RDF 入门 推荐标准

目录

1. 介绍(Introduction)
2. 关于资源的陈述(Making Statements About Resources)
   2.1 基本概念(Basic Concepts)
   2.2 RDF 模型(The RDF Model)
   2.3 结构化属性值与空白节点(Structured Property Values and Blank Nodes)
   2.4 类型文字(Typed Literals)
   2.5 总结(Concepts Summary)
3. RDF/XML - 一种用于 RDF 的 XML 语法(An XML Syntax for RDF: RDF/XML)
   3.1 基本原理(Basic Principles)
   3.2 简化和组织 RDF URIrefs(Abbreviating and Organizing RDF URIrefs)
   3.3 总结(RDF/XML Summary)
4. 其它 RDF 表达能力（Other RDF Capabilities）
   4.1 RDF 容器（RDF Containers）
   4.2 RDF 集合（RDF Collections）
   4.3 RDF 具体化（RDF Reification）
   4.4 关于结构化值（rdf:value）的更多信息（More on Structured Values: rdf:value）
   4.5 XML 文字（Literal）（XML Literals）
5. RDF Schema - 定义 RDF 的词汇表（Defining RDF Vocabularies: RDF Schema）
   5.1 描述类（Class）（Describing Classes）
   5.2 描述属性（Property）（Describing Properties）
   5.3 理解 RDF Schema 声明（Interpreting RDF Schema Declarations）
   5.4 其它关于 Schema 的信息（Other Schema Information）
   5.5 更丰富的 Schema 语言（Richer Schema Languages）
6. 一些 RDF 的应用：具体领域中的 RDF（Some RDF Applications: RDF in the Field）
   6.1 都柏林核心元数据倡议（Dublin Core Metadata Initiative）
   6.2 PRISM
   6.3 XPackage
   6.4 RSS 1.0：RDF 站点汇总（RSS 1.0: RDF Site Summary）
   6.5 CIM/XML
   6.6 基因本体协会（Gene Ontology Consortium）
   6.7 描述设备性能与用户偏好（Describing Device Capabilities and User Preferences）
7. RDF 规范的其它部分（Other Parts of the RDF Specification）
   7.1 RDF 语义（RDF Semantics）
   7.2 测试用例（Test Cases）
8. 参考文献（References）
   8.1 规范性参考文献（Normative References）
   8.2 参考性资料（Informational References）
9. 致谢（Acknowledgments）

[编辑]

附录

A. 关于 URI 的更多信息(More on Uniform Resource Identifiers (URIs))
B. 关于 XML 的更多信息(More on the Extensible Markup Language (XML))
C. 改动记录(Changes)

[编辑]

1. 介绍(Introduction)

资源描述框架(Resource Description Framework，简称 RDF)是一个用于表达关于万维网(World Wide Web)上的资源的信息的语言．它专门用于表达关于 Web 资源的元数据，比如 Web 页面的标题、作者和修改时间，Web 文档的版权和许可信息，某个被共享资源的可用计划表等。然而，将"Web 资源（Web resource）"这一概念一般化后，RDF 可被用于表达关于任何可在 Web 上被*标识*的事物的信息，即使有时它们不能被直接从 Web 上*获取*。比如关于一个在线购物机构的某项产品的信息（例如关于规格、价格和可用性信息），或者是关于一个 Web 用户在信息递送方面的偏好的描述。

RDF 用于信息需要被应用程序处理而不是仅仅显示给人观看的场合。RDF 提供了一种用于表达这一信息、并使其能在应用程序间交换而不丧失语义的通用框架。既然是通用框架，应用程序设计者可以利用现成的通用 RDF 解析器（RDF parser）以及通用的处理工具。能够在不同的应用程序间交换信息意味着对于那些并非信息的最初创建者的应用程序也是可利用这些信息。

RDF 基于这样的思想：用 Web 标识符（称作*统一资源标识符，Uniform Resource Identifiers* 或 *URIs*）来标识事物，用简单的属性（property）及属性值来描述资源。这使得 RDF 可以将一个或多个关于资源的简单陈述表示为一个由结点和弧组成的*图（graph）*，其中的结点和弧代表资源、属性或属性值。为了让讨论显得尽量具体一些，下面的这组陈述"有一个人由 http://www.w3.org/People/EM/contact#me 标识，他的名字是 Eric Miller，他的电子邮件地址是 em@w3.org,他的头衔是 Dr."可以表示为图 1 所示的图：

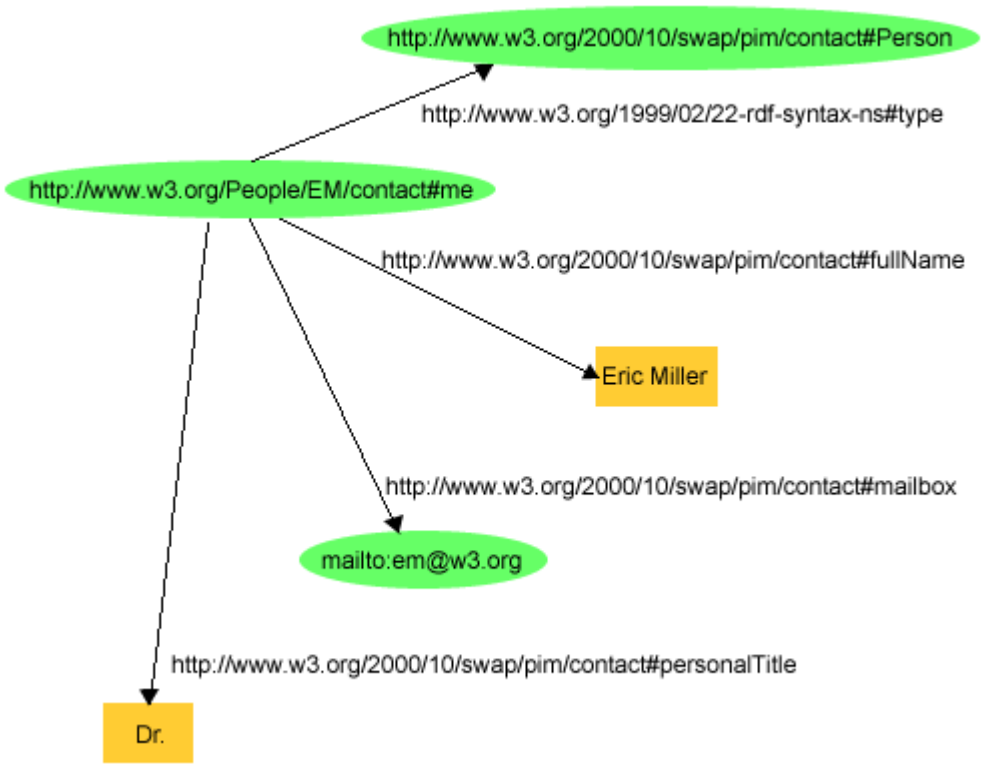

图 1：一个描述 Eric Miller 的 RDF 图

图 1 展示了 RDF 用 URIs 来标识：

个体（individual），例如 Eric Miller，他被标识为 http://www.w3.org/People/EM/contact#me

事物的种类，例如 Person，它被标识为 http://www.w3.org/2000/10/swap/pim/contact#Person

上述事物的属性（property），例如 mailbox，它被标识为 http://www.w3.org/2000/10/swap/pim/contact#mailbox

上述属性的值，例如 mailto:em@w3.org 是 mailbox 属性的值。（RDF 也使用字符串（比如"Eric Miller"）以及其它数据类型中的值（如整数，日期等）作为属性的值）

RDF 提供了一种基于 XML 的语法（称为 *RDF/XML*）用于保存和交换 RDF 图。图 1 所示的 RDF 用 RDF/XML 来书写的话就像下面的例 1 这样：

例 1：一段描述 Eric Miller 的 RDF/XML

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">


<contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">

    <contact:fullName>Eric Miller</contact:fullName>

    <contact:mailbox rdf:resource="mailto:em@w3.org"/>

    <contact:personalTitle>Dr.</contact:personalTitle>

  </contact:Person>


</rdf:RDF>
```

注意：上述 RDF/XML 也包含 URIs，同样地还包含简略形式的属性（比如 mailbox 和 fullName）及各个属性的值（em@w3.org 和 Eric Miller）。

和 HTML 一样，RDF/XML 也是机器可处理的，也使用 URIs，RDF 也可以链接 Web 上任何地方的信息。然而与传统超文本不同的是，RDF URIs 可以引用任何可标识的事物，包括那些不是直接可从 Web 上获取的事物（比如 Eric Miller 这个人）。结果是，RDF 除了能够描述像网页这类事物以外，还可以描述汽车，商业活动，人类，新闻事件等等。此外，RDF 中的属性本身有 URIs，用以准确标识被链接的事物之间的关系。

下列文档都属于 RDF 规范的一部分：

RDF 概念和抽象语法 [RDF-CONCEPTS]
RDF/XML 语法规范 [RDF-SYNTAX]
RDF 词汇描述语言 1.0: RDF Schema [RDF-VOCABULARY]
RDF 语义 [RDF-SEMANTICS]
RDF 测试用例 [RDF-TESTS]
RDF 入门（本文档）

本文档旨在提供一个关于 RDF 的介绍，并描述一些已有的 RDF 应用，以帮助信息系统设计者及应用程序开发者理解 RDF 的特性和如何使用这些特性。特别地，本文档旨对下列这类问题做出回答：

RDF 看起来是怎样的？
RDF 可以表达什么信息？
RDF 信息是如何被创建、访问和处理的？
现有信息如何能够与 RDF 结合？

本文档是一个*非规范性（non-normative）*文档，也就是说它并不是关于 RDF 的权威性的规范。本文档中的例子和其他解释性的资料用于帮助读者理解 RDF，但是它们有可能并不是权威性的和充分完善的。在这些情况下,请参考 RDF 规范中相关的规范性文档。为此，本文档描述了 RDF 规范中的其他文档所担任的角色，并在适当的地方提供了指向规范性文档中相关部分的链接。

需要注意的是，上述 RDF 文档更新并更清晰地解释了先前发布的一些 RDF 规范，资源描述框架(RDF)的模型和语法规范 [RDF-MS]和资源描述框架模式(RDF Schema)规范 1.0[RDF-S]。所以，一些术语、语法和概念已经稍有变化。本文档反映的是在上面的列表中给出的新版本 RDF 规范。因此，对于熟悉旧规范或基于旧规范的教程、入门读物的读者，应留意当前版本的规范与先前版本的一些差异。关于先前版本的 RDF 中出现的一些问题以及它们在当前版本中的解决方案的列表，请参见[http://www.w3.org/2000/03/rdf-tracking/ RDF 问题跟踪（Issue Tracking)。

[编辑]

2. 关于资源的陈述

RDF 是用以提供一种发表关于 Web 资源（如网页）的陈述的简单方法的。这部分描述了 RDF 提供这些能力（描述这些概念的规范标准就是 RDF 概念和抽象语法[RDF-CONCEPTS]）背后的基本思想。

[编辑]

2.1 基本概念

试想一下这样的情形：有一个叫 John Smith 的人创建了某个网页。如果用自然语言（比如英语）来陈述该事实，一种简洁明了的方式是采用下面这种简单陈述的形式：

```
http://www.example.org/index.html has a creator whose value is John Smith
```

强调该陈述的各个部分是为了强调：为了描述某事物的特性，需要某些方法来命名或标识多种事物：

陈述所描述的事物（譬如此例中 John Smith 所创建的网页）
陈述所描述事物的具体属性（property）（譬如本例中的 creator）
陈述所描述的作为该属性（陈述所描述事物对应的）的值的事物（如这个网页的 creator 是谁）。

在上面的陈述中，网页用它的 URL（Uniform Resource Locator，统一资源定位符）来标识。另外，单词“creator”被用来标识事物的属性（property），短语“John Smith”被用来标识作为属性（property）的值的事物（一个人）。

该网页的其他属性（property）可以通过书写其他具有相同形式的英文陈述来描述：用 URL 标识该网页，用单词（或其他表达式）来标识网页的属性（properties）及其值。例如：这个网页的创建日期和所用的语种可以由下列陈述来描述：

```
http://www.example.org/index.html has a creation-date whose value is August 16, 1999

http://www.example.org/index.html has a language whose value is English
```

RDF 是基于这一思想的：被描述的事物具有一些属性 （properties），而这些属性各有其值（values）；对资源的描述可以通过对它作出指定了上述属性及值的陈述（statement）来进行（就像上面例子中的那样）。RDF 用一套特定的术语来表达陈述中的各个部分。确切地说，关于事物（譬如上例中的网页）的陈述中用于识别事物的那部分就叫做主体，而用于区分陈述对象主语的各个不同属性（譬如：作者，创建日期，语种等等）的那部分就叫做谓词，陈述中用于区分各个属性的值的那部分叫做客体。因此，考虑英文陈述：

```
http://www.example.org/index.html has a creator whose value is John Smith
```

这个陈述里的不同部分用 RDF 术语来说就是：

主体是 URL http://www.example.org/index.html

谓词是词“creator”

客体是短语“John Smith”

正如英语是作为（用英语沟通的）人们之间很好沟通工具一样，RDF 是用来做出机器可处理的声明的工具。那么，如果想做出这种适合机器处理的声明，还需要两件事：

一个可用来区分标识一个陈述中的主体，谓词，客体的机器可处理的标识符系统，同时这个标识符系统不会和其他人可能在 Web 上使用的相似的标识符系统混淆；

一种用以表示这些陈述并让这些陈述可在机器间交流的机器可处理的语言；

幸运地是，现有的 Web 体系结构提供了这两个必需的工具。

正如前面所描述的，Web 已经提供了一种形式的标识符—URL（Uniform Resource Locator，同一资源定位符）。在第一个例子中，就采用了一个 URL 来标识 John Smith 所创建的网页。URL 是标识（identify）Web 资源（Web resource）的字符串，这是通过标识资源的首选访问机制来实现的（本质上，即资源的网络“位置”）。然而，对于许多不具有网络地址或 URL 的资源（这一点与网页是不同的），能够记录关于它们的信息也是同等重要。

Web 提供了一套更通用的标识符形式，称为统一资源标识符（Uniform Resource Identifier，URI）。URL 是 URI 的一种具体形式。所有 URI 都具有共同的特征：即不同的人或组织可以彼此独立地创建并使用 URI 来标识事物。但是，URI 并不局限于标识具有网络地址或其他计算机访问机制的资源。实际上，我们可以创建 URI 来引用陈述中需要被标识的任何资源，包括：

网络可访问资源，譬如，一份电子文档、一个图片、一个服务（例如，“洛山矶的今日天气预报”）或是一组其他的资源；

非网络可访问资源，譬如，人、公司、在图书馆装订成册的书籍；

不物理存在的抽象概念，如“作者（creator）”这个概念；

由于上述通用性，RDF 用 URIs 作为其标识机制（用于标识陈述中的主体、谓词和客体）的基础。更准确地说，RDF 使用的是 URI 引用（URI references）[URIS]。一个 URI 引用（或“URIref”）是一个在尾部附加了可选的“片段识别符（fragment identifier）”的 URI。比如，URI 引用（URIref）http://www.example.org/index.html# 由 URI http://www.example.org/index.html 和（由符号#分隔的）的 section2 （片段标识符）组成。RDF URIrefs 可以包含 Unicode[UNICODE]字符（参见[RDF-CONCEPTS]），这就允许在 URIrefs 中使用多种语言。RDF 将“资源（resource）”定义任何可被为 URI 引用（URIref）标识的事物。因此，使用 URIrefs，RDF 实际上可以描述任何事物，并陈述这些事物之间的关系。URIrefs 和片段标识符将在 Appendix A 和[RDF-CONCEPTS]中作进一步的阐述。

为了用一种机器可处理的（machine-processable）方式来表示 RDF 陈述（RDF statements），RDF 采用了可扩展标记语言（Extensible Markup Language）[XML]。XML 被设计成允许任何人来设计他们自己的文档格式，并可用这种格式书写文档。RDF 定义了一个特殊的 XML 标记语言（称为 RDF/XML）来表示 RDF 信息和在机器间交换这些信息。在第一节中有一个关于 RDF/XML 的例子。这个例子（例 1 ）使用了一些诸如<contact:fullName>和<contact:personalTitle>这样的标签来相应地区分 Eric Miller 和 Dr 这些文本内容。这些标签能让那些理解这些标签含义的程序正确地解释文本内容。XML 的内容和标签（除了一些特例）能够包含统一字符编码[UNICODE] 的字符，这就允许了来自各种语言的信息可以被直接显示出来。附录 B 大体上给出了关于 XML 进一步的背景知识。关于 RDF 的 RDF/XML 专用语法在第三节中有更详细的描述，且定义在规范[RDF-SYNTAX]中。

[编辑]

2.2 RDF 模型

在 2.1 节 中阐述了以下内容：RDF 陈述的基本概念，用 URIref 标识 RDF 陈述中涉及的事物的方法，以及用 RDF/XML 作为一种机器可处理的方式来表示 RDF 陈述。基于这些铺垫，这节将描述 RDF 是如何用 URIs 发表关于资源的陈述的。在介绍（第 1 节）中已经说明了，RDF 的基本思想是：表达简单的资源陈述，其中每个陈述都是由主体（subject），谓词（predicate），客体（object）组成的。在 RDF 中，如下的英文陈述：

```
http://www.example.org/index.html has a creator whose value is John Smith
```

可以由 RDF 陈述来表示：

一个主体 http://www.example.org/index.html

一个谓词 http://purl.org/dc/elements/1.1/creator

和一个客体 http://www.example.org/staffid/85740

请注意是如何用 URIrefs 不仅标识了陈述的主体，还标识了谓词和客体，而不分别用单词“作者”和“John Smith”（这样使用 URIrefs 的效果将会在这节的稍后部分做论述）。

RDF 把图中的节点和弧作为陈述的模型。RDF 的图模型已在[RDF 概念]里有过详细说明。在这个表示法（图示法）中，一个陈述可表示为：

一个表示主体的节点;

一个表示客体的节点;

一个由主体节点指向客体节点的表示谓词的弧;

因此以上的 RDF 陈述可以描述为如图 2 所示的那样：

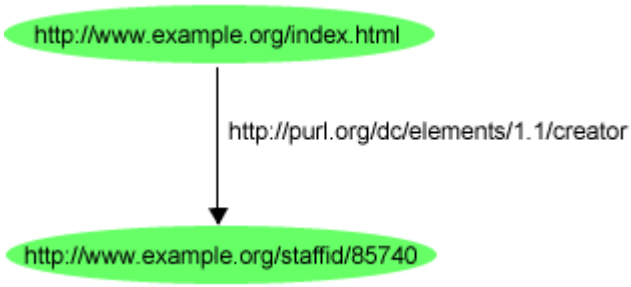

图 2：一个简单的 RDF 陈述

一组陈述相应地被一组节点和弧表示。所以，为了在 RDF 中表示其他的英文陈述：

http://www.example.org/index.html has a **creation-date** whose value is **August 16, 1999**

http://www.example.org/index.html has a **language** whose value is **English**

可以用图 3 所示的图（用合适的 URIrefs 标识属性"creation-date" 和 "language"）：

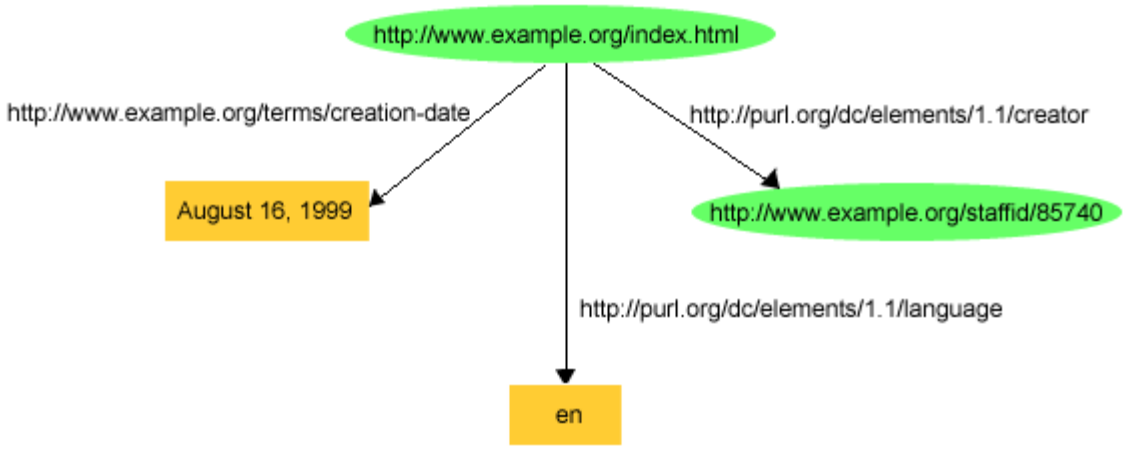

图 3：关于同一资源的多个陈述：

图 3 说明了 RDF 陈述中的客体可以是 URIrefs，也可以是常量值（称作文字），即用于表示某种属性值的字符串。

图 3 显示了：为了表示多种类型的属性值，RDF 陈述中的客体可能是 URIrefs 或字符串表示的常数（称为文字，literal）。（比如在以“http://purl.org/dc/elements/1.1/language” 为谓词的例子中，那个文字就是一个英文的国际标准双字母码。）在 RDF 的陈述中，文字（literal）可能不被用作主体或者谓词。在画 RDF 图时，节点为 URIrefs 的用椭圆来表示，而节点为文字的则用方框来表示（这个例子中的简单字符串文字叫做平凡文字，它与将会在 2.4 节做介绍的类型文字（typed literals）是不同的。在[[RDF-CONCEPTS]中详细说明了各种不同的可用于 RDF 陈述中的文字。平凡文字和类型文字都能容许采统一字符编码[UNICODE]的字符，即允许信息用多种语言描述。）

有时在讨论它们的时候画图不太方便，因此也会用到一个替代的书写陈述的方法，称为三元组。在三元组表示法中，图中的每个陈述都可以写成一个依次为主体，谓词，客体的三元组。如图 3 所表示的陈述用三元组表示法来写就是：

```
<http://www.example.org/index.html&gt; <http://purl.org/dc/elements/1.1/creator&gt; <http://www.example.org/staffid/85740&gt; .

<http://www.example.org/index.html&gt; <http://www.example.org/terms/creation-date&gt; "August 16, 1999" .

<http://www.example.org/index.html&gt; <http://purl.org/dc/elements/1.1/language&gt; "en" .
```

每一个三元组均对应于图中的一条弧，且这个弧的起始节点和终止节点分别是陈述中的主体和客体。和图形表示法不同（倒像是原先的陈述），三元组表示法要求一个节点在它出现的每个陈述中都要有标识。因此，例如， “http://www.example.org/index.html” 在三元组表示法中共出现了三次（在每个三元组中均出现一次），而在图形表示法中只出现了一次。但是，三元组表示法和图示法描述了完全相同的信息，这揭示了一个要点：RDF 的基础是陈述的图模型，而用于表示或描述这个图的表示法则是次要的。

完全的三元组表示法要求写出完整的 URIref（括在尖刮号中），正如上面例子那样，所以导致了在一页中有很多长句。为方便起见，本文档用一种简写法（也在其他的 RDF 规范里使用）来书写三元组。在这种简写法中， 一个不用尖刮号的 XML 限定名（QName）作为一个完整的 URIref 的缩写形式 （QName 会在附录 B 进一步讨论）。一个 QName 包括一个被赋为命名空间 URI 的前缀，其后是一个冒号，然后是个"局部名称"（local name）。由 QName 可以生成完整的 URIref，即将局部名称添加到已经赋了命名空间 URI 的前缀。因此，例如，如果将命名空间 URI“http://example.org/somewhere/”赋值给 QName 前缀 foo，那么 QName “foo:bar”就是 URIref“http://example.org/somewhere/bar”的缩写。在本文档的例子中也会用一些“公认的”QName 前缀（这些前缀无需说明就可使用），定义如下：

```
前缀 rdf:, 命名空间 URI: http://www.w3.org/1999/02/22-rdf-syntax-ns#
```

```
前缀 rdfs:, 命名空间 URI: http://www.w3.org/2000/01/rdf-schema#

前缀 dc:, 命名空间 URI: http://purl.org/dc/elements/1.1/

前缀 owl:, 命名空间 URI: http://www.w3.org/2002/07/owl#

前缀 ex:, 命名空间 URI: http://www.example.org/ (or http://www.example.com/)

前缀 xsd:, 命名空间 URI: http://www.w3.org/2001/XMLSchema#
```

显然，“example”的前缀 “ex:”的变形在需要时也会用在示例中，例如:

```
前缀 exterms:, 命名空间 URI: http://www.example.org/terms/ (作为示例的组织中的词汇),

前缀 exstaff:, 命名空间 URI: http://www.example.org/staffid/ (作为示例的组织中的雇员标识),

前缀 ex2:, 命名空间 URI: http://www.domain2.example.org/ (作为示例的第二个组织中的词汇), and so on.
```

用这种简写法，先前的三元组可以写成：

```
ex:index.html  dc:creator             exstaff:85740 .

ex:index.html  exterms:creation-date  "August 16, 1999" .

ex:index.html  dc:language            "en" .
```

因为 RDF 用 URIref 替代词语来命名陈述中的事物，RDF 称一个 URIref 的集合（特别是为了某个目的集合）为词汇表（vocabulary）。通常，这些词汇表中的 URIrefs 被组织为一个有相同前缀的 QName 的集合。也就是说，一个词汇表中的所有术语都有一个相同的命名空间 URIref，通常这个 URIref（无论是谁控制）定义了这个词汇表。包含在词汇表里的 URIrefs 是通过在公用的 URIref 的末端加上局部名称形成的，这样就构成了一套有着公用前缀的 URIrefs。譬如：正像前面的例子展示的那样，一个组织，比方说是 example.org，可能定义一个由前缀全部为 http://www.example.org/terms/ 的 URIrefs 构成的词汇表，用来表示这个组织在业务中用到的术语（例如：“创建日期”，“产品”等），同时也会定义一个全部由 http://www.example.org/staffid/ 开头的 URIrefs 词汇表来标识这个组织的雇员。RDF 用相同的方法来定义它自己的术语的词汇表，这些术语在在 RDF 中有着特定的含义。RDF 词汇表中的 URIrefs 都以 “http://www.w3.org/1999/02/22-rdf-syntax-ns# ”开头，通常情况下，其 QName 用前缀“rdf:”来表示。RDF 词汇描述语言（将会在 第五节做阐述）定义了另一套都以 http://www.w3.org/2000/01/rdf-schema# 开头的 URIrefs 的术语集合，其 QName 用前缀“rdfs:”来表示。（当一个特定的 QName 前缀以这种方式与一个已给定的术语集相关联的时候，那么这个 QName 前缀有时会用以作为这个词汇表的名称，比如，有人可能说“rdfs: 词汇”）。

使用公用的 URI 前缀提供了一种便捷的方法来组织一套相关的术语集 URIrefs，然而，这仅仅只是一种约定。RDF 模型只认可完整的 URIrefs；它不会去看 URIrefs 的具体内容或使用任何关于它们结构的知识。特别地，RDF 不会仅仅因为 URIrefs 有一个公用的前缀而认定这些 URIrefs 之间有联系（更深入的探讨请看附录 A ）。当 URIrefs 带有不同的前缀时，并没有规定说，这些 URIrefs 就不能被认为属于同一个词汇表。 某个特定的组织、过程（process）或者工具等可以根据自己的需要，来定义词汇。这些词汇的 URIrefs 可以来自于任何其它的词汇，数目不受限制。

另外，有时一个组织将使用词汇表的 URIref 命名空间用作是提供关于该词汇表的详细资料这种 Web 资源所在地的 URL。例如：像著名的 QName 前缀 dc: 将会在本文档的例子中用到，它是和命名空间 URIref http://purl.org/dc/elements/1.1/ 相关联的。事实上，这指在 6.1 节阐述的都柏林核心词汇

表。通过在网页浏览器中访问这个 URIref 命名空间就能获得关于都柏林核心词汇表（明确地说，是一个 RDF Schema）的其他一些信息。然而，这也仅仅只是一种约定。RDF 不会认为每个 URI 命名空间都能确定一个可获取的 Web 资源（更深入的探讨请看附录 B ）。

在这个入门文档的其余部分里，当涉及到一些为特殊目而定义的一套 URIref，比如供 RDF 自身的使用而定义的各种 URIref ，或者是 example.org 定义的用来标识它雇员的一套 URIref，时都将会用到术语“词汇表”。术语“命名空间”只是在特指 XML 命名空间这个语法概念的时候才会使用（或者用来描述 QName 中的 URI 前缀）。

在 RDF 图中可以自由混合来自不同词汇表的 URIrefs。譬如：在图 3 中用到了分别采用了 exterms:,exstaff:,dc:词汇表的 URIref。在 RDF 图中，RDF 也没有限制能用多少个具有同一谓词 URIref 的陈述描述同一个资源。例如：如果资源 ex:index.html 是由约翰·史密斯和其他几名工作人员努力合作创造的话，那么 example.org 可能写以下陈述：

```
ex:index.html dc:creator exstaff:85740 .

ex:index.html dc:creator exstaff:27354 .

ex:index.html dc:creator exstaff:00816 .
```

这些 RDF 陈述的例子开始展现一些使用 URIref 作为 RDF 标识事物的基本方式的优势所在。譬如：在第一个陈述中，不用字符串“John Smith”来作为网页的制作者，而是把一个 URIref（使用基于他的雇员号码的 URIref）http://www.example.org/staffid/85740 赋予给他。这样使用 URIref 的一个优点就是陈述主体可以被更加精确的标识出来。就是说，这个网页的制作者不是字符串“John Smith”，也不是数以千计的名叫 John Smith 的人中的一个，而是与那个 URIref（且不管是谁创建了定义了这种关系的 URIref）相关的那个特殊的 John Smith。而且，因为一个指向 John Smith 的 URIref，他就成为一个成熟的资源，并且仅仅通过增加其他主体为 John 的 URIref 的 RDF 陈述，就可以记录他的一些其他信息。例如：图 4 展示了可给出关于 John 的姓名和年龄的一些陈述。

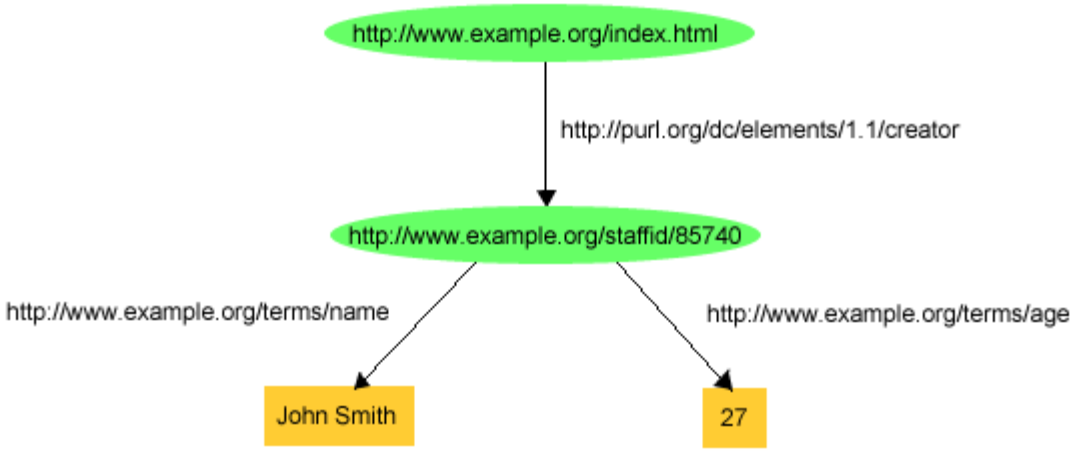

图 4：更多关于 John Smith 的信息

这些例子也说明了 RDF 在 RDF 陈述中用 URIref 作为谓词。就是说，RDF 使用 URIrefs 标识属性，而不是使用像“creator”或者“name”那样的字符串(或词组)。用 URIref 来标识属性的重要性是基于很多原因的。第一，它可以把一个人用的属性和其他人用的属性区别开来，尽管他们可能用相同的字符串来表示属性。例如：在图 4 中的例子，example.org 使用"name"想要使写出的某人的全名作为一个字符串文字（譬如：“John Smith”），但是其他人可能想要使"name"代表某些不同的事物（譬如：在一个程序段中的变量名）。当一段程序遇到“name”作为一个 Web 上的属性标识符（或者当合并来自多个数据源的数据）时将不一定能区分这些使用方法。但是，如果 example.org 用“http://www.example.org/terms/name”当作它定义的“name”的值，并且其他人用“http://www.domain2.example.org/genealogy/terms/name”当作他们定义的“name”的值，那么显然不同的“name”包含着不同的值（即使一个程序不能自动确定它们的具体含义）。此外，使用 URIrefs 来区分属性能使属性被看成是资源本身。因为属性也是资源，仅仅通过增加主体为属性的 URIref 的 RDF 陈述，就能记录关于属性的信息（譬如：example.org 所用的“name”属性的含义的英文描述）。

用 URIref 作为 RDF 陈述的主体，谓词，客体支持了 Web 上的共享词汇表的使用和发展，因为人们可以发现并开始使用已经在用的词汇表来描述事物，这反映了人们对那些概念的共享理解。例如：在三元组“ex:index.html dc:creator exstaff:85740 ”中：

```
ex:index.html   dc:creator   exstaff:85740 .
```

当谓词“dc:crearot”完全展开为一个 URIref 时，就明确的指向了 Dublin Core 元数据属性集（一个广泛使用的描述各种各样信息的一套属性集，将在 6.1 节中做深入的讨论）中的“creator”的属性。这个三元组的作者有效地说明了网页（由 http://www.example.org/index.html 所标识）和网页的作者（一个独一无二的人，由 http://www.example.org/staffid/85740 所标识）之间的关系正是一个由 http://purl.org/dc/elements/1.1/creator 标识的概念。另一个熟悉 Dublin Core 词汇的人，或是查明了“dc:creator”确切含义（通过在 Web 上查找的它的定义）的人，将会明白这个关系的含义。另外，在这种理解的基础上，在处理含有谓词“dc:creator”的三元组时，人们就能编写出行为与这个含义一致的程序来。

当然，这有赖于越来越普遍地使用 URIref 而不是用文字来指代事物；譬如：用 URIref 像“exstaff:85740”和“dc:creator”来代替字符串文字像“John Smith”和“creator”。即使是那样，RDF 对 URIre 的使用仍不能解决所有的标识问题，因为例如：人们仍然能够用不同的 URIref 来指代同一个事物。由于这个原因，尽量使用现有的词汇表（比如 Dublin Core）的术语，而不发明可能与其他词汇表中术语重复的术语，这是个不错的想法。 正如在第 6 节描述的应用展示的那样，在特定领域里适用的词汇表一直都在完善中。然而，即使同义词被建立了，在普遍可访问的“Web 空间”中使用不同的 URIref，即提供了在这些 URIref 中识别等价关系的机遇，也提供了转向使用通用词汇表的机遇。

另外，重要的是，要区分 RDF 本身赋予 RDF 陈述中的词汇（比如在先前例子里的 dc:creator）的含义与人们（或人编写的程序）可能赋予这些词汇的其他外部定义的含义。作为一门语言，RDF 直接定义的只有主体，谓词，客体三元组的图示语法，在 rdf:词汇表中的 URIrefs 的某些含义，和稍后会做介绍的某些其他概念。这些事物在[RDF-CONCEPTS]和[RDF-SEMANTICS]有规范的定义。不过，RDF 没有定义在 RDF 陈述中使用的其他词汇表中的术语的含义，比如：dc:creator。特定的词汇表会被创建且其中的 URIref 会被赋予特定的含义，但这是在 RDF 之外的。使用了这些词汇表中的 URIref 的 RDF 陈述可能会把那些术语的特定含义传达给熟悉这些词汇表的人，或是处理这些词汇表的 RDF 应用程序，而不会把这些含义传达给不是特意处理这些词汇表的通用 RDF 应用程序。

例如，人们可以给一个三元组如：

ex:index.html dc:creator exstaff:85740 .

赋予一定的含义，这是基于单词“creator”作为 URIref“dc:creator”的一部分出现所表示的含义，或者基于在他们理解了“dc:creator”在 Dublin Core 词汇表中的确切定义。 不过，就通用的 RDF 应用程序而言，这个三元组和下面的三元组在内在的含义上是一样的：

```
fy:joefy.iunm   ed:dsfbups   fytubgg:85740 .
```

与此类似，任何可能在 Web 上找到的描述“dc:creator”含义的自然语言文本都无法为一个通用的 RDF 应用程序提供其直接可用的额外的含义信息。

当然，来自一个特定词汇表的 URIrefs 可以在 RDF 陈述中被使用，尽管给定的应用程序可能不能把任何特定的含义赋予他们。例如，通用的 RDF 软件会识别出上述表达式是一个 RDF 陈述，其中“ed:dsfbups”是一个谓词，等等。它不会把词汇表开发人员赋予给一个 URIref（比如 ed:dsfbups）的任何特定含义赋给这个三元组。此外，基于他们对一个给定的词汇表的理解，人们仍可以编写出一个与这个词汇表中的 URIref 的特定含义一致的 RDF 应用程序，尽管这个含义对不是以这种方式编写的 RDF 应用程序是无法理解的。

结果是：RDF 提供了一种发表更易被应用程序处理的陈述的方法。一个应用不能真正理解这些陈述，就如当一个数据库系统处理一个查询语句如：SELECT NAME FROM EMPLOYEE WHERE SLALRY > 35000 的时候，数据库系统对像“employee”或“salary”这类的词汇的理解一样。但是， 如果一个应用程序编写的很合理，那么它处理 RDF 陈述时就好像它确实理解它们一样，这正像是一个数据库系统和它的程序并没有理解什么是“employee”和“payroll”却能在处理雇员和薪水的数据信息时做有用的工作一样。例如：一个人能搜寻 Web 找到全部书评并且为每本书创建一个平均等级。 然后，这个人就可以把这些关于书的信息放到 Web 上。 而另一网站能获取这些书平均等级的列表并且创造一个"评价最高的十本书"页面。 这里，一个关于

书等级的共享词汇表的可用性和及其使用，以及用于标识那些书的一组共享的 URIref，允许个人建造一个在 Web 上可相互理解和用处日益广泛（或说是做了其他贡献那样）的关于书的“信息库”。相同的原则适用于人们每天在 Web 上创建的关于数以千计的主题的大量信息。

RDF 陈述和很多其他信息记录格式类似，像：

在一个数据处理系统里，一个简单的记录或是目录清单中的实体，

一个简单的关系数据库的行，

形式逻辑的简单断言，

并且，这些格式的信息都能看成 RDF 陈述，这使得 RDF 能用于集成来自多个数据源的数据。

[编辑]

2.3 结构化的属性值与空节点

需要记录的信息，如果用简单的 RDF 语句的形式来描述就足够了，那么，一切都将变得很简单，但是，大多数现实世界中的数据，至少表面看起来，要比简单的 RDF 语句所能描述的形式复杂得多。例如，在最初的那个例子中，用来记录创建网页的是一个简单的数据：creation-date 属性，这个属性的值是简单的字符型，但是，假设这个 creation-date 属性的值需要分别记录年、月、日，或者，在描述 John Smith 的个人信息的情况下，我们来考虑 John 的地址，整个的地址可以被写作一个简单的字符串，或者一个三元组"1501 Grant Avenue, Bedford, Massachusetts 01730"。

```
exstaff:85740   exterms:address   "1501 Grant Avenue, Bedford, Massachusetts 01730" .
```

然而,设想一下约翰的地址需要记录为由一个街道，城市，州和邮政编码组成的结构，这些在 RDF 中是如何做到的？

像这样的结构化的数据信息在 RDF 中是通过以下方式描述的：把被描述的事物聚集体（比如：John Smith 的住址）看成一个资源，然后发表关于这个新资源的陈述。所以，在 RDF 图中，为了将 John Smith 住址分解成它的各个组成部分，一个用来描述 John Smith 住址这一概念的新节点就随之产生了，并用一个新的 URIref 来标识，如 http://www.example.org/addressid/85740 （可缩写为 exaddressid:85740）。把这个节点作为主体，RDF 陈述（附加的弧和节点）可用来描述附加的信息，如图 5 所示：

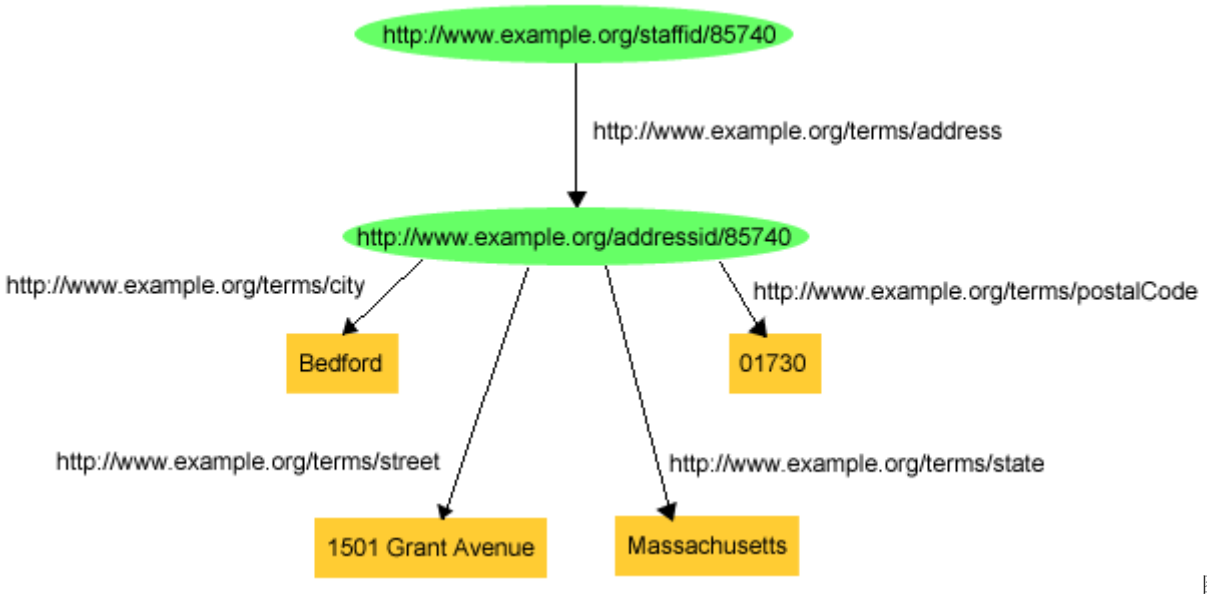

图 5：分解 John 的住址：

相应的三元组表示如下：

```
exstaff:85740       exterms:address        exaddressid:85740 .

exaddressid:85740   exterms:street         "1501 Grant Avenue" .

exaddressid:85740   exterms:city           "Bedford" .

exaddressid:85740   exterms:state          "Massachusetts" .

exaddressid:85740   exterms:postalCode     "01730" .
```

这中描述 RDF 结构化数据的方法会产生很多的“中间的”URIrefs，比如像描述聚集体概念（如 John's address）的 URIref exaddressid:85740。这些概念可能从来不会被从 RDF 图的外部引用，因此可能不需要“通用的”标识符。另外，在用于描述一组陈述的图 图 5 中，用来标识“John Smith's address”的 URIref 并不是真正需要的，因为这个图可以简单地标识为如图 6 所示：

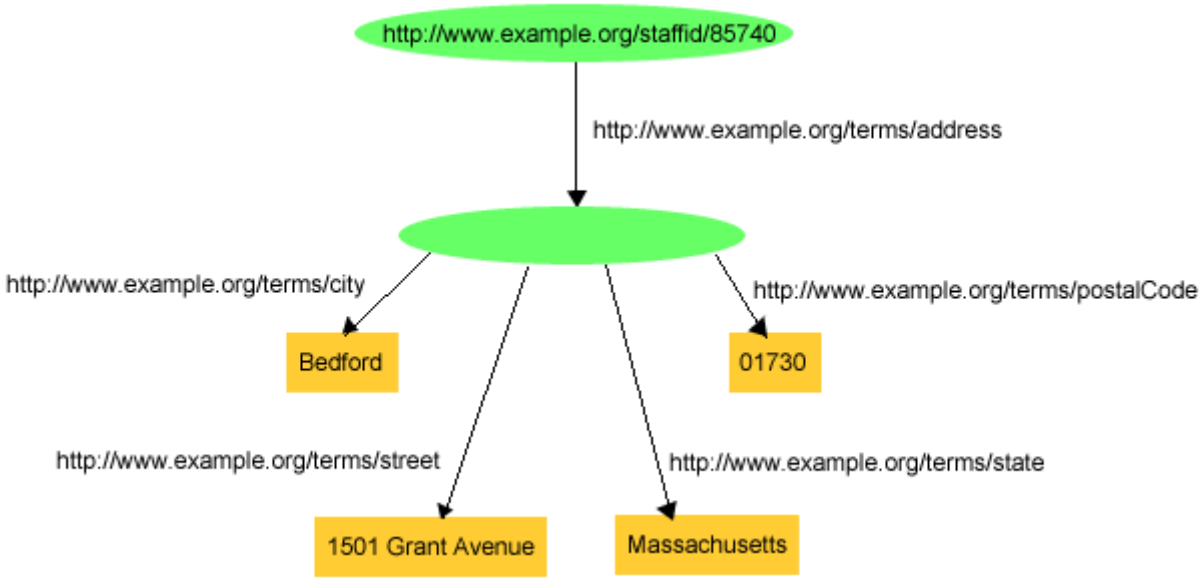

图 6：使用一个空节点

图 6 是一个近乎完美的 RDF 图，它使用了一个没有 URIref 的节点来表示“John Smith's address”这一概念。这个空节点虽然没有 URIref，但表达了它应该表达的含义，因为这个空节点本身提供了图中各个部分之间必需的连通作用（空节点在[RDF-MS]中被称作匿名资源（anonymous resources））。然而，为了把这个图表示为三元组的形式，就需要一个某种形式的能清楚表示那个空节点的标识符。这里试着写出了与图 6 所示的内容相应的三元组：

```
exstaff:85740   exterms:address        ??? .

???             exterms:street         "1501 Grant Avenue" .

???             exterms:city           "Bedford" .

???             exterms:state          "Massachusetts" .
```

```
???             exterms:postalCode      "01730" .
```

“???”出现的地方正是出空节点出现过的地方。因为一个复杂的图包含的空节点可能会不只一个，所以需要一种区分在图的三元组表示法中出现的不同空节点的办法。因此，三元组使用空节点标识符，以“_:name”的形式来表示空节点。例如：在这个例子中，空节点标识符“_:johnaddress”可以用来表示空节点，那么相应的三元组可以写成如下的形式：

```
exstaff:85740   exterms:address         _:johnaddress .

_:johnaddress   exterms:street          "1501 Grant Avenue" .

_:johnaddress   exterms:city            "Bedford" .

_:johnaddress   exterms:state           "Massachusetts" .

_:johnaddress   exterms:postalCode      "01730" .
```

在表示一个图的三元组中，图中每个不同的空节点都被赋予一个不同的空节点标识符。与 URIref 和文字不一样，空节点标识符并不被认为是 RDF 图的一个实际组成部分（这从图 6 所示的就可以看出来，并且注意在图中空节点并没有空节点标识符）。空节点标识符仅仅是在把 RDF 图表示成三元组形式的时候，用来表示图中的空节点的（并且区分不同的空节点）。空节点标识符只是在用三元组表示单一的图的时候才有意义（两个有相同空节点数目的 RDF 图可能会独自地使用相同的空节点标识符来区别那些空节点，但是如果假设不同 RDF 图中具有相同标识符的空节点是相同的就不对了）。如果希望图中的一个节点需要从图的外部来引用，那么就应该赋予一个 URIref 值来标识它。最后，因为空节点标识符表示的是（空）节点而非弧，所以在一个图的三元组表达式中：空节点标识符只能出现在三元组主体和客体的位置上；不能出现在谓词的位置上。

这一节的开头部分记录了聚集体结构（比如 John Smith's address）可以通过如下的方式来描述：聚集体被作为一种资源描述，然后发表关于这个新资源的陈述。这个例子阐明了 RDF 的一个重要方面：RDF 只能直接表示二元关系，例如：John Smith 和描述他住址的文字（literal）之间的关系。要描述 John 和由这个地址的每个单独部分组成的组的关系的时候，就要涉及到处理一个 N 元（n-ary）关系（在这里 N＝5），其中这五元分别是 John，街区（street），城市（city），州（state），和邮政编码（postal code）。为了要在 RDF 中直接的描述这种结构（例如：把地址看作是由街区（street），城市（city），州（state），和邮政编码（postal code）这四个部分构成的一个组），就必须把这个 N 元关系分解为一组二元关系。空节点提供了一种完成这个任务的方法：对于每一个 N 元关系，选择其中的一元（ participant）作为这个关系的主体（比如 John），创建一个空节点来描述其余的关系（比如 John's address），这个 N 元关系的其他元（比如 city）则被描述成由空节点标识的新资源的各个单独的属性。

有些资源可能没有 URI，空节点也同样提供了一种更准确的发表关于这些资源的陈述的方法，但这是通过和那些有 URI 的资源的关系来描述的。例如：当发表一个关于某人（比如 Jane Smith）的陈述的时候，可能会很自然的想到要用基于这人的 email 地址的 URI 作为她的 URI（mailto:jane@example.org ），但是，这种方法可能会导致一些问题。如果需要同时记录关于 Jane（比如她目前的实际住址）和她的邮箱（比如，它所在的服务器）两者的信息，那么，还用基于她邮箱地址的 URI 作为她的 URIref 的话，就会很难区分表述的到底是 Jane 还是她的邮箱。同样的问题也会出现在下面这种情况：当一个公司用它公司的网页 URL（比如：http://www.example.com/）作为自己的 URI 的时候。同样地，当需要同时记录关于网页（比如：是谁于何时创建的）和这个公司两者的信息时，如果还用“http://www.example.com/”作为两者的标识符，就会导致很难区分谁是真正意义上的主体。

这些问题出现的根本原因就是：用 Jane 的邮箱来代表 Jane 是不正确的，因为 Jane 和她的邮箱根本就是两码事，因此她和它应该区别对待。当 Jane 自己没有 URI 时，空节点提供了一条为这种情形更正确的建模方法：Jane 可以由一个空节点表示，并且用这个以“exterms:mailbox”为属性的空节点作为陈述的主体，且用 URIref“mailto:jane@example.org” 作为它的这个属性的值。这个空节点也可以用以“exterms:Person”为值的一个“rdf:type”属性来表述（类型（type）将在以后章节做更详细的讨论），或者是其他有用的描述性信息，正如下列三元组所示的：

```
_:jane    exterms:mailbox    <mailto:jane@example.org&gt; .

_:jane    rdf:type           exterms:Person .

_:jane    exterms:name       "Jane Smith" .

_:jane    exterms:empID      "23748"  .

_:jane    exterms:age        "26" .
```

（注意在第一个三元组中“ mailto:jane@example.org”写在两个尖括号里面，这是因为“mailto:jane@example.org” 在 mailto URI 模式中是个完整的 URIref，而不是一个 QName 缩写，在三元组表示法里一个完整的 URIref 必需写在一对尖括号内。）

这些三元组准确地说明了：“有一个资源，其类型为 exterms:Person，其电子邮箱是由“mailto:jane@example.org” 来标识的，其名字是 Jane Smith，等等”。就是说，空节点可以读作“有一个资源”。以这个空节点作主体的陈述就提供关于这个资源特性方面的信息。

实际上，在这些情况下用空节点替代 URIref 并不会在信息处理方面带来太多改变。例如：人们知道一个 email 地址唯一确定在 example.org 里的某人（特别地，如果这个地址不能被重用的话），那么，这个 email 地址会被用来聚集来自多个数据源的关于这个人的信息。尽管这个 email 地址不是这人的 URI。在这种情况下，如果在 Web 上找到了一些描述一本书的 RDF，并且给出了作者的联系资料如 mailto:jane@example.org，那么把这条新信息和先前的一套三元组结合起来，然后推断作者的名字是 Jane Smith，这样做就可能是合理的。这种说法的关键有点像"这书的作者是 mailto:jane@example.org"是“这书的作者是邮箱是 mailto:jane@example.org 的某人"的一种典型的缩写一样。用一个空节点来表示这个“某人”是一种更准确地来表示真实世界的方法。（顺便说一下，一些基于 RDF 的模式语言特别指出允许某些属性可以作为它们描述的资源的”唯一标识符“。这些将在 5.5 节做更深入的探讨。）

这种使用空节点的方法也可以帮助人们避免文字（literals）在可能不适当情况下的使用.例如：当描述 Jane 的书而又缺少一个识别作者的 URIref 时，出版商就可能这样写（使用出版商自己的 ex2terms:词汇表）：

```
ex2terms:book78354   rdf:type          ex2terms:Book .

ex2terms:book78354   ex2terms:author   "Jane Smith" .
```

但是，书的作者却不是字符串“Jane Smith”，而是一个名叫 Jane Smith 的人。于是，当出版商用一个空节点时相同的信息就可能会被更准确的表达出来：

```
ex2terms:book78354   rdf:type          ex2terms:Book .

ex2terms:book78354   ex2terms:author   _:author78354 .

_:author78354        rdf:type          ex2terms:Person .

_:author78354        ex2terms:name     "Jane Smith" .
```

这实质上说，“ex2terms:book78354”的类型是“ex2terms:book”，并且它的作者是类型为“ex2terms:Person”的一个资源，作者的名字是“Jane Smith”。当然，在这个特例中，出版者为了要鼓励对作者的外部引用，可能对作者指派他自己的 URIrefs 而不是使用空节点识别他们。

最后，上面的例子给出了 Jane 的年龄是 26 说明了这么一件事实就是：有时一个属性的值可能表面上看来很简单，但实际上可能会更复杂。在这种情况，Jane 的年龄实际上是 26 岁，但是单位量（岁）没有明确给出。当确信某个访问文本中属性值的人由此能推定所使用的单位量时类似的单位量经常被省略。但是，在 Web 上无际的文本海洋里，这样的“确信”通常是不可靠的，例如：一个美国地址也许给定一个重量单位——磅，但是某个从外部访问这个数据的人可能会推想给定的单位是千克。总的来说，对于单位或是类似的信息应该经过缜密的考虑然后进行明确的表述。这个问题将会在 4.4 节做更深入的探讨，在这节中将会讲述能把这样的信息表述成构造值（structured values）是 RDF 特色之一，同时也会涉及到另外一些用于表述这类信息的技术。

[编辑]

2.4 类型文字

上一节讲述了怎样处理如下这种情况：用平凡文字表示的属性值分解为能表示那些文字每个单独部分的构造值。例如：如果要记录一个网页的创建日期的话，这种方法用一个专门的平凡文字作为它的值，这个值会以一个由年，月，日组成的结构体作为资料的各个部分，并分别用平凡文字来表示相应的值，就取代了通过创建一个单独 exterms:creation-date 属性来记录一个网页创建日期的方法。但是，到目前为止，所有可以在 RDF 中作为客体的常量值都用这些平凡（非类型）文字表示，甚至当意图是用数字（例如一个年份属性或是年龄属性的值）或者其他更特殊的值当作属性值时。

例如，如图 4 所示的一个记录关于 John Smith 资料的 RDF 图，这个图中记录的 John Smith 的属性“exterms:age”值为平凡文字“27”，正如图 7 所示的那样：

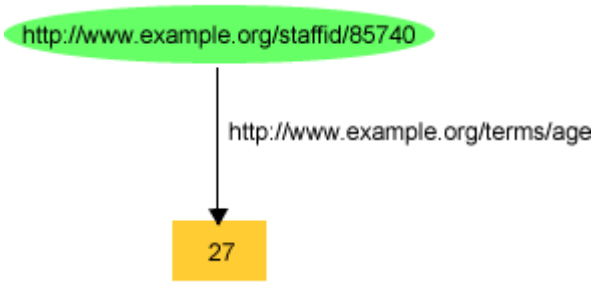

图 7：表示 John Smith 的年龄

在这个例子中，假设的组织 example.org 可能想要用一个数字来表示“27”，而不是用由“2”和后面的“7”组成的字符串来表示（因为文字表示了一个属性“age”的值）。但是，图 7 并没有明确指出“27”应当被解释成一个数字。同样地，example.org 可能也想把“27”表示成一个十进制数（比如，一个值为二十七的十进制数），而不是表示成一个八进制数（比如，一个值为二十三的八进制数）。然而，同样地，图 7 也没有明确指出这些信息。一些特殊的应用程序可能会在编码时把属性 exterms:age 的值解释成十进制数，但这意味着对这个 RDF 的正确解释会依赖 RDF 图中没有明确提供的信息，因而还依赖一些其他需要解释这个 RDF 的应用程序不一定能获取的信息。

在程序设计语言和数据库系统里通常都会提供一个关于如何解释文字的附加信息：即给文字关联一个数据类型，比如在这个例子里，数据类型是小数或者是整数。一个能理解数据类型的应用程序就会知道，比如，把文字“10”表示成数字十，还是数字二，还是由字符“1”和后面的字符“0”组成的一个字符串，取决于指定的数据类型是整数，二进制数，还是字符串。（虽然这本入门文档不会详细说明关于数据类型的概念，但在 2.3 节的结尾处已经提到过更多可以用来包含单位信息的专门的数据类型，比如：一个为 intergerYears 数据类型。）在 RDF 中，类型文字 是用来提供这类信息的。

一个 RDF 类型文字是通过把一个字符串与一个能确定一个特殊数据类型的 URIref 配对形成的。结果在 RDF 图中的一个文字节点就是这个配对。类型文字表示的值就是把指定的字符串赋值给指定的数据类型的值。例如，使用一个类型文字，John Smith 的年龄将会被表述成在三元组中使用的整数 27：

```
<http://www.example.org/staffid/85740&gt; <http://www.example.org/terms/age&gt; "27"^^<http://www.w3.org/2001/XMLSchema#integer&gt; .
```

或者，使用 QName 来简化 URI 则可写成：

```
exstaff:85740  exterms:age  "27"^^xsd:integer .
```

或者，如图 8 所示的：

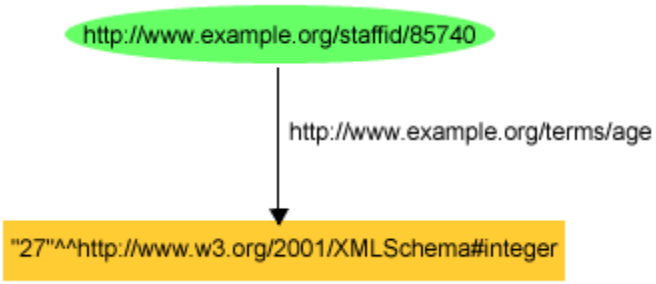

图 8：一个表示 John Smith 年龄的类型文字

同样地，如图 3 所示它表述了关于一个网页的信息资料，网页的 exterms:creation-date 属性值写成了平凡文字“August 16,1999”。然而，如果使用一个类型文字，那么这个网页的创建日期就可以明确的表述成 1999 年 8 月 16 号（August 16, 1999），用三元组表示就是：

```
ex:index.html  exterms:creation-date  "1999-08-16"^^xsd:date .
```

或正如图 9 所示的那样：

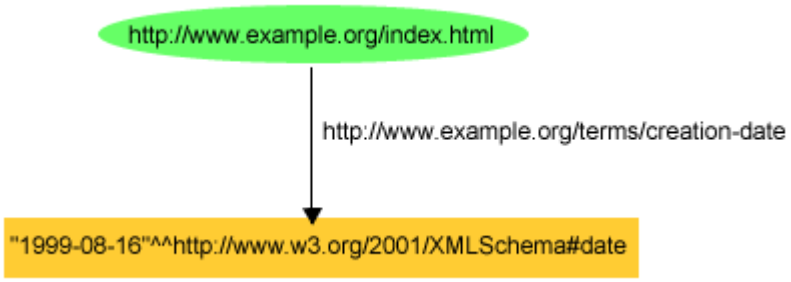

图 9：一个表示一个网页创建日期的类型文字

不像典型的程序设计语言和数据库系统那样，RDF 没有自己的内置数据类型，比如，整型，实型，字符型，或者日期类型。但是，RDF 类型文字为一个已给定的文字提供了一种简单的方法来指出应该用什么样的数据类型来说明它。在类型文字中使用的类型在外部定义，并且由它们的数据类型 URI 来确定。（有一个例外，RDF 用 URIref“rdf:XMLLiteral”定义了一个内置数据类型用来把 XML 内容表示成一个文字值。这个数据类型定义在 [RDF-CONCEPTS]，并在 4.5 节讲述的它的使用方法。）例如：图 8 和图 9 的例子都使用了整数的数据类型和来自 XML 数据类型的数据，其中，在 XML 模型的第二部分：数据类型 [XML-SCHEMA2]中定义了 XML 数据类型。用这种方法的优点之一便是它赋予了 RDF 直接表述信息的机动灵活性，因为在表述那些来自不同信息源的信息时无需进行信息源和 RDF 本身的数据类型集之间的转换。（当在不同的数据类型集系统之间移动信息时，还是要进行类型转换的，但当信息在 RDF 中移入或移出时，RDF 无需额外的数据类型转换。）

正如在 RDF 概念（RDF Concepts）和抽象语法（Abstract Syntax）[RDF-CONCEPTS]中叙述的那样，RDF 数据类型概念是基于 XML 数据类型 [XML-SCHEMA2]的概念框架上的，这种概念框架定义了包含以下部分的数据类型：

一个要用数据类型文字表示的值的集合，被称作值空间（value space）。例如，XML 模型数据类型 xsd:date 的值空间就是日期的集合。

一个数据类型用来表示它的值的字符串的集合，被称作是词法空间（lexical space）。词法空间决定了哪一种字符串能够合乎规则地被用来表示这种数据类型的文字。例如：数据类型 xsd:date 定义了“1999-08-16”是书写这种类型文字的一种合乎规则的方法（相反的如果写成“August 16,1999”则可认为是不合乎规则的）。正如在[RDF-CONCEPTS]中定义的，一种数据类型的词法空间是一个双字节字符[UNICODE]串集，它允许直接表示来自很多语言的信息。

一个从词空间到值空间的“词－值（lexical-to-value）映射”.这决定了这样的一个字符串的值：一个为表示这种特殊数据类型的取自词法空间里的一个特定的字符串。例如：一个映射到数据类型 xsd:date 的词－值映射，决定了对于这种数据类型，字符串“1999-08-16”表示日期“August 16, 1999”。词－值映射是个重要因素因为相同的字符串可能为不同数据类型表示不同的值。

并不是所有数据类型都适和在 RDF 中使用。一种适和在 RDF 中使用的数据类型必须符合上述的概念框架。主要意思就是：对于一个特定的字符串，数据类型必须明确地说明这个字符串是否在它的词法空间内，以及这个字符串在它的值空间里所表示的值。例如：基本的 XML 数据类型（比方说是 xsd:string，xsd:boolean，xsd:date 等等）就适合用在 RDF 当中。然而，一些 XML 内建数据类型就不适合用在 RDF 当中。例如：xsd:duration 没有一个良式定义（well-defined）的值空间，并且 xsd:QName 需要一个封装的 XML 文本文档。到目前为止所认为适合和不适合用在 RDF 中的 XML 数据类型的清单已经在[RDF-SEMANTICS]中给出了。

因为一个特定类型文字表示的值是由类型文字的数据类型决定的，且 RDF 没有定义任一种数据类型(rdf:XMLLiteral 是例外)，对一个出现在 RDF 图中的类型文字的实际解释（例如：决定它所表示的值）必须由那些编写成能正确处理 RDF 和类型文字的数据类型的软件完成。也就是说，这种软件必须能处理一种 RDF 的扩展语言，即把 RDF 和数据类型都作为其内置词汇表的一部分。这使得关于哪种将会在 RDF 软件中普遍用到的数据类型的争论日益激烈。通常，那些在[RDF-SEMANTICS]罗列出的适合在 RDF 中使用的 XML 模型数据类型在 RDF 中有着“首屈一指”的地位。正如已注明的，图 8 和图 9 的例子使用了 XML 数据类型中的一些数据类型，并且这本入门书也将在类型文字的其他例子中使用这些数据类型（另外，XML 数据类型已经被赋值为用来查阅它们的 URIref 了，这一点在[XML-SCHEMA2]有详细的说明）。比较起其他的数据类型来这些 XML 数据类型并没受到区别对待，但人们期望它们能被广泛的应用，因此它们最有可能被不同的软件共同使用。然而，RDF 也应该能处理其他的数据类型集，因为正如已经阐述的，它们也是适合使用在 RDF 中的。

通常，RDF 软件可能被要求处理一些包含引用了软件不能处理的数据类型 的 RDF 数据，这样就有些事情软件是无法做到的。一个原因是，除了·rdf:XMLLiteral 外，RDF 本身不能定义标识数据类型的 URIref。因此，除非 RDF 软件理解特定的 URIref，否则将不能决定一个类型文字中的 URIref 是否实际上标识了一个数据类型。此外，即使一个 URIref 确实标识了一个数据类型，RDF 自身也无法定义这个数据类型和一个特定文字配对的合法性。只有那些能够正确理解这种数据类型的软件才能判断这种合法性。

例如，在如下三元组中的类型文字：

```
exstaff:85740  exterms:age  "pumpkin"^^xsd:integer .
```

或如图 10 所示的那样：

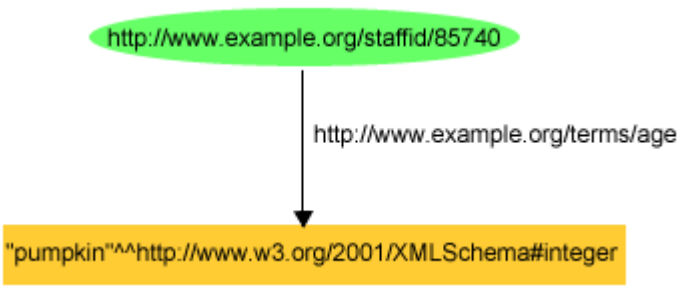

图 10：一个用作 John Smith 的年龄的非法类型文字

是有效的 RDF，但是很明显，一个错误与数据类型 xsd:integer 有关，因为“pumpkin”没有在 xsd:integer 的词法空间里定义。不能处理数据类型 xsd:integer 的 RDF 软件将不会发现这个错误。

然而，适当地使用 RDF 类型文字可以提供更多关于文字值的解释方面的信息，并因此使得 RDF 陈述成为一种更好的在应用程序中交换信息的方式。

[编辑]

2.5 总结

综上所述，基本上说，RDF 是比较简单的：由节点和弧构成的图解释说明了有关被 URIref 识别事物的陈述。在这一部分已经介绍了这些概念。如早先注明的，描述这些概念标准的（即权威的）RDF 规范是 RDF 概念（RDF Concepts）和抽象语法（Abstract Syntax） [RDF-CONCEPTS]，如果做更深入的研究就应当参考它们。在（标准的）RDF 语义学 （RDF Semantics）[RDF-SEMANTICS] 文档中详细说明了这些概念的形式语义。

然而，描述事物除了用目前为止所讨论的 RDF 陈述的基本技术之外，人们或组织也需要一种能够描述他们想要用在那些陈述中的词汇表（术语（term））的方法，显然，如下列的这些词汇表：

描述事物类型的词汇表（比如：exterms:Person）
描述事物属性的词汇表（比如：exterms:age 和 exterms:creation-date）
和牵涉到那些属性并可以作为陈述中的主体和客体的事物的类型的词汇表（比如：通常用 xsd:integer 词汇表来确定属性“exterms:age”的值）。

在 RDF 中描述这类词汇表是依据 RDF Vocabulary Description Language 1.0: RDF Schema] [RDF-VOCABULARY]，这将在第 5 节做介绍。

另外，RDF 基本思想的背景知识和它在提供一种能描述 Web 上信息的语言中扮演的角色，这一点可以在[WEBDATA]中找到。RDF 从知识表示，人工智能和数据管理领域中借鉴了一些思想，具体包括：概念图（Conceptual Graphs），基于逻辑的知识表示（logic-based knowledge representation），框架（frames），关系数据库（relational databases）。关于这些学科的背景知识可能的资料来源包括：[SOWA]，[CG]，[KIF]，[HAYES]，[LUGER]，and [GRAY]。

[编辑]

3. 表示 RDF 的 XML 语法: RDF/XML

如第 2 部分所描述的，RDF 的概念模型是一张图（graph）。RDF 提供了一种被称为 *RDF/XML* 的 XML 语法来书写和交换 RDF 图。与 RDF 的简略记法——三元组（triples）不同，RDF/XML 是书写 RDF 的规范性语法（normative syntax）。RDF/XML 定义于 RDF/XML 语法规范[RDF-SYNTAX]。本节描述 RDF/XML 的语法。

[编辑]

3.1 基本原理

RDF/XML 语法的基本思想可以通过前面的一些例子来说明。以下面这句英文为例：

```
http://www.example.org/index.html has a creation-date whose value is August 16, 1999
```

上面这个陈述可以用 RDF 图来表示（其中的 creation-date 属性已指定了 URIref），如图 11 所示:

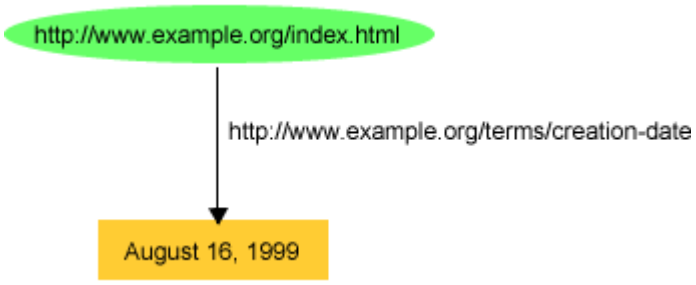

图 11: 描述一个网页的创建日期(Creation Date)

用三元组表示就是：

```
ex:index.html    exterms:creation-date    "August 16, 1999" .
```

（注意：在这个例子中没有用类型文字（typed literal）作为日期的值。类型文字在 RDF/XML 中的表示将在后面描述）

例 2 显示了图 11 所对应的 RDF/XML 语法:

例 2：描述网页创建日期的 RDF/XML

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.              xmlns:exterms="http://www.example.org/terms/">
4.   <rdf:Description rdf:about="http://www.example.org/index.html">
5.       <exterms:creation-date>August 16, 1999</exterms:creation-date>
6.   </rdf:Description>
7. </rdf:RDF>
```

（加入行号仅仅是为了便于解释例子，RDF/XML 中并不包含行号）

看起来有些摸不着头脑(a lot of overhead)，依次考虑该 XML 的各个部分将利于理解其作用。（附录 B 提供了一个关于 XML 的简单介绍）。

第 1 行是 *XML 声明（XML declaration）*：<?xml version="1.0"?>。它表明以下内容将是 XML，XML 版本号是 1.0。

第 2 行以 rdf:RDF 元素开始。它表明以下 XML 内容（从这里开始，直到第 7 行的</rdf:RDF>为止）用于表达 RDF。同一行紧随 rdf:RDF 其后的是 XML 命名空间声明（XML namespace declaration），即 rdf:RDF 首标签的 xmlns 属性。该声明指明在当前内容中出现的所有前缀为 rdf:的标签都属于由下

列 URIref 所标识的命名空间：http://www.w3.org/1999/02/22-rdf-syntax-ns#。以 http://www.w3.org/1999/02/22-rdf-syntax-ns#打头的 URIrefs 用于标识来自 RDF 词汇表中的术语。

第 3 行是另一个 XML 命名空间声明（关于前缀 exterms:的）。该声明用 rdf:RDF 元素的另一个 xmlns 属性来表示。它指明前缀 exterms:与命名空间 URIref http://www.example.org/terms/关联。以 http://www.example.org/terms/开始的 URIrefs 用于由 example 组织（example.org）[译注：example 组织为本文档所举示例]定义的词汇表中的术语。第 3 行末尾的">"符号表明 rdf:RDF 首标签的结束。第 1 至 3 行是常规的、必备的部分，用以表明当前的内容是 RDF/XML，并声明内容中所使用的命名空间。

第 4-6 行是 Figure 11 中所示陈述的 RDF/XML 主要部分。谈及 RDF 陈述时，显而易见，陈述是一种"description（描述）"。并且，它是一种"about（有关）"陈述主体的描述（在本例中，是有关 http://www.example.org/index.html 的描述）。RDF/XML 表示陈述的方式正是如此。第 4 行中 rdf:Description 的起始标签表明某个资源描述的开始，然后标识了陈述所"about（针对）"的资源（也就是陈述的主体）。RDF/XML 采用了 rdf:about 属性来指定主体资源的 URIref。第 5 行用 QName exterms:creation-date 作为标签，提供了"属性元素（property element）",来表示谓词以及陈述的客体。选择 QName exterms:creation-date，可以将本地名称 creation-date 扩展为 http://www.example.org/terms/creation-date，这个属性元素（property element ）的内容就是陈述的客体－即平凡文字"August 19, 1999 "（主体资源的 creation-date 属性的值）。属性元素（property element）在 rdf:Description 元素所包含内容中以嵌套的形式存在，意味着该属性（property ）应用于 rdf:Description 元素的 rdf:about 属性（attribute）所指定的资源。第 6 行表明这个 rdf:Description 到此结束。

最后，第 7 行表明从第 2 行开始的 rdf:RDF 元素到此结束。在能够通过上下文确定 XML 内容为 RDF/XML 的情况下，可以不用 rdf:RDF 元素来包括 RDF/XML 的内容。这在[RDF-SYNTAX]中将进一步讨论。但是，使用 rdf:RDF 元素在任何情况下均没有坏处，本文档中的示例一般都会（但并不总是）使用 rdf:RDF 元素。

例 2 展示了 RDF/XML 如何把一个 RDF 图编码为 XML 元素、属性、元素内容和属性值的基本思想。谓词（以及一些结点）的 URIrefs 被写作 XML *QNames*，即包含一个简短的*前缀（prefix）*（代表命名空间 URI）和一个*内部名（local name）*（代表命名空间中的元素或属性）（详见附录 B）。对于一个（命名空间 URIref，内部名）对，将命名空间 URIref 和内部名连接起来将形成结点或谓词的 URIref（像图 11 中的那样）。主体结点（subject nodes）的 URIrefs 被写为 XML 属性值（客体结点（object nodes）的 URIrefs 有时也可被写为属性值）。文字结点（Literal nodes）（总是客体结点）写为元素的文本内容或属性值。（本文档后面的部分将对这里的部分写法作进一步描述；关于所有写法的描述，请参见[RDF-SYNTAX]。）

对于包含多个陈述的 RDF 图可以用下面 RDF/XML 方式书写：用类似于例 2 中第 4 至 6 行的方式分别表示 RDF 图中的各个陈述。举例来说，对于下面两个陈述：

```
ex:index.html   exterms:creation-date   "August 16, 1999" .

ex:index.html   dc:language             "en" .
```

可以用例 3 中的 RDF/XML 来表示：

例 3: 表达两条陈述的 RDF/XML

```
1.  <?xml version="1.0"?>

2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

3.            xmlns:dc="http://purl.org/dc/elements/1.1/"

4.            xmlns:exterms="http://www.example.org/terms/">
```

```
5.    <rdf:Description rdf:about="http://www.example.org/index.html">
6.        <exterms:creation-date>August 16, 1999</exterms:creation-date>
7.    </rdf:Description>

8.    <rdf:Description rdf:about="http://www.example.org/index.html">
9.        <dc:language>en</dc:language>
10.   </rdf:Description>

11. </rdf:RDF>
```

例 3 和例 2 一样，只是多了一个表达第二条陈述的 rdf:Description 元素（第 8 至 10 行）。(第 3 行给出了另一个命名空间声明，以标识在第二条陈述中用到的命名空间）。可以用同样的方式添加任意多个陈述，只需对每条陈述分别使用一个 rdf:Description 元素。如例 3 所示，一旦完成了书写 XML 和命名空间声明，RDF/XML 中各个 RDF 陈述的书写将简单明了。

RDF/XML 语法提供了若干种简略表达形式以便于书写。比如在例 3 中，用多个属性和属性值来描述同一资源，这里资源 ex:index.html 同时作为多个陈述的主体（subject）。在这种情况下，RDF/XML 允许在标识该主体的 rdf:Description 元素下嵌入多个属性元素来表达属性。比如，要表达下列一组关于资源 http://www.example.org/index.html 的陈述：

```
ex:index.html   dc:creator              exstaff:85740 .
ex:index.html   exterms:creation-date   "August 16, 1999" .
ex:index.html   dc:language             "en" .
```

它所对应的 RDF 图（和图 3 相同）如图 12 所示：

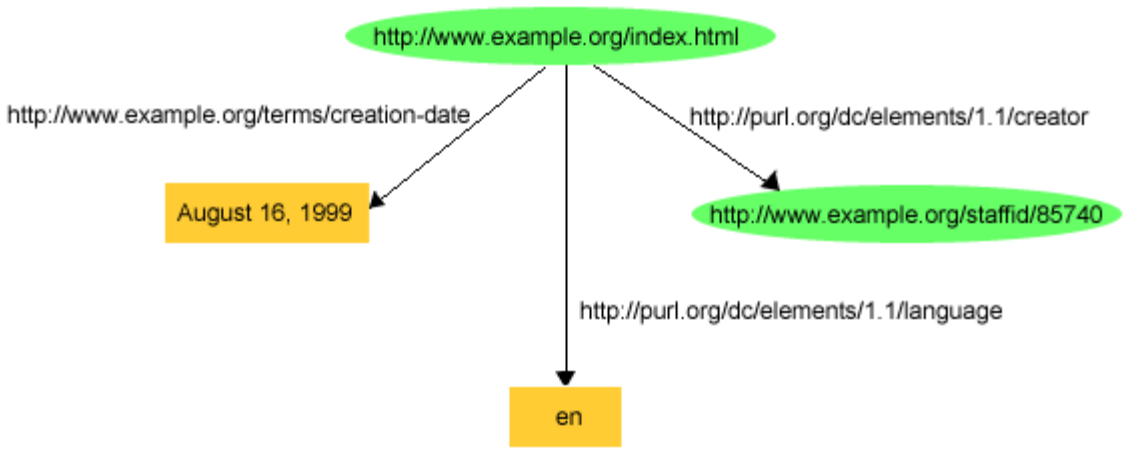

图 12：关于同一资源的多个陈述

例 4 是它所对应的 RDF/XML：

例 4： 简略表达多个属性

```
1.  <?xml version="1.0"?>

2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

3.              xmlns:dc="http://purl.org/dc/elements/1.1/"

4.              xmlns:exterms="http://www.example.org/terms/">


5.    <rdf:Description rdf:about="http://www.example.org/index.html">

6.          <exterms:creation-date>August 16, 1999</exterms:creation-date>

7.          <dc:language>en</dc:language>

8.          <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>

9.    </rdf:Description>


10. </rdf:RDF>
```

与前面两个例子相比，例 4 中多了一个 dc:creator 属性元素（在第 8 行）[译注//属性元素（property element）指表示 RDF 中的属性（property）的 XML 元素(XML element)]。另外，三个属性元素（表达主体 http://www.example.org/index.html 的三个属性）都嵌在同一个 rdf:Description 元素（标识主体 http://www.example.org/index.html）里，而不是为各个陈述单独写一个 rdf:Description 元素。

第 8 行引入了一种新的属性元素形式。第 7 行中的 dc:language 元素与例 2 中的 exterms:creation-date 元素类似，他们都是用平凡文字（plain literal）来表示属性的值，并且都用与属性名称（property name）对应的首标签（start-tag）和尾标签（end-tag）将上述文字括起来。然而，第 8 行中的 dc:creator 元素表示一个属性值为*另一个资源*（而不是平凡文字）的属性。如果把该资源的 URIref 写成被括在首标签和尾标签中的平凡文字（像前两个元素那样），这将表示 dc:creator 元素的值是*字符串* http://www.example.org/staffid/85740 ，而不是由该字符串代表的 URIref 所标识的资源。为了表明这一区别，dc:creator 元素被写成*空元素标签（empty-element tag）*（即没有尾标签）的形式，同时用一个 rdf:resource 属性（attribute）来表达属性的值（property value）。rdf:resource 属性（attribute）表明属性元素（property element）的值是另一个用 URIref 标识资源（resource）。由于该 URIref 要作为属性(attribute)*值* RDF/XML 要求这个 URIref 必须被写成绝对 URIref 或相对 URIref 的形式，而不能像写元素*名*（element name）或属性*名*（attribute name）那样被简略为 QName(关于绝对 URIref 和相对 URIref，请参见附录 A)。

例 4 中的 RDF/XML 是一种*简略形式*，理解这一点很重要。例 5 是描述同一个 RDF 图（图 12）的 RDF/XML，其中各个陈述被分开书写：

例 5：用分开的陈述书写例 4

```
<?xml version="1.0"?>
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:exterms="http://www.example.org/terms/">
  <rdf:Description rdf:about="http://www.example.org/index.html">
      <exterms:creation-date>August 16, 1999</exterms:creation-date>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:language>en</dc:language>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
  </rdf:Description>
</rdf:RDF>
```

下面部分将描述另外一些 RDF/XML 简略形式。[RDF-SYNTAX]提供了关于所有简略形式的更全面的描述。

RDF/XML 同样可以表示包含空白结点（*blank node*）（即没有 URIref 的结点，详见 2.3 节）的图。比如，图 13（取自[RDF-SYNTAX]）显示了一个表达下列信息的图：“文档'http://www.w3.org/TR/rdf-syntax-grammar'有一个 title（标题）（'RDF/XML Syntax Specification (Revised)'）和一个 editor（编者），而 editor 的 name（姓名）为'Dave Beckett'、home page（主页）为'http://purl.org/net/dajobe/'”。

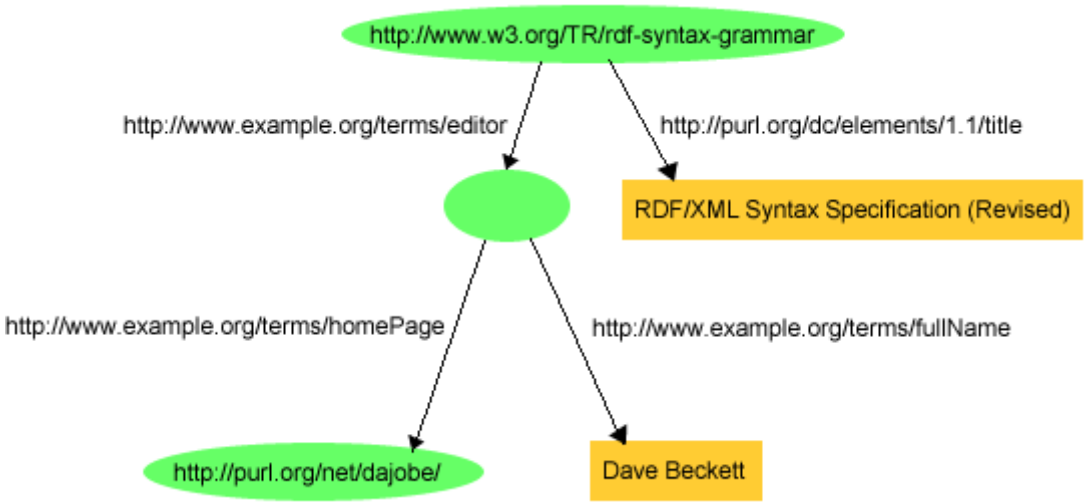

图 13：一个包含空白结点的图

上面的例子展示了 2.3 节中曾讨论过的想法，即用一个空白结点（blank node）来表示某个没有 URIref、但可用其它信息来描述的事物。在上面的例子中，该空白结点表示一个人，即文档的 editor（编者），这个人是用他的 name（姓名）和 homepage（主页）来描述的。

RDF/XML 提供了多种方式来表示包含空白结点的图。[RDF-SYNTAX]描述了所有这些方式。这里所描述的方式是其中最直接的一种，即为各个空白结点指定一个*空白结点标识符（blank node identifier）*。空白结点标识符用于标识某个特定 RDF/XML 文档内部的一个空白结点。但是与 URIref 不同，空白结点标识符在所属文档外部是不可识别的[译注//即仅在它被指定为空白结点标识符的那个文档内有效]。在 RDF/XML 中，所有可以出现资源 URIref 的地方都可以用 rdf:nodeID 属性（attribute）来引用空白结点（用一个空白结点标识符作为该属性的值）。 具体地，一个以空白结点为*主体*的陈述，在 RDF/XML 中可以用一个拥有 rdf:nodeID（而不是 rdf:about）属性（attribute）的 rdf:Description 元素来描述。同样地，一个以空白结点为*客体*的陈述可以用一个拥有 rdf:nodeID（而不是 rdf:resource）属性（attribute）的属性元素（property element）来描述。例 6 展示了如何用 rdf:nodeID 来描述图 13 对应的 RDF/XML：

例 6：描述空白结点的 RDF/XML

```
1.  <?xml version="1.0"?>

2.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

3.              xmlns:dc="http://purl.org/dc/elements/1.1/"

4.              xmlns:exterms="http://example.org/stuff/1.0/">


5.     <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">

6.       <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>

7.       <exterms:editor rdf:nodeID="abc"/>

8.     </rdf:Description>


9.     <rdf:Description rdf:nodeID="abc">

10.       <exterms:fullName>Dave Beckett</exterms:fullName>

11.       <exterms:homePage rdf:resource="http://purl.org/net/dajobe/"/>

12.    </rdf:Description>


13. </rdf:RDF>
```

在例 6 中，第 9 行使用空白结点标识符 abc 来标识作为多个陈述的主体的空白结点，然后在第 7 行中利用该空白结点标识符来表明它所对应的空白结点是某个资源的 exterms:editor 属性（property）。相对于[RDF-SYNTAX]中描述的其它方式而言，使用空白结点标识符的优点是可以在一个 RDF/XML 文档中多次引用同一个空白结点

最后，2.4 节中介绍的*类型文字（typed literals）*可以替代前面例子中的平凡文字（plain literals）作为属性的值（property values）。类型文字（typed literal）在 RDF/XML 中的表示方法为：为包含该文字的属性元素（property element）增加一个 rdf:datatype 属性（attribute），并通过该属性指定数据类型的 URIref。

比如，将例 2 中的陈述改为用类型文字而不是平凡文字来作为 exterms:creation-date 属性的值（property values），那么三元组将这样书写：

```
ex:index.html    exterms:creation-date    "1999-08-16"^^xsd:date .
```

它所对应的 RDF/XML 如例 7 所示：

例 7：使用类型文字的 RDF/XML

```
1. <?xml version="1.0"?>

2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

3.           xmlns:exterms="http://www.example.org/terms/">

4.   <rdf:Description rdf:about="http://www.example.org/index.html">

5.     <exterms:creation-date rdf:datatype=

         "http://www.w3.org/2001/XMLSchema#date">1999-08-16

       </exterms:creation-date>

6.   </rdf:Description>

7. </rdf:RDF>
```

例 7 中，在第 5 行给出的 exterms:creation-date 属性元素（property element）的值是一个类型文字（而不是平凡文字），这是通过为 exterms:creation-date 元素的首标签（start-tag）增加一个 rdf:datatype 属性（attribute）并由此指定数据类型实现的。rdf:datatype 属性（attribute）的值应是某个数据类型的 URIref，在本例中它是 XML Schema 中的 date 数据类型的 URIref。作为属性值（attribute value），URIref 必须被写全[译注//即写为相对 URIref 或绝对 URIref]，而不是像在三元组中那样写出它的 QName 形式 xsd:date。用一个正确的数据类型文字作为元素内容（element content），在本例中是文字 1999-08-16，即用 XML Schema 中的 date 数据类型表示的一个代表 1999 年 8 月 16 日的文字。

在本文档下面的示例中，我们将避免使用平凡文字（无类型的），而是使用具有正确数据类型（datatype）的类型文字（typed literals）。这样做的目的在于强调类型文字可以传递更多关于文字值（literal values）解释的信息。（但下列情况例外，即示例取自实际应用，而在该实际应用中没有使用类型文字。在这种情况下，示例将保留原应用中的平凡文字，以精确反映 RDF 在该应用中的用法。）在 RDF/XML 中，平凡文字和类型文字（以及某些例外情况下的标签（tag））都可以包含 Unicode[UNICODE]字符，以允许直接表达多语言的信息。

例 7 举例说明了使用类型文字需要为每一个元素值为类型文字的元素书写一个 rdf:datatype 属性（attribute），并用一个标识具体数据类型的 URIref 作为属性值。如前面提到的，RDF/XML 要求用作属性值（attribute value）的 URIrefs 必须书写完整，而不能写成 QName 形式。在这种情况下，RDF/XML 允许使用 XML *实体（entities）*以提高可读性，即为 URIrefs 提供一个另外的简写形式。XML 实体声明的本质是将一个 XML 名称（XML name）与一个字符串相关联。如果实体名在 XML 文档中被引用，XML 处理器（XML processors）将用相应的字符串来替换该引用。比如，下列 ENTITY 声明（被写在 RDF/XML 文档头部的 DOCTYPE 声明中）：

```
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
```

把实体 xsd 定义为一个代表命名空间 URIref（对应于 XML Schema 数据类型的命名空间）的字符串。这一声明使得完整的命名空间 URIref 可在 XML 文档中被简略为实体引用&xsd;。在例 7 中使用上述简略形式便可得到例 8.

例 8: 使用类型文字和 XML 实体的 RDF/XML

```
1. <?xml version="1.0"?>
2. <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

3. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4.           xmlns:exterms="http://www.example.org/terms/">

5.   <rdf:Description rdf:about="http://www.example.org/index.html">
6.     <exterms:creation-date rdf:datatype="&xsd;date">1999-08-16
       </exterms:creation-date>
7.   </rdf:Description>

8. </rdf:RDF>
```

第 2 行中的 DOCTYPE 声明定义了实体 xsd，该实体被用在第 6 行中。

是否在 RDF/XML 中用 XML 实体作为简略机制是可选的，因此是否在 RDF/XML 中使用 XML DOCTYPE 声明也是可选的。（对于熟悉 XML 的读者，RDF/XML 只需是“良构的（well-formed）”XML。RDF/XML 没有被设计为将用验证型 XML 处理器（validating XML processor）验证它对于某个 DTD 的有效性。附录 B 将对此作全面论述,并提供了关于 XML 的更多信息。 ）

出于可读性考虑，本文剩余部分的示例将使用 XML 实体 xsd（如前面描述的那样）。附录 B 将对 XML 实体作进一步讨论。如附录 B 的说明，其他 URIrefs（而且更一般地，字符串）也可以被简略为使用 XML 实体。然而，在本文档的示例中，我们仅对 XML Schema 数据类型采用这种简略方式。

尽管还有其他用于书写 RDF/XML 的简略形式，但到目前为止所介绍的方法提供了一种简单而一般的用 RDF/XML 来表达 RDF 图的方法。要应用这些方法，一个 RDF 图可以这样书写 RDF/XML：

所有空白结点（blank nodes）被指定空白结点标识符（blank node identifiers）。

依次列出各个结点，将它作为一个非嵌套的 rdf:Description 元素的主体（subject）。若该结点有 URIref，则 rdf:Description 元素使用 rdf:about 属性（attribute）；若该结点是空结点，则 rdf:Description 元素使用 rdf:nodeID 属性（attribute）。

对于各个以该结点作为主体（subject）的三元组（triple），创建一个正确的属性元素（property element）。该属性元素或者是元素内容（element content）为文字（literal）（可能为空）、或者是有一个指定该三元组的客体（object）的 rdf:resource 属性（attribute）（对于客体有 URIref 的情况）、或者是有一个指定该三元组的客体的 rdf:nodeID 属性（attribute）（对于客体结点为空结点的情况）。

与更多在[RDF-SYNTAX]中描述的简略方法相比，上述这种简单的方式提供了实际图结构的最直接的表示。另外，如果应用中的输出 RDF/XML 将在进一步的 RDF 处理中被使用的话，这种方式是特别推荐的。

[编辑]

3.2 简写与组织 RDF URIrefs

到目前为止，所有的示例都假定所描述资源已指派了 URIrefs。比如，前面的几个例子提供了关于 example.org 网页的描述性信息，该网页的 URIref 为 http://www.example.org/index.html 。在 RDF/XML 中，这个资源（resource）的标识是通过使用一个 rdf:about 属性（attribute）并用资源的 URIref 作为属性值实现的。尽管 RDF 并没有规定或限定如何为资源（resources）指派 URIrefs，有时希望为一些有组织的资源指派 URIrefs。比如，设想一个运动产品公司 example.com 要为它的产品（比如帐篷、旅行鞋等等）提供一个基于 RDF 的目录，该目录是一个 RDF/XML 文档，用 http://www.example.com/2002/04/products 来对它进行标识和定位。在该资源中，各个产品可能会有一个单独的 RDF 描述。例 9 所示的 RDF/XML 描述了这个产品目录和一种型号为“Overnighter”的帐篷所对应的目录项及其描述：

例 9：example.com 产品目录的 RDF/XML

```
1.  <?xml version="1.0"?>

2.  <!DOCTYPE rdf:RDF [<!ENTITY
xsd "http://www.w3.org/2001/XMLSchema#">]>

3.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

4.               xmlns:exterms="http://www.example.com/terms/">


5.    <rdf:Description rdf:ID="item10245">

6.          <exterms:model
rdf:datatype="&xsd;string">Overnighter</exterms:model>

7.          <exterms:sleeps
rdf:datatype="&xsd;integer">2</exterms:sleeps>

8.          <exterms:weight
rdf:datatype="&xsd;decimal">2.4</exterms:weight>

9.          <exterms:packedSize
rdf:datatype="&xsd;integer">784</exterms:packedSize>
```

```
10.     </rdf:Description>

  ...other product descriptions...

11.  </rdf:RDF>
```

与前面例子的一个重要*区别*在于第 5 行中的 rdf:Description 元素，它有一个 rdf:ID 属性（attribute）而不是 rdf:about 属性（attribute）。rdf:ID 用于指定一个*片断标识符（fragment identifier）*作为资源完整 URIref 的简略形式，该片断标识符在 rdf:ID 属性值（attribute）（在本例中为 item10245，它可能是由 example.com 指定的目录号）中给出。片断标识符 item10245 的解析是相对于*基准 URI（base URI）*（在本例中，基准 URI 为目录文档的 URI）。该帐篷的完整 URIref 是这样形成的：取（目录的）基准 URI，在后面添加字符“#”（表明后面跟随的是片断标识符）和字符串“item10245”，这样得到了绝对 URIref http://www.example.com/2002/04/products#item10245.

。

这里的 rdf:ID 属性有些类似 XML 和 HTML 中的 ID 属性（attribute），因为它定义了一个相对于当前基准 URI（在本例中，基准 URI 是目录的基准 URI）[译注//其原因参见 XML Base 或其简体中文翻译[http://xmlbase.w3china.org/]]必须唯一的名称。在本例中，rdf:ID 属性（attribute）为这种特定的帐篷指派一个名称（item10245）。同一文档中的其他 RDF/XML 可以用两种方式来引用这个帐篷：使用绝对 URIref http://www.example.com/2002/04/products#item10245 ，或使用*相对 URIref* #item10245。相对 URIref 可被理解为一个相对目录的基准 URIref 定义的 URIref。使用同样的简略形式，帐篷的 URIref 也可以在目录项中通过使用 rdf:about="#item10245"来给出（也就是说,直接给出帐篷的相对 URIref），而不是使用 rdf:ID="item10245"。作为一种简略机制，这两种方式本质上是相同的：RDF/XML 在这两种方式下形成的完整 URIref 完全相同，都是 http://www.example.com/2002/04/products#item10245 。但是，使用 rdf:ID 提供了一种机制，可以在指派一组互不相同的名称集合时进行检测，因为 rdf:ID 属性（attribute）给定的值在隶属同一个基准 URI 的范围（在本例中为产品目录文档）内只能出现一次。如果用另一种方式的话，example.com 将通过两步给出该帐篷的 URIref：首先指定整个产品目录的 URIref，然后在产品目录中的帐篷描述中使用一个相对 URIref 来表明该帐篷被指派的 URIref。另外，相对 URIref 的使用可被理解为是对 RDF 中这个帐篷指派的完整 URIref 的一种简略形式，也可以被认为是对产品目录中这个帐篷的 URIref 的指派。

位于产品目录*外部*的 RDF 可以用完整的 URIref（通过连接该帐篷的相对 URIref #item10245 和产品目录的基准 URI 形成绝对 URIref http://www.example.com/2002/04/products#item10245）来引用这个帐篷。例如，一个户外运动网站 exampleRatings.com 也许会用 RDF 来提供各种帐篷的排名。例 9 所描述的帐篷被给予的等级为（5-star），这一信息可以在 exampleRatings.com 网站上由例 10 所示的 RDF/XML 来表示：

例 10: exampleRatings.com 给上述帐篷的等级

```
1.  <?xml version="1.0"?>

2.  <!DOCTYPE rdf:RDF [<!ENTITY
xsd "http://www.w3.org/2001/XMLSchema#">]>

3.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"

4.
xmlns:sportex="http://www.exampleRatings.com/terms/">


5.     <rdf:Description
```

```
rdf:about="http://www.example.com/2002/04/products#item10245">

6.          <sportex:ratingBy rdf:datatype="&xsd;string">Richard
Roe</sportex:ratingBy>

7.          <sportex:numberStars
rdf:datatype="&xsd;integer">5</sportex:numberStars>

8.     </rdf:Description>

9.  </rdf:RDF>
```

在例 10 中，第 5 行使用的 rdf:Description 元素带有一个 rdf:about 属性（attribute），该属性的值为被描述的帐篷的完整 URIref。使用这个 URIref 可以精确标识被排名所引用的帐篷。

上面这些例子说明了若干要点。首先，尽管 RDF 没有规定或限定如何为资源（如本例中的各种帐篷和其他产品）指派 URIrefs，但在 RDF 中可通过下面的方法获得为资源指派 URIrefs 的*效果*：（在 RDF 之外）标识一个文档（如本例中的产品目录），并将它作为资源描述的来源；而在该文档中描述各个资源时使用相对 URIref[译注//该相对 URIref 被作为资源在所属文档范围内的标识符]。比如，example.com 可以将这个产品目录作为描述它的产品的总的来源，如果一个产品的产品号不在产品目录中的任何条目中出现，那么它就不是一个 example.com 所知道的产品。（注意：RDF 并不会仅根据两个资源的 URIrefs 有相同的基准（base）或在某些方面相像而假定它们之间存在任何特定的关系。也许 example.com 了解这些关系，但它们没有直接在 RDF 中定义。）

以上的示例说明了 Web 体系结构的一个基本原则，即任何人都应能够用他们愿意使用的任何词汇自由地添加关于某一现存资源的信息 [BERNERS-LEE98]。这些例子进一步说明了描述某个特定资源的 RDF 不需要全部存放于某一处；相反，它可以被分布于 Web 上。这一点不仅对上述情形（即一个组织对其他组织定义的资源给出排名或评论）适用，对于下面这种情形也是适用的，即资源的最初定义者希望通过增加关于资源（或其他人）的信息来充实资源的描述。这可以通过修改 RDF 文档（即资源被最初定义的文档）来实现：即增加为了描述附加信息所必需的属性（properties）和值。另一种实现方式（如本例所展示的）是：创建一个单独的文档，并在其中通过使用 rdf:Description 元素为资源（用 rdf:Description 元素的 rdf:about 属性（attribute）的值来标识）提供附加的属性（properties）和值。

上面的论述表明，相对 URIrefs（比如#item10245）的解析将是相对于某个*基准 URI（base URI）*的。缺省情况下，这个基准 URI 可以是该相对 URI 所在资源的 URI。但是，在某些情况下也许希望能够显式指定基准 URI。比如，设想除位于 http://www.example.com/2002/04/products 的产品目录以外，example.org 希望在某个镜像站点上提供一个产品目录的副本（比如，位于 http://mirror.example.com/2002/04/products ）。这可能会造成一些问题，因为如果是通过镜像站点获得产品目录的话，帐篷的 URIref 将会通过产品目录的 URI 产生，也就是说形成的绝对 URIref 为 http://mirror.example.com/2002/04/products#item10245 ，而不是 http://www.example.com/2002/04/products#item10245 。显然，这将引用一个非期望的资源。作为另一个选择，example.org 也许希望为它的产品的 URIrefs 指定一个基准 URIref，而*不是*使用产品目录文档的位置作为基准 URIref。

为了处理上述情况，RDF/XML 支持 XML Base [XML-BASE]。XML Base 允许 XML 文档指定一个基准 URI，而不是将 XML 文档本身的 URI 作为基准 URI。例 11 显示了产品目录是如何用 XML Base 来描述的：

例 11: 在 example.com 的产品目录中使用 XML Base

```
1.  <?xml version="1.0"?>

2.  <!DOCTYPE rdf:RDF [<!ENTITY
xsd "http://www.w3.org/2001/XMLSchema#">]>

3.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
```

```
ns#"

4.               xmlns:exterms="http://www.example.com/terms/"

5.
xml:base="http://www.example.com/2002/04/products">


6.     <rdf:Description rdf:ID="item10245">

7.          <exterms:model
rdf:datatype="&xsd;string">Overnighter</exterms:model>

8.          <exterms:sleeps
rdf:datatype="&xsd;integer">2</exterms:sleeps>

9.          <exterms:weight
rdf:datatype="&xsd;decimal">2.4</exterms:weight>

10.         <exterms:packedSize
rdf:datatype="&xsd;integer">784</exterms:packedSize>

11.    </rdf:Description>


  ...other product descriptions...


12.  </rdf:RDF>
```

在例 11 中，第 5 行的 xml:base 声明为 rdf:RDF 元素里的内容指定了基准 URI（直到在遇到其他被指定的 xml:base 为止）http://www.example.com/2002/04/products 。这样，在元素里引用的所有相对 URIrefs 的解析将相对于 xml:base 所指定的基准 URI，而与这些相对 URIrefs 所在文档的 URI 是什么无关。因此，所描述帐篷的相对 URIref #item10245 将被解析为同一个绝对 URIref http://www.example.com/2002/04/products#item10245，不管产品目录文档的实际 URI 是什么，也不管基准 URIref 是否真的标识了某一具体文档。

到目前为止，所有示例使用的都是同一个产品描述——example.com 产品目录中的某个具体型号的帐篷。然而，example.org 也许希望提供多种不同型号的帐篷，同样对旅行背包、旅行鞋等多个不同类型产品也提供多种选择。把事物划分为不同的*种类（kings）*或*类别（categories）*这一想法与程序设计语言里的对象（objects）有不同的*类型（types）*或*类（classes）*类似。RDF 通过提供预定义属性（property）rdf:type 来支持这种概念。当我们用 rdf:type 属性（property）来描述一个 RDF 资源时，该属性的值就是被看作表达这种事物的*种类（kings）*或*类别（categories）*的资源。该属性的主体被看作为该*种类（kings）*或*类别（categories）*的一个*实例（instance）*。例 12 展示了 example.com 是如何使用 rdf:type 来表达产品描述是关于一个帐篷的：

例 12: 在帐篷的描述中使用 rdf:type

```
1.   <?xml version="1.0"?>

2.   <!DOCTYPE rdf:RDF [<!ENTITY
```

```
xsd "http://www.w3.org/2001/XMLSchema#">]>

3.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"

4.               xmlns:exterms="http://www.example.com/terms/"

5.
xml:base="http://www.example.com/2002/04/products">


6.     <rdf:Description rdf:ID="item10245">

7.          <rdf:type
rdf:resource="http://www.example.com/terms/Tent"/>

8.          <exterms:model
rdf:datatype="&xsd;string">Overnighter</exterms:model>

9.          <exterms:sleeps
rdf:datatype="&xsd;integer">2</exterms:sleeps>

10.         <exterms:weight
rdf:datatype="&xsd;decimal">2.4</exterms:weight>

11.         <exterms:packedSize
rdf:datatype="&xsd;integer">784</exterms:packedSize>

12.    </rdf:Description>


  ...other product descriptions...


13.  </rdf:RDF>
```

在例 12 中，第 7 行的 rdf:type 属性（property）表示被描述的资源是 URLref http://www.example.com/terms/Tent 所标识的类的一个实例（instance）。这假定了 example.com 已经在某个词汇表中描述了这个类（class），因此才可以在上面的例子中用该类的绝对 URIref 来引用它。如果 example.com 是在产品目录所在文档中描述了这个类，那么可以用相对 URIref #Tent 来引用它。

RDF 本身并没有提供方法来定义特定应用中的类（class），比如本例中的 Tent、或它们的属性（properties）如 exterms:weight 等。这些类是在 *RDF schema* 中使用 *RDF Schema* 语言（将在第 5 节介绍）来描述的。其他用于类描述的方法也可以被定义，比如 *DAML+OIL* 和 *OWL* 语言（将在 http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#richerschemas 5.5 节]介绍）。

在 RDF 中，为被描述的资源指定 rdf:type 属性（property）以指明它是某个特定类型或类的实例是非常普遍的手法。这样的资源在 RDF 图中被称作 *有类型结点（typed node）*，在 RDF/XML 中被称作 *有类型结点元素（typed node elements）*。RDF/XML 提供了一种特殊的简略形式以描述这些有类型结点。在这种简略形式中，rdf:type 属性及其的值被移去，而结点对应的 rdf:Description 元素被替换为一个以 QName 为名称的元素（QName 对应于被移去的那个 rdf:type 属性的值，也就是某个类的 URIref）。使用这种简略形式，例 12 中 example.com 的帐篷也可用例 13 所示的 RDF/XML 来描述：

例 13：简略帐篷的类型

```
1.  <?xml version="1.0"?>

2.  <!DOCTYPE rdf:RDF [<!ENTITY
xsd "http://www.w3.org/2001/XMLSchema#">]>

3.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"

4.               xmlns:exterms="http://www.example.com/terms/"

5.
xml:base="http://www.example.com/2002/04/products">


6.     <exterms:Tent rdf:ID="item10245">

7.          <exterms:model
rdf:datatype="&xsd;string">Overnighter</exterms:model>

8.          <exterms:sleeps
rdf:datatype="&xsd;integer">2</exterms:sleeps>

9.          <exterms:weight
rdf:datatype="&xsd;decimal">2.4</exterms:weight>

10.         <exterms:packedSize
rdf:datatype="&xsd;integer">784</exterms:packedSize>

11.    </exterms:Tent>


  ...other product descriptions...


12.  </rdf:RDF>
```

因为一个资源可能被描述为是多个类的实，因此一个资源可能有多个 rdf:type 属性。但是，在这些 rdf:type 属性中，只有一个可被写为简略形式，其余的必须像例 12 中的那样用 rdf:type 属性写出。

有类型结点的简略形式除用于描述用户自定义类（user-defined classes）（比如 exterms:Tent）的实例以外，在描述内建 RDF 类（built-in RDF classes，将在第 4 节中介绍）（比如 rdf:Bag）和内建 RDF Schema 类（built-in RDF Schema classes，将在第 5 节中介绍）（比如 rdfs:Class）时也是被普遍使用的。

例 12 和例 13 展示了一点，即用 RDF/XML 写出的 RDF 陈述可以非常类似直接用 XML（而不是 RDF）书写的描述。在 XML 正被使用于越来越多的应用这一背景下，这是一个重要的考虑，因为这意味着 RDF 可在这些应用中被使用，而不需对信息的构造方式作较大改动。

[编辑]

3.3 总结

上面的例子已经展现了 RDF/XML 背后的一些基本思想。这些例子为开始书写有用的 RDF/XML 提供了足够的信息。关于用 XML 进行 RDF 陈述的建模（被称作 *striping*）背后的理论、以及其他可用的 RDF/XML 简略形式、和其他关于用 XML 书写 RDF 的细节和示例，请参见（规范性）文档 RDF/XML 语法规范 [RDF-SYNTAX]。

[编辑]

4. 其他 RDF 表达能力

RDF 提供了一些额外 的表达能力，如内嵌的表示资源组和 RDF 陈述组的类和属性，还能让 XML 片断做为属性的值。下面讨论这些额外的表达能力。

[编辑]

4.1 RDF 容器

我们常常需要描述一组事物：例如，说一般书是由多个作者写的，或把上某门课程的学生都列出来，或一个软件包下的所有模块。RDF 提供了一些预定义的类型和属性用以描述一组事物。

首先，RDF 提供了容器词汇，包括三个预定义的类型，以及他们的一些属性。一个容器是一个包含了一些事物的资源，这些被包含的事物称为成员。容器的成员可能是资源（包括匿名节点）或文字。RDF 定义了三种类型的容器：

rdf:Bag

rdf:Seq

rdf:Alt

一个包（Bag,是类型为 rdf:Bag 的资源）表示了一组可能包含重复成员的资源或文字，且成员之间是无序的。例如，包可以用来描述对于成员的添加或处理顺序没有特别要求的组。

一个序列（Sequence,是类型为 rdf:Seq 的资源）表示了一组资源或文字，其中可能有重复的成员，而且成员之间是有序的。例如，序列可以用来描述一组必须按字母顺序排列的事物。

一个替换（Alternative,是类型为 rdf:Alt 的资源）表示了一组可以选择的资源或文字（常常是属性的一个值）。例如，序列可以用来描述一组可以互相替换的关于著作的不同语言的翻译，或者描述一个资源可能出现的几个因特网镜像站点。在应用中，如果属性的值是一个替换，这可以选择替换中任意一个合适的成员作为属性的值。

为了表示一个资源是一个容器，这个资源必须有个属性 rdf:type 且值为预定义的三个资源：rdf:Bag, rdf:Seq, or rdf:Alt 之一。这个容器资源（可能是匿名节点或由 URIref 标识的节点）代表了作为一个整体的一组事物。容器的成员和容器资源之间的关系是由一组专门为此定义的表示成员关系的属性描述的，这些属性的主体是容器资源，客体是其成员。这些成员关系属性的名字形为 rdf:_*n*，其中 *n* 是一个大于 0 的十进制整数，如 rdf:_1, rdf:_2, rdf:_3 等等。除了成员关系属性和 rdf:type 属性外，容器资源还可以有其他的属性。

很重要的一点是：尽管这些容器可以用预定义的 RDF 类型和属性定义，但并没有为这些容器提供特定的意义，例如，一个替换容器的成员的值是可替换的，这只是 RDF 的设计用途。这些容器类型和他们的定义，目的是在需要描述一组事物的时候建立一个大家认同的惯例。RDF 所做的只是提供预定以的类型和属性用以构造 RDF 图来描述各种容器，相对于一个普通的资源如 ex:Tent (如 3.2 节讨论的)，它并没有给作为包的资源更多的内置的解释。在这种情况下，应用程序必须注意到不同类型的容器有不同的含义，这点会在下面的例子中进一步讨论：

容器的一个惯常用法是表明有个属性的值是一组事物。例如，为了表示句子“参加课程 6.001 的学生有：Amy, Mohamed, Johann, Maria, 和 Phuong”，可以给课程一个 s:students 属性(来自一个合适的词汇集)，它的值是一个包容器（表示一组学生），然后再用容器成员关系属性，每个学生可以标识为这个组的成员，如 RDF 图图 14 所示：

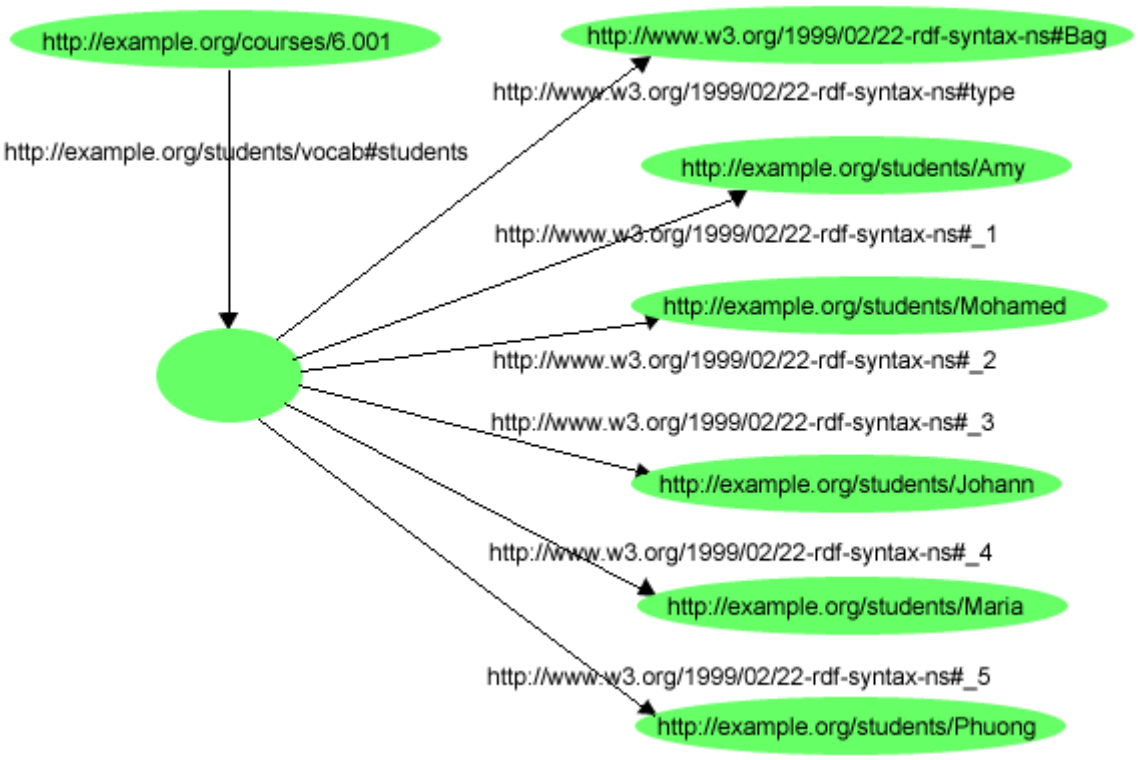

图 14：一个简单的包容器描述

因为这个例子中属性 s:students 的值是一个包，因此学生的 URIrefs 的顺序是无关紧要的，尽管容器成员属性的名字包含了一些整数。应用程序应该在创建或处理图的时候忽略掉属性名字中显示出来的顺序。

RDF/XML 提供了一些特殊的语法和缩写来简化容器的描述。例如例 14 描述了例 14 所示的图。

例 14：用 RDF/XML 书写的一个包含学生的包（bag）

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:s="http://example.org/students/vocab#">



   <rdf:Description rdf:about="http://example.org/courses/6.001">

      <s:students>

         <rdf:Bag>

            <rdf:li rdf:resource="http://example.org/students/Amy"/>
```

```
            <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
            <rdf:li rdf:resource="http://example.org/students/Johann"/>
            <rdf:li rdf:resource="http://example.org/students/Maria"/>
            <rdf:li rdf:resource="http://example.org/students/Phuong"/>
          </rdf:Bag>
       </s:students>
    </rdf:Description>
 </rdf:RDF>
```

例 14 展示了 RDF/XML 用 rdf:li 作为一个便利元素来避免显式地指定属性的名字。那些标号了的属性，如 rdf:_1, rdf:_2 是由图中的 rdf:li 自动生成的。元素名 rdf:li 是借鉴 HTML 的列表的。<rdf:Bag>元素又是一个例 13 演示的缩写的例子，即当描述一个类的实例时，用单个元素替换一个 rdf:Description 元素和 rdf:type 元素。因为没有指定 URIref，这个包资源是一个匿名节点。另外，它是属性元素<s:students> 的子节点，这也是一种缩写，表示这个匿名节点是属性的值。[RDF-SYNTAX]讲解了更详细的缩写方法。

容器 rdf:Seq 的图结构和对应的 RDF/XML 写法和 rdf:Bag 相似（唯一的区别是类型是 rdf:Seq）。同样地，虽然 rdf:Seq 是用来描述序列，但应用程序要负责在创建和处理 RDF 图时正确解释体现在属性名字中的顺序。

为了演示替换容器的用法，句子“X11 的源代码可能可以在网站 ftp.example.org, ftp1.example.org, 或 ftp2.example.org 上找到”可用如图 15 所示的图表示：

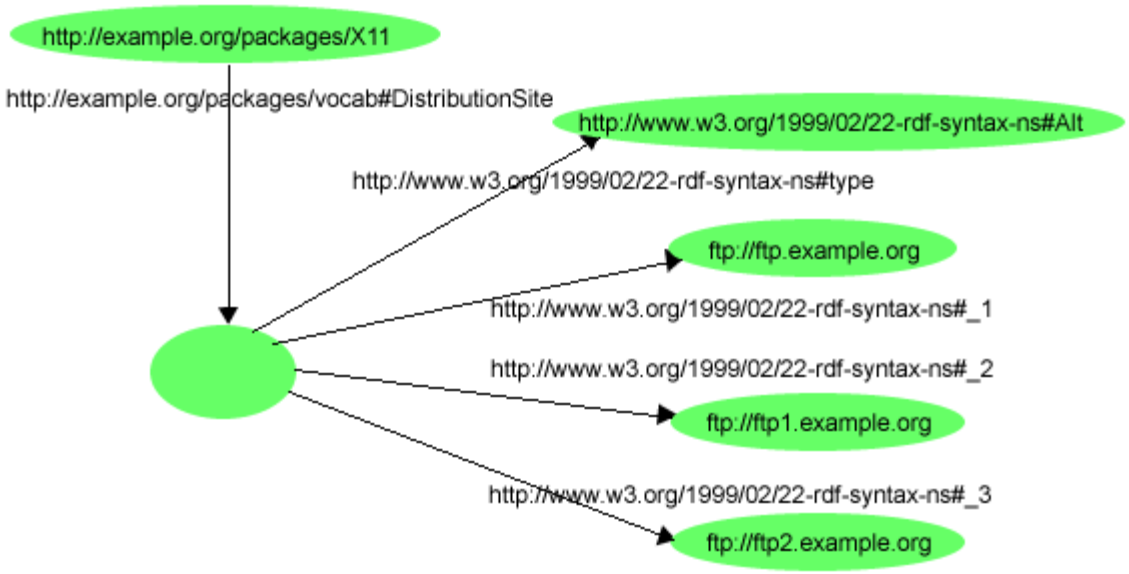

图 15：一个简单的替换容器描述

图 15 代表的图可以在 RDF/XML 中如例 15 所示：

例 15：用 RDF/XML 书写的一个替换容器（Alt Container）

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:s="http://example.org/packages/vocab#">



   <rdf:Description rdf:about="http://example.org/packages/X11">

      <s:DistributionSite>

         <rdf:Alt>

            <rdf:li rdf:resource="ftp://ftp.example.org"/>

            <rdf:li rdf:resource="ftp://ftp1.example.org"/>

            <rdf:li rdf:resource="ftp://ftp2.example.org"/>

         </rdf:Alt>

      </s:DistributionSite>

   </rdf:Description>

</rdf:RDF>
```

一个替换容器被设计为至少有一个成员，由属性 rdf:_1 标识。这个成员被认为是属性的缺省值或优先值。除了 rdf:_1 外，其他剩下的成员的顺序是无关紧要的。

图 15 中的 RDF 图表示属性 s:DistributionSite 的值是一个类型是替换容器的资源。关于这个 RDF 图更多的意义必须由应用程序来解释，例如，替换容器的一个成员可以看成属性 s:DistributionSite 的值,或者 ftp://ftp.example.org 是这个属性的缺省值或优先值。

替换容器常常和语言标记一起使用(RDF/XML 允许用在[XML]定义的属性 xml:lang 表示元素节点的内容是用什么语言书写的。 [RDF-SYNTAX]描述了 xml:lang 的用法, 6.2 节也演示它的用法。)。例如，如果一个著作被翻译成了多种语言，这这本著作的标题属性可以是一个替换容器，这个容器包含了用多种语言书写的标题。

包和替换的用法的区别可以通过以下例子更清楚地看出来。哈克贝里?芬历险记(“Huckleberry Finn”)这本书只有一个作者，但作者有两个名字（马克?吐温是笔名，塞谬尔?克莱门斯是原名），每个名字都可以表示作者。因此，包含这两个名字的替换比包更适合作为作者名这个属性的值，因为包容器隐含了这是两个不同的作者的意思。

用户可以不用 RDF 容器词汇，随意选择自己的方式表达一组资源。这些 RDF 容器只是提供一种通用的方式，如果能得到广泛应用，可以使描述一组资源的数据具有更好的互操作性。

有时候，有明显的方法可以代替 RDF 容器。例如，要描述一个资源和一组资源的关系，可以用多个陈述（属性是相同的）来表示，每个陈述的主体都是第一个资源，客体是一组资源的一个资源。这和用一个陈述表示，主体是第一个资源，客体是包含一组资源的容器的方式在结构上是不同的。有些时候，这两种表示方式有相同的意义，但也有时候不一样。如何在不同的情况下去选择不同的方式要考虑以下问题：

考虑一个例子表示作家和其的作品之间关系的例子，如下面的句子：

Sue 已经写了"Anthology of Time", "Zoological Reasoning", 和 "Gravitational Reflections"这三本书。

在这个例子中，有三个资源都是由相同的作者独立完成的，这可以用重复的属性表示如下：

```
exstaff:Sue    exterms:publication    ex:AnthologyOfTime .

exstaff:Sue    exterms:publication    ex:ZoologicalReasoning .

exstaff:Sue    exterms:publication    ex:GravitationalReflections .
```

在这个例子中，只说了作品之间的关系是他们由同一个作者写成。每个陈述是一个独立的事实，因为用重复的属性也是合理的选择。当然，用一个关于 Sue 写的一组书的陈述表示的话，也是合理的。

```
exstaff:Sue    exterms:publication    _:z .

_:z            rdf:type               rdf:Bag .

_:z            rdf:_1                 ex:AnthologyOfTime .

_:z            rdf:_2                 ex:ZoologicalReasoning .

_:z            rdf:_3                 ex:GravitationalReflections .
```

在另一方面，句子

```
这个决议被由 Fred, Wilma 和 Dino 组成的章程委员会批准了。
```

```
```

说这个委员会作为一个整体批准了这个决议；并不是说每个单独的委员会成员都投票同意了这个决议。在这个例子中，如果把这个句子用三个独立的 exterms:approvedBy 陈述表示，每一个陈述的主体是一个委员会成员，如下所示，则是不恰当的，

```
ex:resolution   exterms:approvedBy   ex:Fred .

ex:resolution   exterms:approvedBy   ex:Wilma .

ex:resolution   exterms:approvedBy   ex:Dino .
```

因为这些陈述是说每个成员都批准了这个决议。

如果将这句话表示为一个陈述，其主体是决议，客体是委员会，则更为恰当。这个委员会可以用一个包含这三个委员会成员的包容器来描述，如下面的这些三元组所示：

```
ex:resolution       exterms:approvedBy   ex:rulesCommittee .

ex:rulesCommittee   rdf:type             rdf:Bag .

ex:rulesCommittee   rdf:_1               ex:Fred .

ex:rulesCommittee   rdf:_2               ex:Wilma .

ex:rulesCommittee   rdf:_3               ex:Dino .
```

当用 RDF 容器时，要明白这些陈述不是像程序语言中的结构那样在构造容器，而是描述已经存在的容器。在刚给出的章程委员会例子中，章程委员会是一组没有排序的人，不管 RDF 的描述方式是怎样的。说资源 ex:rulesCommittee 的类型是 rdf:Bag，并不等于说章程委员会是一个数据结构，或说构造了一个包含这个组的成员的数据结构（章程委员会可以描述为一个包即使没有描述他的任意成员。反之，它描述了章程委员会和包容器有对应的特点，即有成员且成员之间的序是无关紧要的。类似地，用容器成员属性只是描述了一个容器资源有一些事物作为其成员，并不是说这些事物是容器仅有的成员。例如，上面的三元组只说 Fred, Wilma,和 Dino 是委员会的成员，但没有说他们是委员会的所有成员。

同样，例 14 and 例 15 显示了描述容器的一个通用模式。然而，很重要的一点是，RDF 并不强制使用 RDF 容器词汇的方式，因此，可能会有其他使用容器的方式。例如，有时候让容器资源有一个 URIref 比作为一个匿名节点更为恰当。此外，还可能有不像前面的例子那样格式规范的容器词汇使用方式，例如，例 16 显示了一个用 RDF/XML 表示的和图 15 中的替换容器类似的图，但它显式地写出了容器成员属性，而不是由 rdf:li 自动生成：

例 16：一个格式不正确的替换容器的 RDF/XML 表示

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
         xmlns:s="http://example.org/packages/vocab#">



  <rdf:Description rdf:about="http://example.org/packages/X11">

    <s:DistributionSite>

      <rdf:Alt>

        <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>

        <rdf:_2 rdf:resource="ftp://ftp.example.org"/>

        <rdf:_2 rdf:resource="ftp://ftp1.example.org"/>

        <rdf:_5 rdf:resource="ftp://ftp2.example.org"/>

      </rdf:Alt>

    </s:DistributionSite>

  </rdf:Description>

</rdf:RDF>
```

如[RDF-SEMANTICS]所说，RDF 没有对 RDF 容器词汇的使用施加格式正确性约束,因此例 16 是合法的，尽管这个容器的类型即是包又是替换，属性 rdf:_2 还具有两个不同的值，还没有属性 rdf:_1, rdf:_3, 或 rdf:_4。

结果是，RDF 应用如果需要容器是格式正确的，则为了应用的鲁棒性，应该写代码检查容器词汇的使用恰当与否。

[编辑]

4.2 RDF 集合

在 4.1 节中描述的容器的一个缺点是没有办法封闭它，即没有办法说这些是容器的所有成员。一个容器只说一些有标识的资源是它的成员，无法说没有其他的成员了。而且，如果有一个图描述它的一些成员，我们没法排除在其他地方有图也描述这个容器的其它成员的可能。RDF 以 RDF 集合（collection）的形式提供了对描述特定成员的组的支持。一个 RDF 集合是用列表结构表示的一组事物，这个列表结构是用一些预定义的集合词汇表示的。RDF 的集合词汇包括属性 rdf:first 和 rdf:rest，和资源 rdf:nil.

为了展示这个，句子“上课程 6.001 的学生有：Amy, Mohamed, 和 Johann”可以用图 16 所示的图表示：

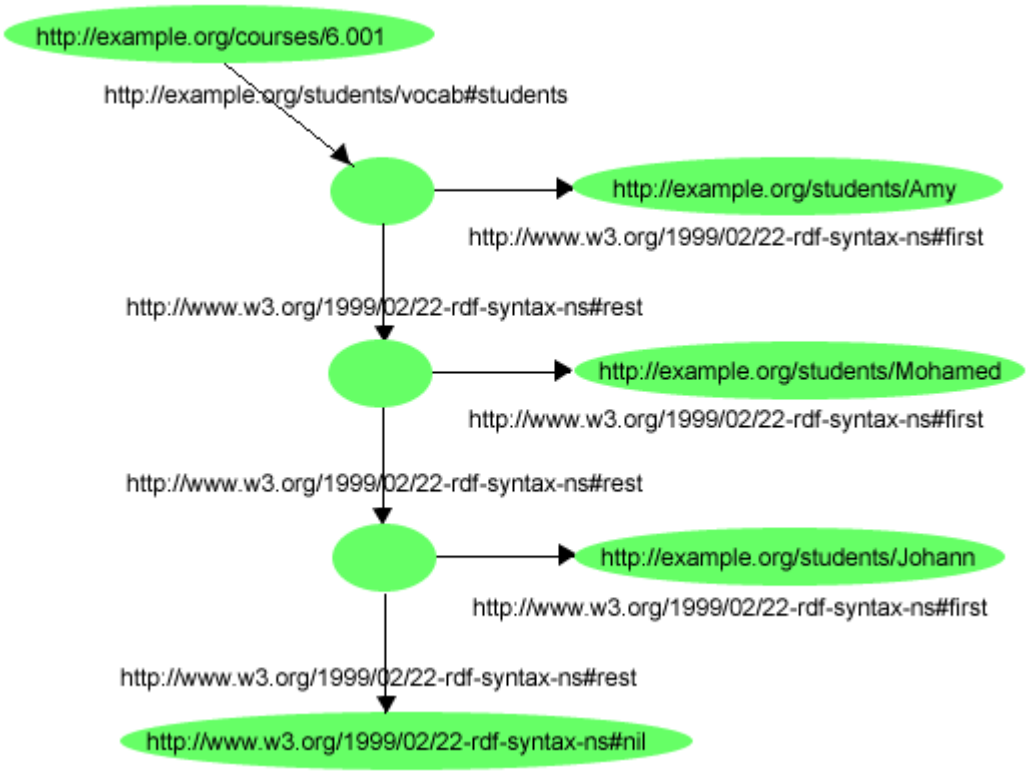

图 16：一个 RDF 集合（列表结构）

在这个图中，集合中的每个成员，如 s:Amy,是属性 rdf:first 的客体，这个属性的主体是表示一个列表的资源（这个例子中是一个匿名节点）。这个列表通过属性 rdf:rest 链到列表的剩下的元素，列表的结束用一个值为 rdf:nil 的属性 rdf:rest 表示（rdf:nil 表示一个空列表，它的类型是 rdf:List）。这种结构和 Lisp 语言很像，像 Lisp 一样，属性 rdf:first 和 rdf:rest 允许程序遍历这个结构。组成列表结构的每个匿名节点的类型默认为 rdf:List,尽管这没有在图中显示出来。RDF Schema 语言[RDF-VOCABULARY]定义了属性 rdf:first 和 rdf:rest 的主体的类型是 rdf:List,因此，这些匿名节点的类型信息可以推导出来，而不用每次都写出来。

RDF/XML 提供了一种特殊的图的记法使描述集合更为简单。在 RDF/XML 中，一个集合可以用一个代表属性的元素节点表示，且这个元素节点有个属性节点是 rdf:parseType="Collection"，这表示元素节点下的内容要以一种特殊的方式解释。在这里，rdf:parseType="Collection"表示包含的元素要用来创建 RDF 图中（其它的 rdf:parseType 属性的值将在后面的章节中讲述）。

为了展示 rdf:parseType="Collection"是怎么工作的，例 17 中的 RDF/XML 写法对应的图如 图 16 说示：

例 17:一个关于学生的集合的 RDF/XML 表示

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:s="http://example.org/students/vocab#">



   <rdf:Description rdf:about="http://example.org/courses/6.001">

      <s:students rdf:parseType="Collection">
```

```
            <rdf:Description rdf:about="http://example.org/students/Amy"/>

            <rdf:Description rdf:about="http://example.org/students/Mohamed"/>

            <rdf:Description rdf:about="http://example.org/students/Johann"/>

      </s:students>

   </rdf:Description>

</rdf:RDF>
```

在 RDF/XML 中用 rdf:parseType="Collection" 常定义了一个如 图 16 所示的列表结构：一个固定有限长度的列表，它用 rdf:nil 表示列表的结束，还用了一些相对于列表唯一的匿名节点。然后，RDF 并没有强制集合词汇只能用于这种方式，因此也可以用于其他方式。有些方式可能没有描述列表或者说封闭的集合。为了知道原因，注意到图 16 所示的图也可以用集合词汇的普通格式（不用 rdf:parseType="Collection"）写，如例 18 所示：

例 18：一个关于学生的集合的 RDF/XML 普通格式表示

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:s="http://example.org/students/vocab#">



<rdf:Description rdf:about="http://example.org/courses/6.001">

   <s:students rdf:nodeID="sch1"/>

</rdf:Description>



<rdf:Description rdf:nodeID="sch1">

   <rdf:first rdf:resource="http://example.org/students/Amy"/>

   <rdf:rest rdf:nodeID="sch2"/>

</rdf:Description>



<rdf:Description rdf:nodeID="sch2">
```

```
    <rdf:first rdf:resource="http://example.org/students/Mohamed"/>
    <rdf:rest rdf:nodeID="sch3"/>
 </rdf:Description>

 <rdf:Description rdf:nodeID="sch3">
    <rdf:first rdf:resource="http://example.org/students/Johann"/>
    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
 </rdf:Description>
 </rdf:RDF>
```

注意到在 [RDF-SEMANTICS] (和容器词汇相同的情况 4.1 节)中, RDF 并没有对集合词汇施加格式正确性约束（well-formedness），因此，当用普通格式写时，有可能写出的列表结构并不像由用 rdf:parseType="collection"自动生成的那样正确，如一个节点的 rdf:first 属性有两个不同的值是合法的，或列表结构的最后的节点不是空列表，或简单地忽略了结合的某些部分。另外，用普通格式写的集合词汇定义的图可以用 URIrefs 标识列表中的部分节点，而不是在列表中是唯一的匿名节点，这样，有可能在别的图中有三元组为集合添加元素，而使之不封闭。

结果是，RDF 应用如果需要集合是格式正确的，为了应用的鲁棒性，就应该检查集合词汇时被正确的使用。另外，语言 OWL [OWL], 能够对 RDF 图的结构定义额外的限制，可以避免一些集合格式不正确的情况。

[编辑]

4.3 RDF 具体化

有时候 RDF 应用需要描述 RDF 陈述，例如，为了记录 RDF 陈述发表的时间，作者和其他类似的信息（有时这些信息称为"来源"信息）。在 3.2 节的例 9 中，我们描述了一个帐篷，其 URIref 是 exproducts:item10245, 被 example.com 出售。其中一个三元组描述了帐篷的重量：

```
 exproducts:item10245    exterms:weight    "2.4"^^xsd:decimal .
```

如果能记录是谁提供了这个信息，可能对 example.com 有用。

RDF 提供了用以描述 RDF 陈述的内置词汇。用这些词汇对一个陈述的描述称为这个陈述的具体化(reification)。RDF 具体化词汇包含类 rdf:Statement，属性 rdf:subject, rdf:predicate, 和 rdf:object。然而，用 RDF 具体化词汇需要谨慎，因为很容易想到这些词汇定义了一些实际上没有定义事物。这个问题会再这节的后面讨论。

用 RDF 具体化词汇，关于帐篷重量的陈述的具体化可以通过给这个陈述一个 URIref，如 exproducts:triple12345(这样可以写陈述去描述它)，可以通过以下陈述描述它：

```
exproducts:triple12345   rdf:type        rdf:Statement .

exproducts:triple12345   rdf:subject     exproducts:item10245 .

exproducts:triple12345   rdf:predicate   exterms:weight .

exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .
```

这些陈述是说：由 URIref exproducts:triple12345 标识的资源是一个 RDF 陈述，且这个陈述的主体是 exproducts:item10245 标识的资源，这个陈述的谓词是 exterms:weight 标识的资源，这个陈述的客体是有类型文字"2.4"^^xsd:decimal 标识的小数值。假设原来的陈述的确是被 exproducts:triple12345 标识的，通过比较原来的陈述和这个陈述的具体化，很清楚具体化的确描述了原来的陈述。RDF 具体化词汇的惯用法通常包含了用这个模式描述的四个陈述，因此，这四个陈述有时被称为“具体化四元组”（reification quad）。

根据这个惯例用 RDF 具体化，example.com 可以这样记录 John Smith 发表了关于帐篷重量的陈述的事实：首先赋予这个陈述一 URIref（如，exproducts:triple12345），用上面描写的具体化描述这个陈述，然后再加一个语句，即 exproducts:triple12345 是由 John Smith 写的(用一个 URIref 标识“John Smith”)。结果是:

```
exproducts:triple12345   rdf:type        rdf:Statement .

exproducts:triple12345   rdf:subject     exproducts:item10245 .

exproducts:triple12345   rdf:predicate   exterms:weight .

exproducts:triple12345   rdf:object      "2.4"^^xsd:decimal .

exproducts:triple12345   dc:creator      exstaff:85740 .
```

这原来的陈述，和具体化以及作为陈述的作者 John Smith 形成了如图 17 所示的图:

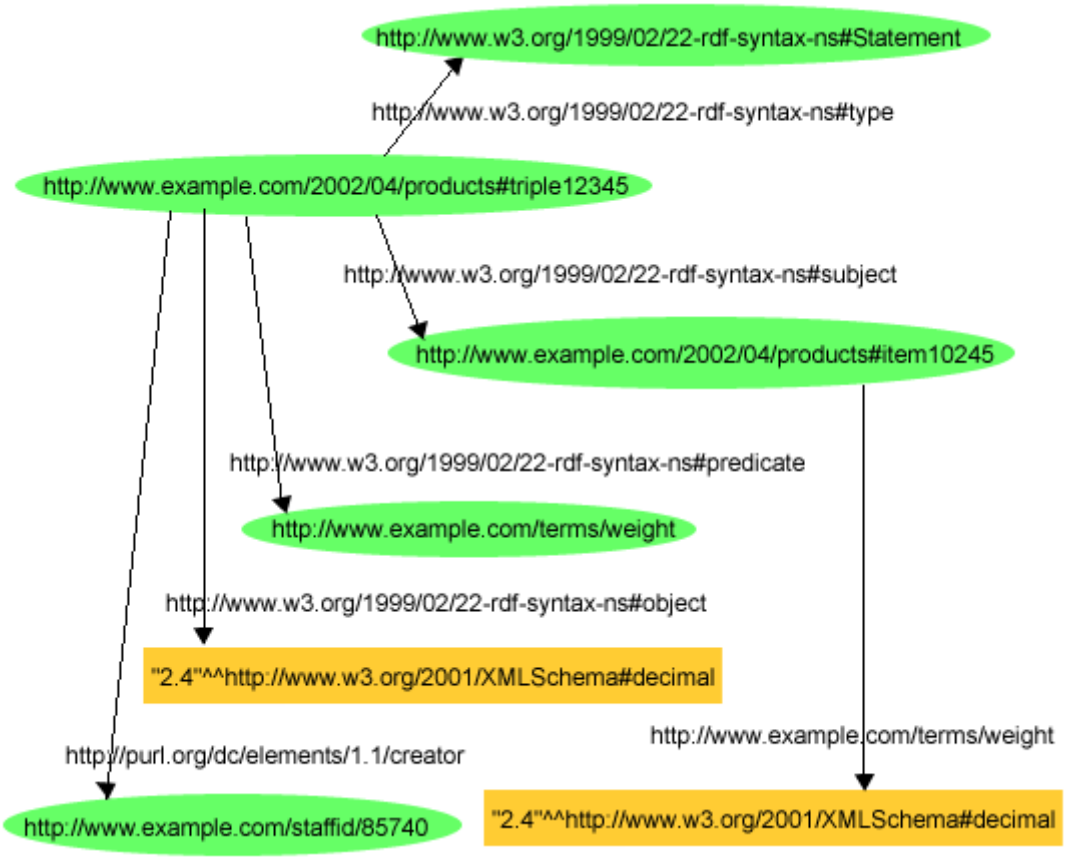

图 17：一个陈述及其具体化和属性

这个图用 RDF/XML 的写法如例 19 所示：

例 19：一个具体化例子的 RDF/XML 表示

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:dc="http://purl.org/dc/elements/1.1/"

         xmlns:exterms="http://www.example.com/terms/"

         xml:base="http://www.example.com/2002/04/products">

  <rdf:Description rdf:ID="item10245">

    <exterms:weight rdf:datatype="&xsd;decimal">2.4</exterms:weight>

  </rdf:Description>

  <rdf:Statement rdf:about="#triple12345">
```

```
    <rdf:subject rdf:resource="http://www.example.com/2002/04/products#item10245"/>
    <rdf:predicate rdf:resource="http://www.example.com/terms/weight"/>
    <rdf:object rdf:datatype="&xsd;decimal">2.4</rdf:object>
    <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Statement>
</rdf:RDF>
```

3.2 节介绍了使用 RDF/XML 的 rdf:ID 属性于 rdf:Description 元素中来简化称述主体的 URIref。rdf:ID 也可以用在属性元素节点中，用来自动产生一个属性元素生成的三元组的具体化。例 20 显示了这样可以表示和例 19 同样的图:

例 20：用 rdf:ID 生成具体化表示

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:exterms="http://www.example.com/terms/"
         xml:base="http://www.example.com/2002/04/products">
  <rdf:Description rdf:ID="item10245">
    <exterms:weight rdf:ID="triple12345" rdf:datatype="&xsd;decimal">2.4
    </exterms:weight>
  </rdf:Description>
  <rdf:Description rdf:about="#triple12345">
    <dc:creator rdf:resource="http://www.example.com/staffid/85740"/>
  </rdf:Description>
```

```
</rdf:RDF>
```

在这种情况下，把属性节点 rdf:ID="triple12345"放在 exterms:weight 元素中，达到与原来描述帐篷重量的三元组同样的效果：

```
exproducts:item10245    exterms:weight    "2.4"^^xsd:decimal .
```

和一些具体化三元组：

```
exproducts:triple12345    rdf:type         rdf:Statement .

exproducts:triple12345    rdf:subject      exproducts:item10245 .

exproducts:triple12345    rdf:predicate    exterms:weight .

exproducts:triple12345    rdf:object       "2.4"^^xsd:decimal .
```

这些具体化三元组的主体是一个 URIref，它是由文档的基准 URI（在 xml:base 中声明），字符“#“和 rdf:ID 的值连接而成。也就是说，这些三元组和前面的例子一样，具有相同的主体：exproducts:triple12345。

注意：对具体化做断言和对原来的陈述做断言是不同的，并且，它们中的一个成立不能推出另一个成立。也就是说，当有的人说，John 说了些关于帐篷重量的东西，他们并不是在陈述一个帐篷的重量，他们是在陈述 John 说了什么。相反地, 当某人（用一个陈述）描述了一个帐篷的重量，他并没有对这个陈述做出其他的陈述（因为他可能没有兴趣说关于“陈述”的事情）。

上述说法在一些地方被称为“具体化的惯例用法”。就像前面说到的一样，用 RDF 具体化词汇的时候要谨慎，因为很容易设想这些词汇定义了一些其实并没有定义的事情。如果有应用程序成功使用具体化词汇，他们是通过遵循某些惯例，做了一些假设做到的，这些惯例和假设都是在 RDF 为具体化词汇定义的真正意义之外的，也是在 RDF 为了支持他们而提供的真正辅助之外的。

首先，重要的是，在具体化的惯例用法中，具体化三元组的主体是假设为标识了在一个特定的 RDF 文档中特定的三元组，而不是一些任意的具有相同的主体，谓词，客体的三元组。使用这个惯例是因为具体化是被设计用于表达一些属性，如书写的日期，RDF 数据远信息，就像例子中已经给出的那样，这些属性必须应用于三元组的特定实例。有可能有多个三元组具有相同的主体，谓词，客体，另外，尽管图是定义为一个三元组的“集合”，多个结构相同的三元组实例可能出现在不同的文档中。因此，为了完全支持这个惯例，需要一些方式来将具体化三元组的主体和一些文档中个体三元组关联起来。然而，RDF 没有提供办法实现它。

例如，在上面的例子中，在三元组中和 RDF/XML 中都没有显式的信息说那个描述帐篷重量的陈述就是资源 exproducts:triple12345，它是四个具体化三元组的主体，也是 John Smith 做出的陈述。这可以从图 17 看出。原来的陈述当然是图的一部分，但就图上的信息而言，exproducts:triple12345 是一个单独的资源，而不是标识了图中的那个陈述。RDF 没有提供内置的方式来表明一个像 exproducts:triple12345 的 URIref 是怎样关联到一个特定的陈述或图的，和没有提供内置的方式来表明一个像 exproducts: item10245 的 URIref 是怎样关联到一个具体的帐篷一样。关联特定的 URIref 到特定的资源（这里是陈述）必须通过 RDF 以外的机制实现。

用如例 20 所示的 rdf:ID 能够自动生成具体化三元组，这提供了一种方便的办法来表明用在具体化中的陈述的主体 URIref。而且，它提供了一个“窍门”用以把具体化三元组和创建他们的 RDF/XML 语法部分关联起来，因为 rdf:ID 属性的值 triple12345 是用于生成具体化三元组的主体 URIref。然而，这个关联还是在 RDF 之外的，因为生成的三元组中没有任何东西显式地说原来的三元组有个 URIref 是 exproducts:triple12345（RDF 并不假设在一个 URIref 和任何使用了这个 URIref 或其缩写形式的 RDF/XML 之间有任何关系）。

缺乏内置的方式赋予陈述 URIref 并不意味着这种类型的来源（provenance）信息无法在 RDF 中表示，而是这无法仅仅用 RDF 关联给具体化词汇的语义达到。例如，如果一个 RDF 文档（如，一个网页）有个 URI，这可以对这个 URI 表示的资源做出陈述，然后，通过一些应用相关的对这些陈述的解释，一个应用可以认为这些陈述对文档中的全部陈述起作用。另外，如果有些 RDF 之外的机制可用以赋予 RDF 陈述 URI，当然可以做出关于陈述(用这些 URI 标识)的陈述。然而，这样的话，没有必要严格遵照具体化词汇的惯例用法。

为了明白这点，假设原来的陈述：

```
exproducts:item10245    exterms:weight    "2.4"^^xsd:decimal .
```

有一个 URIref exproducts:triple12345，这个陈述可以简单地通过下面的陈述让其归属于 John Smith：

```
exproducts:triple12345    dc:creator    exstaff:85740 .
```

而没有用具体化词汇（尽管加上 exproducts:triple12345 的 rdf:type 是 rdf:Statement 可能是有益的）。

另外，具体化词汇在使用时，可以直接遵照上述的惯例，并附带应用相关的对如何把特定的三元组和其具体化关联起来的理解。然而，其他收到这个 RDF 数据的应用不一定能够分享这个应用相关的理解，因此不一定能够正确地解释这个图。

还有一点需要重视：这里讲述的具体化的解释和一些语言中的“引用”（quotation）不同。具体化描述了一个特定的三元组的实例和这三元组提及的资源之间的关系，具体化可以直观地理解为：这个三元组说的是这些事物，而不是在引用中的：这个三元组有这个形式。例如，在这节中的具体化例子中，三元组

exproducts:triple12345 rdf:subject exproducts:item10245 .

描述了原来的陈述的 rdf:subject, 是说陈述的主体是由 URIref exproducts:item10245 标识的资源（那个帐篷）。它不是像引用那样说陈述的主体是这个 URIref 本身（一个由一些字符开头的字符串）。

4.4 关于结构化值（rdf:value）的更多信息

2.3 节注意到 RDF 模型本质上只支持二元关系，也就是说，一个陈述表示了两个资源之间的关系。例如，陈述：

```
exstaff:85740    exterms:manager    exstaff:62345 .
```

说两个雇员之间有 exterms: manager 关系（一个管理另一个）。

然而，有些情况下有必要在 RDF 中表示多元关系（多于两个资源之间的关系）。2.3 节 讨论了一个例子，这个例子的问题是要表示 John Smith 和他的地址信息，即一个由街道，城市，州，邮政编码组成的结构。可以看出这个地址信息是一个 5 元关系。

```
address(exstaff:85740, "1501 Grant Avenue", "Bedford", "Massachusetts", "01730")
```

2.3 节注意到这种结构化的信息可以通过聚集一部分信息作为一个单独的资源（这里，John's 的地址信息），然后对这个新资源发表单独的陈述，如下面的三元组所示：

```
exstaff:85740   exterms:address        _:johnaddress .

_:johnaddress   exterms:street         "1501 Grant Avenue" .

_:johnaddress   exterms:city           "Bedford" .

_:johnaddress   exterms:state          "Massachusetts" .

_:johnaddress   exterms:postalCode     "01730" .
```

（其中_:johnaddress 是表示 John's 的地址的匿名节点的标识）这是在 RDF 中表示 N 元关系的一般方法：选出关系中的一个元素（这里是 John）作为关系（这里是 address）的主体，指定一个中间的资源来表示关系中剩下的部分，然后把关系剩下部分都作为这个中间资源的属性。

在 John 的地址信息中，没有一个部分能够作为属性 exterms:address 的主值（main value）,所有部分对值的贡献是相同的。然后，有时候，结构中的一部分可以作为结构的主值，其他部分提供上下文信息或其他对主值的修饰信息。例如，在 3.2 节的例 9 中，一个帐篷的重量的值是小数 2.4,也就是，

```
exproduct:item10245   exterms:weight   "2.4"^^xsd:decimal .
```

实际上，更为复杂的对重量的描述应该是 2.4 千克（ kilograms），而不是仅仅小数 2.4。为了表示这个，属性 exterms:weight 的值还需要 2 个部分：作为类型文字的数值和量度的单位（千克）。在这种情况下，数值可以看成属性 exterms:weight 的主值，因为一般这个值都是一个类型文字，而依赖对上下文信息来填充这个值的单位。

在 RDF 模型中，这种类型的带有修饰的属性值可以看成一种简单的结构化值。为了表达这个，一个单独的资源可以用来表示结构化值，且作为原来陈述的客体。这个资源可以有属性表示结构值中的部分。在这里，应该有个属性表示数值，还一个属性表示单位。RDF 提供了一个预定义的属性 rdf:value 来描述结构值中的主值。因此，这里的数值可以作为 rdf:value 的值，资源 exunits:kilograms 作为属性 exterms:units 的值（假设资源 exunits:kilograms 定义为 example.org 的词汇集的一部分）。结果，这些三元组为：

```
exproduct:item10245   exterms:weight   _:weight10245 .

_:weight10245         rdf:value        "2.4"^^xsd:decimal .

_:weight10245         exterms:units    exunits:kilograms .
```

这可以用 RDF/XML 表示，如例 21 所示：

例 21：用 rdf:value 的 RDF/XML 表示

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:exterms="http://www.example.org/terms/">



  <rdf:Description rdf:about="http://www.example.com/2002/04/products#item10245">

    <exterms:weight rdf:parseType="Resource">

      <rdf:value rdf:datatype="&xsd;decimal">2.4</rdf:value>

      <exterms:units rdf:resource="http://www.example.org/units/kilograms"/>

    </exterms:weight>

  </rdf:Description>



</rdf:RDF>
```

例 21 也显示了对 4.2 节介绍的属性 rdf:parseType 的用法，在这里，rdf:parseType="Resource"用以表示节点的内容同时被解释为一个新的匿名节点，而不用写一个嵌套的 rdf:Description 元素。rdf:parseType="Resource" 用在属性 exterms:weight 元素节点中表示创建了一个匿名节点作为属性的值，且包含的元素 (rdf:value 和 exterms:units) 描述了这个匿名节点的属性。关于 rdf:parseType="Resource"更详细的信息在[RDF-SYNTAX].

相同的方法也可用于表示具有任何单位的数量，用 rdf:value 属性表示主值，用其他的属性表示关于数量的其他信息。

但没有必要用 rdf:value 去取代用户自定义的属性，如例 21 中的属性 exterms:amount, 因为 RDF 并没有给 rdf:value 特殊的解释。它只是作为表示这种情形的惯例。

然而，尽管大部分数据库中和 Web 上的数据用简单的数值作为属性（如，重量，价格等）的值，这种简单的数值往往不足够描述这些值。在像 Web 这样的环境，一般来说，假设其他人访问属性的值的时候能知道它的单位（或其他上下文相关的信息）是不稳妥的。例如，一个美国的网站给出重量的单位默认是磅，但非美国的人访问这个数据的时候常假设单位是公斤。对 Web 上数据的正确解释可能需要其他的信息，如单位信息。这有很多种方法：例如，用 rdf:value,把单位信息放在属性名字种，如属性 exterms:weightInKg 包含了数据的单位信息。或者，加上用户自定义的的信息，如 exterms:unitOfWeight, 在物品或产品实例的描述中，在一个数据集的描述中，或者在 Schema 中（参考第 5 节）

[编辑]

4.5 XML 文字

有时候，属性的值可能是一个 XML 的片断，或包含 XML 标记的文本。例如，出版社可能要维护一些 RDF 元数据，包括书和文章的标题。当然，这些标题可能是简单的字符串，但并不总是这样。例如，数学书的标题可能包含用 MathML [MATHML]表示的数学公式。标题还可能因为其他原因包含标记，如 Ruby 标注 [RUBY], 或者双相的显示特殊的浮雕型变量。 ([CHARMOD])

RDF/XML 提供了一个特殊的符号，使书写这种文字变得简单。即通过属性 rdf:parseType 第三个值，如果一个元素节点的有属性节点 rdf:parseType="Literal",表明这个元素节点的内容应该被解释为一个 XML 片断。例 22 演示了这种用法。

例 22：一个 XML 文字的 RDF/XML 表示

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:dc="http://purl.org/dc/elements/1.1/"

         xml:base="http://www.example.com/books">



  <rdf:Description rdf:ID="book12345">

    <dc:title rdf:parseType="Literal">

      <span xml:lang="en">

        The <em>&lt;br /&gt;</em> Element Considered Harmful.

      </span>

    </dc:title>

  </rdf:Description>



</rdf:RDF>
```

Example 22 中的 RDF/XML 描述了一个包含了一个三元组的图，这个三元组的主体是 ex:book12345, 谓词是 dc:title. 属性节点 rdf:parseType="Literal" 表示元素节点 <dc:title> 下的内容是一个 XML 片断，即作为属性 dc:title 的值。在这个图中，值是类型为 rdf:XMLLiteral 的类型文字。[RDF-CONCEPTS] 特意定义 rdf:XMLLiteral 作为表示 XML 片断的数据类型。这些 XML 片断都是根据 XML Exclusive Canonicalization recommendation [XML-XC14N]规范化了的，这导致了一些转换，如为 XML 片断加上已用的名字空间声明，统一的字符转义或去转义，空元素的扩展。（因为这些原因以及三元组的标注本身需要一些转义符，这里不显示真正的类型文字。RDF/XML 提供了属性 rdf:parseType="Literal" 使 RDF 用户不用进行这些转换。详细信息请参考[RDF-CONCEPTS] 和 [RDF-SYNTAX].) 一些上下文属性，如 xml:lang 和 xml:base 并不会从 RDF/XML 文档中继承，如果需要的化，如上例所示，必须在 XML 片断中给出。

这个例子表明了书写 RDF 数据是要谨慎。乍看，标题的值是用平凡文字表示的简单字符串，后来又发现会包含 XML 标记。如果一个属性的值有时包含 XML 标记，有时有不包含，应该都用 rdf:parseType="Literal"，或者软件应该能处理属性的值可能是平凡文字和 rdf:XMLLiteral 类型的文字的情况。

[编辑]

5. RDF Schema — 定义 RDF 的词汇表

RDF 使用命名特性和值来表达与资源有关的简单声明。但是，在某些情况下，用户希望能够根据需要自定义一些词汇，然后用这些词汇来描述资源。这些词汇表明用户正在描述某种资源，并且会采用某些特定的特性来描述。例如，3.2 节所举例子中，example.com 公司想要描述一个类 exterms:Tent，并且使用特性 exterms:model，exterms:weightInKg 和 exterms:packedSize 来描述它们。（这些类和特性的名字之前带有各种“example”名称空间前缀，表明：在 RDF 中，这些名字其实都是 RUI 引用，2.1 小节中已经讨论过这个问题）。类似的情况，那些对图书资源比较感兴趣的 RDF 开发人员可能会描述 ex2:Book 或者 ex2:MagazineArtical 这样的类，并使用 ex2:author, ex2:title ,ex2:subject 之类的特性来描述这些类。其它的一些应用可能会描述诸如 ex3:Person、ex3:Company 这样的类和 ex3:age 、ex3:jobTitle、ex3:stockSymbol 、ex3:numberofEmployee 之类的一些特性。RDF 本身并不能针对特定应用需求来定义一些类和特性。这些类和特性被称为 RDF 词汇，它们需要通过 RDF 词汇描述语言：RDF Schema（RDF 的一种扩展语言） 来定义。

RDF Schema 并没有针对特定应用提供诸如 exterms:Tent , ex2:Book 或者 ex3:Person 这样的一些类或者是诸如 exterms:weightInKg ex2:author 或者 ex3:JobTitle 这样的一些特性词汇。RDFS 只是提供了描述一种这些类和特性的能力 (facilities)，并且可以暗示某些类和特性期望合在一起使用（例如,特性 ex3:jobTitle 应该用来描述 ex3:Person）。换句话说，RDF Schema 为 RDF 提供了一个类型系统。RDF Schema 类型系统在某些方面类似于 Java 这样的面向对象编程语言的类型系统。例如，RDF Schema 允许资源被定义为一个或者多个类的实例。另外，RDFS 通常把类组织成为一种分级结构；例如，类 ex:Dog 可以定义为 ex:Mammal 的子类，而 ex:Mammal 又是 ex:Animal 的子类。 如果某个资源是类 ex:Dog 的实例，那么隐含意味着它也是 ex:Animal 的实例。然而，RDF 类和特性在某些方面又与编程语言的类型系统有着明显的差异。RDF 类和特性并没有过多束缚资源的描述方式，而是提供了一些关于 RDF 资源的额外描述信息。这些添加的信息可以通过各种方式来使用，具体内容将在 5.3 节中介绍。

RDF Schema 所具有的这些能力本身也是以 RDF 词汇形式提供的。也就是说，这些 RDF 词汇是一组带有特殊含义的、预定义的 RDF 资源。 这些资源的（RDF Schema 词汇）URI 带有前缀 http://www.w3.org/2000/01/rdf-schema# （QName 通常采用前缀 rdfs:）。采用 RDF Schema 语言所定义的词汇描述（schemas）也是合法的 RDF 图。因此，即使一个软件不是专为处理新加的 RDF Schema 词汇而开发的，它仍然可以将 schema 解释为一个包含了各种资源和特性的合法 RDF 图，但是这个软件并不能“理解”新添加的 RDF Schema 术语的内在含义。为了理解新加术语的含义，RDF 软件必须能够处理一种扩展语言。这种扩展语言不仅仅包含 rdf:前缀的词汇，而且还包含了 rdfs:前缀的词汇，以及这些词汇的内在含义。下一小节中将对这一点进行详细分析。

下一小节将详细分析 RDF Schema 的基本资源和特性。

[编辑]

5.1 描述类

描述过程通常从划分被描述事物的种类开始。RDF Schema 把*事物的种类*称之为*类（class）*。RDF Schema 中的*类（class）*与我们通常所说的*类型（Type）*或者*分类（Category）*基本相同，有点类似于面向对象编程语言（比如 Java）中的类（class）的概念。RDF 类可以用来表示事物的任何分类，例如网页、人、文档类型、数据库或者抽象概念等。类可以通过 RDF Schema 中的资源（rdfs:Class 和 rdfs:Resource）以及特性（rdf:type 和 rdfs:subClassOf）来表示。

例如，假设 example.org 这个组织想用 RDF 来提供有关不同种类机动车（Motor vehicles）的信息，那么它在 RDF Schema 中首先需要一个代表机动车这一分类（category）的类（class）。属于某个类（class）的资源被称为该类的*实例（instances）*。在本例中，所有是机动车的资源都是这个类的实例。

在 RDF Schema 中，一个*类*是任何具有 rdf:type 特性、并且该特性的值为 rdfs:Class 的资源。因此，可以这样来描述一个机动车类：为该类指定一个 URIref，比如 ex:MotorVehicle（其中的前缀 ex: 代表 URIref http://www.example.org/schemas/vehicles，ex: 被用作 example.org 词汇表中的所有 URIrefs 的前缀），然后将这个机动车类描述为一个具有 rdf:type 特性、并且特性值为 rdfs:Class 的资源。也就是说，example.org 应该编写如下的 RDF 声明：

```
ex:MotorVehicle   rdf:type   rdfs:Class .
```

正如在 3.2 节中所讨论的那样，特性 rdf:type 用来表明一个资源是某个类的实例。因此，将 ex:MotorVehicle 描述为类之后，可以用下面的 RDF 声明来描述资源 exthing:companyCar 是一个机动车：

```
exthings:companyCar   rdf:type   ex:MotorVehicle .
```

（在书写上述 RDF 声明时，遵守了一些约定，即：将类名的首字母大写，而将特性和实例名称的首字母小写。但是，RDF Schema 并不强制要求这一点。上述 RDF 声明也假定：example.org 已分别为事物的类（classes）和事物的实例（instances）定义了两个不同的词汇表[译注//这两个词汇表分别在 ex:和 exthings:所代表的命名空间里定义]。）

rdfs:Class 本身也是资源，而且也有一个 rdf:type 特征并且该特性的值为 rdfs:Class。一个资源可以是一个或多个类的实例.

example.org 在描述了 ex:MotorVehicle 类之后，也许还要再定义一些类来代表各种特定种类的机动车，例如客车（passenger vehicles）、大篷货车（vans）、小型货车（minivans）等等。这些类可以采用和类 ex:MotorVehicle 同样的方法来定义：首先为每个新建的类指定一个 URIref，然后编写如下的 RDF 声明将这些资源定义为类：

```
ex:Van     rdf:type   rdfs:Class .

ex:Truck   rdf:type   rdfs:Class .
```

但是上面这些声明本身只是对单个的类的定义。example.org 可能还要描述上面定义的类与类 ex:MotorVehicle 之间的特定关系，比如它们是机动车（MotorVehicle）中的*一种*

这种两个类之间的特化关系（specialization relationship）可以用预定义的特性 rdfs:subClassOf 来描述。例如，可以通过编写下面这条 RDF 声明来描述 ex:Van 是一种特殊的 ex:MotorVehicle：

```
ex:Van   rdfs:subClassOf   ex:MotorVehicle .
```

在上面的例子中，rdfs:subClassOf 的含义是：任何 ex:Van 类的实例同时也是 ex:MotroVehicle 类的实例。因此，如果资源 exthings:companyVan 是 ex:Van 的一个实例，那么理解 RDF Schema 词汇表的 RDF 软件可以根据上面声明的 rdfs:subClassOf 关系推理出额外的有用信息，即 exthings:companyVan 也是 ex:MotorVehicle 的一个实例。

exthings:companyVan 的例子说明了一点，即 RDF Schema 可以定义一种扩展语言。但是，RDF 本身并没有定义 RDF Schema 中的词汇（比如 rdfs:subClassOf）所具有的含义。因此，即时某个 RDF schema[译注//注意 RDF Schema 与 RDF schema 的区别]定义了 ex:Van 和 ex:MotorVehicle 之间的 rdfs:subClassOf 关系（通过编写一个 RDF 声明），但是对于不理解 RDF Schema 术语的 RDF 软件来说，这个 RDF 声明只是一个以 rdfs:subClassOf 为谓词的三元组，该 RDF 软件并不能理解 rdfs:subClassOf 所代表的含义，因此也不能据此得到额外的结论：exthing:companyVan 也是 ex:MotorVehicle 的实例。

rdfs:subClassOf 特性具有*传递性（transitive）*。也就是说，如果给出下列 RDF 声明：

```
ex:Van       rdfs:subClassOf   ex:MotorVehicle .

ex:MiniVan   rdfs:subClassOf   ex:Van .
```

这些声明可以推理出 ex:MiniVan 同时也是 ex:MotroVehicle 的子类。相应的，如果 RDF Schema 定义了某个资源是 ex:MiniVan 的实例，那么它也是 ex:MotorVehicle 的实例（同时也是类 ex:Van 的实例）。一个类可以是一个或多个类的子类。（例如：ex:MiniVan 可以同时为 ex:Van 和 ex:PassengerVehicle 的子类）。RDF Schema 规定：所有的类总是 rdfs:Resource 的子类（因为任何类的实例都是资源）。

图 18 是上面这些例子的一个完整的类层次图。

图 18：车辆类层次图

为了简化起见，图 18 中将连接类和 rdfs:Class 的 rdf:type 特性全部省略了。事实上，根据 RDF Schema 的定义，一个以 rdfs:subClassOf 为谓词的声明中的主体和客体都应该是 rdfs:Class 类型的资源。因此，这些信息可以推理出来，不过在实际编写 schema 的时候，明确声明这些信息是一种比较好的编码风格。

上图所示的 schema 可以用下列三元组来描述：

```
ex:MotorVehicle       rdf:type          rdfs:Class .

ex:PassengerVehicle   rdf:type          rdfs:Class .

ex:Van                rdf:type          rdfs:Class .
```

```
ex:Truck              rdf:type          rdfs:Class .

ex:MiniVan            rdf:type          rdfs:Class .

ex:PassengerVehicle   rdfs:subClassOf   ex:MotorVehicle .

ex:Van                rdfs:subClassOf   ex:MotorVehicle .

ex:Truck              rdfs:subClassOf   ex:MotorVehicle .

ex:MiniVan            rdfs:subClassOf   ex:Van .

ex:MiniVan            rdfs:subClassOf   ex:PassengerVehicle .
```

上述 schema 可用例 23 所示的 RDF/XML 来书写。

例 23：用 RDF/XML 书写的车辆类层次图

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF

  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

  xml:base="http://example.org/schemas/vehicles">


<rdf:Description rdf:ID="MotorVehicle">

  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>

</rdf:Description>


<rdf:Description rdf:ID="PassengerVehicle">

  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>

  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>

</rdf:Description>
```

```
<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

</rdf:RDF>
```

正如 3.2 节以及例 13 中所讨论的，RDF/XML 提供了一种简写形式来描述具有 rdf:type 特性的资源（*类型结点*）。因为 RDF Schema 的类同时也是 RDF 资源，这种简写形式可以应用于对类的描述。上述的 schema 也可以使用这种简写形式来描述，如例 24 所示：

例 24：使用类型节点的机动车类层次简写形式

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

</rdf:RDF>
```

本节其余部分都将采用相似的类型节点简写形式。

例 23 和例 24 中的 RDF/XML 为那些具有 rdf:ID 特性的资源（类）指定了资源名称（例如：MotorVehicle），这相当于为资源（类）指派了一个相对于 schema 文档的 URIrefs（参见 3.2 节的描述）。这里的 rdf:ID 非常有用，因为它不仅仅是 URIrefs 的简写形式，同时还提供了一种额外检查，即确保 rdf:ID 属性的值相对于当前的基准 URI（通常的情况下就是本文档 URI）是唯一的。在 RDF schemas 中定义类和特性的名称时，这种机制有助于发现重复的 rdf:ID 值。在同一个 schema 中，基于这些名字的相对 URIrefs 可以在其它类的定义中被引用。(例如，可以在其它类的描述中用#MotorVehicle

来引用 MotorVehicle 类)。假定 schema 本身的 URI 是 http://example.org/schemas/vehicles ，那么 MotorVehicle 类的完整 URIref 就是 http://example.org/schemas/vehicles#MotorVehicle （如图 18 所示）。正如 3.2 节所指出的，schema 可能被移动或复制（或者只是为 schema 中的类指定一个基准 URIref，而不假定它们被发布在同一个位置）。为了确保在这些情况下对 schema 中类的引用能够维持一致，可以在定义这些类的时候显式声明一个基准 URI（即增加属性 xml:base="http://example.org/schemas/vehicles" ）。显式声明 xml:base 是一种良好的编码风格，前面两个例子就是这样做。

为了在别处的 RDF *实例数据*（即描述属于某个类的个体的数据）中引用这些类，example.org 可采取下列方式：声明一个恰当的 xml:base，并书写相对 URIrefs，然后根据这两者所确定的绝对 URIrefs 来标识类；或者，声明一个恰当的名字空间，并书写 QNames，然后根据由 QName 展开得到的绝对 URIrefs 来标识类。例如，在例 25 所示的 RDF/XML 中，资源 exthings:companyCar 可被描述为类 ex:MotorVehicle（在例 24 中定义的）的一个实例：

例 25：一个 ex:MotorVehicle 类的实例

```
<?xml version="1.0"?>

<rdf:RDF

  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

  xmlns:ex="http://example.org/schemas/vehicles#"

  xml:base="http://example.org/things">


  <ex:MotorVehicle rdf:ID="companyCar"/>

</rdf:RDF>
```

注意：QName ex:MotorVehicle 将根据命名空间声明 xmlns:ex="http://example.org/schemas/vehicles#" 展开为完整的 URIref http://example.org/schemas/vehicles#MotorVehicle，即 MotorVehicle 类的 URIref（见图 18）。xml:base 声明 xml:base=http://example.org/things 用于将 rdf:ID="companyCar"展开为 exthings:companyCar 所对应的 URIref（这里没有采用前一种机制是因为 QName 不能作为 rdf:ID 属性的值）。

[编辑]

5.2 描述特性

用户除了描述他们想要描述的*类（classes）*，通常还需要能够定义刻画这些类的*特性（properties）*（例如用 rearSeatLegRoom 来描述一个客车）。在 RDF schema 中，特性是用 RDF 类 rdf:Property 以及 RDF Schema 特性 rdfs:domain（定义域）、rdfs:range（值域）以及 rdfs;subPropertyOf 来描述的。

RDF 中的所有特性都被描述为类 rdf:Property 的实例。因此一个新特性（例如 exterms:weigntInKg）的描述是通过为它指派一个 URIref，并使用一个值为 rdf:Property 的 rdf:type 特性来完成的。例如，书写如下的 RDF 声明：

```
exterms:weightInKg   rdf:type   rdf:Property .
```

RDF Schema 还提供了一些词汇用于描述如何在 RDF 数据中正确使用特性和类。其中最重要的一个信息是由 RDF Schema 特性 rdfs:range 和 rdfs:domain 提供的，它们用于进一步描述与应用相关的特性。

rdfs:range 用于表明某个特性的值（定义域）是给定类的实例。例如，如果 example.org 想要表明“特性 ex:author 的值是类 ex:person 的实例”，那么可以写出如下的 RDF 声明：

```
ex:Person   rdf:type     rdfs:Class .

ex:author   rdf:type     rdf:Property .

ex:author   rdfs:range   ex:Person .
```

上述声明表明：ex:person 是一个类，ex:author 是一个特性，并且对于使用 ex:author 特性的 RDF 声明，其客体（Object）是 ex:person 类的实例。

一个特性（比如 ex:hasMother）可以有零个、一个、或多个 range 特性。如果 ex:hasMother 没有 range 特性，那么对 ex:hasMother 特性的值而言，就没有什么限制。如果 ex:hasMother 有一个 range 特性限制，例如指定 ex:Person 作为其值域，这表明 ex:hasMother 特性的值是类 ex:Person 的实例。如果 ex:hasMother 有多个 range 特性，例如在指定 ex:person 为它的 range 特性的同时，还指定了 ex:Femal 作为它的 range 特性，这说明了 ex:hasMother 特性的值是*所有*被指定的类的实例。例如，ex:hasMother 的值既是一个 *ex:Femal* 又是一个 *ex:Person*。

最后一点并不是十分明显，即：实际上，为特性 ex:hasMother 指定两个 range（ex:Femal 和 ex:person）涉及到两个独立的声明：

```
ex:hasMother   rdfs:range   ex:Female .

ex:hasMother   rdfs:range   ex:Person .
```

对于任何用到上述特性的声明，比方说：

```
exstaff:frank   ex:hasMother   exstaff:frances .
```

为确保两个 rdfs:range 声明*都*正确，exstaff:frances 必须同时是 ex:Femal 和 ex:Person 类的实例。

rdfs:range 特性也可用于表明特性的值是一个类型文字（typed literal，见 2.4 节）。例如，如果 example.org 要表明特性 ex:age 的值来自 XML Schema 数据类型 xsd:integer，它应书写如下 RDF 声明：

```
ex:age   rdf:type     rdf:Property .

ex:age   rdfs:range   xsd:integer .
```

数据类型 xsd:integer 是通过 URIref（完整的 URIref 为 http://www.w3.org/2001/XMLSchema#integer ）来标识的。可以不在 schema 中显式声明该 URIref 标识某个数据类型而直接使 t 用它。然而，显式声明一个给定的 URIref 标识某个数据类型常常是有用的。这可以通过使用 RDF Schema 类 rdfs:Datatype 来完成。要声明 xsd:integer 是一个数据类型，example.org 应书写如下 RDF 声明：

```
xsd:integer   rdf:type   rdfs:Datatype .
```

这个声明的含义是：xsd:integer 是某个数据类型（它被假定符合[RDF-CONCEPTS]中所描述的 RDF 数据类型要求）的 URIref。这一声明并*不*构成一个数据类型的*定义*，即仿佛 example.org 是在定义一个新的数据类型。在 RDF Schema 中，没有可用来定义数据类型的方式。正如在 2.4 节中提到的，数据类型是在 RDF（甚至是 RDF Schema）之外定义、然后在 RDF 中通过它们的 URIrefs 来*引用*的。上述声明的作用仅仅是记录该数据类型的存在，并显式表明它在 schema 中被用到。

rdfds:domain 用于表明某个特性应用于指定的类（定义域）。例如，如果 example.org 要表明特性 ex:author 应用于类 ex:Book 的实例上，它应书写如下 RDF 声明：

```
ex:Book     rdf:type      rdfs:Class .

ex:author   rdf:type      rdf:Property .

ex:author   rdfs:domain   ex:Book .
```

上述声明表明 ex:Book 是类，ex:author 是特性。而使用 ex:author 特性的 RDF 声明以 ex:Book 的实例为主体。

一个给定的特性（例如 exterms:weight）可能有零个、一个或多个 domain 特性。如果 exterm:weight 没有 domain 特性，那么便没有规定 exterms:weight 特性对应的主体是某类资源，也就是说任何资源都可以作为 exterms:weight 特性的主体。如果 exterms:weight 有一个 domain 特性，例如指定 ex:Book 作为其定义域，这表明 exterms:weight 特性应用于类 ex:Book 的实例。如果 exterms:weight 有多个 domain 特性，例如一个 domain 特性指定其定义域为 ex:Book，另一个 domain 特性指定其定义域为 ex:MotorVehicle，这表明具有 exterms:weight 特性的资源是*所有*被指定为定义域的类的实例，即具有 exterms:weight 特性的资源既是 ex:Book 类的实例*又是* ex:MotorVehicle 类的实例（对待具体问题，需要认真分析、仔细地指定定义域和值域）。

与 rdfs:range 的情况一样，最后一点可能不是非常明显，即：实际上，为特性 exterms:weight 声明两个定义域（ex:Book 和 ex:MotorVehicle）涉及到两个独立的声明：BR>

```
exterms:weight   rdfs:domain   ex:Book .

exterms:weight   rdfs:domain   ex:MotorVehicle .
```

对于任何用到该特性的声明，比方说：

```
exthings:companyCar   exterms:weight   "2500"^^xsd:integer .
```

为了保证两个 rdfs:domain 声明*都*是正确的，必须保证 exthings:companyCar 同时是 ex:Book 和 ex:MotorVehicle 类的实例。

我们可以扩展车辆类的 schema 来举例说明值域（range）和定义域（domain）描述的使用方法：增加两个特性 ex:registeredTo 和 ex:rearSeatLegRoom，然后添加一个新的类 ex:Person，并显式地将 xsd:integer 声明为数据类型。ex:registerdTo 特性应用于任何 ex:MotorVehicle，并且它的值是 ex:Person。对本例而言，ex:rearSeatLegRoom 仅仅应用到 ex:PassengerVehicle 类的实例上，其值是一个 xsd:integer（该值给定了后座的脚部活动空间的厘米数）。例 26 显示了这些描述的 RDF/XML：

例 26：车辆 schema 的一些特性描述

```
<rdf:Property rdf:ID="registeredTo">

  <rdfs:domain rdf:resource="#MotorVehicle"/>

  <rdfs:range rdf:resource="#Person"/>

</rdf:Property>


<rdf:Property rdf:ID="rearSeatLegRoom">

  <rdfs:domain rdf:resource="#PassengerVehicle"/>

  <rdfs:range rdf:resource="&xsd;integer"/>

</rdf:Property>


<rdfs:Class rdf:ID="Person"/>


<rdfs:Datatype rdf:about="&xsd;integer"/>
```

注意：例 26 中没有使用<rdf:RDF>元素，因为这里的 RDF/XML 片断将被添加到了例 24 所示的车辆 schema 中。同理，这里可以使用相对 Urirefs（比如#MotorVehicle）来引用该 schema 中的类。

RDF Schema 提供了一种方法来像描述类的特化关系一样来描述特性之间的特化关系。这种两个特性之间的特化关系可以使用预定义的 rdfs:subPropertyOf 特性来描述。例如，如果 ex:primaryDriver 和 ex:driver 都是特性，则 example.org 可以通过下列 RDF 声明来描述“ex:primaryDriver 是 ex:driver 的特化”：

```
ex:driver          rdf:type             rdf:Property .

ex:primaryDriver   rdf:type             rdf:Property .

ex:primaryDriver   rdfs:subPropertyOf   ex:driver .
```

rdfs:subPropertyOf 关系的含义在于，如果一个实例 exstaff:fred 是实例 ex:companyVan 的一个 ex:primaryDriver，那么 RDF Schema 定义 exstaff:fred 也是 ex:companVan 的一个 ex:driver 。例 27 所示的 RDF/XML 描述了这些特性（同样，该 RDF/XML 片断将被添加到例 24 所示的车辆 schema 中）

例 27：车辆 schema 中其它的特性

```
<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>
```

一个特性可以是零个、一个或多个特性的子特性（subproperty）。当 RDF Schema 中的 rdfs:range 和 rdfs:domain 特性应用于某个 RDF 特性时，它们也会应用于该 RDF 特性的子特性。因此，在上面的例子中，RDF Schema 定义了 ex:primaryDriver 也具有 rdfs:domain 约束（值为 ex:MotorVehicle），因为 ex:primaryDriver 是 ex:driver 的子特性。

例 28 是车辆 schema 的完整 RDF/XML 代码，包括了到目前为止的所有描述。

例 28：完整的车辆 schema

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">

<rdfs:Class rdf:ID="MotorVehicle"/>

<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
```

```
</rdfs:Class>

<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Person"/>

<rdfs:Datatype rdf:about="&xsd;integer"/>

<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>

<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
```

```
</rdf:Property>

<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>

<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>

</rdf:RDF>
```

前面说明了如何使用 RDF Schema 来描述类和特性，现在可以讲述使用这些类和特性的实例了。例如，例 29 描述了一个 ex:passengerVehicle 类（见例 28）的实例，并为它的一些特性赋了值。

例 29：一个 ex:PassengerVehicle 类的实例

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:ex="http://example.org/schemas/vehicles#"
            xml:base="http://example.org/things">

  <ex:PassengerVehicle rdf:ID="johnSmithsCar">
       <ex:registeredTo rdf:resource="http://www.example.org/staffid/85740"/>
       <ex:rearSeatLegRoom
           rdf:datatype="&xsd;integer">127</ex:rearSeatLegRoom>
       <ex:primaryDriver rdf:resource="http://www.example.org/staffid/85740"/>
```

```
  </ex:PassengerVehicle>

</rdf:RDF>
```

本例假定实例与 schema 是分别在不同的文档中描述的。由于 schema 以 http://example.org/schemas/vehicles 作为其 XML base（即作为 xml:base 属性的值），因此在实例数据[译注//如 5.1 节所述，即描述属于某个类的个体的数据]中用命名空间声明 xmlns:ex="http://example.org/schemas/vehicles#" 来保证 QNames（如 ex:registeredTo）得以展开为正确的、标识 schema 中类和特性的 URIrefs。实例描述中也使用了一个 xml:base 声明，用于将 rdf:ID="johnSmithsCar"中的 johnSmithsCar 展开为正确的绝对 URIref，而绝对 URIrefs 的展开过程是根据 xml:base 进行的，与当前文档的位置无关。

注意，ex:registeredTo 特性可被用来描述 ex:PassengerVehicle 类的实例，因为 ex:PassengerVehicle 是 ex:MotorVehicle 的子类。也要注意到，在实例描述中，ex:rearSetLegRoom 特性的值是一个类型文字（typed literal），而不是平凡文字（plain literal）（它不是通过 <ex:rearSeatLegRoom>127</ex:rearSeatLegRoom>的方式来指定值）。因为 schema 中描述了该特性的值域（range）为一个 xsd:integer，因此该特性的值应该是一个数据类型为 xsd:interger 的类型文字以满足值域限制（即值域声明并不会自动地将平凡文字*指派*为某种数据类型，因此必须显式提供一个具有正确数据类型的类型文字）。正如 4.4 节中所讨论的那样，可以在 schema 或者附加的实例数据中提供一些额外信息来显式指定 ex:rearSetLegRoom 特性的*单位*（厘米）。

[编辑]

5.3 解释 RDF Schema 声明

正如前所述，RDF Schema 的类型系统与面向对象编程语言（如 Java）的类型系统很相似，然而，RDF 与大部分程序设计语言的类型系统在几个重要方面是有区别：

一个重要的区别是：RDF Schema 描述属性可以作用于哪些特定的类（通过 *domain* 和 *range* 属性），而不是（像面向对象类型系统）描述类具有哪些特定属性的集合。例如：在一个典型的面向对象编程语言中，类 Book 会定义为有一个属性 author,且 author 的类型是 Person；而在 RDF Schema 中，类 ex:Book 和属性 ex:author 都会被分开定义，另外，属性 ex:author 的定义域是 ex:Book,值域是 ex:Person。

这两种定义方法的区别表面上看起来只是语法层次上的，其实，他们有着重大的区别。在程序设计语言中，属性 author 是关于类 Book 的描述的一部分，而且只能应用于类 Book 的实例。如果另外一个类，例如 SoftwareModule 也有个属性 author,这两个 author 属性被看成是不同的属性。因为在大多数编程语言中，属性的作用域局部与它被定义的类或类型的。而在 RDF 中，缺省地，属性的描述是独立于类描述的，并且属性的作用域是*全局的*（尽管可以它们可被声明为只能应用于某些类）。

一个结果是：RDF Schema 可以描述一个不指定定义域的属性，如 exterms:weight 这个属性可以用来描述任何具有重量（weight）属性的类的实例。RDF 的以属性为中心的方法的一个好处是：扩展属性的用途使其应用于最初定义时未预知的情形更为容易。同时，这个“好处”也要谨慎使用，以免属性错误应用于不恰当的情形。

RDF 属性的全局作用域的另外一个结果是：无法在 RDF Schema 中定义一个根据所描述类（即定义域）的不同而具有不同值域的属性。例如，在定义属性 ex:hasParent 时，如果这个属性用于描述类 ex:Human，则这个属性的值域是 ex:Human；如果这个属性用于描述类 ex:Tiger，则这个属性的值域是 ex:Tiger，这样定义属性显然更为合适。但这种定义方法无法在 RDF Schema 中实现。对一个 RDF 属性的值域的任何定义将应用于用到该属性的*所有*地方。因此，定义值域时应谨慎。然而，虽然不能在 RDF Schema 中定义这种局部相异的属性值域，但在 5.5 节中讨论的表达能力更强的模式语言中可以定义。

另外一个重要的区别是：RDF Schema 的描述不一定像程序设计语言的类型声明那样是*规约性的（prescriptive）*。例如，如果在某个程序设计语言中，类 Book 被定义为有一个属性 author，且 author 的类型是 Person，这通常被解释为一组*约束*。该语言不会允许 Book 的一个实例没有 author 属性，也不会允许 Book 的一个实例的 book 属性值的类型不是 Person。更为重要的是，如果 author 的定义为类 Book 的唯一属性，该语言不会允许 Book 的实例具有其他的属性。

相反，RDF Schem 提供的模式信息是作为对资源的额外*描述（descriptions）*，并不会限制这些描述怎样被用于一个应用。例如，假设一个 RDF Schema 中定义了属性 ex:author 的值域（rdfs:range）是 ex:Person,这仅仅是一个简单的 RDF 陈述：谓词是 ex:author 的陈述的客体是类 ex:Person 的实例。

这个 Schema 提供的信息可能以多种方式得到利用。一个应用可能把这个陈述解释为创建 RDF 数据的模板的一部分，且用它来限制任意 ex:author 的值都是 ex:Person 的实例。也就是说，这个应用和程序设计语言一样，把模式信息解释为约束（constraint）。然而，另外一个应用可能把这个陈述解释为获取的数据的额外信息，这个信息并没有在原数据中显式表示。例如，第二个应用可能收到一些 RDF 数据，这些数据包含了一个值为未知类型的 ex:author 属性，则可以利用 RDF Schema 中的那个陈述推导出这个 ex:author 属性的值是类 ex:Person 的实例。第三个应用可能收到一些 RDF 数据，这些数据包含了一个值为 ex:Corporation 的实例的 ex:author 属性，利用 RDF Schema 的信息，可以发出警告：“数据可能不一致，也可能不是”。在其他地方，可能存在一个声明消除了这个数据不一致性（例如，声明一个“Corporation”也是一个“Person”）。

此外，根据应用解释属性描述的方式，一个实例仍然可以被认为是合法的，即使没有 RDF Schema 指定的属性（如可能有一些 ex:Book 的实例没有 ex:author 属性，即使 ex:author 的定义域为 ex:Book），或者具有其他的属性（如可能有一些 ex:Book 的实例具有 ex:technicalEditor 属性，即使描述 ex：Book 的 Schema 没有对这个属性的描述）。

换句话说，RDF Schema 中的陈述通常是*描述（descriptions）*。他们也可以是*规约性的（prescriptive）*，如果应用要这样解释的话。RDF Schema 所做的是提供额外的信息。这些信息是否和显式指定的实例数据有冲突，取决于具体的应用。

[编辑]

5.4 其他 Schema 信息

RDF Schema 还提供了一些其他的内嵌属性，这些属性可以用于为 RDF Schema 和实例提供资料记录和其他信息。例如，rdfs:comment 属性可用于提供关于资源的易读的描述，rdfs:label 属性可用于提供关于资源的更易读的名字，rdfs:seeAlso 属性可用于提示其他地方可能有关于此资源的更多的描述，rdfs:isDefinedBy 是 rdfs:seeAlso 的子属性，可用于提示其他地方有关于此资源的定义。对这些属性更多的讨论，请参考 RDF 词汇描述语言 1.0: RDF Schema[RDF-VOCABULARY]。

对于其他的内嵌 RDF 属性（如 rdf:value）它们的用途就是规范中的*设计用途（intended use）*，文档[RDF-SEMANTICS]对这些属性没有定义特定的语义，RDF Schmea 也没有基于他们的设计用途定义任何的约束。例如，没有约束限制属性 rdfs:seeAlso 的客体必须为它所在的陈述的主体提供额外的信息。

[编辑]

5.5 表达更丰富的 Schema 语言

RDF Schema 提供了描述 RDF 词汇集的基本能力，更强的表达能力是可能的，也是有用的。这些能力可以通过进一步发展 RDF Schema 来提供，也可以通过其他基于 RDF 的语言来提供。其他已被认为有用但 RDF Schema 没有提供的、更丰富的表达能力包括：

对属性的*基数限制*。例如，一个人*有且只有一个*生物学意义上的父亲。

指定一个属性（如 ex:hasAncestor）是*传递的*，即：如果 A ex:hasAncestor B，且 B ex:hasAncestor C，则 A ex:hasAncestor C。

指定一个属性是一个类的实例的唯一表示符（或说，*主键*）。

指定两个不同的类（具有不同的 URIrefs）实际代表同一个的类。

指定两个不同的实例（具有不同的 URIrefs）实际代表同一个实例。

指定属性的值域或基数限制取决于属性应用到的类，例如，说一个足球队的属性 ex:hasPlayers 值的个数为 11，同时，这个属性应用于篮球队时，值的个数应该是 5。

能够通过对类的组合（如，并，交）得到新的类，或者说两个类是相离的，即两个类没有共同的实例。

上面提到这些丰富的表达能力，以及其他一些没提到的，正是*本体*语言（如 DAML+OIL[DAML+OIL]和 OWL[OWL]）的目标。这些本体语言都是基于 RDF 和 RDF Schema 的。这些语言的目的是为资源提供更多的机器可处理的语义信息，也就是说，使资源的机器表示能够更紧密模拟真实世界对应的部分。虽然这些语言对构造基于 RDF 的有用应用（第 6 节有一些已有的 RDF 应用的描述）来说不是必须的，但这些语言的研究是语义 Web 研究中的一个非常活跃的主题。

[编辑]

6. 一些 RDF 应用：具体领域中的 RDF

在前面的章节里，已经描述了 RDF 和 RDF Schema 的大体的表达能力，并且举了一些例子来演示这些能力，其中有些例子还可能暗示了一些潜在的 RDF 应用，但是并没有讨论任何真正的应用。在这节中，将会描述一些真正已经发布的 RDF 应用，说明了 RDF 是怎样支持各种不同的现实世界的关于表示和处理关于各种事物的信息的需求。

[编辑]

6.1 都柏林核心元数据倡议

元数据是关于数据的数据。这个术语指的是用来识别、描述、查找信息资源的数据，而不管这些资源是物理存在的或电子化的。虽然用计算机处理的结构化元数据相对较新，但在帮助管理和使用海量信息时的元数据的基本概念已经用了多年。图书目录卡就是一个大家熟悉的这种元数据例子。

都柏林核心是描述文件(因此，也是为了记录元数据)的一个“元素”集(属性)。这个元素集是由元数据工作组在 1995 年 3 月在俄亥俄州的都柏林(Dublin, Ohio)首先开发出来的。稍后，在都柏林核心元数据工作组基础上，都柏林核心进行了一系列的修改，这就是我们现在看到的都柏林核心元数据倡议。都柏林核心的目的是提供一个描述性的元素的最小集，以便对类似文档的网络对象进行描述和自动索引，就像一张图书目录卡一样。都柏林核心元数据集的设计目标是适合让因特网上资源发现工具使用，如通行的万维网搜索引擎使用的网络爬虫（“Webcrawlers”）。此外，都柏林核心有意为足够简单，使得为因特网提供信息的各种各样的作者和出版商都能够理解和使用。都柏林核心元素已广泛地应用于记录因特网资源的有关信息(都柏林核心元素 dc:create 已经在以前的例子中用到)。 都柏林核心元素的当前版本定义在柏林核心元数据元素集，版本 1.1：参考描述 [DC]，它包含了下列属性的定义：

Title: 资源的名字。

Creator: 一个主要负责创建资源内容的实体。

Subject: 资源内容的主题.

Description: 资源内容的描述.

Publisher: 一个负责使得资源内容可用的实体

Contributor: 一个负责为资源内容作出贡献的实体(如作者)。

Date: 在资源生命周期中某时间关联的日期。

Type: 资源内容的类型。

Format: 资源的物理形式或数据形式.

Identifier: 一个在给定上下文中明确标识资源的标识符。

Source: 一个对作为目前资源的来源的资源引用。

Language: 资源内容采用的语言

Relation: 一个对相关资源的引用

Coverage: 资源内容所在的范围或区域

Rights: 关于资源的权限信息.

使用都柏林核心元素的信息可以用任意适合的语言(例如，HTML 的 meta 元素)表示。 然而，RDF 是都柏林核心信息的一种理想的表示。 下面的例子表示了在 RDF 中，用都柏林核心词汇表示的关于一个资源集合的简单描述。注意，这里显示的都柏林核心 RDF 词汇不是权威的描述，都柏林核心参考描述[DC]才是最权威的参考。

第一个例子(例 30)，使用都柏林核心属性描述一个网站主页：

例 30：一个用都柏林核心属性描述的网页

```
<rdf:RDF

    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    xmlns:dc="http://purl.org/dc/elements/1.1/">

    <rdf:Description rdf:about="http://www.dlib.org">

      <dc:title>D-Lib Program - Research in Digital Libraries</dc:title>

      <dc:description>The D-Lib program supports the community of people

       with research interests in digital libraries and electronic

       publishing.</dc:description>

      <dc:publisher>Corporation For National Research Initiatives</dc:publisher>

      <dc:date>1995-01-07</dc:date>

      <dc:subject>

        <rdf:Bag>

          <rdf:li>Research; statistical methods</rdf:li>

          <rdf:li>Education, research, related topics</rdf:li>

          <rdf:li>Library use Studies</rdf:li>

        </rdf:Bag>

      </dc:subject>

      <dc:type>World Wide Web Home Page</dc:type>

      <dc:format>text/html</dc:format>

      <dc:language>en</dc:language>

    </rdf:Description>

</rdf:RDF>
```

我们注意到，RDF 和都柏林核心都定义了一个(XML)元素称为为“Description”，(虽然都柏林核心元素名称用了小写字母)。 即使开头的字母都是大写字母，还是可以用 XML 的名字空间机制将这两个元素区别开来(一是 rdf：Description，另外一个是 dc：description)。 同时，有趣的是，在 Web 浏览器中访问 http://purl.org/dc/elements/1.1/ 时(名字空间 URI 在这里用于标识都柏林核心词汇)，可以得到一个[DC]的 RDF Schema 声明。

第二个例子，例 31 描述了一个出版了的杂志

```
<rdf:RDF

    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

    xmlns:dc="http://purl.org/dc/elements/1.1/"

    xmlns:dcterms="http://purl.org/dc/terms/">

    <rdf:Description rdf:about="http://www.dlib.org/dlib/may98/05contents.html">

      <dc:title>DLIB Magazine - The Magazine for Digital Library Research

        - May 1998</dc:title>

      <dc:description>D-LIB magazine is a monthly compilation of

       contributed stories, commentary, and briefings.</dc:description>

      <dc:contributor>Amy Friedlander</dc:contributor>

      <dc:publisher>Corporation for National Research Initiatives</dc:publisher>

      <dc:date>1998-01-05</dc:date>

      <dc:type>electronic journal</dc:type>

      <dc:subject>

        <rdf:Bag>

          <rdf:li>library use studies</rdf:li>

          <rdf:li>magazines and newspapers</rdf:li>

        </rdf:Bag>

      </dc:subject>

      <dc:format>text/html</dc:format>

      <dc:identifier rdf:resource="urn:issn:1082-9873"/>
```

```
      <dcterms:isPartOf rdf:resource="http://www.dlib.org"/>
    </rdf:Description>
  </rdf:RDF>
```

例 31 使用了都柏林的限定词“isPartOf”（来自一个单独的词汇表）来说明这份杂志是以前描述过的 Web 网站的一部分。

第三个例子,例 32, 描述了例 31 中描述过的杂志中的一篇特定的文章

例 32：描述一篇杂志文章

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:dcterms="http://purl.org/dc/terms/">
    <rdf:Description rdf:about="http://www.dlib.org/dlib/may98/miller/05miller.html">
      <dc:title>An Introduction to the Resource Description Framework</dc:title>
      <dc:creator>Eric J. Miller</dc:creator>
      <dc:description>The Resource Description Framework (RDF) is an
       infrastructure that enables the encoding, exchange and reuse of
       structured metadata. rdf is an application of xml that imposes needed
       structural constraints to provide unambiguous methods of expressing
       semantics. rdf additionally provides a means for publishing both
       human-readable and machine-processable vocabularies designed to
       encourage the reuse and extension of metadata semantics among
       disparate information communities. the structural constraints rdf
       imposes to support the consistent encoding and exchange of
       standardized metadata provides for the interchangeability of separate
```

```
        packages of metadata defined by different resource description

        communities. </dc:description>

       <dc:publisher>Corporation for National Research Initiatives</dc:publisher>

       <dc:subject>

         <rdf:Bag>

           <rdf:li>machine-readable catalog record formats</rdf:li>

           <rdf:li>applications of computer file organization and

            access methods</rdf:li>

         </rdf:Bag>

       </dc:subject>

       <dc:rights>Copyright ? 1998 Eric Miller</dc:rights>

       <dc:type>Electronic Document</dc:type>

       <dc:format>text/html</dc:format>

       <dc:language>en</dc:language>

       <dcterms:isPartOf rdf:resource="http://www.dlib.org/dlib/may98/05contents.html"/>

     </rdf:Description>

</rdf:RDF>
```

例 32 中也使用了限定词“isPartOf”,这次是指这篇文章是先前描述过的杂志中的一部分。

计算机语言和文件格式并非总能为他们描述的数据嵌入元数据提供明确的方式。在许多情况下，元数据不得不被指定为一种单独的资源，并被明确地链接到数据(如，描述本入门文档的 RDF 元数据就是这样的；在本文档的的最后一部分有一个对元数据的显式链接)。 然而，应用程序和语言日益为元数据直接嵌入数据提供明确的方式。例如， W3C 的可伸缩矢量图形语言[SVG] (另一个以 XML 为基础的语言)提供了一个明确的元数据元素用以记录关于其他 SVG 数据的元数据。 任何以 XML 为基础的元数据语言都能包含在这个元素里。[SVG] 包含了例 33 所示的例子，它描述了一个 SVG 文件的元数据是如何嵌入在 SVG 文件本身的。这个例子使用了都柏林核心词汇和 RDF/XML 来记录元数据。

例 33：在 SVG 文件中嵌入元数据

```
<?xml version="1.0"?>
```

```
<svg width="4in" height="3in" version="1.1"
    xmlns = 'http://www.w3.org/2000/svg'>
    <desc xmlns:myfoo="http://example.org/myfoo">
      <myfoo:title>This is a financial report</myfoo:title>
      <myfoo:descr>The global description uses markup from the
        <myfoo:emph>myfoo</myfoo:emph> namespace.</myfoo:descr>
      <myfoo:scene><myfoo:what>widget $growth</myfoo:what>
      <myfoo:contains>$three $graph-bar</myfoo:contains>
        <myfoo:when>1998 $through 2000</myfoo:when> </myfoo:scene>
   </desc>
    <metadata>
      <rdf:RDF
          xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
          xmlns:dc = "http://purl.org/dc/elements/1.1/" >
        <rdf:Description rdf:about="http://example.org/myfoo"
            dc:title="MyFoo Financial Report"
            dc:description="$three $bar $thousands $dollars $from 1998 $through 2000"
            dc:publisher="Example Organization"
            dc:date="2000-04-11"
            dc:format="image/svg+xml"
            dc:language="en" >
          <dc:creator>
            <rdf:Bag>
```

```
            <rdf:li>Irving Bird</rdf:li>

            <rdf:li>Mary Lambert</rdf:li>

          </rdf:Bag>

        </dc:creator>

      </rdf:Description>

    </rdf:RDF>

  </metadata>

</svg>
```

Adobe 公司的可扩展元数据平台(XMP)是另一个允许关于一个文件的元数据被嵌入到文件本身中的技术的例子。 XMP 使用 RDF/XML 作为其元数据表示的基础。 一些 Adobe 产品已经支持 XMP。

[编辑]

6.2 PRISM

PRISM:PRISM: 出版业标准元数据需求[PRISM]是在出版业中发展完善的一个元数据规范。PRISM 工作组是由杂志出版商和杂志的销售商构成的，他们确定行业所需的元数据（metadata）并为之指定一些列相关的规范。出版商想以多种方式利用现有内容（资料），目的是获取更多的对创建资源时的投资回报。一个例子是为了发布在 Web 上而把杂志文章转换成 HTML；而把它许可给一个叫 LexisNexis 的聚合器则是另外一个例子。所有的这些都是内容的“首次使用”；当杂志发行时，其内容也跟着公诸于世了，出版商也想让他们的内容变成“常青树”。它有可能用于其他的期号，如在一个回顾文章中。它也可能用于公司的其他部门，如把杂志中的图片等编辑成书。另一个用途则是许可给外人使用，如关于产品评论的再版，或其他出版商的精选辑中。全部达到这些目的就需要一种元数据方式，它着重于"发现"，"权限追踪"和"端到端的元数据"。

发现（Discovery）：发现是关于寻找资料的一个笼统的术语，包含了搜索，浏览，内容的传输，以及其他技术。关于发现的讨论经常以客户在公众网站的搜索为中心。不过，内容发现的范畴要比这宽广得多。（发现的）主体可能由客户组成，或可能由内部用户（如研究人员，设计者，图片编辑，许可证代理人，等等）组成。为了辅助法内容发现，PRISM 提供了一系列属性来描述资源的主题，格式，类型，来源和上下文。它同时也提供了一种用多主题描述分类系统对资源分类的方法。

权限追踪（Rights Tracking）:杂志经常包含了经其它人许可使用的材料。除了文章，工具条和其他所有内容可能已被许可的类型之外，特许资料的最普遍的类型就是从一个图片代理处获得的图片。一份资料是否有一次性使用权（当然，这也是要交特许使用金的），还是被出版商完全占有；仅仅想知道这些就需要一番努力了。PRISM 为简单的权限跟踪提供了一些元素。PRISM 规范定义了一个单独的词汇集用来支持一些描述（description），这些描述记录了资料可以或不可以被使用的地方，次数，行业等相关内容。

端到端的元数据（End-to-end metadata）：大多数出版的内容都已经有了为它们创建的元数据。遗憾的是，当在系统之间传输内容时，元数据经常会丢失，那么只有在以后的生产过程中以相当的开销来重建它们。为了减少这种问题的发生，PRISM 提供了一种能在内容生产线的多个阶段中使用的规范。PRISM 规范的一大特色是对其他现有的规范的使用。工作组坚决尽可能多的使用现有的规范，而不是创建一个全新的事物，并且只是在必需的地方定义新事物。因此，RPISM 规范使用 XML，RDF，Dublin Core，和各种 ISO 格式和词汇表。

一个 PRISM 描述可能和一些值为频繁文字的 Dublin Core 属性那么简单。例 34 描述了一个照片，并给出了它的名称，摄影师，格式等一些基本的信息：

例 34：一个照片的 PRISM 描述

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:dc="http://purl.org/dc/elements/1.1/"

         xml:lang="en-US">



 <rdf:Description rdf:about="http://travel.example.com/2000/08/Corfu.jpg">

  <dc:title>Walking on the Beach in Corfu</dc:title>

  <dc:description>Photograph taken at 6:00 am on Corfu with two models

  </dc:description>

  <dc:creator>John Peterson</dc:creator>

  <dc:contributor>Sally Smith, lighting</dc:contributor>

  <dc:format>image/jpeg</dc:format>

 </rdf:Description>

</rdf:RDF>
```

PRISM 也对 Dublin Core 的作了补充用以允许更详细的描述。补充的内容通过三个新的词汇集来说明，通常通过前缀 prism:，pcv:和 prl 来引用：

prism: 这个前缀指主要的 RPISM 词汇集，它的术语使用了 URI 前缀“http://prismstandard.org/namespaces/basic/1.0/”。这个词汇集中的大部分属性是来自 Dublin Core 属性的更细化版本。例如，dc:date 的细化版本是由像 prism:publicationTime，prism:releaseTime，prism:expirationTime 等等这样的属性提供的。

pcv:这个前缀指 PRISM 受控词汇表（PRISM Controlled Vocabulary(pcv)），它的术语使用了 URI 前缀“http://prismstandard.org/namespaces/pcv/1.0/”。通常，描述文章的主题（可能不止一个）的常见做法是提供描述性的关键词。遗憾的是，由于不同的人会使用不同的关键词[BATES96]，简单的关键词对提高检索性能上并没有太大作用。最好的办法是用取自某“受控词汇表”的主题术语（subject terms）为文章编码，这个词汇集应当为词汇集中的术语提供尽可能多的同义词。这样，受控的术语就会为搜索者和编索引者提供共用的关键词。pcv 词汇集提供了属性，用以说明词汇集中的术语，术语之间的关系，和术语的别称。

prl:这个前缀指 PRISM 权限语言词汇集(PRISM Rights Language vocabulary)，它的术语使用 URI 前缀“http://prismstandard.org/namespaces/prl/1.0/”。数字权限管理（Digital Rights Management）是个承受着剧变的领域。有很多关于权限管理语言的提议，但没有一个能在整个行业中明显受欢迎。因为没有明确的可推荐选择，于是 PRISM Rights Language (PRL)就被定义为一个过渡性的标准。它提供了依赖于时间，地理，行业等条件的属性，这些属性可以让人们表示一个术语能或不能被使用。人们相信这是一个将会帮助出版商在跟踪权限时省钱的“80/20 平衡”【译者注：意思是说 80%的人经

常使用占术语总数 20%的那部分术语，所以，确定一个标准（即限定这 20%的术语）可以使出版商在跟踪权限时省时省力，也就省了钱。】。它不是作为一种通用的权限语言的，也不能让出版商对客户的内容使用作出限制。

PRISM 采用 RDF 是因为它有处理复杂程度不同的描述的能力。目前，很多元数据使用普通的字符串（平凡文字）值，例如：

```
<dc:coverage>Greece</dc:coverage>
```

长时间以来 PRISM 的开发者期待着 PRISM 规范的使用变得更完善,由简单文字值向更具构造性的值转化。事实上，那个范围的值正是现在有待解决的。一些出版商已经在使用完善的受控词汇集了，而其他的才刚刚使用手工提供（manually-supplied）的关键词。为了说明这些，能赋值给属性 dc:coverage 的不同类型的值的例子是：

```
<dc:coverage>Greece</dc:coverage>

<dc:coverage rdf:resource="http://prismstandard.org/vocabs/ISO-3166/GR"/>
```

(也就是说，用平凡文字或 URIref 标识这个国家)和

```
<dc:coverage>
  <pcv:Descriptor rdf:about="http://prismstandard.org/vocabs/ISO-3166/GR">
    <pcv:label xml:lang="en">Greece</pcv:label>
    <pcv:label xml:lang="fr">Grèce</pcv:label>
  </pcv:Descriptor>
</dc:coverage>
```

（使用一个构造值来提供一个 URIref 和在多种语言中的名称）

同时注意有些含义近似，或可说是其他属性的子集的属性。例如：一个资源的地理主题可以像这样给出：

```
<prism:subject>Greece</prism:subject>
<dc:coverage>Greece</dc:coverage>
```

或

```
<prism:location>Greece</prism:location>
```

那些属性当中的任何一个都可能使用简单文字值或是一个更复杂的结构值。DTD，甚至是更新的 XML Schemas，都不能充分的描述出这种多变的可能性。虽然要处理这种范围广泛的句法变化，RDF 的模型图却有一种简单的结构——三元组集。在三元组领域处理元数据使得旧软件适应有新扩展的内容轻松多了。

这一节由最后举两个例子结束。例 35 说明了：图像（.../Corfu.jpg）不能够被用于（#none）烟草工业（在 SIC（Standard Industrial Classifications, 标准工业分类）中的代码为 21 ）。

例 35：一个图片的 PRISM 描述：

```
<rdf:RDF xmlns:prism="http://prismstandard.org/namespaces/basic/1.0/"
         xmlns:prl="http://prismstandard.org/namespaces/prl/1.0/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
 <rdf:Description rdf:about="http://travel.example.com/2000/08/Corfu.jpg">
  <dc:rights rdf:parseType="Resource"
         xml:base="http://prismstandard.org/vocabularies/1.0/usage.xml">
    <prl:usage rdf:resource="#none"/>
    <prl:industry rdf:resource="http://prismstandard.org/vocabs/SIC/21"/>
  </dc:rights>
 </rdf:Description>
</rdf:RDF>
```

例 36 说明了图片“Corfu”的摄影师员工编号是“3845”（employee 3845），就是人们熟知的 John Peterson（better known as John Peterson）。它还说明了图片所显示的地理位置为希腊。例 36 之所以能够做到这一点，在于它不仅仅提供了来自受控词汇表的术语代码（code），而且它还可以提供该词汇表中术语代码所隐含的信息。

例 36：关于例 35 中图片的附加说明：

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

         xmlns:pcv="http://prismstandard.org/namespaces/pcv/1.0/"

         xmlns:dc="http://purl.org/dc/elements/1.1/"

         xml:base="http://travel.example.com/">



  <rdf:Description rdf:about="/2000/08/Corfu.jpg">

    <dc:identifier rdf:resource="/content/2357845" />

    <dc:creator>

      <pcv:Descriptor rdf:about="/emp3845">

        <pcv:label>John Peterson</pcv:label>

      </pcv:Descriptor>

    </dc:creator>

    <dc:coverage>

      <pcv:Descriptor

          rdf:about="http://prismstandard.org/vocabs/ISO-3166/GR">

        <pcv:label xml:lang="en">Greece</pcv:label>

        <pcv:label xml:lang="fr">Grece</pcv:label>

      </pcv:Descriptor>

    </dc:coverage>

  </rdf:Description>

</rdf:RDF>
```

6.3 XPackage

在很多情况下都需要维持 关于资源的结构化分组和他们用作单位的联系的信息.The XML 包 (XPackage) 规格说明书 [XPACKAGE 提供了一个定义这样的分组的框架,叫做"包". XPackage 详细说明了一个框架来描述包含在包里面的资源,这些资源的特性,包含的方法,还有他们之间的相互关系. XPackage 应用包括指定一个文档用的样式表,声明多个文档共用的图片, 指明一个文档的作者和其他元数据, 描述 XML 资源是如何使用名字空间的, 还提供了一个清单来把资源和一个档案文件捆在一起.

XPackage 框架是基于 XML, RDF 和 XML 链接语言 [XLINK], 还提供了多个 RDF 词汇表: 一个是用在一般的包装描述, 还有几个词汇表是用于对包装处理器提供一些有用的信息.

对 XPackage 的一个应用是描述 XHTML 文档和他们支持的资源.当一个站点要获取一个 XHTML 文档的时候,或许还会牵涉到其他的资源,像样式表和图像文件也需要被获取.然而,如果不处理这个文档,这些资源的特性就不是很明显了.关于这个文档的其他信息,例如作者的名字,如果不处理这个文档,可能也是不可取的. XPackage 允许这些叙述性情报(如作者等信息)标准地储存在一个包含 RDF 的包描述文档.例 37 (为了简明, 删除了名字空间声明):描述一个 XHTML 文档的包描述文档的外部元素:

例 37: 一个 XPackage 包描述文档的外部元素

```
<?xml version="1.0"?>

<xpackage:description>

  <rdf:RDF>



    (description of individual resources go here)



  </rdf:RDF>

</xpackage:description>
```

在包描述文档里面,资源(如 XHTML 文档, 样式表, 和图片) 被标准的 RDF/XML 语法描述. 每一个资源描述元素都包含在不同的词汇表里面的 RDF 属性(XPackage 使用术语"ontology"来表示 RDF 里面的"词汇表"). 除了那个主要的包词汇表, XPackage 本身自带了几个补充的词汇表,包括:

```
*一个描述文件(包含属性如 file:size)的词汇表(使用前缀 file:)

*一个提供 MIME 信息(包含属性如 mime:contentType)的词汇表(使用前缀 mime:)

*一个提供字符使用信息(包含属性如 unicode:sript)的词汇表(使用前缀 unicode:)

*一个描述基于 XML 的资源(包含属性如 x:namespace 和 x:style)的词汇表(使用前缀 x:)
```

在例 38, 使用一个标准的属于 XPackage MINE 词汇表的 XPackage 属性(mime:contentType),定义了文件的 MIME 内容类型 ("application/xhtml+xml"). 另一个属性,使用了一个 Dublin Core 词汇表里面的属性来描述文档的作者 (在这个例子里是"Garret Wilson").这个属性是在 XPackage 外部定义的,所有就使用了 dc:creator 这个属性.

例 38: 对一个 XHTML 文档的描述

```
<?xml version="1.0"?>

<xpackage:description

            xmlns:xpackage="http://xpackage.org/namespaces/2003/xpackage#"

            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"

            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

            xmlns:dc="http://purl.org/dc/elements/1.1/"

            xmlns:mime="http://xpackage.org/namespaces/2003/mime#"

            xmlns:x="http://xpackage.org/namespaces/2003/xml#"

            xmlns:xlink="http://www.w3.org/1999/xlink">

    <rdf:RDF>



    <!--doc.html-->

    <rdf:Description rdf:about="urn:example:xhtmldocument-doc">

      <rdfs:comment>The XHTML document.</rdfs:comment>

      <xpackage:location xlink:href="doc.html"/>

      <mime:contentType>application/xhtml+xml</mime:contentType>

      <x:namespace rdf:resource="http://www.w3.org/1999/xhtml"/>

      <x:style rdf:resource="urn:example:xhtmldocument-stylesheet"/>

      <dc:creator>Garret Wilson</dc:creator>

      <xpackage:manifest rdf:parseType="Collection">
```

```
        <rdf:Description rdf:about="urn:example:xhtmldocument-stylesheet"/>
        <rdf:Description rdf:about="urn:example:xhtmldocument-image"/>
      </xpackage:manifest>
    </rdf:Description>

    </rdf:RDF>
</xpackage:description>
```

属性 xpackage:manifest 指明,在处理的时候,样式表和图像资源都需要用到; 那些资源在包描述文档里面被分别的描述. 在描述样式表的例子例 39 里面, 使用在 general XPackage 词汇表里面的 xpackage:location 属性列出了它在包里面的路径("stylesheet.css"), 也用 XPackage MIME 词汇表的 mime:contentType 属性来说明了这是一个 CSS 样式表 ("text/css").

例 39: 一个样式表的资源描述

```
<?xml version="1.0"?>
<xpackage:description
            xmlns:xpackage="http://xpackage.org/namespaces/2003/xpackage#"
            xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:mime="http://xpackage.org/namespaces/2003/mime#"
            xmlns:x="http://xpackage.org/namespaces/2003/xml#"
            xmlns:xlink="http://www.w3.org/1999/xlink">
    <rdf:RDF>

    <!--stylesheet.css-->
    <rdf:Description rdf:about="urn:example:xhtmldocument-css">
```

```
        <rdfs:comment>The document style sheet.</rdfs:comment>

        <xpackage:location xlink:href="stylesheet.css"/>

        <mime:contentType>text/css</mime:contentType>

      </rdf:Description>



      </rdf:RDF>

  </xpackage:description>
```

这个例子的完整版本可以在[XPACKAGE]上看到.

[编辑]

6.4 RSS 1.0: RDF 站点汇总

人们常常需要每天访问网上各种各样的信息，例如计划安排、任务表、新闻标题、搜索结果，最新消息等。随着万维网上信息资源和多样性的增加，管理这些信息并将之集成为一个整体的难度越来越大。RSS1.0（http://purl.org/rss/1.0）(“RDF Site Sumary”)是一个 RDF 词汇表，它提供一种轻量级但功能强大的方式来描述这些实时的、大规模分布和可重用的信息。RSS1.0 同时可能也是万维网上最为广泛的 RDF 应用。

举例来说，W3C 主页（http://www.w3.org）是 W3C 组织与保持公众联系的主要方式，服务于传播有关该组织的研究信息。图 19 是某一天的 W3C 主页，(http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#figure19).中间一列是新闻条目，经常发生改变。为了能够及时传播这些信息，W3C 小组实现了一个 RDF Site Summary(RSS 1.0 (http://purl.org/rss/1.0/))新闻种子，这样中间列的内容就可以被其它人按需获取或者重用。新闻综合站点可以将（种子站点的）新闻标题融合进自己当天新闻综述之中。其它的站点也可以将新闻标题作为一种链接显示给它们的访问者，并且，越来越多的个人会用自己的桌面应用来订阅种子（站点的内容）。这些桌面的 RSS 阅读器允许它们的用户不用浏览器去访问每一个站点，就能追踪上百个站点。

图 19：W3C 主页

万维网上大量的站点都提供 RSS1.0 种子。例 40 (http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#example40)是某一天 W3C 种子的实例。(http://www.w3.org/2000/08/w3c-synd/home.rss)

例 40: W3C RSS 1.0 种子示例

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF xmlns="http://purl.org/rss/1.0/"

    xmlns:dc="http://purl.org/dc/elements/1.1/"

    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <channel rdf:about="http://www.w3.org/2000/08/w3c-synd/home.rss">

    <title>The World Wide Web Consortium</title>

    <description>Leading the Web to its Full Potential...</description>

    <link>http://www.w3.org/</link&gt;

    <dc:date>2002-10-28T08:07:21Z</dc:date>

    <items>

        <rdf:Seq>

            <rdf:li rdf:resource="http://www.w3.org/News/2002#item164"/>

            <rdf:li rdf:resource="http://www.w3.org/News/2002#item168"/>

            <rdf:li rdf:resource="http://www.w3.org/News/2002#item167"/>

        </rdf:Seq>

    </items>

  </channel>

  <item rdf:about="http://www.w3.org/News/2002#item164">

    <title>User Agent Accessibility Guidelines Become a W3C

       Proposed Recommendation</title>

    <description>17 October 2002: W3C is pleased to announce the

       advancement of User Agent Accessibility Guidelines 1.0 to
```

```
      Proposed Recommendation. Comments are welcome through 14 November.
      Written for developers of user agents, the guidelines lower
      barriers to Web accessibility for people with disabilities
      (visual, hearing, physical, cognitive, and neurological).
      The companion Techniques Working Draft is updated. Read about
      the Web Accessibility Initiative. (News archive)</description>
    <link>http://www.w3.org/News/2002#item164</link&gt;
    <dc:date>2002-10-17</dc:date>
  </item>
  <item rdf:about="http://www.w3.org/News/2002#item168">
    <title>Working Draft of Authoring Challenges for Device
      Independence Published</title>
    <description>25 October 2002: The Device Independence
      Working Group has released the first public Working Draft of
      Authoring Challenges for Device Independence. The draft describes
      the considerations that Web authors face in supporting access to
      their sites from a variety of different devices. It is written
      for authors, language developers, device experts and developers
      of Web applications and authoring systems. Read about the Device
      Independence Activity (News archive)</description>
    <link>http://www.w3.org/News/2002#item168</link&gt;
    <dc:date>2002-10-25</dc:date>
  </item>
  <item rdf:about="http://www.w3.org/News/2002#item167">
```

```
    <title>CSS3 Last Call Working Drafts Published</title>

    <description>24 October 2002: The CSS Working Group has

       released two Last Call Working Drafts and welcomes comments

       on them through 27 November. CSS3 module: text is a set of

       text formatting properties and addresses international contexts.

       CSS3 module: Ruby is properties for ruby, a short run of text

       alongside base text typically used in East Asia. CSS3 module:

       The box model for the layout of textual documents in visual

       media is also updated. Cascading Style Sheets (CSS) is a

       language used to render structured documents like HTML and

       XML on screen, on paper, and in speech. Visit the CSS home

       page. (News archive)</description>

    <link>http://www.w3.org/News/2002#item167</link&gt;

    <dc:date>2002-10-24</dc:date>

  </item>

</rdf:RDF>
```

正如例 40 所示 (http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#example40)，种子的格式经过专门设计，这样内容就可以非常容易地打包进不同的部分。新闻站点，网站日志，体育比分（sports scores）、股票报价 一类的应用都是 RSS1.O 的典型用例。

RSS 种子可以被任何能够“讲”HTTP 的应用所请求。最近，RSS1.0 应用分成了以下三种不同的大类:

在线聚合器——一些站点例如：Meerkat (http://www.oreillynet.com/meerkat/index.php?&c=4743&t=ALL) 和 NewsIsFree (http://www.newsisfree.com/sources/info/906/),如图 20 所示 (http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#figure20)。这些站点从上千个信息源那里收集种子信息，从中分离出<item>所包含的内容，然后将这些条目添加到一起，聚合成一个更大的组，从中搜索。通过这种方式，一个人如果想要搜索最新的新闻，例如从这几千个站点中搜索有关 JAVA 的最新消息，（就只需要在这个组中搜索就可以了），无需到这几千个站点中去搜索。

桌面阅读器——一些工具如 Amphetadesk (http://www.disobey.com/amphetadesk/) 和 NetNewsWire Lite (http://ranchero.com/netnewswire/)可以让用户从桌面上订阅上百个种子（站点）。阅读器可以每个小时定制刷新种子站点内容，这样用户就可以知道最新消息。

脚本－RSS 的最初目的是让网站站长能够将其它站点的内容包含进自己的站点。RSS1.0 仍然在以这种方式使用。许多站点（例如 Slashdot (http://slashdot.org/)）将 RSS 种子集成进网站的首页（front page）。

图 20：MeerKat 和 NewsIsFree

RSS1.0 遵照可扩展原则进行设计。通过引入其它的 RDF 词汇（RSS 开发团体称之为模块（modules）），RSS1.0 作者可以提供大量的元数据并处理。模块可以和其它更通用的 RDF 词汇一样，可以由感兴趣的人员来开发。当前，已经开发了三个官方认可的模块(http://web.resource.org/rss/1.0/)，和 19 个建议模块(http://web.resource.org/rss/1.0/modules/proposed.html)。这些模块已经被大量团体所认同，其覆盖范围从完整的都柏林核心模块（Dublin Core module） (http://web.resource.org/rss/1.0/modules/dc/) 到更加专业的 RSS 中心模块，例如 Aggregation module (http://web.resource.org/rss/1.0/modules/aggregation/).

应当注意一点，当我们在 RDF 的范畴讨论 RSS 的时候，当前有两中 RSS 规范标准。一个分支(RSS 0.91,0.92,0.93,0.94 和 2.0)没有使用 RDF，另外一个分支(RSS 0.9 和 1.0)使用了 RDF。

[编辑]

6.5 CIM/XML

电业部门使用电力系统模型以完成各种不同的任务。例如，在计划和安全分析中电力系统的模拟就是必不可少的。电力系统模型同时也应用于实际的操作中，比如它可以被能源管理系统(Energy Management Systems ,EMS)用于能源控制中心中。一个实际运作的电力系统模型可能由几千种信息类别组成。除了在内部使用这些模型，电业部门为了规划和一些运作目的，比如为了达到协调电力传输或者确保操作可靠性的目的，还需要相互交换系统模型信息。然而，各个电业部门使用了不同的软件来完成这些任务，因此，系统模型以不同的格式存储，这使得这些模型的交换相当困难。

为了使电力系统模型间能够进行信息交换,电业部门需要就电力系统的实体以及实体间关系的定义达成一致。一个非盈利性的电力科学研究协会——电力能源研究协会(Electric Power Research Institute,EPRI)，开发了一个[通用信息模型](Common Information Model,CIM)来完成这一任务。在 CIM 模型中,明确了电力系统的资源,属性以及关系等的通用语义。除此之外，为了能够进一步增强电子化地交换 CIM 模型的能力，电力业已经开发了一种用 XML 表达 CIM 模型的语言 CIM/XML。CIM/XML 是一种 RDF 的应用,使用 RDF 和 RDF Schema 来组织 CIM 模型的 XML 文档结构。北美电力可靠性理事会(North American Electric Reliability Council,NERC)(一个业界组织的以提高北美地区电力传送可靠性为目的的理事会)已经采纳了 CIM/XML 作为电力传输系统运营商间交换模型信息的标准。同时，CIM/XML 格式也通过了国际电工委员会的国际标准化认证过程。在[DWZ01]上有关于 CIM/XML 的大量精彩讨论.[注:请不要将电力业 CIM 与分布式管理任务组(Distributed Management Task Force, DMTF)所开发的 CIM 混淆。DMTF 开发的 CIM 是用以为分布式软件、网络和企业环境表示管理信息的，它也是采用 XML 进行表示的，目前并未采用 RDF，但关于这个方向的独立研究已经展开。

CIM 模型能将一个电业部门的所有主要对象均表示为类、属性以及它们之间的关系.CIM 模型使用这些类和属性以支持独立开发的应用软件之间的集成，这些软件可能应用于特定厂商的 EMS 之间，也可能是应用于某个 EMS 和其他的与系统运作有关的系统之间，像发电和送电的管理等。

CIM 模型被明确表示为一系列使用统一建模语言(Unified Modeling Language，UML)描述的类图。CIM 模型的基类是 PowerSystemResource 类，同时 CIM 模型还包括了 Substation 、Switch、Breaker 等一些特殊化的子类。CIM/XML 将 CIM 表示为 RDF 模式的一个词汇表，并将 RDF/XML 用作为交换特定系统模型间的语言。例 41 展示了 CIM/XML 类和属性定义的例子。

例 41: CIM/XML 类和属性定义的例子

```
<rdfs:Class rdf:ID="PowerSystemResource">
  <rdfs:label xml:lang="en">PowerSystemResource</rdfs:label>
  <rdfs:comment>"A power system component that can be either an
    individual element such as a switch or a set of elements
    such as a substation. PowerSystemResources that are sets
    could be members of other sets. For example a Switch is a
    member of a Substation and a Substation could be a member
    of a division of a Company"</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Breaker">
  <rdfs:label xml:lang="en">Breaker</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Switch" />
  <rdfs:comment>"A mechanical switching device capable of making,
     carrying, and breaking currents under normal circuit conditions
     and also making, carrying for a specified time, and breaking
     currents under specified abnormal circuit conditions e.g. those
     of short circuit. The typeName is the type of breaker, e.g.,
     oil, air blast, vacuum, SF6."</rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="Breaker.ampRating">
   <rdfs:label xml:lang="en">ampRating</rdfs:label>
   <rdfs:domain rdf:resource="#Breaker" />
```

```
    <rdfs:range rdf:resource="#CurrentFlow" />
    <rdfs:comment>"Fault interrupting rating in amperes"</rdfs:comment>
 </rdf:Property>
```

CIM/XML 为了简化模型表达，只使用了 RDF/XML 完全语法的一个子集。并且，CIM/XML 对 RDF Schema 词汇表进行了一些扩展。这些扩展支持逆属性的描述以及多值(基数)约束。多值（基数）约束用来描述对于一个给定资源的一个给定属性允许的作为其值的实例个数（对于一个多值声明允许的值是“零值或单值”、“单值”、“零值或多值”、“单值或多值”）。例 42 中的属性阐述了这些扩展(这些扩展的部分用 QName 前缀 cims 标识)：

例 42：一些 RDF 模式的 CIM/XML 扩展

```
 <rdf:Property rdf:ID="Breaker.OperatedBy">
    <rdfs:label xml:lang="en">OperatedBy</rdfs:label>
    <rdfs:domain rdf:resource="#Breaker" />
    <rdfs:range rdf:resource="#ProtectionEquipment" />
    <cims:inverseRoleName rdf:resource="#ProtectionEquipment.Operates" />
    <cims:multiplicity rdf:resource="http://www.cim-logic.com/schema/990530#M:0..n" />
    <rdfs:comment>"Circuit breakers may be operated by
        protection relays."</rdfs:comment>
 </rdf:Property>

 <rdf:Property rdf:ID="ProtectionEquipment.Operates">
    <rdfs:label xml:lang="en">Operates</rdfs:label>
    <rdfs:domain rdf:resource="#ProtectionEquipment" />
    <rdfs:range rdf:resource="#Breaker" />
    <cims:inverseRoleName rdf:resource="#Breaker.OperatedBy" />
    <cims:multiplicity rdf:resource="http://www.cim-logic.com/schema/990530#M:0..n" />
    <rdfs:comment>"Circuit breakers may be operated by
```

```
        protection relays.″</rdfs:comment>

 </rdf:Property>
```

通过使用 CIM/XML 交换各个厂商产品之间的实际的大规模模型(包括像在一个测试中使用到的 2000 多所变电站的描述数据)，并通过验证这些模型能被有代表性的应用软件所正确解释,电力科学研究院已经成功的进行了互操作的测试。尽管 CIM 模型原先是被计划用于电力管理系统，但现在它已经被扩展到电力传输以及其他方面的一些应用领域了。

对象管理组织（Object Management Group，OMG）已经采用了一个称为数据访问辅助工具（Data Access Facility ，[DAF]）的对象接口标准来对 CIM 电力系统模型进行访问。类似 CIM/XML 语言，DAF 也是基于 RDF 模型的，并且与 CIM/XML 共享相同的 CIM 模式。.然而，CIM/XML 目的是使一个模型能够以文档的方式被交换，而 DAF 则是使应用软件能够像访问一个对象集合一样的访问这个模型。

基于 XML 的信息交换很自然地能被表示为实体－关系模型或者面向对象的类、属性和关系(甚至不要求这些信息是能通过 Web 访问的)，CIM/XML 阐明了 RDF 在这其中能够扮演的重要角色。在这些情况下，RDF 为 XML 提供了一个基本架构以支持对象标识，以及支持在结构化的关系中使用这些对象。大量使用 RDF/XML 进行信息交换的应用软件阐明了这一关系，同时大量研究 RDF(或者像 OWL 等的本体语言)和 UML(以及它的 XML 表示)之间联系的工程也阐明了这一关系。CIM/XML 需要扩展 RDF 模式才能支持基数约束和逆属性，这类的需要促使人们开发更强大的基于 RDF 的 Schema 或本体语言，如 DAML+OIL 和在 5.5 节描述的 OWL 等。在将来，这些语言在支持一些类似的模型应用就可能更为合适。

最后，CIM/XML 还为那些正在寻找另外的“RDF 在领域中的应用”的例子的那些人阐述了一个重要事实：一些语言被描述为“XML”语言，或者系统被描述为使用“XML”的，而它们实际使用的“XML”就是 RDF/XML,也即它们就是关于 RDF 的应用。有时需要相当深入地了解这些语言或者这些系统的描述才能发现这点(在某些已知的例子中，RDF 甚至根本没有被明确的提及，但是样例数据清楚地显示这是 RDF/XML)。同时，在类似于 CIM/XML 的这些应用中，由于创建的 RDF 是被计划用作软件部件间的信息交换而不是用于通用性的访问(尽管我们可以假想在未来，更多这种类型的 RDF 能够通过 Web 访问)，因此创建出的 RDF 在 Web 上并不那么容易找到。

[编辑]

6.6 基因本体协会

使用像 SNOMED RT (医学文献术语系统化命名法, Systematized Nomenclature of Medicine Reference Terminology) 和 MeSH (医学学科标题，Medical Subject Headings) 之类的受控词汇表的结构化元数据在医学界扮演着越来越重要的角色。这些结构化元数据提高了文献搜索的效率，并有助于医学知识[COWAN]的传播与交换。但是，由于医学领域知识在不断地发生快速变化，这就要求人们开发额外的词汇表以供使用。

基因本体协会(Gene Ontology (GO) Consortium)的[基因本体([GO])]旨在提供一个受控词汇表，用以描述基因产品某些方面的特征。相互协作的数据库使用基因本体术语标注它们的基因产品（或者基因），引用并指明了哪些证据可用以支持这些注释。这些数据库使用通用基因本体术语，更加容易实现数据库之间的统一查询。结构化的基因本体可以在不同的粒度等级上进行归因与查询。由于有关基因和蛋白质在细胞中功能的知识在不断积累和更新，因此基因本体词汇表也是动态变化的。

基因本体的三个组织原则是“分子功能”，“生物过程”和“细胞成分”。某个基因产品可能有一种或多种分子功能，并有可能作用于一个或多个生物过程之中；该产品可能包含了一种或多种的细胞成分，也可能与一种或多种细胞成分相关联。这三种本体中所有的术语定义都包含在一个单独的（文本格式的）定义文件中。每月都会定时生成一些 XML 格式的文件，其中包含了这三种本体文件以及所有可用的（本体）定义。

功能、过程和成分被表示为有向无环图（DAGs）或者网络。一个子术语可能是其父术语的一个“实例”（isa-关系），或者是其父术语的一个成分（part-of 关系）。一个子术语可能有多个父术语并可能与其不同的父术语间有着不同的关系类别。本体还可以表示同义词和对外部数据库的交叉引用。由于 RDF/XML 在表示图结构时有很大的灵活性，并且有着广泛的工具支持，基因本体使用 RDF/XML 来简化 XML 版本的本体中的术语间的关系表示。但是，基因本体目前在术语描述中使用了“非”RDF 嵌套的 XML 结构，所以现在所使用的基因本体描述语言并不是纯粹的 RDF/XML。

例 43 展示了一些基因本体文档中的基因本体信息的样例:

例 43：基因本体信息样例

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE go:go>

<go:go xmlns:go="http://www.geneontology.org/xml-dtd/go.dtd#"

       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <go:version timestamp="Wed May 9 23:55:02 2001" />


  <rdf:RDF>

     <go:term rdf:about="http://www.geneontology.org/go#GO:0003673">

        <go:accession>GO:0003673</go:accession>

        <go:name>Gene_Ontology</go:name>

        <go:definition></go:definition>

     </go:term>


     <go:term rdf:about="http://www.geneontology.org/go#GO:0003674">

        <go:accession>GO:0003674</go:accession>

        <go:name>molecular_function</go:name>

        <go:definition>The action characteristic of a gene product.</go:definition>

        <go:part-of rdf:resource="http://www.geneontology.org/go#GO:0003673" />

        <go:dbxref>

           <go:database_symbol>go</go:database_symbol>

           <go:reference>curators</go:reference>

        </go:dbxref>

     </go:term>
```

```
<go:term rdf:about="http://www.geneontology.org/go#GO:0016209">
   <go:accession>GO:0016209</go:accession>
   <go:name>antioxidant</go:name>
   <go:definition></go:definition>
   <go:isa rdf:resource="http://www.geneontology.org/go#GO:0003674" />
   <go:association>
      <go:evidence evidence_code="ISS">
         <go:dbxref>
            <go:database_symbol>fb</go:database_symbol>
            <go:reference>fbrf0105495</go:reference>
         </go:dbxref>
      </go:evidence>
      <go:gene_product>
         <go:name>CG7217</go:name>
         <go:dbxref>
            <go:database_symbol>fb</go:database_symbol>
            <go:reference>FBgn0038570</go:reference>
         </go:dbxref>
      </go:gene_product>
   </go:association>
   <go:association>
      <go:evidence evidence_code="ISS">
         <go:dbxref>
            <go:database_symbol>fb</go:database_symbol>
```

```
            <go:reference>fbrf0105495</go:reference>
         </go:dbxref>
      </go:evidence>
      <go:gene_product>
         <go:name>Jafrac1</go:name>
         <go:dbxref>
            <go:database_symbol>fb</go:database_symbol>
            <go:reference>FBgn0040309</go:reference>
         </go:dbxref>
      </go:gene_product>
   </go:association>
</go:term>
</rdf:RDF>
</go:go>
```

例 43 说明了 go:term 是基本元素。在某些情况下，基因本体没有使用 RDF Schema，而定义了它自己的术语。例如，术语 GO:0016209 有一个元素 <go:isa rdf:resource="http://www.geneontology.org/go#GO:0003674" />。这一标记表示了"GO:0016209 isa GO:0003674"这一关系，或者用英语来说，表示了“Molecular function is part of the Gene Ontology（抗氧化剂是一个分子功能）”。另一个关系特例是 go:part-of。例如，GO:0003674 有一个元素是<go:part-of rdf:resource="http://www.geneontology.org/go#GO:0003673" />。这表明了“Molecular function is part of the Gene Ontology（分子功能是基因本体的一部分）”。

任何一个标注必须被归因到一个源，这个源可以是一份参考文献、一个数据库或者一个计算分析。标注必须指明在引用的源中找到了哪些证据以支持基因产品与这一基因术语的关联。一个简单的受控词汇表可以用来记录这些证据。例子包括：

ISS 意思是“inferred from sequence similarity [with <database:sequence_id>]（根据基因序列相似性推断[用<database:sequence_id>]）”

IDA 意思是“inferred from direct assay（根据直接化验推断）”

TAS 意思是“traceable author statement（可追踪的作者论述）”

go:dbxref 元素表示在外部数据库的术语，而 go:association 表示每个术语的基因关联。go:association 能同时含有 go:association 和 go:gene_product，前者包含了指向支持关联证据的 go:dbxref，后者则包含了基因符号与 go:dbxref。这些元素说明了基因本体的 XML 语法并不是“纯粹的”RDF/XML，因为这些元素中嵌套着其他元素这一事实并未遵循[RDF-SYNTAX]的 2.1 节与 2.2 节中所描述的交替节点/谓词弧的条状嵌套语法。

基因本体说明了许多有趣的要点。首先，它表明使用 RDF 结构化 XML 能增强 XML 在信息交换中的作用。在数据结构整体上是一个图结构或者网络结构而不是一个严格层次结构的情况下，这一点将体现得更加明显。同时，基因本体作为一个例子表明了使用 RDF 的数据并不一定要以在 Web 上直接使用的形式出现（尽管所有的文件通过 Web 都是可访问的）。而基因本体也可作为一个例子，表明数据表面上使用“XML”进行描述，但是进一步观察可发现其使用的是 RDF/XML（尽管不是“纯粹的”RDF/XML）。最后，基因本体阐明了 RDF 在本体本体中所扮演的基础性角色。在 5.5 节中讨论过 DAML+OIL 与 OWL 等基于 RDF 的更加丰富的本体描述语言,一旦这些语言得到广泛使用，RDF 的作用将会更进一步得到增强。实际上，一个下一代基因本体（Gene Ontology Next Generation）的项目已经开始使用这些更丰富的语言来展开基因本体表示的相关研究。

[编辑]

6.7 描述设备性能和用户参数

近几年新出现了很多用于浏览 Web 的移动设备（mobile devices）。这些设备中有许多具有高度分散的能力，包括选择广泛的输入与输出能力，以及不同级别的语言支持。移动设备在网络连接能力方面也可能有着广泛的不同。这些新设备的用户期望有一个便于使用的外观显示，而不管其设备能力与当前的网络特性如何。同样地，用户希望在内容或应用呈现给他们时能够考虑到他们动态变化的偏好（比如是否开启声音）。然而在现实中，由于设备的异构性以及缺少一种让用户向服务器传达个人偏好的标准方式，这会造成：得到的内容可能不能存贮在设备里、不能显示出来或者违反了用户意愿。此外，最终的内容可能会在网络传输上花很多时间才到达客户端设备。

处理这些问题的一个办法是让客户端编码*递送上下文（delivery context）*（即设备能力、用户偏好、网络特性等等）使得服务端可以利用这个上下文为设备和用户进行内容的定制（关于递送上下文的定义，请参见[DIPRINC]）。W3C 的组合能力/偏好特征(Composite Capabilities/Preferences Profile, CC/PP)规范[CC/PP]通过定义一个描述递送上下文的通用框架促进了这个问题的解决。

CC/PP 框架定义了一个相对简单的结构，即一个由组件和属性/值对构成的 2 层结构。一个*组件（component）*可被用来描述递送上下文的一部分（如网络特性、设备支持的软件或设备的硬件特性）。一个组件可以包含一个或多个*属性（attributes）*。例如，一个表示用户偏好的组件可能会需要包含一个指定是否打开*音频输出*的属性。

CC/PP 用 RDF Schema 定义上述层次结构（关于此结构的 Schema 详细信息，请参见[CC/PP]）。一个 CC/PP *词汇表*定义了一些特定的组件和各自的属性，但[CC/PP]不定义这些词汇。这些词汇是由其他机构或应用（如下所述）定义的。[CC/PP]也没有定义传输 CC/PP 词汇实例的协议。

CC/PP 词汇表的一个实例称为一个*特征文件（profile）*。在特征文件中，CC/PP 属性(attribute)用 RDF 属性(property)表示。例 44 所显示的是一个特征文件的片断，它描述了用户的打开音频输出的偏好：

例 44：一个 CC/PP 特征文件的片断

```
<ccpp:component>
   <rdf:Description rdf:ID="UserPreferences">
   <rdf:type rdf:resource="http://www.example.org/profiles/prefs/v1_0#UserPreferences"/>
   <ex:AudioOutput>Yes</ex:AudioOutput>
   <ex:Graphics>No</ex:Graphics>
   <ex:Languages>
    <rdf:Seq>
     <rdf:li>en-cockney</rdf:li>
```

```
      <rdf:li>en</rdf:li>
     </rdf:Seq>
    </ex:Languages>
   </rdf:Description>
 </ccpp:component>
```

在这个应用中使用 RDF 有几个优势。第一，以 CC/PP 编码的特征文件包含的属性可能是在由不同组织创建的 Schema 中定义的。RDF 很自然地适合表示这些特征文件，因为没有单个组织能够为聚集的特征数据创建一个*超级* Schema。 使用 RDF 的第二个好处是，它使得在特征文件中插入任意属性变得更为容易（由于 RDF 是基于图的数据模型），这对包含了频繁变化的数据（如地点信息）的特征文件尤为有用。

开放移动联盟（Open Mobile Alliance）已经定义了用户代理特征文件（User Agent Profile, UAProf），[UAPROF]。它是一个基于 CC/PP 的框架，包含了一个用于描述设备能力、用户代理能力、网络特性等的词汇表，并定义了一个传输特征文件的协议。UAProf 定义了六个组件，包括*硬件平台（HardwarePlatform）*、*软件平台（SoftwarePlatform）*、*网络特性（NetworkCharacteristics）*和*用户浏览器（BrowserUA）*等。它还为每个组件定义了一些属性，尽管组件的属性并是不固定的——可能被补充或撤销。20040210/#example45 例 45 显示了一个关于*硬件平台*组件的 UAProf 片断：

例 45：一个关于硬件平台组件的 UAProf 片断：

```
 <prf:component>
  <rdf:Description rdf:ID="HardwarePlatform">
   <rdf:type rdf:resource="http://www.openmobilealliance.org/profiles/UAPROF/ccppschema-20021113#HardwarePlatform"/>
   <prf:ScreenSizeChar>15x6</prf:ScreenSizeChar>
   <prf:BitsPerPixel>2</prf:BitsPerPixel>
   <prf:ColorCapable>No</prf:ColorCapable>
   <prf:BluetoothProfile>
    <rdf:Bag>
     <rdf:li>headset</rdf:li>
     <rdf:li>dialup</rdf:li>
     <rdf:li>lanaccess</rdf:li>
    </rdf:Bag>
```

```
  </prf:BluetoothProfile>
 </rdf:Description>
</prf:component>
```

UAProf 协议支持*静态*特征文件和*动态*特征文件。一个*静态*特征文件是通过 URI 访问的。这有几个优点：在客户发给服务器的请求中只含有一个 URI，而不是一个冗长的 XML 文档（这将减少网络拥塞）；客户不需要存储和/或创建这个特征文件；客户端的实现负担是相对轻量级的。*动态*特征文件是动态创建的，因此没有相关联的 URI。它可以包含一个描述与静态特征文件*差别*的特征文件片断，也可以包含客户的静态特征文件中所没有的数据。一个请求可以包含任意数量的静态特征文件和动态特征文件。然后，特征文件的顺序是重要的，因为在请求中后面的特征文件会覆盖前面的。关于 UAProf 的协议和处理多个特征文件的规则，详见[UAPROF]。

一些其他的团体（例如，3GPP's TS 26.234 [3GPP]和 WAP 论坛的多媒体消息服务客户端事务规范（Multimedia Messaging Service Client Transactions Specification [MMS-CTR]））已经定义了一些基于 CC/PP 的词汇表。因此，一个特征文件可以利用 RDF 的分布式特点，包含来自不同词汇表定义的组件。http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#example46 例 46]显示了一个这样的特征文件：

例 46：一个用到多个词汇表的特征文件

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:prf="http://www.wapforum.org/profiles/UAPROF/ccppschema-20010330#"
         xmlns:mms="http://www.wapforum.org/profiles/MMS/ccppschema-20010111#"
     xmlns:pss="http://www.3gpp.org/profiles/PSS/ccppschema-YYYYMMDD#">
 <rdf:Description rdf:ID="SomeDevice">
  <prf:component>
   <rdf:Description rdf:ID="Streaming">
    <rdf:type rdf:resource="http://www.3gpp.org/profiles/PSS/ccppschema-PSS5#Streaming"/>
    <pss:AudioChannels>Stereo</pss:AudioChannels>
    <pss:VideoPreDecoderBufferSize>30720</pss:VideoPreDecoderBufferSize>
    <pss:VideoInitialPostDecoderBufferingPeriod>0</pss:VideoInitialPostDecoderBufferingPeriod>
    <pss:VideoDecodingByteRate>16000</pss:VideoDecodingByteRate>
   </rdf:Description>
  </prf:component>
```

```
<prf:component>
 <rdf:Description rdf:ID="MmsCharacteristics">
  <rdf:type rdf:resource="http://www.wapforum.org/profiles/MMS/ccppschema-20010111#Streaming"/>
  <mms:MmsMaxMessageSize>2048</mms:MmsMaxMessageSize>
  <mms:MmsMaxImageResolution>80x60</mms:MmsMaxImageResolution>
  <mms:MmsVersion>2.0</mms:MmsVersion>
 </rdf:Description>
</prf:component>

<prf:component>
 <rdf:Description rdf:ID="PushCharacteristics">
  <rdf:type rdf:resource="http://www.openmobilealliance.org/profiles/UAPROF/ccppschema-20010330#PushCharacteristics"/>
  <prf:Push-MsgSize>1024</prf:Push-MsgSize>
  <prf:Push-MaxPushReq>5</prf:Push-MaxPushReq>
  <prf:Push-Accept>
   <rdf:Bag>
    <rdf:li>text/html</rdf:li>
    <rdf:li>text/plain</rdf:li>
    <rdf:li>image/gif</rdf:li>
   </rdf:Bag>
  </prf:Push-Accept>
 </rdf:Description>
</prf:component>
```

```
  </rdf:Description>

 </rdf:RDF>
```

关于递送上下文以及其中数据的定义将继续发展。从而，RDF 天生的可扩展性以及因此而对动态变化的词汇表的支持使得 RDF 成为一个表达递送上下文的理想框架。

[编辑]

7. RDF 规范的其它部分

第 1 节 指出了 RDF 规范包含了一系列的文档(除了这个入门外):

RDF 概念和抽象语法 [RDF-CONCEPTS]

RDF/XML 语法规范 [RDF-SYNTAX]

RDF 词汇描述语言 1.0: RDF Schema [RDF-VOCABULARY]

RDF 语义 [RDF-SEMANTICS]

[http://www.w3.org/TR/rdf-testcases/ RDF 测试用例 [RDF-TESTS]

The Primer has already discussed the subjects of several of these documents, basic RDF concepts (in [http://www.w3.org/TR/2004/REC-rdf 这个入门文档已经讨论了这些文档的一些主题，如基本的 RDF 概念(在第 2 节), RDF/XML 语法 (在 第 3 节) 和 RDF Schema (在 第 5 节). 这节简要地讲述一下剩下的文档(尽管已经有很多对 [RDF-SEMANTICS]的引用)，目的是解释他们在完整的 RDF 规范中的角色.

[编辑]

7.1 RDF 语义

就像前面章节讨论的那样, RDF 是用以表示关于资源的陈述的，它的表示形式是图并使用了一些特定的词汇(资源,属性,类的名字等等). RDF 还是用以作为高级语言的基础，如在 5.5 节中讨论的一些语言. 为此, RDF 图的含义必须得到精确的定义.

一般来说,到底什么组成了 RDF 图的"含义"依赖于很多因素，包括在用户团体中解释用户以特定方式定义的类和属性的一些惯例，自然语言书写的注释，对其他具有内容的信息的链接. 如 2.2 节所述，大部分以这种形式表达的含义不能直接被机器处理，尽管这些含义可能被人们解释 RDF 信息时用到,或被程序员在编写软件以对 RDF 信息作出各种处理时用到. 然而, RDF 陈述也具有一种"形式化"的语义，它决定了机器用一种数学的精确方式，从一个给定的 RDF 图中可以得到的结论. 文档 RDF 语义 [RDF-SEMANTICS] 用一个规范说明形式语言语义的技术，称为模型论，来定义这个形式化语义. [RDF-SEMANTICS] 也定义了 RDF 语言的扩展 RDF Schema 和单独的数据类型的语义扩展. 换句话说, RDF 模型论语义提供了所有 RDF 概念的理论基础. 基于在模型论中定义的语义, RDF 图可以容易地翻译到具有相同含义的逻辑表达式上.

[编辑]

7.2 测试用例

文档 RDF 测试用例 [RDF-TESTS] 为用文字描述的 RDF 规范补充了测试用例(例子)，这些用例对应于 RDF 核心工作组解决的一些特定的技术问题. 为了方便描述这些用例，那个测试用例文档引入了一个称为 N-Triples 的表示法，这也是本入门文档中所用的三元组表示法的基础. 这些测试用例以机器可读的方式发布在这个测试用例文档所在的 Web 站点，因此，开发人员可以把这些作为自动化测试 RDF 软件的基础.

这些测试用例被划分为以下几类:

肯定的和否定的语法分析器(Parser)测试：他们测试 RDF/XML Parser 是否能够为合法的 RDF/XML 输入文档产生正确的 N-Triples 输出图，或为不合法的 RDF/XML.输入文档正确地报告错误.

肯定的和否定的蕴涵(Entailment)测试：他们测试是否能够从指定的 RDF 陈述集合中得到正确的或不正确的结论.

带有数据类型的蕴涵测试：他们是用到了数据类型的肯定的或否定的蕴涵测试，因此需要对测试中用到的数据类型的支持.

其他测试：他们是不在上述类别中的测试.

这些测试用例不是对 RDF 的完全规格说明，也没有优先于其他规范文档的意思。相反，它们是用来阐明 RDF 核心工作组（RDF Core Working Group）在 RDF 的设计方面的意图。开发人员也许会觉得这些测试用例很有帮助，如果那些规范在某些细节的措辞上不够清晰的话

[编辑]

8. 参考文献

[编辑]

8.1 规范性文献

[RDF-CONCEPTS]

Resource Description Framework (RDF): Concepts and Abstract Syntax, Klyne G., Carroll J. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-primer-20040210/. The latest version is http://www.w3.org/TR/rdf-concepts/.

[RDF-MIME-TYPE]

MIME Media Types, The Internet Assigned Numbers Authority (IANA). This document is http://www.iana.org/assignments/media-types/ . The registration for application/rdf+xml is archived at http://www.w3.org/2001/sw/RDFCore/mediatype-registration .

[RDF-MS]

Resource Description Framework (RDF) Model and Syntax Specification, Lassila O., Swick R. (Editors), World Wide Web Consortium, 22 February 1999. This version is http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. The latest version is http://www.w3.org/TR/REC-rdf-syntax/.

[RDF-SEMANTICS]

RDF Semantics, Hayes P. (Editor), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. The latest version is http://www.w3.org/TR/rdf-mt/.

[RDF-SYNTAX]

RDF/XML Syntax Specification (Revised), Beckett D. (Editor), W3C Recommendation, 10 February 2004. This version http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. The latest version is http://www.w3.org/TR/rdf-syntax-grammar/.

[RDF-TESTS]

RDF Test Cases, Grant J., Beckett D. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/. The latest version is http://www.w3.org/TR/rdf-testcases/.

[RDF-VOCABULARY]

RDF Vocabulary Description Language 1.0: RDF Schema, Brickley D., Guha R.V. (Editors), W3C Recommendation, 10 February 2004. This version is http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. The latest version is http://www.w3.org/TR/rdf-schema/.

[UNICODE]

The Unicode Standard, Version 3, The Unicode Consortium, Addison-Wesley, 2000. ISBN 0-201-61633-5, as updated from time to time by the publication of new versions. (See http://www.unicode.org/unicode/standard/versions/ for the latest version and additional information on versions of the standard and of the Unicode Character Database).

[URIS]

RFC 2396 - Uniform Resource Identifiers (URI): Generic Syntax, Berners-Lee T., Fielding R., Masinter L., IETF, August 1998, http://www.isi.edu/in-notes/rfc2396.txt.

[XML]

Extensible Markup Language (XML) 1.0, Second Edition, Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (Editors), World Wide Web Consortium, 6 October 2000. This version is http://www.w3.org/TR/2000/REC-xml-20001006. The latest version is http://www.w3.org/TR/REC-xml.

[XML-BASE]

XML Base, Marsh J. (Editor), World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xmlbase-20010627/. The latest version is http://www.w3.org/TR/xmlbase/.

[XML-NS]

Namespaces in XML, Bray T., Hollander D., Layman A. (Editors), World Wide Web Consortium, 14 January 1999. This version is http://www.w3.org/TR/1999/REC-xml-names-19990114/. The latest version is http://www.w3.org/TR/REC-xml-names/.

[XML-XC14N]

Exclusive XML Canonicalization Version 1.0, Boyer J., Eastlake D.E. 3rd, Reagle J. (Authors/Editors), World Wide Web Consortium, 18 July 2002. This version is http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/. The latest version is http://www.w3.org/TR/xml-exc-c14n/.

[编辑]

8.2 参考性文献

[3GPP]

3GPP TS 26.234. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Transparent end-to-end packet switched streaming service; Protocols and codecs V5.2.0 (2002-09). This document is available at http://www.3gpp.org/specs/specs.htm via directory ftp://ftp.3gpp.org/specs/2002-09/Rel-5/26_series/.

[ADDRESS-SCHEMES]

Addressing Schemes, Connolly D., 2001. This document is http://www.w3.org/Addressing/schemes.html.

[BATES96]

Indexing and Access for Digital Libraries and the Internet: Human, Database, and Domain Factors, Bates M.J., 1996. This document is http://is.gseis.ucla.edu/research/mjbates.html.

[BERNERS-LEE98]

What the Semantic Web can represent, Berners-Lee T., 1998. This document is http://www.w3.org/DesignIssues/RDFnot.html.

[CC/PP]

Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies, Klyne G., Reynolds F., Woodrow C., Ohto H., Hjelm J., Butler M., Tran, L., W3C Recommendation, 15 January 2004. This version is http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/. The latest version is http://www.w3.org/TR/CCPP-struct-vocab/.

[CG]

Conceptual Graphs, Sowa J., ISO working document ISO/JTC1/SC32/WG2 N 000, 2 April 2001 (work in progress). Available at http://users.bestweb.net/~sowa/cg/cgstand.htm.

[CHARMOD]

Character Model for the World Wide Web 1.0, Dürst M., Yergeau F., Ishida R., Wolf M., Freytag A., Texin T. (Editors), World Wide Web Consortium, 20 February 2002 (work in progress). This version is http://www.w3.org/TR/2002/WD-charmod-20020220/. The latest version is http://www.w3.org/TR/charmod/.

[CIM]

Common Information Model (CIM): CIM 10 Version, EPRI, Palo Alto, CA: 2001, 1001976. This document is available at http://www.epri.com/attachments/286161_1001976(1).pdf (267pp.).

[COWAN]

Metadata, Reuters Health Information, and Cross-Media Publishing , Cowan J., 2002. Presentation at Seybold New York 2002 Enterprise Publishing Conference. This document is http://seminars.seyboldreports.com/seminars/2002_new_york/presentations/014/cowan_john.ppt. An accompanying transcript is http://seminars.seyboldreports.com/2002_new_york/files/transcripts/doc/transcript_EP7.doc

[DAF]

Utility Management System (UMS) Data Access Facility, version 2.0, Object Management Group, November 2002. This document is available at http://www.omg.org/technology/documents/formal/UMS_Data_Access_Facility.htm.

[DAML+OIL]

DAML+OIL (March 2001) Reference Description, Connolly D., van Harmelen F., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A., World Wide Web Consortium, 18 December 2001. This document is http://www.w3.org/TR/daml+oil-reference.

[DC]

Dublin Core Metadata Element Set, Version 1.1: Reference Description, 02 June 2003. This version is

http://dublincore.org/documents/2003/06/02/dces/. The latest version is http://dublincore.org/documents/dces/.

[DIPRINC]

Device Independence Principles. Gimson, R., Finkelstein, S., Maes, S., Suryanarayana, L., World Wide Web Consortium, 18 September 2001 (work in progress). This version is http://www.w3.org/TR/2001/WD-di-princ-20010918. The latest version is http://www.w3.org/TR/di-princ/.

[DWZ01]

XML for CIM Model Exchange , deVos A., Widergreen S.E., Zhu J., Proc. IEEE Conference on Power Industry Computer Systems, Sydney, Australia, 2001. This document is http://www.langdale.com.au/PICA/.

[GO]

Gene Ontology: tool for the unification of biology, The Gene Ontology Consortium, *Nature Genetics*, Vol. 25: 25-29, May 2000. Available at http://www.geneontology.org/GO_nature_genetics_2000.pdf

[GRAY]

Logic, Algebra and Databases, Gray P., Ellis Horwood Ltd., 1984. ISBN 0-85312-709-3, 0-85312-803-0, 0-470-20103-7, 0-470-20259-9.

[HAYES]

In Defense of Logic, Hayes P., Proceedings from the International Joint Conference on Artificial Intelligence, 1975, San Francisco. Morgan Kaufmann Inc., 1977. Also in *Computation and Intelligence: Collected Readings*, Luger G. (ed), AAAI press/MIT press, 1995. ISBN 0-262-62101-0.

[KIF]

Knowledge Interchange Format, Genesereth M., draft proposed American National Standard NCITS.T2/98-004. Available at http://logic.stanford.edu/kif/dpans.html.

[LBASE]

LBase: Semantics for Languages of the Semantic Web, Guha R. V., Hayes P., W3C Note, 10 October 2003. This version is http://www.w3.org/TR/2003/NOTE-lbase-20031010/. The latest version is http://www.w3.org/TR/lbase/.

[LUGER]

Artificial Intelligence: Structures and Strategies for Complex Problem Solving (3rd ed.), Luger G., Stubblefield W., Addison Wesley Longman, 1998. ISBN 0-805-31196-3.

[MATHML]

Mathematical Markup Language (MathML) Version 2.0, Carlisle D., Ion P., Miner R., Poppelier N. (Editors); Ausbrooks R., Buswell S., Dalmas S., Devitt S., Diaz A., Hunter R., Smith B., Soiffer N., Sutor R., Watt S. (Principal Authors), World Wide Web Consortium, 21 February 2001. This version is http://www.w3.org/TR/2001/REC-MathML2-20010221/. The latest version is http://www.w3.org/TR/MathML2/.

[MMS-CTR]

Multimedia Messaging Service Client Transactions Specification. WAP-206-MMSCTR-20020115-a. This document is available at http://www.openmobilealliance.org/.

[NAMEADDRESS]

Naming and Addressing: URIs, URLs, ..., Connolly D., 2002. This document is http://www.w3.org/Addressing/.

[OWL]

OWL Web Ontology Language Reference, Dean M., Schreiber G (Editors); van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A. (Authors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/owl-ref/.

[PRISM]

PRISM: Publishing Requirements for Industry Standard Metadata, Version 1.1, 19 February 2002. The latest version of the PRISM specification is available at http://www.prismstandard.org/.

[RDFISSUE]

RDF Issue Tracking, McBride B., 2002. This document is http://www.w3.org/2000/03/rdf-tracking/.

[RDF-S]

Resource Description Framework (RDF) Schema Specification 1.0 , Brickley D., Guha, R.V. (Editors), World Wide Web Consortium. 27 March 2000. This version is http://www.w3.org/TR/2000/CR-rdf-schema-20000327/.

[RSS]

RDF Site Summary (RSS) 1.0, Beged-Dov G., Brickley D., Dornfest R., Davis I., Dodds L., Eisenzopf J., Galbraith D., Guha R.V., MacLeod K., Miller E., Swartz A., van der Vlist E., 2000. This document is http://purl.org/rss/1.0/spec.

[RUBY]

Ruby Annotation, Sawicki M., Suignard M., Ishikawa M., Dürst M., Texin T. (Editors), World Wide Web Consortium, 31 May 2001. This version is

http://www.w3.org/TR/2001/REC-ruby-20010531/. The latest version is http://www.w3.org/TR/ruby/.

[SOWA]

Knowledge Representation: Logical, Philosophical and Computational Foundations, Sowa J., Brookes/Cole, 2000. ISBN 0-534-94965-7.

[SVG]

Scalable Vector Graphics (SVG) 1.1 Specification, Ferraiolo J., Fujisawa J., Jackson D. (Editors), World Wide Web Consortium, 14 January 2003. This version is http://www.w3.org/TR/2003/REC-SVG11-20030114/. The latest version is http://www.w3.org/TR/SVG11/.

[UAPROF]

User Agent Profile. OMA-WAP-UAProf-v1_1. This document is available at http://www.openmobilealliance.org/.

[WEBDATA]

Web Architecture: Describing and Exchanging Data, Berners-Lee T., Connolly D., Swick R., World Wide Web Consortium, 7 June 1999. This document is http://www.w3.org/1999/04/WebData.

[XLINK]

XML Linking Language (XLink) Version 1.0, DeRose S., Maler E., Orchard D. (Editors), World Wide Web Consortium, 27 June 2001. This version is http://www.w3.org/TR/2001/REC-xlink-20010627/. The latest version is http://www.w3.org/TR/xlink/.

[XML-SCHEMA2]

XML Schema Part 2: Datatypes, Biron P., Malhotra A. (Editors), World Wide Web Consortium. 2 May 2001. This version is http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/. The latest version is http://www.w3.org/TR/xmlschema-2/.

[XPACKAGE]

XML Package (XPackage) 1.0 , Wilson G., Editor's Working Draft, 6 March 2003. This version is http://www.xpackage.org/specification/xpackage-draft-20030306.html. The latest version is http://www.xpackage.org/specification/.

[编辑]

9. 致谢

本文档得益于 RDF 核心工作组中诸多成员的共同努力。尤其感谢 Art Barstow, Dave Beckett, Dan Brickley, Ron Daniel, Ben Hammersley, Martyn Horner, Graham Klyne, Sean Palmer, Patrick Stickler, Aaron Swartz, Ralph Swick 以及 Garret Wilson，正是他们以及其它许多研究人员在对上一个版本的 RDF 入门提出了许多宝贵意见之后，才得以产生本文。

另外，Pat Hayes, Sergey Melnik 和 Patrick Stickler 也对本文做出了重要贡献，他们领导开发了 RDF 规范系列中所描述的 RDF 数据类型工具。

Frank Manola 还要感谢， MITRE 公司,Frank 的主管， 在其完成本文的绝大部分时间里所给予的大力支持，RDF 核心工作组的活动得到了该公司资助的研究授权。

[编辑]

附录 A：更多关于统一资源标识符（Uniform Resource Identifiers，URIs）的更多信息

注意：附录 A 旨在提供一个关于 URIs 的简要介绍。URIs 的权威规范是 RFC 2396[URIS]，如需了解更多信息请参阅该文档。关于 URIs 的补充论述可以在下列地址处获得：Naming and Addressing: URIs, URLs, ... [NAMEADDRESS]。

如 2.1 节所述，Web 提供了一种通用形式的标识符——统一资源标识符（Uniform Resource Identifier）（URI），用于标识（命名）Web 上的资源。与 URLs 不同，URIs 并不局限于标识有网络位置或使用其它计算机访问机制的资源。已有许多不同的 *URI 方案（URI scheme）*被开发并正在用于不同用途，比如：

http: （超文本传输协议（Hypertext Transfer Protocol[译注：简称 HTTP]），用于标识网页）

mailto: （email 地址），例如 mailto:em@w3.org

ftp: （文件传输协议（File Transfer Protocol[译注//简称 FTP]））

urn: （统一资源名称（Uniform Resource Names[译注：简称 URN]），用于标识永久与位置无关的资源），例如 urn:isbn:0-520-02356-0（标识一本书）

现有 URI 方案（URI schemes）的列表可在 Addressing Schemes [ADDRESS-SCHEMES]获得。一个不错的想法是考虑在各种专门的标识用途中利用现有 URI 方案，而不是试图创造一种新的方案。

URI 由谁创建和如何使用并不是由某个人或组织控制的。然而某些 URI 方案（比如 URL 的 http:） 要依赖于集中式系统（比如 DNS），而其它一些方案（比如 freenet:）则是完全分散式的。这意味着，如其它类型的名称一样，人们不需要特别的权威或权限来为事物创建 URI。而且，任何人都可以创建 URIs 来引用不为他们所拥有的事物，就像在日常语言中任何人都可以用任何他们乐意的名称来称呼不为他们所有拥的事物一样。

如 2.1 节所谈到的，RDF 使用 *URI 引用（URI references）*[URIS]来命名 RDF 陈述中的主体、谓词和客体。一个 URI 引用（或简称 *URIref*）是一个可以在末尾附加（也可以不附加）*片段标识符（fragment identifier）*[译注//片段标识符这一术语有历史渊源，它的实际作用也许并不是标识某个片段。]的 URI。举例来说，URI 引用 http://www.example.org/index.html#section2 包含了 URI http://www.example.org/index.html 和（以"#"字符分隔的）片段标识符 Section2。RDF URIrefs 中可以包含 Unicode [UNICODE]字符（参见[RDF-CONCEPTS]），以允许在 URIrefs 中使用多种语言。

URIrefs 可以是*绝对的（absolute）*或*相对的（relative）*。一个*绝对（absolute）*URIref 是对一个与 URIref 所在上下文无关的资源的引用，例如 URIref http://www.example.org/index.html。*相对（relative）*URIref 是绝对 URIref 的一种简略记法，其中略去了绝对 URIref 中的部分前缀。被略去的部分需要通过相对 URIref 所在上下文中的信息获得。例如，当相对 URIref otherpage.html 出现在资源 http://www.example.org/index.html 中时，它所对应的绝对 URIref 为 http://www.example.org/otherpage.html。 一个没有 URI 部分的 URIref[译注//即一个不含任何字符的 URIref（空 URIref）或一个只包含片段标识符的 URIref]被认为是对当前文档的引用（即它所在的文档）。于是，文档里的空 URIref[译注//即不包含任何字符的 URIref]等价于该文档自身的 URIref；而一个只包含片段标识符（fragment identifier）的 URIref 等价于它所在文档的 URIref 加上该片段标识符。例如，在 http://www.example.org/index.html 里，如果 #section2 作为一个 URIref 出现的话，它将被看作等价于绝对 URIref http://www.example.org/index.html#section2.

[RDF-CONCEPTS]中提到 RDF graphs（RDF 的抽象模型）不使用任何相对 URIrefs，即 RDF 陈述中的主体、谓词和客体（以及类型文字中的数据类型）的标识必须是与上下文无关的。然而，某个特定的具体 RDF 语法（比如 RDF/XML）可以允许在某些情况下将相对 URIrefs 作为一种绝对 URIrefs 的简略写法。RDF/XML 的确允许这样使用相对 URIrefs，前面的一些 RDF/XML 示例已经说明了这一点。更多细节请参见[RDF-SYNTAX]。

RDF 和 Web 浏览器都使用 URIrefs 来标识事物，然而它们在解释 URIrefs 的方式上略有不同。因为在 RDF 中，URIrefs *仅*被用来标识事物，而浏览器还用 URIrefs 来*获取*事物。通常，这并没有经常有实际的区别，但在某些情况下区别是显著的。一个显而易见的区别是：当一个 URIref 被用在浏览器中时，它被期望标识某个可实际获取的资源，即某个确实位于该 URI 所标识的位置的资源。然而在 RDF 中，一个 URIref 可以被用来表示像人这样的*不能*在 Web 上获取的事物。人们在用 RDF 时有时会有这样的惯例：即当一个 URIref 被用于标识某个 RDF 资源时，人们习惯于在该 URI 所标识的位置放一个包含关于这个资源的描述性信息的页面，这样可在浏览器中用该 URIref 来获取信息。这在某些情况下，这是非常有用的，尽管它在原始资源的标识和描述它的 Web 页面（详见 2.3 节）的标识的区分上造成了一些困难。然而，这一惯例在 RDF 定义中并没有被明确规定，RDF 本身不假定 URIref 标识任何可获取的事物。

另一个区别是它们对带片段标识符的 URIref 的不同处理方式。在 URLs（标识某 HTML 文档）中通常用片段标识符（fragment identifiers）来标识 URL 所标识文档中的某个特定位置。按正规的 HTML 用法，URI 引用（URI references）是用于获取资源的，因此下面两个 URIrefs：

```
http://www.example.org/index.html

http://www.example.org/index.html#Section2
```

是相关的（它们都引用了同一个 HTML 文档，而第二个标识第一个中的一个位置）。但是正如前面谈到的，在 RDF 中，URI 引用纯粹被用于*标识*资源，而不是获取资源，并且 RDF 也不假定上述两个 URIrefs 之间存在任何关系。就 RDF 来说，它们在语法上是不同的 URI 引用，因此它们可以引用着不相关的事物。这并不意味着由 HTML 定义的限制关系不存在，而只表明 RDF 不会仅根据 URI 引用的 URI 部分相同而假定它们之间存在某个关系。

就这一点再进一步地，RDF 不假定具有相同前导字符串的 URI 引用（不论是否包含片段标识符）之间存在任何关系。例如，就 RDF 来说，下面两个 URIrefs:

```
http://www.example.org/foo.html

http://www.example.org/bar.html
```

没有特定的关系，尽管它们有共同的前导字符串 http://www.example.org/。 对于 RDF 来说，它们仅仅是不同的资源，因为它们的 URIrefs 是不同的。（尽管它们可能确实是位于同一目录的两个文件，但 RDF 不假定这一点或任何其他关系存在于它们之间。）

[编辑]

附录 B：关于可扩展标记语言（Extensible Markup Language，XML）的更多信息

注意：附录 A 旨在提供一个关于 XML 的简要介绍。XML 的权威规范是[XML]，如需了解更多信息请参阅该文档。

可扩展标记语言（Extensible Markup Language）[XML]被设计为允许任何人设计他们自己的文档格式并依照该格式书写文档。像 HTML 文档（网页）一样，XML 文档包含文本。文本主要由平凡文本内容（plain text content）和以*标签（tag）*为形式的标记（markup）的构成。标记使得处理程序可以解释各个内容片断（称作*元素（elements）*）。XML 内容和（在某些特例中）标签（tags）可以包含 Unicode[UNICODE]字符，以允许直接表达多种语言的信息。在 HTML 中，允许的标签集合以及它们的解释是由 HTML 规范定义的。而 XML 允许终于用户定义他们自己的标记语言（markup languages）（标签及标签可出现的结构），以满足在他们的特定需求（第 3 节所描述的 RDF/XML 语言就是这样一种 XML 标记语言）。例如，下面这段简短的文字就是用一个基于 XML 的标记语言书写的：

```
<sentence><person webid="http://example.com/#johnsmith">I</person>

just got a new pet <animal>dog</animal>.</sentence>
```

以标签为界的元素（<sentence>，<person>等）反映了与这段文字相关的特定结构。这些标签允许理解这些特定元素以及它们的构造方式的程序正确地解释上面的文字。比如，本例中的一个元素是<animal>dog</animal>，它包含*首标签*<animal>、元素*内容*和一个相应的*尾标签*</animal>。这个 animal 元素以及 person 元素被嵌套在 sentence 元素中。如果按下面的方式来书写这个句子，也许嵌套体现得更为清楚（并更接近于某些将在后面出现的更"结构化的"XML）：

```
<sentence>

    <person webid="http://example.com/#johnsmith">I</person>

    just got a new pet

    <animal>dog</animal>.

</sentence>
```

在某些情况下，一个元素可以没有内容。可以用两种方式来书写这样的元素：不在首、尾标签括起的部分写入内容，比如<animal></animal>，或者用被称作*空元素标签（empty-element tag）*的简写形式，比如<animal/>。

在某些情况下，首标签（或空元素标签）除了包含标签名称（tag name）以外，也可以包含限定信息（以*属性（attributes）*的形式出现）。例如，<person>元素的首标签包含属性 webid="http://example.com/#johnsmith"（可能是标识被引用的那个人）。一个属性（attribute）包含名称、等号和值（用引号括起）。

这一具体的标记语言使用单词“sentence”、“person”和“animal”作为标签名称，以试图传达元素的某些含义；它们*将*传达含义给懂英语的读者，或一个专门解释该词汇表的程序。然而，这里没有内嵌的含义。例如，对于不懂英语的读者或一个不理解这些标记的程序而言，元素<person>也许没有任何含义。以下面这段文字为例：

```
<dfgre><reghh bjhbw="http://example.com/#johnsmith">I</reghh>

just got a new pet <yudis>dog</yudis>.</dfgre>
```

对一个机器来说，这段文字和前面的例子有着完全相同的结构。然而，对于懂英语的读者来，他不会明白这段文字表达了什么，因为这里的标签不是英文单词。此外，别人也许已在他们的标记语言中使用了相同的单词，而具有完全不同的含义。例如，在另一个标记语言中“sentence”可能引用一个被定罪的罪犯必须在服役机构服役的时间。因此，必须提供附加的机制来帮助保持 XML 词汇的条理性。

为避免混淆，有必要对元素进行唯一标识。在 XML 中，这是通过使用 XML 命名空间（Namespaces）[XML-NS]实现的。一个*命名空间（namespace）*即一种标识部分 Web（空间）的方式，它作为一个特定集合的名称的限定符（qualifier）。为一个 XML 标记语言创建命名空间是通过为它创建 URI 来实现的。通过使用命名空间的 URI 来限定标签名称，任何人都可以创建他们自己的标签，并正确地区分于别人创建的具有相同拼写的标签。一个时常被遵循的惯例是：创建一个网页来描述标记语言（以及其中标签的含义）并使用该网页的 URL 作为命名空间的 URI。然而，这只是一个惯例，XML 和 RDF 都没有假定一个命名空间 URI 标识的是一个可获取的 Web 资源。下面的例子展示了 XML 命名空间的使用。

```
<user:sentence xmlns:user="http://example.com/xml/documents/">

   <user:person user:webid="http://example.com/#johnsmith">I</user:person>

just got a new pet <user:animal>dog</user:animal>.

</user:sentence>
```

本例中，属性（attribute）xmlns:user="http://example.com/xml/documents/"为上述 XML 片段声明了一个命名空间。它将*前缀（prefix）*user 映射到命名空间 URI http://example.com/xml/documents/ 上。这样，XML 内容可以使用*限定名称*（简称 *QNames*）作为标签，比如 user:person。QName 以标识命名空间的前缀打头，后面紧跟一个冒号，然后是一个用于标签或属性名称的*内部名（local name）*。通过使用命名空间 URIs 来区分名称的分组，并用命名空间的 URIs 来限定标签（如本例中这样），就不必为标签名称冲突而担心了。两个拼写相同的标签仅当它们的命名空间 URIs 也相同时才被认为是相同的标签。

每个 XML 文档必须是良构的（well-formed）。这意味着 XML 文档必须满足许多语法条件，例如，每个首标签必须有一个与之匹配的尾标签，并且元素之间的嵌套必须是正确的（元素之间不能重叠）。[XML]定义了完整的良构条件集合。

另外，一个 XML 文档可以自由选择是否包含一个 XML 文档类型声明（document type declaration）。文档类型声明对文档的结构定义了附加的限制，并支持在文档中使用预定义的文本单元。文档类型声明（通过 DOCTYPE 引入 XML 文档)包含或指向定义了文档语法的声明。这个语法被称为文档类

型定义（document type definition，简称 DTD）。DTD 中的声明指定了哪些 XML 元素（elements）和属性（attributes）可以在符合该 DTD 的 XML 文档中出现、这些元素和属性的关系（例如哪些元素可以嵌套在另一些元素中，哪些属性可以在另一些元素中出现），以及元素或属性是必须的还是可选的。文档类型声明可以指向文档外部的一组声明（称作外部子集，它们可被用来在多个文档间共享一些公用的声明），也可以包含直接在文档中出现的声明（称做内部子集），或者既包含内部子集也包含外部子集。一个文档的完整 DTD 是它的内部子集与外部子集的总和。例 47 是一个带有文档类型声明的 XML 文档的简单例子：

例 47：一个带有文档类型声明的 XML 文档

```
<?xml version="1.0"?>

<!DOCTYPE greeting SYSTEM "http://www.example.org/dtds/hello.dtd">

<greeting>Hello, world!</greeting>
```

本例中，文档仅有外部 DTD 子集，系统标识符（system identifier）[译注//系统标识符一词有其历史渊源，将它理解为一种标识符就可以了，不要对其中的"系统"二字产生误解。]为 http://www.example.org/dtds/hello.dtd 给出了它的位置（一个 URIref）。

如果一个 XML 文档有与之关联的文档类型声明，并且符合该文档类型声明中的限制，则该 XML 文档是有效的（valid）。

一个 RDF/XML 文档仅需是良构的（well-formed）XML；它不用被验证是否符合某个 XML DTD 或 XML 模式（XML Schema），并且[RDF-SYNTAX]也没有指定一个可用于验证 RDF/XML 的规范性 DTD（但[RDF-SYNTAX]在附录中为 RDF/XML 给出了一个非规范性的模式）。关于 XML DTD 语法的更详尽的讨论超出了本文档的范围。欲了解更多关于 XML DTDs 以及 XML 验证的信息，请参见[XML]和众多关于 XML 的书籍。

然而，有一个 XML 文档类型声明的使用*是*与 RDF/XML 相关的，并且是用在定义 XML *实体*中。实际上，XML 实体定义是将一个名称与一个字符串相关联。当实体名称在文档中的别处[译注//指除了它被定义的地方以外]出现时，XML 处理器（XML processors）将用相应的字符串来替换该实体名称。这为长字符串（比如 URIrefs）的书写提供了一种简略方式，并有助于增强 XML 文档的可读性。使用只包含 XML 实体声明的文档类型声明是合法的，并且有时是有用的，即使在文档不用被验证时（比如在 RDF/XML 中）。

在 RDF/XML 文档中，实体通常在文档内部声明，即只使用内部 DTD 子集（这样做的一个原因是 RDF/XML 不用被验证，而非验证的 XML 处理器不被要求处理外部 DTD 子集）。例如，在 RDF/XML 文档的开始部分提供如例 48 所示的文档类型声明将允许在文档中分别用&rdf;、&rdfs;和&xsd;作为命名空间 rdf、rdfs 和 xsd 所对应 URIrefs 的简写。

例 48：一些 XML 实体声明

```
<?xml version='1.0'?>


<!DOCTYPE rdf:RDF [

    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">

    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">

]>
```

```
<rdf:RDF

    xmlns:rdf = "&rdf;"

    xmlns:rdfs = "&rdfs;"

    xmlns:xsd = "&xsd;">


...RDF statements...


</rdf:RDF>
```

[编辑]

附录 C：改动

本文档仅对建议推荐标准版本（Proposed Recommendation version）仅作了微小的编辑与排字上的改动。关于建议推荐标准版本（Proposed Recommendation version）对更早版本所做的改动请参见它的改动记录.