Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™

Beginning XML, 5th Edition

XML入门经典（第5版）

Joe Fawcett
[美] Liam R. E. Quin 著
Danny Ayers
刘云鹏 王超 译

清华大学出版社

XML 入门经典

(第 5 版)

Joe Fawcett
[美] Liam R. E. Quin 著
Danny Ayers

刘云鹏 王超 译

清华大学出版社

北 京

Joe Fawcett, Liam R. E. Quin, Danny Ayers
Beginning XML, 5th Edition
EISBN：978-1-118-16213-2
Copyright © 2012 by John Wiley & Sons, Inc., Indianapolis, Indiana
All Rights Reserved. This translation published under license.

本书中文简体字版由 Wiley Publishing, Inc. 授权清华大学出版社出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

北京市版权局著作权合同登记号 图字：01-2013-2511

Copies of this book sold without a Wiley sticker on the cover are unauthorized and illegal.
本书封面贴有 Wiley 公司防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

XML 入门经典(第 5 版) / (美)福思特(Fawcett, J.)，(美)奎思(Quin, L. R. E.)，(美)艾尔斯(Ayers, D.) 著；刘云鹏，王超 译. —北京：清华大学出版社，2013

书名原文：Beginning XML, 5th Edition

ISBN 978-7-302-34271-7

Ⅰ. ①X… Ⅱ. ①福… ②奎… ③艾… ④刘… ⑤王… Ⅲ. ①可扩充语言—程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2013)第 249404 号

责任编辑：王　军 杨信明
装帧设计：孔祥峰
责任校对：成凤进
责任印制：
出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　印　　张：44.25　　字　　数：1188 千字
版　　次：2013 年 12 第 1 版　　印　　次：2013 年 12 月第 1 次印刷
印　　数：1～4000
定　　价：98.00 元

产品编号：

译 者 序

物竞天择，适者生存——在以计算机与互联网技术为代表的 IT 时代，各种各样的新技术如雨后春笋般涌现，然而真正能够历经磨练生存下来的却寥寥无几。毫无疑问，XML 便是其中的佼佼者。尽管在 XML 诞生之前，已经出现了 SGML。作为标记语言历史上的一个里程碑，SGML 拥有如同超文本标记语言(HTML)一样的历史地位。虽然 SGML 强大的灵活性使其可以适用于各种不同场景，但它还是不可避免地遭受失败。原因很简单，它太复杂了。正如 OSI 模型最终被 TCP/IP 打败一样，XML 的诞生标志着 SGML 的彻底失败。XML 来源于 SGML 的一个子集，保留了灵活性，去掉了复杂性。很快 XML 便获得了巨大的成功，XML 标准开始突飞猛进地发展，大批的软件开发商争先恐后地采纳这个标准，这一切令人叹为观止。如今 XML 在 IT 领域已经拥有无可动摇的地位，很难想象有一个重要的应用程序不使用 XML 来保存它的配置文件或数据文件。

作为 XML 技术的经典读物，本书几乎囊括与 XML 相关的一切。从 XML 诞生与发展到 XML 的定义与构造；从 XML 的验证(DTD、XML Schema、RELAX NG 和 Schematron)到 XML 的处理(XPATH 和 XSLT)；从与数据库的结合(XQuery)到在通信中的应用(RSS、SOAP、WSDL 和 AJAX)。这些错综复杂的知识点在作者笔下以一种循序渐进的方式串联起来，并以一种游刃有余的方式向读者娓娓道来。所谓庖丁解牛，大抵如此。

更为难能可贵的是，在本书中作者使用了平实朴素的语言辅以大量真实可信的实例来帮助读者消化理解一些艰难晦涩的知识点，使读者能够从一名初学者晋升为一名专家。在本书的翻译过程中，我无时无刻不被作者渊博的学识与贴切的教授方式所折服。作者似乎总能找到一些极富代表性的简单实例来阐述一些难点，通过实例的练习，读者能够更加深刻体会这些难点。所谓大道至简，大抵如此。

作为本书的第五版，其覆盖内容相较于第四版有了极大的改动。主要原因在于在过去的几年中 XML 不断地进化，围绕在 XML 周边的技术也是日新月异，以至于作者在写作第五版时竟然发现有四分之三的内容已经过时。本着严谨的治学态度，作者对第四版内容进行了大幅度的修编，补充了当今最前沿最流行的技术。这些新增的内容便构成了本书第五版的基础。所谓与时俱进，大抵如此。

时光荏苒，岁月如梭。回想自己初识 XML 是在大三的编程课上，迄今已多年。在此期间，本人先后积累了几段不同的学习经历，无论是在实验室使用 C++开发了人生第一个大型应用服务器，还是在互联网公司使用 php 与 C++编写了基于搜索的后台转码服务器，抑或是现在正在经历的使用 Java 开发分布式监控平台。可以说，每个阶段的学习与工作都离不开 XML 的身影，而本人也在这些经历中积累了丰富的 XML 开发经验。正是由于与 XML 的这种亲密接触才使得本人有信心有动力完成本书的翻译工作。翻译是一项考验人耐力与信心的工作，厚厚的一本书承载了本人与另一位译者的心血；翻译又是一项充满挑战的工作，正是

由于它的挑战性，才能够使我们有勇气第一次如此系统地学习 XML 技术。此时此刻，再一次向大家真诚地推荐这本书，只要静下心来细细品味，你就能真正理解这本经典所蕴含的无穷魅力。

本书的另一位译者王超多年来一直醉心于数据库方面的研究工作。在本书的翻译过程中，二人相互配合，建立了深厚的友谊。没有他，本人断然不能完成如此浩大的翻译工作，再次真诚地对其说一声感谢。同时感谢我们的母校北京邮电大学，感谢它多年的培养，带领我们进入了 IT 技术的殿堂。本书的翻译工作均在业余时间完成，因此牺牲了许多本应陪伴家人的时间，在此感谢他们的理解和支持。最后，感谢清华大学出版社李阳编辑对我的信任，将如此重要的工作交予本人，并在翻译过程中提出了许多宝贵的意见和建议。

本书全部章节由刘云鹏、王超翻译，参与翻译活动的还有孔祥亮、陈跃华、杜思明、熊晓磊、曹汉鸣、陶晓云、王通、方峻、李小凤、曹晓松、蒋晓冬、邱培强、洪妍、李亮辉、高娟妮、曹小震、陈笑。由于译者才疏学浅，译文中难免会有错漏与偏颇之处，望诸位海涵。如有关于本书或者 XML 技术的任何意见或想法，敬请将反馈发送邮件至 liuyunpeng275@gmail.com。我会认真阅读您发来的每一封邮件。

刘云鹏于北京

作 者 简 介

Joe Fawcett (http://joe.fawcett.name)已经断断续续从事软件开发四十年有余。他是最早荣获 Microsoft 颁发的最有价值专家荣誉的成员之一。Joe 是位于伦敦的卡普兰金融学院软件开发的负责人，卡普兰金融学院专注于为人们提供商业和会计培训，拥有英国最先进的在线会计学习系统之一。除本书的前一版外，这是他为 Wrox 编写的第三本书籍。

Liam R. E. Quin (http://www.holoweb.net/~liam)在万维网联盟(W3C)负责 XML 相关工作。在 20 世纪 80 年代初他就已经在从事标记语言和文本方面的工作，并自 XML 的诞生之日起便从事相关工作。他拥有计算机科学以及数字排版的背景知识，同时还出于对书籍和插图的热爱，维护着网站 www.fromoldbooks.org。他住在加拿大安大略省米尔福德峡湾附近一个古老的农场中。

Danny Ayers (http://dannyayers.com)是一名 Web 技术——主要是与链接数据相关的技术——的独立研究员与开发者。他在 XML 的早期便热衷于该技术。他的专业背景是电子音乐，尽管自从 Web 出现以来该兴趣已经退居次要位置。生活中他是一个木雕业余爱好者，来自英国，现在他在托斯卡纳农村与两只狗和两只猫生活在一起。

技术编辑简介

Karen Tegtmeyer 是一名拥有超过 10 年经验的独立顾问与软件开发者。她从事过多种工作，包括设计、开发、培训以及架构。同时她还是得梅因社区学院的一名兼职计算机科学讲师。

致　　谢

我诚挚地感谢编辑Victoria Swider以及组稿编辑Carol Long的协助，当本项目看上去停滞不前时是他推进了整个项目。我想感谢前一版本的作者，特别是Jeff Rafter和David Hunter，他们使得我们在必要时能够借鉴他们的工作成果。还要感谢妻子Gillian以及我的孩子Persephone和Xavier，他们在过去一年中忍受着我的缺席与蹩脚的幽默；我保证会补偿你们的。

—Joe Fawcett

感谢我的合著者，感谢我的宠物们忍受冗长而又飘忽不定的时间，当然还要感谢 Alexander Chalmers 创建 1810 年的传记词典。

—Liam R. E. Quin

非常感谢 Victoria、Carol 以及整个团队所做的一切。同样感谢 Joe 提供本项目背后的动力，感谢 Liam 使它能持续进展。

—Danny Ayers

前　言

作为一本书的第 5 版，这足以证明其在专业开发人员和学术机构中的流行程度。本书致力于传授这样一种知识，它起初看起来只是另一种昙花一现的技术，但是相反却不断发展成熟以致成为一种理所当然的选择。自本书前一版本算起已将近六年光阴——在 IT 中这是一个名副其实的生命周期。在复审第 4 版中哪些应该保留，哪些应该更新，哪些新材料需要补充的过程中，作者发现大约四分之三的材料已经完全过时。相较于于五年前 XML 有了更多用途，同时在它背后也有了更多依赖 XML 的技术。现在我们不再需要手工配置深奥的配置文件来让一个 Web 服务启动并运行。此外在某些领域人们也发现 XML 并不是总是最合适的。这样或那样的情况，伴随着一次对内容的完整翻修，便形成了这一新版本的基础。

因此，什么是 XML？XML 代表可扩展标记语言(eXtensible Markup Language)，它是一门用于以一种有意义的方式描述数据的语言。实质上无论哪里都需要存储数据，尤其是在数据可能需要被多个应用处理时，XML 便是一个很好的起点。它已经在那些重视交互性的场景上取得了良好的声誉从而成为候选技术，这些场景既可以是不同企业间的两个应用，也可以是一个公司内的两个应用。目前存在数百种标准化 XML 格式，称为架构(schema)，它们已经由各个企业达成共识来表示不同类型的数据，从病历到金融交易再到用于表示旅程的 GPS 坐标。

本书读者对象

本书面向广泛的读者群体。大多数开发者听说过 XML 但是却对其存有一丝恐惧。XML 如今已经习惯在幕后使用，只有在其不能正常工作或者开发者希望做一些与众不同的事情时，用户才意识到他们必须关注内部原理。对于这些人我们有一句忠告：不再恐惧。它同样适用于那些在其他领域拥有丰富经验但是却从没有正式接触过 XML 的开发者。最后，当你需要第一次尝试一些新东西时，它可以作为你的参考。本书中几乎所有技术都有一个相关的“试一试”练习，它将引导你运行一个简单的示例并对该示例的工作过程进行解释。

本书大体上并不需要你有任何标记语言的知识。这些知识都将在前面几章覆盖。尽管预计大部分读者都有一些 Web 编程的知识与经验，但是我们已经试着扩展示例使得这些知识包括使用 Microsoft 堆栈、Java 或者其他诸如 PHP 或 Python 的开源框架。

不用在意书名中的“入门”一词，那只是 Wrox 用于描述本书风格的词汇而不是指你的经验水平。本文中的许多概念，特别是最后几章，均来自于现实生活，已经远远超出了入门的范畴。

本书的内容

本书旨在教授你需要知道的XML的所有知识——它是什么，它如何工作，它的相关技术，以及如何使用它来进行工作(从简单的数据转换到提供多通道内容)。本书着手回答下面这些基础问题：

- 什么是 XML
- 如何使用 XML
- 它是如何工作的
- 它能用来做什么

XML 的基本概念自出现以来一直没有改变过，但是其周边的技术却发生了显著的变化。本书对每种技术以及它是如何崛起的都做了一个基本的回顾，但是大部分例子使用的都是最新版本的技术。这些例子同样来自多个平台，其中 Java 和.NET 占据大多数。XML 产品同样有所涉及，一度曾经有许多免费和商业的可扩展样式表语言转换(XSLT)处理器；例如 XSLT 用于操纵 XML，将其由一种结构转换为另一种，这将在第 8 章介绍，但是自从版本 2 出现之后，由于需要开发和维护软件的工作增多，这些处理器数量已经大幅减少。

本书的组织结构

我们已经尝试将本书中覆盖到的主题以一种尽可能符合逻辑的方式来编排使你能够从一名初学者蜕变成一名专家。每一章节都覆盖一个不同的专业领域。除非你对基础知识相当了解，我们建议你阅读第 1 部分的介绍性章节，尽管对于精明的读者来说略读就已经足够了。其他章节可以按照顺序阅读，或者如果你对其中某一领域尤其感兴趣，可以直接跳到那一章阅读。例如，当你的老板突然告知你下一个版本必须提供一个 XQuery 插件，你就可以直接跳到第 9 章。本书一个简要的概述如下：

- 首先你将学习 XML 的精确定义以及为什么人们觉得它有用。
- 之后带你学习如何创建 XML 以及应该遵循哪些规则。
- 一旦你掌握了它，你将继续学习一个合法的XML文档是什么样子的，以及你如何确定你的文档是合法的。
- 之后将了解如何能够操纵 XML 文档来提取数据并将其转换成其他格式。
- 接下来将会在数据库中存储 XML——介绍这么做的优点与缺点，以及如何对数据库中的数据进行查询。
- 之后将学习提取数据的其他方法，特别是哪些适用于处理大文档的方法。
- 然后我们介绍了 XML 的一些用途，如何以 XML 格式发布数据，如何创建与使用基于 XML 的 Web 服务。我们解释了 AJAX 的由来以及它的工作原理，同时还有一些 XML 的替代品以及何时可以考虑使用它们。
- 之后是关于如何在网页中以及图片显示中使用 XML 的一些章节。
- 最后是一个案例分析。它将许多基于 XML 的技术结合应用于一个现实世界中的例子。

我们已经试着将本书按照一种符合逻辑的方式进行组织，首先向你介绍基础知识，之后带领你学习与XML相关的不同技术。这些技术被分成六个部分，它覆盖了大部分你能遇到的XML的主题，从原始数据验证到处理、存储以及展现。

第Ⅰ部分：XML 介绍

大部分读者应该从这里开始。该部分的各章覆盖了XML的目标以及构建规则。在阅读了这一部分之后你应该理解基本概念与术语。如果你已经熟悉XML，你可以快速浏览这些章节。

第 1 章：什么是 XML——第 1 章介绍了 XML 的历史，为什么需要它，以及创建 XML 文档的基本规则。

第 2 章：良构的 XML——该章进一步详细描述了一个文档如果称为 XML，则它应该包含什么，不应该包含什么。它还包括了用于描述一个 XML 文档中不同组成部分的现代命名系统。

第 3 章：XML 名称空间——每个人的最爱，可怕的名称空间主题以一种简单易懂的方式进行解释。在阅读完该章之后，你将成为专家，而其他人则还在抓耳挠腮。

第Ⅱ部分：验证

本部分涵盖了多种技术来协助你验证创建或接收的 XML 的格式是否正确。

第 4 章：文档类型定义——DTD 是最初的 XML 验证机制。该章展示了如何使用它们来既对文档进行约束又能提供额外内容。

第 5 章：XML Schema——XML Schema 是一种更现代的描述一个 XML 文档格式的方式。该章描述了它们的工作方式并讨论了 DTD 的优缺点。

第 6 章：RELAX NG和Schematron——有时无论是DTD还是Schema都无法提供你需要的功能。该章介绍了另外两种方法，可用来检查XML是否合法，此外还包含混合了多种验证技术的例子。

第Ⅲ部分：处理

该部分包含了从一个 XML 文档提取数据和将一种 XML 格式转换为另一种格式的技术。它包含了 XPath 基础，它是许多 XML 技术的基石。

第 7 章：从 XML 中提取数据——该章介绍了文档对象模型(DOM)，一种最早设计出来用于从 XML 中提取数据的方式。之后继续描述了 XPath，XML 技术的基石之一，它能精确定位到感兴趣的一个或多个项。

第 8 章：XSLT——XSLT 是一种将 XML 从一种格式转换成另一种格式的方法，如果你是从一个外部源接收文档并且需要自己的系统能够读取该文档，则它是一种基本方法。它覆盖了版本 1 的基础，当前版本的更高级特性，并展示了下一个版本中计划中的一些特性。

第Ⅳ部分：数据库

在过去的许多年中，数据库中存储的数据与 XML 中的数据一直存在差距。本部分内容将两者结合在一起，并向你展示了如何能充分利用两种方式。

第 9 章：XQuery——XQuery 是一种被设计成查询已存在文档并创建新 XML 文档的机制。它尤其适用于存储在数据库中的 XML 数据，该章将展示它的工作原理。

第 10 章：XML 与数据库——许多数据库系统现在都有特意为 XML 设计的功能。该章检验了三种上述产品，展示了如何同时查询与更新已有数据，并在有需要时创建新的 XML。

第Ⅴ部分：程序设计

该部分专注于两种处理 XML 的编程技术。第 11 章覆盖了处理大文档的技术，第 12 章展示了 Microsoft 最新的通用数据访问策略，LINQ 是如何与 XML 结合使用的。

第 11 章：事件驱动程序设计——该章介绍了两种特别适用于处理大文件的 XML 处理方法。一个是基于开源 API 的，示例代码使用 Java 实现。第二个是 Microsoft 的.NET 框架的核心部分，它的示例代码使用 C#实现。

第 12 章：LINQ to XML——该章介绍了 Microsoft 最新处理 XML 的方法，从创建到查询再到转换。它包含了许多使用 C#和 VB.NET 的示例，目前相较于.NET 版本它们具有更多功能。

第Ⅵ部分：通信

本部分共有 5 个章节，它们专注于使用 XML 作为一种通信方式。它覆盖了将数据以一种许多不同系统都能够利用的形式来展示，之后说明了 Web 服务如何能使数据被大量不同的客户端所使用。最后讨论了如何以一种标准的方式描述复杂数据使得它能够被访问。

第 13 章：RSS、Aton 和内容聚合——本章覆盖了将内容(比如新闻 feed)以一种平台独立的方式展示的主要方法。它还介绍了相同的 XML 格式如何能用于展示诸如客户列表或销售业绩之类的结构化数据。

第 14 章：Web 服务——Web 服务是过去十年中最大的软件成就之一。该章分析了它们如何工作，XML 如何与其结合，如果要排除遇到的问题，这些是必须了解的基础知识。

第 15 章：SOAP 和 WSDL——该章深入挖掘 Web 服务并描述了内在使用的两个主要系统：SOAP(它决定了服务如何被调用)，以及 Web 服务描述语言(WSDL)，WSDL 用于描述 Web 服务应该提供什么内容。

第 16 章：AJAX——该部分的最后一章介绍了 AJAX 以及它如何能够让你的网站在保持响应并使用更少的带宽的情况下提供即时信息。很显然，它使用了 XML，但是该章还介绍了你希望抛弃 XML 转而使用一种替代技术的情况。

第Ⅶ部分：显示

本部分展示了两种使用 XML 以一种用户友好的方式显示信息同时其格式又能够被计算机识别的方法。

第 17 章：XHTML 和 HTML5——该章介绍了如何以及何时使用 XHTML 以及为什么它相较于传统的 HTML 更受欢迎。之后进一步展示了 HTML 5 的新特性以及它如何消除其中的一些障碍。

第 18 章：可伸缩矢量图形(SVG)——该章介绍了如何将图片存储在 XML 格式中以及这种做法的优点。之后展示了这种格式如何与其他的诸如 HTML 的格式相结合，以及这样做的原因。

第Ⅷ部分：案例分析

该部分包含了一个案例分析，它结合了 XML 的许多用途，并以一个真实世界的例子来展示了它们之间是如何交互的。

第 19 章：案例分析：XML 在出版业中的应用——该案例分析展示了一个虚构的出版社如何从一个基于专有的出版软件转移到一个基于 XML 的工作流，以及在商业上带来了哪些好处。

附录

三个附录包含了参考资料以及每章最后练习的答案。

附录 A：习题参考答案——该附录包含全书所有出现过的每章最后练习的答案。

附录 B：XPath 函数——该附录包含大部分 XPath 函数的信息、它们的签名、返回值以及如何使用它们的例子。

附录 C：XML Schema 数据类型——该附录包含由 XML Schema 定义的众多内置数据类型的信息。它展示了这些数据类型之间的联系以及它们如何被不同的面(facet)所限制。

使用本书的预备知识

运行本书中的示例并不需要购买任何软件，所有例子都可以使用免费可用的软件来编写并运行。你将需要一个装有标准浏览器的计算机——Internet Explorer、Firefox、Chrome 或 Safari，只要它们是较新的版本都可以工作。如果你想亲自创建实例而不是从 Wrox 网站下载，那么你将需要一个基本的文本编辑器，即使是记事本也照样可以。对于一部分代码你也需要运行一个 Web 服务器，Windows 下的 IIS 服务器的免费版本或者其他系统的众多开源实现服务器(比如 Apache)都可以满足需求。对于一些代码示例，你将需要使用 Visual Studio。你既可以使用 Microsoft 的商业版本，也可以使用供下载的免费版本。

如果想使用免费版本 Visual Studio Express 2010，可以前往 www.microsoft.com/visualstudio/en-us/products/2010-editions/express 下载。每个版本的 Visual Studio 都专注于某个特定领域，比如 C#或 Web 开发，因此为了验证所有的示例，你需要下载 C#版本，VB.NET 版本以及 Web 版本。你还应该安装 Visual Studio 2010 的 1.0 服务包，它可以在 www.microsoft.com/download/en/details.aspx?id=23691 下载到。一旦一切准备就绪你就能够打开样例解决方案，或者如果失败的话，可以通过选择 File | Open | Project/Solution...的方式打开解决方案中的一个样例项目，并浏览解决方案文件或者你希望运行的特定项目。在本书即将出版之时，Microsoft 正在准备推出新版本 Visual Studio 2011。尽管截图可能会略有不同，但是本书中的例子都应该可以在新版本中运行。

约定

为了帮助你从本书中获得尽可能多的知识以及清楚地知道它的内容，在本书中我们使用了大量的约定。

试一试

“试一试”是一个你应该遵循本书中的文字说明而进行实践的练习。

(1) 它们通常由许多步骤组成。
(2) 每一步骤都有一个编号。
(3) 使用你自己的数据库副本来按照步骤进行练习。

示例说明

在每个“试一试”之后，将会详细讲解你所输入的代码。

警告：像这样带有警告图标的方框非常重要，它是与上下文内容紧密相关的必须牢记的信息。

注意：铅笔图标表示当前讨论的注释、提示、暗示、技巧以及旁白。

我们使用如下两种方式表示代码：

- 对于大部分示例我们使用不带突出效果的 monofont 类型。
- 对于当前上下文中尤其重要的代码我们使用加粗字体来突出显示。

源代码

在读者学习本书中的示例时，可以手动输入所有的代码，也可以使用本书附带的源代码文件。本书使用的所有源代码都可以从本书合作站点 http://www.wrox.com 或 www.tupwk.com.cn/downpage 上下载。登录到站点 http://www.wrox.com，使用 Search 工具或书名列表就可以找到本书。接着单击本书细目页面上的 Download Code 链接，就可以获得所有的源代码。

注释：由于许多图书的标题都很类似，因此按 ISBN 搜索是最简单的，本书英文版的 ISBN 是 978-1-118-16213-2。

在下载了代码后，只需要用自己喜欢的解压缩软件对它进行解压缩即可。另外，也可以进入 http://www.wrox.com/dynamic/books/download.aspx 上的 Wrox 代码下载主页，查看本书和其他 Wrox 图书的所有代码。

勘误表

尽管我们已经尽了各种努力来保证文章或代码中不出现错误，但是错误总是难免的，如果您在本书中找到了错误，例如拼写错误或代码错误，请告诉我们，我们将非常感激。通过勘误表，可以让其他读者避免受挫。当然，这还有助于提供更高质量的信息。

请给 wkservice@vip.163.com 发电子邮件，我们就会检查您的信息，如果是正确的，我们将在本书的后续版本中采用。

要在网站上找到本书的勘误表，可以登录 http://www.wrox.com，通过 Search 工具或书名列表查找本书，然后在本书的细目页面上，单击 Book Errata 链接。在这个页面上可以查看到 Wrox 编辑已提交和粘贴的所有勘误项。完整的图书列表还包括每本书的勘误表，网址是 www.wrox.com/misc-pages/booklist.shtml。

P2P.WROX.COM

要与作者和同行讨论，请加入 p2p.wrox.com 上的 P2P 论坛。这个论坛是一个基于 Web 的系统，便于您张贴与 Wrox 图书相关的消息和相关技术，与其他读者和技术用户交流心得。该论坛提供了订阅功能，当论坛上有新的消息时，它可以给您传送感兴趣的话题。Wrox 作者、编辑和其他业界专家和读者都会到这个论坛上来探讨问题。

在 http://p2p.wrox.com 上，有许多不同的论坛，它们不仅有助于阅读本书，还有助于开发自己的应用程序。要加入论坛，可以遵循下面的步骤：

(1) 进入 p2p.wrox.com，单击 Register 链接。

(2) 阅读使用协议，并单击 Agree。

(3) 填写加入该论坛所需要的信息和自己希望提供的其他信息，单击 Submit。

(4) 您会收到一封电子邮件，其中的信息描述了如何验证账户，完成加入过程。

不加入 P2P 也可以阅读论坛上的消息，但要张贴自己的消息，就必须加入该论坛。

加入论坛后，就可以张贴新消息，响应其他用户张贴的消息。可以随时在 Web 上阅读消息。如果要让该网站给自己发送特定论坛中的消息，可以单击论坛列表中该论坛名旁边的 Subscribe to this Forum 图标。

关于使用 Wrox P2P 的更多信息，可阅读 P2P FAQ，了解论坛软件的工作情况以及 P2P 和 Wrox 图书的许多常见问题。要阅读 FAQ，可以在任意 P2P 页面上单击 FAQ 链接。

目　录

第Ⅲ部分　处理

第Ⅳ部分　数据库

第 I 部分

XML介绍

- 第 1 章　什么是 XML
- 第 2 章　良构的 XML
- 第 3 章　XML 名称空间

第 1 章 什么是 XML

本章内容：

- XML 出现之前的历史
- XML 的诞生
- XML 文档的基本格式
- XML 的应用领域
- XML 相关技术的简介

XML 是可扩展标记语言(Extensible Markup Language)的缩写(大概是原作者认为 XML 比 EML 听起来更加令人兴奋)。在软件和 IT 领域，XML 的发展和流行遵循一种常规的模式。XML 出现于十多年前，起初使用它的应用程序廖廖无几，不过后来它就引起了公众广泛关注，它的使用开始遍及数据交换领域。随后，XML 的处理和管理工具不断完善，许多人在不知道 XML 存在的情况下就已经在使用它。近来人们对它的一些缺陷与缺点有了反应，不过也随之产生了许多改进以及替代措施。如今 XML 在 IT 领域已经拥有无可动摇的地位，很难想象有一个重要的应用程序不使用 XML 来保存它的配置文件或数据文件。出于以上原因，有必要让现代程序开发人员透彻地了解 XML 的原理，XML 能胜任的场景以及如何利用 XML 的最大优势。而这本书正好可以提供这些内容。

注意：虽然本章提供了一些简短的 XML 例子，但是并不要求你现在就能理解它们的含义。我们的想法只是向你介绍语言背后的重要概念，这样，在你看完全书后，不仅能够掌握 XML 的用法，而且知道它的处理过程。

1.1 初识 XML：数据描述和标记语言

XML 主要有两种应用：第一种是用 XML 来表述底层数据，如配置文件。第二种是利用 XML

为文档添加元数据。例如，在一份报告中用*斜体字*或**粗体字**强调一个特殊的句子。

上面提到的第一种 XML 的用法是用它来替代传统数据表示方式(如 Windows 系统上 INI 文件或 Java 的 Property 文件中的名称/值的表示方式)。第二种 XML 的应用类似于 HTML 文件的工作方式。文档内容包含在<body>元素代表的整体容器内，以<i>或<b>标签使这部分内容以斜体或粗体显示以表示强调。针对这两类应用场景，多年来涌现了多种技术。随着互联网的普及和分布式应用的广泛使用，尤其在软件的不同组件由不同的团队负责开发时，这些不同技术间的差异所造成的问题日益凸显。这个问题体现在数据的通用性上。一个分布式系统很可能包含这样两个组件，其中一个使用 Windows 系统的 INI 文件输出数据，而另一个组件需要将这些数据转换为 Java Properties 格式。这给两个组件的开发人员都增加了不必要的负担，占用了原本应该用于开发新业务功能的资源。

XML则可以用于解决这类问题，它的目的是使不同组件间的数据交互更容易，降低转换输入输出数据格式时的资源需求，使开发人员能专注于其他更重要的编码工作，如编写代码实现业务逻辑。XML也常用于解决文件可读性问题，使文件既方便被人们阅读又能被软件解析。本书也会剖析基于文档的XML和基于数据的XML之间的区别，不过在这之前，让我们先稍微深入地了解一下，在XML出现前计算机是用什么方式存储和交互数据的。

本节内容着眼于数据的一般表述方式，并不会花太多时间解释底层数据(如内存地址等)的详细信息。这里将文件格式分为两大类：二进制文件和文本文件。

1.1.1　二进制文件

简单来说，二进制文件就是一个比特流(0 和 1 的序列)。只有创建二进制文件的应用程序才会理解整个比特流的含义。这就是为什么二进制文件只能由某些特定的计算机程序读取和生成的原因，这些程序就是专门为理解这些二进制文件而设计的。例如，使用 Microsoft Word 2003 之前的版本保存了一份文档，这个文件(以 doc 为扩展名)就是以二进制格式创建的。如果你试图使用文本编辑器(如 Notepad)打开这个文件，会发现无法看到之前在 Word 文档内保存的图片。最好的情况下你能偶尔看到一些被乱码环绕的文本行，但肯定不是原来保存的，有着使用粗体或斜体样式的散文。文档中不仅保存了简单的文本，还保存了我们称之为元数据(metadata)的内容，或者称之为关于信息的信息。混合数据和元数据在二进制文件中非常的常见且简单。元数据能指定各种文件样式，例如以粗体显示文字，在表格中显示文字等。你需要创建这个二进制文件的应用程序把它翻译成你能读懂的文字。离开了这样一个能完全理解二进制格式数据的转换器，即使用 WordPerfect 这样的具有类似功能的软件，也无法打开一个用 Word 创建的文档。二进制格式的主要优点在于内容简洁并且可以被压缩保存到一个相对较小的空间内。这样可以保存更多的文件(例如保存在硬盘上)，更重要的一点是通过网络传输它们时所占用的带宽更小。

1.1.2　文本文件

文本文件和二进制文件相比，最大的不同之处是文本文件是人机均可读的。不需要一个专门的应用程序去解读它，文本文件中的每一组比特流均表示一个已知字符集中的一个字符。这意味着许多不同应用程序都能解读文本文件。在一台标准 Windows 平台的计算机上，可使用 Notepad、WordPad 或者其他任何基于命令行的工具，如 Edit 来解读文本文件。在非 Window 平台的计算机上也有各种类似的系统内的嵌程序可以解读文本文件，如 Emacs 和 Vim。

注意：使用底层数据流表示字符的方式被称为文件的编码方式。一个文件具体使用何种编码方式可从文件的前几个字节中得知。应用程序会在打开这个文件时检查这几个字节，并通过它们来确定显示和操作数据的方式。如果文件的前几个字节为空，则采用应用程序预先设定好的默认字符集来解读文件。XML 使用另一种方法指定文件的编码方式，这将在后面的章节中学习。

文本文件的优点不仅是人机均可读，而且它们也比二进制文件更容易被解析。而文本文件的主要缺点是文件较大。为了使文本文件包含元数据(例如，一段长文本被标记为“重要”)，通常会使用一些其他符号将这些文本标记出来。在 HTML 文件中很容易看到这样的例子，它使用一对尖括号内的信息来表示文本的处理方式。例如我希望在 HTML 页面中标识一个重要语句，那么相应的 HTML 代码为：

```
<strong>returns must include the item order number</strong>
```

另外，文本文件的另一个缺点是缺乏元数据支持。如果将一个包含有几种字体风格的 Word 文档保存为一个文本文件，那我们只能得到一个毫无格式的副本，所有的元数据都丢失了。人们希望找到一种两全其美的方法——一种既方便人类阅读又能被各种应用程序解析，还可以保存元数据的文件格式。于是标记语言应运而生了。

1.1.3　标记语言简史

文本文件的优点使人们在二进制文件和文本文件中更倾向于使用文本文件，但文本文件的缺点也使人们希望能有一套添加元数据的标准规范让大家遵循。大多数人认同使用一些额外的文本来记录元数据信息的标记语言，并把它作为技术发展的方向，但仍然有许多事情有待决定。其中最主要的两个问题是：

- 如何把元数据和原文区分开来？
- 文件中允许添加什么样的元数据？

例如，一些文档需要把内容标记为粗体或斜体，而另一些文档更关心文档的原作者是谁，它是在什么时侯创建的，谁曾经修改过它。为了解决这些问题，标准通用标记语言(Standard Generalized Markup Language，SGML)出现了。SGML 是标记语言历史上的一个里程碑，就像超文本标记语言的历史地位(HyperText Markup Language，HTML)一样。和使用标准限制标记语言正相反，SGML 允许人们使用标准的语法语义创建他们自己的标记语言。任何基于 SGML 语言的应用程序都可以使用这种语言编写的文件并进行相应的处理。正如之前提到的，最常见的例子就是 HTML。HTML 使用尖括号(<and>)来区分原文和元数据，HTML 还定义了一系列可以用在尖括号内的代码，如 em 表示文本，tr 表示表格，td 表示表格内的数据。

1.2　XML 的诞生

虽然 SGML 是一种经过深思熟虑的可应用于各种不同场景的标记语言，但它还是遭受了重大失败，因为它太复杂了。它所有的灵活性都是有代价的，而且只有很少的应用程序可以读取以 SGML

方式定义的标记语言并用它来正确地处理文档。SGML 的理念是正确的，但需要被简化。以此为目标，在 20 世纪 90 年代中期一个小开发团队和大量的对此感兴趣的组织开始致力于研究 SGML 的一个子集，这个子集被称为可扩展标记语言(eXtensible Markup Language，XML)。XML 的第一个草案发布于 1996 年，两年后的 1998 年 2 月 10 日，W3C 发布了它的修订版推荐标准。

注意：万维网联盟（The World Wide Web Consortium，W3C）是主要的国际万维网标准制定组织，它拥有大量的工作组并致力于为各种在 Web 上使用的技术制定推荐标准。一个推荐标准需要经过多个阶段的修改，如工作草案，候选推荐标准最后才能成为推荐标准。这个过程可能需要耗费数年时间。而最终协议被称为推荐标准而不是被称为标准的原因是，你依旧可以使用你自己的方法，并不强制使用它。所有 Web 开发人员都知道在开发应用程序时，有一些问题在所有的浏览器上都普遍存在，而许多这类问题出现的原因在于浏览器厂商没有遵循 W3C 的推荐标准，或者浏览器的某些特性在 W3C 推荐标准定稿前无法开发。本书涉及的大部分 XML 技术都有与之相对应的 W3C 推荐标准，但也有一些仍处于草案阶段。此外有一些与 XML 相关的标准源自 W3C 以外的其他标准组织，这些标准也没有正式的 W3C 推荐标准，如第 11 章中讨论的 SAX。

XML 来源于 SGML 的一个子集，而 HTML 是基于 SGML 的一个应用程序。XML 并没有规定一个文件的整体格式，也没有规定可以添加的元数据，它仅指定了一些规则。这意味着 XML 保留了 SGML 的灵活性，去掉了 SGML 的复杂性。例如，有一个标准的文本文件，其中包含了应用程序的用户列表：

```
Joe Fawcett
Danny Ayers
Catherine Middleton
```

这个文件并不包含元数据；你知道这是用户列表。这些人名在 XML 文件中表述的方式为：

```
<applicationUsers>
  <user firstName="Joe" lastName="Fawcett" />
  <user firstName="Danny" lastName="Ayers" />
  <user firstName="Catherine" lastName="Middleton" />
</applicationUsers>
```

虽然应用程序只通过这个文件依然无法知道如何去处理 user，也不知道 firstName 表示什么，但与原来的文本文件相比，XML 文件更能直观地表达这些数据的含义。比起毫无格式的文本数据，应用程序更容易读取使用 XML 格式的数据，并对这些数据进行正确处理。

几乎所有 XML 文件都是由两种内容组成的，它们被称为元素和属性。在之前的例子里，元素是 applicationUsers 和 user，而属性指的是 firstName 和 lastName。

不过这种记录元数据的方式也有缺点，那就是会增加文件大小。原文件中只有 43 个字符，但为了保存元数据，增加了近 130 个字符，加入元数据后文件比原来大了 3 倍多。XML 的开发者相信，XML 的优势会弥补这一劣势，而事实上 XML 开发者们的一个箴言正是“使文件变简洁并不是我们的目标”，而这一决定后来让无数人追悔莫及。

注意：在本书随后的章节中，你会学习到多种优化 XML 文大小的方法。但这些方法都在一定程度上牺牲了文件的可读性和易用性。

通过下面的习题，你会了解应用程序在处理简单的文本文件和 XML 文件上的不同点。在这个练习中，浏览器事先并没有缓存这两个文件，你会发现比起文本文件，浏览器能从 XML 文件中获得更多的元数据。

试一试　使用浏览器打开一个 XML 文件

这个例子展示了应用程序在处理 XML 文件和纯文本文件上的差异。

(1) 使用记事本或者文本编辑器创建一个文本文件，文本文件的内容是上面提到的不包含元数据的用户列表。

(2) 将这个文件保存为 appUsers.txt。

(3) 打开一个浏览器窗口，在地址栏输入 appUsers.txt 的文件路径。可以看到类似图 1-1 中展现的内容，它只是一个简单的列表。

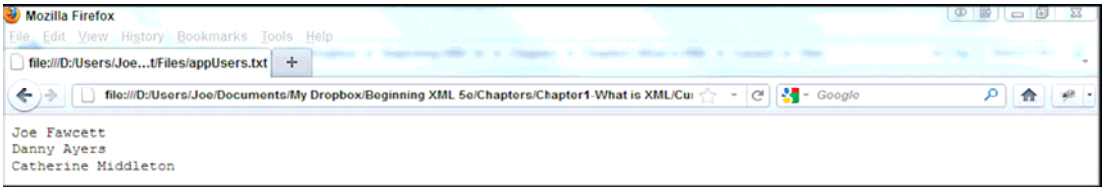

图 1-1

(4) 创建一个 XML 格式的文本文件，并把它保存为 appUsers.xml。如果使用记事本来创建这个文件，在保存时请务必用引号把整个文件名括起来请确保这个文件不会以.txt 为扩展名。

(5) 使用浏览器打开这个文件后，你将会看到类似图 1-2 中展示的内容。

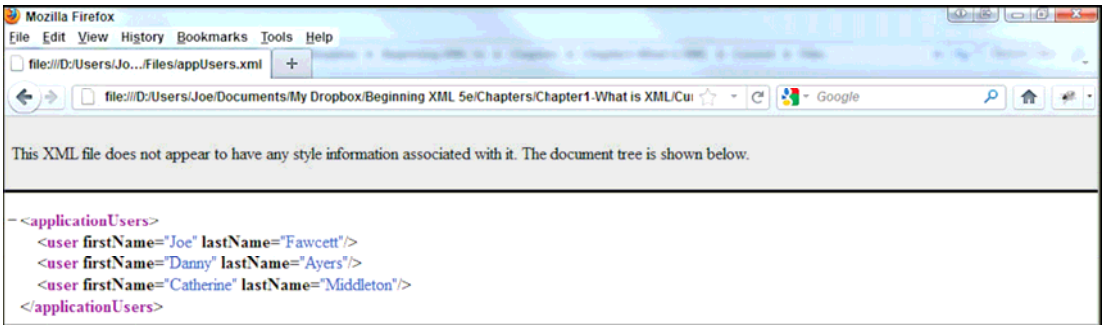

图 1-2

警告：如果使用 IE 浏览器来运行本书中的脚本，可能需要单击浏览器菜单栏上的“工具 | Internet 选项”，并选择“高级”选项卡。在“安全”项中勾选“允许活动内容在我的计算机上的文件中运行”来允许执行本地脚本。

正如上面所看到的，应用程序处理 XML 文件的方式和纯文本文件不同。浏览器用不同的颜色标识出了元数据，而且 applicationUsers 这个标签可以被收缩或展开。即使浏览器不知道这个文件里保存了三个不同的应用程序用户，它也能知道这个文件中有一部分数据要以与其他数据不同的方式来处理，并以一种简单易懂的方式将数据分级显示出来。

示例说明

浏览器通过 XML 样式表文件或 XML 样式转换插件来处理 XML 文件。XML 样式表文件是一种基于文本的 XML 格式的文件，通过它可以转换数据的格式。常应用于将 XML 格式的数据转换为 HTML 格式或其他格式的数据场景中，也能通过它将 XML 格式的数据转换为纯文本文件。在上面的习题中，浏览器把 XML 格式的数据转换为 HTML 格式，以不同的颜色显示标签，而且标签还可以折叠。样式转换插件的内容将会在第 8 章“XSLT”中讨论。

注意：如果你希望看到火狐浏览器使用的默认样式表，可以在火狐浏览器的地址栏中输入 chrome://global/content/xml/XMLPrettyPrint.xsl。IE 浏览器也有类似的内嵌的样式表，但无法通过类似方式查看。IE 浏览器的内嵌样式表是 Microsoft 在相关标准出台前编写的 XSLT 版本样式表，且早已过时。

注意：在本章中你会多次利用浏览器来查看 XML 文件。因为这种方法有许多优点——方便，当 XML 文件编写错误时能获得合理的提示，而且如今的计算机一般来说都安装有浏览器。但浏览器并不适合在正式开发 XML 时使用，尤其不适合在下一个“试一试”那样的场景中使用。因为大部分浏览器都可以显示编写风格较差的 HTML 文件，而这种情况下你很难发现其中编写错误的地方，也无法进行调试。所以我们建议你在开发时使用专用的 XML 编辑器。第 2 章中会介绍一些 XML 编辑器。

1.3　XML 的优点

XML的目标之一是把数据和它的显示方式完全分离开来。同样的数据可用不同的方式显示出来。当通过互联网等方式传输数据时，带宽不会被浪费在传输数据的样式信息上。XML也不像HTML那样内嵌有数据的显示方式，它分离数据和显示方式的方法相当简单，这也是XML的主要优点之一。

1.3.1　XML 规则

为了保证显示方式和数据完全分离，XML 有着一套非常严格的规则，不过这对于用户来说也有好处。例如，在 appUsers.xml 文件中你可以看到用户的姓和名都被双引号括起来了。这是 XML 文件的必要条件，下面的内容就不是 XML 数据：

```
<applicationUsers>
  <user firstName=Joe lastName=Fawcett />
  <user firstName=Danny lastName=Ayers />
```

```
  <user firstName=Catherine lastName=Middleton />
</applicationUsers>
```

使用引号也很容易发现数据缺失的情况，例如：

```
<applicationUsers>
  <user lastName="Fawcett" />
  <user lastName="Ayers" />
  <user lastName="Middleton" />
</applicationUsers>
```

所有的用户名都缺失了。因为所有数据都在双引号中，所以这是一组合法的 XML 数据，应用程序很容易发现并处理这种情况。不合法的文件输入在影响应用程序正常工作前就能被拒绝。验证 XML 文件有效性的方法会在本书第 II 部分中介绍。

使用 XML 文件的另一个好处是，在向文件中添加新数据时，它很容易被扩展。例如应用程序的用户数据需要包含中间名，那么你可以通过增加一个名为 middleName 的属性的方式来达到这个目的：

```
<applicationUsers>
  <user firstName="Joe" middleName="John" lastName="Fawcett" />
  <user firstName="Danny" middleName="John" lastName="Ayers" />
  <user firstName="Catherine" middleName="Elizabeth" lastName="Middleton" />
</applicationUsers>
```

设想有这样一个应用程序，它读取的数据文件中只包含姓和名这两个字段，并将这些数据输出到屏幕的列表上。最初这个软件就是被设计成只显示用户的姓和名的，现在新的需求要求一部分应用程序也能显示用户的中间名。新版本的应用程序使用的 XML 文件只要增加一个名为 middleName 的属性就能满足这个要求。旧版本的应用程序也依然还可以使用这个 XML 文件，它只是忽略了新版本应用程序需要的中间名字段。如果使用纯文本文件来保存用户数据，就很难满足这种需求。如 appUsers.txt:

```
Joe John Fawcett
Danny John Ayers
Catherine Elizabeth Middleton
```

由于无法通过分隔符来界定数据的起止位置，旧版本的应用程序很可能会错误地解读文本文件中的第二列数据，即使中间名被放置到第三列上，也很可能使应用程序在解析文件时出现问题。而 XML 格式的数据很容易区分用户名的不同部分。

1.3.2　数据的分层表示形式

XML 格式的数据和纯文本文件相比另一个好处在于分层显示数据。假设有这样一个场景：在文件系统中，根目录内有许多文件和文件夹，而每一个文件夹内又包含子目录和文件，而子目录内还可能包含子目录和文件。如果使用文本文件来记录就可能是这样的一种形式：文件内容包含两列，一列表示文件路径，另一列说明这是一个文件还是一个子目录。

```
Path Type
C:\folder
C:\pagefile.sys file
```

```
C:\Program Files folder
C:\Program Files\desktop.ini file
C:\Program Files\Microsoft folder
C:\Program Files\Mozilla folder
C:\Windows folder
C:\Windows\System32 folder
C:\Temp folder
C:\Temp\~123.tmp file
C:\Temp\~345.tmp file
```

文件的内容既不优美也不方便阅读。编写这一段文档时也无法方便地将各列对齐。再来看看 XML 文件是如何表示这些信息的：

```
<folder name="C:\">
  <folder name="Program Files">
    <folder name="Microsoft">
    </folder>
    <folder name="Mozilla">
    </folder>
  </folder>
  <folder name="Windows>
    <folder name="System32">
    </folder>
  </folder>
  <folder name="Temp">
    <files>
      <file name="~123.tmp"></file>
      <file name="~345.tmp"></file>
    </files>
  </folder>
  <files>
    <file name="pagefile.sys"></file>
  </files>
</folder>
```

这种分层显示的方式更易于阅读，冗余数据更少，更易于解析。

1.3.3　通用性

XML 的主要优点在于它的通用性。在不同的应用程序间传递数据时，使用 XML 格式的数据能很容易达成格式上的共识，因为 XML 文件本身包含了元数据，应用程序不需要对数据格式再做加工处理。由于可以读取并解析 XML 文件的解析器性价比很高，任何人都可以按他们的要求生成应用程序能够处理的 XML 文件，同时这些文件也能被其他应用程序解读或重建。通用性的重要性可以参见本章开头描述二进制文件时所举的例子。在 Microsoft Word 2003 以前，Word 生成的文件都是二进制格式的。编写一个可以读取和创建这些文件的应用程序是一件相当费事的工作，而且这些应用程序常常不能正常工作。从 Word 2003 以后，所有 Word 文档都可以被保存为带文档结构的 XML 格式，这样其他应用程序(如 OfficeLibre)也能通过一些简单手段来创建 Word 文档。现在通常可以通过文本编辑器来修复损坏的 Word 文档，而在以前文档损坏就意味着数据全部丢失。由此可见 XML 确实是保存数据时的最佳选择。

注意：OfficeLibre 是一款开源应用程序，它的功能和其他办公软件（如 Microsoft Office）类似。它最初被称为 OpenOffice，当 OpenOffice 被 Oracle 收购后它便从中独立出来。你可以在 www.libreoffice.org 上下载到它。

1.4　XML 的实际应用

XML 语言的格式和上个世纪 90 年代中期它刚被创造出来时几乎相同，但现在 XML 文件能包含的内容更加自由。一些在初期版本时不允许使用的字符现在也允许使用了。不过 XML 的应用领域以及它所衍生出的相关技术标准却有了很大的变化。XML 的管理工具也发生了巨大的变化。在过去几年中尤为明显，五年前在浏览器上对 XML 数据进行的任何操作都需要大量的 JavaScript 的支持，也只有有限的浏览器支持 XML 格式的数据。如今已经有了许多优秀的脚本库，它们使 XML 文件可以被很容易地发送、接受或处理。主流浏览器上 XML 的差异性也在逐渐变小。近年来 XML 的另一个变化是，大家对 XML 语言不适用的场景达成了共识，只有少数顽固分子认为 XML 可以用来解决任何问题。接下来的章节会讨论不适合使用 XML 语言的场景。本章涉及的 XML 使用场景也会在以后的章节中进一步深入讨论。

注意：你可以在 www.w3.org/TR/xml 上看到 XML 的最新推荐标准文档。

注意：我们会在第 14 章和第 16 章中详细探讨 JavaScript 对象标记法（JavaScript Object Notation，JSON），它与 Web 服务和 Ajax 密切相关。如果你现在就希望了解到 JSON 的更多内容，可以访问 www.json.org。

1.4.1　数据和文档

到目前为止，你在本书中看到的 XML 使用的例子都是以数据为核心的。在这些例子里都是在原始数据上加上标记，使之更容易理解，更易用，并提高了数据的通用性。另一种 XML 的主要使用方式是以文档为核心的。这是一种带有元数据的较松散的数据结构(如一本书的某个章节或一份法律文件)。HTML 就是 SGML 语言在这方面的一种应用，而 XHTML 则更接近于 XML 语言在面向文档方面的应用。HTML 的设计目的就是希望人们能更方便地阅读这些数据而不是方便软件运行。XML 的设计者则希望这些数据既能方便人们阅读也能方便软件处理。在本书接下来的内容中，你会发现不同样式的 XML 数据处理方式也迥然不同。

以文档为核心的 XML 常用于为一个应用程序在多种平台上发布时提供重用支持。在支持文件格式不同的介质上发布应用时非常的实用。例如，数年前我曾经开发过一个用于提供财务培训资料的系统，后台数据库为这个系统保存了大量的文章、测验题目以及测验答案。所有这些东西都是以

一种类似 XHTML 的 XML 方式保存的，XHTML 是 HTML 的 XML 版。当负责准备培训资料的编辑在数据库里敲定了他所需要的培训资料内容以后，这些内容会被 XSLT(详细内容会在第 8 章中探讨)转存为适合 Web 显示以及打印的格式。在这种系统中使用以文档为核心的 XML 时，无论输出的文档内容需要怎么变化，都只要修改底层使用的数据就行了，因为这些修改可以被反映到当前使用的所有输出介质上。另外，如果需要在其他介质上输出数据，如在移动终端的 Web 浏览器上显示，那么只需要另外指定一种转换方式就可以了。

1.4.2　XML 场景

除了以文档为核心的场景外，XML也常被当作一种表现数据和存储数据的手段。这主要是由于XML灵活的特性，而且以XML方式存储或表示的数据易于人们阅读和计算机解析。本书的这一部分内容描述了一些XML的常用场景，并简单描述XML适用于这些场景的原因。

1. 配置文件

几乎所有的现代配置文件都使用 XML。Visual Studio 项目文件和 Ant(Java 中用于控制软件构建过程的工具)使用的构建脚本都是这方面的例子。在配置文件中使用 XML 的原因在于，这种配置文件比使用传统的键值对方式的配置文件更易于解析也更容易体现层次关系。

2. Web 服务

冗长的 SOAP 风格的服务，简洁的 RESTful 的 Web 服务，以及现在许多人都热衷的 JSON，使用的都是 XML。XML 可以作为一种序列化跨平台对象的便捷方法，也可以作为一种通用的应用程序返回结果集的手段。SOAP 风格的服务(将会在第 15 章和第 16 章中深入讨论)也使用一种被称为 Web 服务描述语言(Web Services Description Language，WSDL)的 XML 格式描述的。WSDL 提供一个 Web 服务和功能的完整描述，包括这个 Web 服务的初始化请求的格式、确认响应以及正确调用服务的细节、主机名、端口号和 URL 格式。

3. 网页内容

虽然许多人认为 XHTML(XML 版本的 HTML)并没有真正流行起来，而且终将被 HTML5 取代，但它目前仍然被广泛应用在网络上。有许多网页内容被保存为 XML 格式，这些网页内容在需要时可以在服务器端和客户端之间转换。把网页内容保存为 XML 格式的原因在于这些内容可以被重用，也因为这样可以节省传输带宽和存储成本。网页内容需要以 HTML 表的形式显示，当 XML 把它和用于传输的代码一起编译时更节省空间。

4. 文档管理

XML 除了被用于存储通过网络提交数据的实际内容以外，它还在文档管理系统被广泛用于保存文件书签和管理元数据，常结合传统的关系型数据库系统使用。XML 用于保存文档的作者、创建日期、修订情况等信息。把这些额外信息和文件的具体内容集中保存意味着文档的所有相关内容都被保存在一处，使这些信息在额外提取时更方便，也确保元数据在描述数据时不被孤立。

5. 数据库系统

大多数现代高端数据库系统，如 Oracle 和 SQL Server，都能保存 XML 格式的文件。这对 XML 来说是个好消息，因为在传统的数据库中许多类型的 XML 数据不适合存储为关系型结构(如表和联接)。例如，一个产品表可能需要保存一些 XML 格式的结构信息，在网页显示或打印时需要这些结构信息。这些信息不能被缩减为简单的表单，几乎不需要修改，或者只需要添加一个新的部分就可以为另一种语言提供支持。如果要修改的文档在数据库中有专门的一列用于保存它所包含的 XML 信息，那么这种修改就非常容易而且简单。这一列 XML 信息需要能以 XQuery 语言来进行修改，XQuery 会在本章的后续内容中探讨。Oracle 和 SQL Server 以及一些开源数据库(如 MySQL)提供了用于保存 XML 信息的数据类型。这些数据类型提供访问和修改 XML 信息的相关方法。

6. 图像表示

矢量图可以使用 XML 来表示。可伸缩矢量图形格式(SVG)是目前最流行的矢量图格式。与传统位图图像格式相比，XML 格式的好处在于其表示的图像更易于操作。图像伸缩等操作由 XML 支持，而不需要其他复杂计算。

7.商务互用性

如今数以百计的公司使用标准 XML 来描述日常交易中使用的不同实体，这是 XML 最广泛的应用。详细包括：

- 医疗数据
- 如证券、股权、货币兑换等金融交易
- 商用和民用不动产
- 法律文书
- 数学和科学公式

1.4.3　XML 技术

你可以使用一些相关技术、标准和模型来实现上面提到的使用场景。这也是本书的主要内容，下面是本书内容的概述。

1. XML 解析器

在对 XML 文档进行操作前首先需要解析它，所谓解析就是将其内容进行分解并按一定内部规则重建。虽然 XML 文件是简单的文本文件，但传统的字符串处理方法(如取子串、计算字符串长度等)和正则表达式并不适合用于从中提取信息。由于 XML 内容丰富，使用灵活，基于字符串的处理方式并不适合处理 XML 数据。

现在有许多 XML 解析器——有的是免费的，有的是收费的——可以很方便得到解析结果。在本书中你会用到各种各样的解析器。早期应用程序使用自建的 XML 解析器的原因之一，是无法在项目中使用当时的外部解析器，当时的外部解析器无论在使用的内存上还是占用的磁盘空间上都过于庞大而导致无法封装在项目文件中。不过现在出现了许多高效的轻量级的解析器，这意味着自己再去开发一个解析器只是一个浪费资源的工作。

下面列举了目前常见的解析器：

- 微软的核心 XML 服务(Microsoft Core XML Services，MSXML)：这是 Microsoft 的标准 XML 工具包，包含有一个解析器。它对外公开了许多 COM 对象接口，供旧版本的 Visual Basic(vb6 或者更低的版本)通过 C++或脚本调用。Visuual Basic 最新的版本是 6.0，它已经不再开发新版本，只有关于修复地址错误和其他安全性措施的补丁包还在发布。在你编写应用程序时很可能不会用到这个解析器，这只给你在使用旧版本 IE 浏览器(IE6 或更低版本)来解析 XML 时提供一个选择。这些旧版本浏览器使用 ActiveX 技术调用 MSXML 解析器，这在某些情况下会导致安全性问题。IE7 及其以后的版本提供了一个内置的解析器和跨浏览器的 XML 支持库。如果高版本的 IE 浏览器可以在你的开发中使用，那么请尽量选用它。
- System.Xml.XmlDocument：这是 Microsoft 的.NET 库中的一个类，.NET 库中包含大量的 XML 开发时使用的类。它们都是标准的文件对象模型(Document Object Model，DOM)并额外引入了一些功能，使 XML 的阅读、编写和处理变得更容易。这部分内容将在下一节详细描述。目前业内均趋于使用 DOM 解析器，不过微软也提供了许多其他 XML 处理方式，我们将会在后续章节中探讨它们。
- Saxon：大多数专业 XML 开发团队都会认为，Saxon 是一款引领 XML 的产品。Saxon 提供 XML 的解析、转化和查询工具，它来自 Dr. Michael Kay 的软件公司。Dr. Michael Kay 撰写过许多关于 XML 和相关技术的高级编程教材。Saxon 有一个非常友好的界面，它通过使用 DOM 提供 XML 交互。Saxon 有 Java 和.Net 版本，其基础版可以免费下载和使用。
- Java 内置的解析器：Java 库也提供 XML 解析器。它以功能简单但适用于处理大多数 XML 工作(如解析和验证文档)闻名。你可以使用其他第三方解析器(如 Apache 的 Xerces 或 Saxon)来替代它。
- Xerces：Xerces 是由知名开源组织，Apache 软件基金会开发，它可以在 Java 上使用。它是许多基于 Java 的 XML 应用程序的基础，比 Java 内置的解析器更常见。

2. 文件对象模型(Document Object Model，DOM)

一旦 XML 解析器完成了它的工作，就会在内存中生成相应的数据模型。这些模型提供了提取和修改 XML 信息的方法。例如，可调用诸如 createElement 的方法来在要输出的文档中生成新的元素，使用诸如 documentElement 的属性定位文档的根元素(在之前的例子中就是 applicationUsers 节点)。

DOM 是最早使用的模型之一。当时这一模型并没有 XML 相关的标准，它只能用于 HTML 文件。DOM 的核心是使用树型结构表现一个 XML 文档。从这个树型结构的根节点开始遍历整个树结构，按照使用者的要求提取或插入数据。虽然 DOM 被使用了很多年，但它依然难以使用。而且 DOM 需要耗费的内存也很大。打开一个占用 1MB 磁盘空间的 XML 文件需要 5MB 的内存。当需要打开大 XML 文件时这显然会带来麻烦。由于 DOM 有这些问题，而且它只是处理 XML 文件的一个中间步骤，这导致了其他数据模型的出现。不过在从 XML 或 HTML 中提取一小部分信息时，DOM 还是有着广泛的应用，尤其是在跨浏览器支持上，DOM 也被许多目前流行的脚本库使用，例如 jQuery。

3. DTD 和 XML Schemas

文件类型定义(document type definitions，DTD)和 XML Schemas 均用于定义 XML 文档的结构和

其包含的数据。它们可以用来验证接受的文档是否符合规定格式。DTD 是一个旧版本的标准，这个标准也被 SGML 囊括了。它们目前正逐步被 XML Schemas 所取代，不过仍普遍被用于(X)HTML 中。它们拥有一些 XML 没有的特性，如可以创建实体声明(这部分内容会在第 4 章中探讨)，可以增加默认属性。XML Schemas 通常提供更多功能，它们在编写 XML 代码上也有优点，数据和数据架构都可以使用同样的工具。而另一方面 DTD 的数据和数据架构则使用完全不同的格式，这使 DTD 难以使用。另外 DTD 和 XML Schema 也都被用于帮助 XML 开发者验证文件。现在大多数 XML 编辑器都允许用户创建指定格式的 XML 文件。在用户编辑文件时，它们会提示用户可用的选择，当用户在错误的地方使用某个元素或属性时也会发出警告。虽然人们对 XML Schemas 的发展前景心存疑虑，但越来越多的人加入到使用 XML Schemas 的队伍中来。

除此之外也有其他方法可以验证接受的文档的格式，比如无法使用 DTDs 或 XML Schemas 的场景下就应该使用其他方法。验证的替代方案将在第 6 章中详细介绍。我们将在第 4 章和第 5 章中分别讨论 DTD 和 XML Schemas。

注意：当阅读 XHTML 文档的源码时，会在文件开头看到类似如下内容表示的对 DTD 的引用：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

4. XML 名称空间

在最初的 XML 推荐标准发布后不久，XML 名称空间就被引入到 XML 的定义中。它曾以难懂和难用闻名。名称空间本质上来说是一种 XML 的分组方式。如果一种或两种不同的格式需要在一起使用，它们自己的元素名就能被分组到各自的名称空间下，确保不会有歧义。在开发人员给不同格式下的元素命名相同的名称时这种分组尤为重要。同样的理念在.Net 和 Java 软件开发中一直在使用，例如开发人员设计了一个用于描述某种 XML 文档类，名为 XmlDocument。为了防止这个类和其他可能存在的同名类起冲突，这个类被放置于一个名称空间(.Net 术语)或一个包(Java 术语)内。这个类的全名可能就是 Wrox.Entities.XmlDocument，这样把它和 Microsoft 的 System.Xml.XmlDocument 区分开。第 3 章详细描述了名称空间。

5. XPath

许多 XML 技术都用到了 XPath。通过它可以定位指定的元素或属性(或其组建码块)。它的工作方式类似文件系统，从根节点开始按层次遍历，直到找到目标。例如我们想在 appUsers.xml 这个文件中找到所有用户。XPath 返回的路径就是：

```
/applicationUsers/user
```

路径遍历始于根节点(以斜杆“/”表示)，选择 applicationUsers 元素，发现所有的 user 元素都位于此。XPaths 的功能十分强大，它支持多种文档遍历方式也支持针对特定区域的谓词查询。XPath 除了被用在 XSLT 里外也被用于 XQuery、XML Schemas 和其他 XML 相关的技术中。我们会在第 7 章中详细学习 XPath。

6. XSLT

XPath 主要应用于可扩展样式表语言转换(eXtensible Stylesheet Language Transformations，XSLT)中。XSLT 是一种强有力的转换文件格式的方法。起初它只能将 XML 文件转化为其他形式的文本文件。从 2.0 版起，它也接受任意的文本文件作为输入。XSLT 是一种描述性语言，它根据源文件使用模板定义结果。

服务器端或浏览器端在将 XML 转换成(X)HTML 时也常用到 XSLT。在客户端做文件转换的好处是，把文件处理的表述过程放在应用层上处理，解决了显示问题。另外还能减轻服务器端的资源占用，提高服务器端的响应时间，也减少了服务器和浏览器间需要传输的数据量。这里有一个很特殊的案例：要处理的数据实际上是要以表格的形式展示，大部分数据内容都差不多。这种情况下直接传递 HTML 数据可能导致服务器和客户端之间要传输的数据成倍地增长。

在接下来的“试一试“中，我们可以看到经过特殊设计的浏览器是如何接受 XML 格式的输入并把它转换成指定格式的数据。在这里你不用深究 XSLT(把它留到第 8 章)，但你需要对 XML 是如何使你从看到的内容中获取你关注的数据有一个清楚的认识。

试一试　XSLT 在浏览器中的应用

使用之前创建的 appUsers.xml 文件示范如何在浏览器中实现一个基本转换：

(1) 使用文本编辑器在 appUsers.xml 所在的文件目录下创建包含以下内容的文件，并将其保存为 appUsers.xslt：


可从
wrox.com
下载源代码

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
     <html>
       <head>
         <title>Application Users</title>
       </head>
       <body>
         <table>
           <thead>
             <tr>
               <th>First Name</th>
               <th>Last Name</th>
             </tr>
           </thead>
           <tbody>
             <xsl:apply-templates select="applicationUsers/user" />
           </tbody>
         </table>
       </body>
     </html>
   </xsl:template>

  <xsl:template match="user">
    <tr>
      <td>
        <xsl:value-of select="@firstName"/>
      </td>
      <td>
        <xsl:value-of select="@lastName"/>
```

```
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

代码段 appUsers.xslt

(2) 按下面的内容修改appUsers.xml，使浏览器在打开它时使用我们刚才创建的XSLT来转换它，而非使用浏览器内置的默认转换器。把修改后的文件保存为 appUsersWithXslt.xml。

可从 wrox.com 下载源代码

```
<?xml-stylesheet type="text/xsl" href="appUsers.xslt" ?>
<applicationUsers>
  <user firstName="Joe" lastName="Fawcett" />
  <user firstName="Danny" lastName="Ayers" />
  <user firstName="Catherine" lastName="Middleton" />
</applicationUsers>
```

代码段 appUsersWithXslt.xml

(3) 使用浏览器打开 appUsersWithXslt.xml。你会得到类似图 1-3 所示的结果。

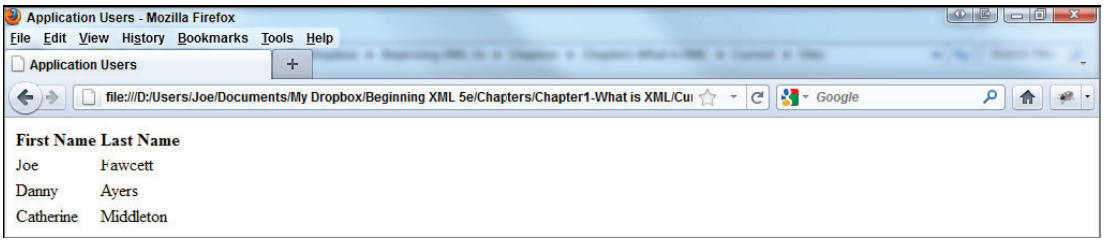

图 1-3

示例说明

当浏览器发现 XML 文件的第一行包含以下内容时：

```
<?xml-stylesheet type="text/xsl" href="appUsers.xslt" ?>
```

浏览器会使用我们创建的 appUsers.xslt 中指定的样式来显示内容，而不是像图 1-2 那样使用默认样式来转换显示。

appUsers.xslt 里有两组 xsl:template 标签。第一组标签指定了要显示的 HTML 文件的基本结构，为其绘制了一个 HTML 表格；第二组标签对 xml 文件中所有的 user 元素起作用，为每个 user 元素生成一列数据。转换完成后，生成的代码就和传统的 HTML 页面代码一样。该转换操作实际生成的代码如下所示：

```
<html>
  <head>
    <title>Application Users</title>
  </head>
  <body>
    <table>
      <thead>
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
```

```
      </thead>
      <tbody>
        <tr>
          <td>Joe</td>
          <td>Fawcett</td>
        </tr>
        <tr>
          <td>Danny</td>
          <td>Ayers</td>
        </tr>
        <tr>
          <td>Catherine</td>
          <td>Middleton</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

7. XQuery

XQuery 与 XSLT 有许多共同特征，所以在 XML 开发时一个很普遍的问题是“该用 XSLT 还是 XOquery 呢？”，回答是“依情况而定。”XQuery 和 XSLT 类似，都能操作单文档，但它更常用于多文档处理上，当这些文档保存在关系型数据库上时这种用法更为普遍。如果使用 XQuery 来处理前面的“试一试”中的 appUsers.xml 文件，为其生成以表格形式显示用户的 HTML 页面，所需的 XQuery 文件如下所示：

```
<html>
  <head>
    <title>Application Users</title>
  </head>
  <body>
    <table>
      <thead>
        <tr>
          <th>First Name</th>
          <th>Last Name</th>
        </tr>
      </thead>
      <tbody>
    {for $user in doc("appUsers.xml")/applicationUsers/user
    return <tr><td>{data($user/@firstName)}</td>
    <td>{data($user/@lastName)}</td></tr>}
      </tbody>
    </table>
  </body>
</html>
```

许多查询语句和前面的 XSLT 文件中的类似。和 XSLT 文件相比，一个主要的不同之处是 XQuery 本身不是 XML 格式的。这使需要编写的代码更少，比 XSLT 更容易编写。另一方面，由于它不是 XML 格式的，它不能通过标准 XML 编辑器编写也无法被 XML 解析器解析。只能通过专门的编辑器来编写，自定义构建软件来处理。

注意：XQuery 有一个基于 XML 的版本，称为 XQueryX。不过它并没有被大众接受，几乎所有在线使用的 XQuery 都是非 XML 的。

编写 XQuery 与 XSLT 相比，主要的不同体现在语法上。XQuery 使用花括号({})标记文档中需要使用处理器处理的部分，文档其他内容则简单地逐字输出。

在上面的例子中，实际要处理的代码部分就是：

```
{for $user in doc("appUsers.xml")/applicationUsers/user
   return <tr><td>{data($user/@firstName)}</td>
   <td>{data($user/@lastName)}</td></tr>}
```

这段代码使用 doc()函数从外部文件 appUsers.xml 中读取信息，对文件中的每个 user 元素创建一个<tr>元素。XQuery 将在第 9 章中深入探讨。

多数情况下，选择 XSLT 还是 XQuery 只由开发人员的个人喜好决定。如果开发人员希望代码风格简洁，可读性较高，或应用程序需要处理数据库中的大量文档，那么有着简单语法且针对多文档处理的 XQuery 会是一个更好的选择。如果开发人员希望代码更容易被支持标准 XML 的软件所解析，或者应用程序的主要功能是将现有的 XML 文件转换为其他格式的文件，而不是重新创建一个新文件，那么使用 XSLT 将是一个更好的选择。

8. XML 管道

当单个步骤不能获得用户希望的输出时，就需要使用 XML 管道技术。例如，我们无法用一个 XML 转换来应对应用程序所能接受的所有类型的文件。这种情况下，可能需要先针对输入数据做初步的转换，然后再使用一种通用的转换方法来实现最终的转换。另一个 XML 管道应用的例子是，输入数据在进行转换前需要经过验证。过去这些管道或工作流是以非正式的方式创建的。如今已经有了一个公认的标准来定义管道。XProc 是 W3C 的推荐标准，你可以在 www.w3.org/TR/xproc 上找到相关资料。XML 管道目前只有极少数的应用，不过如果你需要使用这种类型的工作流，那么 XProc 还是值得考虑的。

1.5　小结

- 在 XML 出现前，处理二进制和纯文本文件的方式以及存在的问题。
- XML 是如何从 SGML 发展而来的
- XML 的基本模块：元素和属性
- XML 的利弊
- 基于文档的 XML 和基于数据的 XML
- XML 的实际应用
- XML 相关技术，如解析器、架构、XPath、XSLT 和 XQuery 转换

第 2 章将讨论 XML 的构建规则和 XML 文件的不同组成部分。

习　题

习题的参考答案在附录 A 中。

1. 修改本章中的 appUsers.xml 文档，移除属性而使用元素来存储数据。

2. 说明上题中创建的文件的主要缺点。牢记数据不仅占用磁盘空间，更重要的是它需要通过网络来传输。

本章要点

主　题	要　点
XML 出现前	大多数数据格式都是专有的，只能被一小部分应用程序所使用，不适合当今的分布式系统
XML 的目标	提高数据通用性，使用人机均可读的文件格式，将开发人员从每次读取或写入数据时都需要编写底层代码的负担中解放出来
标准化负责人	无，不过许多 XML 规范都是由万维网联盟(W3C)制定的
基于文档的 XML 和基于数据的 XML	有两种主要的 XML 格式：一种用于存储纯数据，如配置文件；另一种用于向文档中添加元数据，如 XHTML
依赖于 XML 的技术	有数百种依赖于 XML 的技术，最主要的是以下几种：XML Schemas，它用于验证文档格式； XSLT，主要用于将 XML 格式的文档转换为其他格式； XQuery，用于诸如保存在数据库内文档这种大量的文档集合中的查询操作； SOAP，使用 XML 向 Web 服务传递数据，接收 Web 服务返回的结果

第 2 章 良构的 XML

本章内容：

- 什么是良构的 XML 文档
- XML 文档的组成部分
- XML 文档是如何组织的

迄今为止，你已经了解了 XML 之前的历史、XML 出现的原因以及它的一些优缺点。你也已经快速浏览了本书中提到的与 XML 相关的一些技术。

在本章中将会使用一些规则来检验一个文档是否符合 XML 规范。这一知识点在两种情况下是必需的：第一种情况是，当你为自定义的数据设计 XML 格式时，你可以保证文档能够被标准 XML 解析器解析；另一种情况是，当你在设计一个从外部接收 XML 输入的系统时，可以保证接收到的 XML 的合法性。遗憾的是，许多用于将数据导出成 XML 的系统却破坏了规则。这也就意味着除非你能从根源上解决这个问题，否则只能诉诸一些非 XML 的工具来解决输入问题。这就带来了大量不必要的开发工作，并且违背了使用一种通用方法来进行数据表示的初衷。

除此之外，你还会学习基本的以及高级的 XML 组成部分。从最基本的元素和属性开始，了解它们如何构建为一个完整的文档的。你还会学习描述这些组成部分的现代术语。这也是本书较早期版本的重大变化之一，我们做了很大的努力使得 XML 世界能够拥有一个独立但是足够精确和广泛的处理 XML 的技术词汇表。它使得 XML 标准能够被清晰地书写并且形成了技术发展的基础。

2.1 良构的定义

纯粹地讲并没有什么所谓良构的 XML。一个文档要么是符合良好规范的 XML，要么是文本。但是通常来讲良构 XML 意味着它遵循 W3C 的 XML 推荐标准的如下规则：

- 文档内容与元数据分隔方式(标记)
- 标记的标识方式
- 它的组成部分
- 这些部分的显示顺序以及显示位置

不同的 XML 术语

当谈论 XML 时总会提到一个小问题就是它的组成部分有多种描述方式。之所以会产生这些不同的描述方式，主要有两个原因：

- 这些与 XML 相关的不同技术都有它们自己的行业术语，它们当中仅有少量是通用的。例如，文档对象模型(第 7 章所述)与 XSLT(第 8 章所述)对于同样一个概念有完全不同的词汇。
- 官方的 W3C XML 推荐远远滞后于 XML 的实际应用。在这些文档中使用的术语往往不同于白话。

本章试图在两处坚持使用 W3C 推荐的两个术语。第一个，Extensible Markup Language(可扩展性标记语言 www.w3.org/TR/xml)，它描述了词汇表示形式。简而言之，即 XML 如何在文本编辑器中创建。第二个，Infoset Recommendation(推荐信息集 www.w3.org/TR/xml-infoset/)，描述了 XML 文档的理想化抽象表示。

2.2　在文本编辑器中创建 XML

使用文本编辑器创建 XML 与在 Windows 中使用记事本或者在 linux 中使用 vim 编辑器一样简单。它也是我们讨论 XML 元素的第一步。通过创建 XML，你会逐渐创建一个示例文档。每一步你都会认识 XML 的组成部分并掌握创建它们需要遵守的规则。

2.2.1　禁止的字符

在编写 XML 之前需要知道的第一件事就是在 XML 文档中有一些字符是被禁止使用的。这些规则根据你所使用是 1.0 还是 1.1 版本而略有不同，1.1 版本的要求略微宽松一些。两个版本都禁止在文档中使用 null，它的十六进制表示为 0x0。在 1.0 版本中，也不能使用十六进制码在 0x01 和 0x19 之间的字符，除了以下三个：制表符(0x9)、换行(0xA)和回车(0xD)。

注意：这三个字符以及第四个字符：标准空格字符(0x20)统称为空白符，它们在 XML 有特殊的规则。本章后面将介绍这些规则。

例如，不能使用字符 0x7，它是响铃符(bell)，因为它在某些系统发出钟声或哔声。在 1.1 版本中，尽管这些控制字符的用途不常见，你还是可以使用它们。在下一节中你会看到如何设置特定的版本。在 Unicode 规范中有一些字符同样不能在 XML 中使用，但是一般不会遇到这些字符。可在 W3C 的 XML 推荐标准中找到一个完整列表。

2.2.2　XML 序言

文档的第一部分是序言。它是可选的并不一定必须存在，但是一旦出现则必须出现在开始部分。序言以一个 XML 声明开始，一个最简单的形式如下所示：

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

这个声明仅包含一条版本号信息。目前它不是 1.0 就是 1.1。有时声明还包含文档中用到的编码信息：

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

这里编码使用的是 UTF-8，它是 Unicode 编码的一种。

1. 使用 Unicode 编码

编码是一个将字符转化为等价的二进制表达的过程。一些编码只使用一个字节也就是 8 个比特(bit)，其他则使用多个。使用单字节的劣势在于没有追索的情况下能够被编码的字符个数是有限的。追索是指使用一个特殊的比特序列来表明其后续两个字节代表一个字符的方法或其他相近的解决方法。当 XML 处理器读入文档时，它必须知道使用的是什么编码。但是，这又是一个“鸡生蛋”问题，如果不知道编码方式怎么能知道在声明里放的是什么字符。这个问题的答案很简单，就是文件最开始的几个字节可以包含字节顺序标记，即 BOM。这使得解析器能够读懂声明中定义的编码方式。一旦解析器知道了编码，它便可以解码文档剩余的内容。如果由于某种原因，指定的编码方式并不是文档实际使用的，通常会得到一个异常或者在翻译内容时出错。如果你想了解确定编码方式整个工作过程，你可以参照如下 URL：www.w3.org/TR/2008/REC-xml-20081126/#sec-guessing。

Unicode是一种遵从国际化理念而设计的文本编码标准。它试图定义每一个可能的字符，确定它的名字并分配一个代码点。代码点是一个能够代表字符的数字。Unicode也为每一个字符分配了一个类别，比如字母、数字或标点符号。在本章后续部分你会看到如何在字符引用中使用这些代码点。

Unicode 的两个主要使用编码系统为：UTF-8 和 UTF-16。UTF 是 UCS 转换格式(UCS Transformation Format)的简称，UCS代表通用字符集。当中的数字代表了一个字符使用多少个比特来表示，8或16(也就是1或2个字节)。UTF-8之所以能够使用一个字节而UTF-16需要两个字节是因为UTF-8使用一个字节来表示最常用的字符，使用两个或三个字节表示不常用的字符。UTF-16使用两个字节表示大部分常用字符，其余的使用三个字节。这有点像你的键盘，小写字母和数字只需要按一次，而是用大写字母和其他字符则需要使用shift键。UTF-16的优势在于它对字符采用使用固定的两个字节来表示(极少数使用3个)，这使得解码变得更容易。而它的劣势在于当你只使用拉丁字母和标准数字以及标点符号时，它的文件大小比使用UTF-8要大得多。

所有 XML 处理器需要能够解析 UTF-8 和 UTF-16，即使这些是它们仅有的能够读入的。在没有指明编码信息的情况下 UTF-8 是默认的编码方式。尽管 Unicode 有诸多优势，许多文档仍然使用一些其他的编码方式，例如 ISO-8859-1、windows-1252 或 EBCDIC(一种在许多大型机上的编码方式)。你也会遇到一些使用 ASCII 编码的文件，这是一种基本字符集。曾经几乎所有的文件都是通过它编码的。ASCII 是 Unicode 的子集，因此任何支持 Unicode 的程序都可以对其进行解析。

注意：当你在浏览网页时，经常会看到在一个系统中编码并在另一个系统中解码的副作用，这就是会有一些无意义的字符穿插在可读的文本中。这是一些在一台机器上创建进而上传到第二台网络服务器上并被第三台运行着浏览器的机器阅读的文件的副产品。如果这三台相关机器不能够正确地翻译这些编码，那么就会出现一些字符被曲解的情况。你会注意到通常这些乱码不是 ASCII 编码，它们在不同系统中有不同的代码点。

在实际使用中 UTF-8 编码可能是最好的。因为它有一个广泛的字符集并且被所有的 XML 解析器支持。在没有指定编码方式的情况下，UTF-8 便是默认的编码方式。如果你真的在创建或者解析文件遇到了 UTF-8 不识别字符的问题，你仍然可以轻易地通过自己创建这些字符来解决。你将会在“实体与字符引用”一节中学习这种方法。另外，Unicode 规范随着字符增多而不断扩展。目前的版本可在 http://unicode.org 中找到。

2. 完成声明

既然已经确定了使用的编码方式，你就可以完成声明了。声明的最后一个部分决定了这个文档是不是独立的。

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

Example.xml

独立只应用于那些定义了 DTD 的文档，并且只有在使用 DTD 添加或修改内容时才适用。Example.xml 并没有使用 DTD(记住大多数的现代 XML 格式依赖于架构)，因此你可以将独立声明设置为“是”或者不用管它。

注意：DTD 意为文档类型定义。它是一种定义 XML 格式，描述 XML 默认内容以及规定 XML 内部的引用该如何翻译的方式。第 4 章将讲述 DTD。

如果你曾经使用过 DTD，那么一个 XHTML 文档的例子看起来应该是这样：<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN""www.w3.org/TR/xhtml1/DTD/xhtml1- transitional.dtd">。第 4 章将讲述关于 DTD 声明的更多细节。

有时 XML 序言还包含一些额外元素。这些可选的内容包括注释和处理指令。处理指令将在本章后续内容中讨论。注释通常为人所用并且并不是文档中的实际数据。它们以 <!--开头，以 -->结尾。下面是带有注释的 example.xml。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This is a comment that follows the XML declaration -->
```

总之，注释完全是为人所读的。你也许会将创建文件的日期、姓名以及其他作者信息包含到注释中。不过如果你认为文件只是被软件应用程序处理，就没有必要插入它们。

一旦完成了 XML 序言，就需要创建文档的根元素。接下来的部分详细描述了元素以及如何创建它们。

2.2.3　创建元素

元素是 XML 基本组成块，所有的文档都必须至少包含一个元素。所有元素都是使用两种方式之一来定义。最简单的情况，一个元素包含一个开始标签，它是一个左尖括号“<”，接着是元素的名称，比如 myElement。之后是右尖括号“>”。一个完整的开始标签是 <myElement>。元素的最后是一个结束标签，它由一个左尖括号、一个斜线、元素名称以及右尖括号组成。因此 <myElement>

的结束标签应该是</myElement>。你可以在开始标签的名称后面加入空格，比如<myElement >，但是不能在名字前面加空格，比如< myElement>。将这个加入 Example.xml，如下：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This is a comment that follows the XML declaration -->
  <myElement></myElement>
```

Example.xml

定义一个元素还有一种可选的语法，这种方式只适用于那些没有内容的元素：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This is a comment that follows the XML declaration -->
  <myElement />
```

这类元素称之为自封闭元素。

1. 命名风格

除了以上两种定义方式之外，元素有一些不同的命名风格，正如许多 IT 相关的事物一样，人们因此而受益。一种几乎大家都公认的风格是保持一致性。选择一种文档风格并贯穿始终。以下是一些如何命名元素的命名风格，它们的主要思想是如何在元素名中区分单词。

- Pascal-casing：将包含第一个单词在内的首字母大写：<MyElement />
- Camel-casing：与 Pascal 相似，区别是第一个单词字母小写：<myElement />.
- Underscored names：使用下划线来分隔单词：<my_element />.
- Hyphenated names：使用连字符分隔单词：<my-element />.

还有一些其他的命名风格，不过这四个似乎效果最好。

2. 命名规范

在对元素进行命名时，除了使用命名风格之外，也有一些你需要遵守的命名规范。主要规则如下：

- 元素名可以是下划线或是 Unicode 字符集中任何大小写字母。这就意味着你可以使用在英语和许多其他西方语言使用的罗马字母，在俄语及其相关语言使用的西里尔字母，希腊字母，或者许多其他在 Unicode 标准中定义的文字，例如泰语一级阿拉伯语等。
- 后缀字符可以是连接符或数字
- 名字区分大小写，因此开始和结束标签必须精确匹配
- 名字不能包含空格
- 所有以 XML 开头的名字，无论大小写，都是预留的并且不能被使用(尽管实际上许多解析器也允许这种情况存在)。

注意：元素名称是大小写敏感的并不意味着使用两个仅仅通过大小写区分的做法是明智的，例如<myElement />和<MyElement />。正如在 C#之类的大小写敏感编程语言中是一种不好的实践那样，在 XML 中不应该让元素有相似的名称。

理论上你可以使用冒号“：”作为名称的一部分，但是这和 XML 名称空间(第 3 章讲解)的处理方式相冲突，因此在实际中应避免使用。如果你想查看完整的元素命名规范，请访问：www.w3.org/TR/2008/REC-xml-20081126/#NT-Name.。

正确的规范元素对于创建一个良好规范的 XML 是至关重要的。表 2-1 提供了一些正确以及错误构建元素的例子。

表 2-1　合法与非法的元素

合 法 元 素	原　　因	非 法 元 素	原　　因
<myElement></myElement>	名字之后可以有空格	<my Element />	名字中不能包含空格
<my1stElement/>	名字内部可以包含数字	<1stName />	名字不能以数字开头
<myElement />	在自关闭元素中，名字与斜线之间可以有空格	< myElement />	禁止在开头使用空格
<my-Element />	名字内部可以出现连字符	<-myElement/>	第一个字母不能是连字符
<όνομα/>	Unicode 定义的非罗马字母也是允许出现的。本例中元素名称为希腊语“名字”	<myElement></MyElement>	开始与结束标签必须大小写一致

3. 根元素

写完序言的下一步就是创建根元素。所有的文档都必须有且仅有一个根元素。文档中其他元素在根元素下形成一个分层树。只能有一个根元素是 XML 的基本规定之一。不过这也招致了许多抱怨，许多人提出了一些比较适合拥有多个根元素的例子。例如当使用 XML 作为日志格式时，一个典型的日志文件如下：

```
<entry date="2012-03-03T10:09:53" type="audit">Failed logon attempt
  with username jfawcett</entry>
<entry date="2012-03-03T10:11:01" type="audit">Successful
  logon attempt with username jfawcett</entry>
<entry date="2012-03-03T10:12:11" type="information">Successful folder
  synchronisation for use jfawcett</entry>
```

这种格式很易于管理。每当机器试图添加一条记录条目时，它只需要打开相应的文件在最后写入一行。这可以说是所有系统的一个标准操作。但是这种格式的问题在于它并没有唯一的根元素，你必须添加一个根元素使得它符合良构性。

```
<log>
  <entry date="2012-03-03T10:09:53" type="audit">Failed logon attempt
    with username jfawcett</entry>
  <entry date="2012-03-03T10:11:01" type="audit">Successful
    logon attempt with username jfawcett</entry>
  <entry date="2012-03-03T10:12:11" type="information">
  Successful folder synchronisation for use jfawcett</entry>
</log>
```

但是现在由于仅有一个根元素，要想添加新的条目是很难的。一个简单的文件记录器不得不打开文件并找到结束日志标签(</log>)，然后才能添加一个新行。除此之外，也可以使用解析器来打开

文件，找到根元素(<log>)，在它所有的孩子元素 <entry>之后添加一个新的 <entry>。这个过程需要更多处理，当每分钟需要增加几十个条目时可能会带来问题。

尽管如此 XML 标准委员会仍然坚持他们的观点，他们认为拥有一个独立的全方位的根元素(主要一点是它更容易解析)比它所带来的诸如创建日志文件的问题更为重要。不过他们已经认同有必要有这样一种结构，我们称之为文档片段。文档片段不需要有一个根元素，但却不能被单独处理。它们需要嵌入到一个有唯一根元素的文档中进行处理。这个过程可以有许多方法来实现，第 4 章的“实体声明”一节会介绍其中的一些。

4. 其他元素

根元素下面可以是遵循相同命名规则的其他元素、属性，也可以是之前提到的自由文本。这些嵌套的元素可以用来表示独立或重复的数据，这取决于你想怎么表达。例如，根节点可以是<person>，它下面的元素可以表示一个人的特征，比如<biography>和 <address>。根元素也可以是<people>，它下面可以有一个或多个 <person>元素，每个<person>都有自己的子元素。可以向示例文档添加更多的元素和注释，如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is a comment that follows the XML declaration -->
<!DOCTYPE myElement [
  <!ENTITY nbsp " ">
]>
<myElement myFirstAttribute="One"mySecondAttribute="Two">
Here is some text with a non-breaking space in it.
  <anotherElement>
    <aNestedElement anotherAttribute="Some data here">
Some more text</aNestedElement>
    <!-- a second comment -->
  </anotherElement>
</myElement>
```

记住所有元素必须嵌套在根元素之下，因此下面这种有可能在 HTML 中侥幸通过的标记方式是不允许出现的：

```
<myElement>
  <elementA><elementB></elementA></elementB>
</myElement>
```

不能让元素的结束标签在它所嵌套的元素的结束标签之前出现。

2.2.4　属性

元素是 XML 的两大组成要素之一，另一种就是属性。属性是一个与元素相关的名称-值对。可在示例文档中添加许多属性，如下：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This is a comment that follows the XML declaration -->
  <myElement myFirstAttribute="One" mySecondAttribute="Two"></myElement>
```

属性的命名风格应该与元素所选择的风格保持一致。因此不要混淆搭配成这样：<applicationUser

first-name="Joe" />。即元素使用 camel-casing 风格而属性使用连字符风格。

属性也有一些规则：

- 属性由名称与取值组成，中间用等号连接。属性名的命名风格与对应元素的风格一致，如上例：myFirstAttribute。
- 属性值必须使用引号。你可以自主选择使用单引号或者双引号。可以在一些属性中使用单引号，另外一些属性中使用双引号，但是却不能在同一个属性中混用。
- 属性必须有取值部分，即使它可以是空引用。不允许有像HTML中可能出现的<option selected>出现。
- 每个元素中的属性名必须唯一。
- 如果使用双引号作为分隔符，则在取值部分不能使用它们。对单引号同样适用。

表 2-2 列举了一些正确和错误的使用例子：

表 2-2　合法与非法属性

合 法 属 性	原　　因	非 法 属 性	原　　因
<myElement value="Joe's attribute" />	单引号可以出现在双引号之中	<myElement 1stAttribute= "value" />	属性名不能以数字开头
<myElement value=' "a quoted value" ' />	双引号可以出现在单引号之中	<myElement value= 'Joe's attribute' />	单引号不能出现在单引号之后
		<myElement name="Joe" name="Fawcett" />	两个属性名称不能相同
		<myElement name='Joe" />	分隔符不匹配

2.2.5　元素与属性内容

属性值与元素均可包含字符数据(通常称之为文本)。在之前的代码段中已经出现过属性值的例子。类似元素的例子如下：

```
<myElement>Here is some character content</myElement>
```

除了之前介绍的规则，对于字符内容也有两个额外限制。有两个字符不能出现在属性值或者元素内容中：取值符(&)和左尖括号(<)。后者之所以不能使用是因为它用来分隔元素，这会使解析器无法分辨。前者不能使用是因为它用在实体与字符引用的前面。

1. 实体与字符引用

有些字符或者由于被规范禁止，或者由于不在文档选择的编码中，导致它们不能直接被文档使用。可以采取两种方法将它们插入到文档中。第一种方法称为实体引用。XML 中共有 5 种实体引用，如表 2-3 所示。

表 2-3　实体引用

字　符	引　用
&	&amp;
<	&lt;
>	&gt;
“	&quot;
‘	&apos;

引用以取址符开始和以分号结束。实际的引用出现在中间部分，它们是字符的缩写。例如，lt 代表 less than(小于)。因此不能使用&或<，而应该使用它们的引用来代替。当一个属性值中包含两种类型引号时，'和"就变得非常有用。

表 2-3 中的引用仅是内置的实体引用。如果想使用 DTD 也可以声明自己的引用——很快我们就会讲到这个例子。

字符引用形式相似。它以&#开头以分号结束，中间部分不是使用缩写而是使用字符的 Unicode 代码点。数字可以是十六进制或十进制。例如可以使用十六进制Ω 或十进制&937#;来代表希腊字母(Ω)。

在 XML 论坛上的一个普遍问题是如何表示“插入空格”——那些没有可视输出却用来分隔两个单词以防止重叠的字符。它在网页中一般用于格式化，并且用 来表示。有四种方法可以将该字符插入到 XML 文档中。第一种方法直接将其作为字符插入，这种方法完全不需要使用引用。例如，在 Microsoft word 中你可以通过键入 3A9 然后单击 Alt+X 的方式敲入Ω。不同的编辑器有不同的方法。“插入空格”的 Unicode 代码点是 xA0，因此你也可以直接使用这种方式处理。第二种和第三种方式分别使用十六进制的 和十进制 的字符引用。第四种方法需要在文档的开头创建 DTD 并声明实体。当这个字符在 XML 出现多次并且你想让读者更容易认识它时，你可以使用这种方法。在 HTML 中，引用 可以插入一个“插入空格”。类似的，在 XML 文档中可进行如下操作：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This is a comment that follows the XML declaration -->
<!DOCTYPE myElement [
  <!ENTITY nbsp " ">
]>
<myElement myFirstAttribute="One" mySecondAttribute="Two">
Here is some text with a non-breaking space in it.
</myElement>
```

DTD(第4章会详细描述)声明根元素名称为 myElement，之后是一个实体声明。无论 在文档的任何地方出现，解析器都会将其作为 Unicode 字符 A0来处理，也就是一个“插入空格”。

这种方式也可用来给多个字符添加引用。比如你想为版权创建一个引用，当你在任何地方输入&copyright 时都会输出© Wrox 2012。如果你想改变所有的引用都读入© Wrox 2013，你可以只在一个地方更新 DTD。其实现方法与前例类似，如下：

```
<!DOCTYPE myElement [
  <!ENTITY copyright "© Wrox 2012">
]>
```

更多引用类型参见第 4 章。

警告：务必记住，不能通过实体或字符引用的方式将禁止字符(例如 null)加入到文档中。

2. 元素与属性的取舍

在许多情况下必须对使用元素还是属性来代表数据进行选择。列举第 1 章中的 appUsers.xml 的例子来讲(程序清单 2-1)：

程序清单 2-1　appUsers.xml

```
<applicationUsers>
  <user firstName="Joe" lastName="Fawcett" />
  <user firstName="Danny" lastName="Ayers" />
  <user firstName="Catherine" lastName="Middleton" />
</applicationUsers>
```

可以选择使用元素来代表用户的名和姓，如程序清单 2-2 所示：

程序清单 2-2　appUsers-elementCentric.xml

```
<applicationUsers>
  <user>
    <firstName>Joe</firstName>
    <lastName>Fawcett</lastName>
  </user>
  <user>
    <firstName>Danny</firstName>
    <lastName>Ayers</lastName>
  </user>
  <user>
    <firstName>Catherine</firstName>
    <lastName>Middleton</lastName>
  </user>
</applicationUsers>
```

在使用元素还是属性上并没有固定的规则，但是下面是一些在做决定时需要考虑的东西。

使用属性的时机

当只有一条数据需要展示时，属性往往是一个好的选择。在列表 2-1 中一个人只能有一个名字，因此使用属性是最佳选择。属性名不能重复，在属性中放入一个用逗号分开的角色名字列表将使提取并修改这些数据变得困难。因此，如果需要为用户展示多次类似角色名这样的数据，则不得不使用元素。

使用属性会使文件变小，因为每个包含数据的元素中都需要一个结束标签和一个尖括号的小开销。这就意味着同样的数据量，元素会使用更多字符。当文件需要经常在网络上传输而带宽有限时，这是一个值得考虑的问题。

通常情况下，除非有特别原因，使用属性是一个好的选择。

使用元素的时机

当数据不是一个简单类型时，使用元素是非常有用的。所谓简单类型数据是指一些能以字符串形式作为属性值表达的一些文本或日期。因此像地址这种数据，最好按其组成部分分开以元素的形式展示，而不是通过具有分隔符的字符串将其挤压成属性值的形式来展示。

因此可以这样使用：

```
<person firstName="Joe" lastName="Fawcett">
  <address>
    <line1>Chapter House</line1>
    <line2>Crucifix Lane</line2>
    <city>London</city>
    <postCode>SE1 3JW</postCode>
    <country>England</country>
  </address>
</person>
```

而不是这样：

```
<person
    firstName="Joe"
    lastName="Fawcett"
    address="Chapter House, Crucifix Lane, London, SE1 3JW, England" />
```

当条目需要重复时也最好使用元素。为了将角色名关联到前面所述文件中的用户，最好使用下面这样的结构：

```
<applicationUsers>
  <user firstName="Joe" lastName="Fawcett">
    <roles>
      <role name="administrators" />
      <role name="general" />
    </roles
  <!-- other users here -->
</applicationUsers>
```

注意到每个角色只有一个名字，因此那部分使用的是属性而不是元素。

另一个使用元素的好处是它可以排序，而属性是无序的。可以将属性按照某种顺序放置在文档中，但是 XML 解析器在处理时会忽略这种先后顺序。如果需要数据条目保持某种特殊顺序，就需要选择元素。

其他使用元素的最主要场景是当需要有大量文本数据需要展示时。从技术角度看，你也可以在这种情况下使用属性，但你会得到一个如下所示的文件：

```
<longDocument data="In here is a very long piece of text that goes on for many,
many,
many,
```

```
many,
lines" />
```

这看起来很不规范。当一个文件拥有大量文本时，使用元素表示内容是更加直观的。可以使用元素中的 CDATA 来防止转义特殊字符。

2.2.6　处理指令

XML文档的另一个通用组件是处理指令。你已经在第 1 章中使用基于浏览器的XSL转换中见过一种处理指令。处理指令(PI)用来与使用XML的应用进行交互。它并不能直接被XML解析器使用。

处理指令明确了哪个应用应该执行指令以及它所需要的数据。一个常用的处理指令的例子是通知浏览器对 XML 进行转换，如下所示：

```
<?xml-stylesheet type="text/xsl" href="appUsers.xslt" ?
```

在这个例子中，xml-stylesheet 是目标。它后面是两个伪属性：type="text/xsl"和 href="appUsers.xslt"。它们并不是真实的属性，因为它们不需要遵守属性必须有名字和取值以及使用引号的规定。它们只是目标应用程序需要使用的数据。在此例中，浏览器会识别目标。并且知道 XML 在显示之前应该进行转换。第一个属性表明进行的是 XSL 的转换，并且在第二个属性中指出了它的位置。这个特殊的处理指令仅适用于少量应用，大部分情况下为浏览器。如果使用标准文本编辑器打开这个文件，里面的处理指令会被忽略。

注意：处理指令与 XML 声明看起来很相似，但是声明从技术角度看并不是一个处理指令，它也不能像处理指令那样处理。

2.2.7　CDATA 节

在文档中使用的一种更高级的结构称为 CDATA 节(CDATA 代表字符数据，表示它不需要被标记)。这是一种避免重复转义字符的方法。例如，假设你有一个简单的文档，它里面使用了小于号(<)。正常情况下这会被认为是标记的一部分，所以必须使用实体引用<来对它进行转义。转义的文档内容如下：

```
<conversionData>
  1 kilometer &lt; 1 mile
  1 pint &lt; 1 liter
  1 pound &lt; 1 kilogram
</conversionData>
```

如果你更倾向于在文本中直接使用更易读写小于号(<)，可将这部分元素的内容标记为CDATA节：

```
<conversionData><![CDATA[
  1 kilometer < 1 mile
  1 pint < 1 liter
  1 pound < 1 kilogram
]]></conversionData>
```

CDATA 节以 <![CDATA[开始，以]]>结束。CDATA 中的任何字符都会被当成纯文本而非标记，因此你可以在其中使用任何通常都需要进行转义的字符，比如小于号和取址符。如果需要表示 CDATA 节的结束符]]>，可以将其转义为：]]>。

CDATA 通常在 XHTML 中使用，它是 HTML 的 XML 版本。当在 XHTML 网页中嵌入 JavaScript 代码时，许多字符通常都需要转义。这常常会给 Javascript 解析器带来麻烦。相较而言，你可以像下面的例子一样将整个脚本部分放在 CDATA 节中。这个例子用来测试一个顾客是否试图转移出超过他账户余额的钱。

```
<script type=text/javascript>
//<![CDATA[
function validateTransfer(currentBalance, transferAmount)
{
  if (currentBalance > 0 && transferAmount < currentBalance)
  {
    return true;
  }
  alert("Insufficient funds to transfer the requested amount.");
  return false;
}
//]]>
</script>
```

由于文本被嵌套在 CDATA 节中，你就可以在其中书写标准格式的 javascript 了。相反如果测试的是转移的钱是否小于当前账户余额，你还需要转移&&和<，如下：

```
if (currentBalance > 0 &amp;&amp; transferAmount &lt; currentBalance)
```

这样的一行代码无论是人还是脚本解析器都很难翻译。

另一点值得注意是在 CDATA 节的开始和结束符之前出现的 Javascript 注释符(//)。这是为了让一些较老的浏览器知道如何处理这种结构。

记住，CDATA 节只是为了让人更容易理解。XML 解析器在处理上述两个例子时是没有区别的。一旦 XML 已经被解析，就无法知道字符是通过引用还是 CDATA 转义的。有人使用 CDATA 将一个 XML 文档嵌套在另一个里面，如下：

```
<myDocument>
  <someData>
    <myNestedDocument><![CDATA[
      <anotherDocument>This is bad practice</anotherDocument>
    ]]></myNestedDocument>
  </someData>
</myDocument>
```

这类 XML 很难工作，它应该被禁止。正确地处理多个文档而不让它们互相混淆的方式是使用名称空间，这将在第 3 章介绍。

2.3　高级 XML 解析

迄今已经介绍了 XML 文档所有的通用组件，下面来介绍一些更高级的组件。下面是三个与高级 XML 解析相关的主要讨论议题：

- XML 等价性——通过不同方式创建的文档如何能够被 XML 解析器等价地解析。
- 空白处理——空格与制表符如何被特殊对待
- 错误处理——文档包含错误的后果

2.3.1　XMl 等价性

XML 等价性是指文档虽然有不同的词汇表达方式，但是却能被 XML 解析器一视同仁。一旦文档被解析，将无法知道创建 XML 过程中是否使用了某种特殊风格。如下面两个文档所示，程序清单 2-3 和程序清单 2-4 有三处不同：

程序清单 2-3　文档 1

```
<exampleData source="web">
  <section><![CDATA This is some example data ]]></section>
  <section>Here's some more data</section>
</exampleData>
```

程序清单 2-4　文档 2

```
<exampleData source='web'>
  <section>This is some example data</section>
  <section>Here&apos;s some more data</section>
</exampleData>
```

三个词汇上的差异体现在：

(1) 第一个文件中属性值使用双引号，第二个使用单引号。

(2) 第一个文件中的第一个<section>元素中使用了 CDATA 节，而第二个没有。

(3) 第二个文件在第二个<section>元素中对撇号(')使用了实体引用，而第一个没有。

这两种表述都没有问题，完全取决于个人喜好。一旦其中一个文件被解析之后，你将无法确认哪个是它的源文件。对于解析器来说，属性值如何被引用以及其他的区别，对于 XML 的核心表述没有任何影响。因此，这些文档便实现了 XML 等价性。

多个版本的 XML 文档却形成一样的内存表述，这个现象也有一些消极的影响。例如，当你在转换文件时，如果希望能够将 CDTA 节中的数据与之外的文本区别对待，很遗憾这只能通过其他方式来实现。如果你想这样，可以通过其他的方式来处理这种变化。你可以使用一个非 XML 的工具对文件进行预处理，在那些需要特殊处理的元素上添加标记识别符。

类似地，在 XML 论坛中的一个普遍需求是希望能创建一个 XML 文档，它包含各种由引用而不是它本身所表示的字符。这与程序清单 2-4 类似，其中撇号用 '来表示。原因在于处理 XML 的软件“需要”这种特殊格式。事实上由于这两种格式是 XML 等价的，这说明接收应用程序并兼容 XML，否则这两种格式它都会接受。处理这种需求最好的方式是丢回去让开发者去修改相关的应用程序。显然有时这也是不太可能的，但是再怎么样也无法坚持一个单引号的表示方式。如果你

并没有从根源上解决这个问题的话，你将不得不找一个非基于 XML 的解决方法来满足这个特殊需求。

2.3.2　空白处理

空白符由空格符、制表符、换行符、回车符组成(对应 Unicode 编码为 0x20、0x9、0xA 和 0xD)。空白符并不包含本章前面介绍的广泛使用在网页上的“插入空格”。尽管“插入空格”并可不见，但是它也和其他字母或标点符号一样被处理。

空白符看上去并不会带来麻烦，人们读文档时也经常遇到空白行或有时两个空格连在一起的情况。但在 XML 中空白符需要小心使用，主要有如下两个原因：

- 首先，有时空白符是有意义的。拿一个标准的英文句子为例，每个单词都依靠空格来区分。如果没有这些空格，文本将会非常难读。另一方面，有时空白符则是无关痛痒。许多书籍在开头和最后都会有一些空白页，这些不会被错过但是也不会增加内容。XML 中的情况也类似，很快就可以看到一些相关的例子。
- 空白符在 XML 中之所以重要的第二个原因是由于不同的操作系统对于文件的行尾结束符有不同的转化方式。行尾结束符由换行符和回车符组成。例如在 Windows 生成的文件的行尾结束符是：回车+换行，这是为了模拟旧打印机的动作。相反，在 UNIX 环境中创建的文件的行尾结束符却只有一个换行符。

空白符的处理规则使得 XML 文档成了便携式数据格式，保证了它的一致性体验。比如，当处理 XML 文件时，行尾结束符(无论是回车加换行、换行或者是回车)都统一转换成换行符。

有意义的空白符与无意义的空白符

空白符在 XML 文档中被 XML 解析器处理的方式与在 HTML 中被浏览器处理的方式有很大的不同。如果你熟悉 HTML 开发你就会知道连续的空白字符会被合并，并且换行符和回车符通常会被忽略。下面的 HTML 标记例子中在单词之间有许多空白字符，中间有两个换行符，如下：

```
<p>Here is some text    with    lots of    whitespace

            That won't show in a browser</p>
```

图 2-1 显示了在浏览器中的效果。

图 2-1

可看到连续空格和换行符都没有被显示。这是因为 HTML 浏览器将多个空格合并成一个并且忽略了换行符。如果你想在 HTML 中显示一个换行符需要使用
元素来标记。

XML 并不像 HTML 那样严格，解析器保留了那些不属于标记部分的空白字符。如下面的代码段所示：

```
<chapter title="What is XML? ">
   <para> XML stands for Extensible Markup Language (presumably the original authors
  thought that sounded more exciting than EML) and has followed a path similar to
others in the software and IT world.</para>
</chapter>
```

在这个设计中，<chapter>元素中不会直接添加任何文本内容，而仅包含<para>元素。因此，在开始标签后面的空白和一般缩进只是为了使文档更加易读和易懂。在<chapter>元素的名称和属性title之间也有一些空格。所有这些空白都是毫无意义的。而在<para>元素中出现的空白都是有意义的。

XML解析器可以选择忽略没有意义的空白，但是它如何知道该元素没有直接的文本内容呢？如果文档有XML schema或者DTD的话它也只能知道元素的内容类型。Schema和DTD将会在第 4、5 章中完整介绍，我只想说无意义的空格不需要被解析器保留。如果你确实想让文档中所有的空格保留，可以在根元素或者其下的元素中添加一个特殊的属性：xml:space="preserve"。这会告诉解析器文档完整地保留。将其加入到前面所述例子中的 <chapter>元素：

```
<chapter xml:space="preserve" title="What is XML? ">
   <para> XML stands for Extensible Markup Language (presumably the original authors
  thought that sounded more exciting than EML) and has followed a path similar to
others in the software and IT world.</para>
</chapter>
```

如果你想将空白符处理设置回默认值，可以使用 xml:space="default"。

警告：如前所述，除非XML解析器断然知道这些空白是无意义的，否则它必须对其保留。遗憾的是微软基于COM的解析器(MSXML核心服务)并不是这么处理的，它会将它认为无意义的空格忽略掉。这给过去那些被迫使用这种解析器的开发者(例如使用IE工作的开发者)带来了许多问题。在某些情况下可以通过将解析器的preserveWhitespace属性设置为true来解决这个问题。

2.3.3 错误处理

为了确保你的文档符合 XML 规范，你需要解决一些障碍，例如：匹配标签、引用属性值、必要转义字符等。当 XML 解析器发现其中一条规范被打破时，它有两种处理方式。这取决于标准将其定义为“错误”还是“致命错误”。

可从一个错误中恢复并继续进行文档解析。而致命错误，正如它的名字那样，是不能恢复的。解析器只能选择报告该问题。如果必要时它也会报告其他错误，但是处理的最后将不能产生一个解析之后的文档。和良构性相关的错误都是致命的。

之所以采取这个严格的规定是因为看到了允许 HTML 开发者使用宽松的语法而对网络造成了不利的影响。浏览器接收了错误的 HTML，它便尝试猜测作者的意图。由于每种浏览器对于错误格式的内容有着一套不同的规则，这使得该 HTML 页显示方式不统一。这也意味着如果浏览器没有首先将内容通过各种算法提取有用信息，它将很难对网页进行处理。这也是基于的 XML 的 HTML(XHTML)为了解决的问题之一。遗憾的是在实际中却很难起效，因为它对于大多数人而言很难学习。此外，它也缺乏足够的工具集支持，许多浏览器，特别是 IE，不能正确地处理它。

最后可以比较一下 XML 和标准的编程语言严格错误检查机制。一些在编译时就进行严格类型检查，另一些则只在运行时出错。XML 的观点是尽量早些发现错误。这也意味着可能由于微小的瑕疵而导致文档解析失败。

大部分浏览器都有良好的错误报告机制，这也经常用来寻找那些文档不明显的错误。它们非常严格，即使遇到那些没有被定义为致命的错误也会终止处理。这一点对于几乎所有浏览器都很常见。这也符合只报告它们“可能”恢复的标准。在实际中，遇到错误时终止处理比试图通过猜测作者意图来修复它要容易得多。这会使解析器在处理文档时产生差异，会遇到之前提到的网页处理时我们不想看到的相似问题。下面的“试一试”故意创建了一个格式错误的文件用以证明浏览器中如何进行错误报告。

试一试　用浏览器寻找错误

要了解浏览器如何报告错误，请使用下列代码：

可从
Wrox.com
下载源代码

```
<?xml version="1.0" encoding="utf-8"?>
<pangrams createdOn="2012-01-04T10:19:45'>
  <!-- This file is designed to show
  how errors are reported in a browser -->
  <pangram>The quick brown fox jumps over the lazy dog.</pangram>
  <pangram>Pack my box with five dozen liquor jugs.</pangram>
  <pangram>Glib jocks quiz nymph to vex dwarf.</pangram>
  <pangram>The five boxing wizards jump quickly.</Pangram>
  <pangram>What you write deserves better than a jiggling, shaky,
     inexact & questionably fuzzy approximation of blur</pangram>
</pangrams>
```

XmlFileWithErrors.xml

该文件包含 3 个错误。它们可能不是很明显，这取决于你对 XML 的熟悉程度以及校对能力。在图 2-2 中浏览器对这些错误的报告形式与你自己所想的可能有所不同。在本例中浏览器在报告错误之后停止工作并将到该点为止的内容尽可能显示出来。示例中使用的是谷歌的 Chrome 浏览器。

(1) 在本地浏览文件，会看到第一个错误报告，如图 2-2 所示。

图 2-2

(2) 仔细查看源文件，可以看到 createdOn 属性使用了不匹配的引号——使用双引号开头却使用单引号结尾。但是这并不是报告的错误。解析器报告有一个未转义的<。这是因为它认为属性定义还没有结束但是却突然遇到一个非法字符。迄今它还没有读到任何文本内容，因此除了错误消息之处没有显示任何东西。

(3) 通过使用双引号替换单引号来修正这个错误后，在 Chrome 中重新打开该文件，将得到图 2-3 所示的一条新信息。

图 2-3

(4) 这一次 Chrome 报告不匹配的标签名称。第四个<pangram>元素的开始标签为<pangram> 而结束标签为 </Pangram>，标签必须大小写匹配。但是这次已经读入了一些内容，所以开始的三个<pangram>元素的内容显示了出来。修改这个不匹配并重新打开文件，会报告最后一个错误，如图 2-4 所示。

图 2-4

(5) 这个有点让人疑惑的报告声称有一个实体引用没有名称。这是因为在 XML 中正如之前描述的那样取址符(&)用来开始一个字符或实体引用。在文件中没有名称也没有结束分号，因此解析器认为是引用异常，而不是给出忘记转义&这种简单的解释。修改最后一个错误，图 2-5 所示为整个文件的内容。

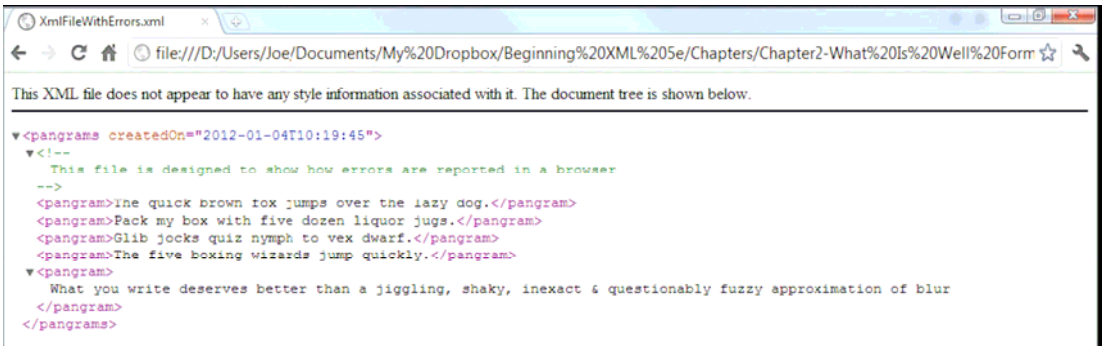

图 2-5

这次文件显示为一个熟悉的树型，可以看到注释，也可以展开和折叠各个部分。

示例说明

XML解析器在检查错误时使用的是一种序列化的方式。它们开始通读文件，一旦发现错误立即报告，并停止进一步的解析工作。这就解释了为什么Chrome每次只显示一个错误，当修改了错误之

后，解析器便可以进一步解析下面的内容。直到所有三个错误被修复，浏览器就可以显示整个XML。

注意：如果你想知道文件的内容，一个 pangram 就是在一种语言中每一个字母表的字母都至少使用一次的句子。

仔细阅读 XML 标准以了解更多关于哪些错误是致命错误而哪些不是的规定。致命错误需要被明确指出，其他损害都是可恢复错误。正如之前所述，大部分解析器将错误全部归结为致命错误。关于这一点的最佳实践是确保所有创建的文件都不包含错误，所有接收的文件都是正确的。如果确实收到了包含错误的文件，在不能拒绝它们的情况下，你的最佳选择是在将它们送入 XML 处理流程之前使用非 XML 的方法修正这些错误。

既然你已经了解了 XML 文件的组成构建块，创建 XML 文件时需要遵守的规则以及错误发生时如何报告，就可以进入下一个阶段了。下一节将讲述解析后的文档是什么。

2.4　XML 信息集

在 W3C 的 XML 推荐规范出版之后又提出一个新的需求，即如何以一种通用的方式来对解析之后的 XML 文档的结构进行定义。

迄今为止有许多方法可以对文档进行描述，这取决于其所使用的技术。文件对象模型(DOM)包含元素和属性这些不同类型的节点，也包含注释和处理指令的其他构建块。其他技术(后续章节会介绍)有其他的方式，新模型的部分工作就是要拿出一个通用的词汇表。新模型也旨在抽象出由不同方式创建的文件的个体差异性，比如该文档是否使用单引号或双引号来标记属性值。此外，新模型使得 XML 相关应用(例如用于转换 XML 的应用)能够工作在一个理想化的文档结构之上。这个新模型就叫做 XML 信息集，通常简称为 XML Infoset。

XMLInfoset 共包含 11 个组件。这一节简要回顾了每一个组件以及它如何进行相关的词汇表述。这些组件的官方名称为：信息项。

2.4.1　文档信息项

每一个 XML 文档都拥有一个文档信息项。这个信息项使得程序能够在访问其他信息项的同时，也能够处理它本身的一系列属性。这些属性包括在 XML 声明中定义的字符编码属性以及文档独立属性等。其他属性包括基本 URI 和文档元素等。基本 URI 是一种指向文档源文件的指针。文档元素是最外层的元素(也就是目前提到的根元素)。在旧的 DOM 术语中，文档信息项与 XML 的根节点最相似。

当需要从当前信息项导航到其他项时，可使用文档元素属性。文档元素属性是指根元素和子属性。使用子属性可以访问任何 XML 序言(根元素之前的内容)中的注释或者处理指令。

2.4.2　元素信息项

使用元素信息项可以访问任何元素相关的信息，每个元素都有一个相关的元素信息项。元素信息项有许多的属性，包括：

- 本地名称：没有任何名称空间前缀(第3章会介绍)的元素名称。例如，<pangram>和<ns:pangram>的本地名称为pangram。
- 孩子：在该元素之下的任何元素、注释、处理指令以及引用。
- 属性：该元素的所有元素的无序集合。注意属性并不是元素的孩子。
- 双亲：将该元素作为孩子的元素。如果该元素为文档元素则为文档。

2.4.3　属性信息项

使用属性信息项可以访问元素中的任意属性。其包含属性如下：

- 本地名称：与元素一样，它是一个没有任何名称空间前缀的名称。
- 归一化值：一个经过了标准空白字符归一化以及将所有引用展开化的属性值。例如将多种行尾符统一成一个换行符后的属性值。
- 属主元素：该属性信息所处的元素。

2.4.4　处理指令信息项

文档中的每一个处理指令都有一个对应的处理指令信息项。它包含 PI 的目标以及剩余文本的内容。虽然并不是强制性的，它的内容经常会被分隔开来而被当成名称/取值结合，这有点像属性。因此信息项并不会进一步去处理内容。

2.4.5　字符信息项

理论上将任何文档中出现的字符，无论它是一个字符引用还是一个 CDATA 中的字符，都会有一个相关的字符信息项。它包含如下属性：

- 字符编码：标识字符编码从 0 到#x10FFFF 的值。这些编码符合 ISO 10646 标准定义，也就是说它和 Unicode 编码是一致的。由于一些诸如 0 之类的编码不能出现的 XML 文档中，因此如果 XML 符合良好规范，你不会在文档中看到这些编码。
- 元素内容空白：这是一个标识元素中的字符是否是空白的布尔值。
- 双亲：将其作为子属性的元素。

在实际中 XML 应用程序经常将多个字符连接成字符串，因为你肯定不想每次只处理一个字符。

2.4.6　注释信息项

注释信息项是指源文档中的注释。它只有两个属性：内容，也就是注释内的文本和双亲。

2.4.7　名称空间信息项

文档内的每个元素都有一个其所属域内名称空间的信息项。名称空间的精彩内容将在第 3 章介绍。

2.4.8　文档类型声明信息项

如果一个文档包含文档类型定义，那么它就一定有一个文档类型声明信息项。它的属性包含可以使 XML 具有检索 DTD 功能的系统标识符、公共标识符等。这种信息项(也包括如下三个：意外的实体引用、未解析实体和符号)只有当文档具有文档类型定义时才适用。

2.4.9　未扩展实体引用信息项

几乎不可能遇到这些信息项，它们是一些还未被扩展的外部实体的占位符。不过大部分解析都将会扩展这些引用，它们很少见。

2.4.10　未解析实体信息项

这些同样是在 DTD 中定义的，一般也不可能遇到。

2.4.11　符号信息项

DTD 中定义的每个符号都有一种对应符号信息项。符号通过在 DTD 中声明一个引用，可将类似图片格式的非 XML 内容引入到 XML 文档中。

XML 信息集还有一个版本叫做后架构验证信息集(PSVI)。正是基于 XML 有一个相关的架构并且使用它来验证这个事实，PSVI 又引入了一些额外信息。在第 5 章中将会介绍这些额外的信息。

2.5　小结

- 良好规范 XML 的含义
- XML 文档中不能出现的字符
- 编码的含义以及 Unicode 的含义
- XML 的基本构建块，包含元素和属性
- 这些构建块的组成方式以及遵循的规则
- 某些字符的转义方式以及所选编码之外的字符的表示方式
- 空白字符的处理方式
- 错误的处理方式
- 选择元素或属性时的参考建议
- XML 信息集以及它成为 XML 文档理想化模型的方式

习　题

习题的参考答案参见附录 A。

1. 下面文件有哪些不符合良好规范的错误？

```
<xmlLibrary>
  <play publicationYear=1898>
    <title>Arms & The Man</title>
    <author>George Bernard Shaw</author>
    <play description>Comedy dealing with the futility of war
and the hypocrisy of human nature.</play description>
  <play>
  <play publicationYear=1950>
    <title>The Mousetrap</title>
    <author>Agatha Christie</author>
    <play description>A traditional whodunnit
  with an extraordinary twist.</play description>
    <play>
```

```
</xmlLibrary>
```

2. 在使用实体引用来定义你的邮箱地址时，如何做到需要修改时只需进行一处修改而不是多处。请给出完整示例。

本章要点

主　　题	要　　点
XML 必须符合良好规范的原因	如果 XML 不符合良好规范，它就不是 XML，也就不能被 XML 解析器解析
某些字符不允许出现在 XML 中	空字符(0x0)被禁止在 XML1.0 中使用，它也是 XML1.1 中唯一禁止的 Unicode 字符。XML1.0 还禁止使用除了空格字符(0x9)、换行符(0x1)和回车符(0xd)之外的所有 0x1 到 0x19 之间的字符
某些字符具有特殊含义	&和<用于引用和标签。当需要将其作为文本使用时，需要将它们转义为& 和 <
XML 文件可以有多种编码	如果文件不是使用 UTF-8 编码，则必须在 XML 声明部分进行声明
基本结构	XML 文件由元素和属性组成。XML 文档必须有一个全包括的根元素或文档元素
空白字符需要特殊处理	除非解析器认为这些空白字符有意义，否则通常将换行和回车符组成的行尾结束符替换为一个换行符，将多个空格压缩成一个
元素和属性	通常结构复杂或者重复的数据选用元素。而单独原子性的数据选用属性
XML 信息集	信息集是 XML 文档被成功解析后的一个理想化版本。为了不依赖于由于文档的不同创建方式而产生的细微差别，使用 XML 的应用应该遵循这种架构