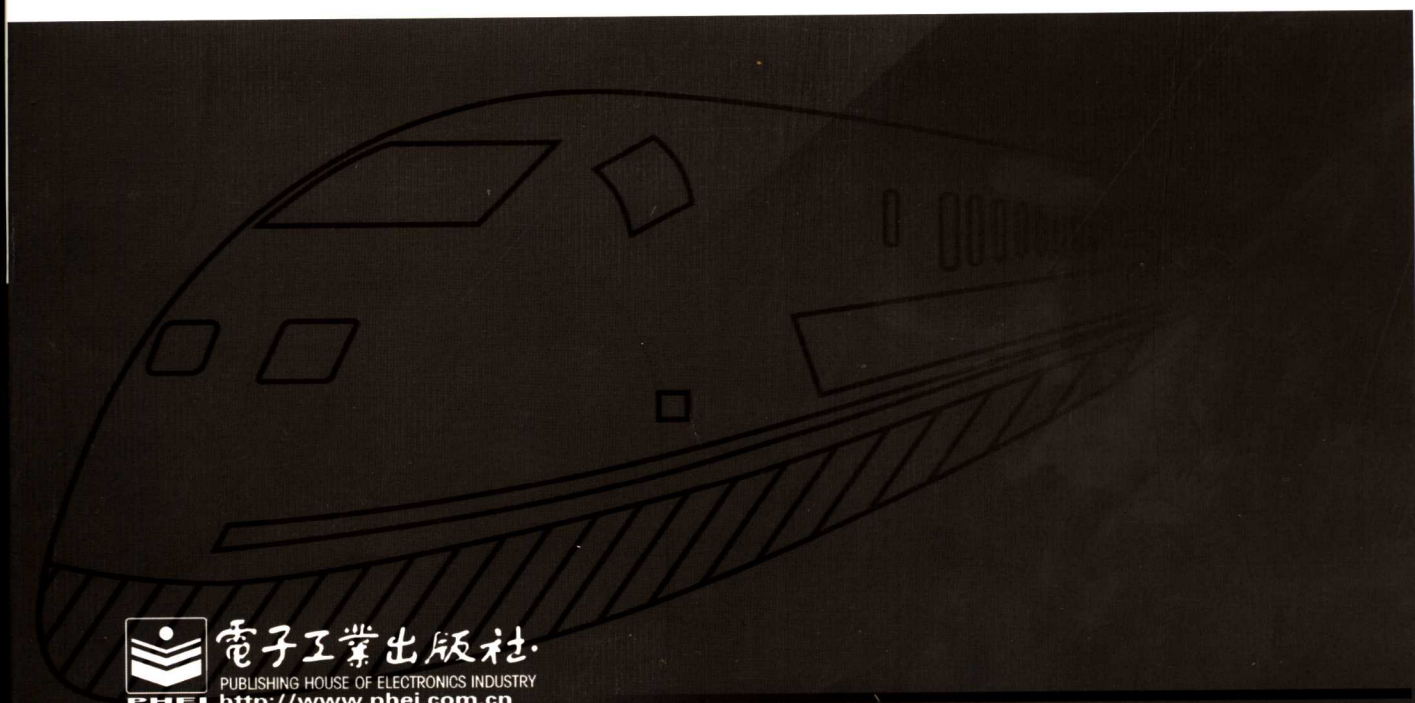
Broadview
WWW.BROADVIEW.COM.CN
PEARSON Addison Wesley
计 算 机 专 业 人 员 书 库

Effective XML: 50 Specific Ways to Improve Your XML

Effective XML: 有效使用 XML 的 50 种方法

[美] Elliotte Rusty Harold 著
徐罡 黄涛 译



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
PHEI http://www.phei.com.cn

计算机专业人员书库

Effective XML:
有效使用 XML 的 50 种方法

Effective XML:
50 Specific Ways to Improve Your XML

[美] Elliotte Rusty Harold 著
徐罡 黄涛 译

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书主要介绍如何有效地使用 XML，以创建高速运行且最小尺寸的代码。本书作者将自己多年从事 XML 教学和编写工作的经验融入其中，旨在告诉读者什么时候、为什么，以及如何有效地使用一些合适的工具。本书从 XML 的底层技术讲起，再逐步到 XML 的高层技术，主要分为四部分：XML 的句法；XML 的结构、文档组织和注释信息；运用不同语言处理 XML 技术和相关的 API，以及 XML 标记结构的本地语义；建立在 XML 文档基础上的系统所能使用的有效技术。

本书适合所有希望成为高效 XML 开发者的程序开发员、测试人员等相关技术人员阅读和参考。

Simplified Chinese edition copyright © 2004 by PEARSON EDUCATION ASIA LIMITED and Publishing House of Electronics Industry.
Effective XML: 50 Specific Ways to Improve Your XML First Edition, ISBN: 0-321-15040-6 by Elliotte Rusty Harold, Copyright © 2004.All Rights Reserved.
Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Addison Wesley, Inc..
This edition is authorized for sale only in the People's Republic of China (excluding the Special Administrative Region of Hong Kong and Macau).

本书中文简体字翻译版由电子工业出版社和 Pearson Education 培生教育出版亚洲有限公司合作出版。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。

本书封面贴有 Pearson Education 培生教育出版集团激光防伪标签，无标签者不得销售。

版权贸易合同登记号　图字：01-2004-3053

图书在版编目（CIP）数据

Effective XML：有效使用 XML 的 50 种方法 /（美）哈乐德（Harold, E. R.）著；徐罡，黄涛译. —北京：电子工业出版社，2005.3
（计算机专业人员书库）
书名原文：Effective XML: 50 Specific Ways to Improve Your XML
ISBN 7-121-00924-2

I. E…　II. ①哈…　②徐…　③黄…　III. 可扩充语言，XML－程序设计　IV. TP312

中国版本图书馆 CIP 数据核字（2005）第 009136 号

责任编辑：寇国华　孙学瑛
印　　刷：北京智力达印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编 100036
经　　销：各地新华书店
开　　本：787×1092　1/16　印张：18.5　字数：339 千字
印　　次：2005 年 3 月第 1 次印刷
定　　价：36.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

序

编程人员学习 XML 基础可能需要花费一个星期的时间，但学习如何有效地使用 XML 却可能需要花费一生的时间。虽然已经有很多书讲述如何使用 XML 的基本句法，但这本书却是目前惟一一本讲述如何有效地使用 XML 的。本书不是一本 XML 指南，它不讲述什么是标签，也不讲述如何写一个 DTD，我认为了解这些知识是学习本书的基础。本书旨在告诉你在什么时候、为什么，以及如何有效地使用这些工具（同样重要的是什么时候不应该使用它们）。

本书包含了我多年来从事 XML 教学和编写工作的经验。在过去的 5 年中，我写了多部关于XML的书并教授了多门XML课程。我逐渐发现大部分读者和学员已经熟悉了XML基础，他们知道什么是标签、如何使用 DTD 验证文档的有效性以及如何使用 XSLT 样式表转换文档。仍然保留的问题是，XML 到底是什么？我们为什么要使用它？尽管 XML 的基本句法和支撑技术是易于理解的，正如绝大部分开发人员知道 CDATA 片断是什么，但他们不能够确定我们为什么要使用它；尽管程序员知道如何给元素添加属性和子节点，但他们同样不能确定究竟该使用属性还是子节点。尽管程序员知道模式（Schema）是什么，但他们并不知道如何选择一个模式语言。

自从XML逐步成为新软件系统的基础支柱以来，新问题——不再是关于XML是什么，而是如何有效地使用它显得更为重要。我们应该采用哪些技术？避免采用哪些技术？进一步来讲，也就是说，哪些技术表面上很好，但不能进一步支撑系统的升级。我在大学里教书时，告诉学生的第一件事情就是编写程序不仅仅是通过编译产生希望得到的结果，更重要的是代码的可扩展性、易读性及可维护性。XML 既可用来产生稳健、可扩展、可维护并且易于理解的系统，同样它也可以产生不可维护、难于理解、脆弱并且封闭的代码。如同 Eric Clapton 的不朽名言——“你如何使用它也就决定了它本身”。

XML 不是编程语言，而是一种标记语言，目前许多程序员都能成功地运用它。在 XML 之前已有许多标记语言，并且在程序开发人员的团体中并不认为 XML 是最成功的标记语言。新的不熟悉的标记语言给使用者带来的问题是不能如同以往的标记语言一样立即有效地使用它。许多能够工作的系统本身不够稳健、可扩展性不强且可移植性差，这样的系统已经被抛弃。XML 承诺稳健型、可扩展性及可移植性，XML 的这一特点正符合人们的需求。使用 XML 的程序员为软件开发开辟了新的领域，进行了新的展望，完成前几年无法完成的事情。然而，也有很多 XML 的前驱者从 XML 使用的战场上负伤而归。

从 XML 发布到现在已经 5 年了，正确设计 XML 应用模式和反模式也逐步清晰，XML 社团的所有人在开拓 XML 的这 5 年中也犯了不少错误。但是，我们从错误中吸取教训，并且我们设计了一些原则来帮助后来者防止再犯同样的错误。目前正是在路上摆放警告标

记的时候，我们不敢说“小心，路上有恐龙”，至少我们可以说“路并没有你先前所看到的那么平坦，可能你并不认为如此，但是，更平滑的路却不在这条路上”。

该书分为四部分，首先从 XML 的底层技术讲起，再逐步到 XML 的高层技术。

- 第一部分覆盖 XML 的句法，XML 的句法并不能真正地影响 XML 文档的内容，但是它们影响这些 XML 文档是否更易于编辑和处理。
- 第二部分研究 XML 的结构、XML 文档组织和注释信息。
- 第三部分谈论使用不同的语言（如 C++、C#、Java、Python 或 Perl）处理 XML 的技术和相关的 API，以及 XML 标记结构的本地语义。
- 第四部分探究建立在 XML 文档基础上的系统所能使用的有效技术，而不研究单个 XML 文档。

本书虽然采用这种组织结构，但并不限定读者的阅读顺序，读者可以根据自己的情况选择任何一章开始阅读。当然，希望读者都能够阅读全书，至少应该先阅读简介，在简介中定义了全书使用、易于混淆和误用的术语。然后，读者可再选择感兴趣的题目阅读，并可以交叉参考全书内容。

我希望本书是一个开始，而不是结束。本书只是处于 XML 发展的早期阶段，还有很多问题和方法有待于发现和创造。你可能开发了你自己的最好实践，而本书并没有提到，我也很愿意知道这些最好实践。你也可能对其中的某些观点有异议，同样我也很愿意知道这些异议。我们也通过 xml-dev 邮件列表讨论了本书中的观点，而且这种讨论仍然在继续。建议感兴趣的读者能够订阅和参加到 xml-dev 邮件列表讨论中——http://lists.xml.org/。另一方面，如果有错误（如 ID 属性丢失了一个引号，误拼了“cat”单词），请写信给我——elharo@metalab.unc.edu。我的网页 http://www.cafeconleche.org/books/effectivexml/列举了本书的勘误表和更新。

最后，我希望本书能够使你更有效，并且更愉快地使用 XML。

此为试读，需要完整PDF请访问：www.ertongbook.com

简　介

与本书序言所声明的一样，本书既不是一本介绍 XML 的书，也不是一本 XML 指南。使用本书之前，你至少应该熟悉 XML 文档的基本结构，知道如何在你所使用的程序语言中使用解析器来读一个 XML 文档，并且能够使用样式表等。

然而，在过去的几年里，我们发现 XML 中的特定名词和短语有不同的含义，而且经常得不到统一使用。由此造成的名词和短语的歧义性，有时候会产生混淆，并导致严重的过程失败。XML 书的作者及 XML 培训人员不够严谨地使用 XML 词汇，如元素及标签等，是造成这种现象的原因之一（当然，也包括本书作者）。名词和短语的混淆同样存在于 W3C 的 XML 工作组中，这个工作组对名词和短语的使用彼此不一致，甚至在一个规范中同一名词的含义不同。我们有必要在展开本书的讲述之前花一些时间仔细地定义要用到的术语，确保术语的意义与实际的使用相一致，当然也不排除对术语和短语的其他观点和解释。

为此，列举了以下经常会混淆的术语。

- 元素与标签。
- 属性与属性值。
- 实体与实体引用。
- 实体引用与字符引用。
- 子、子元素与内容。
- 文本、字符数据与标记语言。
- 命名空间、命名空间命名与命名空间 URI。
- XML 文档与 XML 文件。
- XML 应用与 XML 软件。
- 合式与有效。
- DTD 与 DOCTYPE。
- XML 声明与处理指令。
- 字符集与字符编码。
- URI、URI 引用与 IRI。
- 模式（Schema）与 W3C XML 模式语言。

这些术语的混淆经常引起对 API 和 XML 工具如何运行的误解，例如，如果你将字符引用与实体引用相等同，则可能非常困惑为什么 SAX 解析器在字符引用时不调用 StartEntity 方法。当你使用邮件向其他人询问一个问题时，不能够使其他人明白你的问题，也有可能导致你花费几个小时来编写一个测试案例来修补一个程序上的错误。

当你能够准确地说明你的问题时，这些表面上困难的问题立即变得很简单了。因此，

我们需要仔细地定义术语，以避免问题的混淆带来不必要的困难。

元素与标签

元素不等同于标签，同样，标签也不等同于元素。元素以开始标签开始，包含特定的内容，然后以结束标签结束。标签定义了元素的界限，它们是元素的一部分，但其本身不是元素，如同三明治不仅仅包含一片面包一样。元素是整个三明治，包括面包、芥末、蛋黄酱、肉和奶酪，标签就像面包片。例如，<Headline>是一个开始标签，</Headline>是一个结束标签。<Headline>Crowd Hears Beth Giggle</Headline>是一个完整的元素，元素可以嵌套其他元素，但标签不能包含其他标签。

单个空元素标签可以表示一个完整的元素是元素退化的结果。例如，<Headline/>既是 headline 的标签，也是 headline 元素，这种表示只是元素表示的一种特例。从语义上来讲，<Headline/>空标签完全等同于<Headline></Headline>两个标签，绝大多数处理 XML API 并不区分这两种形式，其效果是相同的。

总之，嵌套的元素构成 XML 文档的结构，标签确定了各个元素的界限。

属性与属性值

属性用来描述元素的特征，它有一个名称和一个值，包含在元素开始标签中（属性可以在 DTD 中设定默认值）。例如，考虑下面这个元素：

```
<Headline page="10">Crowd Hears Beth Giggle</Headline>
```

headline 元素有一个 page 属性，其值是 10。headline 元素的 page 属性包括一个名称和一个值，属性名称是 page，其值是简单字符串 10。属性值使用单引号或双引号加以标注，单引号或双引号是等价的。下面这个元素与前一个等价：

```
<Headline page='10'>Crowd Hears Beth Giggle</Headline>
```

一个元素可以有多个属性，属性的排列不区分次序，以下两个元素同样是等价的：

```
<Headline id="A3" page="10">Crowd Hears Beth Giggle</Headline>
<Headline page="10" id="A3">Crowd Hears Beth Giggle</Headline>
```

因此，解析器也不区分属性次序。如果你想关注属性的次序，那么可以使用子元素代替属性，如下所示：

```
<Headline>
  <id>A3</id> <page>10</page>
  Crowd Hears Beth Giggle
</Headline>
```

这种方法与术语混淆无关，而是一种探索使用属性和子元素来表示相同事物的技术。

元素和属性的差别很大，次序只是元素与属性的差别之一，其他重要的差别还包括类型、合式和拥有子结构的能力。

实体与实体引用

实体是一个存储单元，包含部分 XML 文档。实体可以是网络服务器或其他应用系统返回的一个文件、一个数据库记录、一个对象或一个字节流，它可以包含一个完整的 XML 文档或者只是一些元素或声明。

实体引用指向这些实体。有通用实体引用和参数实体引用两类实体引用。通用实体引用开始于&，例如&或&chapter1;。可以在 DTD 中定义 chapter1 实体如下：

```
<!ENTITY SYSTEM chapter1 "http://www.example.com/chapter1.xml">
```

在 XML 文档中，可以如下所示引用该实体：

```
<book>
  &chapter1;
...
</book>
```

&chapter1;只是一个实体引用，http://www.example.com/chapter1.xml 文档中的实际内容才是这个实体。它们彼此关联，但并不是相同的事物。

参数实体和参数实体引用也遵循同样的模式，差别是参数引用包含 DTD 片断，而不是实例文档片断。参数引用开始于%，而不是&，实体引用代表和指向实际的实体。

XML API 处理实体引用时，可以报告实体或者报告实体引用，或者两者都报告，也有的 API 两者都不报告。一些 XML API，如 XOM，只是简单地用相应的实际实体替换实体引用，而不会报告它们对实体引用的操作。其他如 JDOM，只是报告没有被替换的实体，而不报告已被替换的实体。另外还有其他的，如 DOM 和 SAX 依赖于调用参数和解析器的功能可以报告实体和实体引用两者。但有 5 种预定义的实体引用（&，<，>，"和'）无论是哪种情况都不报告。

实体引用与字符引用

不是每一个以&开头的引用都是实体引用。实体引用只用于命名实体，包括 5 个预定义实体引用和在 DTD 中以 ENTITY 声明定义的实体。如<和上例中的&chapter1。

相比而言，字符引用使用十六进制或十进制 Unicode 值来引用特定的字符，而不使用名称。字符引用每次引用一个字符，而不是一组字符。例如， 是一个十六位字符引用，引用空格字符。 是一个十进制字符引用，引用空格字符。XHTML 的 是一个实体引用，引用空格字符。

即使报告实体引用的 XML API 也很少报告字符引用，解析器只是替换被引用的字符，

并与整个文本相合并。程序代码并不关注字符是以字符引用方式出现，还是以实体引用方式出现。

子、子元素与内容

元素的内容位于元素的开始标签和结束标签之间，例如，DocBook 中的 para 元素。

```
<para>
  As far as we know, the Fibonacci series was first discovered
  by Leonardo of Pisa around 1200 C.E. Leonardo was trying to
  answer the question, <!-- Scritti di Leonardo Piasano. Rome:
  Baldassarre, 1857. Volume I, pages 283 - 284.Fibonacci,
  Leonardo. --> <quote lang="la"><foreignphrase>Quot paria
  coniculorum in uno anno ex uno pario germinatur?</foreign
  phrase></quote>, or, in English, <quote>How many pairs of
  rabbits are born in one year from one pair?</quote> To solve
  Leonardo&rsquo;s problem, first estimate that rabbits have
  a one month gestation period, and can first mate at the age
  of one month, so that each doe has its first litter at two
  months. Then make the simplifying assumption that each litter
  consists of exactly one male and one female.
</para>
```

para 元素的内容是文本，它也包含空格、注释、其他文本、一个 quote 子元素、纯文本、另一个 quote 子元素、纯文本、’实体引用，然后以其他文本结束。所有这些包括子元素的内容，如 quote 构成 para 元素的内容。

para 元素有两个子元素，都命名为 quote。para 元素不仅包含子元素，还包含一个注释、一些字符数据和一个实体引用。虽然不同的 API 和系统对这些子元素、注释、字符数据和实体引用的表示方法和个数多少有所不同，但这些都可以看做 para 元素的子。在极端情况下，每个分离的字符都可以作为一个独立的子。另一种极端情况为在所有实体引用被替换后，每个文本节点包含最大数量的连续的文本块，这样 para 元素只有 4 个文本子节点。

另一方面，虽然 foreignphrase 元素和 quote 元素的内容是 para 元素的后代，但是它们不是 para 元素的子。

混淆子和子元素的普遍原因是忘记了组合内容的实际可能性，即使文档采用记录结构，子与子元素的差别也是重要的。例如，考虑下面的 presentation 元素。

```
<presentation>
  <title>DOM</title>
  <date>Thursday, November 21, 2002</date>
  <host>Software Development 2002 East</host>
  <copyright>2000-2002 Elliotte Rusty Harold</copyright>
  <last_modified>November 26, 2002</last_modified>
  <author_name>Elliotte Rusty Harold</author_name>
```

```
  <author_url>http://www.elharo.com/</author_url>
  <author_email>elharo@metalab.unc.edu</author_email>
  <abstract>Elliotte Rusty Harold's DOM tutorial</abstract>
</presentation>
```

presentation 元素看起来好像只有子元素。但统计子节点时，你必须也统计空格，至少有 10 个空格子节点。此外，title、date、host 和其他元素又如何？它们每一个都有一个子节点，但不包含子元素。底线：子不仅仅只是元素。

文本、字符数据与标记

XML 文档由文本构成，可以分为标记和字符数据两个互不交叉的集合。标记包括所有的标签、注释、处理指令、实体引用、字符引用、CDATA 限定符、XML 声明、文本声明、文档类型声明，以及除根元素之外的空白空间。除上述之外，其余部分都是字符数据。例如，DocBook 中的 para 元素，标记采用黑体书写，字符数据采用平体书写。

```
<para>
  As far as we know, the Fibonacci series was first discovered
  by Leonardo of Pisa around 1200 C.E. Leonardo was trying to
  answer the question, <!-- Scritti di Leonardo Piasano. Rome:
  Baldassarre, 1857. Volume I, pages 283 - 284.Fibonacci,
  Leonardo. --> <quote lang="la"><foreignphrase>Quot paria
  coniculorum in uno anno ex uno pario germinatur?</foreign
  phrase></quote>, or, in English, <quote>How many pairs of
  rabbits are born in one year from one pair?</quote> To solve
  Leonardo&rsquo;s problem, first estimate that rabbits have
  a one month gestation period, and can first mate at the age
  of one month, so that each doe has its first litter at two
  months. Then make the simplifying assumption that each litter
  consists of exactly one male and one female.
</para>
```

DocBook 中 para 元素的标记包括<para>和</para>、<quote>和</quote>、<foreignphrase>和</foreignphrase>标签、注释和实体引用’，其余部分均为字符数据。

其余部分有时也称为解析字符数据或 *PCDATA*，在 DTD 中使用 PCDATA 关键字声明元素，如下面的 interfacename 元素的声明：

```
<!ELEMENT interfacename (#PCDATA)>
```

但是，这样把解析字符数据与字符数据等同使用是不够精确的。通常来讲，解析字符数据是指解析器已经替换了实体引用和字符引用后的字符数据，其中包含字符数据和标记语。

命名空间、命名空间命名与命名空间 URI

XML 命名空间是名称的集合，例如，在 XHTML 中定义的所有元素名称（html、head、title、body、p、div、table 和 hl 等）构成 XHTML 的命名空间，SVG 命名空间是在 SVG 中使用的元素名称的集合（svg、rect、polygon 和 polyline 等）。不是所有预先定义的元素名称都属于命名空间，只有局部预先定义的元素名称才属于命名空间。

通过 URI 识别的每个引用命名空间叫做命名空间名称，例如，XHTML 的命名空间名称是 http://www.w3.org/1999/xhtml。SVG 的命名空间是 http://www.w3.org/2000/svg。命名空间名称确定了命名空间，但不等同于命名空间。

命名空间名称假定为一个 URI 引用。当然，如果命名空间命名不是一个 URI 引用，则也不是一个技术上的错误。例如，命名空间命名可以包含{字符或希腊字符 λ，这两个字符在 URI 中都是非法的。在实际使用中，大多数命名空间命名都是合法的 URI 引用，命名空间命名也称做命名空间 URI。实际上，命名空间命名是命名空间 URI 引用，但是目前绝大多数开发人员并不区分命名空间 URI 和命名空间 URI 引用。

XML 文档与 XML 文件

在技术上来讲，XML 文档是遵循 XML 1.0 规范的 Unicode 字符序列，XML 文档可以以文件的方式或以其他方式存储。例如 XML 文档可以采用数据库记录的方式存储，或者放在程序的内存中，或者从网络数据流中读取。再如打印在一本书中，画在广告牌中，甚至在公共汽车的广告窗中。因此，XML 文档不必是一个文件。如果 XML 文档采用文件来存放，则可以存放在一个单一的文件中，也可分散在多个文件中使用实体引用加以链接。甚至多个 XML 文档可以存放在一个文件中，但这种方式不经常使用。

此外，在分析 XML 文档时，区分 XML 文档与 DTD 及其他形式的模式是必要的。遵循特定模式的 XML 文档称为实例文档，也就是说，XML 文档是特定模式的实例。

XML 应用与 XML 软件

XML 应用是模式、规范和其他规则集定义的 XML 文档的一个类，例如，SVG（可扩展向量图，Scalable Vector Graphics）、XHTML、MathML、GedML、XSL 格式对象和 DocBook 都是 XML 应用。你自己创造的用来分类喜剧书的 XML 文档也是一个 XML 应用，即使它不遵循某个 DTD、模式（Schema）和规范。即使应用软件处理 XML 应用，如 XMLSPY 编辑器、Mozilla Web 浏览器或者 XEP XSL-FO 转换器，XML 应用也不是该应用软件的一部分。

合式与有效

合式与有效是评价 XML 文档优劣程度的两个级别，合式是指强制地遵循句法约束，

此为试读，需要完整PDF请访问：www.ertongbook.com

有效是指可选择的结构和语义约束。在英文中，使用合式来描述文档的正确性已是一种趋势，XML 与英文不同的是它有更多的特定含义。文档是正确的，但它仍然是不可处理的。

合式性是 XML 文档的基本要求，合式包括不同的句法约束，如每个开始标签必须有与其相匹配的结束标签，每个 XML 文档必须有且只能有一个根元素。如果文档是非合式的，那么它就不是一个 XML 文档。解析器遇到非合式的 XML 文档将报告出错，并停止解析。解析器并不试图猜测文档作者的意图，它们也不能够修正错误，只是抛弃该文档。

有效是比合式更强的约束，对处理 XML 文档来说不是必需的。有效性决定不同的元素和属性应该出现在不同的位置，有效验证文档是否遵循文档类型定义（DTD）和文档类型声明（DOCTYPE）定义的约束。即使文档不遵循这些约束，在某些情况下仍然是可处理的。客户应用程序决定是否及如何抛弃非法性文档，而不是由解析器决定。

文档有效通常是指文档遵循模式（Schema）有效，而不是指遵循 DTD。特别是当同时使用 DTD 和模式验证某些文档时，这一点易于混淆。因此引入模式合法（schema-valid）。如果使用 DTD 合法，由客户应用程序决定是否及如何处理模式合法文档。模式合法，则解析器通知客户应用程序文档模式非法。但是可以继续解析，客户应用程序决定是否接受这个模式非法文档。

DTD 与 DOCTYPE

XML 文档类型定义是描述有效文档的 ElEMENT、ATTLIST、ENTITY 和 NOTATION 的集合。文档类型声明位于 XML 文档的序言中，它包含文档类型定义或者指向文档的文档类型定义。文档类型定义和文档类型声明是两个密切关联的概念，但不是相同的事物。DTD 缩写只是指文档类型定义，而不是文档类型声明。缩写形式 DOCTYPE 只是指文档类型声明，而不是文档类型定义。

例如，如下文档类型声明：

```
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
                         "docbook/docbookx.dtd" >
```

这个文档类型指向具有-//OASIS//DTD DocBook XML V4.1.2//EN 标识的 DTD，该标识位于 docbook/docbookx.dtd 相对 URL 地址中。

下面是另一个文档类型声明：

```
<!DOCTYPE book SYSTEM "http://www.example.com/docbookx.dtd">
```

该文档类型指向 DTD，该 DTD 在绝对地址 URL http://www.example.com/docbookx.dtd。

也有的文档类型声明在 XML 文档中包含了整个 DTD，并用中括弧加以限定。

```
<!DOCTYPE book [
  <!ELEMENT book (title, chapter+)>
  <!ELEMENT chapter (title, paragraph+)>
  <!ELEMENT title (#PCDATA)>
```

```
  <!ELEMENT paragraph (#PCDATA)>
]>
```

最后，这个例子的文档类型声明既指向外部的 DTD，也包含一个内部的 DTD 子集，完整的 DTD 由外部和内部 DTD 子集混合形成。

```
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
                         "docbook/docbookx.dtd" [
 <!- add XIncludes ->
 <!ENTITY % local.para.char.mix " | xinclude:include">
 <!ELEMENT xinclude:include EMPTY>
 <!ATTLIST xinclude:include
    xmlns:xinclude CDATA #FIXED "http://www.w3.org/2001/XInclude"
    href           CDATA #REQUIRED
    parse          (text | xml) "xml"
 >
]>
```

DTD 可以由外部、内部的 DTD 或者由两者混合而成。DTD 与文档类型声明不是相同的事物，文档类型声明指定根元素，而 DTD 不需要指定根元素。DTD 声明内容模型和元素的属性列表，文档类型声明不需要内容模型和元素的属性列表。绝大多数 API 提供对文档类型声明内容的访问，而不提供对文档类型定义内容的访问。

XML 声明与处理指令

XML 声明是 XML 规范中最不应该混淆的构件之一，由于多种技术原因，该构件位于 XML 文档的最顶部，事实上它不是处理指令。

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

XML 声明看起来像一个处理指令，但它不是处理指令，它只是一个 XML 声明。处理指令的处理对象中禁止出现 xml、XML、Xml 或者任何 XML 的不同大小写组合。

有的 XML API 允许客户程序访问 XML 声明中的信息，但是如果 API 支持客户程序访问 XML 声明中的信息，API 不能采用与报告处理指令相同的机制。例如，在 SAX 2.1 中，Locator2 接口提供某些 XML 声明中的信息，该解析器不能通过调用 ContentHandler 中的 processingInstruction 方法查看 XML 声明中的信息。

字符集与字符编码

XML 是基于 Unicode 字符集，字符集是分配了编码的字符集合。Unicode 4.0 定义了超过 9 万个独立字符，该字符集中的每个字符映射为一个数字，如 64、812 或 87 000。这些数字不是整型、短整型、字节、长整型或者其他任何数字类型，而只是简单的数字。Shift-JIS、Latin-1 等其他字符集是分配了不同编码的不同字符集合，并与 Unicode 字符集

相重叠。

也就是说，其他字符集与 Unicode 字符集都为相同的字符分配相同的数字。

字符编码是以某种特定的方式将字符的编码表示为字节类型，UTF-8、UTF-16、UCS-2、UCS-4、UTF-32 及其他不常使用的方法都是 Unicode 的编码方法。不同的编码方法使用不同字节序列和字节数解码相同的编码。可以采用 big-endian 形式或 little-endian 形式的数据，可以采用非二补角机制表示，可以采用两个字节或 4 个字节表示一个字符，甚至可以采用不同的字节数表示不同的字符。

改变所采用的字符集也意味着改变字符集所包含的字符范围，例如，ISO-8859-7 字符集包括希腊字母，而 ISO-8859-1 字符集则不包括改变字符编码，并不改变字符集所包含的字符范围，而仅仅是改变了每个字符的编码方法。

XML 解析器在向客户应用程序输出之前将其他字符集的字符转换为 Unicode 字符，其结果是，XML 解析器将其他字符集作为 Unicode 子字符集的不同编码处理。实际上，XML 总是采用 Unicode 字符集，不允许改变字符集，只允许选择不同的编码方法。

URI、URI 引用与 IRI

一个 URI 等同于一个资源，一个 URI 引用等同于一个资源的一部分。URI 引用可以包含部分标识符，使用井号(#)与 URI 分离，而 URI 不包含井号(#)。例如，http://www.w3.org/TR/REC-xml-names/是一个 URI，而 http://www.w3.org/TR/REC-xml-names/#Philosophy 是一个 URI 引用。

大部分与 XML 相关的规定都采用 URI 引用的方式定义，而不采用 URI 方式来定义，如 XML 中的命名空间。例如，在 W3C XML Schema 语言中，简单类型 xsd:anyURI 是指元素的类型是 URI 引用。通常在使用中，人们并不区分两者。然而，区分两者的异同是重要的。例如，在文档类型声明中的系统标识符是 URI，而不是 URI 引用。

说明 有的观点认为相对 URI 是 URI 引用，不是真正的 URI，XML 规范的作者好像认同此观点。但是，URI 规范（RFC2396）并不支持此观点。规范清楚地阐述了相对 URI 与相对 URI 引用，这种观点可能希望所有的 URI 都是绝对 URI。如果是这种情况，则这种观点是失败的。URI 与 URI 引用的区别在于后者允许部分引用，而前者不允许。

目前，IETF 正从事国际化资源标识（Internationalized Resource Identifiers，IRIs）的研究。IRIs 与 URI 相似，不同的是 IRIs 是非 ASCII 字符，如ζ和é，而这些在 URI 中是不允许的。IRIs 规范正在制定中，一些 XML 规范已经开始引用 IRIs 规范。例如，XLink href 属性实际上已经包含了 IRI，而不只是 URI。

模式与 W3C XML Schema 语言

模式是一个用来规定文档的布置及所允许的内容的文档通用术语，它最早出现在数据库研究领域。对于 XML 来说，有多种不同的模式语言存在。每种语言都有各自的优缺点，如 DTD、RELAX NG、Schematron 及 W3C XML Schema 语言。

XML 的开发人员更喜欢使用模式或 XML Schema 引用 W3C XML Schema 语言。有一点要说明的是，W3C XML Schema 语言不是惟一的这样的语言，同时也不是最简单、最强大，并且设计最好的，它只是一种已对外发布的语言，有其缺点和优点。我们也不应忽略其他模式语言，更不应该用模式只指定引用 W3C XML Schema 语言。

不幸的是，W3C 并没有给它的模式语言一个更简洁的称呼。因此，为了避免烦琐，我也偶尔屈服于诱惑使用模式（Schema）引用 W3C XML Schema 语言，但我也只是在 W3C XML Schema 语言范围内使用模式引用 W3C XML Schema 语言。在 W3C XML Schema 语言范围之外，要明确模式的引用。我们也可以在模式语言中认为模式不仅是名词，而且还是代词，在 W3C 中仅仅是个名词。

每个词都有多种不同的解释，XML 是一种精确定义的语言，它的每个词都需要有确定的意义。即便是这样，在实际使用中，在 XML 的某些方面也经常易于混淆。由于混淆而使问题更难于解决是没有意义的，使用正确的词表示正确的概念能够简化很多不必要的复杂问题，并且节省精力来解决真正复杂的问题。

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com