# 中文知识组织系统形式化语义描述

# 标准体系

# （三）

## 中文领域本体的 OWL 描述规范

http://nkos.lib.szu.edu.cn/2013/11/owl-guide

深圳大学图书馆 **NKOS** 研究室

**2013** 年 **11** 月

# 前言

“中文知识组织系统形式化语义描述标准体系研究”得到国家社科基金项目（编号：12BTQ045）和广东省哲学社会科学“十一五”规划项目（编号: GD10CTS02）资助。其总体目标是基于国内知识组织系统的特点和发展需要，参考国际上网络知识组织系统的现有成果和发展趋势，提出中文知识组织系统的形式化语义描述标准体系，为国内相关团体和个人构建各类网络知识组织系统提供一系列标准规范选择。进一步，为该标准体系提供配套的技术服务，推动国内知识组织系统的网络化发展和应用。

“中文知识组织系统形式化语义描述标准体系”由以下三份标准文本构成：

- 通用 CNKOS 语义描述规范
- 高受控词表的 OntoThesaurus 描述规范
- 中文领域本体的 OWL 描述规范

“中文领域本体的 OWL 描述规范”是在全面梳理 W3C 推荐标准 OWL1 和 OWL2 的标准文本的基础上，针对国内中文领域本体研究中所涉及的描述需求，编制的一套易于入门和操作的参考性描述规范，由一份“中文领域本体的 OWL 应用指南”及本课题组翻译的 5 份 OWL1 和 OWL2 标准文本中译文组成，方便中文领域本体的建模者参考和使用 OWL 的标准定义。这 5 份中译文文本同时发布在 W3C 官方网站 W3C Translations 上，供全球需要者参考使用。

“中文领域本体的 OWL 描述规范”由深圳大学图书馆曾新红主持研制和起草。OWL 标准文本的翻译参与者见各标准文本中译文首页。

受研制者的学识限制，本规范中难免会有错漏和不完善之处，敬请批评指正。

# 目录

# 1 中文领域本体的 OWL 应用指南

细粒度的中文领域本体建议采用 OWL 进行描述。网络本体语言 OWL（Web Ontology Language）是实现语义 Web 的核心语言工具。它为网络本体的构建提供了丰富的建模原语。

OWL（OWL2 发布之后改称为 OWL1）由 W3C 的 Web 本体工作小组（Web-Ontology（WebOnt）Working Group）开发，于 2004 年 2 月 10 日发布为 W3C 推荐标准（Recommendation）。OWL2 是 OWL1 的扩展和修订版，由后续小组，即 W3C OWL 工作组（W3C OWL Working Group）开发，于 2009 年 10 月 27 日发布为 W3C 推荐标准。二者采用相同的命名空间 owl（URI: http://www.w3.org/2002/07/owl#）。

OWL1 包含 6 个标准文档，详见表 1。对于中文领域本体建模初学者而言，推荐阅读顺序为：1 Overview（从一无所知到初步入门），2 Guides（获知使用该语言的理论方法和例子），3 Reference（详细使用手册和主要参考）。4、5、6 主要面向工具开发者，可以在需要的时候查阅。Overview 和 Guides 已由“W3CHINA 开放翻译计划（OTP）”译为中文，Reference 则由深圳大学图书馆 NKOS 研究室（本课题组）译为中文，点击表 1 中相应中文标题可链接到这些中文文档。

表 1. OWL1 文档

| 序号 | 类别 | 文档 |
|---|---|---|
| 1 | 用户指南 | OWL 概述( OWL Overview )。OWL1 的快速概览，是 OWL1 的入门和初步参考资料。 |
| 2 | 用户指南 | OWL 指南(OWL Guides)。可获知使用该语言的理论方法和例子。 |
| 3 | 用户指南 | OWL 参考(OWL Reference)。是 OWL1 的详细使用手册，可查阅建模原语的相关描述、使用限制、注意事项等。其附录及参考文献提供 OWL 每个语言元素的索引及快速参考。 |
| 4 | 核心规范 | OWL 语义与抽象语法(Semantics and Abstract Syntax)。含标准语法，精确定义语义，技术细节。面向工具开发者。 |
| 5 | 补充性文档 | OWL 测试用例(Test cases)。面向工具开发者。 |
| 6 | 补充性文档 | OWL 应用案例和需求(Use Cases and Requirements)。面向工具开发者。 |

注：此表是笔者仿照 OWL2 的文档分类进行的总结。部分内容参考了以下文献：王梅. 网络本体语言（OWL）的标准体系解析. 图书情报工作, 2005(7).

OWL2 包含 13 个标准文档，详见表 2。作为 OWL1 的升级版本，W3C 建议用户直接使用 OWL2。对于初学者来说，这可能会很困难。因此，笔者建议读者结合 OWL1 文档来全面了解 OWL2。建议阅读顺序：1 Document Overview，8 Primer，9 New Features and Rational。通过以上三个文档先全面了解 OWL2 的基本内容和使用方法，然后在具体的建模过程中，主要参考以下三个文档：OWL1 的 3 Reference，OWL2 的 10 Quick Reference Guide，以及 9 New Features and Rational。

OWL2 的文档 2-7 以及 11-13 主要面向工具开发者，可以在需要的时候查阅。OWL2 的所有用户指南（1，8，9，10）均已由本课题组译为中文。点击表 2 中的相应中文标题可以链接到这些中文文档。

上文提到的所有 OWL1 和 OWL2 的文档中译文均可在 W3C Translations（http://www.w3.org/2005/11/Translations/Lists/ListLang-zh-hans.html）上获得。

表 2. OWL2 文档

| 序号 | 类别 | 文档 |
|---|---|---|
| 1 | 用户指南 | 文档概述( Document Overview )。OWL2 的快速概览，它描述了 OWL2 与 OWL1 的关系。该文档是 OWL2 的入门和初步参考资料。 |
| 2 | 核心规范 | 结构化规范与函数式语法 (Structural Specification and Functional-Style Syntax)从结构与函数式语法的角度定义了 OWL2 本体的结构，并从 OWL2 本体全局限制的角度定义了 OWL2 DL 本体。 |
| 3 | 核心规范 | 映射到 RDF 图(Mapping to RDF Graphs) 定义了 OWL2 结构到 RDF 图的映射，因而也就定义了在语义网中交换 OWL2 本体的主要方法。 |
| 4 | 核心规范 | 直接语义(Direct Semantics)从模型－理论语义的角度定义了 OWL2 的含义。 |
| 5 | 核心规范 | 基于 RDF 的语义 (RDF-Based Semantics)通过 RDF 语义(RDF Semantics)的一个扩展定义了 OWL2 本体的含义。 |
| 6 | 核心规范 | 一致性准则(Conformance) 提出了对 OWL2 工具的要求，也提供了帮助判定一致性的测试用例集。 |
| 7 | 规范 | 配置语言(Profiles) 定义了 OWL2 的三个子语言，它们在特定应用场景下可以发挥重要的作用。 |
| 8 | 用户指南 | OWL2 入门(OWL 2 Primer) 对 OWL2 作了较通俗的介绍，其中包括对来自于其他学科交叉知识的介绍。 |
| 9 | 用户指南 | OWL2 新特性与原理(OWL 2 New Features and Rationale) 对 OWL2 的主要新特性作了概述，并鼓励将它们应用到语言中。 |
| 10 | 用户指南 | OWL2 快速参考指南(OWL 2 Quick Reference Guide）概要地说明了 OWL2 的结构，同时也指出了相对于 OWL1 的变动。 |
| 11 | 规范 | XML 序列化(XML Serialization)定义了交换 OWL2 本体的 XML 语法，适合与基于模式的编辑工具及 XQuery/XPath 这样的 XML 工具一起使用。 |
| 12 | 规范 | 曼彻斯特语法(Manchester Syntax)(WG Note) 定义了易读但形式化较弱的 OWL2 语法，用于某些 OWL2 用户界面工具，在入门(Primer)中也用到了。 |
| 13 | 规范 | 数据值域扩展：线性等价(Data Range Extension: Linear Equations) |

| | |
|---|---|
| | (WG Note) 指定了 OWL2 的可选扩展，它支持对属性值的高级约束。 |

注：此表译自 OWL2 Document Overview。

# 2　OWL 标准文本中译文

## 2.A　OWL Web 本体语言参考

本文档《OWL Web 本体语言参考》是 W3C 发布的 ***OWL Web Ontology Language Reference***（2004-02-10）的中文译本。文中若存在译法不当和错误之处，欢迎批评指正，请发邮件至：**zengxh@szu.edu.cn**，谢谢！

**翻译说明：**

- 本文档的英文版是唯一正式版本。此处的中文译本仅供学习与交流。
- 中文译本的内容是非正式的，仅代表译者的个人观点。
- 中文译本的内容会根据反馈意见随时进行修订。
- 中文译本同时通过 W3C Translations 网站发布。
- 转载本文，请注明译者和原链接。

**译者：**
曾新红（Xinhong Zeng），深圳大学图书馆 NKOS 研究室
徐　乐（Le Xu），深圳大学计算机与软件学院

**资助声明：**
本次翻译工作得到广东省哲学社会科学“十一五”规划项目（批准号：GD10CTS02）和国家社科基金项目“中文知识组织系统的形式化语义描述标准体系研究”（批准号：12BTQ045）的资助。

发布时间：　2013 年 1 月 18 日

---

## OWL Web 本体语言参考

## W3C 推荐标准　2004 年 2 月 10 日

**当前版本:**
http://www.w3.org/TR/2004/REC-owl-ref-20040210/
**最新版本:**
http://www.w3.org/TR/owl-ref/
**上一版本:**
http://www.w3.org/TR/2003/PR-owl-ref-20031215/
**编者:**
Mike Dean, BBN Technologies

Guus Schreiber, Free University Amsterdam

**贡献者:**

Sean Bechhofer, University of Manchester
Frank van Harmelen, Free University Amsterdam
Jim Hendler, University of Maryland
Ian Horrocks, University of Manchester
Deborah L. McGuinness, Stanford University
Peter F. Patel-Schneider, Bell Labs Research, Lucent Technologies
Lynn Andrea Stein, Franklin W. Olin College of Engineering

请参考本文的*勘误表*，那里会有一些规范性的修正。

也可以查看相关翻译 。



---

## 摘要

Web 本体语言 OWL 是一个语义标记语言，用于在万维网上发布和共享本体。OWL 以 RDF（资源描述框架）词汇扩展的形式开发，它源于 DAML+OIL Web 本体语言。本文档包含了对整套 OWL 语言结构的结构化的非形式化描述，目的是为想要构建 OWL 本体的 OWL 用户提供一个参考。

## 本文档的状态

本文档已经由 W3C 成员及其他兴趣组织审阅，并已被 W3C 主管（W3C Director）批准为 W3C 推荐标准（W3C Recommendation）。W3C 在制定推荐标准过程中担任的角色是使规范受到关注，并促使其被广泛应用。这将增强 Web 的功能性与互操作性。

本文档是 W3C 关于 Web 本体语言 OWL 的推荐标准的六个部分之一。它由 Web 本体工作小组开发，作为 W3C 语义 Web 行动（行动声明，小组章程）的一部分于 2004 年 2 月 10 日发布。

本文档的早期版本中所描述的关于 OWL 的设计已被广泛校阅，并已满足工作小组的技术需求。工作小组充分考虑了所有收到的意见，并做了必要的修改。本文档自从建议推荐标准版本以来的所有修改都详细列在文后的变动日志中。

欢迎通过 public-webont-comments@w3.org（历史存档）提出您的意见，也可以通过 www-rdf-logic@w3.org（历史存档）参与相关技术的一般讨论。

可以访问到有关实现的一个列表。

W3C 维护着一个与这些工作相关的专利声明的目录。

*这节描述了本文档在发布时的状态。其他文档可能替代该文档。当前 W3C 的最新出版物的目录和这个技术报告的最新版本在 W3C 技术报告索引*

*http://www.w3.org/TR/中可以找到。*

# 致谢

本文档有一部分源自 DAML+OIL (2001.3) Reference Description [*DAML+OIL*]（作为 DAML+OIL W3C Note 的一部分提交）。我们对本文档和之前文档的赞助者表示诚挚的感谢。

Jeremy Carroll, Jim Hendler, Brian McBride 和 Peter Patel-Schneider 对本文档进行了实质性的审查，并为本文档贡献了文字。Jeff Heflin 撰写了关于过时（deprecation）的章节。Jerome Euzenat 为一个枚举数据类型提供了示例。

这个文档是 Web 本体工作组作为一个整体进行广泛讨论的结果。该工作组的参与者包括：Yasser alSafadi, Jean-François Baget, James Barnette, Sean Bechhofer, Jonathan Borden, Frederik Brysse, Stephen Buswell, Jeremy Carroll, Dan Connolly, Peter Crowther, Jonathan Dale, Jos De Roo, David De Roure, Mike Dean, Larry Eshelman, Jérôme Euzenat, Tim Finin, Nicholas Gibbins, Sandro Hawke, Patrick Hayes, Jeff Heflin, Ziv Hellman, James Hendler, Bernard Horan, Masahiro Hori, Ian Horrocks, Jane Hunter, Francesco Iannuzzelli, Rüdiger Klein, Natasha Kravtsova, Ora Lassila, Massimo Marchiori, Deborah McGuinness, Enrico Motta, Leo Obrst, Mehrdad Omidvari, Martin Pike, Marwan Sabbouh, Guus Schreiber, Noboru Shimizu, Michael Sintek, Michael K. Smith, John Stanton, Lynn Andrea Stein, Herman ter Horst, David Trastour, Frank van Harmelen, Bernard Vatant, Raphael Volz, Evan Wallace, Christopher Welty, Charles White，以及 John Yanosy。

# 目录

---

# 1. 引言

## 1.1 本文档的目的

本文档给出了 OWL 的所有建模原语（modelling primitives）的系统、紧凑和资料性的描述，使用了 OWL 的 RDF/XML 交换语法。我们希望本文档能够为 OWL 语言的用户提供参考指南。

本文档是 OWL 即 Web 本体语言描述的一部分，由 W3C Web 本体工作组制定。“OWL 概述”的文档路线图部分描述了文档的每一个不同部分以及它们是如何组合在一起的。对 OWL 不熟悉的读者不妨先查阅“OWL 概述”文档[*OWL Overview*]，然后再查阅“OWL 指南”[*OWL Guide*]上关于 OWL 语言使用的更具叙述性的说明和例子。

本文档假定读者熟悉 RDF 的基本概念[*RDF Concepts*] ，具有 RDF/XML 语法[*RDF/XML Syntax*]和 RDF Schema[*RDF Vocabulary*]的应用知识。

可以在“OWL 语义和抽象语法”文档[*OWL S&AS*]中找到关于 OWL 语言结构（constructs）精确语法的规范参考。那份文档还包含一个语言结构含义的确切定义，这个定义以模型-理论（model-theoretic）语义的形式给出。诸如 OWL 本体的一致性等概念在该文档中也有讨论。

在“OWL 需求”文档[*OWL Requirements*]中描述了 OWL 语言的用例和需求。在测试文档[*OWL Test Cases*]中对 OWL 工具的测试用例（*例如*蕴含测试、一致性测试）做了详细说明。

## 1.2 OWL Full / DL / Lite

正如在 OWL 概述文档[*OWL Overview*]以及随后的 OWL 指南[*OWL Guide*]中所讨论的，我们相信 OWL 语言提供的两种特定的子集对于实现者和语言使用者是有用的。OWL Lite 专门为易于实现的开发而设计，它提供了一个功能子集帮助使用者开始使用 OWL。设计 OWL DL（DL 即“Description Logic”）是为了支持现有的描述逻辑业务，OWL DL 提供了一个对推理系统有着理想计算性能的语言子集。完整的 OWL 语言（称为 OWL Full 以区别于其子集）放宽了一些对 OWL DL 的约束，以使得许多数据库和知识表达系统可以使用那些对它们可能有用的特性，但是这违反了描述逻辑推理器的约束条件。

> 注意：除非专门以 OWL DL 或 Lite 构造，否则 RDF 文档通常都采用 OWL Full。

OWL Full 和 OWL DL 支持相同的 OWL 语言结构集合。它们的差别在于对那些特性和 RDF 特性的使用限制。OWL Full 允许 OWL 和 RDF Schema 自由混合，它和 RDF Schema 一样，不强制要求类、属性、个体和数据值严格分离。OWL DL 限制了与 RDF 的混合，要求类、属性、个体和数据值不相交。OWL DL 子语言之所以存在，是因为工具制造者已经开发出了强大的推理系统，支持 OWL DL 所需限制条件的本体约束。OWL Full 和 OWL DL 之间差异的形式定义，读者可以参考语义和抽象语法文档[*OWL S&AS*]。Sec. 8.2 "OWL DL"对 OWL Full 和 OWL DL 之间的差异做了总结。

OWL Lite 是 OWL DL 的子语言，仅支持 OWL 语言结构的一个子集。OWL Lite 特别针对这样一类工具制造者，他们想要支持 OWL，但想从一个相对简单的语言特性的基本集合开始。OWL Lite 和 OWL DL 遵守相同的语义限制，允许推理引擎保证某些所需的性能。Sec. 8.3 给出了 OWL Lite 中被允许的语言结构的总结。读者可以参考语义和抽象语法文档[*OWL S&AS*]获得 OWL Lite 支持的 OWL 语言结构子集的更形式化的描述。

> 注意：升级到 OWL 的 RDF 用户应注意 OWL Lite 不是 RDF Schema 的一个简单扩展。OWL Lite 是 OWL DL 的精简版本，它对 RDF 词汇的使用施加了限制（*例如*，类和属性不能相交等）。OWL Full 是为与 RDF 的最大兼容性而设计的，因此，RDF 用户可以很自然地从 OWL Full 着手。当 RDF 用户在 OWL DL 和 OWL Lite 二者中抉择时，应该考虑 OWL DL/Lite 的优势（*例如*，对推理的支持）是否胜过对 OWL 和 RDF 结构使用的 DL/Lite 限制。
>
> 注意：本文档中 OWL Lite 被定义为对 OWL DL 的许多额外限制。这意味着，除非明确说明，否则 OWL DL 结构也是 OWL Lite 的一部分。Sec. 8.3 总结了这些附加的 OWL Lite 限制。

## 1.3 OWL 语法

OWL 本体是一个 RDF 图[*RDF 概念*]，而 RDF 图又是一个 RDF 三元组集合。正如任何 RDF 图，OWL 本体图可以写成多种不同的句法格式（如同“RDF/XML 语法规范”（修订版）[*RDF/XML 语法*]中的描述）。当前文档使用了一些特定的 RDF/XML 句法格式表示三元组（指南文档中也是如此）。然而 OWL 本体的意义完全由 RDF 图确定。因此，我们可以使用其他 RDF/XML 句法格式，只要所代表的 RDF 三元组相同。那么，其他句法格式就可以携带和本文档中所使用的句法格式相同的含义。

作为一个替代句法格式引至相同 RDF 三元组的简单例子，我们考虑以下的 RDF/XML 语法：

```
<owl:Class rdf:ID="Continent"/>
```

下面的 RDF/XML 语法：

```
<rdf:Description rdf:about="#Continent">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
</rdf:Description>
```

为相同的 RDF 三元组编码，因此传达相同的含义。

## 1.4 OWL 和 RDF 语义

OWL 是 RDF 的一个词汇扩展[*RDF 语义*]，因此任何 RDF 图都形成一个 OWL Full 本体。此外，OWL 所赋予 RDF 图的含义包括了 RDF 所赋予该图的含义。因此，OWL Full 本体可以包含 RDF 的任意内容，这和 RDF 的处理方式是一致的。OWL 赋予某些 RDF 三元组一个附加含义。“OWL 语义和抽象语法”文档[*OWL S&AS*]明确列举了哪些三元组被赋予了特定的含义以及这些含义是什么。

> 注意：正如之前讨论过的，OWL DL 和 OWL Lite 扩展了 RDF 词汇，但也对词汇的使用加以限制。因此，除非专门以 OWL DL 或 Lite 构造，否则 RDF 文档通常都采用 OWL Full。

## 1.5 例子的说明

为了便于阅读，假设本文档中的例子对 XML 实体&rdf;、&rdfs;、&owl;以及&xsd;（XML Schema 数据类型）的定义和附录 B 中的相同。关于相应的命名空间 `rdf:`、`rdfs:`、`owl:`和 `xsd:`同样的假设也成立。

本文档中的例子是为了说明 OWL 语言结构的使用。它们不形成一个一致的本体。读者可以参考“指南”文档获取一个扩展的例子[*OWL Guide*]。

## 1.6 数据聚合与隐私

OWL 在表达出现在多个文档中个体的本体信息的能力方面，以一种有原则的方式支持连接来自异源的数据。下层语义为这些数据提供推理支持，这可能产生意外的结果。特别地，采用 owl:sameAs 表达等价的能力，可以被用来表达表面上不同的个体实际上是相同的。同样地，owl:InverseFunctionalProperty 可被用来将个体连接在一起。例如，如果一个属性，比如“`SocialSecurityNumber`”，是一个 `owl:InverseFunctionalProperty`，那么两个独立的个体如果具有相同的 `SocialSecurityNumber` 属性值，则可被推理出是相同的个体。当个体以这种方式被确定为相同时，来自异源的关于这些个体的信息可以被合并。这种*聚合*（aggretion）可用来得出未*直接*在单源表达的事实。

语义网的连接来自多源信息的能力是一个理想的、强大的特性，它可以用在许多应用中。但是合并来自多源数据的能力，加上 OWL 的推理能力，确实存在被滥用的可能。在出台了数据保护法的国家，特别是欧盟，如果没有一个正当的法律上的原因，创建或者处理这样的连接信息甚至可能是非法的。因此，在使用 OWL 处理可能会连接到其他数据源或者本体的任何个人数据时需要非常谨慎。具体的安全方案超出了工作组的工作范畴。目前正在进行的各种不同的安全和偏好方案有望解决这些问题，比如 SAML 和 P3P。

## 1.7 本文档的附录

本文档有一组包含附加信息的附录。

本文档中附带在语言结构定义上的链接提供了对“OWL 语义和抽象语法”的访问[*OWL S&AS*]。对于每个语言结构对应于指南和 S&AS 文档的相应部分，附录 A 包含了一组系统的链接。

附录 B 包含一个适用于 OWL 语言结构的 RDF schema。这个 schema 提供了关于 OWL 词汇的信息，这可能对本体建造者和工具开发者是一个有用的参考点。该 schema 提供的关于 OWL 类和属性的限制条件是资料性的，并不完整。此外，该 schema 不区分 OWL Full、OWL DL 和 OWL Lite。按照惯例，类的第一个字符大写，属性的第一个字符小写。因此，`owl:Ontology` 表示类，`owl:imports` 表示属性。

> 注意：RDF Schema 文件并不被希望明确导入到本体（*即*，`owl:imports`）。schema 具有资料性状态，其目的是以 RDF/XML 语法的方式提供类和属性的使用。导入此 schema 的人应该期望产生的本体是一个 OWL Full 本体。

附录 C 依据内置的 OWL 类和属性给出了 OWL 词汇表的表格概述（同时给出了属性的值域和定义域）。

对于熟悉 DAML+OIL 的读者，附录 D 列举了许多 DAML+OIL 和 OWL 之间的变化。

最后，附录 E 提供了一套实用指南来具体说明 RDF 格式的 OWL DL 本体。

# 2. OWL 文档

OWL 格式的信息被收集进本体，这样可以以文档的形式储存到万维网上。OWL 的一个方面，本体的导入，依赖于 OWL 本体能够存储在网上的能力。

## 2.1 内容

一个 OWL 文档由可选的本体头（通常至多一个）加上任意数量的类公理、属性公理和关于个体的事实组成。请注意，“公理（axiom）”是在 S&AS 文档里使用的正式术语。而在指南和概述这两个文档里，这些公理略显非正式地称为“定义（definitions）”。

> 注意：OWL 不会强制限定 OWL 组件的顺序。人们在写本体时可能用到某种顺序，比如把本体头部放在开头，但不同的顺序不会对本体的意义造成影响。工具不应该假设任何的顺序。

和大多数 RDF 文档一样，OWL 代码应该是一个 `rdf:RDF` 元素的子元素。这个封闭的元素通常拥有 XML 命名空间和基（base）声明。并且，一个 OWL 本体文档通常以若干实体声明开始。有关这方面信息的典型例子，请参阅指南文档里讨论的酒和食物的例子[*OWL 指南*]。

## 2.2 OWL 内置词汇

OWL 的内置词汇都来自于 OWL 命名空间

`http://www.w3.org/2002/07/owl#`,

按照惯例与命名空间名称 owl 相关联。除了内置词汇，我们不建议本体使用来自此命名空间的名称。如果来自此命名空间的其他名称被使用，那么 OWL 工具制造者应该随时标记警告，但是应该以另外的方式继续正常使用。

## 2.3 MIME 类型

Web 本体工作组并没有要求 OWL 文档有一个单独的 MIME 类型。我们建议使用 RDF 核心工作组要求的 MIME 类型，即 `application/rdf+xml`[*RDF 概念*]，或者 XML MIME 类型 `application/xml` 代替。

至于文件扩展，我们建议使用 `.rdf` 或者 `.owl`。

# 3. 类

类提供了一种根据相似的特征将资源分类的抽象机制。如同 RDF 类，每一个 OWL 类都与一组被称为*类外延*（class extension）的个体相关联。类外延中的个体被称作类的*实例*（instance）。一个类有与类外延有关但并不等同的内涵（底层概念）。因此，有相同类外延的两个类仍然可以是不同的类。

在本文档里出现的用词如“一类个体…”我们应该解读为“一个有着一个包含个体的类外延的类…”。

> 注意：在 OWL Lite 和 OWL DL 里，一个个体不能同时是一个类：类和个体的定义域不相交（属性和数据值也一样）。而 OWL Full 则对 RDF Schema 没有限制：一个类同时可以充当另外一个（元）类的实例。

OWL 类通过“类运算式”（class descriptions）来描述，这一描述可以结合到“类公理”。我们首先描述类运算式，随后描述类公理。

## 3.1 类运算式

本文档（也包括“语义和抽象语法”文档）里使用的类运算式是用来描述类公理（“概述”和“指南”文档里非正式地称为类定义）的基本构建块的术语。一个类运算式这样描述一个 OWL 类：要么通过一个类名，要么通过指定一个未命名的匿名类的类外延。

OWL 对类运算式做了六种类型区分：

1. 类标识符（一个 URI 引用）
2. 详尽的共同构成一个类实例的个体枚举
3. 属性约束
4. 两个或多个类运算式的交集
5. 两个或多个类运算式的并集

6. 一个类运算式的补集

第一种类型在描述类时使用了*类名*（句法上表示为一个 URI 引用），从这个意义上来讲它是特殊的。其他五种类运算式类型通过*限制类外延*来描述匿名类。

2-6 种类运算式类型分别描述了一种类：完全包含列举个体的类（第 2 种类型），满足一个特定属性约束的所有个体的类（第 3 种类型），或者是满足类运算式的布尔组合的类（第 4、第 5 和第 6 种类型）。交集、并集和补集可分别被看作逻辑运算符 AND、OR 和 NOT。后面四种类运算式类型允许类运算式的嵌套，因此，理论上可以形成任意复杂的类运算式。在实践中，通常会限制嵌套的级别。

类型 1 类运算式在句法上被描述成 `owl:Class` 的一个具名实例，`rdfs:Class` 的一个子类：

```
<owl:Class rdf:ID="Human"/>
```

这声明了一个三元组“`ex:Human rdf:type owl:Class .`”，`ex:`代表相关本体的命名空间。

> 注意：在 OWL Lite 和 OWL DL 里，所有的类运算式都要用到 `owl:Class`（或者 `owl:Restriction`，参见下文）。
>
> 注意：`owl:Class` 被定义为 `rdfs:Class` 的一个子类。拥有一个单独的 OWL 类结构这一基本原理存在于对 OWL DL 以及 OWL Lite 的限制条件中，这意味着不是所有的 RDFS 类都是合法的 OWL DL 类。在 OWL Full 中不存在这些限制条件，因此 `owl:Class` 和 `rdfs:Class` 是等价的。

另外五种类运算式形式由 RDF 三元组集合组成，三元组中的空节点代表被描述的类。此空节点有一个 `rdf:type` 属性，其属性值是 `owl:Class`。

> 注意：如果有人为枚举、交集、并集或者补集类型的类运算式提供了一个 RDF 标识，那么我们就不认为它是类运算式，而是一种支持完整类的特殊类公理。详情参见 Sec. 3.2.3。
>
> 注意：为了提高文档的可读性，在文档中使用简约表达“类运算式”来表示“被类运算式所描述的类”。严格来讲，在 2-6 种类运算式类型的情况下是有区别的：类由相应的空节点表示；类运算式由有着空节点作为主体的三元组表示。

有两个 OWL 类标识是预定义的，即类 owl:Thing 和 owl:Nothing。owl:Thing 的类外延是所有个体的集合。owl:Nothing 的类外延是空集合。因此，每一个 OWL 类都是 owl:Thing 的子类，而 owl:Nothing 则是每一个类的子类（子类关系的含义，请参阅有关 `rdfs:subClassOf` 的章节）。

### 3.1.1 枚举

“枚举”类运算式的定义中有 owl:oneOf 属性。这个内置的 OWL 属性值必须是个体的列表，而这些个体是该类的实例。这就使得一个类能够通过详尽地列举它的实例来描述。有 `owl:oneof` 属性的类运算式的类外延正好包含所列举的个体，不多不少。个体列表通常借助于 RDF 结构 `rdf:parseType="Collection"`来表达，这一结构为书写列表元素集合提供了便利的简约表达。例如，下面的 RDF/XML 语法定义了一个包含所有大陆的类：

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Eurasia"/>
    <owl:Thing rdf:about="#Africa"/>
```

```
    <owl:Thing rdf:about="#NorthAmerica"/>
    <owl:Thing rdf:about="#SouthAmerica"/>
    <owl:Thing rdf:about="#Australia"/>
    <owl:Thing rdf:about="#Antarctica"/>
  </owl:oneOf>
</owl:Class>
```

RDF/XML 语法`<owl:Thing rdf:about="..."/>`指的是一些个体（记住：按照定义所有的个体都是 `owl:Thing` 的实例）。

在数据类型章节，我们将会看到 `owl:oneof` 结构的另外一种用法，即定义一个数据值的枚举。

注意：枚举不是 OWL Lite 的一部分。

### 3.1.2 属性限制

属性限制是一种特殊的类运算式。它描述了一个匿名类，即满足这个限制的所有个体的类。OWL 对两种属性限制作了区分：值约束和基数约束。

值约束对属性的值域施加了限制，*当应用到这一特殊的类运算式的时候*。例如，我们可能想要参考那些 `adjacentTo` 属性值是一些 `Region` 的个体，然后在类公理中使用这一限制，甚至可能是在一个 `Region` 类公理本身之内使用。注意到这不同于 `rdfs:range`，后者适用于使用此属性的所有情况。

基数限制对一个属性能够取属性值的数量提出了限制，*在这个特殊类运算式的上下文中*。例如，我们可能会说一个足球队的属性 `hasPlayer` 有 11 个值。对于篮球队来说，相同的属性只有 5 个值。

OWL 还支持有限的结构集用以定义全局属性基数，即 owl:FunctionalProperty 和 owl:InverseFunctionalProperty（参见有关属性的章节）。

属性限制的一般形式如下：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="(some property)" />
  (precisely one value or cardinality constraint, see below)
</owl:Restriction>
```

类 owl:Restriction 定义为 owl:Class 的子类。一个限制类应该有且只有一个三元组链接限制到特定的属性上，使用属性 owl:onProperty。限制类也应该只有一个三元组表示对所考虑属性的值约束或（c.q.）基数约束，*例如*，此属性的基数正好是 1。

属性限制既可以应用于数据类型属性（值是一个数据文本的属性），也可以应用于对象属性（值是一个个体的属性）。关于这一区别的更多信息，请参阅有关属性章节。

#### *3.1.2.1 值约束*

##### 3.1.2.1.1 OWL:ALLVALUESFROM

值约束 owl:allValuesFrom 是内置的 OWL 属性，把一个限制类链接到类运算式或者数据值域。一个含有 `owl:allValuesFrom` 约束的限制用来描述一个类，其所有个体所考虑的所有属性值，要么是类运算式的类外延的成员，要么是指定数据值域内的数据值。换句话说，它定义个体 x 的一个类，满足如果（x,y）是 P

（相关属性）的一个实例，那么 y 应该是类运算式的一个实例或者数据值域中的一个值。
一个简单的例子：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:allValuesFrom rdf:resource="#Human"  />
</owl:Restriction>
```

这个例子描述了一个匿名 OWL 类，其所有个体的 hasParent 属性只有类 Human 的值。注意这一类运算式并非声明这个属性总是有这个类的值；只是对于属于这个匿名限制类的类外延的个体才确实如此。

注意：在 OWL Lite 中，允许作为 owl:allValuesFrom 的客体的唯一类运算式类型是一个类名。

owl:allValuesFrom 约束类似于谓词逻辑的全称（所有）量词——对于被描述的类的每个实例，每一个属性 P 的值都必须满足约束。还要注意 owl:allValuesFrom 与全称量词的对应，意味着针对属性 P 的 owl:allValuesFrom 约束显而易见（trivially）满足根本没有属性 P 的值的个体。要知道为什么是这样，观察 owl:allValuesFrom 约束要求所有 P 的值应该是类型 T，如果不存在这样的值，那么这一约束显然正确(trivially true)。

#### 3.1.2.1.2 OWL:SOMEVALUESFROM

值约束 owl:someValuesFrom 是内置 OWL 属性，它把一个限制类链接到类运算式或者数据值域。包含 owl:someValuesFrom 约束的限制描述了一个类，对于其所有个体，至少一个所考虑属性的值是该类运算式的一个实例或者是数据值域内的一个数据值。换句话说，它定义了个体 x 的一个类，满足至少有一个 y（类运算式的一个实例或者数据值域内的值）构成的实例（x,y）是 P 的一个实例。这并不排除当 y'不属于类运算式或者数据值域，（x,y'）也可以是 P 的实例这一情况。

下面的例子定义了一个类，其个体有至少一个父母是医生：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:someValuesFrom rdf:resource="#Physician" />
</owl:Restriction>
```

owl:someValuesFrom 约束类似于谓词逻辑的存在量词——对于所定义的类的每个实例，至少存在一个 P 的值满足约束。

注意：在 OWL Lite 中，允许作为 owl:someValuesFrom 的客体的唯一类运算式类型是一个类名。

#### 3.1.2.1.3 OWL:HASVALUE

owl:hasValue 值约束是内置 OWL 属性，它把一个限制类链接到值 V，V 可以是个体或者数据值。一个包含 owl:hasValue 约束的限制描述了一个类，对于其所有个体的所考虑属性，至少有一个值 *语义*上等价于 V（也可能有其他值）。

注意：对于数据类型，“语义上等价”的意思是字面值的词汇表达映射到相同的值。对于个体，“语义上等价”的意思是两者要么有相同的 URI 引用要么定义成相同的个体（参见 owl:sameAs）。

注意：OWL Lite 不包含值约束 owl:hasValue。

下面的例子描述了有个体的父母名为 Clinton 的类：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:hasValue rdf:resource="#Clinton" />
</owl:Restriction>
```

#### *3.1.2.2 基数约束*

如同 RDF，在 OWL 中，假定一个类的任何实例对于特定属性都可能有一个任意数目（0 或多个）的值。基数约束的使用可以达到以下目的：使一个属性成为必备的（至少有一个），使一个属性只能有特定数量的值，或者使得一个属性不出现。OWL 提供了三种结构来在类上下文内本地限制属性的基数。

注意：OWL Lite 包括全部三种类型的基数约束的使用，但仅仅是当值是"0"或"1"时。

##### 3.1.2.2.1 OWL:MAXCARDINALITY

基数约束 owl:maxCardinality 是内置的 OWL 属性，它把一个限制类链接到一个属于 XML Schema 数据类型 nonNegativeInteger 值空间的数据值。包含 owl:maxCardinality 约束的限制描述了一个类，其所有个体对于所考虑的属性至多有 N 个语义上不同的值（个体或数据值），在这里 N 是基数约束的值。在句法上，基数约束被表示成有相应 rdf:datatype 属性的 RDF 属性元素。

下面的例子描述了一个至多有两个父母的个体的类：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality
rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>
```

我们将在 Sec. 6 中更详细地讨论 RDF 数据类型。

##### 3.1.2.2.2 OWL:MINCARDINALITY

基数约束 owl:minCardinality 是内置的 OWL 属性，它把一个限制类链接到一个属于 XML Schema 数据类型 nonNegativeInteger 值空间的数据值。包含 owl:minCardinality 约束的限制描述了一个类，其所有个体对于所考虑的属性至少有 N 个语义上不同的值（个体或数据值），在这里 N 是基数约束的值。在句法上，基数约束被表示为有相应 rdf:datatype 属性的 RDF 属性元素。

下面的例子描述了一个至少有两个父母的个体的类：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
</owl:Restriction>
```

需要注意一个或多个 owl:minCardinality 意味着类的所有实例必须拥有一个此属性的值。

##### 3.1.2.2.3 OWL:CARDINALITY

基数约束 owl:cardinality 是内置的 OWL 属性，它把一个限制类链接到一个属于 XML Schema 数据类型 `nonNegativeInteger` 范围的数据值。包含 `owl:cardinality` 约束的限制描述了一个类，其所有个体对于所考虑的属性恰好有 N 个语义上不同的值（个体或数据值），在这里 N 是基数约束的值。在句法上，基数约束被表示成有相应 `rdf:datatype` 属性的 RDF 属性元素。

下面的例子描述了一个恰好有两个父母的个体的类：

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:cardinality
rdf:datatype="&xsd;nonNegativeInteger">2</owl:cardinality>
</owl:Restriction>
```

### 3.1.3 交集，并集和补集

本节的三种类运算式代表了在描述逻辑中使用的更高级的类构造器。它们可以被看作是对于类的 AND，OR 和 NOT 运算符。我们给出了三种运算符的标准集合运算符名称：交集、并集和补集。这些语言结构有着共同的特征，它们包含一个（补集）或者多个（并集、交集）嵌套的类运算式。

#### *3.1.3.1 owl:intersectionOf*

属性 owl:intersectionOf 链接一个类到类运算式列表。`owl:intersectionOf` 声明描述了一个类，其类外延正好包含这样的个体，它们是列表中的所有类运算式的类外延都包含的成员。

例如这样一个例子：

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Turandot" />
        <owl:Thing rdf:about="#Tosca" />
      </owl:oneOf>
    </owl:Class>
  </owl:intersectionOf>
</owl:Class>
```

在上面的例子中，`owl:intersectionOf` 的值是两个类运算式的列表，即两个枚举，它们都描述了有两个个体的类。最终交集是一个有一个个体 Tosca 的类，Tosca 是唯一的两个枚举中都有的个体。

> 注意：假定这三个个体各不相同。事实上在 OWL 中并没有定义成这样。由于 OWL 并没有“唯一名称”这一假定，因此不同的 URI 引用可以指向相同的个体。在 Sec. 5 中，OWL 语言结构对个体的等价性和差异性做了约束。
>
> 注意：在这个例子中，我们利用枚举来弄清楚语言结构是什么意思。更多典型的例子，请参阅 OWL 指南[*OWL Guide*]。

注意：OWL Lite 使用 owl:intersectionOf 是受限的。本文档后面对这点做了讨论，参阅 Sec. 3.2.3。

owl:intersectionOf 可以被看做类似于逻辑与（logical conjunction）。

#### *3.1.3.2 owl:unionOf*

属性 owl:unionOf 链接一个类到类运算式列表。owl:unionOf 声明描述了一个匿名类，其类外延包含的个体至少存在于类运算式列表的其中一个类外延中。

例如这样一个例子：

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Tosca" />
        <owl:Thing rdf:about="#Salome" />
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Turandot" />
        <owl:Thing rdf:about="#Tosca" />
      </owl:oneOf>
    </owl:Class>
  </owl:unionOf>
</owl:Class>
```

上面的类运算式描述了一个类，它的类外延包含三个个体，即 Tosca，Salome 和 Turandot（假设它们都不相同）。

注意：owl:unionOf 不是 OWL Lite 的一部分。

owl:unionOf 类似于逻辑或（logical disjunction）。

#### *3.1.3.3 owl:complementOf*

属性 owl:complementOf 链接一个类到正好一个类运算式上。owl:unionOf 声明描述了一个类，其类外延恰好包含那些*不*属于声明客体类运算式的类外延的个体。owl:complementOf 类似于逻辑非：类外延包含那些不是补集类（complement class）的类外延成员的个体。

下面是关于补集用法的例子，“not meat”表达式可以写成：

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Meat"/>
  </owl:complementOf>
</owl:Class>
```

上面类运算式的外延包含了不属于类 Meat 的所有个体。

注意：owl:complementOf 不是 OWL Lite 的一部分。

## 3.2 类公理

类运算式通过类公理来形成构建块以定义类。类公理的最简单形式是类型 1 的类运算式，它仅仅使用具有一个类标识符的 `owl:Class` 声明了类的存在。

例如，下面的类公理声明了 URI 引用`#Human` 是一个 OWL 类的名称：

```
<owl:Class rdf:ID="Human"/>
```

这是正确的 OWL，但是它没有告诉我们太多的关于类 `Human` 的信息。典型地，类公理包含用来声明一个类的必要和/或充分特征的附加成分。OWL 包含了三种语言结构将类运算式结合成类公理：

- `rdfs:subClassOf` 允许声明一个类运算式的类外延是另一类运算式的类外延的子集。
- `owl:equivalentClass` 允许声明一个类运算式有着和另外一个类运算式完全相同的类外延。
- `owl:disjointWith` 允许声明一个类运算式的类外延和另一个类运算式的类外延没有共同成员。

在句法上，这三种语言结构都是有一个类运算式作为其定义域和值域的属性。我们将会在接下来的子章节中对这些属性做更详细的讨论。

此外，在 OWL 中，允许类公理给枚举或集合运算符类型的类运算式指定一个名称。这些类公理语义上等价于有 `owl:equivalentClass` 声明的类公理，因此那些类公理将会在后面的小节讨论（参见 Sec. 3.2.3“不使用 owl:equivalentClass 的完整类的公理”）。

### 3.2.1 rdfs:subClassOf

AXIOM SCHEMA：*class description* `rdfs:subClassOf` *class description*

`rdfs:subClassOf` 结构被定义为 RDF Schema[*RDF Vocabulary*]的一部分。它在 OWL 中的含义完全一样：如果类运算式 C1 定义为类运算式 C2 的子类，那么 C1 的类外延的个体集应该是 C2 的类外延的个体集的子集。从定义上讲，一个类是其自身的子类（因为子集可以是完全集）。

一个例子：

```
<owl:Class rdf:ID="Opera">
  <rdfs:subClassOf rdf:resource="#MusicalWork" />
</owl:Class>
```

上面的类公理声明了两个通过名称（`Opera` 和 `MusicalWork`）描述的 OWL 类之间的子类关系。子类关系为属于一个类提供了必要条件。在这种情况下，是一个歌剧的个体也应该是一个音乐作品的个体。

注意：在 OWL Lite 中，一个 `rdfs:subClassOf` 声明的主体必须是一个类标识符。客体必须是一个类标识符或者属性限制。

任何类都有可能有任意数量的 subClassOf 公理。例如，我们可以给类 Opera 添加下面的公理：

```
<owl:Class rdf:about="#Opera">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasLibrettist" />
```

```
      <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

这个类公理包含一个属性限制。上面给出的例子表明了 Opera 是一个匿名 OWL 类的子类（记住：owl:Restriction 是 owl:Class 的一个子类），它的类外延所有个体的集合对于属性 hasLibrettist 至少有一个值。因此，歌剧应该至少有一个剧本作者。

当类运算式嵌套以后，类公理会变得更加复杂。例如，带有 owl:allValuesFrom 或 owl:someValuesFrom 声明的属性限制可以指向任何类运算式。考虑下面的例子：

```
<owl:Class rdf:ID="TraditionalItalianOpera">
  <rdfs:subClassOf rdf:resource="#Opera"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasOperaType"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:oneOf rdf:parseType="Collection">
            <owl:Thing rdf:about="#OperaSeria"/>
            <owl:Thing rdf:about="#OperaBuffa"/>
          </owl:oneOf>
        </owl:Class>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

上面的例子显示了 owl:oneOf 结构的用法。这个类公理将传统的意大利歌剧定义为歌剧类的一个子类，歌剧类有 *opera seria* 或者 *opera buffa*（如果没有附加的基数约束，实际上它可以同时有两个值）作为它的歌剧类型。

在指南文档[*OWL Guide*]里可以看到更多的例子。子类公理为我们提供了部分定义：它们代表建立一个个体的类成员的必要但不充分的条件。在接下来的小节我们将会看到 OWL 提供 owl:equivalentClass 结构来定义充分必要条件。作为此类公理的垫脚石，我们来考虑下面的例子：

```
<owl:Class rdf:ID="Operetta">
  <rdfs:subClassOf rdf:resource="#MusicalWork"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasLibrettist" />
      <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class>
      <owl:complementOf rdf:resource="#Opera"/>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

这个类公理声明了 operetta 是一个音乐作品，它至少有一个剧本作者，并且它不是歌剧。运用子类关系保留了还有其他音乐作品的可能性，这样的音乐作品有一个作者并且不是歌剧。如果我们想要指明 operetta's *恰好*是那些有一个作者但不是歌剧的音乐作品，我们需要使用 owl:equivalentClass 结构。

### 3.2.2 owl:equivalentClass

AXIOM SCHEMA: *class description* `owl:equivalentClass` *class description*

一个类公理可能包含(多个)owl:equivalentClass 声明。`owl:equivalentClass` 是一个内置属性，它把一个类运算式连接到另外一个类运算式。这类公理的意义是两个有关的类运算式有相同的类外延（即，两个类运算式含有完全一样的个体集）。

在其最简单的形式里，等价类公理声明了两个具名类的等价关系（根据它们的类外延）。实例如下：

```
<owl:Class rdf:about="#US_President">
  <equivalentClass
rdf:resource="#PrincipalResidentOfWhiteHouse"/>
</owl:Class>
```

注意：使用 `owl:equivalentClass` 并不意味着类相等(class equality)。类相等意味着类有相同的内涵（intensional meaning）（表示相同的概念）。在上面的例子中，“美国总统”的概念与某一地产主要居民的概念有关，但并不等于那个概念。真正的类相等只能用 `owl:sameAs` 结构表达。因为类相等需要把类当做个体，所以只能在 OWL Full 中表达。

通过链接类型 1 类运算式（类标识符）到类型 2 类运算式（枚举），使用 `owl:equivalentClass` 的类公理也能够用来定义一个枚举类。一个例子：

```
<owl:Class rdf:ID="DaPonteOperaOfMozart">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <Opera rdf:about="#Nozze_di_Figaro"/>
        <Opera rdf:about="#Don_Giovanni"/>
        <Opera rdf:about="#Cosi_fan_tutte"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

这个类公理定义了共同表示“莫扎特的达蓬特歌剧”（音乐学的热门主题）的歌剧类。通过等价类结构的使用，我们能够为类成员声明充分必要条件，在此例中包含三个个体的枚举，不多不少。

注意：OWL DL 没有对可以用来作为 `owl:equivalentClass` 声明的主体和客体的类运算式类型做任何限制。在 OWL Lite 中，主体必须是类名，客体必须是类名或属性限制。

注意：虽然原则上不同类型的类运算式允许作为等价类声明的主体，实际上主体通常使用类标识符。本节中的例子也是如此。

同一个类可以有多个等价类公理。但是，这就需要小心。两个公理必须得出相同的结果，即完全相同的类外延。例如，另外一个莫扎特的“达蓬特歌剧”等价类公理可以是下面的写法：

```
<owl:Class rdf:about="#DaPonteOperaOfMozart">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
```

```
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasComposer"/>
          <owl:hasValue
rdf:resource="#Wolfgang_Amadeus_Mozart"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty
rdf:resource="#hasLibrettist"/>
          <owl:hasValue
rdf:resource="#Lorenzo_Da_Ponte"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
```

上面的例子表明莫扎特的达蓬特歌剧的类外延完全对应于由莫扎特作曲达蓬特写剧本的那些歌剧（注意：交集=“与”）。上面的公理确实定义了一个和之前的公理有着完全相同实例的类。

注意：如果我们想要把一个“A subClassOf B”形式的公理“升级”到“A equivalentClass B”（意味着 A 的类外延不只是子集，实际上是和 B 的类外延相同的集合），我们可以添加第二个 subClassOf 公理，形如（B subClassOf A），这两个类外延定义上就等价了（从而具有和“A 类等价于 B”相同的意义）。这样的子类关系“循环”是明确允许的。因为 OWL 可以用在分布式环境中，这个特性非常有用。

### 3.2.3 不使用 owl:equivalentClass 的完整类的公理

AXIOM SCHEMA: named *class description* of type 2 (with `owl:oneOf`) or type 4-6 (with `owl:intersectionOf`, `owl:unionOf` or `owl:complementOf`

OWL 允许用户通过指定枚举或集合运算符类型的类运算式名称来定义类公理。这种类公理为建立类成员定义了充分必要条件。一个例子：

```
<owl:Class rdf:ID="DaPonteOperaOfMozart">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Nozze_di_Figaro"/>
    <owl:Thing rdf:about="#Don_Giovanni"/>
    <owl:Thing rdf:about="#Cosi_fan_tutte"/>
  </owl:oneOf>
</owl:Class>
```

这个公理的解释如下：类 `DaPonteOperaOfMozart` 的类外延完全由枚举定义。

这个类公理语义上等价于上一节中的包含了附加 `owl:equivalentClass` 声明的第一个歌剧例子。这一类型的公理也可以用 `owl:intersectionOf`、`owl:unionOf` 和 `owl:complementOf` 构造。一个使用并集的例子如下：

```
<owl:Class rdf:ID="LivingBeing">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Plant"/>
    <owl:Class rdf:about="#Animal"/>
  </owl:unionOf>
</owl:Class>
```

这个类公理声明了 LivingBeing 的类外延完全对应于 Plant 和 Animal 类外延的并集。

注意：OWL Lite 只包括用 owl:intersectionOf 属性构造的这类公理。intersectionOf 列表的值必须是类标识符和/或属性限制。因此，使用枚举、补集和并集的“完整类”公理不能在 OWL Lite 中使用。

### 3.2.4 owl:disjointWith

AXIOM SCHEMA: *class description* owl:disjointWith *class description*

一个类公理也许包含（多个）owl:disjointWith 声明。owl:disjointWith 是一个以类运算式为定义域和值域的内置 OWL 属性。每个 owl:disjointWith 声明断言所涉及的两个类运算式的类外延没有共同的个体。如同含有 rdfs:subClassOf 的公理，声明两个类不相交是部分定义：它把一个必要但不充分条件施加于此类。

下面是一个关于不相交类的通俗例子：

```
<owl:Class rdf:about="#Man">
  <owl:disjointWith rdf:resource="#Woman"/>
</owl:Class>
```

这个例子正确与否实际上是由生物学家来决定的。下面的例子给出了在子类等级结构中类不相交性的常见用法：

```
<owl:Class rdf:about="#MusicDrama">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Opera"/>
        <owl:Class rdf:about="#Operetta"/>
        <owl:Class rdf:about="#Musical"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="#Opera">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
</owl:Class>

<owl:Class rdf:about="#Operetta">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
  <owl:disjointWith rdf:resource="#Opera"/>
</owl:Class>

<owl:Class rdf:about="#Musical">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
  <owl:disjointWith rdf:resource="#Opera"/>
  <owl:disjointWith rdf:resource="#Operetta"/>
</owl:Class>
```

在这里，为了定义一个父类的一组互不相交且完整的子类，owl:disjointWith 声明和 owl:unionOf 一起使用。在自然语言中：每一个 MusicDrama 要么是一个 Opera 要么是一个 Operetta 或者 Musical（子类的划分是完整的），并且个体如果属于其中一个子类，例如 Opera，就不能属于另外一个子类，比如 Musical（不相交或不重叠的子类）。这是在许多数据-建模符号（data-modelling notations）中常用的建模概念。

注意：OWL Lite 中不允许使用 owl:disjointWith。

# 4. 属性

OWL 对本体建造者可能想要定义的属性进行两大主要类别的区分：

- *对象属性（Object properties）*链接个体到个体。
- *数据类型属性（Datatype properties）*链接个体到数据值。

注意：OWL 也有注释属性（owl:AnnotationProperty）和本体属性（owl:OntologyProperty）的概念。在 OWL DL 里因语义原因需要这些属性。参见 Sec. 7 和“OWL 语义和抽象语法”文档[*OWL S&AS*]。

对象属性被定义为内置 OWL 类 owl:ObjectProperty 的一个实例。数据类型属性被定义为内置 OWL 类 owl:DatatypeProperty 的一个实例。owl:ObjectProperty 和 owl:DatatypeProperty 都是 RDF 类 rdf:Property 的子类（参见附录 B）。

注意：在 OWL Full 中，对象属性和数据类型属性并不互斥。因为数据值可以被视为个体，数据类型属性是对象属性的有效子类。OWL Full 中，owl:ObjectProperty 等价于 rdf:Property。在实践中，这主要对 owl:InverseFunctionalProperty 的使用产生影响。参见 Sec. 8.1OWL Full 特性。

属性公理定义一个属性的特征。一个属性公理在其最简单的形式中只定义一个属性的存在。例如：

```
<owl:ObjectProperty rdf:ID="hasParent"/>
```

这个例子定义了一个属性，限制条件为其值必须是个体。

属性公理常常会定义附加属性特征。OWL 支持下面的属性公理结构：

- RDF Schema 结构：rdfs:subPropertyOf，rdfs:domain 和 rdfs:range
- 其他属性关系：owl:equivalentProperty 和 owl:inverseOf
- 全局基数约束：owl:FunctionalProperty 和 owl:InverseFunctionalProperty
- 逻辑属性特征：owl:SymmetricProperty 和 owl:TransitiveProperty

在接下来的小节中，会更加详细地讨论各种类型的属性公理。

注意：在这一节中我们使用的术语“属性外延”类似于“类外延”。属性外延是与该属性相关的实例集合。属性的实例不是单一元素，而是属性声明的主-客体对。在关系数据库术语里，属性实例被称为二元关系（属性）的“元组”（tuples）。

## 4.1 RDF Schema 属性结构

本节中的结构在 RDF Schema 文档[*RDF Vocabulary*]中做了更详细的讨论。本节中的描述仅提供这些结构的概要和一些专用于 OWL 的方面及例子。

### 4.1.1 rdfs:subPropertyOf

rdfs:subPropertyOf 公理定义了一个属性是另一属性的子属性。这意味着如果 P1 是 P2 的子属性，那么 P1 的属性外延（一个对集合）应该是 P2 的属性外延（也是一个对集合）的子集。

一个例子：

```
<owl:ObjectProperty rdf:ID="hasMother">
  <rdfs:subPropertyOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

这表明属性“hasMother”属性外延中的所有实例（对）也是属性“hasParent”属性外延的成员。

子属性公理可以应用于数据类型属性和对象属性。

注意：OWL DL 中，一个子属性声明的主体和客体必须两者都是数据类型属性或者对象属性。

### 4.1.2 rdfs:domain（定义域）

对于一个属性可以定义（多个）rdfs:domain 公理。在句法上，rdfs:domain 是内置属性，它链接一个属性（类 rdf:Property 的某个实例）到类运算式。rdfs:domain 公理断言这样的属性声明的主体必须属于指定类运算式的类外延。

多重 rdfs:domain 公理是允许的，并且应该被解释为一个合取：这限制了该属性的定义域是那些属于类运算式*交集*的个体。如果想要声明多个类可以充当定义域，那么应该使用一个 owl:unionOf 形式的类运算式。例如，如果我们想要声明属性 hasBankAccount 的定义域是 Person 或者 Corporation，我们需要声明类似下面的东西：

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Corporation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

注意：OWL Lite 中，rdfs:domain 的值必须是一个类标识符。

### 4.1.3 rdfs:range（值域）

对于一个属性可以定义（多个）rdfs:range 公理。在句法上，rdfs:range 是内置的属性，它链接一个属性（类 rdf:Property 的某个实例）到一个类运算式或者数据值域。rdfs:range 公理断言这个属性的值必须属于类运算式的类外延或者指定数据值域内的数据值。

多重值域约束解读为声明属性的值域是所有值域的*交集*（即，类运算式的类外延的交集或数据值域的交集）。类似于 rdfs:domain，多个供选择的值域可以使用 owl:unionOf 形式的类运算式来指定（参见上一小节）。

需要注意的是，不同于本节中描述的关于类运算式的任何值约束，rdfs:range 限制是全局的。值约束（比如 owl:allValuesFrom）用在一个类运算式中，只有当约束应用到那个类时，才会对该属性起作用。与此相反，rdfs:range 限制适用于该属性而与它被用于哪个类无关。因此，使用 rdfs:range 的时候要小心。

注意：OWL Lite 中，允许作为 rdfs:range 的客体的唯一类运算式类型是类名。

## 4.2 与其他属性的关系

### 4.2.1 owl:equivalentProperty

owl:equivalentProperty 结构能够用来声明两个属性有相同的属性外延。在句法上，owl:equivalentProperty 是内置的 OWL 属性，它以 rdf:Property 为定义域和值域。

注意：属性等价（property equivalence）不同于属性相等（property equality）。等价的属性有着相同的“值”（即，相同的属性外延），但可以有不同的内涵（即，表示不同的概念）。属性相等应该用 owl:sameAs 结构来表达。由于这样的公理要求把属性看作个体，因此它只能在 OWL Full 中使用。

### 4.2.2 owl:inverseOf

属性具有一个从定义域到值域的方向性。在实践中，人们会经常发现在两个方向定义关系很有用：人拥有车，车属于人。owl:inverseOf 结构可以用来定义属性之间的这种逆关系。

在句法上，owl:inverseOf 是一个内置的 OWL 属性，它以 owl:ObjectProperty 作为定义域和值域。一个 P1 owl:inverseOf P2 形式的公理断言 P1 属性外延中的每一对（x，y），在 P2 属性外延中都有一对（y,x）与之对应，反之亦然。因此，owl:inverseOf 是一个对称属性。

一个例子：

```
<owl:ObjectProperty rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

## 4.3 关于属性的全局基数约束

### 4.3.1 owl:FunctionalProperty

函数型属性是这样的一个属性，对于每个 x 实例，它只能有一个（唯一）值 y 与之对应，即不存在两个不同的值 y1 和 y2 使得（x,y1）和（x,y2）都是这个属性的实例。对象属性和数据类型属性都可以声明为“函数型”（functional）。为了这一目的，OWL 定义内置类 owl:FunctionalProperty 为 RDF 类 rdf:Property 的一个特殊子类。

下面的公理表明了 husband 属性是函数型的，即一个女人最多有一个丈夫（本体的文化依赖的一个好例子）：

```
<owl:ObjectProperty rdf:ID="husband">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Woman" />
  <rdfs:range rdf:resource="#Man" />
```

```
</owl:ObjectProperty>
```

一如往常有不同的句法变种。上例和下例在语义上等价：

```
<owl:ObjectProperty rdf:ID="husband">
  <rdfs:domain rdf:resource="#Woman" />
  <rdfs:range  rdf:resource="#Man" />
</owl:ObjectProperty>

<owl:FunctionalProperty rdf:about="#husband" />
```

### 4.3.2 owl:InverseFunctionalProperty

如果一个属性声明为反函数型（inverse-functional），那么属性声明的客体唯一确定主体（某一个体）。更形式化地表述，如果我们声明 P 是一个 `owl:InverseFunctionalProperty`，那么就可断言，值 y 只能是 P 的对于单个实例 x 的值，即不存在两个不同的实例 x1 和 x2 使得（x1,y）和（x2,y）都是 P 的实例。

在句法上，通过声明一个属性是内置 OWL 类 owl:InverseFunctionalProperty 的一个实例来指定一个反函数型属性公理，上面所述的内置 OWL 类是 OWL 类 `owl:ObjectProperty` 的子类。

> 注意：因为在 OWL Full 中数据类型属性是对象属性的子类，所以一个反函数型属性可以被定义为数据类型属性。OWL DL 中，对象属性和数据类型不相交，因此一个反函数型属性不能被定义为数据类型属性。也参见 Sec. 8.1 和 Sec. 8.2。

一个反函数型属性的典型例子：

```
<owl:InverseFunctionalProperty rdf:ID="biologicalMotherOf">
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:InverseFunctionalProperty>
```

这个例子表明对 `biologicalMotherOf` 声明的每一个客体（某一人）应该能唯一确定一个主体（某一女性）。反函数型属性类似于数据库中键的概念。

一个与函数型属性不同的方面是，对于反函数型属性，不需要附加的对象属性或数据类型属性公理：反函数型属性在定义上就是对象属性。

我们注意到 `owl:FunctionalProperty` 和 `owl:InverseFunctionalProperty` 指定全局基数约束。也就是说，无论这个属性应用到哪个类，这些基数约束都必须遵守。这不同于属性限制中的基数约束。后者是类运算式，当应用于那个类时才会施加在属性上。

## 4.4 属性的逻辑特征

### 4.4.1 owl:TransitiveProperty

当一个属性 P 被定义为传递属性时，意味着如果(x,y)是 P 的一个实例，(y,z)也是 P 的一个实例，那么我们能推断出（x,z）也是 P 的一个实例。

在句法上，通过使其成为内置 OWL 类 owl:TransitiveProperty 的一个实例来定义一个属性的传递性，这个内置类是 OWL `owl:ObjectProperty` 的子类。

代表某些部分-整体关系的属性是传递属性的典型例子。例如，我们可能想要说明地区之间的属性 `subRegionOf` 具有传递性：

```
<owl:TransitiveProperty rdf:ID="subRegionOf">
  <rdfs:domain rdf:resource="#Region"/>
  <rdfs:range  rdf:resource="#Region"/>
</owl:TransitiveProperty>
```

根据此描述，一个 OWL 推理器应该能够推理出这样的结论：如果 ChiantiClassico、Tuscany 和 Italy 是地区，并且 ChiantiClassico 是 Tuscany 的子区域，Tuscany 是 Italy 的子地区，那么 ChiantiClassico 也是 Italy 的子区域。

我们注意到由于 owl:TransitiveProperty 是 owl:ObjectProperty 的子类，因此下面的句法变体与上面的例子等价：

```
<owl:ObjectProperty rdf:ID="subRegionOf">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Region"/>
  <rdfs:range  rdf:resource="#Region"/>
</owl:ObjectProperty>
```

注意：OWL DL 要求，对于一个传递属性，其本身、其父属性、其本身或其父属性的逆属性都不能声明局部或全局基数约束。

### 4.4.2 owl:SymmetricProperty

对称属性支持这一事实：如果（x,y）是 P 一个实例，那么（y,x）也是 P 的一个实例。句法上，通过成为内置 OWL 类 owl:SymmetricProperty 的一个实例来定义一个属性的对称性，这个内置 OWL 类是 owl:ObjectProperty 的子类。

friendOf 关系是对称属性的一个通俗的例子：

```
<owl:SymmetricProperty rdf:ID="friendOf">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range  rdf:resource="#Human"/>
</owl:SymmetricProperty>
```

一个对称属性的定义域和值域是相同的。

# 5. 个体

个体都定义为个体公理（也称作“事实”（facts））。我们讨论两种类型的事实：

1. 关于类成员和个体属性值的事实
2. 关于个体同一性的事实

## 5.1 类成员和属性值

许多事实通常是表示个体类成员和个体属性值的声明。例如，我们考虑下面关于类 Opera 实例的一组声明：

```
<Opera rdf:ID="Tosca">
  <hasComposer rdf:resource="#Giacomo_Puccini"/>
  <hasLibrettist rdf:resource="#Victorien_Sardou"/>
  <hasLibrettist rdf:resource="#Giuseppe_Giacosa"/>
  <hasLibrettist rdf:resource="#Luigi_Illica"/>
  <premiereDate
rdf:datatype="&xsd;date">1900-01-14</premiereDate>
  <premierePlace rdf:resource="#Roma"/>
```

```
  <numberOfActs
rdf:datatype="&xsd;positiveInteger">3</numberOfActs>
</Opera>
```

这个例子包含了个体 Tosca 的一些事实，个体 Tosca 是类 `Opera` 的一个实例。`Tosca` 由 Giacomo Puccini 作曲。这一歌剧有三个剧本作者。属性 `premiereDate` 链接歌剧到类型为 XML Schema 数据类型 `date` 的文字。关于数据类型的 XML schema 文档[*XML Schema Datatypes*]包含这种数据类型的语法和语义的相关信息。

个体公理不一定是关于具名个体：它们也可以涉及匿名个体。例如，我们考虑下面的 RDF/XML 片段。该示例定义了类 `Measurement` 的一个匿名实例的一些事实，列出了对哪些事实的定量观察，例如所观察的主体，观察到的现象，观察到的值，观察时间：

```
<Measurement>
  <observedSubject rdf:resource="#JaneDoe"/>
  <observedPhenomenon rdf:resource="#Weight"/>
  <observedValue>
    <Quantity>
      <quantityValue
rdf:datatype="&xsd;float">59.5</quantityValue>
      <quantityUnit rdf:resource="#Kilogram"/>
    </Quantity>
  </observedValue>
  <timeStamp
rdf:datatype="&xsd;dateTime">2003-01-24T09:00:08+01:00</timeStamp
>
</Measurement>
```

该个体公理包含两个匿名个体，即某一 `Measurement` 和某一 `Quantity`。在自然语言中，所测得的主体 Jane Doe 的现象 `Weight` 的值是某一数量，其值是以千克为单位的 59.5。测量时间是 UTC+1 时区（阿姆斯特丹、柏林、巴黎的冬令时），2003 年 1 月 24 日，上午 9 点过 8 秒。如前所述，`float` 和 `dateTime` 是 XML Schema 数据类型，在相关的 XML Schema 文档[*XML Schema Datatypes*]里可以找到其句法和语法细节。

## 5.2 个体同一性

许多语言都有一个所谓的“唯一名称”假设：不同的名称指向世界上不同的事物。但在网络上，这一假设是不可能的。例如，同一个人可以通过多种不同方式提及（即有不同的 URI 引用）。为此，OWL 没有作出这一假设。除非明确地声明两个 URI 引用指向相同或者不同个体，否则 OWL 工具应该原则上假定两种情况都有可能。

OWL 提供了三个结构来表达个体同一性的事实：

- `owl:sameAs` 用来表达两个 URI 引用指向相同的个体。
- `owl:differentFrom` 用来表达两个 URI 引用指向不同的个体。
- `owl:AllDifferent` 提供一个术语来表达列表中的个体都不相同。

### 5.2.1 owl:sameAs

内置 OWL 属性 owl:sameAs 链接两个个体。此 owl:sameAs 声明表明两个 URI 引用实际上指向相同的事物：这些个体拥有相同的“身份”。

对于比如“人”这一概念的个体来说相对容易理解。例如，我们可以声明下面两个 URI 引用实际上指向同一个人：

```
<rdf:Description rdf:about="#William_Jefferson_Clinton">
  <owl:sameAs rdf:resource="#BillClinton"/>
</rdf:Description>
```

owl:sameAs 声明经常用于定义本体间的映射。假设每个人都使用相同的名称指向个体是不切实际的。这将需要宏大的设计，有违网络精神。

OWL Full 中，一个类可以被当做（元）类的实例，我们可以使用 owl:sameAs 结构来定义类相等，从而表明这两个概念有相同的内涵。例如：

```
<owl:Class rdf:ID="FootballTeam">
  <owl:sameAs rdf:resource="http://sports.org/US#SoccerTeam"/>
</owl:Class>
```

我们可以想象这个公理是欧洲体育本体的一部分。这两个类在这里被当做个体，作为类 owl:Class 的实例。这使得我们可以声明某个欧洲体育本体里的类 FootballTeam 和某个美国体育本体里的类 SoccerTeam 表示相同的概念。我们注意到这样一个声明的不同：

```
<footballTeam owl:equivalentClass us:soccerTeam />
```

它表明这两个类有相同的类外延，但概念不（一定）相同。

> 注意：URI 引用对照详情请参见 RDF 概念文档[*RDF Concepts*]里的 RDF URI 引用相关章节。

### 5.2.2 owl:differentFrom

内置 OWL owl:differentFrom 属性链接两个个体。owl:differentFrom 声明表示两个 URI 引用指向不同的个体。

一个例子：

```
<Opera rdf:ID="Don_Giovanni"/>

<Opera rdf:ID="Nozze_di_Figaro">
  <owl:differentFrom rdf:resource="#Don_Giovanni"/>
</Opera>

<Opera rdf:ID="Cosi_fan_tutte">
  <owl:differentFrom rdf:resource="#Don_Giovanni"/>
  <owl:differentFrom rdf:resource="#Nozze_di_Figaro"/>
</Opera>
```

上述例子声明了三个不同的歌剧。

### 5.2.3 owl:AllDifferent

对于唯一名称假设成立的本体，当所有个体不得不声明两两不相交时，使用 owl:differentFrom 可能导致大量的声明。针对这一情况，OWL 提供了一个 owl:AllDifferent 结构形式的专门术语。owl:AllDifferent 是一个特殊的内置 OWL 类，此类要定义属性 owl:distinctMembers，用来链接 owl:AllDifferent 的一个实例到个体列表。这个声明的含义是列表中的所有个体彼此互不相同。

一个例子：

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Opera rdf:about="#Don_Giovanni"/>
    <Opera rdf:about="#Nozze_di_Figaro"/>
    <Opera rdf:about="#Cosi_fan_tutte"/>
```

```
    <Opera rdf:about="#Tosca"/>
    <Opera rdf:about="#Turandot"/>
    <Opera rdf:about="#Salome"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

上述例子表明这六个 URI 引用均指向不同的歌剧。

注意：`owl:distinctMembers` 是一个专门的句法结构，它的增加带来了便利，它应该始终与作为其主体的一个 `owl:AllDifferent` 个体一起使用。

# 6. 数据类型

我们在本文档的许多地方已经看到过数据值域的概念，用以指定一个数据值的范围。OWL 允许三种类型的数据值域规范：

- RDF 数据类型规范。
- RDFS 类 rdfs:Literal。
- 枚举数据类型，使用 `owl:oneof` 结构。

在 Sec. 6.3 里对工具支持数据类型的最低标准做了讨论。

## 6.1 RDF 数据类型

OWL 使用 RDF 数据类型方案，它提供了一个机制指向 XML Schema 数据类型[*XML Schema Datatypes*]。有关的详细描述，读者可参考 RDF 文档，例如[*RDF Concepts*]。为方便读者，在这里我们提供了 RDF 数据类型使用的概要。

数据值是 RDF Schema 类 `rdfs:Literal` 的实例。文本可以是普通的（没有数据类型）或者类型化的。数据类型是类 `rdfs:Datatype` 的实例。在 RDF/XML 中，一个文本的类型由 `rdf:datatype` 属性指定，建议其值为下面列举之一：

- 对 XML Schema 数据类型的一个规范的 URI 引用：
  - `http://www.w3.org/2001/XMLSchema#NAME`

  “NAME”处应替换为*简单* XML Schema *内置*数据类型的名称，如同[*XML Schema Datatypes*]的 Section 3 中定义的那样，带有下面指定的附带条件。
- 数据类型 `rdf:XMLLiteral` 的 URI 引用。这一数据类型用来把 XML 内容包含到一个 RDF/OWL 文档中。这一数据类型的 URI 引用是：
  - `http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral`

  有关这个数据类型的详情，请参见“RDF 概念”文档[*RDF Concepts*]。

“RDF 语义”文档[*RDF Semantics*，Section 5]推荐使用下面的简单内置 XML Schema 数据类型。

- 原始数据类型 xsd:string，再加上从 xsd:string 派生的下列数据类型：xsd:normalizedString、xsd:token、xsd:language、xsd:NMTOKEN、xsd:Name 和 xsd:NCName 。
- 原始数据类型 xsd:boolean。
- 原始数值数据类型 xsd:decimal、xsd:float 和 xsd:double，再加上从 xsd:decimal 派生的所有类型（xsd:integer、xsd:positiveInteger、xsd:nonPositiveInteger、

xsd:negativeInteger、xsd:nonNegativeInteger、xsd:long、xsd:int、xsd:short、xsd:byte、xsd:unsignedLong、xsd:unsignedInt、xsd:unsignedShort、xsd:unsignedByte）。

- 原始的与时间有关的数据类型：xsd:dateTime、xsd:time、xsd:date、xsd:gYearMonth、xsd:gYear、xsd:gMonthDay、xsd:gDay 和 xsd:gMonth。
- 原始数据类型 xsd:hexBinary、xsd:base64Binary 和 xsd:anyURI。

注意：虽然我们不建议这么做，但应用程序通过定义 `rdfs:Datatype` 的一个实例来定义它们自己的数据类型并不非法。这样的数据类型是“未识别的数据类型”，但会以类似“不支持的数据类型”的方式来处理。（OWL 工具应如何处理的更多细节参见 Sec. 6.3）。

请注意，当使用数据类型时，即使一个属性定义为有某一数据类型值域，RDF/XML 仍然要求每次使用该属性都要指定数据类型。我们在前面使用过的 `Measurement` 例子中的属性声明可以作为这里的一个示例：

```
<owl:DatatypeProperty rdf:about="#timeStamp">
  <rdfs:domain rdf:resource="#Measurement"/>
  <rdf:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<Measurement>
  <timeStamp
rdf:datatype="&xsd;dateTime">2003-01-24T09:00:08+01:00</timeStamp
>
</Measurement>
```

## 6.2 枚举数据类型

除了 RDF 数据类型之外，OWL 还提供了另外一个结构来定义数据值的范围，即枚举数据类型。此数据类型格式使用了 `owl:oneof` 结构，这个结构也可用来描述一个枚举类。在枚举数据类型的情况下，`owl:oneof` 的主体是 `owl:DataRange` 类的一个空节点，客体是一个文本列表。可惜的是，我们不能使用术语 `rdf:parseType="Collection"`来指定文本列表，因为 RDF 要求这个集合（collection）是一个 RDF 节点元素列表。因此，我们不得不使用基本列表结构 `rdf:first`、`rdf:rest` 和 `rdf:nil` 指定数据值列表。

注意：枚举数据类型不是 OWL Lite 的一部分。

下面的例子指定了属性 `tennisGameScore` 的范围是整数值列表{0, 15, 30, 40}：

```
<owl:DatatypeProperty rdf:ID="tennisGameScore">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf>
        <rdf:List>
          <rdf:first rdf:datatype="&xsd;integer">0</rdf:first>
          <rdf:rest>
            <rdf:List>
              <rdf:first rdf:datatype="&xsd;integer">15</rdf:first>
              <rdf:rest>
                <rdf:List>
                  <rdf:first rdf:datatype="&xsd;integer">30</rdf:first>
                  <rdf:rest>
                    <rdf:List>
                      <rdf:first rdf:datatype="&xsd;integer">40</rdf:first>
                      <rdf:rest rdf:resource="&rdf;nil" />
```

```
                </rdf:List>
              </rdf:rest>
            </rdf:List>
          </rdf:rest>
        </rdf:List>
      </rdf:rest>
    </rdf:List>
  </owl:oneOf>
 </owl:DataRange>
</rdfs:range>
</owl:DatatypeProperty>
```

## 6.3 对数据类型推理的支持

工具可能在对数据类型推理的支持方面有所不同。作为最低要求，工具必须支持对 XML Schema 数据类型 `xsd:string` 和 `xsd:integer` 的数据类型推理。OWL Full 工具还必须要支持 `rdf:XMLLiteral`。对于不支持的数据类型，词法相同的文本应该被视为相等，而词法不同的文本既不被视为相等也不被视为不等。无法识别数据类型的处理方式应该和不支持数据类型的处理方式一样。

# 7. 注释、本体头、导入和版本信息

## 7.1 注释

OWL Full 没有对一个本体中的注释做任何约束。OWL DL 允许注释类、属性、个体和本体头，但仅在下列情况下：

- 对象属性、数据类型属性、注释属性和本体属性集合必须互不相交。因此，在 OWL DL 中 dc:creator 不能既是一个数据类型属性同时又是一个注释属性。
- 注释属性必须有一个显式的定义类型的三元组形式：
  - `AnnotationPropertyID rdf:type owl:AnnotationProperty .`
- 注释属性一定不能用在属性公理中。因此，在 OWL DL 中我们不能为注释属性定义子属性或定义域/值域约束。
- 一个注释属性的客体必须是数据文本、URI 引用或个体。

OWL 预定义了五种注释属性，即：

- `owl:versionInfo`
- `rdfs:label`
- `rdfs:comment`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`

OWL DL 中注释属性合法使用的例子如下：

```
<owl:AnnotationProperty rdf:about="&dc;creator"/>

<owl:Class rdf:about="#MusicalWork">
  <rdfs:label>Musical work</rdfs:label>
  <dc:creator>N.N.</dc:creator>
</owl:Class>
```

上述例子假定&dc;和 dc:分别指向 Dublin Core URI 和命名空间。因此，在 OWL DL 中 Dublin Core 属性当做注释属性来使用需要一个显式的定义类型三元组。这确保了注释在语义正确的方式下由 OWL DL 推理器处理（更多详情参见“OWL 语义和抽象语法”文档[*OWL S&AS*]）。

一旦我们将 dc:creator 定义为注释属性，OWL DL 就不允许有比如下面的值域约束这样的属性公理：

```
<-- 这在 OWL DL 中是非法的 -->

<owl:AnnotationProperty rdf:about="&dc;creator">
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:AnnotationProperty>
```

我们注意到仍可以在注释-属性声明中指定一个文本类型的值：

```
<Opera rdf:about="#Tosca">
  <dc:creator rdf:datatype="&xsd;string">Giacomo
Puccini</dc:creator>
</Opera>
```

## 7.2 本体头

描述一个本体的文档通常包含本体本身的信息。本体是一种资源，所以可以使用 OWL 的属性和其他命名空间来描述它，例如：

```
<owl:Ontology rdf:about="">
  <owl:versionInfo> ... </owl:versionInfo>
  <rdfs:comment>...</rdfs:comment>
  <owl:imports rdf:resource="..."/>
</owl:Ontology>
```

这通常被称为本体头，我们通常可在 RDF/XML 文档头附近看见它。其中这一行

```
<owl:Ontology rdf:about="">
```

表明本模块描述了当前本体。更确切地说，它表明当前基准 URI 标识类 owl:Ontology 的一个实例。建议在文档的开头使用<rdf:RDF>元素中的 xml:base 属性定义基准 URI。

一个样本本体头看起来是这样的：

```
<owl:Ontology rdf:about="">
  <owl:versionInfo>v 1.17 2003/02/26 12:56:51
mdean</owl:versionInfo>
  <rdfs:comment>An example ontology</rdfs:comment>
  <owl:imports rdf:resource="http://www.example.org/foo"/>
</owl:Ontology>
```

下面的章节描述了通常用在头部的各种声明类型。

> 注意：在 OWL 词汇表中将本体-导入结构 owl:imports 和本体版本结构 owl:priorVersion、owl:backwardCompatibleWith 及 owl:incompatibleWith 定义为 OWL 内置类 owl:OntologyProperty 的实例。owl:OntologyProperty 的实例必须要以类 owl:Ontology 作为它们的定义域和值域。允许定义其他 owl:OntologyProperty 的实例。在 OWL DL 中，对于本体属性的约束和 Sec. 7.1 中为注释属性指定的约束一样。

## 7.3 导入本体

一个 owl:imports 声明引用另外一个包含定义的 OWL 本体，其含义被认为是该导入本体含义的一部分。每个引用都包含一个 URI 指明该本体从何处被导入。在句法上，owl:imports 是一个以类 owl:Ontology 为定义域和值域的属性。

owl:imports 声明是传递性的，也就是说，如果本体 A 导入了 B，B 导入了 C，那么 A 就导入了 B 和 C。

一个本体导入自己本身被认为是空动作，因此，如果本体 A 导入 B，B 也导入 A，那么就认为 A 和 B 等价。

注意：一个 OWL 工具是否必须加载一个被导入本体取决于工具的目的。如果该工具是一个完整的推理器（包括完整的一致性检查器），那么它必须加载所有的被导入本体。其他工具，比如简单的编辑器和不完整的推理器可以选择性地加载一些甚至不加载被导入本体。

虽然 owl:imports 和命名空间声明可能显得多余，实际上它们用于不同的目的。命名空间声明仅仅为引用标识符设立一个简写。它们没有隐含定位于 URI 的文档的含义。另一方面，owl:imports 没有为引用被导入文档的标识符提供任何简写符号。因此，任何被导入的本体都有一个相应的命名空间声明。

注意：owl:imports 是 owl:OntologyProperty 的一个实例。

## 7.4 版本信息

### 7.4.1 owl:versionInfo

owl:versionInfo 声明通常都有一个字符串作为其客体提供这一版本的信息，例如 RCS/CVS 关键词。除了 RDF(S)模型理论给定的意义，此声明对本体的逻辑意义没有贡献。

虽然这个属性通常用来进行本体的声明，但它可以应用于任何 OWL 结构。例如，我们可以为一个 OWL 类附加一个 owl:versionInfo 声明。

注意：owl:versionInfo 是 owl:AnnotationProperty 的一个实例。

### 7.4.2 owl:priorVersion

owl:priorVersion 声明包含一个到另外一个本体的引用。它标识了指定的本体是包含（containing）本体的以前版本。除了 RDF(S)模型理论给定的意义，此声明在模型-理论语义中没有任何意义。然而，软件可以用它来根据版本组织本体。

owl:priorVersion 是以类 owl:Ontology 作为其定义域和值域的内置 OWL 属性。

注意：owl:priorVersion 是 owl:OntologyProperty 的一个实例。

### 7.4.3 owl:backwardCompatibleWith

owl:backwardCompatibleWith 声明包含对另外一个本体的引用。它标识了指定的本体是包含本体的以前版本，并进一步表明新版本向后兼容。具体而言，它表明以前版本中的所有标识符在新版本中有相同的预期解释。因此，此声明暗示文档作者可以放心地将他们的文档改为新版本（通过简单地更新命名空间声明和 owl:imports 陈述指向新版本的 URL）。如果没有为两个版本声明 owl:backwardCompatibleWith，那么就不能假定它们的兼容性。

除了 RDF(S)模型理论给定的含义，owl:backwardCompatibleWith 在模型-理论语义中没有任何意义。

owl:backwardCompatibleWith 是以类 owl:Ontology 作为其定义域和值域的内置 OWL 属性。

注意：owl:backwardCompatibleWith 是 owl:OntologyProperty 的一个实例。

### 7.4.4 owl:incompatibleWith

owl:incompatibleWith 声明包含对另外一个本体的引用。它表明了包含本体是所引用本体的更高版本，但新版本不向后兼容。对于那些想要明确说明在没有检查是否需要更改的情况下文档不能升级到新版本的本体作者而言，此声明的使用是必要的。

除了 RDF(S)模型理论给定的含义，owl:incompatibleWith 在模型-理论语义中没有任何意义。

owl:incompatibleWith 是以类 owl:Ontology 作为其定义域和值域的内置 OWL 属性。

注意：owl:incompatibleWith 是 owl:OntologyProperty 的一个实例。

### 7.4.5 owl:DeprecatedClass 和 owl:DeprecatedProperty

过时是版本控制软件中常用的特性（例如，参见 Java 编程语言），它表示保留一个特定功能以便向后兼容的目的，但将来可能会逐步淘汰。在这里，一个特定标识符的类型可以是 owl:DeprecatedClass 或 owl:DeprecatedProperty，其中 owl:DeprecatedClass 是 rdfs:Class 的子类，owl:DeprecatedProperty 是 rdf:Property 的子类。让一个术语过时，意味着提交给本体的新文档里不应该使用该术语。这使得一个本体可以保持向后兼容性，同时逐步淘汰旧词汇（因此，过时与向后兼容性结合使用才有意义）。其结果是旧数据和应用程序更容易移植到新版本，因此可以提高新版本的采用程度。除了 RDF(S)模型理论给定的含义，这在模型-理论语义中没有任何意义。然而，当检查 OWL 标记时程序编写工具可以用来向用户发出警告。

一个关于过时的例子：

```
<owl:Ontology rdf:about="">
  <rdfs:comment>Vehicle Ontology, v. 1.1</rdfs:comment>
  <owl:backwardCompatibleWith
          rdf:resource="http://www.example.org/vehicle-1.0"/>
  <owl:priorVersion
rdf:resource="http://www.example.org/vehicle-1.0"/>
</owl:Ontology>

<owl:DeprecatedClass rdf:ID="Car">
  <rdfs:comment>Automobile is now preferred</rdfs:comment>
```

```
  <owl:equivalentClass rdf:resource="#Automobile"/>
  <!-- note that equivalentClass only means that the classes have the
same
       extension, so this DOES NOT lead to the entailment that
       Automobile is of type DeprecatedClass too -->
</owl:DeprecatedClass>

<owl:Class rdf:ID="Automobile" />

<owl:DeprecatedProperty rdf:ID="hasDriver">
  <rdfs:comment>inverse property drives is now
preferred</rdfs:comment>
  <owl:inverseOf rdf:resource="#drives" />
</owl:DeprecatedProperty>

<owl:ObjectProperty rdf:ID="drives" />
```

# 8. OWL Full，OWL DL 和 OWL Lite

在引言中，我们简要地讨论了 OWL 的三种子语言。在本节中，我们将给出 OWL 的三个“种类”之间差别的资料性说明。“语义和抽象”文档[*OWL S&AS*]里给出了这些差别的形式化解释。

## 8.1 OWL Full

OWL Full 实际上不是子语言。OWL Full 包括所有 OWL 语言结构，并提供 RDF 结构自由的、不受限制的使用。在 OWL Full 中资源 `owl:Class` 相当于 `rdfs:Class`。这一点和 OWL DL 及 OWL Lite 不同，在后两者中 `owl:Class` 是 `rdfs:Class` 的真子类（proper subclass）（这意味着在 OWL DL 和 OWL Lite 中并非所有 RDF 类都是 OWL 类）。OWL Full 还允许类被当做个体。例如，在 OWL Full 中有一个“Fokker-100”标识符同时作为类名（表示一组环球飞行的 Fokker-100 飞机）和个体名（*例如，*类 `AirplaneType` 的一个实例）是完全合法的。

OWL Full 中所有数据值也被认为是个体定义域的一部分。事实上，OWL Full 中个体的总体由所有资源（`owl:Thing` 相当于 `rdfs:Resource`）组成。这意味着对象属性和数据类型属性并不互斥。在 OWL Full 中 `owl:ObjectProperty` 相当于 `rdf:Property`。其结果是，数据类型属性实际上是对象属性的子类。（注意：`owl:ObjectProperty` 和 `owl:DatatypeProperty` 都是 `rdf:Property` 的子类的事实与这点并不矛盾）。

对于想要结合 OWL 的表达性和 RDF 的灵活性以及元建模特性的人，OWL Full 将会很有用。然而，OWL Full 特性的使用意味着会失去 OWL DL 和 OWL Lite 可以提供给推理系统的一些保证（参见下文）。

> 注意：除非专门以 OWL DL 或 Lite 构造，否则 RDF 文档通常都采用 OWL Full。
>
> 注意：因此，OWL Full 中 `owl:Thing` 等价于 `rdfs:Resource`，`owl:Class` 等价于 `rdfs:Class`，`owl:ObjectProperty` 等价于 `rdf:Property`。

## 8.2 OWL DL

OWL DL 是 OWL 的子语言，它对 OWL 语言结构的使用做了许多限制。简要来讲，这些约束如下：

- OWL DL 要求类、数据类型、数据类型属性、对象属性、注释属性、本体属性（*即*，导入和版本控制属性）、个体、数据值和内置词汇之间两两分离。这意味着，例如，一个类不能同时是一个个体。
- OWL DL 中对象属性和数据类型属性集不相交。这意味着下面四个属性特征：
  - 
  - 逆，
  - 反函数型，
  - 对称性，和
  - 传递性

  永远不能为数据类型属性指定。
- OWL DL 要求不能对传递属性或它们的逆，或其父属性及父属性的逆进行基数约束（局部或全局）。
- 只允许在一定的条件下注释。详情参见 Sec. 7.1。
- 大多数 RDF(S)词汇不能在 OWL DL 中使用。更多详情参见“OWL 语义和抽象语法”文档[*OWL S&AS*]。
- 所有公理必须格式良好，没有丢失或存在多余的成分，而且必须形成树状结构。

  最后一个约束意味着人们引用的所有类和属性要分别显式地定型为 OWL 类或属性。例如，如果本体包含下面的成分：

```
<owl:Class rdf:ID="C1">
  <rdfs:subClassOf rdf:resource="#C2" />
</owl:Class>
```

  则该本体（或一个被导入此本体的本体）应该为 C2 包含一个 `owl:Class` 三元组。
- 关于个体相等和差异的公理（事实）必须关于具名个体。

这些 OWL DL 约束可能看起来像一个任意集合，但事实并非如此。这些约束基于针对描述逻辑的推理器领域的工作，这些工作需要这些限制向本体构建者或用户提供推理支持。具体而言，OWL DL 限制允许了 OWL Full 的最大子集，在此之上现有研究可以确保一个 OWL 推理器存在一个可判定的推理过程。

注意：附录 E 为采用 RDF 表示 OWL DL 本体提供了一套实践指南。

## 8.3 OWL Lite

OWL Lite 遵守所有 OWL DL 针对 OWL 语言结构的使用而制定的限制。此外，OWL Lite 将禁止使用：

- owl:oneOf
- owl:unionOf
- owl:complementOf
- owl:hasValue
- owl:disjointWith
- owl:DataRange

OWL Lite 还要求：

- owl:equivalentClass 三元组的主体是类名，owl:equivalentClass 三元组的客体是类名或者限制；
- rdfs:subClassOf 三元组的主体是类名，rdfs:subClassOf 三元组的客体是类名或者限制；
- owl:intersectionOf 只能用于长度大于 1 且只包含类名和限制的列表。

> 注意：这是 OWL Lite 中合法使用 owl:intersectionOf 的一个典型例子：

```
<owl:Class rdf:ID="Woman">
     <owl:intersectionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#Female"/>
       <owl:Class rdf:about="#Human"/>
     </owl:intersectionOf>
   </owl:Class/>
```

- owl:allValuesFrom 和 owl:someValuesFrom 三元组的客体是类名或者数据类型名；
- rdf:type 三元组的客体是类名或者限制；
- rdfs:domain 三元组的客体是类名；
- rdfs:range 三元组的客体是是类名或者数据类型名。

OWL Lite 表达力限制背后的理念是它们提供语言特性的最小有用子集，使得工具开发者更容易支持。OWL Lite 语言结构为子类等级结构构造提供了基础：子类和属性限制。此外， OWL Lite 允许属性是可选的或必备的。OWL Lite 的限制将其置于复杂度比 OWL DL 低的位置。这一点可以对针对 OWL Lite 的完整推理器的效率产生积极的影响。

只支持 OWL Lite 词汇，但在其他方面放宽 OWL DL 限制的实现并不能确保一致性和复杂性方面的计算要求。然而，这样的实现可能对提供 OWL 系统与 RDFS 模型、数据库、标记工具或其他非推理工具之间的互操作性有帮助。Web 本体工作组并没有为这一可能有益的子集提供一个名称。

# 附录 A. 所有语言元素的索引

> 注意：本附录只包含 OWL 专用结构。RDF/RDFS 结构请参见相关 RDF 文档，尤其是 RDF Schema 文档[*RDF Vocabulary*]。

| [OWL Reference]<br>（本文档） | [*OWL Semantics*]<br>（标准） | [*OWL Guide*]<br>（例子） |
|---|---|---|
| owl:AllDifferent | owl:AllDifferent | owl:AllDifferent |
| owl:allValuesFrom | owl:allValuesFrom | owl:allValuesFrom |
| owl:AnnotationProperty | owl:AnnotationProperty | |
| owl:backwardCompatibleWith | owl:backwardCompatibleWith | owl:backwardCompatibleWith |
| owl:cardinality | owl:cardinality | owl:cardinality |

| | | |
|---|---|---|
| owl:Class | owl:Class | owl:Class |
| owl:complementOf | owl:complementOf | owl:complementOf |
| owl:DataRange | owl:DataRange | |
| owl:DatatypeProperty | owl:DatatypeProperty | owl:DatatypeProperty |
| owl:DeprecatedClass | owl:DeprecatedClass | owl:DeprecatedClass |
| owl:DeprecatedProperty | owl:DeprecatedProperty | owl:DeprecatedProperty |
| owl:differentFrom | owl:differentFrom | owl:differentFrom |
| owl:disjointWith | owl:disjointWith | owl:disjointWith |
| owl:distinctMembers | owl:distinctMembers | owl:distinctMembers |
| owl:equivalentClass | owl:equivalentClass | owl:equivalentClass |
| owl:equivalentProperty | owl:equivalentProperty | owl:equivalentProperty |
| owl:FunctionalProperty | owl:FunctionalProperty | owl:FunctionalProperty |
| owl:hasValue | owl:hasValue | owl:hasValue |
| owl:imports | owl:imports | owl:imports |
| owl:incompatibleWith | owl:incompatibleWith | owl:incompatibleWith |
| owl:intersectionOf | owl:intersectionOf | owl:intersectionOf |
| owl:InverseFunctionalProperty | owl:InverseFunctionalProperty | owl:InverseFunctionalProperty |
| owl:inverseOf | owl:inverseOf | owl:inverseOf |
| owl:maxCardinality | owl:maxCardinality | owl:maxCardinality |
| owl:minCardinality | owl:minCardinality | owl:minCardinality |
| owl:Nothing | owl:Nothing | owl:Nothing |
| owl:ObjectProperty | owl:ObjectProperty | owl:ObjectProperty |
| owl:oneOf | owl:oneOf | owl:oneOf |
| owl:onProperty | owl:onProperty | owl:onProperty |
| owl:Ontology | owl:Ontology | owl:Ontology |
| owl:OntologyProperty | owl:OntologyProperty | |
| owl:priorVersion | owl:priorVersion | owl:priorVersion |
| owl:Restriction | owl:Restriction | owl:Restriction |
| owl:sameAs | owl:sameAs | owl:sameAs |
| owl:someValuesFrom | owl:someValuesFrom | owl:someValuesFrom |
| owl:SymmetricProperty | owl:SymmetricProperty | owl:SymmetricProperty |
| owl:Thing | owl:Thing | owl:Thing |
| owl:TransitiveProperty | owl:TransitiveProperty | owl:TransitiveProperty |

| | | |
|---|---|---|
| owl:unionOf | owl:unionOf | owl:unionOf |
| owl:versionInfo | owl:versionInfo | owl:versionInfo |

# 附录 B. OWL 的 RDF Schema

本附录目的的描述请参见 Sec. 1.7。这个附录的 RDF/XML 版本可以在 http://www.w3.org/2002/07/owl 上找到。

```
<?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#" >
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
   ]>

<rdf:RDF
  xmlns     ="&owl;"
  xmlns:owl ="&owl;"
  xml:base  ="http://www.w3.org/2002/07/owl"
  xmlns:rdf ="&rdf;"
  xmlns:rdfs="&rdfs;"
>

<Ontology rdf:about="">
  <imports rdf:resource="http://www.w3.org/2000/01/rdf-schema"/>
  <rdfs:isDefinedBy
rdf:resource="http://www.w3.org/TR/2004/REC-owl-semantics-2004021
0/" />
  <rdfs:isDefinedBy
rdf:resource="http://www.w3.org/TR/2004/REC-owl-test-20040210/"
/>
  <rdfs:isDefinedBy
rdf:resource="http://www.w3.org/TR/2004/REC-owl-features-20040210
/" />
  <rdfs:comment>This file specifies in RDF Schema format the
    built-in classes and properties that together form the basis of
    the RDF/XML syntax of OWL Full, OWL DL and OWL Lite.
    We do not expect people to import this file
    explicitly into their ontology. People that do import this file
    should expect their ontology to be an OWL Full ontology.
  </rdfs:comment>
  <versionInfo>10 February 2004</versionInfo>
</Ontology>

<rdfs:Class rdf:ID="Class">
  <rdfs:label>Class</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<Class rdf:ID="Thing">
  <rdfs:label>Thing</rdfs:label>
  <unionOf rdf:parseType="Collection">
    <Class rdf:about="#Nothing"/>
    <Class>
      <complementOf rdf:resource="#Nothing"/>
    </Class>
```

```
  </unionOf>
</Class>

<Class rdf:ID="Nothing">
  <rdfs:label>Nothing</rdfs:label>
  <complementOf rdf:resource="#Thing"/>
</Class>

<rdf:Property rdf:ID="equivalentClass">
  <rdfs:label>equivalentClass</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="&rdfs;subClassOf"/>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="disjointWith">
  <rdfs:label>disjointWith</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="equivalentProperty">
  <rdfs:label>equivalentProperty</rdfs:label>
  <rdfs:subPropertyOf rdf:resource="&rdfs;subPropertyOf"/>
</rdf:Property>

<rdf:Property rdf:ID="sameAs">
  <rdfs:label>sameAs</rdfs:label>
  <rdfs:domain rdf:resource="#Thing"/>
  <rdfs:range rdf:resource="#Thing"/>
</rdf:Property>

<rdf:Property rdf:ID="differentFrom">
  <rdfs:label>differentFrom</rdfs:label>
  <rdfs:domain rdf:resource="#Thing"/>
  <rdfs:range rdf:resource="#Thing"/>
</rdf:Property>

<rdfs:Class rdf:ID="AllDifferent">
  <rdfs:label>AllDifferent</rdfs:label>
</rdfs:Class>

<rdf:Property rdf:ID="distinctMembers">
  <rdfs:label>distinctMembers</rdfs:label>
  <rdfs:domain rdf:resource="#AllDifferent"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdf:Property rdf:ID="unionOf">
  <rdfs:label>unionOf</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdf:Property rdf:ID="intersectionOf">
  <rdfs:label>intersectionOf</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

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

```
  <rdfs:label>complementOf</rdfs:label>
  <rdfs:domain rdf:resource="#Class"/>
  <rdfs:range rdf:resource="#Class"/>
</rdf:Property>

<rdf:Property rdf:ID="oneOf">
  <rdfs:label>oneOf</rdfs:label>
  <rdfs:domain rdf:resource="&rdfs;Class"/>
  <rdfs:range rdf:resource="&rdf;List"/>
</rdf:Property>

<rdfs:Class rdf:ID="Restriction">
  <rdfs:label>Restriction</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Class"/>
</rdfs:Class>

<rdf:Property rdf:ID="onProperty">
  <rdfs:label>onProperty</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&rdf;Property"/>
</rdf:Property>

<rdf:Property rdf:ID="allValuesFrom">
  <rdfs:label>allValuesFrom</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>

<rdf:Property rdf:ID="hasValue">
  <rdfs:label>hasValue</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
</rdf:Property>

<rdf:Property rdf:ID="someValuesFrom">
  <rdfs:label>someValuesFrom</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&rdfs;Class"/>
</rdf:Property>

<rdf:Property rdf:ID="minCardinality">
  <rdfs:label>minCardinality</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="maxCardinality">
  <rdfs:label>maxCardinality</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Property>

<rdf:Property rdf:ID="cardinality">
  <rdfs:label>cardinality</rdfs:label>
  <rdfs:domain rdf:resource="#Restriction"/>
  <rdfs:range rdf:resource="&xsd;nonNegativeInteger"/>
</rdf:Property>

<rdfs:Class rdf:ID="ObjectProperty">
  <rdfs:label>ObjectProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="DatatypeProperty">
  <rdfs:label>DatatypeProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdf:Property rdf:ID="inverseOf">
  <rdfs:label>inverseOf</rdfs:label>
  <rdfs:domain rdf:resource="#ObjectProperty"/>
  <rdfs:range rdf:resource="#ObjectProperty"/>
</rdf:Property>

<rdfs:Class rdf:ID="TransitiveProperty">
  <rdfs:label>TransitiveProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>

<rdfs:Class rdf:ID="SymmetricProperty">
  <rdfs:label>SymmetricProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="#ObjectProperty"/>
</rdfs:Class>

<rdfs:Class rdf:ID="FunctionalProperty">
  <rdfs:label>FunctionalProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:ID="InverseFunctionalProperty">
  <rdfs:label>InverseFunctionalProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&owl;ObjectProperty"/>
</rdfs:Class>

<rdfs:Class rdf:ID="AnnotationProperty
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<AnnotationProperty rdf:about="&rdfs;label"/>
<AnnotationProperty rdf:about="&rdfs;comment"/>
<AnnotationProperty rdf:about="&rdfs;seeAlso"/>
<AnnotationProperty rdf:about="&rdfs;isDefinedBy"/>

<rdfs:Class rdf:ID="Ontology">
  <rdfs:label>Ontology</rdfs:label>
</rdfs:Class>

<rdfs:Class rdf:ID="OntologyProperty">
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdf:Property rdf:ID="imports">
  <rdfs:label>imports</rdfs:label>
  <rdf:type rdf:resource="#OntologyProperty"/>
  <rdfs:domain rdf:resource="#Ontology"/>
  <rdfs:range rdf:resource="#Ontology"/>
</rdf:Property>

<rdf:Property rdf:ID="versionInfo">
  <rdfs:label>versionInfo</rdfs:label>
  <rdf:type rdf:resource="#AnnotationProperty"/>
</rdf:Property>
```

```
<rdf:Property rdf:ID="priorVersion">
  <rdfs:label>priorVersion</rdfs:label>
  <rdf:type rdf:resource="#OntologyProperty"/>
  <rdfs:domain rdf:resource="#Ontology"/>
  <rdfs:range rdf:resource="#Ontology"/>
</rdf:Property>

<rdf:Property rdf:ID="backwardCompatibleWith">
  <rdfs:label>backwardCompatibleWitesh</rdfs:label>
  <rdf:type rdf:resource="#OntologyProperty"/>
  <rdfs:domain rdf:resource="#Ontology"/>
  <rdfs:range rdf:resource="#Ontology"/>
</rdf:Property>

<rdf:Property rdf:ID="incompatibleWith">
  <rdfs:label>incompatibleWith</rdfs:label>
  <rdf:type rdf:resource="#OntologyProperty"/>
  <rdfs:domain rdf:resource="#Ontology"/>
  <rdfs:range rdf:resource="#Ontology"/>
</rdf:Property>

<rdfs:Class rdf:ID="DeprecatedClass">
  <rdfs:label>DeprecatedClass</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdfs;Class"/>
</rdfs:Class>

<rdfs:Class rdf:ID="DeprecatedProperty">
  <rdfs:label>DeprecatedProperty</rdfs:label>
  <rdfs:subClassOf rdf:resource="&rdf;Property"/>
</rdfs:Class>

<rdfs:Class rdf:ID="DataRange">
  <rdfs:label>DataRange</rdfs:label>
</rdfs:Class>

</rdf:RDF>
```

# 附录 C. OWL 快速参考

OWL 词汇表中的类:

| rdfs:Class |
| --- |
| owl:AllDifferent |
| owl:AnnotationProperty |
| owl:Class |
| owl:DataRange |
| owl:DatatypeProperty |
| owl:DeprecatedClass |
| owl:DeprecatedProperty |

| |
|---|
| owl:FunctionalProperty |
| owl:InverseFunctionalProperty |
| owl:Nothing |
| owl:ObjectProperty |
| owl:Ontology |
| owl:OntologyProperty |
| owl:Restriction |
| owl:SymmetricProperty |
| owl:Thing |
| owl:TransitiveProperty |

OWL 词汇表中的属性:

| rdf:Property | rdfs:domain | rdfs:range |
|---|---|---|
| owl:allValuesFrom | owl:Restriction | rdfs:Class |
| owl:backwardCompatibleWith | owl:Ontology | owl:Ontology |
| owl:cardinality | owl:Restriction | xsd:nonNegativeInteger |
| owl:complementOf | owl:Class | owl:Class |
| owl:differentFrom | owl:Thing | owl:Thing |
| owl:disjointWith | owl:Class | owl:Class |
| owl:distinctMembers | owl:AllDifferent | rdf:List |
| owl:equivalentClass | owl:Class | owl:Class |
| owl:equivalentProperty | rdf:Property | rdf:Property |
| owl:hasValue | owl:Restriction | |
| owl:imports | owl:Ontology | owl:Ontology |
| owl:incompatibleWith | owl:Ontology | owl:Ontology |
| owl:intersectionOf | owl:Class | rdf:List |
| owl:inverseOf | owl:ObjectProperty | owl:ObjectProperty |
| owl:maxCardinality | owl:Restriction | xsd:nonNegativeInteger |
| owl:minCardinality | owl:Restriction | xsd:nonNegativeInteger |
| owl:oneOf | owl:Class | rdf:List |
| owl:onProperty | owl:Restriction | rdf:Property |
| owl:priorVersion | owl:Ontology | owl:Ontology |
| owl:sameAs | owl:Thing | owl:Thing |
| owl:someValuesFrom | owl:Restriction | rdfs:Class |

| | | |
|---|---|---|
| owl:unionOf | owl:Class | rdf:List |
| owl:versionInfo | | |

# 附录 D. 相对于 DAML+OIL 的改变

本节总结了从 DAML+OIL [*DAML+OIL*] 到 OWL 的变化。

1. 语义有重大改变。对于三个子语言而言，DAML+OIL 的语义最接近 OWL DL 的语义。
2. 命名空间改为 http://www.w3.org/2002/07/owl。
3. 从 RDF Core Working Group 以来对 RDF 和 RDF Schema 的不同更新都被合并进来，包括
    4.
    - 允许循环子类
    - 多重 `rdfs:domain` 和 `rdfs:range` 属性作为交集处理
    - RDF 语义 [*RDF Semantics*]
    - 数据类型
        -
        - RDF 和 OWL 使用 XML Schema 命名空间 http://www.w3.org/2001/XMLSchema 而不是 http://www.w3.org/2000/10/XMLSchema。
        - OWL 不支持将数据类型作为类型（types）使用，例如
            - `<size>`
            - `  <xsd:integer rdf:value="10"/>`
            - `</size>`

            而采用

            ```
            <size
            rdf:datatype="http://www.w3.org/2001/XMLSchema#
            integer">10</size>
            ```

    3.
    - 用来表示闭集（closed collections）的 `daml:List` 结构大部分并入 RDF
        -
        - rdf:parseType="Collection" 代替 `rdf:parseType="daml:collection"`
        - `rdf:List`, `rdf:first`, `rdf:rest` 和 `rdf:nil` 代替 `daml:List`, `daml:first`, `daml:rest` 和 `daml:nil`
        - 不支持 `daml:item`。因为这个特性主要用来创建类型化列表，我们在这里包含了一个不使用 `daml:item` 创建这样一个列表的例子：
            - `<rdfs:Class rdf:ID="OperaList">`
            - `  <rdfs:subClassOf rdf:resource="&rdf;List"/>`
            - `  <rdfs:subClassOf>`
            - `  <owl:Restriction>`
            - `  <owl:onProperty rdf:resource="&rdf;first"/>`
            - `  <owl:allValuesFrom rdf:resource="#Opera"/>`
            - `  </owl:Restriction>`
            - `  </rdfs:subClassOf>`
            - `  <rdfs:subClassOf>`

```
            <owl:Restriction>
              <owl:onProperty
rdf:resource="&rdf;rest"/>
              <owl:allValuesFrom
rdf:resource="#OperaList"/>
            </owl:Restriction>
          </rdfs:subClassOf>
        </rdfs:Class>
```

这个例子定义了一个列表，其元素大部分是歌剧。这通过两个限制来实现，一个针对 rdf:first 值（表示列表元素的类型），另一个针对 rdf:rest 值（它应该是被定义列表的名称）。

4. 受限限制（qualified restrictions）被从该语言中删除，导致以下属性的删除：
   5.
   - daml:cardinalityQ
   - daml:hasClassQ
   - daml:maxCardinalityQ
   - daml:minCardinalityQ
6. 许多属性和类被重命名，如下表所示：

| DAML+OIL | OWL |
|---|---|
| daml:differentIndividualFrom | owl:differentFrom |
| daml:equivalentTo | owl:sameAs |
| daml:sameClassAs | owl:equivalentClass |
| daml:samePropertyAs | owl:equivalentProperty |
| daml:hasClass | owl:someValuesFrom |
| daml:toClass | owl:allValuesFrom |
| daml:UnambiguousProperty | owl:InverseFunctionalProperty |
| daml:UniqueProperty | owl:FunctionalProperty |

6. 增加 owl:SymmetricProperty。
7. 增加 owl:AnnotationProperty, owl:OntologyProperty 和 owl:DataRange。
8. 在 OWL Full 中一个 owl:DatatypeProperty 可以是一个 owl:InverseFunctionalProperty 。
9. RDF 和 RDF Schema 的类和属性的同义词被从该语言中删除，导致以下删除：
   10.
   - daml:comment
   - daml:domain
   - daml:label
   - daml:isDefinedBy
   - daml:Literal
   - daml:Property
   - daml:range
   - daml:seeAlso
   - daml:subClassOf
   - daml:subPropertyOf
   - daml:type
   - daml:value
11. daml:disjointUnionOf 被从该语言中删除，因为它可以使用 owl:unionOf 或 rdfs:subClassOf 和 owl:disjointWith 实现。
12. daml:equivalentTo 被重命名为 owl:sameAs，并且不再是 owl:equivalentClass 和 owl:equivalentProperty 的父属性。

13. 加入以下属性和类以支持版本管理：
    14.
    - owl:backwardCompatibleWith
    - owl:DeprecatedClass
    - owl:DeprecatedProperty
    - owl:incompatibleWith
    - owl:priorVersion
15. 增加 owl:AllDifferent 和 owl:distinctMembers 处理唯一命名假设（Unique Names Assumption）。

# 附录 E. OWL DL 本体的经验法则

“OWL 语义和抽象语法”文档 [*OWL S&AS*] 从抽象语法以及到 RDF 的映射的角度提供了 OWL 本体的特性描述。

以下规则给出了一个 RDF 图是一个 DL 本体的条件的非形式化特性描述。这并不是想要代替 S&AS 中给出的特性描述，而是想提供一些大体上的建议——想法就是如果你遵守这些法则，你就更容易创建一个 OWL DL 本体。给出这些规则也不是想告诉你如何把三元组表示转换成更接近于抽象语法的某种东西。

## 不要搞糟词汇表

内置的属性和类不应该重新定义。通常这意味着在 OWL，RDF 和 RDFS 命名空间中的东西不应该作为三元组的主体出现。

## 提供显式的类型定义（typing）

任何事物都应该有一个类型 [1]。如果任一 URI 引用用在一个希望类出现的地方，那么该图应该包含一个三元组声明

```
x rdf:type owl:Class
```

类似地，如果一个属性 `p` 用在一个希望对象属性出现的地方，那么应该有一个三元组 [2]

```
p rdf:type owl:ObjectProperty
```

如果一个属性 `q` 用在一个希望数据类型属性出现的地方，那么应该有一个三元组

```
q rdf:type owl:DatatypeProperty
```

如果一个属性 `o` 用在一个希望本体属性出现的地方，那么它应该要么是内置本体属性之一(`owl:imports`, `owl:priorVersion`, `owl:backwardCompatibleWith`, 和 `owl:incompatibleWith`) ，要么有一个三元组：

```
o rdf:type owl:OntologyProperty
```

如果一个属性 a 用在一个希望注释属性出现的地方，那么它应该要么是内置注释属性之一（owl:versionInfo, rdfs:label, rdfs:comment, rdfs:seeAlso 和 rdfs:isDefinedBy），要么有一个三元组：

```
a rdf:type owl:AnnotationProperty
```

出现在本体中的任何个体都应该至少指定一个类型，即对于一个个体 i，必须有一个三元组：

```
i rdf:type c
```

其中 c 是一个 owl:Class 或 owl:Restriction。

## 保持名称的独立

类、属性（对象，数据类型，本体和注释）以及个体的 URI 引用都应该是互斥的。因此我们不能有下面这样的东西：

```
x rdf:type owl:Class
x rdf:type owl:ObjectProperty
```

特别地，这意味着我们不能把类用作实例，即

```
x rdf:type owl:Class
y rdf:type owl:Class
x rdf:type y
```

在 OWL DL 中是无效的。在这里一般规则是，如果在该图中存在一个节点 x 有三元组：

```
x rdf:type owl:Class
```

那么 x 不应该作为任何其他有谓词 rdf:type 的三元组的主体出现。[3]

## 限制

如果一个节点 x 有 rdf:type owl:Restriction，那么应该如下所示：

- 它是一个空节点（即不具名）。
- 它不是任何其他谓词是 rdf:type 的三元组的主体 [3]。
- 它是仅仅一个谓词是 owl:onProperty 的三元组的主体，该三元组的客体是一个 owl:ObjectProperty 或 owl:DatatypeProperty。
- 它是仅仅一个以下三元组的主体：
  -

    - 有谓词 owl:someValuesFrom 的三元组。在这种情况下，owl:onProperty 三元组的客体的属性类型应该是适当的。这意味着如果这个三元组的客体是一个数据类型，这个属性就应该是一个 owl:DatatypeProperty。如果这个客体是一个类运算式，该属性就应该是一个 owl:ObjectProperty。这种类型定义信息应该展现出来（由于以上列出的限制）。
    - 有谓词 owl:allValuesFrom 的三元组。类似的限制对 owl:someValuesFrom 也成立。
    - 有谓词 owl:hasValue 的三元组。如果包含在 owl:onProperty 三元组中的属性的类型是 owl:ObjectProperty，那么这个三元组的客体就应该是一个个体。如果包含在 owl:onProperty 三元组中的属性的类型是 owl:DatatypeProperty，那么这个三元组的客体就应该是数据文本。
    - 有谓词 owl:minCardinality 的三元组。这个三元组的客体应该是一个数据文本，表示一个非负整数。
    - 有谓词 owl:maxCardinality 的三元组。限制同 owl:minCardinality。
    - 有谓词 owl:cardinality 的三元组。限制同 owl:minCardinality。
- 任何其他 x 是其主体的三元组都应该有谓词 owl:equivalentClass 或 owl:disjointWith。

# 类公理

对于任何有谓词 rdfs:subClassOf 或 owl:equivalentClass 或 owl:disjointWith 的三元组，其主体和客体都应该是一个 owl:Class 或 owl:Restriction，即，如果有

```
x rdfs:subClassOf y
```

那么该图就必须包含以下之一：

```
x rdf:type owl:Class
```

或

```
x rdf:type owl:Restriction.
```

以及以下之一

```
y rdf:type owl:Class
```

或

```
y rdf:type owl:Restriction.
```

# 属性公理

对于任何有谓词 rdfs:subPropertyOf 或 owl:equivalentProperty 的三元组，其主体和客体应该有同样的类型（owl:ObjectProperty 或 owl:DatatypeProperty 之一）。即，如果有

```
p owl:equivalentProperty q
```

那么该图必须包含

```
p rdf:type owl:ObjectProperty
q rdf:type owl:ObjectProperty.
```

或

```
p rdf:type owl:DatatypeProperty
q rdf:type owl:DatatypeProperty.
```

有谓词 rdfs:domain 的三元组应该有一个 owl:ObjectProperty 或 owl:DatatypeProperty 作为其主体，有一个 owl:Class 或 owl:Restriction 作为其客体。

有谓词 rdfs:range 的三元组应该有一个 owl:ObjectProperty 或 owl:DatatypeProperty 作为其主体。对于前者，其客体应该是一个 owl:Class 或 owl:Restriction。对于后者，其客体应该是一个 XML Schema 类型，rdfs:Literal，或一个 owl:oneof，指定类型为 owl:DataRange 的数据值域。

一个 owl:inverseOf 三元组的主体和客体都必须有类型 owl:ObjectProperty。

# 个体公理

对于任何有谓词 owl:sameAs 或 owl:differentFrom 的三元组，其主体和客体都必须是个体。

注意：用 owl:sameAs 关联两个类与用 owl:equivalentClass 关联两个类是很不相同的事情。前者意味着这两个对象事实上是相同的，实际上是类作为实例的一个例子，从而将该本体推出 OWL DL 之外。 后者是一个断言：这两个类的外延（例如成员集合）是等同的。

类似地，用 owl:differentFrom 关联类与用 owl:disjointWith 关联类（又是一个 OWL Full 结构的例子）也不是一回事。两个类可以是不同的对象，但仍然共享相同的外延。

如果一个节点 x 有 rdf:type owl:AllDifferent，那么应该如下所述：

- 它是一个空节点（即未具名）。
- 它是一个谓词是 owl:distinctMembers 的三元组的主体，其客体应该是（良好构造的）rdf:List，其所有元素都是个体。
- 它不是任何其他三元组的主体（或客体）。

## 布尔类表达式

布尔运算符 (and, or, not) 在 OWL 中用 owl:intersectionOf，owl:unionOf 和 owl:complementOf 来表示。

一个 owl:complementOf 三元组的主体必须是一个 owl:Class，客体必须是一个 owl:Class 或 owl:Restriction。

一个 owl:unionOf 或 owl:intersectionOf 三元组的主体必须是一个 owl:Class，客体必须是一个（良好构造的）rdf:List，其所有元素都必须是 owl:Class 或 owl:Restriction。这可以显式地用扩展的 rdf:Lists 来表示，或者如果采用的是 RDF-XML，则用一个 rdf:parseType="Collection" 属性来表示。

```
<owl:Class>
 <owl:intersectionOf rdf:parseType="Collection">
  <owl:Class rdf:about="x"/>
  <owl:Class rdf:about="y"/>
 </owl:intersectionOf>
</owl:Class>
```

如果 owl:Class 是一个空节点（即这个类未具名），那么它可以只是最多一个有谓词 owl:intersectionOf，owl:unionOf 或 owl:complementOf 的三元组的主体。如果这个类是具名的，任意数量的这种三元组都是允许的。

## 枚举

有谓词 owl:oneOf 的任一三元组的主体都必须是一个 owl:Class 或 owl:DataRange。对于前者，客体必须是一个（良好构造的）rdf:List，其所有元素都是个体。对于后者，客体必须是一个（良好构造的）rdf:List，其所有元素都是数据文本。当存在布尔运算符时，还是可以使用 rdf:parseType="Collection"。

## 本体和注释属性

有一个本体谓词的任一三元组，其主体和客体都应该是一个本体，例如节点 x 存在一个三元组：

```
x rdf:type owl:Ontology
```

有一个注释谓词的任一三元组，其主体应该是一个具名（即非空节点）类，一个属性，一个个体或一个本体。有一个注释谓词的三元组的客体应该是一个个体，一个数据文本或一个任意的 URI 引用。

作为 OWL，RDF 和 RDFS 词汇表中谓词的例外，注释属性是仅有的应该出现在有一个类或属性作为其主体的三元组中的谓词。

注释属性和本体属性本身应该是类型化的，并且不应该作为三元组的主体或客体出现，除非是作为一个有谓词 `rdf:type` 或一个注释属性的三元组的主体出现。

## 避免结构共享

通常 OWL 的 S&AS 描述不允许 RDF 表示中存在结构共享（*structure sharing*）。这实际上意味着 RDF 图中表示一个特定描述的一个匿名节点应该只出现一次（作为一个三元组的客体）。因此，像：

```
x1 rdf:type owl:Class
x1 rdfs:subClassOf _:y
x2 rdf:type owl:Class
x2 rdfs:subClassOf _:y
_:y rdf:type owl:Class
_:y owl:complementOf z
```

这样的东西应该避免。在一些棘手的角落情况下这是*允许*的。但是一般情况下，无论何时一个类运算式被用在多个地方，RDF 图都应该使用不同的空节点。

## 避免孤立的空节点

通常情况下空节点出现在图中，要么表示未具名个体，要么应该是下述情况中的一种：

- 一个 `rdfs:subClassOf`, `owl:equivalentClass`, `owl:disjointWith`, `owl:someValuesFrom`, `owl:allValuesFrom` 或 `rdf:type` 三元组的客体。
- 一个 `rdf:type` 三元组的主体，客体是 `owl:AllDifferent`。
- 一个 `rdf:List` 中的一个元素。

孤立空节点（orphan blank nodes），即不是一个三元组的客体的空节点，通常是不允许的（除非是在上面描述的 `owl:AllDifferent` 情况下）。

## 基本事实

本体可以包含基本事实（ground facts）的断言（例如断言个体的属性的三元组）。用于这些断言的属性必须是一个 `owl:ObjectProperty` 或

owl:DatatypeProperty。任何这种三元组的主体都必须是一个个体（应该是类型化的）。客体可以是对一个个体的引用（如果该属性是 owl:ObjectProperty）或数据文本（如果该属性是 owl:DatatypeProperty）。

## OWL Lite

OWL Lite 文档应该遵循与 OWL DL 文档相同的规则，另外还有许多额外限制，主要涉及允许使用的词汇。OWL Lite 文档不应该使用以下任何一个词汇：

- owl:unionOf
- owl:complementOf
- owl:oneOf
- owl:hasValue
- owl:disjointWith

有谓词 owl:equivalentClass 的三元组，作为其客体或主体的任何对象都不应该是空节点。

有谓词 owl:minCardinality, owl:maxCardinality 或 owl:cardinality 的任何三元组的客体，都应该是一个数据文本，表示整数 0 或 1。

在 OWL Lite 中涉及 owl:intersectionOf 使用的情况稍微复杂些。这个谓词不应该用来构成任意的表达式，但是在表示完整类定义时是需要的。上述限制告诉我们，任何有谓词 owl:intersectionOf 的三元组的主体都应该是一个 owl:Class。在 OWL Lite 中有更进一步的限制，这个类应该是具名的，即主体不能是空节点。

## 其他

要小心使用 owl:Thing。例如，下面的 OWL-RDF 片断：

```
<owl:Class rdf:about="#A">
  <rdfs:subClassOf>
    <owl:Thing/>
  </rdfs:subClassOf>
</owl:Class>
```

并没有描述类 A 是 owl:Thing 的子类，实际上描述了类 A 是 owl:Thing 的某个匿名实例的子类。因此这是在把类用作实例，已超出了 OWL DL 的范围。所需的 owl:Thing 子类的效果可以通过：

```
<owl:Class rdf:about="#A">
  <rdfs:subClassOf>
    <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
  </rdfs:subClassOf>
</owl:Class>
```

来获得。要小心不要混淆 owl:Class 和 rdfs:Class。由于没有给 c 一个合适的类型，下面这个不属于 OWL DL。

```
c rdf:type rdfs:Class
```

## 注释

[1] 必须为任何事物进行类型定义的要求，当然不适用于来自 OWL, RDF 或 RDFS 命名空间的事物。

[2] 严格来讲，如果属性定义成一个 owl:TransitiveProperty，owl:SymmetricProperty 或 owl:InverseFunctionalProperty，这就没有必要。

[3] 在这里一个例外是我们可以有：

```
x rdf:type rdfs:Class
x rdf:type owl:Class
p rdf:type rdf:Property
p rdf:type owl:ObjectProperty
```

或

```
q rdf:type rdf:Property
q rdf:type owl:DatatypeProperty
```

另外，对于限制，我们可以有：

```
x rdf:type owl:Restriction
x rdf:type rdfs:Class
x rdf:type owl:Class
```

# 附录 F. 自 PR 之后的变动日志

1. 删除附录 E 中的似是而非的尾注[4]。
2. 在 Jacco van Ossenbruggen 评注之后，在附录 B(RDF Schema of OWL) 中给 AnnotationProperty 和 OntologyProperty 增加了 rdfs:label 元素。
3. 修复断开的章节参考以及更正章节参考格式。
4. 标准化参考文献章节。
5. 在 Minsu Jang 评注之后，对 rdf:RDF 元素的需求描述进行了编辑修改。
6. 针对 Lacy 所作的评论作了若干编辑修改。
7. 在一些公开评注之后（例如，见 Benjamin Nowack 的评注），对 Sec. 7.1 中关于 OWL DL 对注释属性的使用的约束增加了解释性文字。也在 Sec. 7.2 中增加了一个句子，指出同样的约束对本体属性也有效。
8. 在编辑最后通读之后进行了若干小的编辑性更正。

# 参考文献


**[OWL Overview]**
*OWL Web Ontology Language Overview*, Deborah L. McGuinness and Frank van Harmelen, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-features-20040210/ . Latest version available at http://www.w3.org/TR/owl-features/ .

**[OWL Guide]**
*OWL Web Ontology Language Guide*, Michael K. Smith, Chris Welty, and Deborah L. McGuinness, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-guide-20040210/ . Latest version available at http://www.w3.org/TR/owl-guide/ .

**[OWL Semantics and Abstract Syntax]**
*OWL Web Ontology Language Semantics and Abstract Syntax*, Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/ . Latest version available at http://www.w3.org/TR/owl-semantics/ .

**[OWL Test]**
*OWL Web Ontology Language Test Cases*, Jeremy J. Carroll and Jos De Roo, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-test-20040210/ . Latest version available at http://www.w3.org/TR/owl-test/ .

**[OWL Requirements]**
*OWL Web Ontology Language Use Cases and Requirements*, Jeff Heflin, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-webont-req-20040210/ . Latest version available at http://www.w3.org/TR/webont-req/ .

**[RDF Concepts]**
*Resource Description Framework (RDF): Concepts and Abstract Syntax*, Graham Klyne and Jeremy J. Carroll, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ . Latest version available at http://www.w3.org/TR/rdf-concepts/ .

**[RDF Syntax]**
*RDF/XML Syntax Specification (Revised)*, Dave Beckett, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/ . Latest version available at http://www.w3.org/TR/rdf-syntax-grammar/ .

**[RDF Semantics]**
*RDF Semantics*, Pat Hayes, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ . Latest version available at http://www.w3.org/TR/rdf-mt/ .

**[RDF Vocabulary]**
*RDF Vocabulary Description Language 1.0: RDF Schema*, Dan Brickley and R. V. Guha, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ . Latest version available at http://www.w3.org/TR/rdf-schema/ .

**[DAML+OIL]**
*DAML+OIL (March 2001) Reference Description*. Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. W3C Note 18 December 2001. Latest version is available at `http://www.w3.org/TR/daml+oil-reference`.

**[XML-SCHEMA2]**
*XML Schema Part 2: Datatypes - W3C Recommendation*, World Wide Web Consortium, 2 May 2001.

## 2.B　OWL2 Web 本体语言文档概述

本文档《OWL2 Web 本体语言文档概述》是 W3C 发布的 ***OWL 2 Web Ontology Language Document Overview***（2009-10-27）的中文译本。文中若存在译法不当和错误之处，欢迎批评指正，请发邮件至：**zengxh@szu.edu.cn**，谢谢！

**翻译说明：**

- 本文档的英文版是唯一正式版本。此处的中文译本仅供学习与交流。
- 中文译本的内容是非正式的，仅代表译者的个人观点。
- 中文译本的内容会根据反馈意见随时进行修订。
- 中文译本同时通过 W3C Translations 网站发布。
- 转载本文，请注明译者和原链接。

**译者：**

曾新红（Xinhong Zeng），深圳大学图书馆 NKOS 研究室

蔡庆河（Qinghe Cai），深圳大学计算机与软件学院

**资助声明：**

本次翻译工作得到广东省哲学社会科学“十一五”规划项目（批准号：GD10CTS02）和国家社科基金项目“中文知识组织系统的形式化语义描述标准体系研究”（批准号：12BTQ045）的资助。

翻译时间：2011 年 8 月

发布时间：2012 年 9 月 21 日

---

# OWL2 Web 本体语言
# 文档概述

## W3C 推荐标准 2009 年 10 月 27 日

**当前版本：**

http://www.w3.org/TR/2009/REC-owl2-overview-20091027/

**最新版本（OWL2 系列）：**

http://www.w3.org/TR/owl2-overview/

**最新推荐标准：**

http://www.w3.org/TR/owl-overview

**上一版本：**

http://www.w3.org/TR/2009/PR-owl2-overview-20090922/ （彩色标注不同之处）

**编者：**

W3C OWL 工作组（见致谢）

请参阅本文档的勘误表，那里可能会有一些规范的校正。

本文档也可以以如下的非规范格式查看：PDF 版本。

另见：译文。



# 摘要

OWL2 Web 本体语言，或者简略地记作 OWL2，是一个本体语言，用于带有形式化定义含义的语义网。OWL2 本体提供了类、属性、个体及数据值，以语义网文档的形式存储。OWL2 本体可以与以 RDF 记载的信息结合使用，并且 OWL2 本体本身也主要是以 RDF 文档的形式进行交换。

本文档是对 OWL2 以及其他不同 OWL2 文档的介绍。它阐述了 OWL2 的语法，不同类型的语义，可用配置语言（profiles）（子语言），以及 OWL1 与 OWL2 之间的关系。

# 本文档的状态

### 可能已被替代

*本节描述的是本文档在发布时的状态。其他的文档可能替代了该文档。在 http://www.w3.org/TR/的 W3C technical reports index 中，可以找到当前的 W3C 出版物列表和该技术报告的最新版本。*

### XML Schema 数据类型依赖

根据定义，OWL2 使用的是 XML Schema Definition Language (XSD)中定义的数据类型。对本文档中的 OWL2 而言， XSD 的最新 W3C 推荐标准是版本 1.0，此时，版本 1.1 正在向推荐标准演进。OWL2 的设计已经利用了 XSD1.1 中的新数据类型和更清晰的注释，但是目前这些利用有一部分暂时搁置。特别地，OWL2 中

基于 XSD1.1 的那些元素将被当作是*可选的*，直到 XSD1.1 成为 W3C 的推荐标准为止，详见 Conformance, section 2.3。等到 XSD1.1 发布为 W3C 的推荐标准时，这些元素才会终止可选状态而与其它指定元素一样进入必备状态。

我们提议，目前开发人员和用户遵循 XSD 1.1 Candidate Recommendation。根据 Schema 和 OWL 工作组之间的讨论，当 XSD1.1 演进成为推荐标准时，我们并不希望任何实现会有必须的改动。

### 未变的文档

本文档与先前版本并没有主体上的改变。关于更早版本的变动细节，请见 change log。

### 请发表意见

请将意见发送至 public-owl-comments@w3.org（公开文档）。虽然由 OWL 工作组执笔的本文档已经完成，但是您的意见依旧可能在勘误表或者未来的修订版中得到解决。欢迎开发人员在 public-owl-dev@w3.org(公开文档)公开讨论。

### 由 W3C 批准

*本文档已经由 W3C 成员、软件开发人员以及其他 W3C 小组和兴趣组织审查，并由 W3C 主管（Director）批准成为 W3C 推荐标准。这是一个稳定的文档，可以作为参考资料或被其他文档引用。W3C 在制作推荐标准过程中担任的角色，是要引起人们对该规范的注意并促进它的广泛应用。这提升了 Web 的功能性和交互性。*

### 专利

*本文档由遵循 5 February 2004 W3C Patent Policy 的小组完成。它只是一个提供信息的文档。W3C 维护着一个专利披露公开列表（public list of any patent disclosures），它与该组织的可交付成果一起制作；此页面也包含披露专利的说明。*

## 目录

# 1. 引言

本文档对 OWL2 Web 本体语言作了非规范性高层次概述，作为定义和描述 OWL2 的文档的路线图。

本体是形式化的术语词汇表，通常覆盖一个特定的领域并为某个用户社区所共享。它们通过描述本体中术语间的关系来给术语下定义。OWL2 是 OWL Web Ontology Language 的扩展和修订版，OWL 由 W3C Web 本体工作组（W3C Web Ontology Working Group）开发并于 2004 年发布（今后称为“OWL1”）。OWL2 是由后续小组，即 W3C OWL 工作组（W3C OWL Working Group）开发的（本文档也由该小组撰写）。与 OWL1 一样，OWL2 也是为便于 Web 上的本体开发与共享而设计的，它的最终目标是使得 Web 上的内容能够更容易地被机器访问。

# 2. 概述

图 1 给出了 OWL2 语言的一个概貌，展示了它主要的构成模块以及彼此之间的关系。中心的椭圆是本体的抽象符号，它可以被想象成一个抽象的结构或者一个 RDF 图（见 2.1 本体）。顶部是不同的具体语法（见 2.2 语法），它们可以用来序列化及交换本体。底部是两个语义规范，它们定义了 OWL2 本体的含义（见 2.3 语义）。

OWL2 的大部分用户只需要一种语法和一种语义；对他们而言，该图表会变得更简单一些，只需要顶部的一种语法，底部的一种语义，并且几乎不需要看到中心椭圆内部的内容。

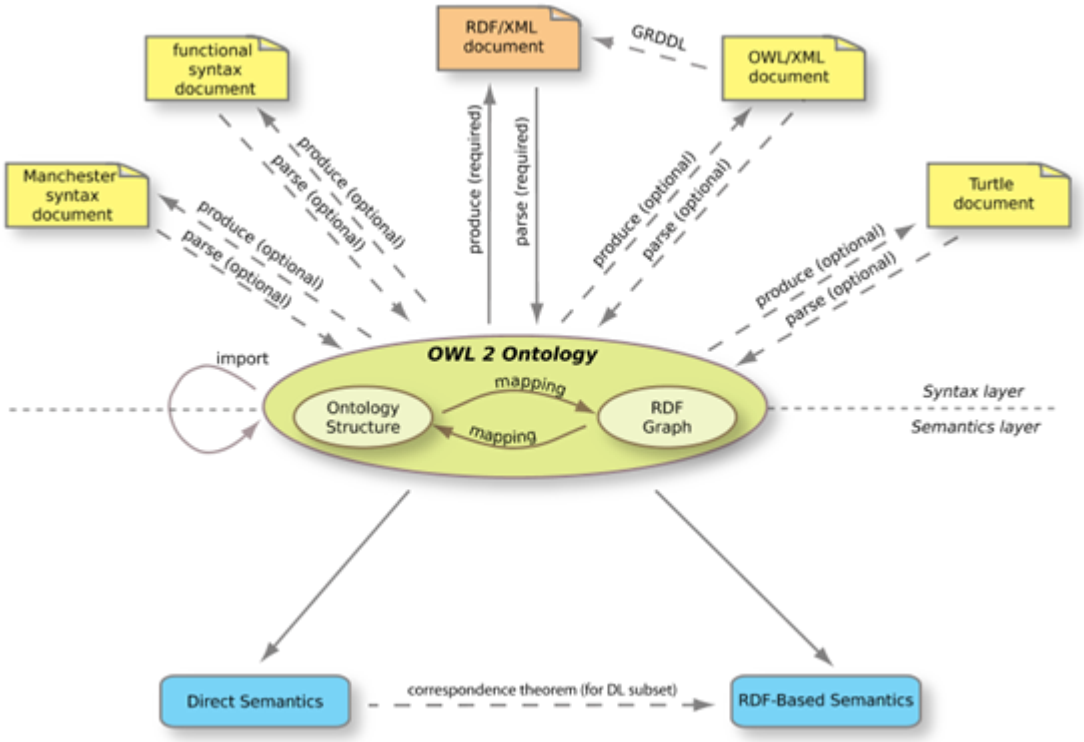


图 1. OWL2 的结构

## 2.1 本体

“OWL2 结构化规范”文档[*OWL 2 Structural Specification*]中定义了 OWL2 本体的概念性结构。它使用 UML[UML]来定义 OWL2 中可用的结构化元素，使用抽象术语解释了它们所扮演的角色和发挥的功能，未引用任何特别的语法。它也定义了函数式语法（functional-style syntax），该语法严格遵从结构化规范并允许以紧凑的方式书写 OWL2 本体。

任何的 OWL2 本体都可以视为是一个 RDF 图。两者之间的关系由“映射到 RDF 图”文档[*OWL 2 RDF Mapping*]定义，该文档定义了结构化格式与 RDF 图格式之间的相互映射。OWL2 快速参考指南[*OWL 2 Quick Guide*]，将 OWL2 的这两种展现形式一并列出，并做了简单概述。

## 2.2 语法

在实践中，需要一套具体的语法来存储 OWL2 中的本体并使之能够在工具与应用间交换。用于 OWL2 的主要交换语法是 RDF/XML[*RDF Syntax*]；它事实上是所有的 OWL2 工具都必须支持的唯一语法。（见“OWL2 一致性准则”文档[*OWL2 Conformance*]）

RDF/XML 提供了 OWL2 工具间的互操作，但是也可能需要用到其他的具体语法。这包括可供选择的其他 RDF 序列化语法，例如 Turtle[*Turtle*]，XML 序列化语法[*OWL 2 XML*]，以及更加“易读的”语法，即曼彻斯特语法[*OWL 2 Manchester Syntax*]，若干本体编辑工具都在使用它。最后要说的是，虽然函数式语法的主要

目的是指定语言的结构[*OWL 2 Structural Specification*]，但是它也可以用于序列化。

| 语法名 | 规范 | 状态 | 用途 |
|---|---|---|---|
| RDF/XML | Mapping to RDF Graphs, RDF/XML | 强制 | 交换(所有遵循 OWL2 的软件都可以用它读写) |
| OWL/XML | XML Serialization | 可选 | 更易于利用 XML 工具处理 |
| 函数式语法（Functional Syntax） | Structural Specification | 可选 | 更易于看到本体的形式化结构 |
| 曼彻斯特语法 | Manchester Syntax | 可选 | 更易于读/写 DL 本体 |
| Turtle | Mapping to RDF-Graphs, Turtle | 可选，非来自 OWL-WG | 更易于读/写 RDF 三元组 |

## 2.3 语义

“OWL2 结构化规范”文档定义了 OWL2 本体的抽象结构，但是并没有定义它们的含义。“直接语义”[*OWL 2 Direct Semantics*]和“基于 RDF 的语义”[*OWL 2 RDF-Based Semantics*]提供了给 OWL2 本体赋予语义的两种可选途径，同时也提供了描述两者之间联系的对应定理（correspondence theorem）。推理工具和其他工具会用到这两种语义，例如用它们来响应类一致性、包容性以及实例检索查询。

“直接语义”直接将含义赋予本体结构（construct），使得其与 SROIQ 描述逻辑的模型理论语义（model theoretic semantics）相兼容（SROIQ 是具有有用计算性能的一阶逻辑片段）。这种紧密联系的好处在于大量的描述逻辑文献和实现经验可以直接用于 OWL2 工具。不过，必须施加一些条件在本体结构上，以确保它们能够被转化为 SROIQ 知识库；例如，在数字限制中不能使用传递属性(在“OWL2 结构化规范”文档[*OWL 2 Structural Specification*]的 Section 3 可以看到一个这些条件的完整列表)。满足这些语法条件的本体称为 *OWL2 DL* 本体。“OWL2 DL” 用来简略地表示使用直接语义[*OWL 2 Direct Semantics*]解释的 OWL2 DL 本体。

“基于 RDF 的语义”[*OWL 2 RDF-Based Semantics*] 直接给 RDF 图赋予含义，也就间接地通过到 RDF 图的映射给本体结构赋予了含义。“基于 RDF 的语义”与“RDF 语义”[*RDF Semantics*]完全兼容，并扩展了 RDF 中定义的语义条件。基于 RDF 的语义可以不加限制地应用到*任何* OWL2 本体上去，这是由于任何 OWL2 本体都可以映射到 RDF。“OWL2 Full” 用来简略地表示被作为 OWL2 本体并使用基于 RDF 语义解释的 RDF 图。

在“基于 RDF 的语义”文档[*OWL 2 RDF-Based Semantics*]的 Section 7.2 中，对应定理（correspondence theorem）定义了直接语义与基于 RDF 的语义之间的精确而紧密的关系。该定理实质上阐述了这样一个事实：给定一个 OWL 2 DL 本体，如果该本体被映射成 RDF 图并且使用基于 RDF 的语义进行解释，使用直接语义得出的推论将仍然是有效的。

## 2.4 配置语言

OWL2 配置语言 [*OWL 2 Profiles*] 是 OWL2 的子语言（语法子集），它们在特定的应用场景中具有重要的作用。OWL2 共定义了三种不同的配置语言：OWL2 EL，OWL2 QL 和 OWL2 RL。每种配置语言都定义为“OWL2 结构化规范”的一个*语法限制*，即作为可用于相容本体(conforming ontology)的结构化元素的子集，并且它们都比 OWL DL 更具有限制性。每种配置语言为了获得不同的计算和/或实现便利，对 OWL 的表达能力的不同方面进行了折衷。

**OWL2 EL** 支持所有的用于标准推理任务的多项式时间算法；它尤其适用于需要超大规模本体的应用以及为了保证性能可以牺牲掉表达力的应用。**OWL2 QL** 支持在 LogSpace（更准确地说是 $AC^0$）中，使用标准关系型数据库技术来响应合取查询；它尤其适合于使用相对轻量级本体来组织大量个体，并且直接通过关系查询（例如 SQL）访问数据比较有用或者有必要的应用。**OWL2 RL** 支持多项式时间的推理算法实现，这些算法使用扩展了规则的数据库技术直接对 RDF 三元组进行操作；它尤其适用于使用相对轻量级本体来组织大量个体，并且对 RDF 三元组形式的数据进行直接操作很有用或者有必要的应用。

当然，任何的 OWL2 EL、QL 或者 RL 本体也是 OWL2 本体，可以使用直接语义或者基于 RDF 的语义进行解释。当使用 OWL2 RL 时，基于规则的实现可以在 RDF 三元组上直接操作，因此可以直接应用到任意的 RDF 图中去，即应用到任意的 OWL2 本体上去。在这种情况下，推理通常是*可靠的*（即：只会计算出正确的查询结果），但可能是不完整的（即：并不能确保能够得到所有正确的查询结果）。不过，“配置语言”文档的 PR1 定理表明：通常地，当本体与 OWL2 RL 的结构化定义相一致时，执行基础原子查询（ground atomic queries）的基于规则的适当实现既是可靠的，又是完整的。

# 3 与 OWL1 的关系

OWL2 与 OWL1 拥有非常相似的整体架构。如图 1 所示，尽管名称可能不同，但是 OWL2 的所有构建模块在 OWL1 中也都有。

RDF/XML 的核心角色，其他语法的角色以及直接语义和基于 RDF 的语义之间的关系（即对应定理）并没有改变。更重要的是，对于所有的意图和目的，OWL2 与 OWL1 是完全向后兼容的：所有的 OWL1 本体依旧是有效的 OWL2 本体，它们在所有的实际用例中都有相同的推论（见“OWL2 新特性与原理”[*OWL 2 New Features and Rationale*]的 Section 4.2）。

OWL2 为 OWL1 添加了一些新的功能。一些新特性是语法上的改进（“语法糖”，例如，类的不相交并集），其他的则是提供了新的表达能力，包括：

- 键（keys）；
- 属性链（property chains）；
- 更丰富的数据类型，数据定义域；
- 有条件的基数限制；
- 非对称属性，自反属性和不相交属性；
- 增强的注释能力。

OWL2 也定义了三种新的配置语言[*OWL 2 Profiles*] 和一种新的语法 [*OWL 2 Manchester Syntax*]。另外，放松了适用于 OWL DL 的一些限制；因此，OWL2 中可以为描述逻辑推理器所操控的 RDF 图集合稍微大一些。

以上内容在“OWL2 新特性与原理”文档[*OWL 2 New Features and Rationale*]中都有详述。“OWL 2 快速参考指南”[*OWL 2 Quick Guide*] 也对 OWL2 的特性作了概述，清晰地指出了哪些是新的。

# 4 文档路线图

OWL2 本体语言是通过五个核心的规范文档规范定义的，它们分别描述了它的概念性结构，主要的交换语法（RDF/XML），两种可选的语义（直接语义和基于 RDF 的语义），以及一致性要求。三个附加的规范文档则描述了一些实现可能会支持的可选特性：语言配置和两种可选的具体语法（OWL/XML 和 Manchester）。

然而，这些文档都相当具有技术性，主要针对 OWL2 的实现者及工具开发人员。需要更加易用的 OWL2 特性和用法指南的人可以参考查询用户文档（“OWL2 入门”，“OWL2 新特性与原理”和“OWL2 快速参考指南”）。

| 序号 | 类别 | 文档 |
|---|---|---|
| 1 | 用户指南 | 文档概述( Document Overview )。OWL2 的快速概览，它描述了 OWL2 与 OWL1 的关系。该文档是 OWL2 的入门和初步参考资料。 |
| 2 | 核心规范 | 结构化规范与函数式语法 (Structural Specification and Functional-Style Syntax)从结构与函数式语法的角度定义了 OWL2 本体的结构，并从 OWL2 本体全局限制的角度定义了 OWL2 DL 本体。 |
| 3 | 核心规范 | 映射到 RDF 图(Mapping to RDF Graphs) 定义了 OWL2 结构到 RDF 图的映射，因而也就定义了在语义网中交换 OWL2 本体的主要方法。 |
| 4 | 核心规范 | 直接语义(Direct Semantics)从模型－理论语义的角度定义了 OWL2 的含义。 |
| 5 | 核心规范 | 基于 RDF 的语义 (RDF-Based Semantics)通过 RDF 语义(RDF Semantics)的一个扩展定义了 OWL2 本体的含义。 |
| 6 | 核心规范 | 一致性准则(Conformance) 提出了对 OWL2 工具的要求，也提供了帮助判定一致性的测试用例集。 |
| 7 | 规范 | 配置语言(Profiles) 定义了 OWL2 的三个子语言，它们在特定应用场景下可以发挥重要的作用。 |
| 8 | 用户指南 | OWL2 入门(OWL 2 Primer) 对 OWL2 作了较通俗的介绍，其中包括对来自于其它学科交叉知识的介绍。 |
| 9 | 用户指南 | OWL2 新特性与原理(OWL 2 New Features and Rationale) 对 OWL2 的主要新特性作了概述，并鼓励将它们应用到语言中。 |

| | | |
|---|---|---|
| 10 | 用户指南 | OWL2 快速参考指南(OWL 2 Quick Reference Guide）概要地说明了 OWL2 的结构，同时也指出了相对于 OWL1 的变动。 |
| 11 | 规范 | XML 序列化(XML Serialization)定义了交换 OWL2 本体的 XML 语法，适合与基于模式的编辑工具及 XQuery/XPath 这样的 XML 工具一起使用。 |
| 12 | 规范 | 曼彻斯特语法(Manchester Syntax)(WG Note) 定义了易读但形式化较弱的 OWL2 语法，用于某些 OWL2 用户界面工具，在入门(Primer)中也用到了。 |
| 13 | 规范 | 数据值域扩展：线性等价(Data Range Extension: Linear Equations)(WG Note) 指定了 OWL2 的可选扩展，它支持对属性值的高级约束。 |

# 5 附录: 变动日志 (资料性)

## 5.1 相对于建议推荐标准的变动

自 2009 年 9 月 22 号的建议推荐标准以来，没有发生变动。

## 5.2 相对于上一征求意见版本的变动

该部分总结了自 2009 年 6 月的上一征求意见版本的工作草案以来，本文档的变动。

- 做了一些很小的编辑性变动。

# 6 致谢

OWL2 的开发始于 OWL1.1 成员提交（其本身是用户和开发者反馈的结果），尤其是在 OWL 体验与研究方向工作组(OWLED)系列中积累的信息。该工作组也考虑了来自于 WebOnt 工作组（WebOnt Working Group）的待解决问题（postponed issues）。

本文档由 OWL 工作小组（见下）执笔，它的内容反映了作为一个整体的该工作小组内部的广泛讨论。编者向 Ivan Herman (W3C/ERCIM)，Ian Horrocks (牛津大学)和 Peter F. Patel-Schneider (Bell Labs Research, Alcatel-Lucent) 的仔细审阅致以特别的谢意。

在本文档发布时，经常参加 OWL 工作组会议的与会者有：Jie Bao (RPI)、Diego Calvanese (Free University of Bozen-Bolzano)、Bernardo Cuenca Grau (牛津大学计算实验室)、Martin Dzbor (公开大学)、Achille Fokoue (IBM 公司)、Christine Golbreich (Université de Versailles St-Quentin and LIRMM)、Sandro Hawke (W3C/MIT)、Ivan Herman (W3C/ERCIM)、Rinke Hoekstra (阿姆斯特丹大学)、Ian Horrocks (牛津大学

计算实验室)、Elisa Kendall (Sandpiper Software)、Markus Krötzsch (FZI)、Carsten Lutz (不来梅大学)、Deborah L. McGuinness (RPI)、Boris Motik (牛津大学计算实验室)、Jeff Pan (阿伯丁大学)、Bijan Parsia (曼彻斯特大学)、Peter F. Patel-Schneider (Bell Labs Research, Alcatel-Lucent)、Sebastian Rudolph (FZI)、Alan Ruttenberg (科学共享组织)、Uli Sattler (曼彻斯特大学)、Michael Schneider (FZI)、Mike Smith (Clark & Parsia)、Evan Wallace (NIST)、Zhe Wu (甲骨文公司)和 Antoine Zimmermann (DERI Galway)。我们还要感谢以前的工作组成员：Jeremy Carroll、Jim Hendler 和 Vipul Kashyap。

# 7 参考


**[OWL 2 Conformance]**
*OWL 2 Web Ontology Language: Conformance* Michael Smith, Ian Horrocks, Markus Krötzsch, Birte Glimm, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-conformance-20091027/. Latest version available at http://www.w3.org/TR/owl2-conformance/.

**[OWL 2 Direct Semantics]**
*OWL 2 Web Ontology Language: Direct Semantics* Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-direct-semantics/.

**[OWL 2 Manchester Syntax]**
*OWL 2 Web Ontology Language: Manchester Syntax* Matthew Horridge, Peter F. Patel-Schneider. W3C Working Group Note, 27 October 2009, http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-manchester-syntax/.

**[OWL 2 New Features and Rationale]**
*OWL 2 Web Ontology Language: New Features and Rationale* Christine Golbreich, Evan K. Wallace, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-new-features-20091027/. Latest version available at http://www.w3.org/TR/owl2-new-features/.

**[OWL 2 Primer]**
*OWL 2 Web Ontology Language: Primer* Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-primer-20091027/. Latest version available at http://www.w3.org/TR/owl2-primer/.

**[OWL 2 Profiles]**
*OWL 2 Web Ontology Language: Profiles* Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/. Latest version available at http://www.w3.org/TR/owl2-profiles/.

**[OWL 2 Quick Reference Guide]**

*OWL 2 Web Ontology Language: Quick Reference Guide* Jie Bao, Elisa F. Kendall, Deborah L. McGuinness, Peter F. Patel-Schneider, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/. Latest version available at http://www.w3.org/TR/owl2-quick-reference/.

**[OWL 2 RDF Mapping]**

*OWL 2 Web Ontology Language: Mapping to RDF Graphs* Peter F. Patel-Schneider, Boris Motik, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-mapping-to-rdf-20091027/. Latest version available at http://www.w3.org/TR/owl2-mapping-to-rdf/.

**[OWL 2 RDF-Based Semantics]**

*OWL 2 Web Ontology Language: RDF-Based Semantics* Michael Schneider, editor. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-rdf-based-semantics/.

**[OWL 2 Specification]**

*OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax* Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-syntax/.

**[OWL 2 XML Serialization]**

*OWL 2 Web Ontology Language: XML Serialization* Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/. Latest version available at http://www.w3.org/TR/owl2-xml-serialization/.

**[RDF Semantics]**

*RDF Semantics*. Patrick Hayes, ed., W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Latest version available as http://www.w3.org/TR/rdf-mt/.

**[RDF Syntax]**

*RDF/XML Syntax Specification (Revised)*. Dave Beckett, ed. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Latest version available as http://www.w3.org/TR/rdf-syntax-grammar/.

**[Turtle]**

*Turtle - Terse RDF Triple Language*. David Beckett and Tim Berners-Lee. W3C Team Submission, 14 January 2008, http://www.w3.org/TeamSubmission/2008/SUBM-turtle-20080114/. Latest version available at http://www.w3.org/TeamSubmission/turtle/.

**[UML]**

*OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*. Object Management Group, OMG Available Specification, November 2007, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/.

## 2.C　OWL2 Web 本体语言入门

本文档《OWL2 Web 本体语言入门》是 W3C 发布的 ***OWL 2 Web Ontology Language Primer***（2009-10-27）的中文译本。文中若存在译法不当和错误之处，欢迎批评指正，请发邮件至：**zengxh@szu.edu.cn，谢谢！**

**翻译说明：**

- 本文档的英文版是唯一正式版本。此处的中文译本仅供学习与交流。
- 中文译本的内容是非正式的，仅代表译者的个人观点。
- 中文译本的内容会根据反馈意见随时进行修订。
- 中文译本同时通过 W3C Translations 网站发布。
- 转载本文，请注明译者和原链接。

**译者：**

曾新红（Xinhong Zeng），深圳大学图书馆 NKOS 研究室
吴　鹏（Peng Wu），深圳大学计算机与软件学院
林伟明（Weiming Lin），深圳大学图书馆 NKOS 研究室

**资助声明：**

本次翻译工作得到国家社科基金项目“中文知识组织系统的形式化语义描述标准体系研究”（批准号：12BTQ045）和广东省哲学社会科学“十一五”规划项目（批准号：GD10CTS02）的资助。

翻译时间：2013 年 9 月

发布时间：2013 年 11 月 13 日

---

# OWL 2 Web 本体语言

# 入门

## W3C 推荐标准 2009 年 10 月 27 日

**当前版本:**

http://www.w3.org/TR/2009/REC-owl2-primer-20091027/

**最新版本 (系列 2):**

http://www.w3.org/TR/owl2-primer/

**最新推荐标准版本:**

http://www.w3.org/TR/owl-primer

**上一版本:**

http://www.w3.org/TR/2009/PR-owl2-primer-20090922/ (彩色标注不同之处版本)

**贡献者:**

Pascal Hitzler, Wright State University
Markus Krötzsch, FZI Research Center for Information Technology
Bijan Parsia, University of Manchester
Peter F. Patel-Schneider, Bell Labs Research, Alcatel-Lucent
Sebastian Rudolph, FZI Research Center for Information Technology

请参阅本文档的**勘误表**，那里可能会有一些规范的校正。

本文档也可以以如下的非规范格式查看：PDF 版本。

另见译文。



---

# 摘要

OWL2 Web 本体语言，简称 OWL2，是一种本体语言，用于带有形式化定义含义的语义网。OWL2 本体提供了类、属性、个体及数据值，以语义网文档的形式存储。OWL2 本体可与以 RDF 记载的信息结合使用，并且 OWL2 本体本身也主要是以 RDF 文档的形式进行交换。OWL2 文档概述（Document Overview）阐述了 OWL2 的整体状态，因此，应当在阅读其他 OWL2 文档之前先阅读该文档。

本文档提供了对 OWL2 语言通俗易懂的介绍，包括对来自于其他学科的内容的定位，通过一个连续示例来展示 OWL2 如何首先用来表示简单的信息，然后表示更复杂的信息，OWL2 如何管理本体，最后介绍 OWL2 本体语言的各种子语言之间的区别。

# 本文档的状态

**可能已被替代**

*本节描述的是本文档在发布时的状态。其他的文档可能替代了该文档。在 http://www.w3.org/TR/ 的 W3C technical reports index 中，可以找到当前的 W3C 出版物列表和该技术报告的最新版本。*

### XML Schema 数据类型依赖

根据定义，OWL2 使用的是 XML Schema Definition Language (XSD)中定义的数据类型。对本文档中的 OWL2 而言，XSD 的最新 W3C 推荐标准是版本 1.0，此时，版本 1.1 正在向推荐标准演进。OWL2 的设计已经利用了 XSD1.1 中的新数据类型和更清晰的注释，但是目前这些利用有一部分暂时搁置。特别地，OWL2 中基于 XSD1.1 的那些元素将被当作是*可选的*，直到 XSD1.1 成为 W3C 的推荐标准为止，详见 Conformance, section 2.3。等到 XSD1.1 发布为 W3C 的推荐标准时，这些元素才会终止可选状态而与其他指定元素一样进入必备状态。

我们提议，目前开发人员和用户遵循 XSD 1.1 Candidate Recommendation。根据 Schema 和 OWL 工作组之间的讨论，当 XSD1.1 演进成为推荐标准时，我们并不希望任何实现会有必须的改动。

### 未变的文档

本文档与先前版本并没有主体上的改变。关于更早版本的变动细节，请见 change log 和 color-coded diff。

### 请发表意见

请将意见发送至 public-owl-comments@w3.org（公开文档）。虽然由 OWL 工作组执笔的本文档已经完成，但是您的意见依旧可能在勘误表或者未来的修订版中得到解决。欢迎开发人员在 public-owl-dev@w3.org(公开文档)公开讨论。

### 由 W3C 批准

*本文档已经由 W3C 成员、软件开发人员以及其他 W3C 小组和兴趣组织审查，并由 W3C 主管（Director）批准成为 W3C 推荐标准。这是一个稳定的文档，可以作为参考资料或被其他文档引用。W3C 在制作推荐标准过程中担任的角色，是要引起人们对该规范的注意并促进它的广泛应用。这提升了 Web 的功能性和交互性。*

### 专利

*本文档由遵循 5 February 2004 W3C Patent Policy 的小组完成。它只是一个提供信息的文档。W3C 维护着一个专利披露公开列表（public list of any patent disclosures），它与该组织的可交付成果一起制作；此页面也包含披露专利的说明。*

## 目录

# 1 简介

W3C的OWL2 Web本体语言(OWL)是一种语义Web语言，它被设计用来表示关于事物、事物群和事物之间关系的丰富和复杂知识。OWL是一个以计算逻辑为基础的语言，于是用OWL表示的知识可以通过计算机程序进行推理，从而验证知识一致性，或者使隐性知识显性化。OWL文档，也被称为本体，能够在万维网上发布，可以引用其他的OWL本体或者被其他的OWL本体引用。OWL是W3C Semantic Web技术堆栈（其中还包括RDF [*RDF Concepts*]和SPARQL[*SPARQL*] ）的一部分。

入门文档编写的目的是帮助读者深入理解OWL，以及它的优势和劣势。此文档的核心是通过一个连续示例来介绍OWL的大多数语言特性。本文很多例子来自一个样本本体（在附录an appendix中完整展示）。这个样本本体被设计用来以一种容易理解的方式接触OWL的关键语言特性，它本身并不是一个好的本体。

## 1.1 本文档指南

这篇文档拟提供对 OWL2 的初步了解，特别适用于对 OWL2 不熟悉的读者。因此，我们在 Section 2 对 OWL2 的性质进行一些高层介绍，在 Section 3 提供一些很基本的知识表达概念，并解释它们如何与 OWL2 使用的术语相关。熟悉知识表达和推理的读者，也许只需快速浏览本节就可以获得 OWL2 的术语集。

第 4 至 8 节描述了 OWL 提供的大多数语言特性，从非常基础的开始，逐步推进到更复杂的特性。Section 4 呈现和讨论了 OWL2 的基本建模特性，之后 Section 5 引入了复杂类。Section 6 专门介绍用于属性的高级建模特性。Section 7 关注与数据类型相关的高级建模。Section 8 总结了主要用于本体管理目的的超逻辑（extralogical）特性。

在 Section 9 我们专注于 OWL2 DL 和 OWL2 Full 这两个 OWL 语义视图的不同之处，而在 Section 10 我们描述了 OWL2 的三个易处理的子语言，称为配置语言。Section 11 专门介绍对 OWL2 的工具支持，而在 Section 12 中我们指出，在看完本文档的 OWL2 非正式介绍之后，应该到哪儿去找继续阅读的资料。

最后，Section 13 列出了在本文档中使用的完整本体样例。

如果想要一个全面的 OWL2 语言特性清单，请查看 OWL2 快速参考指南[*OWL 2 Quick Guide*]，它提供了链接指向相应章节中关于语义和例子的适当文档。

对于已经熟悉了 OWL1 的读者，OWL2 新特性与原理[*OWL 2 New Features and Rationale*]提供了一个 OWL2 中所有变化的全面概述。

## 1.2 OWL 语法

OWL是一个在语义Web上使用的语言，因此OWL中的名称是国际资源标识

符(IRIs) [*RFC 3987*]。因为IRI很长，所以我们会常常使用缩写。缩写的工作方式特定于每个句法格式，可用于编码OWL本体，但本文档中的例子通常不用知道它们的细节就可以理解。在Section 8.2会进一步给出适当的命名空间声明和相关机制。

OWL提供了各种可用的语法，服务于不同的目的。设计函数式语法[*OWL 2 Specification*]是为了更容易达到规范目的，并且给OWL2工具（例如APIs和推理器）的实现提供一个基础。用于OWL的RDF/XML语法，就是RDF/XML，再加上一个对于OWL结构的特定翻译 [*OWL 2 RDF Mapping*]，这是唯一一个所有的OWL2工具都必须支持的语法。曼彻斯特语法[*OWL 2 Manchester Syntax*]是一个OWL语法，它的设计目的是让非逻辑学家更好阅读。OWL XML语法是一种用于OWL的XML语法，用一个XML schema定义[*OWL 2 XML*]。有一些工具能够为OWL转换不同的语法。OWL语言结构在很多句法形式中也用IRIs表示，很多声明需要用到缩写形式，如例子所示。必要的细节也请见Section 8.2。

附录中的例子和样本本体可用4种不同的语法浏览，基于RDF语法序列化方式我们既提供RDF/XML [*RDF Syntax*]，也提供Turtle [*RDF Turtle Syntax*]。在整个文档，你可以通过下边的按钮控制显示哪个语法。

下边的按钮可以用来显示或者隐藏可用的语法。（译者注：此功能略）

# 2 OWL2 是什么？

OWL2是一种表示本体的语言。本体这个术语在计算机科学之内和之外都有一个复杂的历史，但是我们用它来表示某种计算工件（computational artifact），即类似于一个程序，一个XML schema，或者一个网页的东西，通常都以文档的形式表示。本体是一组对世界某一部分（通常被称为本体的兴趣领域或题材）的精确的描述性陈述。精确的描述要满足几个因素：最重要的，要防止在人类交流中产生误解，要保证软件以统一且可预测的方式表现，并且能够与其他的软件很好地协同工作。

为了精确描述一个兴趣领域，提出一组中心术语（通常也叫词汇表）并且确定它们的含义是有用的。除了一个简短的自然语言定义之外，这个词的含义还可以通过说明这个词怎样与其他词相互联系来进行描述。提供词汇表以及相互关系信息的术语集（terminology），是典型的OWL2文档的组成要素。除了术语知识，本体也可能包含所谓的断言知识，处理具体的域对象而不是笼统的观念考虑。

OWL2不是一种编程语言：OWL2是声明性的（declarative），也就是说，它用逻辑的方式描述一个事务的状态。合适的工具（所谓的推理器）于是可以进一步用来推断关于这种状态的信息。这些推断怎样用算法实现不是OWL文档的一部分，但是依赖于具体的实现。不过，任何这种问题的正确答案都由形式语义（有两个版本：直接语义[OWL 2 Direct Semantics]和基于RDF的语义 [OWL 2 RDF-Based Semantics]）预先决定。只有遵守这些语义的实现才能叫OWL2一致（参考[OWL 2 Conformance]）。透过其声明性的本质，创建OWL2文档的活动在概念上就不同于编程。不过，在这两种情况下都需要创建复杂的形式化文档，可以将某些观念从软件工程转移到本体工程，如方法论和协同工作、模块化、模式等。

OWL2不是一个用于语法一致性的schema语言。不同于XML，OWL2不提

供精确的方法来规定一个文档应该如何在句法上结构化。特别是，它没有方法执行某些信息(如一个人的社会安全号)必须按句法呈现。这一点要记住，因为OWL有一些特性可能会让使用者产生这样的误解。

OWL2不是一个数据库框架。当然，OWL2文件存储信息，数据库也一样。此外，断言信息和数据库内容之间以及术语信息和数据库模式之间的某种相似也可以被描绘出来。然而，它们在基本假设上通常有重要区别（技术层面：使用的语义）。如果一些事实不存在于数据库中，它通常被认为是假的（所谓的封闭世界假设），而对于一个OWL 2文档它可能只是丢失了（但可能是真的），遵循开放世界假设。此外，数据库模式常常会有上面提到的规定性约束语义。不过，在技术上，数据库为许多面向本体的系统提供了可行的支柱。

# 3 知识建模：基本概念

OWL2是一种知识表示语言，旨在对兴趣领域的知识进行明确表达(formulate)、交换和推理。应该首先解释一些基本概念，以方便理解知识在OWL2中是如何被表达的。这些基本概念如下：

- **公理(Axioms):**,一个 OWL 本体表达的基本陈述
- **实体(Entities)：**用来指向现实世界对象的元素
- **表达式(Expressions)：**实体的组合，从简单的描述形成复杂的描述

OWL2的目的是捕获知识，可以用OWL表示的这种“知识”当然不能反映人类知识的所有方面。OWL2可以被看作是一种针对人类知识某些部分的强大而通用的建模语言。建模的结果被称为本体，这个术语也有助于避免混乱，因为在知识表示领域，“模型”这个术语经常以一种相当不同的意义在使用。

现在，为了明确地阐述知识，假设它由被称为陈述（statements）或命题（propositions）的基本块组成是有用的。如陈述“下雨了”或“每个人终有一死”这样的基本命题，都是典型的例子。事实上，每一个OWL2本体实际上就是这样一组基本的“知识块”。本体中的陈述在OWL2中被称为公理，该本体断言其公理是真的。一般来说，给定事务的某种状态，OWL陈述可以是真的或假的。这可以将它们与实体和表达式区分开来，下面会进一步描述。

当人类思考时，他们从自己的知识中推出结果。OWL的一个重要特点，就是它会以其可以表示的知识形式捕捉人类智力的这一方面。那么通常所说的一个陈述是其他陈述的推论是什么意思呢？它基本上意味着只要其他陈述是真的，这个陈述也是真的。用OWL的术语来说，就是如果在任何状态下，A中的所有陈述是真的，a也是真的，那么陈述集合A*蕴涵*（entails）陈述a。此外，一个陈述集合可能是*一致的*（即存在一个可能的状态，其中的所有陈述均为真）或*不一致的*（即不存在这样的状态）。OWL的形式语义规定，在本质上，对于那种可能的“状态”一个特定的OWL陈述集是真的。

有OWL工具（推理器）可以自动计算结果。这种本体公理互动的方式非常微妙，人类很难理解。对于OWL2，这既是优势又是劣势。它的优势是用OWL2工具能够发现人类没有发现的信息。这允许知识工程师更直接地建模，系统能够对建模提供有用的反馈和评论。它的劣势是人类较难立即预见各种组合中的各种结构的实际效果。工具的支持可以改善情况，但成功的知识工程往往仍然需要一

定的培训和经验。

仔细看一下OWL的陈述，我们发现它们很少是“单一整体结构”，但经常有一些可以显式表示的内部结构。它们通常指世界中的对象，并被放入类别（category）中（如“Mary是女人”）或说一些关于其关系的事情（“John和Mary结婚了”）来进行描述。陈述中的所有原子成分被称为实体（entities），它们是对象（John，Mary），类别（女性）或关系（结婚）。在OWL2中，对象用*个体*(individuals)表示，类别用*类*(classes)表示，关系用*属性*(properties)表示，属性将会进一步细分。*对象属性*（object properties）将对象关联到对象（如一个人与其配偶的关联），而*数据类型属性*（datatype properties）分配数据值给对象（如将年龄赋予一个人）。*注释属性*（annotation properties）被用于为本体本身（或其部分）进行信息编码（如一个公理的作者和创建日期），取代兴趣领域。

作为OWL的中心特征，可以用所谓的*构造器*（constructors）将实体的名称组合进*表达式*（expressions）。一个基本例子，原子类“女性”和“教授”可以结合起来描述类“女教授”。后者可以用一个OWL类表达式描述，该表达式可用于陈述或其他表达式中。从这个意义上讲，表达式可以被视为被其结构（structure）定义的新实体。在OWL中，用于每种类型实体的构造器（constructors）差异很大。用于类的表达式语言十分丰富和复杂，而用于属性的表达式语言则要少得多。这些差异有历史和技术的原因。

# 4 类、属性和个体——及其基本建模

介绍完基本概念，我们现在要深入 OWL2 建模的细节了。在随后的段落中，我们将介绍 OWL2 提供的基本建模特性，为如何使用它们提供例子和一般性意见。我们从基本特性（这些基本特性任何建模语言都会具备）出发，逐步向更高级的结构推进。

因此我们将展示一个特定家庭的信息。注意我们并不打算用这个例子作为这类应该用 OWL 描述的领域的代表，或作为用 OWL 建模的好典型，或者是相当复杂、多变和具有文化依赖性的家庭领域的正确表示。相反，这只是一个展示 OWL 各种特性的相当简单的例子。

## 4.1 类（Classes）和实例（Instances）

我们先介绍我们正在讨论的人，做法如下：

**Functional-Style Syntax**

```
ClassAssertion(  :Person  :Mary )
```

这个陈述谈到一个名叫 Mary 的个体并且声明这个个体是人。更专业地，*是一个人*通过说明 Mary 归属于（或者“is a member of”，甚至更专业地称“is an

instance of”）所有人类的这个 *class* 来表达。通常，类被用来对一些具有共性的个体分组，以便引用它们。因此，类本质上代表了个体集合。建模时，类经常被用来表示由一组人类思维概念组成的对象，如概念 *person* 或者 *woman*。所以，我们可以用同样的方式表明 Mary 是女人，即她是 woman 类的实例：

**Functional-Style Syntax**

```
ClassAssertion(  :Woman  :Mary )
```

由此也可以知道类成员关系不是排他的：因为可能有各种各样的标准对个体分组（例如性别、年龄、鞋码等等），一个个体可能同时属于多个类。

## 4.2 类层次结构

在上节中，我们提到两个类：所有人的类和所有女人的类。对于人类读者而言很明显这两个类有一个特殊的关系：人比女人更广泛，也就是说，只要我们知道一些个体是女人，那么这些个体也一定是人。然而，这个对应关系并非源自“person”和“woman”这两个标签，它是关于世界和这些术语用法的人类背景知识的一部分。因此，为了让一个系统能够得出预期的结论，必须告诉它这些对应关系。在 OWL2 中，这通过一个所谓的子类公理来实现：

**Functional-Style Syntax**

```
SubClassOf(  :Woman  :Person )
```

在一个本体中这个公理的存在使得推理器可以针对每一个被指定为 woman 类的实例的个体，推断出她同样也是 person 这个类的实例。作为一个经验法则，如果短语“每一个 A 都是一个 B”有意义且正确，那么类 A 跟类 B 之间的子类关系就能够指定。

在本体建模中经常使用子类陈述，不仅用来偶尔声明这种相关性，而且通过指定兴趣领域所有类的泛化关系对整个类层次结构建模。假设我们想表示所有的母亲都是女人：

**Functional-Style Syntax**

```
SubClassOf(  :Mother  :Woman )
```

那么推理器不仅可以推导出每个归类为母亲的单独个体，也是一个女人（因此也是人），而且可以推导出 Mother 也是 Person 的子类——和我们的直觉一致。从专业上讲，这意味着类和子类之间的关系是传递的（transitive）。除此之

外，它也是自反的（reflexive），意味着每一个类都是它自己的子类，这也很直观，很明显每个人都是一个人。

在我们的词汇表中，类可以有效指向相同的集合，而 OWL 提供了一种机制，表明它们在语义上被认为是等价的。例如，我们交替使用 Person 和 Human 这两个术语，意味着每一个 Person 类的实例也是 Human 类的实例，反之亦然。如果两个类包含完全相同的个体，这两个类被认为是等价的。下面的例子指出 Person 类和 Human 类是等价的。

Functional-Style Syntax

```
EquivalentClasses(  :Person  :Human )
```

上述陈述声明了 Person 类和 Human 类是完全等价的，相当于同时声明 Person 类是 Human 类的子类以及 Human 类是 Person 类的子类。

## 4.3 类不相交

在 4.1 节，我们说明了一个个体可以是几个类的实例。然而在某些情况下，一个类的成员明显不包括另外一个类的成员。例如，就类 Man 和 Woman 而论，可以排除存在一个个体同时是这两个类的实例（为了这个示例，我们忽略生物学上的边界个案）。这种在两个类之间的“不相容关系”被称为*（类）不相交*((class) disjointness)。两个类不相交的信息又是我们背景知识的一部分，必须为推理系统明确声明才能使用。做法如下：

**Functional-Style Syntax**

```
DisjointClasses(  :Woman  :Man )
```

实际上，不相交陈述经常被忘记或者忽略。一个不太确定的原因可能是，直观而言，除非有其他证明才会考虑类不相交。如果不相交陈述被忽略，很多可能有用的结果会丢失。注意在我们的例子中，需要不相交公理来推导出 Mary 不是一个男人。此外，通过上述公理，推理器才能推断出类 Mother 和类 Man 之间的不相交性。

## 4.4 对象属性

在前面章节，我们关注单一的个体、它们的类成员关系以及类是如何基于它们的实例相互关联的。但通常一个本体也意味着指定了一个个体如何与其他个体关联。当描述一个家庭时，这些关系是核心。我们首先说明 Mary 是 John 的妻子。

**Functional-Style Syntax**

```
ObjectPropertyAssertion(  :hasWife  :John  :Mary )
```

在此，描述个体以何种方式关联的实体，称作*属性* (properties)，如我们例子中的 hasWife。

注意，个体书写的顺序很重要。“Mary is John's wife”可能是真，“John is Mary's wife”则一定为假。事实上，这是一种常见的建模错误，通过使用只允许一种直观阅读的属性名称可以避免。例中的名词（如 wife），这种无歧义的名称可以用“of”或者“has”来构建（wifeOf 或 hasWife）。对于动词（例如“to love”），变化词形（loves）或者使用带有 “by”的被动语态（lovedBy）就可以防止误解。

我们也能够声明两个个体不是通过一个属性关联。例如下面的例子，指出 Mary 不是 Bill 的妻子。

**Functional-Style Syntax**

```
NegativeObjectPropertyAssertion(  :hasWife  :Bill  :Mary )
```

否定属性断言提供了一个唯一的机会声明我们知道为假的事物。这类信息在 OWL 中特别重要，OWL 的默认立场是任何事情都是可能的，除非你另外说明。

## 4.5 属性层次结构

在 4.2 节，我们讨论过指定一个类成员关系蕴涵（implies）另外一个是有用的。本质上，同样的情形也会发生在属性上：如果知道 B 是 A 的妻子，也就知道了 B 是 A 的配偶（注意，顺序反过来就不成立）。OWL 允许如下陈述：

**Functional-Style Syntax**

```
SubObjectPropertyOf(  :hasWife  :hasSpouse )
```

这也是用于属性等价（类似于类等价）的一个语法快捷方式。

## 4.6 定义域和值域限制

两个个体通过某个属性相互关联的信息，经常允许推出关于个体本身的进一步结论。特别是可以推断出类成员关系(membership)。例如，陈述“B 是 A 的妻子”显然隐含着 B 是女人而 A 是男人。所以在某种程度上，关于两个个体通过某个属性相关联的陈述，携带着有关这些个体的隐含的额外信息。在我们的例

子中，这些额外信息能够通过类成员关系来表示。OWL 提供一种方法说明这种对应关系：

**Functional-Style Syntax**

```
ObjectPropertyDomain(  :hasWife  :Man )
ObjectPropertyRange(  :hasWife  :Woman )
```

有了这两个公理，并且给定信息，例如 Sasha 和 Hillary 通过属性 hasWife 关联，推理器就能够推断出 Sasha 是男人和 Hillary 是女人。

## 4.7 等价和非等价个体

请注意，到目前为止，从给出的信息可以推断出 John 和 Mary 是不同的个体，因为他们分别是不相交类 Man 和 Woman 的实例。然而，如果我们添加另一个家庭成员信息，例如 Bill，指出他是男性，且至今为止也没有说什么来暗示 John 和 Bill 不一样。OWL 没有假设不同的名字是不同个体的名称（这种对必需的“unique names assumption（唯一名称假设）”的缺失特别适用于语义 Web 应用程序，那里的名称可能由不同组织在不同时间创建，不知不觉指向同一个个体）。因此，如果我们想排除 John 和 Bill 是相同个体这种选项，就必须明确定义如下：

**Functional-Style Syntax**

```
DifferentIndividuals(  :John  :Bill )
```

也有可能要声明两个名称指向（代表）同一个个体。例如，我们可以说 James 和 Jim 是同一个个体。

**Functional-Style Syntax**

```
SameIndividual(  :James  :Jim )
```

这可以使推理器推断出任何关于个体 James 的信息也适合 Jim 这个个体。

## 4.8 数据类型

到目前为止，我们已经了解了如何通过类成员关系以及与其他个体的关系来描述个体。然而在很多情况下，个体要通过数据值描述。例如一个人的生日、年龄、电子邮箱地址等。为此，OWL 提供了另一种属性，称为*数据类型属性*（Datatype properties）。这些属性将个体与数据值（而不是其他个体）关联，

很多 XML Schama 数据类型[*XML Schema Datatypes*]可以利用。下面是一个使用数据类型属性的示例，它描述了 John 的年龄是 51 岁。

**Functional-Style Syntax**

```
DataPropertyAssertion(  :hasAge  :John "51"^^xsd:integer )
```

同样的，我们也能够描述 Jack 的年龄不是 53 岁。

**Functional-Style Syntax**

```
NegativeDataPropertyAssertion(  :hasAge  :Jack "53"^^xsd:integer )
```

定义域和值域也可以为数据类型属性声明，就像为对象属性那样。然而，在这种情况下，值域将是一个数据类型而不是一个类。下面声明了 hasAge 属性仅用来将人与非负整数关联。

**Functional-Style Syntax**

```
DataPropertyDomain(  :hasAge  :Person )
DataPropertyRange(  :hasAge xsd:nonNegativeInteger )
```

现阶段我们要指出使用属性定义域和值域时很容易发生的一个常见错误。在刚才给出的例子中 hasAge 属性仅用来关联人和非负整数，假如我们也要指定信息“Flix 是 Cat 类并且 9 岁”。从这种合并信息中，可能推出 Flix 也是 Person 类，这可能不是所预期的。这是一个常见的建模错误：注意一个定义域（或值域）声明不是一个对知识的约束，但是允许推理器进一步推出知识。在我们的例子中，如果我们声明年龄只能赋予给人，那么一切我们赋予年龄的东西自然而然是一个人。

# 5 高级类关系

在前面的章节，我们把类当作是某种有名字的“不透明的”东西。我们用它们来描述个体，并且通过子类或不相交陈述将它们与其他的类关联。

现在我们将演示如何将具名的类、属性和个体用作构建块来定义新的类。

## 5.1 复杂类

利用迄今为止描述过的语言元素，可以建模简单的本体。为了表达更复杂的知识，OWL 提供了逻辑类构造器（logical class constructors）。具体而言，OWL 为逻辑与、或、非提供了语言元素。相应的 OWL 术语是从集合论借用过来的：

（*类的*）*交集*（intersection）、*并集*（union）和*补集*（complement）。这些结构将原子类（即有名字的类）结合起来构建复杂类。

两个类的*交集*只包含那些同时是这两个类的实例的个体。下述例子描述了 Mother 类只包含了那些同时是 Woman 类和 Parent 类的实例的对象：

**Functional-Style Syntax**

```
EquivalentClasses(
   :Mother
  ObjectIntersectionOf(  :Woman  :Parent )
)
```

从上例中可以得到一个推论：Mother 类的所有实例也是 Parent 类的实例。

两个类的*并集*包含了每一个至少包含于其中一个类的个体。因此我们可以把所有父母的类描述成 Mother 类和 Father 类的并集：

**Functional-Style Syntax**

```
EquivalentClasses(
   :Parent
  ObjectUnionOf(  :Mother  :Father )
)
```

类的*补集*对应于逻辑非：它完全是由不属于该类本身成员的对象组成。下面用类的补集定义了没有孩子的人，也演示了类构造器（class constructors）可以嵌套：

**Functional-Style Syntax**

```
EquivalentClasses(
   :ChildlessPerson
  ObjectIntersectionOf(
     :Person
    ObjectComplementOf(  :Parent )
  )
)
```

以上所有的例子演示了类构造器的用法，为的是把新类*定义*为其他类的组合。但是，当然也可能同时使用类构造器和一个子类声明来指出一个类的必要非充分条件。下述声明指出每个 Grandfather 都是一个 man 和一个 parent（而反过来就不一定）：

**Functional-Style Syntax**

```
SubClassOf(
   :Grandfather
  ObjectIntersectionOf(  :Man  :Parent )
)
```

一般情况下，复杂类可以用在具名类可以出现的每个地方，因此也可以用在类断言中。下面的例子演示了这一点，断言 Jack 是一个人但不是一个父母。

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectIntersectionOf(
     :Person
    ObjectComplementOf(  :Parent )
  )
   :Jack
)
```

## 5.2 属性限制

属性限制为复杂类提供了另一种类型的基于逻辑的构造器（constructors）。顾名思义，属性限制使用涉及属性的构造器。

一种叫做*存在量化*（existential quantification）的属性限制定义了一个类，它是以下所有个体的集合：这些个体通过一个特定属性关联另一个个体（是某个类的实例）。这可以通过以下例子得到最好的解释，下例定义了 Parent 类，其个体通过 hasChild 这个属性关联到一个人。

**Functional-Style Syntax**

```
EquivalentClasses(
   :Parent
  ObjectSomeValuesFrom(  :hasChild  :Person )
)
```

这意味着存在一个预期，对于每个 Parent 类的实例，存在至少一个孩子，而且这个孩子是 Person 类的成员。这对捕获*不完整*知识是很有用的。例如，Sally 告诉我们 Bob 是一个父母，因此我们就能推断出他至少有一个孩子，即使我们不知道他们的名字。使用存在量化的自然语言指示物是像“some”或“one”这样的单词。

另外一种叫*全称量化*（universal quantification）的属性限制用来描述一个类，其包含的所有个体都必须是某给定类的实例。我们可以用以下陈述说明：某人是一个快乐的人，如果其所有孩子都是快乐的人。

**Functional-Style Syntax**

```
EquivalentClasses(
   :HappyPerson
  ObjectAllValuesFrom(  :hasChild  :HappyPerson )
)
```

这个例子也说明了 OWL 陈述被允许自引用，HappyPerson 这个类同时用在等价陈述的两边。

属性限制的使用可能对于“建模初学者”来说会导致一些概念混淆。作为一个经验法则，在将一个自然语言语句转换成逻辑公理时，存在量化的出现要频繁得多。使用全称量化的自然语言指示物是像“only”，“exclusively”或“nothing but”这样的单词。

关于全称角色限制有一种特别的误解。作为一个例子，考虑一下上面的快乐公理，直观读起来暗示着：一个人想要快乐必须有至少一个快乐的小孩。但是情况并不是这样：不是属性 hasChild 的“起始点”的任何个体，是用全称量化定义在 hasChild 上的任何类的类成员。因此，通过以上的陈述，每一个无子女的人也有资格快乐。要形式化前面提及的预期读法，必须声明如下：

**Functional-Style Syntax**

```
EquivalentClasses(
   :HappyPerson
  ObjectIntersectionOf(
    ObjectAllValuesFrom(  :hasChild  :HappyPerson )
    ObjectSomeValuesFrom(  :hasChild  :HappyPerson )
  )
)
```

这个例子也说明了属性限制也可以和复杂类嵌套。

属性限制也可以用来描述其个体与一个特殊的个体关联的类。例如我们可以定义 John 的孩子这个类：

**Functional-Style Syntax**

```
EquivalentClasses(
   :JohnsChildren
```

```
  ObjectHasValue(  :hasParent  :John )
)
```

作为一个被属性相互连接的个体的特殊案例，个体可能被连接到其自身。下面的例子展示了如何表示所有自恋者都爱他们自己这样一个观点。

**Functional-Style Syntax**

```
EquivalentClasses(
   :NarcisticPerson
  ObjectHasSelf(  :loves )
)
```

## 5.3 属性基数限制

使用全称和存在量化，我们可以表示分别至少有某人孩子中的一个的所有情况。然而，我们可能想要指定限制内的个体的数量。事实上，我们能够构建依赖于孩子的数量的类。下面的例子表明了 John 有至多 4 个其本身也是父母的孩子。

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectMaxCardinality( 4  :hasChild  :Parent )
   :John
)
```

注意，这个陈述允许 John 有任意多个其本身不是父母的孩子。

同样，也可以声明最小数量，说 John 是某个类的实例，该类的个体有至少两个其本身是父母的孩子：

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectMinCardinality( 2  :hasChild  :Parent )
   :John
)
```

如果我们知道 John 的孩子中本身是父母的孩子的确切数量，可以指定如下：

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectExactCardinality( 3  :hasChild  :Parent )
   :John
)
```

在基数限制里，提不提供类是可选的；如果我们只想谈谈 John 所有孩子的数量，我们可以写成：

**Functional-Style Syntax**

```
ClassAssertion(
  ObjectExactCardinality( 5  :hasChild )
   :John
)
```

## 5.4 个体的枚举

描述一个类的一个非常简单的方法就是枚举它所有的实例。OWL 提供了这种可能性，例如，我们可以创建一个生日来宾类：

**Functional-Style Syntax**

```
EquivalentClasses(
   :MyBirthdayGuests
  ObjectOneOf(  :Bill  :John  :Mary)
)
```

注意，这个公理比 4.1 节中描述的简单断言 Bill、John 和 Mary 的类成员关系提供更多信息。除此之外，它还规定 Bill、John 和 Mary 是 MyBirthdayGuests 类的*唯*一成员。因此，以这种方式定义的类有时也被称为*封闭类*或者枚举集。如果我们现在断言 Jeff 是 MyBirthdayGuests 类的一个实例，结果是 Jeff 必须等同于上面三个人中的一个。

# 6 属性的高级使用

直到现在，我们关注的是仅仅用作类表达式构建块的类和属性。接下来，我们将看看用 OWL2 提供的属性可以实现什么其他建模功能。

## 6.1 属性特征

有时候，一个属性可以通过用另外一个属性并改变它的方向来获得，也就是取逆。例如，属性 hasParent 可以定义为 hasChild 的逆属性：

**Functional-Style Syntax**

```
InverseObjectProperties(  :hasParent  :hasChild )
```

这允许推断，例如，任意个体 A 和 B，其中 A 通过 hasChild 属性关联到 B，B 和 A 也通过 hasParent 属性内联。然而，如果我们仅仅想使用它，比如，在一个类表达式里，我们不需要显式指定一个名称给逆属性。例如为了定义 Orphan 类，我们可以不使用新属性 hasParent，而是直接称它为 hasChild-inverse：

**Functional-Style Syntax**

```
EquivalentClasses(
   :Orphan
  ObjectAllValuesFrom(
    ObjectInverseOf(  :hasChild )
     :Dead
  )
)
```

在某些情况下，一个属性和它的逆一致，或者换句话说，属性的方向不重要。例如属性 hasSpouse，如果 B 用它关联 A，那么 A 也用它关联 B。原因很明显，具有这种特征的属性是所谓的*对称性*（属性），它可以如下指定：

**Functional-Style Syntax**

```
SymmetricObjectProperty(  :hasSpouse )
```

另一方面，一个属性也可以非对称，意思就是说如果用它关联 A 到 B，则永远不会用它关联 B 到 A。显然（排除时空穿梭造成的矛盾场景），属性 hasChild 就是这种情况，表示如下：

**Functional-Style Syntax**

```
AsymmetricObjectProperty(  :hasChild )
```

注意，非对称（asymmetric）是一个比不对称（non-symmetric）更强的概念。同样，对称（symmetric）是一个比不非对称（non-asymmetric）更强的概念。

之前，我们认为子属性类似于子类。事实上，将类不相交的概念转移到属性上也成立：如果没有两个个体同时通过某两个属性相互关联，则这两个属性不相交。按照普通法（common law），我们可以因此陈述父母-子女（parent-child）婚姻不可以出现：

**Functional-Style Syntax**

```
DisjointObjectProperties(  :hasParent  :hasSpouse )
```

属性也能够*自反*（reflexive）：这样的属性将一切事物关联到其自身。如下面的例子，指出每个人都是他自己的亲属。

**Functional-Style Syntax**

```
ReflexiveObjectProperty(  :hasRelative )
```

注意，这并不一定意味着任何两个通过一个自反属性相关联的个体是等同的。

而且，属性可以*非自反（irreflexive）*，意味着没有个体能够通过这样的角色关联其自身。以下是一个典型例子，简单陈述了没有人可以是他自己的父母。

**Functional-Style Syntax**

```
IrreflexiveObjectProperty(  :parentOf )
```

接下来，考虑 hasHusband 属性。因为每个人只能有一个丈夫（为了此示例我们认为理所当然），每个个体通过 hasHusband 属性最多能够关联另外一个个体。这类属性称作*函数型*（functional），如下描述：

**Functional-Style Syntax**

```
FunctionalObjectProperty(  :hasHusband )
```

注意，这个陈述不要求每个个体有一个丈夫，它仅仅陈述了不能多于一个。另外，如果我们有一个陈述说 Mary 的丈夫是 James，而又有一个陈述说 Mary 的丈夫是 Jim，可以推论 Jim 和 James 必须指的是同一个个体。

也有可能指出一个给定属性的逆是函数型：

**Functional-Style Syntax**

```
InverseFunctionalObjectProperty(  :hasHusband )
```

这表明了一个个体可以是至多一个其他个体的丈夫。这个例子也表明了函数型和反函数型之间的区别，在一夫多妻制下，前一条公理是有效的而后者不是。

现在来看看属性 hasAncestor，这个属性意味着不论何时 A 是 B 的直系后裔，个体 A 都和 B 关联。显然，属性 hasParent 是 hasAncesor 的一个“特例”，由此它可以被定义为一个子属性。而且，如果能“自动地”包含父母的父母（以及父母的父母的父母）也很好。这个可以通过将 hasAncestor 定义为*传递性*（transitive）属性来实现。对于某个个体 B，每当传递属性关联个体 A 和 B 并且关联 B 和 C，它也关联个体 A 和 C。

**Functional-Style Syntax**

```
TransitiveObjectProperty(  :hasAncestor )
```

## 6.2 属性链

前面一节的最后一个例子意味着，当有 hasParent 属性的一个链时，存在一个属性 hasAncestor。我们可能想要更具体一点定义，比如说，用属性 hasGrandparent 代替。从技术上说，这意味着我们想用 hasGrandparent 关联所有通过两个 hasParent 属性的链来关联的个体。与前面 hasAncestor 例子形成对比，我们不希望 hasParent 是 hasGrandparent 的特例，也不想用 hasGrandparent 指曾祖父母等。我们可以表示，每一个这样的链必须通过一个 hasGrandparent 属性涵盖，如下：

**Functional-Style Syntax**

```
SubObjectPropertyOf(
  ObjectPropertyChain(  :hasParent  :hasParent )
   :hasGrandparent
)
```

## 6.3 键

在 OWL2 里，一个（数据或对象）属性集合能够被指定为一个类表达式的键。这意味着，该类表达式的每一个具名实例，都可以用这些属性获得的关于该实例的值的集合来唯一标识。

一个简单的例子就是，一个人可以由她的社会安全号识别：

**Functional-Style Syntax**

```
HasKey(  :Person () (  :hasSSN ) )
```

# 7 数据类型的高级使用

在 4.8 节，我们知道个体可以被赋予数值信息，基本上是通过数据类型属性连接它们和数据值，就像对象属性关联其他领域个体。实际上，这些相似之处延伸到很多数据类型高级特性的使用上。

首先，数据值被分组为数据类型，我们已经在 4.8 节看到一个数据类型属性的值域限制如何可以用来指明该属性可以关联到的值的种类。此外，可以通过约束或组合现有的数据类型来表示和定义新的数据类型。数据类型通过所谓的*分面（facets）*(借用自 XML Schema Datatype [*XML Schema Datatypes*])进行限制。接下来的例子，我们通过限定整数数据类型的值在（包含）0 到 150 之间，为一个人的年龄定义一个新数据类型。

**Functional-Style Syntax**

```
DatatypeDefinition(
   :personAge
  DatatypeRestriction( xsd:integer
    xsd:minInclusive "0"^^xsd:integer
    xsd:maxInclusive "150"^^xsd:integer
  )
)
```

同样，数据类型也可以被组合，就像类的补集、交集和并集。因此，假设我们已经定义了一个数据类型 minorAge，我们可以通过从 personAge 排除 minorAge 的所有数据值，定义数据类型 majorAge。

**Functional-Style Syntax**

```
DatatypeDefinition(
   :majorAge
  DataIntersectionOf(
     :personAge
    DataComplementOf(  :minorAge )
  )
)
```

而且，一个新的数据类型也可以仅仅通过枚举它所包含的数据值而创建。

**Functional-Style Syntax**

```
DatatypeDefinition(
   :toddlerAge
  DataOneOf( "1"^^xsd:integer "2"^^xsd:integer )
)
```

在 6.1 节，我们见到了描述对象属性特征的多种方式。其中一些也可用于数据类型属性。例如，我们可以通过描述数据对象属性 hasAge 为函数型，表示每个人仅仅有一个年龄：

**Functional-Style Syntax**

```
FunctionalDataProperty(  :hasAge )
```

新类可以通过施加在数据类型属性上的限制来定义。下面的例子定义了青少年类为所有年龄在 13 到 19 岁之间的个体。

**Functional-Style Syntax**

```
SubClassOf(
   :Teenager
  DataSomeValuesFrom(  :hasAge
    DatatypeRestriction( xsd:integer
      xsd:minExclusive "12"^^xsd:integer
      xsd:maxInclusive "19"^^xsd:integer
    )
  )
)
```

# 8 文档信息和注释

接下来，我们描述的 OWL2 特性不会给本体中指定的“逻辑”知识带来实际贡献。这些特性用来为本体自身、公理甚至单个的实体提供额外信息。

## 8.1 注释公理和实体

在很多情况下，我们希望用信息配备我们的 OWL 本体的某些部分，这些信息实际上不描述领域本身，但是谈及有关领域的描述。OWL 为此目的提供了注释。OWL 注释简单地把属性-值对与本体的部分或整个本体自身联系在一起。甚至注释本身也可以加注释。注释信息实际上不是本体的逻辑意义的一部分。所以，

例如，我们可以为我们本体中的一个类增加信息，对其含义给出一个自然语言描述。

**Functional-Style Syntax**

```
 AnnotationAssertion( rdfs:comment  :Person "Represents the set of all people." )
```

下面是一个带有注释的公理的例子。

**Functional-Style Syntax**

```
SubClassOf(
  Annotation( rdfs:comment "States that every man is a person." )
   :Man
   :Person
)
```

这种注释经常在工具中使用，提供自然语言文本，显示在帮助界面中。

## 8.2 本体管理

在 OWL 中，有关一个主题的一般信息几乎总是聚集成一个本体，然后被各种应用程序使用。我们也可以为 OWL 本体提供一个名字，通常就是本体文档位于网络中的位置。如果有关一个主题的特定信息被不同的应用程序使用，也可以放置在本体中。

**Functional-Style Syntax**

```
Ontology(<http://example.com/owl/families>
  ...
)
```

我们把 OWL 本体放进 OWL 文档，然后这些文档被放置到本地文件系统或万维网上。除了包含一个 OWL 本体，OWL 文档也包含有关通常用在 OWL 本体中的简短名字（例如 Person）转换为 IRIs 的信息（通过为前缀提供展开）。于是，IRI 就是一串前缀展开和引用。

到目前为止，在我们的例子中已经使用了大量的前缀，包括xsd和空前缀。前缀xsd已被用在XML Schema数据类型的简约名字中，其IRI已经由XML Schema推荐标准确定。因此我们必须使用xsd的标准展开（也就是http://www.w3.org/2001/XMLSchema#）。我们为其他前缀选择的展开方式会影

响我们本体中的类、属性和个体的命名，也会影响本体自身的命名。如果我们把本体放到网上，就应该选择一个展开方式，让它在我们控制的网络的某一部分上，这样就不会意外地用到别人的名字。（这里我们使用了一个没有人控制的虚构位置。）两种基于XML的语法需要用于内置名称的命名空间，也可以为命名空间使用XML实体。一般来说，应该要注意：可用的缩写机制及其特定语法在OWL的每一个序列化中是不同的，甚至在用到类似关键词的案例中（也是这样）。

**Functional-Style Syntax**

```
Prefix(:=<http://example.com/owl/families/>)
Prefix(otherOnt:=<http://example.org/otherOntologies/families/>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)

Ontology(<http://example.com/owl/families>
   ...
)
```

在 OWL 中重用已被存储在其他本体中的一般信息很常见。不需要复制这个信息，OWL 允许使用引入陈述来引入在其他本体中的整个本体的内容，如下：

**Functional-Style Syntax**

```
Import( <http://example.org/otherOntologies/families.owl> )
```

因为语义 Web 和本体构建是分布式的，本体为同样的概念、属性或个体使用不同的名称很常见。正如我们所见到的，OWL 中有几个结构可以用来陈述不同名称指向同一个类、属性或个体，所以，例如，我们可以将我们本体中使用的名称绑定到一个引入本体所使用的名称上（而不是冗长乏味地重命名实体），如下：

**Functional-Style Syntax**

```
SameIndividual(  :John otherOnt:JohnBrown )
SameIndividual(  :Mary otherOnt:MaryBrown )
EquivalentClasses(  :Adult otherOnt:Grownup )
EquivalentObjectProperties(  :hasChild otherOnt:child )
EquivalentDataProperties(  :hasAge otherOnt:age )
```

## 8.3 实体声明

为了帮助管理本体，OWL 有声明的概念。基本思想就是认为在一个本体中要声明每一个类、属性或个体，然后，它就能够在这个本体中使用，且其他本体可以引用这个本体。

在曼彻斯特语法中，声明是隐式的。如果需要的话，提供关于一个类、属性或个体信息的结构（constructs），隐式地声明那个类、属性或个体。其他的语法有显式的声明：

**Functional-Style Syntax**

```
Declaration( NamedIndividual(  :John ) )
Declaration( Class(  :Person ) )
Declaration( ObjectProperty(  :hasWife ) )
Declaration( DataProperty(  :hasAge ) )
```

然而，一个 IRI 可能同时表示不同的实体类型（例如既是一个个体又是一个类）。这种可能性（称作“双关（punning）”）已被引入，以允许一定数量的元建模，第 9 节我们给出了这样一个例子。不过，OWL2 在使用和重用名称上仍需要一些规则。为了允许一个可读性更好的语法和其他技术原因，OWL2 DL 要求一个名称不能用于一个以上的属性类型（对象、数据类型或注释属性），一个 IRI 也不能既表示一个类又表示一个数据类型。此外，“内置”名称（例如那些 RDF、RDFS 还有 OWL 的各种语法使用的名称）不能在 OWL 中随意使用。

# 9 OWL2 DL 和 OWL2 Full

在 OWL2 中，对本体指定含义有两种可选的方式，分别是所谓的直接模型理论语义[*OWL 2 Direct Semantics*]和基于 RDF 的语义[*OWL 2 RDF-Based Semantics*]。非正式地，概念“OWL2 DL”用来指使用直接语义解释 OWL2 本体，当考虑基于 RDF 的语义时，则使用概念“OWL2 Full”。OWL2 函数式语法文档[*OWL 2 Specification*]另外列出了一些条件，满足这些条件的 OWL2 本体才是 OWL2 DL 本体。

直接模型理论语义[*OWL 2 Direct Semantics*]以一种描述逻辑[*Description Logics*]的方式为 OWL2 提供含义。基于 RDF 的语义[*OWL 2 RDF-Based Semantics*]是对 RDFS 语义 [*RDF Semantics*]的扩展，并建立在将 OWL2 本体视为 RDF 图的基础上。

当考虑本体的时候，这两个语义间的差异一般是很微小的。事实上，给定一个 OWL2 DL 本体，使用直接语义得到的推论，在基于 RDF 的语义之下仍然有效，请参考“基于 RDF 的语义”文档[*OWL 2 RDF-Based Semantics*]的 Section 7.2 中的对应定理（correspondence theorem）。二者的两个主要区别是：基于直接模型理论语义的注释没有形式化的含义，而基于 RDF 的语义有一些额外的由 RDF 全集视图（RDF view of the universe）产生的推论。

从概念上讲，我们可以从两个方面考虑 OWL2 DL 和 OWL2 Full 间的区别：

- OWL DL 可以看作是 OWL2 Full 的一个句法上受到限制的版本，设计限制的目的是为了让实现更简单。实际上，因为 OWL2 Full（基于 RDF 语义）是不可判定的，OWL2 DL（基于直接模型理论语义）使得编写一个原则上能够返回所有“是或否”答案的推理器（受资源约束）成为可能。由于它的设计，有若干具备产品质量的推理器覆盖了整个 OWL2 DL 语言（基于直接模型理论语义）。针对基于 RDF 语义的 OWL 2 Full 则没有这样的推理器。

- OWL2 Full 可以看作是 RDFS 最直接的扩展。基于 RDF 语义的 OWL2 Full 遵循 RDFS 语义以及通用的句法原理（即一切事物都是一个三元组并且完全反映语言）。

当然，这两个语义一起设计，因此也影响了彼此。例如，OWL2 的一个设计目的就是让 OWL2 DL 在句法上更接近 OWL2 Full（即，允许更多的 RDF 图/OWL2 Full 本体成为合法的 OWL2 DL 本体）。这导致了所谓的*双关（punning）*被合并进了 OWL2，比如，一个类和一个个体使用相同的 IRI 作为名称。这个例子的用法如下所示，陈述了 John 是一个父亲而父亲是一个社会角色：

**Functional-Style Syntax**

```
ClassAssertion(  :Father  :John )
ClassAssertion(  :SocialRole  :Father )
```

注意第一个陈述，Father 被用作一个类，而在第二个陈述中它被用作一个个体。在这个意义上，SocialRole 充当 Father 类的一个元类（metaclass）。

在 OWL1 中，一个包含这两个陈述的文档将是一个 OWL1 Full 文档，但不是一个 OWL1 DL 文档。然而，在 OWL2 DL 中这是被允许的。但必须注意到，OWL2 DL 的直接模型理论语义通过将 Father 类和 Father 个体理解为同一 IRI 的两个不同视图去适应它，也就是说，在进行语义解释时它们被区别对待。

# 10 OWL2 配置语言

除了 OWL2 DL 和 OWL2 Full 之外，OWL2 指定了三种配置语言。一般来说，OWL2 是一种非常具有表达力的语言（不管是在计算上还是对用户），因此很难实现好并使用其进行工作。这些附加的配置语言被设计成易于接受的 OWL2 子集，可满足各种应用程序的需要。正如 OWL2 DL，这些配置语言的主要要求是计算上的考虑（并且它们都具备强大的可扩展性，更容易在给定现有技术上实现），但是有很多具备良好计算性能的 OWL2 子集。已选定的 OWL2 配置语言被认为是已经具有了大量的用户社区，尽管有几个其他的配置语言没有被包含进

来并且人们应该会期待更多配置语言出现。文档[*OWL 2 Profiles*]为指定附加配置语言提供了一个清晰的模板。

为了保证可扩展的推理，现有的配置语言共享一些涉及其表达力的限制。一般而言，它们不允许否定和析取，因为这些结构会使推理复杂化并且已经证明建模时鲜有需要。例如，没有一个配置语言可以指定每一个人是男性或者女性。配置语言更具体的建模限制将在单个配置语言的章节中论述。

我们讨论每一个配置语言及其设计原理，并为用户选择哪个配置语言进行工作提供一些指导。请注意这个讨论是不全面的，也不可能全面。任何一种决策都必须部分基于可用的工具以及如何适应你的系统或工作流程的其余部分。

大体上，不同的配置语言可以从句法上进行区分，因为在各种配置语言之间存在包含关系。例如，OWL2 DL 可以被看作是 OWL Full 的一个句法片段，OWL2 QL 是 OWL2 DL 的一个句法片段（因此也是 OWL Full 的句法片段）。（但）下面这些配置语言（彼此之间）都不是另一个配置语言的子集。理想情况下，可以在子配置语言上使用一个符合父配置语言的推理器（或者其他工具），且获得的结果没有变化。对于与 OWL2 DL 相关的配置语言 OWL2 EL 和 OWL2 QL 而言，这个原则确实成立：每个符合 OWL2 DL 的推理器都是一个 OWL2 EL 推理器和 OWL2 QL 推理器（但是可能在性能上不同，因为 OWL2 DL 推理器面向的是更通用的案例集合）。

## 10.1 OWL2 EL

使用 OWL2 EL 非常类似于使用 OWL2 DL：在一个子类（subClassOf）陈述的两边都能使用类表达式甚至推断这个关系。对于很多大型的、面向类表达式的本体，只要稍作简化就可以获得一个 OWL2 EL 本体，并且保留原本体的大部分含义。

OWL2 EL 是随着大型生物健康本体而设计的（例如 SNOMED CT、NCI 叙词表和 Galen）。这类本体的共同特征是包含复杂的结构描述（例如，根据它们包含和被包含在什么部位来定义某些身体部位，或者沿着部分-子部分（part-subpart）关系传播疾病），巨量的类，大量使用分类来管理术语，将结果术语应用到巨量数据上。因此，OWL2 EL 拥有一个具备相对表达力的类表达式语言，并且它对类表达式如何可以用在公理中没有限制。它也拥有相当有表达力的属性表达式，包括属性链，但排除逆属性。

OWL2 EL 的合理使用明显不限于生物健康领域：正如其他配置语言，OWL2 EL 是独立于领域的。然而，当你的领域和你的应用程序需要识别结构上复杂的对象时，OWL2 EL 作用明显。这类领域包括系统配置、产品库存和许多科学领域。

除了否定和析取，OWL2 EL 也不允许对属性使用全称量化。因此，不能陈述像“富人的所有孩子都富有”这样的命题。而且，因为各种角色逆都不可用，所以没有办法说明，例如 parentOf 和 childOf 是彼此的逆。

缩略词 EL 反映了该配置文件是基于所谓的描述逻辑 EL 家族 [*EL++*]的；它们是主要提供存在量化变量的语言。

下面的例子使用了一些在 OWL2 EL 中可用的典型建模特性：

**Functional-Style Syntax**

```
SubClassOf(
   :Father
  ObjectIntersectionOf(  :Man  :Parent )
)

EquivalentClasses(
   :Parent
  ObjectSomeValuesFrom(
     :hasChild
     :Person
  )
)

EquivalentClasses(
   :NarcisticPerson
  ObjectHasSelf(  :loves )
)

DisjointClasses(
   :Mother
   :Father
   :YoungChild
)

SubObjectPropertyOf(
  ObjectPropertyChain(  :hasFather  :hasBrother )
   :hasUncle
)

NegativeObjectPropertyAssertion(
   :hasDaughter
   :Bill
   :Susan
```

```
)
```

## 10.2 OWL 2 QL

OWL2 QL 可以使用标准关系型数据库技术（例如 SQL）来实现，仅需要依据类公理扩展查询。这意味着它可以与 RDBMSs 紧密结合并且受益于它们的健壮实现和多用户特性。此外，它可以不必“接触数据”即可实现，从而真正作为一个翻译/预处理层。在表达力上，它可以表示实体关系和 UML 图（至少是那些有功能限制的）的主要特性。因此，它既适合表示数据库模式也适合通过查询重写整合它们。其结果是，它也可以直接作为一个高层次数据库模式语言使用，尽管用户可能更喜欢基于语法的图表。

OWL2 QL 也捕获许多 RDFS 的常用特性并由此进行了小扩展，如逆属性和子属性层次。OWL2 QL 非对称地限制类公理，也就是，你可以把结构（constructs）用作一个子类，但你不能将它用作超类。

在其他结构（constructs）中，OWL2 QL 不允许对一个类表达式使用角色的存在量化，例如，它可以描述每个人都有 parent，但不能描述每个人都有一个 female parent。而且不支持属性链公理。

缩略词 QL 反映了这样一个事实：在这个配置语言中，通过将查询重写成标准关系型查询语言（**Q**uery **L**anguage）[*DL-Lite*]就可以实现查询应答。

下面的例子使用了一些在 OWL2 QL 中可用的典型建模特性。第一个公理陈述了对于每一个没有孩子的人，不存在任何人称这个人为父母：

**Functional-Style Syntax**

```
SubClassOf(
   :ChildlessPerson
  ObjectIntersectionOf(
     :Person
    ObjectComplementOf(
      ObjectSomeValuesFrom(
        ObjectInverseOf(  :hasParent )
        owl:Thing
      )
    )
  )
)

DisjointClasses(
   :Mother
```

```
    :Father
    :YoungChild
)

DisjointObjectProperties(
    :hasSon
    :hasDaughter
)

SubObjectPropertyOf(
    :hasFather
    :hasParent
)
```

## 10.3 OWL 2 RL

OWL2 RL 配置语言面向需要可扩展推理又不能牺牲太多表达能力的应用程序。它被设计用来适应可以牺牲语言的完整表达力以换取效率的 OWL2 应用程序，以及需要一些来自 OWL2 的额外表达力的 RDF(S)应用程序。这可以通过定义一个 OWL2 句法子集来达到目标，它可以使用基于规则的技术实现，并且以一阶蕴涵的形式展现一个基于 RDF 语义的 OWL2 部分公理化，用作这样一个实现的基础。

OWL2 RL 的适当的基于规则的实现可以与任意的 RDF 图一起使用。因此，对于丰富 RDF 数据，OWL2 RL 是理想的，特别是当数据必须由附加规则修改的时候。然而，从建模的角度来看，这个会促使我们远离使用类表达式：OWL2 RL 保证我们不能（容易地）在父类表达式里谈及未知的个体（此限制由规则特性产生）。与 OWL2 QL 相比，当你已经把数据整理成 RDF 并且将它作为 RDF 工作的时候，OWL2 RL 效果更好。

在其他结构中，OWL2 RL 不允许这样的陈述：一个个体的存在迫使另一个个体的存在。例如，陈述“每个人有一个父母”不可以在 OWL2 RL 中表示。

OWL2 RL 非对称地限制类公理，也就是，你可以把结构（constructs）用作一个子类，但你不能将它用作超类。

缩略词 RL 反映了这样的一个事实：在这个配置语言中的推理可以使用标准的规则语言（**R**ule **L**anguage）[*DLP*]来实现。

下面的例子使用了一些在 OWL2 RL 中可用的典型建模特性。第一个公理（有点人为编造）陈述：对于 Mary、Bill 和 Meg（都是女性）中的每一个，下

述成立：她是一个有至多一个孩子的父母，所有她的孩子（如果她有的话）都是女性。

**Functional-Style Syntax**

```
SubClassOf(
  ObjectIntersectionOf(
    ObjectOneOf(  :Mary  :Bill  :Meg )
     :Female
  )
  ObjectIntersectionOf(
     :Parent
    ObjectMaxCardinality( 1  :hasChild )
    ObjectAllValuesFrom(  :hasChild  :Female )
  )
)

DisjointClasses(
   :Mother
   :Father
   :YoungChild
)

SubObjectPropertyOf(
  ObjectPropertyChain(  :hasFather  :hasBrother )
   :hasUncle
)
```

# 11 OWL 工具

为了使用 OWL 本体进行工作，工具支持是必要的。主要有两种类型的工具专注于本体生命周期的两个主要阶段：*本体编辑器*用来创建和编辑本体，而*推理器*用来查询本体以获得隐性知识，即，它们决定一个正在考虑中的陈述是否是一个本体的逻辑结果。

目前用得最广泛的 OWL 编辑器是 Protégé，一个由斯坦福大学开发的免费开源编辑框架。凭借其开放的插件结构，它允许特殊用途本体编辑组件的简易集成。其他编辑器包括 TopQuadrant 的商用 TopBraid Composer，以及开源系统 SWOOP 和 NeOn-Toolkit。

有若干用于 OWL DL 的推理器，其支持的推理特性覆盖范围稍有不同。对于其中的一些，OWL2 一致（OWL2 conformance）目前已有规划，相应的实现

正在进行中。Test Suite Status 文档罗列了，下面提及的一些推理器在何种程度上遵从测试用例。

对于 OWL DL 内的推理，最突出的系统是曼彻斯特大学的 Fact++，牛津大学计算实验室的 Hermit，“Clark & Parsia, LLC”的 Pellet 和 Racer Systems 的 RacerPro。

除了那些支持所有 OWL DL 的一般用途的推理器，还有为易处理的 OWL 配置语言定制的推理系统。德累斯顿工业大学（Dresden University of Technology）的 CEL 支持 OWL EL。罗马 Sapienza 大学（Sapienza Università di Roma）的 QuOnto 支持 OWL QL。ORACLE 11g 支持 OWL RL。

开源的 OWL API 作为目前最重要的 OWL 开发工具，扮演着相当重要的角色。

必须提到，在本文档创建时，有几个 OWL 工具还在开发中，因此当前的概述应该看做是此开发的快照而不是最新的概述。OWL 工具的全面列表可以在 semanticweb.org 和 ESW-Wiki 上找到。

# 12 下一步读什么

这篇简短的入门只触及了 OWL 的表面。有很多更长和内容更丰富的关于 OWL 以及如何使用 OWL 工具的教程可以在网上搜索到。阅读其中一个文档并使用工具构建一个 OWL 本体，可能是获得 OWL 应用知识的最佳方式。如果要学习更多的有关 OWL 的基础知识，我们建议首先翻阅一本教科书[*FOST*]然后再追踪其中引用的原始文章。

这篇简短的入门也不是 OWL 的规范定义。OWL 语法的规范定义以及每一个 OWL 结构的资料性描述可以查看“OWL2 结构化规范与函数式语法”文档[*OWL 2 Specification*] 。

当要寻找有关一个特定语言特性的信息时，OWL2 快速参考指南[*OWL 2 Quick Guide*]是一个方便的参考。

对于那些对更形式化的文档感兴趣的读者，OWL2 的形式化含义可以在 OWL2 语义文档[*OWL 2 Direct Semantics*] [*OWL 2 RDF-Based Semantics*]中找到。

OWL 语义与 RDF 三元组的映射请见“映射到 RDF 图”文档[*OWL 2 RDF Mapping*]。

# 13 附录：完整的样本本体

这里包含的是完整的 OWL 本体样例。本体公理按照其使用的顶级表达特性排序。此外，我们遵循一种常用的排序，依次是本体和声明信息、有关属性的信息、类和数据类型、个体。

### Functional-Style Syntax

```
Prefix(:=<http://example.com/owl/families/>)
Prefix(otherOnt:=<http://example.org/otherOntologies/families/>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Ontology(<http://example.com/owl/families>
  Import( <http://example.org/otherOntologies/families.owl> )

  Declaration( NamedIndividual(  :John ) )
  Declaration( NamedIndividual(  :Mary ) )
  Declaration( NamedIndividual(  :Jim ) )
  Declaration( NamedIndividual(  :James ) )
  Declaration( NamedIndividual(  :Jack ) )
  Declaration( NamedIndividual(  :Bill ) )
  Declaration( NamedIndividual(  :Susan ) )
  Declaration( Class(  :Person ) )
  AnnotationAssertion( rdfs:comment  :Person "Represents the set of all
people." )
  Declaration( Class(  :Woman ) )
  Declaration( Class(  :Parent ) )
  Declaration( Class(  :Father ) )
  Declaration( Class(  :Mother ) )
  Declaration( Class(  :SocialRole ) )
  Declaration( Class(  :Man ) )
  Declaration( Class(  :Teenager ) )
  Declaration( Class(  :ChildlessPerson ) )
  Declaration( Class(  :Human ) )
  Declaration( Class(  :Female ) )
  Declaration( Class(  :HappyPerson ) )
  Declaration( Class(  :JohnsChildren ) )
  Declaration( Class(  :NarcisticPerson ) )
  Declaration( Class(  :MyBirthdayGuests ) )
  Declaration( Class(  :Dead ) )
  Declaration( Class(  :Orphan ) )
  Declaration( Class(  :Adult ) )
  Declaration( Class(  :YoungChild ) )
  Declaration( ObjectProperty(  :hasWife ) )
  Declaration( ObjectProperty(  :hasChild ) )
  Declaration( ObjectProperty(  :hasDaughter ) )
  Declaration( ObjectProperty(  :loves ) )
```

```
Declaration( ObjectProperty(  :hasSpouse ) )
Declaration( ObjectProperty(  :hasGrandparent ) )
Declaration( ObjectProperty(  :hasParent ) )
Declaration( ObjectProperty(  :hasBrother ) )
Declaration( ObjectProperty(  :hasUncle ) )
Declaration( ObjectProperty(  :hasSon ) )
Declaration( ObjectProperty(  :hasAncestor ) )
Declaration( ObjectProperty(  :hasHusband ) )
Declaration( DataProperty(  :hasAge ) )
Declaration( DataProperty(  :hasSSN ) )
Declaration( Datatype(  :personAge ) )
Declaration( Datatype(  :majorAge ) )
Declaration( Datatype(  :toddlerAge ) )

SubObjectPropertyOf(  :hasWife  :hasSpouse )
SubObjectPropertyOf(
  ObjectPropertyChain(  :hasParent  :hasParent )
   :hasGrandparent
)
SubObjectPropertyOf(
  ObjectPropertyChain(  :hasFather  :hasBrother )
   :hasUncle
)
SubObjectPropertyOf(
   :hasFather
   :hasParent
)

EquivalentObjectProperties(  :hasChild otherOnt:child )
InverseObjectProperties(  :hasParent  :hasChild )
EquivalentDataProperties(  :hasAge otherOnt:age )
DisjointObjectProperties(  :hasSon  :hasDaughter )
ObjectPropertyDomain(  :hasWife  :Man )
ObjectPropertyRange(  :hasWife  :Woman )
DataPropertyDomain(  :hasAge  :Person )
DataPropertyRange(  :hasAge xsd:nonNegativeInteger )

SymmetricObjectProperty(  :hasSpouse )
AsymmetricObjectProperty(  :hasChild )
DisjointObjectProperties(  :hasParent  :hasSpouse )
ReflexiveObjectProperty(  :hasRelative )
IrreflexiveObjectProperty(  :parentOf )
FunctionalObjectProperty(  :hasHusband )
```

```
InverseFunctionalObjectProperty(  :hasHusband )
TransitiveObjectProperty(  :hasAncestor )
FunctionalDataProperty(  :hasAge )

SubClassOf(  :Woman  :Person )
SubClassOf(  :Mother  :Woman )
SubClassOf(
   :Grandfather
  ObjectIntersectionOf(  :Man  :Parent )
)
SubClassOf(
   :Teenager
  DataSomeValuesFrom(  :hasAge
    DatatypeRestriction( xsd:integer
      xsd:minExclusive "12"^^xsd:integer
      xsd:maxInclusive "19"^^xsd:integer
    )
  )
)
SubClassOf(
  Annotation( rdfs:comment "States that every man is a person." )
   :Man
   :Person
)
SubClassOf(
   :Father
  ObjectIntersectionOf(  :Man  :Parent )
)
SubClassOf(
   :ChildlessPerson
  ObjectIntersectionOf(
     :Person
    ObjectComplementOf(
      ObjectSomeValuesFrom(
        ObjectInverseOf(  :hasParent )
        owl:Thing
      )
    )
  )
)
SubClassOf(
  ObjectIntersectionOf(
    ObjectOneOf(  :Mary  :Bill  :Meg )
```

```
        :Female
    )
    ObjectIntersectionOf(
        :Parent
      ObjectMaxCardinality( 1  :hasChild )
      ObjectAllValuesFrom(  :hasChild  :Female )
    )
  )

  EquivalentClasses(  :Person  :Human )
  EquivalentClasses(
     :Mother
    ObjectIntersectionOf(  :Woman  :Parent )
  )
  EquivalentClasses(
     :Parent
    ObjectUnionOf(  :Mother  :Father )
  )
  EquivalentClasses(
     : childlessPerson
    ObjectIntersectionOf(
        :Person
      ObjectComplementOf(  :Parent )
    )
  )
  EquivalentClasses(
     :Parent
    ObjectSomeValuesFrom(  :hasChild  :Person )
  )
  EquivalentClasses(
     :HappyPerson
    ObjectIntersectionOf(
        ObjectAllValuesFrom(  :hasChild  :HappyPerson )
        ObjectSomeValuesFrom(  :hasChild  :HappyPerson )
    )
  )
  EquivalentClasses(
     :JohnsChildren
    ObjectHasValue(  :hasParent  :John )
  )
  EquivalentClasses(
     :NarcisticPerson
    ObjectHasSelf(  :loves )
```

```
)
EquivalentClasses(
   :MyBirthdayGuests
  ObjectOneOf(  :Bill  :John  :Mary)
)
EquivalentClasses(
   :Orphan
  ObjectAllValuesFrom(
    ObjectInverseOf(  :hasChild )
     :Dead
  )
)
EquivalentClasses(  :Adult otherOnt:Grownup )
EquivalentClasses(
   :Parent
  ObjectSomeValuesFrom(
     :hasChild
     :Person
  )
)

DisjointClasses(  :Woman  :Man )
DisjointClasses(
   :Mother
   :Father
   :YoungChild
)
HasKey(  :Person () (  :hasSSN ) )

DatatypeDefinition(
   :personAge
  DatatypeRestriction( xsd:integer
    xsd:minInclusive "0"^^xsd:integer
    xsd:maxInclusive "150"^^xsd:integer
  )
)
DatatypeDefinition(
   :majorAge
  DataIntersectionOf(
     :personAge
    DataComplementOf(  :minorAge )
  )
)
```

```
DatatypeDefinition(
   :toddlerAge
  DataOneOf( "1"^^xsd:integer "2"^^xsd:integer )
)

ClassAssertion(  :Person  :Mary )
ClassAssertion(  :Woman  :Mary )
ClassAssertion(
  ObjectIntersectionOf(
     :Person
    ObjectComplementOf(  :Parent )
  )
   :Jack
)
ClassAssertion(
  ObjectMaxCardinality( 4  :hasChild  :Parent )
   :John
)
ClassAssertion(
  ObjectMinCardinality( 2  :hasChild  :Parent )
   :John
)
ClassAssertion(
  ObjectExactCardinality( 3  :hasChild  :Parent )
   :John
)
ClassAssertion(
  ObjectExactCardinality( 5  :hasChild )
   :John
)
ClassAssertion(  :Father  :John )
ClassAssertion(  :SocialRole  :Father )

ObjectPropertyAssertion(  :hasWife  :John  :Mary )
NegativeObjectPropertyAssertion(  :hasWife  :Bill  :Mary )
NegativeObjectPropertyAssertion(
   :hasDaughter
   :Bill
   :Susan
)
DataPropertyAssertion(  :hasAge  :John "51"^^xsd:integer )
NegativeDataPropertyAssertion(  :hasAge  :Jack "53"^^xsd:integer )
```

```
   SameIndividual(  :John  :Jack )
   SameIndividual(  :John otherOnt:JohnBrown )
   SameIndividual(  :Mary otherOnt:MaryBrown )
   DifferentIndividuals(  :John  :Bill )
)
```

# 14 附录：变动日志（资料性）

## 14.1 相对于建议推荐标准的变动

本节总结了该文档相对于 2009 年 9 月 22 日的建议推荐标准的变动。

- 做了少量的编辑性阐释和改进。

## 14.2 相对于上一征求意见版本的变动

本节总结了该文档相对于 2009 年 6 月 11 日候选推荐标准的变动。

- 修改了一些例子中存在的错误。
- 修改了示例本体以保持一致和句法正确。
- 添加了一节“OWL2 工具”。
- 添加了到其他 OWL2 文档的各种链接。
- 新添加了一条注释，指出属性的非对称（asymmetric）是一个比不对称（non-symmetric）更强的概念，对称（symmetric）是一个比不非对称（non-asymmetric）更强的概念。
- 新添加了关于配置语言名（profile names）起源的注释，指出没有配置语言是其他配置语言的子集。
- 对于 OWL2 语法的 Post Last Call 变更被合并进来。
- 做了少量的编辑性阐释和改进，小的更正和修改，以及装饰性变更。

# 15 致谢

OWL2 的开发始于 OWL1.1 成员提交（其本身是用户和开发者反馈的结果），尤其是在体验与研究方向工作组（OWLED）系列中积累的信息。该工作组也考虑了来自于 WebOnt 工作组（WebOnt Working Group）的待解决问题（postponed issues）。

本文档由 OWL 工作小组（见下）执笔，它的内容反映了作为一个整体的该工作小组内部的广泛讨论。编者向 Jie Bao （RPI），Michel Dumontier（卡尔顿大学），Christine Goldbreich（凡尔赛大学和 LIRMM），Henson Graves（洛

克希德马丁公司），Ivan Herman（W3C/ERCIM），Rinke Hoekstra（阿姆斯特丹大学），Doug Lenat（Cycorp），Deborah L. McGuinness（RPI），Alan Rector（曼彻斯特大学），Alan Ruttenberg（Science Commons），Uli Sattler（曼彻斯特大学），Michael Schneider （FZI）和 Mike Smith（Clark & Parsia）的仔细审阅和有益评论致以特别的谢意。

在本文档发布时，经常参加 OWL 工作组会议的与会者有：Jie Bao （RPI），Diego Calvanese（Bozen-Bolzano 大学），Bernardo Cuenca Grau（牛津大学计算实验室），Martin Dzbor（英国公开大学），Achille Fokoue（IBM 公司），Christine Golbreich（凡尔赛大学和 LIRMM），Sandro Hawke （W3C/MIT），Ivan Herman（W3C/ERCIM），Rinke Hoekstra（阿姆斯特丹大学），Ian Horrocks（牛津大学计算实验室），Elisa Kendall（Sandpiper 软件），Markus Krötzsch（FZI）， Carsten Lutz（不来梅大学），Deborah L. McGuinness （RPI），Boris Motik（牛津大学计算实验室），Jeff Pan（阿伯丁大学），Bijan Parsia（曼彻斯特大学），Peter F. Patel-Schneider（贝尔实验室，阿尔卡特公司），Sebastian Rudolph（FZI），Alan Ruttenberg（Science Commons），Uli Sattler（曼彻斯特大学），Michael Schneider（FZI），Mike Smith （Clark & Parsia），Evan Wallace （NIST），Zhe Wu（甲骨文公司）和 Antoine Zimmermann（DERI Galway）。我们还要感谢以前的工作组成员：Jeremy Carroll、Jim Hendler 和 Vipul Kashyap。

# 16 参考文献

**[Description Logics]**

*The Description Logic Handbook: Theory, Implementation, and Applications, second edition*. Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, eds. Cambridge University Press, 2007. Also see the *Description Logics Home Page*.

**[DLP]**

*Description Logic Programs: Combining Logic Programs with Description Logic*. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. in Proc. of the 12th Int. World Wide Web Conference (WWW 2003), Budapest, Hungary, 2003. pp.: 48–57

**[DL-Lite]**

*Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family*. Diego Calvanese, Giuseppe de Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati. J. of Automated Reasoning 39(3):385–429, 2007

**[EL++]**

*Pushing the EL Envelope*. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the 19th Joint Int. Conf. on Artificial Intelligence (IJCAI 2005), 2005

**[FOST]**

*Foundations of Semantic Web Technologies*. Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. Chapman & Hall/CRC, 2009, ISBN: 9781420090505.

**[OWL 2 Conformance]**

*OWL 2 Web Ontology Language: Conformance* Michael Smith, Ian Horrocks, Markus Krötzsch, Birte Glimm, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-conformance-20091027/. Latest version available at http://www.w3.org/TR/owl2-conformance/.

**[OWL 2 Manchester Syntax]**

*OWL 2 Web Ontology Language: Manchester Syntax* Matthew Horridge, Peter F. Patel-Schneider. W3C Working Group Note, 27 October 2009, http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-manchester-syntax/.

**[OWL 2 New Features and Rationale]**

*OWL 2 Web Ontology Language: New Features and Rationale* Christine Golbreich, Evan K. Wallace, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-new-features-20091027/. Latest version available at http://www.w3.org/TR/owl2-new-features/.

**[OWL 2 Profiles]**

*OWL 2 Web Ontology Language: Profiles* Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/. Latest version available at http://www.w3.org/TR/owl2-profiles/.

**[OWL 2 Quick Reference Guide]**

*OWL 2 Web Ontology Language: Quick Reference Guide* Jie Bao, Elisa F. Kendall, Deborah L. McGuinness, Peter F. Patel-Schneider, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/. Latest version available at http://www.w3.org/TR/owl2-quick-reference/.

**[OWL 2 RDF-Based Semantics]**

*OWL 2 Web Ontology Language: RDF-Based Semantics* Michael Schneider, editor. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-rdf-based-semantics/.

**[OWL 2 RDF Mapping]**

*OWL 2 Web Ontology Language: Mapping to RDF Graphs* Peter F. Patel-Schneider, Boris Motik, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-mapping-to-rdf-20091027/.

Latest version available at
http://www.w3.org/TR/owl2-mapping-to-rdf/.

**[OWL 2 Direct Semantics]**

*OWL 2 Web Ontology Language: Direct Semantics* Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-direct-semantics/.

**[OWL 2 Specification]**

*OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax* Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-syntax/.

**[OWL 2 XML Serialization]**

*OWL 2 Web Ontology Language: XML Serialization* Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/. Latest version available at http://www.w3.org/TR/owl2-xml-serialization/.

**[pD*]**

*Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*. Herman J. ter Horst. J. of Web Semantics 3(2–3):79–115, 2005

**[RDF Concepts]**

*Resource Description Framework (RDF): Concepts and Abstract Syntax*. Graham Klyne and Jeremy J. Carroll, eds. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. Latest version available as http://www.w3.org/TR/rdf-concepts/.

**[RDF Semantics]**

*RDF Semantics*. Patrick Hayes, ed., W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Latest version available as http://www.w3.org/TR/rdf-mt/.

**[RDF Turtle Syntax]**

*Turtle - Terse RDF Triple Language*. David Beckett and Tim Berners-Lee, 14 January 2008

**[RDF Syntax]**

*RDF/XML Syntax Specification (Revised)*. Dave Beckett, ed. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. Latest version available as http://www.w3.org/TR/rdf-syntax-grammar/.

**[RFC 3987]**

*RFC 3987: Internationalized Resource Identifiers (IRIs)*. M. Duerst and M. Suignard. IETF, January 2005, http://www.ietf.org/rfc/rfc3987.txt

**[SPARQL]**

*SPARQL Query Language for RDF*. Eric Prud'hommeaux and Andy Seaborne, eds. W3C Recommendation, 15 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. Latest version available as http://www.w3.org/TR/rdf-sparql-query/.

**[XML Schema Datatypes]**

*XML Schema Part 2: Datatypes Second Edition*. Paul V. Biron, and Ashok Malhotra, eds. W3C Recommendation 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. Latest version available as http://www.w3.org/TR/xmlschema-2/. This reference is to be considered a reference to XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes upon its expected publication as a W3C Recommendation (see Section 2.3 in OWL 2 Conformance). The (non-normative) version of the XSD 1.1 document available at publication time is the 30 April 2009 Candidate Recommendation.

## 2.D　OWL2 Web 本体语言快速参考指南

本文档《OWL2 Web 本体语言快速参考指南》是 W3C 发布的 ***OWL 2 Web Ontology Language Quick Reference Guide***（2009-10-27）的中文译本。文中若存在译法不当和错误之处，欢迎批评指正，请发邮件至：**zengxh@szu.edu.cn**，谢谢！

**翻译说明：**

- 本文档的英文版是唯一正式版本。此处的中文译本仅供学习与交流。
- 中文译本的内容是非正式的，仅代表译者的个人观点。
- 中文译本的内容会根据反馈意见随时进行修订。
- 中文译本同时通过 W3C Translations 网站发布。
- 转载本文，请注明译者和原链接。

**译者：**

曾新红（Xinhong Zeng），深圳大学图书馆 NKOS 研究室

蔡庆河（Qinghe Cai），深圳大学计算机与软件学院

**资助声明：**

本次翻译工作得到广东省哲学社会科学“十一五”规划项目（批准号：GD10CTS02）和国家社科基金项目“中文知识组织系统的形式化语义描述标准体系研究”（批准号：12BTQ045）的资助。

翻译时间：2011 年 9 月

发布时间：2012 年 9 月 25 日

---

# OWL2 Web 本体语言
# 快速参考指南
## W3C 推荐标准 2009 年 10 月 27 日

当前版本：

http://www.w3.org/TR/2009/REC-owl2-quick-reference-20091027/

最新版本(系列 2)：

http：//www.w3.org/TR/owl2-quick-reference/

最新推荐标准：

http：//www.w3.org/TR/owl-quick-reference

上一版本：

http://www.w3.org/TR/2009/PR-owl2-quick-reference-20090922/ (彩色标注

不同之处)
编者：
Jie Bao，伦斯勒理工学院
Elisa F. Kendall，Sandpiper 软件公司
Deborah L. McGuinness，伦斯勒理工学院
Peter F. Patel-Schneider，Bell Labs Research, Alcatel-Lucent
贡献者：
Li Ding，伦斯勒理工学院
Ankesh Khandelwal，伦斯勒理工学院

请参阅本文档的**勘误表**，那里可能会有一些规范的校正。
本文档也可以以如下的非规范格式查看：PDF 版本，参考卡片。
另见 译文。


---

# 摘要

OWL2 Web 本体语言，或者简略地记作 OWL2，是一个本体语言，用于带有形式化定义含义的语义网。OWL2 本体提供了类、属性、个体及数据值，以语义网文档的形式存储。OWL2 本体可以与以 RDF 记载的信息结合使用，并且 OWL2 本体本身也主要是以 RDF 文档的形式进行交换。OWL2 文档概述（Document Overview）阐述了 OWL2 的整体状态，因此，应当在阅读其他 OWL2 文档之前先阅读该文档。

本文档提供的是 OWL2 语言的非规范性快速参考指南。它也提供了到其他文档的链接，包括用于语言介绍与示例的 OWL2 入门、包含了更多函数式语法细节的 OWL2 结构化规范与函数式语法文档和用于新特性描述的 OWL2 新特性与原理文档。

# 本文档的状态

## 可能已被替代

*本节描述的是本文档在发布时的状态。其他的文档可能替代了该文档。在* http://www.w3.org/TR/*的 W3C technical reports index 中，可以找到当前的 W3C 出版物列表和该技术报告的最新版本。*

## XML Schema 数据类型依赖

根据定义，OWL2 使用的是 XML Schema Definition Language (XSD)中定义的数据类型。对本文档中的 OWL2 而言，XSD 的最新 W3C 推荐标准是版本 1.0，此时，版本 1.1 正在向推荐标准演进。OWL2 的设计已经利用了 XSD1.1 中的新数据类型和更清晰的注释，但是目前这些利用有一部分暂时搁置。特别地，OWL2 中基于 XSD1.1 的那些元素将被当作是*可选的*，直到 XSD1.1 成为 W3C 的推荐标准为止，详见 Conformance, section 2.3。等到 XSD1.1 发布为 W3C 的推荐标准时，这些元

素才会终止可选状态而与其他指定元素一样进入必备状态。

我们提议，目前开发人员和用户遵循 XSD 1.1 Candidate Recommendation。根据 Schema 和 OWL 工作组之间的讨论，当 XSD1.1 演进成为推荐标准时，我们并不希望任何实现会有必须的改动。

## 变动总结

自上一版本以来，并没有什么实质性的改变。若需了解细微的变动细节，请参见变动日志和彩色标注不同之处版本。

## 请发表意见

请将意见发送至 public-owl-comments@w3.org（公开文档）。虽然由 OWL 工作组执笔的本文档已经完成，但是您的意见依旧可能在勘误表或者未来的修订版中得到解决。欢迎开发人员在 public-owl-dev@w3.org(公开文档)公开讨论。

## 由 W3C 批准

*本文档已经由 W3C 成员、软件开发人员以及其他 W3C 小组和兴趣组织审查，并由 W3C 主管（Director）批准成为 W3C 推荐标准。这是一个稳定的文档，可以作为参考资料或被其他文档引用。W3C 在制作推荐标准过程中担任的角色，是要引起人们对该规范的注意并促进它的广泛应用。这提升了 Web 的功能性和交互性。*

## 专利

*本文档由遵循 5 February 2004 W3C Patent Policy 的小组完成。它只是一个提供信息的文档。W3C 维护着一个专利披露公开列表（public list of any patent disclosures），它与该组织的可交付成果一起制作；此页面也包含披露专利的说明。*

---

## 目录

# 1 名称（Names），前辍（Prefixes）与符号（Notation）

OWL2 中的名称是国际化资源标识符（IRI），通常简写为“prefix: localname”(前辍:本地名)。其中，“prefix”是展开为 IRI 的前辍名；而本地名则是名称的剩余部分。OWL2 中的标准前辍名有：

| 前辍名 | 展开形式 |
|---|---|
| rdf: | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs: | http://www.w3.org/2000/01/rdf-schema# |
| owl: | http://www.w3.org/2002/07/owl# |
| xsd: | http://www.w3.org/2001/XMLSchema# |

我们使用下表中的符号约定（notation conventions）*：

| 字母 | 含义 | 字母 | 含义 | 字母 | 含义 | 字母 | 含义 |
|---|---|---|---|---|---|---|---|
| C | 类表达式 | CN | 类名 | D | 数据值域 | DN | 数据类型名 |
| P | 对象属性表达式 | PN | 对象属性名 | R | 数据属性 | A | 注释属性 |
| a | 个体 | aN | 个体名 | _:a | 匿名个体（一个空节点标签） | v | 文本 |
| n | 非负整数 ** | f | 分面 | ON | 本体名 | U | IRI |
| s | IRI 或匿名个体 | t | IRI，匿名个体或文本 | p | 前辍名 | _:x | 空节点 |
| ($a_1$ … $a_n$) | RDF 列表 | | | | | | |

* 上述所有内容均可以有下标。** 是 "n"^^xsd:nonNegativeInteger 的缩略写法。

# 2 OWL2 结构与公理

对于一个 OWL2 DL 本体，对公理有一些全局性限制。

在下面的这些表中，第一列提供了与入门（Primer）的链接（如果适用），第二列提供了到函数式语法（Functional Syntax）的链接，第三列则给定了用 Turtle

语法表示的 RDF 三元组。

## 2.1 类表达式

### 预定义与具名类

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 具名类 (named class) | CN | CN |
| 通用类 (universal class) | owl:Thing | owl:Thing |
| 空类 (empty class) | owl:Nothing | owl:Nothing |

### 布尔连接与个体枚举

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 交集（intersection） | ObjectIntersectionOf($C_1$ … $C_n$) | _:x rdf:type owl:Class.<br>_:x owl:intersectionOf ( $C_1$ … $C_n$ ). |
| 并集（union） | ObjectUnionOf($C_1$ … $C_n$) | _:x rdf:type owl:Class.<br>_:x owl:unionOf ( $C_1$ … $C_n$ ). |
| 补集（complement） | ObjectComplementOf(C) | _:x rdf:type owl:Class.<br>_:x owl:complementOf C. |
| 枚举（enumeration） | ObjectOneOf($a_1$ … $a_n$) | _:x rdf:type owl:Class.<br>_:x owl:oneOf ( $a_1$ … $a_n$ ). |

### 对象属性限制

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 全称量化（universal） | ObjectAllValuesFrom(P C) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:allValuesFrom C |
| 存在量化（existential） | ObjectSomeValuesFrom(P C) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:someValuesFrom C |
| 个体值 (individual value) | ObjectHasValue(P a) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:hasValue a. |
| 本地自反（local reflexivity） | ObjectHasSelf(P) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:hasSelf "true"^^xsd:boolean. |
| 精确基数 (exact cardinality) | ObjectExactCardinality(n P) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:cardinality n. |
| 限定精确基数 (qualified exact cardinality) | ObjectExactCardinality(n P C) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:qualifiedCardinality n.<br>_:x owl:onClass C. |

| | | |
|---|---|---|
| 最大基数<br>(maximum cardinality) | ObjectMaxCardinality(n P) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:maxCardinality n. |
| 限定最大基数<br>(qualified maximum cardinality) | ObjectMaxCardinality(n P C) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:maxQualifiedCardinality n.<br>_:x owl:onClass C. |
| 最小基数<br>(minimum cardinality) | ObjectMinCardinality(n P) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:minCardinality n. |
| 限定最小基数<br>(qualified minimum cardinality) | ObjectMinCardinality(n P C) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty P.<br>_:x owl:minQualifiedCardinality n.<br>_:x owl:onClass C. |

**数据属性限制**

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 全称量化<br>（universal） | DataAllValuesFrom(R D) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:allValuesFrom D. |
| 存在量化<br>（existential） | DataSomeValuesFrom(R D) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:someValuesFrom D. |
| 文本值<br>（literal value） | DataHasValue(R v) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:hasValue v. |
| 精确基数<br>（exact cardinality ） | DataExactCardinality(n R) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:cardinality n. |
| 限定精确基数<br>（qualified exact cardinality ） | DataExactCardinality(n R D) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:qualifiedCardinality n.<br>_:x owl:onDataRange D. |
| 最大基数<br>(maximum cardinality) | DataMaxCardinality(n R) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:maxCardinality n. |
| 限定最大基数<br>(qualified maximum cardinality) | DataMaxCardinality(n R D) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:maxQualifiedCardinality n.<br>_:x owl:onDataRange D. |

| | | |
|---|---|---|
| 最小基数<br>(minimum cardinality) | DataMinCardinality(n R) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:minCardinality n. |
| 限定最小基数<br>(qualified minimum cardinality) | DataMinCardinality(n R D) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperty R.<br>_:x owl:minQualifiedCardinality n.<br>_:x owl:onDataRange D. |

**使用 n 元数据值域的限制**

下表中的‘$D^n$’是 n 元数据值域。

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| n 元全称量化<br>（n-ary universal） | DataAllValuesFrom($R_1$ … $R_n$ $D^n$) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperties ( $R_1$ … $R_n$ ).<br>_:x owl:allValuesFrom $D^n$. |
| n 元存在量化<br>（n-ary existential） | DataSomeValuesFrom($R_1$ … $R_n$ $D^n$) | _:x rdf:type owl:Restriction.<br>_:x owl:onProperties ( $R_1$ … $R_n$).<br>_:x owl:someValuesFrom $D^n$. |

## 2.2 属性

### 对象属性表达式

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 具名对象属性<br>(named object property ) | PN | PN |
| 通用对象属性<br>( universal object property) | owl:topObjectProperty | owl:topObjectProperty |
| 空对象属性<br>( empty object property) | owl:bottomObjectProperty | owl:bottomObjectProperty |
| 逆属性<br>( inverse property) | ObjectInverseOf(PN) | _:x owl:inverseOf PN |

### 数据属性表达式

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 具名数据属性<br>(named data property) | R | R |
| 通用数据属性<br>( universal data property) | owl:topDataProperty | owl:topDataProperty |
| 空数据属性<br>(empty data property) | owl:bottomDataProperty | owl:bottomDataProperty |

## 2.3 个体与文本

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 具名个体<br>(named individual) | aN | aN |
| 匿名个体<br>( anonymous individual) | _:a | _:a |
| 文本(literal) (数据类型值) | "abc"^^DN | "abc"^^DN |

## 2.4 数据值域

### 数据值域表达式

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 具名数据类型<br>(named datatype) | DN | DN |
| 数据值域补集<br>(data range complement) | DataComplementOf(D) | _:x rdf:type rdfs:Datatype.<br>_:x owl:datatypeComplementOf D. |
| 数据值域交集<br>(data range intersection) | DataIntersectionOf($D_1 \cdots D_n$) | _:x rdf:type rdfs:Datatype.<br>_:x owl:intersectionOf ($D_1 \cdots D_n$). |
| 数据值域并集<br>( data range union) | DataUnionOf($D_1 \cdots D_n$) | _:x rdf:type rdfs:Datatype.<br>_:x owl:unionOf ($D_1 \cdots D_n$). |
| 文本枚举<br>(literal enumeration) | DataOneOf($v_1$ … $v_n$) | _:x rdf:type rdfs:Datatype.<br>_:x owl:oneOf ( $v_1$ … $v_n$ ). |
| 数据类型限制<br>(datatype restriction) | DatatypeRestriction(DN $f_1$ $v_1$ … $f_n$ $v_n$) | _:x rdf:type rdfs:Datatype.<br>_:x owl:onDatatype DN.<br>_:x owl:withRestrictions (_:$x_1$ … _:$x_n$).<br>_:$x_j$ $f_j$ $v_j$. j=1…n |

## 2.5 公理

### 类表达式公理

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 子类<br>(subclass) | SubClassOf($C_1$ $C_2$) | $C_1$ rdfs:subClassOf $C_2$. |
| 等价类<br>( equivalent classes) | EquivalentClasses($C_1$ … $C_n$) | $C_j$ owl:equivalentClass $C_{j+1}$. j=1…n-1 |
| 不相交类<br>( disjoint classes) | DisjointClasses($C_1$ $C_2$) | $C_1$ owl:disjointWith $C_2$. |
| 两两不相交类<br>( pairwise disjoint classes) | DisjointClasses($C_1$ … $C_n$) | _:x rdf:type owl:AllDisjointClasses.<br>_:x owl:members ( $C_1$ … $C_n$ ). |
| 不相交并<br>( disjoint union) | DisjointUnionOf(CN $C_1$ … C | CN owl:disjointUnionOf ( $C_1$ … $C_n$ ). |

## 对象属性公理

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 子属性 (subproperty) | SubObjectPropertyOf($P_1$ $P_2$) | $P_1$ rdfs:subPropertyOf $P_2$. |
| 属性链包含 (property chain inclusion) | SubObjectPropertyOf(ObjectPropertyChain($P_1$ … $P_n$) P) | P owl:propertyChainAxiom ($P_1$ … $P_n$). |
| 属性定义域 (property domain) | ObjectPropertyDomain(P C) | P rdfs:domain C. |
| 属性值域 (property range) | ObjectPropertyRange(P C) | P rdfs:range C. |
| 等价属性 (equivalent properties ) | EquivalentObjectProperties($P_1$ … $P_n$) | $P_j$ owl:equivalentProperty $P_{j+1}$. j=1…n-1 |
| 不相交属性 (disjoint properties) | DisjointObjectProperties($P_1$ $P_2$) | $P_1$ owl:propertyDisjointWith $P_2$. |
| 两两不相交属性 (pairwise disjoint properties) | DisjointObjectProperties($P_1$ … $P_n$) | _:x rdf:type owl:AllDisjointProperties. _:x owl:members ( $P_1$ … $P_n$ ). |
| 逆属性 (inverse properties) | InverseObjectProperties($P_1$ $P_2$) | $P_1$ owl:inverseOf $P_2$. |
| 函数型属性 (functional property) | FunctionalObjectProperty(P) | P rdf:type owl:FunctionalProperty. |
| 反函数型属性 (inverse functional property) | InverseFunctionalObjectProperty(P) | P rdf:type owl:InverseFunctionalProperty. |
| 自反属性 (reflexive property) | ReflexiveObjectProperty(P) | P rdf:type owl:ReflexiveProperty. |
| 非自反属性 (irreflexive property) | IrreflexiveObjectProperty(P) | P rdf:type owl:IrreflexiveProperty. |

| | | |
|---|---|---|
| 对称属性 (symmetric property) | SymmetricObjectProperty(P) | P rdf:type owl:SymmetricProperty. |
| 非对称属性 (asymmetric property) | AsymmetricObjectProperty(P) | P rdf:type owl:AsymmetricProperty. |
| 传递属性 (transitive property) | TransitiveObjectProperty(P) | P rdf:type owl:TransitiveProperty. |

## 数据属性公理

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 子属性 ( subproperty) | SubDataPropertyOf($R_1$ $R_2$) | $R_1$ rdfs:subPropertyOf $R_2$. |
| 属性定义域 (property domain) | DataPropertyDomain(R C) | R rdfs:domain C. |
| 属性值域 (property range) | DataPropertyRange(R D) | R rdfs:range D. |
| 等价属性 (equivalent properties) | EquivalentDataProperties($R_1$ … $R_n$) | $R_j$ owl:equivalentProperty $R_{j+1}$. j=1…n-1 |
| 不相交属性 (disjoint properties) | DisjointDataProperties($R_1$ $R_2$) | $R_1$ owl:propertyDisjointWith $R_2$. |
| 两两不相交属性 (pairwise disjoint properties) | DisjointDataProperties($R_1$ … $R_n$) | _:x rdf:type owl:AllDisjointProperties. _:x owl:members ( $R_1$ … $R_n$ ). |
| 函数型属性 (functional property) | FunctionalDataProperty(R) | R rdf:type owl:FunctionalProperty. |

## 数据类型定义

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 数据类型定义 (datatype definition) | DatatypeDefinition(DN D) | DN owl:equivalentClass D. |

## 断言

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 个体等价 (individual equality) | SameIndividual($a_1$ … $a_n$) | $a_j$ owl:sameAs $a_{j+1}$. j=1…n-1 |
| 个体不等价 (individual inequality) | DifferentIndividuals($a_1$ $a_2$) | $a_1$ owl:differentFrom $a_2$. |
| 个体两两不等价 (pairwise | DifferentIndividuals($a_1$ … $a_n$) | _:x rdf:type owl:AllDifferent. _:x owl:members ($a_1$ … $a_n$). |

| | | |
|---|---|---|
| individual inequality)_ | | |
| 类断言 (class assertion) | ClassAssertion(C a) | a rdf:type C. |
| 肯定的对象属性断言 (positive object property assertion) | ObjectPropertyAssertion( PN $a_1$ $a_2$ ) | $a_1$ PN $a_2$. |
| 肯定的数据属性断言 (positive data property assertion) | DataPropertyAssertion( R a v ) | a R v. |
| 否定的对象属性断言 (negative object property assertion) | NegativeObjectPropertyAssertion(P $a_1$ $a_2$ ) | _:x rdf:type owl:NegativePropertyAssertion.<br>_:x owl:sourceIndividual $a_1$.<br>_:x owl:assertionProperty P.<br>_:x owl:targetIndividual $a_2$. |
| 否定的数据属性断言 (negative data property assertion ) | NegativeDataPropertyAssertion(R a v ) | _:x rdf:type owl:NegativePropertyAssertion.<br>_:x owl:sourceIndividual a.<br>_:x owl:assertionProperty R.<br>_:x owl:targetValue v. |

**键**

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 键(Key) | HasKey(C ($P_1$ … $P_m$) ($R_1$ … $R_n$) ) | C owl:hasKey ($P_1$ … $P_m$ $R_1$ … $R_n$). m+n>0 |

## 2.6 声明

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 类 (class) | Declaration( Class( CN ) ) | CN rdf:type owl:Class. |
| 数据类型 (datatype) | Declaration( Datatype( DN ) ) | DN rdf:type rdfs:Datatype. |
| 对象属性 (object property) | Declaration( ObjectProperty( PN ) ) | PN rdf:type owl:ObjectProperty. |
| 数据属性 (data property) | Declaration( DataProperty( R ) ) | R rdf:type owl:DatatypeProperty. |
| 注释属性 (annotation property) | Declaration( AnnotationProperty( A ) ) | A rdf:type owl:AnnotationProperty. |
| 具名个体 | Declaration( NamedIndividual( aN ) ) | aN rdf:type |

| | | |
|---|---|---|
| (named individual) | | owl:NamedIndividual. |

## 2.7 注释

### 注释

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 注释断言<br>( annotation assertion) | AnnotationAssertion(A s t) | s A t. |
| 公理注释<br>(annotation of an axiom)<br>其中，用 RDF 表示的公理是一个或者更多的形如 si U ti 这样的三元组，**即有相同的谓词 U**。 | AXIOM(Annotation(A t) ⋯) | _:$x_i$ A t.<br>$s_i$ U $t_i$.<br>⋯<br>_:$x_i$ rdf:type owl:Axiom.<br>_:$x_i$ owl:annotatedSource $s_i$.<br>_:$x_i$ owl:annotatedProperty U.<br>_:$x_i$ owl:annotatedTarget $t_i$. |
| 公理注释<br>(annotation of an axiom)<br>其中，用 RDF 表示的公理是 _:x U t1。 | AXIOM(Annotation(A t) ⋯ ) | _:x A t.<br>_:x U $t_1$.<br>⋯ |
| 其他注释的注释<br>(annotation of another annotation)<br>(用 RDF 表示的其他注释以 s1 开头) | Annotation(Annotation(A t) ⋯ $A_1$ $t_1$) | _:x A t.<br>$s_1$ $A_1$ $t_1$.<br>⋯<br>_:x rdf:type owl:Annotation.<br>_:x owl:annotatedSource $s_1$.<br>_:x owl:annotatedProperty $A_1$.<br>_:x owl:annotatedTarget $t_1$. |

### 注释属性

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 具名注释属性<br>( named annotation property) | A | A |
| 人类可读的名称<br>(human-readable name) | rdfs:label | rdfs:label |
| 人类可读的评注<br>(human-readable comment) | rdfs:comment | rdfs:comment |

| | | |
|---|---|---|
| 附加信息<br>(additional information) | rdfs:seeAlso | rdfs:seeAlso |
| 定义实体<br>(defining agent) | rdfs:isDefinedBy | rdfs:isDefinedBy |
| 版本信息<br>(version information) | owl:versionInfo | owl:versionInfo |
| 过时<br>(deprecation) | owl:deprecated | owl:deprecated |
| 向后兼容<br>(backwards compatibility) | owl:backwardCompatibleWith | owl:backwardCompatibleWith |
| 不兼容<br>(incompatibility) | owl:incompatibleWith | owl:incompatibleWith |
| 先前版本<br>(prior version) | owl:priorVersion | owl:priorVersion |

**注释公理**

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| 注释子属性<br>(annotation subproperties) | SubAnnotationPropertyOf($A_1$ $A_2$) | $A_1$ rdfs:subPropertyOf $A_2$. |
| 注释属性定义域<br>（annotation property domain） | AnnotationPropertyDomain(A U) | A rdfs:domain U. |
| 注释属性值域<br>（annotation property range） | AnnotationPropertyRange(A U) | A rdfs:range U. |

## 2.8 本体

**本体**

| 语言特性 | 函数式语法 | RDF 语法 |
|---|---|---|
| OWL 本体<br>(引入)[1][2] | Ontology([ON [U]]<br>Import($ON_1$)...<br>Annotation(A t)<br>...<br>) | ON rdf:type owl:Ontology.<br>[ON owl:versionIRI U.]<br>ON owl:imports $ON_1$. ...<br>ON A t.<br>... |
| 前辍声明<br>(prefix declaration)[3] | Prefix(p=U) | @prefix p U. |

1. [ ]表示可选结构
2.在 RDF 语法中，如果没有本体名，那么使用_:x 来代替 ON。
3.这里的 RDF 语法是 Turtle 表示法，其他的 RDF 序列化方法可能会有不同。

# 3 内置数据类型与分面（Facets）

## 3.1 内置数据类型

| | | |
|---|---|---|
| 通用数据类型 | rdfs:Literal | |
| Numbers | owl:rational | owl:real |

| | | | | |
|---|---|---|---|---|
| | xsd:double | xsd:float | xsd:decimal | xsd:integer |
| | xsd:long | xsd:int | xsd:short | xsd:byte |
| | xsd:nonNegativeInteger | | xsd:nonPositiveInteger | |
| | xsd:positiveInteger | | xsd:negativeInteger | |
| | xsd:unsignedLong | | xsd:unsignedInt | |
| | xsd:unsignedShort | | xsd:unsignedByte | |
| Strings | rdf:PlainLiteral (RDF 普通文本) | | | |
| | xsd:string | xsd:NCName | xsd:Name | xsd:NMTOKEN |
| | xsd:token | xsd:language | xsd:normalizedString | |
| Boolean Values | xsd:boolean (值空间：*true* 和 *false*) | | | |
| Binary Data | xsd:base64Binary | | xsd:hexBinary | |
| IRIs | xsd:anyURI | | | |
| Time Instants | xsd:dateTime (时区偏移为可选) | | | |
| | xsd:dateTimeStamp (时区偏移为必需) | | | |
| XML Literals | rdf:XMLLiteral | | | |

### 3.2 分面

| 分面 | 值 | 适用数据类型 | 说明 |
|---|---|---|---|
| xsd:minInclusive<br>xsd:maxInclusive<br>xsd:minExclusive<br>xsd:maxExclusive | 用相应数据类型表示的文本 | Numbers,<br>Time Instants | 限制值空间大于（等于）或者小于（等于）某个值 |
| xsd:minLength<br>xsd:maxLength<br>xsd:length | 非负整数 | Strings,<br>Binary Data,<br>IRIs | 根据文本长度，限制值空间 |
| xsd:pattern | 作为正则表达式的 xsd:string 文本 | Strings,<br>IRIs | 限制值空间为与正则表达式相匹配的文本 |
| rdf:langRange | 作为正则表达式的 xsd:string 文本 | rdf:PlainLiteral | 限制值空间为带有语言标签且与正则表达式相匹配的文本 |

# 4 附录

### 4.1 OWL2 中的新特性

| | |
|---|---|
| 类表达式 | • 本地自反（local reflexivity） (自我限制)<br>• 限定精确/最大/最小基数限制的对象和数据<br>• 在 n 元数据值域上的全称量化（universal）和存在量化（existential）限制 |
| 类公理<br>(Class Axioms) | • 两两不相交类<br>• 类不相交并 |
| 属性表达式<br>(Property | • 通用对象属性和空对象属性<br>• 通用数据属性和空数据属性 |

| | |
|---|---|
| Expressions) | • 逆对象属性表达式 |
| 属性公理<br>(Property Axioms ) | • 属性链包含<br>• 不相交对象属性<br>• 不相交数据属性<br>• 自反，非自反和非对称对象属性 |
| 数据值域<br>(Data Ranges) | • 数据类型定义<br>• 数据值域补集，交集和并集<br>• 数据类型限制和分面（facets）<br>• n 元数据类型的钩连（hook） |
| 断言<br>(Assertions) | • 否定的对象属性断言<br>• 否定的数据属性断言 |
| 注释<br>( Annotation) | • 注释断言<br>• 公理或注释的注释<br>• 注释子属性<br>• 注释属性定义域和值域<br>• owl:deprecated 注释属性 |
| 附加内置数据类型<br>(Extra Built-in Datatypes) | • owl:rational，owl:real，xsd:dateTimeStamp，rdf:PlainLiteral |
| 其他<br>(Others) | • 键<br>• 声明<br>• 元建模能力（metamodeling capabilities） (双关（Punning）)<br>• 匿名个体 |

## 4.2 OWL2 RDF 语法中的附加词汇

| 特性 | 词汇 | 注释 |
|---|---|---|
| 数据值域<br>(data range) | owl:DataRange | 在 OWL2 中已过时，由 rdfs:Datatype 代替 |
| 个体两两不同集合中的成员<br>(membership of a set of pairwise different individuals) | owl:distinctMembers | 也可选择使用 owl:members |
| 本体属性<br>(ontology property) | owl:OntologyProperty | |
| 过时<br>(deprecation) | owl:DeprecatedClass,<br>owl:DeprecatedProperty | 可选的 RDF 语法：<br>s rdf:type owl:DeprecatedClass.<br>或<br>s rdf:type owl:DeprecatedProperty.<br>**可由**<br>s owl:deprecated "true"^^xsd:boolean.<br>**代替** |

# 5 附录：变动日志 (资料性)

## 5.1 相对于建议推荐标准的变动

本节总结了该文档相对于 2009 年 9 月 22 日的建议推荐标准的变动。

- “（注释）Annotations”表有少量编辑性的改动。
- 对表头和其他内容的解释有少量编辑性的改动。
- 链接了本指南的 pdf 版本，即 OWL2 参考卡片。

## 5.2 相对于上一征求意见版本的变动

本节总结了该文档相对于 2009 年 6 月 11 日的候选推荐标准的变动。

- 关于 owl:rational 和 rdf:XMLLiteral 数据类型的“有风险的特性”（Features At Risk）标注已经删除；实现支持已经被充分证明，因此这些特性已不再被认为是有风险的（见 2009 年 8 月 5 日的 Resolution 5 和 Resolution 6）。
- 做过少量的编辑性改动。

# 6 致谢

OWL2 的开发始于 OWL1.1 成员提交（其本身是用户和开发者反馈的结果），尤其是在 OWL 体验与研究方向工作组(OWLED)系列中积累的信息。该工作组也考虑了来自于 WebOnt 工作组（WebOnt Working Group）的待解决问题（postponed issues）。

本文档由 OWL 工作小组（见下）执笔，它的内容反映了作为一个整体的该工作小组内部的广泛讨论。编者向 Bernardo Cuenca Grau（牛津大学），Christine Golbreich（Université de Versailles St-Quentin and LIRMM），Ivan Herman（W3C/ERCIM）和 Bijan Parsia (曼彻斯特大学) 的仔细审阅致以特别的谢意。

在本文档发布时，经常参加 OWL 工作组会议的与会者有：Jie Bao (RPI)、Diego Calvanese (Free University of Bozen-Bolzano)、Bernardo Cuenca Grau (牛津大学计算实验室)、Martin Dzbor (公开大学)、Achille Fokoue (IBM 公司)、Christine Golbreich (Université de Versailles St-Quentin and LIRMM)、Sandro Hawke (W3C/MIT)、Ivan Herman (W3C/ERCIM)、Rinke Hoekstra (阿姆斯特丹大学)、Ian Horrocks (牛津大学计算实验室)、Elisa Kendall (Sandpiper Software)、Markus Krötzsch (FZI)、Carsten Lutz (不来梅大学)、Deborah L. McGuinness (RPI)、Boris Motik (牛津大学计算实验室)、Jeff Pan (阿伯丁大学)、Bijan Parsia (曼彻斯特大学)、Peter F. Patel-Schneider (Bell Labs Research, Alcatel-Lucent)、Sebastian Rudolph (FZI)、Alan Ruttenberg (科学共享组织)、Uli Sattler (曼彻斯特大学)、Michael Schneider (FZI)、Mike Smith (Clark & Parsia)、Evan Wallace (NIST)、Zhe Wu (甲骨文公司)和 Antoine Zimmermann (DERI Galway)。我们还要感谢以前的工作组成员：Jeremy Carroll、Jim Hendler 和 Vipul Kashyap。

# 2.E　OWL2 Web 本体语言新特性与原理

本文档《OWL2 Web 本体语言新特性与原理》是 W3C 发布的 ***OWL 2 Web Ontology Language New Features and Rationale***（2009-10-27）的中文译本。文中若存在译法不当和错误之处，欢迎批评指正，请发邮件至：**zengxh@szu.edu.cn**，谢谢！

**翻译说明：**

- 本文档的英文版是唯一正式版本。此处的中文译本仅供学习与交流。
- 中文译本的内容是非正式的，仅代表译者的个人观点。
- 中文译本的内容会根据反馈意见随时进行修订。
- 中文译本同时通过 W3C Translations 网站发布。
- 转载本文，请注明译者和原链接。

**译者：**

曾新红（Xinhong Zeng），深圳大学图书馆 NKOS 研究室

蔡庆河（Qinghe Cai），深圳大学计算机与软件学院

**资助声明：**

本次翻译工作得到广东省哲学社会科学“十一五”规划项目（批准号：GD10CTS02）和国家社科基金项目“中文知识组织系统的形式化语义描述标准体系研究”（批准号：12BTQ045）的资助。

翻译时间：2011 年 9 月

发布时间：2012 年 10 月 24 日

---

# OWL2 Web 本体语言新特性与原理

# W3C 推荐标准 2009 年 10 月 27 日

当前版本：

http://www.w3.org/TR/2009/REC-owl2-new-features-20091027/

最新版本（OWL2 系列）：

http://www.w3.org/TR/owl2-new-features/

最新推荐标准：

http://www.w3.org/TR/owl-new-features

上一版本：

http://www.w3.org/TR/2009/PR-owl2-new-features-20090922/ (彩色标注不同之处版本)

编者：

Christine Golbreich，University of Versailles Saint-Quentin and LIRMM

Evan K. Wallace，美国国家标准与技术研究院 (NIST)

贡献者：

Peter F. Patel-Schneider，Bell Labs Research, Alcatel-Lucent

请参阅本文档的**勘误表**，那里可能会有一些规范的校正。

本文档也可以以如下的非规范格式查看： PDF 版本。

另见 译文。



---

# 摘要

OWL2 Web 本体语言，或者简略地记作 OWL2，是一个本体语言，用于带有形式化定义含义的语义网。OWL2 本体提供了类、属性、个体及数据值，以语义网文档的形式存储。OWL2 本体可以与以 RDF 记载的信息结合使用，并且

OWL2 本体本身也主要是以 RDF 文档的形式进行交换。OWL2 文档概述（Document Overview）阐述了 OWL2 的整体状态，因此，应当在阅读其他 OWL2 文档之前先阅读该文档。

本文档是对 OWL2 Web 本体语言新特性的简介，同时包含了对 OWL2 与 OWL 初始版本之间不同点的解释。它也介绍了激发设计主要新特性的需求，并从理论和实现的角度阐述了这些新特性的原理。

# 本文档的状态

## 可能已被替代

*本节描述的是本文档在发布时的状态。其他的文档可能替代了该文档。在* http://www.w3.org/TR/ *的 W3C technical reports index 中，可以找到当前的 W3C 出版物列表和该技术报告的最新版本。*

## XML Schema 数据类型依赖

根据定义，OWL2 使用的是 XML Schema Definition Language (XSD)中定义的数据类型。对本文档中的 OWL2 而言，XSD 的最新 W3C 推荐标准是版本 1.0，此时，版本 1.1 正在向推荐标准演进。OWL2 的设计已经利用了 XSD1.1 中的新数据类型和更清晰的注释，但是目前这些利用有一部分暂时搁置。特别地，OWL2 中基于 XSD1.1 的那些元素将被当作是*可选的*，直到 XSD1.1 成为 W3C 的推荐标准为止，详见 Conformance, section 2.3。等到 XSD1.1 发布为 W3C 的推荐标准时，这些元素才会终止可选状态而与其他指定元素一样进入必备状态。

我们提议，目前开发人员和用户遵循 XSD 1.1 Candidate Recommendation。根据 Schema 和 OWL 工作组之间的讨论，当 XSD1.1 演进成为推荐标准时，我们并不希望任何实现会有必须的改动。

## 变动总结

自上一版本以来，并没有什么实质性的改变。若需了解细微的变动细节，请参见变动日志和彩色标注不同之处版本。

## 请发表意见

请将意见发送至 public-owl-comments@w3.org（公开文档）。虽然由 OWL 工作组执笔的本文档已经完成，但是您的意见依旧可能在勘误表或者未来的修订版中得到解决。欢迎开发人员在 public-owl-dev@w3.org(公开文档)公开讨论。

## 由 W3C 批准

*本文档已经由 W3C 成员、软件开发人员以及其他 W3C 小组和兴趣组织审查，并由 W3C 主管（Director）批准成为 W3C 推荐标准。这是一个稳定的文档，可*

*以作为参考资料或被其他文档引用。W3C 在制作推荐标准过程中担任的角色，是要引起人们对该规范的注意并促进它的广泛应用。这提升了 Web 的功能性和交互性。*

### 专利

*本文档由遵循 5 February 2004 W3C Patent Policy 的小组完成。它只是一个提供信息的文档。W3C 维护着一个专利披露公开列表（public list of any patent disclosures），它与该组织的可交付成果一起制作；此页面也包含披露专利的说明。*

---

# 目录

# 1 引言

本文档概述了主要的 OWL2 新特性及其原理。这些特性基于实际应用、用户和工具开发者的经验而确定，有些特性在 OWLED 工作组系列中已有存档。提供给 W3C OWL 工作组的那些用例为这些特性的引入提供了支撑，一些用例在第 7 部分中列出。本文档也讲述和推介了一些其他的设计决策，这些决策是在 OWL2 的开发过程中作出的，或者是从 OWL Web 本体语言（OWL1）特意保留下来的，尤其是用于 OWL2 的不同具体语法以及 OWL2 与 RDF 的关系（第 4 部分）。OWL2 对 OWL1 进行了扩展，并继承了它的语言特性、设计决策和用于 OWL1 的用例。本文档也因此而形成了在 OWL1 下的用例及需求[*OWL Use Cases and Requirements*]的扩展版本。

OWL2 为 OWL1 新添了若干新特性，包括增强的对属性的表达能力、对数据类型的扩展支持、简单的元建模能力、扩展的注释能力以及键（第 2 部分）。OWL2 也定义了若干种配置语言——它们是能更好地满足特定的性能需求或者更易于实现的 OWL2 的语言子集（第 3 部分）。

# 2 特性&原理

OWL2 的新特性如以下类目所示：

1.使常用语句更易使用的语法糖（syntactic sugar），

2.增强了表达能力的新结构，

3.对数据类型的扩展支持，

4.简单的元建模能力，

5.扩展的注释能力，

6.其他创新及次要特性。

每种特性都按以下的共同模式描述：

- 一个解释添加该新特性理由的简短句，
- 特性描述，包含非形式化涵义、非形式化语法以及来自用例的一个简单示例，
- 新特性的理论与实现意义（implications），
- 与相关用例的链接。

读者可以通过切换下面的按钮，有选择地显示或隐藏示例以及示例中的函数式语法(FSS)或 RDF 语法。

**（译者注：此功能略）**

## 2.1 语法糖

OWL2 新添了语法糖以便更容易地书写一些常见模式（patterns）。由于所有的这些结构都只是简化的写法，所以它们并没有改变语言的表达力、语义和复杂性。但是，为了能够使处理更高效，实际实现可能更倾向于关注这些结构。

### 2.1.1 F1: 不相交并

虽然 OWL1 提供了定义一组子类为不相交的方法，并通过使用一些公理完成了对父类的覆盖（covering），但是，这并不能简洁地做到。

**DisjointUnion** 将一个类定义为其他类的并集，并且这些类是两两不相交的。它是若干单独的两两不相交类公理和并类创建的简化写法。 规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **DisjointUnion** ({ A } C $CE_1$ ... $CE_n$ ) ，其中 C 是一个类，$CE_i$, 1 ≤ i ≤ n 是类表达式，{ A }表示 0 或 0 个以上注释。 |

**示例：**

• HCLS

| | |
|---|---|
| DisjointUnion(*:BrainHemisphere :LeftHemisphere :RightHemisphere*) (*UC#2*) | 一个 *:BrainHemisphere* 只能是 *:LeftHemisphere* 或 *:RightHemisphere* 中的一个，而不能同时是两者。 |
| DisjointUnion(*:Lobe :FrontalLobe :ParietalLobe :TemporalLobe :OccipitalLobe :LimbicLobe*) (*UC#1*) | 一个 *:Lobe* 只能是 *:FrontalLobe*，*:ParietalLobe*，*:TemporalLobe*，*:OccipitalLobe* 或 *:LimbicLobe* 中的一个 ，而不能同时是它们中的两个或两个以上。 |

• 化学

| | |
|---|---|
| DisjointUnion(*:AmineGroup :PrimaryAmineGroup :SecondaryAmineGroup :TertiaryAmineGroup* ) (*UC#3*) | 一个 *:AmineGroup* 只能是 *:PrimaryAmineGroup*，*:SecondaryAmineGroup* 或 *:TertiaryAmineGroup* 中的一个，而不能同时是它们中的两者。 |

• 自动化

| | |
|---|---|
| DisjointUnion(*:CarDoor :FrontDoor :RearDoor :TrunkDoor*) (*UC#4*) | 一个 *:CarDoor* 只能是 *:FrontDoor*，*:RearDoor* 或 *:TrunkDoor* 中的一个，而不能是它们中的两个或两个以上。 |

用例#1 用例#2 用例#3 用例#4

### 2.1.2 F2: 不相交类

虽然 OWL1 为声明两个子类不相交提供了方法，但是不能简洁地声明多个子类不相交。

**DisjointClasses** 指出集合中的所有类都是两两不相交的。它是类之间的（多个）二元不相交公理的简易写法。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **DisjointClasses** ({ A } $CE_1$ ... $CE_n$ ) ，其中，$CE_i$, 1 ≤ i ≤ n 是类表达式，{ A } 表示 0 或 0 个以上注释。 |

**示例：**

• HCLS

| | |
|---|---|
| DisjointClasses( *:UpperLobeOfLung :MiddleLobeOfLung :LowerLobeOfLung* ) (*UC#2*) | *:UpperLobeOfLung*，*:MiddleLobeOfLung*，*:LowerLobeOfLung* 是两两不相交的。 |
| DisjointClasses( *:LeftLung :RightLung* ) (*UC#2*) | 既是 *:LeftLung* 又是 *:RightLung* 的事物是不存在的。 |

注意：FMA 使用了大量的不相交类 [*FMA* C]：3736 个形如 *Left X vs Right X*（例如 Left lung vs Right lung），13989 个形如 *X of left Y vs X of right Y*（例如 Skin of right breast vs Skin of left breast），75 个形如 X of *male Y vs X of female Y*（例如 Right side of male chest vs Right side of female chest）。

用例#1 用例#2

### 2.1.3 F3: 否定的对象属性断言和否定的数据属性断言

虽然 OWL1 为断言个体的属性值提供了方法，但是它并没有提供结构以直接断言个体所没有（否定事实）的值。

**NegativeObjectPropertyAssertion**（或：**NegativeDataPropertyAssertion**）指出给定的个体不拥有给定的属性（或：文本）。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **NegativeObjectPropertyAssertion**( { A } OPE $a_1$ $a_2$ )，其中 OPE 是一个对象属性表达式，$a_1$、$a_2$ 是个体，{ A }表示 0 或 0 个以上注释。 |
| **NegativeDataPropertyAssertion**( { A } DPE a lt )，其中 DPE 是数据属性表达式，a 是个体，lt 是文本，{ A }表示 0 或 0 个以上注释。 |

**示例：**

• HCLS

| | |
|---|---|
| NegativeObjectPropertyAssertion( *:livesIn :ThisPatient :IleDeFrance* ) (*UC#9*) | *:ThisPatient* 不在 *:IleDeFrance* 区域居住。 |
| NegativeDataPropertyAssertion( *:hasAge :ThisPatient* 5^^*xsd:integer* ) (*UC#9*) | *:ThisPatient* 不是 5 岁。 |

用例#9

## 2.2 属性的新结构

OWL1 主要关注表达类及个体信息的结构，对属性的表达则略显不足。OWL2 提供了新的结构，来表达对属性的附加限制、属性的新特征、属性的不兼容性、属性链和键。

### 2.2.1 F4: 自我限制

OWL1 不允许通过给定的属性来定义一类与自身相关联的对象，例如控制自身的进程类。这种“本地自反性（local reflexivity）”在许多应用中都很有用处，尤其是在全局自反性对某属性一般不成立，但本地自反性却对一些对象类成立时。OWL2 结构 **ObjectHasSelf** 允许在类描述中使用本地自反性。自我限制是 SROIQ[*SROIQ*]的一部分，而 SROIQ 是作为 OWL-DL 基础的描述逻辑 (SHOIN) 的扩展，用于提供用户要求的附加条件，同时不影响它的可判定性和实用性。SROIQ 得到了若干推理器的支持，包括 FaCT++、HermiT 和 Pellet [*TOOLS*]。

使用 ObjectHasSelf 限制定义的类表达式，代表了一类通过给定对象属性与自身相关联的所有对象。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| ObjectHasSelf (OPE) ，其中 OPE 是对象属性表达式。 |

**示例：**

• HCLS

| | |
|---|---|
| SubClassOf ( *:AutoRegulatingProcess* ObjectHasSelf ( *:regulate* ) ) | 自动控制进程（Auto-regulating processes）*控制（regulate）*其自身。 |
| SubClassOf ( *:Auto-Phosphorylating-Kinase* ObjectHasSelf ( *:phosphorylate* )) (*UC#20*) | 自动磷酸化激酶 (Auto-Phosphorylating-Kinases)磷酸化其自身。 |

用例#5 用例#3

### 2.2.2 F5: 属性限定基数限制

虽然 OWL1 允许对属性实例的数量进行限制，例如定义拥有至少 3 个孩子的人，但是它并没有为约束被计数的实例的*类*或者*数据值域* (限定基数限制)提

供方法，例如指定一类拥有至少 3 个孩子并且都是女孩的人。在 OWL2 中，限定的和非限定的基数限制都是可能的。限定的对象和数据基数限制在 SROIQ 中已有表示并且已经成功实现。它们已经得到了不同的工具和推理器的支持（例如 Protégé4，FACT++，HermiT, KAON2，PELLET 和 RACER）[*TOOLS*] [*OWL API*]。

**ObjectMinCardinality**，**ObjectMaxCardinality** 和 **ObjectExactCardinality**（或：**DataMinCardinality**，**DataMaxCardinality** 和 **DataExactCardinality**）允许为对象（或：数据）属性进行最小、最大或精确的限定基数限制断言。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

对象属性基数限制

| |
|---|
| **ObjectMinCardinality** ( n OPE [ CE ])，其中 n 是一个非负整数，OPE 是对象属性表达式，[ CE ]表示 0 或一个类表达式。 |

| |
|---|
| **ObjectMaxCardinality** ( n OPE [ CE ]) ，其中 n 是一个非负整数，OPE 是对象属性表达式，[ CE ]表示 0 或一个类表达式。 |

| |
|---|
| **ObjectExactCardinality** ( n OPE [ CE ]) ，其中 n 是一个非负整数，OPE 是对象属性表达式，[ CE ]表示 0 或一个类表达式。 |

**示例：**

- HCLS

| | |
|---|---|
| ObjectMinCardinality( 5 *:hasDirectPart owl:Thing* ) | 一类拥有至少 5 个 *direct part* 的对象。 |
| ObjectExactCardinality( 1 *:hasDirectPart :FrontalLobe* ) (*UC#1*) | 一类只拥有一个类型为 *frontal lobe* 的 direct part 的对象。 |

在 OWL1 中，可以表述一个大脑半球有至少 5 个 direct part，但是不可以表述像*用例#1* 中所需要的，只有一个特定类型（*frontal*、 *parietal*、 *temporal*、 *occipital* 或 *limbic lobe*）的 direct part。而在 OWL2 中，如上面的例子所示，这两种陈述都是可以的。

- 化学

| | |
|---|---|
| ObjectMaxCardinality ( 3 *:boundTo :Hydrogen*) (UC#3) | 一类至多结合三个不同*:Hydrogen* 的对象。 |

• 自动化

| | |
|---|---|
| ObjectMaxCardinality( 5 *:hasPart :Door* ) (*UC#4*) | 一类有至多 5 个*:Door* 的对象。 |
| ObjectExactCardinality( 2 *:hasPart :RearDoor* ) (*UC#4*) | 一类只能拥有两个*:RearDoor* 的对象。 |

数据属性基数限制

| |
|---|
| **DataMinCardinality** ( n DPE [ DR ] ) ，其中 n 是非负整数，DPE 是数据属性表达式，[ DR ]表示 0 或 1 个数据值域。 |

| |
|---|
| **DataMaxCardinality** ( n DPE [ DR ] ) ，其中 n 是非负整数，DPE 是数据属性表达式，[ DR ]表示 0 或 1 个数据值域。 |

| |
|---|
| **DataExactCardinality** ( n DPE [ DR ] ) ，其中 n 是非负整数，DPE 是数据属性表达式，[ DR ]表示 0 或 1 个数据值域。 |

**示例：**

• HCLS

| | |
|---|---|
| DataMaxCardinality ( 1 *:hasSSN* ) | 每个个体拥有至多一个社会保障号。 |

用例#1 用例#2 用例#3, 用例#4 用例#8

### 2.2.3 F6: 自反、非自反和非对称对象属性

虽然 OWL1 允许断言对象属性是对称或传递的，但是不可能断言属性是自反的，非自反的和非对称的。

OWL2 结构 **ReflexiveObjectProperty** 允许断言对象属性表达式是全局自反的，即所有的个体都满足该属性。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **ReflexiveObjectProperty** ( { A } OPE ) ，其中 OPE 是对象属性表达式，{ A }表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| ReflexiveObjectProperty( *:sameBloodGroup* ) (*UC#9*) | 任何事物都与它自身有相同的血型。 |
| ReflexiveObjectProperty( *:part_of* ) (*UC#2*) | 任何事物都是它自身的一部分。 |

注意：对部分－整体关系（mereological relations）存在不同的解释。例如，OBO (用例#5) 声明:*part_of* 是自反的，而在用例#1 中，解剖实体间的部分－整体关系 *anatomicalPartOf* 却被断言为是非自反的。

OWL2 结构 **IrreflexiveObjectProperty** 允许断言对象属性表达式是非自反的，即所有的个体都不满足该属性（自反）。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **IrreflexiveObjectProperty** ( { A } OPE )，其中 OPE 是对象属性表达式，{ A }表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| IrreflexiveObjectProperty( *:proper_part_of* ) (*UC#5*) | 任何事物不可能是它自身的一部分（proper part）。 |
| IrreflexiveObjectProperty( *:boundBy* ) (*UC#1*) | 任何事物都不可以结合自身。 |

- 地球与空间

| | |
|---|---|
| IrreflexiveObjectProperty( *:flowsInto* )(*UC#6*) | 任何事物都不可以流进自身。 |

**注意：给出的这些例子对应于给定用例（例如用例#1）中关于部分－整体属性（*anatomicalPartOf*）以及拓扑属性（*:boundBy*）的陈述。但是，其他的应用可能将这些术语用于具有不同特征的属性。**

OWL2 结构 **AsymmetricObjectProperty** 允许断言对象属性表达式是非对称的，即，如果属性表达式在个体 x 与 y 之间成立，那么它在 y 和 x 之间就不成立。注意“非对称”的语义比简单的“不对称（not symmetric）”更强。规范语法

（Normative Syntax）直接语义（Direct Semantics）基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **AsymmetricObjectProperty** ( { A } OPE ) ，其中 OPE 是对象属性表达式，{ A } 表示 0 或 0 个以上注释。 |

**示例：**

• HCLS

| | |
|---|---|
| AsymmetricObjectProperty( *:proper_part_of* )(*UC#8*) | 属性:*proper_part_of* 是非对称的。 |

这些结构是 SROIQ 的一部分，并且在 SROIQ 推理器（如 FaCT++、 HermiT 和 Pellet）中已经得到实现。

用例#5 用例#6 用例#8

注意：很多用例都举例说明了对自反、非自反、非对称和本地自反属性的诉求。医疗保健和生命科学兴趣小组在他们的上一征求意见版本中明确提到了这些特性的用途。语义网部署工作小组（SWD）也明确提到自反属性和非对称属性的潜在用途，例如，用于指定 SKOS 语义关系的特定应用的特定化 (见来自于 SWD 的意见)。例如，在部分－整体关系中，*partOf* 关系被定义为是传递的、自反的和反对称的。很多描述复杂结构的应用，例如在生命科学和系统工程中，需要以这种方式进行公理化，扩展使用部分－整体关系。在本体建模中遇到的其他关系也需要这样的公理化，不过可能具有不同的特征 (例如，[*OBO*] [*RO*])。例子包括：严格的部分关系（proper part of）与方位关系（典型的传递和非自反关系），因果关系（典型的传递和非自反关系）和成员关系（典型的非自反关系）。另外一个例子是 skos:broader 关系。SKOS 规范[*SKOS*]没有对 skos:broader 的自反和非自反进行声明，以允许两种解释：例如，对一个推理得到的 OWL 子类等级结构的直接转换（translation）而言，它应当被认为是自反的；但是对大多数的叙词表和分类法而言，它应当被认为是非自反的。OWL2 自反属性/非自反属性允许按需添加这两种特性中的一个。自我限制甚至是更细粒度的，对于一个给定的 skos:Concept，它允许 skos:broader 被处理为仅仅是本地自反或非自反（通过 SubClassOf 公理）。

### 2.2.4 F7: 不相交属性

虽然 OWL1 为声明类的不相交提供了方法，但不可能声明属性不相交。

OWL2 结构 **DisjointObjectProperties** 允许断言若干对象属性是两两不兼容的（排斥的）；也就是说，不可能通过该集合中的两个不同属性将两个个体连接起来。这个结构是 SROIQ 的一部分，并且在 SROIQ 推理器中已经得到实现。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **DisjointObjectProperties**( { A } $OPE_1$ ... $OPE_n$ ) ， 其中 $OPE_i$, $1 \leq i \leq n$ 是对象属性表达式，{ A }表示 0 或 0 个以上注释。 |

**示例：**

• HCLS

| | |
|---|---|
| DisjointObjectProperties( *:connectedTo :contiguousWith* ) (*UC#1*) | *:connectedTo* 与 *:contiguousWith* 是相互排斥的属性。 |

注意：用例#1 将通过第三方解剖实体关联起来的两个解剖实体定义为*有联系的*（*connected*），但是当它们相邻时，则被称为是*邻近的*（*contiguous*）。

**DisjointDataProperties** 允许断言若干数据属性是两两不兼容的（排斥的）。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **DisjointDataProperties** ( { A } $DPE_1$ ... $DPE_n$ )，其中 $DPE_i$, $1 \leq i \leq n$ 是数据属性表达式，{ A }表示 0 或 0 个以上注释。 |

**示例：**

| | |
|---|---|
| DisjointDataProperties( *:startTime :endTime* ) | 某事物（例如外科手术）的开始时间，必须不同于它的结束时间。 |

用例#1 用例#2 用例#3

### 2.2.5 F8: 属性链包含（Property Chain Inclusion）

OWL1 没有提供方法将属性定义为其他属性的组合（composition），就像定义“uncle”时需要做的（译者注：定义 uncle 需要用到两个或两个以上属性，例如 brother 和 father）；因此，不可能将一个属性（例如 *locatedIn*）与另一个属性（例如 *part of*）一起传递（propagate）。在一个 **SubObjectPropertyOf** 公理中的 OWL2 结构 **ObjectPropertyChain** 允许将属性定义为若干属性的组合。这种公理就是 SROIQ 中著名的*复杂角色包含（complex role inclusions）*（它也定义了对判定性而言所必需的正则条件），并在 SROIQ 推理器中已经得到实现。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

公理“SubObjectPropertyOf ( ObjectPropertyChain( $OPE_1$ ... $OPE_n$ ) OPE)”指出：通过对象属性表达式链 $OPE_1$, ..., $OPE_n$ 与另一个体 y 连接起来的个体 x，必然通过对象属性 OPE 与 y 相连。

| |
|---|
| SubObjectPropertyOf ( { A } **ObjectPropertyChain**( $OPE_1$ ... $OPE_n$ ) OPE )，其中 $OPE_i$, $1 \le i \le n$ 是对象属性，{ A }表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| SubPropertyOf (<br>ObjectPropertyChain(<br>*:locatedIn :partOf* )<br>*:locatedIn* )<br>(UC#7) | 如果 x 位于 y 内，并且 y 是 z 的一部分，那么 x 位于 z 内，例如，位于一个局部上的疾病也位于整体上。 |

用例#1 用例#5 用例#7 用例#8

### 2.2.6 F9:键

OWL1 没有为定义键提供方法。不过，键对于很多想通过（一组）键属性的值来唯一标识给定类的个体的应用而言，显然极为重要。OWL2 结构 **HasKey** 允许为给定的类定义键。虽然在 OWL2 中键属性不要求是函数型属性或全部（total）属性，但是如果需要，总是有可能单独地声明键属性是函数型属性。OWL2 中的键是 DL 安全规则[*DL-Safe*]的一种形式。它们在 HermiT、KAON2 和 Pelle 中已得到实现，也可以被加入到其他的推理器中。

**HasKey** 公理指出：类的每个具名实例都由一个（数据或对象）属性或者一组属性唯一标识——也就是说，如果类的两个具名实例彼此的某个键属性值完全相同，那么这两个个体就是一样的。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **HasKey** ( { A } CE ( $OPE_1$ ... $OPE_m$ ) ( $DPE_1$ ... $DPE_n$ ) ) ，其中 CE 是类表达式，$OPE_i$ , $1 \leq i \leq m$ 是对象属性表达式；$DPE_j$, $1 \leq j \leq n$ 是数据属性表达式，{ A }表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| HasKey( *:RegisteredPatient :hasWaitingListN* ) | 每个注册的患者——[在 ABM 国家器官轮候名单上] ——都通过他的轮候名单号唯一标识。（*UC#9*） |
| ClassAssertion( *:RegisteredPatient :ThisPatient* ) | *:ThisPatient* 是一个 *:RegisteredPatient*。 |
| DataPropertyAssertion( *:hasWaitingListN :ThisPatient* "123-45-6789" ) | *:ThisPatient* 拥有轮候名单号“123-45-6789”。 |

在这个例子中，由于 *:hasWaitingListN* 是类*:RegisteredPatient* 的键，所以号码“ 123-45-6789 ”就唯一地标识了 *:ThisPatient* 。公理 HasKey( *:RegisteredPatient :hasWaitingListN* ) 仅仅指出：被分配了号码的两个不同的患者，在轮候名单上不可能拥有相同的号码，如果类*:RegisteredPatient* 的两个具名实例的*:hasWaitingListN* 值一样，那么这两个个体将会是同一个个体。HasKey 公理与 InverseFunctionalProperty 公理相似，主要的不同在于它只适用于显式命名的个体。它并没有声明每个注册患者有至少或至多一个*:hasWaitingListN* 值。不能得出这样的推论：每个拥有一个*:hasWaitingListN* 的患者都属于*:RegisteredPatient* 类。

| | |
|---|---|
| HasKey( *:Transplantation* :donorId :recipientId :ofOrgan ) | 每项移植都由一个供体、一个受体和一个器官唯一标识。(*用例#9*) |

需要若干属性的集合来标识一项移植：实际上，一个供体可能为一个人捐赠多个器官，例如肾和肝脏，或者将同种器官捐赠给两个受体，例如将肾或其他不同的器官捐赠给不同的受体。

用例#2 用例#7 用例#9

## 2.3 扩展的数据类型能力

### 2.3.1 F10: 附加的数据类型和数据类型限制

OWL1 只支持整数和字符串数据类型，不支持这些数据类型的任何子集。例如，我们可以说每个人都有年龄，这是一个整数，但是不可以限制该数据类型的值域说成年人的年龄大于 18。OWL2 给数据类型提供了新的能力，如同 XML Schema 那样，通过分面（facets）支持更加丰富的数据类型集合和数据类型限制。

OWL2 数据类型包括：a) 不同种类的数字（numbers），增加了对更大范围的 XML Schema 数据类型 (double、float、decimal、positiveInteger 等)的支持，并提供了它自己的数据类型，例如 owl:real；b) 带有（或不带有）语言标签（使用 rdf:PlainLiteral 数据类型）的字符串； c) 布尔值、二进制数据、IRI、时刻（time instants）等。

**DatatypeRestriction** 也使通过限制*分面*来指定对数据类型的限制成为可能，分面限制给定数据类型所允许的范围，通过限制长度（针对字符串，例如 minLength、maxLength）和最小/最大值（例如 minInclusive、maxInclusive）来实现。扩展数据类型在很多的描述逻辑中都是允许的，并且得到了一些推理器的支持。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **DatatypeRestriction** ( DT $F_1$ $lt_1$ ... $F_n$ $lt_n$ )，其中 DT 是一元数据类型，⟨ $F_i$ $lt_i$ ⟩, $1 \leq i \leq n$ 是限制性分面和文本对（pairs）。 |

**示例：**

- HCLS

| | |
|---|---|
| DatatypeRestriction (xsd:integer minInclusive 18) (*用例#9*) | 在 XML Schema 数据类型 xsd:integer 之上建立的下界为 18 的新数据类型 |

例如，定义 18 岁以下（儿童）患者就需要这种数据类型，他们依靠医院的儿科服务，而那些 18 岁以上的（成人）患者则依靠成人服务。

用例#9 用例#11 用例#12 用例#18 用例#19

### 2.3.2 F11: n 元数据类型

在 OWL1 中，不可能表示一个对象的值之间的关系，例如无法表示正方形就是长等于宽的矩形。n 元数据类型支持并**没有**被添加到 OWL2 中，因为到底什么样的支持应该添加进来还存在争论。但是，OWL2 包含了 n 元数据类型所需的句法结构，为扩展提供了共同的基础。数据值域扩展：线性等价（W3C note）从有理系数线性（不）等价的角度，提出了对 OWL2 的扩展以定义数据值域。

- HCLS

| | |
|---|---|
| DataAllValuesFrom<br>( *:admissionTemperature :currentTemperature*<br>DataComparison(Arguments(x y) leq( x y ))))<br>(*UC#11*) | 其 *:admissionTemperature* 小于等于*:currentTemperature* 的个体 |

用例#10 用例#11

### 2.3.3 数据类型定义

OWL1 允许通过类描述来定义新类，但并没有为显式地定义新数据类型提供方法。为了易于书写、阅读和维护本体，OWL2 为定义数据类型提供了新的结构。如果相同的数据类型在一个本体中多次使用，这就非常有用。

**DatatypeDefinition** 允许显式地为新数据类型命名。规范语法（Normative Syntax） 直接语义（Direct Semantics） 基于 RDF 的语义（RDF-Based Semantics）

| |
|---|
| **DatatypeDefinition** ( { A } DT DR )，其中 DT 是数据类型，DR 是数据值域，{ A } 表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| DatatypeDefinition( *:adultAge*<br>DatatypeRestriction(xsd:integer<br>minInclusive 18) (*UC#9*) | 通过使用下界为 18 的 XML Schema 数据类型 xsd:integer，定义了成年人的年龄。 |

用例#9

### 2.3.4 数据值域组合

虽然 OWL1 允许通过组合类来构建新类，但是并没有提供通过组合其他数据类型来构建新数据类型的方法。而在 OWL2 中，就可以使用这种方式定义新的数据类型。

在 OWL2 中，可以使用数据值域的交运算（DataIntersectionOf）、并运算（ DataUnionOf）和补运算（DataComplementOf）来构建数据值域组合。

| |
|---|
| **DataIntersectionOf** ( { A } $DR_1$ ... $DR_n$ ) ，其中 $DR_i$, $1 \le i \le n$ 是数据值域，{ A } 表示 0 或 0 个以上注释。 |

| |
|---|
| **DataUnionOf** ( { A } $DR_1$ ... $DR_n$ ) ，其中 $DR_i$, $1 \le i \le n$ 是数据值域，{ A }表示 0 或 0 个以上注释。 |

| |
|---|
| **DataComplementOf** ( { A } DR) ， 其中 $DR_i$, $1 \le i \le n$ 是数据值域，{ A }表示 0 或 0 个以上注释。 |

**示例：**

| | |
|---|---|
| DataComplementOf( *:adultAge* ) | 该数据值域包含了所有不是大于等于 18 的正整数的文本。 |

用例#9

## 2.4 简单的元建模能力

### 2.4.1 F12: 双关 (Punning )

OWL1 DL 要求在名称之间有一个严格的区分，例如类名和个体名。OWL2 DL 放松了这种区分，在某种程度上允许同一术语有不同用法，例如 Eagle，可以用于类，指所有鹰类，也可以用于个体，表示属于所有植物和动物（元）类的鹰这种物种。但是，OWL2 DL 还是施加了某些限制：它要求同一名称不能既用于类又用于数据类型，一个名称只能用于一种属性。OWL2 直接语义对同一名称的不同用法完全区别对待，而这正是 DL 推理器所要求的。

**示例：**

- 电信

| | |
|---|---|
| Declaration( Class( *:Person* ) ) (*UC#13*) (1) | *:Person* 被声明为一个类。 |
| ClassAssertion( *:Service :s1* ) (2) | *:s1* 是 *:Service* 的一个个体。 |
| ObjectPropertyAssertion( *:hasInput :s1 :Person* ) (3) | 个体*:s1* 通过 *:hasInput* 与个体 *:Person* 连接。 |

同一术语“:Person”既在(1)中代表类又在(3)中代表个体。因为有了*双关* (Class ↔ Individual) ，这在 OWL2 中是可能的。

• 协作环境 (Wiki)

| | |
|---|---|
| Declaration( Class( *:Deprecated_Properties* ) ) (*UC#14*)(1) | *:Deprecated_Properties* 被声明为 *Class* |
| Declaration( ObjectProperty( *:is_located_in* ) ) (2) | *:is_located_in* 被声明为 *ObjectProperty* |
| ClassAssertion( *:Deprecated_Properties :is_located_in* ) (3) | *:is_located_in* 是*:Deprecated_Properties* 的一个个体。 |

同一术语“is_located_in”既在(2)中代表属性又在(3)中代表个体。因为有了 *双关* (Property ↔ Individual) ，这在 OWL2 中是可能的。

用例#14 也可以使用对属性*:is_located_in* 的注释 *deprecated property* 来表示，这种方法可能更直观或建模更佳。

• UML 设计

| | |
|---|---|
| Declaration( Class( *:Person* ) ) Declaration( Class( *:Company* ) ) (*UC#15*) (1) | *:Person* 与 *:Company* 被声明为类。 |
| SubClassOf ( *:PersonCompany :Association*) (2) | *:PersonCompany* 代表*:Association* 的一个子类，用于建模类*:Person* 与类 *Company* 之间的关联。 |
| ObjectPropertyDomain( *:PersonCompany :Person* ) (3) | 属性 *:PersonCompany* 的定义域是*:Person*。 |
| ObjectPropertyRange( *:PersonCompany :Company* ) (4) | 属性 *:PersonCompany* 的值域是 *:Company*。 |

同一术语“*:PersonCompany*” 既代表了类(2)也代表了对象属性(3, 4)。因为有了 *双关* (Class ↔ ObjectProperty)，这在 OWL2 中是可能的。

用例#12 用例#13 用例#14 用例#15

## 2.5 扩展的注释

OWL1 允许给每个本体实体提供超逻辑的（extralogical）注释，如标签（label）或注解（comment），但是不允许对公理进行注释，例如提供关于谁或什么时候断言了公理的信息。而 OWL2 允许注释本体、实体、匿名个体、公理和注释本身。

### 2.5.1 F13: 注释

对本体实体和匿名个体的注释 OWL2 提供了结构 **AnnotationAssertion** 用于本体实体（如类或属性）及匿名个体的注释。这些注释在 OWL2 直接语义中并不携带语义，允许 DL 推理器直接使用。

| |
|---|
| **AnnotationAssertion**( { A } AP s v )，其中 AP 是注释属性，s 是 IRI 或匿名个体，v 是文本、IRI 或匿名个体，{A} 表示 0 或 0 个以上（注释断言的）注释。 |

**示例：**

- HCLS

| | |
|---|---|
| AnnotationAssertion (*rdfs:label CARO:0000003 "anatomical structure"* ) (*UC#5*) | CARO 本体的 IRI *CARO:0000003* 由人类可读标签 *"anatomical structure"* 注释，该标签是 rdfs:label 注释属性的一个值。 |
| AnnotationAssertion (*FMA:UWDAID FMA:Heart* 7088 ) (*UC#2*) | FMA 的 IRI *FMA:Heart* 由整数 *7088*（其 FMA Id）注释，该整数是注释属性 *FMA:UWDAID* 的一个值。 |

对公理、注释、本体的注释 OWL2 提供了结构 Annotation 用于公理和本体的注释。它也可以用于注释本身的注释。这些注释在 OWL2 直接语义中并不携带语义，允许 DL 推理器直接使用。

| |
|---|
| **Annotation**( {A} AP v ) ，其中 AP 是注释属性，v 是文本、IRI 或匿名个体，{A} 表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| SubClassOf ( Annotation( *rdfs:comment* "Middle lobes of lungs are necessarily right lobes since left lungs do not have middle lobe.") *:MiddleLobe :RightLobe* ) (*UC#2*) | 注解"Middle lobes of lungs are necessarily right lobes…" 是子类公理的一个注释，解释了为什么*:MiddleLobe* 是*:RightLobe* 的子类。 |

用例#2 用例#5 用例#12 用例#19

### 2.5.2 关于注释属性的公理

注释属性可以被赋予定义域(**AnnotationPropertyDomain**)和值域(**AnnotationPropertyRange**)，并参与到一个注释属性等级结构（SubAnnotationPropertyOf）中。这些特殊的公理在 OWL2 直接语义中并没有语义含义，但是在基于 RDF 的语义中却带有标准的 RDF 语义（通过映射到 RDF 词汇表）。

注释属性的子属性

| |
|---|
| **SubAnnotationPropertyOf** ( { A } $AP_1$ $AP_2$ ) ，其中 $AP_1$ 和 $AP_2$ 是注释属性，{A}表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| SubAnnotationPropertyOf (*:narrow_synonym :synonym* ) (*UC#5*) | 属性 *:narrow_synonym* 是 *:synonym* 的子属性。<br><br>OBO 本体，尤其是 Gene Ontology（基因本体），区别不同种类的同义词：exact_synonym，narrow_synonym 和 broad_synonym。 |

注释属性的定义域

| |
|---|
| **AnnotationPropertyDomain** ( { A } AP U ) ，其中 AP 是注释属性，U 是 IRI，{A}表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| AnnotationPropertyDomain ( *FMA:UWDAID FMA:AnatomicalEntity* )(UC#2) | 只有 *FMA: AnatomicalEntity* 可以拥有一个 FMA:UWDAID (即：一个 FMA ID) 。 |

注释属性的值域

| |
|---|
| **AnnotationPropertyRange** ( { A } AP U ) ，其中 AP 是注释属性，U 是 IRI，{A}表示 0 或 0 个以上注释。 |

**示例：**

- HCLS

| | |
|---|---|
| AnnotationPropertyRange ( *FMA:UWDAID* xsd:positiveInteger ) (UC#2) | *FMA: AnatomicalEntity* 的 ID 是一个正整数。 |

用例#2 用例#5

## 2.6 其他的创新

### 2.6.1 F14: 声明

在 OWL1 中，实体（如类或对象属性）可以无需预先声明就在本体中使用，所以无法确保实体名在不同的公理中都能匹配。在实践中，如果实体名在公理中不匹配，就没有办法捕获错误。在 OWL2 中，一个声明意味着实体是本体词汇表的一部分。声明也使实体类别（类、数据类型、对象属性、数据属性、注释属性或个体）与被声明的实体间建立了关联。声明并不总是必需的（见语法）。声明不对 OWL2 本体的含义造成影响，因此也不影响推理。如果需要，实现可以选择检查每个名称是否已声明。

| |
|---|
| **Declaration**( A E ) ，其中 A 是注释，E 是实体。 |

**示例：**

- 工具

下面的声明指出 IRI *:Person* 用作一个类，而 IRI *:Peter* 是一个个体。

| | |
|---|---|
| Declaration( Class( *:Person* ) ) (*UC#17*) | *:Person* 被声明为一个类。 |
| Declaration( NamedIndividual( *:Peter* ) ) | *:Peter* 被声明为一个个体。 |

- HCLS

| | |
|---|---|
| Declaration( Class( *CARO:0000003* ) ) (*UC#5*) | *CARO:0000003* 被声明为一个类。 |

用例#17 用例#5

### 2.6.2 Top 和 Bottom 属性

OWL1 只有用于类的预定义顶层和底层实体，即 owl:Thing 和 owl:Nothing 这两个类，而 OWL2 又提供了顶层和底层对象属性和数据属性，即

**owl:topObjectProperty**、**owl:bottomObjectProperty**、**owl:topDataProperty** 和 **owl:bottomDataProperty**。

- 所有的个体对都由 owl:topObjectProperty 连接；
- owl:bottomObjectProperty 不连接个体；
- 所有可能的个体都经由 owl:topDataProperty 与所有的文本连接；
- 没有个体经由 owl:bottomDataProperty 连接到文本。

### 2.6.3 IRIs

统一资源定位符（URIs）在 OWL1 中用来标识类、本体和其他的本体元素。URI 是使用 ASCII 的一个子集形成的字符串。这是相当有局限性的，尤其是对于非英语名称，因为 ASCII 只包含了英语字母表中的字符。为了满足广泛的国际化需要，OWL2 使用了国际化资源标识符(IRI) [*RFC3987*] 来标识本体及其元素。

### 2.6.4 引入与版本

采用 OWL1，本体可以存储为语义网文档，且本体可以引入其他的本体。OWL2 使之更加明晰，通过本体文档的定位达到引入的目的。

OWL2 也澄清了本体名（IRI）与其位置之间的关系，为了响应某些请求，通过版本名（IRI）提供了简单的版本（控制）机制。每个 OWL2 本体都可以有一个本体 IRI，它用来标识该本体。OWL2 本体也可以有一个版本 IRI，用来标识该本体的特定版本。

OWL2 本体以其版本 IRI 存储，并且拥有本体 IRI 的某个本体也以该本体 IRI 存储。如果需要哪个版本并不重要，那么引入时可以使用本体 IRI，但是如果需要得到特定的版本，就要使用版本 IRI。

**Ontology** ( [O [ V ]] { Import ( O' ) } { A } { AX } )，其中[O]和[V]表示 0 或一个本体和版本 IRI，{Import(O')} 表示 0 或更多引入，O'是一个本体 IRI，{A}表示 0 或更多注释，{AX}表示 0 或更多公理。

该本体以其版本 IRI V 存储。使用了本体 IRI O 的某个版本也应该以 O 存储，这被认为是该本体的当前版本。

## 2.7 次要特性

在 OWL2 的语法中也引入了一些其他的改变，但是相对于 OWL1 而言，这些并不是表达力上的改变。

### 2.7.1 匿名个体

在 OWL1 中，匿名个体是作为无标识符的个体引入的。

**示例：**

| | |
|---|---|
| Individual(value( *:city :Paris* ) value( *:region :IleDeFrance* )) | 该公理并没有包含*:city* 和*:region* 三元组中主体的个体名，因此这个引入的个体是一个匿名个体。 |

相比之下，在 OWL2 中匿名个体是使用节点 ID 标识的。

**示例：**

| | |
|---|---|
| ObjectPropertyAssertion( *:city* _:a1 *:Paris* ) (*UC#9*) | 该公理为巴黎城中的这个未知地址引入了一个显式的匿名个体_:a1。 |
| ObjectPropertyAssertion( *:region* _:a1 *:IleDeFrance* ) | 该公理为法兰西岛地区中的未知地址引入了一个显式的匿名个体_:a1。 |

这个变动主要是由一个与新的函数语法相关的需求促成的。由于抽象语法结构的（嵌套）帧结构（frame structure），使用空节点的模式可以不使用节点 ID 指定，但是在函数语法中不可以这样做。没有表达力上的改变，在 RDF 方面也没有任何变动，并且 OWL2 中匿名个体的处理与 OWL1 完全向后兼容。在上面的例子中，“_:a1”只是表示 RDF 图中的一个空节点。

用例#9

### 2.7.2 逆属性

在 OWL1 中，所有的属性都是原子的，但是有可能断言某些对象属性是其他属性的逆属性。在 OWL2 中，属性表达式（如 ObjectInverseOf( P )）可以在类表达式中直接使用。它通过规避给逆属性命名的需要，使得书写本体更加容易。

逆对象属性表达式 ObjectInverseOf( P )连接个体 $a_1$ 和 $a_2$，当且仅当对象属性 P 连接 $a_2$ 和 $a_1$。

| |
|---|
| **ObjectInverseOf**( P )，其中 P 是一个对象属性。 |

**示例：**

| | |
|---|---|
| ObjectInverseOf( *:partOf* ) | 这个表达式表示了*:part of* 的逆属性。 |

逆对象属性公理 **InverseObjectProperties**( $OPE_1$ $OPE_2$ )指出两个属性是互逆的。

| |
|---|
| **InverseObjectProperties**( $OPE_1$ $OPE_2$ )，其中 $OPE_1$ 和 $OPE_2$ 是对象属性表达式。 |

**示例：**

下面是一个 OWL1 逆属性公理的例子。

| | |
|---|---|
| ObjectProperty( *:hasPart* inverse *:partOf* ) | *:hasPart* 有一个名为*:part of* 的逆属性。 |

它在 OWL2 中，可以用下面的公理表示 *:hasPart* 是 *:part of* 的逆属性。

| | |
|---|---|
| EquivalentProperties( *:hasPart* ObjectInverseOf( *:partOf* ) ) | *:partOf* 与 *:hasPart* 的逆属性等同。 |

由于这样的公理相当普遍，OWL2 也提供了下面的简写句法。

| | |
|---|---|
| InverseObjectProperties( *:hasPart :partOf* ) | *:hasPart* 和*:partOf* 是互逆属性。 |

# 3 配置语言

## 3.1 F15: OWL2 EL, OWL2 QL, OWL2 RL

OWL1 定义了两种主要的变体，OWL DL 和 OWL Full，以及一个句法子集(OWL Lite)。但是，事实表明，它对解决随后由 OWL 本体部署（deployment）所确定的需求是不够的。

- 许多应用，尤其是在生命科学领域中的应用，使用超大规模的本体，例如 FMA、NCI 叙词表、SNOMED CT、Gene Ontology 和一些 OBO 本体。这样的本体经常需要表示（相当）复杂的实体（例如由复杂方式连接起来的部件组成的解剖实体）或者允许属性延伸（例如疾病的位置从局部到整体）；它们也会有海量的类，以及由分类构成的重量级应用，以便于开发和维护。因此，这些应用主要与语言的扩展性和推理性能问题（见，例如围绕 FMA[*FMA*]的话题）相关，并且愿意牺牲一些表达力以换取计算性能保证，尤其是分类方面的。

- 涉及经典数据库的许多应用都关注 OWL 与数据库技术和工具之间的互操作性。虽然这些应用中所使用的本体一般是相对轻量级的，但是它们经

常被用来查询存储在标准关系数据库中的大量个体。因此，就产生了这样一个需求：经由关系查询（例如 SQL）直接访问这些数据。

•其他的应用关注本体语言与规则和现有规则引擎之间的互操作性。虽然这些应用中使用的本体也一般是相对轻量级的，但是它们可以被用来查询大型的数据集，并且对 RDF 三元组形式的数据进行直接操作可能是有用或有必要的。典型的案例既包括愿意牺牲语言的完整表达力以换取效率的 OWL 应用，也包括需要来自于 OWL2 的某些额外表达力的 RDF(S)应用。

为了满足以上需求，OWL2 定义了 3 种不同的配置语言：OWL2 EL、OWL2 QL 和 OWL2 RL——它们是具有有效计算性能（例如从 LOGSPACE 到 PTIME 范围内的推理复杂度）或者实现可能性（例如使用 RDBs 的片段可实现（fragments implementable））的 OWL2 的子语言（句法子集）。以下是简明描述，若需完整描述，见配置语言 [*OWL2 Profiles*]。

### 3.1.1 OWL2 EL

OWL2 EL 捕获了许多大规模本体（例如 SNOMED CT 和 NCI 叙词表）所用到的表达能力。

OWL2 EL 为该语言设置了若干句法上的限制：

- 对结构的限制： OWL2 EL 支持对类表达式或数据值域的存在量化，对个体（ObjectHasValue）或文本（DataHasValue）的存在量化，自我限制，涉及单个个体或单个文本的枚举，类和数据值域的交运算。摒弃的特性包括对类表达式或数据值域的全称量化，基数限制(min、max 和 exact)、析取 (ObjectUnionOf、DisjointUnion 和 DataUnionOf)，类的否定（class negation）以及许多其他的特性；“OWL2 配置语言”文档[*OWL2 Profiles*]中给出了 missing features 的完整列表。

- 对公理的限制：OWL2 EL 支持多数的公理，例如子类、等价类、类不相交、值域与定义域、对象属性包含(SubObjectPropertyOf)，可能涉及到属性链和数据属性包含 (SubDataPropertyOf)、传递属性、键 (HasKey)，…

- 应该注意到，除了句法限制外，OWL2 EL 在一个附加条件下扩展了对 OWL2 结构规范 [*OWL2 Specification*]中定义的公理的全局限制（见 OWL2 配置语言[*OWL2 Profiles*]的 2.2.6 Global Restrictions）。

由于这些限制，OWL2 EL 推理器 (例如 CEL [*CEL*]) 可以开发推理算法，包括查询应答算法，已知其最大时间复杂度是多项式（见 OWL2 配置语言[*OWL2 Profiles*]的 Computational Properties)。缩略词 EL 反映出该配置语言的基础是描述逻辑的 EL 家族[*EL++*] [*EL++ Update*]，即只提供存在量化的逻辑。

### 3.1.2 OWL2 QL

OWL2 QL 捕获了典型地用于简单本体（如叙词表）的表达能力，以及 ER/UML 模式所用到的（大部分）表达能力。

OWL2 QL 为该语言设置了若干句法上的限制：

- 对结构的限制：特性包括：存在限制的一个有限形式，子类，等价类，不相交，值域与定义域，对称属性等。摒弃的特性包括：对类表达式或数据值域的存在量化，自我限制，对个体或文本的存在量化，个体和文本枚举，对类表达式或数据值域的全称量化，基数限制 (min、max 和 exact)，析取 (ObjectUnionOf、DisjointUnion 和 DataUnionOf)，属性包含 (涉及属性链的 SubObjectPropertyOf)，函数型和反函数型属性，传递属性，自反属性，非自反属性，非对称属性，键；OWL2 配置语言[*OWL2 Profiles*]中给出了 missing features 的完整列表。

- 对公理的限制：除了不允许 DisjointUnion 外，OWL2 QL 与结构化规范[*OWL2 Specification*]支持相同的类公理。

这些限制使其能够与 RDBMS 紧密集成在一起，并且推理器可以在标准关系型数据库之上实现。因此，该配置语言尤其适合那些只需要相对轻量级本体又有大量个体的应用，以及通过关系查询（例如 SQL）直接访问数据有用或有必要的应用。推理，包括查询应答，可以使用查询重写技术高效地实现，已知其最大时间复杂度是 NLogSpace(见 OWL2 配置语言[*OWL2 Profiles*]的 Computational Properties)。缩略词 QL 反映了这样一个事实：查询应答可以通过将查询改写为标准的关系型查询语言来实现。

### 3.1.3 OWL2 RL

OWL2 RL 设计用于愿意牺牲语言的完整表达力以换取效率的 OWL 应用，以及需要来自于 OWL2 的某些额外表达力的 RDF(S)应用。这是通过定义一个 OWL2 句法子集来达到的，该子集适用于采用基于规则技术的实现。

OWL2 RL 为该语言设置了若干句法上的限制：

- 对结构的限制：支持大部分的 OWL2 类表达式结构，但是它们被限制在某些句法位置使用（见 OWL2 配置语言 [*OWL2 Profiles*]的 Table 2）。例如，对类的存在量化以及对类表达式(ObjectUnionOf)的并运算都不允许出现在公理的右边。

- 对公理的限制：除了类的不相交并、自反的对象属性公理和否定的对象及数据属性断言之外，OWL2 RL 支持 OWL2 的所有公理。

这些限制使 OWL2 RL 能够使用基于规则的技术来实现，例如扩展了规则的 DBMS，并且使推理（包括查询应答）的最大时间复杂度是多项式（见 OWL2 配置语言[*OWL2 Profiles*]的 Computational Properties)。基于规则的实现能够直接操作 RDF 三元组 (例如 Oracle OWL Prime [*OWL Prime*])，也因此可以被应用到任意的一个 RDF 图，即任意的 OWL2 本体上。在这种情况下，只会计算出针对查

询的正确答案（推理将会是*可靠的*），但是它并不能保证获取到所有正确的答案（它可能是不*完整的*）。该配置语言受到过 DLP[*DLP*]和 pD*[*pD**]的启发，缩略词 RL 反映了这样的一个事实：推理可以使用标准的规则语言来实现。

用例#2 用例#3 用例#4 用例#8 用例#16

### 3.2 选择哪种配置语言？

应用开发人员可能会问自己哪种配置语言才能最大限度地满足他们的需求。这些不同配置语言的选择主要取决于应用所需要的表达力、赋予类或数据的推理优先级、数据集的大小以及扩展性的重要程度等。以下建议可能会有用：

- 如果用户需要一种可扩展配置语言，用于大而（相当）简单的本体且本体（TBox/Schema）推理也需要好的时间性能，那么可能需要考虑 OWL2 EL。

- 如果用户需要一种容易与关系型数据库系统互操作的配置语言，而且对大型数据集的可扩展推理又是最重要的任务时，可能需要考虑 OWL2 QL。

- 如果用户需要一种容易与规则引擎和扩展了规则的 DBMS 互操作的配置语言，而且对大型数据集的可扩展推理又是最重要的任务时，可能需要考虑 OWL2 RL。

注意 OWL2 QL 和 OWL2 RL 都能很好地适用于使用了相对轻量级的本体以及非常大的数据集的应用。选择哪一种配置语言可能取决于要处理的数据类型：如果通过关系查询(例如 SQL)直接访问数据有用或有必要，那么 OWL2 QL 可能更好一些；如果直接操作 RDF 三元组形式的数据有用或有必要，那么 OWL2 RL 可能更好一些。

# 4 其他的设计选择与原理

虽然 OWL2 与 OWL1 是完全向后兼容的，但是它的概念性设计却有轻微的不同，尤其是关于 OWL2 语法（的设计）。

### 4.1 语法

有不同的可用方法来序列化和交换 OWL2 本体。OWL2 的主要交换语法是 RDF/XML 语法[*RDF/XML*]，它是实现**必须**支持的唯一语法。正如下面所解释的，函数式语法[*OWL2 Specification*]的主要目的是指定该语言的结构。OWL/XML [*OWL2 XML*]是一种 XML 序列化，它由能更好与基于 XML 的工具和语言互操作的愿望所激发。

#### 标准语法

正如在“一致性准则”文档[*OWL2 Conformance*]的 Section 2.1 中所明确指出的，OWL2 本体的唯一必备的交换语法是 RDF/XML：

“已经为 OWL2 本体文档定义了若干语法，它们中的一些或全部都可以被 OWL2 工具用来交换文档。但是，将本体文档作为输入的那些遵循 OWL2 的工具，必须接受使用 RDF/XML 序列化[*OWL2 RDF Mapping*]的本体文档，并且，发布本体文档的那些遵循OWL2的工具，如果可能的话，也必须能够以RDF/XML序列化的方式（例如通过 HTTP 内容协商）发布本体文档（如果被要求这么做）。”

**函数式语法**

OWL1 的语法是通过抽象语法（AS）定义的。函数式语法（FS）在 OWL2 中扮演相似的角色：它定义了该语言的语法（grammar）。但是 OWL2 既在语法方面又在结构方面进行了详细说明。实际上，除了函数式语法之外，OWL2 也引进了*结构化规范*来精确地指定 OWL2 本体的概念性结构。结构化规范是使用统一建模语言(UML)定义的。它使用了 UML 图的一种很简单的形式，希望那些熟悉面向对象系统的读者能够很容易理解。结构化规范为 OWL2 的所有语法（规范的和非规范的）提供了一个规范的抽象模型。它独立于 OWL2 本体的任何具体的交换语法。函数式语法严格遵循该结构化规范。语法的清晰与易读是函数式语法设计的重要因素。引进函数式语法是为了能够更容易书写 OWL2 的公理。OWL2 函数式语法的另一个好处在于它与一阶逻辑中使用的语法更接近，这也使得不同的规范问题以及将 OWL2 结构与一般的文献关联起来更容易。它是 OWL2 众多语法（例如，RDF/XML、曼彻斯特语法）中的一员。

OWL1 提供了类帧（frame-like）语法，它允许类、属性或个体的若干特性能够在单个公理中一次性定义。这可能在实践中引发问题。首先，它将给定实体的许多不同方面绑定在单个的公理中。虽然这在设计本体时可能是方便的，但是编程处理它们时却不方便。实际上，OWL1 的大部分实现都将这样的公理分成若干个“原子”公理，每个“原子”公理只处理该实体的单个特性。但是，它可能引发往返(round-tripping)问题，因为该本体的结构可能会在这个过程中被破坏。其次，这种类型的公理经常会被误解释为是给定实体的一个声明或者唯一“定义”。不过，在 OWL1 中，不作为任何此类公理的主体，实体也可以被使用，并且，可能有很多这样的公理与同一实体相关联。OWL2 已经通过若干途径解决了这些问题。首先，OWL2 已经摒弃了这种类帧记法（frame-like notation），代之以更细粒度的公理结构：每个公理都只描述给定实体的一个特性。其次， OWL2 提供了结构一致性概念（notion of structural consistency）的显式声明和一个显式定义。虽然 OWL2 显得更加冗长，但也是期望不产生大多数使用本体工程工具创建 OWL 本体时产生的问题。

**示例：**

下面是 OWL1 中类帧公理的示例。

| | |
|---|---|
| ObjectProperty( *:partOf* ObjectInverseOf( *:containedIn* ) inverseFunctional transitive<br><br>Annotation( *rdfs:comment* "an object is a part of another object.")) | 属性*:partOf* 有一个名为 *containedIn* 的逆属性，属性*:partOf* 是一个反函数型属性和传递属性，并有方便人类使用的注解“Specifies that an object is a part of another object。” |

**示例：**

上例在 OWL2 中可以使用以下公理来表示。

| | |
|---|---|
| Declaration( ObjectProperty( *:partOf* ) ) | 对象属性 *:partOf* 的声明 |
| AnnotationAssertion( *rdfs:comment :partOf* "*partOf* means that an object is a part of another object." ) | 该断言提供了一个对属性*:part of* 的注解，即“*partOf* means that an object is a part of another object.” |
| InverseObjectProperties( *:partOf :containedIn* ) | *:partOf* 与*:containedIn* 是逆属性。 |
| InverseFunctionalObjectProperty( *:partOf* ) | *:partOf* 是一个反函数型属性。 |
| TransitiveObjectProperty( *:partOf* ) | *:partOf* 是传递属性。 |

至于 OWL2 中的抽象语法(AS)，如果 AS 作为一种交换语法使用，那么用 AS 编写的 OWL1 本体就可以输入到 OWL2 工具并依旧是有效的本体。但是要强调的是，这是工具供应者的问题：OWL2 本体必备的唯一交换语法是 RDF/XML，是否要接受用 AS（或 FS，就此而言）序列化的本体取决于工具。

**OWL/XML 语法**

OWL 工作组已根据 XML Schema [*XML Schema*]，为 OWL2 定义了一个 XML 语法，称为 XML_Serialization 或 OWL/XML[*OWL2 XML*]。该语法同 OWL2 的结构化规范 [*OWL2 Specification*]相一致。XML 语法由以下的 OWL 用户需求所激发：这些用户希望与基于 XML 的工具和语言（例如 WSDL、XSLT/XQuery/XPath，或模式感知编辑器）有更好的互操作性。这是一个标准格式，OWL 工具供应商

可以选择性地支持它，以提供对 XML 模式可用的大量工具链的访问。从而，使用这些供应商工具的 OWL 工具开发人员和用户就可以书写 XPath、XSLT、XQuery 和 CSS，与 OWL 协同工作。RDF/XML 作为 OWL1 中唯一可用的 XML 格式，使用它却很难做到这一点。另外一个好处在于，使用 GRDDL 可以将 XML 数据提供给 RDF/OWL 应用。OWL/XML 的引入也为精通 XML 的人理解 OWL 提供了方便之道，也使得 OWL 对于那些在 XML 工具和培训上做了相当大投资的组织或个人变得更有吸引力。该格式与必备的交换格式 RDF/XML 之间的转换已有可用的开源工具包。因此，OWL/XML 与现有的 OWL1 工具和数据集成在一起而又没有破坏工具间的互操作性。

## 4.2 向后兼容

与 OWL1 相比，OWL2 的整体结构没有变化——尽管可能命名不同，但是几乎所有的 OWL2 构造模块在 OWL1 中都已存在。

- 在 OWL1 中，抽象语法（见 OWL1 语义[*OWL1 Semantics*]的 Section 2)扮演着 OWL2[*OWL2 Specification*]中结构语法与函数式语法的双重角色。虽然 OWL2 的函数式语法与 OWL1 的抽象语法在形式上不同，但是它们在 OWL 的整体结构中的角色是一致的：指定了该语言的结构。OWL2 函数式语法与 RDF 图表示更接近，它可以捕获更多的 RDF 图；它与 UML [*UML*]中的结构化规范也有直接的对应。

- 像 OWL1 一样，OWL2 指定了一个从本体结构（使用抽象/函数式语法表示）到 RDF 图的精确映射。不过，OWL2 也受益于一个显式指定的从 RDF 图到本体结构的反映射[*OWL2 RDF Mapping*]。

- OWL2 的两种语义（直接语义 [*OWL2 Direct Semantics*] 和基于 RDF 的语义[*OWL2 RDF-Based Semantics*]）在 OWL1 中有它们自己的直接对应，分别对应于 Direct Model-Theoretic Semantics 和 RDF-Compatible Model-Theoretic Semantics [*OWL1 Semantics*]。

- XML 表示语法（XML Presentation Syntax）对 OWL1 而言也是可用的 [*OWL1 XML Syntax*] (虽然并不是推荐标准)。另一方面，OWL1 中并不存在曼彻斯特语法[*OWL2 Manchester Syntax*]。

- OWL1 定义了一种子语言(OWL Lite)，而 OWL2 定义了三种 (EL、QL 和 RL) [*OWL2 Profiles*]。 OWL Lite 还没有为 OWL2 重新规范，但是由于向后兼容，OWL Lite 终究还是 OWL2 的子语言。

RDF/XML 作为 OWL2 工具唯一必备的交换语法的核心角色并没有改变，直接语义与基于 RDF 的语义之间的关系（即对应定理）也没有改变。更为重要的是，它与 OWL1 不管在语法上还是在语义上，都是完全向后兼容的。

• 正如在 OWL1 中那样，OWL2 可以处理所有的 RDF 图。OWL2 中被赋予特定涵义的词汇表包含了 OWL1 中的特定词汇表。但是，owl:DataRange 虽然仍旧可能使用，但已过时——应该使用 rdfs:Datatype 代替它。

• OWL2 [*OWL2 Direct Semantics*]使用的直接语义与 OWL1 [*OWL1 Semantics*]的几乎完全兼容。唯一的不同在于注释在 OWL2 的直接语义中是语义无关的。不过，用户不大可能注意到下面的这些不同：首先，OWL1 直接语义中赋予注释的语义是极弱的，不大可能产生任何有意义的蕴含（entailment）；其次，使用直接语义的 OWL1 工具处理注释时一般会认为（好像）它们是语义无关的。

• OWL2 的基于 RDF 的语义[*OWL2 RDF-Based Semantics*]与 OWL1 的基于 RDF 的语义[*OWL1 Semantics*]是完全兼容的。这些语义的某些细节发生了改变，不过，推论集仍然是一样的。

• 如果 RDF 要遵循 OWL2 DL 本体文档 [*OWL2 Conformance*]，RDF 文档中的引入处理就会发生轻微的改变。在 OWL1 中，先发生引入，因此整个合并的图被认为是一个单元 [*OWL1 Semantics*]。在 OWL2 中，多数情况下单个文档会被认为是独立的 [*OWL2 Specification*]。这意味着，为了成为合格的 OWL2 DL 本体文档，没有完好设置本体头部的 OWL1 DL RDF 文档需要作轻微的改动。

# 5 总结表

这张表总结了这些主要的新特性，其中每个特性都有一个示例。它总结了用例（第 1 列）、特性（第 2 列）和示例（第 3 列）之间的关系。对于每个用例，都选用一个用粗体字显示名称的特定特性。第 3 列给出了相应的示例，第 4 列则以粗体字给出了它的参考来源。该用例涉及到的其他特性则以从 F1 至 F15 的数字标注。 (示例的选取旨在为每种特性提供一个容易理解的例证，许多领域及真实示例来源于网上可获取的论文)。

| 用例 | 特性 | 示例 | 参考 |
|---|---|---|---|
| UC#1 | **DisjointUnion**<br>F2 F5 F7 F8 F11 | DisjointUnion(*:Lobe :FrontalLobe :ParietalLobe :TemporalLobe : OccipitalLobe :LimbicLobe*)<br><br>*:Lobe 是 :FrontalLobe :ParietalLob :TemporalLobe :OccipitalLobe :LimbicLobe* 的不相交并。 | **[*MEDICAL REQ*]**<br><br>[*Ontology with rules*] [*Brain Imaging* ] |
| UC#2 | **DisjointClasses**<br>F1 F2 F5 F7 F9 | DisjointClasses( *:LeftLung :RightLung* )<br><br>*:Lung 不能同时是:LeftLung 和 :RightLung。* | **[*FMA*]** |

| | | | |
|---|---|---|---|
| UC#20 | **Local reflexivity** | ObjectHasSelf( *:phosphorylates*)<br><br>*一个类，其所有个体都:phosphorylates（磷酸化）自身。* | **[*BIO*]** |
| UC#4 | **Qualified Cardinality**<br>F1 F15 | ExactCardinality( 2 *:hasPart :RearDoor* )<br><br>一个类，其对象都刚好有 2 个*:RearDoor*。 | **[*Auto*]** |
| UC#5 | **Asymmetric property**<br>F6 F8 F13 | AsymmetricProperty( *:proper_part_of*)<br><br>*如果 p 是 q 的一部分（proper part），那么 q 就不可能是 p 的一部分。* | **[*OBO*]**<br><br>[*RO*]<br>[*OBO2OWL*] |
| UC#6 | **Irreflexive property** | IrreflexiveProperty( *:flowsInto* )<br><br>任何事物都不可以 *:flowsInto* （流入）它自己。 | **[*Ordnance*]** |
| UC#7 | **Property chain**<br>F9 | SubPropertyOf( ObjectPropertyChain( *:locatedIn :partOf* ) *:locatedIn* )<br><br>*:locatedIn*（位于）局部的任何事物也*:locatedIn*（位于）整体，例如某种疾病。 | **[*SNOMED REQ*]** |
| UC#8 | **Reflexive property**<br>F5 F8 | ReflexiveProperty( *:partOf* )<br><br>[*Part Whole*] 认为 *partOf* 是自反属性，例如“汽车是汽车的一部分”。 | **[*Part Whole*]** |
| UC#9 | **Negative property**<br>F9 F10 | NegativePropertyAssertion( *:hasAge :ThisPatient* 5^^*xsd:integer* )<br><br>该*患者*的年龄不是 5 岁。 | **[*Transplant Ontology*]**<br><br>[*Agence Biomedecine*] |
| UC#10 | **N-ary** | AllValuesFrom( *:testDate :enrollmentDate* x > y + 30)<br><br>那些*:testDate* 大于*:enrollmentdate* + 30 的个体。 | **[*N-ary*]** |
| UC#11 | **N-ary**<br>F10 | AllValuesFrom( *:admissionTemperature :currentTemperature* x < y)<br><br>那些*:admissionTemperature* 小于*:currentTemperature* 的个体。 | **[*N-ary*]** |
| UC#12 | **Datatype restriction** | DatatypeRestriction(xsd:integer minInclusive 18) | **[*Protege*]** |

| | | | |
|---|---|---|---|
| | F5 F12 F13 | 在 XML Schema 数据类型 xsd:integer 基础上，下界为 18 的新数据类型，用来描述 *Adult* 类。 | |
| UC#13 | **metamodeling** | Declaration( Class( *:Person* ) )<br>*:Person* 被声明为一个类；<br>ClassAssertion( *:Service :s1* )<br>*:s1* 是*:Service* 的实例；<br>PropertyAssertion( *:hasInput :s1 :Person* )<br>*:s1* 有一个输入*:Person*<br><br>*这是 Class ↔ Individual* 双关的一个例子。 | **[*Web Service*]**<br><br>[*Punning*] |
| UC#14 | **metamodeling** | Declaration( ObjectProperty( *:is_located_in* ) )<br><br>*:is_located_in* 被声明为一个*对象属性*；<br>ClassAssertion( *:Deprecated_Properties :is_located_in* )<br>*:is_located_in* 是类*:Deprecated_Properties* 的一个个体；<br>*这是 Property ↔Individual* 双关的一个例子。 | **[*Wiki*]**<br><br>[*Punning*] |
| UC#15 | **metamodeling** | Declaration( Class( *:Person* ) ) Declaration( Class( *:Company* ) )<br><br>*:Person* 和 *:Company* 被声明为类；<br>SubClassOf ( *:PersonCompany :Association*) )<br>类*:Person* 和 *:Company* 之间的关联；<br>PropertyDomain( *:PersonCompany :Person* )<br>属性*:PersonCompany* 的定义域是*:Person*。<br>PropertyRange( *:PersonCompany :Company* )<br>属性*:PersonCompany* 的值域是*:Company*。<br>*这是 Class ↔ ObjectProperty* 双关的一个例子。 | **[*UML Association Class*]**<br><br>[*Punning*] |
| UC#16 | **Profiles** | *该用例促生了一种配置语言，例如 OWL QL，其合取查询应答使用传统的关系型数据库系统来实现。* | **[*Who reads?*]** |
| UC#17 | **Declaration** | Declaration( Class( *:Person* ) )<br><br>*:Person* 被声明为一个类。 | **[*Syntax Problem*]**<br><br>[*TOOLS*]<br>[*OBO2OWL*] |
| UC#18 | **Datatype**<br>F5 | DatatypeRestriction( xsd:integer minInclusive "18000"^^xsd:integer maxExclusive "19600"^^xsd:integer )<br><br>*大气层的数据值域介于 18000 [英尺] 与 19600 [英尺] 之间。* | **[*VSTO*]** |

| | | | |
|---|---|---|---|
| UC#19 | **Annotation**<br>F10 | SubClassOf( *rdfs:comment* ("data generated by the LogParser using the ObserverLog") :LogInformation :Information)<br><br>*这是一个公理注释的示例。* | **[*NCAR*]** |

*说明*:

| **F1** | **F2** | **F3** | **F4** | **F5** | **F6** | **F7** | **F8** |
|---|---|---|---|---|---|---|---|
| Disjoint Union | Disjoint Classes | Negative Property Assertion | Local reflexivity | Qualified Cardinality | Reflexive, Irreflexive, Asymmetric | Disjoint properties | Property chain inclusion |
| **F9** | **F10** | **F11** | **F12** | **F13** | **F14** | **F15** | |
| Keys | Datatype restriction | N-ary datatype | Simple metamodeling capabilities | Extended annotations | Declarations | Profiles | |

# 6 参考


[OWL2 Specification]

*OWL2 Web Ontology Language: Structural Specification and Functional-Style Syntax* Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-syntax/.

[OWL2 Direct Semantics]

*OWL2 Web Ontology Language: Direct Semantics* Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-direct-semantics/.

[OWL2 RDF-Based Semantics]

*OWL2 Web Ontology Language: RDF-Based Semantics* Michael Schneider, editor. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-rdf-based-semantics/.

[OWL2 RDF Mapping]

*OWL2 Web Ontology Language: Mapping to RDF Graphs* Peter F. Patel-Schneider, Boris Motik, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-mapping-to-rdf-20091027/. Latest version available at http://www.w3.org/TR/owl2-mapping-to-rdf/.

[OWL2 Profiles]

*OWL2 Web Ontology Language: Profiles* Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, Carsten Lutz, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/. Latest version available at http://www.w3.org/TR/owl2-profiles/.

[OWL2 Conformance]

*OWL2 Web Ontology Language: Conformance* Michael Smith, Ian Horrocks, Markus Krötzsch, Birte Glimm, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-conformance-20091027/. Latest version available at http://www.w3.org/TR/owl2-conformance/.

[OWL2 XML Serialization]

*OWL2 Web Ontology Language: XML Serialization* Boris Motik, Bijan Parsia, Peter F. Patel-Schneider, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/. Latest version available at http://www.w3.org/TR/owl2-xml-serialization/.

[OWL2 Manchester Syntax]

*OWL2 Web Ontology Language: Manchester Syntax* Matthew Horridge, Peter F. Patel-Schneider. W3C Working Group Note, 27 October 2009, http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-manchester-syntax/.

[OWL1 Semantics]

*OWL Web Ontology Language: Semantics and Abstract Syntax*. Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks, eds., W3C Recommendation, 10 February 2004.

[OWL1 XML Syntax]

*OWL Web Ontology Language: XML Presentation Syntax*. Masahiro Hori, Jérôme Euzenat and Peter F. Patel-Schneider, eds., W3C Note, 11 June 2003.

[RFC 3987]

*RFC 3987: Internationalized Resource Identifiers (IRIs)*. M. Duerst and M. Suignard. IETF, January 2005, http://www.ietf.org/rfc/rfc3987.txt

[RDF/XML]

*RDF/XML Syntax Specification (Revised)*. Dave Beckett and Brian McBride, eds., W3C Recommendation 10 February 2004.

[OWL Use Cases and Requirements]

*OWL Web Ontology Language: Use Cases and Requirements* Jeff Heflin, ed. W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-webont-req-20040210/. Latest version available at http://www.w3.org/TR/webont-req/.

[SROIQ]

*The Even More Irresistible SROIQ*. Ian Horrocks, Oliver Kutz, and Uli Sattler. In Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006). AAAI Press, 2006.

[SHOIQ]

*A Tableaux Decision Procedure for SHOIQ*. Horrocks, I., and Sattler, U. In Proc. of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005) (2005), Morgan Kaufmann, Los Altos.).

[Next Steps]

*Next Steps to OWL*. B. Cuenca Grau, I. Horrocks, B. Parsia, P. Patel-Schneider, and U. Sattler. In Proc. of OWL: Experiences and Directions, CEUR, 2006.

[Syntax Problem]

*Problem with OWL Syntax*. Boris Motik and I. Horrocks, OWLED 2006, 2006.

[CEL]

*CEL—A Polynomial-time Reasoner for Life Science Ontologies*. F. Baader, C. Lutz, and B. Suntisrivaraporn. In U. Furbach and N. Shankar, editors, Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06), volume 4130 of Lecture Notes in Artificial Intelligence, pages 287-291. Springer-Verlag, 2006.

[SNOMED EL+]

*Replacing SEP-Triplets in SNOMED CT using Tractable Description Logic Operators*. B. Suntisrivaraporn, F. Baader, S. Schulz, K. Spackman, AIME 2007

[EL++]

*Pushing the EL Envelope*. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the 19th Joint Int. Conf. on Artificial Intelligence (IJCAI 2005), 2005.

[EL++ Update]

*Pushing the EL Envelope Further*. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the Washington DC workshop on OWL: Experiences and Directions (OWLED08DC), 2008.

[DL-Lite]

*Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family*. Diego Calvanese, Giuseppe de Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati. J. of Automated Reasoning 39(3):385-429, 2007.

[DLP]

*Description Logic Programs: Combining Logic Programs with Description Logic*. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. in Proc. of the 12th Int. World Wide Web Conference (WWW 2003), Budapest, Hungary, 2003. pp.: 48-57

[pD*]

*Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*. Herman J. ter Horst. J. of Web Semantics 3(2-3):79-115, 2005.

[OWLPrime]

*Implementing an Inference Engine for RDFS/OWL Constructs and User-Defined Rules in Oracle*. Zhe Wu Eadon, G. Das, S. Chong, E.I. Kolovski, V. Annamalai, M. Srinivasan, J. Oracle, Nashua, NH; Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on, pages 1239-1248, Cancun, 2008.

[Metamodeling]

*On the Properties of Metamodeling in OWL*. Boris Motik. On the Properties of Metamodeling in OWL. Journal of Logic and Computation, 17(4):617-637, 2007.

[Datatype]

*OWL Datatypes: Design and Implementation*. Boris Motik, Ian Horrocks, ISWC 2008, Karlsruhe, Deutshland, 2008.

[XML Schema]

*W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures*. Shudi Gao, C. M. Sperberg-McQueen, and Henry S. Thompson, eds. W3C Candidate Recommendation, 30 April 2009, http://www.w3.org/TR/2009/CR-xmlschema11-1-20090430/. Latest version available as http://www.w3.org/TR/xmlschema11-1/.

[DL-Safe]

*Query Answering for OWL-DL with Rules*. Boris Motik, Ulrike Sattler and Rudi Studer. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 3(1):41-60, 2005.

[UML]

*OMG Unified Modeling Language (OMG UML), Infrastructure, V2.1.2*. Object Management Group, OMG Available Specification, November 2007, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF/.

# 7 附录：用例

## 7.1 用例↔特性

| 用例 | 不相交并 | 不相交类 | 否定的属性 | 本地自反 | 限定基数 | 自反非自反非对称 | 不相交属性 | 属性链 | 键 | 数据类型限制 | N 元数据类型 | 元建模 | 扩展注释 | 声明 | 配置文件 | 匿名个体 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UC#1 | * | * | - | - | * | - | * | * | - | - | - | - | - | - | - | - |
| UC#2 | * | * | - | - | * | - | * | - | * | - | - | - | - | - | - | - |
| UC#3 | * | * | - | - | * | - | - | - | - | - | - | - | - | - | * | - |
| UC#4 | * | - | - | - | * | - | - | - | - | - | - | - | - | - | * | - |
| UC#5 | - | - | - | * | - | * | - | * | - | - | - | - | * | * | - | - |
| UC#6 | - | - | - | - | - | * | - | - | - | - | - | - | - | - | - | - |
| UC#7 | - | - | - | - | - | - | - | * | * | - | - | - | - | - | - | - |
| UC#8 | - | - | - | - | * | * | - | * | - | - | - | - | - | - | - | - |
| UC#9 | - | - | * | - | - | - | - | - | * | * | - | - | - | - | - | - |
| UC#10 | - | - | - | - | - | - | - | - | - | - | * | - | - | - | - | - |
| UC#11 | - | - | - | - | - | - | - | - | - | * | * | - | - | - | - | - |
| UC#12 | - | - | - | - | * | - | - | - | - | * | - | * | * | - | - | - |
| UC#13 | - | - | - | - | - | - | - | - | - | - | - | * | - | - | - | - |
| UC#14 | - | - | - | - | - | - | - | - | - | - | - | * | - | - | - | - |
| UC#15 | - | - | - | - | - | - | - | - | - | - | - | * | - | - | - | - |
| UC#16 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | * | - |

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UC#17 | - | - | - | - | - | - | - | - | - | - | - | - | - | * | - | - |
| UC#18 | - | - | - | - | * | - | - | - | - | * | - | - | - | - | - | - |
| UC#19 | - | - | - | - | - | - | - | - | - | * | - | - | * | - | - | - |
| UC#20 | - | - | - | * | - | - | - | - | - | - | - | - | - | - | - | - |

译者注：最后一行由译者根据 7.21 节的内容补充。

下面的用例列表并不全面。列表中所包含的用例只是激发 OWL2 *新*特性的许多用例中的一部分——无论是用户、实现者还是理论原因——它们此时被 OWL2 工作小组收录。一些在论文中指出的、将来可能需要的其他扩展（诸如规则、默认等）则在括号中标明。

所有的用例都按以下模式描述：*概述*、*特性*、*示例*、*参考*。*概述*只对用例作概括性的描述。*特性*罗列了论文中用例所需要的若干特性。*示例*指向一个特性以及一个被选来例证 OWL2 的特定新特性的简短例子。同样的信息也可以在表 3.2（译者注：似乎应该是“5 总结表”）中看到（缩略形式）。为了便于访问，*参考*指向网上可获取的相关论文，它们的 URL 在附录的参考书目中提供。

## 7.2 用例#1 -用于神经外科的大脑图像注释 [HCLS]

*概述*：要开发的系统涉及到神经外科中外科手术的准备。具体而言，其目标是要协助用户在一个环绕损伤(其切除术是主要目的)的区域标出大脑皮层沟回。给定解剖标记，尤其是在脑胶质瘤上，对外科手术是极为重要的。大脑图像注释对临床案例的文档化也是有益的，它使人们能够检索类似的案例，为将来的手术提供决策支持。集成以解剖特征为索引的多个分布式图像源，也需要一个共享的大脑解剖本体。这对用于大脑神经病理学的大脑图像统计分析的大规模联合系统是有用的。

*特性*：**不相交并，不相交类，限定基数限制，不相交属性，属性链包含公理，[n-元]，[规则]**

*示例*：不相交并

- 例如：Lobe 是 :FrontalLobe :ParietalLob :TemporalLobe :OccipitalLobe 和 :LimbicLobe 的不相交并。

*参考* ：[*MEDICAL REQ*] [*Ontology with rules*] [*Brain Imaging* ]

## 7.3 用例#2 -解剖学基础模型[HCLS]

*概述：*解剖学基础模型 (FMA) 是人类“权威”解剖学的最全面的本体。解剖学在生物医学中扮演重要角色，许多生物医学本体和应用都会引用解剖实体。FMA 是生物信息学的一个极佳的资源，它为使用解剖知识的应用之间提供信息共享的便利。正如其作者所言，FMA“……是生物医学信息学领域中的一个参考本体，用于关联不同的解剖视角，对齐生物信息学中已有的和将要出现的本体……”。解剖学（Anatomy）本体，连同基因（Gene）及疾病（Disease）参考本体，一起构成了未来生命科学语义网的支柱。但是在表示某些属性是排他属性（例如 proper-part 与 boundBy）方面，FMA 将会从 OWL 的新特性中受益。由于许多的生物医学本体和应用通过交叉引用（cross-references）指向 FMA 解剖学实体，所以键也会是有用的。

*特性*：**不相交并，不相交类，限定基数限制，不相交属性，键，扩展注释，配置语言**

*示例*：不相交类

- 例如：既是*:LeftLung* 又是*:RightLung* 的事物是不存在的。

*参考*：[*FMA*]

## 7.4 用例#3 -化合物分类[HCLS]

*概述*：官能团从原子及其连接性（connectivity）的角度，描述了化学反应的语义。当官能团出现在化合物中时，会表现出特有的化学行为。在此用例中，作者们走出了第一步，为化合物的分类设计了一个官能团的 OWL-DL 本体，并从领域需求的角度强调了 OWL1 及建议的 OWL1.1 的能力与限制。他们也描述了表达性特性在基础关系本体设计中的应用，以及怎样才能将上层本体用于引导生命科学知识的的形式化（formulation）。他们还介绍了增强现有本体以方便知识表示和问题应答的经验。

“单环与多环结构是参与若干类型化学反应的分子的重要组成部分。”诸如限定基数限制这样的新 OWL 语言特性将会有助于描述官能团的数量和类型。

*特性*：**不相交并，不相交类，限定基数限制，配置语言**

*示例*：限定基数限制

- 例如：用于指定官能团的数量和类型 。

*参考*：[*Chemistry*]

## 7.5 用例#4 -某自动化公司的多源查询[自动化]

*概述*：大型公司经常使用不同的模型和格式将信息及知识存储在多个信息系统中。该用例的主要目的是从一个大型自动化公司的多个数据和知识源中检索相

关信息。对这个应用而言，一种便于对多个数据库进行查询且易于表示 PLIB（Parts Library ISO 13584 Standard）产品本体的配置语言，将会是大有裨益的。

*特性*：**不相交并，限定基数限制，配置语言(OWL2 QL)**

*示例*：限定基数限制

- 例如：有刚好两个后门的汽车的类。

*参考*：[*Auto*]

## 7.6 用例#5 -用于生物医学数据集成的 OBO 本体 [HCLS]

*概述*：开放生物医学本体（Open Biomedical Ontologies, OBO）联盟一直在追求一种策略，通过其采用了通用受控本体的注释，来为集成生物医学数据提供便利。包括 Gene 本体在内，现有的 OBO 本体正在经历综合配套改革，新本体将会在一个正在演进的控制本体开发的共享准则集的基础上创建。其结果就是一个扩张的 OBO 本体家族，被设计成可互操作，并结合生物学现实的准确表示。在这一工作中，将 OBO 关系本体设计成定义一组基础关系及其语义。OBO 使用传递、对称、自反、反对称(anti-symmetric)等特征来限定每个关系。更广泛地，OBO 格式提供了一些在 OBO 本体中用到的结构，如 is_reflexive、is_symmetric、is_cyclic、is_anti_symmetric 等等。转换 OBO 本体需要将新的 OWL2 属性公理自反、非自反、非对称(asymmetric)等映射成相应的 OBO 结构，否则它们将会转换成注释。

*特性*：**本地自反，自反，非自反，非对称，属性链包含公理，声明[反对称]**

*示例*： 非对称

- 例如：如果 p 是 q 的一部分（proper part），那么 q 就不能是 p 的一部分。

*参考*： [*OBO*] [*RO*] [*OBO2OWL*]

## 7.7 用例#6 -（英国）陆地测量部的空间与拓扑关系[地球与空间]

*概述*: 陆地测量部（Ordnance Survey）是英国的国家测绘局（National Mapping Agency）。目前它维护着一个持续更新的大不列颠地形数据库。该数据库包含大约 4.4 亿人造及自然景观特征。从森林、道路和河流，到个人住所、园地，甚至是邮筒，一切都包含在这些特征中。除了这个地形测绘，整个的新信息层也在逐步地添加到此数据库中，比如与地图精确匹配的航空摄影图像，提供所有属性地址的数据以及集成的交通信息。对拓扑和空间关系及许多其他方面而言，“为了捕获我们的公理的真正意图，我们需要能够说明一个属性是否是自反、非自反、非对称（asymmetric）或反对称的（antisymmetric）”。

*特性*：**自反，非自反，非对称，[反对称]**

*示例*：非自反

- 例如：没有事物流进它自己。

*参考*：[*Ordnance*]

## 7.8 用例#7 -医学系统命名法 [HCLS]

*概述*：医学系统命名法－临床术语(SNOMED CT) 是一项在医疗保健领域具有广泛覆盖面的临床术语学成果，并且已被包括美国、英国、加拿大、澳大利亚、丹麦等在内的许多国家，选为用于电子健康应用的国家标准。SNOMED 最初在 1976 年发布，随着 SNOMED RT 与英国临床术语（Clinical Terms）第 3 版的合并，SNOMED CT 作为主要的扩展在 2002 年启用。一个使之不同于先前版本的主要特色在于，它使用了描述逻辑（DL）来定义和组织编码及术语。SNOMED 的另外一个主要特色是它的规模和复杂性。SNOMED 拥有超过 350,000 个概念编码，每一个概念编码都表示一个不同的类，比我们已知的第二大的基于 DL 的本体的数量级还要大。

在没有属性链包含公理的情况下，SNOMED 社区采用 OWL 将会需要难以实施的解决方法(awkward workarounds)，其伴随而来的混乱和复杂性将会有效地扼杀其在该方向上的发展。针对这些问题，我们有了更清晰的途径，使用 OWL2 进行进一步的开发以及与其他生物医学本体进行集成。必备的属性链包含公理允许对另一属性（例如 *part-of*）的属性继承进行编码，这在解剖学中是极其重要的。例如，有了诸如 has-location。proper-part-of < has-location 这样的公理，就可以将对手指的伤害推理成对手的伤害。正如[*SNOMED EL+*]中所报道的，使用这种方法重整的 SNOMED-CT，解剖学类的数目从 54,380 降到了 18,125，而 CEL 推理器[*CEL*] (version 0.94) 花费的时间也从 900.15s 降至 18.99s。

与 FMA 一样，通过概念 ID 给定了 SNOMED 与其他生物医学本体之间交叉引用的共同用法，键也将是极为有用的。

*特性*：**属性链包含公理，键，配置语言(OWL2 EL)**

*示例*：属性链

- 例如：位于局部的任何事物也位于整体。

*参考*：[*SNOMED REQ*]

## 7.9 用例#8 -OWL 本体中简单的部分－整体关系[HCLS]

*概述*：对那些为语义网开发本体的人而言，表示部分－整体关系是一个非常常见的问题。OWL 没有为部分－整体关系提供任何的内置原语（primitive）（像处理子类关系那样），但是它拥有能够表述大多数（但不是全部）此类常见案例的足够表示能力。部分－整体关系的研究本身已经是一个完整领域—“mereology (分体论，或称总分学)”，该文件（note）只是为了处理一些简单易懂的案例，

它们定义涉及了部分－整体关系的类。在这项研究中讨论了部分－整体关系需要的一些扩展，即：限定基数限制、自反以及从部分到整体的延伸（propagation）的需求。

*特性*：**限定基数限制，自反，属性链包含**

*示例*：自反

- 例如：额叶是脑半球的一部分（*part of* 关系），或者汽车是汽车的一部分（*part of* 关系）。

注意：根据 OBO 中所给的定义，整体也被认为是部分[*Part Whole*]，但也有一些反对的观点，认为“part of”不是自反的。

*参考*：[*Part Whole*]

## 7.10 用例#9 -法国的肾脏分配政策[HCLS]

*概述*：在法国，肾脏分配是生物医学机构（Agence de la biomedicine）的责任。它包含了一般的规则，例如：供体-受体 ABO 血型标识，在国家器官轮候名单中的唯一注册（注册号是在轮候名单中注册时分配的，它在该轮候名单中唯一地标识了该患者），以及一些特定器官的在全国范围分配优先级的定义。对于每个肾脏受体而言，都可以指定最小的 HLA 匹配和禁止抗原。儿科受体的优先级高于儿科供体。肾脏的建议优先级依次为：(1) 急症患者，(2) 患者的群体反应性抗体水平= 80%（包含在一个特定可接受抗原协议中）或=1（HLA 与供体不匹配）， (3) 0 失配患者，(4) 具有较低移植易接近性的患者。也涉及到地理标准：（移植地图中的）每个区域（例如法兰西岛）都应该只负责生活在本区域内的患者。这种现实中的应用和分配系统表明，成人与儿童的区分如何在医疗保健领域有强烈的影响：在医院里，18 岁以下患者（儿童）依靠儿科服务，而 18 岁以上患者（成人）依靠成人服务；只有等候移植的 16 岁以下儿童才在轮候名单中有优先权。

*特性*：**否定的属性断言，数据类型限制，键**

*示例*：否定的属性断言

- 例如：这个患者不是 5 岁。

*参考*：[*Agence Biomedecine*] [*Transplant Ontology*]

## 7.11 用例#10 -患者招募的资格标准

*概述*：这个用例基于一项正在进行的致力于临床观察互操作性（Clinical Observations Interoperability）的 W3C 特别工作，其目标就是使在临床实验（Clinical Trials）背景下的医疗保健服务中所创建的医疗数据能够重用和共享。特别地，被选来证明这种互操作方法可行性的第一个应用，就是患者招募。在该

案例中，一个临床实验协议样本集可从 http://www.clinicaltrials.gov 获得，其中每一个都包含一个候选资格列表（接纳与排除标准）。这些资格标准用于识别合格的患者，并潜在地形成 SPARQL 查询条件或者被表示为 OWL 类。正如在上面的用例中所讨论的，也需要将它们进行映射。一个基于这些临床实验协议分析的列表可以从 http://esw.w3.org/topic/HCLS/ClinicalObservationsInteroperability?action=AttachFile&do=get&target=FunctionalRequirements_v1.xls 获得。

特别地，这些临床实验的一个要求是临床实验参与者的登记日期，应当在他开始接受某项特定治疗之后的 30 天之内。这激发了对带有非等价表达式的 n 元数据类型的需求。

*特性*：**[n 元]**

*示例*：**n 元数据类型**

- 例如：临床实验参与者的登记日期，应当在他开始接受某项特定治疗之后的 30 天之内。

## 7.12 用例#11 -关于数据类型的多个用例 [HCLS]

*概述*：[*n 元*] 提供了很多用例，这些用例将受益于不同的数据类型扩展。

*特性*：**数据类型限制，[n 元]**

*示例*：**n 元数据类型**

- 例如：像时间间隔（interval）这样的数据类型限制，或者像用例#10 所需的带有不等价表达式的 n 元数据类型。

*参考*：[*N-ary*]

## 7.13 用例#12 - Protégé对 OWL 用户体验的报告 [工具]

*概述*：[*Protege*]在 2005 年报道了随着 OWL 支持的发展，Protégé的体验以及那时的 OWL 用户社区的体验。虽然源于这些社区的反馈整体上是积极的，但是他们的经验表明，在用户需求、OWL 表达力和用户对 OWL 的理解之间仍有相当大的鸿沟。Protégé开发人员根据其经验进行了总结，对 OWL 的未来版本提出了很多的扩展，即：对用户定义的数据类型的集成（尤其是数值值域（numeric ranges））、限定基数限制、不相交的管理(owl:AllDisjoint)、更为灵活的注释属性(至少最实用)。该报告强调，用户经常抱怨的是 OWL 对数值表达式的弱表示，这是 OWL 语言中的一个毗漏。除了那些开发传统医学术语的小组，几乎所有的小组都强烈地需要能够表示量化信息。典型的例子包括，介于 1mm 和 2mm 之间的长度，大于 18 岁的年龄，在 1030mb 与 1035mb 之间的压力。人们需要这样的值域声明来对个体进行分类以及创建诸如 *Adult* 这样的类定义，因此，也应该得到推理器的支持。用户群指出，当前的 OWL 数据类型形式体系太弱而不能为

现实世界中的大部分应用提供支持，许多潜在的用户也因此而不能采用 OWL。*“用户社区迫切等待 OWL 规范的一个扩展，来表示具有 XML Schema 分面（如 xsd:minInclusive）的用户自定义数据类型。”*它也从实现者的角度，指出了一些与注释或元建模相关的局限：*“在 OWL DL 中尽管有注释属性的值，被声明为注释属性的属性目前仍极为受限，它们既不可以拥有值域和定义域约束，也不能出现在子属性等级结构中。这类关于属性的信息使工具能够控制注释属性可以获得的值。由于没有值域约束，很难为用户提供合适的输入小部件（widget）。在类似的意义上，也有助于声明元类(meta-class)，从而可以将类分成不同的类型，并给每种类型提供不同的接口。当前，使用这些特性意味着将本体强行转为 OWL Full。”*

*特性*：**限定基数限制，数据类型限制，注释，元建模**

*示例*：附加数据类型

- 例如：成年人是年龄大于 18 岁的个体。

*参考*：[*Protege*]

## 7.14 用例#13 - Web 服务建模 [电信]

*概述*：人们经常想用一个类来指定某个属性的值。一个来自于卡尔斯鲁厄大学的例子[*Web Service*] 是服务建模。服务(Service)被建模为:Service 类的实例。对于每个具体的服务 (即每个:Service 的实例)，用户都需要声明服务的输入是什么。 这里有一个服务描述的示例：

(1) :Service rdf:type owl:Class
(2) :Person rdf:type owl:Class
(3) s1 rdf:type :Service
(4) s1 :input :Person

根据(1)和(3)，可得 s1 是类 :Service 的个体，而根据(2)，可得 :Person 是一个类；因此，在(4)中我们有一个在个体与类之间的关系 :input。因此，你需要某种元建模来解决这种问题。一种可能的方法是，名称“Person”既可以指一个 Person 类又可以指一个 Person 个体，表示作为一个整体的 Person (Class ↔ Individual)。

*特性*：**元建模**

*示例*：简单元建模

- 例如：类和个体 : *Person* 可以既用作类又用作个体。

*参考*：[*Web Service*] [*Punning*]

## 7.15 用例#14 -协作环境下的管理词表[维基]

*概述*：为了捕获模式元素(class/property)之间的实际关系，将模式元素关联起来会很有用。在 Semantic MediaWiki（一个简单但使用广泛的、带有轻量级表达力的基于 OWL 的语义内容管理系统）[*OWL1.1 Wiki*] 里观察到的一个例子是，用户希望将模式元素关联起来指明特定域的关系，一般是组织本体词表。例如这样的陈述：

- “属性 *is_located_in* 在类 *Deprecated_Properties* 中，并被属性 *has_location* 替代。”

- “类 *City* 的对象应该有一个属性 *population* 的值。” (通过将类与属性关联起来表示)

这些仅仅是实用的描述，没有模式级（schema-level）的逻辑关系。但是，在协作词表的创建过程中却是相关的，用户可以表达这种他们需要的关系。Semantic MediaWiki 的一个重要的方面是用户也可以查询语义信息，而其已通过双关实现。通过使用现成的 OWL 推理器，已经将 Semantic MediaWiki 进行了扩展，如果这样的系统能够在这种简单用例中解决双关(Class/Property ↔ Individual)的使用问题，那么将很受欢迎。

*特性*：**元建模**

*示例*：简单元建模

- 例如：一个属性和一个个体：作了一个陈述，断言一个属性是类 Deprecated_properties 的个体。

*参考*：[*Wiki*] [*Punning*]

## 7.16 用例#15 - UML 关联类[设计人员]

*概述*：统一建模语言(UML)包含了一个被称为 Association Class（关联类）的建模元素，它联合了 UML Class 和 UML Association（UML 中用于定义类类关系 Association 的结构）的特性。Association Class，例如 Person 类与 Company 类的关联，允许建模人员将关系定义为一个关联（association），并同时将其具体化。当建模人员想要建模关系本身的属性时，这就很方便。类和对象属性双关（Class ↔ ObjectProperty）可能是支持这种案例的一种方式。

*特性*：**元建模**

*示例*：简单元建模

- 例如：对象属性和类: *PersonCompany* 既可以用作对象属性又可以用作类。

*参考*：[*UML Association Class*] [*Punning*]

## 7.17 用例#16 -数据库联邦[设计人员]

*概述*：某位生命科学应用的设计者一直都在创建一种数据库联邦模式。此模式涉及到设计一种描述不同数据库的域和值的 XML 模式，以及关联查询工具，可以从一种查询接口（使用若干 SQL 变体）将查询写入具备相关信息的数据库中。那些结果都呈现在一个单一的集成视图中。他听说 OWL 和语义网技术可能是实现相同功能的合适技术并可以利用 Web 标准使之可用，但是他不知道要从哪里入手。本应用阐明了相当多用户社区的共同需求，这些用户想使用他们的数据库并可以很容易地以一种友好的方式进行查询。这就促生了一种配置语言，在该语言中，合取查询应答使用传统的关系型数据库系统实现。

*特性*：**配置语言(OWL2 QL)**

*示例*：配置语言

- 例如：OWL2 QL 配置语言易于以一种友好的方式对数据库联邦进行查询。

*参考*：[*Who reads?*]

## 7.18 用例#17 -工具开发人员[工具]

*概述*：一个用户给本体添加了一个断言，但是他偶然错误录入了个体的 IRI。通过比较公理中个体的 IRI 和显式声明为本体的一部分的 IRI，这种错误应该是可能检测到的。如果没有将个体 IRI 显式地引入本体中，也应该给用户更正这个错误的机会。由于缺乏声明，诸如涉及 Protégé-OWL 工具集架构[*TOOLS*]的工具开发人员经常会谈及由此引发的问题，例如针对 API [*OWL API*]。“首要问题是 OWL 不允许显式声明，即断言某个类、属性或个体在某个本体中存在。OWL 标准的这一方面经常被误解释，也由此而引发 OWL API 的设计错误”[*Syntax Problem*]。

*特性*：**声明**

*示例*：声明

- 例如：将人声明为本体的一个类。

*参考*：[*Syntax Problem*]

## 7.19 用例#18 - 虚拟日地天文台[地球与空间]

*概述*：大量的单学科和多学科虚拟天文台（例如 http://vsto.org，http://vmo.nasa.gov/ ）正在开始使用语义技术来提供数据访问与集成。虚拟天文台是一套放在一组计算机上的软件应用，它允许用户从一群分布式产品库及服务供应商那里统一地寻找、访问和使用资源（数据、软件、文档和图像产品及使用这些资源的服务）。VO 是一种结合服务和/或多个库的服务（见

http://lwsde.gsfc.nasa.gov/VO_Framework_7_Jan_05.doc）。一些虚拟天文台非常关注数据摄取时的起源编码（例如 http://spcdis.hao.ucar.edu/ ）。虚拟日地天文台(VSTO)是国家科学基金会（National Science Foundation）和国家大气研究中心（National Center for Atmospheric Research）支持的一项成果，它允许研究者查找太阳和日地数据。它为语义增强的 Web 报务提供了一个本体增强接口，而语义增强的 Web 服务有助于访问大量的在线科学数据库。后台的 OWL 本体包含了对包括仪器、天文台、参数等在内的科学术语的术语描述。用户必须为他们想要检索的数据指定一个数据描述，包含一个特定的仪器类(或者该仪器类的描述)，所需数据的日期范围，以及参数。为了在相关的科学术语中指明这一点，科学家需要能够表示数值值域及数值比较，这超出了 OWL1 的数值支持。该应用也需要扩展以包含空间描述。它会使用（如果）提供给空间/地理包含（containment）的表达力。

*特性*：**限定基数，数据类型限制，[默认]**

*示例*：数据类型限制

- 例如：大气(层)的值域介于 18000 与 19600[英尺]之间 。

*参考*：[*VSTO*]

## 7.20 用例#19 -语义源捕获[地球与空间]

*概述*：在一项为科学数据所附带的元（数据）信息提供更好搜索能力的工作中，SPCDIS 项目提供一个基础框架，捕获在数据摄取时期对有关科学源信息的声明性描述。该项目的初始领域是日冕物理学。这项工作（除了其他事项以外）需要扩展的注释及数据类型限制。

*特性*：**数据类型限制，扩展的注释**

*示例*：对附属注释的扩展注释

- 例如：对公理（如 SubClass 公理）的注解，以表达（比如）子类的元素是由一个日志解析器生成的数据。

*参考*：[*NCAR*]

## 7.21 用例#20 -生物化学自相互作用[化学领域]

*概述*：在生物化学中，一些生物分子会使其自身发生化学改变，这样一种方式在生物学上有重要意义。i) 蛋白激酶是能够为某些在靶蛋白中发现的氨基酸添加磷酸基的酶。一些被称为自动磷酸化激酶的激酶，会给某些靶氨基酸（其自身的一部分）添加磷酸基。ii) 核酶是有催化活性的 RNA 分子，它们中已知有 7 种自然类型会切割它们自己的 RNA 序列。这种切割可能会引起重要的变动，这些变动可能发生在病毒复制、基因表达上，也可能发生在不同蛋白质转录的生成上。这些有催化活性的自切割 RNA 组成了一个核酶的子类，称为自切割核酶

(Self-Cleaving Ribozymes)。这些生物医学上的自相互作用可以通过断言属性的本地自反来捕获。

*特性*：**本地自反**

*示例*：本地自反

- 例如：自动磷酸化激酶(是一种激酶) :phosphorylates（磷酸化）其自身。

*参考*：[*BIO*]

## 7.22 用例参考书目


[Medical Req]

*Web ontology language requirements w.r.t expressiveness of taxonomy and axioms in medicine In Proc. of ISWC 2003*

[Micro Theory]

*Creation and Usage of a "Micro Theory" for Long Bone Fractures: An Experience ReportHoward Goldberg, Vipul Kashyap and Kent Spackman, In Proc. of KR-MED 2008..*

[Ontology with Rules]

*Ontology enriched by rules for identifying brain anatomical structures In RIF 2004, Washington, 2004. and Annex.*

[Brain Imaging]

*Towards an Hybrid System Using an Ontology Enriched by Rules for the Semantic Annotation of Brain MRI Images In Proc. of RR 2007*

*The Brain Anatomy Case Study In Proc. of Protege 2005.*

[FMA]

*The Foundational Model of Anatomy A*

*The Foundational Model of Anatomy B*

*The Foundational Model of Anatomy C.*

[Chemistry]

*Describing chemical functional groups in OWL-DL for the classification of chemical compounds Natalia Villanueva-Rosales and Michel Dumontier. In OWL: Experiences and Directions (OWLED 07), Innsbruck, Austria.*

*Modelling Life Sciences knowledge with OWL1.1 (OWLED 08 DC)*

[Auto]

*An exploratory study in an automotive company*.

[OBO]

*The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Barry Smith et al.* .

[RO]

*Relations in Biomedical Ontologies.* .

[OBO2OWL]

*OBO to OWL: Go to OWL1.1!* *(OWLED 07)*

*OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences* *In Proc. of ISWC 2007.*

[Ordnance]

*Experiences of using OWL at the Ordnance Survey*.

[SNOMED REQ]

*An examination of OWL and the requirements of a large health care terminology*.

[Agence Biomedecine]

*Changing Kidney Allocation Policy in France: the Value of Simulation*.

[Transplant Ontology]

*Construction of the dialysis and transplantation ontology*.

[Little Web]

*A little semantic web goes a long way in biology* Wolstencroft, K., Brass, A., Horrocks, I., Lord, P., Sattler, U., Stevens, R., Turi, D. In Proceedings of the 2005 International Semantic Web Conference (ISWC 2005), pp. 786-800. Springer, Berlin Heidelberg New York (2005).

[Part Whole]

*Simple part-whole relations in OWL Ontologies* *Alan Rector, Chris Welty. W3C Editor's Draft 11 Aug 2005* .

[TOOLS]

*Supporting Early Adoption of OWL1.1 with Protege-OWL and FaCT++.* *Matthew Horridge and Dmitry Tsarkov and Timothy Redmond. In OWL: Experiences and Directions (OWLED 06), Athens, Georgia*.

[OWL API]

*Igniting the OWL1.1 Touch Paper: The OWL API* Matthew Horridge and Sean Bechhofer and Olaf Noppens (2007). In OWL: Experiences and Directions (OWLED 07), Innsbruck, Austria.

[Protege OWL]

*The Protégé OWL Experience* Holger Knublauch, Matthew Horridge, Mark Musen, Alan Rector, Robert Stevens, Nick Drummond, Phil Lord, Natalya F. Noy2, Julian Seidenberg, Hai Wang. In OWL: Experiences and Directions (OWLED 05), Galway, Ireland, 2005.

[N-ary]

*N-ary Data predicate use case*.

[Web Service]

*Preference-based Selection of Highly Configurable Web Services* Steffen Lamparter, Anupriya Ankolekar, Stephan Grimm, Rudi Studer: WWW-07, Banff, Canada, 2007.

[Wiki]

*Reusing Ontological Background Knowledge in Semantic Wikis* Denny Vrandecic, Markus Krötzsch, Proceedings 1st Workshop on Semantic Wikis. Budva, Montenegro, June 2006 .

[UML Association Class]

*Association*.

[Punning]

*Punning Use Cases*.

[Who reads?]

*Who reads our documents?*

*NIF*

*NIF Data-Integration slides*

[VSTO]

*The Virtual Solar-Terrestrial Observatory: A Deployed Semantic Web Application Case Study for Scientific Research* McGuinness, D.L., Fox, P., Cinquini, L., West, P., Garcia, J., Benedict, J.L., Middleton, D..

*VSTO2*.

*VMO*.

[NCAR]

*Semantic Provenance Capture in Data Ingest Systems* .

[BIO]

*Springer*.

*pnas*.

[SKOS]

*W3C Working Draft 29 August 2008* .

# 8 附录：变动日志（资料性）

## 8.1 相对于建议推荐标准的变动

本节总结了该文档相对于 2009 年 9 月 22 日的建议推荐标准的变动。

- 做了少量的编辑性改动。

## 8.2 相对于上一征求意见版本的变动

本节总结了该文档相对于 2009 年 6 月 11 日候选推荐标准的变动。

- 新添加了一条注释，该注释指出属性的非对称（asymmetric）是一个比不对称（non-symmetric）更强的概念。
- 新添加了关于配置语言名（profile names）起源的注释。
- 做了一些小的编辑性变动。

# 9 致谢

OWL2 的开发始于 OWL1.1 成员提交（其本身是用户和开发者反馈的结果），尤其是在 OWL 体验与研究方向工作组(OWLED)系列中积累的信息。该工作组也考虑了来自于 WebOnt 工作组（WebOnt Working Group）的待解决问题（postponed issues）。

本文档由 OWL 工作小组（见下）执笔，它的内容反映了作为一个整体的该工作小组内部的广泛讨论。编者向 Elisa Kendall（Sandpiper 软件），Peter F. Patel-Schneider（Bell Labs Research, Alcatel-Lucent））和 Alan Ruttenberg（科学共享组织）的仔细审阅致以特别的谢意。

在本文档发布时，经常参加 OWL 工作组会议的与会者有：Jie Bao (RPI)、Diego Calvanese (Free University of Bozen-Bolzano)、Bernardo Cuenca Grau (牛津大学计算实验室)、Martin Dzbor (公开大学)、Achille Fokoue (IBM 公司)、Christine Golbreich (Université de Versailles St-Quentin and LIRMM)、Sandro Hawke (W3C/MIT)、Ivan Herman (W3C/ERCIM)、Rinke Hoekstra (阿姆斯特丹大学)、Ian

Horrocks (牛津大学计算实验室)、Elisa Kendall (Sandpiper Software)、Markus Krötzsch (FZI)、Carsten Lutz (不来梅大学)、Deborah L. McGuinness (RPI)、Boris Motik (牛津大学计算实验室)、Jeff Pan (阿伯丁大学)、Bijan Parsia (曼彻斯特大学)、Peter F. Patel-Schneider (Bell Labs Research, Alcatel-Lucent)、Sebastian Rudolph (FZI)、Alan Ruttenberg (科学共享组织)、Uli Sattler (曼彻斯特大学)、Michael Schneider (FZI)、Mike Smith (Clark & Parsia)、Evan Wallace (NIST)、Zhe Wu (甲骨文公司)和 Antoine Zimmermann (DERI Galway)。我们还要感谢以前的工作组成员：Jeremy Carroll、Jim Hendler 和 Vipul Kashyap。