8. 查找“panther”和“tiger”的 WordNet 上位词的 RDQL 查询

```
SELECT
                    ?wordform, ?definition
                    WHERE
                    (?firstconcept, <wn:wordForm>, "panther"),
                    (?secondconcept, <wn:wordForm>, "tiger"),
                    (?firstconcept, <wn:hyponymOf>, ?hypernym),
                    (?secondconcept, <wn:hyponymOf>, ?hypernym),
                    (?hypernym, <wn:wordForm>, ?wordform),
                    (?hypernym, <wn:glossaryEntry>, ?definition)
                    USING
                    wn FOR <http://www.cogsci.princeton.edu/~wn/schema/>
```

此处，查询的意思是“查找单词 'panther' 和 'tiger' 所指的概念；查找第三个概念，前两个概念是其下位词；查找第三个概念的可能的单词和词会条目”，如图 4 所示：

图 4. 清单 8 中 WHERE 子句匹配的图形

wordform 和 definition 都在 SELECT 子句中声明，所以它们都是输出。尽管词查询仅匹配了一个 WordNet 概念，查询的图形可以以两种方式匹配，因为该概念有两个不同的单词形式：

```
wordform  | definition
                        ========================================
========================================
                        "big cat" | "any of several large cats typica
lly able to roar and living in the wild"
                        "cat"     | "any of several large cats typica
lly able to roar and living in the wild"
```

使用 Jena 中的 RDQL

Jena 的 com.hp.hpl.jena.rdql 包包含在 Java 代码中使用 RDQL 所需的所有类和接口。要创建 RDQL 查询，将 RDQL 放入 String 中，并将其传送给 Query 的构造函

数。通常直接设置模型用作查询的源，除非在 RDQL 中使用 FROM 子句指定了其他的源。一旦创建了 Query ，可以从它创建 QueryEngine ，然后执行查询。清单 9 中说明了此过程：

清单 9. 创建和运行 RDQL 查询

```
// Create a new query passing a String containing the RDQL to execute
                        Query query = new Query(queryString);
                        // Set the model to run the query against
                        query.setSource(model);
                        // Use the query to create a query engine
                        QueryEngine qe = new QueryEngine(query);
                        // Use the query engine to execute the query
                        QueryResults results = qe.exec();
```

使用 Query 的一个非常有用的方法是在执行之前将它的一些变量设置为固定值。这种使用模式与 javax.sql.PreparedStatement 的相似。变量通过 ResultBinding 对象与值绑定，执行时该对象会传送给 QueryEngine 。可以将变量与 Jena Resource 或与常量值绑定。在将常量与变量绑定之前，通过调用 Model.createLiteral 将其打包。清单 10 说明了预先绑定方法：

清单 10. 将查询变量与值绑定

```
// Create a query that has variables x and y
                        Query query = new Query(queryString);
                        // A ResultBinding specifies mappings between
 query variables and values
                        ResultBinding initialBinding = new ResultBind
ing() ;
                        // Bind the query's first variable to a resou
rce
                        Resource someResource = getSomeResource();
                        initialBinding.add("x", someResource);
```

```
                    // Bind the query's second variable to a lite
ral value
                    RDFNode foo = model.createLiteral("bar");
                    initialBinding.add("y", foo);
                    // Execute the query with the specified value
s for x and y
                    QueryEngine qe = new QueryEngine(query);
                    QueryResults results = qe.exec(initialBindin
g);
```

QueryEngine.exec() 返回的 QueryResults 对象执行 java.util.Iterator 。 next() 方法返回 ResultBinding 对象。查询中使用的所有变量都可以凭名称通过 ResultBinding 获得，而不管它们是否是 SELECT 子句的一部分。清单 11 显示了如何进行此操作，仍使用 清单 6 中的 RDQL 查询：

清单 11. 查找“domestic dog”的 WordNet 词汇条目的 RDQL 查询

```
SELECT
                    ?definition
                    WHERE
                    (?concept, <wn:wordForm>, "domestic dog"),
                    (?concept, <wn:glossaryEntry>, ?definition)
                    USING
                    wn FOR <http://www.cogsci.princeton.edu/~wn/s
chema/>";
```

运行此查询获得的 ResultBinding 如期望的那样包含常量词汇条目。另外，还可以访问变量 concept 。变量通过调用 ResultBinding.get() 凭名称获得。通过此方法返回的所有变量都可以转换成 RDFNode ，如果您想将这些变量绑定回更进一步的 RDQL 查询，这将非常有用。

这种情况下， concept 变量表示 RDF 资源，所以从 ResultBinding.get() 获得的 Object 可以转换成 Resource 。然后可以调用 Resource 的查询方法来进一步探查这部分模型，如清单 12 中所示：

清单 12. 使用查询结果

```
// Execute a query
                        QueryResults results = qe.exec();
                        // Loop over the results
                        while (results.hasNext()) {
                        ResultBinding binding = (ResultBinding)result
s.next();
                        // Print the literal value of the "definition
" variable
                        RDFNode definition = (RDFNode) binding.get("d
efinition");
                        System.out.println(definition.toString());
                        // Get the RDF resource used in the query
                        Resource concept = (Resource)binding.get("con
cept");
                        // Query the concept directly to find other w
ordforms it has
                        List wordforms = concept.listObjectsOfPropert
y(wordForm);
                        }
```

程序源码下载中包含的程序 FindHypernym.java（请参阅 参考资料）汇总了您这里研究的区域。它查找命令行上给定单词的上位词，清单 13 中显示了使用的查询：

清单 13. 查找概念的上位词的单词形式和词汇条目的 RDQL 查询

```
SELECT
                        ?hypernym, ?definition
```

```
                        WHERE
                        (?firstconcept, <wn:wordForm>, ?hyponym),
                        (?firstconcept, <wn:hyponymOf>, ?secondconcep
t),
                        (?secondconcept, <wn:wordForm>, ?hypernym),
                        (?secondconcept, <wn:glossaryEntry>, ?definit
ion)
                        USING
                        wn FOR <http://www.cogsci.princeton.edu/~wn/s
chema/>
```

命令行上给出的单词与 hyponym 词语绑定，查询查找该单词表示的概念，查找第二个概念（第一个概念是它的下位词），然后输出第二个概念的单词形式和定义。清单 14 显示了输出：

清单 14. 运行示例 FindHypernym 程序

```
$ java FindHypernym "wisteria"
                        Hypernyms found for 'wisteria':
                        vine:weak-stemmed plant that derives support
from climbing, twining,
                        or creeping along a surface
```

使用 OWL 添加意义

您可能想知道为什么“wisteria”的上位词搜索仅返回它的直接上位词“vine”。如果从植物学观点，您可能还希望显示“traceophyte”也显示为上位词，以及“plant”。实际上，WordNet 模型表明“wisteria”是“vine”的下位词，“vine”是“traceophyte”的下位词。直观地，您知道“wisteria”因此是“traceophyte”的下位词，因为您知道“hyponym of”关系是 *可传递的*。所以您需要有一种方法将这种认识合并到 FindHypernym 程序中，从而产生了 OWL。

Web Ontology Language 或 OWL 是 W3C 推荐标准，设计用来“明确表示词汇表中词语的意义以及那些词语之间的关系”。与 RDF Schema 一起，OWL 提供了一种正式地描述 RDF 模型的机制。除了定义资源可以属于的层次结构类，OWL 还允许表达资源的属性特征。例如，在 清单 1 中使用的 Relationship 词汇表中，可以使用 OWL 说明 childOf 属性与 parentOf 属性相反。另一个示例说明 WordNet 词汇表的 hyponymOf 属性是可传递的。

可传递关系

关系对于三个元素 *a*、 *b* 和 *c* 是可传递的，从而 *a* 和 *b* 之间及 *b* 和 *c* 之间存在关系意味着 *a* 和 *c* 之间存在关系。

可传递关系的一个示例是“大于”关系。如果 *a* 大于 *b*， *b* 大于 *c*，因而 *a* 肯定大于 *c*。

在 Jena 中，本体被看作一种特殊类型的 RDF 模型 OntModel 。此接口允许程序化地对本地进行操作，使用便利方法创建类、属性限制等等。备选方法将本体看作特殊 RDF 模型，仅添加定义其语义规则的语句。清单 15 中说明了这些技术。注意还可以将本体语句添加到现有数据模型中，或使用 Model.union() 将本体模型与数据模型合并。

清单 15. 创建 WordNet 的 OWL 本体模型

```
// Make a new model to act as an OWL ontology for WordNet
                        OntModel wnOntology = ModelFactory.createOntologyModel();
                        // Use OntModel's convenience method to describe
                        // WordNet's hyponymOf property as transitive
                        wnOntology.createTransitiveProperty(WordnetVocab.hyponymOf.getURI());
                        // Alternatively, just add a statement to the underlying model to express that
                        // hyponymOf is of type TransitiveProperty
```

```
wnOntology.add(WordnetVocab.hyponymOf, RDF.type, OWL.TransitiveProperty);
```

使用 Jena 推理

给定了本体和模型后，Jena 的推理引擎可以派生模型未明确表达的其他语句。Jena 提供了多个 Reasoner 类型来使用不同类型的本体。因为要将 OWL 本体与 WordNet 模型一起使用，所以需要 OWLReasoner 。

下例显示了如何将 OWL WordNet 本体应用到 WordNet 模型自身以创建推理模型。这里我实际将使用 WordNet 模型的子集，仅包含下位词层次结构中“plant life”之下的那些名词。仅使用子集的原因是推理模型需要保存在内存中，WordNet 模型对于内存模型过大而不能实现。我用来从整个 WordNet 模型中提取 plants 模型的代码包含在文章来源中，名为 ExtractPlants.java（请参阅 参考资料）。

首先我从 ReasonerRegistry 中获得 OWLReasoner 。 ReasonerRegistry.getOWLReasoner() 在它的标准配置中返回 OWL reasoner，这对于此简单情况已经足够。下一步是将 reasoner 与 WordNet 本体绑定。此操作返回可以应用本体规则的 reasoner。然后，将使用绑定的 reasoner 从 WordNet 模型创建 InfModel 。

从原始数据和 OWL 本体创建了推理模型后，它就可以像任何其他 Model 实例一样进行处理。因此，如清单 16 所示，通过 FindHypernym.java 与正常 Jena 模型一起使用的 Java 代码和 RDQL 查询可以重新应用到推理模型，而不进行任何更改：

清单 16. 创建和查询推理模型

```
// Get a reference to the WordNet plants model
```

```
                        ModelMaker maker = ModelFactory.createModelRD
BMaker(connection);
                        Model model = maker.openModel("wordnet-plants
",true);
                        // Create an OWL reasoner
                        Reasoner owlReasoner = ReasonerRegistry.getOW
LReasoner();
                        // Bind the reasoner to the WordNet ontology
model
                        Reasoner wnReasoner = owlReasoner.bindSchema
(wnOntology);
                        // Use the reasoner to create an inference mo
del
                        InfModel infModel = ModelFactory.createInfMod
el(wnReasoner, model);
                        // Set the inference model as the source of t
he
                        query query.setSource(infModel);
                        // Execute the query as normal
                        QueryEngine qe = new QueryEngine(query);
                        QueryResults results = qe.exec(initialBindin
g);
```

文章来源中有完整清单，名为 FindInferredHypernyms.java。清单 17 显示了当对推理模型查询“wisteria”的上位词时的结果：

清单 17. 运行示例 FindInferredHypernyms 程序

```
$ java FindInferredHypernyms wisteria
                        Hypernyms found for 'wisteria':
                        vine:weak-stemmed plant that derives support
from climbing, twining, or creeping along a surface
                        tracheophyte:green plant having a vascular sy
```

```
stem:ferns, gymnosperms, angiosperms
                        vascular plant:green plant having a vascular
system:ferns, gymnosperms, angiosperms
                        plant life:a living organism lacking the powe
r of locomotion
                        flora:a living organism lacking the power of
locomotion
                        plant:a living organism lacking the power of
locomotion
```

OWL 本体中包含的信息已经使 Jena 可以推断“wisteria”在模型中有上位词。

结束语

本文说明了 Jena Semantic Web Toolkit 的一些最重要的功能，并用示例说明了如何创建、导入和持久化 RDF 模型。您已经了解了查询模型的不同方法，并看到了如何使用 RDQL 简明地表达任意查询。另外，您还了解了如何使用 Jena 的推理引擎对基于本体的模型进行推理。

本文中的示例已经说明了将数据表示为 RDF 模型的一些效果，以及 RDQL 从这些模型中提取数据的灵活性。当在您自己的 Java 应用程序中使用 RDF 模型时，这里说明的基本方法将是非常有用的起点。

Jena 是综合的 RDF 工具集，它的功能远不止您这里了解的这些。Jena 项目的主页是开始学习其功能的好地方。

通过 Jena Semantic Web Framework 在 Jave 应用程序中使用 RDF 模型

Philip McCarthy (phil@planetrdf.com), 开发人员, SmartStream Technologies Ltd

2004 年 7 月 01 日

RDF 越来越被认为是表示和处理半结构化数据的一种极好选择。本文中，Web 开发人员 Philip McCarthy 向您展示了如何使用 Jena Semantic Web Toolkit，以便在 Java 应用程序中使用 RDF 数据模型。

“资源描述框架（Resource Description Framework，RDF）”最近成为 W3C 推荐标准，与 XML 和 SOAP 等 Web 标准并排。RDF 可以应用于处理特殊输入数据（如 CRM）的领域，已经广泛用于社会网络和自助出版软件（如 LiveJournal 和 TypePad）。

Java 程序员将越来越多地得益于具有使用 RDF 模型的技能。在本文中，我将带您体验惠普实验室的开放源代码 Jena Semantic Web Framework（请参阅 参考资料）的一些功能。您将了解如何创建和填充 RDF 模型，如何将它们持久存储到数据库中，以及如何使用 RDQL 查询语言以程序方式查询这些模型。最后，我将说明如何使用 Jena 的推理能力从本体推断模型知识。

本文假设您已经就图形、三元组和模式等概念方面对 RDF 比较熟悉，并对 Java 编程有基本的了解。

创建简单的 RDF 模型

我们从基本操作开始：从头创建模型并向其添加 RDF 语句。本节，我将说明如何创建描述一组虚构家庭成员之间关系的模型，如图 1 中所示：

图 1. 虚拟家庭树

将使用来自“关系”词汇表（请参阅 参考资料）的属性 siblingOf 、 spouseOf 、 parentOf 和 childOf 来描述不同的关系类型。为简单起见，家庭成员用来自虚构名称空间的 URI （ http://family/ ）进行标识。词汇表 URI 通常以 Jena 代码形式使用，所以将它们声明为 Java 常量会非常有用，减少了错误输入。

Schemagen

当您通过 Jena 的 API 来使用模型时，为模型词汇表中的每个属性定义常量非常有用。如果有词汇表的 RDF、DAML 或 OWL 表示，Jena 的 Schemagen 工具可以自动生成这些常量，使您的工作更加容易。

Schemagen 在命令行中运行，使用的参数包括模式或本体文件的位置、要输出的类的名称

和 Java 包。然后可以导出生成的 Java 类，其 Property 常量用于访问模型。

还可以使用 Ant 将 Schemagen 作为构建处理的一部分来运行，保持 Java 常量类与正在变化的词汇表保持同步。

Jena 的 ModelFactory 类是创建不同类型模型的首选方式。在这种情况下，您想要空的、内存模型，所以要调用的方法是 ModelFactory.createDefaultModel() 。这种方法返回 Model 实例，您将使用它创建表示家庭中每个成员的 Resource 。创建了资源后，可以编写关于这些资源的语句并添加到模型中。

在 Jena 中，语句的主题永远是 Resource ，谓词由 Property 表示，对象是另一个 Resource 或常量值。常量在 Jena 中通过 Literal 类型表示。所有这些类型共享公共接口 RDFNode 。将需要四个不同的 Property 实例表示家庭树中的关系。这些实例使用 Model.createProperty() 创建。

将语句添加到模型中的最简单方法是通过调用 Resource.addProperty() 。此方法以 Resource 作为主题在模型中创建语句。该方法使用两个参数，表示语句谓词的 Property 和语句的对象。addProperty() 方法被过载：一个过载使用 RDFNode 作为对象，所以可以使用 Resource 或 Literal 。还有有益过载，它们使用由 Java 原语或 String 表示的常量。在示例中，语句的对象是表示其他家庭成员的 Resource 。

通过使用三元组的主题、谓词和对象调用 Model.createStatement() ，还可以直接在模型上创建语句。注意以此种方式创建 Statement 不将其添加到模型中。如果想将其添加到模型中，请使用创建的 Statement 调用 Model.add() ，如清单 1 所示：

清单 1. 创建模型来表示虚构的家庭

```
// URI declarations
String familyUri = "http://family/";
String relationshipUri = "http://purl.org/vocab/relationship/";
// Create an empty Model
Model model = ModelFactory.createDefaultModel();
// Create a Resource for each family member, identified by their
URI Resource adam = model.createResource(familyUri+"adam");
Resource beth = model.createResource(familyUri+"beth");
Resource chuck = model.createResource(familyUri+"chuck");
Resource dotty = model.createResource(familyUri+"dotty");
// and so on for other family members
// Create properties for the different types of relationship to represent
Property childOf = model.createProperty(relationshipUri,"childOf");
Property parentOf = model.createProperty(relationshipUri,"parentOf");
Property siblingOf = model.createProperty(relationshipUri,"siblingOf");
Property spouseOf = model.createProperty(relationshipUri,"spouseOf");
```

```
// Add properties to adam describing relationships to other family members
adam.addProperty(siblingOf,beth);
adam.addProperty(spouseOf,dotty);
adam.addProperty(parentOf,edward);
// Can also create statements directly .. .
Statement statement = model.createStatement(adam,parentOf,fran);
// but remember to add the created statement to the model
model.add(statement);
```

整个代码示例 FamilyModel.java 还说明了语句批量如何一次添加到模型中，或者作为一个数组或者作为 java.util.List 。

构建了家庭模型后，我们看一下如何使用 Jena 的查询 API 从模型中提取信息。

回页首

查询 RDF 模型

程序化地查询 Jena 模型主要通过 list() 方法在 Model 和 Resource 接口中执行。可以使用这些方法获得满足特定条件的主题、对象和 Statement 。它们还返回 java.util.Iterator 的特殊化，其具有返回特定对象类型的其他方法。

我们返回 清单 1 的家庭模型，看一下可以查询它的不同方法，如清单 2 所示：

清单 2. 查询家庭模型

```
// List everyone in the model who has a child:
ResIterator parents = model.listSubjectsWithProperty(parentOf);
// Because subjects of statements are Resources, the method returned a ResIterator
while (parents.hasNext()) {
// ResIterator has a typed nextResource() method
Resource person = parents.nextResource();
// Print the URI of the resource
System.out.println(person.getURI()); }
```

```
// Can also find all the parents by getting the objects of all "childOf" statements
// Objects of statements could be Resources or literals, so the Iterator returned
// contains RDFNodes
NodeIterator moreParents = model.listObjectsOfProperty(childOf);
// To find all the siblings of a specific person, the model itself can be queried
NodeIterator siblings = model.listObjectsOfProperty(edward, siblingOf);

// But it's more elegant to ask the Resource directly
// This method yields an iterator over Statements
StmtIterator moreSiblings = edward.listProperties(siblingOf);
```

最通用的查询方法是 Model.listStatements(Resource s, Property p, RDFNode o) ，下面说明的便利方法都是以其为基础。所有这些参数都可以保留为 null ，在这种情况下，它们作为通配符，与任何数据都匹配。清单 3 中显示了 Model.listStatements() 的一些使用示例：

清单 3. 使用选择器查询模型

```
// Find the exact statement "adam is a spouse of dotty"
model.listStatements(adam,spouseOf,dotty);

// Find all statements with adam as the subject and dotty as the object
model.listStatements(adam,null,dotty);

// Find any statements made about adam
model.listStatements(adam,null,null);
// Find any statement with the siblingOf property
model.listStatements(null,siblingOf,null);
```

回页首

导入和持久化模型

不是所有的应用程序都从空模型开始。更常见的是，在开始时从现有数据填充模型。在这种

情况下，使用内存模型的缺点是每次启动应用程序时都要从头重新填充模型。另外，每次关闭应用程序时，对内存模型进行的更改都将丢失。

一种解决方案是使用 Model.write() 序列化模型到文件系统，然后在开始时使用 Model.read() 将其取消序列化。不过，Jena 还提供了持久化模型，它们会被持续而透明地持久存储到后备存储器。Jena 可以在文件系统中或在关系数据库中持久化它的模型。当前支持的数据库引擎是 PostgreSQL、Oracle 和 MySQL。

WordNet

WordNet 是“英文语言的词汇数据库”。我使用的是 Sergey Melnik 和 Stefan Decker 的 RDF 表示。它具有四个单独的模型，本文示例中将使用其中三个模型。

WordNet-nouns 模型包含 WordNet 表示的所有“词汇概念”和用于表示每个概念的“单词形式”。例如，它包含由单词形式“domestic dog”、“dog”和“Canis familiaris”表示的词汇概念。

第二个模型是 WordNet-glossary。它提供模型中每个词汇概念的简短定义。“dog”的词汇概念具有词汇条目“a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times.”

WordNet-hyponyms 是第三个模型。它定义模型中概念的层次结构。概念“dog”是概念“canine”下位词，而“canine” 本身是概念“carnivore”的下位词。

为了说明如何导入和持久化模型，我将 WordNet 1.6 数据库的 RDF 表示导入到 MySQL 中。因为我使用的 WordNet 表示采用多个单独 RDF 文档的形式，将这些文档导入到一个 Jena 模型中会合并它们的语句。 图 2 说明了 Nouns 和 Glossary 模型合并后 WordNet 模型的片段的结构：

图 2. 合并的 WordNet nouns 和 glossary 模型的结构

创建数据库后台模型的第一步是说明 MySQL 驱动类，并创建 DBConnection 实例。DBConnection 构造函数使用用户的 ID 和密码登录到数据库。它还使用包含 Jena 使用的 MySQL 数据库名称的数据库 URL 参数，格式为 "jdbc:mysql://localhost/dbname" 。Jena 可以在一个数据库内创建多个模型。 DBConnection 的最后一个参数是数据库类型，对于 MySQL，该参数为 "MySQL" 。

然后 DBConnection 实例可以与 Jena 的 ModelFactory 一起使用来创建数据库后台模型。

创建了模型后，可以从文件系统中读入 WordNet RDF 文档。不同的 Model.read() 方法可以从 Reader 、 InputStream 或 URL 填充模型。可以通过 Notation3 、N-Triples 或默认

情况下通过 RDF/XML 语法解析模型。WordNet 作为 RDF/XML 进行序列化，所以不需要指定语法。读取模型时，可以提供基准 URI。基准 URI 用于将模型中的任何相对 URI 转换成绝对 URI。因为 WordNet 文档不包含任何相对 URI，所以此参数可以指定为 null 。

清单 4 显示了将 WordNet RDF/XML 文件导入到 MySQL 持久化模型的完整过程：

清单 4. 导入和持久化 WordNet 模型

```
// Instantiate the MySQL driver
Class.forName("com.mysql.jdbc.Driver");
// Create a database connection object
DBConnection connection = new DBConnection(DB_URL, DB_USER, DB_PASSWORD, DB_TYPE);

// Get a ModelMaker for database-backed models
ModelMaker maker = ModelFactory.createModelRDBMaker(connection);
// Create a new model named "wordnet."Setting the second parameter to "true" causes an
// AlreadyExistsException to be thrown if the db already has a model with this name
Model wordnetModel = maker.createModel("wordnet",true);
// Start a database transaction.Without one, each statement will be auto-committed
// as it is added, which slows down the model import significantly.
model.begin();
// For each wordnet model .. .
InputStream in = this.getClass().getClassLoader().getResourceAsStream(filename);
model.read(in,null);
// Commit the database transaction model.
commit();
```

由于已经填充了 wordnet 模型，以后可以通过调用 ModelMaker.openModel("wordnet",true); 来访问该模型。

仅使用 Jena 的 API 查询像 WordNet 这样巨大的模型将有一定的限制性，因为要执行的每类查询都将需要专门编写多行的代码。幸运的是，Jena 以 RDQL 形式提供了一种表达通用查询的机制。

回页首

RDF 数据查询语言（RDQL）

RDQL 是 RDF 的查询语言。虽然 RDQL 还不是正是的标准，但已由 RDF 框架广泛执行。RDQL 允许简明地表达复杂的查询，查询引擎执行访问数据模型的繁重工作。RDQL 的语法表面上类似 SQL 的语法，它的一些概念对已经使用过关系数据库查询的人来说将比较熟悉。在 Jena Web 站点中可以找到极好的 RDQL 指南，但几个简单的示例会对说明基础知识大有帮助。

使用 jena.rdfquery 工具可以在命令行上对 Jena 模型执行 RDQL 查询。RDFQuery 从文本文件中获取 RDQL 查询，然后对指定的模型运行该查询。对数据库后台模型运行查询需要相当多的参数。清单 5 中显示了运行下列示例需要的完整命令行：

清单 5. 从命令行运行 RDQL 查询

```
$java jena.rdfquery --data jdbc:mysql://localhost/jena --user dbuser --password dbpass
--driver com.mysql.jdbc.Driver --dbType MySQL --dbName wordnet --query example_query.rdql
```

正如您看到的，这些参数中的大多数参数都提供了创建与 MySQL 的连接所需的详细信息。其中重要的部分是 --query example_query.rdql ，它是 RDQL 文件的位置。还要注意运行 jena.rdfquery 需要 Jena 的 lib 目录中的所有 JAR 文件。

清单 6 显示了您将检查的第一个查询：

清单 6. 查找“domestic dog”的 WordNet 词汇条目的 RDQL 查询

```
SELECT
    ?definition
WHERE
    (?concept, <wn:wordForm>, "domestic dog"),
    (?concept, <wn:glossaryEntry>, ?definition)
USING
    wn FOR <http://www.cogsci.princeton.edu/~wn/schema/>
```

SELECT 部分声明查询要输出的变量 — 在本例中，是名为 definition 的变量。 WHERE 子句引入第二个变量 concept 并定义与图形匹配的三元组。查询在具有 WHERE 子句中的所有三元组的图形中查找语句。所以，在英语中， WHERE 子句的意思为“查找具有 'domestic dog' 作为单词形式的概念，并查找这些概念的词汇条目”，如图 3 所示。 USING 子句提供一种便利，用于声明名称空间的前缀。

图 3. 清单 6 中的 WHERE 子句匹配的图形

运行查询的结果为：

```
definition
=========================================================================
=========
"a member of the genus Canis (probably descended from the common wolf) that has
been domesticated by man since prehistoric times; occurs in many breeds; "the
dog barked all night""
```

所以这种情况仅有一个结果。清单 7 中显示的下个查询的意思为“查找单词 'bear' 表示的概念，并查找这些概念的词汇条目”。

清单 7. 查找“bear”的 WordNet 词汇条目的 RDQL 查询

```
SELECT
    ?definition
WHERE
    (?concept, <wn:wordForm>, "bear"),
    (?concept, <wn:glossaryEntry>, ?definition)
USING
    wn FOR <http://www.cogsci.princeton.edu/~wn/schema/>
```

此查询返回 15 个结果，因为此单词形式表示多个不同的概念。结果为：

```
definition
=========================================================================
=========
"massive plantigrade carnivorous or omnivorous mammals with long shaggy coats
and strong claws"
"an investor with a pessimistic market outlook"
"have on one's person; "He wore a red ribbon"; "bear a scar""
"give birth (to a newborn); "My wife had twins yesterday!""
```

清单 8 中显示了另一个示例，查找其他两个单词的上位词（母词）：

清单 8. 查找“panther”和“tiger”的 WordNet 上位词的 RDQL 查询

```
SELECT
    ?wordform, ?definition
WHERE
    (?firstconcept, <wn:wordForm>, "panther"),
    (?secondconcept, <wn:wordForm>, "tiger"),
    (?firstconcept, <wn:hyponymOf>, ?hypernym),
    (?secondconcept, <wn:hyponymOf>, ?hypernym),
    (?hypernym, <wn:wordForm>, ?wordform),
    (?hypernym, <wn:glossaryEntry>, ?definition)
USING
    wn FOR <http://www.cogsci.princeton.edu/~wn/schema/>
```

此处，查询的意思是“查找单词 'panther' 和 'tiger' 所指的概念；查找第三个概念，前两个概念是其下位词；查找第三个概念的可能的单词和词会条目”，如图 4 所示：

图 4. 清单 8 中 WHERE 子句匹配的图形

wordform 和 definition 都在 SELECT 子句中声明，所以它们都是输出。尽管词查询仅匹配了一个 WordNet 概念，查询的图形可以以两种方式匹配，因为该概念有两个不同的单词形式：

```
wordform   | definition
================================================================================
"big cat" | "any of several large cats typically able to roar and living in the wild"
"cat"         | "any of several large cats typically able to roar and living in the wild"
```

回页首

使用 Jena 中的 RDQL

Jena 的 com.hp.hpl.jena.rdql 包包含在 Java 代码中使用 RDQL 所需的所有类和接口。要创建 RDQL 查询，将 RDQL 放入 String 中，并将其传送给 Query 的构造函数。通常直接设置模型用作查询的源，除非在 RDQL 中使用 FROM 子句指定了其他的源。一旦创建了 Query ，可以从它创建 QueryEngine ，然后执行查询。清单 9 中说明了此过程：

清单 9. 创建和运行 RDQL 查询

```
// Create a new query passing a String containing the RDQL to execute
Query query = new Query(queryString);
// Set the model to run the query against
query.setSource(model);

// Use the query to create a query engine
QueryEngine qe = new QueryEngine(query);
// Use the query engine to execute the query
QueryResults results = qe.exec();
```

使用 Query 的一个非常有用的方法是在执行之前将它的一些变量设置为固定值。这种使用模式与 javax.sql.PreparedStatement 的相似。变量通过 ResultBinding 对象与值绑定，执行时该对象会传送给 QueryEngine 。可以将变量与 Jena Resource 或与常量值绑定。在将常量与变量绑定之前，通过调用 Model.createLiteral 将其打包。清单 10 说明了预先绑定方法：

清单 10. 将查询变量与值绑定

```
// Create a query that has variables x and y
Query query = new Query(queryString);
// A ResultBinding specifies mappings between query variables and values
ResultBinding initialBinding = new ResultBinding() ;
// Bind the query's first variable to a resource
Resource someResource = getSomeResource();
initialBinding.add("x", someResource);
// Bind the query's second variable to a literal value
RDFNode foo = model.createLiteral("bar");
initialBinding.add("y", foo);
// Execute the query with the specified values for x and y
QueryEngine qe = new QueryEngine(query);
QueryResults results = qe.exec(initialBinding);
```

QueryEngine.exec() 返回的 QueryResults 对象执行 java.util.Iterator 。 next() 方法返回 ResultBinding 对象。查询中使用的所有变量都可以凭名称通过 ResultBinding 获得，而不管它们是否是 SELECT 子句的一部分。清单 11 显示了如何进行此操作，仍使用 清单 6 中的 RDQL 查询：

清单 11. 查找“domestic dog”的 WordNet 词汇条目的 RDQL 查询

```
SELECT
    ?definition
WHERE
    (?concept, <wn:wordForm>, "domestic dog"),
    (?concept, <wn:glossaryEntry>, ?definition)
USING
    wn FOR <http://www.cogsci.princeton.edu/~wn/schema/>";
```

运行此查询获得的 ResultBinding 如期望的那样包含常量词汇条目。另外，还可以访问变量 concept 。变量通过调用 ResultBinding.get() 凭名称获得。通过此方法返回的所有变量都可以转换成 RDFNode ，如果您想将这些变量绑定回更进一步的 RDQL 查询，这将非常有用。

这种情况下， concept 变量表示 RDF 资源，所以从 ResultBinding.get() 获得的 Object 可以转换成 Resource 。然后可以调用 Resource 的查询方法来进一步探查这部分模型，如清单 12 中所示：

清单 12. 使用查询结果

```
// Execute a query
QueryResults results = qe.exec();
// Loop over the results
while (results.hasNext()) {
    ResultBinding binding = (ResultBinding)results.next();

    // Print the literal value of the "definition" variable
    RDFNode definition = (RDFNode) binding.get("definition");
    System.out.println(definition.toString());
    // Get the RDF resource used in the query
    Resource concept = (Resource)binding.get("concept");
    // Query the concept directly to find other wordforms it has
    List wordforms = concept.listObjectsOfProperty(wordForm);
}
```

程序源码下载中包含的程序 FindHypernym.java（请参阅 参考资料）汇总了您这里研究的区域。它查找命令行上给定单词的上位词，清单 13 中显示了使用的查询：

清单 13. 查找概念的上位词的单词形式和词汇条目的 RDQL 查询

```
SELECT
    ?hypernym, ?definition
WHERE
    (?firstconcept, <wn:wordForm>, ?hyponym),
    (?firstconcept, <wn:hyponymOf>, ?secondconcept),
    (?secondconcept, <wn:wordForm>, ?hypernym),
    (?secondconcept, <wn:glossaryEntry>, ?definition)
USING
    wn FOR <http://www.cogsci.princeton.edu/~wn/schema/>
```

命令行上给出的单词与 hyponym 词语绑定，查询查找该单词表示的概念，查找第二个概念（第一个概念是它的下位词），然后输出第二个概念的单词形式和定义。清单 14 显示了输出：

清单 14. 运行示例 FindHypernym 程序

```
$ java FindHypernym "wisteria"
Hypernyms found for 'wisteria':
vine:weak-stemmed plant that derives support from climbing, twining,
or creeping along a surface
```

回页首

使用 OWL 添加意义

您可能想知道为什么“wisteria”的上位词搜索仅返回它的直接上位词“vine”。如果从植物学观点，您可能还希望显示“traceophyte”也显示为上位词，以及“plant”。实际上，WordNet

模型表明“wisteria”是“vine”的下位词，“vine”是“traceophyte”的下位词。直观地，您知道“wisteria”因此是“traceophyte”的下位词，因为您知道“hyponym of”关系是 可传递的。所以您需要有一种方法将这种认识合并到 FindHypernym 程序中，从而产生了 OWL。

可传递关系

关系对于三个元素 a、 b 和 c 是可传递的，从而 a 和 b 之间及 b 和 c 之间存在关系意味着 a 和 c 之间存在关系。

可传递关系的一个示例是“大于”关系。如果 a 大于 b， b 大于 c，因而 a 肯定大于 c。

Web Ontology Language 或 OWL 是 W3C 推荐标准，设计用来“明确表示词汇表中词语的意义以及那些词语之间的关系”。与 RDF Schema 一起，OWL 提供了一种正式地描述 RDF 模型的机制。除了定义资源可以属于的层次结构类，OWL 还允许表达资源的属性特征。例如，在 清单 1 中使用的 Relationship 词汇表中，可以使用 OWL 说明 childOf 属性与 parentOf 属性相反。另一个示例说明 WordNet 词汇表的 hyponymOf 属性是可传递的。

在 Jena 中，本体被看作一种特殊类型的 RDF 模型 OntModel 。此接口允许程序化地对本地进行操作，使用便利方法创建类、属性限制等等。备选方法将本体看作特殊 RDF 模型，仅添加定义其语义规则的语句。清单 15 中说明了这些技术。注意还可以将本体语句添加到现有数据模型中，或使用 Model.union() 将本体模型与数据模型合并。

清单 15. 创建 WordNet 的 OWL 本体模型

```
// Make a new model to act as an OWL ontology for WordNet
OntModel wnOntology = ModelFactory.createOntologyModel();
// Use OntModel's convenience method to describe
// WordNet's hyponymOf property as transitive
wnOntology.createTransitiveProperty(WordnetVocab.hyponymOf.getURI());
// Alternatively, just add a statement to the underlying model to express that
// hyponymOf is of type TransitiveProperty
wnOntology.add(WordnetVocab.hyponymOf, RDF.type, OWL.TransitiveProperty);
```

回页首

使用 Jena 推理

给定了本体和模型后，Jena 的推理引擎可以派生模型未明确表达的其他语句。Jena 提供了多个 Reasoner 类型来使用不同类型的本体。因为要将 OWL 本体与 WordNet 模型一起使用，所以需要 OWLReasoner 。

下例显示了如何将 OWL WordNet 本体应用到 WordNet 模型自身以创建推理模型。这里我实际将使用 WordNet 模型的子集，仅包含下位词层次结构中“plant life”之下的那些名词。仅使用子集的原因是推理模型需要保存在内存中，WordNet 模型对于内存模型过大而不能实现。我用来从整个 WordNet 模型中提取 plants 模型的代码包含在文章来源中，名为 ExtractPlants.java（请参阅 参考资料）。

首先我从 ReasonerRegistry 中获得 OWLReasoner 。 ReasonerRegistry.getOWLReasoner() 在它的标准配置中返回 OWL reasoner，这对于此简单情况已经足够。下一步是将 reasoner 与 WordNet 本体绑定。此操作返回可以应用本体规则的 reasoner。然后，将使用绑定的 reasoner 从 WordNet 模型创建 InfModel 。

从原始数据和 OWL 本体创建了推理模型后，它就可以像任何其他 Model 实例一样进行处理。因此，如清单 16 所示，通过 FindHypernym.java 与正常 Jena 模型一起使用的 Java 代码和 RDQL 查询可以重新应用到推理模型，而不进行任何更改：

清单 16. 创建和查询推理模型

```
// Get a reference to the WordNet plants model
ModelMaker maker = ModelFactory.createModelRDBMaker(connection);
Model model = maker.openModel("wordnet-plants",true);
// Create an OWL reasoner
Reasoner owlReasoner = ReasonerRegistry.getOWLReasoner();
// Bind the reasoner to the WordNet ontology model
Reasoner wnReasoner = owlReasoner.bindSchema(wnOntology);
// Use the reasoner to create an inference model
InfModel infModel = ModelFactory.createInfModel(wnReasoner, model);
// Set the inference model as the source of the
query query.setSource(infModel);
// Execute the query as normal
QueryEngine qe = new QueryEngine(query);
QueryResults results = qe.exec(initialBinding);
```

文章来源中有完整清单，名为 FindInferredHypernyms.java。清单 17 显示了当对推理模型查询“wisteria”的上位词时的结果：

清单 17. 运行示例 FindInferredHypernyms 程序

```
$ java FindInferredHypernyms wisteria
Hypernyms found for 'wisteria':
vine:weak-stemmed plant that derives support from climbing, twining, or creeping along a surface
tracheophyte:green plant having a vascular system:ferns, gymnosperms, angiosperms
vascular plant:green plant having a vascular system:ferns, gymnosperms, angiosperms
plant life:a living organism lacking the power of locomotion
flora:a living organism lacking the power of locomotion
plant:a living organism lacking the power of locomotion
```

OWL 本体中包含的信息已经使 Jena 可以推断“wisteria”在模型中有上位词。

回页首

结束语

本文说明了 Jena Semantic Web Toolkit 的一些最重要的功能，并用示例说明了如何创建、导入和持久化 RDF 模型。您已经了解了查询模型的不同方法，并看到了如何使用 RDQL 简明地表达任意查询。另外，您还了解了如何使用 Jena 的推理引擎对基于本体的模型进行推理。

本文中的示例已经说明了将数据表示为 RDF 模型的一些效果，以及 RDQL 从这些模型中提取数据的灵活性。当在您自己的 Java 应用程序中使用 RDF 模型时，这里说明的基本方法将是非常有用的起点。

Jena 是综合的 RDF 工具集，它的功能远不止您这里了解的这些。Jena 项目的主页是开始学习其功能的好地方。

回页首

下载

名字 大小 下载方法
j-jena-examples.zip HTTP

关于下载方法的信息

参考资料

您可以参阅本文在 developerWorks 全球站点上的 英文原文.

jena.sourceforge.net 是 Jena 项目的主页，其根据 BSD-style 许可证分发。在这里您将找到大量的文档，可以下载 Jena 框架的最新版本。

当以 RDF 描述人们相互间的关系时， Relationship 词汇表非常有用。

Jena-dev 致力于 Jena 的开发人员的邮件列表。如果您有关于 Jena 的问题，可以从这里得到答案。

HP Labs Semantic Web Research Group 的其他语义 Web 工具和出版物可以在他们的 Web 站点上找到。

在 Jena 站点上可以找到 “ A Programmer's Introduction to RDQL”，这是一本带有许多示例的综合教程。

在 The WordNet project 的主页上可以找到更多关于它的有用资源信息。

可以从 SemanticWeb.org 获得本文中使用的 WordNet RDF 表示。还可以获得 RDF 表示的 RDF Schema。

SchemaWeb 是 RDF Schema 和 OWL 本体的目录。当查找用来表达数据的词汇表时，可以从这里开始。

Dave Beckett 在 RDF Resource Guide 页面维护语义 Web 和 RDF 链接的完整集合。

Uche Ogbuji 在他的 Basic XML and RDF techniques for knowledge management 系列文章中研究了 RDF 的应用程序（developerWorks, July 2001).

Stefano Mazzocchi 的文章“ It's All About Graphs” 讨论了以 RDF 图形代替数据库表和 XML 文档表示数据的灵活性。

Shelley Power 的书 Practical RDF (O'Reilly，2003) 从 RDF 的基本概念到实际的应用程序对 RDF 进行了研究。

" An introduction to RDF" (developerWorks, December 2000) 是 Uche Ogbuji 的另一篇文章，概述了 RDF 的一些基础知识。

W3C 的 OWL Overview 是了解 OWL 功能的最好地方。

请访问 Developer Bookstore，获取技术书籍的完整列表，其中包括数百本 Java 相关的图书。

在 developerWorksJava 技术专区 可以找到有关 Java 编程各个方面的数百篇文章。

关于作者

Philip McCarthy 是一名 Web 开发人员，专门研究 J2EE 和前沿技术。他有四年的 Java 编程经验，曾在 Orange 从事客户 Internet 应用程序开发。他目前在 SmartStream Technologies 开发基于 Web 的财务应用程序。