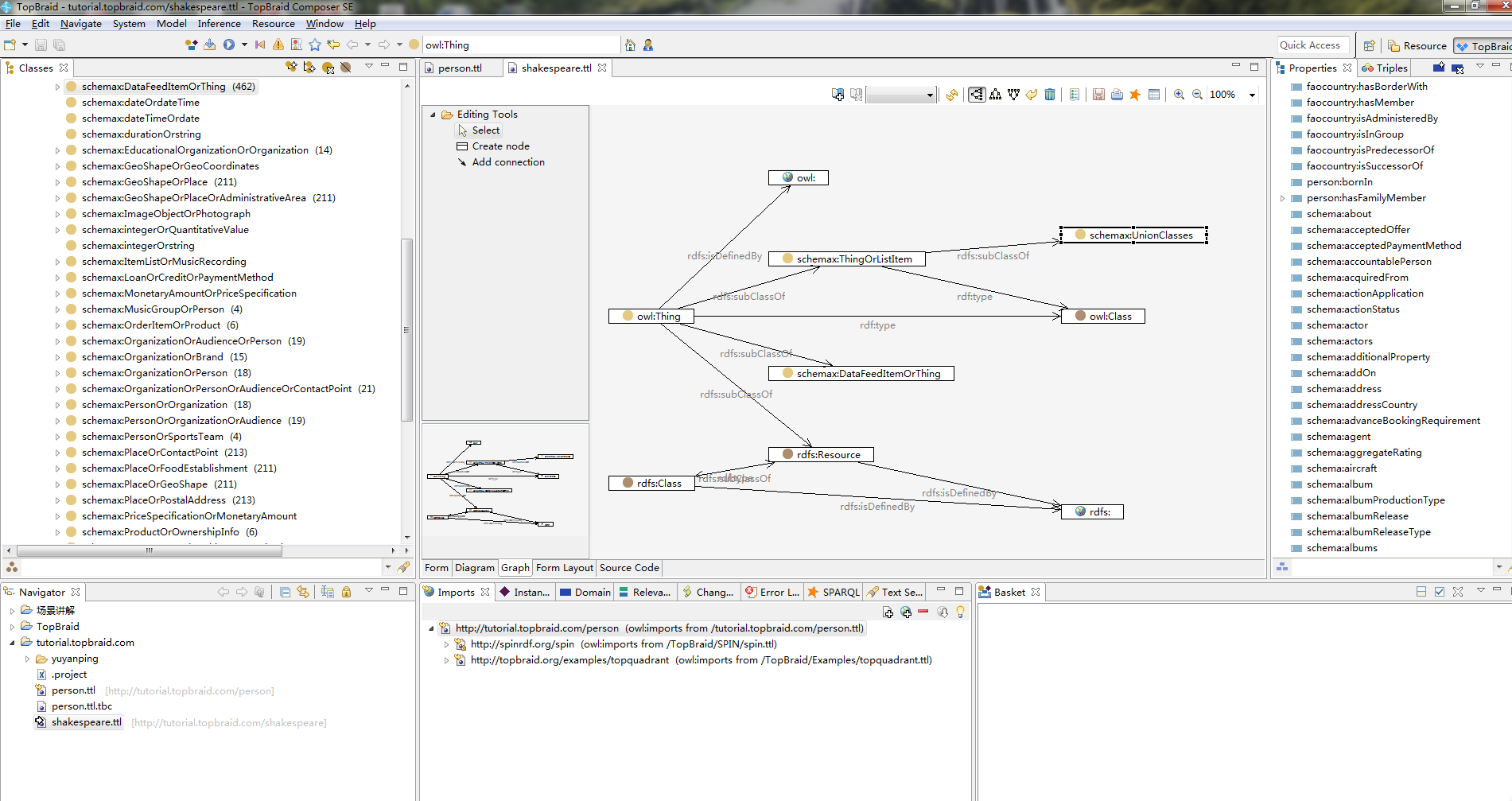
**TopBraid Composer使用指南**

Maestro Edition Version 5.2

重庆邮电大学工业物联网与网络化控制教育部重点实验室 网关组



2019

**作者：于彦平，修订者：文艺、高岐**

**2019/4/25**

**TopBraid Composer使用指南**

**1 引言**

本指南简要介绍TopBraid Composer(Maestro Edition)—一种开发语义Web本体和构建语义应用程序的企业级建模环境。本软件完全支持W3C标准，是领先的工业级RDF编辑器和OWL本体编辑器，同时也是市面上最好的SPARQL工具。

本文中，TopBraid ComposerTM、TBC和Composer为同义词。

Composer以Eclipse插件的形式实现，它可以用于开发本体模型、将数据和模型转换为RDF/XML或者将RDF/XML转换为其他格式数据和模型、集成数据源整合以及开发语义Web服务和应用。本文档仅包含Composer功能的一小部分。

第二部分讲解如何安装TBC。这一部分展示如何打开和浏览已有的本体、如何组织TBC workspace以及如何创建一个新的工程文件。

第三部分的重点集中于构建一个简单的本体（仅限于RDFS vocabulary），并执行测试查询。本部分还包括一个RDFS推理练习。

第四部分解释导入功能和方法来处理多本体操作。

第五部分描述如何创建关键的OWL构造，包括约束。

第六部分介绍SHACL（Shape Constraint Language，模型约束语言），并提供了一个SHACL独立教程的链接。

附录A提供关于TBC支持的标准：RDF、RDFS、OWL、SPARQL和SHACL的一些额外信息。对以上标准不熟悉的读者可以先看附录A，然后再转到第二部分进行阅读。

一旦您掌握了本文档包含的基本功能，我们建议您仔细查看Composer自带的Help菜单。许多文档不包含的功能在Help文档中有相关解释。可以通过选择**Help >Help Contents**菜单来访问Help，然后单击**TopBraid Composer**。你也可以在TopQuadrant官网上找到一些视频和样例。

***1.1 Conventions（约定）***

类(class)、属性(property)和个体名(individual name)以Times New Roman粗体字书写。

用户接口窗口小部件的名称和菜单选项以Arial粗体字体显示。

当需要往TBC中输入信息的时候，使用的是Times Courier New字体。

练习和教程步骤以如下方式呈现：

**Exercise N:Accomplish this**

1. Do this.

2. Then do this.

3. Now do this.

|  |
| --- |
| 使用TBC的提示和建议以及构建本体以此图标呈现。 |

|  |
| --- |
| 潜在的陷阱和警告以此图标呈现。 |

|  |
| --- |
| 一般性的注释以此图标呈现。 |

|  |
| --- |
| 高级功能以此图标呈现。我们推荐读者首次阅读本指南时跳过高级功能。 |

**2 安装和设置**

***2.1 系统要求***

TopBraid Composer作为一个Eclipse插件的形式实现。Eclipse是一个为各种建模和编程语言开发的功能强大的开源平台。将Composer集成到Eclipse中意味着你可以利用Eclipse这个集成开发环境的优势。Eclipse也提供了一个更新机制允许用户在有新版本可用时，便捷地更新诸如TopBraid Composer这样的插件。

系统要求和Eclipse 3.6平台的要求相同。

***2.2 安装***

TopBraid Composer其实是免安装的，要使用TopBraid Composer，首先要到<http://www.topquadrant.com/downloads/topbraid-composer-install>下载相应的软件包。软件包共分为Free Edition(TBC-FE)、Standard Edition(TBC-SE)和Maestro Edition（TBC-ME，大师版）三个版本，除了Free Edition外，其他两个版本都是收费的，但是会提供一个月的免费试用期。

本指南以Maestro Edition为例，其他版本同样适用。

下载软件包时，会让你选择软件运行的主机操作系统。下载好软件包之后，双击其中的TopBraid Composer.exe文件打开。

启动TopBraid Composer后会要求你选择workspace，这是Eclipse的一个术语，该workspace是用于后续存储一系列工程文件的文件夹。选择默认的workspace名（记录下该名称，后面的“Organize the Workspace”部分会用到），然后它会自动为你创建工作空间。

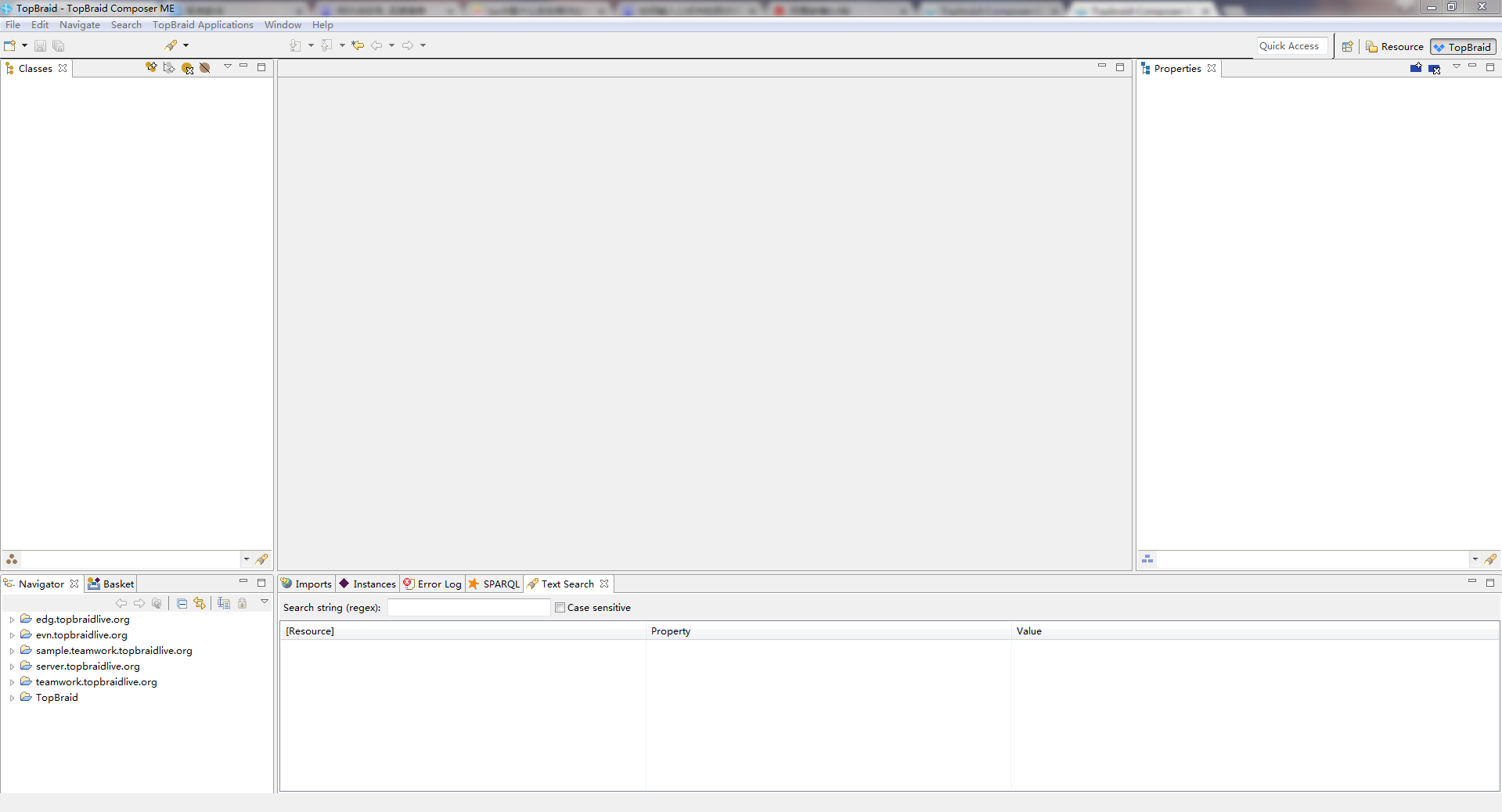


图2.1 TopBraid Composer的默认视图

***2.3 熟悉TBC***

**2.3.1 TBC视图**

TBC用户界面是一个“视图”的集合或者说是一个应用程序窗口的集合。每个视图都有一个标签，显示视图的名字和针对该视图的图标集合。以下是默认的视图：

* **Navigator（导航）视图：**该视图显示Eclipse工作空间中的工程和文件。

⭘ 你会看到随TBC-ME自动生成的至少5个工程文件。现在我们重点关注名为**TopBraid**的工程文件，该工程文件中包含了许多样例本体。

* **Classes、Properties和Association视图：**这三个视图展示了当前模型的层次性。
* **Resource Editor：**这个视图是TopBraid Composer的主工作区域。当你打开一个RDF/OWL图时，在导航视图、类视图和属性视图中可以看到当前选中的资源。

⭘ 该窗口包含**Form**、**Graph**、**Source Code**和**Browser**（只有大师版具有）4个基本标签。

⭘ 当选择的资源是一个类时，还会有一个**Diagram**标签。

⭘ 当选择的资源是一个本体时，会出现**Statistics（统计数据）**、**Profile（配置文件）**和**Overview**标签。

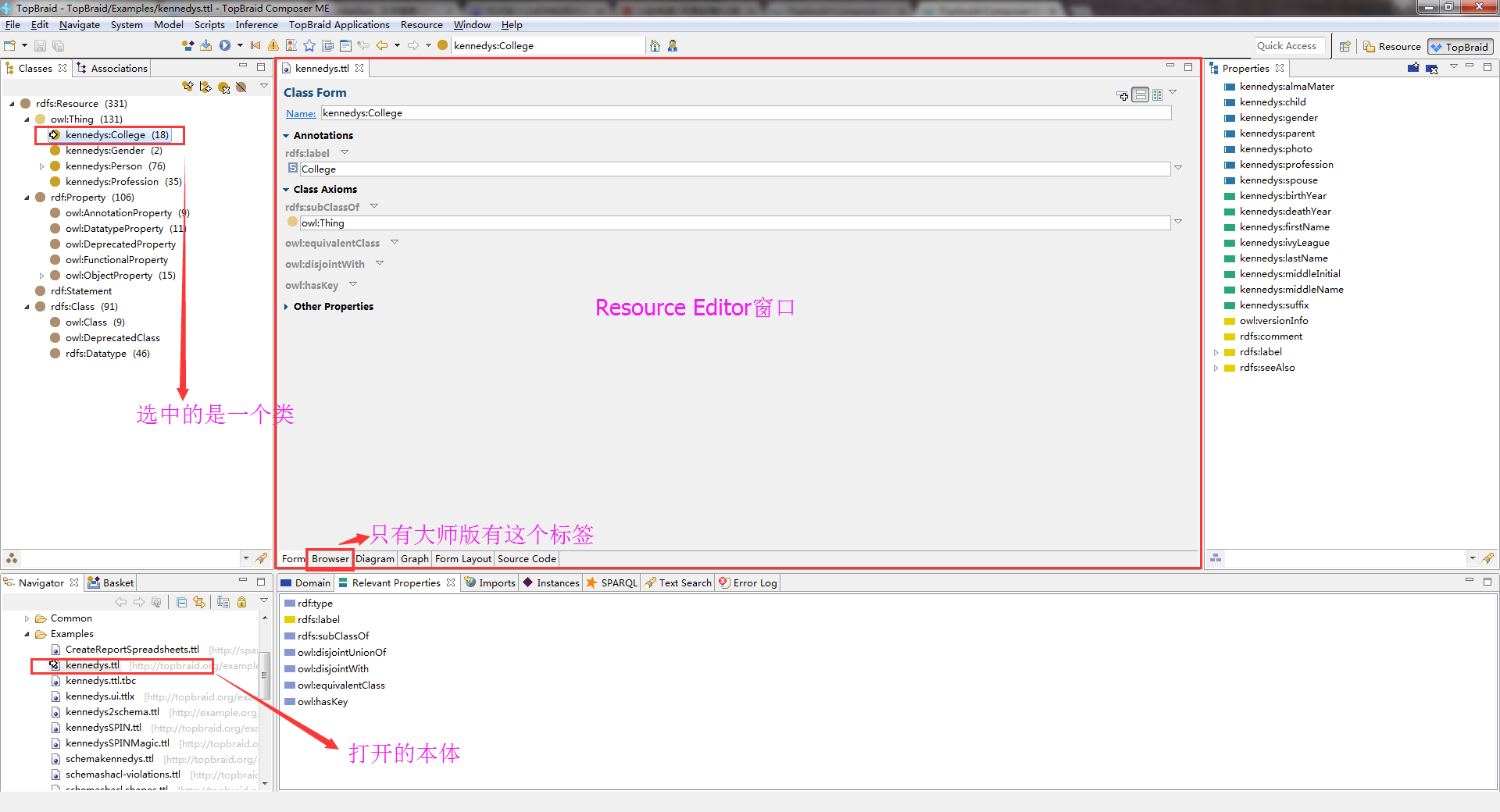


图2.2 当选中资源是一个类时的界面

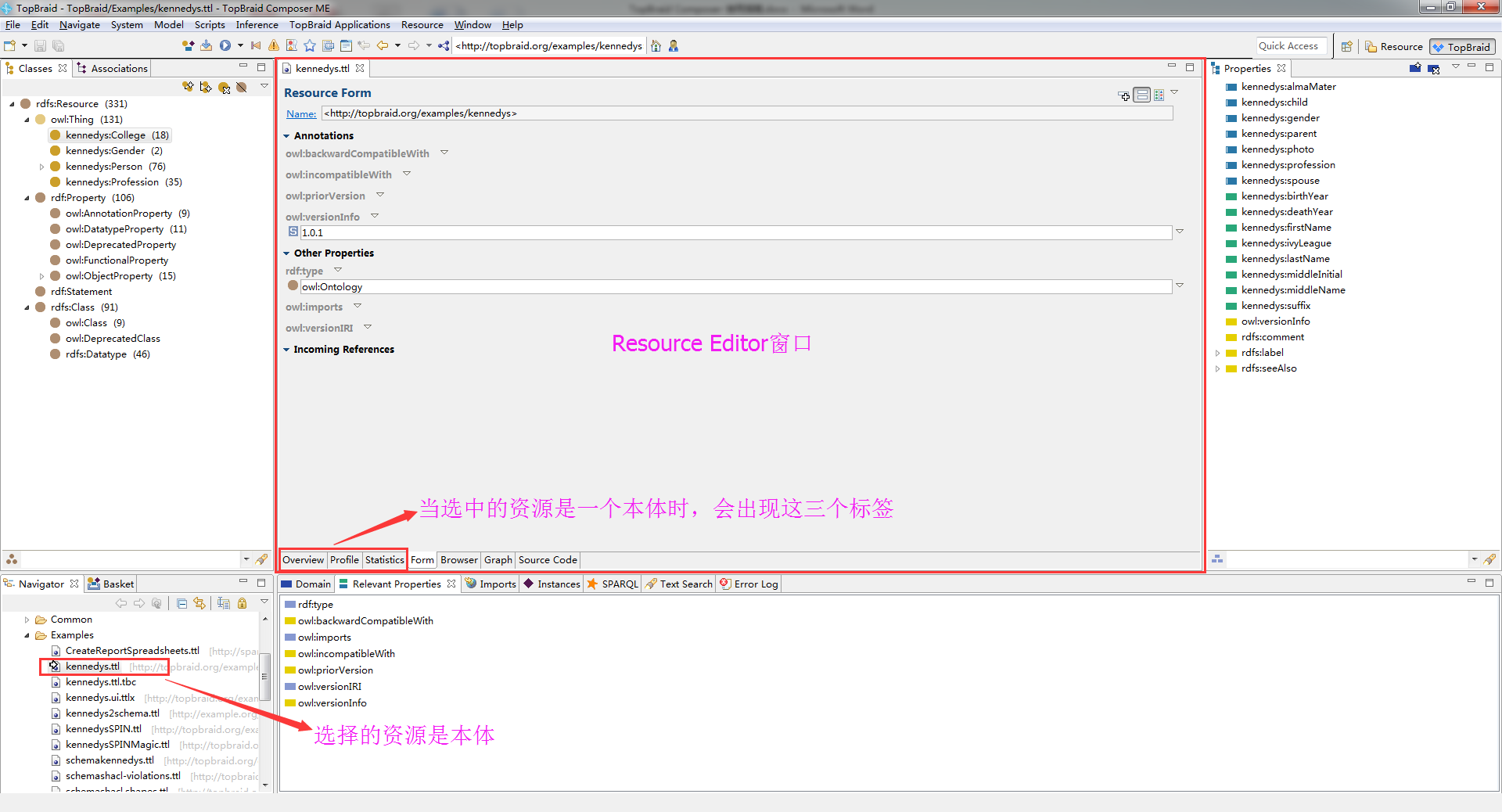


图2.3 当选中的资源是一个本体时的界面

* **Imports视图：**显示本体的导入（层次性地）。
* **Instances视图：**该视图显示了在**Classes**视图中选中的类包含的实例。
* **Domain视图：**显示当前选中的类领域中的所有属性（如果当前选中的资源不是一个类的话，则显示最近选择的一个类的领域中的所有属性）。
* **Relevant Properties（关联属性）视图：**显示当前选择的资源可能会用到的属性列表。
* **SPARQL视图：**提供一个创建、执行和管理SPARQL查询的界面。
* **Basket视图：**在多种用途下，Basket视图可以用作一个灵活的拖放区域，用于暂存拖拽进来的资源。
* **Text Search视图：**让你能够基于文本匹配找到相关资源。它跨一个资源的所有文本属性进行搜索。
* **Shapes、Target和SHACL Validation（模型约束语言验证）：**提供对新的W3C RDF Shapes(SHACL)标准的支持（其中有些视图已在TBC5.1版本中介绍，有些只在TBC5.2中可用或者进行了更新，TBC在不断跟进标准的进展）。

除了以上视图，还有许多其他可用视图。想要研究和选择其他视图，从顶端**Window > Show View**菜单进行选择。你将会看到大量的其他选项。所有的Composer视图都可以根据需要进行拖拽和重新组织，也可以改变视图的大小、最大化和最小化视图。

要想最大化一个视图，可以双击该视图名字所在标签区域。再次双击该标签区域将会还原之前的视图大小。想要移动一个视图，点击视图标签，将视图移动到你想要放置的位置。

每个视图都有许多菜单选项。这些菜单选项以图标的形式列于视图的标题栏，一些额外菜单选项可以通过点击视图标题栏向下的三角图标进行选择。

下面我们将要打开和研究随TBC自带文件中的其中一个。

**Exercise 1：Navigating in TBC**

1. 在**Navigator**视图中展开**TopBraid**工程文件，然后展开**Examples**文件夹，双击topquarant.ttl打开这个本体。现在，在**Resource Editor**视图中可以看到本体的相关信息，**Classes**和**Properties**视图会显示这个本体中的类和属性。

2. 在**Classes**视图中，展开**owl:Thing**，双击**schema:Country**类来选中它。关于**schema:Country**的信息将会显示在屏幕中间的**Resource Editor**视图中。

|  |
| --- |
| 或者，你可以使用以下方式来选择一个资源：  **a.** 单击**schema:Country**前面额图标（由于这是一个类，所以前面的图标为一个金黄色的圆圈）。  **b.** 在**Resource Editor**视图上面的工具栏文本框中输入**schema:Country**，按回车键将会选中这个资源。 |

3. **schema:Country**是被选中资源的qname或者说是“qualified name（限定名称）”，由于URI引用可能会很长，qname提供了一种URI的简化方式。qname由namespace prefix（命名空间前缀）、冒号和一个本地无前缀的资源名组成。单击**Resource Editor**视图中带下划线的**Name**标签，你将会看到这个资源的完整URI：<http://schema.org/Country>。

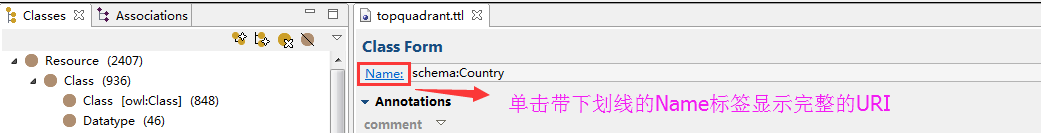


图2.4 查看资源完整的URI

4. 将鼠标悬停在**Resource Editor**视图中Form标签中的rdfs:subClassof窗口小部件内的**schema:AdministrativeArea**上，并且按住CTRL键不放。如下图所示这时候，**schema:AdministrativeArea**会变成一个超链接。点击这个超链接，跳转到一个表，显示**schema:AdministrativeArea**的信息。

|  |
| --- |
| 总是存在一个被选中的本体模型，和一个被选中的本体资源（class、property、individual等）。被选中的资源总是来自于被选中的本体模型。Resource Editor展示当前选中的资源，当然，在工具栏的文本框中也会显示。 |

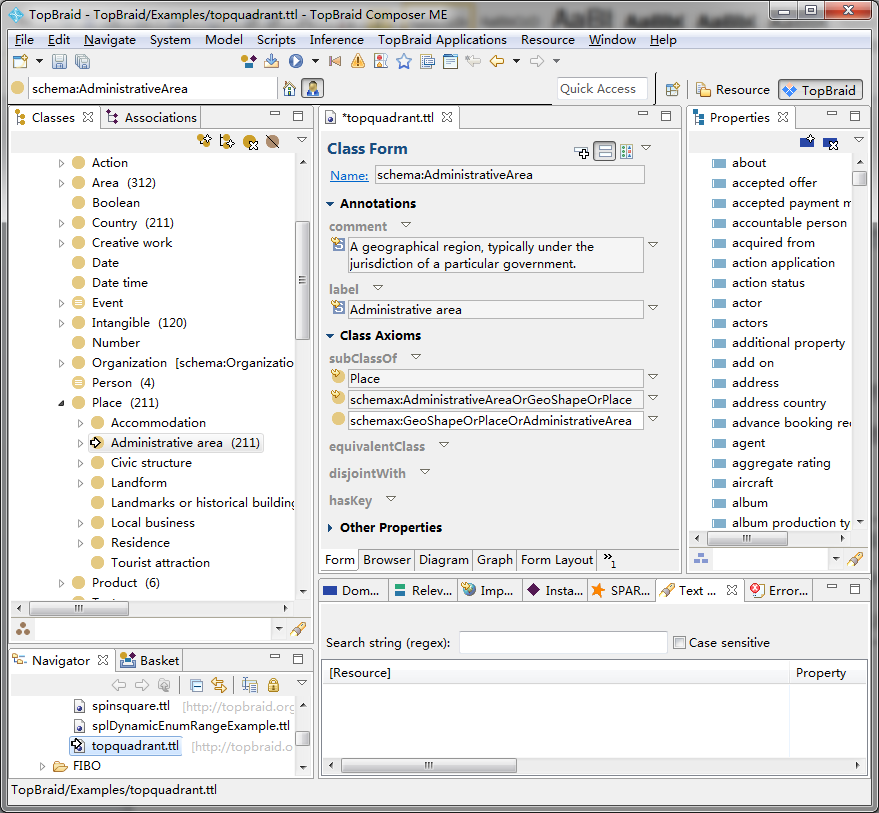


图2.5 使用超链接进行导航

5. 注意Composer工具栏上的后向箭头和前向箭头。他们能够让你在当前选中资源和先前选中资源之间进行快速导航，这点和web浏览器上的前进、后退类似。点击后向箭头，然后点击前向箭头，注意观察**Resource Editor**视图中显示资源的变化。

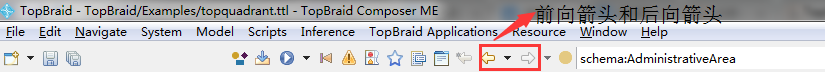


图2.6 前向箭头和后向箭头进行导航

6. 选中**schema:Country**类，点击屏幕底端的**Instances**视图，在实例视图中将会显示一系列的国家实例。

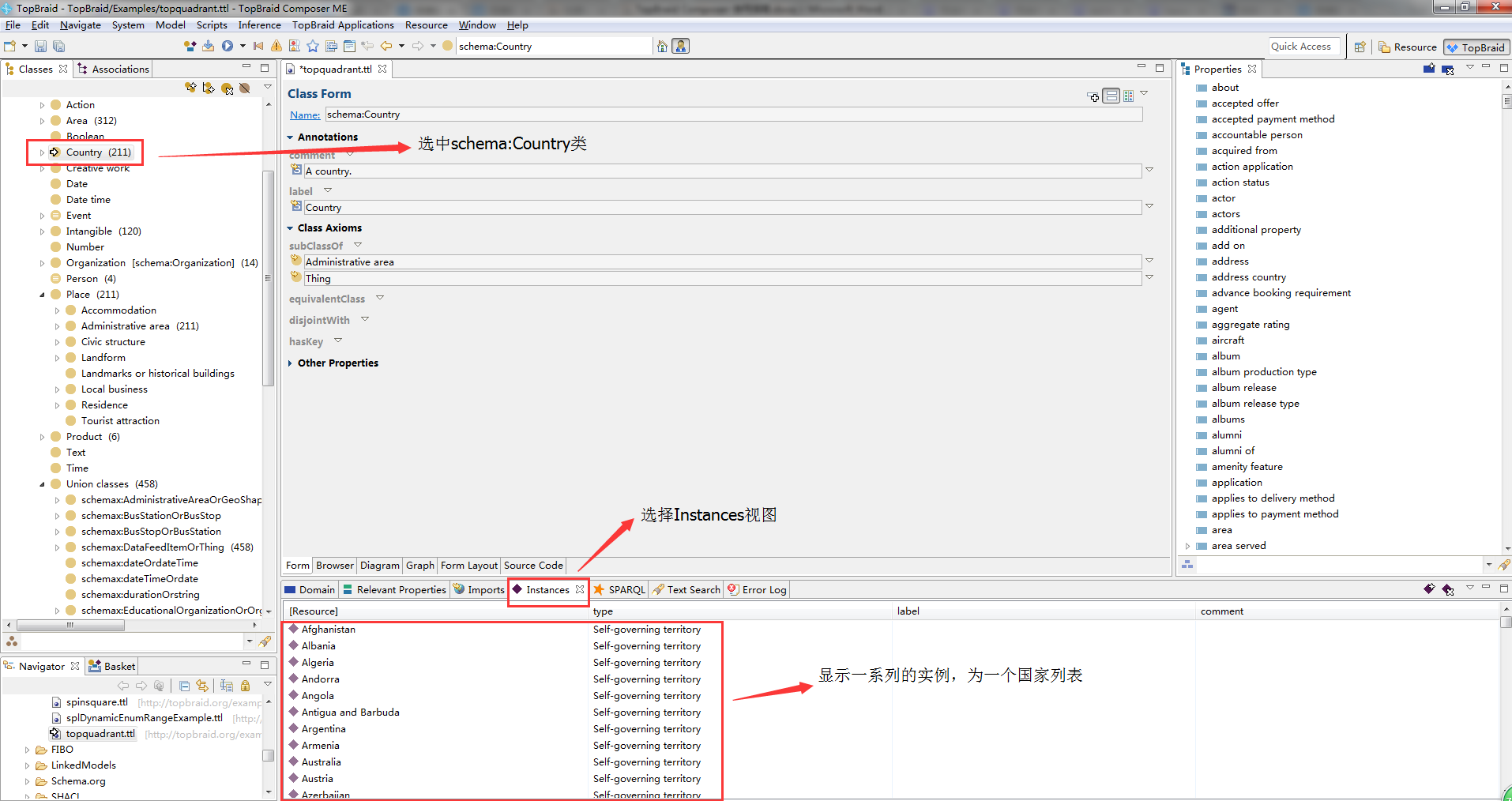


图2.7 查看类的实例

7. 点击TopBraid Composer菜单区域内的**Toggle between qnames (with prefixes) and human readable rdfs:labels**图标，切换为基于标签的显示模式。这种模式下，以更加可读的方式显示资源的rdf:label值，而不是显示qname。再次点击图标，切换到基于qname的显示模式。

**Exercise 2：Switching between ontologies**

1. 到**Navigator**视图中打开Kennedys本体(**kennedys.ttl**)。现在你的界面将会显示如图2.8所示。

|  |
| --- |
| TopBraid Composer可以打开各种不同格式的文件。包括所有常见的序列化 RDF：RDF/XML（这些文件一般具有.rdf扩展名）、Turtle（具有.ttl扩展名）和N-triples（具有.nt扩展名）。注意TBC不能打开.owl扩展名的文件。如果你有一个具有.owl扩展名的文件，请重命名为以上扩展名的文件。  使用TopBraid的Semantic XML技术和Semantic Tables技术，通过动态转化为RDF文件的方式，TopBraid Composer也可以打开XML文件和Excel文件（具有.xls或者.xlsx扩展名）。你可以尝试一下这个功能，将一个电子表格复制到你的workspace中，双击进行打开。TopBraid可以打开数据库连接文件，这些文件用于存储配置连接到RDF数据库或者是关系数据库的设置信息。介绍如何将TBC与数据库之间的协作已经超出了本指南的范围，要想了解更多信息，请参阅Help菜单。 |

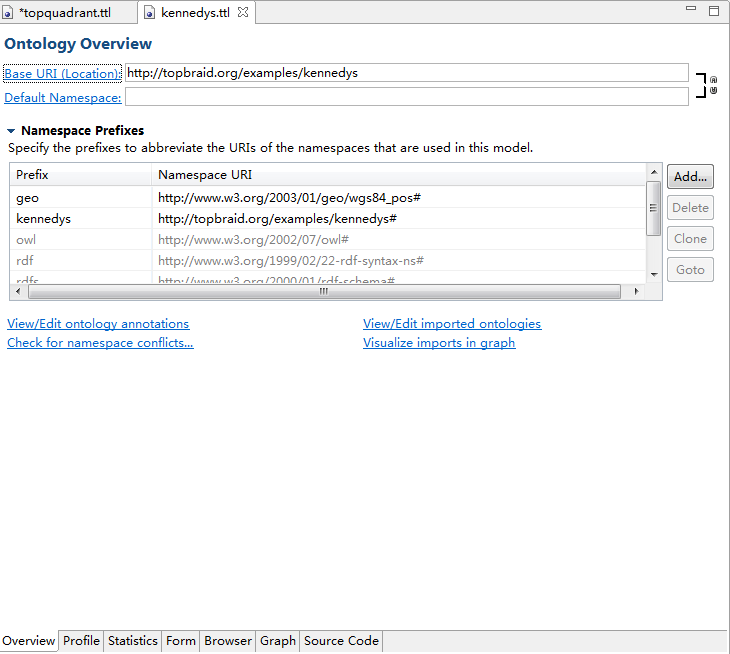


图2.8 打开多个本体

2. 注意**kennedys.ttl**是你当前选中的本体，但是**topquadrant.ttl**依然被打开。点顶端标签上的**topquadrant.ttl**就能切换回这个本体。

3. 关闭**topquadrant.ttl**。可以通过点击标签上的“×”，或者通过**File**菜单中的**Close**选项进行关闭。

3. 关闭**kennedys.ttl**。方法同上。当然你也可以通过**Flie**菜单的**Close All**选项关闭所有的本体。

**2.3.2 创建一个Eclipse工程**

在下面的练习中，你将学习创建自己的工程文件。

|  |
| --- |
| 工程是Eclipse工作空间中组织信息的一种方式，在工作空间中，信息可以 被进一步地组织成层次性的文件夹。 |

**Exercise 3：Creating a new project**

1. 在菜单中选择**File > New > Project…**

2. 在弹出的窗口中展开**General**文件夹。选择**Project**并单击**Next**键，然后键入tutorial.topbriad.com作为工程名（可随意指定），点击**Finish**。

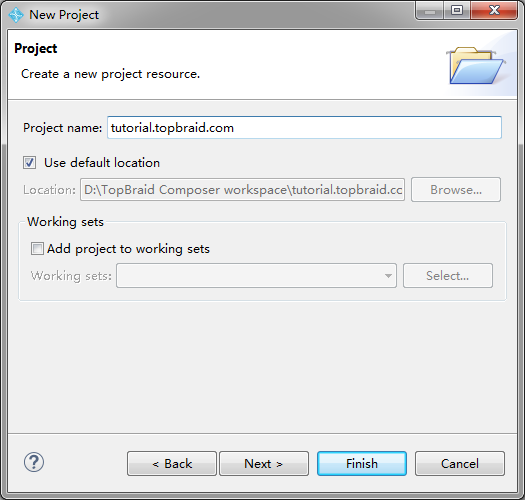
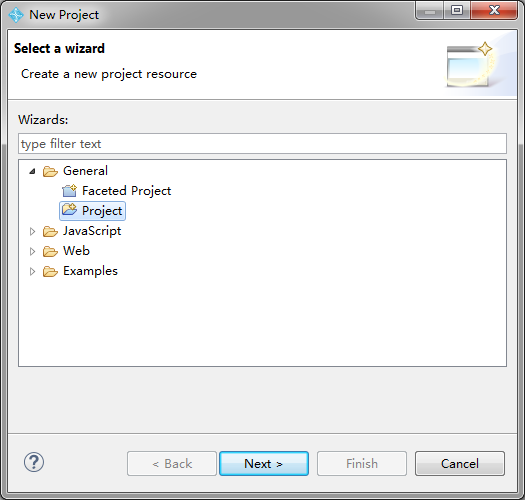


图2.9 创建一个工程

现在你可以：

* 在工程中创建文件夹；
* 将已有的文件复制到工程中；
* 创建一个新的文件。

**2.3.3 组织工作空间**

**Exercise 4：Create folders in a project**

1. 在**Navigator**视图中，右键单击你创建的工程文件，选择**New > Folder**。

2. 在弹出的窗口中对新建的文件夹命名，然后单击**Finish**。

3. 在**Navigator**视图中你将看到刚才创建的新文件夹。

4. 找到你打开TopBraid Composer的时候选择的workspace所在的路径，进入到workspace文件夹，你将会发现在这个文件夹中已经存在刚才创建的文件夹。

**2.3.4 复制文件到一个工作空间**

**Exercise 5：Copy file to a workspace and open it**

1. 找到一个你想要复制到工作空间的文件，可以是RDF/XML文件，也可以是能够转换为RDF的文件，如.xls文件或者是.xlsx文件。右键单击这个文件，选择“复制”。

2. 在**Navigator**视图中，找到你刚才创建的文件夹，右键单击这个文件夹，选择“**Paste**”。

3. 现在，这个文件已经被复制到你新建的文件夹中，你可以双击这个文件打开它。

|  |
| --- |
| 一个工作空间对应于你电脑文件系统中的一个文件夹。你可以通过电脑的文件管理系统（如Windows Explorer）将文件复制到工作空间中任意文件夹中。但是，在你刷新**Navigator**视图中相应的文件夹之前，在**Navigator**视图中你看不到复制进去的文件。  右键单击**Navigator**视图中的文件夹，选择**Refresh**更新文件夹。如果你用其他方式（例如你在Eclipse环境之外使用文本编辑器重新编辑了文件）对文件进行了更新，在刷新文件夹之前，TBC不知道你已经对文件进行了更新。  你也可以用同样的方式将工程复制到工作空间中。例如，TopQuarant提供了一些样例工程文件供你下载。当你复制一个新工程到一个工作空间时，需要刷新，TBC才能知道工作空间进行了更新。如果刷新工作空间之后你还是没有见到你新复制进去的工程，那么你可以通过以下方式之一解决：   * 在**Navigator**视图中，右键，选择**Import… > General > Existing Projects into Workspace**，然后浏览选择你要导入的工程，或 * 创建一个新的工程，给它一个和你已经复制过来的工程相同的名字，或 * 重启TBC。 |

**2.3.5 偏好设置**

|  |
| --- |
| TBC是一个可高度配置的软件。各视图中显示的内容都可以根据用户喜好进行配置。偏好设置可以通过**Classes**和**Properties**视图中向下的三角图标进行选择。在**Preferences**对话框中，展开**TopBraid Composer**，查看可用的偏好设置对话框。**Classes View**的偏好对话框如下图所示。 |

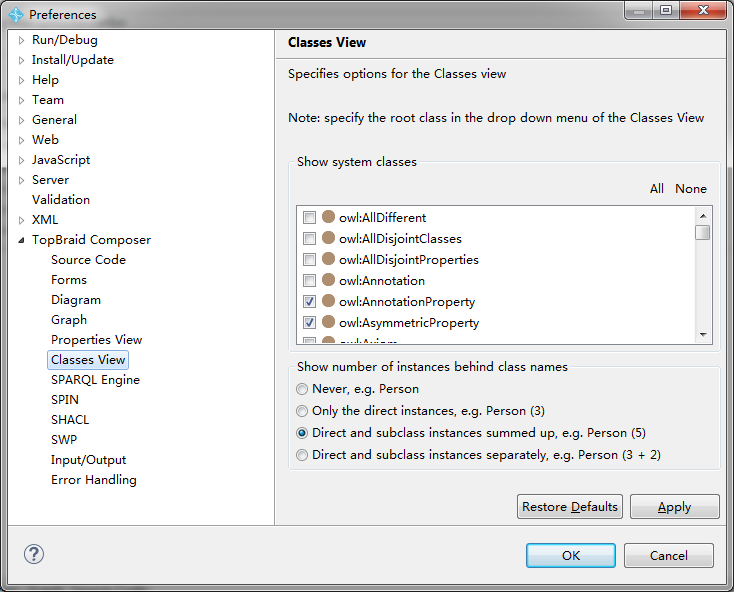


图2.10 类的偏好设置对话框

**2.3.6 使用Help菜单**

本指南仅涵盖了Composer的一小部分功能。如果你想了解本部分功能的详细信息或者想要学习TBC的其他功能，你应该参考**Help**功能菜单。

Help功能可以通过TopBraid Composer下的**Help > Help Contents**菜单进行访问。展开左侧的TopBraid Composer目录，你会看到许多可用的帮助页面。注意，在Help窗口的左上角有个搜索框，Help内容是可以检索的。

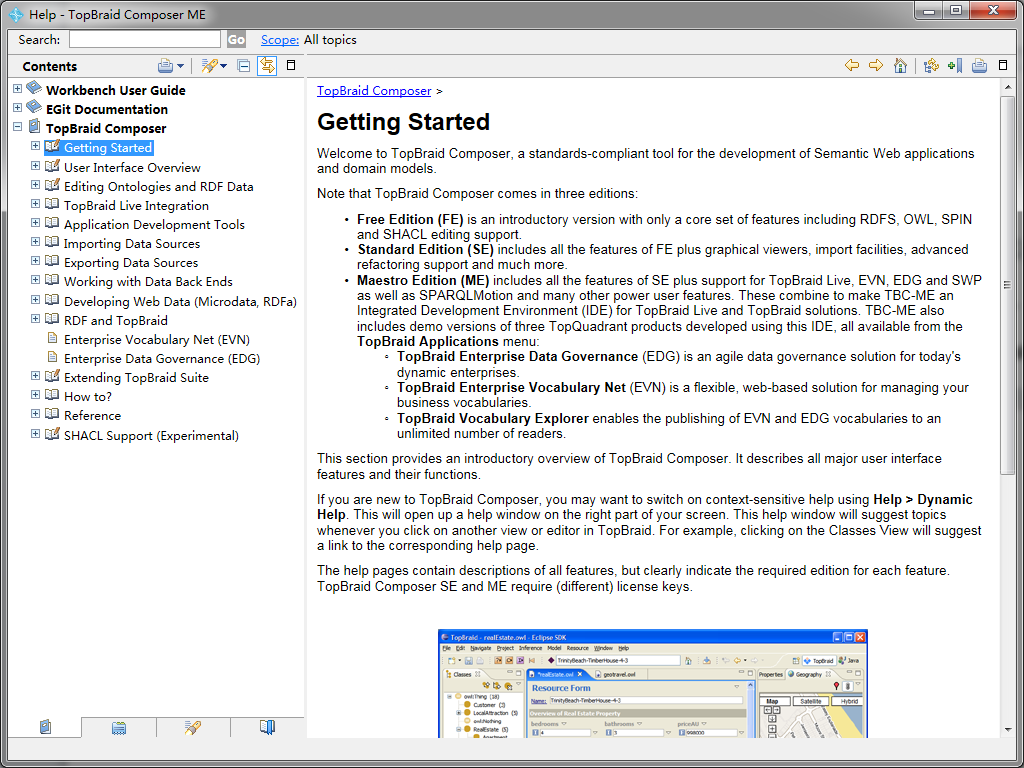


图2.11 Help功能

**3 用TopBraid Composer构建第一个本体**

本章描述如何创建一个关于人和家庭关系的简单本体。

**Exercise 6：Create a new file**

1. 在**Navigator**视图中，右键单击你新建的tutorial.topbraid.com工程文件，选择**New > RDF File**。

2. 在Base URI中将unnamed用person替换掉。这时候你注意观察，在File name文本框中自动生成名为person的文件名。默认的文件扩展名为.ttl（turtle文件）；你也可以选择其他RDF序列化扩展名。

3. 点击**Finish**。

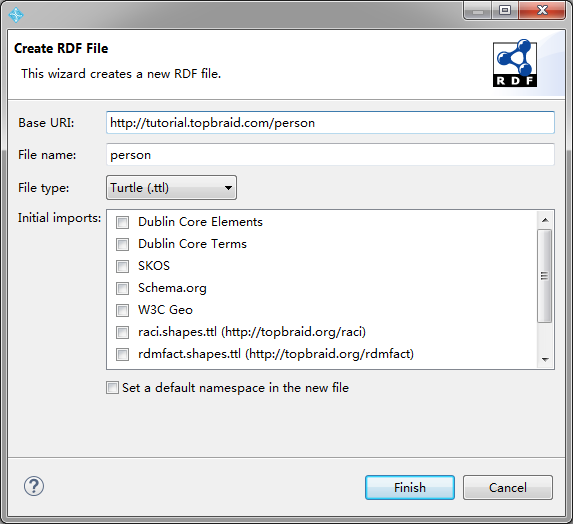


图3.1 创建RDF文件的对话框

***3.1 创建类***

当一个新的RDF文件被创建后，界面显示如图2.13所示。根据你的设置，**rdfs:Resource**可能是**Classes**视图中的根节点，但是你可以通过点击**Classes**视图中向下的三角图标并选择下拉菜单中的**Start hierarchy with owl:Thing**选项切换为以**owl:Thing**作为根节点的模式。我们推荐在这练习中使用这种模式，这样**Classes**视图就不会显得很混乱；当你不想使用OWL类时，使用**rdfs:Resource**作为根节点很重要。

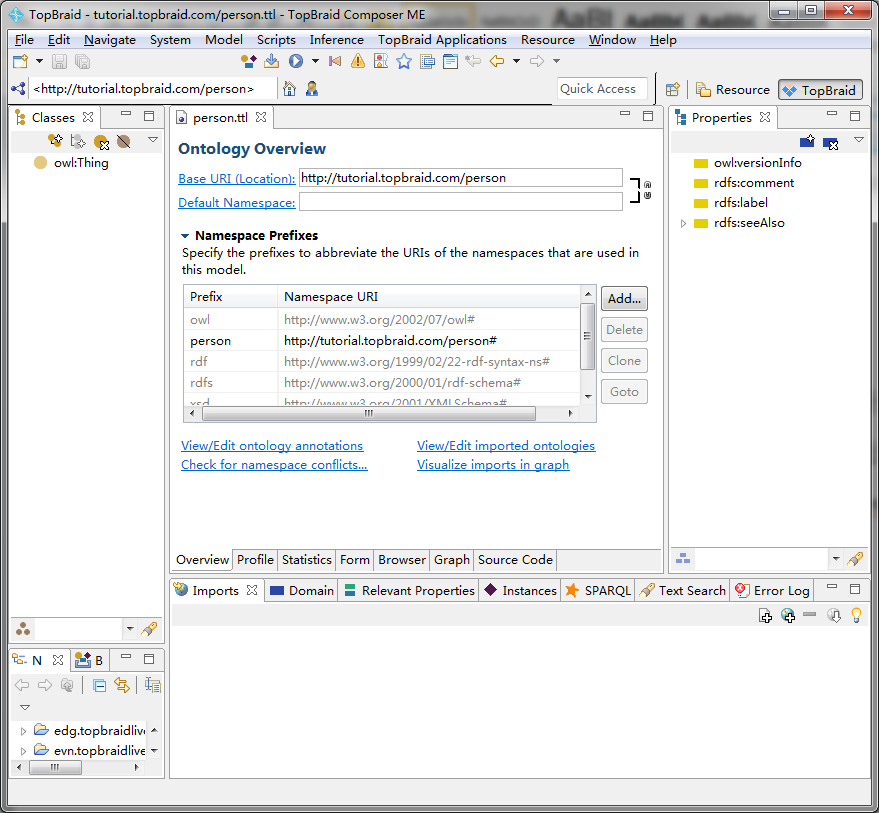


图3.2 创建person本体并设置**Start hierarchy with owl:Thing**模式后的初始界面

OWL类被解释为individual resources的集合。类**owl:Thing**表示包含全部individual的集合。因此，所有的类都是**owl:Thing**的子类。

让我们往本体中加入一些类。

**Exercise 7：Create classes Person，FemalePerson and MalePerson**

1. 在**owl:Thing**上右键单击。

2. 选择图2.14菜单中的**Create subclass**选项。这一步骤将会为当前选中的类创建一个新的子类（在本示例中，为类**owl:Thing**创建了一个子类）。你也可以通过**Classes**视图标题栏的图标创建一个子类。

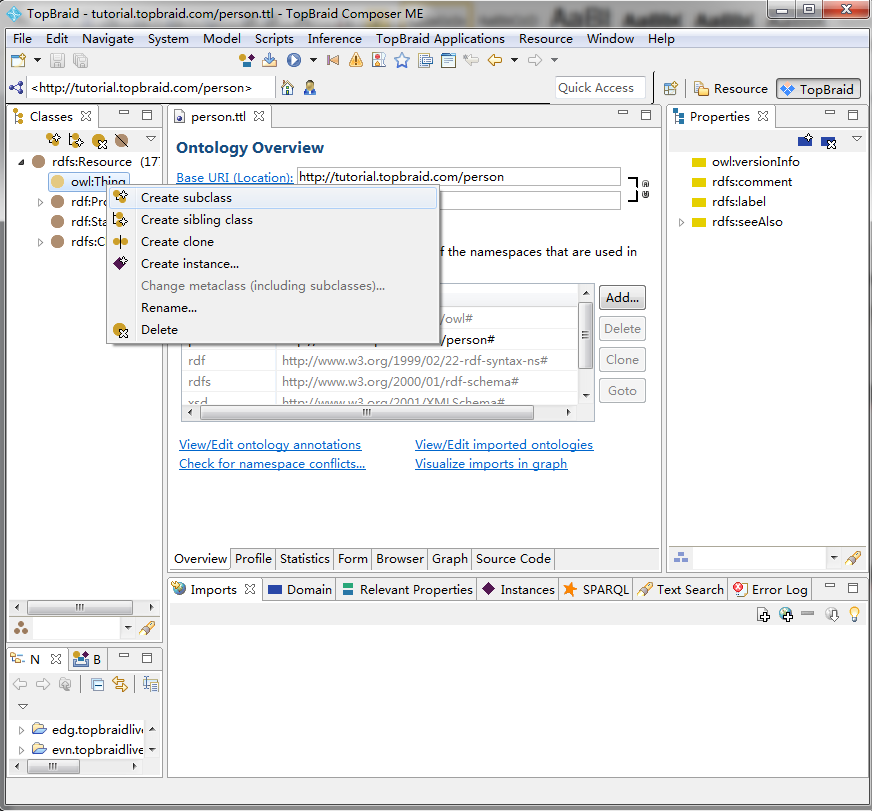


图3.3 **Classes**视图—菜单选项

3. **Create classes**对话框中默认的名称是**Thing\_1**，将其重命名为person:Person。

在对话框中的**Annotations Template**（注释模板）中选中**rdfs:label**，并在**Initial Value**文本域中输入{name}。这会告诉TBC从类名中生成一个人类可读的标签。点击**OK**。（当类名是一个复杂、多词汇的类名时，这将非常有用）。至此，新建的类将会在**Form**视图中显示。

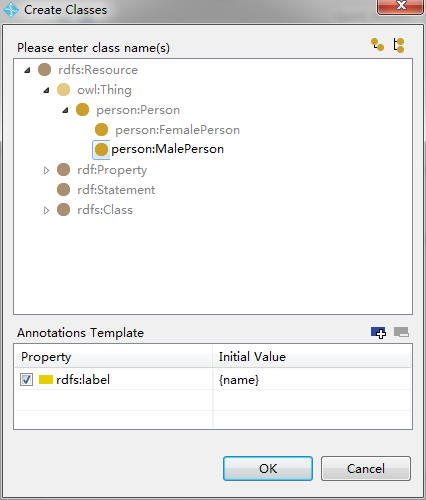


图3.4 创建一个类

4. 每个资源的资源编辑器中已经具有为该资源定义的所有字段（属性）；在本示例中，它展示了你用**Create Classes**对话框设置的rdfs:label和rdfs:subClassOf属性。如果资源编辑器中不存在某个字段，那么可以很容易地进行添加。例如，从**Properties**视图中将**rdfs:comment**拖拽到**Class Form**中**Annotation**区域。

5. 在rdfs:comment文本框中输入‘Human being’，点击文本框右侧的图标，或者在键盘上敲击Enter键。文本框周围的黑框将会消失，这时候添加的Human being被确定。

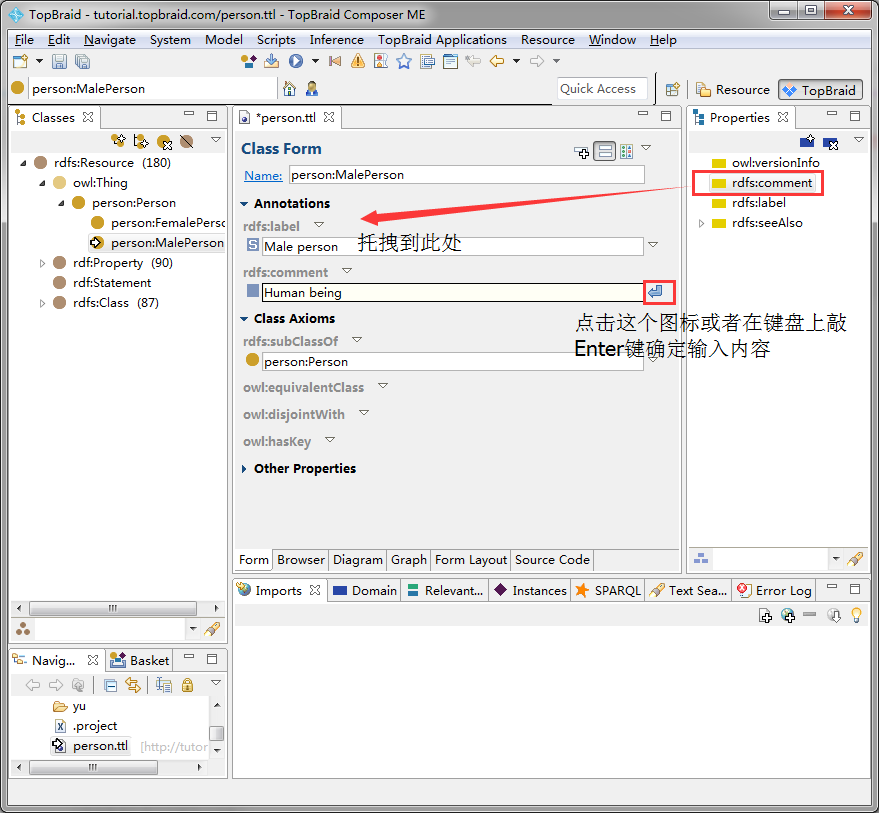


图3.5 将属性拖拽到Class Form中

6. 重复步骤2和步骤3添加类person:FemalePerson和类person:MalePerson，确保每次点击**Create subclass**之前选中的是Person这个类，这样创建的类才是Person的子类。（如果你不小心将person:MalePerson创建为person:FemalePerson的一个子类，你可以将Person:MalePerson拖拽到Person节点上，这样可以就把Person:MalePerson变为Person的子类。这种用拖拽的方式编辑类的层次性，使得TBC对本体的编辑更加方便和灵活）。

7. 在**Classes**视图中双击person:MalePerson可以查看它的**Class Form**视图。将你的鼠标悬停在资源编辑器中的rdfs:subClassOf属性值左侧的，这时候图标上会出现一个“+”，点击这个“+”，会显示类person:MalePerson的**Class Form**中嵌套的Person类的**Class Form**。

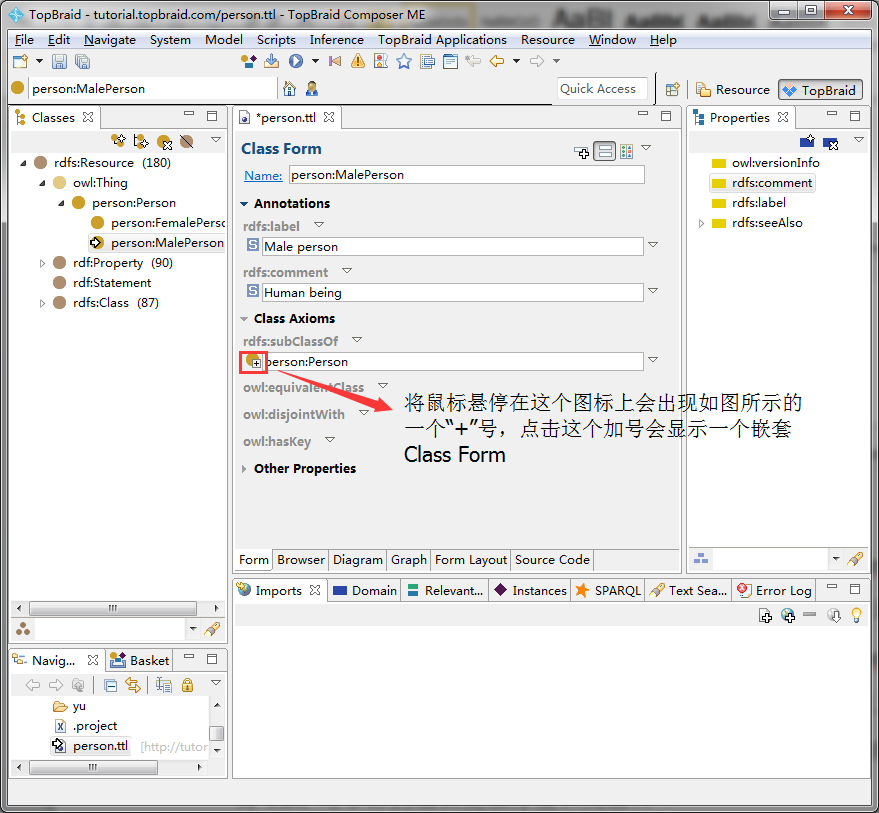


图3.6 嵌套**Class Form**

类的层次性和person:MalePerson的**Class Form**视图应该如图3.7所示。

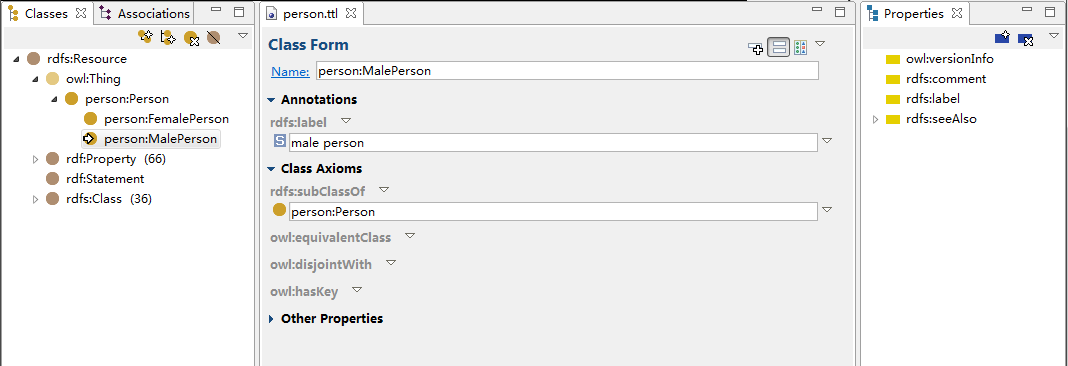


图3.7 Person本体初始类的层次性

8. 注意，在**Class Form**上面本体名之前有一个星号。这说明本体被修改过，尚未保存。选择**File > Save**或者点击左上角的保存图标或者用组合键Ctrl+S保存修改，这时候星号消失。

|  |
| --- |
| 在Composer中，**Classes**视图中类的名字前面有一个金色圆形图标，当前选中的类（也就是**Class Form**中当前显示的类），在其金色图标上有一个横向的前头表示当前选中。 |

**Classes**视图有许多图标选项，如下图所示。将鼠标悬停在这些图标上会出现一个工具提示解释这个工具的作用；这些功能的大多数也可以从右键单击类名出现的快捷菜单中进行选择。



图3.8 Classes视图中的图形选项

**Resource**视图也有许多图标和选项；大多数图标和选项在下图中进行了解释。在**Resource**视图的底部你也将看到下图上没有的**Browser**标签。这个标签能够让你看到关于一个资源的信息的不同HTML表示方式。



图3.9 TBC **Resource**视图（当选中**Form**标签时）可用的图标和选项

***3.2 创建属性***

属性表示RDF资源和另一个资源或者字面量(literal)之间的连接(connection)。例如，类**MalePerson**和**Person**资源之间有一个rdfs:subClassOf连接（属性），与“Male Person”字符串之间有一个rdfs:label连接（属性）。

OWL中有两种主要的属性—Object properties（对象属性）和Datatype properties（数据类型属性）。对象属性链接一个资源和另一个资源。数据类型属性链接一个资源和一个字面量(literal)。字面量是XML Schema数据类型值，例如字符串和整型数字。

OWL还有第三种属性类型—Annotation（标注属性）。Annotation属性用来存储被推理工具忽略的信息，包括label（标签）、comment（说明）和note（注释）。Exercise 5中我们使用标注属性rdfs:comment为Person类添加一个comment。

所有类型的OWL属性都是rdf:Property的子类。

在Composer的**Properties**视图中，属性在其名字之前有一个矩形图标。其中对象属性用蓝色的矩形图标表示，数据类型属性用绿色矩形图标表示，标注属性用黄色矩形图标表示。RDF属性具有深蓝色图标。Form视图中当前显示的属性（当前选中的属性），在其矩形图标上有一个横向的箭头。

属性视图也有一些功能图标，在图3.10中对各个图标进行解释。



图3.10 **Properties**视图功能图标

属性可以通过上图中添加一个新属性功能图标创建。在**Properties**视图中属性名字上右键单击，会出现一个快捷菜单，你可以选择**Create subproperty…**来创建一个属性的子属性，这点和**Classes**视图类似。注意，用这种方式创建的属性是当前选中属性的一个子属性。

**Exercise 8：Create datatype properties called firstName and lastName**

1. 点击添加一个新属性图标。创建属性的对话框显示如图3.11所示。

2. 选择**owl:DatatypeProperty**。重命名新属性为person:firstName。在Range中选择**xsd:string**，表明firstName值总是这种类型的。点击**OK**。

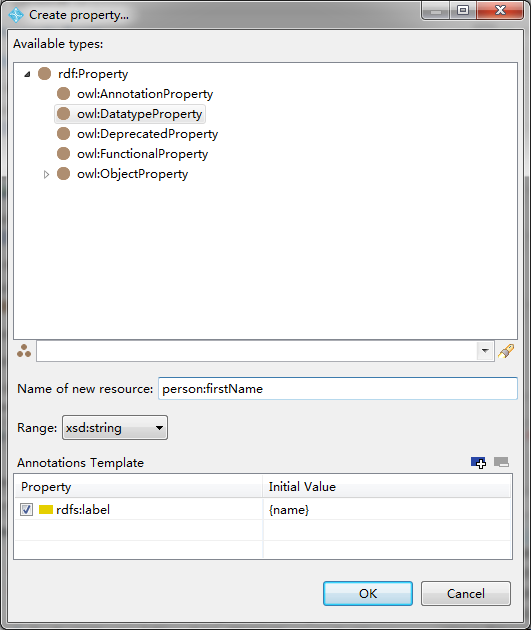


图3.11 创建一个新属性

3. 将类**Person**添加到新建的person:firstName属性的定义域(domain)中，表明一个具有firstName属性的资源是类**Person**的一个成员。这个操作可以通过以下方式中的一种实现：

a. 选中person:firstName属性状态下，将**Classes**视图中的**Person**拖拽到**Form**视图中**Property Axioms**下方的rdfs:domain处。

b. 点击rdfs:domain右侧的倒三角下拉菜单，选择**Add empty row**，在出现的文本框中输入：person:Person，点击**OK**。

c. 点击rdfs:domain右侧的倒三角下拉菜单，选择**Add existing…**，从对话框中选择类**Person**，点击OK。

4. 你的屏幕将显示如下图所示。

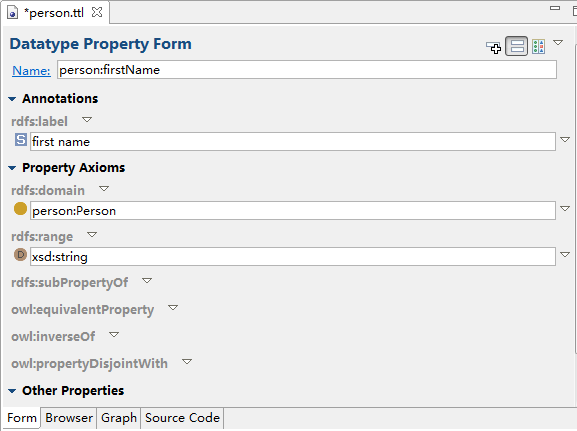


图3.12 firstName属性定义

5. 重复以上操作，创建一个person:lastName属性。

6. 在**Classes**视图中选中类**Person**。

7. 点击**Domain**视图。你可以在这个视图中看到刚才创建的两个属性。

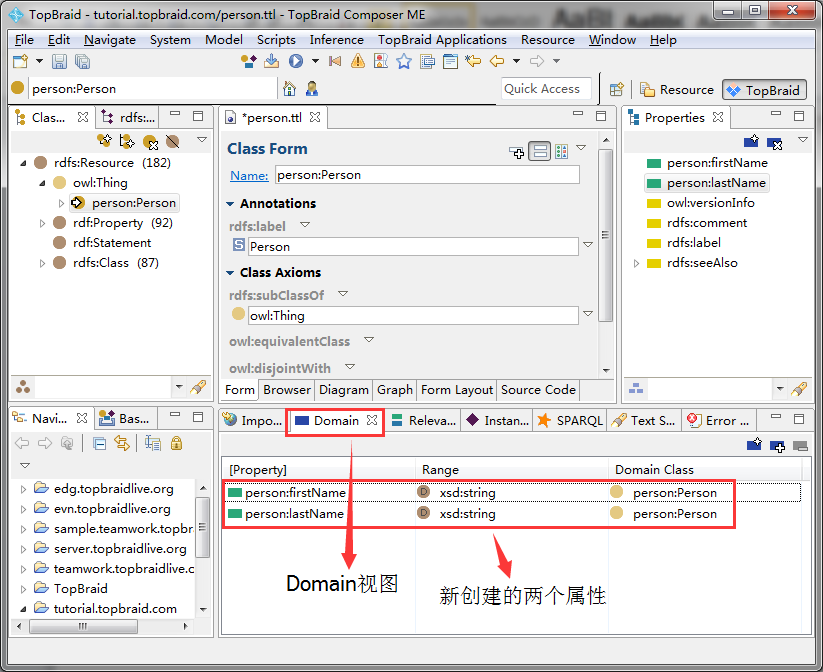


图3.13 类**Person**的**Domain**视图

**Exercise 9 Create object properties called hasDaughter，hasSon and hasChild**

1. 点击添加一个新属性图标，这时候弹出创建属性对话框。

2. 选择**owl:ObjectProperty**，因为这个属性将会有非字符串值的其他对象值。将新属性命名为person:hasDaughter，点击**OK**。

3. 将person:hasDaughter属性的定义域(domain)设置成类**Person**，将值域(range)设置成类**FemalePerson**。

4. 添加另一个对象属性person:hasSon。

5. 将person:hasSon属性的定义域(domain)设置成类**Person**，把值域(range)设置成类**MalePerson**。

6. 添加第三个对象属性person:hasChild。

7. 将**hasChild**属性设为**hasDaughter**和**hasSon**的父属性。可以用以下方法实现这个操作：

a. 选中**hasDaughter**，将**hasChild**拖拽到**rdfs:subPropertyOf**区域。

b. 在**Properties**视图中，将**hasSon**拖拽到**hasChild**上。

|  |
| --- |
| 定义域和值域的声明是全局的。他们应用于任何使用属性的地方。Semantic Web（语义Web）标准RDFS和OWL定义了产生于已有声明的推理和新声明（三元组）。  根据我们刚才下的定义，当应用RDFS推理时，任何资源，如果它是RDF三元组中的一个主语，并且这个RDF三元组有一个谓语**hasDaughter**，那么这个资源会被推理成类**Person**的一个成员。任何资源，如果它是这样一个RDF三元组的宾语，那么它将会被推理为类**FemalePerson**的一个成员。因为我们将**hasDaughter**设为**hasChild**的一个子属性，那么还会推理出主语相对于宾语有一个**hasChild**值。 |

|  |
| --- |
| 和其他数据模型（如关系数据库或者面向对象建模）不同，属性是独立于类的，并且可以在不同语境中被重用。这可以通过在模型中将一个属性连接指定到一个或者多个类来实现。而且，在TBC中，任何时候都可以拖拽一个属性字段到任意表中，即使没有模型定义说明这个属性可以被使用。  可以为一个属性指定多个类作为其定义域和值域。例如，可以拖拽多个类到rdfs:domain文本框中，如下图所示。    图3.14 多个类作为rdfs:domain  多定义域声明被解释为交集(intersection)。如果一个属性的定义域中有两个类**MalePerson**和**FemalePerson**，在这个属性定义域中的任何实例将会被假定为同是这两种类型。  如果你想表达一个定义域是两个类的并集(union)（例如一个实例可以是一个**MalePerson**或者是一个**FemalePerson**），那么你应该将**MalePerson**和**FemalePerson**放到同一文本框，并在两者之间加上一个‘or’，如下图所示。    图3.15 定义域为一个并集  尝试两种定义方式，并点击**Source Code**标签查看两种定义方式下OWL语法表示的不同，如图3.16所示。    图3.16定义域为一个交集和一个并集时的Source Code  谨记虽然你对你的本体具有完全控制权，并且能够保证多定义域和值域的并集，使用相同的属性但为它们指定不同的定义域和值域将会导致定义域和值域的交集。而这不是我们想要或者期望的结果。因此，我们建议定义域和值域应该被谨慎使用。后续要讲的OWL约束是一个连接类和属性的替代方法。 |

***3.2 创建实例***

现在我们将要向本体中添加一个实例。

**Exercise 10：Create instances of the Person class:Susanna\_Shakespeare，Judith\_Shakespeare，Hamnet\_Shakespeare and William\_Shakespeare**

1. 选中类**FemalePerson**。

2. 点击**Instances**视图。点击添加一个新实例图标，**Create FemalePerson**对话框出现后，将新实例的默认名字改为person:Susanna\_Shakespeare，点击**OK**。**Instances**视图的功能图标解释如图3.17所示。

3. 重复以上步骤，创建person:Judith\_Shakespeare实例。

4. 选中类**MalePerson**，添加实例person:Hamnet\_Shakespeare实例和person:William\_Shakespeare实例。至此，你的界面应如图3.18所示。

图3.17 **Instances**视图的功能图标

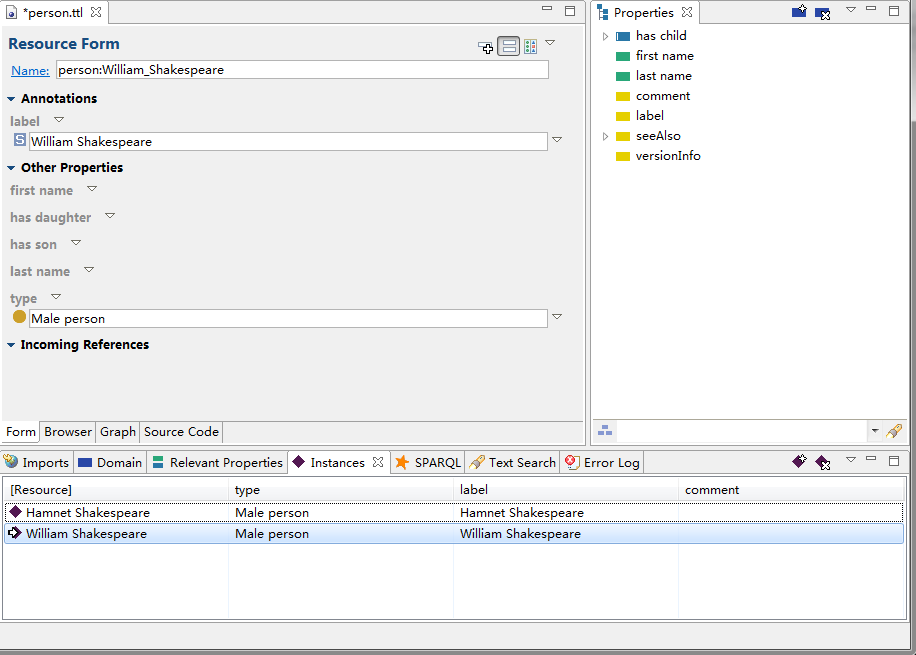


图 3.18 实例William\_Shakespeare的资源表

5. 将**Properties**视图中的**hasSon**和**hasDaughter**拖拽到**Instances**视图中的William\_Shakespeare上。

6. 将**Instances**视图中的Hamnet\_Shakespeare拖拽到William\_Shakespeare的**Resource Form**中的hasSon小部件处。

7. 声明Judith和Susanna是William的女儿。除了之前使用的拖拽的方式之外，这个声明也可以使用几种其他的方式实现。尝试以下几种不同方法，体会TBC的工作方式：

a. 使用“Add existing…”菜单选项：

* 点击**hasDaughter**旁边的倒三角图标，选择**Add existing…**。
* 从弹出的**Add existing**界面中选择**Judith\_Shakespeare**和**Susanna\_Shakespeare**，点击**OK**。

b. 手动输入需要的信息：

* 点击**hasDaughter**旁边的倒三角图标，选择**Add empty row**。
* 输入Judith\_Shakespeare。
* 为Susanna重复以上操作，体验自动补全快捷键。输入**Sus**使用组合快捷键Ctrl+Space。

c. 使用**Basket**：

* 选中类**FemalePerson**。
* 点击**Instances**视图。
* 将**Instances**视图中的**Judith\_Shakespeare**和**Susanna\_Shakespeare**拖拽到**Basket**视图中。
* 选中类**MalePerson**，点击**Instances**视图，选中**WillIan\_Shakespeare**。
* 将**Basket**视图中**Judith\_Shakespeare**拖拽到**William\_Shakespeare**的**Resource Form**中的**hasDaughter**处。
* 单击选中**Basket**视图中的**Susanna\_Shakespeare**。
* 将**William\_Shakespeare**的**Resource Form**中**Judith\_Shakespeare**前面的菱形图标拖拽到**Basket**视图中。这一步操作将Judith\_Shakespeare声明为Susanna\_Shakespeare的女儿。在图3.20中的**Incoming References**处，你可以看到Susanna\_Shakespeare和William\_Shakespeare共同拥有女儿Judith\_Shakespeare。
* 你将看到如下图所示的对话框，点击OK。在这个实例中使用这种方式并不节省时间，但是你可以同时选中Basket视图中的多个资源（使用Ctrl或者Shift键），当你需要添加多个具有相同主语和谓语的声明或者添加多个具有不同主语的声明时，这种方式非常方便。

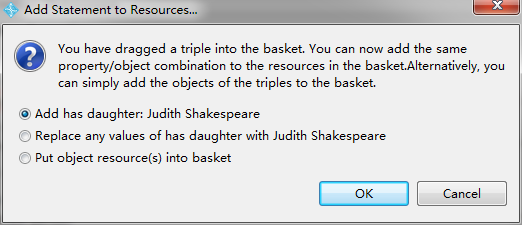


图3.19 向资源中添加声明

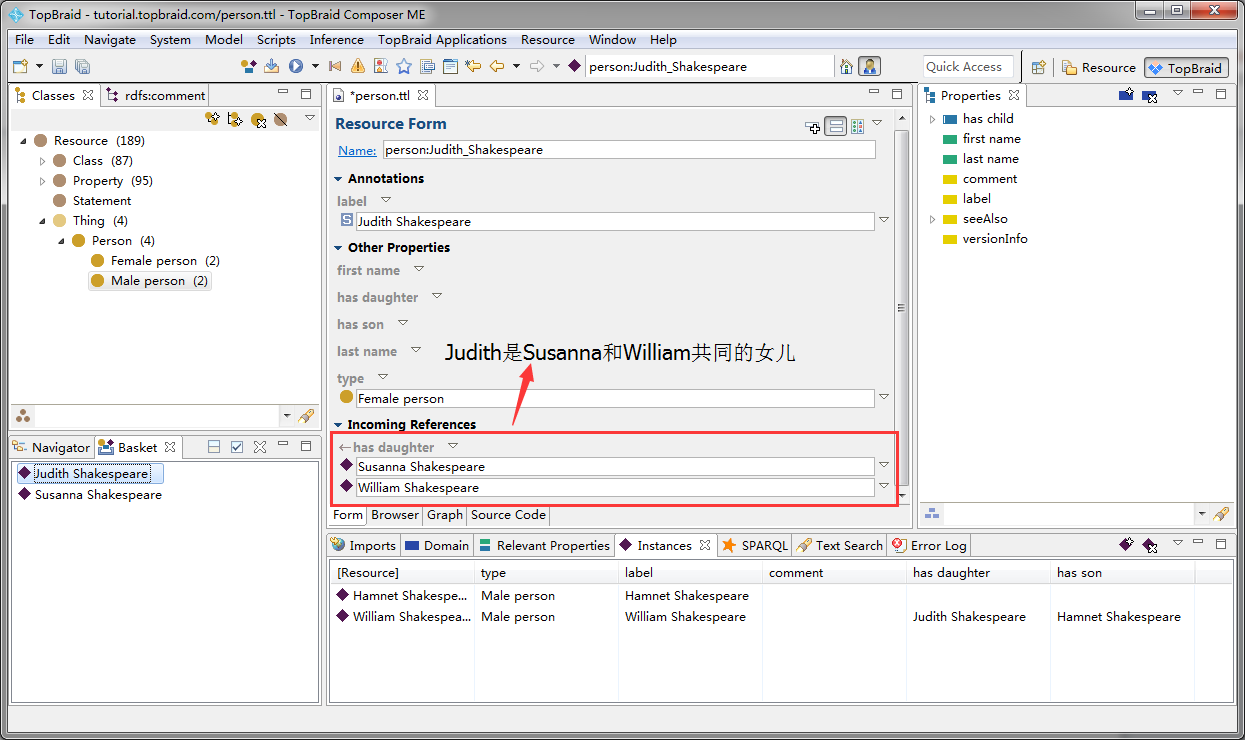


图3.20 将Judith声明为Susanna和William共同的女儿

|  |
| --- |
| 你可以很容易地改变任何已经创建资源的类型。比如说，你创建了类**Person**的一个实例**Susanna\_Shakespeare**。现在你想把**Susanna\_Shakespeare**作为类**FemalePerson**的一个实例。通过选中**Susanna\_Shakespeare**并将**FemalePerson**拖拽到**rdf:type**文本框中。 |

***3.4 执行SPARQL查询***

SPARQL(SPARQL Protocol and RDF Query Language)是一个用于查询RDF/OWL数据的标准。TBC自带一个内置SPARQL查询引擎。下面让我们体验一下SPARQL查询。

**Exercise 11：Run a default query**

1. 点击靠近界面底部的**SPARQL**视图。这个视图的选项和布局如图3.21所示。

图3.21 **SPARQL**视图

2. 打开**SPARQL**视图在查询面板上就会出现一个示例查询。这个查询将会列出所有与‘x rdfs:subClassOf y’模式相匹配的三元组，并将显示其中的x和y。换而言之，它会列出所有是另一个类的子类的资源。查询结果包括资源（主语）和作为资源父类的类（宾语）。如图3.22查询结果所示，第一列显示的是主语，第二列显示的是宾语。同一行中的主语为宾语的子类，这和‘x rdfs:subClassOf y’模式相匹配。

3. 点击绿色箭头或者在选中**SPARQL**视图时，通过Ctrl+Enter快捷键运行查询。可以观察到查询结果包括我们刚才创建的类以及内置的OWL和RDFS类。

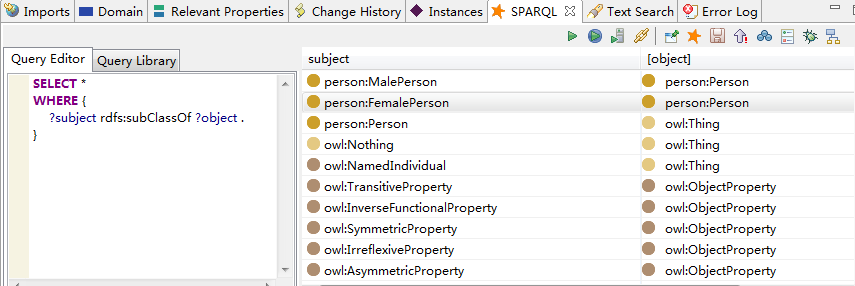


图3.22 示例查询结果

让我们写一个查询来检索具有女儿的所有人。根据上面的例子，一个返回所有父母和他们的女儿的查询应当如下所示：

|  |
| --- |
| **SELECT** ?subject ?object  **WHERE** { ?subject person:hasDaughter ?object. } |

将以上查询语句替换掉之前查询面板上的查询语句并点击运行。查询结果如图3.23所示。

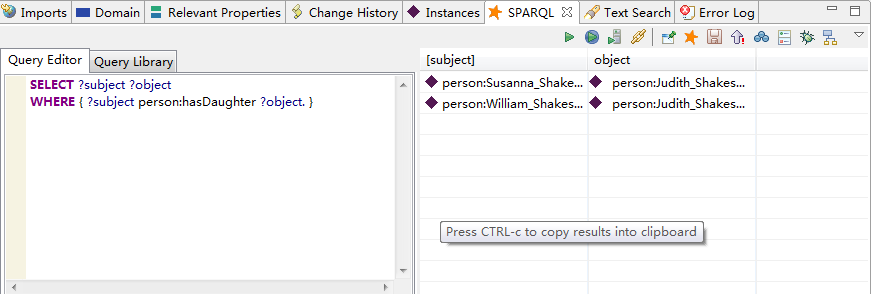


图3.23 查询具有女儿的人

**SPARQL**视图的**Query Editor**为你在书写查询语句的时候提供帮助，它用紫色高亮SPARQL关键词并且在不识别的属性名下面用红色波浪线标出。例如，你试图在属性person:hasDaughter后面加上一个X，那么在person:hasDaughterX下面立即就会出现红色的波浪线。这表明person:hasDaughterX不是现有本体中的属性，在查询中将不起任何作用。（去掉“X”继续）。

|  |
| --- |
| 当你创建的工程文件中包含很多命名空间和模块时，引入一个新的前缀很有用。当你创建一个RDF文件的时候，在Create RDF File对话框中取消勾选”Set a default namespace in the new file”会自动为base URI创建一个前缀。 |

为了学习如何在Composer中指定前缀，让我们为命名空间http://tutorial.topbaid.com/person#创建一个前缀。

**Exercise 12：Create a namespace prefix**

1. 点击TBC菜单下面的图标，在**Resource Form**中显示本体自身的信息。

2. 选择**Overview**标签，点击**Namespace Prefixes**表格右侧的图标。

3. 输入到**Prefix**列中，输入http://tutorial.topbraid.com/person#到**Namespace URI**列中。

4. 按Enter键。

5. 点击**File > Save**或者使用快捷键**Ctrl+S**进行保存。

**Exercise 13：Query for all parents of females**

1. 点击**SPARQL**视图，将**rdfs:hasDaughter**改为**person:hasDaughter**。

2. 点击运行SPARQL图标。你的界面将显示如下图所示：

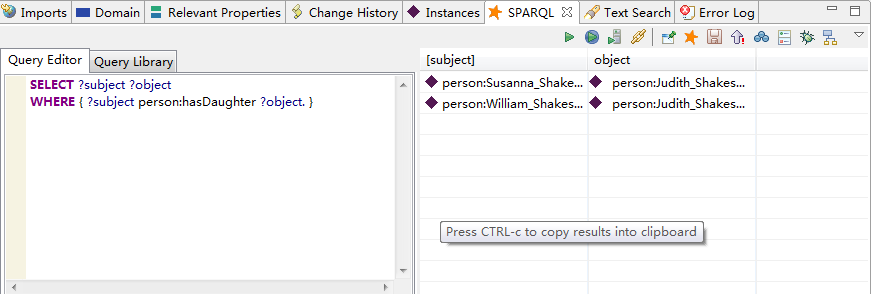


图2.24 **SPARQL**视图

3. 将查询语句中的person:hasDaughter改为person:hasChild。

4. 运行查询，你将看不到任何结果。

为什么我们查询不到任何结果？hasDaughter和hasSon是hasChild的子属性，按照RDFS推理规则（见附录A），按理说查询结果应该返回William Shakespeare和他所有的孩子才对。

在这里做一个小小的解释。我们刚才的查询只是在断言（声明）的三元组上做的查询，并没有在推理的三元组之上进行查询。我们没有声明具有person:hasChild谓语的三元组。

有两种方法处理这个问题：

* 直接进行查询；
* 使用一个推理引擎。

5. 将查询语句改为如下所示。这个查询出任何具有person:hasChild谓语的三元组的主语和宾语或者查询出所有person:hasChild的超级属性(superproperty)。

|  |
| --- |
| **SELECT** ?subject ?object  **WHERE** {  ?p rdfs:subPropertyOf \* person:hasChild .  ?subject ?p ?object .  }  6. 运行以上查询，你将看到如下图所示。 |

图2.25 查询具有person:hasChild谓语的主语和宾语

下面的练习中，我们将使用推理引擎运行推理，观察它是如何改变查询结果的。

**Exercise 14：Run Inferences and query for all parents**

1. 选择**Inferences > Run Inferences…**菜单选项。（如果你运行的是TBC免费版（其他版本也可能出现，如果执行**Inferences > Run Inferences…**后出现如图2.26的错误提示，请按这个步骤解决），在选择**Inferences**之前，你需要首先单击home图标，选择本体的**Profile**标签，勾选上OWL 2 RL来打开SPIN library，点击保存），如图2.27所示。

2. 推理的三元组将会显示在**Inferences**视图中，如图2.27所示。

3. 在SPARQL视图中，点击使用“使用当前配置的推理”图标，这个选项在使用推理进行查询与不使用推理进行查询之间切换。点击这个图标之后，使得下次查询时使用推理。然后运行以下查询：

|  |
| --- |
| **SELECT** ?subject ?object  **WHERE** { ?subject person:hasDaughter ?object. } |

你会看到William Shakespare具有两三个孩子，如图3.28所示。（如果使用的是免费版的TBC，那么就没有“使用当前配置的推理”图标。一旦OWL 2 RL推理被打开，使用绿色三角形图标运行SPARQL查询，将会使用推理进行查询）。

4. 在**Instances**视图中，点击**William\_Shakespeare**，观察其**Resource Form**。这时候**Resource Form**中就已经包含了推理的属性。这些属性值的背景以浅蓝灰色显示，如图2.29所示。

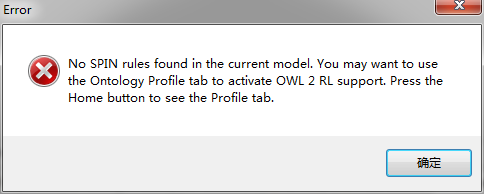


图2.26 如果没选中OWL 2 RL可能出现的错误提示

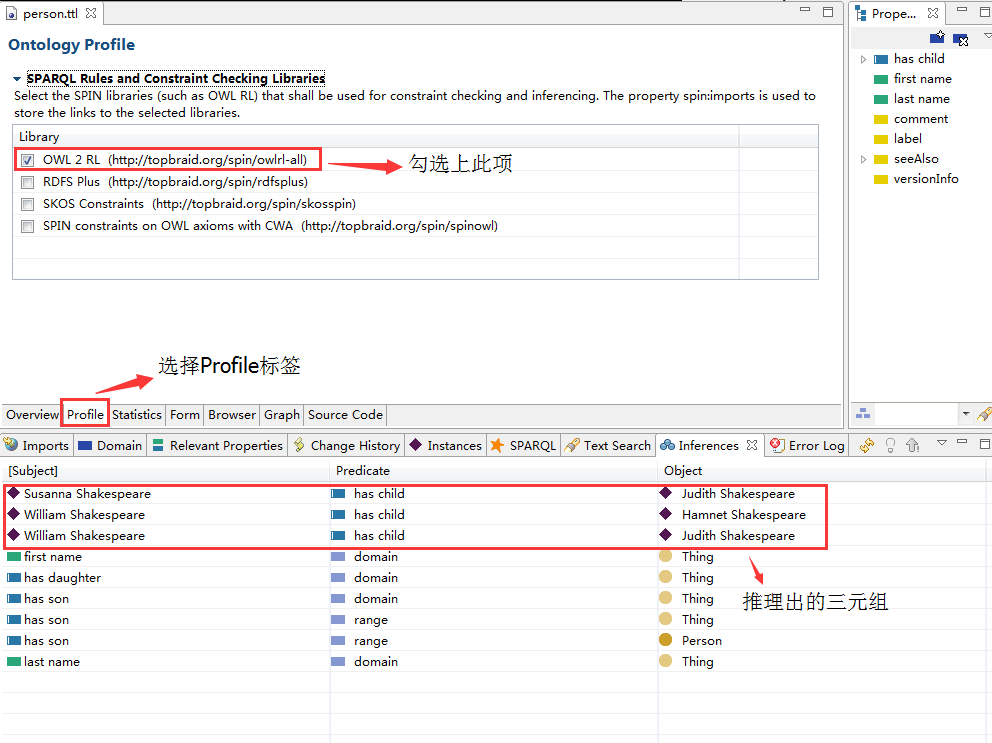


图2.27 选中OWL 2 RL后推理出的三元组

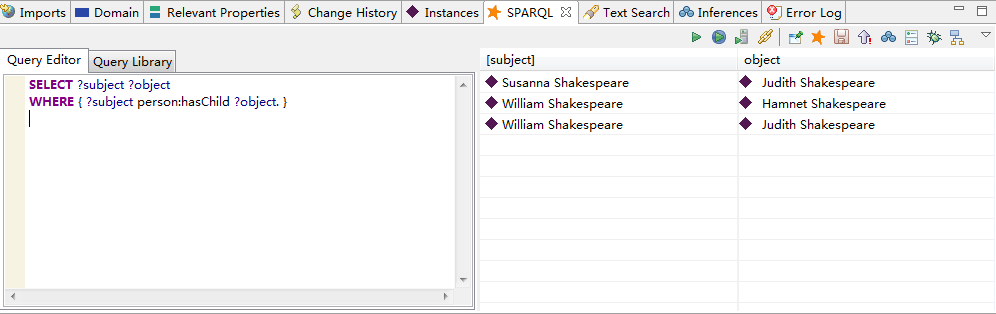


图2.28 使用推理后的查询结果



图2.29 **Resource Form**中推理出的属性值以浅蓝灰背景显示

|  |
| --- |
| TBC保留对声明的(asserted)和推理的(inffered)陈述区分。如果现在你关闭本体并重新打开它。你会看到推理陈述已经不存在。  然而，通过将推理的陈述转换为声明的陈述的方式，可以将单个推理的陈述或者全部推理的陈述持久化保存：   * 通过点击一个推理陈述右侧的**Show widget menu**图标（倒三角图标）（例如，William Shakespeare的**Resource Form**中person:hasChild的两个属性值之一），你可以选择**Assert inferred Statement**（声明推理的陈述）选项。     图2.30 Widget Menu   * 或者，在菜单选项中选择**Inference - > Save Inference graph…**保存全部的推理三元组图。 |

5. 保存查询以便将来再次使用

* 选中**Classes**视图中的**person:Person**类。
* 点击**SPARQL**视图中的‘将当前查询附加到选定的资源’图标，在弹出的询问是否导入SPIN命名空间确认对话框中，点击“确定”；这会将查询附加到类上。你将看到查询已经被附加到**Person**类资源编辑器的**spin:query**文本框中。
* 有两种方式来运行保存的查询：

a. 点击**spin:query**文本框（不是**spin:query**名字）右侧的**Show widget menu**三角图标，选择**Execute as SPARQL query**。

b. 选择**SPARQL**视图的**Query Library**标签，你将看到本体中的这个保存的查询和其他保存的查询。确保这个查询旁边的复选框被勾选上，并且“使用当前配置的推理”图标已被选中。点击执行**SPARQL**查询图标。**Query Library**标签中将会出现如下图所示的查询结果：

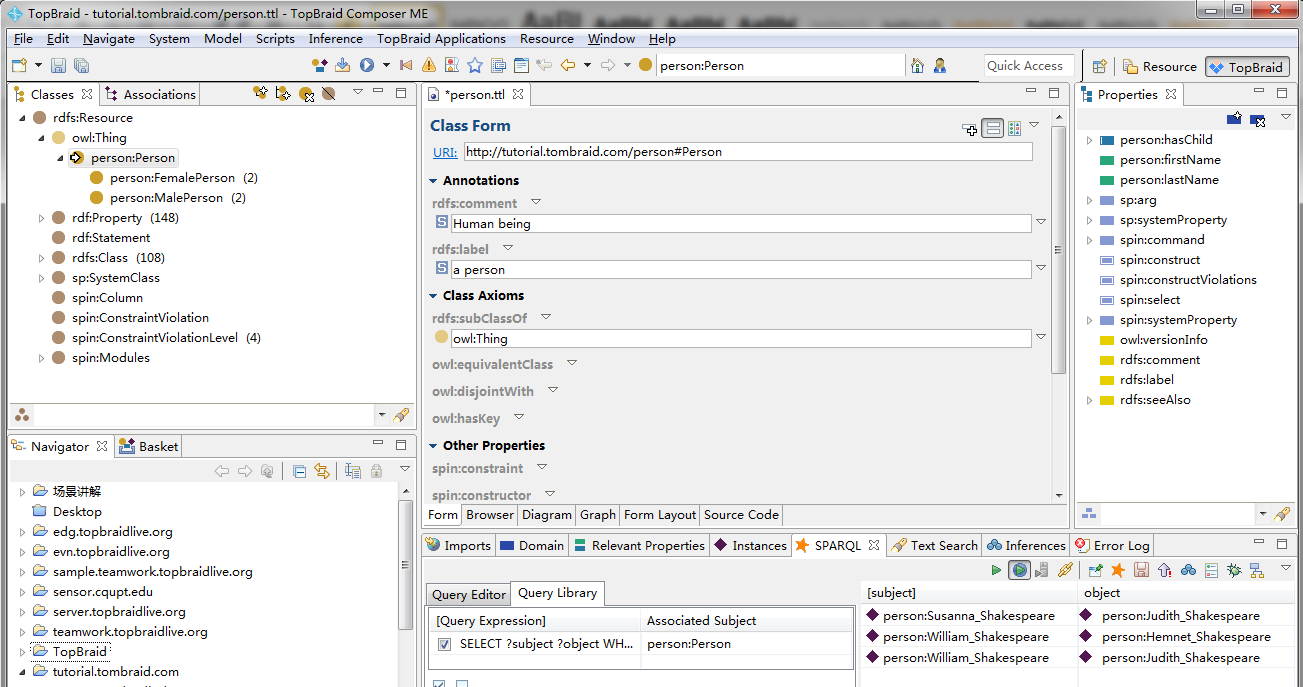


图2.31 **SPARQL Query Library**

|  |
| --- |
| TBC将会识别类的**spin:query**属性值中的SPARQL语法并提供菜单选项来执行它们。要想在**Properties**视图中找到**spin:query**属性，可以使用**Properties**视图底部的查询文本框。在查询文本框中输入**query**，点击右侧的查询图标就能搜索出**spin:query**属性。用同样的方式也能在**Classes**视图中查找到想找的类名。 |

|  |
| --- |
| 你也可以将规则和约束与本体的资源联系起来。想要学习更多关于这部分的知识，请参阅SPIN教程：<http://www.topquadrant.com/spin/tutorial/index.html> |

6. 导出你的查询结果：

* 在**SPARQL**视图中点击**Export results to file**图标。
* 在出现的对话框中，为导出的文件创建一个名字，选择保存格式，然后点击**Save**。

***3.5 扩展本体***

在这一部分我们将会添加另外两个对象属性并观察其涉及的推理。

**Exercise 15：Create an object property called hasSpouse**

1. 点击**Properties**视图中的**Create property**图标，这时**Create property**对话框将会出现。

2. 展开**owl:ObjectProperty**。选择**owl:SymmetricProperty**(对称属性)。

3. 将新属性命名为person:hasSpouse，点击**OK**。

4. 在新属性的**Property Form**中，设置新属性的**rdfs:domain**和**rdfs:range**为**person:Person**。

在本练习之前，我们只适用到RDFS词汇来定义person本体中的属性。（更多关于RDFS和OWL的信息请参加Appendix A）。这是首次用到OWL概念。

|  |
| --- |
| 虽然在前面的各个例子中，我们将类定义为OWL类，将属性定义为OWL属性，但是我们并没有做任何需要OWL表达性的建模。到目前为止，我们做的所有练习都可以不使用任何OWL声明来实现。例如，创建一个owl:Thing的子类可以用创建rdfs:Resource的子类来代替，这样就将类声明为RDFS类。  通过‘隐藏’所有的OWL约束，Composer可以被配置为仅使用RDFS的编辑器。这可以通过转到**Window > Preferences**，展开**TopBraid Composer**目录，在**Classes View**和**Properties View**中做适当的选择实现。 |

|  |
| --- |
| 对称属性（symmetric property，此处定义为**p**）涉及到以下推理：  1. 如果有一个三元组**a p b**，它将会推理出**b p a**。如果我们的例子中已经包含了资源**Anne\_Hathaway**，那么，如果**William\_Shakespeare hasSpouse Anne\_Hathaway**，会推理出**Anne\_Hathaway has Spouse William\_Shakespeare**，因为**hasSpouse**被定义为一个对称属性。  2. 如果**p rdfs:domain a**，那么**p rdfs:domain b**。类似地，如果**p rdfs:range a**，那么**p rdfs:range b**。 |

|  |
| --- |
| 如果你显式地运行推理，大多数推理将会仅仅被显示在Composer中。然而，Composer也有有限数量的琐碎推理交互性的执行或者‘just in time’。这些推理是：   * 互逆一致性。如果声明了属性**p owl:inverseOf q**，TBC将会推理出属性**q owl:inverseOf** 属性**p**。 * 互逆属性定义域和值域的一致性。如果声明了**p rdfs:domain a**和**p owl:inverseOf q**，TBC将会推理出**q rdfs:range a**。   因为TBC维持这些自动推理，我们建议如果你使用互逆属性，你应该指定属性一个方向的定义域和值域，让TBC推断属性逆向的定义域和值域。  给出上面单向的声明，你就能够使用SPARQL进行双向的查询。我们通常不鼓励用户指定逆向属性。 |

**Exercise 16：Create object property called hasFamilyMember**

1. 点击**Create property**图标。**Create property**对话框将会出现。

2. 选择**owl:ObjectProperty**。重命名属性名为**person:hasFamilyMember**，点击**OK**。

3. 在**Properties**视图中，将**hasChild**和**hasSpouse**拖拽到**hasFamilyMember**上，将**hasChild**和**hasSpouse**变为**hasFamilyMember**的子属性。

4. 选择**File > Save**保存你的工作。

|  |
| --- |
| 当子属性已经指定了定义域和值域后，那么就不需要甚至不建议为其父属性指定定义域和值域。另一种本体设计模式（不常使用，但是在某些情况下适用）是指定父属性的定义域和值域而省略对子属性定义域和值域的指定。 |

**4 使用Imports和处理多个本体**

在这一章中，我们将会把topquadrant.ttl和person.ttl连接起来，用于描述人出生的地点。

topquadrant.ttl本体可以在**Navigator**视图中通过展开目录**TopBraid > Examples >topquadrant.ttl**访问。

我们不会去将person.ttl本体中的信息复制到topquadrant.ttl本体，而是使用导入并重用本体的方式连接两个本体。

OWL本体可以导入一个或者多个OWL本体。本体一旦被导入，不但类、属性和个体可以被其他本体引用，被导入本体中的公理(axiom)和事实(fact)实际上也被包含于原来的本体中。OWL允许循环导入本体，例如，topquadrant本体可以导入到person本体中，person本体也可以导入到topquadrant本体中。在我们的练习中，我们将topquadrant本体导入到person本体中。

注意使用命名空间在另一个本体中引用类、属性和个体和完全导入本体之间的区别。

**Exercise 17：Import topquadrant ontology and make changes**

1. 在**Navigator**视图中，打开person.ttl文件。

2. 在**Classes**视图菜单中，点击倒三角图标选择**Start hierarchy with owl:Thing**。

3. 点击**Imports**视图，将**Navigator**视图中**TopBraid > Examples**目录下的**topquadrant.ttl**拖拽到**Imports**视图中。

另一种指定导入的方式是通过点击图4.1所示的Import local file…图标来实现的。



图4.1 **Imports**视图图标

当**Import local file**对话框出现后，展开workspace文件夹直到你已经定位到topquadrant.ttl，点击**OK**。

|  |
| --- |
| 要想导入位于web上的RDF图，使用Import from URL图标。 |

4. 现在你的界面应当如图4.2所示。注意有些类和属性（例如，**owl:Thing > schema:Country**）以褪色的图标显示。这些类和属性来自被导入模型中的资源。

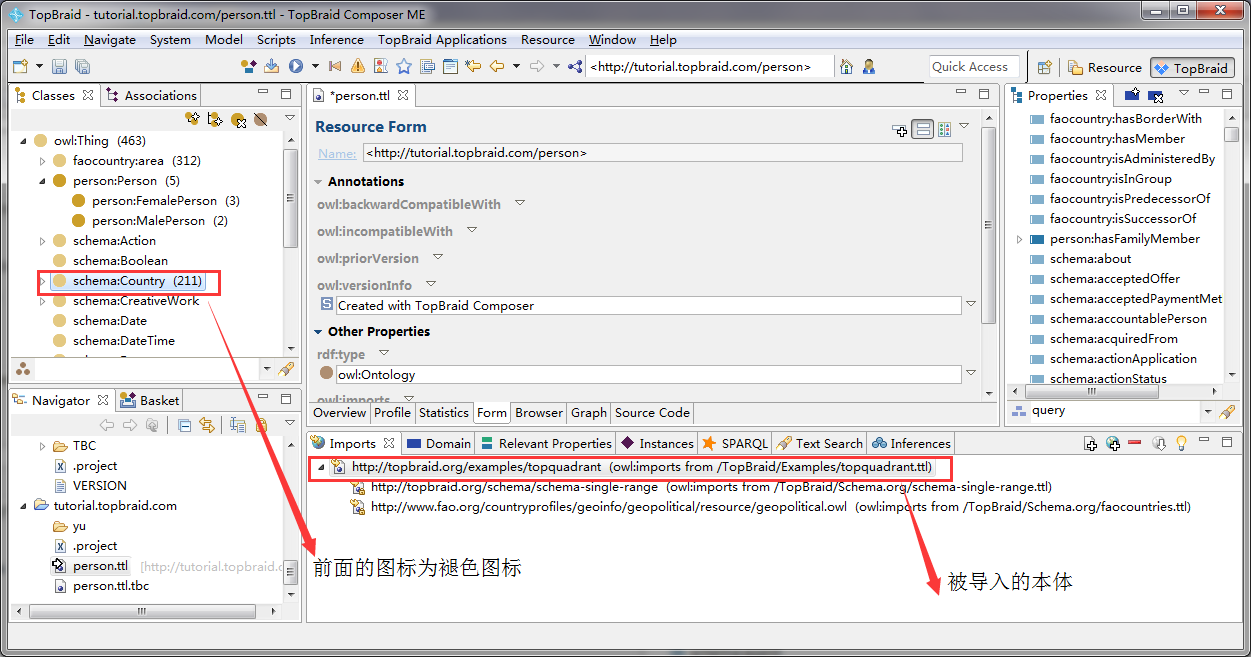
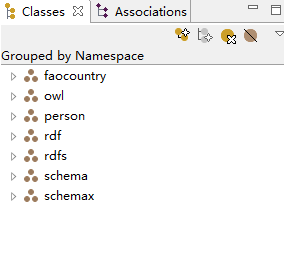


图4.2 Person本体和导入的topquadrant本体

5. 创建一个名为person:bornIn的对象属性。将它的定义域设为**Person**（来自源本体person.ttl），值域设为**Country**（来自topquadrant本体）。我们刚才通过导入本体的方式创建了两个本体之间的连接桥梁。

6. 通过**File > Save**保存修改。

7. 当处理多个导入的本体时，按命名空间排序来查看类和属性非常有用。在**Classes**视图中的左下方，点击**Group by namespaces**图标来按照命名空间组织目录视图。用这种显示方式，rdfs:subClassOf层次性将不会被显示。再次点击这个图标将会还原类目录视图。注意**Properties**视图也有一个类似的功能。

8. 选中**Person**类，将其名字改为person:HumanBeing。点击**Enter**。

9. 通过使用**File > Close**关闭person本体。

10. 一个对话框将会出现来提供对修改的保存。因为我们不想最近做的修改，点击**Deselect All**，然后点击**OK**。

图4.3 按命名空间组织的Classes视图

|  |
| --- |
| 当处理多个本体的时候，很重要的一点是知道相对于旧的声明哪些新声明和/或改变被保存。Composer遵循以下规则：   * 新声明被添加到当前选中的本体。属性person:bornIn被添加到person.ttl中。 * 被导入资源的**Form**中的数据可以被编辑。对现有声明的任何改变都将会写入到他们源自的本体中。例如，如果将topquadrant.ttl导入到person.ttl，在查看topquadrant.ttl本体时我们将**FemalePerson**是**Person**的子类这一事实做了一个修改（实质上移除了声明**person:Person rdfs:subClassOf person:FemalePerson**的三元组），这个改变将会写入到person.ttl文件。 * 如果我们说**hasSon**的定义域不再是**Person**，而是变为**Parent**（一个为了这个目的创建的新类），新三元组的位置将依赖于以什么方式做的改变： * 如果我们用**Parent**重写**Person**，则更改将作为对该文件中先前存在的三元组的更新保存在person.ttl中。 * 如果我们删除了关于定义域的条目，然后添加了一个新的定义域，删除将在person.ttl中完成，新的三元组将会保存在topquadrant.ttl中（如果我们从topquadrant.ttl中做了这样一个改变，并且是topquadrant.ttl导入到person.ttl中）。 * 当我们将类person:Person的URI改变为person:HumanBeing时，这一改变是对person.ttl本体中Person类定义做出的修改。TBC分析和更新person.ttl中所有对这类的引用。它还会扫描查看是否有其他的本体也导入了这个本体。   当处理模块化导入的本体时，因此可能有意或者无意地对导入的文件做了更改。如果导入的文件来自于Web，这种改变在关闭本体的时候将会消失。对于本地文件，这些改变将会被保存。  Composer保留了一个对所有更改的日志—在**Change History**视图中可以访问。未被保存的修改可以通过使用**Edit > Undo**来回滚。  当使用他人的本地本体时，一个比较好的做法是将它们锁定（使其为只读模式），以防意外更新。锁定可以通过在**Navigator**视图中选中这个文件并点击图标来实现。 |

person本体具有一些关于人的一般性（schema层面）信息。它同样也具有关于Shakespeare家庭非常特定的信息。因为我们对人和他们的旅行爱好之间的关系这种一般性情形感兴趣，所以将关于Shakespeare家庭的信息分离到一个它自己的文件中很有意义。

**Exercise 18：Move Resources between ontologies**

1. 在**Navigator**视图中，在**tutorial.topbraid.com**工程中创建一个名为**shakespeare.ttl**的新RDF文件。为它创建一个base URI。

2. 通过将**Navigator**视图中的**person.ttl**拖拽到**Imports**视图的方式，将**person.ttl**导入到**shakespeare.ttl**。现在你的界面应当如图4.4所示。

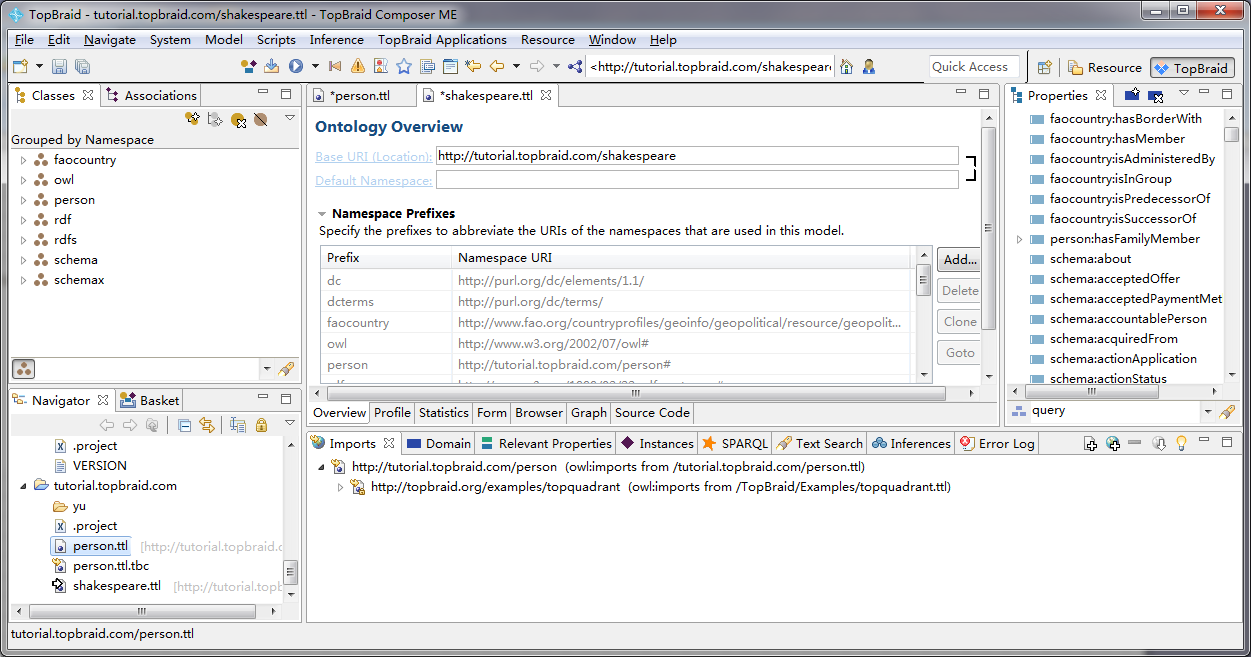


图4.4 新创建的shakespeare.ttl文件

3. 选择**File > Save**菜单选项。注意观察文件名前的星号消失，说明文件已经被保存。

4. 打开person.ttl。

5. 选中**Classes**视图中的类**MalePerson**，点击**Instances**视图。

6. 将**Instances**视图中的**William\_Shakespeare**和**Hamnet\_Shakespeare**拖拽到**Basket**视图中。

7. 选中类**FemalePerson**。

8. 将**Instances**视图中的**Judith\_Shakespeare**和**Susanna\_Shakespeare**拖拽到**Basket**视图中。

9. 选中**Basket**视图中的这四个资源并将它们拖拽到**Navigator**视图中的Shakespeare.ttl文件上，如图4.5所示。

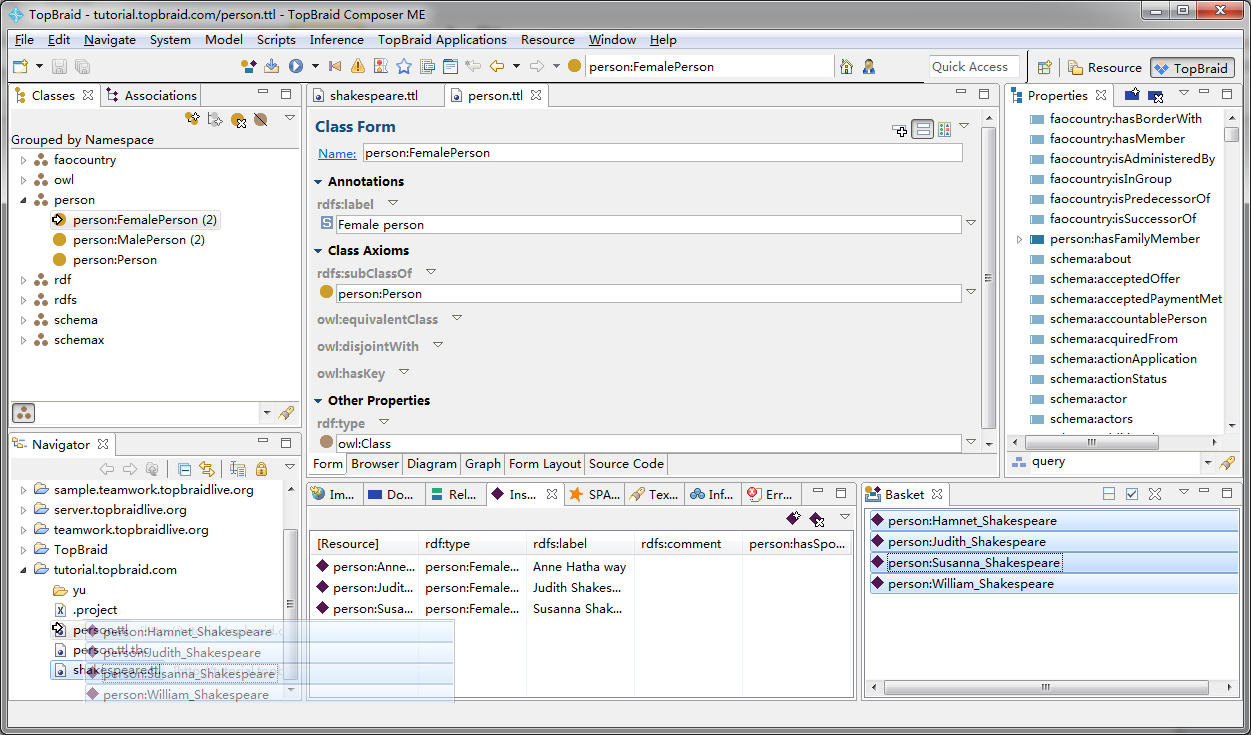


图4.5 将资源移动到shakespeare.ttl文件中

10. 在弹出的确认是否移动资源的对话框中，点击“是”。

11. 你可以观察到person.ttl本体中**Person**类中不再含有以上四个实例。

12. 切换到shakespeare.ttl文件，定位到William Shakespeare资源。可以观察到William Shakespeare和他孩子之间的关联关系依然存在。

|  |
| --- |
| 注意，在前面的练习中，所有这四个个体是被一次性移动的。下面让我们看一下如果先移动**person:William\_Shakespeare**，然后再移动**person:Susanna\_Shakespeare**会发生什么：   * 在出现的‘should the namespace be adjusted’询问中选择‘yes’，这会把**person:William\_Shakespeare**的记录ID修改为**shakespeare:Shakespeare**。 * 至此，**person:Susanna\_Shakespeare**的记录ID保持不变。因此，shakespeare.ttl文件中现在将会具有以下三元组：**shakespeare:William\_Shakespeare person:hasDaughter person:Susanna\_Shakespeare**。 * 如果我们现在开始移动Susanna，并且对是否改变命名空间这个问题选择‘yes’，她的记录ID将会变为shakespeare:Susanna\_Shakespeare，并且她和William之间的联系将会消失。   如果你一次性移动所有资源，Composer将会在它们的URI修改之后依然维持相连接资源之间的关系。  如果你不能一次性地移动所有相互连接的资源，那么除了第一个移动的资源之外，移动其他资源时在出现‘should the namespace be adjusted’问题选项时选择‘no’。并且在移动过去之后更改资源ID，使它们手动包含正确的命名空间。  如Exercise17 所示，如果你改变了一个资源的URI，TBC将会检查是否有文件导入了这个模型并且这些文件可能因此受到重命名的影响。对话框将会列出潜在受影响的模型。在你的指引下，TBC将会把改变扩散到所有受影响的文件中。  为了更精确地控制移动操作，可以通过选择**Triples**视图来实现。通过选择**Window > Show View > Triples**来访问**Triples**视图。 |

**5 使用OWL定义类**

如前所述，所有关于属性的RDF声明都是全局性的。如果声明了**Person**在**hasChild**属性的定义域内，这个声明在任何使用**hasChild**属性的地方都保持正确。换句话说，它指定的是属性**hasChild**而不是类**Person**。

让我们思索以下例子：

* 我们已经通过使用rdfs:domain声明说明**Person bornIn Country**。
* 我们现在想要声明一个人出生的地方只能有一个国家。

这里就需要引入OWL约束。与定义域和值域不同，约束定义的是类。约束用来限制属于一个类的个体。OWL支持以下约束：

* 量词约束(Quantifier Restrictions)—**allValuesFrom**和**someValuesFrom**。

allValuesFrom也叫做通用量词(universal quantifiers)；

someValuesFrom也叫做存在量词(existential quantifiers)。

* 基数约束(Cardinality Restrictions)—**minCardinality**，**cardinality**和**maxCardinality**。
* **hasValue**约束。

约束可以使用**rdfs:subClassOf**或者**owl:equivalentClass**声明。两者之间的不同在于从声明中获得的推理结果不同。例如：

* 假如US Citizen是所有国籍属性值等于(hasValue)’USA’的事物的一个子类，意味着：
* 如果已知一个个体是US Citizen，可以被推理出他的国籍是’USA’。
* 假如US Citizen等价于所有国籍属性值等于(hasValue)’USA’的所有事物，意味着：
* 如果已知一个个体是US Citizen，那么可以推理出他的国籍是’USA’，并且
* 如果已知一个个体的国籍是’USA’，可以推理出他是一个USA Citizen。

让我们用一个例子来开始这一部分并创建一个约束。

**Exercise 19：Create someValuesFrom restriction using the Edit Restriction dialog**

1. 在类person:Person的**Class Form**中，点击靠近**owl:equivalentClass**小部件的**Show Widget menu**图标，选择**Create restriction…**，如图5.1所示。这时，**Edit Restriction**对话框将会出现，如图5.2所示。

2. 在对话框中，从**On Property**目录中选择**bornIn**属性，在**Restriction Type**选项中选择**cardinality(exactly)**单选按钮。在Filter文本框中，输入值**1 schema:Country**来表明一个人只可以在一个确切的国家出生。点击**OK**。

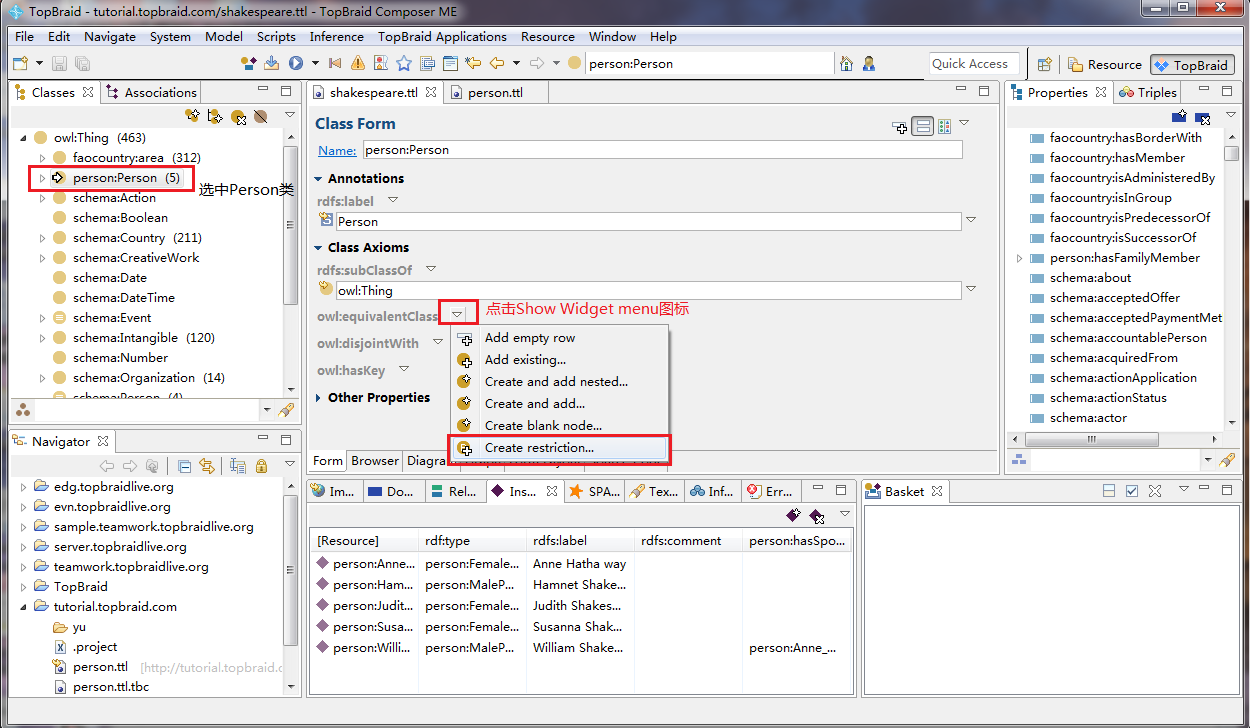


图5.1 创建约束

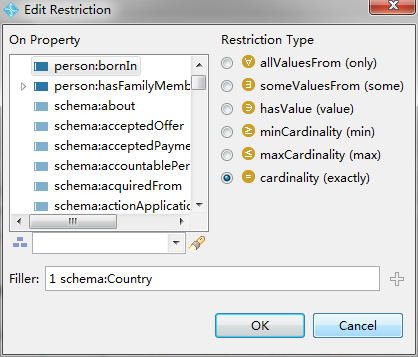


图5.2 **Edit Restriction**对话框

现在你的界面应该如图5.3所示。注意约束使用叫做曼彻斯特OWL语法（Manchester OWL syntax，由曼彻斯特大学发明。请参考：<http://www.w3.org/2007/OWL/wiki/ManchesterSyntax>）的de-facto标准概念来定义。

作为使用对话框创建约束的替代方式，你也可以通过在**owl:equivalentClass**或者**rdfs:subClassOf**窗口小部件处添加一个空行然后根据曼彻斯特语法直接键入约束的方式创建一个约束。

下面的部分解释常用的曼彻斯特语法关键词。

***5.1 约束关键词***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OWL** | **DL**  **Symbol** | **Manchester Syntax Keyword** | **Example** | **Inferences** |
| someValuesFrom |  | some | hasChild **some** man | 当与owl:equivalent类一起使用时，启用对三元组主语类型的推理 |
| allValuesFrom |  | all | hasSibling **all** woman | 启用对三元组主语类型的推理 |
| hasValue |  | has | hasCountryOfOrigin **has** England | 当与owl:equivalent类一起使用时，启用对三元组主语类型的推理 |
| minCardinality |  | min | hasChild **min** 3 | 当与owl:equivalent类一起使用时，启用对三元组主语类型的推理 |
| Cardinality |  | exactly | hasChild **exactly** 3 | 当在开放世界假设下使用时，不会导致任何分类推理 |
| maxCardinality |  | max | hasChild **max** 3 | 当在开放世界假设下使用时，不会导致任何分类推断 |

注意OWL允许hasValue约束有一个数据类型文字作为填充。TopBraid Composer中的语法示例如下：

* “**value**”作为**xsd:string**字面量
* **42**作为**xsd:int**字面量
* **4.2**作为**xsd:float**字面量
* **true**或者**false**作为**xsd:boolean**字面量

***5.2 布尔类构造函数(Bool Class Constructors)***

|  |  |  |  |
| --- | --- | --- | --- |
| OWL | DL  Symbol | TopBraid Composer Syntax Keyword | Example |
| intersectionOf |  | **and** | Doctor and Female |
| unionOf |  | **or** | Man or woman |
| complementOf |  | **not** | not Child |

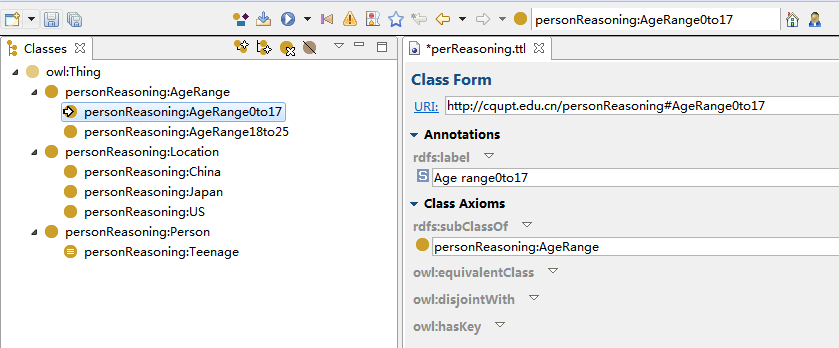
***5.3 复杂类表达式***

复杂类表达式可以使用上面的布尔构造函数和约束来构造。例如：

Person **and** hasChild **some** (Person **and** (hasChild **all** man) **and** (hasChild **some** Person))描述了一类人的集合，这类人至少有一个孩子，这些孩子有一些仅为男性的孩子（例如，爷爷只有孙子）。注意应该使用括号来消除表达式的歧义。

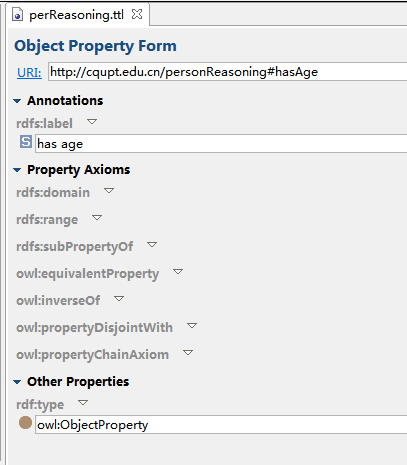
***5.4 Demo1——使用TBC的ObjectProperty来约束类型***

Step1：类别初步划分结构如图所示：



其中，

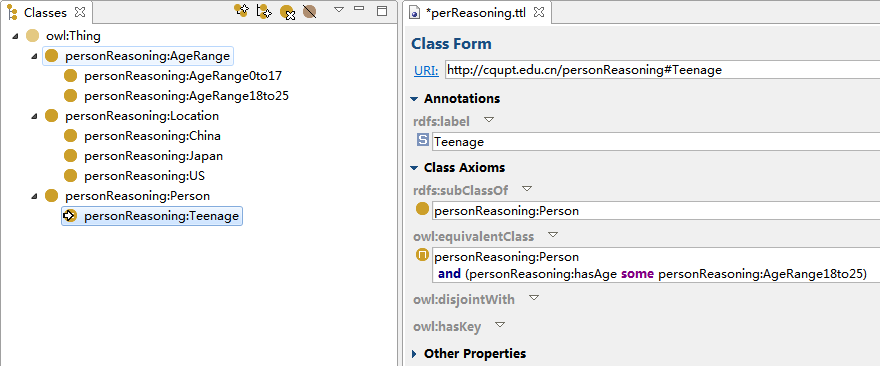
has Age属性定义为：ObjectProperty（ObjectProperty与DataProperty的区别见3.2节）



其中，Person的子类Teenage的定义使用类型公理/规则来描述，即：

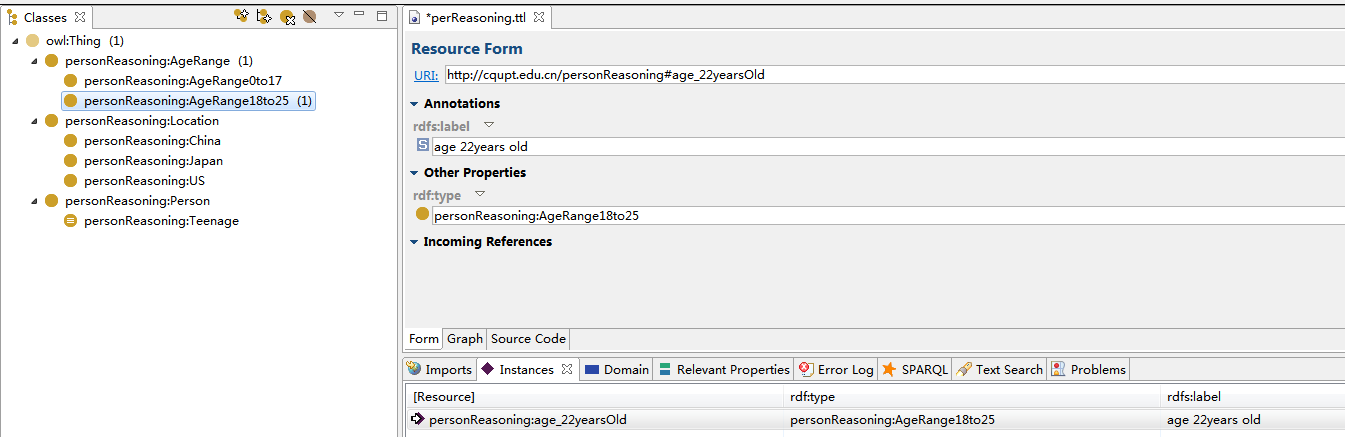
Teenage为：既为人类，同时年龄属性的值在“18to25”类中

:Person and (:has Age some :AgeRange18to25)

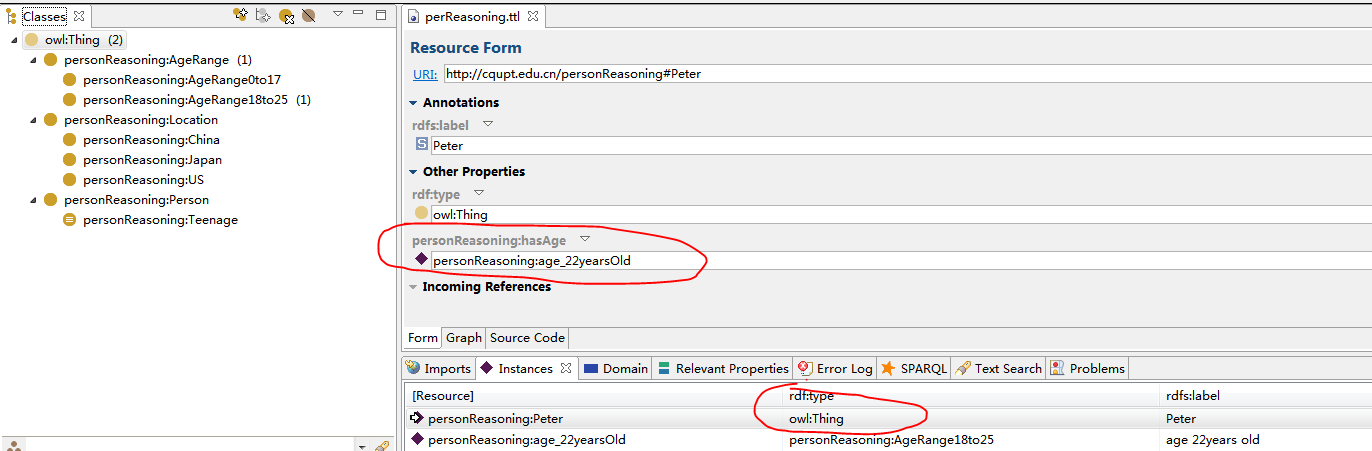


Step2:创建实例对象，并为其赋予相应的属性及其值。

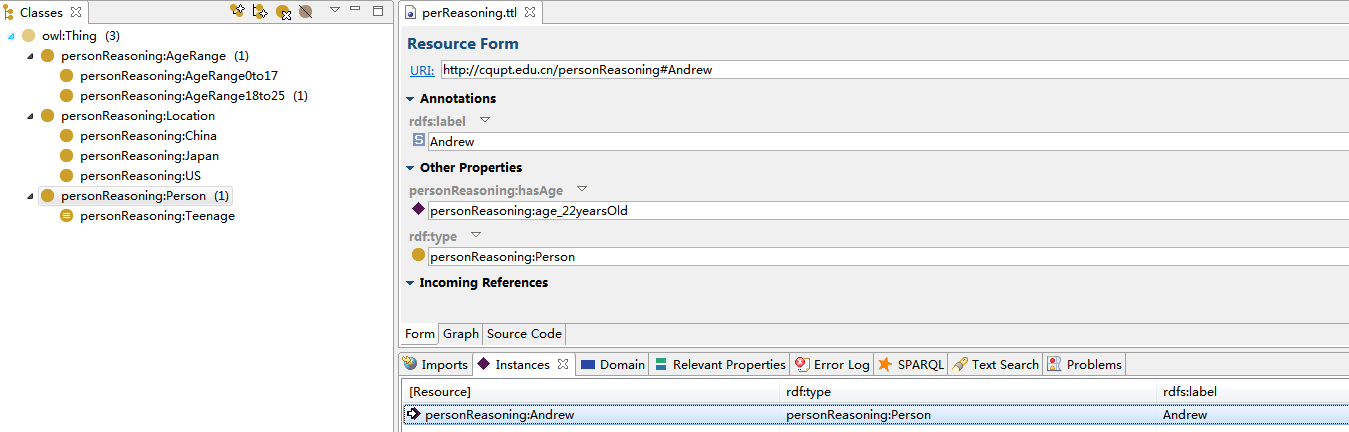
创建一个AgeRange18to25的年龄，如：age\_22yearsOld，即：



创建一个Thing的实例，并添加属性hasAge，及其值“age 22years old”。如：Peter，即：

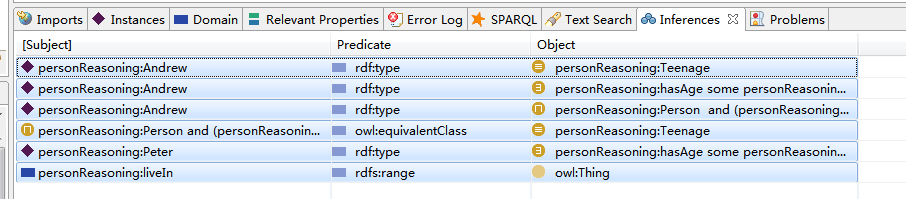


创建另一个Person类的实例，并添加属性hasAge，及其值“age 22years old”，即：



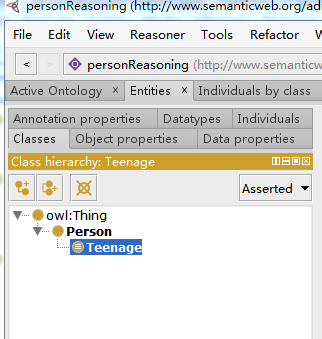
Step3:执行类型推理

结果：Andrew的类型推理得到rdf:type Teenage而Peter不是Teenage类型，因为根据事实不能得到Peter是Person这一类，不满足Teenage定义的条件。

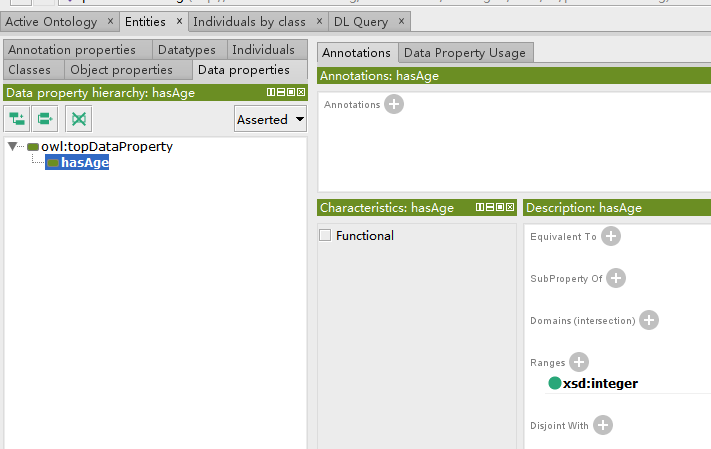


***5.5 Demo2——使用Protege的DataProperty来约束类型***

Step1：定义类Person，及其子类Teenage，即：

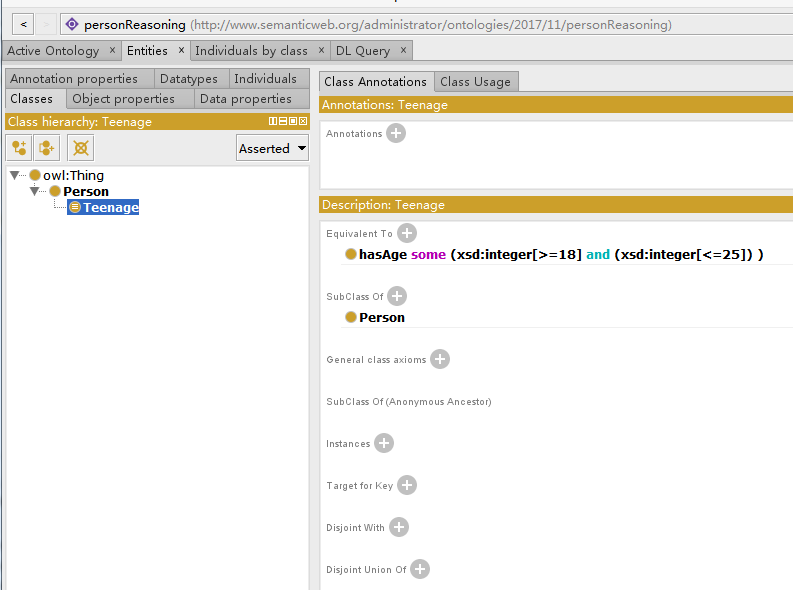


Step2：定义属性hasAge为Dataproperty类型属性，并说明其值域为xsd:integer,即：



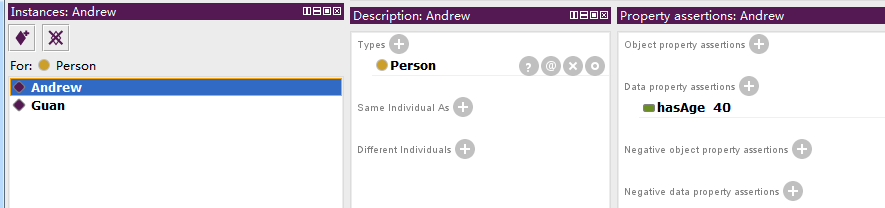
Step3：定义Teenage的等价公理：

Teenage :has:Age some (xsd:integer[>=18] and (xsd:integer[<=25]) )

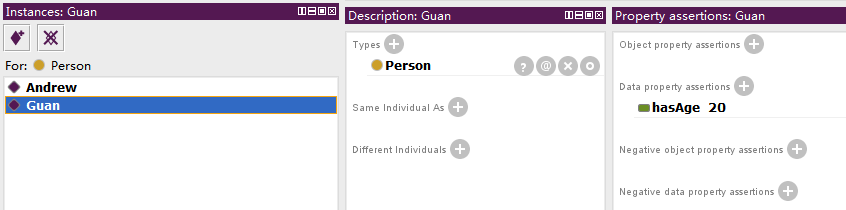


Step4:创建需要归类的Person实例，Andrew和Guan，分别赋予其hasAge属性值为40和20，即：

Andrew：

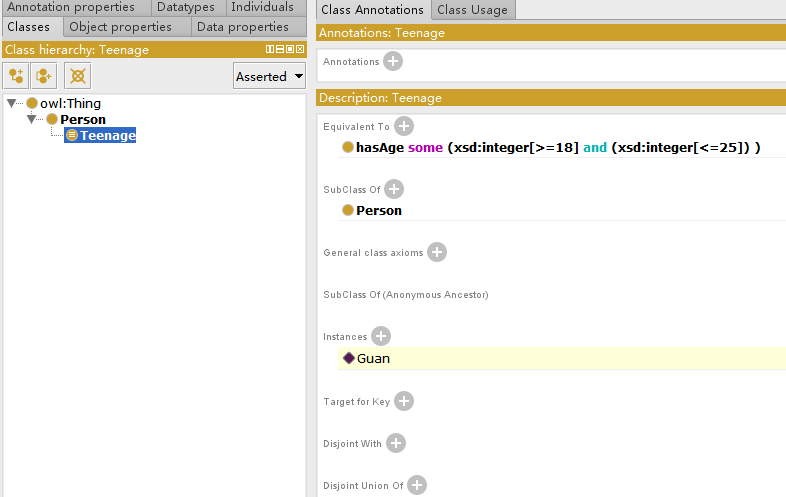


Guan：



Step5：执行推理

结果：Guan归为Teenage类，而Andrew不属于Teenage类，即：

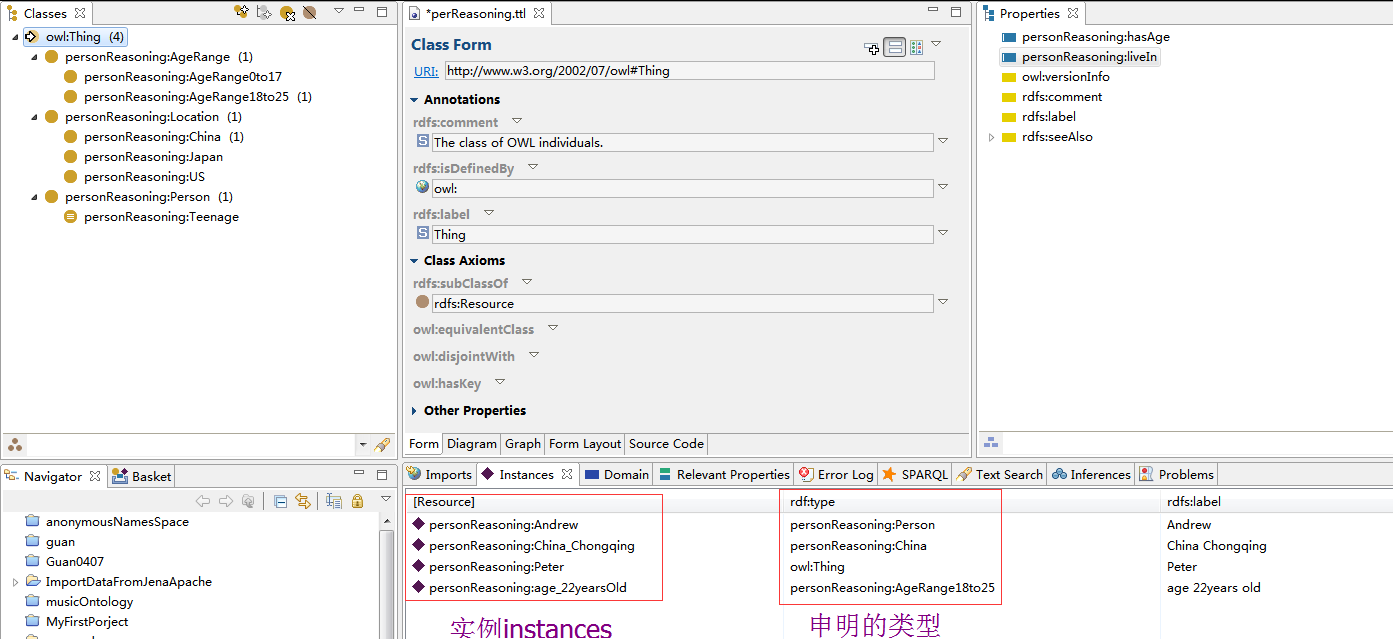


***5.6 Demo1 & Demo2小结***

仔细对比Demo1和Demo2，发现Demo1中使用的hasAge被定义为TBC（TopBraidComposer）中的推理，不支持DataProperty的数值推理，这在Protégé中得到了支持（如Demo2所示）。但由于我们的JenaAPI也不支持Dataproperty这种类型属性的推理，所以在需要使用这种类型的推理时候，建议使用Demo1中的方法，将数据类型归到某个独立的类中去，然后使用ObjectProperty来指向这些独立的数据类（代替内建的数据类型int,float之类的）。

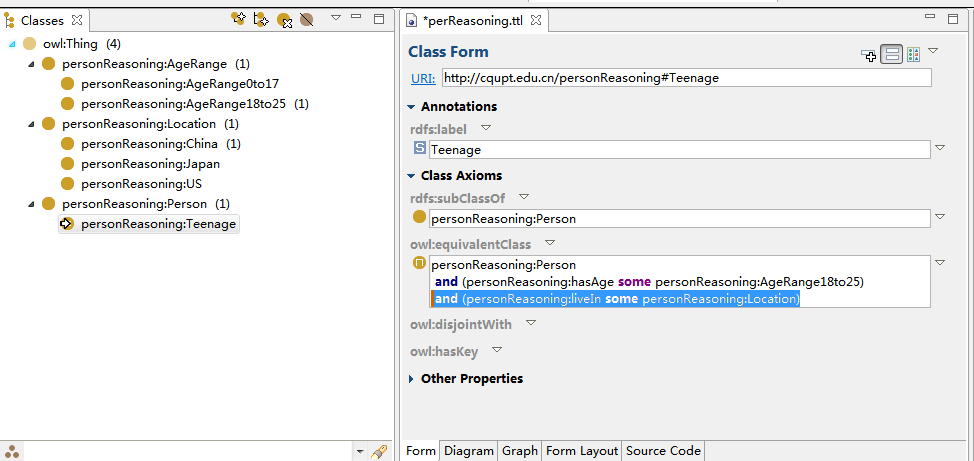
***5.7 Demo3——并列属性定义***

Step1：在Demo1中添加Location（China，US，Japan归为子类）及ObjectProperty属性：liveIn，即：



同时，使用布尔类构造函数 “and（合取/并集）” 为Teenage增添一条约束信息（直接使用曼切斯特语法在白色框内书写），即：

liveIn some Location:



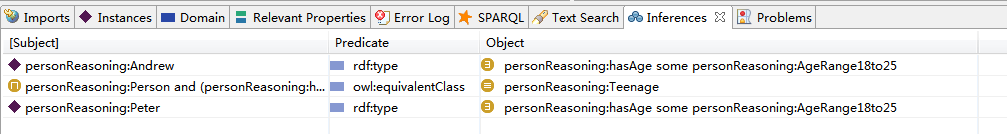
Step 2:推理-1

Andrew has Age 22；type Person.

Peter has Age 22;type Thing.

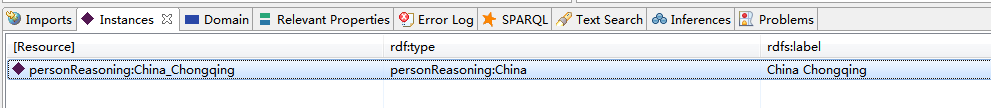
两者都没有liveIn属性及值。

推理结果如下：两者均不为Teenage类型。

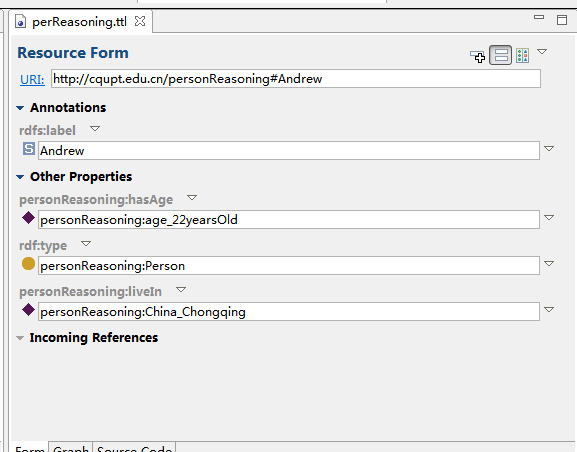


Step 2:推理-2

执行1：创建一个China的“Location”实例，如 “China\_Chongqing”，即：

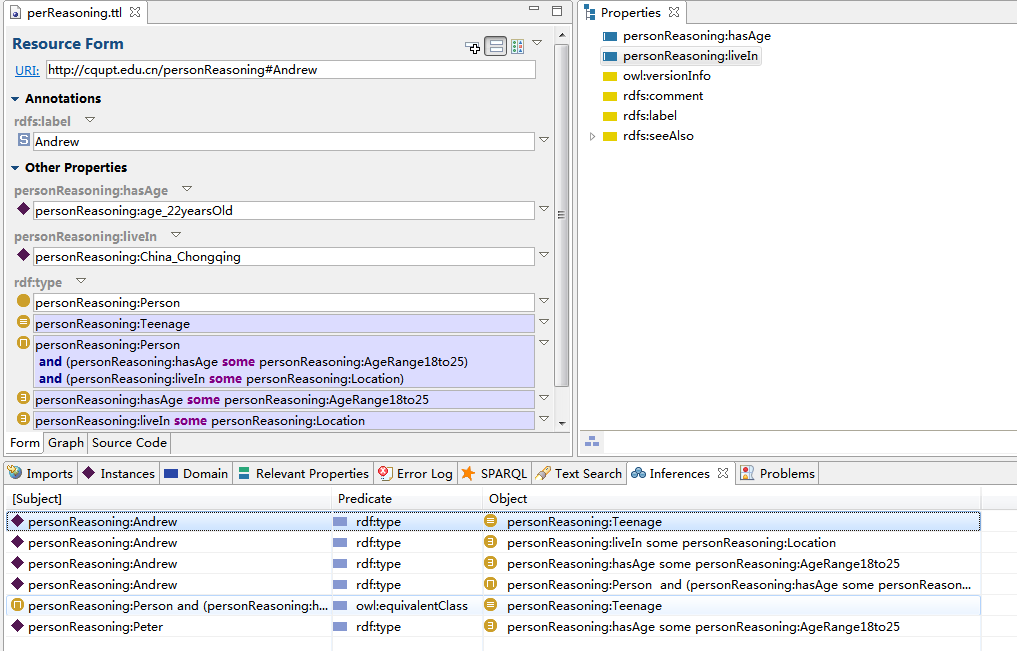


执行2：为Andrew添加liveIn属性及值，即：



Step3：推理-3

在添加了如上所述的信息后，推理得到，Andrew的类型是Teenage，Peter则不是，因为Peter有两个条件没有满足：①rdf.type Person；②liveIn Location：



**6 用SHACL建模**

SHACL(Shapes Constraint Language)是2017年初W3C指定的正式数据建模语言。随着越来越多的人使用来自不同数据源的数据，特别是对于数据集成项目，SHACL为他们提供了一种方法来描述他们正在使用的数据的”shapes”，以便应用程序可以更好地利用这些数据。除了描述哪些属性与哪些类匹配（例如，驱动用户输入表单）之外，SHACL还允许您在数据被应用程序使用时定义数据约束，，可以使用标准化模型而不是使用程序代码更轻松地提高数据质量。

TopQuadrant是开发SHACL标准的积极参与者，其所有产品都支持SHACL标准。和OWL不同，SHACL（和SPARQL一样）以“封闭世界”方式解释模型。这对数据验证以产生预期的结果是必要的。SHACL模型和约束可以与OWL本体一起使用，或者你也可以不使用OWL而完全使用SHACL建模。SPIN是SHACL的关键输入，并且在数据约束方面，SHACL可以被看作是SPIN 2.0。但是，SPIN与SHACL相比具有更加广泛的范围。例如，SHACL不支持能够推理出新事实的规则。SHACL和SPIN可以一同使用来涵盖整个建模和应用的需要。

对于初学者，关于如何使用SHACL在TBC中建模，请参考：

<http://www.topquadrant.com/technology/shacl/tutorial/>

**附录 A：Semantic Web Standards**

***A.1 RDF***

资源描述框架(RDF，Resource Description Framework)是用于表示资源信息的语言，这些资源位于internet架构之上并可以被internet所访问。每个RDF资源都有一个基于统一资源描述符(URI)的唯一ID。URI类似于URL，是一个唯一的符号字符串，但是它不需要解析为实际的位置。RDF具有XML序列化，许多熟悉XML的人会感觉RDF基于XML语法。然而这种对RDF的理解是不正确的。RDF应该根据它的数据模型来理解。RDF数据可以被表示为（序列化为）XML，但是它也可以具有其他序列化语法，例如N3和Turtle。

RDF声明经常被称为“三元组”，因为它包含三部分：主语(rdf:subject)、谓语(rdf:predicate)和宾语(rdf:object)。每一条记录都是一个RDF资源。包含具有相同ID的资源的声明将会被合并。例如，以下三个三元组将会合并到一起：

• flights:FlightDL210 flights:hasDestination geo:London

• geo:London rdf:type geo:City

• :John :livesIn geo:London

以图的形式表示合并的结果为：



图A.1 RDF图—例1

RDF数据应该被理解为一个带有节点和弧线的图，其中节点为主语和宾语，弧线为谓语。一个三元组声明本身可以使用rdf:Statement类的实例来给出一个资源ID。

一个声明中的谓语可以是另一个声明中的主语或宾语，如下图所示。下图中，livesIn是上图中的一个谓语，它还是下图中livesIn rdf:type owl:ObjectProperty声明中的一个主语。



图A.2 RDF图—例2

你可以在Composer中使用Resource Editor中的Graph面板来生成上面的这种图表。

资源ID前面的前缀（如rdf:或者owl:）表示命名空间。在RDF中使用XML命名空间具有一套完整的（有些晦涩）语法规则。Composer为你实现这些规则。作为用户，你只需要为每个本体或者RDF数据集提供一个命名空间。关于命名空间的命名约定可以通过TBC中的help控制面板的“Naming Conventions and Namespaces”获得。（你可以在Help功能的Table of Contents或者通过Help的搜索功能找到）。

虽然RDF包含rdf:type属性，但是RDF数据模型没有提供一种表示RDF数据的自定义模式的方法。也就是说，在RDF中没有类或者类定义的概念。模式由构建于RDF上层的语言—RDFS和OWL提供。RDFS引入了类，这对指定模式至关重要。类是一个个体的集合，这些个体在类中的身份可以由rdf:type声明显式决定—这意味着个体是该类的类型，或者其在类中的身份也可以从其他声明中推理出来。

所有内置RDF语言构造在Composer中都可用。这些可以在Classes视图和Properties视图中看到。使用Classes View Preferences和Properties View Preferences可以配置哪些RDF元素可见。

***A.2 RDFS***

RDFS是RDF的一种模式语言。RDFS是一种相当小的词汇表。除此之外，它定义了本体开发中常用的以下资源：

* rdfs:Class
* rdfs:subClassOf
* rdfs:subPropertyOf
* rdfs:domain
* rdfs:range
* rdfs:Resource

RDFS还定义了4个注释属性以及一些陈旧的元素，例如很少被使用的rdfs:Container。

RDFS词汇表中所有资源在TopBraid Composer中均可使用。它的类和属性分别显示于Classes视图和Properties视图。

和XML Schema不同，RDFS不能被用于验证。它被用于基于本体模式(ontology schema)和给定的（断言的）声明推理出额外信息。一个RDFS本体永远不可能在语义上失效。

当在文本编辑器中创建RDF时，或许（甚至很可能）会犯一些语法错误。使用本体开发工具（例如TopBraid Composer）能够确保没有语法错误。

RDFS和OWL具有形式定义良好的语义。这些语义是根据每个RDFS或者OWL语句所涉及的推理进行定义。“所涉及”是指推理源自RDF中一个断言的陈述。RDFS定义如下：

* rdfs:subClassOf

**If** x rdf:type rdfs:Class1 **and** rdfs:Class1 rdfs:subClassOf

rdfs:Class2, **then** x rdf:type rdfs:Class2

* rdfs:subPropertyOf

**If** x property1 y **and** property1 rdfs:subpropertyOf property2, **then**

x property2 y

* rdfs:domain

**If** Class1 rdfs:domain property1 **and** x property1 y, **then** x rdf:type

Class1

* rdfs:range

**If** Class2 rdfs:range property1 **and** x property1 y, **then** y rdf:type

Class2

***A.3 OWL***

OWL Web本体语言是定义和实例化Web本体的W3C标准语言。OWL词汇表定义在RDFS词汇表之上：

* owl:Class **is a subclass of** rdfs:Class.
* **While RDFS properties are instances** rdf:Property, **OWL introduces**

owl:ObjectProperty **and** owl:DatatypeProperty **which are both subclasses of** rdf:Property.

* **OWL Object properties link two resources within the ontology. For example, the property** livesIn **might link** John (**an instance of the** Person **class**) **with** London (**an instance of the** City **class**).
* **OWL Datatype properties link a resource to an XML Schema Datatype value or an RDF literal. For example**, **the property** hasTelephoneNumber **might link** John to **the string literal** “44-10-1233400”.

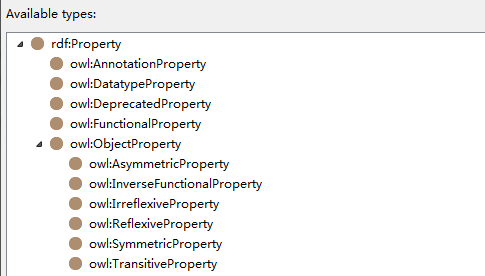
另外，OWL提供了其他描述属性的方式，包括：

* **Functional (any instance using this property can have only one distinct value for the property)**
* **Inverse Functional**
* **Transitive** (**if** partOf is **an** owl:TransitiveProperty **and** x partOf y **and** x

partOf z, **then** x partOf z)

* **Symmetric** (**if** siblingOf is **a**n owl:SymmetricProperty **and** x siblingOf y, **then** y siblingOf x)

所有OWL语言约束在TopBraid Composer中均可用。下图显示了属性的内置OWL类。



图A.3 rdfs:Property的子类

除了全局约束（属性描述），OWL还提供了一个创建指定类的本地约束的词汇表。OWL是一种丰富的语言。本指南没有包含所有的OWL语法。

OWL形式语义规范了如何从本体中创建的声明中得出逻辑结果（推理），例如事实不是字面上存在于本体中，而是由语义引起的。Composer提供对生成OWL推理的支持。这可以通过在Composer中配置使用其中一个内置推理引擎来实现。

***A.4 SPARQL***

SPARQL是用于查询RDF图的一个W3C推荐标准。它的语法和关系数据库查询语言SQL类似，但是它的关注点在于三元组匹配。

SPARQL为不同的目的提供了五种不同的查询变体。

SELECT query—用于以表的形式返回查询结果；

CONSTRUCT query—用于返回RDF三元组；

ASK query—用于为查询提供一个简单的True/False结果；

DESCRIBE query—用于从SPARQL端点提取出一个RDF图。DESCRIBE请求描述特定资源的三元组。SPARQL规范将其留给查询处理器来决定哪些三元组作为对指定资源的描述被返回。这将导致DESCRIBE query实现的不一致，所以这种查询形式不是很受欢迎，但是值得去尝试一下。

UPDATE query—用于向图中插入或者删除一个三元组。

每种查询格式都使用WHERE代码块来约束查询。请参考以下网址获取对SPARQL查询语言的精确描述：

<http://www.w3.org/TR/rdf-sparql-query/>

***A.5 SHACL***

SHACL Shapes Constraint Language是一种根据一组条件验证RDF图的W3C标准。这些条件被以RDF图格式表达的RDF模型和其他构造的方式提供。模型（Shapes）可被用于除验证意外的其他多种目的，包括用户接口的建立、代码的生成和数据的集成。作为一种建模语言，SHACL是一种OWL的替代物。它基于封闭世界语义。

可以将SHACL看作是SPIN 2.0—SPIN标准化的产物。在撰写这个文档的时候，SHACL还处于工作草案状态。