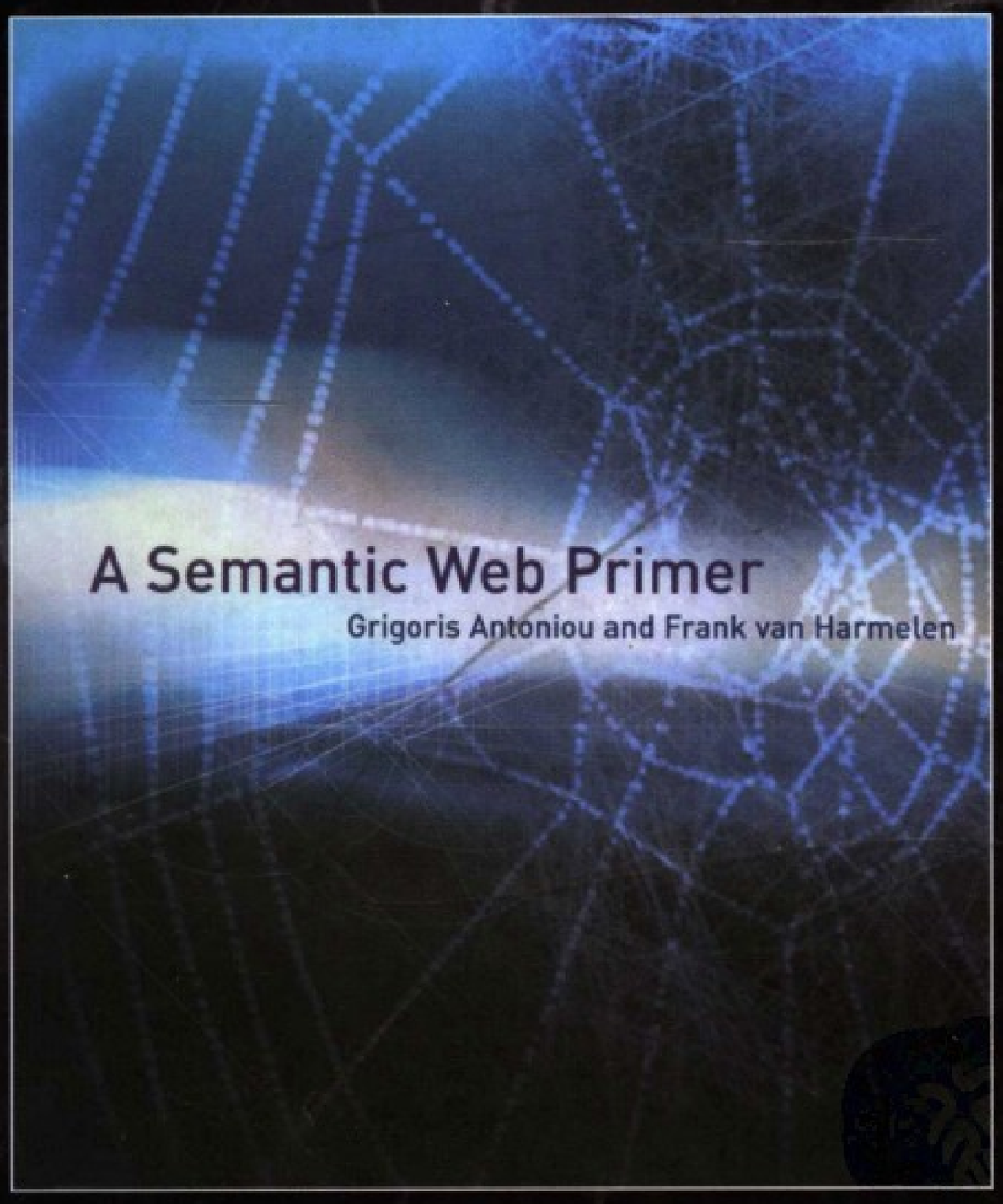
HZ BOOKS
华章教育

计 算 机 科 学 丛 书

语义网基础教程

(希) Grigoris Antoniou 赫拉斯研究与技术基金会
(荷) Frank van Harmelen Vrije大学 著
陈小平 中国科学技术大学 等译



A Semantic Web Primer

机 械 工 业 出 版 社
China Machine Press

语义网基础教程

“这是一本我们一直期待的书，一部语义网基本概念和方法的简明导论。”

—— *Rudi Studer*，德国卡尔斯鲁厄大学AIFB学院

“这是一本关于语义网特别是最近成为标准的新语言RDF Schema和OWL的极为出色的导论。我正在使用这本书作为本科生语义网课程的教科书，学生们认为它写得很好、清晰明了。对于那些希望着手学习这门新兴技术的人来说，本书将成为一件强有力的工具。”

—— *James Hendler*，马里兰大学计算机科学系

“这是一本杰出且非常必需的优秀书籍。它向读者全面介绍了语义网的动机、应用和支撑技术。”

—— *Ian Horrocks*，英国曼切斯特大学计算机科学系

随着网络内容的机器解读，语义网的发展孕育着万维网及其应用的一场革命。在本书中，两位作者为这个新兴领域提供了一个导引，描述了它的核心思想、语言和技术。本书可用作教材或专业人员的自学读本，主要介绍适合本科程度的基础概念和技术，并提供了习题、项目说明、参考文献及其注解，旨在帮助读者通过学习得以自行开发有关的应用。

本书是现今唯一一本系统讲述作为语义网核心内容的语言（XML、RDF、OWL和规则）与技术（显式元数据、本体、逻辑和推理）的书籍。书中还涉及本体工程及应用场景等重要内容。各章内容包括概览，XML以及支持语义可共用性的相关技术，作为机器可处理语义的标准数据模型的RDF和RDF Schema，作为RDF Schema的扩展和Web本体语言的W3C推荐标准的OWL，语义网框架中的单调和非单调规则，若干精选的应用领域以及语义网技术在其中的作用的介绍，基于本体的系统开发，当前热点的讨论和未来展望。

作者简介

Grigoris Antoniou 希腊赫拉斯研究与技术基金会计算机科学研究所教授。

Frank van Harmelen 荷兰Vrije大学人工智能系教授。

投稿热线：(010) 88379604
购书热线：(010) 68995259, 68995264
读者信箱：hzjsj@hzbook.com

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

封面设计：锡 杉

上架指导：计算机/网络理论/语义Web

ISBN 978-7-111-23734-1

9 787111 237341

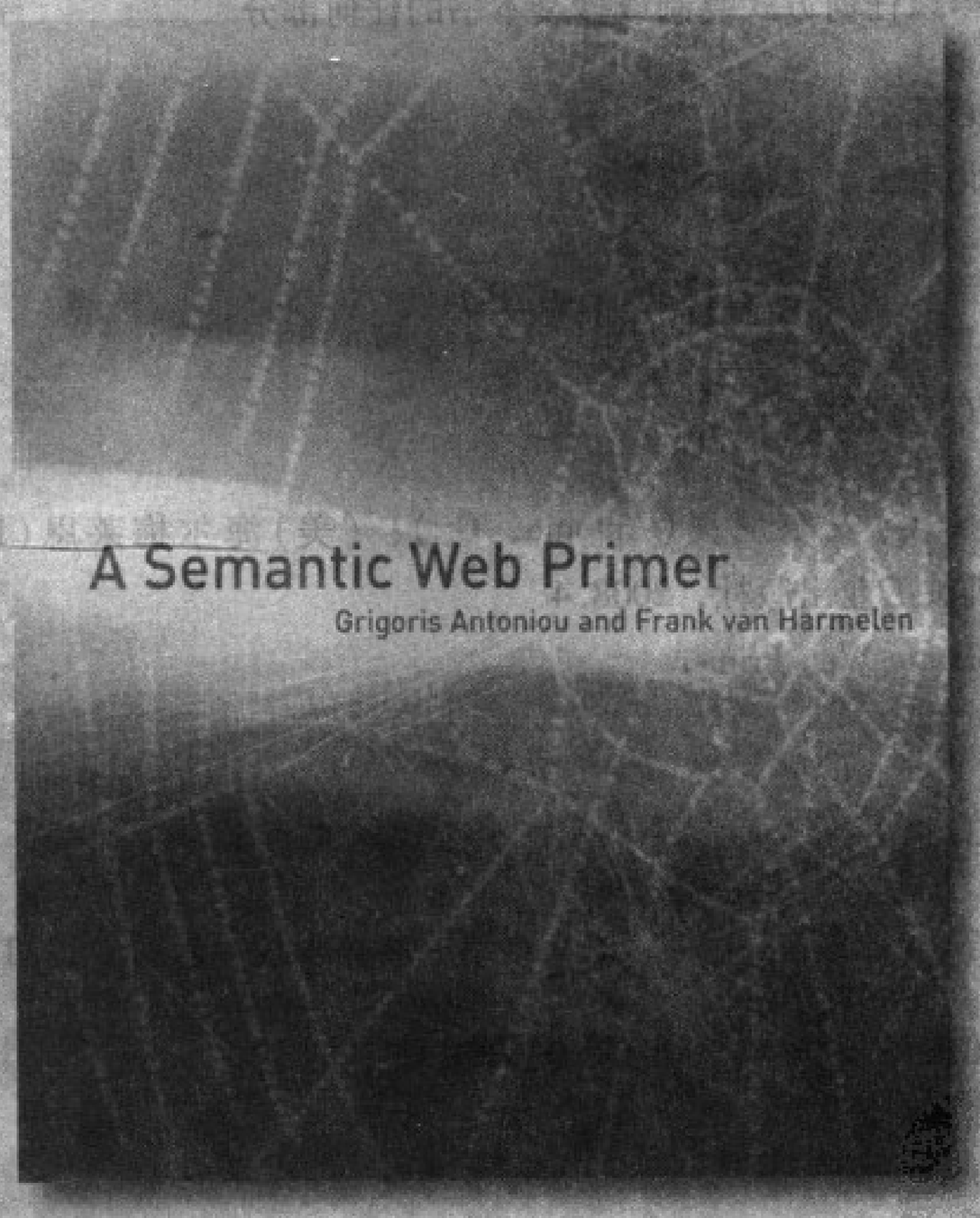
定价：32.00元

TP18/166
2008

计 算 机 书

语义网基础教程

(希) Grigoris Antoniou 赫拉斯研究与技术基金会 (荷) Frank van Harmelen Vrije大学 著 陈小平 中国科学技术大学 等译



A Semantic Web Primer

机械工业出版社
China Machine Press

本书是一本语义网的入门性教科书，内容包括语义网技术概述、XML 语言、结构化文档、RDF 和 RDF Schema、网络本体语言 OWL、与语义网相关的逻辑与推理规则和本体工程等方面。本书还介绍了一些知名行业公司的语义网应用实例，极具实用参考价值。

本书主要用于大学语义网课程的教学，也是语义网技术专业人士十分难得的参考读物。本书还可作为大学计算机专业本科生教材。

Grigoris Antoniou, Frank van Harmelen: A Semantic Web Primer(ISBN 0-262-01210-3).
Authorized translation from the English language edition published by MIT Press.
Copyright © 2004 by MIT Press.
All rights reserved. No part of this publication may be reproduced or distributed in any means, or stored in a database or retrieval system, without the prior written permission of the publisher.
Chinese simplified language edition published by China Machine Press.
Copyright © 2008 by China Machine Press.

本书中文简体字版由美国麻省理工学院出版社授权机械工业出版社独家出版。未经出版者预先书面许可，不得以任何方式复制或抄袭本书的任何部分。

版权所有，侵权必究。
本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2006-4280

图书在版编目(CIP)数据

语义网基础教程/(美)安东尼奥(Antonio, G.)；(美)海尔梅莱恩(Harmelen, F.)著；陈小平译. —北京：机械工业出版社，2008.4
书名原文：A Semantic Web Primer
(计算机科学丛书)

ISBN 978-7-111-23734-1

Ⅰ. 语…　Ⅱ. ①安…　②陈…　Ⅲ. 语义网络－高等学校－教材　Ⅳ. TP18

中国版本图书馆 CIP 数据核字(2008)第 036291 号

机械工业出版社(北京市西城区百万庄大街 22 号　邮政编码　100037)
责任编辑：王　玉
北京京北制版厂印刷 · 新华书店北京发行所发行
2008 年 4 月第 1 版第 1 次印刷
184mm×260mm · 12.5 印张
标准书号：ISBN 978-7-111-23734-1
定价：32.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：(010)68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自 1998 年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与 Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann 等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出 Tanenbaum，Stroustrup，Kernighan，Jim Gray 等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近 260 个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”。为了保证这两套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大

学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这两套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业的教学度身订造的。其中许多教材均已为 M. I. T.，Stanford，U. C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色—有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzjsj@ hzbook. com

联系电话：(010)68995264

联系地址：北京市西城区百万庄南街 1 号

邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

中文版前言

自 20 世纪 90 年代末发端以来，语义网研究已经成为一项全球性事业，相关的各种研究项目在欧洲和美国几乎同时出现，亚洲的研究者也表现出极大的兴趣。

从那时以来，中国学者在本体映射、推理技术、基于本体的浏览与搜索等各个方面展开了研究工作，形成了非常活跃的研究局面。通过这些工作，中国已经成为国际语义网研究领域中的一个重要成员。2006 年第一届亚洲语义网大会在北京举行。中国学者在语义网的其他国际学术会议上也占据了重要位置，最近还在一个语义匹配的竞赛中取得了领先地位。此外，中国还有一个非常活跃的讨论语义网网站(http：//bbs. w3china. org/)。

语义网研发的一个首要目标是实现语义可共用性(semantic interoperability)。当简单的语法可共用性无法实现或难以满足应用需求时，就必须借助语义可共用性。在教学中，我们经常通过欧洲语言与汉语之间的巨大语法差异来说明不同语言背景的读者仅借助语法手段实现不同语种材料的共用是多么困难。因此，本书中文版的出版是很有意义的。这部十分出色的中译本由中国科学技术大学陈小平教授主持完成。

我们非常高兴陈小平教授和他的团队愿意承担本书的翻译工作，他们做得非常出色。我们在访问中国科学技术大学期间，与译者对翻译中遇到的问题进行了广泛、深入的讨论。我们通过回答这些问题对原著的内容做进一步澄清，但更多情况下，我们的讨论涉及原著的更深层内容，甚至远远超越原著的范围，进入对前沿问题的探讨。在翻译过程中，荷兰阿姆斯特丹 Vrije 大学的高级研究员黄智生博士也对翻译提供了专业性支持，帮助澄清了翻译中的很多技术性问题。

作为作者，我们认为他们是最好的翻译团队。我们完全相信中译本的质量和完整性；并且诚挚地希望本书中文版将为从事语义网的教与学的师生提供有益的帮助，并对进一步提升语义网的研发兴趣发挥积极作用。

Since it' s inception in the late '90s, the Semantic Web programme has been an example of a truly global enterprise. Almost simultaneously, research programs arose in Europe and the US, and there was significant interest from researchers in Asia.

Since those early years, Chinese researchers have built a very active research community, with activities in many different research areas, such as ontology mapping, inference techniques, ontology-based browsing and searching, and many others.

Through these activities, China has become a significant player in Semantic Web research. China was hosting the first Asian Semantic Web Conference in Beijing in 2006, Chinese research groups have a significant presence at international Semantic Web conferences, there is a very active web-site and bulletin board on Semantic Web topics (http://bbs.w3china.org/), and a Chinese entry gained a leading position in recent Ontology Alignment Challenges.

A major goal of Semantic Web research and development is to enable semantic interoperability. Semantic interoperability is required when simple syntactic interoperability is not achievable, or not sufficient for a particular task. In lectures, we often use the large syntactic differences between European languages and the Chinese language to illustrate how hard is to achieve interoperability using only syntactic means. It is therefore fitting that this Chinese translation of our Semantic Web Primer has now become available. This very fine translation is from the hand of Prof. XiaoPing Chen of the University of Science and Technology (USTC) in Hefei.

We were very pleased with the offer of Prof. XiaoPing Chen of the University of Science and Technology (USTC, Hefei) to translate this volume. He and his team have done a remarkable job. During an invited visit to their group they asked many knowledgable, detailed and probing questions about unclarities they had come across during the translation process. Sometimes, answering their questions was a matter of providing some clarification, but more often, their questions pointed at deep issues underlying the text, and often lead to technical discussions reaching far beyond the text of the Primer, and into open research issues. Prof. XiaoPing Chen and his team were professionally supported by dr. Zhisheng Huang, a senior researcher at the Vrije Universiteit Amsterdam, who helped to clarify many detailed translation issues.

We could not have wished for a better translating team. We have all confidence in the quality and integrity of the translation, and we sincerely hope that this translated edition of the Semantic Web Primer will contribute to further strengthen the interests in Semantic Web research and development, and that it will be a useful aid to students and teachers alike.

Frank van Harmelen

Grigoris Antoniou

译 者 序

语义网是一个有些“奇特”的新兴领域。它是现实主义与理想主义的奇妙组合。其现实主义主要表现为坚持以“易行性”(tractability)为语义网技术的首要条件，从而确保技术成果计算上的可行性，这与传统人工智能的风格不尽相同。另一方面，它又设立了很高的目标，试图通过网络信息“理解”的机械化而实现网络信息处理的根本性变革，这与万维网及其应用的现行技术路线也不一样。可以预见，语义网技术一旦市场化，必然引起万维网应用的全面改观。

语义网飞速发展导致本领域信息严重过量，从而给初学者带来极大不便。如原版前言指出的那样，本书的主旨是要解决这一难题。确实，作为一本语义网技术的导论性教材，尤其相对于国内众多读者，本书在很多方面是相当出色的——简练而不简陋、通俗而不肤浅、前沿而不学究……原著的出版对于读者来说是值得庆幸的，对于译者而言却实属不幸。如果译者的工作能对中文读者有所助益，将是不幸中之万幸！经过两年多的努力，中译本终于与读者见面了。对于翻译中的种种疏漏差错，请读者批评指正。

本书的翻译是集体合作的结果，并经过了多轮翻译和修订。分工如下：周熠(第1、8章)、杨斌(第7章)、姜杰汇、蒙云松(第6章)、宋志伟、李永喜(第4章)、李欢、吉建民(第5章)、杨方凯(第3章)、范正洁(第2章)。陈小平对所有译稿作了全面修订。

原作者 Frank van Harmelen 解答了一些疑难问题。原作者的同事、语义网专家黄智生博士为本书的翻译提供了大量帮助。对机械工业出版社的编辑所做的工作，在此一并致以谢忱！

译 者

2007 年 6 月

前　言

万维网(WWW)的出现改变了人与人之间交流的方式，改变了信息传播、获取的方式和商业运作的方式。语义网(Semantic Web)包含着能够显著改进当前万维网及其应用的技术。本书旨在介绍这门发展中的新兴技术。

判断一本书的成功与否应当参照作者的写作目的。本书是一本语义网的入门性教科书，主要用于大学语义网课程的教学，同时也适用于语义网技术的自学。

既然所有关于语义网的信息都可以在网上找到，到底有没有必要编写这样一本教材呢？作者认为有这个必要，因为在网上有太多质量参差不齐的资源和过量的信息。这些信息有的是正确的，有的却是过时的或错误的，而且大部分讨论的是具体细节。任何初学者或想开设语义网课程的教师都面临着这些问题。本书的目的就是帮助读者解决这个难题。

一部教科书必须对所涵盖的内容有所选择，尤其是在这样一个飞速发展的领域里，更要着眼于那些直到将来仍然有价值的基础内容。当然，作者总是难以摆脱自己的偏爱。

即便对于被涵盖的内容，本书也无意成为一本包含全部细节的工具书。对于某些专题，比如 XML，已经有很多专著。由于关于语义网技术的所有定义和手册都可以在网上找到，所以没有必要为语义网这个领域专门写一本工具书。因此，本书着眼于介绍语义网的基本思想和主要技术，并为读者灵活地掌握和应用这些基础内容提供充分的细节，进而使读者有能力从其他来源自行获得所需的其余细节。在每章的最后，给出一些相关参考文献并附带必要的说明。

致谢

感谢 Jeen Broekstra、Michel Klein 和 Marta Sabou，为我们在荷兰 Vrije 大学开设基于万维网的知识表示课程提供很多第一手资料；感谢 Annette ten Teije、Zharko Aleksovski 和 Wouter Jansweijer 仔细阅读本书的初稿。感谢 Christoph Grimmer 和 Peter Koenig 校对部分书稿并帮助完成图表的生成和 LaTeX 处理。此外，感谢为本书终稿提供专业协助的 MIT 出版社员工和定义 LaTeX 2ε 宏命令的 Christopher Manning。

目　　录

出版者的话

专家指导委员会

中文版前言

译者序

前言

第 1 章　语义网概览 …… 1

1.1　万维网现状 …… 1

1.2　从当前万维网到语义网：实例分析 …… 3

1.2.1　知识管理 …… 3

1.2.2　企业对个人电子商务 …… 3

1.2.3　企业对企业电子商务 …… 4

1.2.4　个人代理：未来的展望 …… 5

1.3　语义网技术 …… 5

1.3.1　显式的元数据 …… 6

1.3.2　本体 …… 7

1.3.3　逻辑 …… 9

1.3.4　代理 …… 11

1.3.5　语义网和人工智能的对比 … 12

1.4　语义网的逐层递进 …… 12

1.5　全书总览 …… 14

1.6　小结 …… 14

推荐读物 …… 15

第 2 章　XML 结构化网络文档 …… 17

2.1　引言 …… 17

2.2　XML 语言 …… 20

2.2.1　序言 …… 20

2.2.2　元素 …… 20

2.2.3　属性 …… 21

2.2.4　注释 …… 21

2.2.5　处理指令 …… 22

2.2.6　合式 XML 文档 …… 22

2.2.7　XML 文档的树形模型 …… 22

2.3　结构化 …… 23

2.3.1　DTD 方式 …… 24

2.3.2　XML Schema …… 28

2.4　命名空间 …… 33

2.5　XML 文档的寻址和查询 …… 35

2.6　处理 …… 38

2.7　小结 …… 43

推荐读物 …… 44

练习与设计 …… 45

第 3 章　用 RDF 描述网络资源 …… 47

3.1　引言 …… 47

3.2　RDF 的基本思想 …… 49

3.2.1　资源 …… 49

3.2.2　属性 …… 49

3.2.3　陈述 …… 49

3.2.4　关于陈述的三种观点 …… 49

3.2.5　陈述的具体化 …… 51

3.2.6　数据类型 …… 51

3.2.7　对 RDF 的评论 …… 52

3.3　基于 XML 的 RDF 语法 …… 53

3.3.1　rdf: resource 属性 …… 55

3.3.2 嵌套描述 …… 56
3.3.3 rdf: type 元素 …… 57
3.3.4 简略语法 …… 57
3.3.5 容器元素 …… 58
3.3.6 具体化 …… 62
3.4 RDF Schema 的基本思想 …… 62
3.4.1 类和属性 …… 62
3.4.2 类层次结构和继承 …… 63
3.4.3 属性层次结构 …… 64
3.4.4 RDF 与 RDFS 层次 …… 64
3.5 RDF Schema 语言 …… 65
3.5.1 核心类 …… 66
3.5.2 用于定义关系的核心属性 … 66
3.5.3 用于约束属性的核心属性 … 67
3.5.4 用于具体化的常用属性 …… 68
3.5.5 容器类 …… 68
3.5.6 工具属性 …… 68
3.5.7 例子：大学 …… 69
3.5.8 例子：机动车 …… 70
3.6 用 RDF Schema 描述 RDF 和 RDF Schema …… 71
3.6.1 RDF …… 71
3.6.2 RDF Schema …… 72
3.7 RDF 和 RDF Schema 的公理语义 …… 73
3.7.1 基本方法 …… 73
3.7.2 基本谓词 …… 74
3.7.3 RDF …… 74
3.7.4 RDF Schema …… 76
3.8 RDF 和 RDFS 的直接推理系统 … 78
3.9 使用 RQL 查询 …… 79
3.9.1 基本查询 …… 80
3.9.2 使用 select-from-where 查询 …… 80
3.9.3 模式查询 …… 81
3.10 小结 …… 82
推荐读物 …… 82
练习与设计 …… 83
第 4 章 网络本体语言 OWL …… 85
4.1 引言 …… 85
4.1.1 本体语言需求 …… 85
4.1.2 RDF Schema 表达能力的局限性 …… 86
4.1.3 OWL 与 RDF/RDFS 的兼容性 …… 87
4.1.4 三个 OWL 子语言 …… 87
4.2 OWL 语言 …… 89
4.2.1 语法 …… 89
4.2.2 头部 …… 89
4.2.3 类元素 …… 90
4.2.4 属性元素 …… 90
4.2.5 属性约束 …… 92
4.2.6 特殊性质 …… 94
4.2.7 布尔组合 …… 95
4.2.8 枚举 …… 96
4.2.9 实例 …… 96
4.2.10 数据类型 …… 97
4.2.11 版本控制信息 …… 98
4.2.12 OWL 分层 …… 98
4.3 例子 …… 99
4.3.1 一个非洲野生生物本体 …… 99
4.3.2 一个打印机本体 …… 103
4.4 OWL 自描述 …… 107
4.4.1 命名空间 …… 107
4.4.2 元类 …… 108
4.4.3 类等价 …… 108
4.4.4 类的组合 …… 110
4.4.5 类属性约束 …… 110

4.5 未来扩展 …… 112
4.5.1 模块与导入 …… 112
4.5.2 默认值 …… 113
4.5.3 封闭世界假设 …… 113
4.5.4 唯一名假设 …… 113
4.5.5 过程附加 …… 113
4.5.6 属性链接规则 …… 113
4.6 小结 …… 114
推荐读物 …… 114
练习与设计 …… 115
第5章 逻辑和推理：规则 …… 117
5.1 引言 …… 117
5.2 单调规则的例子：家庭关系 …… 119
5.3 单调规则：语法 …… 120
5.3.1 规则 …… 120
5.3.2 事实 …… 121
5.3.3 逻辑程序 …… 121
5.3.4 目标 …… 121
5.4 单调规则：语义 …… 122
5.4.1 谓词逻辑语义 …… 122
5.4.2 常证据和带参证据 …… 123
5.5 非单调规则：动机和语法 …… 124
5.5.1 非正式讨论 …… 124
5.5.2 语法定义 …… 125
5.6 非单调规则的例子：有经纪人的交易 …… 126
5.6.1 卡洛斯需求的形式化描述 …… 126
5.6.2 可选公寓的表达 …… 127
5.6.3 选择一间公寓 …… 128
5.7 XML 中的规则标记：单调规则 …… 129
5.7.1 项 …… 129
5.7.2 原子公式 …… 130
5.7.3 事实 …… 130
5.7.4 规则 …… 131
5.7.5 查询 …… 131
5.7.6 一个 DTD …… 132
5.7.7 RuleML 的另一种数据模型 …… 132
5.8 XML 中的规则标记：非单调规则 …… 134
5.8.1 例子 …… 134
5.8.2 一个 DTD …… 135
5.9 小结 …… 136
推荐读物 …… 136
练习与设计 …… 137
第6章 应用 …… 139
6.1 引言 …… 139
6.2 Elsevier 的横向信息产品 …… 139
6.2.1 背景 …… 139
6.2.2 问题 …… 139
6.2.3 语义网技术的作用 …… 140
6.3 奥迪的数据整合 …… 141
6.3.1 背景 …… 141
6.3.2 问题 …… 141
6.3.3 语义网技术的作用 …… 141
6.4 瑞士人寿保险公司的技能寻获 …… 143
6.4.1 背景 …… 143
6.4.2 问题 …… 144
6.4.3 语义网技术的作用 …… 144
6.5 EnerSearch 的智囊门户网站 …… 145
6.5.1 背景 …… 145
6.5.2 问题 …… 146
6.5.3 语义网技术的作用 …… 146
6.6 在线学习 …… 148
6.6.1 背景 …… 148

6.6.2 问题 …… 149
6.6.3 语义网技术的作用 …… 149
6.6.4 在线学习的本体 …… 150
6.7 Web 服务 …… 151
6.7.1 背景 …… 151
6.7.2 问题与语义网技术的作用 …… 151
6.8 其他场景 …… 154
6.8.1 伦敦警察局的多媒体收藏索引 …… 154
6.8.2 戴姆勒 - 克莱斯勒的在线采购 …… 155
6.8.3 诺基亚的设备可共用性 …… 155
推荐读物 …… 156
第 7 章 本体工程 …… 158
7.1 引言 …… 158
7.2 手工构建本体 …… 158
7.2.1 确定范围 …… 158
7.2.2 考虑复用 …… 159
7.2.3 列举术语 …… 159
7.2.4 定义分类 …… 159
7.2.5 定义属性 …… 159
7.2.6 定义侧面 …… 160
7.2.7 定义实例 …… 160
7.2.8 检查异常 …… 160
7.3 复用现成本体 …… 161
7.3.1 专家知识汇编 …… 161
7.3.2 整合词汇表 …… 161
7.3.3 高层本体 …… 161
7.3.4 主题层次体系 …… 161
7.3.5 语言学资源 …… 162
7.3.6 本体图书馆 …… 162
7.4 使用半自动方法 …… 162
7.5 On-To-Knowledge 语义网体系结构 …… 164
7.5.1 知识获取 …… 165
7.5.2 知识存储 …… 165
7.5.3 知识维护 …… 166
7.5.4 知识使用 …… 166
7.5.5 技术可共用性 …… 166
推荐读物 …… 167
设计 …… 167
第 8 章 结论和展望 …… 170
8.1 集成 …… 170
8.2 一些技术问题 …… 170
8.2.1 网络本体语言：少就是多 …… 170
8.2.2 规则和本体 …… 170
8.3 未来预测 …… 171
附录 OWL 抽象语法 …… 173
索引 …… 180

第 1 章　语义网概览

1.1　万维网现状

万维网改变了人们彼此交流的方式和商业运作的方式。发达社会正在向知识经济和知识社会转型，而万维网处于这场革命的核心位置。

这种发展使得人们对计算机的看法也发生了变化。起初，计算机仅仅用做数值计算，而现在则主要用于信息处理，典型的应用包括数据库、文档处理和游戏等等。眼下，人们对计算机关注的焦点正在经历新的转变，将其视为信息高速公路的入口。

绝大部分现有的网络内容适合于人工处理。即使是从数据库自动生成的网络内容，通常也会丢弃原有的结构信息。目前万维网的典型应用方式是，人们在网上查找和使用信息、搜索和联系其他人、浏览网上商店的目录并且填表格定购商品等等。

现有软件工具没有很好地支持这些应用。除了建立文件间联系的链接之外，最有价值和必不可少的工具是搜索引擎。

基于关键词的搜索引擎，比如 Alta Vista、Yahoo、Google 等，是使用现有万维网的主要工具。毫无疑问，假如没有这些搜索引擎，万维网不会取得现在这么大的成功。然而，搜索引擎的使用也存在一些严重的问题：

- 高匹配、低精度。即使搜到了主要相关页面，但它们与同时搜到的 28 758 个低相关或不相关页面混在一起，检索的效果就很差。太多和太少一样令人不满意。
- 低匹配或无匹配。有时用户得不到任何搜索结果，或者漏掉了一些重要的相关页面。虽然对于现在的搜索引擎来说，这种情况发生的频率不高，但确实会出现。
- 检索结果对词汇高度敏感。使用最初填写的关键词往往不能得到想要的结果，因为相关的文档里使用了与检索关键词不一样的术语。这当然令人不满意，因为语义上相似的查询理应返回相似的结果。
- 检索结果是单一的网页。如果所需要的信息分布在不同的文档中，则用户必须给出多个查询来收集相关的页面，然后自己提取这些页面中的相关信息并组织成一个

整体。

有趣的是，尽管搜索引擎技术在发展，但主要的困难还是上述几条。技术的发展速度似乎落后于网上内容量的增长速度。

此外，即使搜索是成功的，用户仍必须自己浏览搜索到的文档，从中提取所需的信息，也就是说，对极其耗时的信息检索本身，搜索引擎并没有提供很多支持。因此，用信息检索(information retrieval)来描述搜索引擎为用户提供的功能，是不确切的；用信息定位(location finder)可能更加合适。另外，由于现有网络搜索的结果不易直接被其他软件进一步处理，因此搜索引擎的应用往往是孤立的。

目前，为网络用户提供更强大支持的主要障碍在于，网上内容的含义不是机器可解读的(machine-accessible)。当然，有一些工具能够检索文档、把它们分割成更小的部分、检查拼写并统计词频等等。可是，一旦牵涉到解释句子含义和提取对用户有用的信息，现有的软件就能力有限了。举一个简单的例子。对现有技术而言，以下两个句子的含义是难以区分的：

我是一个计算机科学的教授。

2 你不妨认为，我是一个计算机科学的教授。

使用文本处理技术，怎样才能改善现在的状况呢？一种解决方案是沿用现有表达方式来表示网上内容，并发展基于人工智能和计算语言学的一些日趋复杂的技术以解决现有难题。这条途径迄今已探索多时，尽管取得了一些进展，但其任务仍然显得过于困难。

另一条途径是用一种更容易被机器处理[⊖]的表示方法来描述网上内容，并采用智能技术来利用这种表示方法所提供的便利。我们把这个革命性的方案称为语义网(Semantic Web)运动。值得注意的是，语义网并不是一种和现有万维网平行的新的全球化信息高速公路，相反，它将在现有万维网的基础上逐渐成长。

语义网运动由——万维网联盟(W3C)发起，其倡导者正是在20世纪80年代后期发明万维网的Tim Berners-Lee。他期望通过这场运动来实现他的万维网初始蓝图，在这个蓝图中，信息含义所起的作用比现今万维网中重要得多。

语义网的发展得到了产业界的大力支持和各国政府的大量资助。美国政府设立了DAML(DARPA Agent Markup Language)项目，欧盟第六研究框架计划也将语义网作为关键内容之一。

⊖ 文献中经常使用另一个词语“机器可理解”，我们认为不妥，因为会产生误解。智能代理不必真正“理解”信息，只要有效地处理信息就够了，尽管这种处理有时让人以为机器具有理解能力。

1.2 从当前万维网到语义网：实例分析

1.2.1 知识管理

知识管理所关注的是在一个组织中获取、处理和维护知识。由于大型商业机构把内部知识视为一种能够提高生产力、创造新的价值和增强竞争力的智力资源，知识管理已成为他们的一项关键性工作，对于地理分布广泛的国际组织来说尤其如此。 3

目前大多数可用信息只具有弱结构组织形式，如文本、音频和视频等。从知识管理的角度来说，现有技术存在以下诸方面的局限：

- 信息搜索。公司通常依靠基于关键词的搜索引擎，这方面的局限性上文已经做了简要阐述。
- 信息抽取。需要人工浏览文档，以便从中找出相关的信息。现有的智能代理(intelligent agent)还不能令人满意地完成这项任务。
- 信息维护。现在这方面仍然存在一些问题，比如术语的不相容性和无法移除过时信息等等。
- 信息挖掘。虽然可以用数据挖掘(data mining)等手段提取隐藏在公司数据库中的新知识，但对于分布式的、弱结构化的文档集合，这个任务仍然是困难的。
- 信息视图。经常需要限制某些雇员对某些信息的浏览权限。“视图”意味着隐藏某些信息，这个概念在数据库领域里为人们所熟知，但对企业内部网(或者万维网)尚难以实现。

语义网的目的是允许更高级的知识管理系统，使得：

- 可以根据含义用概念空间组织知识；
- 具备相容性检查和新知识提取功能的自动工具支持知识维护；
- 问答系统取代关键词搜索——用人性化的方式检索、提取和表示所需知识；
- 支持跨文档的问答功能；
- 能够规定用户察看信息的指定部分(甚至文档的某些部分)的权限。 4

1.2.2 企业对个人电子商务

企业对个人(B2C)电子商务是万维网用户最主要的商业活动。典型场景是用户访问一个或者多个网上商店，浏览他们的报价，选择并定购产品。

理想化的情况是，用户可以收集所有(至少是主要的)网上商店的信息，包括价格、条款、说明(比如有效期)等信息，从而选择最好的产品。但是如此规模的手工浏览太费时间，

通常用户在做决定之前只会访问一个或者少数几个网上商店。

为了改善这种情形，以购物机器人的形式出现了一些网上购物工具，这种软件代理(software agent)[⊖]能够访问多家商店、提取产品和价格信息、汇总市场情况。这些功能是借助于从网上商店提取信息的打包程序来实现的。对于不同的网上商店，必须开发不同的打包程序。这种方法是有缺陷的。

有关信息是通过关键词搜索和其他文本分析手段从网上商店获取的。这个过程利用了特定信息段之间临近关系的假设以进行信息打包(比如，价格是通过单词 price 后面接着的 $ 符号和一个正数来表示的，则根据三者之间的邻近关系把价格信息“打包”)。这种假设不一定正确，因而这种启发式方法未必总能奏效。所以，能够如此提取的信息是有限的。像运输费用、交货时间、关于目的地国家的限制、安全等级和隐私保护措施等信息一般不能提取出来，而所有这些因素可能对用户的决策起重要作用。另外，编制打包程序是费时的，而且网上商店的设置一旦发生变动，则需要重新编制打包程序。

语义网技术支持软件代理对产品信息和服务条款的解读，使得：

- 能够正确地提取价格和产品信息，并根据用户需求解释配送和隐私保护措施；
- 能够从其他来源，比如独立的排名机构或客户群体，补充提供网上商店的信誉信息；

- 不再需要低层次的打包程序；
- 更加高级的购物代理(shopping agent)能够代表购买者的利益与商店代理(shop agent)进行自动协商。

1.2.3 企业对企业电子商务

大多数用户在网上的商业活动只与 B2C 电子商务有关，但是，网络技术的最大经济价值体现在企业对企业(B2B)电子商务中。

企业间数据交换的传统方式是通过电子数据交换(EDI)途径。然而，这个技术太复杂了，只能为专家所理解，编程和维护都很困难，且易出错。每一对企业间的数据交换都需要专门编程，开发费用很大。另外，EDI 是一种孤立的技术手段，所交换的数据无法方便地与其他商业应用集成使用。

因特网似乎是企业间交流的理想的基础设施。企业越来越关注基于因特网的解决方案，已经出现了诸如企业间门户(B2B portal)这样的商业模型。然而，由于没有统一的标准，企业对企业电子商务模式的发展遇到了阻碍。HTML(hypertext markup language，超文本标记

⊖ 在最一般的意义下，人工智能领域中的 agent 指能够在动态不确定环境中，独立完成复杂任务的物理或计算实体，译为“主体”。本书中，附带各种修饰词的 agent 主要指各种代表用户利益，独立完成复杂任务的计算机程序，这种情况下一般译为“代理”。——译者注

语言)不包含信息的结构和信息的语义，所以，它无法对上述事务提供有效的支持。XML这种新标准是一个很大的进展，但它只适用于事先对词汇表及其含义达成共识的情况。

语义网的实现将使企业间的合作不需要太大的开销。借助于标准的抽象领域模型(abstract domain model)可以消除术语差异，实现用不同术语书写的文档之间的翻译转换，从而实现数据交换。拍卖、协商和草拟合同将由软件代理自动(或半自动)地完成。

1.2.4 个人代理：未来的展望

迈克尔出了一场轻微车祸，他觉得颈部有点疼。他的私人医生建议他去做一组理疗。迈克尔要他的语义网代理(Semantic Web agent)设计出一些可能的就医方案。

语义网代理从医生的自动代理(the doctor's agent)那里查到了医生推荐的治疗信息，并查看了迈克尔的医疗保险公司提供的理疗师名单，从中找出距离迈克尔的办公室或住处不超过10公里的理疗师和权威排名机构对他们的评价，然后，寻找和迈克尔的日程表不冲突 6
的预约时间。几分钟后，它给出了两个建议方案。但迈克尔都不满意。其中一个方案的预约时间在两星期后，而在另一个方案里，迈克尔则必须在交通高峰时刻驱车前往。所以，迈克尔把时间限制得更严格一些，让他的语义网代理重新设计方案。

几分钟后，语义网代理给出了一个的新方案，可以在两天之内预约一个声誉很高的理疗师，但有一些小问题需要解决。首先，迈克尔的一些不太重要的工作安排需要调整，语义网代理承诺它将调整迈克尔的工作时刻表，如果迈克尔采纳这个方案的话。另外，该理疗师不在保险公司网站公布的名单里，因为他的收费超过了保险公司规定的最高限额。不过，语义网代理从另一张名单上找到了这个理疗师的名字，并且得知，根据保险公司的有关规定，迈克尔可以接受超过最高限额的治疗。此外，迈克尔的语义网代理还和理疗师的自动代理谈成了优惠价格。该理疗师最近才决定收取高额治疗费，所以急于招徕新的病人。

既然增加的费用有限，迈克尔对这个新方案很感兴趣。不过，这个语义网代理是几天前才安装的，所以迈克尔要求它回答几个问题：对理疗师声誉的评价是怎样做出的？为什么迈克尔需要调整他的工作日程表？议价过程是怎样进行的？语义网代理一一给出了适当的解释。

迈克尔觉得很满意，他的语义网代理将使他繁忙的生活变得更轻松。他要求他的语义网代理采取所有必要的步骤，落实这个治疗方案。

1.3 语义网技术

1.2节所描绘的场景不是科幻小说，它的实现也不需要科学上的革命性突破。我们有理由断定，这是一个工程技术挑战而不是科学挑战，其中所有重要方面已经得到了部分的

解决。目前，最需要的是集成、标准化、工具的开发和用户的采用。当然，未来的技术进
7 步会产生更加先进的语义网。

在下面各小节里，我们将简要描述实现上述功能所需的几个必备技术。

1.3.1 显式的元数据

目前的网络内容更适合让人而不是程序来阅读。HTML 是直接或借助工具书写网页的主导语言。一个理疗师的网页片断看上去是这样的：

```
<h1 >Agilitas Physiotherapy Centre </h1 >
Welcome to the home page of the Agilitas Physiotherapy Centre.
Do you feel pain? Have you had an injury? Let our staff
Lisa Davenport, Kelly Townsend (our lovely secretary)
and Steve Matthews take care of your body and soul.

<h2 >Consultation hours </h2 >
   Mon 11am - 7pm <br >
   Tue 11am - 7pm <br >
   Wed 3pm - 7pm <br >
   Thu 11am - 7pm <br >
   Fri 11am - 3pm <p >
   But note that we do not offer consultation
   during the weeks of the
   <a href = ". . . " >State Of Origin </a > games.
```

这些信息以适合人类阅读的方式显示，但对于机器来说就有问题了。基于关键词的搜索能够识别理疗(physiotherapy)和咨询时间(consultation hours)这些单词。一个智能代理甚至能够识别该理疗中心的职员。但是，它分不清谁是医生、谁是秘书，识别具体的咨询时间就更加困难了(这需要链接到 the State Of Origin 游戏的页面里去浏览)。

通过语义网途径解决这个问题的办法不是开发更高级的智能代理，而是从网页表示的角度入手。如果用更合适的语言取代 HTML，用它们编写的网页就可以更好地表达其内容。这种网页不仅可以包含格式信息以生成便于人类阅读的文档，还可以包含描述内容的其他
8 信息。比如在我们的例子中，可以包含下面这些信息：

```
<company >
   <treatmentOffered >Physiotherapy </treatmentOffered >
   <companyName >Agilitas Physiotherapy Centre </companyName >
   <staff >
      <therapist >Lisa Davenport </therapist >
      <therapist >Steve Matthews </therapist >
      <secretary >Kelly Townsend </secretary >
```

```
    </staff>
</company>
```

与 HTML 相比，这种表达方式更容易让机器处理。元数据(metadata)指的是“关于数据的数据”。元数据刻画的是数据的含义，也就是语义网中的语义。

在 1.2 节的场景里，从网页中提取治疗细节、日程表、预约时间、价格和产品描述等信息似乎没有什么困难，似乎所有这些信息都可以从已有的网页内容中直接检索出来。但正如刚解释的那样，如果采用基于文本的信息操作而不是利用机器可处理的元数据的话，这是不可能做到的。⊖

对于目前的网页开发而言，用户没有必要一定是计算机科学专家，他们可以用一些工具来达到自己的目的。于是，问题来了，用户为什么要放弃 HTML 转而采用语义网语言？通过把当前情形和万维网初创时的情形加以比拟，我们可以给出一个乐观的回答。第一批用户之所以决定采用 html 是因为当时 html 已经成为一种标准，他们想成为最早的用户而获益。随着网络工具变得越来越多、越来越好，其他人也跟着采用 HTML。很快，HTML 就变成了一种全球通用的标准。

类似地，我们现在正目睹 XML 普及的早期阶段。虽然 XML 本身对于实现语义网蓝图来说是不充分的，但它是重要的第一步。XML 和 RDF 是当前与语义网相关的 W3C 标准，采纳它们的最早用户可能是一些对知识管理和企业对企业电子商务感兴趣的大型机构。这一势头将推动越来越多的工具销售商和终端用户采用这项技术。

这将是语义网探索具有决定性意义的一步，但同时也是一场挑战。就像前面提到的，当前最大的挑战不在于科学方面，而在于技术是否被普遍接受。 [9]

1.3.2 本体

本体论(ontology)这个词来源于哲学。在哲学中，本体论是形而上学的一个分支，研究存在的本质(根据希腊文字$Ovτολογια$的直译)，即什么是最一般意义上真实存在的，以及如何描述它们。例如，下面这个观察反映了典型的本体论承诺(ontological commitment)：这个世界是由具体对象组成的，这些对象可以按照共有属性划分为抽象的类。

最近几年，计算机科学借用了很多其他领域的术语并赋予它们与原有含义大不相同的技术性含义，本体这个词就是其中之一。现在我们可以谈论“一个本体”，原来只能说“本体”。本书沿用由 T. R. Gruber 提出、后经 R. Studer 改进的定义：一个本体是一个概念体系(conceptualization)的显式的形式化规范。

⊖ 这里似乎没有明确解释本节标题中的定语“显式的”。“显式的”(explicit)可理解为“借助表达式本身直接表示出来的”，与之相反的概念是“隐式的”(implicit)。上面给出的理疗师网页片断的前一种表达是隐式的，后一种是显式的。——译者注

一般地说，一个本体形式地刻画一个论域(domain of discourse)。一个典型的本体由有限个术语以及它们之间的关系组成。术语(term)指给定论域中的重要的概念(对象的类)。例如，以一个大学为论域，则教工、学生、课程、教室和学科是一些重要的概念。

概念间的关系通常包括类的层次结构。一个层次结构规定类 C 是另一个类 C' 的子类，如果 C 中的所有对象都包含在 C' 中。例如，所有教职人员都是教工。图 1-1 表示一个大学论域的层次结构。

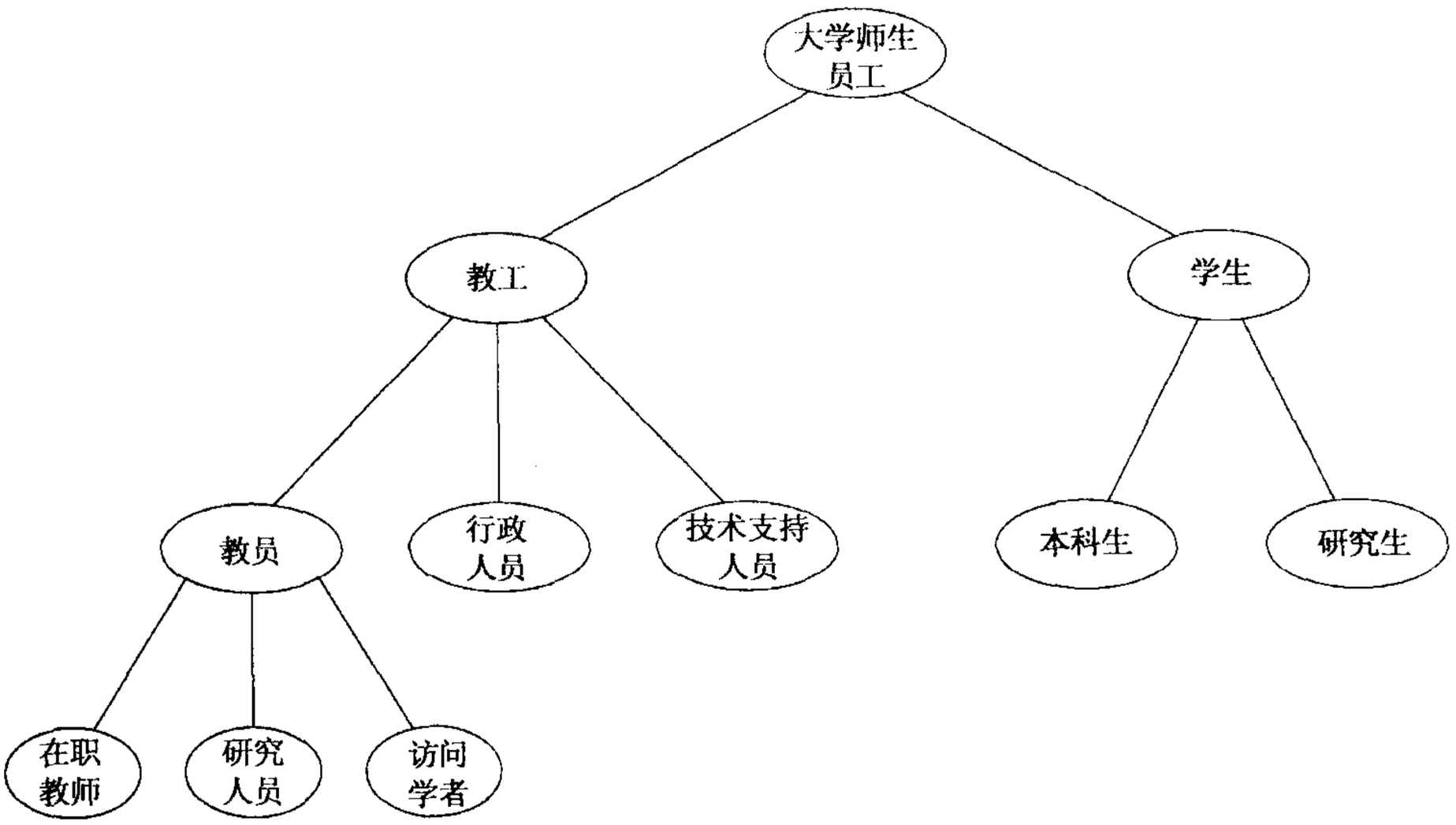


图 1-1 一个层次结构

除了子类关系之外，本体还可以包括以下信息：

- 属性（例如，X 教 Y）
- 值约束(例如，只有教职人员才能授课)
- 不相交描述(例如，教职人员和普通员工不相交)
- 对象间逻辑关系的规定(例如，一个系至少要有 10 个教职人员)

在万维网这个环境中，本体提供了对给定领域的一种共识。这种共识对于消除术语差别是必要的。比如，一个应用中的邮政编码可能正好是另一个应用中的区域代码。另一种情况是两个不同的应用可能使用同样的术语来代表不同的含义。在大学 A 中，术语
10 “course”可能指学位(比如计算机科学)，而在大学 B 中可能指课程(比如 CS 101)。通过把各自的术语映射到一个公共本体或者定义本体之间的直接映射，可以消除这些术语差异。容易看出，不管采用哪种方案，本体都支持语义可共用性(semantic interoperability)。⊖

⊖ 语义网研究的一个目标是提供一套技术，使任何一个网页上的信息可以被机器自动地解读，从而可以被各种软件代理“共同使用”，这就是“可共用性”在语义网领域中的基本含义。——译者注

本体可用于网站的组织和导航。现有很多网站在页面左边列出了概念层次结构中最高层的术语，用户可以点击其中之一来浏览相关子目录。

本体也可用于提高网络搜索的精确度。搜索引擎可以精确地根据本体中的概念查找相关页面，而不是收集所有出现某些(通常是含混的)关键词的页面。这样一来，在查询和网页中存在的术语差异就可以消除了。

另外，可以利用本体在网络搜索中试探更一般或更特殊的查询。如果一个查询失败了，没有找到相关文档，搜索引擎可以向用户推荐更一般的查询。甚至可以考虑让搜索引擎主 11
动执行这样的查询，在碰巧用户接受建议的场合，这样做就缩短了响应时间。反之，如果查到的结果太多，搜索引擎可以建议用户使用更特殊的查询。

在人工智能(AI)领域，开发和使用本体语言是一个长期的传统，语义网研究可以在此基础上进一步推进。目前网络领域中最重要的本体语言如下：

- XML 为结构化文档提供一种表层语法，但没有对这些文档的含义提供语义约束。
- XML Schema 是一种定义 XML 文档结构的语言。
- RDF 是一种描述对象(“资源”)和对象间关系的数据模型，并为这种数据模型提供一个简单的语义，这些数据模型可以用 XML 语法来表示。
- RDF Schema 是一种刻画 RDF 资源的属性和类的词汇描述语言，带有关于这些属性和类的一般 - 特殊关系的层次结构语义。
- OWL 是一个更丰富的词汇描述语言，可以刻画诸如类间关系(如不相交性)、类的基数(如“恰有一个”)、相等、更丰富的属性类型、属性的特征(如对称性)和枚举类等。

1.3.3 逻辑

逻辑是研究推理原理的学科，可以追溯到亚里士多德。一般而言，逻辑首先提供了表示知识的形式语言。其次，逻辑提供易于理解的形式语义——在绝大多数逻辑系统中，句子含义的定义无需涉及对知识的操作。通常我们称之为陈述性知识，就是只描述什么是成立的而不必关心是怎样推出的。

逻辑的第三个作用是，可以借助自动推理机从给定的知识演绎(推导)出一些结论，从而使隐含的知识外显出来。[⊖]这样的推理机在人工智能中得到了广泛的研究。举一个推理的例子。假设我们知道所有教授都是教职人员，所有教职人员都是员工，并且迈克尔是一个教授。在谓词逻辑中，这些信息表达如下： 12

⊖ 至少在目前的情况下，任何推理机都必须以某个逻辑为基础，用此逻辑来规定前提集与结论之间的“可推关系”。——译者注

$prof(X) \rightarrow faculty(X)$

$faculty(X) \rightarrow staff(X)$

$prof(michael)$

于是，可以推出如下结论：

$faculty(michael)$

$staff(michael)$

$prof(X) \rightarrow staff(X)$

注意，这个例子包含着通常在本体中的知识。因此，逻辑可以用来推出被隐式表达的本体知识。这样做也有助于发现意料之外的关系和不相容性。

不过，逻辑比本体更具一般性。智能代理可以用逻辑来做决策和选择行动序列。例如，一个商店代理可以根据下列规则给顾客打折

$loyalCustomer(X) \rightarrow discount(5\%)$

其中顾客的信用度是根据公司数据库中存储的数据决定的。一般来说，在表达能力和计算效率之间存在着相互制约关系。一种逻辑的表达能力越强，它推出结论的计算开销就越大。如果遇到不可计算性的障碍，有些结论是不可能推出的。幸运的是，绝大多数和语义网相关的知识具有相对严格的形式。例如，上面的例子涉及形为“如果前提条件成立，则某结论成立”的规则，而且只需要考虑有限多个对象。逻辑的这个部分是易处理的，并且已有支持它的高效率推理工具。

逻辑的一个重要优点是它提供了对结论的解释——推导步骤的序列是可重现的。不仅如此，人工智能研究者还发明了一些表达解释的人性化方法，把证明组织成自然演绎形式，把多个低层推理步骤合成一个直观上的简单步骤。在此基础上，把对一个结论的推理还原为推理中用到的事实和推理规则的集合，从而给出这个推理的解释。

13 解释可以提高用户对语义网代理的信心（见 1.2.4 节理疗的例子），因此它对语义网来说是重要的。Tim Berners-Lee 曾经谈到用“Oh yeah?”按钮来要求代理做出解释。

对代理之间的互动行为来说，解释同样是重要的。某些代理可以有能力推导逻辑推论，而其他代理只有证明验证的能力，也就是检验推导序列是不是合法的证明的能力。举一个简单的例子，假设某个网上商店的自动代理甲对某个个人的自动代理乙发送如下消息“你欠我 80 美元”（当然不是用自然语言，而是用一种机器能够处理的形式语言）。代理乙可以要求代理甲做出解释，代理甲可以用如下形式的序列给出回答：

网上购物记录 80 美元

配送证明（例如，UPS 跟踪号）

本店计价规则：

$$purchase(X, Item) \wedge price(Item, Price) \wedge delivered(Item, X) \rightarrow owes(X, Price)$$

典型情况下，事实来源于某些指定的网页地址(其信誉可由自动代理验证)，而规则来源于共享的商业本体或者在线商店的规定。

为了使逻辑能够在网上应用，必须使它能够和其他数据一起使用，而且使它能被机器处理。为此，目前正在研究如何用网络语言表示逻辑知识和逻辑证明。最初的工作在 XML 层次上展开，但是将来，规则和证明需要在 RDF 和本体语言(如 DAML + OIL 和 OWL)层表示。

1.3.4 代理

代理是能够自主地(autonomously)和主动地(proactively)运作的软件。其概念来源于面向对象程序设计和基于组件的软件开发。

语义网上的个人代理(图 1-2)将接收用户给出的任务和偏好，在网上查找相关信息，与其他代理交流，根据用户需求和偏好分析信息做出选择，最后将答案提交给用户。1.2.4 节理疗例子中迈克尔的私人代理就是一个这样的代理。 14

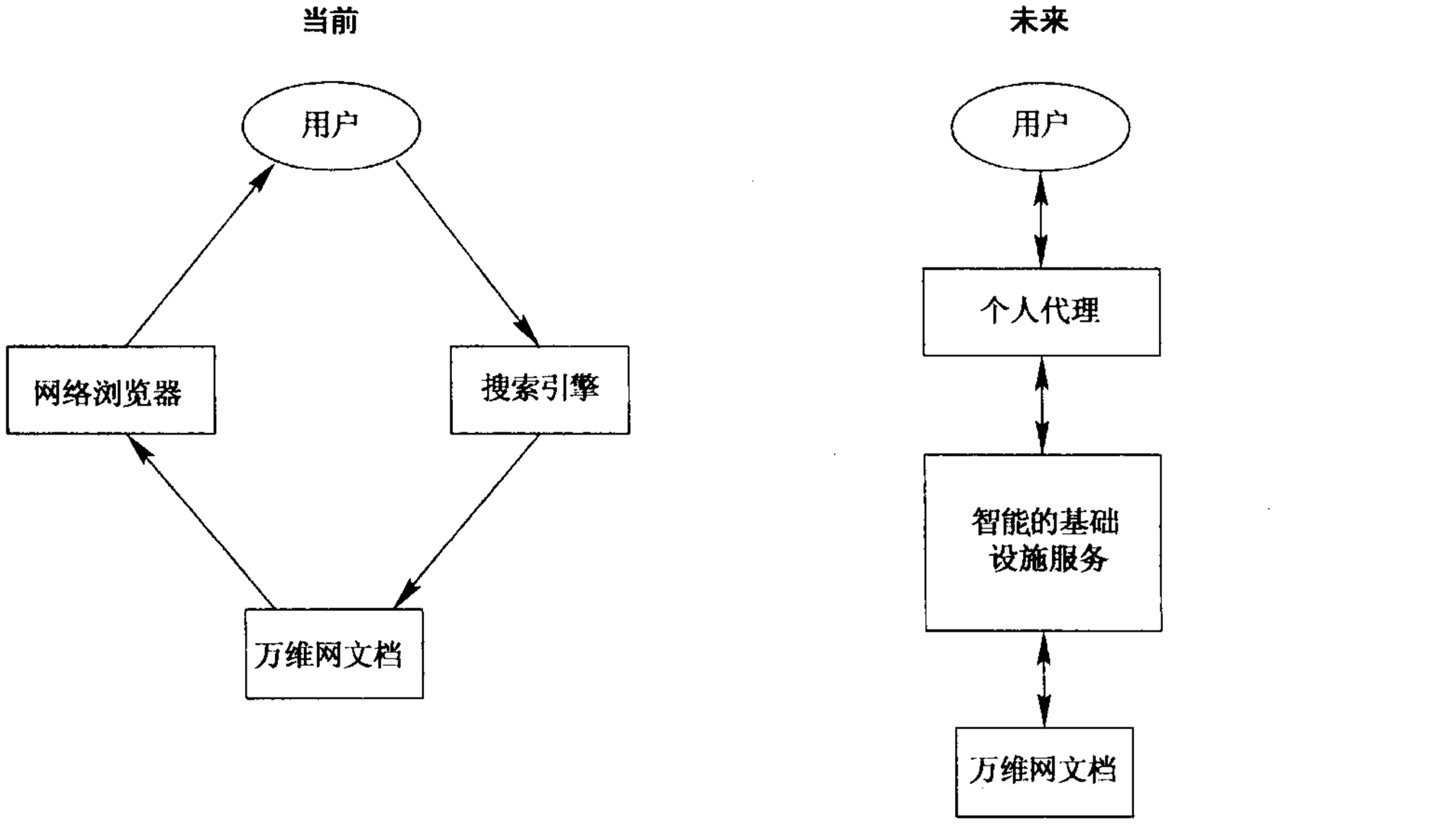


图 1-2 智能个人代理

值得注意的是，代理将不会取代语义网上的人类用户，也不是非得由它们来做决定。在很多情况下，它们的作用是收集和整理信息，为用户提供备选方案。例如，迈克尔的个人代理所做的工作是提供它能够找到的两个最好的解决方案，而一个旅行代理则可以根据用户偏好寻找合适的旅游线路。

语义网代理会用到我们上面概述的所有技术：

- 使用元数据从网络资源识别和提取信息；
- 使用本体辅助网络搜索，解释检索到的信息，并与其他代理进行交流；
- 用逻辑处理检索到的信息并推导结论。

15 此外，还需要一些其他技术，比如代理的通信语言等。同时，对于一些高级的应用，需要形式化地表示代理的信念、愿望和意图，并建立和维护用户模型。不过，这些课题和语义网技术可以各自独立地加以研究，所以本书对这些课题不做进一步讨论。

1.3.5 语义网和人工智能的对比

前面提到，实现语义网所需要的大多数技术建立在人工智能已有工作的基础之上。考虑到人工智能已有很长的历史，但在实用中并不总是很成功，有人可能会担心出现最坏的情况，即语义网将重蹈人工智能的覆辙：承诺太多，导致期望太高而最后却没有实现（至少在承诺的时间范围里没有实现）。

这个担心是站不住脚的。语义网蓝图的实现并不需要达到人类智能的层次。就像我们试图解释的那样，实际上这个挑战是通过另一条途径来应对的。完全地解决人工智能问题是一个深层次的科学问题，或许比得上物理学（解释物理世界）和生物学（解释生命世界）的核心问题。所以，如果像过去某些时刻承诺的那样，要在十年到二十年之内实现人类层次的人工智能，则遇到困难是不足为奇的。

但对语义网来说，部分解决问题是可行的。即使一个智能代理不能推出一个人类用户可以推出的所有结论，它仍然可以为一个比现有万维网高级得多的万维网做出贡献。这又引出了人工智能和语义网的另一区别。如果说人工智能的最终目标是建造出呈现人类或更高级智能的智能代理的话，那么语义网的目标是协助人类用户处理网上的日常事务。

显然，语义网将广泛地利用现有人工智能技术，而这些技术的进步将促进语义网的发展。但是，并不需要等到人工智能达到很高的成就，现有人工智能技术已足以帮助语义网向其蓝图前进很大一步了。

1.4 语义网的逐层递进

语义网的研究要一步一步地推进，每一步都要在前一层之上搭建新的一层（layer）。这样做的理由是，小步前进比较容易达成共识。一般情况下，会有多个团队沿着不同方向研

16 究同一个问题，由此产生的不同想法之间的竞争是科学发展的主要驱动力。但是，从工程角度看问题，标准化是必要的。如果大多数研究者在某些方面达成一致而在另一些方面不一致，将有助于确立哪些是共同点。这样一来，即使更宏伟的目标失败了，至少还能得到

一些积极的部分成果。

一个标准一旦建立，更多团体和公司就会采纳它，而不会继续观望哪个研究方案将最终胜出。语义网的本性促使公司和个人用户必须采取行动——开发工具、添加内容和使用这些内容，而不能坐等语义网蓝图的全部实现(那可能还需要十年左右的时间)。

要在语义网的某一层上搭建更高一层，应该遵循两个原则：

- 向下可兼容性。位于某一层的代理应能解释和使用低层的信息。例如，掌握 OWL 语义的代理要具备足够的能力，以充分利用由 RDF 和 RDF Schema 描述的信息。
- 向上部分可理解性。另一方面，位于某一层的代理应能部分地使用更高层的信息。例如，一个只掌握 RDF 和 RDF Schema 语义的代理可以部分地解释用 OWL 表达的知识，而无视 RDF 和 RDF Schema 之上的其他成份。

图 1-3 描绘了语义网设计蓝图的主要层次(Tim Berners-Lee 称之为“layer cake”)。

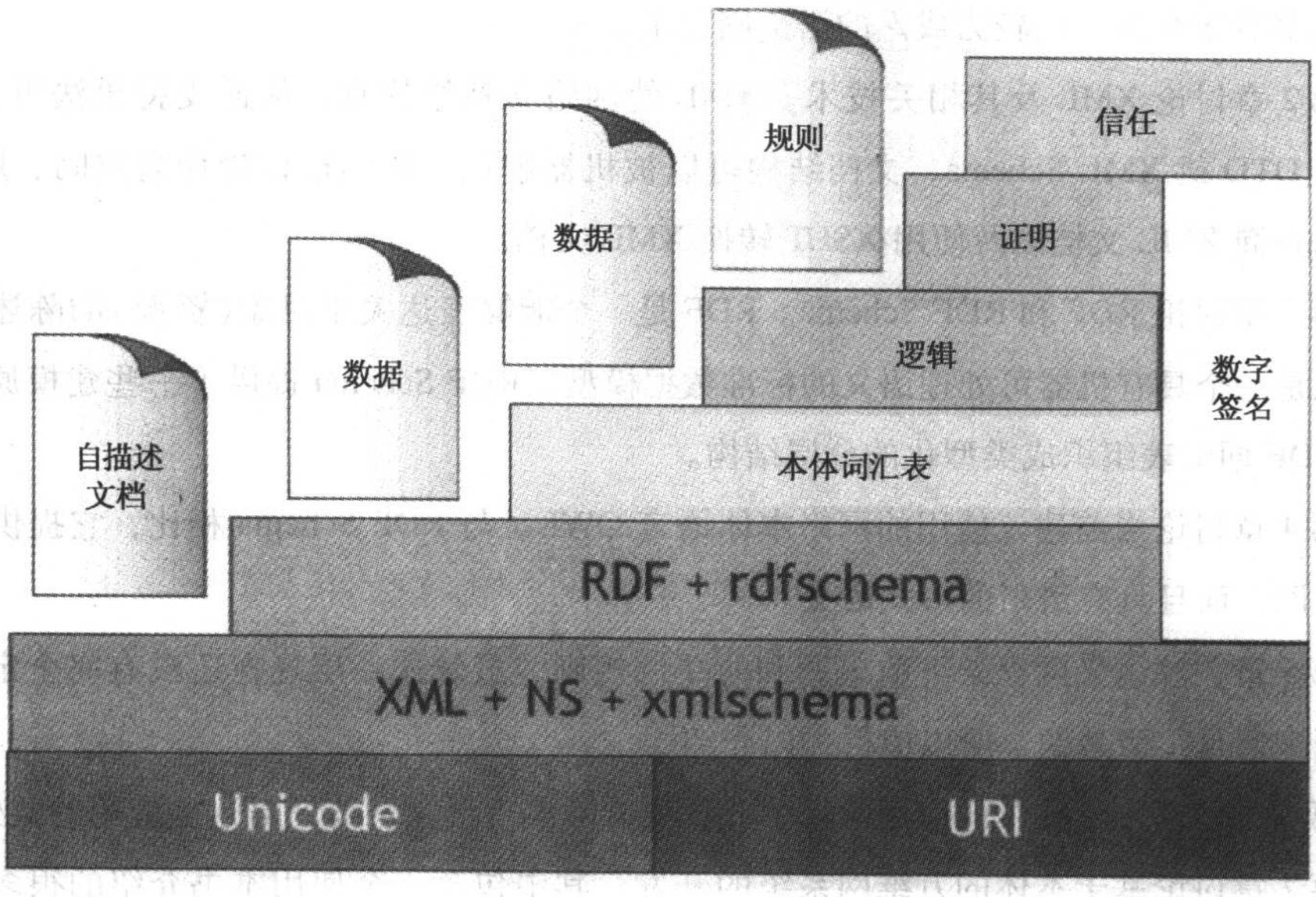


图 1-3　语义网的分层方案

底层是 XML，它是一种根据用户自定义的词汇表编写结构化网络文档的语言。XML 特别适合在万维网上传递文件。

RDF 如实体 - 关系模型，是一种基本数据模型，用来编写关于网络对象(资源)的简单陈述句。RDF 的数据模型并不依赖于 XML，虽然 RDF 有基于 XML 的词法。因此，在图 1-3 中，它位于 XML 层之上。

RDF Schema 提供将网络对象组织成层次结构的建模原语。其中，关键原语是类和属性、子类和子属性关系、定义域和值域限定等，RDF Schema 是基于 RDF 的。

RDF Schema 可以看作是一种编写本体的初始语言。但需要更强大的本体语言扩展 RDF Schema，以表达网络对象之间更加复杂的关系。逻辑层用于进一步增强本体语言的能力，以便表达应用相关的陈述性知识。

证明层(Proof layer)涉及实际的演绎过程、证明的(低层)网络语言表示以及证明的验证。

最后，信任层(Trust layer)将随着数字签名(digital signature)和其他种类知识的使用而出现，并以其他可信任的代理的推荐或其他机构以及顾客群体的排名和认证为基础。有时，“信任的网络”这个词用来表明，信任将按照与万维网自身同样的分布式和混沌方式加以组织。处于分层结构这座金字塔的顶端，信任是一个高层而且至关重要的概念：只有当用户信任它的操作(安全)和它所提供信息的质量时，万维网才能发挥它的全部潜力。

17
|
18

1.5 全书总览

本书着重介绍一些较为成熟的语义网技术。

第2章讨论 XML 及其相关技术。XML 使网络文档结构化，从而支持语法可共用性。借助于 DTD 或 XML Schema，文档结构可以被机器解读。我们还讨论命名空间、用 XPath 访问和查询 XML 文档以及使用 XSLT 转换 XML 文档。

第3章讨论 RDF 和 RDF Schema。RDF 是一个能够表达关于对象(资源)的陈述句的语言，也是一个具有机器可处理语义的标准数据模型。RDF Schema 提供了一些建模原语，用于将 RDF 词汇表组织成类型化的分层结构。

第4章讨论当前建议使用的网络本体语言 OWL。与 RDF Schema 相比，它提供更多的模型原语，而且具有清晰的形式语义。

第5章关注语义网框架下的单调和非单调规则。虽然这一层现在还没有完全定义，但基本原理还是很清楚的，所以值得加以介绍。

第6章讨论几个具体的应用领域，并说明实际实现语义网蓝图对这些应用的益处。

第7章讨论基于本体的万维网系统的开发，其中包含一个使用本书介绍的很多技术的微型项目。

最后，第8章简要地介绍语义网领域正在争论的几个问题。

1.6 小结

- 语义网是一个旨在改进万维网现状的运动。
- 语义网的核心想法是使用机器可处理的网络信息。
- 语义网的关键技术包括显式的元数据、本体、逻辑和推理以及智能代理。
- 语义网的研发逐层进行。

19

推荐读物

一篇出色的介绍性文章，1.2.4 节中的场景部分源出于此。

- T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American* 284 (May 2001): 34-43.

一本关于万维网历史和未来的鼓舞人心的书籍：

- T. Berners-Lee, with M. Fischetti. *Weaving the Web*. San Francisco: Harper, 1999.

介绍语义网的一些在线读物：

- T. Berners-Lee. Semantic Web Road Map. September 1998. <http://www.w3.org/DesignIssues/Semantic.html>. 20
- T. Berners-Lee. Evolvability. March 1998. <http://www.w3.org/DesignIssues/Evolution.html>.
- T. Berners-Lee. What the Semantic Web Can Represent. September 1998. <http://www.w3.org/DesignIssues/RDFnot.html>.
- E. Dumbill. The Semantic Web: A Primer. November 1, 2000. <http://www.xml.com/pub/a/2000/11/01/semanticweb/>.
- F. van Harmelen and D. Fensel. Practical Knowledge Representation for the Web. <http://www.cs.vu.nl/~frankh/postscript/IJCAI99-III.html>.
- J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems* 16 (March-April 2001): 30-37.
  预印版在 <http://www.cs.umd.edu/users/hendler/AgentWeb.html>.
- S. Palmer. The Semantic Web, Taking Form. <http://infomesh.net/2001/06/swform/>.
- S. Palmer. The Semantic Web: An Introduction. <http://infomesh.net/2001/Swintro/>.
- A. Swartz. The Semantic Web in Breadth. <http://logicerror.com/semanticWeb-long>.
- A. Swartz and J. Hendler. The Semantic Web: A Network of Content for the Digital City. <http://blogspace.com/rdf/SwartzHendler.html>.
- R. Jasper and A. Tyler. The Role of Semantics and Inference in the Semantic Web: A Commercial Challenge. <http://www.semanticweb.org/SWWS/program/position/soi-jasper.pdf>.

几门语义网课程(包含在线资料)：

- F. van Harmelen et al. Web-Based Knowledge Representation. <http://www.cs.vu.nl/~marta/wbkr.html>.
- J. Heflin. The Semantic Web. <http://www.cse.lehigh.edu/~heflin/courses/semweb/>.
- A. Sheth. Semantic Web. <http://lsdis.cs.uga.edu/SemWebCourse/index.html>.
- H. Boley, S. Decker, and M. Sintek. Tutorial on Knowledge Markup Techniques. <http://www.dfki.uni-kl.de/km/knowmark/>.

语义网及其相关问题的最新动态：

- <http://www.SemanticWeb.org>.
- <http://www.w3.org/2001/sw/>.
- <http://www.ontology.org>.

部分提供语义网相关技术信息的研究论文：

- D. Fensel, J. Hendler, H. Lieberman and W. Wahlster, eds. *Spinning the Semantic Web*. Cambridge, MA: MIT Press, 2003.
- J. Davies, D. Fensel and F. van Harmelen, eds. *Towards the Semantic Web: Ontology-Driven Knowledge Management*. New York: Wiley, 2002.

21

- The conference series of the *International Semantic Web Conference* (see <http://www.semanticweb.org/>).

第2章　XML结构化网络文档

2.1　引言

HTML(hypertext markup language，超文本标记语言)源于SGML(standard generalized markup language，标准通用标记语言)，是目前书写网页的标准语言。SGML是一个国际标准(ISO 8879)，定义一种独立于设备和系统且人机均可读的信息表示方法。这种标准的重要性在于提供有效的交流途径，以支持技术进步和商业合作。在万维网这样的分布式环境中，不存在中央集权，不能强制推行通常意义下的标准，故而由W3C(World Wide Web Consortium，万维网联盟)制订的一系列标准称为推荐标准(recommendation)。

遵从SGML的语言称为SGML应用。HTML就是一种这样的应用，引入它的原因是SGML对于和因特网相关的某些用途过于复杂。为了克服HTML的一些缺陷，另一种SGML应用——XML(extensible markup language，可扩展性标记语言)应运而生。下面通过一个简单的例子来考察XML语言产生的动机，此例中的网页包含一本书的信息。

```
<h2 >Nonmonotonic Reasoning: Context-Dependent
     Reasoning </h2 >
<i >by <b >V. Marek </b > and <b >M. Truszczynski </b > </i > <br >
Springer 1993 <br >
ISBN 0387976892
```

相同信息的典型XML表达如下：

```
<book >
  <title >
    Nonmonotonic Reasoning: Context-Dependent Reasoning
  </title >
  <author >V. Marek </author >
  <author >M. Truszczynski </author >
  <publisher >Springer </publisher >
  <year >1993 </year >
  <ISBN >0387976892 </ISBN >
</book >
```

在分析HTML和XML这两种表示方式的区别之前，先考察它们的一些相似之处。首

先，这两种表示方式都用到了标签(tag)，例如 <h2> 和 </year>。诚然，HTML 和 XML 均是标记语言(markup language)，用户可以用它们书写一些内容并提供信息以说明这些内容所起的作用。

XML 和 HTML 类似，也是建立在标签之上的。这些标签可以嵌套使用(标签中含有标签)。XML 的所有标签都必须按一定规则配对使用(因此，若使用了一个起始标签 <title>，就必须有一个对应的结尾标签 </title>)，而 HTML 允许一些标签如
无需配对。夹在起始、结尾标签之间的内容和起始、结尾标签一起，构成一个元素(element)。(近期发展的语言 XHTML 使 HTML 朝 XML 的方向靠拢：任何有效的 XHTML 文档也是一个有效的 XML 文档，因而 XHTML 的起始、结尾标签也应是配对的)。

粗略地看来，人类用户可以很容易地阅读 HTML 和 XML 文档，这两种语言都设计得易于人们理解和使用。但对于机器，情况又是怎样的呢？设想一下，一个智能代理试图从上例网页中检索此书的作者姓名。假设上述 HTML 页面可以通过网络搜索来定位(这在实际中是有疑问的，现有搜索引擎的局限性已得到证明)。没有任何显式信息指出作者是谁。一个合乎情理的猜测是：作者的姓名会紧接在书名之后或在单词 by 之后，但无法保证这种书写习惯总被遵守。即使在遵守这个习惯的场合，究竟是有两个作者 V. Marek 和 M. Truszczynski，还是只有一个叫“V. Marek and M. Truszczynski”的作者呢？显然，需要进行更多的文本处理才能回答这个问题，而这种处理不能保证正确性。

这个问题源于 HTML 文档不包含结构信息，即文档成分及其相互关系的信息。相对而
[24] 言，XML 文档远比 HTML 文档更容易被机器解读，因为它描述了文档的每一成分，并借助 XML 嵌套结构定义了文档成分之间的关系。例如，<author>标签出现在 <book>标签的辖域内，所以，<author>标签描述的是这本特定的书的属性。于是，处理 XML 文档的机器可以演绎出，这个 author 元素所指涉的就是包含它的那个 book 元素，而不必像对 HTML 文档那样只能根据临近关系推测这个事实。XML 的另一个优点是允许定义值约束(比如，年的取值必须是小于 3000 的四位数)。*XML 允许机器可解读的信息表示方式。*

当然，必须承认，HTML 表示还描述了文档的格式，这是 XML 表示所没有的。然而，这一特点并非 HTML 的优势，反而是它的弱点：HTML 必须规定格式。事实上，HTML 文档的主要用途在于显示信息(除了与其他文档建立链接这种功能之外)。另一方面，*XML 的内容和格式是分开的*。相同的信息可以用不同的方式显示出来，而不需要同一内容的许多副本；再者，除了用于显示信息，这些内容可以用来做很多其他的事情。

再以一个著名的物理定律为例。HTML 的表示形式为

```
<h2>Relationship force-mass</h2>
<i>F=M×a</i>
```

XML 的表示形式为

```
<equation>
   <meaning>Relationship force-mass</meaning>
   <leftside>F</leftside>
   <rightside> M×a</rightside>
</equation>
```

把此处的 HTML 文档与先前的 HTML 文档比较，就会发现这两个文档基本上用的是相同的标签。这并不奇怪，因为这些标签在 HTML 中是预定义的。相比之下，第二个 XML 文档用了与第一个 XML 文档截然不同的标签。这与表示的预期用途有关。HTML 表示用于显示信息，所以标签集是固定的，如列表、字体和颜色等。在 XML 中，可以用不同的方式来使用信息，因此应该由用户定义适用于特定应用的词汇表。因此，*XML 是一种关于标记的元语言——没有固定的标签集，而是由用户自行定义自己的标签。* 25

正像人们如果不使用同样的语言，彼此之间就无法进行有效的交流，如果万维网上的应用需要交流与合作，那么就必须就公用词汇表达成一致。各种团体和商界已经着手定义自己特定的词汇表，并创建基于 XML 的应用(或其扩展，因此名字 XML 中含有 extensible 一词)。在很多领域中，人们已经定义了这种基于 XML 的应用，比如数学领域(MathML)、生物信息领域(BSML)、人力资源领域(HRML)、天文学领域(AML)、新闻领域(NewsML)和投资领域(IRML)。

同样，W3C 也在 XML 之上定义了多种语言，比如 SVG 和 SMIL。这种方法也应用于定义 RDF(见第 3 章)。

注意，XML 语言可以作为不同应用之间的统一数据交换格式(uniform data exchange format)。事实上，现今 XML 的这种用法远远超过了原先期望它作为文档标记语言的用法。一些公司常常需要从他们的客户和商业伙伴那儿检索信息，并相应地更新公司的数据库。如果没有如 XML 这样统一的标准，那么各方都需要开发特定的操作和查询软件，这样就会增加技术方面的开支。此外，如果一方决定更改它的数据库格式的话，那么所有软件都需要更新。

本章 2.2 节详细描述 XML 语言，2.3 节介绍 XML 文档的结构化。在关系数据库中，必须定义表的结构。类似地，XML 文档的结构也需要定义。这可以通过书写 DTD(document data definition，文档类型定义)这种较陈旧的方法来实现，或者通过 XML Schema(XML 模式)这种将要逐渐取代 DTD 的现代方法来实现。

2.4 节描述支持 DTD 和 XML Schema 模块化的命名空间。2.5 节介绍用 XPath 存取和查询 XML 文档。最后，2.6 节介绍如何用 XSL 和 XSLT 来转化 XML 文档，以便加以显示(或达到其他目的)。 26

2.2 XML 语言

一个 XML 文档包含一个序言、一系列元素和一个可选的尾注(此处不做讨论)。

2.2.1 序言

XML 文档的序言由一个 XML 声明和一项可选的对外部结构化文档的引用组成。下面是 XML 声明的一个例子:

```
<? xml version = "1.0"encoding = "UTF-16"? >
```

上述声明指定当前文档是 XML 文档,并且定义了文档的版本号和特定系统下的字符编码方式(如 UTF-8、UTF-16 和 ISO 8859-1)。字符编码方式不是强制的,但是指定编码是个好习惯。有时也会指明文档是否是自包含的,也就是说,它是否要引用外部结构化文档:

```
<? xml version = "1.0"encoding = "UTF-16"standalone = "no"? >
```

对外部结构化文档的引用可以是这样的:

```
<! DOCTYPE book SYSTEM "book.dtd" >
```

此例中文档的结构信息在本地文件 book.dtd 中,当然引用也可以是一个 URL。如果仅使用本地可识别的名字或者仅使用 URL,那么就用 SYSTEM 标记;如果这两者都用的话,就要用 PUBLIC 标记。

2.2.2 元素

XML 的元素代表 XML 文档描述的"事物",比如书籍、作者和出版商。这些元素构成了 XML 文档的主要概念。一个元素由一个起始标签、元素内容(content)和一个结尾标签构成。例如:

```
<lecturer > David Billington </lecturer >
```

用户几乎可以随意选择标签名,只有很少的限制。最重要的限制是,标签名的首字符必须是字母、下划线或冒号,而且不能以字符串"xml"的任何组合方式开头(如"Xml"和"xML")。

27

元素内容可以是文本,或是其他的元素,或为空。例如,

```
<lecturer >
   <name > David Billington </name >
   <phone > +61-7-3875 507 </phone >
```

```
</lecturer>
```

如果一个元素不包含任何内容，则称为空元素(element)。形如

```
<lecturer></lecturer>
```

的空元素可以缩写为：

```
<lecturer/>
```

2.2.3 属性

一个空元素未必毫无意义，它可以拥有属性(attributes)形式的一些特性。属性是元素的起始标签中的名字-值(name-value)对，例如：

```
<lecturer name="David Billington"phone="+61-7-3875 507"/>
```

下面是一个非空元素的属性的例子：

```
<order orderNo="23456" customer="John Smith"
       date="October 15, 2002">
  <item itemNo="a528" quantity="1"/>
  <item itemNo="c817" quantity="3"/>
</order>
```

这条信息也可以通过把属性改写为嵌套元素的形式来表达：

```
<order>
  <orderNo>23456</orderNo>
  <customer>John Sm_th</customer>
  <date>October 15, 20025</date>
  <item>
    <itemNo>a528</itemNo>
    <quantity>1</quantity>
  </item>
  <item>
    <itemNo>c817</itemNo>
    <quantity>3</quantity>
  </item>
</order>
```

使用元素写法还是属性写法通常只是个人偏好。但要注意属性不能嵌套。

2.2.4 注释

注释是一段文本，分析器会忽略这行文本。注释形式如下

```
<! --This is a comment-->
```

2.2.5 处理指令

处理指令(Processing Instruotion，PI)提供一种向处理元素的应用传递信息的机制，一般形式如下：

```
<? target instruction? >
```

例如，

```
<? stylesheet type = "text/css" href = "mystyle. css"? >
```

处理指令提供了一种在另外的说明性表示中增加过程性处理的可能。

2.2.6 合式 XML 文档

如果一个 XML 文档的语法正确，那么它就是一个合式 XML 文档。部分语法规则如下：

- 在文档的最外层只能有一个元素(称为根元素(root element))。
- 每个元素都包含一个起始标签和一个与之相对应的结尾标签。
- 标签之间不可以重叠，如 <author> <name> Lee Hong </author> </name> 的形式违法。
- 同一元素内部的属性名不可相同。
- 元素名和标签名必须合法。

29

2.2.7 XML 文档的树形模型

可以把合式 XML 文档表示成树形结构，因此树形结构为 XML 提供了一个形式化的数据模型。这种表达方法往往是有益的。以下列文档为例：

```
<? xml version = "1. 0"encoding = "UTF-16"? >
<! DOCTYPE email system "email. dtd" >
<email >
 <head >
   <from name = "Michael Maher"
         address = "michaelmaher@ cs. gu. edu. au" / >
   <to   name = "Grigoris Antoniou"
         address = "grigoris@ cs. unibremen. de" / >
   <subject > Where is your draft? </subject >
 </head >
 <body >
Grigoris, where is the draft of the paper
```

```
  you promised me last week?
  </body>
</email>
```

图 2-1 显示了这个 XML 文档的树形表示。这是一棵含有序关系并带有标记的树：

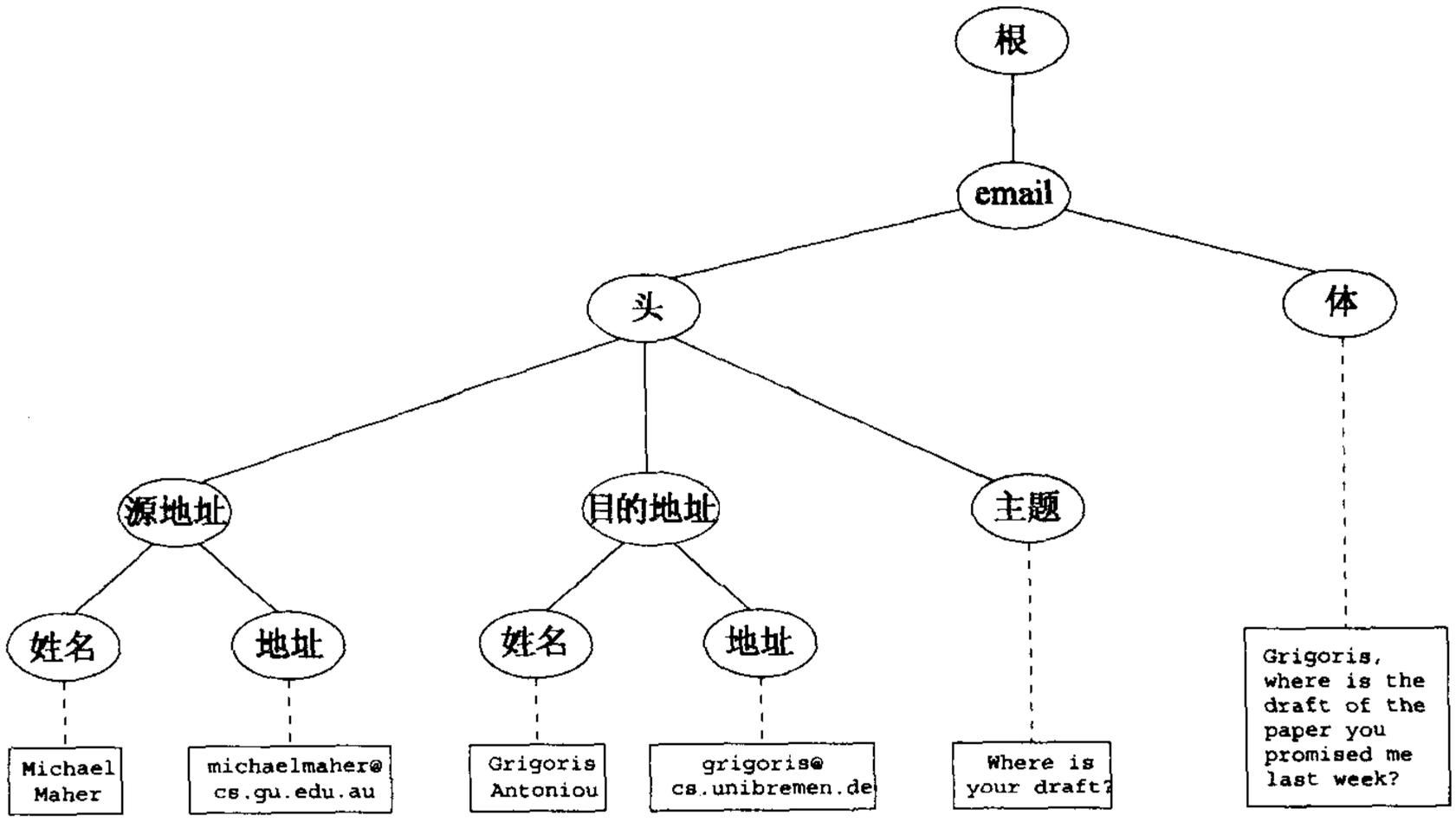


图 2-1 一个 XML 文档的树形表达

- 仅有一个根节点。
- 树中无环。
- 除了根节点之外，每个节点仅有一个父节点。
- 每个节点都有一个标记。
- 元素的次序是有意义的。

尽管元素的次序有意义，属性间的次序却是无关的。故而下面两个元素是等价的：

```
<person lastname="Woo" firstname="Jason"/>
<person first="Jason" lastname="Woo"/>
```

在树形表示中没有充分地反映出这一点。一般地说，需要一种更为细致的树形概念，比如要区分不同类型的节点(元素节点、属性节点等等)。这里仅用树形结构做图示说明，不再讨论更深入的细节。

在 email 元素的例子中，图 2-1 表现了根(代表 XML 文档)与根元素之间的区别。在 2.5 节讨论 XML 文档的寻址和查询时，这一区别将发挥作用。

2.3 结构化

当一个 XML 文档遵循一定语法规则时，称其为合式 XML 文档。但是，这些规则没有

对文档的具体结构做出任何规定。设想有两个应用希望通过使用同一词汇表来交流。为了实现这一目标，有必要定义所有可能用到的元素名和属性名。同时，还要定义文档的结构：一个属性可以使用什么样的值，哪些元素可以出现在其他元素中，等等。

30
|
31

有了这些结构信息，就可以提高文档有效的可能性。如果一个合式 XML 文档使用了结构化信息并且遵循这些结构化信息，则称之为*有效的*(valid)。

有两种定义 XML 文档结构的方式：DTD 方式和 XML Schema 方式。前一种方法较陈旧，限制较多；后一种方法提供了扩充的可能性，主要用于定义数据类型。

2.3.1 DTD 方式

外部 DTD 和内部 DTD

一个 DTD 的组成部分可以定义在另一个独立的文件(外部 DTD)中，也可以定义在该 XML 文档自身(内部 DTD)中。通常情况下，使用外部 DTD 较好，因为它的定义可以被几个不同文档使用；否则就会出现重复，而且一致性的维护也会变得困难。

元素

考察上一节例子中的元素：

```
<lecturer>
   <name>David Billington</name>
   <phone> +61-7-3875 507</phone>
</lecturer>
```

这个元素类型⊖的 DTD 形如：

```
<! ELEMENT lecturer (name, phone) >
<! ELEMENT name (#PCDATA) >
<! ELEMENT phone (#PCDATA) >
```

这个 DTD 的含义如下：

- 此文档可能会用到元素类型 lecturer(讲师)、name(姓名)和 phone(电话)；
- 一个 lecturer 元素会按顺序包含一个 name 元素和一个 phone 元素；

32

- name 元素和 phone 元素可以有任何内容。在 DTD 中，#PCDATA 是元素唯一的原子类型。

如果一个 lecturer 元素包含一个 name 元素，或者包含一个 phone 元素，那么，对应的 DTD 表示如下：

```
<! ELEMENT lecturer (name | phone) >
```

⊖ 应该明确区分元素类型 lecturer 和这个类型的一个实例(如 David Billington)。元素类型 lecturer 的所有实例在此定义下拥有同样的结构。

如果想把一个 lecturer 元素指定为以任意顺序包含一个 name 元素和一个 phone 元素，那么，难度就更大了，只能用下面这种办法：

```
<! ELEMENT lecturer ((name, phone) | (phone, name)) >
```

但是，这种办法实际操作上有局限性(设想 10 个元素以任意顺序的排列)。

属性

考察上节中的元素：

```
<order orderNo = "23456" customer = "John Smith"
       date = "October 15, 2002" >
   <item itemNo = "a528" quantity = "1"/ >
   <item itemNo = "c817" quantity = "3"/ >
</order >
```

此元素的 DTD 形如：

```
<! ELEMENT order (item + ) >
<! ATTLIST order
   orderNo    ID       #REQUIRED
   customer   CDATA    #REQUIRED
   date       CDATA    #REQUIRED >
<! ELEMENT item EMPTY >
<! ATTLIST item
   itemNo     ID       #REQUIRED
   quantity   CDATA    #REQUIRED
   comments   CDATA    #IMPLIED >
```

和上例相比，出现了两种新用法，一是 item(项目)元素类型定义为空，二是在 order(顺序)元素类型的定义中，item 后面出现了 + 。 + 是下列基数操作符(cardinality operators)中的一种，它们的含义是：

33

?: 出现 0 次或 1 次

*: 出现 0 次或多次

+: 出现 1 次或多次

若没有基数操作符，则表示“出现且仅出现 1 次”。

除了定义元素之外，还要在属性表(attribute list)中定义属性。属性定义的第一分量是此属性表所适用的元素类型名称，后跟一个三元组表，其中的三元组由属性名、属性类型和值类型构成。属性名是在使用 DTD 的 XML 文档中可能使用的名称。

属性类型

属性类型类似于预定义的数据类型，但可供选择的属性类型有限。最重要的一些属性类型为：

- CDATA，串类型(字符序列)
- ID，在整个 XML 文档中唯一确定的名字
- IDREF，指向另一个元素的索引，该元素与本 IDREF 属性拥有同样的 ID 属性值
- IDREFS，一系列 IDREF
- $(v_1 | \cdots | v_n)$，所有可能值的一个枚举

仅这些属性类型并不能满足应用的需要。比如，不能指定日期和数字，必须将它们译成字符串类型(CDATA)，因而无法规定它们的特定结构。

值类型

有四种值类型：

- #REQUIRED，此属性必须出现在 XML 文档中元素类型出现的任何地方。在前例中，itemNo(项目序号)和 quantity(数量)必须永远在 item 元素中出现。
- #IMPLIED，此属性的出现是可选的，如上例中的 comments(注释)。
- #FIXED "value"，每个元素必须具有这一属性，该属性的值永远等于在 DTD 中 #FIXED之后给定的值。在 XML 文档中的赋值是无意义的，因为它将被这个固定值覆盖。
- "value"，规定属性的默认值。如果 XML 文档规定了一个属性的特定值，那么它将覆盖该属性的默认值。例如，邮件系统的默认编码方式可以是"mime"，若用户显式指定的编码方式是"binhex"，则将采用"binhex"方式。

引用

下面是一个使用 IDREF 和 IDREFS 的例子。首先给出 DTD：

```
<! ELEMENT family (person* ) >
<! ELEMENT person (name) >
<! ELEMENT name (#PCDATA) >
<! ATTLIST person
   id          ID      #REQUIRED
   mother      IDREF   #IMPLIED
   father      IDREF   #IMPLIED
   children    IDREFS  #IMPLIED >
```

遵循此 DTD 结构的一个 XML 元素如下：

```
<family>

  <person id="bob" mother="mary" father="peter">
    <name>Bob Marley</name>
  </person>
```

```
<person id = "bridget" mother = "mary" >
  <name > Bridget Jones < /name >
< /person >

<person id = "mary" children = "bob bridget" >
  <name > Mary Poppins < /name >
< /person >

  <person id = "peter" children = "bob" >
    <name > Peter Marley < /name >
  < /person >

< /family >
```

35

请读者体会本例中人与人之间的引用关系。

综合性实例

2.2.7 节给出的 email 元素的 DTD 的最后一个例子如下：

```
< ! ELEMENT email (head, body) >
< ! ELEMENT head (from, to + , cc* , subject) >
< ! ELEMENT from EMPTY >
< ! ATTLIST from
    name        CDATA   #IMPLIED
    address     CDATA   #REQUIRED >
< ! ELEMENT to EMPTY >
< ! ATTLIST to
    name        CDATA   #IMPLIED
    address     CDATA   #REQUIRED >
< ! ELEMENT cc EMPTY >
< ! ATTLIST cc
    name        CDATA   #IMPLIED
    address     CDATA   #REQUIRED >
< ! ELEMENT subject (#PCDATA) >
< ! ELEMENT body (text, attachment* ) >
< ! ELEMENT text (#PCDATA) >
< ! ELEMENT text (#PCDATA) >
< ! ELEMENT attachment EMPTY >
< ! ATTLIST attachment
    encoding (mime | binhex) "mime"
    file        CDATA   #REQUIRED >
```

这个 DTD 的一些有趣之处如下：

- 一个 head(头)元素按顺序包含一个 from(源地址)元素、至少一个 to(目的地址)元素、0 或多个 cc(抄送地址)元素和一个 subject(主题)元素。

- from、to 和 cc 这些元素中的 name(姓名)属性是可选的；address(地址)属性是必须的。
- 一个 body(体)元素包含一个 text(文本)元素，在 text 元素后可能还有一些 attachment(附件)元素。
- attachment 元素中的 encoding(代码)属性必须取值为“mime”或“binhex”，且前者是默认值。

36

最后，再看看 DTD 另外两个值得注意的地方。首先，DTD 可以化成扩展的巴克斯-努尔范式(Extended Backus-Naur Form，EBNF)。比如，声明

```
<! ELEMENT email (head, body) >
```

与规则 email :: = head body 等价，它表示一个邮件按顺序包含头和体。第二，在 DTD 中允许递归定义。比如，

```
<! ELEMENT bintree ((bintree root bintree) | emptytree) >
```

定义了一棵二叉树，此树是棵空树，或者由一棵左子树、一个根节点和一棵右子树组成。

2.3.2 XML Schema

XML Schema 为定义 XML 文档结构提供了一种更为丰富的语言。它的特点之一是其语法基于 XML 自身。这种设计极大地提高了可读性，而且更重要的是，这使得技术的可重用性有了显著提高。对于 XML Schema，不必像 DTD 一样，为了支持单独的语法而编写独立的分析器、编辑器和精巧的打印机等等。一个尤其重要的改进是提供模式(schemas)重用和精化的可能性。XML Schema 允许用户通过扩展或限制已有的类型来定义新的类型。结合基于 XML 的语法，这个特性使用户可以在模式的基础上建立新的模式，以此减少工作量。最后，XML Schema 提供了一套成熟的可运用于 XML 文档的数据类型集(DTD 的数据类型仅限于字符串类型)。

一个 XML Schema 是一个含起始标签的元素，如：

```
<xsd: schema
  xmlns: xsd = "http: //www. w3. org/2000/10/XMLSchema"
37      version = "1.0" >
```

此元素使用 W3C 网站上 XML Schema 模式。这是建立其他新模式的基础。前缀 xsd 指定模式的命名空间(在下节将介绍命名空间)。若在 xmlns 属性中省去此前缀，则使用下列命名空间中元素作为默认值：

```
<schema
  xmlns = "http: //www. w3. org/2000/10/XMLSchema"
```

```
version = "1.0" >
```

后面将略去 xsd 前缀。

下面讨论 schema 元素。最重要的内容是用数据类型定义元素和属性的类型。

元素类型

元素类型的语法是：

```
<element name = ". . ."/ >
```

它可以有许多可选的属性，如类型

type =". . ."（后面将对类型做更多讨论）

或基数限制：

- minOccurs =“x”，其中 x 可为任意自然数(包括 0)
- maxOccurs =“x”，其中 x 可为任意自然数(包括 0)或者 unbounded

minOccurs 和 maxOccurs 是 DTD 提供的基数操作符?，* 和 + 功能上的推广。当没有显式地给定基数限制时，minOccurs 和 maxOccurs 的默认值是 1。

下面是几个例子：

```
<element name = "email"/ >
<element name = "head" minOccurs = "1" maxOccurs = "1"/ >
<element name = "to" minOccurs = "1"/ >
```

属性类型

属性类型的语法是：

```
<attribute name = ". . ."/ >
```

38

而且还可以有一些可选的属性，如类型，

type = ". . ."

或存在(对应于 DTD 中的#OPTIONAL 和#IMPLIED)：

use = "x"，x 可以是 optional 或 required。

或默认值(对应于 DTD 中的#FIXED 和默认值)：

use = "x" value = ". . .", x 可为 default 或 fixed

例如：

```
<attribute name = "id" type = "ID" use = "required"/ >
<element name = "speaks" type = "Language" use = "default"
    value = "en"/ >
```

数据类型

我们已经认识到，DTD 的一个重要缺陷是数据类型非常有限。XML Schema 具有定义数据类型的强大功能。首先，它有很多内嵌数据类型(built-in data types)，在此列出一些：

- 数字型数据类型，包括 integer，Short，Byte，Long，Float，Decimal
- 字串型数据类型，包括 string，ID，IDREF，CDATA，Language
- 日期和时间型数据类型，包括 time，Date，Month，Year

此外，还有用户定义数据类型(user-defined data types)，包括不可使用元素或属性的简单数据类型(simple data type)以及可以使用元素和属性的复杂数据类型(complex data type)。下面先讨论复杂数据类型，之后在介绍类型限制的时候再讨论简单数据类型。复杂数据类型的定义通过在已有数据类型上定义一些属性(如果有的话)或使用下述手段完成：

- sequence，已有数据类型元素的序列，按照预定义的次序出现
- all，必须出现的元素的集合，但出现的次序无关紧要；
- choice，元素的集合，其中之一将被选择。

39

例如：

```
<complexType name = "lecturerType" >
  <sequence >
    <element name = "firstname" type = "string"
      minOccurs = "0" maxOccurs = "unbounded"/ >
    <element name = "lastname" type = "string"/ >
  </sequence >
  <attribute name = "title" type = "string" use = "optional"/ >
</complexType >
```

此例表明，XML 文档中的类型声明为 lecturerType(讲师类型)的元素可以有一个 title(头衔)属性，还可以包含任意多个 firstname(姓)元素以及恰好一个 lastname(名)元素。

数据类型的扩展

可以用新的元素和属性来扩展已有数据类型。下面以扩展 lecturer 数据类型作为例子。

```
<complexType name = "extendedLecturerType" >
  <extension base = "lecturerType" >
    <sequence >
      <element name = "email" type = "string"
        minOccurs = "0" maxOccurs = "1"/ >
    </sequence >
    <attribute name = "rank" type = "string" use = "required"/ >
  </extension >
</complexType >
```

在本例中，lecturerType 类型扩展了一个 email 元素和一个 rank(级别)属性，形成的数据类型如下：

```
<complexType name = "extendedLecturerType" >
  <sequence >
    <element name = "firstname" type = "string"
      minOccurs = "0" maxOccurs = "unbounded"/ >
    <element name = "lastname" type = "string"/ >
    <element name = "email" type = "string"
      minOccurs = "0" maxOccurs = "1"/ >
  </sequence >
  <attribute name = "title" type = "string" use = "optional"/ >
  <attribute name = "rank" type = "string" use = "required"/ >
</complexType >
```

40

源类型和扩展类型之间有层次关系。扩展类型的实例也是源类型的实例。扩展类型可以包含比源类型更多的信息，但决不会包含比源类型少或者错误的类型信息。

数据类型的限制

可以通过对某些值的约束来限制已有的数据类型。例如，可以增加新的 type 和 use 属性，或者对 minOccurs 和 maxOccurs 加以收缩。

要注意，限制并非扩展的逆过程。限制不是通过删除元素或属性来实现的。因此，下面的层次关系依然成立：限制生成的新类型的实例也是源类型的实例。它们至少满足源类型的约束和一些新加的约束。

对 lecturer 数据类型作限制的一个例子如下：

```
<complexType name = "restrictedLecturerType" >
  <restriction base = "lecturerType" >
    <sequence >
      <element name = "firstname" type = "string"
        minOccurs = "1" maxOccurs = "2"/ >
    </sequence >
    <attribute name = "title" type = "string" use = "required"/ >
  </restriction >
</complexType >
```

所施加的约束用黑体字标出。读者可以把它同原例比较一下。

通过限制已有数据类型也可以定义简单数据类型。例如，可以如下定义一个取值 1 到 31 的类型——月中日期(dayOfMonth)：

```
<simpleType name = "dayOfMonth" >
  <restriction base = "integer" >
    <minInclusive value = "1"/ >
```

```
    <maxInclusive value="31"/>
  </restriction>
</simpleType>
```

还可以通过列出所有可能的值来定义新的数据类型。例如，可以定义一个数据类型——周中日期(dayOfWeek)如下：

```
<simpleType name="dayOfWeek">
  <restriction base="string">
    <enumeration value="Mon"/>
    <enumeration value="Tue"/>
    <enumeration value="Wed"/>
    <enumeration value="Thu"/>
    <enumeration value="Fri"/>
    <enumeration value="Sat"/>
    <enumeration value="Sun"/>
  </restriction>
</simpleType>
```

41

综合性实例

这里定义一个 e-Mail 的 XML Schema，以便与 2.3.1 节的综合性实例中的 DTD 作比较。

```
<element name="email" type="emailType"/>
<complexType name="emailType">
  <sequence>
    <element name="head" type="headType"/>
    <element name="body" type="bodyType"/>
  </sequence>
</complexType>

<complexType name="headType">
  <sequence>
    <element name="from" type="nameAddress"/>
    <element name="to" type="nameAddress"
     minOccurs="1" maxOccurs="unbounded"/>
    <element name="cc" type="nameAddress"
     minOccurs="0" maxOccurs="unbounded"/>
    <element name="subject" type="string"/>
  </sequence>
</complexType>

<complexType name="nameAddress">
  <attribute name="name" type="string" use="optional"/>
```

```
    <attribute name = "address" type = "string" use = "required"/ >
</complexType >                                                          42

<complexType name = "bodyType" >
  <sequence >
    <element name = "text" type = "string"/ >
    <element name = "attachment" minOccurs = "0"
        maxOccurs = "unbounded" >
      <complexType >
        <attribute name = "encoding" use = "default"
            value = "mime" >
          <simpleType >
            <restriction base = "string" >
              <enumeration value = "mime"/ >
              <enumeration value = "binhex"/ >
            </restriction >
          </simpleType >
        </attribute >
        <attribute name = "file" type = "string"
            use = "required"/ >
      </complexType >
    </element >
  </sequence >
</complexType >
```

注意，某些数据类型是单独定义的，并有名字，而其他的一些数据类型却是在别的数据类型中定义的且没有名字(比如 attachment 元素的类型和 encoding 属性的类型)。一般来说，如果一个类型只使用一次，可以在用它的地方匿名地定义。然而，当嵌套变得很深的时候，这种方法的局限性很快就会暴露出来。

2.4 命名空间

把 XML 用作一个通用元(meta)标记语言的主要优点之一是，可以获取来自不同来源的信息。从技术角度来说，一个 XML 文档可能会用到多个 DTD 或模式。但是，由于每个结构化文档是独立开发的，因此命名冲突(name clashes)在所难免。假设在 DTD A 和 DTD B 中用不同方式定义一个元素类型 *e*，一个分析器试图核查一个出现 *e* 元素的 XML 文档是不是有效的，那么就应该告诉它该用哪个 DTD 来做有效性核查。 43

解决办法在技术上是简单的：每个 DTD 或模式使用不同的前缀，以消除二义性。其中，用冒号分隔前缀和本地名：

前缀：名字

例如，设想由一所澳大利亚大学(想像为 Griffith 大学)和一所美国大学(假设是 Kentucky 大学)共同为在线学生建立统一视图。每个学校各用自己的术语表，而且两个表是不同的。例如，美国的讲师(lecturer)不是正式教师，在澳大利亚却是正式教师(实际上，相当于美国的助理教授)。以下的例子说明如何消除二义性。

```
<? xml version = "1.0" encoding = "UTF-16"? >
<vu: instructors
     xmlns: vu = "http: //www. vu. com/empDTD"
     xmlns: gu = "http: //www. gu. au/empDTD"
     xmlns: uky = "http: //www. uky. edu/empDTD" >
  <uky: faculty
    uky: title = "assistant professor"
    uky: name = "John Smith"
    uky: department = "Computer Science"/ >
  <gu: academicStaff
    gu: title = "lecturer"
    gu: name = "Mate Jones"
    gu: school = "Information Technology"/ >
</vu: instructors >
```

所以，命名空间的声明是在元素中作出的，并在此元素和它的任何孩子(元素或属性)中使用。命名空间的声明具有下列形式：

```
xmlns: 前缀 = "位置"
```

其中，位置是 DTD 或模式的地址。如果没有指定前缀，如在

```
xmlns = "位置"
```

44 中，那么就使用默认位置值。例如，上述例子等价于下列文档：

```
<? xml version = "1.0" encoding = "UTF-16"? >
<vu: instructors
     xmlns: vu = "http: //www. vu. com/empDTD"
     xmlns = "http: //www. gu. au/empDTD"
     xmlns: uky = "http: //www. uky. edu/empDTD" >
  <uky: faculty
    uky: title = "assistant professor"
    uky: name = "John Smith"
    uky: department = "Computer Science"/ >
  <academicStaff
    title = "lecturer"
    name = "Mate Jones"
    school = "Information Technology"/ >
</vu: instructors >
```

2.5 XML 文档的寻址和查询

在关系数据库中，可以用诸如 SQL 之类的查询语言来筛选和检索数据库。对于 XML 文档来说，也有一些查询语言，比如 XQL、XML-QL 和 XQuery。

XML 查询语言的核心概念是路径表达式(path expression)，它规定如何在 XML 文档的树形表示中到达一个节点或一个节点集。下面介绍 XPath 形式的路径表达式，这种形式还可用于查询之外的其他目的，如用来转换 XML 文档。

XPath 是一种对 XML 文档的组件(parts)进行寻址的语言，它对 XML 的树形数据模型进行操作，并具有一种非 XML 语法。关键概念是路径表达式。XPath 的路径表达式可为

- 绝对的(从树形模型的根开始)；语法表达上用指示文档根节点的符号/开头，位于文档根元素的上一层；
- 相对于背景(context)节点的。

考察如下 XML 文档：

```
<? xml version = "1.0" encoding = "UTF-16"? >
<! DOCTYPE library PUBLIC "library. dtd" >                    45
<library location = "Bremen" >
  <author name = "Henry Wise" >
    <book title = "Artificial Intelligence"/ >
    <book title = "Modern Web Services"/ >
    <book title = "Theory of Computation"/ >
  </author >
  <author name = "William Smart" >
    <book title = "Artificial Intelligence"/ >
  </author >
  <author name = "Cynthia Singleton" >
    <book title = "The Semantic Web"/ >
    <book title = "Browser Technology Revised"/ >
  </author >
</library >
```

此文档的树形表示见图 2-2。

下面举几个路径表达式的例子来说明 XPath 的功能。

1. 寻址所有 author(作者)元素。

```
/library/author
```

此路径表达式寻址所有这样的 author 元素：它们是直接位于根节点之下的 library (图书馆)元素节点的孩子。用序列/t_1/…/t_n(其中 t_{i+1} 是 t_i 的孩子节点)来定义树形表示中 46

的一条路径。

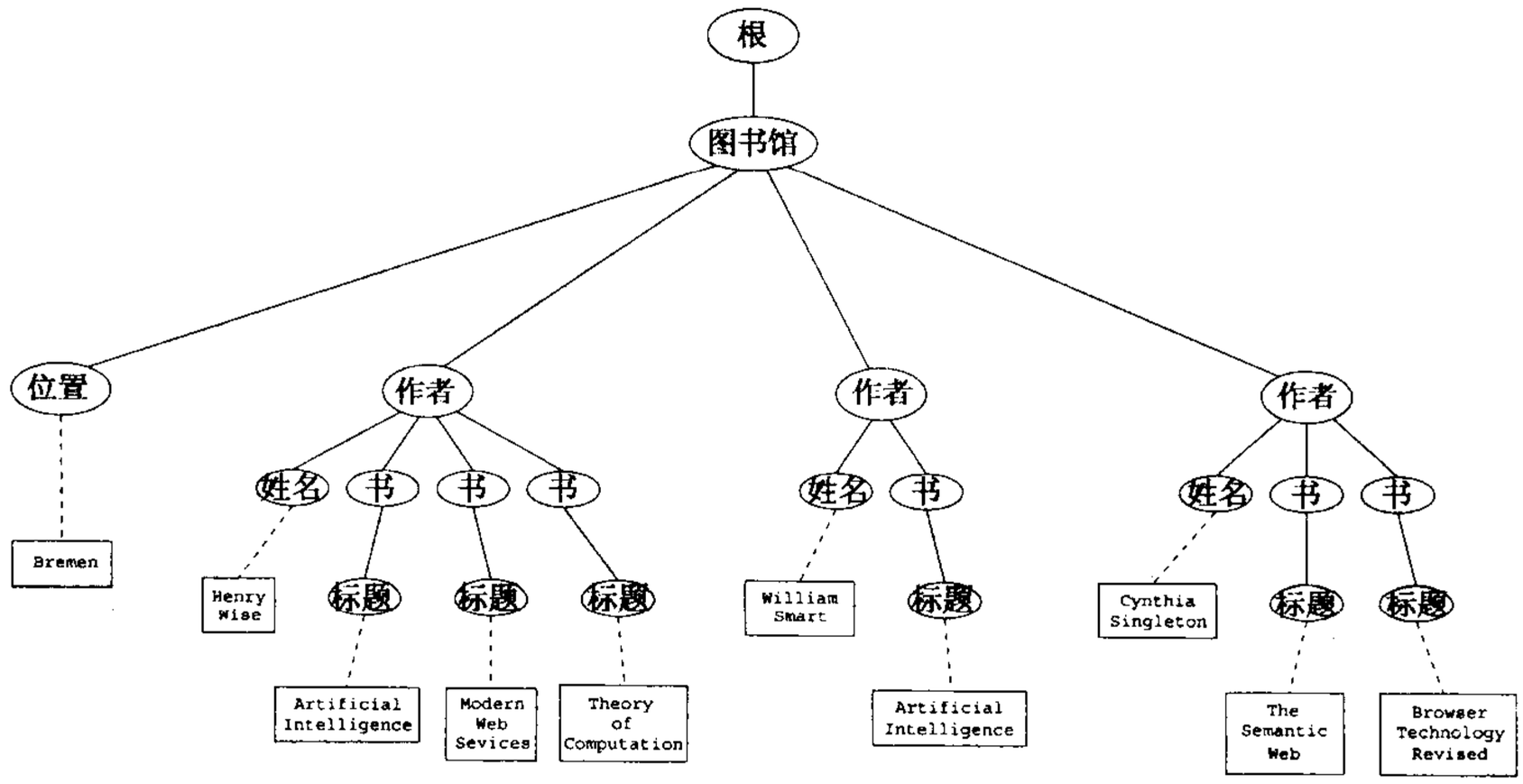


图 2-2 一个图书馆文档的树形表示

2. 上例的另一可选方法是：

```
//author
```

这里的//是指应该考查文档中的所有元素，检查它们是否属于 author 类型。换句话说，此路径表达式寻址文档中的所有 author 元素。因为这个 XML 文档的特殊结构，此表达式才与上一表达式产生相同的结果。一般情况下，用这两种方法进行查询的结果是不同的。

3. 在 library(图书馆)元素节点下寻址 location(位置)属性节点。

```
/library/@ location
```

符号@ 用于表示属性节点。

4. 在文档的任意 book(书)元素下寻址所有属性节点，其 title(标题)取值为“Artificial Intelligence”(见图 2-3)。

```
//book/@ title = "Artificial Intelligence"
```

5. 寻址所有 title 为“Artificial Intelligence”的书(见图 2-4)。

```
//book[@ title = "Artificial Intelligence"]
```

方括号中的检验称为过滤表达式(filter expression)。它限制被寻址的节点集。

注意此表达式和查询 4 中表达式的区别。此表达式寻址其 title 满足一定条件的 book 元素。查询 4 寻址 book 元素的 title 属性节点。比较图 2-3 与图 2-4 可知它们的区别。

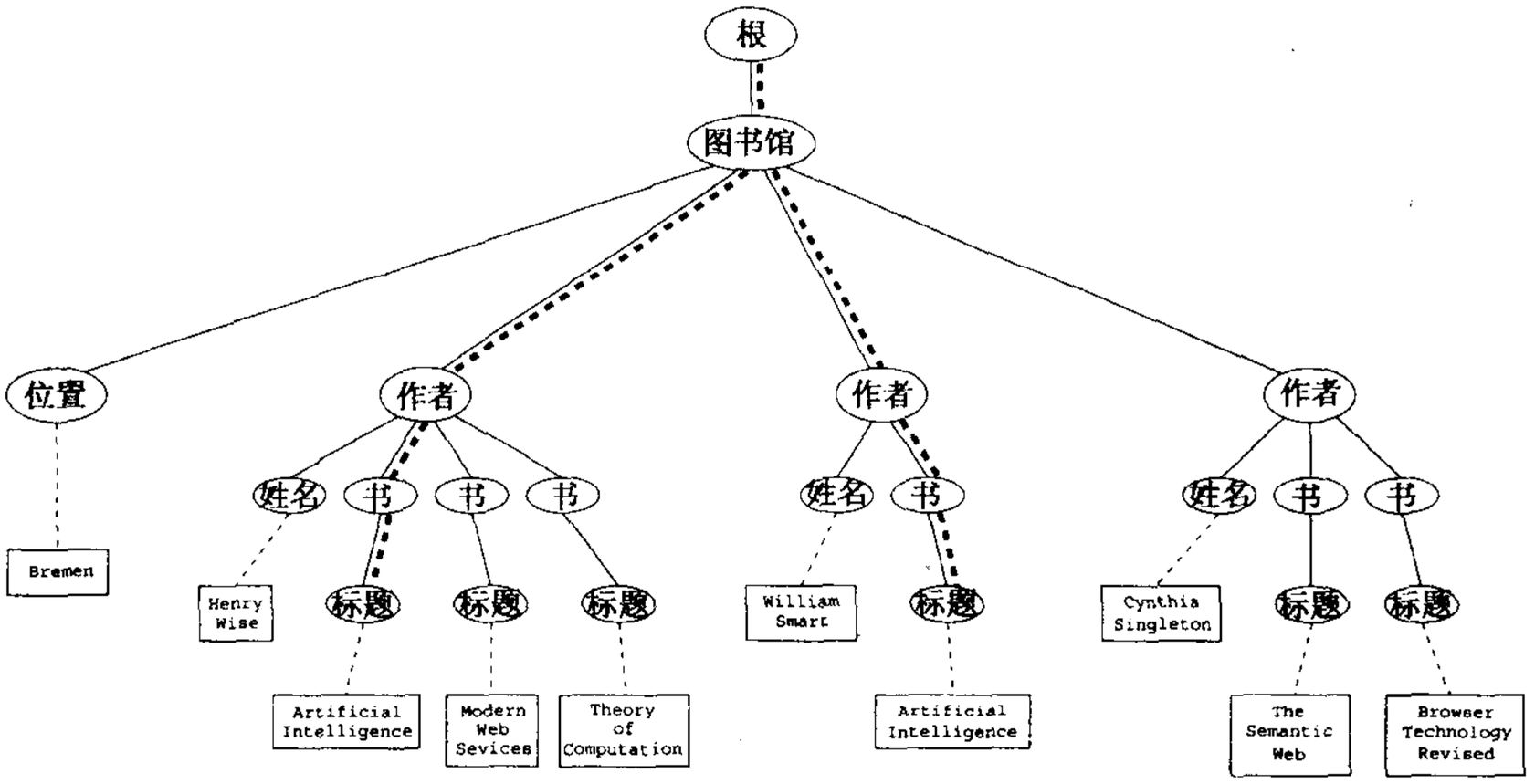


图 2-3　查询 4 的树形表示

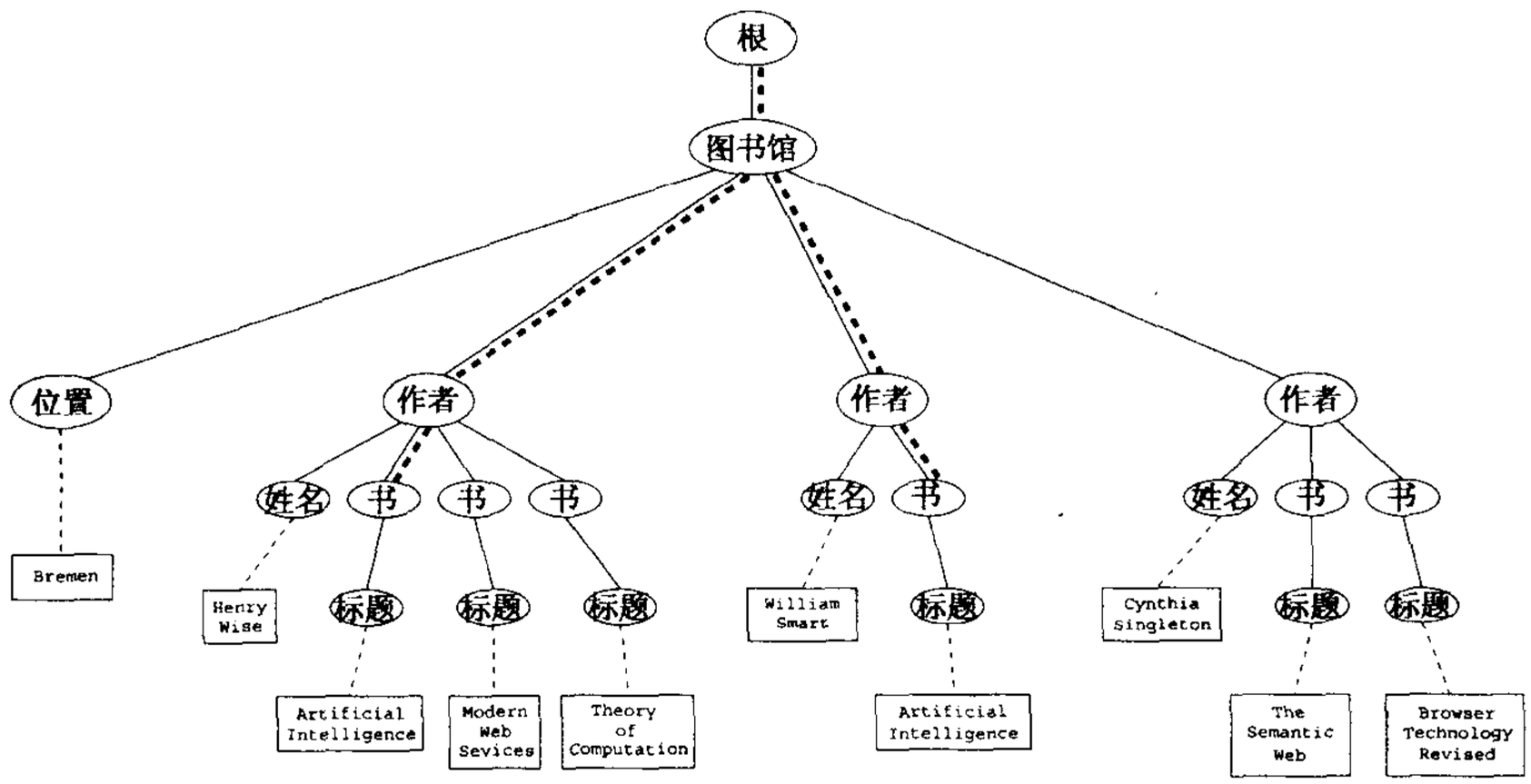


图 2-4　查询 5 的树形表示

6. 寻址 XML 文档中的第一个 author 元素节点。

47
|
48

```
//author[1]
```

7. 从文档中，在第一个 author 元素节点下寻址最后一个 book 元素。

```
//author[1]/book[last()]
```

8. 寻址没有 title 属性的所有 book 元素节点。

```
//book[not @ title]
```

列举这些例子是为了说明路径表达式的表达能力。总体上说，一个路径表达式由一系列被斜线符号分隔的步骤组成。一个步骤(step)由一个轴确定符、一个节点检验和一个可

选的谓词组成。

- 轴确定符(axis specifier)指定了待寻址的节点和背景节点的树形关系。比如父节点、祖先节点、孩子节点(默认值)、兄弟节点和属性节点。//就是轴确定符，表示子孙或它自己。
- 节点检验(node test)指定要寻址的节点。最常见的节点检验是元素名字(可使用命名空间信息)，也可有其他节点检验。比如，*寻址所有的元素节点，comment()寻址所有的注释节点，等等。
- 谓词(predicates)(或过滤表达式)是一个限定待寻址节点集合的可选项。比如，表达式[1]搜寻第一个节点，[position()=last()]搜寻最后一个节点，[position() mod 2 =0]搜寻其位置是偶数的节点，等等。

此处只列出了简略的语法，实际上 XPath 的语法更完全且更复杂，在本章结尾处有相关参考文献。

2.6 处理

到目前为止，还没有介绍关于 XML 文档显示的信息。这种信息是必要的，因为 XML 文档不像 HTML 文档那样含有格式信息。这种情况的好处在于，对于一个给定的 XML 文
49 档，应用不同的样式表(style sheets)能够以不同的方式显示该文档。比如，考虑 XML 元素

```
<author>
   <name>Grigoris Antoniou</name>
   <affiliation>University of Bremen</affiliation>
   <email>ga@ tzi.de</email>
</author>
```

使用一个样式表，此文档可能显示如下：

Grigoris Antoniou
University of Bremen
ga@ tzi. de

如果使用另一个不同的样式表，那么此文档可能显示为：

Grigoris Antoniou
University of Bremen
ga@tzi de

样式表可以用不同的语言书写，比如 CSS2，即 2 号标准层叠样式表(cascading style sheets level 2)。另一种可能是 XSL，即可扩展样式表语言(extensible stylesheet language)。

XSL 包含一个转换语言(XSLT)和一个格式语言。当然，这两种语言都是 XML 的应

用。XSLT 规定了一套将输入 XML 文档转换成另一个 XML 文档、HTML 文档或纯文本的规则。输出文档可以使用与输入文档相同的 DTD 或模式，或者使用完全不同的词汇表。

XSLT(XSL 的转换语言)可以独立于上述格式语言使用，它能将数据或元数据从一种 XML 表示转换为另一种 XML 表示，这使得它成为基于 XML 的应用中最有价值的工具。当使用不同的 DTD 或模式的应用要求彼此交流时，一般选择 XSLT。作为一种工具，XSLT 可用于内容的机器处理，这种处理不考虑方便人类阅读的信息显示。尽管如此，下面仅讨论用 XSLT 来显示 XML 文档。

定义 XML 文档显示的一种方法是将其转换为 HTML 文档。下例定义一个应用于作者例子的 XSLT 文档。 50

```
<? xml version = "1.0" encoding = "UTF-16"? >
<xsl: stylesheet version = "1.0"
    xmlns: xsl = "http: //www.w3.org/1999/XSL/Transform" >

  <xsl: template match = "/author" >
    <html >
      <head > <title > An author < /title > </head >
      <body bgcolor = "white" >
        <b > <xsl: value-of select = "name"/ > </b > <br >
        <xsl: value-of select = "affiliation"/ > <br >
        <i > <xsl: value-of select = "email"/ > </i >
      </body >
    </html >
  </xsl: template >
</xsl: stylesheet >
```

此样式表应用于前面的 XML 文档，将产生下面的 HTML 文档输出(此 HTML 文档定义了显示方式)：

```
<html >
   <head > <title > An author < /title > </head >
   <body bgcolor = "white" >
      <b > Grigoris Antoniou </b > <br >
      University of Bremen <br >
      <i > ga@ tzi.de </i >
   </body >
</html >
```

现在做一些观察。XSLT 文档是 XML 文档。故而 XSLT 是建立在 XML 之上的(就是说，它是一个 XML 应用)。XSLT 文档定义一个模板(template)。在本例中，定义的是一个带有一些占位符的 HTML 文档，使得内容可被插入(参见图 2-5)。

```
<html>
<head><title>An author</title></head>
<body bgcolor="white">
  <b>...</b><br>
  ...<br>
  <i>...</i>
</body>
</html>
```

图 2-5 一个模板

在上述 XSLT 文档中，xsl: value-of 检索元素的值并把它复制到输出文档中。也就是在模板中填入内容。

现在，假定有一个含有几个作者详细信息的 XML 文档。分别处理每个 author 元素显然会浪费力气。此种情况下，为 author 元素定义一个特殊模板，这个模板可以为主模板
51 所用，通过参考下列输入文档来阐述这种方法：

```
<authors>
  <author>
    <name>Grigoris Antoniou</name>
    <affiliation>University of Bremen</affiliation>
    <email>ga@ tzi.de</email>
  </author>
  <author>
    <name>David Billington</name>
    <affiliation>Griffith University</affiliation>
    <email>david@ gu.edu.net</email>
  </author>
</authors>
```

定义下列 XSLT 文档：

```
<? xml version="1.0" encoding="UTF-16"? >
<xsl: stylesheet version="1.0"
    xmlns: xsl="http: //www.w3.org/1999/XSL/Transform" >

  <xsl: template match="/" >
    <html>
    <head> <title>Authors< /title> </head>
    <body bgcolor="white" >
      <xsl: apply-templates select="authors"/ >
      <! -- Apply templates for AUTHORS children -- >
    </body>
```

```
</html>
</xsl: template>
<xsl: template match = "authors" >
   <xsl: apply-templates select = "author"/ >
</xsl: template >
<xsl: template match = "author" >
  <h2 > <xsl: value-of select = "name"/ > </h2 >
  Affiliation: <xsl: value-of select = "affiliation"/ > <br >
   Email: <xsl: value-of select = "email"/ >
   <p >
  </xsl: template >
</xsl: stylesheet >
```

产生的输出为：

```
<html >
<head > <title > Authors < /title > </head >
<body bgcolor = "white" >
   <h2 > Grigoris Antoniou </h2 >
   Affiliation: University of Bremen <br >
   Email: ga@ tzi. de
   <p >
   <p >
   <h2 > David Billington </h2 >
   Affiliation: Griffith University <br >
   Email: david@ gu. edu. net
   <p >
</body >
</html >
```

xsl: apply-templates 元素使得背景节点的所有孩子与搜寻的路径表达式相匹配。比如，若当前模板应用于/(即当前背景节点是根节点)，则 xsl: apply-templates 元素应用于根元素，在这个例子中就是 authors 元素(注意，/位于根元素之上)。若当前背景节点是 authors 元素，那么 xsl: apply-templates select = "author"元素使得 author 元素的模板应用于 authors 元素的所有 author 孩子。

为文档中的每个元素类型定义模板是个好习惯。即使有些元素用不着特别处理，如本例中的 authors，仍然应该使用 xsl: apply-templates 元素。这样，从树的根节点到叶节点，所有的模板实际上都用到了。

现在转到属性的处理。假设要用 XSLT 来处理元素

```
<person firstname = "John" lastname = "Woo"/ >
```

先尝试一下似乎最简单的做法，即把元素转换到它自身。用户可能会写：

```
<xsl: template match = "person" >
   <person
      firstname = " <xsl: value-of select = "@ firstname" > "
      lastname = " <xsl: value-of select = "@ lastname" > "/ >
</xsl: template >
```

然而，这不是一个合式 XML 文档，因为 XML 的语法不允许在属性值的内部使用标签。但用户的意图是清楚的，即希望将属性的值增加到模板中。在 XSLT 中，花括号中的数据占据 xsl: value-of 元素的位置。正确地定义此模板的方法如下：

```
<xsl: template match = "person" >
   <person
      firstname = "{@ firstname}"
      lastname = "{@ lastname}"/ >
</xsl: template >
```

最后，给出把一个 XML 文档转换成另一个 XML 文档的例子，其中不指定显示方式。再次使用 authors 文档作为输入，定义 XSLT 文档如下：

```
<? xml version = "1.0" encoding = "UTF-16"? >
<xsl: stylesheet version = "1.0"
      xmlns: xsl = "http: //www.w3.org/1999/XSL/Transform" >

<xsl: template match = "/" >
  <authors >
     <xsl: apply-templates select = "authors"/ >
  </authors >
</xsl: template >
<xsl: template match = "authors" >
  <xsl: apply-templates select = "author"/ >
</xsl: template >

<xsl: template match = "author" >
  </author >
    <name > <xsl: value-of select = "name"/ > </name >
    <contact >
      <institution >
        <xsl: value-of select = "affiliation"/ >
      </institution >
      <email > <xsl: value-of select = "email"/ > </email >
    </contact >
  </author >
</xsl: template >
```

```
</xsl: stylesheet>
```

输出文档应该显而易见。图 2-6 中的树形表示说明了 XSLT 的树形转换特性。

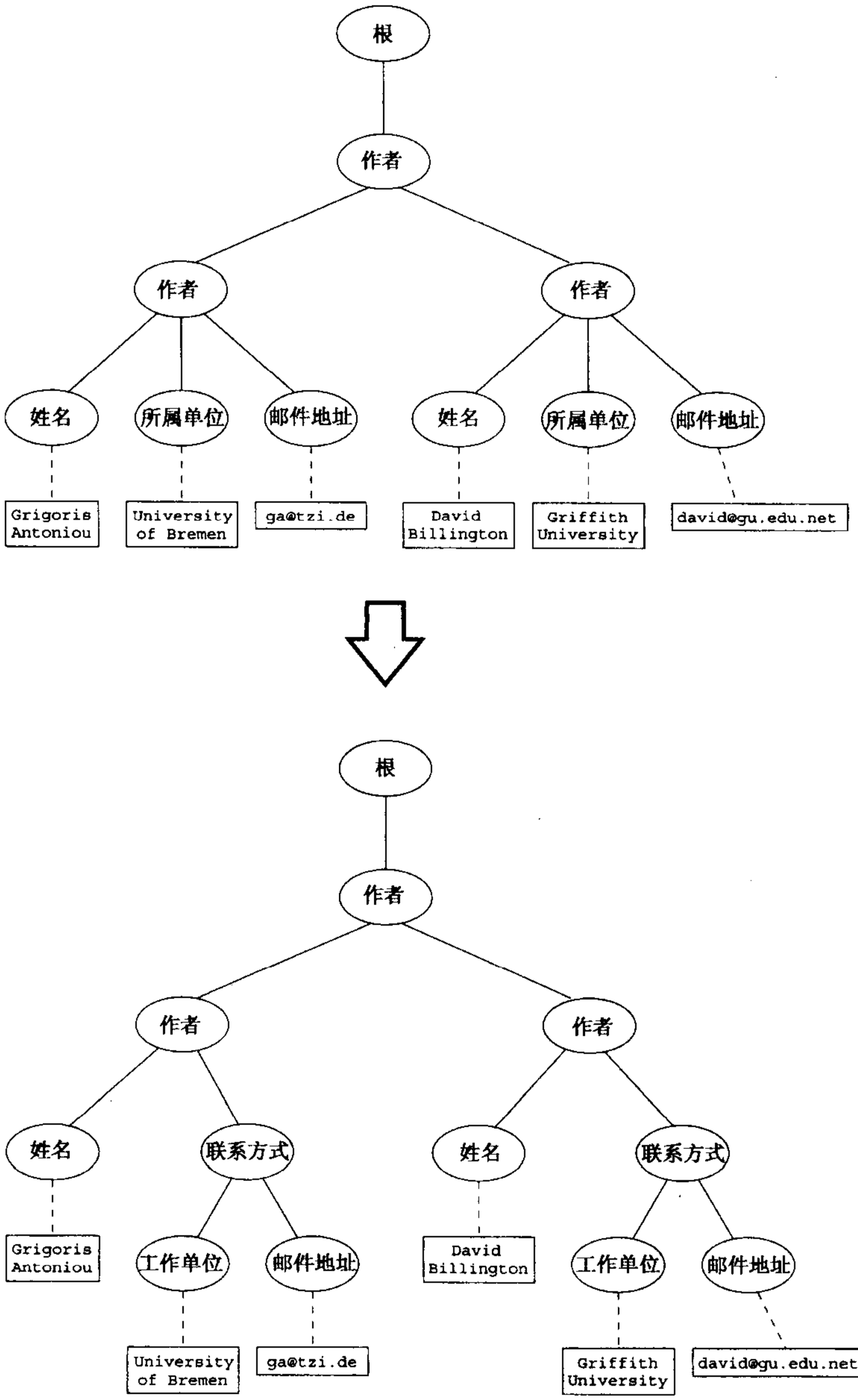


图 2-6　用做树形表示转换方式的 XSLT

2.7　小结

- XML 是一种元语言，允许用户借助标签为文档定义标记。

- 标签的嵌套引入了结构。可以用 XML schema 或 DTD 来规定文档的结构。
- XML 将内容和结构与格式相分离。
- XML 是网上结构化信息表示的默认标准，支持信息的机器处理。
- XML 借助标记、结构和转换支持不同应用之间的结构化信息的交换。
- 查询语言支持 XML。

55 | 56

以下几点将在后继章节中讨论：

- 标签嵌套没有标准含义。
- 机器不能解读 XML 文档的语义，只有人可以做到这点。
- 如果对词汇表存在共同理解，则 XML 将支持协作和交流。因此，在使用领域专用或团体专用词汇表的紧密合作场合，XML 的适用性很好，但对于全球性交流的适用性则没有这么好。

推荐读物

通常，W3C 官方文档可在 http：//www. w3. org 中找到。这里给出最重要的几个链接，外加一些有用的参考文献。

- T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, eds. Extensible Markup Language (XML) 1.0, 2nd ed, W3C Recommendation, October 6, 2000. <http://www.w3.org/TR/REC-xml>.
- T. Bray, D. Hollander, A. Layman, eds. Namespaces in XML, January 14, 1999. <http://www.w3.org/TR/REC-xml-names/>.
- J. Clark, S. DeRose, eds. XML Path Language (XPath) Version 1.0, W3C Recommendation, November 16, 1999. <http://www.w3.org/TR/xpath>.
- S. Adler et al. Extensible Stylesheet Language (XSL) Version 1.0, W3C Recommendation, October 15, 2001. <http://www.w3.org/TR/xsl/>.
- J. Clark, ed. XSL Transformations (XSLT) Version 1.0, W3C Recommendation, November 16, 1999. <http://www.w3.org/TR/xslt>.

XML 查询的前沿工作可在这里找到：

57

- <http://www.w3.org/XML/Query.html>.

XML 在产业界已引起了人们的大量关注，很多深入讨论技术性细节的书籍也已问世。进一步阅读 XML，可以选用以下两本书：

- E. R. Harold. *XML Bible*, 2nd ed. New York: Wiley (Hungry Minds), 2001.
- D. Mercer. *XML: A Beginner's Guide*. New York: McGraw Hill (Osborne), 2001.

一些关于 XML 的教材和相关技术的站点为：

- <http://www.xml.com>, 从这个站点中可以找到以下论文：
  - N. Walsh. A Technical Introduction to XML. October 3, 1998.
  - T. Bray. XML Namespaces by Example. January 19, 1999.
  - E. van der Vlist. Using W3C XML Schema. October 17, 2001.
  - G. Holman. What Is XSLT? (I): The Context of XSL Transformations and the XML Path Language. August 16, 2000.
- <http://www.w3schools.com>
- <http://www.topxml.com>
- <http://www.zvon.org>
- <http://www.xslt.com>

练习与设计

2.1 本书的 e-mail(邮件)例中定义了一个邮件的 body(体)，它给含一个 text(文本)元素和多个 attachment(附件)。试修改此模式，使其含有任意顺序的任意多个 text 和 attachment。

2.2 用“XML DTD”或“XML schema”等关键字上网搜索 XML 的应用。

2.3 阅读关于命名空间、XPath、XSL 和 XSLT 的 W3C 官方文档。找出本章未讨论的问题，尤其是 XPath 的通用记号和表达能力。用这些新知识写一些小型文档。

2.4 本章中未涉及网页的重要组成部分——链接。XLink 提供了比 HTML 链接更强大的链接功能。在 W3C 的官方网页上找出 XLink。注意，可以像如下这般创建简单的链接：

```
<mylink xmlns: xlink =http: //www. w3. org/1999/xlink"
   xlink: type ="simple" xlink: href ="target. html" >
Click here </mylink >
```

58

2.5 讨论 XLST 与定义网站视图(views)的关联(“视图”隐藏网站上的某些部分，只显示对特定用户有用的部分)。

2.6 比较使用 XML 标记文档和使用 TeX/LaTeX 标记文档间的区别；比较 XML 转换方式和 BibTeX 的区别。

下列设计作业要求“设计词汇表”，包括定义词汇表、编写相应的 DTD 或模式、书写 XML 文档样本，并把这些文档转换成 HTML 以及在 Web 浏览器中浏览。

2.7 设计一个(部分)模拟自己工作环境的词汇表。例如，假如你在大学工作，设计一个关于课程、授课教师、教室、出版物等等的词汇表。

2.8 针对自己的一项爱好，设计一个与爱好相同者交换信息时使用的词汇表。

2.9 如果你阅读某些类型的书籍，设计一个描述这类书籍的词汇表并与他人进行交流。

2.10 如果你是一名投资者，设计一个潜在投资选项及其属性的词汇表(比如风险、回报、投资者年龄和投资者性格)。

2.11 如果你爱好烹调，设计一个有关食物、口味和食谱的词汇表。

2.12 为上面每一个词汇表再写一个 XSL 样式表。这次不是把 XML 转换成 HTML，而是转换成另一种标记语言，比如 WML，一个支持 WAP 的移动电话所使用的标记语言。这种样式表应适用于在有限带宽和有限屏幕大小的小型移动设备上显示信息。你可以使用一种现成的 WAP 模拟器来显示结果。

59

第3章　用 RDF 描述网络资源

3.1　引言

XML 是一种定义标记的通用元语言。它向用户提供统一的框架及一系列工具软件，如分析器，以便在不同的应用之间交换数据和元数据。然而，XML 并没有提供任何表示数据语义(semantics)的手段。例如，XML 中的标签嵌套没有预定含义，完全由应用程序自行解释。我们用下例说明这一点。假定要表示下述事实：

David Billington 讲授离散数学课程。

使用 XML，这句话有多种表达方式，以下三种都是可能的：

```
<course name = "离散数学" >
   <lecturer > David Billington </lecturer >
</course >

<lecture name = "David Billington" >
   <teaches > 离散数学 </teaches >
</lecturer >

<teachingOffering >
   <lecturer > David Billlington </lecturer >
   <course > 离散数学 </course >
</teachingOffering >
```


注意，前两种表示尽管表达的是相同信息，但使用了相反的嵌套。所以，XML 没有为标签嵌套的含义制定任何标准。

虽然资源描述框架(Resource Description Framework，RDF)通常称为一种“语言”(本书也这样称呼)，但它实际上是一个数据模型(data-model)。它由一系列陈述(statement)即“对象 - 属性 - 值”三元组(object-attribute-value triple)组成。上述关于 Billington 的表示就是一个陈述。当然，为了刻划和传输一个抽象数据模型，需要一套具体的语法。已经用 XML 给出了 RDF 的一个语法，RDF 因此继承了 XML 的优点。然而，很重要的一点是，不采用 XML 语法而用其他语法刻画 RDF 也是可能的。基于 XML 的语法并不是 RDF 模型的一个必要组成部分。

RDF 是领域独立的，即没有假定 RDF 的特定应用领域。用户可以使用称为 RDF Schema(RDFS)的模式语言定义自己的术语表。目前普遍认为 RDF Schema 这一命名是一个错误的选择。从 RDF Schema 这个名字看，它与 RDF 的关系类似于 XML Schema 与 XML 的关系，但实际上并非如此。XML Schema 规定 XML 文档的结构，而 RDF Schema 则定义 RDF 数据模型所使用的词汇(vocabulary)。RDFS 可以定义词汇，规定什么属性可以作用于什么类型的对象，属性可以取什么值，也可以描述对象之间的关系。例如，我们可以写

授课者是教师的一个子类。

这个句子的意思是，所有授课者都是教师。“是……的一个子类”是有预定义含义的，理解这一点非常重要。应用程序不能随意解释这个词语，它的预定义含义必须被所有 RDF 处理程序共同遵守。通过对特定的语法符号赋予语义，可以用 RDF/RDFS 对特定的领域建模。

下面用一个例子来说明 RDF Schema 的重要性。考虑下列 XML 元素：

```
<academicStaffMember>Grigoris Antoniou</academicStaffMember>
<professor>Michael Maher</professor>
<course name="离散数学">
   <isTaughtBy>David Billington</isTaughtBy>
</course>
```

62

假定要找出所有教师。一个可能的 XPath 路径表达式是：

```
//academicStaffMember
```

其结果只有 Grigoris Antoniou。虽然从 XML 角度看这是正确的，但这个结果在语义上并不令人满意，因为人们都会认为 Michael Maher 和 David Billington 也应包含在结果中，因为：

- 所有教授都是教师(即 `professor` 是 `academic StaffMember` 的一个子类)。
- 课程只能由教师讲授。

这种信息利用特定领域的语义模型(semantic model)，而不能用 XML 或 RDF 来表达，但却是一种可以用 RDFS 描述的典型知识。因此，从语义网观点看来，RDFS 使机器可以解读语义信息。

3.2 节和 3.3 节讨论 RDF，包括 RDF 的基本思想和基于 XML 的 RDF 语法。3.4 节和 3.5 节分别介绍 RDF Schema 的基本概念和语言。

3.6 节介绍 RDF 和 RDFS 命名空间的一些元素的定义。3.7 节给出 RDF 和 RDFS 的公理语义，这些语义使用谓词逻辑描述，将 RDF/RDFS 建模原语的直观含义形式化。

3.8 节给出一个基于推理规则的直接语义(direct semantics)，3.9 节则介绍用资源查询语言 RQL 对 RDF/RDFS 文档进行查询。

3.2 RDF 的基本思想

RDF 的基本概念是资源、属性和陈述。

3.2.1 资源

可以将资源视为一个对象，也就是打算谈论的“事物”。资源可以是作者、书籍、出版社、地点、人、旅馆、房间或查询请求等等。每个资源都有一个通用资源标识符 URI(Universal Resource Identifier。)URI 可以是 URL(统一资源定位符，或称为网络地址)，也可以是某个其他形式的唯一标识符。注意，标识符并不一定要直接达到(access to)资源。URI 系统 63
不仅可表示网络地址，也可以标记许多其他对象，如电话号码、ISBN 码或地理位置。关于 URI 的本质可以进行一番长篇大论，甚至涉及到哲学问题(如怎样适当地标识一个人)，但这里不详细讨论了。一般地，我们假定 URI 是一个网络资源的标识符。

3.2.2 属性

属性是一类特殊的资源，描述资源之间的关系，如“由…编写”、“年龄”、“头衔”等等。RDF 中的属性仍然用 URI(或者实际中的 URL)标识。用 URI 标识“事物”及其关系的想法非常重要，因为这使用户有了一个全球通用的唯一命名系统，从而极大地减轻了此前一直困扰着分布式数据表示的同名问题。

3.2.3 陈述

陈述(statement)用于描述资源所具有的属性。一个陈述是一个“对象-属性-值”三元组，由一个资源、一个属性和一个值组成。值可以是资源，也可以是文字(literal)。文字是原子值(字符串)，它的结构本书不再详细讨论。

3.2.4 关于陈述的三种观点

陈述的一个例子如下：

David Billington 是网页 http：//www. cit. gu. edu. au/ ~db 的拥有者。

解释该陈述的最简单方法是使用定义，比如下面的三元组：

(http：//www. cit. gu. edu. au/ ~db, http：//www. mydomain. org/site-owner, #DavidBillington).

一般地，可以把三元组(x，P，y)看作一个逻辑公式 $P(x, y)$，其中二元谓词[⊖] P 将对象 x 和对象 y 关联起来。实际上，RDF 只提供二元谓词(属性)。注意上例中的属性“site-

⊖ 一个二元谓词代表一个二元关系，比如“x 是 y 的老师”。——译者注

owner”和两个对象都使用 URL 加以标识，而另一个对象简单地用一个字符串表示。

64 解释陈述的第二种观点是基于图的。图 3-1 给出了与上文陈述对应的图形表示，它是顶点和弧都带标记的有向图，弧由资源(陈述的主语)指向值(陈述的宾语)。这种图在人工智能中被称为语义网络(semantic net)。

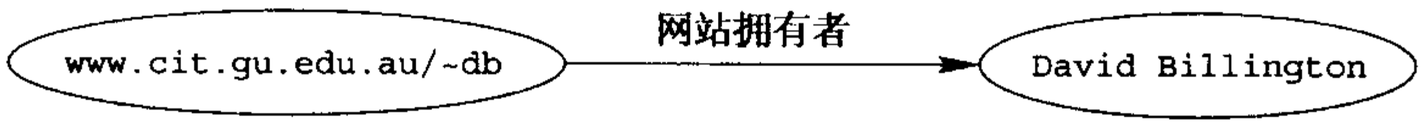


图 3-1 三元组的图表示

上文已提到，陈述的值可以是资源，因此，它也可以指向其他资源。考虑下面的三元组：

(http：//www. cit. gu. edu. au/_ db， http：//www. mydomain. org/siteowner， # David Billington)

(#David Billington，http：//www. mydomain. org/phone，“3875507”)

(#David Billington， http：//www. mydomain. org/uses，

http：//www. cit. gu. edu. au/_arock/defeasible/Defeasible. cgi)

(www. cit. gu. edu. au/_ arock/defeasible/Defeasible. cgi， http：//www. mydomain. org/site-owner，“Andrew Rock”)

它的有向图表示如图 3-2 所示。

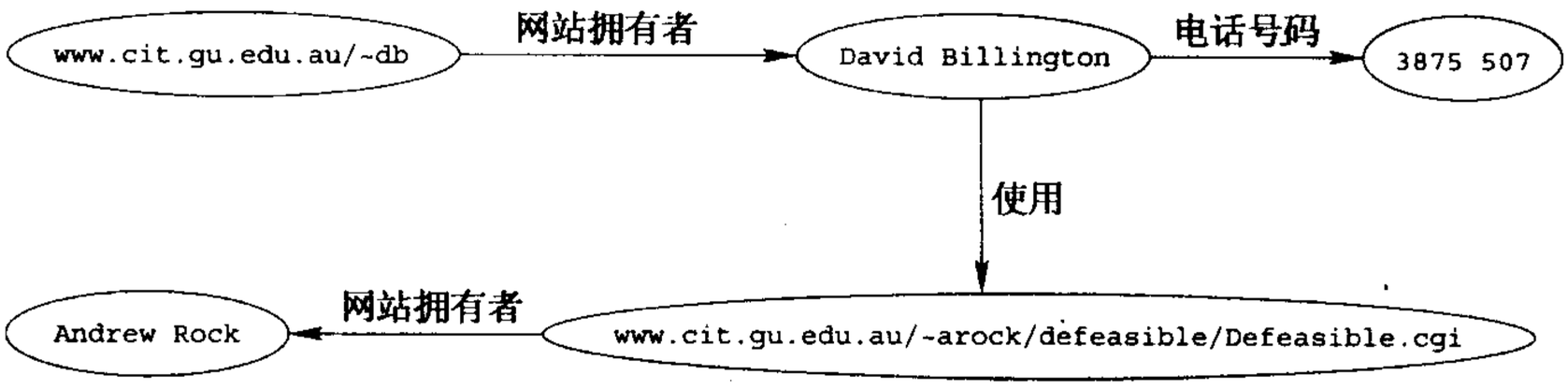


图 3-2 语义网络

65 图是人们理解事物的有用工具，但语义网要求机器可解读和机器可处理的表示。

因此，就有了第三种可能的表示法，它基于 XML。根据这个可能性，一个 RDF 文档表示为一个带有标签 rdf：RDF 的 XML 元素，该元素的内容是一系列描述(description)，这些描述使用标签 rdf：Description。每个描述都对资源给出一条陈述，资源的标识可采用下列三种方式之一：

- about 属性，用于引用已经存在的资源
- ID 属性，用于创建新的资源
- 无名，用于创建匿名资源

3.3 节将讨论 RDF 的基于 XML 的语法，这里只给出上述第一个陈述的 RDF 表示：

```
<? xml version = "1.0" encoding = "UTF-16"? >
<rdf: RDF
  xmlns: rdf = "http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
  xmlns: mydomain = "http: //www. mydomain. org/my-rdf-ns" >

 <rdf: Description rdf: about = "http: //www. cit. gu. edu. au/ ~db" >
  <mydomain: 网站拥有者 >
    David Billington
  </mydomain: 网站拥有者 >
 </rdf: Description >

</rdf: RDF >
```

第一行表明使用 XML。在下面的例子中都省略这一行，但要记住，在任何基于 XML 语法的 RDF 文档中，这一行都是需要的。

rdf: Description 元素对资源 http: //www. cit. gu. edu. au/ ~db 做出了陈述。在这个描述中，属性是一个标签，而它的内容是属性的值。

多个描述肯定是以一定的顺序出现，也就是说，XML 语法强制了描述之间的串行化(serialization)。然而，根据 RDF 抽象数据模型，描述(或资源)的顺序是无关紧要的。这再次表明，图模型才是真正的 RDF 数据模型，而 XML 只是图模型的一种可能的串行化表示而已。 [66]

3.2.5 陈述的具体化

RDF 允许对陈述的陈述，例如：

Grigoris 相信 David Billington 是网页 http：//www. cit. gu. edu. au/ ~db 的创建者。

一个这样的陈述可用于描述对另一个陈述的看法，比如是否相信另一个陈述是真的，这在某些应用中是重要的。为此，对一个陈述赋予唯一的标识符，用以指代该陈述。RDF 通过一种具体化机制(reification mechanism)达到这个目的(见 3.3.6 节)。

关键想法是引入一个辅助对象，比如 *belief*1，并通过主语(subject)、谓语(predicate)和宾语(object)三个属性把它与原陈述的三个部分联系起来。上例中，对象 *belief*1 的主语是 *David Billington*，谓语是创建者，宾语是 *http：//www. cit. gu. edu. au/ ~db*。采用这种复杂的办法是迫不得已的，因为 RDF 只允许三元组，故无法直接给一个三元组增加一个标识符(那就成了四元组)。

3.2.6 数据类型

考虑电话号码“3875507”，程序读到这个 RDF 数据模型时，并不知道文字“3875507”应该解释为一个整数还是一个字符串，如果它确实是一个整数(因而可以应用整数运算，比如

被17整除），那么究竟是用十进制还是八进制表示。仅当显式地获得该文字是否表示数字以及表示的是哪个数字的信息时，程序才知道如何解释这个资源。在程序设计语言或数据库系统中，通常给文字赋予一个数据类型来提供这类信息，在这种情况下，数据类型就类似于上述的十进制数或者整数。RDF借助类型化文字（typed literal）提供这类信息。

利用类型化文字，可以用下面的三元组描述David Billington的年龄是27岁，27是一个整数：

（“David Billington”，http：//www. mydomain. org/age，
“27”^^http：//www. w3. org/2001/XMLSchema#integer ）

这个例子主要说明两件事：使用“^^”标记来标明文字的类型⊖，使用XML Schema预定义的数据类型。严格地说，在RDF文档中允许使用任何外部定义的数据类型模式，但在实际应用中，最广泛使用的数据类型模式是用XML Schema定义的。它预定义了大量数据类型，包括布尔值、整数、浮点数、时间和日期。

67

3.2.7 对RDF的评论

上文已经指出，RDF只使用二元属性。这个限制看起来非常苛刻，因为人们经常使用二元以上的谓词。幸运的是，这些谓词可以使用几个二元谓词来模拟。下面用三元谓词*referee*来说明这个技巧。*referee*(*X*，*Y*，*Z*)的直观含义是：

*X*是棋手*Y*和*Z*的裁判。

现在引入新的辅助资源*chessGame*和二元谓词*ref*，*player*1，*player*2。于是可以如下表示*referee*(*X*，*Y*，*Z*)：

ref(*chessGame*，*X*)
*player*1(*chessGame*，*Y*)
*player*2(*chessGame*，*Z*)

它的图表示如图3-3所示。虽然这种解决方法是可靠的，但原先使用的三元谓词表示却更加简单自然。

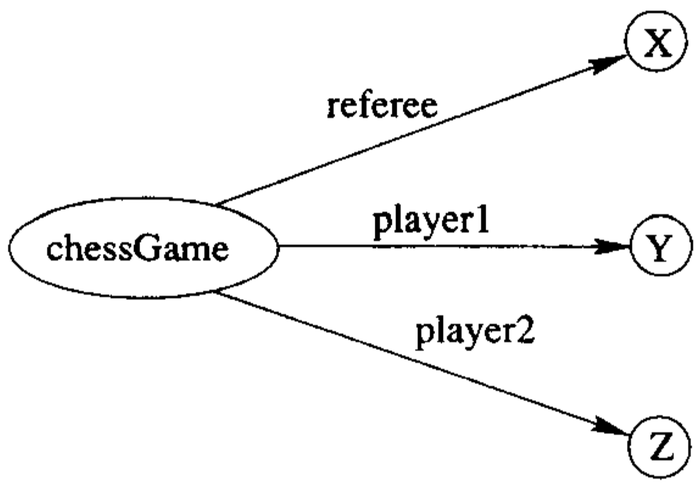


图3-3 三元谓词的表示

68

⊖ 3.3节描述的XML型语法中将采用这个记号的另一种形式。

RDF 的另一个问题与属性处理有关。上文已经提到，属性是一类特殊的资源。因此，属性本身也可以在陈述的对象-属性-值三元组中以对象的形式出现。虽然这种表示提供了灵活性，但对建模语言来说却非常少见，也会给用户带来混乱。

此外，具体化机制是如此强大，以至于似乎不应该将它放在像 RDF 这样简单的语言中。对陈述做陈述会使复杂性上升一级，对于语义网中如此基础的一层这是不必要的；相反，在需要提供更强表达能力的较高层次中引入该机制看起来会更加自然。

最后，基于 XML 语法的 RDF 虽然很利于机器处理，但是对用户而言却不够友好。

总而言之，RDF 具有很多独特之处，但还不是一个最优的建模语言。然而，我们不得不承认，它实际上已经逐渐成为了一个标准。在技术发展的历史上，最好的技术往往得不到采用。比如，视频系统 VHS 在技术上可能是市场上同期出现的三种电视系统中最弱的（另两个是 Beta 和 Video 2000），但是它却被最终采用；更不用说个人计算机领域的诸多软件或硬件标准了，它们中很多在技术上有很多优点，但却最终没有被采用，因而总是引起很多争议。

从正面来看，RDF 确实具有足够的表达能力（至少可以作为更高层之下的基础层），而且最终的语义网也不会直接用 RDF 来编程，而是用一些界面友好的工具来实现，它们会自动地把更高层表示转化为 RDF。使用 RDF 的好处是，信息可以无歧义地映射到一个模型中。既然 RDF 将会成为一个标准，在现阶段用 RDF 编写数据模型也可看作类似于在网络早期用 HTML 编写网页。

3.3 基于 XML 的 RDF 语法

一个 RDF 文档由一个 rdf: RDF 元素构成，其内容是一系列描述。例如，Griffith 大学 2001 年度的课程和授课者的领域知识可描述如下：

```
<! DOCTYPE owl [
    <! ENTITY xsd "http: //www. w3. org/2001/XMLSchema#" >
] >
<rdf: RDF
  xmlns: rdf = "http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
  xmlns: xsd = "http: //www. w3. org/2001/XMLSchema#"
  xmlns: uni = "http: //www. mydomain. org/uni-ns#" >
<rdf: Description rdf: about = "949352" >
  <uni: name > Grigoris Antoniou </uni: name >
  <uni: title > 教授 </uni: title >
</rdf: Description >

<rdf: Description rdf: about = "949318" >
```

```
  <uni: name >David Billington </uni: name >
  <uni: title >副教授 </uni: title >
  <uni: age rdf: datatype ="&xsd; integer" >27 </uni: age >
</rdf: Description >

<rdf: Description rdf: about ="949111" >
  <uni: name >Michael Maher </uni: name >
  <uni: title >教授 </uni: title >
</rdf: Description >

<rdf: Description rdf: about ="CIT1111" >
  <uni: courseName >离散数学 </uni: courseName >
  <uni: isTaughtBy >David Billington </uni: isTaughtBy >
</rdf: Description >

<rdf: Description rdf: about ="CIT1112" >
  <uni: courseName >具体数学 </uni: courseName >
  <uni: isTaughtBy >Grigoris Antoniou </uni: isTaughtBy >
</rdf: Description >

<rdf: Description rdf: about ="CIT2112" >
  <uni: courseName >程序设计Ⅲ </uni: courseName >
  <uni: isTaughtBy >Michael Maher </uni: isTaughtBy >
</rdf: Description >

<rdf: Description rdf: about ="CIT3112" >
  <uni: courseName >计算理论 </uni: courseName >
  <uni: isTaughtBy >David Billington </uni: isTaughtBy >
</rdf: Description >

<rdf: Description rdf: about ="CIT3116" >
  <uni: courseName >知识表示 </uni: courseName >
  <uni: isTaughtBy >Grigoris Antoniou </uni: isTaughtBy >
</rdf: Description >
</rdf: RDF >
```

下面来简要分析一下。首先，使用扩展的 XML 命名空间机制。在 XML 中，命名空间只用于避免二义性。而在 RDF 中，预期的外部命名空间是定义资源的 RDF 文档，于是命名空间也可以用于 RDF 文档的导入。这一机制允许资源被众多用户所复用，其他用户可以在这些资源中加入一些额外的特征，从而产生巨大的分布式知识聚积。

其次，严格地说，rdf: Description 元素中的 rdf: about 属性等价于一个 ID 属性，但它通常用于暗示：这个被某陈述断定的对象已在别处被“定义”了。从形式化的角度

看，一组 RDF 陈述形成一张大的图，资源之间通过属性相连，没有资源在一处定义而在另一处引用。然而在串行化的 XML 语法中，建议把资源的一处出现作为“定义”位置，而在其他位置则陈述它的“额外”属性，有时这很有用(只要能提高相对于人类读者的可读性)。

实际上，前面的例子有点容易产生误解。如果要写得绝对正确，需要把每个课程和员工的 ID，如 949352 和 CIT3112 都替换成对外部名字空间的指称，例如，

```
<rdf: Description
  rdf: about = "http: //www. mydomain. org/uni-ns/#CIT3122" >
```

前例中之所以没有这么写，是因为当时的内容基本着眼点是 RDF 的基本思想，然而读者应该意识到，这样书写 RDF 文档是不够精确的。

rdf: Descritption 元素的内容称为*属性元素*(property element)，如下面描述中，

```
<rdf: Description rdf: about = "CIT3116" >
  <uni: courseName >知识表示 </uni: courseName >
  <uni: isTaughtBy >Grigoris Antoniou </uni: isTaughtBy >
</rdf: Description >
```

71

两个元素 uni: courseName 和 uni: isTaughtBy 都定义为 CIT3116 的属性-值二元组的取值，因此，上例对应了两个 RDF 陈述。

其三，属性 rdf: datatype =“&xsd; interger”用来表示年龄属性的数据类型。即使年龄属性的值域已经定义为“&xsd; integer”，仍然需要在每次使用该属性时标明值的数据类型。这是为了确保 RDF 处理器在不参考对应 RDFS 定义的情况(这种情况在不受任何约束的万维网中是非常常见的)下，仍然可以为属性的值赋予正确的类型。

最后，一个描述的多个属性元素之间的关系必须理解为合取关系。比如在上面的例子中，课程的名称为“知识表示”，并且由 Grigoris Antoniou 讲授。

3.3.1 rdf: resource 属性

前面的例子在一个方面不令人满意：课程和授课者之间的关系没有得到严格定义，只是借助相同的名字而隐含地存在。对于机器来说，使用相同名字的情况可能只是一种巧合，例如，讲授课程 CIT3112 的 David Billington 可能和 ID 为 949318 的那个 David Billington 不是一个人。因此，需要一个形式化的规定，比如，规定讲授课程 CIT1111 的教师就是 ID 为 949318 的那个教师，他的名字就叫 David Billington。使用 rdf: resource 属性可以达到这个效果：

```
<rdf: Description rdf: about = "CIT1111" >
  <uni: courseName >离散数学 </uni: courseName >
  <uni: isTaughtBy rdf: resource = "949318"/ >
```

```
</rdf: Description>

<rdf: Description rdf: about="949318">
  <uni: name>David Billington</uni: name>
  <uni: title>副教授</uni: title>
</rdf: Description>
```

需要注意的是，如果已经在 RDF 文档中使用 ID 属性而非 about 属性来定义 ID 号为 72 939318 教师的这个资源，那么必须在 rdf: resouce 属性的值 949318 前面加上#号：

```
<rdf: Description rdf: about="CIT1111">
   <uni: courseName>离散数学</uni: courseName>
   <uni: isTaughtBy rdf: resource="#949318"/>
</rdf: Description>

<rdf: Description rdf: ID="#949318">
   <uni: name>David Billington</uni: name>
   <uni: title>副教授</uni: title>
</rdf: Description>
```

对于外部定义的资源，这个规定也同样有效。例如，引用外部定义的资源 CIT1111 时，就把

```
http: //www.mydomain.org/uni-ns/#CIT1111
```

作为 rdf: about 的值，其中 www.mydomain.org/uni-ns/则是定义 CIT1111 的 URI。换句话说，使用 ID 属性的描述定义了 URI 段，使用该 URI 段可以引用该描述。

3.3.2 嵌套描述

可以在其他描述的内部定义描述。例如，可以把前例中的描述替换成下述嵌套描述。

```
<rdf: Description rdf: about="CIT1111">
  <uni: courseName>离散数学</uni: courseName>
  <uni: isTaughtBy>
    <rdf: Description rdf: about="949318">
      <uni: name>David Billington</uni: name>
      <uni: title>副教授</uni: title>
    </rdf: Description>
  </uni: isTaughtBy>
</rdf: Description>
```

其他课程，如 CIT3112，仍然可以和新资源 949318 发生联系。换句话说，虽然一个描述可以在其他描述内部定义，但它还是全局有效的。

3.3.3 rdf: type 元素

在上述例子中，所有描述都针对两类不同资源：课程和授课教师。这个事实对于读者来说是不言而喻的，但由于没有明确声明而使得机器无法解读。在 RDF 中，可以使用 [73] rdf: type 元素作出这样的陈述。下面是两个包含类型信息的描述。

```
<rdf: Description rdf: about = "CIT1111" >
<rdf: type rdf: resource = "&uni; course"/ >
   <uni: courseName > 离散数学 </uni: courseName >
   <uni: isTaughtBy rdf: resource = "949318"/ >
</rdf: Description >

<rdf: Description rdf: about = "949318" >
  <rdf: type rdf: resource = "&uni; 授课者"/ >
  <uni: name > David Billington </uni: name >
  <uni: title > 副教授 </uni: title >
</rdf: Description >
```

注意，rdf: type 元素允许用户在 RDF 文档中引入某种结构。更多的结构将在本章后面讨论 RDF Schema 时介绍并详细讨论。

3.3.4 简略语法

RDF 文档的语法可以简化，简略语法规则如下：

1. 描述元素中的无子属性(childless property)元素可以被 XML 属性替代，类似于 XML 中的情况；

2. 对于有类型元素的描述元素，可用 rdf: type 元素中指定的名字替换 rdf: Description。

例如，描述

```
<rdf: Description rdf: ID = "CIT1111" >
 <rdf: type rdf: resource = "&uni; 课程"/ >
 <uni: courseName > 离散数学 </uni: courseName >
 <uni: isTaughtBy rdf: resource = "#949318"/ >
</rdf: Description >
```

等价于(对 uni: courseName 使用规则 1)

```
<rdf: Description rdf: ID = "CIT1111"
  uni: courseName = "离散数学" >                                    [74]
   <rdf: type rdf: resource = "&uni; 课程"/ >
   <uni: isTaughtBy rdf: resource = "#949318"/ >
```

```
</rdf: Description>
```

以及(根据规则2)

```
<uni: course rdf: ID="CIT1111"
             uni: courseName="离散数学">
 <uni: isTaughtBy rdf: resource="#949318"/>
</uni: course>
```

记住这三种表示仅仅是相同 RDF 陈述的语法变形，也就是说，它们相对于 RDF 数据模型是等价的，只是 XML 语法形式不同而已。

3.3.5 容器元素

容器元素包含多个资源或属性，这些资源或属性被视为一个整体。比如在上面的例子中，用户希望谈论由某位授课者教授的多门课程。RDF 中有三种不同类型的容器：

`rdf: Bag` 无序容器，允许存储的元素多次重复出现(因此不是一个集合)。典型的例子是工会成员，或者文件夹中的文件，这些例子中元素之间的顺序无关紧要。

`rdf: Seq` 有序容器，元素也可以多次出现，但规定顺序。典型的例子是课程的各个环节，议程的各阶段，以及按字母顺序排列的成员名单，在这些例子中，元素之间是有确定顺序的。

`rdf: Alt` 选项集合(多选一)。典型例子是一个文档的原本和副本，或者是一个文档的不同语言的译本。

容器元素的内容是一些元素，它们使用 `rdf: _1`，`rdf: _2` 等标签标识。下面重述一个完整的 RDF 文档作为例子。

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

<uni: lecturer rdf: about="949352"
               uni: name="Grigoris Antoniou"
               uni: title="教授">
 <uni: coursesTaught>
  <rdf: Bag>
   <rdf: _1 rdf: resource="CIT1112"/>
   <rdf: _2 rdf: resource="CIT3116"/>
  </rdf: Bag>
 </uni: coursesTaught>
</uni: lecturer>
```

```
<uni: lecturer rdf: about = "949318"
               uni: name = "David Billington"
               uni: title = "副教授" >
<uni: coursesTaught >
  <rdf: Bag >
    <rdf: _1 rdf: resource = "CIT1111"/ >
    <rdf: _2 rdf: resource = "CIT3112"/ >
  </rdf: Bag >
 </uni: coursesTaught >
</uni: lecturer >

<uni: lecturer rdf: about = "949111"
               uni: name = "Michael Maher"
               uni: title = "教授" >
 <uni: coursesTaught rdf: resource = "CIT2112"/ >
</uni: lecturer >

<uni: course rdf: about = "CIT1111"
             uni: courseName = "离散数学" >
 <uni: isTaughtBy rdf: resource = "949318"/ >
</uni: course >

<uni: course rdf: about = "CIT1112"
             uni: courseName = "具体数学" >
 <uni: isTaughtBy rdf: resource = "949352"/ >
</uni: course >

<uni: course rdf: about = "CIT2112"
             uni: courseName = "程序设计Ⅲ" >
 <uni: isTaughtBy rdf: resource = "949111"/ >
</uni: course >

<uni: course rdf: about = "CIT3112"
             uni: courseName = "计算理论" >
 <uni: isTaughtBy rdf: resource = "949318"/ >
</uni: course >

<uni: course rdf: about = "CIT3116"
             uni: courseName = "知识表示" >
 <uni: isTaughtBy rdf: resource = "949352"/ >
</uni: course >

</rdf: RDF >
```

也可以不使用 rdf: _1、rdf: _2 这类标签而使用 rdf: li。下例使用了这一语法变形。假定课程 CIT1111 要么由 Grigoris Antoniou 讲授，要么由 David Billington 讲授：

```
<uni: course rdf: about = "CIT1111"
             uni: courseName = "离散数学" >
 <uni: lecturer >
  <rdf: Alt >
   <rdf: li rdf: resource = "949352"/ >
   <rdf: li rdf: resource = "949318"/ >
  </rdf: Alt >
 </uni: lecturer >
</uni: course >
```

容器元素有一个可选的 ID 属性，用来标识容器并且对其加以引用：

```
<uni: lecturer rdf: about = "949318"
               uni: name = "David Billington"
               uni: title = "副教授" >
 <uni: coursesTaught >
  <rdf: Bag rdf: ID = "数据库课程" >
   <rdf: _1 rdf: resource = "CIT1111"/ >
   <rdf: _2 rdf: resource = "CIT3112"/ >
  </rdf: Bag >
 </uni: coursesTaught >
</uni: lecturer >
```

容器元素的一个典型应用是表示两个以上参数的谓词。重新考虑例子 *referee*(*X*, *Y*, *Z*)，这里 *X* 是棋手 *Y* 和棋手 *Z* 的裁判。一个解决方法是区分裁判和棋手。图形表示如图 3-4。它的基于 XML 的语法如下：

77

```
<referee rdf: about = "…#X" >
 <players >
  <rdf: Bag >
   <rdf: li rdf: resource = "…#Y"/ >
   <rdf: li rdf: resource = "…#Z"/ >
  </rdf: Bag >
 </players >
</referee >
```

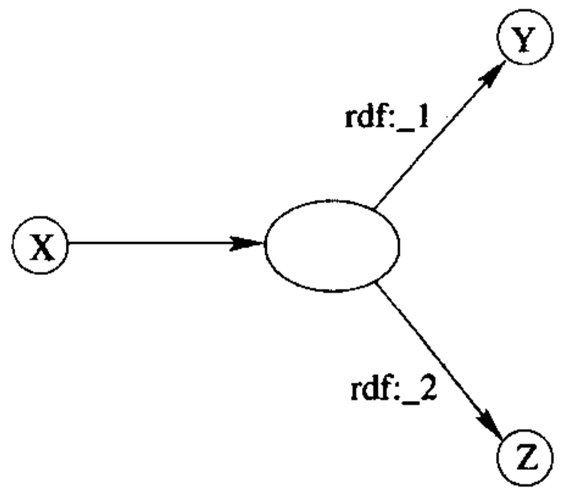


图 3-4　三元谓词的表示

注意，rdf: Bag 定义一个匿名辅助资源。选择 Bag 无序容器是因为假定棋手之间没有区别。如果需要强调顺序的话，比如排在前面的棋手执白棋、排在后面的棋手执黑棋，就应该使用 Seq 有序容器。

这些容器的一个局限是用户无法封闭它们，即无法表达“这些是容器中的所有元素”这种信息。这是因为尽管一张图可以描述它的一些成员，却无法阻止另一张图来补充描述另外一些成员。RDF 允许使用 RDF 集合(collection)的方式来描述只含有特定成员的“群组”(groups)。一个 RDF 集合是一组对象，在 RDF 图中使用表结构表示。这个表结构是用预定义的集合词汇构造的，该词汇由一个预定义的类型 rdf: List，两个预定义的属性 rdf: first 和 rdf: rest 以及一个预定义的资源 rdf: nil 组成。可以这样书写 78

```
<rdf: Description rdf: about = "CIT2112" >
 <uni: isTaughtBy >
  <rdf: List >
   <rdf: first >
    <rdf: Description rdf: about = "949111"/ >
   </rdf: first >
   <rdf: rest >
    <rdf: List >
     <rdf: first >
      <rdf: Description rdf: about = "949352"/ >
     </rdf: first >
     <rdf: rest >
      <rdf: List >
       <rdf: first >
        <rdf: Description rdf: about = "949318"/ >
       </rdf: first >
       <rdf: rest >
        <rdf: Description rdf: about = "&rdf; nil"/ >
       </rdf: rest >
      </rdf: List >
     </rdf: rest >
    </rdf: List >
   </rdf: rest >
  </rdf: List >
 </uni: isTaughtBy >
</rdf: Description >
```

这表明 CIT2112 仅是由标记为资源的 949111，949352，949318 三位教师教授的，不再有其他教师(使用终止符 nil 表示)。一个简洁的语法是使用“Collection”值和 rdf: parseType 属性表示：

```
<rdf: Description rdf: about = "CIT2112" >
  <uni: isTaughtBy rdf: parseType = "Collection" >
    <rdf: Description rdf: about = "949111"/ >
    <rdf: Description rdf: about = "949352"/ >
    <rdf: Description rdf: about = "949318"/ >
```

```
        </uni: isTaughtBy>
79    </rdf: Description>
```

3.3.6 具体化

前面已经说过，有时希望对陈述做出陈述。为此，必须用一个标识符来指代一个陈述。RDF 使用的具体化(reification)机制可以把陈述映射为资源。例如，描述

```
<rdf: Description rdf: about="949352">
 <uni: name>Grigoris Antoniou</uni: name>
</rdf: Description>
```

可具体化为

```
<rdf: Statement rdf: about="StatementAbout949352">
 <rdf: subject rdf: resource="949352"/>
 <rdf: predicate rdf: resource="&uni; name"/>
 <rdf: object>Grigoris Antoniou</rdf: object>
</rdf: Statement>
```

注意，使用 rdf: subject、rdf: predicate 和 rdf: object 可分别提取一个陈述的三个成分。

可以用一个陈述的 ID 来指代它，就像指代其他描述一样。如果不想进一步谈论这个陈述，就用 rdf: Decription；否则就用 rdf: Statement。

如果描述元素中有多个属性元素，它们就对应于多个陈述。这些陈述可以放在无序容器中作为一个整体被引用，也可以分别具体化(见习题 3.1)。

3.4 RDF Schema 的基本思想

RDF 是一种允许用户使用自己的词汇描述资源的通用语言。RDF 是领域无关的，没有定义任何领域的语义，这要由用户借助 RDF Schema(RDFS)来完成。

3.4.1 类和属性

如何描述一个特定的领域呢？考虑 Griffith 大学的课程和授课者这样一个领域。首先必须明确要谈论的“东西”。这是第一个也是最基本的区分。例如，用户希望谈论一个具体的授课者，如

80 David Billington，和一个具体的课程，如离散数学；RDF 已经实现了这一点。与此同时，用户还希望谈论课程，第一学年的课程、授课者、教授等等。这两者有什么区别呢？区别在于：第一种情况谈论的是个体对象(资源)，而在第二种情况谈论的是定义对象类型的类(class)。

类可以看作元素的集合。属于类的个体对象称为该类的实例(instance)。RDF 中，使用

rdf: type 规定实例和类之间的关系。

类的一个重要作用是运用模式(schema)对 RDF 文档所能表达的东西加以限制。程序设计语言中，借助类型可以防止用户写出无意义的表达式(比如 $A+1$，其中 A 是一个数组；因此，规定加法 + 的运算对象必须是数)。RDF 也需要这样。至少，如下陈述是不应该出现的

离散数学由具体数学讲授。

MZH5760 号房间由 David Billington 讲授。

第一个陈述是无意义的，因为课程只能由授课者讲授。这意味着属性“由...讲授”的值受到了某种约束，用数学语言说就是属性的值域受到了约束。

第二个陈述也没有意义，因为只有课程才能被讲授。这意味着属性的主语受到了约束，用数学语言说就是属性的定义域受到约束。

3.4.2 类层次结构和继承

一旦有了类，就希望在它们之间建立关系。例如下述一些类：

职员，助理教授，教师，行政人员，教授，技术人员，副教授

这些类彼此并非没有联系。比如，每一个教授都是教师。我们称“教授”是“教师”的子类(subclass)，或者等价地，称“教师”是“教授”的父类(superclass)。子类关系定义了类的层次结构，如图 3-5 所示。一般而言，如果类 A 的所有实例都是类 B 的实例，A 就是 B 的子类。RDF Schema 并不要求所有的类要形成一个严格层次结构。换句话说，图 3-5 这样的子类图并非一定是树。一个类可以有多个父类。如果类 A 是 B_1 和 B_2 两个类的子类，这只是意味着类 A 的所有实例既是 B_1 类的实例，也是 B_2 类的实例。 81

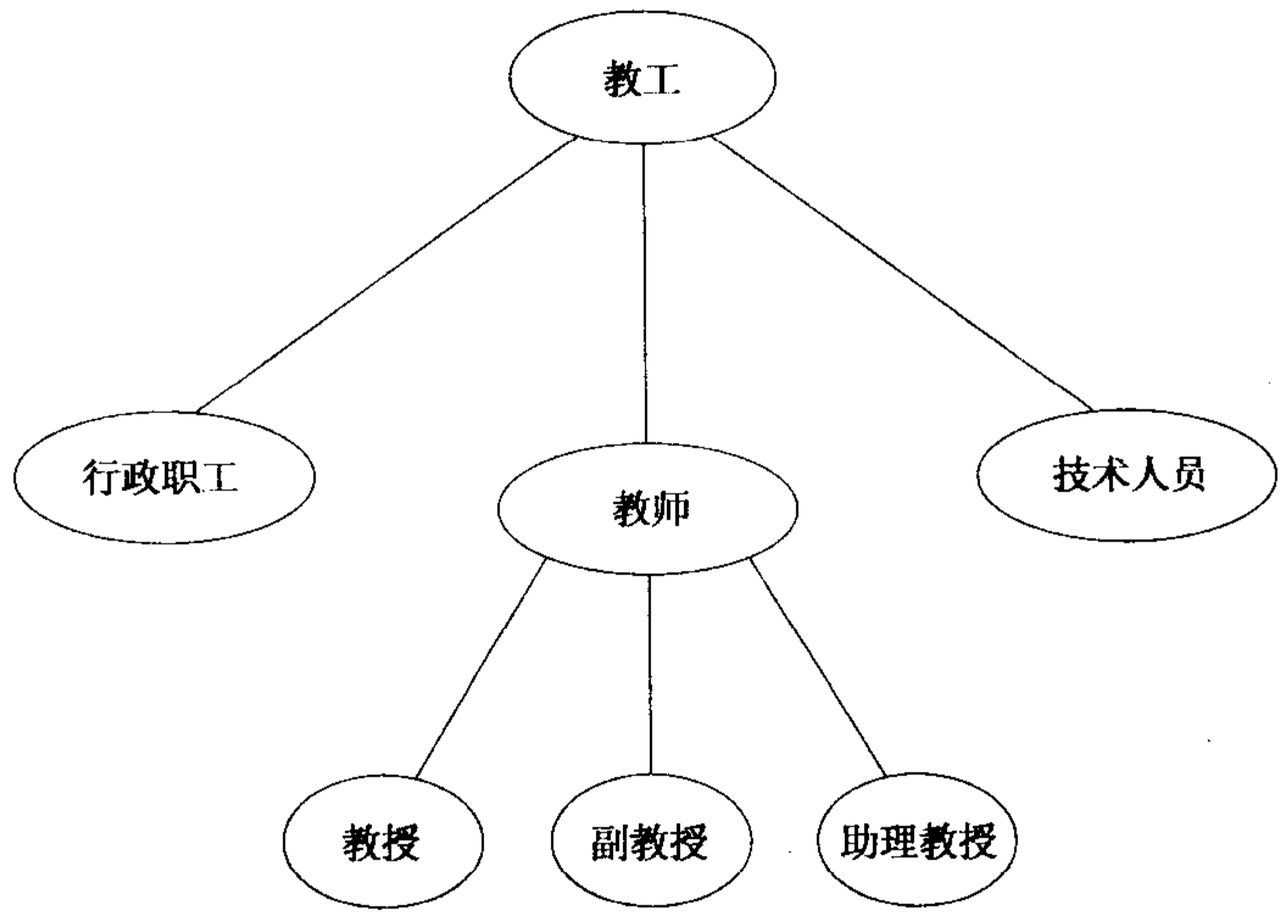


图 3-5 类层次

类的层次化组织具有非常重要的实际意义。考虑下面的值域限制

课程只能由教员讲授。

假定 Michael Maher 定义为教授。那么，根据上述限制，他不能讲授课程，因为没有陈述表明 Michael Maher 也是一个教师。如果在描述中显式地加入该陈述，就可以克服这个困难。但是这么做违背直觉。[⊖]取而代之的做法是让 Michael Maher 从教师这个类“继承”(inherit)授课能力。RDF Schema 正是这么做的。

这样一来，RDF Schema 就限定了“子类”的语义。结果，应用程序就不能随意解释“子类”的含义了；相反地，所有 RDF 处理程序都必须遵循这一预定义语义。通过这样的语义定义，RDFS 成为可以为特定领域定义语义的语言(虽然还很有限)。换句话说，RDF
82 Schema 是一个原始本体语言(ontology language)。

类、继承和属性当然也出现在其他计算领域中，如面向对象程序设计。尽管有很多相似之处，差别也是存在的。在面向对象程序设计中，一个对象类定义可施用于该类的属性。给一个类增加新的属性就意味着改变这个类。

然而在 RDFS 中，属性是全局定义的，也就是说，它们并不在类定义中作为属性封装起来。用户可以定义作用于一个已有类的新属性而不改变这个类。

一方面，这是一个影响深远的强有力的机制——可以使用别人定义的类，通过引入新的属性对其进行改写以满足自己的需要；另一方面，这种对属性的处理背离了在面向对象程序设计和建模领域已经广泛使用的标准方法。这是 RDF/RDFS 的另一个独特风格。

3.4.3 属性层次结构

RDFS 可以定义类之间的层次关系，对于属性也同样如此。例如，“由…讲授”是“参与”的子属性。如果课程 c 是由教师 a 讲授的，则 a 也参与到 c 中。但反过来未必成立。例如，a 可能是课程的组织者，或者是助教，只负责批改学生作业而不教授课程 c。

一般而言，如果 $P(x, y)$ 成立必然有 $Q(x, y)$ 成立，则称 P 是 Q 的子属性。

3.4.4 RDF 与 RDFS 层次

最后，用一个简单的例子来说明 RDF 和 RDFS 所涉及的不同层次。考虑一个 RDF 陈述

离散数学由 David Billington 讲授。

⊖ 这并不是说，“Michael Maher 是一个教师”这个陈述不成立或者违反直觉，而是说增加这个陈述并以此消除上述困难这两种做法是违背直觉的。直观上，教授是教师的子类，因此一般情况下，教授应该“不言自明地”具有教师的属性(这就是所谓属性的“继承”)，故教师能做的事情(比如授课)教授也可以做。书中下面给出的办法(即利用子类关系和“继承”解决问题)就是基于这个直觉的。——译者注

这个陈述的模式可能由下列类和属性组成：授课者、教师、教工、第一学年课程，由…讲授，参与，电话，员工 ID。图 3-6 描述了例子中 RDF 和 RDF Schema 的层次。图中，方块是属性，虚线上面的椭圆表示类，虚线下面的椭圆表示实例。 83

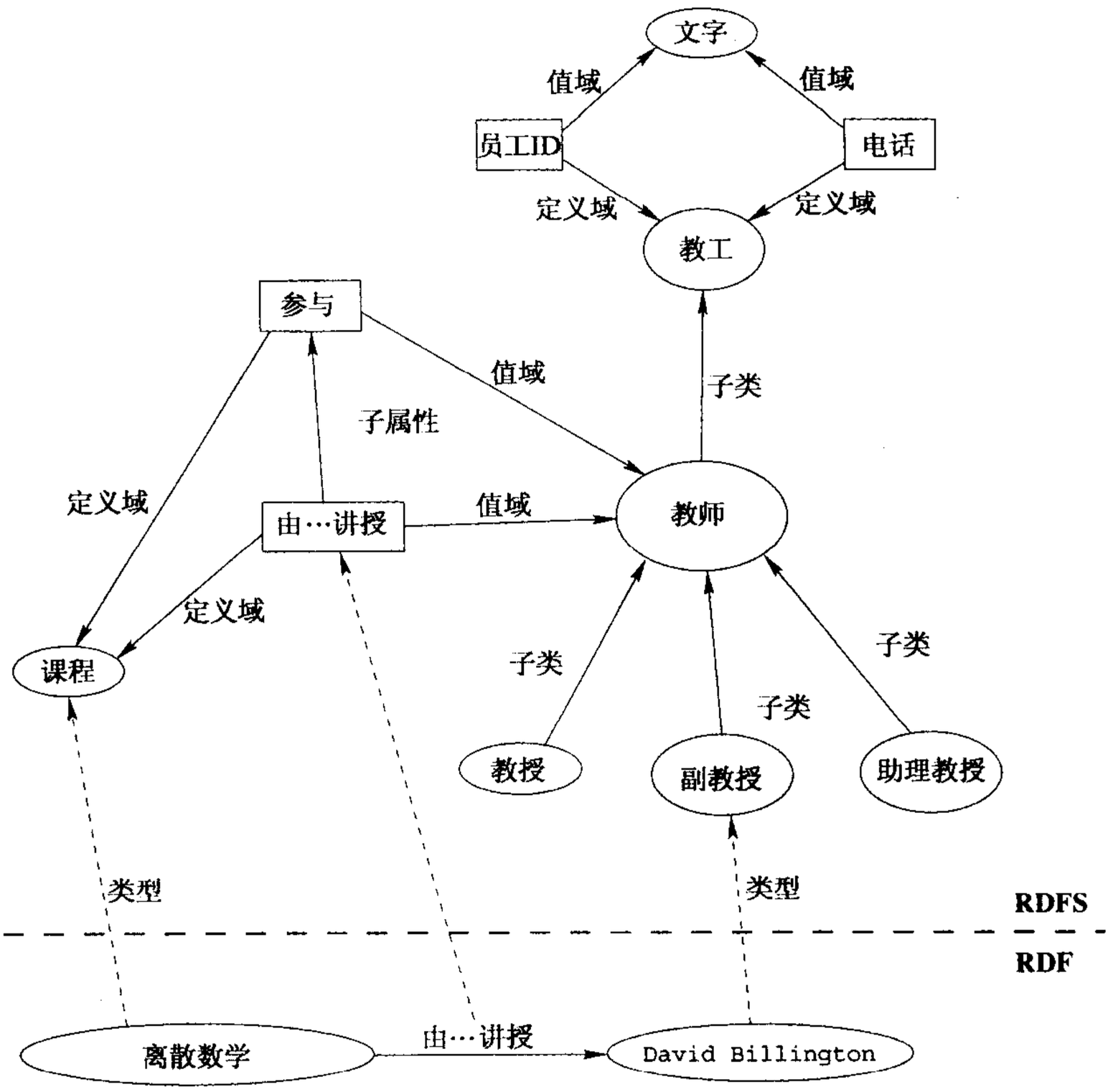


图 3-6 RDF 和 RDFS 层次

图 3-6 所示的模式可以用 RDF Schema 形式化地表达出来，包括如下关键字：subClassOf，Class，Property，subPropertyOf，Resource 等等。下面详细介绍 RDF Schema 的语法。

3.5 RDF Schema 语言

RDF Schema 提供的建模原语可以表达 3.4 节描述的信息。下一步要考虑选择使用什么样的形式语言。使用 RDF 自身是自然的——RDF Schema 的建模原语是用资源和属性定义的。这个选择的合理性在图 3-6 中得到证实。这张图反映了类/属性层次以及对应的实例，显然，它本身也是被标记的图，可以用 RDF 编码。需要记住，RDF 允许用户陈述任何资源，而任何有 URI 的东西都可以成为资源。因此，如果要表达“授课者”类是“教师”类的子 84

类，则需要

1. 定义授课者、教师、子类三个资源

2. 把子类定义为一个属性

3. 书写三元组(子类，授课者，教师)

这三步都可用 RDF 表达。因此，一个 RDFS 文档(它是一个 RDF Schema)也是一个 RDF 文档，并且使用基于 XML 的 RDF 语法。尤其是，它仍然遵循 3.3 节中的语法规定。

下面定义 RDF Schema 的建模原语。

3.5.1 核心类

核心类是

rdfs: Resource，所有资源的类。

rdfs: Class，所有类的类。

rdfs: Literal，所有文字(字符串)的类。到目前为止，文字是 RDF/RDFS 中仅有的“数据类型”。

rdf: Property，所有属性的类。

rdf: Statement，所有具体化陈述的类。

例如，授课者类可以如下定义：

```
<rdfs: Class rdf: ID="授课者">
  ...
</rdfs: Class>
```

85

3.5.2 用于定义关系的核心属性

用于定义关系的核心属性有

rdf: type，把一个资源和它所属的类联系起来(见 3.3.3 节)，将该资源声明为该类的一个实例。

rdfs: subClassOf，把一个类和它的父类联系起来。一个类的所有实例同时也是其父类的实例。注意，一个类可以是多个类的子类。例如，女教授类可以是女性的子类，同时也可以是教授的子类。

rdfs: subPropertyOf，把一个属性和它的一个父属性联系起来。

下例陈述了这样一种子类关系——所有授课者都是教工：

```
<rdfs: Class rdf: about="授课者">
 <rdfs: subClassOf rdf: resource="教工"/>
</rdfs: Class>
```

注意，rdfs: subClassOf 和 rdfs: subPropertyOf 根据定义是传递的。此外，rdfs: Class 是 rdfs: Resource 的子类(类也是资源)，并且 rdfs: Resource 还是 rdfs: Class 的一个实例(rdfs: Resource 是所有资源的类，所以它也是一个类!)。由于同样的原因，每个类都是 rdfs: Class 的实例。

3.5.3 用于约束属性的核心属性

用于对属性加以约束的核心属性如下：

rdfs: domain，限定属性 *P* 的定义域，即规定在谓词 *P* 的三元组中以主语身份出现的资源所属的类。如果不限定定义域，任何资源都可以作为三元组的主语。

rdfs: range，限定属性 *P* 的值域，即规定在谓词 *P* 的三元组中以值的身份出现的那些资源所处的类。

下面的例子中规定，电话号码只能为教工所拥有，并且它的值是总是文字。

```
<rdf: Property rdf: ID="电话号码">
 <rdfs: domain rdf: resource="#教工"/>
 <rdfs: range rdf: resource="&rdf; Literal"/>
</rdf: Property>
```

86

在 RDFS 中，还有

rdfs: ConstraintResource，所有约束的类。

rdfs: ConstraintProperty，包含定义约束的所有属性，它仅有两个实例：rdfs: domain 和 rdfs: range，它也定义为 rdfs: ConstriantResource 和 rdf: Property 的子类。

图 3-7 和图 3-8 描绘了 RDFS 中建模原语之间的关系。

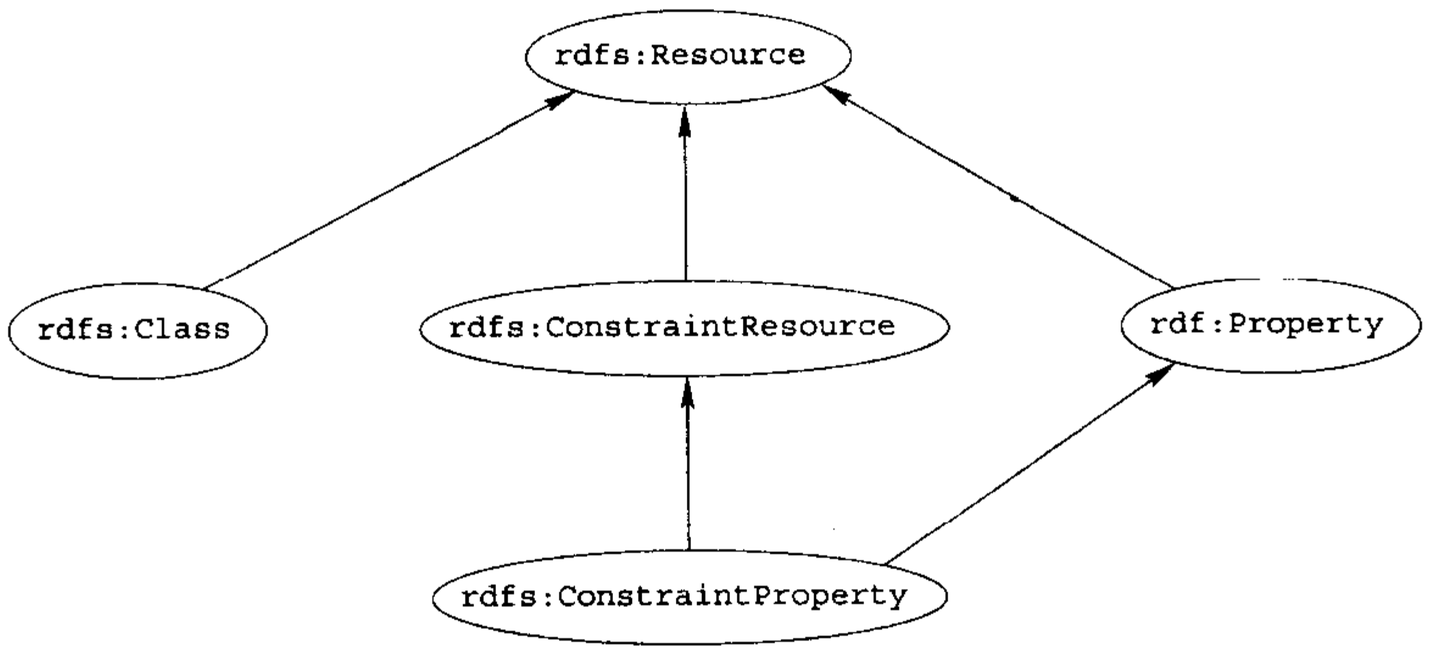
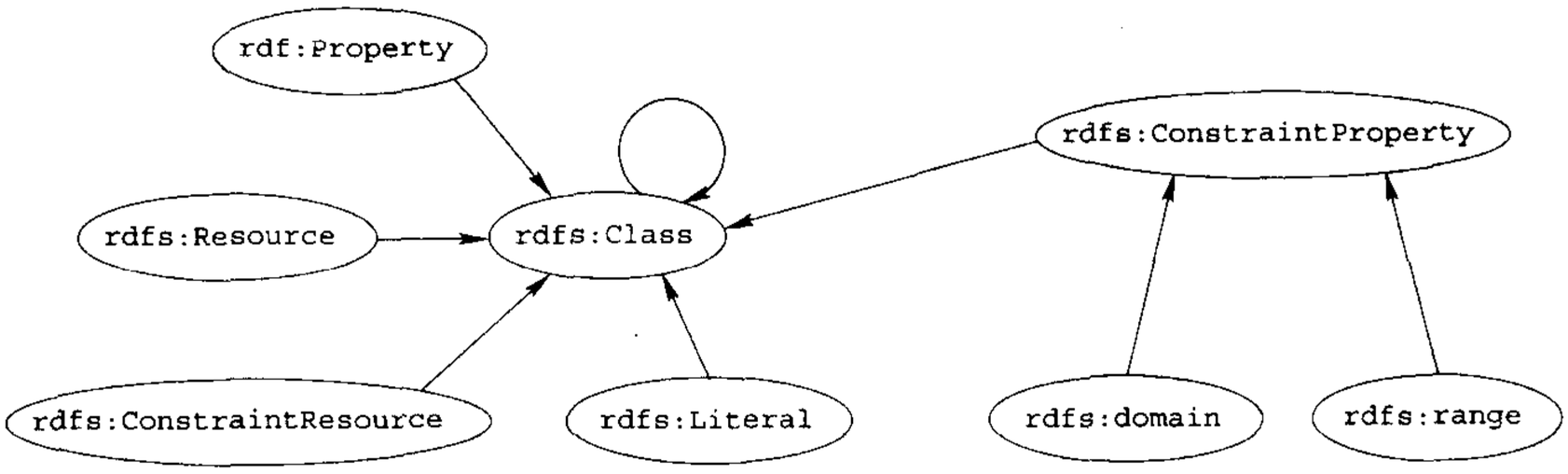


图 3-7 RDFS 部分建模原语的子类层次



87 图 3-8 RDFS 部分建模原语的实例关系

3.5.4 用于具体化的常用属性

下列属性在具体化(见 3.3.6 节)时非常有用:

rdf: subject, 把一个具体化的陈述映射为它的主语;

rdf: predicate, 把一个具体化的陈述映射为它的谓语;

rdf: object, 把一个具体化的陈述映射为它的宾语。

3.5.5 容器类

在 3.3.5 节已经提到, 容器元素是

rdf: Bag, 所有无序容器的类;

rdf: Seq, 所有有序容器的类;

rdf: Alt, 所有可选项容器的类;

rdf: Container, 是所有容器类的父类, 包含上述三个。

3.5.6 工具属性

一个资源可以在网络上的许多不同地方定义和描述。下面的属性允许用户把这些地址链接起来:

rdfs: seeAlso 把资源和解释它的另一个资源联系起来。

rdfs: isDefinedBy 是 rdfs: seeAlso 的子属性, 把资源和定义它的位置(通常是一个 RDF Schema)联系起来。

通常为读者提供更多的信息也很有用, 这可以使用下列属性来实现:

rdfs: comment, 注释, 通常是一段话, 与资源相关联;

rdfs: label, 资源可以附带一个用户友好的标签(名字)。它可以作为 RDF 文档的

88 图形化表示中节点的名称, 此外还有其他用处。

3.5.7 例子：大学

下面来分析课程和授课者例子，并提供该领域的一个概念模型，也就是一个本体。

```
<rdf: RDF
  xmlns: rdf = "http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
  xmlns: rdfs = "http: //www. w3. org/2000/01/rdf-schema#" >

<rdfs: Class rdf: ID = "授课者" >
 <rdfs: comment >授课者类，所有授课者都是教师。
 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#教师"/ >
</rdfs: Class >

<rdfs: Class rdf: ID = "教师" >
 <rdfs: comment >
 教师类
 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#教工"/ >
</rdfs: Class >

<rdfs: Class rdf: ID = "教工" >
 <rdfs: comment >教工类 </rdfs: comment >
</rdfs: Class >

<rdfs: Class rdf: ID = "课程" >
 <rdfs: comment >课程类 </rdfs: comment >
</rdfs: Class >

<rdf: Property rdf: ID = "参与" >
 <rdfs: comment >
 关联授课者和所讲授的课程
 </rdfs: comment >
 <rdfs: domain rdf: resource = "#课程"/ >
 <rdfs: range rdf: resource = "#授课者"/ >
</rdf: Property >

<rdf: Property rdf: ID = "讲授" >
 <rdfs: comment >
 从父属性“参与”继承其定义域(“课程”)和值域(“授课者”)
 </rdfs: comment >
 <rdfs: subPropertyOf rdf: resource = "#参与"/ >
</rdf: Property >
```

```
<rdf: Property rdf: ID = "电话" >
 <rdfs: comment > 它是教工的属性，值是文字类型。
 </rdfs: comment >
 <rdfs: domain rdf: resource = "#教工"/ >
 <rdfs: range rdf: resource = "&rdf; Literal"/ >
 </rdf: Property >
</rdf: RDF >
```

3.5.8 例子：机动车

这里再给出一个关于机动车的简单本体的例子。类关系如图 3-9 所示。

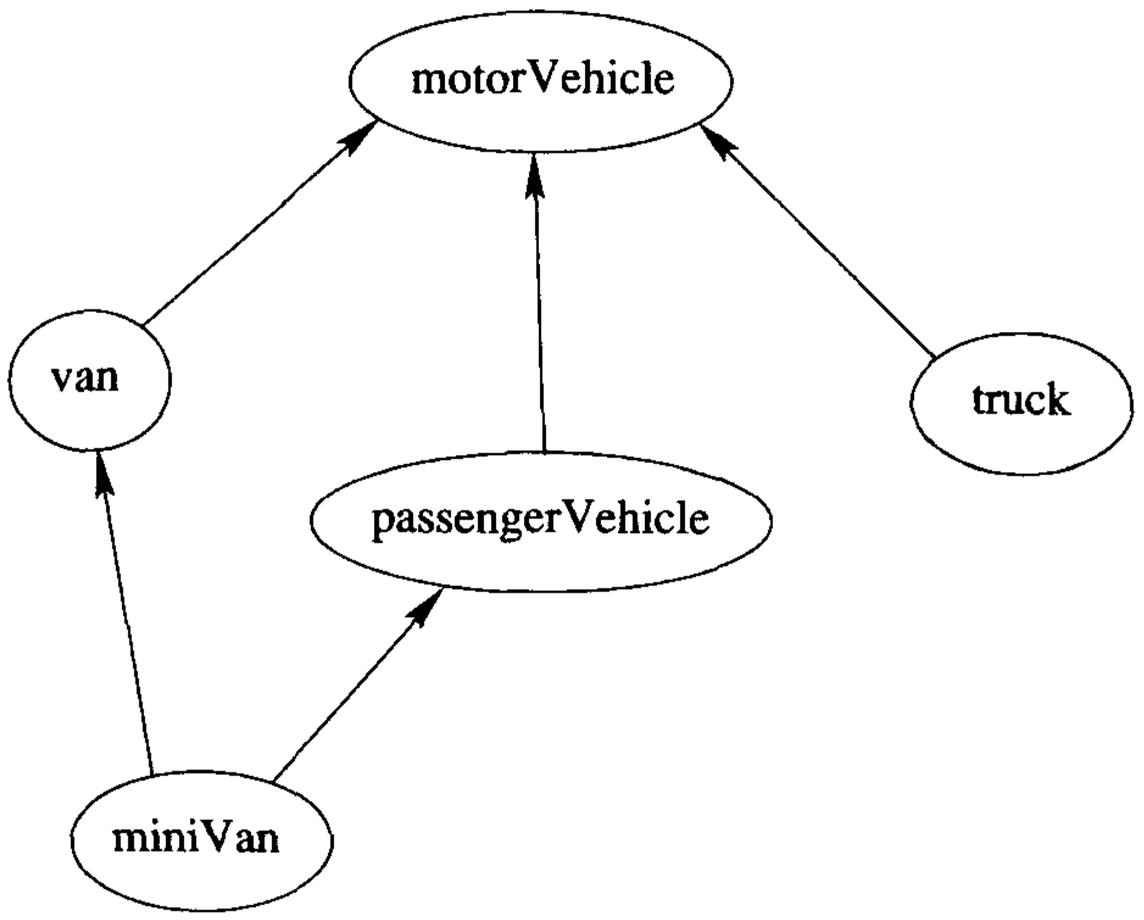


图 3-9 机动车例子的类层次


```
<rdf: RDF
   xmlns: rdf = "http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
   xmlns: rdfs = "http: //www. w3. org/2000/01/rdf-schema# " >

<rdfs: Class rdf: ID = "机动车"/ >

<rdfs: Class rdf: ID = "大货车" >
 <rdfs: subClassOf rdf: resource = "#机动车"/ >
</rdfs: Class >

<rdfs: Class rdf: ID = "卡车" >
 <rdfs: subClassOf rdf: resource = "#机动车"/ >
</rdfs: Class >

<rdfs: Class rdf: ID = "客车" >
 <rdfs: subClassOf rdf: resource = "#机动车"/ >
```

```
</rdfs: Class >

<rdfs: Class rdf: ID = "小货车" >
 <rdfs: subClassOf rdf: resource = "#客车"/ >
 <rdfs: subClassOf rdf: resource = "#大货车"/ >
 </rdfs: Class >
</rdf: RDF >
```

3.6 用 RDF Schema 描述 RDF 和 RDF Schema

我们已经知道了 RDF 和 RDFS 语言的主要语法构成，再了解一下 RDF 和 RDFS 的定义会很有指导意义。这些定义是用 RDF Schema 的语言表达的。既然每一个成分的含义已经规定清楚，这些定义是很容易读懂的。

下面的定义只是全部语言定义的一部分。剩下的部分可以在 rdf: RDF 元素的命名空间中找到。

3.6.1 RDF

```
<? xml version = "1.0" encoding = "UTF-16"? >

<rdf: RDF
   xmlns: rdf = "http: //www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns: rdfs = "http: //www.w3.org/2000/01/rdf-schema#" >

 <rdfs: Class rdf: ID = "Statement"
   rdfs: comment = "The class of triples consisting of a
                    predicate, a subject and an object
                    (that is, a reified statement)"/ >

<rdfs: Class rdf: ID = "Property"
  rdfs: comment = "The class of properties"/ >

<rdfs: Class rdf: ID = "Bag"
  rdfs: comment = "The class of unordered collections"/ >

<rdfs: Class rdf: ID = "Seq"
  rdfs: comment = "The class of ordered collections"/ >

<rdfs: Class rdf: ID = "Alt"
  rdfs: comment = "The class of collections of alternatives"/ >

<rdf: Property rdf: ID = "predicate"
```

91

```
  rdfs: comment = "Identifies the property used in a statement
                   when representing the statement
in reified form" >
  <rdfs: domain rdf: resource = "#Statement"/ >
  <rdfs: range rdf: resource = "#Property"/ >
</rdf: Property >

<rdf: Property rdf: ID = "subject"
  rdfs: comment = "Identifies the resource that a statement is
                   describing when representing the statement
                   in reified form" >
  <rdfs: domain rdf: resource = "#Statement"/ >
</rdf: Property >

<rdf: Property rdf: ID = "object"
  rdfs: comment = "Identifies the object of a statement
                   when representing the statement
                   in reified form"/ >

<rdf: Property rdf: ID = "type"
  rdfs: comment = "Identifies the class of a resource
                   The resource is an instance
of that class. "/ >

92      </rdf: RDF >
```

3.6.2 RDF Schema

```
<rdf: RDF
   xmlns: rdf = "http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
   xmlns: rdfs = "http: //www. w3. org/2000/01/rdf-schema#" >

<rdfs: Class rdf: ID = "Resource"
  rdfs: comment = "The most general class"/ >

<rdfs: Class rdf: ID = "comment"
  rdfs: comment = "Use this for descriptions" >
  <rdfs: domain rdf: resource = "#Resource"/ >
  <rdfs: range rdf: resource = "#Literal"/ >
</rdfs: Class >

<rdfs: Class rdf: ID = "Class"
  rdfs: comment = "The concept of classes
```

```
                    All classes are resources" >
  <rdfs: subClassOf rdf: resource = "#Resource"/ >
</rdfs: Class >

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

<rdf: Property rdf: ID = "subPropertyOf" >
  <rdfs: domain rdf: resource = "&rdf; Property"/ >
  <rdfs: range rdf: resource = "&rdf; Property"/ >
</rdf: Property >

</rdf: RDF >
```

上述命名空间并没有提供 RDF 和 RDFS 的完整定义。考虑 rdfs: subClassOf，命名空间只规定它可以作用于类，并且以类作为取值。“作为一个子类”(即一个类的每一个实例都是另一个类的实例)是什么含义并没有表达出来。实际上，也无法使用 RDF 文档表示。如果可以的话，也就没有必要定义 RDF Schema 了。

下一节将给出形式语义。当然，RDF 分析器和其他 RDF 工具软件(包括查询处理器)必须完全掌握这些语义。 93

3.7 RDF 和 RDF Schema 的公理语义

本节将形式化定义 RDF 和 RDF Schema 建模原语，以刻画 RDF 和 RDFS 的语义。

所使用的形式语言是谓词逻辑(predicate logic)，它通常被认为是所有(基于符号的)知识表示的基础。这个形式化中出现的公式称为公理(axioms)。

用逻辑描述的 RDF 和 RDFS 的语义排除了二义性，并且是机器可解读的，同时也为借助逻辑推理机支持 RDF/RDFS 的自动推理提供了基础。

3.7.1 基本方法

RDF 和 RDFS 的所有原语都表示为常元，如 *Resource*、*Class*、*Property*、*subClassOf* 等等。引入几个预定义谓词描述这些常元之间的关系。

同时还需要一个关于表的辅助理论，它有下述函数词(符号)

nil：空表

cons(x, l)：在表头添加一个元素

first(l)：返回表 l 的第一个元素

rest(*l*)：返回表 l 的其余部分

以及谓词符号

item(*x*, *l*)：测试元素 *x* 是否在表 *l* 中出现

list(*l*)：测试 *l* 是否一个表

表可以用来表示 RDF 容器。在一些更强的本体语言中，还需要用它来表达某些构造(constructs)如基数限制的含义。

大部分公理提供类型信息，例如，

Type(*subClassOf*, *Property*)

的含义是说 *subClassOf* 是一个属性。使用带“等词”[⊖]的谓词逻辑，变量名都以？打头，默认
94 所有公理都是全称封闭的。

本节给出 RDF 和 RDF Schema 大部分元素的定义。完全的公理语义可以参考在线文档(见参考文献 Fikes & McGuinness 2001)。

3.7.2 基本谓词

基本谓词如下

PropVal(*P*, *R*, *V*)，三元谓词，表示一个 RDF 陈述，资源为 *R*，属性为 *P*，值为 *V*；

Type(*R*, *T*)，是 *PropVal*(*type*, *R*, *T*)的简写，表示资源 *R* 的类型为 *T*

$$Type(?r, ?t) \longleftrightarrow PropVal(type, ?r, ?t)$$

3.7.3 RDF

一个 RDF 陈述(三元组)(*P*, *R*, *V*)表示为 *PropVal*(*P*, *R*, *V*)。

类

在 RDF/S 语言中，有常量 *Class*，*Resource*，*Property*，*Literal*。所有类都是 *Class* 类的实例，也就是说，它们的类型为 *Class*：

$Type(Class, Class)$

$Type(Resource, Class)$

$Type(Property, Class)$

$Type(Literal, Class)$

资源(Resource)是最一般的类——每个对象都是资源。因此，类和属性都是资源：

$Type(?p, Property) \longrightarrow Type(?p, Resource)$

$Type(?c, Class) \longrightarrow Type(?c, Resource)$

⊖ 数理逻辑中的“等词”代表直观上的相等关系。——译者注

最后，RDF 陈述中的谓语必须是属性：

$$PropVal(?p, ?r, ?v) \longrightarrow Type(?p, Property)$$ 95

类型属性

类型(trpe)是一个属性

$$Type(type, Property)$$

注意，它也等价于 *PropVal*(*type*，*type*，*Property*)：类型的类型是属性。类型可以作用于资源，它的值是一个类：

$$Type(?r, ?c) \longrightarrow (Type(?r, Resource) \wedge Type(?c, Class))$$

辅助属性 *FuncProp*

函数属性是一个属性，它同时又是函数，它将一个资源与至多一个值相关联。函数属性并非 RDF 的概念，而是用于其他原语公理化的工具。

常元 *FuncProp* 是表示所有函数属性的类。*P* 是函数属性当且仅当它是一个属性，并且不存在 x，y_1，y_2 使得 $P(x, y_1)$，$P(x, y_2)$并且 $y_1 \neq y_2$。

$$Type(?p, FuncProp) \longleftrightarrow$$
$$(Type(?p, Property) \wedge \forall ?r \forall ?v1 \forall ?v2$$
$$(PropVal(?p, ?r, ?v1) \wedge PropVal(?p, ?r, ?v2) \longrightarrow ?v1 = ?v2))$$

具体化陈述

常元 *Statement* 表示所有具体化陈述的类。所有的具体化陈述都是资源，陈述是 *Class* 类的一个实例。

$$Type(?s, Statement) \longrightarrow Type(?s, Resource)$$

$$Type(Statement, Class)$$

一个具体化的陈述可以分解成 RDF 三元组的三部分：

$$Type(?st, Statement) \longrightarrow$$
$$\exists ?p \exists ?r \exists ?v(PropVal(Predicate, ?st, ?p) \wedge$$
$$PropVal(Subject, ?st, ?r) \wedge PropVal(Object, ?st, ?v))$$

主语、谓语和宾语都是函数属性，即：每一个陈述都只有一个主语、一个谓语和一个宾语： 96

$$Type(Subject, FuncProp)$$

$$Type(Predicate, FuncProp)$$

$$Type(Object, FuncProp)$$

它们的类型信息为

$$PropVal(Subject, ?st, ?r) \longrightarrow$$
$$(Type(?st, Statement) \wedge Type(?r, Resource))$$

$$PropVal(Predicate, ?st, ?p) \longrightarrow$$
$$(Type(?st, Statement) \wedge Type(?p, Property))$$

$$PropVal(Object, ?st, ?v) \longrightarrow$$
$$(Type(?st, Statement) \wedge (Type(?v, Resource) \vee Type(?v, Literal)))$$

最后一个公理说明，如果 RDF 陈述的宾语作为属性出现，则它必须作用于具体化陈述，且值为资源或文字。

容器

所有的容器都是资源：

$$Type(?c, Container) \longrightarrow Type(?c, Resource)$$

容器都是表：

$$Type(?c, Container) \longrightarrow list(?c)$$

容器可以是 Bag，或者 Sequence 或者 Alternatives：

$$Type(?c, Container) \longleftrightarrow$$
$$(Type(?c, Bag) \vee Type(?c, Seq) \vee Type(?c, Alt))$$

Bag 和 Sequence 不相交：

$$\neg(Type(?x, Bag) \wedge Type(?x, Seq))$$

对于每一个自然数 $n>0$，存在一个选择符_n，它选取容器中的第 n 个元素。它是一个函数属性

$$Type(_n, FuncProp)$$

并且只作用于容器：

97 $$PropVal(_n, ?c.?o) \longrightarrow Type(?c, Container)$$

3.7.4 RDF Schema

子类和子属性

子类是一个属性：

$$Type(subClassOf, Property)$$

如果类 C 是类 C' 的子类(subClassOf)，那么 C 的所有实例都是 C' 的实例：

$$PropVal(subClassOf, ?c, ?c') \longleftarrow$$
$$(Type(?c, Class) \wedge Type(?c', Class) \wedge$$
$$\forall ?x(Type(?x, ?c) \longrightarrow Type(?x, ?c')))$$

类似地，对于子属性 subPropertyOf：P 是 P' 的子属性如果 $P(x, y)$ 蕴涵 $P'(x, y)$：

$$Type(subPropertyOf, Property)$$

$$PropVal(subPropertyOf, ?p, ?p') \longleftrightarrow$$
$$(Type(?p, Property) \wedge Type(?p', Property) \wedge$$
$$\forall ?r \forall ?v(PropVal(?p, ?r, ?v) \longrightarrow PropVal(?p', ?r, ?v)))$$

约束

每个约束资源(constraint resourse)都是资源

$$PropVal(subClassOf, ConstraintResource, Resource)$$

约束属性正是作为约束资源的属性：

$$Type(?cp, ConstraintProperty) \longleftrightarrow$$
$$(Type(?cp, ConstraintResource) \wedge Type(?cp, Property))$$

domain 和 range 都是约束属性：

$$Type(domain, ConstraintProperty)$$
$$Type(range, ConstraintProperty)$$

domain 和 range 分别定义属性的定义域和值域。前面说过，属性 P 的定义域是 P 作用的所有对象组成的集合。如果 P 的定义域是 D，则对任何 $P(x, y)$，有 $x \in D$。 98

$$PropVal(domain, ?p, ?d) \longrightarrow$$
$$\forall ?x \forall ?y(PropVal(?p, ?x, ?y) \longrightarrow Type(?x, ?d))$$

属性 P 的值域是 P 可以取的所有值。如果 P 的值域是 R，那么对任何 $P(x, y)$，有 $y \in R$。

$$PropVal(range, ?p, ?r) \longrightarrow$$
$$\forall ?x \forall ?y(PropVal(?p, ?x, ?y) \longrightarrow Type(?y, ?r))$$

用前面的规则可以推出下面的公式：

$$PropVal(domain, range, Property)$$
$$PropVal(range, range, Class)$$
$$PropVal(domain, domain, Property)$$
$$PropVal(range, domain, Class)$$

至此得到 RDF 和 RDFS 的形式语义。一个代理只要有了这样的知识，就可以推出一些有趣的结论。例如，只要知道 *teaches* 的定义域是 *academicStaffMember*，*academicStaffMember* 是 *staffMembers* 的子类，以及 *teaches*(*DB*，*DiMa*)，那么代理就可以借助谓词逻辑语义或者某个谓词逻辑证明系统自动推出 *staffMember*(*DB*)这个结论。

3.8 RDF 和 RDFS 的直接推理系统

如上所述，3.7 节中给出的详细公理语义可用于 RDF 和 RDF Schema 的自动推理。然而，这需要借助于一阶逻辑证明系统。这是一个非常苛刻的要求，当需要处理数以百万计的陈述(比如，有数百万的形如 *Type*(？ *r*,？ *c*)的公式)时，通常是不可行的。

由于这个原因，人们直接用 RDF 三元组给出 RDF 的语义，以及相对于该语义可靠性和完全性的一个推理系统。

这个直接推理系统包含如下形式的规则

99 IF E 包含某些三元组

THEN 在 E 中加入一些新的三元组

其中 E 是由 RDF 三元组组成的任意集合。

本节并不重复这个系统的全部推理规则(这可以在 RDF 官方文档中查到)，只给出几个基本的例子：

IF E 含有三元组(？ *x*,？ *p*,？ *y*)

THENE 也包含三元组(？ *p*, rdf: type, rdf: property)

也就是说，从任何在三元组属性位置上的资源？ *p* 都可以推出它是类 rdf: property 的成员。

一个更有趣的例子是下面的规则：

IF E 包含三元组(？ *u*, rdfs: subClassOf,？ *v*)和(？ *v*, rdfs: subClassOf,？ *w*)

THEN E 也包含三元组(？ *u*, rdfs: subClassOf,？ *w*)

它规定子类关系具有传递性。与此密切关联的是下面的规则

IF E 含有三元组(？ *x*, rdf: type,？ *u*)和(？ *u*, rdfs: subClassOf,？ *v*)

THEN E 也包含三元组(？ *x*, rdf: type,？ *v*)

这是 rdfs: subClassOf 含义的本质定义。

初次了解 RDFS 的人会对下面的例子感到惊奇：

IF E 包含三元组 (？ *x*,？ *p*,？ *y*)和 (？ *p*, rdfs: range,？ *u*)

THEN E 也包含三元组 (？ *y*, rdf: type,？ *u*)

这个规则的意思是，如果资源？ *y* 作为属性？ *p* 的值出现，则可以推出它是？ *p* 值域的成员。这表明 RDF Schema 的值域定义用来决定值域的成员，而不是用来限制属性的值域。⊖

整个规则集合大约包含几十条规则，可以无需使用复杂的定理证明技术而实现它们的高效率推导。

⊖ 这里，“限制属性的值域”的意思是“对已经定义的属性值域施加进一步约束并使之缩小”。OWL 有这项功能，参见下一章；但 RDF/RDFS 没有，参见习题 3.3 和 3.19。RDF/RDFS 关于定义域和值域的定义功能译为“定义域和值域限定”。——译者注

3.9 使用 RQL 查询

本节介绍一种 RDF 的查询语言。首先说明为什么不用 XML 查询语言而需要一种新的查询语言。概括地说，这是由于 XML 的抽象层次比 RDF 低，如果用基于 XML 的语言查询 100
RDF 文档，就会非常复杂。下面作进一步解释。

前面已经提到，使用 XML 语法表示 RDF 的陈述有多种方法。例如，假定想检索所有授课者的头衔。对于一个特定的授课者的描述可能是这样的：

```
<rdf: Description rdf: about = "949318" >
 <rdf: type rdf: resource = "&uni; 授课者"/ >
 <uni: name > David Billington < /uni: name >
 <uni: title > 副教授 < /uni: title >
< /rdf: Description >
```

一个合适的 Xpath 查询是

```
/rdf: Description[rdf: type =
  "http: //www. mydomain. org/uni-ns#授课者"]/uni: title
```

但该描述本来也可以写成下面的形式：

```
<uni: lecturer rdf: about = "949318" >
 <uni: name > David Billington < /uni: name >
 <uni: title > 副教授 < /uni: title >
< /uni: lecturer >
```

这样一来，前面的 XPath 查询语句就无效了，因而必须写成

```
//uni: lecturer/uni: title
```

此外，这个描述还有第三种可能的表示

```
<uni: lecturer rdf: about = "949318"
               uni: name = "David Billington"
               uni: title = "副教授"/ >
```

对于这个语法变形，需要再写一个对应的 XPath 查询：

```
//uni: lecturer/@ uni: title
```

对一个授课者的描述可以采取上述三种等价表达方式中的任何一种，但必须对应地写出不同的 XPath 查询。

当然，更好的办法是在 RDF 层写查询语句。一个合适的查询语言必须理解 RDF，也就是说，它不仅要理解 RDF 的语法，还要理解 RDF 的数据模型和它的词汇所表示的语义。

101 此外，查询语言也要了解 RDF Schema 的语义。例如，给出信息

```
<uni: lecturer rdf: about = "949352" >
 <uni: name > Grigoris Antoniou </uni: name >
</uni: lecturer >

<uni: professor rdf: about = "949318" >
 <uni: name > David Billington </uni: name >
</uni: professor >

<rdfs: Class rdf: about = "&uni; 教授" >
 <rdfs: subClassOf rdf: resource = "&uni; 授课者"/ >
</rdfs: Class >
```

如果要查询所有授课者的姓名，则返回结果应该是 Grigoris Antoniou 和 David Billington。

在撰写本书的时候(2003 年中期)，还没有推出 RDF 和 RDFS 查询语言的标准，不仅 W3C 没有推荐，实际上也没有出现。在本节讨论中，将选择 RDF 查询语言 RQL 加以介绍，因为它具备了作为 RDF 和 RDFS 查询语言的许多特征，比如路径表达式和模式敏感性。然而，还存在其他的查询语言(如 RDQL)，即使 RQL 本身也在改进之中。

3.9.1 基本查询

查询语句 Class 可以检索所有的类，查询语句 Property 可以检索所有的属性。要获得一个类的所有实例，比如类 course，就写

```
course
```

这个查询也会返回 course 类的子类的所有实例，这是完全正确的。但是，如果不想检索这些通过子类层次继承下来的实例，就应该写

```
^course
```

简单使用查询 involves 就可以查询到与属性 involves 相联系的资源和值。结果还包括 involves 的子属性。例如，它可以取出所有与继承属性 isTaughtBy 关联的三元
102 组。如果不想得到这些多余的结论，就写^involves。

3.9.2 使用 select-from-where 查询

和 SQL 一样，

select 规定查询数据的数量和顺序

from 用于选取被查询的数据模型

where　对可能的查询结果施加限制

例如，要检索所有教工的电话号码，可以写

```
select X, Y from {X}phone{Y}
```

这里 X 和 Y 都是变元，{X}phone{Y}表示一个资源-属性-值三元组。要检索所有授课者及其电话号码，则用

```
select X, Y from lecturer{X}.phone{Y}
```

这里，lecturer{X}收集类 lecturer 的所有实例，然后把结果与变元 X 绑定。第二部分收集谓词 phone 的所有三元组。但这里还有一个隐含的连接操作，把第二个查询局限于资源部分为变量 X 的那些三元组。在这个例子中是限制 *phone* 的定义域为授课者。这种隐含连接操作记为一个点“.”。

下面的查询选出由 ID 号为 949352 的授课者讲授的所有课程的名称，在这里体现了显式的连接操作。

```
select N
from course{X}.isTaughtBy{Y}, {C}name{N}
where Y = "949352" and X = C
```

除了等号以外，还有其他比较运算符。例如，X < Y 表示“X 低于 Y”。在 X，Y 都是字符串的情况下，X < Y 就是 X 就是按照字典序排在 Y 之前；如果 X 和 Y 都是类，则 X 是 Y 的子类。

3.9.3　模式查询

RQL 允许检索模式信息。模式变量的名称要使用前缀 $（对类）和@（对属性）。例如，　103

```
select X, $X, Y, $Y
from {X: $X}phone{Y: $Y}
```

检索出与 phone 属性相联系的所有三元组的资源和值，phone 的子属性，以及类。注意，这些类不一定与 phone 的定义域和值域完全吻合，因为它们可以是定义域或值域的子类。例如，根据如下描述

phone（“949352”，“5041”）
type（“949352”，*lecturer*）
subclass（*lecturer*，*staffMember*）
domain（*phone*，*staffMember*）
range（*phone*，*literal*）

得到

("949352", *lecturer*, "5041", *literal*)

虽然 *lecturer* 不是 *phone* 的定义域。

一个属性的定义域和值域可以这样检索：

```
select domain(@ P), range(@ P)
from @ P
where @ P = phone
```

更多细节请参考 RQL 用户手册(v2.0)(2003)。

3.10 小结

- RDF 为表示和处理元数据提供了一个基础。
- RDF 有一个基于图的数据模型，其核心概念是资源、属性和陈述。一个陈述是一个资源-属性-值三元组。
- RDF 有一个基于 XML 的语法，支持语法可共用性。XML 和 RDF 彼此互补，因为 RDF 支持语义可共用性。
- RDF 体现了分布式数据的思想，允许知识的递增构建以及共享和复用。

104

- RDF 是领域无关的。RDF Schema 提供了一种描述具体领域知识的机制。
- RDF Schema 是一种原始本体语言，提供了一套具有固定语义的建模原语。RDF-Schema 的核心概念是类、子类关系、属性、子属性关系，以及定义域和值域限定。
- 存在 RDF 和 RDFS 的查询语言。

下一章将讨论以下内容：

- RDFS 作为网络建模语言是相当原始的，很多需要的建模原语尚未定义。
- 因此需要在 RDF/RDFS 之上再建立一个本体层。

推荐读物

下列资源是官方在线文档：

- G. Klyne and J. Carroll, eds. Resource Description Framework (RDF): Concepts and Abstract Syntax. January 23, 2003. <http://www.w3.org/TR/rdf-concepts>.
- D. Brickley and R.V. Guha, eds. RDF Vocabulary Description Language 1.0: RDF Schema, January 23, 2003. <http://www.w3.org/TR/rdf-schema>.
- P. Hayes, ed. RDF Semantics, January 23,2003. <http://www.w3.org/TR/rdf-mt/>.

- D. Beckett, ed. RDF/XML Syntax Specification, January 23, 2003. <http://www.w3.org/TR/rdf-syntax-grammar/>.
- F. Manola and E. Miller, eds. RDF Primer. <http://www.w3.org/TR/rdf-primer/>.
- R. Fikes and D. McGuinness. An Axiomatic Semantics for RDF, RDF Schema and DAML+OIL, October 2001. <http://www.daml.org/2001/03/axiomatic-semantics.html>.
- The RQL v2.0 User Manual, July 12, 2003. <http://139.91.183.30:9090/RDF/RQL/Manual.html>.

下面是一些更有阅读价值的读物： 105

- S. Decker et al. The Semantic Web: The Roles of XML and RDF. *IEEE Internet Computing* 15,3 (October 2000): 63–74.
- D. Dodds et al. *Professional XML Meta Data*. Birmingham, U.K., Wrox Press, 2001.
- J. Hjelm. *Creating the Semantic Web with RDF*. New York, Wiley, 2001.
- G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. *Querying Community Web Portals*. Technical Report, ICS-FORTH, Heraklion, Greece, November 2000. <http://139.91.183.30:9090/RDF/publications/sigmod2000.html>.
- J. Broekstra, Sesame RQL: a tutorial. <http://sesame.aduna.biz/publications/rql-tutorial.html>.
- M. Nic. RDF Tutorial - Part I: Basic Syntax and Containers. <http://www.zvon.org/xxl/RDFTutorial/General/book.html>.

下列网站常年维护关于工具软件及其他资源的详细列表：

- <http://www.ilrt.bris.ac.uk/discovery/rdf/resources/>.
- <http://www.w3.org/RDF>

练习与设计

3.1 研读 RDFS 命名空间并理解本章中没有介绍的元素的含义。

3.2 阅读 RQL 用户手册，注意本章没有讨论的内容。

3.3 RDFS 允许对一个属性定义多个定义域并使用它们的并集。讨论用并集作为定义域和用交集作为定义域的优缺点。

3.4 在 RDFS 的一个早期版本中，`rdfs: subClassOf` 是不允许有环的。试找一个适合用环形类关系描述的例子(提示：考虑类之间的等价关系)。

106 3.5 讨论下列陈述之间的区别，并用图形表示：

a X 支持提议，Y 支持提议，Z 支持提议。

b 由 X，Y，Z 三人组成的小组支持提议。

3.6 比较 rdfs: subClassOf 和 XML Schema 的类型扩展。

3.7 考虑 rdf: _n 在公理语义里的形式定义。它是否正确刻画了 rdf: _n 作为一个集合第 n 个元素选择符的直观含义？如果没有，提出一个完善的公理化定义。

3.8 使用 3.7 节给出的公理证明该节末尾给出的导出公式。

3.9 讨论为什么 RDF/S 不允许逻辑矛盾——任何 RDF/S 文档都是相容的，因此，它至少具有一个模型。

3.10 试把关系数据库模型影射到 RDF 上。

3.11 比较关系-实体建模和 RDF。

3.12 为图书馆写一个 RDF Schema：书，作者，出版社，出版年，印数，出版日期等。再用 RDF 写一些陈述并用 RQL 查询。

3.13 写一个关于地理信息的本体：城市，国家，首都，边界，州等。

3.14 第 2 章要求读者考虑各种不同的领域，为它们构造适当的词汇。试通过定义合适的类和属性为这些领域建主概念模型，然后用 RDF 写一些陈述。

下面要求读者考虑 RDFS 的局限性。具体地说，还有哪些东西是应该表达的？在 RDFS 中能不能表达？这些局限性与第 4 章的内容有关，在那里将介绍表达能力更为丰富的建模语言。

3.15 考虑男性和女性两个类。定义它们之间的一个在本体中应该成立的关系。

3.16 考虑人，男性和女性三个类。定义在一个本体中应该描述这三类之间的关系。这个

107 关系中的哪个部分可以用 RDFS 表达？

3.17 假定有人声称 Bob 和 Peter 都是 Mary 的父亲。显然这里有语义错误。本书介绍的语义模型怎样才能阻止这样的错误产生呢？

3.18 “是…的孩子”和”是…的双亲”两个属性之间有什么关系？

3.19 考虑属性吃，它的定义域是动物，值域是动物或者植物。假设要定义一个新类食草动物。为这个类给出一个合乎需要的对属性吃的限制。这个限制可以用 RDFS 中的 rdfs: range 来表达吗？

3.20 根据 http://sesame.aduna.biz 提供的 RDF 资料计算一些 RQL 查询结果。

3.21 自己选择一个领域，对其构造 RDF Schema 词汇表，并使用 FRODO RDFViz 可视化

108 工具[⊖]为你的词汇表构建类和属性图。

⊖ <http://www.dfki.uni-kl.de/frodo/RDFSViz/>

第 4 章　网络本体语言 OWL

4.1　引言

深究起来，上一章介绍的 RDF 和 RDF Schema 的表达能力还非常有限；粗略地说，RDF 局限于二元常谓词，RDFS 局限于子类分层和属性分层，以及属性的定义域和值域限定。

然而，万维网联盟(W3C)网络本体工作组[㈠]确定的一些语义网特征用例(characteristic use-cases)所需要的表达能力比 RDF 和 RDF Schema 强得多。

美国和欧洲的许多研究机构意识到需要更为强大的本体建模语言，并联合提出了一个语言 DAML + OIL [㈡](美国提议的 DAML-ONT[㈢]及欧洲提议的 OIL[㈣]的合称)。

DAML + OIL 作为网络本体工作组定义网络本体语言(Web Ontology Language)OWL 的起点，而 OWL 的目标是成为语义网的公认标准本体语言。

本章先从需求角度介绍 OWL 的动机及与 RDFS 的重要联系，然后详细描述 OWL 语言的构成元素。

4.1.1　本体语言需求

本体语言用于对领域模型进行显示的形式化描述。其主要需求是

良定义语法，高效率的推理支持，形式语义，充分的表达能力和表达的方便性。

良定义语法的重要性是显然的，这在程序设计语言领域很清楚，它是机器处理信息的必要条件。前面介绍的所有语言都有良定义语法，基于 RDF 和 RDFS 构建的 DAML + OIL 和 OWL 也都具有类似的语法。

当然，基于 XML 的 RDF 语法是否友好还有疑问，还有其他更适合人类用户的语法(如 OIL 语法)。但这个缺点不是很严重，因为终端用户建立本体可以用编辑工具，更一般地说，是本体开发工具，而不是直接使用 DAML + OIL 或 OWL。

㈠ <http://www.w3.org/2001/sw/WebOnt/>

㈡ <http://www.daml.org/2001/03/daml+oil-index.html>

㈢ <http://www.daml.org/2000/10/daml-ont.html>

㈣ <http://www.ontoknowledge.org/oil/>

形式语义能精确刻画知识的含义。这里，所谓精确的含义是，这种语义并不依据主观直觉，也不会因人(或机器)而异。在数理逻辑中，形式语义的重要性早已人所共知。

形式语义的一种用途是允许人们推导知识。对于本体知识，需要进行下列推理

- **类属关系**。若 x 是类 C 的实例，C 是 D 的子类，则可推出 x 是 D 的实例；
- **类等价**。若类 A 与类 B 等价，类 B 与类 C 等价，则 A 与 C 等价；
- **相容**。假设声明 x 是类 A 的实例，A 既是 $B \cap C$ 的子类，也是 D 的子类，而且 B 和 D 不相交，则产生不相容，因为 A 应是空集，但又有实例 x，这显示本体中有错误；
- **分类**。如果已声明一组特定的属性-值对形成类 A 成员的充分条件，那么，若个体 x 满足这样的条件，则可推出 x 是 A 的实例。

110

语义是推理支持(reasong support)的一个先决条件，诸如上面的推导可以用机器代替手工自动完成。推理支持的重要作用包括

- 检查本体和知识的相容性；
- 检查类间的隐含关系；
- 对实例进行自动分类。

自动推理支持比手工能检查得更多，这对多人参与的大型本体设计或者不同来源本体的融合与共享，是十分有益的。

形式语义和推理支持的实现通常是把本体语言对应到已知的逻辑系统，并使用已有的自动推理机。OWL(部分地)对应于描述逻辑(description logic)，利用现有描述逻辑推理机如 FaCT 和 RACER。各种描述逻辑系统是谓词逻辑的一个具有高效率推理支持的子集。

4.1.2 RDF Schema 表达能力的局限性

RDF 和 RDFS 可以表示某些本体知识，主要建模原语涉及以类型层次组织起来的词汇，包括子类关系和子属性关系、定义域和值域限定以及类实例。但是，还有很多特性不支持，比如：

- **属性的局部辖域**。`rdfs: range` 为一个属性(例如吃)定义的值域是相对于所有类的，无法定义只适用于某些类的值域限制(例如，无法定义牛只吃植物，而其他动物还可能吃肉)。
- **类不相交性**。有时需要表示类的不相交特性，如男性和女性两类不相交。但 RDFS 只能规定类间的子类关系，如女性是人类的子类。
- **类的布尔组合**。有时希望通过对已有类的并、交或补等操作，组合产生新的类，比如定义人类为男性和女性的不相交的并。RDFS 不允许这样的定义。

111

- **基数约束**。有时需要对一个属性不同取值的个数加以约束。如一个人恰好有一个父亲和一个母亲，一门课程至少由一个授课者讲授。RDFS 同样不能表达这样的约束。

- **属性的特殊性质。**有时需要规定属性具有传递性(如属性“大于”)、唯一性(如“母亲”),或定义属性的逆属性(如“吃”和“被吃”)等。

因此,需要一个比 RDFS 更强的语言,能提供这些功能甚至更多功能。设计这样的语言,要在表达能力和高效率推理支持之间进行折中。一般来说,表达能力越强的语言,推理的效率就越低,甚至是不可计算的。这就需要一个兼顾两方面需求的语言,在能够高效率推理的同时又有描述各种本体和知识的充分表达能力。

4.1.3 OWL 与 RDF/RDFS 的兼容性

理想情况下,OWL 应是 RDFS 的一个扩展,以便继续使用 RDFS 中类和属性的含义(rdfs: Class, rdfs: subClassOf 等),并增加一些建模原语提供更强的表达能力。这样的扩展也符合语义网的层次结构(见图 1.3)。

遗憾的是,对 RDFS 的简单扩展不能同时保证表达能力和推理效率。RDFS 有一些强有力的建模原语(见图 3-8),象 rdfs: Class(所有类的类)和 rdf: Property(所有属性的类)这样表达能力很强的构造,如果在继承这些原语的基础上继续扩展,则计算性能是无法保证的。 112

4.1.4 三个 OWL 子语言

似乎不太可能同时满足本体语言的全部需求:既具有高效率的推理支持,又具有由 RDF Schema 与完整逻辑的组合所形成的语言一样强大的表达能力。

这促使网络本体工作组定义了 OWL 的三个不同的子语言,每个子语言在不同层次(程度)上实现整体需求。

OWL Full

完整的 OWL 语言称为 OWL Full,使用了 OWL 的所有原语。它还允许这些原语与 RDF/RDFS 的任意组合,甚至通过原语间的相互作用来改变预定义原语的含义。例如,在 OWL Full 中可以对所有类的类施加基数约束,从而限制任何本体可以表达的类的个数。

OWL Full 的优点是语法和语义上都相对于 RDF 完全向上兼容——任何合法的 RDF 文档都是合法的 OWL Full 文档;任何有效的 RDF/RDFS 推论都是有效的 OWL Full 推论。缺点是 OWL Full 的表达能力过于强大,以致于是不可判定的,从而排除了完备或高效推理支持的任何希望。

OWL DL

为了保证计算效率,OWL Full 的子语言 OWL DL(DL 是描述逻辑的缩写)对 OWL 和 RDF 的构造(constructor)的使用做了限制——本质上不允许构造间的相互施用,从而确保这个子语言对应于一个已经得到充分研究的描述逻辑系统。

这个子语言的优点是保证了高效率推理支持，缺点是不能与 RDF 完全兼容——合法的 RDF 文档一般需通过一些扩展或限制才能成为合法的 OWL DL 文档，但每个合法的 OWL
113 DL 文档都是合法的 RDF 文档。

OWL Lite

OWL Lite 是对 OWL DL 的构造施加了进一步限制的子语言。比如 OWL Lite 没有枚举类、类不相交陈述、任意基数约束等。其优点是容易掌握(对用户)和实现(对开发工具)，缺点当然是表达能力有限。

采用 OWL 的本体开发人员需要考虑使用哪个子语言最符合自己的需要。选择 OWL Lite 还是 OWL DL 取决于用户多大程度上需要 OWL DL 和 OWL Full 提供的表达能力；选择 OWL DL 还是 OWL Full 取决于用户多大程度上需要 RDFS 提供的元建模(metamodeling)机制，如定义类的类和为类赋予属性等。与 OWL DL 相比，OWL Full 的推理支持是难以预料的，因为 OWL Full 的完全实现是不可能的。

这三个子语言之间是严格向上兼容的：

- 合法的 OWL Lite 本体都是合法的 OWL DL 本体；
- 合法的 OWL DL 本体都是合法的 OWL Full 本体；
- 有效的 OWL Lite 推论都是有效的 OWL DL 推论；
- 有效的 OWL DL 推论都是有效的 OWL Full 推论。

OWL 在很大程度上仍然使用 RDF 和 RDFS：

- 所有 OWL 变体都使用 RDF 语法；
- 实例声明和 RDF 一样，使用 RDF 描述和类型信息；
- OWL 的构造，如 `owl: Class`、`owl: DatatypeProperty`、`owl: ObjectProperty` 等，是对 RDF 中对应构造的特殊化。

114 图 4-1 显示了 OWL 和 RDF/RDFS 中一些建模原语间的子类关系。

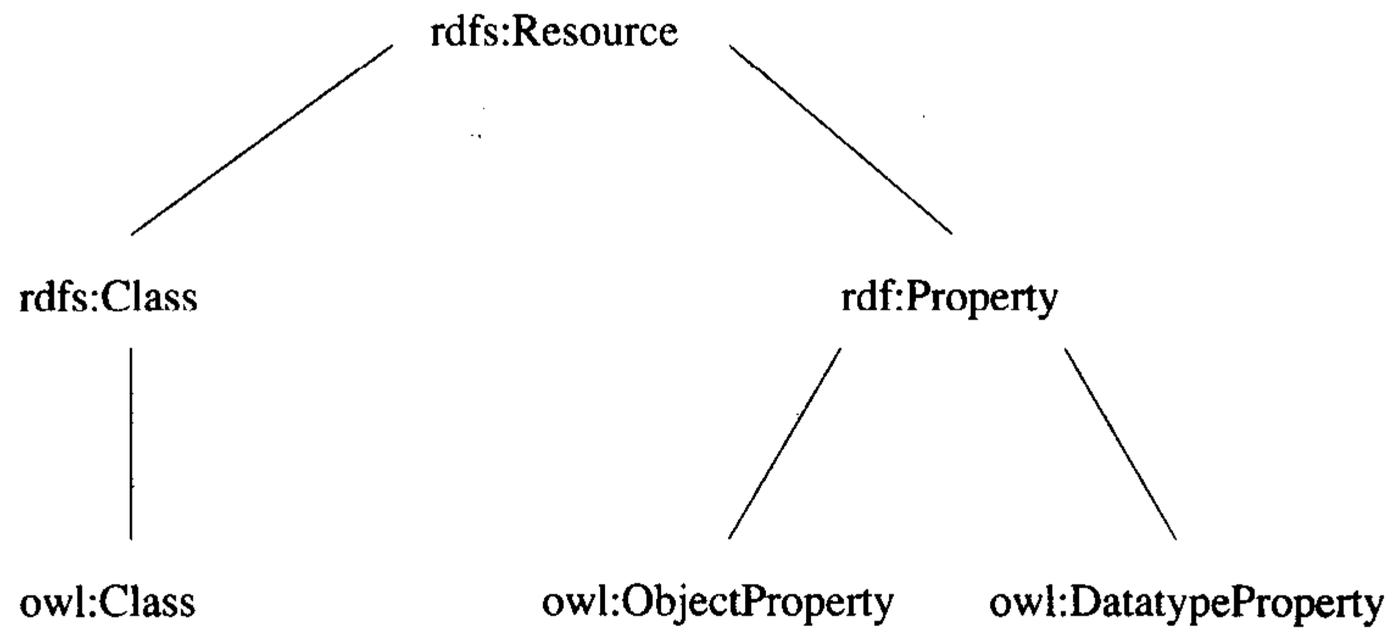


图 4-1 OWL 和 RDF/RDFS 间的子类关系

语义网分层结构(见图 1-3)背后的一个主要动机是希望不同层次的软件通过向下兼容可以得到复用。然而，只有 OWL Full 实现了完全向下兼容(任何掌握 OWL 的处理器都可以正确解释 RDFS 文档)，这是以牺牲可计算性为代价的。

4.2 节较详细地介绍 OWL，4.3 节举例加以说明。部分 OWL 定义可用 OWL 自身书写，见 4.4 节。4.5 节讨论 OWL 不能表达的一些需求，作为未来扩展的目标。

4.2 OWL 语言

4.2.1 语法

OWL 以 RDF 和 RDFS 为基础，使用基于 XML 的 RDF 语法。既然这是 OWL 的基本语法，这里也使用它，尽管 RDF/XML 语法并不易读。所以，也定义了 OWL 的一些其他的语法形式：

- 一种人们更易读懂、但不符合 RDF 惯例的基于 XML 的语法。[⊖] 115
- 语言规范文档中使用的一种抽象语法[⊖]，比 XML 语法或 RDF/XML 语法都更加紧凑和易读。本书附录用此抽象语法列出了本章中的所有 RDF/XML 代码。
- 一种基于流行的统一建模语言(UML)约定的图形化语法，是人们熟悉 OWL 的一条简单途径。

4.2.2 头部

OWL 文档通常称为 OWL 本体或 RDF 文档。一个 OWL 本体的根元素是一个 rdf:RDF 元素，用来指定一系列命名空间：

```
<rdf: RDF
    xmlns: owl ="http: //www. w3. org/2002/07/owl#"
    xmlns: rdf ="http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
    xmlns: rdfs ="http: //www. w3. org/2000/01/rdf-schema#"
    xmlns: xsd ="http: //www. w3. org/2001/XMLSchema#" >
```

一个 OWL 本体可以以一组“簿记”断言(assertions for housekeeping)开始，这些断言聚集在一个 owl: Ontology 元素中，包括注释、版本控制及其他本体导入等。如：

```
<owl: Ontology rdf: about = "" >
  <rdfs: comment >本体例子 </rdfs: comment >
  <owl: priorVersion
```

⊖ 定义见 <http: //www. w3. org/TR/owl-xmlsyntax/ >

⊖ <http: //www. w3. org/TR/owl-semantics/ >

```
        rdf: resource = "http: //www. mydomain. org/uni-ns-old"/ >
    <owl: imports
        rdf: resource = "http: //www. mydomain. org/persons"/ >
    <rdfs: label > 大学本体 </rdfs: label >
</owl: Ontology >
```

这些断言中只有 owl: imports 会影响本体的逻辑含义，它导入其他本体，将其内容
116 作为当前本体的一部分。需要注意的是，命名空间用于消除二义性，导入的本体则提供了可用的定义。通常，每个可用的命名空间都有一个导入元素(import element)，但可以导入附加本体，例如，只提供定义而不引入新名字。

还要注意，owl: imports 具有传递性：若本体 *A* 导入了本体 *B*，*B* 导入了本体 *C*，则本体 *A* 也导入了本体 *C*。

4.2.3 类元素

类的定义使用 owl: Class 元素[⊖]。例如，副教授可以定义如下：

```
<owl: Class rdf: ID = "副教授" >
    <rdfs: subClassOf rdf: resource = "#教师"/ >
</owl: Class >
```

可以用 owl: disjointWith 元素断定副教授与助理教授和教授类都是不相交的。这些断言可以放在上面的定义中，也可以通过 rdf: about 引用其 ID。这种机制是从 RDF 继承而来的。

```
<owl: Class rdf: about = "#副教授" >
    <owl: disjointWith rdf: resource = "#教授"/ >
    <owl: disjointWith rdf: resource = "#助理教授"/ >
</owl: Class >
```

类等价可用 owl：equivalentClass 元素定义：

```
<owl: Class rdf: ID = "教师" >
    <owl: equivalentClass rdf: resource = "#教员"/ >
</owl: Class >
```

有两个预定义的类：owl: Thing 和 owl: Nothing，前者是最一般的类，包含所有
117 东西，后者是空类。每个类都是 owl: Thing 的子类和 owl: Nothing 的父类。

4.2.4 属性元素

OWL 有两种属性：

⊖ owl: Class 是 rdfs: Class 的子类。

- 对象属性，将对象相互关联，例如由…讲授(isTaughtBy)和监督(supervises)是两个对象属性。
- 数据类型属性，将对象与数据类型值相关联，例如电话、标题、年龄等。OWL 没有任何预定义数据类型，也没提供特别的定义机制，但可以使用 XML Schema 提供的数据类型，据此利用语义网的层次特性。

下面是数据属性的一个例子：

```
<owl: DatatypeProperty rdf: ID = "年龄" >
   <rdfs: range rdf: resource = "http: //www. w3. org/2001/XMLSchema
   #nonNegativeInteger"/ >
</owl: DatatypeProperty >
```

用户定义的数据类型一般都收集在一个 XMLS 中，然后可以在 OWL 本体中直接使用。下面是对象属性的一个例子：

```
<owl: ObjectProperty rdf: ID = "由…讲授" >
   <rdfs: domain rdf: resource = "#课程"/ >
   <rdfs: range rdf: resource = "#教师"/ >
   <rdfs: subPropertyOf rdf: resource = "#参与"/ >
</owl: ObjectProperty >
```

可以声明多个定义域和值域。这种情况下将分别取它们的交集。

OWL 可以定义属性的“逆属性”，典型例子是由…讲授和讲授(teaches)：

```
<owl: ObjectProperty rdf: ID = "讲授" >
   <rdfs: range rdf: resource = "#课程"/ >
   <rdfs: domain rdf: resource = "#教师"/ >
   <owl: inverseOf rdf: resource = "#由…讲授"/ >
</owl: ObjectProperty >
```

图 4-2 举例说明了属性和逆属性的关系。其实逆属性的定义域和值域都可以从原属性得到，只要交换定义域与值域。 118

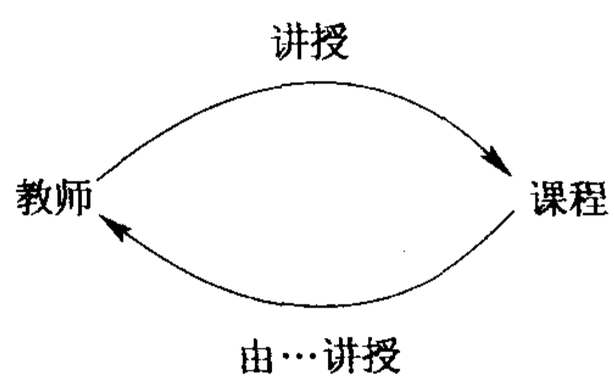


图 4-2 互逆属性

属性等价可用 owl: equivalentProperty 定义：

```
<owl: ObjectProperty rdf: ID = "授课" >
```

```
    <owl: equivalentProperty rdf: resource = "#讲授"/ >
</owl: ObjectProperty >
```

4.2.5 属性约束

用 rdfs: subClassOf 可以规定一个类 C 是另一个类 C' 的子类，于是 C 的所有实例都是 C' 的实例。

若想声明类 C 满足某些条件(即 C 的实例都满足这些条件)，这等价于声明 C 是类 C' 的子类，其中 C' 包含所有满足这些条件的对象。OWL 就是这样做的，注意 C' 通常是匿名的。

下面例子中要求一年级课程只能由教授讲授(根据一种可疑的观点，资历越深教学效果越好)：

```
<owl: Class rdf: about = "#一年级课程" >
   <rdfs: subClassOf >
     <owl: Restriction >
       <owl: onProperty rdf: resource = "#由...讲授"/ >
       <owl: allValuesFrom rdf: resource = "#教授"/ >
     </owl: Restriction >
   </rdfs: subClassOf >
</owl: Class >
```

119 其中 owl: allValuesFrom 用来规定 owl: onProperty 指定属性(这里是由...讲授)的可能取值范围，即要求所有值都只能从指定的类(这里是教授)中选取。

也可以如下声明数学课由 David Billington 教授来教：

```
<owl: Class rdf: about = "#数学课" >
   <rdfs: subClassOf >
     <owl: Restriction >
       <owl: onProperty rdf: resource = "#由...讲授"/ >
       <owl: hasValue rdf: resource = "#949352"/ >
     </owl: Restriction >
   </rdfs: subClassOf >
</owl: Class >
```

根据这个声明，owl: onProperty 指定属性必须取 owl: hasValue 规定的特定值。

也可以规定所有教师至少要教一门本科课程：

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

```
        <owl: someValuesFrom
            rdf: resource = "#本科课程"/ >
      </owl: Restriction >
    </rdfs: subClassOf >
</owl: Class >
```

比较一下 owl: allValuesFrom 和 owl: someValuesFrom。本例中，使用后者要求存在一门本科课程由某教师讲授，当然该教师也可能还讲授研究生课程，这对应于逻辑中的存在量词。对本例来说，也可以使用前者要求一年级课程的每个讲授者都是教授，这对应于逻辑中的全称量词。

通常，元素 owl: Restriction 应包含一个元素 owl: onProperty 及一个或多个约束声明。一种约束声明使用 owl: allValuesFrom、owl: hasValue 或 owl: someValuesFrom 定义被约束属性的取值方式；另一种约束声明定义被约束属性的基数约束。120 如要求每门课程至少由一人来讲授：

```
<owl: Class rdf: about = "#课程" >
 <rdfs: subClassOf >
  <owl: Restriction >
    <owl: onProperty rdf: resource = "#由...讲授"/ >
    <owl: minCardinality
        rdf: datatype = "&xsd; nonNegativeInteger" >
      1
    </owl: minCardinality >
  </owl: Restriction >
 </rdfs: subClassOf >
</owl: Class >
```

注意，必须把文字“1”解释为非负整数(而不是字符串)，另外使用定义在头部的命名空间声明 xsd，指向 XMLS 文档。

如果实际需要，可以规定一个系最少有十个成员，最多有三十个成员：

```
<owl: Class rdf: about = "#系" >
 <rdfs: subClassOf >
  <owl: Restriction >
    <owl: onProperty rdf: resource = "#成员"/ >
    <owl: minCardinality
        rdf: datatype = "&xsd; nonNegativeInteger" >
      10
    </owl: minCardinality >
  </owl: Restriction >
</rdfs: subClassOf >
<rdfs: subClassOf >
```

```
<owl: Restriction>
  <owl: onProperty rdf: resource = "#成员"/>
  <owl: maxCardinality
        rdf: datatype = "&xsd; nonNegativeInteger">
     30
  </owl: maxCardinality>
</owl: Restriction>
</rdfs: subClassOf>
</owl: Class>
```

也可以规定一个具体的数字，如一个博士生必须有两位导师。这可以通过对 owl: minCardinality 和 owl: maxCardinality 赋相同的值得到，更方便的方法是使用 OWL 提供的 owl: cardinality。

小结一下。owl: Restriction 定义了一个没有 ID 的匿名类，它不是由 owl: Class 定义的，并且局部有效——只能在该约束出现的地方使用。在说到类时，记住有两层含义：①由 owl: Class 定义的具有 ID 的类；②由满足一组约束条件的个体组成的局部匿名类，即由其他类的组合得到的匿名类。类的组合有时也叫类表达式。

4.2.6 特殊性质

属性元素的某些性质可以直接定义：

owl: TransitiveProperty 定义传递性，例如“优于”、“高于”、“祖先”等关系都是具有传递性的；

owl: SymmetricProperty 定义对称性，例如“同年级”、“同胞兄弟”等关系具有对称性；

owl: FunctionalProperty 定义函数性，即规定一个属性对任何对象最多只能取一个值，例如“年龄”、“身高”或“direct Superviser”等。

owl: InverseFunctionalProperty 定义逆函数性，即规定一个属性对不同的对象不能取相同的值，例如“身份证号”(每人的身份证号都是不同的)。

用语法表达这些性质的一个例子如下：

```
<owl: ObjectProperty rdf: ID = "同年级">
  <rdf: type rdf: resource = "&owl; TransitiveProperty" />
  <rdf: type rdf: resource = "&owl; SymmetricProperty" />
  <rdfs: domain rdf: resource = "#学生" />
  <rdfs: range rdf: resource = "#学生" />
</owl: ObjectProperty>
```

4.2.7 布尔组合

无论是由 owl: Class 还是由类表达式定义的类，都可以谈论其布尔组合(并、交、补)。例如，可以如下描述课程和教工不相交：

```
<owl: Class rdf: about = "#课程" >
  <rdfs: subClassOf >
    <owl: Class >
      <owl: complementOf rdf: resource = "#教工"/ >
    </owl: Class >
  </rdfs: subClassOf >
</owl: Class >
```

也就是说，每门课程都是教工补集的一个实例，即课程不是教工。当然，也可以用 owl: disjointWith 达到相同的目的。

类的并由 owl: unionOf 定义：

```
<owl: Class rdf: ID = "大学人员" >
  <owl: unionOf rdf: parseType = "Collection" >
    <owl: Class rdf: about = "#教工"/ >
    <owl: Class rdf: about = "#学生"/ >
  </owl: unionOf >
</owl: Class >
```

上面定义的这个新类不是教工和学生并集的子类，而是等价于它们的并，即定义了类的等价。此处没有规定教工和学生不相交，教工也可能同时是学生。

类的交由 owl: intersectionOf 定义：

```
<owl: Class rdf: ID = "计算机系教师" >
  <owl: intersectionOf rdf: parseType = "Collection" >
    <owl: Class rdf: about = "#教师"/ >
    <owl: Restriction >
      <owl: onProperty rdf: resource = "#属于"/ >
      <owl: hasValue rdf: resource = "#计算机系"/ >
    </owl: Restriction >
  </owl: intersectionOf >
</owl: Class >
```

123

上面定义了两个类的交，其中一个是匿名类——所有属于计算机系的对象，它与教师的交集即计算机系教师。

布尔组合可以任意嵌套。下面的例子把行政人员定义为既不是教师也不是技术支撑人员的教工：

```
<owl: Class rdf: ID="行政人员">
 <owl: intersectionOf rdf: parseType="Collection">
  <owl: Class rdf: about="#教工"/>
  <owl: Class>
   <owl: complementOf>
    <owl: Class>
     <owl: unionOf rdf: parseType="Collection">
      <owl: Class rdf: about="#教师"/>
      <owl: Class rdf: about="#技术支撑人员"/>
     </owl: unionOf>
    </owl: Class>
   </owl: complementOf>
  </owl: Class>
 </owl: intersectionOf>
</owl: Class>
```

4.2.8 枚举

一个枚举是一个 owl: one Of 元素，通过列举所有成员定义一个类：

```
<owl: Class rdf: ID="星期">
 <owl: oneOf rdf: parseType="Collection">
  <owl: Thing rdf: about="#星期一"/>
  <owl: Thing rdf: about="#星期二"/>
  <owl: Thing rdf: about="#星期三"/>
  <owl: Thing rdf: about="#星期四"/>
  <owl: Thing rdf: about="#星期五"/>
  <owl: Thing rdf: about="#星期六"/>
  <owl: Thing rdf: about="#星期日"/>
 </owl: oneOf>
</owl: Class>
```


4.2.9 实例

类的实例声明和 RDF 一样：

```
<rdf: Description rdf: ID="949352">
 <rdf: type rdf: resource="#教师"/>
</rdf: Description>
```

这等价于

```
<教师 rdf: ID="949352"/>
```

也可以更详细一些，例如

```
<教师 rdf: ID = "949352" >
 <uni: 年龄 rdf: datatype = "&xsd; integer" >39 </uni:  年龄 >
</教师 >
```

与典型的数据库系统不同，OWL 不采纳命名唯一性假设(unique-names assumption)，因为具有不同名字或 ID 的两个实例不一定是不同的。例如，假设规定每门课程最多由一位教工讲授

```
<owl: ObjectProperty rdf: ID = "由…讲授" >
 <rdf: type rdf: resource = "&owl; FunctionalProperty" / >
</owl: ObjectProperty >
```

随后又规定某一门课由两位教工讲授：

```
<course rdf: ID = "CIT1111" >
 <由…讲授 rdf: resource = "#949318"/ >
 <由…讲授 rdf: resource = "#949352"/ >
</course >
```

这不会导致 OWL 推理机报错，相反，系统会正确推出资源“949318”和“949352”等价。要确保不同个体确实是不同的，必须明确地陈述它们是不相等的：

```
<授课者 rdf: ID = "949318" >
 <owl: differentFrom rdf: resource = "#949352"/ >
</授课者 >
```

经常需要做出这样的不相等(inequality)陈述。若要陈述大量个体不相等，则相应陈述的数量将爆炸性增加。为此，OWL 提供了一个简略表示法，把互不相等的个体放在一个列表中：

125

```
<owl: AllDifferent >
 <owl: distinctMembers rdf: parseType = "Collection" >
  <授课者 rdf: about = "#949318"/ >
  <授课者 rdf: about = "#949352"/ >
  <授课者 rdf: about = "#949111"/ >
 </owl: distinctMembers >
</owl: AllDifferent >
```

注意，owl: distinctMembers 只能和 owl: allDifferent 组合使用。

4.2.10 数据类型

虽然 XMLS 提供了一种机制支持用户自定义数据类型(比如成人年龄的类型是大于 18 的所有整数、所有数字开头的字符串等)，这些派生数据类型不能在 OWL 中使用。事实上，并非所有 XMLS 内嵌(built-in)数据类型都可以在 OWL 中使用。OWL 参考文档列出了

所有可以在 OWL 中使用的 XMLS 数据类型，包括一些最常用的数据类型，如字符串、整数、布尔类型、时间和日期等。

4.2.11 版本控制信息

前面提到 owl: priorVersion 作为头部信息的一部分，指现有本体的先前版本。这个信息没有形式化的模型理论语义，但用户或程序可用它来管理本体。

除了 owl: priorVersion 之外，OWL 还有三个陈述用来指明版本信息。所有这些信息都没有形式化语义。

owl: versionInfo 通常包含一个说明当前版本信息的字符串，例如 RCS/CVS[⊖]关键词。

owl: backwardCompatibleWith 包含对另一个本体的引用，并将被引本体指定为当前本体的一个向后兼容的(backward-compatible)先前版本。特别地，这也表明先前版本中的所有标识符在新版本中都有相同的语义解释。这就给文档作者一个提示，可以安全地变
126 更其文档为新版本(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)。

owl: incompatibleWith 指明被引本体是当前本体的一个不向后兼容的先前版本。这用于表明：本体作者必须在检查文档是否需要修改之后才能升级使用新版本。

4.2.12 OWL 分层

上一节讨论了 OWL 的所有语言构造，现在说明子语言(OWL Full，OWL DL，OWL Lite)可以使用 OWL 的哪功能。

OWL Full

OWL Full 中，OWL 的任何语言构造都可以任意组合使用，只要组合结果是合法的 RDF 语句。

OWL DL

为了利用描述逻辑的形式化基础和高效率计算性能，任何 OWL DL 本体必须遵守下列规定：

- 词汇划分。任何资源只能是下面中的一种：类、数据类型、数据类型属性、对象属性、个体、数据值或内嵌词汇的部分。例如，类不能同时是个体；一个属性不能既有来自数据类型的值又有来自类的值(这会使它既是数据类型属性又

⊖ RCS/CVS 即版本控制系统。——译者注

是对象属性)。

- 显式划分。不仅所有的资源必须进行词汇划分(如上面所规定的那样),而且这种划分必须明确陈述。例如,假设一个本体包含下面的信息:

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

127

这已经蕴涵 C2 是一个类(由 rdfs: subClassOf 的值域限定),但 OWL DL 要求还必须有下面的显式陈述:

```
<owl: Class rdf: ID = "C2"/ >
```

- 属性分离。根据第一条规定,对象属性的集合与数据类型属性的集合不相交,这意味着不能对数据类型属性施加下列规定: owl: inverseOf, owl: FunctionalProperty, owl: InverseFunctionalProperty 或 owl: SymmetricProperty。
- 传递属性无基数约束。不能对传递属性进行基数约束。
- 匿名类限制。匿名类只允许作为 owl: equivalentClass 或 owl: disjointWith 的定义域或值域,或作为 rdfs: subClassOf 的值域(不能作为定义域)。 128

OWL Lite

OWL Lite 本体一定是 OWL DL 本体,并作了进一步限制如下:

- 不允许使用下列构造: owl: oneOf, owl: disjointWith, owl: unionOf, owl: complementOf 和 owl: hasValue。

基数陈述(准确基数,最多个数,最少个数)只能取 0 或 1,不能是任意的非负整数。

- owl: equivalentClass 陈述只能用于类标识符,不能用于匿名类。

4.3 例子

4.3.1 一个非洲野生生物本体

本例是一个描述非洲野生生物的本体,图 4-3 显示了基类(basic classes)与子类的关系。注意子类信息只是该本体中的部分信息,整个图要大得多。图 4-4 给出了陈述"树枝是树的一部分"的图表示。

这个本体还包含用 rdfs: comment 书写的注释信息。

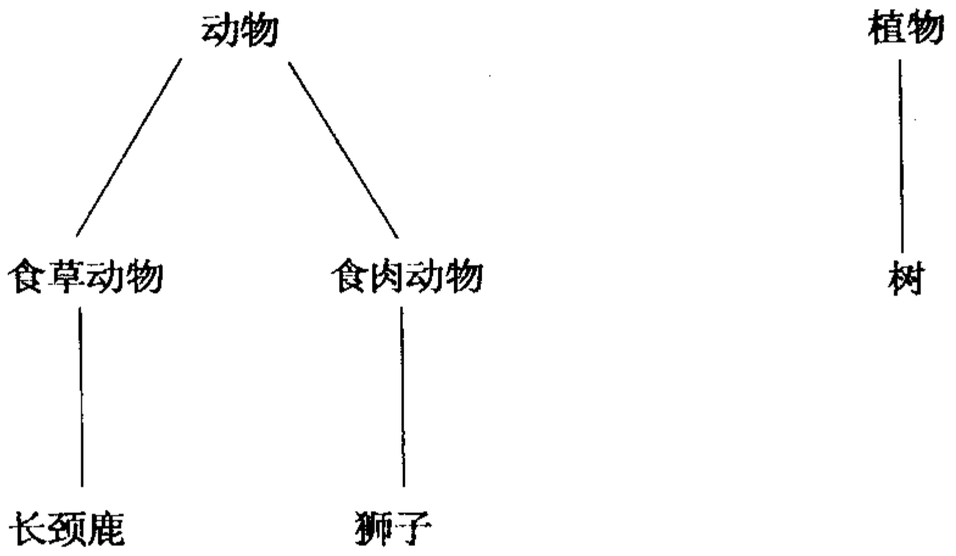


图 4-3　非洲野生生物本体的类和子类

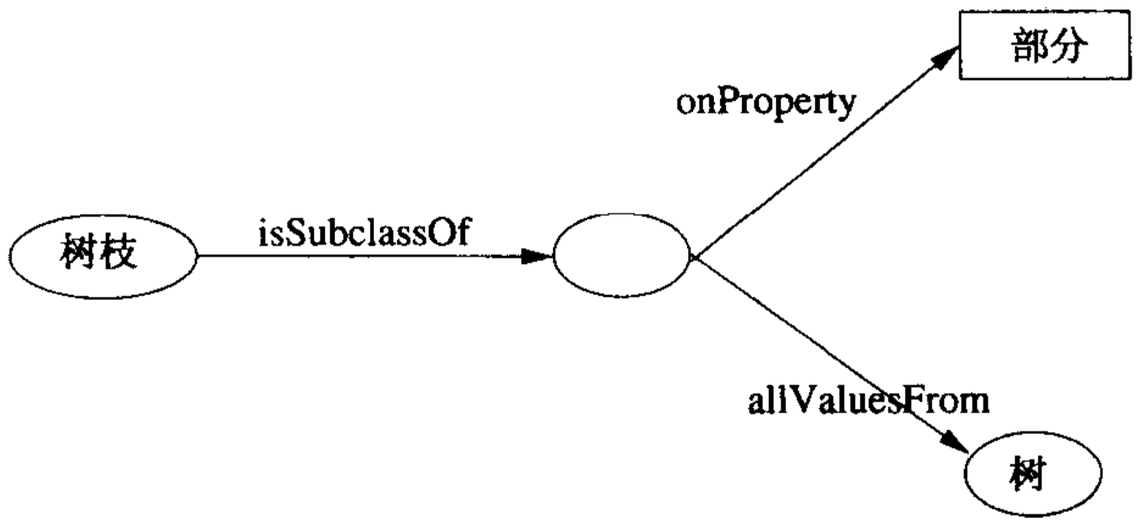


图 4-4　树枝是树的一部分

```
<rdf: RDF
 xmlns: rdf ="http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
 xmlns: rdfs ="http: //www. w3. org/2000/01/rdf-schema#"
129  xmlns: owl ="http: //www. w3. org/2002/07/owl#" >

<owl: Ontology rdf: about ="xml: base"/ >

<owl: Class rdf: ID ="动物" >
 <rdfs: comment >动物类 </rdfs: comment >
</owl: Class >

<owl: Class rdf: ID ="植物" >
 <rdfs: comment >植物和动物不相交
 </rdfs: comment >
 <owl: disjointWith rdf: resource ="#动物"/ >
</owl: Class >

<owl: Class rdf: ID ="树" >
 <rdfs: comment >树是植物 </rdfs: comment >
 <rdfs: subClassOf rdf: resource ="#植物"/ >
</owl: Class >
```

```
<owl: Class rdf: ID="树枝">
 <rdfs: comment>树枝是树的一部分</rdfs: comment>
 <rdfs: subClassOf>
  <owl: Restriction>
  <owl: onProperty rdf: resource="#部分"/>
  <owl: allValuesFrom rdf: resource="#树"/>
 </owl: Restriction>
 </rdfs: subClassOf>
</owl: Class>

<owl: Class rdf: ID="树叶">
<rdfs: comment>树叶是树枝的一部分</rdfs: comment>
 <rdfs: subClassOf>
  <owl: Restriction>
   <owl: onProperty rdf: resource="#部分"/>
   <owl: allValuesFrom rdf: resource="#树枝"/>
  </owl: Restriction>
 </rdfs: subClassOf>
</owl: Class>

<owl: Class rdf: ID="食草动物">
 <rdfs: comment>食草动物是只吃植物(或植物的一部分)的动物
</rdfs: comment>                                                    130
  <owl: intersectionOf rdf: parseType="Collection">
  <owl: Class rdf: about="#动物"/>
   <owl: Restriction>
   <owl: onProperty rdf: resource="#吃"/>
    <owl: allValuesFrom>
     <owl: Class>
      <owl: unionOf rdf: parseType="Collection">
       <owl: Class rdf: about="#植物"/>
       <owl: Restriction>
      <owl: onProperty rdf: resource="#部分"/>
     <owl: allValuesFrom rdf: resource="#植物"/>
     </owl: Restriction>
    </owl: unionOf>
     </owl: Class>
    </owl: allValuesFrom>
   </owl: Restriction>
 </owl: intersectionOf>
</owl: Class>

<owl: Class rdf: ID="食肉动物">
 <rdfs: comment>食肉动物吃动物(也可以吃植物)
 </rdfs: comment>
```

```
    <owl: intersectionOf rdf: parseType = "Collection" >
      <owl: Class rdf: about = "#动物" / >
      <owl: Restriction >
        <owl: onProperty rdf: resource = "#吃" / >
        <owl: someValuesFrom rdf: resource = "#动物" / >
      </owl: Restriction >
    </owl: intersectionOf >
  </owl: Class >

  <owl: Class rdf: ID = "长颈鹿" >
    <rdfs: comment > 长颈鹿是食草动物，只吃树叶 </rdfs: comment >
    <rdfs: subClassOf rdf: resource = "#食草动物" / >
    <rdfs: subClassOf >
131   <owl: Restriction >
        <owl: onProperty rdf: resource = "#吃" / >
        <owl: allValuesFrom rdf: resource = "#树叶" / >
      </owl: Restriction >
    </rdfs: subClassOf >
  </owl: Class >

  <owl: Class rdf: ID = "狮子" >
    <rdfs: comment > 狮子只吃食草动物 </rdfs: comment >
    <rdfs: subClassOf rdf: resource = "#食肉动物" / >
    <rdfs: subClassOf >
      <owl: Restriction >
        <owl: onProperty rdf: resource = "#吃" / >
        <owl: allValuesFrom rdf: resource = "#食草动物" / >
      </owl: Restriction >
    </rdfs: subClassOf >
  </owl: Class >

  <owl: Class rdf: ID = "可口植物" >
    <rdfs: comment > 食草食肉动物都吃的植物 </rdfs: comment >
    <rdfs: subClassOf rdf: resource = "#植物" / >
    <rdfs: subClassOf >
      <owl: Restriction >
        <owl: onProperty rdf: resource = "#被吃" / >
        <owl: someValuesFrom >
          <owl: Class rdf: about = "#食草动物" / >
        </owl: someValuesFrom >
      </owl: Restriction >
    </rdfs: subClassOf >
    <rdfs: subClassOf >
      <owl: Restriction >
        <owl: onProperty rdf: resource = "#被吃" / >
```

```
      <owl: someValuesFrom >
        <owl: Class rdf: about ="#食肉动物"/ >
      </owl: someValuesFrom >
    </owl: Restriction >
  </rdfs: subClassOf >
</owl: Class >
```

132

```
<owl: TransitiveProperty rdf: ID ="部分"/ >

<owl: ObjectProperty rdf: ID ="吃" >
  <rdfs: domain rdf: resource ="#动物"/ >
</owl: ObjectProperty >

<owl: ObjectProperty rdf: ID ="被吃" >
  <owl: inverseOf rdf: resource ="#吃"/ >
</owl: ObjectProperty >

</rdf: RDF >
```

133

4.3.2 一个打印机本体

图 4-5 表示本例中类和子类的关系。

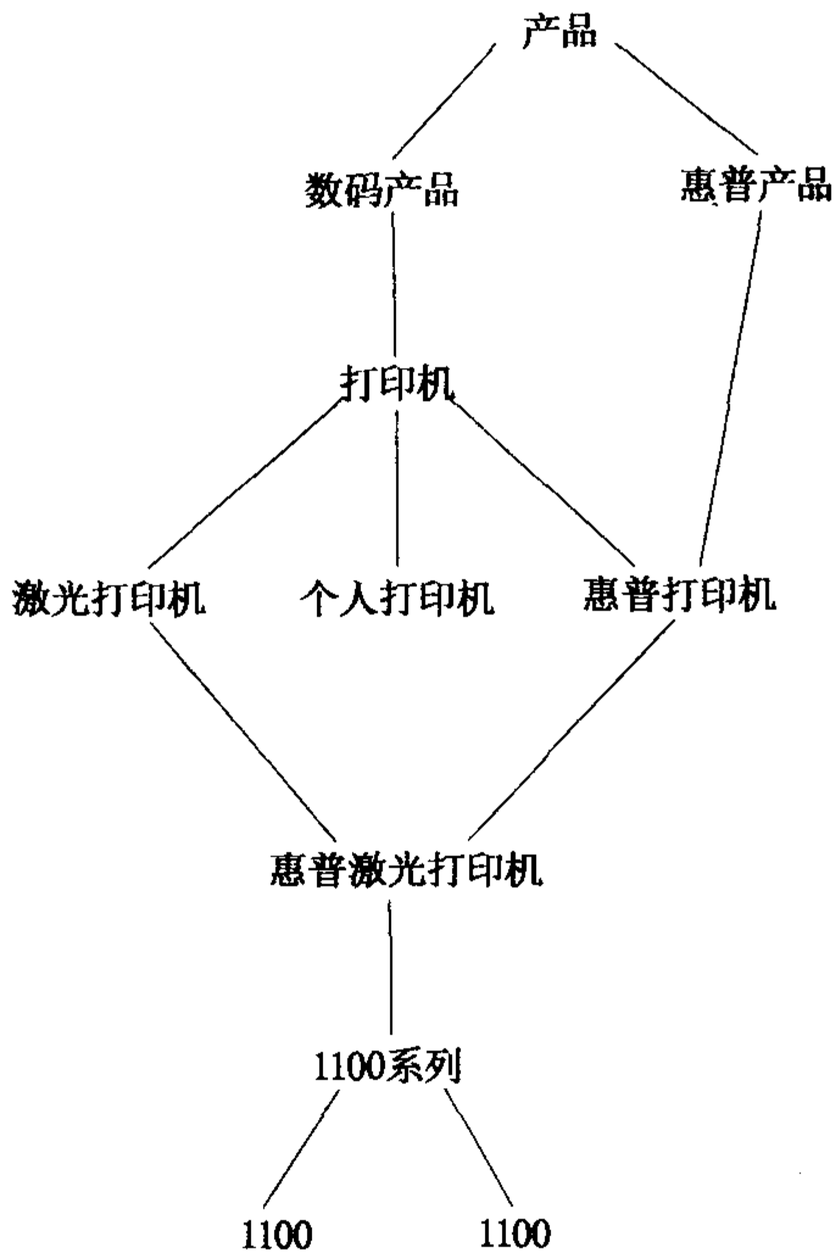


图 4-5 打印机本体的类与子类

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

<rdf: RDF
  xmlns: rdf = "http: //www. w3. org/1999/02/22-rdf-syntax-ns#"
  xmlns: rdfs = "http: //www. w3. org/2000/01/rdf-schema#"
  xmlns: xsd = "http: //www. w3. org/2001/XMLSchema#"
  xmlns: owl = "http: //www. w3. org/2002/07/owl#"
  xmlns = "http: //www. cs. vu. nl/ ~ frankh/spool/printer. owl#" >

<owl: Ontology rdf: about = "" >
 <owl: versionInfo >
  My example version 1. 2, 17 October 2002
 </owl: versionInfo >
</owl: Ontology >

<owl: Class rdf: ID = "产品" >
 <rdfs: comment > 产品类 </rdfs: comment >
</owl: Class >

<owl: Class rdf: ID = "数码产品" >
 <rdfs: comment > 打印和数字图像设备形成产品的子类 </rdfs: comment >
 <rdfs: label > 设备 </rdfs: label >
 <rdfs: subClassOf rdf: resource = "#产品" / >
</owl: Class >

<owl: Class rdf: ID = "惠普产品" >
 <rdfs: comment > 由惠普公司生产的产品 </rdfs: comment >
 <owl: intersectionOf rdf: parseType = "Collection" >
  <owl: Class rdf: about = "#产品" / >
  <owl: Restriction >
    <owl: onProperty rdf: resource = "#生产商" / >
    <owl: hasValue rdf: datatype = "&xsd; string" >
      惠普公司
    </owl: hasValue >
  </owl: Restriction >
 </owl: intersectionOf >
</owl: Class >

<owl: Class rdf: ID = "打印机" >
 <rdfs: comment > 打印机是打印和数字图像设备 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#数码产品" / >
</owl: Class >
```

134

```
<owl: Class rdf: ID = "个人打印机" >
 <rdfs: comment > 个人打印机形成打印机的子类 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#打印机"/ >
</owl: Class >

<owl: Class rdf: ID = "惠普打印机" >
 <rdfs: comment > 惠普打印机既是惠普产品也是打印机 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#打印机"/ >
 <rdfs: subClassOf rdf: resource = "#惠普产品"/ >
</owl: Class >

<owl: Class rdf: ID = "激光打印机" >
 <rdfs: comment > 使用了激光喷墨技术的打印机 </rdfs: comment >
 <owl: intersectionOf rdf: parseType = "Collection" >
  <owl: Class rdf: about = "#打印机"/ >
  <owl: Restriction >
   <owl: onProperty rdf: resource = "#打印技术"/ >
   <owl: hasValue rdf: datatype = "&xsd; string" >
    激光喷墨
   </owl: hasValue >
  </owl: Restriction >
 </owl: intersectionOf >
</owl: Class >

<owl: Class rdf: ID = "惠普激光打印机" >
 <rdfs: comment > 惠普激光打印机既是惠普产品又是激光打印机 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#激光打印机"/ >
 <rdfs: subClassOf rdf: resource = "#惠普打印机"/ >
</owl: Class >

<owl: Class rdf: ID = "1100 系列" >
 <rdfs: comment >
 惠普 1100 系列激光打印机每分钟打印 8 页(8ppm), 分辨率为 600dpi
 </rdfs: comment >
 <rdfs: subClassOf rdf: resource = "#惠普激光打印机"/ >
 <rdfs: subClassOf >
  <owl: Restriction >
   <owl: onProperty rdf: resource = "#打印速度"/ >
   <owl: hasValue rdf: datatype = "&xsd; string" >
    8ppm
   </owl: hasValue >
  </owl: Restriction >
 </rdfs: subClassOf >
```

135

```
<rdfs: subClassOf >
 <owl: Restriction >
  <owl: onProperty rdf: resource ="#分辨率"/ >
  <owl: hasValue rdf: datatype ="&xsd; string" >
    600dpi
  </owl: hasValue >
 </owl: Restriction >
</rdfs: subClassOf >
</owl: Class >

<owl: Class rdf: ID ="1100se" >
 <rdfs: comment >1100se 打印机属 1100 系列，卖 450 美元 </rdfs: comment >
 <rdfs: subClassOf rdf: resource ="#1100series"/ >
 <rdfs: subClassOf >
  <owl: Restriction >
   <owl: onProperty rdf: resource ="#价格"/ >
   <owl: hasValue rdf: datatype ="&xsd; integer" >
     450
   </owl: hasValue >
  </owl: Restriction >
 </rdfs: subClassOf >
</owl: Class >

<owl: Class rdf: ID ="1100xi" >
 <rdfs: comment >1100xi 打印机属 1100 系列，卖 350 美元 </rdfs: comment >
 <rdfs: subClassOf rdf: resource ="#1100series"/ >
 <rdfs: subClassOf >
  <owl: Restriction >
   <owl: onProperty rdf: resource ="#价格"/ >
   <owl: hasValue rdf: datatype ="&xsd; integer" >
     350
   </owl: hasValue >
  </owl: Restriction >
 </rdfs: subClassOf >
</owl: Class >

<owl: DatatypeProperty rdf: ID ="生产商" >
 <rdfs: domain rdf: resource ="#产品"/ >
 <rdfs: range rdf: resource ="&xsd; string"/ >
</owl: DatatypeProperty >

<owl: DatatypeProperty rdf: ID ="价格" >
 <rdfs: domain rdf: resource ="#产品"/ >
 <rdfs: range rdf: resource ="&xsd; nonNegativeInteger"/ >
```

```
</owl: DatatypeProperty>

<owl: DatatypeProperty rdf: ID="打印技术">
 <rdfs: domain rdf: resource="#打印机"/>
 <rdfs: range rdf: resource="&xsd; string"/>
</owl: DatatypeProperty>

<owl: DatatypeProperty rdf: ID="分辨率">
 <rdfs: domain rdf: resource="#打印机"/>
 <rdfs: range rdf: resource="&xsd; string"/>
</owl: DatatypeProperty>

<owl: DatatypeProperty rdf: ID="打印速度">
 <rdfs: domain rdf: resource="#打印机"/>
 <rdfs: range rdf: resource="&xsd; string"/>
</owl: DatatypeProperty>

</rdf: RDF>
```

这个本体说明树中同一层的同胞兄弟可以相交，例如个人打印机可以是惠普打印机或激光打印机，尽管这三类都是打印机的子类。

4.4 OWL 自描述

下面用 OWL 自身给出 OWL 的一些定义，完整的定义可以在网上找到(参见推荐读物)。同时，对 OWL 的一些前文未提及的方面加以解释。

4.4.1 命名空间

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

<! DOCTYPE owl [
  <! 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
 xml: base ="http: //www.w3.org/2002/07/owl"
 xmlns ="&owl;"
 xmlns: owl ="&owl;"
 xmlns: rdf ="&rdf;"
 xmlns: rdfs ="&rdfs;"
 xmlns: dc ="http: //purl.org/dc/elements/1.1/">
```

当前文档(定义这个 OWL)的 URI 被指定为默认命名空间。因此，下文可以不使用前缀 owl:。此外，注意 XML 实体定义的使用，出现在属性值中的 URL 可以缩写。

138

4.4.2 元类

所有 OWL 类组成的类是所有 RDFS 类组成的类的一个子类：

```
<rdfs: Class rdf: ID="Class">
    <rdfs: label>Class</rdfs: label>
    <rdfs: comment>The class of all OWL classes</rdfs: comment>
    <rdfs: subClassOf rdf: resource="&rdfs; Class"/>
</rdfs: Class>
```

Thing 是 OWL 中最一般的对象类，而 Nothing 是最特殊的类，即空对象类。下面的关系成立：

$$Thing = Nothing \cup Nothing^{c}$$

$$Nothing = Thing^{c} = Nothing^{c} \cap Nothing^{cc} = \varphi$$

```
<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>
```

4.4.3 类等价

由 owl: EquivalentClass 表示的类等价隐含着一个子类关系，它总是出现在两个类之间。与之相似的是属性等价 owl: EquivalentProperty。不相交陈述也只能作用于两个类之间。

```
<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"/>
```

139

```
</rdf: Property>

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

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

可以对任意事物陈述它们相等或不相等；在 OWL Full 中，这种相等或不相等陈述也可以应用于类。[⊖] owl: sameAs 是 owl: sameIndividualAs 的同义词。

```
<rdf: Property rdf: ID="sameIndividualAs">
   <rdfs: label>sameIndividualAs</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>

<rdf: Property rdf: ID="sameAs">
   <rdfs: label>sameAs</rdfs: label>
   <EquivalentProperty rdf: resource="#sameIndividualAs"/>
</rdf: Property>
```

owl: distinctMembers 只能用于 owl: AllDifferent:

```
<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"/>
```

140

⊖ 注意“相等/不相等”与“等价/不等价”之间的区别，后者只能应用于两个类之间。——译者注

```
</rdf: Property>
```

4.4.4 类的组合

假定一个列表包含一组类表达式，owl: unionOf 根据该列表构建一个类。

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

owl: intersectionOf 和 owl: oneOf 类似，但假定列表所含的是个体。owl: complementOf 则用一个单独的类定义一个新类：

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

4.4.5 类属性约束

OWL 中的约束用来定义由满足给定条件的对象组成的类：

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

所有下列属性只能出现在约束定义中，它们的定义域都是 owl: Restriction，但值域有所不同：

```
<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>
```

141

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

owl: maxCardinality 和 owl: cardinality 的定义类似于 owl: minCardinality, owl: someValuesFrom 的定义则类似于 owl: allValuesFrom。

还需注意，owl: onProperty 允许对对象属性或数据类型属性加以约束。因此，施加约束的属性(restricting properties)如 owl: allValuesFrom 的值域不是 owl: Class(对象类)，而是更一般的 rdfs: Class。

属性

owl: ObjectProperty 是 rdf: Property 的特殊情况。owl: DatatypeProperty 也类似。

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

传递性 owl: TransitiveProperty 只能用于对象属性。对称性、函数性、可逆性等也类似。

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

142

逆属性 owl: inverseOf 作用于两个对象属性。

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

下面这些虽然没有在在线参考文献中介绍，但也应该是正确的：

```
<TransitiveProperty rdf: ID = "&rdfs; subClassOf"/ >
```

```
<TransitiveProperty    rdf: ID="&rdfs; subProperty/>

<TransitiveProperty    rdf: ID="EquivalentClass"/>
<SymmetricProperty     rdf: ID="EquivalentClass"/>

<SymmetricProperty     rdf: ID="disjointWith"/>

<TransitiveProperty    rdf: ID="EquivalentProperty"/>
<SymmetricProperty     rdf: ID="EquivalentProperty"/>

<TransitiveProperty    rdf: ID="sameIndividualAs"/>
<SymmetricProperty     rdf: ID="sameIndividualAs"/>

<SymmetricProperty     rdf: ID="differentFrom"/>

<SymmetricProperty     rdf: ID="complementOf"/>
<rdf: Property         rdf: about="complementOf">
<rdfs: subPropertyOf   rdf: resource="disjointWith"/>
</rdf: Property>

<rdf: Property         rdf: about="cardinality">
  <rdfs: subPropertyOfrdf: resource="mincardinality"/>
  <rdfs: subPropertyOfrdf: resource="maxcardinality"/>
</rdf: Property>

143  <SymmetricProperty        rdf: ID="inverseOf"/>
     <rdf: Property        rdf: about="inverseOf">
       <inverseOf          rdf: resource="inverseOf"/>
     </rdf: Property>
```

虽然这里用OWL自身刻画了OWL的部分语言，但语义并不完整。所以，单独给出一个语义规范(如由OWL标准给出的那样)是有必要的。

4.5 未来扩展

显然，OWL不会是语义网的最终本体语言。在OWL的需求文档中已经确定了一些新的特性，还有很多正在讨论之中。本节简要列出少量对OWL可能的扩展和改进。

4.5.1 模块与导入

导入由其他人定义的本体在语义网应用中会非常普遍。然而，OWL对本体导入的支持还很弱，只允许导入指定位置的完整本体。即使用户只需要本体的一小部分，也必须导入整个本体。程序设计语言里的模块化结构基于信息隐藏的思想：通过外部接口能够调用一

个模块的某种功能，但不需要知道该功能是如何实现的。什么是对应的本体信息隐藏概念以及基于这一概念的本体导入机制，仍有待研究。

4.5.2 默认值

很多实用知识表示系统将继承值作为默认值，允许继承值被分层结构中更特殊的类覆盖(overridden)。这虽然被广泛使用，但在如何对默认值的非单调行为进行正确形式化的问题上，尚未达成一致。 144

4.5.3 封闭世界假设

目前的OWL语义采用了基于开放世界假设(open-world assumption)的标准逻辑模型，这个假设的含义可以简单地解释为：不能仅根据无法证明一个语句为假而断定它为真。显然，对于巨大和部分可知的万维网，这个假设无疑是正确的。然而，与之相反的封闭世界假设(只要无法证明一个语句为假，则断定该语句是真的)在某些应用中也是有用的。封闭世界假设与默认概念密切相关，会导致同样的非单调行为。因此，OWL不包含封闭世界假设。

4.5.4 唯一名假设

典型的数据库应用假定，名字不同的个体是不同的个体。OWL遵循通常的逻辑规范，不支持这种假设。如果两个个体(或类、属性)名字不同，仍然可以通过推理得出它们是相同的。因为采用了开放世界假设，万维网采用非唯一名假设似乎是合理的。但与上面的情况类似，在某些应用中唯一名假设也是有用的。更微妙的是，用户可能希望区分本体的不同部分，某些部分采用这个假设，而其他部分则不采用。

4.5.5 过程附加

知识表示中规定术语含义的一种常见做法是，附加一段代码，通过它的执行计算术语的含义，而不是在语言中显式给出术语的定义(如在OWL中那样)。但是，这种做法还不能很好地与形式语义整合在同一个系统中，故而没有被OWL采用。

4.5.6 属性链接规则

如前所述，由于可判定性的要求，OWL目前不允许属性的复合，但在许多应用中属性的复合是一种有用的操作。更一般地，甚至希望将属性定义为包含其他属性的一般规则(Horn规则或其他形式的规则)。基于规则的知识表示与描述逻辑风格的知识表示的集成是一个目前非常活跃的研究课题。 145

此处涉及的一些内容(如规则、非单调性)将在第5章加以讨论。在附录中，使用一种

更加紧凑和易读的抽象语法重新描述本章中的所有 RDF/XML 代码。

4.6 小结

- OWL 是网络本体的建议标准，用于描述知识的语义以便供机器处理。
- OWL 以 RDF 和 RDFS 为基础：使用基于 XML 的 RDF 语法，用 RDF 描述定义实例，使用了大部分 RDFS 建模原语。
- 借助 OWL 到逻辑的映射提供了形式语义和推理支持，为此使用了谓词逻辑和描述逻辑。

尽管 OWL 为实践中的运用提供了足够丰富的内容，它仍在继续扩展之中，进一步的逻辑特性包括规则将被引进。

推荐读物

下面是 OWL 的主要参考文献：

- D. McGuinness and F van Harmelen, eds. OWL Web Ontology Language Overview. August 18, 2003. <http://www.w3.org/TR/owl-features/>.

- M. Dean and G. Schreiber, eds. F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, L. Stein, *OWL Web Ontology Language Reference*. August 18, 2003. <http://www.w3.org/TR/owl-ref/>.

- M. Smith, C. Welty, and D. McGuinness, eds. *OWL Web Ontology Language: Guide*. August 18, 2003. <http://www.w3.org/TR/owl-guide/>.

对 DAML + OIL 和 OIL 有兴趣者，可参考

- J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks, Enabling knowledge representation on the Web by Extending RDF Schema. In *Proceedings of the 10th World Wide Web Conference (WWW10)*, 2001. <http://www10.org/cdrom/papers/291/>

146

- D. Fensel, I. Horrocks, F. van Harmelen, D. McGuinness and P. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems* 16 March-April (2001): 38–45. <http://www.cs.vu.nl/ frankh/abstracts/IEEE-IS01.html>.

- D. McGuiness. Ontologies come of age. In *Spinning the Semantic Web*, ed. D. Fensel, J. Hendler, H. Lieberman and W. Wahlster. MIT Press 2003.

- P. Patel-Schneider, I. Horrocks and F. van Harmelen, Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web, In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI02)*. 2002. <http://www.cs.vu.nl/ frankh/abstracts/AAAI02.html>.

关于描述逻辑的一些参考文献：

- F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, eds. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambrdge: Cambridge University Press, 2002.

- E. Franconi. Description Logics Course Informaton. <http://www.cs.man.ac.uk/~franconi/dl/course/>.

- I. Horrocks and U. Sattler. Ontology Reasoning in the *SHOQ(D)* Description Logic. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*. 2001, 199–204.

- I. Horrocks. Tutorial on Description Logic. <http://www.cs.man.ac.uk/~horrocks/Slides/IJCAR-tutorial/Print/>.

两个有趣的网站：

- <http://www.w3.org/2001/sw/WebOnt/>. Information on OWL.

- <http://www.daml.org>. Information on DAML+OIL. See especially the pages /language, /ontologies and /tools.

下面是一些与本体的一般观念有关的链接，但与本章的内容非常不同。辞典(Thesauri)是非形式化本体的一种简单类型。

- <http://www.lub.lu.se/metadata/subject-help.html>. An extensive collection of pointers to thesauri. 147

- <http://www.topicmaps.org>. Topic maps constitute a simple ontology language in use today.

- <http://dublincore.org>. An example of an ontology used extensively in the digital library domain is the Dublin Core.

练习与设计

4.1 阅读 OWL 的在线规范和全部命名空间，见 < http：//w3. org > 。

4.2 用三种方式陈述两个类不相交。

4.3 分别表达两个事实：(a)所有数学课都是由 David Billington 教授一人(没有其他教师参与)讲授的；(b)数学课恰好是 David Billington 教授讲授的课。两者区别明显吗?

4.4 严格地说，对称性 `SymmetricProperty` 在 OWL 中是不必要的，因为它可由其他原语表达出来。请问如何表达？提示：考虑反函数。

4.5 与上一题类似，试用其他 OWL 语言构造表达函数性 `FunctionalProperty`。

4.6 确定哪些特性是 OWL 必须的？哪些仅是为了使用方便而引入、实际上可由其他建模

原语表达的?

4.7 在非洲野生生物本体示例中，若在食肉动物的定义中改 owl: allValuesFrom 为 owl: someValuesFrom，会出现什么问题？提示：考虑食草动物的定义。

4.8 说明三个概念 FunctionalProperty、InverseFunctionalProperty 与 Inverseof 之间的关系。

148 4.9 阐明将 owl: Class 规定为 rdfs: Class 的子类的必要性。

4.10 第3章介绍了 RDF 的公理语义，可为 OWL 构造类似的公理语义。请给出 intersectionOf 的公理语义。

4.11 给出 inverseOf 的公理语义。

4.12 构建一个关于基数约束的公理语义：

(a) 定义无重复列表 noRepeatsList。若 L 中每个元素最多出现一次，则 L 是"无重复列"表。这个概念不属于 OWL 语言，但可用于基数约束的元素计数。

(b) 定义 minCardinality 和 maxCardinality 为定义域是 Restriction、值域是 NonNegativeInteger 的属性。

(c) 给出一条公理刻画 minCardinality 的含义：若 *onProperty*(R, P)，且 *minCardinality*(R, n)则 x 是 R 的实例，当且仅当，存在一个长度≥ n 的"无重复列表"L 使得对所有 $y \in L$ 有 $P(x, y)$。

(d) 用类似的方法刻画 maxCardinality 的含义。

4.13 浏览 <http: //www. daml. org/ontologies > 上的一些本体。

4.14 用 OWL 写一个本体。

4.15 OIL 是 OWL 的前身，阅读 OIL 的相关材料及一些本体例子，比较 OIL 与 OWL 的异同。

4.16 比较 OWL 和 DAML + OIL 的在线文档。

4.17 用 OWL 的词汇重写 DAML + OIL 文档的例子。

4.18 思考 OWL 还缺少哪些特性。[提示：考虑项目(projects)和员工。对每个项目，什么是真的？对每个员工呢(限于对公司有益的)？这些条件能用 OWL 表达吗?]

149

第5章　逻辑和推理：规则

5.1　引言

从抽象的观点看，前面几章的主题是与*知识表示*(representation of knowledge)相关的，包括关于网上资源内容的知识和关于应用领域中的概念及其相互关系的知识(本体)。

早在万维网出现之前，人工智能领域就研究过知识表示。哲学领域对相关课题的研究更早，甚至可以追溯到古希腊。亚里士多德被认为是逻辑学之父，而逻辑特别是*谓词逻辑*(predicate logic)(又称为*一阶逻辑*(first-order logic)一直是知识表示的基础。这里，我们列出一些使得逻辑通用和重要的原因：

- 它提供一种便于表达知识的高层语言，并且具有很强的表达能力。
- 它具有易于理解的形式化语义，将无歧义的含义赋予逻辑公式。
- 提供了精确的逻辑后承(logical consequence)概念，它决定一个公式是否可由另一些公式(前提)在语义上推出。实际上，逻辑最原始的动机正是研究逻辑后承的客观规律。
- 存在由前提集从语法上自动推出结论的证明系统。
- 存在逻辑后承与语法推导相一致的证明系统。证明系统应当是可靠的(所有语法可推出的公式都是前提的逻辑后承)和完全的(前提的所有逻辑后承都可由证明系统推出)。
- 在存在可靠且完全的证明系统的意义上，谓词逻辑是独一无二的。表达能力更强的逻辑(高阶逻辑)没有这样的证明系统。
- 由于存在证明系统，可以追踪逻辑后承的推导过程。在此意义上，逻辑能够提供其结论的理由。

RDF 和 OWL(Lite 和 DL)语言可视为谓词逻辑的特殊形式。其公理语义说明了这一点。

这些特殊语言存在的理由之一是它们提供了一种能够很好地满足预定需求的语法(对我们来说是基于标记的 Web 语言)。另一个主要理由是它们定义了逻辑的可推理子集。4.1 节提到，对于某些特定的逻辑，在表达能力和计算复杂性之间有一种相互制约：语言的表达能力越强，其证明系统的效率越低(最坏情况下)。正如我们说过的，OWL Lite 和 OWL DL

粗略地对应于某种描述逻辑(description logic)，而这种描述逻辑是谓词逻辑的一个存在高效率证明系统的子集。

谓词逻辑的另一个存在高效率证明系统的子集包括所谓的规则系统(rule systems)，也称为 Horn 逻辑(Horn logic)或确定逻辑程序(definite logic program)。一个规则有如下形式

$A_1, \cdots, A_n \rightarrow B$

其中 A_i 和 B 都是原子公式。实际上，对于这样的一条规则有两种直观解释：

1. 如果 $A_1, \cdots, A_n$ 已知为真，那么 B 也真。这样解释的规则称为演绎规则(deductive rules)。

2. 如果条件 $A_1, \cdots, A_n$ 为真，那么执行行动 B。这样解释的规则称为反应式规则(reactive rules)。

152 这两种观点都有重要的应用。但在本章中，我们采用演绎解释，并研究这种语言和可能的查询及其适当的回答。我们也简要描述产生这些回答的推理机制。

需要注意的是，描述逻辑和 Horn 逻辑是正交的，即二者中的任何一个都不是另一个的子集。例如，在 OWL 中不可能做出这样的断言：学习且居住在同一城市的人是“home students”，但这可以很方便地用规则来表达：

studies(*X*, *Y*), *lives*(*X*, *Z*), *loc*(*Z*, *U*) -> *homestudent*(*X*)

另一方面，规则不能表达这样的信息：一个人要么是男性要么是女性，但在 OWL 中这可以很方便地用不可兼或(disjoint union)来表达。

下面关注另一种类型的规则。我们给出一个简单的例子。假设一个网上卖家希望在顾客的生日当天给予特别优惠。可以用如下的规则来简单地表达这个应用：

*R*1：If birthday, then special discount.(如果是生日，则有特别优惠)

*R*2：If not birthday, then not special discount.(如果不是生日，则没有特别优惠)

这种解决方案在已知生日的情况下运转良好。但是，假设一个顾客因为涉及隐私而拒绝提供他的生日。在这种情况下，就不能应用上面的规则了，因为其前提是未知的。为了处理这种情况，需要写出如下的规则：

*R*1：If birthday, then special discount.(如果是生日，则有特别优惠)

*R*2'：If birthday is not known, then not special discount.(如果不知道是生日，则没有特别优惠)

但是，规则 *R*2' 的前提不在谓词逻辑的表达能力之内，于是，就需要一种新的规则系统。注意规则 *R*1 和 *R*2 适用于掌握相关情形的完全信息的情况(比如，或者是生日或者不是生日)。新型的规则系统将应用于信息不完全的情况。

谓词逻辑和它的特殊形式在下面的意义下是单调的：一旦推出一个结论，它将总是有

效的，即使后来又获得了新的信息。如果用规则 *R2'* 推出结论“没有特别优惠”，当后来得知客户的生日并且与购买日期一致时，这个结论将变得不再有效。所以，我们称其为非单调规则（nonmonotonic rules），以便与单调规则（谓词逻辑的特殊形式）相区别。本章我们将同时讨论单调规则和非单调规则。 153

我们最后关注规则在不同应用中的转换。例如，一个网上商店可能会用智能代理可理解的规则来表达定价、退款和自行规定的特别条款。语义网途径正是这样，使用我们讨论过的某种网络语言，把这些知识表达成机器可解读的形式。本章将说明规则是如何在类 XML 语言（“规则标记语言”）中表达的。一些规则系统的应用将在第 6 章讨论。

5.2 节给出一个运用单调规则（谓词逻辑的一个子集——称为 Horn 逻辑）的例子，5.3 节和 5.4 节分别介绍 Horn 逻辑的语法和语义，5.5 节和 5.6 节分别介绍非单调规则的语法和一个例子，最后，5.7 节和 5.8 节分别介绍单调规则和非单调规则的基于 XML 的表示方法。

5.2 单调规则的例子：家庭关系

设想一个关于家庭关系的数据库。假设这个数据库包含以下基本谓词（base predicates）：

mother(*X*, *Y*)	*X* 是 *Y* 的母亲
father(*X*, *Y*)	*X* 是 *Y* 的父亲
male(*X*)	*X* 是男性
female(*X*)	*X* 是女性

然后用合适的规则推导进一步的关系。首先定义谓词 *parent*（父母）。parent 或者是 father（父亲）或者是 mother（母亲）。

$$mother(X, Y) \rightarrow parent(X, Y)$$

$$father(X, Y) \rightarrow parent(X, Y)$$

然后定义 brother（兄弟）是有相同 parent 的男性：

$$male(X), parent(P, X), parent(P, Y), notSame(X, Y) \rightarrow brother(X, Y)$$ 154

谓词 *notSame* 表示不相等，并假定所有关于不相等的事实[⊖]保存在数据库中。虽然每个实用的逻辑系统会提供方便的手段以表达相等和不相等，但是为了保持讨论的一般性，这里选择抽象的解决方法。

类似地，*sister*（姐妹）定义如下：

$$female(X), parent(P, X), parent(P, Y), notSame(X, Y) \rightarrow sister(X, Y)$$

⊖ “事实”的定义见 5.3.2 节。——译者注

uncle(叔叔)是 parent 的 brother:

$brother(X, P), parent(P, Y) \rightarrow uncle(X, Y)$

grandmother(祖母)是 parent 的 mother(母亲):

$mother(X, P), parent(P, Y) \rightarrow grandmother(X, Y)$

ancestor(祖先)可能是 parent 或是某个 parent 的 ancestor:

$parent(X, Y) \rightarrow ancestor(X, Y)$

$ancestor(X, P), parent(P, Y) \rightarrow ancestor(X, Y)$

5.3 单调规则:语法

考察一个简单的规则,它的意思是,所有超过 60 岁的忠实顾客可以享受一种特别优惠:

$loyalCustomer(X), age(X) > 60 \rightarrow discount(X)$

以此为例,一条规则可以包含下列成分:

变量——代表非特定值[⊖],这里是 X

常量——代表特定值,这里是 60

155 谓词——代表值的属性(关系或性质),这里是 *loyalCustomer* 和 >

函数符号——代表函数,这里是 *age*

5.3.1 规则

一条规则形如:

$B_1, \cdots, B_n \rightarrow A$

其中 A, B_1, …, B_n 是原子公式。A 是规则的头(head), B_1, …, B_n 是规则的前提(premises)。集合 $\{B_1, \cdots, B_n\}$ 称作规则的体(body)。

规则体中的逗号代表合取。上述规则直观解释为:如果 B_1, B_2, …, B_n 为真,则 A 也为真(或者等价地,如果证明了所有 B_1, …, B_n,也就证明了 A)。注意,A, B_1, …, B_n 中可以出现变量,例如,

$loyalCustomer(X), age(X) > 60 \rightarrow discount(X)$

这条规则对所有的顾客都适用:如果顾客是忠实的并且超过 60 岁,那么他或她就得到优惠。换句话说,默认变量 X 是全称量化的(在逻辑公式中用 $\forall X$ 表示)。一般情况下,规则中出现的所有变量都默认为是全称量化的。

综上所述,一条规则 r

⊖ 这里的"值"不限于数值,也不限于"值域"中的元素,可以代表任何"对象"(见 5.4 节)。——译者注

$$B_1, \cdots, B_n \rightarrow A$$

解释为下列谓词逻辑公式

$$\forall X_1 \cdots \forall X_k((B_1 \wedge \cdots \wedge B_n) \rightarrow A)$$

记为 $pl(r)$，它义等价于

$$\forall X_1 \cdots \forall X_k(A \vee \neg B_1 \vee \cdots \vee \neg B_n)$$

其中 $X_1, \cdots, X_k$ 是 $A, B_1, \cdots, B_n$ 中出现的所有变量。以后称 $pl(r)$ 为 r 的“谓词逻辑解释”。

5.3.2 事实

一个事实是一个原子公式，比如 *loyalCustomer*(*a*345678)，它表示“ID 为 a345678 的顾客是忠实的”。默认事实中出现的所有变量都是全称量化的。

5.3.3 逻辑程序

逻辑程序[○]P 是事实和规则的一个有限集合。它的谓词逻辑解释 $pl(P)$ 是 P 中所有规则和事实的谓词逻辑解释的集合。 156

5.3.4 目标

目标是对一个逻辑程序提出的查询 G，形如 $B_1, \cdots, B_n \rightarrow$。如果 $n=0$，则为空目标□。

下一个任务是在谓词逻辑中解释目标。根据先前的想法(即将逗号解释为合取，默认全称量化)，一个目标解释为：

$$\forall X_1 \cdots \forall X_k(\neg B_1 \vee \cdots \vee \neg B_n)$$

这个公式与 $pl(r)$ 相似，唯一的不同在于这条规则的头 A 被省略了[⊖]。它在谓词逻辑中等价于

$$\neg \exists X_1 \cdots \exists X_k(B_1 \wedge \cdots \wedge B_n)$$

其中 $X_1, \cdots, X_k$ 是在 $B_1, \cdots, B_n$ 中出现的所有变量。下面进一步说明这个公式的含义。假设我们知道 $p(a)$，并且有目标 $p(X) \rightarrow$。事实上，使用这个目标的动机是希望知道是否存在一个值使得 p 为真。因为有事实 $p(a)$，我们将得到一个肯定的回答，也就是 $\exists X\ p(X)$。但是，为什么目标要取为这个回答的否定呢？原因在于逻辑程序运用了一种数学证明的技巧，称为反证法(proof by contradiction)。为了证明从 B 可以推出 A，这种技巧先假设 A 为

⊖ 由于这里是在单调规则的基础上定义逻辑程序的，这种逻辑程序是单调逻辑程序。——译者注

⊖ 注意，这个公式等价于 $\forall X_1 \cdots \forall X_k(false \vee \neg B_1 \vee \cdots \vee \neg B_n)$，所以，可以认为省略的规则头是矛盾式 *fase*。

假，并与 B 共同作为前提推出矛盾，从而证明 A 假与 B 不能同时成立。因此，由 B 必然推出 A。

在逻辑程序中，要证明一个目标有肯定回答，是通过否定该目标并且证明这个逻辑程
157 序推出矛盾来实现的。比如说，给定上述逻辑程序 $p(a)$ 和目标 $\neg\ \exists Xp(X)$，将得到一个逻辑矛盾：第二个公式说明没有元素具有属性 p，但是第一个公式说明值 a 确实具有属性 p。这样，由 $p(a)$ 就可逻辑地推出 $\exists Xp(X)$。

5.4 单调规则：语义

5.4.1 谓词逻辑语义

回答查询的一种方式是运用上述对规则、事实和查询的谓词逻辑解释，以及众所周知的谓词逻辑语义学。更精确地说，给定一个逻辑程序 P 和一个含变量 X_1，…，X_k 的查询 B_1，…，$B_n\rightarrow$，对该查询的回答是肯定的，当且仅当

$$pl(P)\models \exists X_1\cdots\exists X_k(B_1\wedge\cdots\wedge B_n) \tag{1}$$

或等价地，如果

$$pl(P)\cup\{\neg\ \exists X_1\cdots\exists X_k(B_1\wedge\cdots\wedge B_n)\}\text{不可满足} \tag{2}$$

也就是说，对一个查询的回答是肯定的，当且仅当逻辑程序 P 的谓词逻辑表示 $pl(P)$ 和该查询的谓词逻辑解释的并集是不可满足的(即矛盾的)。

谓词逻辑语义学的严格定义可以在文献中找到，这里只给出非形式化的描述。一般而言，一个逻辑语言可以被赋予不同的含义。而谓词逻辑的一个模型 $\mathcal{A}$(model)赋予一种特定的含义。一个模型 $\mathcal{A}$ 包括：

- 一个论域 $dom(\mathcal{A})$，是一个非空集，其中元素称为对象，公式正是对这些对象做出断定；
- 每个常量对应论域 $dom(\mathcal{A})$ 中的元素；
- 每个函数符号对应的 $dom(\mathcal{A})$ 中的具体函数；
158 • 每个谓词对应 $dom(\mathcal{A})$ 中的关系或性质。

逻辑连接词$\neg$，$\vee$，$\wedge$，$\rightarrow$，$\forall$，$\exists$ 根据它们的直观意义定义为：非，或者，并且，蕴含，对所有，存在。进而可以定义一个公式在模型 $\mathcal{A}$ 下为真，记为 $\mathcal{A}\models\varphi$。

定义公式 φ 是公式集 M 的逻辑后承，或称 M 逻辑地推出 φ，如果在 M 为真(即 M 中所有公式都为真)的所有模型 $\mathcal{A}$ 中 φ 都为真。

现在能够说明(1)和(2)的含义了，这就是：无论如何解释程序 P 及查询中出现的常量、谓词、函数词，只要 P 的谓词逻辑解释为真，则 $\exists X_1\cdots\exists X_k(B_1\wedge\cdots\wedge B_n)$ 也必然为真，

就是说，存在变量 X_1，…，X_k的值，使得所有原子公式 B_i为真。

例如，假设 P 是程序

$p(a)$

$p(X)\rightarrow q(X)$

考虑查询 $q(X)\rightarrow$。显然，$q(a)$可由 $pl(P)$逻辑地推出。因此，$\exists Xq(X)$可由 $pl(P)$逻辑地推出，于是 $pl(P)\cup\{\neg\exists Xq(X)\}$是不可满足的，因此，程序对查询给出肯定回答。但是，如果考虑查询 $q(b)\rightarrow$，则一定给出否定回答，因为 $q(b)$不能由 $pl(P)$逻辑地推出。

逻辑程序的另一种语义——极小 Herbrand 模型语义，涉及更多的技术细节，就不在此讨论了。

5.4.2 常证据和带参证据

到目前为止，讨论集中在查询的是/否回答上。但是，这种回答不一定是最理想的。假设有事实

$p(a)$

和查询

$p(X)\rightarrow$

回答“是”是正确的，但不是令人满意的。这就像有人问你：“你知道现在几点了吗?”你看了看表，然后回答“是”。在上述例子中，恰当的回答应该是替换$\{X/a\}$，它给出 X 的实例 a，使得回答是肯定的和具体的，而这里的常量 a 可称为一个常证据(ground[⊖] witness)。假如给定的事实是

$p(a)$，$p(b)$

则对相同的查询，就有两个常证据 a 和 b。或者等价地，返回两个替换$\{X/a\}$和$\{X/b\}$作为推理结果。

常证据并不总是最佳的回答，尽管有时它是有价值的。考虑逻辑程序

$add(X, 0, X)$

$add(X, Y, Z)\rightarrow add(X, s(Y), s(Z))$

如果我们把 s 理解为“后继函数”，它的返回值是它的参数的值加 1，则这个程序刻画的是加法。add 的第三个参数计算它的前两个参数的和。考虑查询 $add(X, s^8(0), Z)\rightarrow$。可能的常证据由以下替换决定

$\{X/0, Z/s^8(0)\}$

$\{X/s(0), Z/s^9(0)\}$

⊖ 此处 ground 一词的基本含义可理解为“不含变元的”。——译者注

$\{X/s(s(0)),\ Z/s^{10}(0)\}$

……

而带参证据(parameterized witness)$Z = s^8(X)$是查询

160 $\exists X \exists Z\ add(X,\ s^8(0),\ Z)$

的最一般证据。

这种最一般的证据的计算是SLD反驳⊖的主要目的，对它的介绍超出了本书的范围。

5.5 非单调规则：动机和语法

5.5.1 非正式讨论

现在把注意力转向非单调规则系统。到目前为止，一旦规则的前提得以证明，这条规则就被应用，而它的头也当作结论被推出。在非单调规则系统中，即使所有前提已知，规则也可能不会被应用，因为还必须考虑是否同时存在与之相矛盾的推理。一般说来，下面考虑的规则都称为可废止的(defeasible)，因为其他规则可以废止它们。为了允许规则间的冲突，否定的原子公式(negated atomic formulas)可以出现在规则的头或体中。例如，可以有下面的规则

$p(X) \rightarrow q(X)$

$r(X) \rightarrow \neg\ q(X)$

下面使用不同的箭头来区别可废止规则和标准的单调规则：

$p(X) => q(X)$

$r(X) => (q(X)$

在这个例子中，如果同时还给出事实

$p(a)$

$r(a)$

则根据非单调规则，既推不出$q(a)$也推不出$\neg\ q(a)$。这是一个两条规则彼此阻塞的典型例子。这种冲突可以通过使用规则间优先序(priorities among rules)来解决。假设我们知道由于某些原因，第一条规则比第二条规则可靠，那么我们可以确定地推出$q(a)$。

161 在实践中，优先序的出现是很自然的，可以基于各种不同的原则，比如：

- 一条规则的来源可能比另一条的来源更可靠或更权威。例如，在法律领域，联邦法就优先于州法。同样，在商业经营中，高层管理部门比中层管理部门更权威。
- 一条规则可能比另一条更优先，因为它在时间上更近。

⊖ SLD反驳即“程序子句的带选择函数的线性反驳”(selective linear resolution for definite clauses).

- 一条规则可能比另一条更优先，因为它更特殊。典型的例子是一条普遍的规则带有一些对例外情况的特殊规定；在出现这些例外情况时，特殊规定比一般规则本身更应当被遵守。

对于给定的一组规则，特殊性通常可以根据这些规则计算出来，但第一条和第二条原则无法由逻辑推理定义。所以，我们对具体的优先原则加以抽象，假定存在规则集上的一种*外在优先关系*(external priority relation)，用它统一地刻画各种具体的优先原则。为了从语法上表达这种关系，拓展规则语法以包含一个唯一的标号，例如，

$r_1: p(X) \Rightarrow q(X)$

$r_2: r(X) \Rightarrow \neg q(X)$

于是可以用

$r_1 > r_2$

来表示 r_1 比 r_2 更优先。

这里不对 > 施加很多条件，甚至不要求它是规则间的全序关系。我们仅要求优先关系是无环的，也就是，不能有如下形式的环

$r_1 > r_2 > \cdots > r_n > r_1$

注意，优先关系是为了解决竞争规则(competing rules)间的冲突而引进的。在简单情况下，仅当一条规则的头是另一条规则的头的否定时，这两条规则才出现竞争。但实际应用中未必总是如此，常见的情况是，当某个谓词 p 被推出时，不再允许另一些谓词成立。例如，一个投资顾问可能将他的建议建立在投资者可以接受的三种风险级别上：低，中等和高。显然，每个投资者在任何给定时刻只能选择一种风险级别。技术上，可以通过给每个文字 L 维护一个冲突集(conflict set) $C(L)$ 来刻画这种情形。$C(L)$ 总是含有 L 的否定，也可以包含更多文字。 162

5.5.2 语法定义

可废止规则(defeasible rule)有如下形式

$r: L_1, \cdots L_n \Rightarrow L$

其中 r 是标号(label)，$\{L_1, \cdots, L_n\}$ 是体(或前提)，L 是规则的头。$L, L_1, \cdots, L_n$ 是正或负文字(一个文字是一个原子公式 $p(t_1, \cdots, t_m)$ 或它的否定 $\neg p(t_1, \cdots, t_m)$)。在规则中没有函数词出现⊖。有时我们用 $head(r)$ 表示规则的头，$body(r)$ 表示体。有时用标号 r 指代整个规则，虽然这有些不严格。

可废止逻辑程序(defeasible logic program)是一个三元组 $(F, R, >)$，包括事实集 F，

⊖ 由于技术原因而采取这个限制，关于这个问题的讨论已超出了本章的范围。

可废止规则的有限集 R，以及 R 上的无环二元关系 $>$（严格地说是 $r > r'$ 的集合，其中 r，r' 是 R 中规则的标号）。

5.6 非单调规则的例子：有经纪人的交易

这个例子说明，在电子商务（它在语义网上可以理想地运行）领域可以怎样使用规则。有经纪人的交易（brokered trade）通过独立的第三方——经纪人来实现。经纪人匹配买家的需求和卖家的能力，当双方都满意时提议进行交易。

作为一个具体应用，下面讨论公寓租赁这种常见但通常乏味耗时的活动[⊖]。适当的网络服务可以相当大地减少工作量。首先给出一个潜在租赁者的需求。

> 卡洛斯正在找一个至少 45 平方米、至少有两个卧室的公寓。如果是在三楼或三楼以上，大楼必须有电梯。而且可以养宠物。
>
> 卡洛斯愿意为市中心的 45 平方米大小的公寓付 300 美元，为在市郊的类似公寓付 250 美元。并且，他愿意为公寓超出 45 平方米的部分每平方米支付 5 美元，
>
> 163 为花园每平方米付 2 美元。
>
> 他的付款总额不会超过 400 美元。在给定的可选项中，他将选择最便宜的，第二优先的是有花园的，最后才是有额外空间的。

5.6.1 卡洛斯需求的形式化描述

我们用下面的谓词来描述公寓的属性：

size(x, y)	y 是公寓 x 的大小（单位为平方米）
bedrooms(x, y)	x 有 y 个卧室
price(x, y)	y 是 x 的价格
floor(x, y)	x 是在第 y 层楼
garden(x, y)	x 有大小为 y 的花园
lift(x)	在 x 所在的大楼里有电梯
pets(x)	在 x 里允许养宠物
central(x)	x 位于市中心

我们还使用下面的谓词：

acceptable(x)	公寓 x 满足卡洛斯的要求
offer(x, y)	卡洛斯愿意为 x 付 y 美元

⊖ 在这种情况下，房东扮演抽象的卖家角色。

现在我们来表达卡洛斯的需求。不考虑具体需求，任何公寓都是可接受的，表示为：

r_1：=> $acceptable(X)$

但是，如果卡洛斯的某条要求没有被满足，则 X 就是不可接受的：

r_2：$bedroome(X,\ Y)$，$Y<2$ => ¬ $acceptable(X)$

r_3：$size(X,\ Y)$，$Y<45$ => ¬ $acceptable(X)$

r_4：¬ $pets(X)$ => ¬ $acceptable(X)$

r_5：$floor(X,\ Y)$，$Y>2$，($lift(X)$ => ¬ $acceptable(X)$

r_6：$price(X,\ Y)$，$Y>400$ => ¬ $acceptable(X)$

规则 $r_2 - r_6$ 是对规则 r_1 的例外情况的特殊规定，所以这些规则都比 r_1 优先：

$r_2 > r_1$，$r_3 > r_1$，$r_4 > r_1$，$r_5 > r_1$，$r_6 > r_1$ 164

然后，给出计算卡洛斯愿意为一个公寓支付的价钱的规则。

r_7：$size(X,\ Y)$，$Y \geqslant 45$，$garden(X,\ Z)$，$central(X)$ => $offer(X,\ 300+2Z+5(Y-45))$

r_8：$size(X,\ Y)$，$Y \geqslant 45$，$garden(X,\ Z)$，($central(X)$ => $offer(X,\ 250+2Z+5(Y-45))$

仅当卡洛斯愿意付出的价钱少于房东提出的价钱时，此公寓才是可接受的(假设没有议价发生)，这用规则和优先序表示为：

r_9：$offer(X,\ Y)$，$price(X,\ Z)$，$Y<Z$ => ($acceptable(X)$

$r_9 > r1$

5.6.2 可选公寓的表达

每个可供选择的公寓都有唯一的名字，它的属性以事实形式表达。例如，公寓 a_1 可以描述如下：

$bedrooms(a_1,\ 1)$

$size(a_1,\ 50)$

$central(a_1)$

$floor(a_1,\ 1)$

¬ $lift(a_1)$

$pets(a_1)$

$garden(a_1,\ 0)$

$price(a_1,\ 300)$

表 5.1 罗列出所有可选公寓的描述。实际应用中，待租公寓的报价信息可以存储在关系数据库中。

如果将卡洛斯的需求和可选公寓的描述相匹配，我们将发现

- 公寓 a_1 不可接受，因为它只有一个卧室(规则 r_2)
- 公寓 a_4 和 a_6 不可接受，因为不许养宠物(规则 r_4)
- 对于 a_2，卡洛斯愿意付 300 美元，但实际价格更高(规则 r_7 和 r_9)
- 公寓 a_3，a_5 和 a_7 是可接受的(规则 r_1)

165

表 5-1 可选公寓

Flat	Bedrooms	Size	Central	Floor	litf	Pets	Garden	Price
a_1	1	50	是	1	不	是	0	300
a_2	2	45	是	0	不	是	0	335
a_3	2	65	不	2	不	是	0	350
a_4	2	55	不	1	是	不	15	330
a_5	3	55	是	0	不	是	15	350
a_6	2	60	是	3	不	不	0	370
a_7	3	65	是	1	不	是	12	375

5.6.3 选择一间公寓

至此已经识别出卡洛斯可以接受的公寓。这个筛选是有价值的，因为它减少了值得关注的公寓数目，选出来的可以实地考察。不过，通过进一步考虑卡洛斯的偏好(preferences)，[⊖]有可能进一步减少公寓数目，直至减少到一个。卡洛斯的偏好是基于价格、花园大小和房子大小的，而且这三个因素按顺序排列，这可以用规则和优先序表示如下：

r_{10}: $cheapest(X) \Rightarrow rent(X)$

r_{11}: $cheapest(X)$, $largestGarden(X) \Rightarrow rent(X)$

r_{12}: $cheapest(X)$, $largestGarden(X)$, $largest(X) \Rightarrow rent(X)$

$r_{12} > r_{10}$

$r_{12} > r_{11}$

$r_{11} > r_{10}$

此外，我们需要规定最多只有一个公寓会被租用，这可以利用冲突集表示如下：

$C(rent(x)) = \{\neg rent(x)\}(\{rent(y) \mid y \neq x\}$

利用其他规则，从可接受公寓的集合可以推出这些规则的前提。为了保持讨论的简单性，这里只列出这个例子中用到的事实：

166

$cheapest(a_3)$

$cheapest(a_5)$

$largest(a_3)$

⊖ 此例中的“偏好”属于直观上所谓“个人偏好”，与上文提到的三种优先原则不同。但在一定条件下，可以用规则集的外在优先半系表达，正如书中表明的那样。——译者注

$largest(a_7)$

$largestGarden(a_5)$

于是，经过下列推理，可以得出租用 a_5 的决定：

- 规则 r_{11} 的前提被 a_5 满足，所以由 r_{11} 推出 $rent(a_5)$。
- 规则 r_{10} 的前提被 a_3 满足，所以由 r_{10} 推出 $rent(a_3)$。这就构成一个冲突。但由于 r_{11} 优先于 r_{10}，所以这个冲突被化解，$rent(a_3)$ 被 $rent(a_5)$“击败”，换句话说，r_{10} 在当前情况下“失效”。
- 这是唯一的冲突，因为其他公寓都不满足 r_{11} 和 r_{12} 的前提。

这样就做出了一个选择“租用 a_5”，卡洛斯马上就能搬进去。

5.7 XML 中的规则标记：单调规则

本节的目的是依照语义网的观点，用机器可处理的规则形式描述知识。我们勾画出单调规则的一种用 XML 编码的方式。

5.7.1 项

项用 XML 的标签 <term>，<function>，<var> 和 <const> 表达。例如，项

$f(X, a, g(b, Y))$

表达为：

```
<term>
    <function>f</function>
    <term>
        <var>X</var>
    </term>
    <term>
        <const>a</const>
    </term>
    <term>
        <function>g</function>
        <term>
            <const>b</const>
        </term>
        <term>
            <var>Y</var>
        </term>
    </term>
</term>
```

5.7.2 原子公式

用附加的标签 <atom> 和 <predicate> 来表示原子公式。例如，公式

$p(X, a, f(b, Y))$

表示如下：

```
<atom>
    <predicate>p</predicate>
    <term>
        <var>X</var>
    </term>
    <term>
        <const>a</const>
    </term>
    <term>
        <function>f</function>
        <term>
            <const>b</const>
        </term>
        <term>
            <var>Y</var>
        </term>
    </term>
</atom>
```

168

注意函数词、谓词和常量之间的区别在上面使用的逻辑语法中是隐式的，而在 XML 中是显式的。

5.7.3 事实

一个事实就是一个原子公式，由标签 <fact> 开始和结束。例如，事实 $p(a)$ 表示如下：

```
<fact>
    <atom>
        <predicate>p</predicate>
        <term>
            <const>a</const>
        </term>
    </atom>
</fact>
```

5.7.4 规则

规则包括头和体。头是一个原子公式。体是一个原子公式序列(可能为空)。我们使用新的标签 <rule>， <head>和 <body>。例如，规则

$p(X, a), q(Y, b) \rightarrow r(X, Y)$

表示如下：

```
<rule>
    <head>
        <atom>
            <predicate>r</predicate>
            <term>
                <var>X</var>
            </term>
            <term>
                <var>Y</var>
            </term>
        </atom>
    </head>
    <body>
        <atom>
<atom>
    <predicate>p</predicate>
    <term>
        <var>X</var>
    </term>
    <term>
        <const>a</const>
    </term>
</atom>
<atom>
    <predicate>q</predicate>
    <term>
        <var>Y</var>
    </term>
    <term>
        <const>b</const>
    </term>
        </atom>
    </body>
</rule>
```

5.7.5 查询

查询表达为规则的体，用标签 <query>包围。

5.7.6 一个 DTD

一个程序由一些规则和事实构成。

```
<! ELEMENT program ((rule | fact)* ) >
```

一个事实由一个原子公式构成。

```
<! ELEMENT fact (atom) >
```

一条规则由头和体构成。

170

```
<! ELEMENT rule (head, body) >
```

头由一个原子公式构成。

```
<! ELEMENT head (atom) >
```

体由一系列原子公式构成。

```
<! ELEMENT body (atom* ) >
```

一个原子公式由一个谓词和它后面的一些项构成。

```
<! ELEMENT atom (predicate, term* ) >
```

一个项可以是一个常量、一个变量，或者是一个由函数词和它后面的一些项构成的复合项。

```
<! ELEMENT term (const | var | (function, term* )) >
```

谓词、函数词、常量和变量都是原子类型。

```
<! ELEMENT predicate (#PCDATA) >
<! ELEMENT function (#PCDATA) >
<! ELEMENT var (#PCDATA) >
<! ELEMENT const (#PCDATA) >
```

一个查询是一列原子公式。

```
<! ELEMENT query (atom* ) >
```

5.7.7 RuleML 的另一种数据模型

RuleML 是语义网背景下关于规则的一项重要的标准化举措，基于与 DTD 表示相似的想法(图 5-1 展示了 DTD 和 RuleML 所用标签的对比)。但是，RuleML 已经形成了另一种不同的数据模型，结合了 XML 和 RDF 的特点。在 XML 中元素的次序是重要的，而在 RDF

中这种次序被忽略了。

Our DTD	RuleML
program	rulebase
fact	fact
rule	imp
head	_head
body	_body
atom	atom
atom*	and
predicate	rel
const	ind
var	var

图 5-1　单调规则 DTD 与 RuleML 的对比

RuleML 目前以 XML 为基础，但运用类 RDF 的“role 标签”，它在一个表达中的位置是无关紧要的。例如，如果我们运用 role 标签 < _head > 和 < _body >，则以下表达 171

```
<rule>
    <_head>
        <atom>
            <predicate>p</predicate>
            <term>
                <const>a</const>
            </term>
        </atom>
</_ head>
<_body>
    <atom>
        <predicate>q</predicate>
        <term>
            <const>b</const>
        </term>
    </atom>
</_ body>
    </_ body>
</rule>
```

等价于

```
<rule>
    <_body>
        <atom>
            <predicate>q</predicate>                                    172
            <term>
```

```
            <const>b</const>
        </term>
    </atom>
</_body>
<_head>
    <atom>
        <predicate>p</predicate>
        <term>
            <const>a</const>
        </term>
    </atom>
</_head>
</rule>
```

尽管这两种表达在XML数据模型(其中次序是重要的)中是不同的。对这个观点的讨论，参考建议的读物。

有一点必须明确，用XML不仅可以表达程序和查询，还可以表达替换和证明。

5.8 XML中的规则标记：非单调规则

与单调规则相比，非单调规则在语法上有以下几点不同：

- 没有函数词，因此项的结构是平坦的(flat)；
- 否定词在规则的头和体中都可以出现；
- 每一条规则有一个标号；
- 除了规则和事实之外，一个程序还包含优先序语句。

5.8.1 例子

考虑可废止程序

r_1: $p(X) \Rightarrow s(X)$

173 r_2: $q(X) \Rightarrow s(X)$

$p(a)$

$q(a)$

$r_1 > r_2$

我们用标签 <stronger> 描述优先序，用规则的ID标号来指称它们的名字。

规则 r_1 表示如下：

```
<rule id="r1">
    <head>
        <atom>
```

```
        <predicate>s</predicate>
        <term>
            <var>X</var>
        </term>
    </atom>
</head>
<body>
    <atom>
        <predicate>p</predicate>
        <term>
            <var>X</var>
        </term>
    </atom>
</body>
</rule>
```

规则 r_2 相应地表示。事实 $p(a)$ 表示如下：

```
<fact>
    <atom>
        <predicate>p</predicate>
        <term>
            <const>a</const>
        </term>
    </atom>
</fact>
```

优先关系 $r_1 > r_2$ 表示如下：

```
<stronger superior="r1" inferior="r2"/>
```

174

5.8.2 一个 DTD

一个程序由一些规则、一些事实和优先序关系组成。

```
<! ELEMENT program ((rule | fact | stronger)* )>
```

一个事实由一个原子公式或它的否定组成。

```
<! ELEMENT fact (atom | neg)>
<! ELEMENT neg (atom)>
```

一条规则由头和体元素，还有一个 id 属性组成。

```
<! ELEMENT rule (head, body)>
<! ATTLIST rule
    id ID #IMPLIED>
```

规则的头和体的定义类似单调规则，但是可以包含否定原子。

```
<! ELEMENT head (atom | neg) >
<! ELEMENT body ((atom | neg)* ) >
```

一个原子公式由一个谓词和它后面的一些变量和常量组成。

```
<! ELEMENT atom (predicate, (var | const)* ) >
```

优先序中的一项有两种属性，分别代表优先级高的和优先级低的规则。

```
<! ELEMENT stronger EMPTY) >
<! ATTLIST stronger
     superior IDREF #REQUIRED >
     inferior IDREF #REQUIRED >
```

谓词、常量和变量都是原子类型。

```
<! ELEMENT predicate (#PCDATA) >
<! ELEMENT var (#PCDATA) >
<! ELEMENT const (#PCDATA) >
```

175 一个查询是一系列原子公式。

```
<! ELEMENT query (atom* ) >
```

5.9 小结

- Horn 逻辑是谓词逻辑的一个存在高效率推理的子集。它与谓词逻辑的另一个子集描述逻辑是正交的。
- Horn 逻辑是单调规则的基础。
- 非单调规则在信息不完全的情况下是有用的。这些规则可以在相反的证据(其他规则)面前失效。
- 优先序用来解决非单调规则之间的冲突。
- 可以在类 XML 语言中简洁地表达规则。

推荐读物

单调规则是逻辑的一个标准话题。更多的信息可参考下列书籍：

- E. Burke and E. Foxley. *Logic and Its Applications*. Upper Saddle River, N.J: Prentice Hall, 1996.
- M. A. Covington, D. Nute, and A. Vellino. *Prolog Programming in Depth*, 2nd ed. Upper Saddle River, N.J: Prentice Hall, 1997.

- A. Nerode and R. A. Shore. *Logic for Applications*. New York: Springer, 1997.
- U. Nilsson and J. Maluszynski. *Logic, Programming and Prolog*, 2nd ed. New York: Wiley, 1995.
- N. Nissanke. *Introductory Logic and Sets for Computer Scientists*. Boston: Addison-Wesley, 1998.

非单调规则是一个比较新的话题。有关信息可以参考 Covington，Nute 和 Vellino 的 *Prolog Programming in Depth*，以及

- G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Transactions on Computational Logic* 2 (April 2001): 255-287.
- B. N. Grosof. Prioritized Conflict Handling for Logic Programs. In *Proceedings of the International Logic Programming Symposium*. 1997, 197-211.
- B. N. Grosof, Y. Labrou, and H. Y. Chan. A Declarative Approach to Business Rules in Contracts: Courteous Logic Programs in XML. In *Proceedings of the 1st ACM Conference on Electronic Commerce (EC-99)*, 1999. 176
- D. Nute. Defeasible Logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 3*, D. M. Gabbay, C. J. Hogger, and J. A. Robinson, eds. New York: Oxford University Press, 1994.

- <http://www.informatik.uni- bremen.de/~ga/research/ruleml.html>.

关于规则的标记语言及其在语义网中的应用的一般信息可以参考 RuleML 网页：

- <http://www.dfki.uni-kl.de/ruleml/>.

比较详细地描述 RuleML 数据模型的文献是：

- H. Boley. The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. 2001. <http://www.dfki.uni-kl.de/~boley/ruleml-mht.pdf>.

TRIPLE 是为语义网设计的一个推理系统。详情请参考

- <http://triple.semanticweb.org/>.

练习与设计

5.1 参考 5.2 节的例子，定义谓词 `aunt`(姑妈)，`grandfather`(爷爷)，`sibling`(同胞兄弟)和 `descendant`(后代)。

5.2 考虑一个含节点和有向边的图，并且用事实 *edge*(*a*, *b*)来表示一条从节点 *a* 到节点 *b* 的有向边。定义一个二元谓词 *path*，它对节点 *c* 和 *d* 为真当且仅当图中存在从 *c* 到 *d* 的一条路径。

5.3 提出一种非单调规则和本体的组合，特别是提出一种集成，使得

(a)本体用来推出某些事实；

177 (b)可废止规则可以使用来自(a)的事实进行推理；

(c)规则头中的谓词不在本体中出现(就是说，规则只能使用、不能推出新的本体知识)。

5.4 为单调规则给出一种 XML 证明标记，包括替换和 SLD 推理的标记(如果熟悉 SLD 反驳)。

5.5 判断 RDFS 和 OWL 的哪种构造能用单调规则来表达。例如，子类关系可以用 $c(X) \rightarrow c'(X)$ 来表达(*c* 是 *c*' 的子类)。

178

第6章　应　　用

6.1　引言

本章描述一些应用。本书讲述的技术在这些应用中已经派上用场，或有潜在的用武之地。这里打算仅描述现实的场景，这些场景即便还没有实现，至少也是被各行业的大企业认真考虑过的。

本章将为语义网技术的各种用途给出一个广泛的概览，包括横向信息产品(horizontalinformation product)、数据整合(data integration)、技能寻获(skill-finding)、智囊门户网站(think tank protal)、在线学习(e-learning)、Web 服务(web services)、多媒体收藏索引(multimedia collection index)、在线采购(on-line procurement)和设备可共用性(device interoperatability)。

6.2　Elsevier 的横向信息产品

6.2.1　背景

Elsevier 是一家居于领先地位的科技出版商。它的产品和众多竞争者一样，主要是按照传统的刊物订阅的形式组织的。这些刊物的在线可获得性目前尚未真正改变生产线的组织方式。虽然作者的个人论文可以在网络上在线获得，但只能以它们在刊物上登载的形式，而文集则是按照刊登论文的刊物组织的。Elsevier 的用户可以订阅在线内容，但这种订阅仍然按照传统生产线的方式，组织成一本本刊物或刊物合订本的形式。

6.2.2　问题

这些传统刊物可以用纵向信息产品(vertical information product)一词来描述：产品分成许多相互分离的门类(如生物类、化学类、医药类等)，每个产品仅涉及其中一个门类的内容(其实往往只是一个门类中的一部分内容)。可是，随着各个学科(信息科学、生命科学、物理学等)的迅猛发展，这种不同杂志互不相干地分别覆盖不同学科的传统做法已经不再令人满意。Elsevier 的用户所感兴趣的是跨越多个传统学科的主题领域。例如，一家制药公司

可能要购买 Elsevier 的全部有关阿尔茨海默症(Alzheimer's disease)[㊀]的科研资料，而不关心这些资料是来自生物学刊物医学刊物还是化学刊物。这种情况下，所需求的是所谓的横向信息产品，即 Elsevier 所拥有的关于给定主题的全部信息，这种信息跨越了相互分离的所有传统学科和刊物的边界。

当前，像 Elsevier 这样的大出版商很难提供这样的横向产品。由 Elsevier 出版的信息产品被分割封闭在不同的刊物之内，每个刊物都有自己的一套索引系统，每套索引系统都是根据不同的物理标准、语法标准以及语义标准组织起来的。物理的和语法的壁垒是可以解决的。Elsevier 已经把许多产品的内容转译为允许跨杂志查询的 XML 格式。然而，语义的问题大部分还没有解决。当然，从多个刊物中搜索包含相同关键词的文章是可能的，但不太可能提供令人满意的结果，因为同一领域内或者不同领域间普遍存在同音词、同义词问题。我们需要的是用一个相容的概念集合为所有刊物建立索引，并据此搜索各个刊物的方法。

6.2.3 语义网技术的作用

本体和分类辞典(Thesaurus)(一种极轻量级本体)已被证明为高效信息访问的一种关键技术，通过把相关词条根据特定领域加以关联、编组以及为索引信息提供受控词汇表等方式，帮助解决自由文本搜索的一些问题。在各种不同的专业领域中，已经开发出许多分类辞典。医药信息领域的例子有 MeSH[㊁] 和 Elsevier 的生命科学分类辞典 EMTREE[㊂]，这些分类辞典已经被用来访问诸如 MBASE[㊃]和 Science Direct® 这样的信息源。但是，目前尚不存在多个不同的信息源与用于索引、查询这些信息源的分类辞典之间的链接。

180

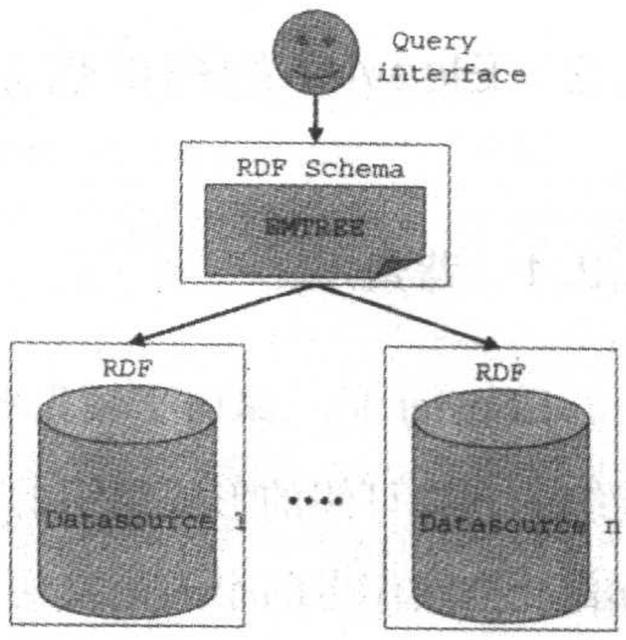


图 6-1 Elsevier 在数据源之间的查询

Elsevier 正在进行实验，探索借助单一界面访问生命科学领域中多个信息源的可能性。实验以 EMTREE 作为单一的底层本体，所有纵向信息源的索引均依据该本体(见图 6-1)。

语义网技术在这个体系结构中扮演多个角色。首先，RDF 被用作一种异质数据源之间的可共用格式。其次，本体(本例中是 EMTREE)本身也用 RDF 表示，尽管这并非它本来的

㊀ 又称早老性痴呆症。据 2007 年 6 月的一项研究显示，全球现有患者约 2660 万，预期 2050 年突破 1 亿。——译者注

㊁ <http://www.nlm.nih.gov/mesh>。

㊂ 42 000 个索引词条，175 000 个同义词词条。

㊃ <http://www.embase.com>；4000 个刊物，8 百万个记录。

格式。各个信息源都被映射到这个统一的本体之上，然后，这个本体即可成为所有这些信息源的单一入口。

这个问题并非仅为 Elsevier 所遇到，整个科技出版业都在为解决诸如此类的问题而努力。实际上，在试图使信息内容适应新的发送和组织形式方面，Elsevier 是先行者之一。 181

6.3　奥迪的数据整合

6.3.1　背景

上一节描述的问题本质上是一个数据整合问题。Elsevier 试图解决这一问题是为了顾客方便。但是在各个公司内部，数据整合也是一个巨大的问题。事实上，它被普遍认为是大公司信息技术预算中最昂贵的开支项目。如奥迪公司这样规模的企业(雇员 51 000 人、每年缴税额 220 亿美元、汽车产量 700 000 辆)，操作数千个数据库，常常对相同的信息多次重复存储，而且因为数据源没有相互连接而时有遗漏。现有处理方式依靠昂贵的手工代码生成和点对点的翻译脚本来完成数据整合。

6.3.2　问题

传统的中间件(middleware)固然能改进和简化整合过程，却需应对来自数据整合的根本挑战——基于预定含义(即数据语义)的信息共享。

6.3.3　语义网技术的作用

使用本体作为语义数据模型可以将各不相同的数据源合理地融合为单一的信息整体。通过为数据和内容源创建本体，以及添加通用的领域知识，整合企业内各不相同的数据源的工作就可以在不干扰已有应用软件的情况下进行。本体被映射到数据源(域、记录、文件、文档)，使得应用软件可以通过本体直接访问数据。

下面用一个照相机的例子来说明一般想法[⊖]。在一个特定的数据源或应用中表达照相机信息的一种方式如下：

```
<SLR rdf: ID = "Olympus-OM-10" >
    <viewFinder > twin mirror < /viewFinder >
    <optics >
        <Lens >
            < focal-length > 75-300mm zoom < /focal-length >
            < f-stop > 4. 0-4. 5 < /f-stop >
```

182

⊖　由 R. Costello 给出，见 http：//www. xfront. com/avoiding-syntactic-rigor-mortis. html。

```
        </Lens>
    </optics>
    <shutter-speed>1/2000 sec. to 10 sec.</shutter-speed>
</SLR>
```

一个人阅读了如上代码，可以得知 Olympus-OM-10 是一台 SLR(根据已有的知识，我们知道 SLR 是某款型号照相机)，有双镜取景器(viewfinder)，并给出焦距范围(focal-length)、光圈间隔(f-stop)以及最大和最小的快门速度(shutter-speed)。这一解释仅是由人类阅读者做出的，一台计算机不可能知道 Olympus-OM-10 是某一型号的 SLR，也不知道 75~300mm 是焦距范围。

这只是在语法上对信息编码的一种方式。另一个信息源可以使用一种完全不同的格式：

```
<Camera rdf: ID="Olympus-OM-10">
    <viewFinder>twin mirror</viewFinder>
    <optics>
        <Lens>
            <size>300mm zoom</size>
            <aperture>4.5</aperture>
        </Lens>
    </optics>
    <shutter-speed>1/2000 sec. to 10 sec.</shutter-speed>
</Camera>
```

人类读者可以看出以上两种格式表达同样的内容。毕竟我们知道 SLR 是一种照相机，而 f-stop 是 aperture 的同义词。当然，通过写一个从这一格式到另一格式的翻译器，可以为上述两个数据源提供一个简单特设的整合。但这只能解决这一特定的整合问题，而当我们遇到另一种照相机数据格式的时候，将不得不针对新格式重复相同的工作。

另一种做法是，用 OWL 写一个简单的照相机本体：

```
<owl: Class rdf: ID="SLR">
   <rdfs: subClassOf rdf: resource="#camera"/>
</owl: Class>

<owl: DatatypeProperty rdf: ID="f-stop">
   <rdfs: domain rdf: resource="#Lens"/>
</owl: DatatypeProperty>

<owl: DatatypeProperty> rdf: ID="aperture">
   <owl: equivalentProperty rdf: resource="#f-stop"/>
</owl: DatatypeProperty> >

<owl: DatatypeProperty rdf: ID="focal-length">
```

```
    <rdfs: domain rdf: resource = "#Lens"/ >
</owl: DatatypeProperty >

<owl: DatatypeProperty > rdf: ID = "size" >
    <owl: equivalentProperty rdf: resource = "#focal-length"/ >
</owl: DatatypeProperty > >
```

它的意思是说，SLR 是一种照相机，f-stop 是 aperture 的同义词，而 focal-length 是 lens size 的同义词。

现在，假定一个应用程序 A 在使用着第二种编码（照相机、aperture、lens size），且会收到应用程序 B 发送的以第一种格式编码（SLR、f-stop、focal length）的数据。当应用程序 A 解析从 B 得来的 XML 文档时，它遇到了 SLR。它“看不懂”SLR 这个词，所以“请教”照相机本体：“你知道 SLR 是什么吗?”本体回答“SLR 是一种照相机”。这个知识提供了令 A 能够“了解”一个它不知道的东西（SLR）和它已经知道的东西（照相机）之间的关系。当应用程序 A 继续解析，它遇到了 f-stop。这个词也看不懂，所以请教照相机本体：“你知道 f-stop 是什么吗?”本体回答：“f-stop 是 aperture 的同义词。”这个知识再一次在应用程序 A 知道的东西和不知道的东西之间搭建了桥梁。对于 focal length 也使用类似的方式处理。

这里的要点是，语法上不同的表达已经不再是障碍了。事实上，可以鼓励语法上不同的表达，这使得每个应用程序使用最适合它自身的语法表达方式。本体提供这些各不相同的语法格式之间的唯一整合，而不是 n 个不同格式之间的 n^2 对两两映射。

奥迪不是唯一一个为了解决数据整合问题而研究语义网技术的公司。波音、戴姆勒-克莱斯勒、惠普等其他大公司都面临着相同的问题（见推荐读物）。这个应用背景现在已经足 184
够现实，使得 Unicorn（以色列）、Ontoprise（德国）、Network Inference（英国）等遍布全球的 IT 公司正在把市场前景寄托在语义网技术的应用上。

6.4　瑞士人寿保险公司的技能寻获

6.4.1　背景

瑞士人寿保险公司在欧洲的人寿保险行业之中居于领先地位，全球雇员 11 000 人，账面保险费约 140 亿美元，其子公司、分支机构、代办处和合作伙伴遍布约 50 个国家。

在所有公司中，关于行业惯例的知识、个人能力和雇员的技能都是应对知识密集型任务的最重要的资源。它们是公司成功所凭借的真正底蕴。建立一个可以电子化访问的关于人的能力、经验和关键知识领域的信息库是建立企业知识管理的主要步骤之一。这样一个

技能信息库可以用于：搜寻拥有特定技能的人员、揭示技能差距和能力等级、指导作为职业生涯计划一部分的培训过程以及为公司的知识资本建立档案。

6.4.2 问题

公司有如此巨大的国际化工作人员群体，分布在地理和文化上各不相同的地区，使得建立整个公司的技能信息库成为一项困难的任务。怎样列出数量繁多的各种技能？怎样组织它们，使得对它们的访问可以跨越地理和文化的疆界？怎样保证该信息库经常更新？

6.4.3 语义网技术的作用

瑞士人寿保险公司在 On-To-Knowledge 项目中的实验(见推荐读物)使用手工建立的本体来覆盖公司的三个组织单位：信息技术、私人保险和人力资源。该本体共包含跨越这三个部门的700个概念，另有180个教育性概念和130个工作职能概念没有根据这三个领域进
185 一步细分。

这里给出该本体的一小部分，以对其表达方式作管中窥豹：

```
<owl: Class rdf: ID="Skills">
 <rdfs: subClassOf>
  <owl: Restriction>
   <owl: onProperty rdf: resource="#HasSkillsLevel"/>
   <owl: cardinality rdf: datatype="&xsd; nonNegativeInteger">
    1
   </owl: cardinality>
   </owl: Restriction>
 </rdfs: subClassOf>
</owl: Class>

<owl: ObjectProperty rdf: ID="HasSkills">
 <rdfs: domain rdf: resource="#Employee"/>
 <rdfs: range  rdf: resource="#Skills"/>
<owl: ObjectProperty>

<owl: ObjectProperty rdf: ID="WorksInProject">
 <rdfs: domain rdf: resource="#Employee"/>
 <rdfs: range  rdf: resource="#Project"/>
 <owl: inverseOf rdf: resource="#ProjectMembers"/>
</owl: ObjectProperty>

<owl: ObjectProperty rdf: ID="ManagementLevel">
 <rdfs: domain rdf: resource="#Employee"/>
 <rdfs: range>
```

```
    <owl: oneOf rdf: parseType = "Collection" >
      <owl: Thing rdf: about = "#member"/ >
      <owl: Thing rdf: about = "#HeadOfGroup"/ >
      <owl: Thing rdf: about = "#HeadOfDept"/ >
      <owl: Thing rdf: about = "#CEO"/ >
    </owl: oneOf >
  </rdfs: range >
</owl: ObjectProperty >

<owl: Class rdf: ID = "Publishing" >
  <rdfs: subClassOf rdf: resource = "#Skills"/ >
</owl: Class >

<owl: Class rdf: ID = "DocumentProcessing" >
  <rdfs: subClassOf rdf: resource = "#Skills"/ >                    186
</owl: Class >

<owl: Class rdf: ID = "DeskTopPublishing" >
  <rdfs: subClassOf rdf: resource = "#Publishing"/ >
  <rdfs: subClassOf rdf: resource = "#DocumentProcessing"/ >
</owl: Class >
```

瑞士人寿保险公司的每个雇员都被要求依据由技能本体驱动的表格填写方式建立“主页”。相应的实例集合可以通过使用生成的 RQL 查询(见第 3 章)的基于表格的界面来查询。

尽管这一系统未曾走出原型阶段，却被瑞士人寿保险公司总部有关部门的 100 个人(后来增加到 150 个人)使用着。

6.5　EnerSearch 的智囊门户网站

6.5.1　背景

EnerSeach 是一个致力于能源信息技术的工业研究联盟。它的目标是建立和传播先进的 IT 技术将如何影响能源公用部门的知识，尤其立足于使这些部门实现全欧洲的市场化。

EnerSeach 的构成有别于传统的研究机构，它的研究项目是由不同国家(瑞典、美国、荷兰、德国和法国)里不固定的各个研究团体执行的，其中有很多团体虽然从事 EnerSearch 资助的工作，却不是它的雇员。从这个意义上来说，EnerSearch 更像一个虚拟组织，因为它所进行的研究是为了满足公用产业(utility industries)与 IT 厂商的兴趣。EnerSearch 具有有限公司的形式，被该公用产业中一些对正在进行的研究有迫切需求的公司所拥有，其股东包括欧洲各国的大公用公司：瑞典的 Sydkraft、葡萄牙的 EDP、荷兰的 ENECO、西班牙的

Iberdrola和德国的 Eon，以及该公用产业的一些全球 IT 供应商如 IBM、ABB 等。由于其组
187 成上广阔的地域跨度，从知识分布的观点看，EnerSearch 也具有虚拟组织的特性。

6.5.2 问题

传播知识是 EnerSearch 的一个重要任务。EnerSearch 网站是知识传播的一条重要途径(实际上，EnerSearch 的一个股东公司正是在了解了它的网站后才加入 EnerSearch 的)。虽然如此，其网站的信息结构仍有很多不足之处，它的主要组织在网站上是以术语“关于我们(about us)”的形式表述的，其内容包括：已经完成的工程，参与工作的研究人员，文章、报告和介绍。然而，这样的信息并不能满足信息搜寻者的需求。信息搜寻者一般不关心工程的内容或者作者等知识，而致力于寻找相关工业领域里重要问题的答案，如：负载管理(load management)能否节约开销？如果能的话，能节约多少？预付的投资需要多少？输电线通信(powerline communication)能在技术上对 ADSL 或电缆调制解调器(cable modem)构成竞争吗？

6.5.3 语义网技术的作用

EnerSearch 网站实际上是被具有不同需求的团体使用着，即这个领域的研究人员、公用产业的职员与管理人员等。为这些群体构建其相关的主题和问题的清晰视图是完全可能的，而且 EnerSearch 涉及的知识领域都被很好地定义了。基于这些因素，定义足够稳定且质量足够好的领域本体是可以实现的。实际上，On-To-Knowledge 项目使用一个轻量级的“EnerSearch 午餐时间本体”(EnerSearch lunchtime ontology)成功地进行了一些实验，该本体的开发顶多只需要几个小时(即“午餐时间”)。

这个轻量级本体只有一个分类层次结构(taxonomical hierarchy)，因此只需要 RDF Schema 的表达能力。下面用非形式化记号来粗略描述这个本体的一个分支：

```
...
 IT
   Hardware
   Software
   Applications
   Communication
     Powerline
     Agent
   Electronic Commerce
   Agents
188    Multi-agent systems
   Intelligent agents
   Market/auction
   Resource allocation
```

Algorithms

这个本体被 EnerSearch 网站以多种不同方式用于其导航工具之中。图 6-2 显示了 EnerSearch 网站关于概念“agent”的子概念的语义图，图 6-3 显示了不同作者由于研究和出版领域的区别而产生的语义距离。[⊖]

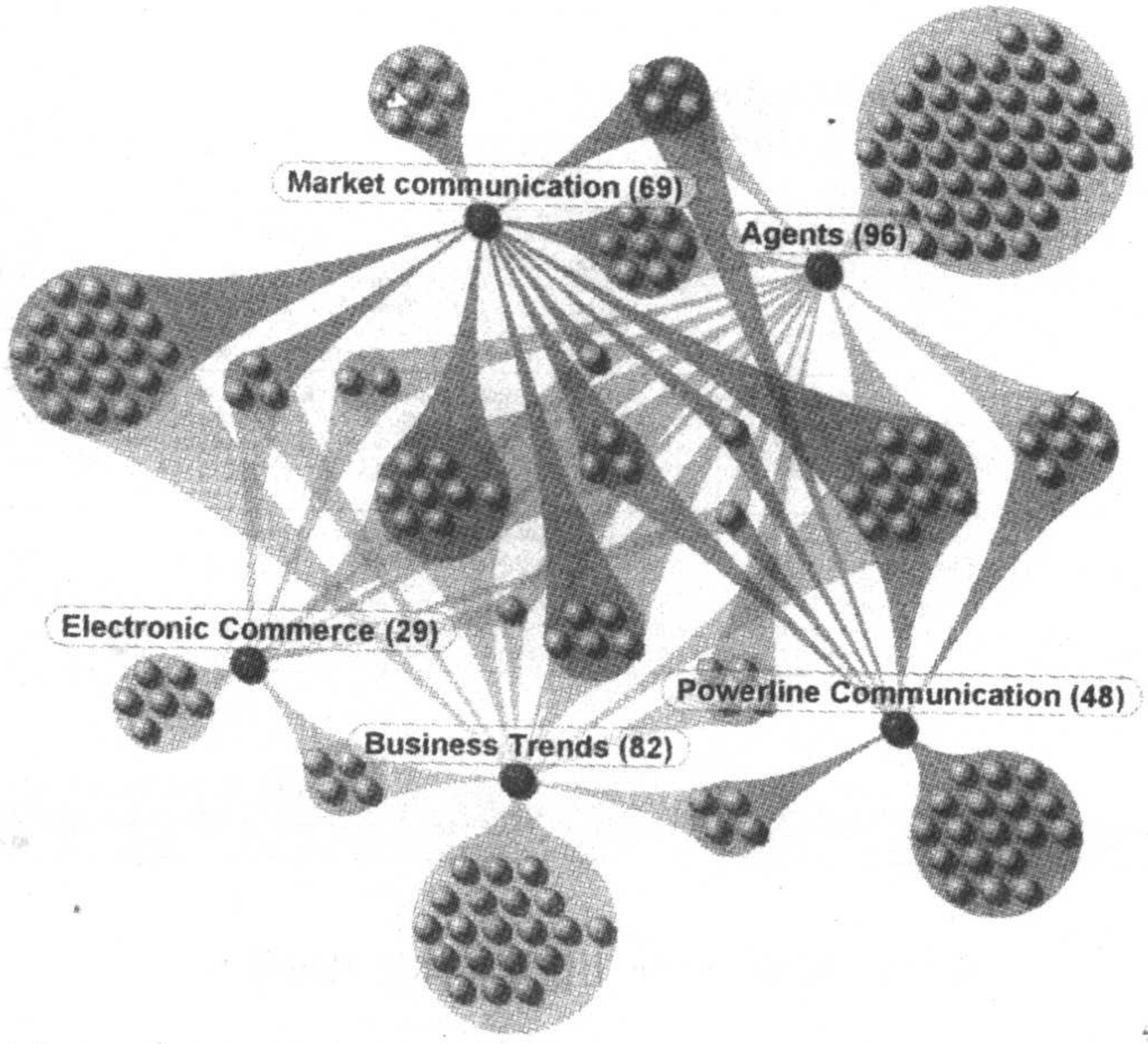


图 6-2　EnerSearch 网站的部分语义图

189

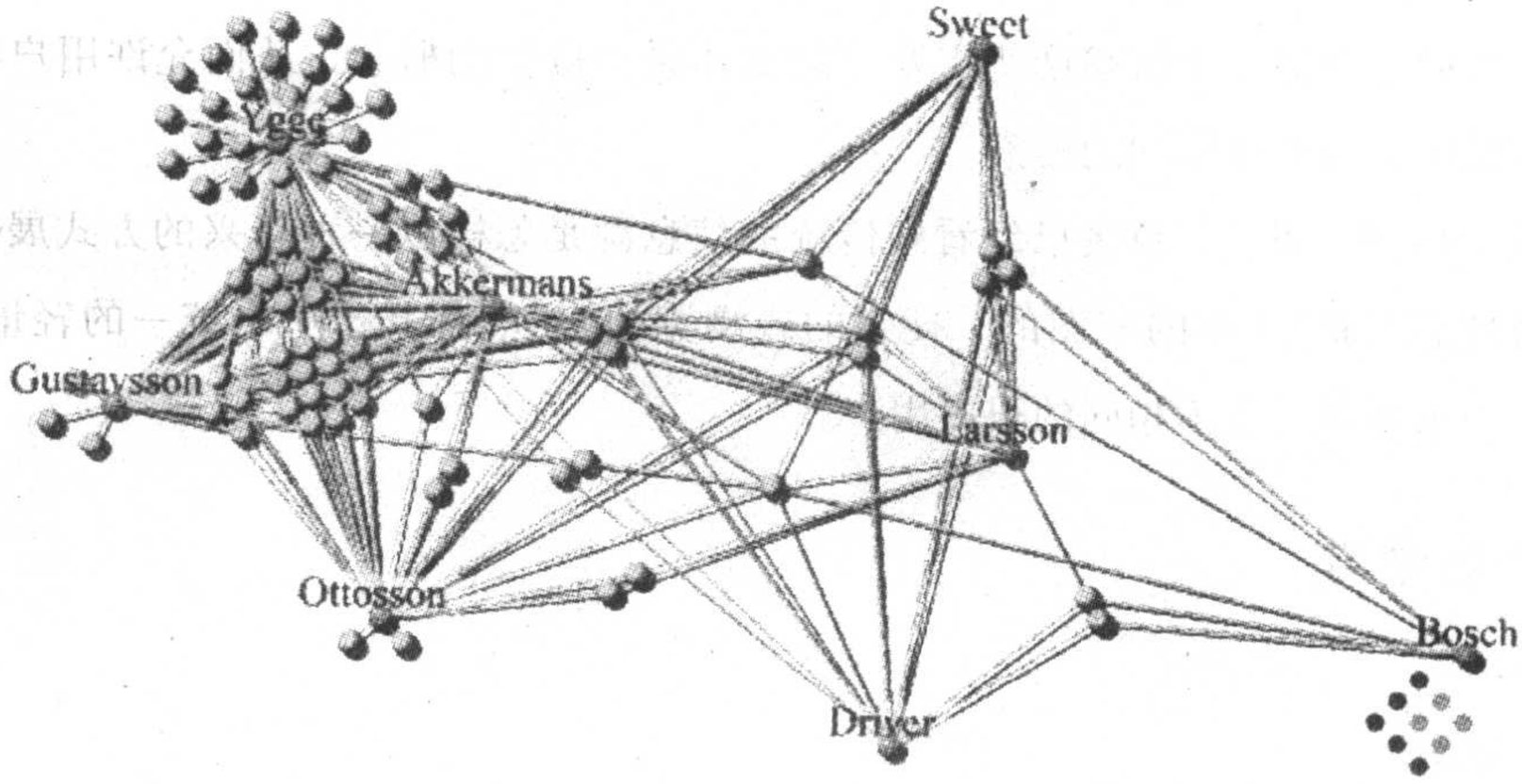


图 6-3　EnerSearch 作者的语义距离

⊖　两个图都是借助 Aduna 的语义聚类可视化软件得到，见 <http：//www. aduna. biz >。

图6-4是相同信息用Spectacle Server语义浏览软件以一种完全不同的方式产生的显示[⊖]。该用户选择“按作者”选项，然后选了作者“Fredrik Ygge”和概念“电缆长度”(cable length)。结果列出了Fredrik Ygge关于“电缆长度”这个主题的所有出版物。

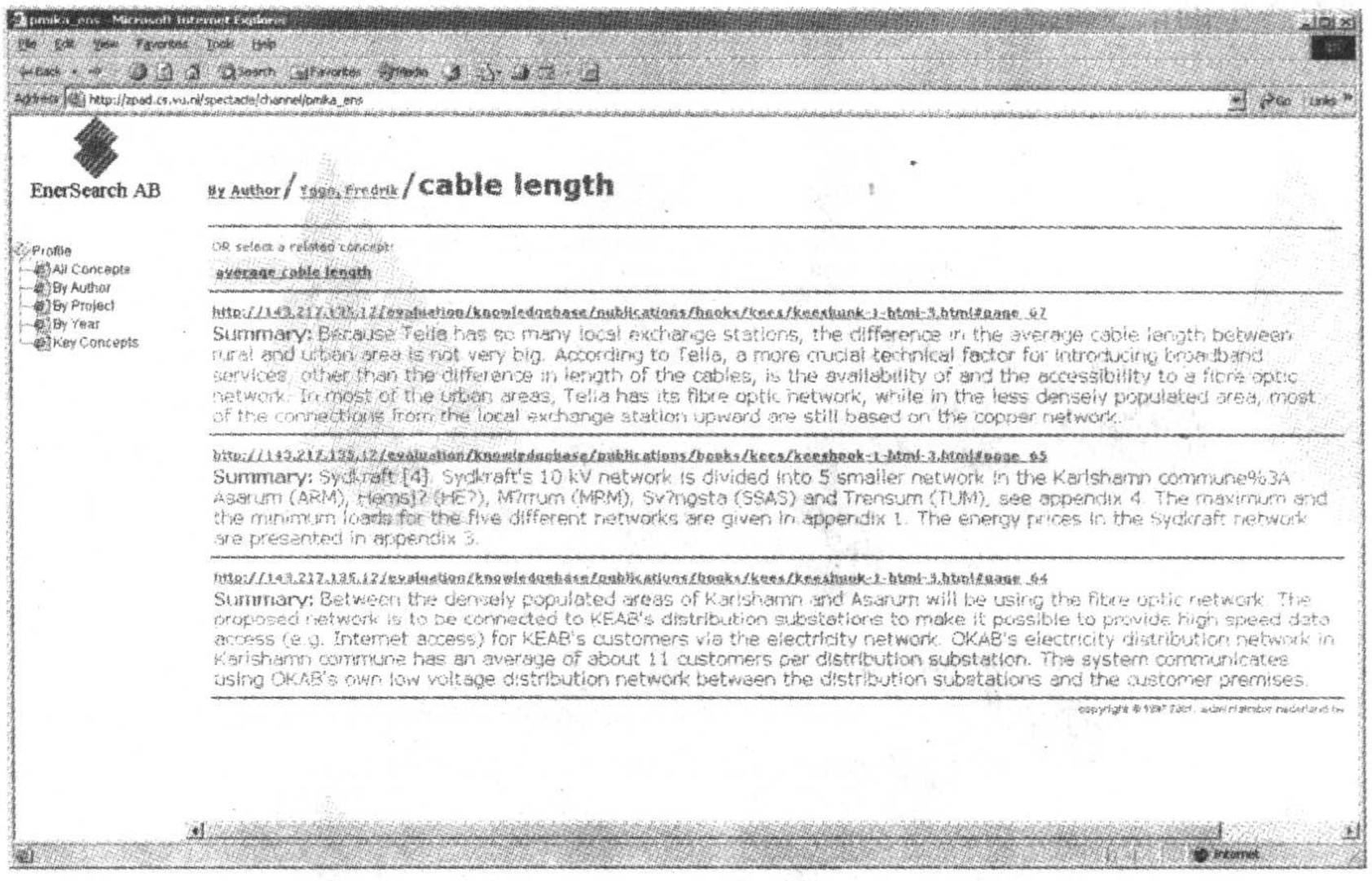


图6-4 在Spectacle上浏览本体组织的页面

第三种显示信息的方法是使用QuizRDF工具[⊜]。与完全基于本体的显示(如前面三个图所示)和没有任何语义基础的传统关键词搜索不同，QuizRDF的目的是两者的结合——用户可以键入一般关键词。这将使结果列表如同传统搜索结果一样，显示出一系列包含关键词的文章，但同时也显示出描述这些文章在层次体系中位置的概念，从而允许用户在关键词搜索的结果中进行本体驱动的搜索。

在这个应用场景里，读者已经看到传统的信息源是怎样以一种新兴的方式展现的。所有这些展现的机制(文本的和图的，搜索的或浏览的)虽然都基于一个统一的轻量级本体，但适合于需求和背景各不相同的很多用户。

6.6 在线学习

6.6.1 背景

万维网正在不断地改变着人类活动的许多方面，其中包括人们的学习。传统学习具有

⊖ 来自Aduna，见<http://www.aduna.biz>。

⊜ 原型由英国电信研究实验室提供。

以下几个特点：

- 由教育者驱动。由教师选择学习内容和教学方法，确定教学计划和学习的进度。
- 线性获取知识。按照预先确定的顺序传授知识，学习者很难偏离这个顺序而选择自己感兴趣的内容。
- 时间和地区依赖。在特定的时间和特定的地方进行学习。

结果，学习是面向大部分参与者而非个性化的，因而并不适合于每一个有天赋的学生的具体情况，尽管这种传统学习过程效率高并且在很多场合是有效的。因特网的出现为全新教育方式的实现铺平了一条道路。

190 | 191

人们已经可以在高等教育中看到这种变化。大学正逐步把活动重心转移到为学生提供更多的灵活性上来。虚拟大学和在线课程只是这些活动中的一小部分，教育灵活性和新的教育方式同样在传统的校园内得以施行，虽然这些学校仍然要求学生到课堂上课，但对他们的限制减少了。渐渐地，学生就能够选择并且决定上课的内容和评估规程，确定学习的进度，采用最适合自己的学习方法。

我们同样期待在线学习会对企业员工的业务素质(work-related qualification)和个人的终生学习活动产生更大的影响。提高企业竞争力的一个重要机制是提高员工的技能。企业需要及时的、满足其特殊需求而且能够完美地融入日常工作方式的学习过程。传统的学习方式不能满足这些需要，而在线学习很有希望解决这个问题。

6.6.2　问题

与传统教育不同，在线学习不受教师的驱动，学生能够以一种非预定的顺序获得学习资料，并通过选择学习资料自己编排课程。要想实现这种方式，必须给学习资料附加额外信息，以支持有效的索引和检索。

元数据的运用是解决上述问题的一种自然方式，并且长期以来被图书管理员或多或少地使用着。在在线学习界(e-learning community)，像 IEEE LOM 这样的标准已经出台。它与学习资料的信息相关联，比如教育和教学属性、访问权限和使用条件、以及与其他教育资源的关系等。虽然这些标准是有用的，但它们面临着与所有完全基于元数据的解决方案(XML 途径)一样的困扰——缺少语义。结果，组合来自不同作者的资料可能很困难，对检索的支持可能不是最佳的，并且学习资源的检索和组织必须人工完成而不是个性化地自动代理(Personalized automated agent)实现。如果使用语义网技术，就能避免这些问题。

192

6.6.3　语义网技术的作用

语义网的关键思想，即语义共享(本体)和机器可处理(machine-processable)的元数据，为满足在线学习的需求提供一条有希望的途径，它能支持学习资料的语义查询和概念化导

航，并具有下列特点：

- 由学习者驱动。不同作者编著的学习资料能够与公认的本体相链接。学习者可以通过语义查询设计自己的个性化课程，并且可以根据实际问题的上下文来检索学习资料。
- 灵活的资料获取。能够按照学习者的兴趣和需要以他们希望的顺序获得知识。尽管在一定条件下语义标注仍然会设置某些限制，但非线性获取方式在总体上是有保证的。
- 集成性。语义网能够为各个机构的商业过程提供一个统一的平台，并将学习活动集成到这些过程之中。这个解决方案或许对商业公司来说极具价值。

6.6.4 在线学习的本体

在在线学习环境中，经常出现不同作者使用不同术语表(terminologies)的情况，使得不同学习资料的结合变得困难。而教师与学生之间知识背景和层次上的典型差异使得这个检索困难更加突出。因此，需要建立一种共享的理解机制，而本体正是实现这个目标的强有力机制。在在线学习环境，区分内容(content)，教学(pedagogy)和结构(structure)这三类知识是有意义的。因此，需要相应地区分三种本体。

内容本体(content ontology)描述学习领域(比如历史或计算机科学)的基本概念，同时包含这些概念之间的关系和一些基本属性。例如，对古代雅典的研究是古希腊历史研究的一部分，又是古代史研究的一部分。这个本体应该包含关系“是……的一部分”，并且这种
193 关系应具有传递性(如在 OWL 中的表达)。于是，学习支持自动代理(automated learning support agent)就能推知，有关古代雅典的知识可以在古代史下面找到。这个内容本体同样可以准确表示同义、缩写等关系。

教学方面的问题可以用教学本体(pedagogy ontology)处理。例如，学习资料可以按讲义、辅导、示例、课堂互动指导、习题和解答等进行归类。最后，结构本体(structure ontology)用来定义学习资料的逻辑结构，其中的典型知识包括层次关系和导航关系如“上一步”、“下一步”、“有……为其一部分”(hasPart)、“作为……的一部分”(isPartOf)、“要求”以及“基于”。这些关系间的关系同样可以定义，比如“有……为其一部分”和“作为……的一部分”之间是逆关系。在网络上开发在线学习系统是十分自然的，因此，应该使用一个网络本体语言。

应该指出，不能期望从学习本体得出的大多数推论具有深层次的含义。人类能够轻松处理“有……为其一部分”和“作为……的一部分”之类的关系以及它们的相互作用。但是，关键在于，这样的推理需要由自动代理(automated agent)完成，而这种语义信息对于自动代理的推理是必须的。

6.7　Web 服务

6.7.1　背景

说到 Web 服务(web services)，意味着网站不仅要提供静止的信息，还要涉及与用户的互动并且常常允许用户采取一些行动。通常在简单 Web 服务和复杂 Web 服务之间加以区别。

简单 Web 服务涉及单一的网络可访问程序、感知器或者装置，它们不依赖于其他 Web 服务，除了对用户需求的一次性响应之外，也不需要与用户做进一步互动。典型例子包括信息供应服务，例如返回给定地址和邮政编码和航班查询服务。

复杂 Web 服务由简单 Web 服务组成，通常需要与用户进行持续的在线交互，以便用户做出选择或者根据情况提供相应信息。例如，用户与一家在线音乐商店的交互包括：根据各种标准寻找 CD 和标题、阅读评论文章和试听音乐样本、把选定的 CD 添加到购物篮里、提供信用卡细节以及装运详单和交付地址。 194

6.7.2　问题与语义网技术的作用

目前，使用 Web 服务需要人的参与，例如需要人们浏览信息和填写表格。面对 Web 服务，语义网蓝图着眼于通过提供机器可理解(machine-interpretable)的服务描述，实现对 Web 服务的发现、调用(invocation)、组合和监控的自动化。

网站应该能够通过声明和描述服务(即服务本体)来使用一组基本类和属性。DAML-S 是开发面向 Web 服务的本体语言的开端。它使用了 DAML + OIL，也就是说它可以被看作 DAML + OIL(一个 DAML + OIL 应用)的上层应用。DAML-S 目前还在发展之中(比如说 DAML-S 已经移植到 OWL 上)，所以，这里不介绍它的技术细节而集中介绍它的基本思想。

有三种基本类型的知识与服务相关联：服务概况(service profile)、服务模型(service model)和服务基底(service grounding).

服务概况是对服务的功能和要求的描述，在某种意义上，也就是服务的说明。这些信息对服务发现(service discovery)来说是必不可少的。根据服务概况，一个服务搜寻代理就可以判断这个服务是否满足自己的需要。值得注意的是，一个服务概况可以不是对一个现有服务的描述，而是由服务需求者提供的对服务需求的说明。

服务模型描述这个服务是如何进行的，也就是说，当提供这个服务的时候会发生什么事情。这些信息对于支持服务搜寻代理去组合一系列服务，以完成一件复杂任务并监控服

务的执行来说，可能是重要的。

服务基底规定了代理如何访问该服务的细节信息。典型情况下，一个服务基底要规定一个通信协议和用于联络该服务的端口号。

下面简要讨论 DAML-S 的服务概况和服务模型。

服务概况

服务概况提供了一种方式，以描述网站供给的服务和用户需要的服务，从而支持供给
195 与需求之间的匹配。一般情况下，一个 DAML-S 的服务概况提供以下信息：

- 关于服务及其提供者的用户可读(human-readable)的描述
- 服务提供的功能规范
- 诸如预期响应时间和地域限制之类的附加信息。

所有这些信息都用 DAML-S 的建模原语——DAML-S 类和属性来编码，这些类和属性又使用 DAML + OIL 语言定义。例如，某服务提供的一个功能是类 offeredService 的一个实例，定义如下：

```
<rdfs: Class rdf: ID = "OfferedService" >
    <rdfs: label > OfferedService </rdfs: label >
    <rdfs: subClassOf rdf: resource = "http: //www. daml. org/
        services/daml-s/2001/10/Service. daml#"/ >
</rdfs: Class >
```

这个类里定义了一组属性：intendedPurpose，serviceName 和 providedBy。前两个属性的值域为字符串，第三个属性的值域为一个新的有各种属性的类 ServiceProvider。下面是一个实例的简单例子：

```
<profile: ServiceProvider rdf: ID = "SportsNews" >
    <profile: phone > 1234 5678 </profile: phone >
    <profile: fax > 1234 5679 </profile: fax >
    <profile: email > abc@ defgh. com </profile: email >
    <profile: webURL > www. defgh. com </profile: webURL >
    <profile: physicalAddress > 150 Nowhere St,
        111 Somewhere, Australia </profile: PhysicalAddress >
</profile: ServiceProvider >
```

服务概况的功能描述定义一些属性，这些属性描述该服务提供的功能。主要的属性有：

输入(input)，描述提供服务所需的必要参数。例如，体育新闻服务可能需要以下输入：日期、运动项目、顾客信用卡的细节。

输出(output)，规定服务的输出。在体育新闻的例子中，输出可以是给定日期某一
196 个体育项目的新闻文章。

前提(precondition)，规定有效地执行服务所需要的条件。输入和前提之间的区别

可用当前这个例子说明：信用卡的细节是输入，而信用卡有效且没有透支是前提。

效果(effect)，描述服务的效果的属性。在我们的例子里，每看一篇新闻文章从信用卡里扣 1 美元是一个效果。

由于现有 DAML + OIL 语言的限制(这些限制在 OWL 同样存在)，DMAL-S 的建模原语对服务的功能描述是非常有限的。比如，无法像软件规范那样定义输入与输出之间的逻辑关系。一旦借助逻辑设施(如规则)加强了网络本体语言，DAML-S 的开发者将尝试实现这种可能性。

服务模型

服务模型的核心概念是过程(process)，它用输入、输出、前提、效果以及子过程的复合(如果有的话)来描述服务。上面已经讨论了服务概况的输入、输出、前提和效果，所以，这里将注意力集中于简单过程到复杂过程的复合。

图 6-5 显示过程本体的上层结构。我们可以看到上层的类 Process 和它的三个子类：

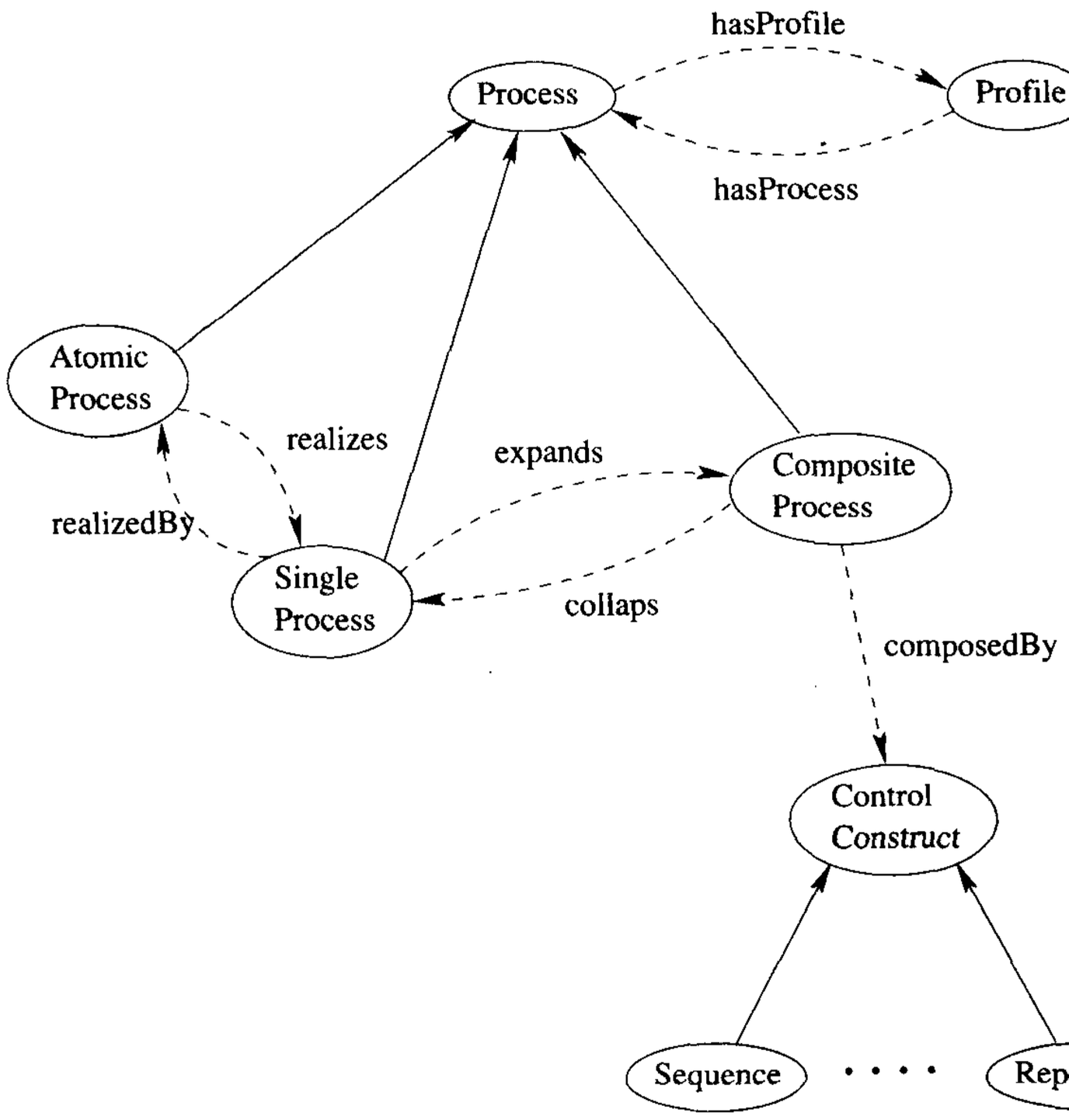


图 6-5　过程本体的上层结构

- 原子过程(atomic process)能够通过向它们传递适合的信息直接调用，它们在一步内执行完毕；

- 简单过程(simple process)是抽象元素，可视为能够单步执行但不是可调用的(invocable)；
- 复合过程(composite process)由其他较简单的过程组成。

图6-5中的几个属性说明如下：

- hasProfile 和 hasProcess 是两个描述过程与其概况间关系的属性。
- realizes 和 realizedBy 表示简单过程可以通过原子过程实现。
- expands 表示简单过程也可以用于抽象目的，并扩展为一个复合过程；expands 的逆属性是 collaps。

最后，一个复合过程由一些控制结构(control construct)组成：

```
<rdf: Property rdf: ID = "composedBy" >
    <rdfs: domain rdf: resource = "#CompositeProcess"/ >
    <rdfs: range rdf: resource = "#ControlConstruct"/ >
</rdf: Property >
```

197～198

DAML-S 目前提供的控制结构包括 sequence，choice，if-then-else 和 repeat-until。

对于服务概况来说，DAML-S 的过程模型仍然在发展之中。

AI 和 Web 服务

Web 服务是一个可以有效地运用人工智能技术的应用领域，例如服务功能与服务需求的匹配，以及在可以运用自动规划的场合，将简单服务组成复杂服务。在推荐读物里还能找到一些相关的参考文献链接。

6.8 其他场景

本节简要描述正在被产业界和研究机构考察的其他一些应用场景。

6.8.1 伦敦警察局的多媒体收藏索引

在诸如伦敦警察局(Scotland Yard)和国际刑警组织(Interpol)这样的公安机关里，有特殊部门专职负责艺术品和古玩盗窃的案件。这类盗窃案件的案犯往往难以追踪，即便成功追踪并追回被盗物品，怎样物归原主也是一个出人意料的难题。虽然现在已经建立了被盗艺术品的国际数据库，但要在这些数据库中找到特定的物品仍然不容易，因为不同团体常常会提供不同的描述。一个举报盗窃案的博物馆也许会把一个物品描述为“宋代莲花瓷瓶”，而一位警官在报告一件寻获的物品时却很可能简单地输入“12.5英寸高有叶状装饰的浅绿色花瓶”。目前，需要人类专家来鉴别失窃花瓶是否就是寻获的花瓶。

解决办法的一部分工作是开发受控的词汇库，例如 Getty Trust 的《艺术与建筑分类辞

典》(AAT)[一]，或 Iconclass 分类辞典[二]，把它们扩展为成熟的本体，开发能够使用本体背景知识根据物理外观描述对各类物品进行自动鉴别的软件，并解决不同团体使用不同本体描述相同物品时出现的本体映射问题。 199

6.8.2　戴姆勒－克莱斯勒的在线采购

与现今所有汽车生产厂商一样，戴姆勒-克莱斯勒为了获得各种汽车零配件要与几百家供货商进行交互。近年来，在线采购已被认为是一种降低成本的重要潜在手段。例如，相互发送合同、订单、发票和财务转账等纸面过程可以变更为在软件应用程序之间进行数据交换的电子过程。同样，与一些固定供货商之间静态的、长期的合作关系可以变更为开放市场上竞争的、动态的、短期的合作关系。每当一个供货商提出了更好的价格，戴姆勒-克莱斯勒公司希望能够立刻与之交易，而不是被困锁在与另一个供货商之间的长期协议之内。

这种在线采购是企业对企业(B2B)电子商务背后最主要的动力之一。B2B 电子商务的现有努力严重依赖于数据格式的预先标准化，也就是说，依赖于整个行业事先对数据格式及其预期语义达成一致。像 Rosetta Net[三]这样的组织是专门负责这种标准化工作的。从 Rosetta Net 的网站上可以看到：

> Rosetta Net[是]一个自筹资金的非营利组织。[它]是一个为信息技术、电子元件、半导体制造和电信等产业的大公司创建和实施全行业开放电子商务过程标准而工作的协会。这些标准形成一个通用的电子商务语言，以协调全球供应链中合作伙伴之间的过程。

由于这些数据格式是用 XML 规定的，单从文件本身读不出任何语义，合作伙伴必须在耗时而且昂贵的标准谈判中达成协议，然后通过手工编码将数据格式所规定的语义写入代码之中。

一个更有吸引力的途径是使用 RDF Schema 和 OWL 这样的格式，因为它们具有显式定义的形式语义。这将使产品描述“自带语义”，从而开启远比现行方式更加自由的在线 B2B 采购过程之门。 200

6.8.3　诺基亚的设备可共用性[四]

近年来，日常环境中数码设备经历了爆炸式的发展——PDA、手机、数码相机、笔记本电脑、公共场所的无线访问，汽车之中的 GPS 设备，等等。在这一背景之下，对这些设

[一] <http://www.getty.edu/research/tools/vocabulary/aat>。

[二] <http://www.iconclass.nl/>。

[三] <http://www.rosettanet.org>。

[四] 本节基于 OWL 需求文档中的一个用例，见推荐读物。——译者注

备之间的可共用性的需求正在变得越来越强烈。这些设备需求具有遍布各地和无线通信的特点，因而需要支持自动、特设配置的网络体系结构。

真正的特设配置网络的一项关键技术是服务发现(service discovery)，即描述、发布和找到其他服务的能力，这里的服务包括蜂窝电话、打印机、传感器等各种设备提供的功能。现有各种设备发现和能力描述机制(例如 Sun 的 JINI、微软的 UPnP)基于特设表示模式，而且严重依赖于标准化(事先确定所有在通信或讨论中可能涉及的东西)。

比这种事先标准化更有吸引力的是“即兴可共用性”(serendipitous interoperability)，这是在“不曾排练过的”条件下的可共用性，也就是说，设备的设计不一定非要考虑到协作，可以在不同的时间由不同的厂商为了不同的目的生产出来，却仍然能够互相发现和利用对方的优势功能。“理解”其他设备并且推断其服务或功能是必要的，因为未来成熟的无所不在的计算场景将涉及数十乃至数百个设备，对这样的场景进行事先标准化是极其困难的。

与在线采购的场景相似，为了实现这种对功能“即兴的”相互理解，需要借助带有标准语义的本体。

推荐读物

一本有关本体在电子商务和知识管理中应用的非技术性书籍：

- D. Fensel. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. New York: Springer 2001.

201 OWL 的用例文档反映了 W3C 网页本体工作组定义 OWL 的动机：

- J. Heflin. OWL Web Ontology Language Use Cases and Requirements. August 18, 2003.<http://www.w3.org/TR/webont-req/>.

下面这本书描述了 On-To-Knowledge 项目中三种不同的用例分析(case-study)，关于这个项目更多的信息可以在 <http://www.ontoknowledge.org> 上查到。

- J. Davies, D. Fensel, and F. van Harmelen. *Towards the Semantic Web: Ontology-Driven Knowledge Management*. New York: Wiley, 2003.

一组关于语义网技术工业应用的文章可以在国际语义网会议的工业序列(Industrial Track)中找到，文章的收录从 2003 年的会议开始：

- D. Fensel, K. Sycara, J. Mylopoulos, eds. *Proceedings of the 2nd International Semantic Web Conference*. New York. Springer, 2003. Lecture Notes in Computer Science, volume 2870.

一篇关于在线学习语义网的推荐文章：

- L. Stojanovic, S. Staab and R. Studer. eLearning Based on the Semantic Web. In *Proceedings of WebNet 2001 - World Conference on the WWW and the Internet*.
  <http://www.aifb.uni-karlsruhe.de/WBS/Publ/2001/WebNet_lstsstrst_2001.pdf>

两篇关于语义网门户应用的参考文章：

- S. Staab et al. Semantic Community Web Portals. In *Proceedings of the 9th International WWW Conference*. 2000
  <http://www9.org/w9cdrom/134/134.html>
- N. Stojanovic et al. SEAL – A Framework for Developing SEmantic PortALs. In *Proceedings of the 1st International Conference on Knowledge Capture (K-CAP)*. 2001.
  <http://www.aifb.uni-karlsruhe.de/WBS/Publ/2001/sealkcap2.pdf>

关于 DAML-S 和支持 DAML 的 Web 服务主页是

- <http://www.daml.org/services/>

一些相关的出版物：

202

- The DAML Services Coalition. DAML-S: Web Service Description for the Semantic Web. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*. New York: Springer 2002. Lecture Notes in AI Volume 2342.
  <http://www.daml.org/services/ISWC2002-DAMLS.pdf>
- M. Paolucci et al. Semantic Matching of Web Services Capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*. New York: Springer 2002. Lecture Notes in AI Volume 2342.
  <http://www.daml.org/services/ISWC2002-Matchmaker.pdf>
- S. McIlraith, T.C. Son, and H. Zeng. Mobilizing the Semantic Web with DAML-Enabled Web Services. In *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb 2001)*.
  <http://www.daml.org/services/SemWeb01-KSL.pdf>

一些提供工具的有用网站：

<http://business.semanticweb.org>提供了语义网技术应用和提供商的信息的好资源。

<http://www.daml.org/tools> 是一个大型工具库，尽管目前限于 DAML + OIL，语言，但其中很多工具将升级到 OWL 语言。

<http://www.w3.org/2001/sw/WebOnt/impls#Implementations> 和 <http://www.cs.man.ac.uk/~horrocks/OntoWeb/SIG/node3.html>罗列了 OWL 标准稳定之后的第一批工具.

<http://www.ilrt.bris.ac.uk/discovery/rdf/resources/>。RDF 和 RDF Schema 的工具、工程和应用。

203

第7章　本体工程

7.1　引言

本书主要介绍语义网的基本技术——表示语言、查询语言、转换和推理技术以及工具。显然，引进如此大量新工具和新技术将同时提出方法学问题——怎样才能最好地应用这些工具和技术？哪些语言和工具应该按照哪种顺序用于哪些场合？质量控制和资源管理会有哪些问题？

关于语义网的许多这样的问题已在其他背景下得到了研究，比如软件工程、面向对象设计以及知识工程等。对这些问题的全面处理超出了本书的范围。因此，本章简要讨论本体构建(特别是手工本体构建)、现有本体复用和半自动化方法使用中出现的一些方法学问题。

7.2　手工构建本体

关于手工构建本体的讨论主要参考 Noy 和 McGuinness 的文章“Ontology Development 101：A Guide to Creating Your First Ontology”。更多的参考文献见推荐读物。

204
\
205

本体的开发过程可以分为下面一些主要阶段：

1. 确定范围
2. 考虑复用
3. 列举术语
4. 定义分类
5. 定义属性
6. 定义侧面
7. 定义实例
8. 检查异常

这个过程与其他开发过程一样，在实施中不是线性的，上述步骤可能会有反复，在过程中的任何一点都有可能退回到早期的某一步。这里不深入讨论这个复杂过程的管理，而是转入各个单独步骤的介绍。

7.2.1　确定范围

开发一个领域本体本身不是目的，就好比定义一组数据及其数据结构为其他程序使用一样。换句话说，本体是为特定目的而建立的特定领域的模型。因此，不存在特定领域的

“正确”本体。本体必须是特定领域的一种抽象，而选择其他抽象总是可能的。抽象中包含的内容取决于这个本体将被如何使用以及预期的扩展。在这个阶段要回答的基本问题是：这个本体将覆盖什么领域？使用该本体的目的是什么？该本体将为什么类型的问题提供回答？谁将使用和维护这个本体？

7.2.2 考虑复用

随着语义网的快速推广，本体也将被更加广泛的利用。在定义本体时，已经很少从零开始。几乎总能从第三方获得可用的本体，至少可以作为构建自己本体的一个有用的起点（见7.3节）。

7.2.3 列举术语

实际定义本体的第一个步骤是写下在该本体中期望出现的所有相关术语的一个非结构化列表。例如，名词是类名的基础，动词或动词短语是属性名（如 *is part of*、*has component* 等）的基础。 206

梯度分析和网格分析（laddering and grid analysis）等传统知识工程工具可以很好地运用于这个阶段，以建立术语集和这些术语的初始结构。

7.2.4 定义分类

在确定了相关术语之后，这些术语必须组织成一种分层结构。至于用自顶向下还是自底向上方式来做这个工作更高效/可靠，还存在不同的看法。

当然，确保这个分层结构是类与子类的层次结构是重要的。这就是说，如果 A 是 B 的子类，那么每个 A 的实例也必须是 B 的一个实例，只有这样才能确保 `owl: subClassOf` 和 `rdfs: subClassOf` 等原语的内嵌语义得到遵守。

7.2.5 定义属性

这一步经常要和上一步交叉进行。在将一些类组织成层次结构时，自然要同时建立关联这些类的属性。

回忆 `subClassOf` 关系的语义要求——只要 A 是 B 的一个子类，则对任何 B 实例成立的每一个属性陈述对任何 A 实例也都成立。由于这种继承性，可以把属性加在它所适用的最高层类上。

对类加属性时，应该立即陈述这些属性的定义域和值域。这里需要考虑一般与具体之间的折中。一方面，给属性尽可能一般的定义域和值域，使之通过继承应用于子类，是有吸引力的；另一方面，尽可能窄地限定定义域和值域，有助于通过检查定义域和值域的违

规来检测出本体中潜在的不一致性和概念偏差。

7.2.6 定义侧面

值得注意的是，以上几个步骤后，只需要 RDF Schema 提供的表达能力而不需要使用
207 OWL 增加的原语。这种情况在当前步骤中改变了，这里需要对以上步骤中定义的属性进一步规定它们的“侧面”(facet)，从而使属性定义更加完整。三类侧面是

- 基数——为尽可能多的属性规定是否允许或必需有一定数目的不同值。常见的情况包括“至少有一个值”(即“必要属性”(required properties))和“至多有一个值”(即单值属性)。
- 特定值——经常用某个属性所具有的一些特殊值来定义类，这些特定值可以用 OWL 中的 owl: hasValue 加以指定。有时这个要求可以放宽，规定属性的值可以取自一个给定的类(owl: someValuesFrom)，而不必是一个特定值。
- 关系特性——最后一类侧面涉及属性的关系特性，包括对称性、传递性、逆属性和函数值。

在本体构建过程的这一步骤完成之后，就可以检查本体内在的不一致性了(在此之前是不可能的，因为 RDF Schema 还不足以表达这些不一致性)。在传递属性、对称属性或逆属性的定义域和值域的定义中出现矛盾，是不一致性的常见情况。类似地，基数属性也常常是不一致性的来源。最后，另一种可能的不一致性来源于对属性值的约束与定义域和值域限定相冲突。

7.2.7 定义实例

当然，定义本体很少是因为自身的缘故，而是为了用本体来组织集合实例。将这些集合实例填入本体是一个独立的步骤。典型情况下，实例的数量比本体中类的数量高很多个数量级。本体所含类的数量从几百个到上万个，而实例的数量则从数百个到几十万个甚至更多。

由于实例的数量如此巨大，所以将它们置入本体通常不是手工完成的。实例经常是从
208 原有数据源，如数据库中检索得到的。另一种常用技术是从文本库中自动抽取实例。

7.2.8 检查异常

使用 OWL 相对于使用 RDF Schema 的一个重要优点是可以检查本体自身或者置入本体的实例集的不一致。一些常见的不一致情况如上所述(见 7.2.6)。

7.3 复用现成本体

本体的构建应该尽可能从现成本体出发。现成本体有很多种。

7.3.1 专家知识汇编

一些本体是由很大的专家组经过多年细心编制而成的。医学领域中的一个例子是美国国家癌症学会制作的癌症本体㊀。文化领域中的例子有包含 125 000 个术语的艺术和建筑辞典(AAT)㊁以及有 220 000 个条目的艺术家名字叙词表(ULAN)㊂。另外一个例子是用 28 000个术语描述文化图像的图像分类辞典 Iconclass㊃。地理学领域的一个例子是涵盖 1 百万个条目的地理名称辞典(TGN)㊄。

7.3.2 整合词汇表

一些各自独立开发的词汇表已经合并成一个大的资源，例如由 100 个生物医学词汇和分类表整合而成的一体化医学语言系统 UMLS㊅。UMLS 包含 750 000 个概念以及它们之间 209
的 1 千万个链接。这样一种整合词汇表的语义质量相当差，但它在很多应用中还是非常有用的，至少可以作为一个起点。

7.3.3 高层本体

上面叙述的本体都是高度专业化的，同时也尝试定义非常一般化的应用本体(有时称作高层(upper-level)本体)。有两个典型的例子，一个是常识知识库 Cyc㊆，带有 6 000 个概念以及它们之间的 60 000 个断言，另一个是“标准高层本体”(SUO)㊇。

7.3.4 主题层次体系

其他“本体”严格说来是名不副实的，它们只是简单的术语集合，以一种特殊的层次关系松散地组织起来。这种层次体系通常不是一个严格的分类结构，而是同时混合了不同的特殊性关系如 *is-a*, *part-of*, *contained-in*。然而，这样的资源作为开发起点经常是十分有用

㊀ <http: //www. mindswap. org/2003/CancerOntology/ >。

㊁ <http: //www. getty. edu/research/tools/vocabulary/aat >。

㊂ <http: //www. getty. edu/research/conducting_research/vocabularies/ulan/ >。

㊃ <http: //www. iconclass. nl >。

㊄ <http: //www. getty. edu/research/conducting_research/vocabularies/tgn/ >。

㊅ <http: //umlsinfo. nlm. nih. gov >。

㊆ <http: //www. opencyc. org/ >。

㊇ <http: //suo. ieee. org/ >。

的。一个大型例子是 Open Directory[一]，包含超过 400 000 个按照层次组织起来的类，并用 RDF 格式表示。

7.3.5 语言学资源

一些资源起初建立时不是作为一个特定领域的抽象，而是作为语言学资源。它们作为本体开发的起点的作用已经显示出来。在这一类中典型的例子是 WordNet[二]，包含 9 万多个词义。

7.3.6 本体图书馆

目前正在尝试构建在线本体的在线图书馆。在本体工程组网[三]和 DAML 网站[四]上可以找
210 到有关例子。虽然对 XML Schema 的开发工作严格说来不是本体，仍可作为开发工作的有用起点。[五]

现成本体很难不加修改就可以复用。概念和属性通常都需要加以精练(使用 owl: subClassOf 和 owl: subPropertyOf)。还需要引入多个可选的名字以更好地适用于特定的领域(使用 owl: equivalentClass 和 owl: equivalentProperty)。RDF 和 OWL 允许对其他本体中定义的类加以精炼，应该充分利用这点。

关于本体导入和在不同本体间建立映射这类问题仍然悬而未决，而且这类问题被认为是语义网研究的最大难点之一(而且是非常紧迫的)。

7.4 使用半自动方法

将语义网蓝图变为现实有两个核心挑战。

第一，必须支持充实语义的再工程化任务以建立元数据 Web。语义网的成功极大地依赖于本体及相关元数据的增殖，这需要高速度低耗费地产生这种元数据。为此，可以用机器学习技术来支持本体的整合以实现语义可共用性。

第二，必须提供一种手段以维护和采用作为语义网基础的机器可处理数据，这就需要支持网络动态特性的机制。

尽管过去十年来已有成熟的本体工程工具，手工获取本体依然是一个耗时、昂贵、高

㈠ <http://dmoz.org>。

㈡ <http://www.cogsci.princeton.edu/_wn>，RDF 格式的在 <http://www.semanticweb.org/library/>。

㈢ <http://www.ontology.or.kr/ontology/onto_lib.asp>。

㈣ <http://www.daml.org>。

㈤ DTD/Schema 的例子可以在 http://XML.org，Rosetta Net 的例子可以在 <http://www.rosettanet.org> 看到。

度技巧化，并且有时很麻烦的任务，很容易导致知识获取瓶颈。

这些问题类似于过去二十年来知识工程师在知识获取方法学或定义知识库工作平台上处理过的问题。事实证明，知识获取和机器学习技术的整合对知识获取是有益的。 211

无论是在知识获取或抽取，还是在知识修订或维护方面，机器学习的研究已有很长的历史，并且提供了大量技术可用来解决这些挑战。机器学习技术可以支持下列任务：

- 从网上现有数据中抽取本体
- 从网上现有数据中抽取关系数据和元数据
- 通过分析概念外延来融合和映射本体
- 通过分析实例数据来维护本体
- 通过观察用户来改进语义网应用

机器学习提供了许多技术可用来支持下列任务：

- 聚类
- 本体的递增更新
- 帮助知识工程师
- 改进大规模的自然语言本体
- 单纯(领域)本体学习

Omalayenko 提出可用机器学习技术支持的三类本体，并且描述了这些领域的当前技术发展水平。

自然语言本体

自然语言本体(NLO)包含语言概念间的词汇关系，规模很大、不需要频繁更新。通常表达为系统的背景知识并且用来扩展用户查询。NLO 学习的现状看上去是十分乐观的，不仅存在稳定的通用 NLO，而且存在自动或半自动构建和丰富领域相关 NLO 的技术。 212

领域本体

领域本体刻画特定领域的知识，例如药理学或打印机知识。这些本体提供一个有限领域的详细概念描述。通常是手工建立的，但不同的学习技术可以协助知识工程师，特别是不熟练的知识工程师。领域本体的学习远没有 NLO 发展得好。领域本体的获取仍然靠人类知识工程师引导，自动化的学习技术在知识获取中起的作用非常小，只能在领域文档中找到统计意义上有效的依赖关系，并把这些建议给知识工程师。

本体实例

当本体保持不变时，本体实例可以自动产生并且频繁更新(如从黄页上得到一个公司概况会频繁更新)。本体实例的学习任务十分符合机器学习框架，也有几个机器学习算法的成功应用。但是，这些应用或者严格依赖于领域本体，或者加入大量与任何领域理论无关的标记。为给定领域本体从文本中抽取本体实例的通用技术还没有开发出来。

除了支持上述不同类型的本体，本体学习还有很多其他用途。下面列表中的前三个任务(也是来自 Omalayenko)在知识工程中与本体获取任务相关，后三个与本体维护任务相关。

- 知识工程师从零开始建立本体。在这个任务中，机器学习通过推荐领域内最重要的联系或检查、验证所构建的知识库来协助知识工程师。
- 从网络文档中抽取本体模式。在这个任务里，机器学习系统以数据和元知识(如同元本体)作为输入，产生可用的本体输出，这可能对知识工程师有所帮助。

213

- 抽取本体实例植入给定本体模式并抽取网络文档中的本体实例。这个任务类似于信息抽取和页面标注，而且可以将这些技术应用于这些领域。
- 本体整合和导航涉及对巨大的、可能经由机器学习获得的知识库的重构和导航。例如，一个具体任务可能是将经由机器学习获得的命题知识库转变为一阶知识库。
- 本体维护任务是更新本体的一些部分，这些部分设计为及时更新的(就像格式化标签必须紧跟页面设计的变化)。
- 本体丰富(或本体调整)包括对现有本体的次要关系的自动修改。这种修改不会改变主要概念和结构，但会使本体更准确。

机器学习提供很多技术、算法和工具。然而，本体表示的一个重要需求是，本体必须是符号化的、人类可读并可理解的。这就迫使人们只能使用符号化学习算法，而跳过神经网络和遗传算法等其他方法。一些潜在可用的算法如下：

- 命题规则学习算法(Propositional rule learning algorithms)学习关联规则或其他形式的属性-值规则。
- 贝叶斯学习(Bayesian learning)的主要代表是朴素贝叶斯分类器(Naive Bayes classifier)。它基于贝叶斯定理，在训练实例属性条件独立的假定下产生概率化属性-值规则。
- 一阶逻辑规则学习(First-order logic rules learning)归纳出称为一阶 Horn 子句的含变量的规则。
- 聚类算法(Clustering algorithms)将实例按照相似度或两个实例的属性值之间的距离测度聚集成组。

概括而言，尽管有很多潜在的机器学习技术可以用于语义网工程，目前还是一个远未得到很好理解的领域。没有现成的技术或工具是立竿见影的，虽然在不远的将来情况会发

214 生改变。

7.5 On-To-Knowledge 语义网体系结构

建立语义网不仅涉及使用本书介绍的新语言，还涉及一种非常不同的工程风格和一条

完全不同的应用集成途径。为了说明这一点，本节介绍如何将一些语义网相关的工具整合到单一的轻量级体系结构中，借助语义网标准在这些独立的工程性工具之间达到可共用性（见图 7-1）。

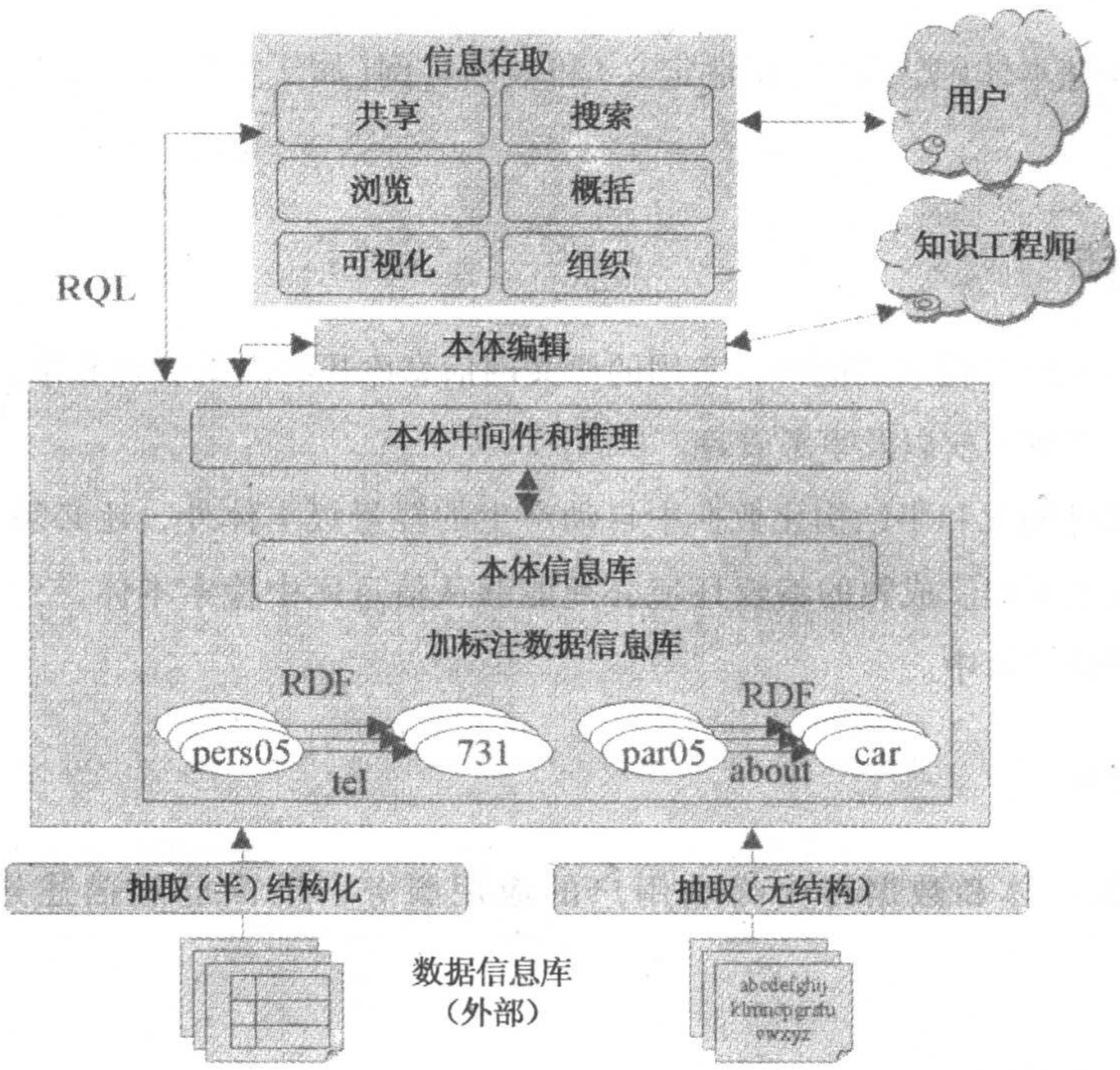


图 7-1　语义网知识管理体系结构 215

7.5.1　知识获取

在图 7-1 的底部可以找到一些使用表面分析技术（surface analysis techniques）从文档中获取内容的工具。这些文档可以是无结构的自然语言文档或者是结构化和半结构化文档（如 HTML 表和电子表格）。

对于无结构文档，这些工具通常结合统计技术和简单的自然语言技术，从文档中抽取关键概念。

对于结构化更强的文档，这些工具使用诸如装订程序、归纳和模式识别技术，从这些文档的弱结构中抽取内容。

7.5.2　知识存储

这些分析工具的输出是概念的集合，只有浅层概念层次，顶多有极少的交叉分类（cross-taxonomical）关系。用 RDF 和 RDF Schema 足够表达这些抽取出来的信息。

除了简单存储这些用抽取工具产生的知识外，信息库还必须提供检索这些知识的方法，最好是使用如第3章介绍的结构化查询语言。任何可推理的RDF Schema信息库也将支持RDF模型理论，包括基于定义域和值域定义的类成员演绎，以及subClassOf关系的传递闭包推导。

信息库将存储本体(类层次、属性定义)和本体实例(属于类的个体、使属性成立的个体对)。

7.5.3 知识维护

除了基本的存储和检索功能外，实用的语义网信息库还将提供管理和维护本体的功能：变更管理、存取和所有权以及事务管理。

除了自动从结构化和半结构化数据中自动产生的轻量级本体外，还必须支持有关知识
216 密集型本体的人工工程。成熟的编辑环境必须能够从信息库中检索本体，允许知识工程师对其操作并存回信息库中。

7.5.4 知识使用

信息库中的本体和数据将为终端用户的应用服务。前面已经描述过很多这样的应用。

7.5.5 技术可共用性

在On-To-Knowledge项目[⊖]中，图7-1的体系结构用组件间极轻量级的连接实现。所有组件通过RDF实现了语法可共用性。所有语义使用RDF Schema表达实现了语义可共用性。实体可共用性的实现是组件间通过简单的HTTP协议建立连接进行联系，除了一个组件(本体编辑器)外，其他所有都实现为远程服务。当从阿姆斯特丹操作On-To-Knowledge系统时，在挪威运行的本体抽取工具对得到的一个文档的伦敦网址进行分析；产生的RDF和RDF Schema上传到Amersfoort(荷兰)的信息库服务器运行。这些数据上传到本地安装的本体编辑器上，经过编辑之后传回到Amersfoort服务器。然后，这些数据提供给瑞典的一个基于本体的网站生成器(见第6章的EnerSearch用例学习)以及一个英国的搜索引擎，均在阿姆斯特丹电脑屏幕的浏览器上显示出结果。

总之，所有这些工具都在远程运行，独立地工程开发，只依靠HTTP和RDF达到高度
217 的可共用性。

⊖ <http://www.ontoknowledge.org>。

推荐读物

作为本章基础的关键性论文有：

- Ontology Development 101: A Guide to Creating Your First Ontology Natalya. F. Noy and Deborah L. McGuinness <http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html>.
- M. Uschold, and M. Gruninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, Volume 11 Number 2, (June 1996).
- B. Omelayenko. Learning of Ontologies for the Web: the Analysis of Existing Approaches, In: *Proceedings of the International Workshop on Web Dynamics, 8th International Conference on Database Theory (ICDTŠ01)*. 2001. <http://www.cs.vu.nl/ borys/papers/WebDyn01.pdf>

两本经常被引用的书：

- A. Maedche, *Ontology Learning for the Semantic Web*, Kluwer International Series in Engineering and Computer Science, Volume 665, 2002.
- J. Davies, D. Fensel, and F. van Harmelen. *Towards the Semantic Web: Ontology-Driven Knowledge Management*. New York: Wiley, 2003.

设计

本设计是一个中等规模的练习，需要 2 ~ 3 人大约 2 ~ 3 周的时间。所需要的软件都是免费软件。下面推荐一些成功使用过的软件，该领域的发展是十分活跃的，所以软件的可用性很可能在迅速地改变。如果没有提到某个软件，并不意味着不赞成使用它。

作业包括三个部分。

1. 在第一部分要建立一个本体，用来描述自己的应用领域，包含应用中需要的信息。可以使用本体中定义的术语来描述具体数据。在这一步中，需要应用本章开始部分概括的本体构建方法，并使用 OWL 作为本体表示语言(见第 4 章)。 218
2. 在第二部分要使用你构建的本体对数据构造不同的视图，查询本体和数据来抽取每一个视图需要的信息。在这部分里，要用到 RDF 存储和查询工具(见第 3 章)。
3. 在第三部分要使用 XSLT 技术对所抽取的数据建立不同的图形表示(见第 2 章)。

第一部分　建立一个本体

作为第一步，需要选定一个应用领域作为本设计的对象。最好选择自己具备足够知识或可以比较容易从专家处得到这些知识的领域。

在本设计说明中，将使用本书用到的一个领域，也就是大学全体教师领域，包括教师、课程和系，当然可以用你自己选择的其他领域来代替这个领域。

第二，用 OWL 表示建立一个本体来描述这个领域(例如全体教师)。本体不需要覆盖整个领域，但至少应包含几十个类。特别注意本体的质量(广度和深度)，并且尽可能充分地利用 OWL 的表达能力。这个阶段有很多可用的工具。根据作者的经验，OILed[㈠]比较好用，但其他编辑器也可以，比如 Protégé[㈡]和 OntoEdit[㈢]。如果有雄心，甚至可以使用本体抽取工具从文本进行本体开发(但本书没有这方面的介绍)，或者使用一些获取本体的工具，导入半结构化数据源，如 Excel 表单、用 tab 划界的文件等，这样的工具有 Excel2RDF 和 ConvertToRDF[㈣]。当然，也可以选择从该领域现有的本体做起[㈤]。

最好使用一个推理引擎来验证本体，检查不一致性。作者使用过与 OILed 紧耦合的
219 FaCT 推理引擎，而 OntoEdit 有自己的推理引擎。如果使用 Protégé，则需要找一些这个编辑器可用的插件，如本体的多重可视化界面，或用 Prolog 或 Jess 推理。

第三，用 RDF Schema 形式输出本体。当然，这会导致信息从丰富的 OWL 本体中丢失，但这是不可避免的，因为在后续步骤中的工具能力有限，并且这也正是现实语义网应用中真实的一幕。

最后，要通过具体的实例和它们的属性来丰富本体。与编辑工具的选择相关，可以用相同工具(OntoEdit)或不同的工具(OILed)。给定 RDF 实例的简单语法结构，你甚至可以手工完成，或写一些简单的脚本从可用在线资源中抽取实例信息(在本书的应用背景中，学生可以从全体教师的电话簿上获取一些信息)。可以使用 W3C 的验证服务[㈥]。这个服务不仅可以验证文件的语法正确性，还可以提供现有三元组的可视化列表。另外，在这个阶段，可以从一些半结构化资源中导入数据。这一步结束时，要产生以下结果：

- 完整的 OWL 本体
- 该本体的简化版，输出为 RDF Schema
- 本体的实例，以 RDF 表示
- 一个报告，描述该本体的范围和建模过程中的主要设计决定。

第二部分　用 RQL 查询建立服务概况

在这一步中，要使用查询工具抽取本体和数据中的相关部分。为此需要某种方法将本

㈠ <http://oiled.man.ac.uk>。

㈡ <http://protege.stanford.edu>。

㈢ <http://ontoprise.de>。

㈣ <http://www.mindswap.org>。

㈤ 例如，可以在 <http://www.daml.org/ontologies> 中找到。

㈥ <http://www.w3.org/RDF/Validator/>。

体存储在一个支持查询工具的信息库中。可以使用 Sesame RDF 存储和查询工具㈠，也可以选择 KAON 服务器㈡或 JENA㈢等。 220

第一步是将本体(RDF Schema 格式)和相关实例上传到信息库。这里包含一些安装过程。

接着，使用信息库自带的查询语言定义不同的用户服务概况，并使用查询来抽取每个服务概况的相关数据。

尽管这些程序支持不同的查询语言(Sesame 支持 RQL，Jena 支持 RDQL，KAON 服务器支持 KAON Query)，它们都提供足够的表达能力来定义丰富的服务概况。在建立自己的全体教师模型例子中，可以定义不同年份学生的服务概况，留学生的服务概况，学生和教师的服务概况，通过宽带或慢调制解调器访问的服务概况等。

定义服务概况查询的典型输出格式是 RDF/XML 或其他 XML 格式。

第三部分　显示基于服务概况的信息

在最后这部分，使用从第二部分查询得到的 XML 输出，产生不同服务概况的人类可读的表示。

显然需要用到的技术是 XML 样式表，特别是 XSLT(见第 2 章)。XSLT 有很多不同的编辑器，XLST 的处理器㈣也很多。

这部分的挑战是定义可浏览的、高度相关的数据表示。这些数据都是从第一和第二部分产生和选择出来的。

小结

在完成这个设计的所有部分之后，应该有效地实现了图 7-1 中所示的体系结构中的大部分。应用本书介绍的大部分语言(XML，XSLT，RDF，RDF Schema，OWL)，建立了一个真正的语义网应用，即，将现实世界的一部分建模为一个本体，用查询语言对这个本体定义特定的用户视图，并且使用 XML 技术定义这些用户视图的可浏览表示。 221

㈠ <http：//sesame. aidministrator. nl>。

㈡ <http：//kaon. semanticweb. org>。

㈢ <http：//www. hpl. hp. com/semweb>。

㈣ 例如 <http：//www. xslt. com>。

第 8 章　结论和展望

8.1　集成

本书介绍了语义网的关键技术。现在从第 1 章开始，做一个总体回顾是有益的。下面通过考察一个自动议价场景，我们来了解所有这些技术是怎样集成到一起的。

- 每一个参与议价的当事人都用一个软件代理表示。本书没有讨论一般意义下的代理，仅向读者提供相关的参考文献。
- 这些软件代理需要对一些特定术语的含义达成一致，为此需要公认某个共同的本体，比如用 OWL 写的本体。
- 事实、报价和决定可以用 RDF 陈述表达。当与一个本体相关联之后，这些陈述就会变得具有实际作用。
- 代理之间的信息交流使用基于 XML(或基于 RDF)的语言。

- 代理的协商策略用逻辑语言描述。
- 代理根据协商策略、事实、以往报价和还价等信息，经过推理，决定下一步行动。

8.2　一些技术问题

8.2.1　网络本体语言：少就是多

在语义网研究中，很多努力用于开发一个合适的网络本体语言，其结果是采纳 OWL 作为现行标准。一个核心问题是，本体语言究竟是否需要非常复杂？总可以设想一些场景是一阶逻辑无法表达的，但这些场景在实际应用中的重要性仍然有疑问。

有理由期望绝大部分本体知识本质上相当简单，使用表达能力较弱的语言就足够了。简单的本体语言意味着更高效的推理支持、更易于维护和更容易使用。最后一点对语义网能否成功是至关重要的。从这个角度来说，OWL Lite 沿着正确方向迈进了一步。

8.2.2　规则和本体

在第 4 章中曾经提到，现有(高级)网络本体语言基于描述逻辑。另一方面，我们又注

意到，规则在很多应用问题中是重要而简单的形式化表达方式。目前，将二者相结合的研究正在进行之中。

我们认为，一种整合描述逻辑和规则语言的全部功能的形式化可能矫枉过正了。姑且不论是否有必要使用这样表达能力丰富的语言，现有研究表明，这种整合存在着一些难以克服的复杂性和可计算性障碍。

一个明智的折中方案是在 RDFS 之上扩展规则语言，以此取代沿着描述逻辑的方向继续扩展。这种语言的表达能力并没有什么技术困难。不像看上去的那样有限，可以用规则语言定义描述逻辑以及 OWL 的很多特性。

8.3 未来预测

那么，语义网运动会成功吗？虽然很多人坚信这一点(并正在投资)，结果仍然是未知的。正如本书开头描述的那样，这个问题与其说是一个技术问题，还不如说是一个实行问题。关键在于能否快速、有力地证明这项技术的有用性以形成强大的发展势头。这与万维 224
网发展的早期情形是相似的。

*本体从何而来？*我们已经看到这个潜在瓶颈的一些解决方案。某些大型本体正成为事实上的标准(如 WordNet，NCIBI 癌症本体)；很多小型本体由一些机构手工编制(如 Rosetta-Net)，或者通过机器学习技术、自然语言分析或借鉴其他资源(如 database schemas)自动生成。

*语义标记从何而来？*显然，大量的语义标注工作不会用手工完成(不同于万维网早期用手工编码 HTML 页面的情形)。基于自然语言和借鉴其他资源技术(如数据库)的文档分析将为此发挥突出作用。

*工具从何而来？*这个潜在的瓶颈正在得到解决。对语义网应用生存周期中的每一方面，大量不同种类的工具(编辑、存储、查询和推理、可视化、版本工具)已经唾手可得。目前这些工具大部分被学术界使用，很快就会被商业界采用，尤其是被美国和欧盟正在启动的创新项目采用。

*怎样处理大量本体？*这称为本体映射问题[⊖]，可能是最难解决的。对此正在进行多种途径的探讨(基于协商代理、机器学习或语言学分析)，但尚未得到很好的解决。

可能最初的成功不会出现在开放、异质的万维网环境中，而是出现在大组织的内部网络中。在那些环境下，通过集中控制可以强制推行某些标准和技术，从而导致首次成功。相信大组织的知识管理可能是最有希望的起点。

⊖ 参见 6.8.1 节。——译者注

估计紧随其后的其他领域是所谓 e 科学，即科学家对语义网的运用，正如科学家对万维网的使用构成了万维网普及的一种重要“催化剂”一样。由于隐私、安全和信任方面的所有问题，电子商务可能是语义网的下一个应用。

总而言之，我们对语义网的前景表示乐观，同时也希望本书作为一本语义网教程能够
225 在“开发万维网全部潜力”的事业中发挥作用。

附录　OWL 抽象语法

第 4 章使用的基于 XML 的 OWL 语法相当繁琐，不易阅读。OWL 还有一个抽象语法[○]，其可读性强得多。

本附录用列出第 4 章中所有 OWL 代码的抽象语法表达。

4.2.2　头部

```
Ontology(

 Annotation(rdfs: comment "An example OWL ontology")
 Annotation(rdfs: label "University Ontology")
 Annotation(owl: imports http: //www.mydomain.org/persons)
)
```

4.2.3　类元素

```
Class(associateProfessor partial academicStaffMember)


Class(professor partial)
DisjointClasses(associateProfessor assistantProfessor)
DisjointClasses(professor associateProfessor)

Class(faculty complete academicStaffMember)
```

4.2.4　属性元素

```
DatatypeProperty(age range(xsd: nonNegativeInteger))


ObjectProperty(isTaughtBy
 domain(course)
 range(academicStaffMember))
SubPropertyOf(isTaughtBy involves)

ObjectProperty(teaches
```

○　定义在 <http: //www. w3. org/TR/ow/-semantics/ > 。

```
 inverseOf(isTaughtBy)
 domain(academicStaffMember)
 range(course))

ObjectProperty(lecturesIn)
EquivalentProperties(lecturesIn teaches)
```

4.2.5 属性约束

```
Class(firstYearCourse partial
 restriction(isTaughtBy allValuesFrom (Professor)))

Class(mathCourse partial
 restriction(isTaughtBy hasValue (949352)))

Class(academicStaffMember partial
 restriction(teaches someValuesFrom (undergraduateCourse)))

Class(course partial
 restriction(isTaughtBy minCardinality(1)))

Class(department partial
 restriction(hasMember minCardinality(10))
 restriction(hasMember maxCardinality(30)))
```

4.2.6 特殊属性

```
ObjectProperty(hasSameGradeAs Transitive Symmetric
 domain(student)
 range(student))
```

4.2.7 布尔组合

```
Class(course partial
 complementOf(staffMember))

Class(peopleAtUni complete
 unionOf(staffMember student))

Class(facultyInCS complete
 intersectionOf(faculty
                restriction(belongsTo
```

```
                                hasValue
                                (CSDepartment))))

Class(adminStaff complete
 intersectionOf(staffMember
                complementOf(unionOf(faculty
                                     techSupportStaff))))
```

4.2.8 枚举

```
EnumeratedClass(weekdays Monday
                         Tuesday
                         Wednesday
                         Thursday
                         Friday
                         Saturday
                         Sunday)
```

4.2.9 实例

```
Individual(949352
 type(academicStaffMember))

Individual(949352
 type(academicStaffMember)
 value(age "39"^^&xsd; integer))

ObjectProperty(isTaughtBy Functional)

Individual(CIT1111
 type(course)
 value(isTaughtBy 949352)
 value(isTaughtBy 949318))

Individual(949318
 type(lecturer))

DifferentIndividuals(949318 949352)
DifferentIndividuals(949352 949111 949318)
```

4.3.1 非洲野生生物本体

```
Ontology(
```

```
ObjectProperty(eaten-by inverseOf(eats))
ObjectProperty(eats domain(animal))
ObjectProperty(is-part-of Transitive)

Class(animal partial
 annotation(rdfs: comment "Animals form a class."))

Class(branch partial
 annotation(rdfs: comment "Branches are parts of trees.")
 restriction(is-part-of allValuesFrom (tree)))

Class(carnivore complete
 annotation(rdfs: comment
            "Carnivores are exactly those animals that eat animals.")
 intersectionOf(animal
                restriction(eats someValuesFrom (animal))))

Class(giraffe partial
 annotation(rdfs: comment
            "Giraffes are herbivores, and they eat only leaves.")
 herbivore
 restriction(eats allValuesFrom (leaf)))

Class(herbivore complete
 annotation(rdfs: comment
            "Herbivores are exactly those animals that
            eat only plants or parts of plants.")
 intersectionOf(
      animal
      restriction(eats
                  allValuesFrom
                  (unionOf(plant
                           restriction(is-part-of
                                       allValuesFrom
                                       (plant))))))))

Class(leaf partial
 annotation(rdfs: comment "Leaves are parts of branches.")
 restriction(is-part-of allValuesFrom (branch)))

Class(lion partial
 annotation(rdfs: comment
            "Lions are animals that eat only herbivores.")
 carnivore
```

```
 restriction(eats allValuesFrom (herbivore)))

Class(plant partial
 annotation(rdfs: comment
            "Plants form a class disjoint from animals."))

Class(tasty-plant partial
 annotation(rdfs: comment
            "Tasty plants are plants that are eaten both by herbivores and
carnivores.")
      plant
      restriction(eaten-by someValuesFrom (herbivore))
      restriction(eaten-by someValuesFrom (carnivore)))

     Class(tree partial
      annotation(rdfs: comment "Trees are a type of plant.")
      plant)

      AnnotationProperty(rdfs: comment)
      DisjointClasses(plant animal)
     )
```

4.3.2 打印机本体

```
Ontology(
 Annotation(owl: versionInfo
            "My example version 1.2, 17 October 2002")

DatatypeProperty(manufactured-by
 domain(product)
 range(xsd: string))

DatatypeProperty(price
 domain(product)
 range(xsd: nonNegativeInteger))

DatatypeProperty(printingResolution
 domain(printer)
 range(xsd: string))

DatatypeProperty(printingSpeed
 domain(printer)
 range(xsd: string))
```

```
DatatypeProperty(printingTechnology
 domain(printer)
 range(xsd: string))

Class(1100se partial
 annotation(rdfs: comment
            "1100se printers belong to the 1100 series and cost $450.")
 1100series
 restriction(price hasValue ("450"^^&xsd; integer)))

Class(1100series partial
 annotation(rdfs: comment
            "1100series printers are HP laser jet printers
            with 8ppm printing speed and 600dpi printing resolution.")
 hpLaserJetPrinter
 restriction(printingSpeed hasValue ("8ppm"^^&xsd; string))
 restriction(printingResolution
            hasValue ("600dpi"^^&xsd; string)))

Class(1100xi partial
 annotation(rdfs: comment
            "1100xi printers belong to the 1100 series and cost $350.")
 1100series
 restriction(price hasValue ("350"^^&xsd; integer)))

Class(hpLaserJetPrinter partial
 annotation(rdfs: comment
            "HP laser jet printers are HP products and laser jet printers.")
 laserJetPrinter
 hpPrinter)

Class(hpPrinter partial
 annotation(rdfs: comment
            "HP printers are HP products and printers.")
 hpProduct
 printer)

Class(hpProduct complete
 annotation(rdfs: comment
            "HP products are exactly those products
            that are manufactured by Hewlett Packard.")
 intersectionOf(
     product
     restriction(manufactured-by
```

```
                    hasValue ("Hewlett Packard"^^&xsd; string))))

Class(laserJetPrinter complete
 annotation(rdfs: comment
             "Laser jet printers are exactly those printers
             that use laser jet printing technology. ")
 intersectionOf(
          printer
          restriction(printingTechnology
                      hasValue ("laser jet"^^&xsd; string))))

Class(padid partial
 annotation(rdfs: comment "Printing and digital imaging devices form
                           a subclass of products. ")
       annotation(rdfs: label "Device")
       product)

Class(personalPrinter partial
 annotation(rdfs: comment
          "Printers for personal use form a subclass of printers. ")
 printer)

Class(printer partial
 annotation(rdfs: comment "Printers are printing and digital imaging devices. ")
 padid)

Class(product partial
 annotation(rdfs: comment "Products form a class. "))
)
```

索　引

索引中的页码为英文原书页码，书中页边标出原书页码。

A

#PCDATA(DTD 中唯一的元素的原子类型)，33

AAT(艺术与建筑分类辞典)，199，219

Aduna(公司名)，189，190

agent(代理，主体)，14

aim of the authors(作者的目的)，xix

Art and Architecture Thesaurus(艺术与建筑分类辞典)，199，209

artificial intelligence(人工智能)，16

attribute types(属性类型)，34，38

axiomatic semantics(公理语义)，94

B

B2B e-commerce(企业对企业电子商务)，6，200

B2B portals(企业间门户)，6

B2C e-commerce(企业对个人电子商务)，5

C

cancer ontology(癌症本体)，209

cardinality restrictions(基数约束)，112，121

CDATA(DTD 中的串类型)，34

class expressions(类表达式)，122

class hierarchy(类层次结构)，81

classes(类)，81

closed-world assumption(封闭世界假设)，145

complete proof system(完全的证明系统)，152

constant(常量)，155

container elements(容器元素)，75

CSS22(号标准层叠样式表)，50

Cyc(一个常识知识库)，210

D

DAMLDARPA(主体标识语言)，3

DAML + OIL(一种本体建模语言)，109

data integration(数据整合)，182

data type(数据类型)，39，67，72

data type extension(数据类型的扩展)，40

data type restriction(数据类型的限制)，41

defaults(默认值)，144

defeasible logic program(可废止逻辑程序)，163

defeasible rule(可废止规则)，163

definite logic program(确定逻辑程序)，152

domain(定义域)，81

domain and range restrictions(定义域和值域限制)，17，105，111，208，209

downward compatibility(向下可兼容性)，17

DTD(文档数据定义)，32

E

e-commerce(电子商务)，200

e-learning(在线学习)，192

element(元素)，24
element types(元素类型)，38
EMTREEElsevier(生命科学主题词表)，181
enumerations(枚举)，124
explicit metadata(显式的元数据)，8

F

fact(事实)，156
filter expression(过滤表达式)，47
first-order logic(一阶逻辑)，151
follows(推出)，159
formal semantics(形式语义)，110
FRODO RDFSViz(一种 RDF 可视化程序)，108
function symbol(函数符号)，155

G

goal(目标)，157

H

Horn logicHorn(逻辑)，152
HTML(超文本标识语言)，23

I

IconclassIronclass(分类辞典)，199，209
ID(DTD 中具有唯一名字的类型)，34
IDREF(DTD 中对 ID 的引用类型)，34
IDREFS(DTD 中对 ID 的多个引用类型)，34
inference system(推理系统)，99
inheritance(继承)，82
instances(实例)，81

K

knowledge management(知识管理)，3，185
knowledge representation(知识表示)，151

L

layer(层、分层)，16
layering of OWL 分层 OWL127
literals(文字)，64
logic(逻辑)，12，151
logic layer(逻辑层)，18

M

machine learning(机器学习)，211
machine-processable Web content(机器可解读的网页内容)，3
markup languages(标记语言)，24
MBASE(一个生物医学资料库)，181
MeSH(一个医学本体)，180
metaclasses(元类)，139
model(模型)，158
modules(模块)，144
monotonic logic program(单调的逻辑程序)，156
monotonic rule(单调规则)，156
multimedia(多媒体)，199

N

namespace(命名空间)，43，71
nonmonotonic rule(非单调规则)，153
nonmonotonic rule system(非单调规则系统)，161

O

OIL(一种本体建模语言)，109
On-To-Knowledge(一个欧盟语义网课题)，215，217
ontology(本体)，10
ontology development process(本体开发过程)，205

Open Directory(一个主题层次系统)，210
OWL(网络本体语言)，109
OWL DLOWL DL(子语言)，113，127
OWL FullOWL FULL(子语言)，113，127
OWL LiteOWL (轻型子语言)，114，128
OWL speciesOWL(的子语言)，113
owl：AllDifferent140
owl：allValuesFrom119，142
owl：backwardCompatibleWith126
owl：cardinality122，142
owl：Class117
owl：complementOf123，141
owl：DatatypeProperty118
owl：differentFrom140
owl：disjointWith117，139
owl：distinctMembers140
owl：equivalentClass117
owl：EquivalentClass139
owl：equivalentProperty119
owl：EquivalentProperty139
owl：FunctionalProperty122
owl：hasValue119
owl：imports116
owl：incompatibleWith127
owl：intersectionOf123，141
owl：InverseFunctionalProperty122
owl：inverseOf118，143
owl：maxCardinality122，142
owl：minCardinality122，142
owl：Nothing117
owl：ObjectProperty118
owl：oneOf124，141
owl：onProperty119，142
owl：Ontology116
owl：priorVersion126
owl：Restriction119，141
owl：sameAs140
owl：sameIndividualAs140
owl：someValuesFrom119，142
owl：SymmetricProperty122
owl：Thing117
owl：TransitiveProperty122
owl：unionOf123，141
owl：versionInfo126

P

path expression(路径表达式)，45
portal(门户)，187
predicate(谓词)，155
predicate logic(谓词逻辑)，151
priority(优先序)，161
procedural attachment(过程附加)，145
proof layer(证明层)，18
proof system(证明系统)，151
property(属性)，81
property chaining(属性链)，145
property hierarchy(属性层次结构)，83
range(值域)，81

R

RDF(资源描述框架)，61
RDF propertyRDF(属性)，64
RDF query languageRDF(查询语言)，100
RDF resourceRDF(资源)，63
RDF SchemaRDF (模式)，80
RDF Schema limitationsRDF Schema(的局限性)，111
RDF statementRDF(陈述)，64
rdf：_175
rdf：about71
rdf：Alt75
rdf：Bag75
rdf：Description66
rdf：first78
rdf：List78

rdf：nil78
rdf：object80
rdf：predicate80
rdf：Property85
rdf：resource72
rdf：rest78
rdf：Seq75
rdf：Statement85
rdf：subject80
rdf：type74，86
rdfs：Class85
rdfs：ConstraintProperty87
rdfs：ConstraintResource87
rdfs：domain86
rdfs：isDefinedBy 88
rdfs：label88
rdfs：Literal85
rdfs：range86
rdfs：Resource85
rdfs：seeAlso88
rdfs：subClassOf86
rdfs：subPropertyOf86
recommendations(推荐标准)，23
reification(具体化)，67，80
rfds：comment88
root(根)，31
root element(根元素)，31
Rosetta Net 200，211
RQL(RDF 查询语言)，100
rule body(规则的体)，156，163
rule head(规则的头)，156，163
rule markup(规则标记)，167，173
rule markup language(规则标记语言)，154
RuleML171
rules(规则)，145，152，224

S

search engines(搜索引擎)，1
select-from-where103
semantic interoperability(语义可共用性)，11
semantics(语义(学))，12
service grounding(服务基底)，195
service models(服务模型)，197
service profiles(服务概况)，195
shopbots(购物机器人)，5
SLD resolutionSLD(反驳)，161
sound proof system(可靠的证明系统)，152
Standard Upperlevel Ontology(高层本体标准)，210
standards(标准)，17，23
style sheet(样式表)，50
subclass(子类)，81
subproperty(子属性)，83
SUO(高层本体标准)，210
superclass(父类)，81

T

tags(标签)，24
TGN(地理名称辞典)，209
thesaurus(分类辞典)，180
Thesaurus of Geographic Names(地理名称辞典)，209
triple(三元组)，64
trust layer(信任层)，18
typed literals(类型化文字)，67

U

ULAN(艺术家名字叙词表)，209
UMLS(医学语言系统)，210

Unified Medical Language System(一体化医学语言系统)，209
Union List of Artist Names(艺术家名字联合列表)，209
unique-names assumption(唯一名假设)，125，145
upward partial understanding(向上部分可理解性)，17

V

variable(变量)，155
versioning(版本控制)，126
visualization(可视化)，189

W

Web Ontology Working Group(网络本体工作组)，109
Web servicesWeb(服务)，194
well-formed XML document 合式 XML(文档)，29
witness(证据)，159
WordNet(词网)，210
World Wide Web(万维网)，1
World Wide Web Consortium(万维网联盟)，3
wrappers 打包程序 5

X

Xlink58
XML(可扩展性标记语言)，23
XML attributesXML(属性)，28
XML declarationXML(声明)，27
XML documentXML(文档)，27
XML elementsXML(元素)，27
XML Schema(一种用于寻址部分 XML 文档的语言)，37
XPath45
Xpath101
XSL(可扩展样式表语言)，50
XSLT(可扩展样式表语言的转换语言)，50
XSLT template(可扩展样式表语言的模板)，51