

第4章 数据建模与XML

XML应用程序的成功与否取决于你真正使用的 XML文档的设计的优劣：这些文档不仅要携带目前人们进行交流所需的信息，而且要足够灵活以适应未来的需要。本章介绍了在你设计XML文档时需要考虑的若干因素。

我们将从以下三方面讨论设计过程：

- 信息建模——理解文档所携带的信息的结构和含义。
- 文档设计——将你的信息模型转换为一组规则（或模式），以便创建真正的文档。
- 模式表示法——一种记录你的文档设计的技术，它使得文档设计能够被处理软件和人类用户所理解。

4.1 信息建模

当我们在前一章介绍图书目录问题时，曾经进行了一些简单的信息建模工作。在本章，我们将详细地讨论建模，并通过一些实际的例子说明它在设计基于XML的信息系统中的作用。

信息建模的第一条规则是将精力集中于“现实世界”，而不是技术。信息模型是对一个组织机构中所用到的信息的描述，它独立于任何IT系统。

- 如何将它结构化？
- 它有什么含义？
- 谁“拥有”它，谁负责保证它的时效性和质量？
- 它来自何方，最终会产生什么样的结果？

因此根据定义，对于信息建模的讨论不能过度针对于XML。在本章稍后，我们将专门阐述有关XML的问题。

那么，在本书中有必要讨论建模吗？当然，因为如果你参加与XML有关的项目，理解信息建模是非常关键的。另外，虽然信息建模是独立于技术的，但是在介绍关系数据库设计的书中常常会讨论建模问题，而且某些方面的论述往往有些偏颇。在这本专门介绍XML的书中，我们能够从另一个角度或许是更加全面的角度看待这个问题。

信息建模为什么如此重要？因为如果没有模型，就谈不上信息，有的只是数据。信息模型定义了数据的含义。事实上，信息模型是必不可少的；唯一可以选择的是采用大家一致同意的共享信息模型，还是冒险允许每个人根据自己的想法采用不同的信息模型（它不可避免地会造成相互之间的误解）。如果使用共享信息模型，将带来无穷的机会。来自Microsoft的Adam Bosworth、Andrew Layman和Michael Rys在一篇文章（<http://biztalk.org/Resources/canonical.asp>）中曾有以下叙述：

.....我们早已开始着手实现一个曾经梦寐以求的目标：无论数据产生者位于何处，任何

数据消费者都能够通过某种工具与他们交互，并且这种通信是基于数据的含义，而不是数据偶然的表现形式。

在本书中，我们将侧重于通用的原则，而非某些特殊的形式。如果你希望了解建模的方法，例如：UML (Unified Modeling Language, 统一建模语言)，有大量这方面的书籍可以参考，另外，<http://www.rational.com/>上也提供了完整的规范。在信息建模中，以下两个目标常常会产生冲突：

- 获得绝对精确的定义。
- 有效地与用户通信。

正式的方法通常在技术精确度方面有着更明显的优势，但是它们却不易于被外行人士所理解，因此，我们试图借助术语和简化的图表表示法从一定程度上弥补这一不足。

有关术语的警告：在实际的信息建模中，实体和属性等术语的含义与它们在 XML规范中的含义有着天壤之别。特别是我们将信息模型中的事物称为对象 (object) 而不是实体 (entity)，将它们的特征称为特性 (property) 而不是通常的属性 (attribute)。这样就能够避免XML技术术语与XML概念之间的混淆。

4.1.1 静态模型和动态模型

信息模型主要分为以下两种类型：静态模型和动态模型。

静态模型侧重于描述系统的状态。它基本上是由以下类型的语句构成的：“一个客户应该有一个或多个帐号”，“一章可以有零个或多个脚注”，“每本书都有一个ISBN”。它们描述了系统中对象的类型、特性以及对象之间的关系。当然，除了描述之外，它们还定义了这些对象一致同意的名称，例如：customer、account、chapter和footnote。对象一致公认的名称是成功的一半，这也是为何XML信息模型有时被称为词汇表。

动态模型侧重于描述对信息的处理，例如：处理模型和工作流图表，数据流模型，以及对象生存周期历史。动态模型包含以下类型的语句：“病理学部门要将检查结果发送给负责为病人提供咨询的顾问”。动态模型描述了信息的交换：出于特定的目的将数据从一个地方发送到另一个地方。

一般而言，静态模型与数据库的设计直接相关，信息被长期保存，并可以用于多种用途；而动态模型是直接与信息的设计相关的，信息存在的时间非常短暂，而且用途专一。

当然，XML能够用来表示系统中两种类型的数据——文档和消息。但是，任何系统设计都需要同时考虑静态模型和动态模型，而且这两个模型是同等重要的。有些人宁愿从静态或动态模型开始，但是在完成一个模型之后才开始另一个模型是不恰当的。我习惯于从静态模型开始，因为它能够建立基本的术语，而且静态信息模型可能是任何信息系统中最持久的——即使在二十年之后，当所有代码都重写若干次之后，它仍然保持不变。

当然，事实上长期的静态信息和短暂的消息之间的界限比较模糊：静态信息模型中的许多对象实际上是事件（例如，产品销售），许多从短暂通信（例如，客户的抱怨）开始的文档将使用很长时间。你可以自行决定将这些对象模型化为静态的还是动态的，它们主要取决于具体的情况。

4.1.2 文档和数据

从传统角度讲，文档和数据是毫不相干的。商业数据处理的对象是高度结构化和形式化的信息——管理企业的分类帐簿。其目的是通过整理数据使得处理过程能够自动化，并汇总信息，以便高层管理人员判断利润额或亏损额。相反，文档发布涉及到如何创建和产生供人们阅读的文本，模仿并增强印刷页面作为人类沟通工具的效率。

因此从某种程度讲，数据领域的关键是分析和整理，将数据处理过程自动化，使得系统更加统一；而在文档领域，核心问题是提供灵活性，使得信息的作者和读者能够尽可能以创造性的方式进行交流。

Web促进了这两个领域的结合。XML可能是在这两方面表现同等出色的第一个技术例子。这种汇合的趋势对于这两个领域来说都是有益的，因为信息系统设计者一直在寻找增强系统灵活性的方法，而文档设计者也一直在探索记录更多结构的方法。如今他们共同的目标是利用组织机构的集体知识进行“知识管理”，其中知识的含义非常广泛：从高度结构化和有组织的数据到特别的非正式数据。目前，还有许多跨越传统分界线的“多媒体”应用程序，例如：我们将在本章使用的产生旅游手册的例子。

说明这两个领域逐渐融合的另一例子是构成信息处理系统一部分的事务文档，例如：订单、发票，以及提供医院预约、收费清单和事故报告的信件。

但是这两个领域的传统仍然是泾渭分明的，对于原来分别从事数据库领域数据建模和文档设计的两个人，你很容易发现他们之间的差别。我们将尝试一条中立的道路——借鉴两个领域的优势。

4.1.3 从何处开始

有一个古老的关于伦敦旅行者的故事，他向一个人询问如何到达 Trafalgar 广场，那个人说：“如果我是你，你就不会从这里开始。”对于信息建模也存在着同样的情况：实际上，通常你无法选择从何处开始，因此问“从何处开始”这个问题是多此一举。

事实上，你总是从目前所在的地方开始，因此第一步是要确定你在哪里。定义系统的范围和目的了吗？是否已经有了格式正规的商业过程集合，或者要开发一个新的集合？确定系统体系结构了吗？你能对结果有多大影响；你所授权的范围有多大？谁将负责作决定；是否有人持反对意见？

在数据处理和文档设计领域，传统的方法是从现有的文书工作开始。找到相关的文档，通过概括和抽象确定它们的结构，与用户讨论文档中信息的来源，如何将信息从一个文档传递到另一个文档，以及如何使用这些信息，然后将所有内容组合在一起，形成一个数据模型。

通常，这种方法在目前来说不是非常出色，因为人们不想创建单纯重复现有处理方式的系统。正如电子商务系统不必精确地反映传统的购买过程，在线旅游手册也不必是印刷的假期指南的准确重复。因此，你应该对要获得的商业目标以及真正能够激发用户的因素有更高层次的透彻的理解。你必须知道存在哪些信息，以及它们为什么存在，并且提出更富创造性和想象力的方法，以实现商业目标。当然，你是否能够做到这种程度取决于你被分配的工作，以及你对

其他人的影响力。

4.1.4 静态信息模型

在本节中，我们将逐步介绍如何定义静态信息模型。它分为以下四个步骤：

- 步骤1——标识事物，并对它们进行命名和定义。
- 步骤2——将事物组织为类层次。
- 步骤3——定义关系、元组数和约束。
- 步骤4——通过添加特性，将与对象相关的值详细化。

1. 步骤1：命名事物

开始信息建模的最好方法是为系统中的事物设置名称。其中“事物”通常是指实体、对象、类或数据元素；这一步并不十分重要。我们将以上“事物”称为对象类型。

首先要列出与系统相关的所有事物，例如：客户、帐号、假期、旅馆、度假胜地、国家、预定或付款。有人建议首先对系统进行文本描述，然后从中选出所有名词。不管采用哪种方法，这一步都不太困难；通常在较短的时间内就能够完成。

下一步是产生对象类型的定义，它可能需要耗费更多的时间。以“假期”一词为例，它的定义要保证当你看到假期时能够识别它，而且对于某个事物是否是假期没有争论的余地。比如你可以问以下问题：

- 如果两个人一起旅游，应该算作一个假期还是两个假期？
- 未出售的假期是否仍然能够算作假期，或者称为其他事物？
- 如果客户预定了一个假期，后来调整到其他日期，这是否仍然算作同一假期？

在建立这些定义时，我发现有两类问题总是很有意义的。第一个是以下形式的问题：

- X是一个假期吗？

通过测试一些不太确定的例子，能够明确概念的边界。第二个问题是：

- X和Y是相同的假期，还是不同的假期？

在此，我们不再讨论X和Y是否是真正的假期，而是寻找能够明确地区分特定假期的规则。对于无形的对象，例如：飞行、频道、服务或广告活动，这类问题尤其重要。

你也许会发现不同的人对于这些问题的答案截然不同。有些人或许对什么是假期有完全不同的看法，例如，他们可能认为2001年的复活节是一个假期。这就体现出信息建模的价值：它能够排除可能的误解。在我曾经工作过的一个机构中，由于一个部门认为零售商意味着位于特定位置的一个商店，而另一个部门认为是一家公司，它可能拥有多个商店，结果造成了数据的丢失。如果问这样一个问题：“达拉斯的QuickFood与匹兹堡的QuickFood是相同的零售商吗？”，就能够暴露这种误解。

经过上述过程，你有可能得到一长串的对象类型列表，其中某些可能有较长的名字，例如：holiday-inventory-item和party-holiday-reservation。如果有可能的话，选择业界人士能够正确理解和解释的名称，因为他们不是总有时间查看你精心编写的定义。

在本阶段，由于我们侧重于事物的标识，因此除了命名对象类型之外，如何识别每个实例也是值得考虑的。如何标识单独的假期？目前可能已经存在用于标识实例的代码，你需要了解这些代码；或者你不得不重新发明一个新代码；或者你可以使用某些特性的组合——例如：假

期可能由客户编号和起始日期的组合来标识。在本阶段中，常常能够发现现有编码机制中存在的问题，例如：公司旅游部门可能用一个客户编码标识来自特定公司的所有职员，而客户服务部门会给每个参加旅游的职员分配一个独立的客户编号。

因此在本步骤的结尾，我们将得到一个对象类型列表，其中的名称和定义都获得了一致的同意。

2. 分类

分类是生物学中的术语，它是指分类系统；在信息建模中，我们也将它称为类型层次（有人愿意将它称为本体论）。既然已经列出了所有对象类型，并给它们命名，现在，我们要将它们组织到一个层次化的分类体系中。当我们编写对象类型的定义时，通常会形成这些层次关系。以下定义都是很常见的，例如：

- 当天往返假期是一个不包括预订食宿的假期。
- 有三类食宿：旅馆食宿、自备食物的食宿和营地食宿。

在这些定义中，关键词是“is a（是一个）”（或“is a kind of（是一种）”）。团体假期是一个假期，取消是一个事务，定金是一个付款，素食菜单选择是一个免费项。如果你能写下类似“A是一种B”或者“每个A都是B”的语句，就能够在分类中标识子类关系。

这有时被称为is-a测试：但是一定要谨慎，因为在英语中我们也常常用is-a表示单个实例与它的类型之间的关系：“Benidorm is a resort（Benidorm是一个旅游胜地）”。实际上，在这里用is-a-kind-of来表述更加安全。

正如我们将在后面看到的，标识子类非常有用，但是更重要的，它有助于理解对象类型定义。举例来说，如果你的类层次错误地将客户标识为旅游者的子类，很可能有人会迅速发现错误，并指出IBM是一个客户，但是仍然要登飞机。

如果你习惯于面向对象编程，你就已经在定义类型层次中领先了，然而，你仍然面临着潜在的威胁，因为程序员通常主要依据系统中的功能模块考虑对象类，而忽略了它们在外在世界中所表示的内容。如果你发现自己使用动词而不是名词来命名对象类型，那么你可能已经落入了这个陷阱。

下面的例子显示了我们将在假期业务中使用的类型层次的部分结构（参见图 4-1）。它使用UML表示法，其中箭头从子类指向超类。

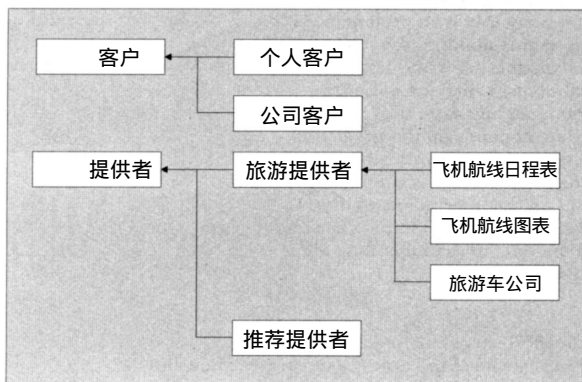


图 4-1

要了解更多有关UML和表示法的信息，参见《Instant UML》(ISBN: 1-86100-087-1)。

你可以进一步细化这张示意图，但是在本阶段，保持它的简明性有助于理解它要传达的主要信息，并且帮助你从用户那里获得反馈，说明他们是否正确理解了你要表述的概念。使用Visio等工具绘出这张示意图是非常有价值的，它使你能够扩展或压缩细节的层次，以便将同一个示意图以不同的形式提供给不同的听众。

步骤2是将你的对象类型组织到类型层次中。

3. 寻找关系

既然已经命名了对象类型，静态信息建模的下一步工作是确定对象类型之间的关系。

我们最好从几个英语句子开始：

- A customer books one or more holidays (一个客户可以预订一个或多个假期)
- Each holiday involves one or more travelers (每个假期包含一个或多个旅游者)
- Each holiday involves zero or more journeys (每个假期包含零个或多个旅行)
- Each holiday involves one or more accommodation-bookings (每个假期包含一个或多个食宿预订)
- Each accommodation-booking is at one hotel (每个食宿预订是位于一个旅馆的)

这些关系 (UML将之称为关联) 可以用示意图来表示 (如图 4-2所示)。我们可以通过多种不同的表示法描述这些对象关系，每种方法各具特色。在本章中，我们将使用 UML表示法，因为它已经被 IT界广泛接受，虽然我发现有时与用户交流时，最好使用更加自然的方式，例如：采用比较通俗的方式说明某个关系是一对多的。就我个人而言，我喜欢将示意图做得尽量简单和直接，将主要精力集中于关键的信息，而将细节留给文本文档，因为它们比示意图更易于维护。无论如何，细化到哪种程度比较合适取决于项目本身：如果你只有三个月的时间建立 Web 站点，就无法为数据模型中的所有对象类型构造详细的定义，即使你通过艰苦的努力完成了，也没有人有时间读它们，更不用说不断更新了。

对于每种关系，必须掌握以下基本点：

关系中的元组数说明了它能够包含的每种对象的数目：

- 最常见的是——一对多关系：一章可以有多个段落，一个人可以参加多个假期，一本书可以有多个版本，一个订单可以包含多个订购项。在上面的示意图中，我们将大多数一对多关系标记为一端只有一个对象 (1..1意味着至少一个，至多一个)，另一端可以有0至n个对象。在有些情况下，使用“1..n”更有意义；例如：如果一个旅游胜地没有旅馆，就不能称之为旅游胜地，因此在旅馆一端使用“1..n”表示。

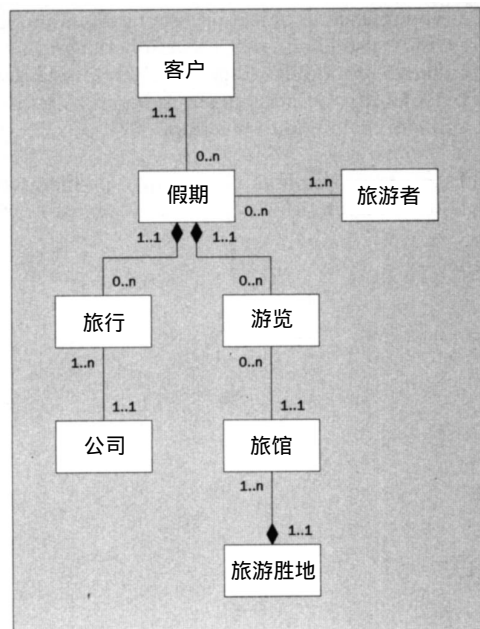


图 4-2

- 另外还有多对多关系：一个作者可以写几本书，而且一本书也可以有几个作者。在上面的示意图中也存在这种关系：几个旅游者可以在同一假期中一起旅游，而且每个旅游者也可以参加几个假期（企业旅游公司可能要跟踪这种情况）。对于多对多关系，通常要将每一对作为一个对象来命名：我们将由一个假期和一个旅馆组成的对称为一次游览（visit）。之所以采用这种方式，是因为它使得你能够有地方放置属于这种关系而不是任何一个对象的特性，例如：每个游览都有与它相关的日期和房间号。
- 一对一关系不太常见：这种关系的例子有人和工作的关系，一个人在任何时刻只能有一份工作，工作只能由一个人完成。

在为最终的XML表示建立信息模型的过程中，有一类关系特别重要——包含关系。它总是一对多或一对一的。包含关系的确切构成没有一定之规，不过我们可以从英语的句法中得到启迪：章包含段落；旅游胜地包含旅馆；旅馆包含旅游者。UML定义了两种形式的包含：聚合（aggregation）和构成（composition），聚合是相对松散的对象组合，它使得一组事物能够暂时被视作一个整体（例如：一个旅行团，某个人可能在不同的时刻属于不同的旅行团）；构成是一种更强的关系，其中的各个组成部分不能独立存在（例如：旅馆中的房间不能独立于旅馆而单独存在）。在UML中，聚合关系的聚合端用菱形表示。

我们还可以延伸聚合的概念：一个假期包含几次飞行；一个时间安排包含多个事件；一个电话服务包含几个产品。但是此时我们需要格外谨慎，因为我们很容易忽略一个事实——这些关系实际上是多对多的关系，我们有时会将它们误认为是一对多的。例如，许多时间安排能够包含相同的事件。虽然概念可以扩展，但是最好确保每个对象“仅包含在”另一个对象中：这与我们对“包含”一词的直观理解相吻合，而且也符合XML的数据模型，虽然XML的数据模型能够表达任意链接，但是它主要用于说明层次型结构。为了明确这一点，每当你在关系图中看到菱形时，在XML中应该将它模型化为包含的元素和重复的子元素，因此对于我们的例子，可以有如下XML结构：

程序清单 4-1

```
<HOLIDAY>
  <JOURNEY>
    <FROM>London Gatwick</FROM>
    <TO>Orlando, Florida</TO>
    <DATE>2000-02-15 11:40</DATE>
    <FLIGHT>BA1234</FLIGHT>
  </JOURNEY>
  <JOURNEY>
    <FROM>Orlando, Florida</FROM>
    <TO>London Gatwick</TO>
    <DATE>2000-03-01 18:20</DATE>
    <FLIGHT>BA1235</FLIGHT>
  </JOURNEY>
  <VISIT>
    <HOTEL>Orlando Hyatt Regency</HOTEL>
    <ARRIVAL>2000-02-15</ARRIVAL>
    <DEPARTURE>2000-03-01</DEPARTURE>
  </VISIT>
</HOLIDAY>
```

为关系寻找合适的名称通常是非常棘手的：这可不是英语语言所擅长的。最终你常常会使用include（包含）、use（使用）或has（有）等不确定的名称。更糟糕的是，根据你看待关系的角度，关系一般会有不同的名称。在描述关系时，最好使用完整的短语，例如：hotel is-located-in resort（旅馆位于旅游胜地中），person is-an-author-of book（人是书的作者）。幸运的是，我们不必将关系的名称作为XML标记：它们仅仅出现在系统文档中。因此，我们无需在示意图中标明关系的名称。

在第3步的结尾，我们已经定义了模型中对象类型之间现有的关系。

4. 定义特性

对象类型和关系构成了静态信息模型的骨架；特性的作用是在骨架上增加血肉。特性是与对象相关联的值。一个人有身高、体重、国籍和职业；一个旅馆有若干房间、一个被评定的星级和一个价目表。

在对象的特性列表中，不需要再次包含关系：如果我们已经建立了旅馆与旅游胜地的关系，就不必将“位置”作为旅馆的特性。

对于特性，最重要的是数据类型。是否要在固定范围内取值，它是数字吗，它的单位是什么？它是可选的还是必需的，是否有缺省值？

如果你曾经设计过关系型数据库，可能对规格化属性值有一定的研究：你应该明白表格中的所有特性值应该是原子的，如果发现非原子特性，你的本能将驱使你为它创建一个新的表格。在XML设计中，最好忘记这些规则。复合特性值不是XML设计中需要避免的问题。复合特性值的例子有：由数字值和测量单位（如：米或加仑）构成的测量结果，或者由钱数和货币单位构成的货币值。另外还有地址、地理位置和薪水历史。最终，你会将它们分为各个组成部分，但是层次化模型的最大特点是你可以将它们组合在一起。这也意味着你可以暂时不考虑详细的内部结构，而将它留至设计过程的后续阶段，这种方式有助于保持高层模型的清晰度，并且使之易于理解。

在UML中，你可以将对象的特性放在描述对象的框中，如图4-3所示。根据我自己的经验，我认为文本形式的特性列表通常更加易于维护，例如将它们存储在电子表格中。然而，为了便于解释，有时也需要在示意图中放入几个特性，以便说明该对象所代表的信息类型：

因此，在第4步的结尾，我们终于完成了静态信息模型：我们已经确定了系统中各个对象类型的定义，它们之间的关系，以及它们的特性。

现在，我们将转向信息建模的另一个重要方面：动态建模，它将描述系统运行过程中对信息可能执行的操作。

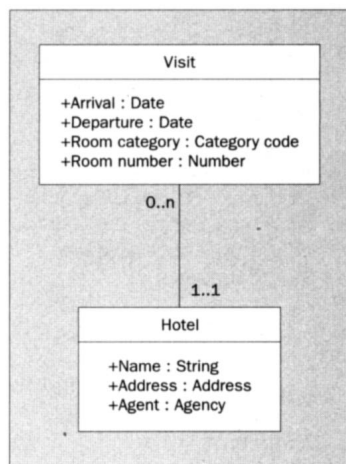


图 4-3

4.1.5 动态建模：对数据进行哪些处理

到目前为止，我们已经了解了静态信息模型。如果你希望使用XML表示系统中流动的消息，

还需要理解以下问题：我们将对数据进行哪些处理，这些数据来自何方，下一步要送往何处。

动态建模可以采用以下几种方法。在一个项目中可能不会用到所有方法，因此我们不像介绍静态建模那样提供简单的按部就班的方法。在本节中，我们将列举一些可选的技术，你可以根据自己项目的特点选择适当的方法。我们将介绍的内容有：

- 处理模型和工作流模型
- 数据流模型
- 对象模型
- 对象生存周期历史
- 使用案例
- 对象交互图

记住，我们的目标是争得大家的同意，获得众人正确的理解：创建模型有助于大家理解系统预期的功能。你只需要将系统中比较困难的部分模型化，而不必为系统的各个部分建立模型。

1. 处理模型和工作流模型

处理模型和工作流模型侧重于人和企业在完成任务过程中所扮演的角色，信息存储和处理阶段是相对次要的。例如，处理模型将描述旅游者在度假过程中遇到事故时怎么办：它定义了旅游胜地的当地代理、本国的代理和总公司的职责，它明确了谁应该负责安排医疗、安排返程和通知亲属。为了使整个过程顺利进行，它可能还规定了要填写和传递的各种表格，它或许根本不包含任何计算机系统。处理模型通常集中于角色、责任，以及系统中各方参与者的任务，而工作流模型更多地关注于在参与者之间传送的文档。

2. 数据流模型

数据流模型与处理模型非常类似，但是它更多地侧重于信息系统，而不是业务。数据流模型描述了数据存储、处理器和数据流。数据存储规定了将信息永久性地保留在何处（例如：计算机数据库，或者仅仅放在档案柜中）；处理器将对数据进行操作；数据流是指将数据从一个处理器或数据存储传送到另一个处理器或数据存储。数据流模型极其依赖于静态信息模型；静态模型通过旅游者或旅馆等概念的定义描述了它们的含义，但是它没有说明信息的存储。相反地，数据流模型将说明有关假期的信息会一直保留在预订数据库中，直至假期结束并结清所有帐目，此时，关于假期详细情况的总结将转移到市场信息系统，其余内容将传送到归档存储。

3. 对象模型

对象模型包含动态部分和静态部分。对象定义的动态部分或称行为部分侧重于每个对象能够做什么，它通过一组操作或方法定义了对象的行为。

就我个人而言，我并不认为行为对象模型对建模有很大的价值；它更适合作为设计工具。因为许多事件都是与多个对象相关的（例如：旅游者到旅馆登记），到底将它与其中的哪个对象相关联纯粹属于设计决策。

4. 对象生存周期历史

对象生存周期历史（UML称之为对象生命线）也关注于单个的对象，但是它是从整体角度

考虑的：它描述了每个对象在生存周期内所执行的操作——如何创建对象，在对象的生存周期中能够发生哪些事件，对象如何响应这些事件，什么情况使得它最终被清除。

我发现对象生存周期历史对于测试模型的完整性非常有价值。人们常常会特别注意一些事件，而忽视另一些事件——例如，模型可能描述了如何处理假期预订，而忽略了如何处理取消预订或要求退款的情况。只有定义了每个对象如何进入系统，以及如何从系统中删除，你才会发现原来的考虑可能存在着纰漏。

5. 使用案例

使用案例分析了特定的用户任务是如何完成的，例如：预订了假期的人如何取消订单？使用案例可以与处理模型非常类似，但是它通常侧重于特定用户的活动。

在将业务模型化和描述 IT 系统的内部行为方面，用户案例都非常有用。但是，这两个层次有可能混淆；你最好将这两者分离，因为它们是针对不同的听众的。虽然如何发挥使用案例的作用要根据实际情况而确定，但是我所见过的最有价值的实例是将使用案例主要应用于用户与系统的交互——描述对话框，例如：“用户通过在地图上点击说明所希望的度假地点。系统将据此提供旅馆列表，每个旅馆都配以一幅照片，并且注明了旅馆的名称和价目表。用户可以点击任何旅馆，进一步了解有关该旅馆的信息。屏幕上显示的信息中将包含一个按钮，用户可以通过它进行预订……”，等等。

UML为描述使用案例提供了图形表示法，但是最好的方法是通过交互式原型，人们常常称它为storyboard。

通过使用案例描述的用户界面对话框将集中说明用户和系统之间的交互信息，而不是解释对话框在屏幕上如何显示。这使得相应的 XML实现中的信息内容自然而然地独立于表示细节。

6. 对象交互图

对象交互图从一个比数据流模型更加详细的层次分析对象之间的消息交互。数据流模型可能仅仅说明在线预订系统向信用卡 clearer发送交易信息，并收到认证信息作为应答，然而，交互图将详细描述构成这次会话的所有消息。

当你需要描述两个独立系统之间的交互时，例如：旅游代理系统与航空公司之间的交互，对象交互图是一种非常出色的方法。它能够帮助你定义将哪些信息封装在消息中。由于这些消息通常是XML消息，因此对象交互图在设计每个消息的 XML结构提供了所需的上下文。

7. 选择动态建模方法

我们在八十年代提倡的某些系统设计方法鼓励你坚持这样一个信念：在开始任何编码之前，必须详细定义所有方法。这种观点并非总是正确的；Internet变化如此之快，使得这种观点难以贯彻。但是，了解能够适应不同情况的各种技术是非常有必要的，因此我们上面介绍的内容对你来说应该具有一定的价值。

在动态建模时，并不一定要建立上述所有模型，你应该学会变通——这些都可以作为非常有用的工具，当你会见用户（或者与设计小组的其他成员交谈）时，可以在白板上进行简要叙述，以征求大家的意见。如前所述，当你在 XML项目中定义从一个子系统流向另一个子系统的消息时，或者当你在电子商务中定义从你的系统流向供应商或客户的系统的消息时，你会发现这些方法自有其用途。

4.2 设计XML文档

本章的第一部分着重分析了现实世界中要表示的信息，它使得每个人能够对这些信息达成共识。分析到此为止，现在让我们再回到 XML。在本节中，我们将假设已经理解了系统的信息需求，现在，我们的核心问题是如何设计能够满足这些需求的 XML 文档。我们首先来看看 XML 在系统体系结构中能够扮演的角色。

4.2.1 XML的两种角色

在讨论数据流模型时，我们看到系统中有两种类型的数据：数据存储和消息流，数据存储提供长期的永久性数据，以便于系统引用，消息流将暂时存在的信息从一个子系统（或者人，或者数据存储）转移到另一个子系统。

XML适用于这两种类型的数据，但是它们在设计时要考虑的问题相差甚远，因此我们将依次进行介绍。

1. 用于消息的XML

将与XML用于永久性数据相比，将 XML用于系统中的消息时引发的设计问题较少。这是因为每个消息通常相当完备，而且一条消息中应该包含哪些内容一般可以由处理模型得出。

当然，我们使用的消息一词有非常广泛的含义。它可以是企业之间发送的用于表示旅馆预订交易的EDI式消息。或者，它也可以是一个子系统向另一个子系统提出请求时传递的数据，例如：在银行中，欺骗检测系统可以要求银行运作系统提供特定帐户的交易历史，并接收 XML消息形式的应答。除此之外，还有为用户提供显示的消息，例如：当你通过浏览器咨询特定旅游胜地的天气预报。在这种情况下，应答将以 XML消息的形式从后台数据库系统返回前端，前端系统（或浏览器）利用适当的样式表进行展示，将应答消息变为可视化页面。有些消息可能源自人，例如：销售查询是由 Web 站点上的表单创建的。

以下通用的设计原则可以应用于所有 XML 消息，无论这些消息具体的用途是什么：

- 设计应该反映信息内容，而不是体现预期的用途。随着时间的推移，信息的用途可能发生变化，但是信息的内容一般是稳定的。这条原则特别针对于表示细节：如果信息是供人类使用的，避免包含屏幕布局和字体大小等表示信息。
- 设计应该预计到可能的变化。当然，XML 本身的设计在这方面是领先的，它能够避免传统的缺陷，例如：固定长度的字段和固定的列顺序。但是，文档设计者在将信息结构化时有责任采用一种能够在一定程度上适应未来变化的方式。
- 尽量使用标准消息类型，减少自己的发明。目前标准化的消息类型越来越多，例如：最初由 Microsoft 发起的 Biztalk (<http://www.biztalk.org/>)，以及独立的 OASIS 协会 (<http://www.oasis-open.org/>) 都提供了许多例子。除此之外、<http://www.ontology.org/>、<http://www.rosettanet.org/>和<http://www.commercenet.org/>等站点也值得一看。第 1 章介绍了业界在为特定应用领域创建标准的 XML 词汇表和消息类型方面所做的努力。即使不能原封不动地使用它们，也能够从中汲取许多灵感，或者可以利用第 7 章介绍的命名空间和模式将它们融入到自己的扩展模型中。

- 在性能允许的范围内，数据编码应该与自然编码尽可能接近。避免过于简洁的编码，例如：将W作为Withdrawn的编码，除非这种编码已经被业界广泛接受。使用该行业中已有的标识符和编码（比如：信用卡号），尽量不要自己杜撰标识符，（特别要）避免会使消息和现有数据库形成密切关系的标识符。消息应该是独立的。

有一个设计决策比较棘手，即：是否包含一些目前尚不需要的信息，以备将来使用。对于这个问题，没有一致的答案。如果包含对象的所有特性比从中选择接收者感兴趣的特性更加简单，可以将它们全部包括在内，由接收者决定舍弃哪些特性。另一方面，发送大量多余的数据是非常愚蠢的，我们也可能出于安全性或隐秘性考虑保留一些特性。这个决策应该是切合实际的。

在消息中包含常规信息是非常明智的，即使它重复了任何用于传递数据的消息系统都能够提供的信息。日期和时间、发送者标识和预期的接收者是显然要包含的信息；你还可以加入序号，它能够检查传输过程中是否有消息丢失，或者是否有消息被传输了两次。之所以在XML消息中包含许多相关信息是因为：它使得接收者能够更方便地从消息中提取信息，而不必使用单独的API；更重要的是，如果消息是为了审计等用途而进行的归档，将所有内容放在一起便于存储。

2. 用于永久性数据的XML

消息的设计主要是由动态信息模型决定的。相反，当将XML用于永久性数据时，静态模型是至关重要的。

(1) 一个文档有多大

在设计过程中，最困难的部分莫过于决定数据的粒度：一个文档中应该包括多少内容？

对于有些应用来说，一个包含几千兆字节的XML文档是比较适宜的，而有些应用宁愿使用小文档。XML不太适于直接访问；就当前技术而言，如果要访问这样一个大文档的任何部分，都必须解析整个文档，这可能要耗费几个小时。

从另一个极端看，拥有许许多多小文档也不太理想，因为它不可能利用XML丰富的结构来表示信息模型中的关系。

有时，当业务对象非常大而且内部结构很复杂时，自然会将每个对象映射为一个XML文档。例如，在人力资源系统中，自然要为每个职员记录使用一个XML文档，或者在病历系统中，每个病历都对应一个XML文档。

当你使用XML保存永久性数据时，寻找信息的过程通常可以分为以下两个步骤：首先找到正确的文档，然后找到你所感兴趣的内容。这两步用到的工具和技术截然不同，因此决定在每个文档中放置哪些信息是相当关键的。

要定位正确的文档，主要有以下可选的四种方法：

- 利用操作系统文件存储中的目录结构通过名称定位文档。文档的文件名可能与系统中使用的对象标识符相关，人事文件可能以职员的人事编号命名。
- 利用文档之间的相互索引，这种方法类似于传统的Web站点，文档总是通过链接找到的，不过在此通常采用更加结构化的文档组织方式。举例来说，如果有一组足球比赛报道，每个报道都是由一个XML文档构成的，可以建立另一个文档作为它们的索引，列出所有比赛

——日期、地点和参赛的球队。当然，不必手工维护这个索引。可以将它设置为自动更新：提交新的比赛报道后，系统自动对它进行分析，并将相关信息添加到索引文档中。XSLT 非常适于编写这类应用。

- 利用关系数据库索引文档。还可以选择将 XML 文档保存为文件形式，并通过数据库引用，或者将 XML 文档也存储在数据库中。越来越多的数据库明确表示支持该功能，如果必要的话，还可以使用“blob”(Binary Large Object, 二进制大对象)字段。如果利用关系数据库索引 XML 文档，就可以通过任意 SQL 语句识别它们，但是只能使用那些数据库直接包含的特性。要深入了解如何使用 XML 和关系数据库，参见第 10 章。
- 利用自由原文搜索引擎索引文档。越来越多的搜索引擎提供对 XML 的支持。通过这种方法，可以根据在文档的任何位置出现的关键字搜索文档。虽然自由原文搜索通常被定位为支持用户对非结构化数据的查询，但是由于 XML 文档中的标记使得文档具有更强的结构性，因此将使这种查询方法变得更加有效。对于数据更新量不太大的应用来说，这种方法比使用关系数据库查询更加高效。

另外，你还可以使用所谓的“XML 服务器”。XML 服务器通常不以未解析的纯文本形式保存 XML 数据，而是将它们存为 DOM 形式——即：它将文档对象模型中的节点保存为对象数据库中的对象。这样，不必首先从磁盘读取整个文档，然后再解析它，而可以直接通过 DOM 接口访问数据。这种方法的主要优点体现在它使得数据访问变得更加平滑，避免使用一个 API 定位正确的文档，使用另一个 API 在文档中搜索信息。然而，XML 服务器将每个 DOM 节点保存为一个独立的数据库对象，这使得它的性能受到一定影响，特别是当加载或更新大型数据库时。另外，由于标准的 DOM 接口只能通过遍历来寻找数据，因此具体的查询方法是每个厂家所特有的。

另一种能够降低解析大型文档的代价的方法是将文档缓存到内存中。例如，对于 Microsoft 的 ASP 页面或 Java Server Pages，你可以在应用程序范围内保存数据，这意味着当 Web 服务器启动时，将数据读入内存，当服务器关闭时，从内存读出数据。对于几千兆字节的文档，这种方法的代价过于昂贵（就大多数 DOM 实现而言，源 XML 文件中每字节的内容大约会占用 10 个字节的内存），但是对于几兆字节的数据来说，这种方法还是切实可行的——添置内存或许比购买 XML 服务器软件更便宜。

凭经验而论，我习惯于根据一个称为“整屏”的概念决定合适的文档大小。从理论上讲，这种方法是不能令人满意的，毕竟 XML 应该是独立于表示的，以每次要显示给用户的信息量作为设计决策的主要依据似乎不太恰当，然而实际上，这是一种很好的测试方法。如果存储的 XML 文档所包含的数据量超过了一个用户愿意在屏幕上看到的内容，最终你所解析的大量数据可能永远得不到用户的青睐；相反，如果你需要访问几百个存储的 XML 文档才能够形成一屏信息，在访问每个用户页面时都会产生大量处理开销。

但是，如果用户希望每次看到不同的一小部分数据呢？可以将整个 DOM 文档作为应用程序的一个对象保存在服务器的内存中，根据用户请求对 DOM 文档进行过滤，产生能够更精确地满足用户请求的较小的 XML 文档。

我们将在第 10 章详细介绍 XML 和数据库。

(2) 需要多少种文档类型

令人惊讶的是，这个问题并不像看上去那么简单，因为 XML并没有严格定义文档类型的概念。特别是有两个独立的概念，它们并不一定是一一对应的：DTD和顶级元素类型。

但是从概念角度讲，模型中可以包含几种不同的文档类型，在 XML中可以通过以下几种方式表示它们：

- 每种文档类型对应于一个不同的 DTD。如果不同的文档类型中有部分内容是公共的，可以定义这些 DTD都能够共享的外部参数实体。
- 可以为所有文档类型使用相同的 DTD，但是每一部分使用不同的顶级元素标记。DTD没有规定哪个元素应该作为文档元素，因此可以在同一 DTD中定义几个文档类型。通过这种方式，各个文档类型之间可以方便地互相引用。

DTD本身不一定是一个孤立的实体；它可以通过外部参数实体引用其他 DTD中的定义，或者通过条件部分进行参数化，因此它实际上能够变得非常灵活。遗憾的是，这同时也意味着复杂性：组织机构常常会发现由于某个 DTD过于复杂，而导致其他人无法理解它。

无论选择哪种方法，都应该尽量保证不同文档类型的通用性。这意味着要将名称和地址等公共部件标准化，将元素和属性的命名规则标准化，将消息的时间标签和序号等重要数据标准化。

本章稍后将进一步讨论 DTD的设计。

4.2.2 将信息模型映射到XML

在本节中，我们将详细介绍如何将信息模型的不同部分映射到 XML文档结构。

1. 表示对象类型

通常，信息模型中的对象类型将转换为 XML结构中的元素类型。

你可以将对象类型名称作为元素名称，或者如果对文档占用的空间很在意，可以采用对象类型名称的缩写。大多数人都喜欢给元素标记使用短名称，这不仅仅是为了节省空间，而且因为它能够增强 XML的可读性：或许它能够避免人们将过多的注意力集中于标记，而忽视了真正的内容。

对象类型是类型层次的一部分，你能够选择从哪一层开始建立 XML元素。举例来说，如果 deposit是payment的子类，可以将<deposit>作为元素标记；或者将<payment>作为元素标记，并通过其他方式说明这是一个 deposit，或许可以借助属性，如：<payment type="deposit">。

使用最特殊的类型（ deposit ）的优越性体现在 DTD能够更精确地定义与该元素相关的属性和子元素。使用更通用的类型（ payment ）的优越性体现在它从一定程度上简化了应用程序代码的编写，你不必关心它到底是哪种类型的 payment。

永远记住，设计数据结构时不要单纯为了程序员编写第一个应用程序方便。因为当这个程序寿终正寝后，数据结构仍然会存在很长时间。

2. 表示关系

模型中的某些关系可以利用 XML文档结构中的元素嵌套表示。显然，“包含”关系可以通过这种方式表示，例如：<footnote>元素可以嵌套在<chapter>元素中，<hotel>元素可以嵌套在<resort>元素中。

毫无疑问，一个元素只能被一个父元素包含，因此必须采用其他方式处理模型中其余类型的关系。实际上，这意味着某种链接。

在XML中，有几种表示从一个元素到另一个元素的链接的方法：

- 可以使用ID和IDREF属性。ID属性包含的值能够在文档中唯一标识元素；IDREF属性的值必须与文档中某个元素的ID相同。因此，ID可以作为关系数据库中的主关键字，IDREF作为外部关键字。另外，不要忘记IDREFS数据类型，关系数据库中没有直接与之对应的结构：IDREFS包含一组以空格分隔的IDREF值，因此它可以作为一对多关系的“一”端，或者多对多关系的任何一端。
- 可以使用XPointer引用。它与HTML中我们熟悉的HREF属性等价。可以通过XPointer值引用同一文档或其他文档中的元素。
- 可以在XML文档中使用应用程序定义的主关键字和外部关键字，而不对它们进行特殊声明。应用程序知道这些值是用于表示关系的，而XML软件会将它们视作普通数据。

以上三种方法各有所长。使用ID/IDREF的主要优势在于XML解析器能够帮助你进行文档的有效性验证。遗憾的是，标准的DOM接口并不提供根据ID寻找元素的方法，但是许多DOM实现都通过专有的接口弥补了这一缺陷，例如Microsoft提供了nodeFromID()方法。XSLT也提供了使用ID值的机制，但是通过将ID值作为关键字访问元素并不比通过非ID值访问元素更加有效，因为它不支持执行更高级的操作。使用ID还有其他局限性，例如，它们在整个文档中必须是唯一的（仅仅保证在一个元素类型中的唯一性是不行的），每个元素只能有一个ID属性，它们必须遵守XML名称的语法规则（这意味着“234”、“I18/296”和“ABC 123”都不是有效的ID）。因此，这种方法不占有绝对优势。

在下面的例子中，使用ID是非常合适的。在团体旅游的旅行计划中，有可选的活动和游览项目，但是旅游团中的每个人并非都有相同的选择。例如，父亲母亲要去娱乐场所，而小孩要去动物园。在旅行计划中，我们将通过以下方式表述这种情况：每名游客都有一个ID值，在游览项目中，利用IDREFS字段列出所有参加该项目的游客，具体代码如下：

程序清单 4-2

```
<itinerary>
  <traveler id="t01">
    <name>Mrs Mary Higgins</name>
  </traveler>
  <traveler id="t02">
    <name>Mr John Higgins</name>
  </traveler>
  <traveler id="t03">
    <name>Rory Higgins</name>
    <age>12</age>
  </traveler>
  <traveler id="t04">
    <name>Kylie Higgins</name>
    <age>9</age>
  </traveler>
  <excursion participants="t01 t02">
    <venue>Casino</venue>
    <date>2000-06-15</date>
```

```
</excursion>
<excursion participants="t03 t04">
  <venue>Zoo</venue>
  <date>2000-06-15</date>
</excursion>
</itinerary>
```

在以上代码中，我们采用了单向链接——从游览项目到游客。当然，我们也可以使用反向链接，在<traveler>元素中设置游览项目属性，或者可以使用双向链接。通常，我尽量避免双向链接，因为它产生的唯一结果就是必须检查一致性，并确定当出现不吻合的情况时该如何处理。如果决定使用单向链接，无论哪个方向都可以。对于一对多关系，你会本能地在“多”端使用IDREFS属性，因为在关系型表中就是这样处理外部关键字的，但是在“一”端使用IDREFS属性也未尝不可。

XPointer引用比ID/IDREF更加灵活，然而遗憾的是它尚未完全标准化，而且未被广泛实现。令人欣慰的是这种局面即将得到改善——XPath标准的制定工作接近尾声，它已经成为W3C建议，它将作为以前的XPointer语法的替代品。XPointer的优势在于它允许从一个文档到另一个文档的交叉引用，虽然你需要知道目标文档的详细结构。XPointer不需要在DTD中标识，因此解析器不会验证它的有效性。如果你选择直接支持XPointer值的XML软件，可以考虑利用它实现链接，然而或许它仅仅是一种我们应该密切监视的未来的技术，它在目前的环境下并不是真正实用的。

显而易见，最简单的XPointer是URL，只要有理由使用独立的文档保存数据，使用URL表示关系就是非常值得的。

假设我们的旅游公司负责组织一次会议。会议的一项安排是为每个与会代表安排单独的出游路线。因此，应该为会议建立一个XML文档，并且分别为每个代表的路线建立一个XML文档。例如：

程序清单 4-3

```
<conference>
  <title>13th Annual High Achievers' Celebration</title>
  <resort>Honolulu</resort>
  <delegates>
    <delegate>
      <name>Jane Sales</name>
      <itinerary>http://high-speed.com/itinerary/JS0002.xml</itinerary>
    </delegate>
    <delegate>
      <name>John Pusher</name>
      <itinerary>http://high-speed.com/itinerary/JP0008.xml</itinerary>
    </delegate>
  </delegates>
</conference>
```

现在，我们只能链接到XML文档，一旦XPointer规范标准化，就能够链接到文档中的某个元素。例如，旅馆的所有菜单应该位于一个文档中，每个代表选择的菜单应该是指向各个菜单的XPointer。

另外，利用应用程序级的关键字处理关系也是一种非常可行的方法，它提供了最大的灵活

性，你可以采用任何有效的方式对关系进行操作。唯一的缺陷是 XML 解析器不能为你提供任何帮助。借助 XSLT 样式表中的 key() 函数，能够以非常直接的方式处理通过这种方法实现的关系。

要了解更多有关链接的信息，参见第 8 章。

3. 表示属性

当你在信息模型中标识了特性之后，就出现了一种经典的进退两难的局面：在 XML 文档中，你应该使用 XML 属性表示它，还是使用嵌套的（子）元素表示它？一旦做出决定，还有其他因素需要考虑。

元素还是属性

在下面的例子中，书的特性表示为 XML 属性：

程序清单 4-4

```
<book
  author="Nelson Mandela"
  title="Long Walk to Freedom"
  publisher="Abacus"
  isbn="0-349-10653-3" />
```

对于相同的信息，下面的例子将特性表示为子元素：

程序清单 4-5

```
<book>
  <author>Nelson Mandela</author>
  <title>Long Walk to Freedom</title>
  <publisher>Abacus</publisher>
  <isbn>0-349-10653-3</isbn>
</book>
```

哪种表示方法更好？这是常常令首次设计 XML 文档的人感到困惑的一个问题，虽然有人会回答“无所谓，它们没什么区别”，但是这个问题的确值得详细探讨，通过我们的分析，你将能够根据实际情况权衡利弊。然而，需要提前说明的是，即使专家在这方面也不能达成一致意见，而且有时他们会产生严重的分歧。

首先，让我们看看如何从哲学角度分析到底应该选择哪种方法。有人认为子元素代表被包含的对象——它有自己的存在和独立于容器的标识——而属性代表与对象相关联的值，它没有自己独立的生命。这种论点的含义非常深刻，如果你准备沿着这条路线走下去，最好在与别人争论之前先研读一下亚里士多德的论著。依据这种逻辑，你最终会决定将 age 表示为属性，而将 place-of-birth 表示为子元素（因为地点是个有自己权利的对象）。这种思路产生的结果不太直观，而且也不是非常有帮助，它会使你陷入对每个特性徒劳的争论中。

另一种推理路线是揣摩 XML 标准设计者的意图。早在 SGML 中就已经存在子元素和属性之间的区别了，因此我们可以问这样一个明显的问题：最初的 SGML 设计者如何看待这两种结构的角色呢？

正如我们所知，SGML 最初是作为一种标记语言为发行提供文本。后来，其他应用程序也将它作为通用的数据交换格式。从标记语言角度考虑，SGML 中的内容（最终用户将在页面上看到

的文本)和元数据(有关内容的信息,它是供执行各种处理操作的软件使用的)有着明显的区别。简单来说,内容是用元素标记中的文本表示的,元数据是用属性表示的。这种差别一直保持到HTML中,而且在HTML的发展过程中引起了许多问题:例如客户端 JavaScript部分不能作为属性被正确处理,因此要将它们放置在<script>元素中;但是由于不希望用户看到这些代码,因此要将它们注释掉——这简直是滥用注释。

所以,SGML中元素内容和属性之间的差别从本质上体现在信息用途上的差异:元素内容是供文档的读者使用的,属性是供印刷商或他们的软件使用的。当然,一旦将SGML或XML用于软件系统之间的数据交换,这种差别就丧失了意义。即使有人类读者,例如电子表格,决定哪些供软件使用,哪些供用户使用也是很困难的。

综上所述,回顾历史并不能使我们得到正确的答案。所以,我们能够得出这样的结论:只要从实用角度考虑,你可以自由选择使用元素或属性。

下面我们将从正反两方面分析一下这两种方法(参见表4-1)。

表 4-1

	优 点	缺 点
XML属性	DTD能够对值进行约束:如果只允许有限的几个值,例如:“yes”和“no”,使用属性非常有效	只支持简单的字符串值
	DTD能够定义缺省值	不支持元数据(即“属性的属性”)
	验证ID和IDREF的有效性	无序
	占用较少的空间(当你通过网络发送几千兆字节的数据时,它的优势非常明显)	
	对于某些数据类型(例如,NMTOKENS),能够进行空白的规格化,它能够减轻应用程序解析的压力	
子元素	便于使用DOM和SAX接口进行处理	
	能够访问未解析外部实体,例如: 二进制数据	
	支持任意复杂的值和重复的值。	空间占用率略高
	有序	
	支持“属性的属性”	编程比较复杂
	当数据模型改变时,子元素可扩展	

至于如何权衡这些因素,取决于应用程序的实际情况。许多有经验的设计者认为最重要的因素是变化的潜力:随着应用程序的发展,是否能够扩展文档或消息格式。这一因素使得子元素优于属性,因为实际应用中最常见的变化是将一个简单的特性(例如:“作者”)扩展为复杂的结构化特性(例如:作者列表,其中每一项都由姓和名标识)。但是对于你的应用程序来说,这是否是正确的选择,只有你自己才能决定。值得一提的是,最初由Microsoft发起的Biztalk提出了截然相反的建议,他们认为属性更加可取,因此你应该根据实际情况来决定。

4. 属性值编码

无论使用元素还是属性表示对象的特性，都必须决定如何对它们的值进行编码。

以下列出了一些常见的情况：

(1) 数量，例如高度、宽度和重量

第一个问题的是否将度量单位标准化（例如，所有长度都以米为单位），还是使用多种不同的度量单位。如果度量单位是数据的一部分，你可以将它作为属性值的一部分（`height="1.86m"`），或者将它分离为独立的项（`<height units="m">1.86</height>`）。最后，必须定义十进制数字的格式，例如，用什么符号表示小数点和千位分隔符。一旦建立了 XML 模式，只需引用标准的数字数据类型，而不必担心词法编码，但是目前这个任务仍然要由文档设计者承担。

要深入了解 XML 模式，参见第 7 章。

(2) Yes/No 值

在英语语境中，使用字符串 `yes` 和 `no` 作为明确的表示，如果空间非常宝贵，可以使用缩写 `y` 和 `n`。如果用属性表示模型中的特性，还可以使用 SGML 的习惯用法，HTML 中也采用这种方式——例如 `border="border"` 意味着 `yes`，如果缺少 `border` 属性意味着 `no`。这种方式在 SGML 中有一定的优越性，但是对于 XML 来说，它已经成为历史。为了保证与 SGML 的互操作性，XML 不提倡同一元素有两个值集合相交的属性，例如，两个属性都允许值 `yes` 和 `no`。

(3) 其他有限的值集合

对于这些有限的集合，最基本的决策是使用代码还是完整的名称，例如：`W` 或 `Withdrawn`，`uk` 或 `United Kingdom`。如果不考虑空间因素，扩展的名称的最主要优点在于它能够避免误解——用户对代码的理解常常与设计者的初衷大相径庭，结果导致数据含义的混淆。

(4) 日期和时间

许多人都提倡根据 ISO 8601 的日期格式（`YYYY-MM-DD`）进行标准化，以避免出现模棱两可的情况。ISO 8601 还定义了时间的表示法，其中包括时区。

实际上有许多标准是关于特定类型的特性的。表 4-2 列出了国际标准化组织（ISO）发布的一些标准。

表 4-2

标 准	范 围	说 明
ISO 2955	公制度量单位	在信息系统中使用有限的字符集（不包括希腊字母）表示 SI 和其他度量单位的方法
ISO 3166	国家代码	表示国家名称代码的规范；它包含两字母代码和三字母代码。两字母代码是广为人们熟悉的 Internet 域名，例如： <code>de</code> 代表德国
ISO 4217	货币代码	国家货币的代码
ISO 5218	人类性别	表示男和女的代码（这肯定是目前所发布的最短的国际标准）
ISO 6093	数值	三种数值表示法：字符串形式，机器可读的形式和人类可识别的形式
ISO 6709	位置	使用经度、纬度和海拔高度唯一地标识位于地球表面、地上和地下的点
ISO 8601	日期和时间	日期表达式，包括：日历的日期、序数的日期、星期几和时间，它使用标点符号进行分隔以免引起混淆

当然，你没有义务一定要使用这些标准，但是遵循标准有两大优点。第一，别人考虑到的困难可能比你想到的多，第二，告诉其他人你将遵照标准执行能够减少徒劳无益的争论。

除此之外，还有许多其他标准：Internet RFC 1766定义了自然语言的编码机制，它已经被广泛采纳；X.500/LDAP标准定义了一套精心研制的在世界范围内表示人名的方案（当然不仅仅是名字，中间名的首字母和姓）；另外还有许多应用于特定领域的标准，例如：出版领域的 ISBN，运输业的机场编码，等等。

在不久的将来，我们期待更多的通用数据类型被标准化，它们将成为 XML模式活动的一部分。到那时，XML解析器将接管解析和验证使用这些类型的数据的任务，它能够从一定程度上减轻应用程序的负担。

(5) 特性的名称

当你使用元素或者属性表示信息模型中的特性时，应该使用什么名称呢？这个问题似乎是显而易见的，然而实际上你可以有两种选择：以数据类型命名，或者以它在父对象中的角色命名。举两个简单的例子，文档的创建日期可以命名为 date-created或date；客户的地址可以命名为 address或billing-address。

当然，如果存在多个日期或地址，你别无选择只能使用较长的形式，你或许认为它的作用仅仅是消除了含糊不清，从某种意义上讲确实如此。遗憾的是，这种形式同时也删除了隐含的数据类型信息。应用程序软件如何知道 billing-address和service-address有相同的有效性约束？另外，如果每个元素都通过这种方式来区别，DTD将变得相当复杂。

相反，如果采用较短的命名形式，有可能使多个对象类型都包含相同的特性名称，这是否会导致该名称重载？例如，旅游胜地、旅馆和客户都有名称——在XML文档中使用<name>标记表示它们是否合理？从本质上讲，在XML中元素名称是全局的（无论出现在哪里，含义都是相同的），而属性名称则是属于特定的元素类型，它对于该类型来说是本地的。这并不是不可违背的规则；你可以重载元素名称，特别是用于表示 NAME或TITLE等简单文本特性的元素，同样，在某些环境下（特别是XSLT样式表），属性名称也可以被视作有全局意义。无论在哪种情况下，最好不要用相同的名称表示含义截然不同的事物，这是一条通用的设计规则，例如：不要用STATUS同时表示个人的信贷价值和航空公司承诺的特性。

对于命名问题，你要考虑以下选择：

- 使用系统的命名习惯，例如：<Billing.Address>和<Creation.Date>。这不仅能够保证含义清晰，而且能够保持简洁性，你甚至还可以将这种命名习惯扩展到应用程序软件中，当然这对于XML软件来说是没有特殊含义的——特别要指出的是，这种结构化命名方式不能应用于XSLT。（XML名称允许句点，它与其他字符的地位平等，没有特殊意义。）其他可选的方法还有：在单词之间使用下划线，使用 camelCase表示法（第二个单词的首字母大写），但是句点表示法对于处理程序来说更有意义。
- 使用能说明含义的名称作为元素名称，用属性表示数据类型，例如：<BillingAddress type="Address">。这意味着应用程序仍然可以获得数据类型。实际上，由于每个BillingAddress都是一个Address，因此每个实例不必都包含 type属性，你可以在DTD中定义FIXED的属性值。然而，这种方法不能在 DTD中自动定义结构化规则，说明所有

type="Address"的元素符合相同的样式。

- 采用完全相反的方法：使用数据类型作为元素名称，用属性表示元素扮演的角色，例如：
<Address role="Billing">。从编写DTD的角度看，这种方法更好，虽然它仍然存在局限性：现在，你的确能够定义地址的规则，但是不能说明每个客户必须有一个帐单地址（Billing Address），而前一种方法能够做到这一点。

- 使用嵌套的元素层：

```
<Customer>
  <BillingAddress>
    <Address>
    </Address>
  </BillingAddress>
</Customer>
```

从理论角度讲，这可能是目前为止最好的方法：你能够定义 Customer必须包含一个BillingAddress，BillingAddress必须包含一个Address（且不能含有其他元素），此处的Address与文档中出现的其他Address将遵循相同的规则。唯一的缺陷是，如果你对于模型中的每个特性都严格应用这个规则，结果将产生大量标记。考虑到 DTD的数据类型验证功能是相当有限的，因此这种方法的前景也不太光明。

(6) 二进制数据

并不是对象的所有属性都能够用字符串表示：多媒体数据，特别是二进制数据就是例外。在XML设计中，如何表示图形等二进制数据呢？

主要有两种方法：二进制对象可以是内部的或者外部的。内部的意味着它们表示为 XML数据流的一部分，外部的意味着它们位于独立的文件中。

对于内部存储，大多数人使用 Base64编码。这种编码技术将每个二进制数字序列编码为一个ASCII字符。它非常适于XML，因为ASCII字符通常不会与在XML中有特殊含义的标记序列（例如：“<”和“]]>”）冲突。这意味着数据中不会偶然出现定界符。当然，二进制数据与Base64字符串之间的转换工作应该完全由应用程序承担。

对于外部存储，“纯XML”方式是使用外部未解析实体和表示法。例如，为了包含对图形文件picture1.gif的引用，你应该编写如下DTD（通常在内部子集中）：

```
<!NOTATION gif SYSTEM "gifeditor.exe">
<ENTITY picture1 SYSTEM "picture1.gif" NDATA gif>
```

其中，gifeditor.exe代表能够处理这种采用特殊格式的数据的应用程序的名称。实际上，除非选择的XML工具集有其他定义，否则它就用于唯一地标识一种格式。对未解析实体的引用是作为ENTITY类型的属性的值出现的，因此DTD将包含以下声明：

```
<!ELEMENT picture EMPTY>
<!ATTLIST picture name ENTITY #REQUIRED>
```

最后，在文档中你希望图形出现的位置，通过未解析实体引用调用它，例如：

```
<picture name="picture1"/>
```

需要注意的是，实体引用中不包含“&”：解析器知道这是一个实体引用，因为它已经在DTD的ATTLIST声明中进行了适当的声明。

最后，如果你认为上述方式过于繁琐，可以仿照 HTML 使用 URL 链接：

```
<IMG SRC="picture.gif"/>
```

对于 XML 解析器来说，这仅仅是一个普通的 CDATA 属性，但是在应用程序级，你可以将它解释为指向 GIF 文件的 URL。

需要强调的是，在我们介绍的技术中，虽然只有一种技术（未解析实体和表示法）使用了特殊的 XML 机制处理二进制数据，但并不意味着这是唯一被认可的方法：我们提到的其他技术也同样可行。

如果你的主要目标是将 XML 翻译为 HTML，以便在浏览器中显示，基于 URL 链接的方法到目前为止是最简单的。如果你在应用程序之间暂时性地传送数据，使用基于 Base64 编码的内部对象可能是最有效的，因为它能够避免与交叉引用的完整性相关的所有问题，例如：链接对象更新后怎么办。

当 XLink 被标准化和实现之后，就有可能在 XML 中嵌入某些二进制数据，例如：图形，它将出现在源文档中。我们将在第 8 章讨论 XLink。

5. 使用 XML 实体

在对如何将信息模型转化为 XML 设计的讨论中，我们一直未说明如何使用 XML 实体。

实际上，这没什么值得惊讶的，因为在 XML 中实体被认为是物理文档结构的一部分，而不属于逻辑结构。实体引用应该被认为实体被内嵌在引用出现的位置。

然而，实体在逻辑上有一些用途。将文档体中的部分内容放入外部实体最主要的好处在于它使得这部分能够独立于文档的其余内容而独立更新。对于那些与文档主体有着不同更新周期或更新权限的文本或其他文档部件，这种方式非常有价值。这种控制能力比由共享公共内容而产生的空间节省更加重要。对于一个复杂的 DTD，如果它的各个部分分别由不同的人控制，这种方法也同样有效。

我们在讨论信息建模时曾经提到信息所有者和信息生存周期的问题。在决定如何将文档分割为物理实体时，或许应该考虑信息模型的这些因素。

4.3 模式语言和表示法

在本章的最后一部分，我们将研究如何以书面形式或电子形式表示文档的设计，使得人类用户和软件都能够访问它。我们将介绍两种主要的模式表示法：DTD，以及各种 XML 模式建议。

我们首先阐述一下我们的目标：我们希望获得什么？

4.3.1 模式的作用

在数据库和文档领域，模式这一概念的提出已经有许多年的历史了——它或许是这两个领域少得可怜的几个共同点之一！模式的正式作用是定义所有可能的有效的文档集合；或者从反面说，它的作用是定义约束，文档除了遵循 XML 规范之外，还必须满足这些约束，才能被认为是有意义的。

在此，我们必须谨慎使用“有效性（validity）”一词。在 XML 标准中，“有效”意味着一些

特殊的规定。非正式地讲，它表示文档必须符合 DTD 中的规则。我们对于 DTD 不能表示的许多约束感兴趣，其中有些约束是 XML 模式也无法表达的——例如，一条消息开始处的序号必须比前一条消息的序号大一。本章后续部分出现的“有效”一词与 XML 中的“有效”含义不同，它是面向用户的：如果文档能够满足信息模型定义的所有约束，就认为它是有效的。

1. 为什么需要模式

以下论述摘自 W3C XML 模式建议的草案：

模式的目的是定义和描述一类 XML 文档，它使用 [markup] 结构约束和说明各个组成部分的含义、用法和关系，这些组成部分包括：数据类型、元素及其内容、属性及其内容、实体及其内容，以及表示法。模式结构还能够提供附加信息的规范，例如：缺省值。模式试图通过通用的文档词汇表说明自身的含义、用法和功能。因此，XML 模式结构能够为各类 XML 文档定义、描述和归类 XML 词汇表。

我们认为以上论述说明了模式的功能，但并未说明原因。通过这段话，我们总结出模式有以下两个作用：约束和解释。

2. 模式的约束功能

模式的作用之一是定义有效文档与无效文档之间的差别。规则的表述应该尽可能使软件能够判断一个文档是否有效；但是在实际应用中，有些规则可以是只有人类才能够解释的。例如，科学杂志的一条规则规定作者的地址必须只能包括城市和国家，或者摘要必须是法语的。

之所以需要这些约束，有以下两点原因：格式上的原因（每个出版物都要维护自己的品牌图像、样式和设计的完整性）和处理的原因。处理的原因定义了处理过程的下一阶段（即：处理文档）对信息的需求，无论这个过程是个商业过程（如：处理职位申请表）、印刷过程，还是内部系统过程（如：更新数据库）。在这两种情况下，约束都可以看作是一种品质控制。

当然，约束并不总是好的。这是一种巨大的诱惑，你可能毫无顾忌地利用规则的约束能力对系统进行过度严格的限制。信息系统常常不够灵活，我们的目标应该是明智地使用约束，使得处理活动中的人能够最大限度地发挥他的聪明才智。如果使用过度，约束本身会对信息的质量产生负作用：我过去不得不给电子商务系统提供错误的数据，因为这个 Web 页面坚持我的地址必须位于美国。

另外，我们的确能够定义明确的规则，并用软件进行检查，但是这并不意味着在可以想象得到的每个处理阶段都利用规则验证文档的有效性。例如，当你从 Web 服务器向外传送文档时对它进行检查就是多余的：当它被放到 Web 服务器上时应该检查有效性。另外，有些人盲目地毫无顾忌地使用验证有效性的解析器。类似地，当 XML 文档从一个软件系统发送到同一组织机构中的另一个软件系统时，在测试阶段有必要验证其有效性，但是当一切正常运转之后，就不必重复地验证有效性，因为软件之间应该相互信任。

3. 模式的解释功能

模式的第二个作用是解释——记录结构的说明和用法，使得发送方和接收方能够对消息有相同的理解。

在文档和数据库领域，模式的这个作用都是次要的，虽然它实质上可能更加重要。部分原因在于模式总是不能提供给需要它的人，即：在屏幕上输入数据的人。这就是为何许多系统都

受到一种称为语义漂移的现象的困扰，随着时间的推移，用户有改变系统使用方式和更改数据字段含义的倾向，即使软件结构并未发生变化。在我遇到的一个例子中，用户故意输入会遭系统拒绝的数据，因为有待纠正的记录文件能够方便地记录明天要打的电话。在另一个系统中，我发现通讯社以发送虚假的新闻文章为幌子向客户发送发票。

你不能阻止这些事情的发生，正如建筑师不能阻止居住者在厨房里看电视。你能所做的是尽量解释你所提供的结构的用途，并为用户提供灵活的空间，使得他们在不违背你的设计的前提下达到自己的目的，而且使你的结构尽可能地直观和自然，以免用户产生被强迫走另一条路的感觉。

端用户不必直接了解模式，这也意味着模式应该是应用程序可读的：例如，应用程序应该能够提取数据字段的解释，并将这些解释作为提示文本显示在输入数据的屏幕上。七、八十年代，模式的这种观念非常流行，它出现在数据字典中：数据字典强调模式在定义系统词汇中名称的含义方面的作用，虽然软件公司对数据字典的滥用（他们将数据字典做成内部系统目录）导致它最终被淘汰。

4.3.2 将DTD作为模式

如果模式的作用是约束和解释，那么 XML DTD的局限性就显而易见了。它主要体现在以下几方面：

(1) DTD能够表示哪些约束

作为一种约束语言，DTD是非常有限的。它能够控制元素的相互嵌套，却对元素中包含的内容无能为力。它对属性的控制略微强一些，但即使如此也是非常有限的，例如：DTD无法规定某个属性值必须是数字。这必然意味着许多实际的有效性验证工作必须由应用程序来完成；事实上，我常常发现DTD的验证功能基本上没什么价值，不值得使用。

(2) 你能够依赖DTD的有效性验证吗

即使在DTD能够定义的有限的规则中，XML文档仍然可以设置自己的议程：是否引用DTD，引用哪个DTD，以及是否用私有的内部子集中的声明覆盖DTD中的声明，这些都是由文档本身决定的。即使应用程序知道将使用验证有效性的解析器，文档乐于接受哪些约束也是由文档本身来决定的，应用程序不能将自己的意愿强加给它。

假设你为旅游路线定义了DTD——itinerary.dtd。为了将旅游路线邮寄给客户，你的应用程序要负责打印旅游路线。如果应用程序不希望由自己来验证有效性，它需要确保输入的文档符合名为itinerary.dtd的DTD。为此，它要保证：

- 它所使用的解析器是能够验证有效性的。（这一条是不言而喻的，但是你的应用程序是使用SAX、DOM还是XSLT，就很难判断了。）
- 文档的<!DOCTYPE>声明中包含对itinerary.dtd的引用。（遗憾的是，大多数标准API都不提供该信息。）
- 解析器使用的是应用程序认为正确的itinerary.dtd版本（这很难做到）。
- 输入文档中的文档元素具有正确的元素类型（DTD并没有说明哪个元素应该是顶级元素，但幸运的是，应用程序很容易进行这项检测）。

- 输入文档的内部 DTD 子集没有覆盖外部 DTD 中关键的有效性规则。举例来说，如果外部 DTD 子集包含以下定义：

```
<!ELEMENT PAYMENT EMPTY>
<!ATTLIST PAYMENT MEANS (check | credit-card) #REQUIRED>
```

则应用程序自然会认为在有效的输入文档中，<PAYMENT>元素应该包含 MEANS 属性，且属性的值是字符串 “ check ” 或 “ credit-card ”。但是，如果输入文档的内部 DTD 子集包含以下声明：

```
<!ATTLIST PAYMENT MEANS (check | credit-card | cash ) #IMPLIED>
```

它将覆盖外部 DTD 中的声明。因此，MEANS 属性可以被省略，或者取值 “ cash ”，解析器仍然会认为输入文档有效。

(3) 编写 DTD

DTD 和 XML 采用不同的语法，因此它不仅难读，而且更难写，它采用的定界符与 SGML 完全不同。DTD 对元素和属性含义的解释作用如同用编译后的 Java 字节码说明商业过程：语法中甚至规定有些地方不能插入注释。

市场上的 DTD 编辑工具能够简化 DTD 的编写，但是归根到底，如果想完全理解 DTD，就必须学习它的语法。DTD 不仅不方便，而且结构上比较笨拙，特别是不能像解析 XML 文件那样解析 DTD。

(4) DTD 和命名空间

最后一个问题是 DTD 几乎不能与 XML 命名空间配合使用。命名空间使你能够在文档中混合来自多个信息模型的元素，例如在关于污染扩散程度的文档中同时使用化学公式和地理编码。你能够选择名称的前缀，而不会改变元素的含义，例如：你可以用 <GEO:LI> 代表立陶宛，用 <CHEM:LI> 代表锂。

DTD 和命名空间的问题主要体现在两方面。首先，文档只能引用一个外部 DTD。其次，通过添加前缀给元素重命名后，将导致 DTD 无法识别，因此如果你确实希望将二者相结合，实际上每次都要创建新的 DTD 版本。

尽管 DTD 存在这些局限性，但是它是目前唯一的标准，因此我们将介绍如何根据信息模型创建 DTD。

根据信息模型创建 DTD

我们已经了解了如何将信息模型中的概念转化为 XML 文档的设计，其中的某些决策将直接反映在 DTD 中。例如，选择将对象的特性表示为元素或者属性。

但是当你真正开始编写 DTD 时，有些问题才能够明朗，下面我们将介绍这些问题。

虽然 DTD 的建模功能相当有限，但是许多功能都能够通过非常灵活的参数实体机制获得。参数实体可以实现 DTD 中不同定义之间的文本共享，除此之外它还有其他用途，DTD 的效率在一定程度上取决于使用参数实体的技巧。

大多数 DTD 都包含两种类型的定义：元素定义和属性定义。元素定义规定了元素的内容；属性定义规定了每个元素中能够出现的属性。我们将依次介绍这两种定义。

(1) 定义元素内容

DTD支持以下五种元素结构定义（参见表 4-3）。

表 4-3

内容模型	举 例
EMPTY内容	<!ELEMENT confirmed EMPTY >
ANY内容	<!ELEMENT description ANY >
元素内容	<!ELEMENT payment (currency?, amount, date, mode-of-payment?) >
复合内容	<!ELEMENT estimated-cost (#PCDATA note)* >
PCDATA内容	<!ELEMENT color (#PCDATA) >

实际上，XML规范仅标识了四种结构：根据语法规定，PCDATA内容只是一种特殊的复合内容，但是从建模的角度看，这两者有很大差别。下面让我们看看如何使用每种结构。

(2) EMPTY内容的元素

从本质上讲，EMPTY元素代表一个布尔值：它可能出现，或者不出现。如果你希望标记特定的预约已经被确认，可以使用空的<CONFIRMED/>子元素来表示。

如果打算用属性而不是子元素表示对象的特性，EMPTY元素也是非常有用的。（注意，EMPTY意味着它没有子元素或文本节点；但并不表示它没有属性。）当选择这种方法时，会发现文档中几乎所有元素都是空的：文档可能包含几千个这种形式的元素：

```
<payment from="1234" to="5678" amount="230.45" date="1999-10-15"/>
```

如果你希望构造一个特殊的文档，它仅包含一个XML元素，且该元素没有子元素，只有许多属性：对于某些类型的消息来说，这种结构是非常合理的。

另外，还可以使用EMPTY元素表示枚举数据类型的值（这种方法的确不太常见）。例如，可以编写以下代码：

```
<TRAVELER>
  <GENDER><FEMALE/></GENDER>
</TRAVELER>
```

GENDER元素的声明可以采用元素内容结构的形式指定有效值列表，例如：

```
<!ELEMENT GENDER ( MALE | FEMALE ) >
```

(3) ANY内容的元素

在由信息模型产生的DTD中，不太可能出现ANY选项；相反地，如果部分信息模型未知，常常会导致DTD中出现ANY内容的元素。实际上，它并不允许一个元素包含任何类型的子元素，只有DTD中定义的类型才能作为子元素出现，因此它等价于列出DTD中的所有元素类型。因为有些元素类型在此出现是没有意义的，所以这仅仅是懒惰的表现：最好使用元素内容或复合内容，并列出现在这种环境下有意义的所有元素。

(4) 元素内容的元素

元素内容定义了哪些元素可以作为被定义元素的子元素，而且定义了它们出现的顺序，是否必须出现，以及是否允许重复。在所有选项中，元素内容使得设计者能够对有效性进行最精确的控制。考虑以下定义：

```
<!ELEMENT RESORT (
```

```
NAME,  
COUNTRY,  
REGION?,  
HOTEL+)>
```

以上声明规定 RESORT 必须有一个名称，必须位于一个特定的国家内，它可以在一个特定的地区，并且必须包含一个或多个旅馆。根据语法规则，子元素可以以空白分隔，但是按照习惯，空白是没有意义的。

使用这种声明最大的缺陷是它不仅说明了能够出现的元素及出现的次数，而且隐含地规定了元素的次序：如果 DTD 中包含以上声明，则当国家出现在旅游胜地的名称之前就被认为是无效的。列出二十个可选的且不重复的子元素，却不限制它们出现的顺序是不可能的。这到底会不会成为一个问题取决于文档是如何产生的。对于由人编写的文档，如果仅仅为了让 XML 解析器验证有效性就要求文档的作者按照指定的顺序排列所有子元素，这显然不太合理。

(5) 复合内容的元素

复合元素内容是文档中最常见的结构，在这种结构中，你可以利用标记以文本形式标志特定的单词。标记当然应该是语义标记，而不能是纯表示标记，例如：

程序清单 4-6

```
<resumé>  
  <languages>  
    I am fluent in <lang>English</lang>, <lang>Spanish</lang>, and to  
    a lesser extent in <lang>Chinese</lang>.  
  </languages>  
</resumé>
```

根据我们在本章描述的这类正式的建模方法不太容易产生复合内容结构——这并不意味着这种结构不好。相反，它是一种非常出色的表示信息的方式，特别是对于那些比较模糊的信息，我们希望捕获自然语言能够表达的微妙的有些模棱两可的含义。有时，试图将所有事物都规格化和代码化是不恰当的。

复合元素内容不仅能够用来表述事实，而且能够说明事实的特性。同样，在大多数流行的信息建模方法中没有复合内容的概念，虽然它有许多实际的用途，特别是在医疗记录、历史研究及假货调查等方面，我们不仅要记录结论，如：“Thomas Wilson 铅中毒”，而且要记录与事实相关的观察资料：谁在何时下的结论？有什么证据？有多大把握？有人持不同意见吗？它将产生以下类型的结构：

程序清单 4-7

```
<weight>182.3  
  <plus-or-minus>0.5</plus-or-minus>  
  <measured-by>John Smith</measured-by>  
  <date-of-measurement>1991-10-15</date-of-measurement>  
  <instrument>567421</instrument>  
  <alternative-value>186.1  
    <measured-by>Mary Jackson</measured-by>  
    <date-of-measurement>1993-01-21</date-of-measurement>  
  </alternative-value>  
</weight>
```

复合元素内容并不非常适合这种目的，因为它总是允许任何子元素以任意顺序出现任意多

次，特别是它允许多个文本内容单元。除了复合元素内容之外，还有几种更好的可选方案：一种方法是将“事实”表示为元素，将“事实的特性”表示为这些元素的属性；另一种方法是为文本内容设定另一级元素，这样上述例子将变为：

程序清单 4-8

```
<weight>
  <value>182</value>
  <plus-or-minus>0.5</plus-or-minus>
  <measured-by>John Smith</measured-by>
  <date-of-measurement>1991-10-15</date-of-measurement>
  <instrument>567421</instrument>
  <alternative-value>
    <value>186.1</value>
    <measured-by>Mary Jackson</measured-by>
    <date-of-measurement>1993-01-21</date-of-measurement>
  </alternative-value>
</weight>
```

但是如果出现大量的没有注释的所谓“事实”元素，且只有一个值，将会难于处理。

(6) PCDATA内容的元素

最后，我们将介绍PCDATA内容。PCDATA元素是构成文档的原子，更高级的结构是以它为基本组成单位的，如果你用元素表示特性，则大部分元素都将是PCDATA元素。

(7) 对象类型层次建模

在XML中，没有明显的类型层次概念，但是可以用参数实体仿真它，有经验的DTD设计者在潜意识中已经形成了这一观念。

在专用类型的定义中，总是要在派生它的类型的定义中添加一些特殊的内容。例如，对象类型<refund>是类型<payment>的专有化形式，它在<payment>定义的基础上增加了<reason>和<authorized-by>等属性。如果我们在DTD中使用参数实体定义类型，这种扩展能够以非常自然的方式来表达。

下面的例子使用子元素表示特性：

程序清单 4-9

```
<!ENTITY % payment "amount, date, account, notes?" >
<!ELEMENT payment ( %payment; ) >
<!ENTITY % refund "%payment;, reason?, authorised-by" >
<!ELEMENT refund ( %refund; )
```

(8) 当几个特性的数据类型相同时

在前面对建模的讨论中，我们曾经提出这样一个问题：如何命名用于代表两个独立特性Billing-Address和服务-Address的元素。对于这种情况，你常常会创建几个有相同数据类型的元素类型，即：这些元素类型有相同的有效性规则。同样，在DTD中，你自然可以用参数实体表示最基础的数据类型：

```
<!ENTITY % address "address-line+, postcode, country" >
<!ELEMENT billing-address ( %address; ) >
<!ELEMENT service-address ( %address; ) >
```


(9) 在DTD中定义属性

当你开始定义属性时,可供选择的余地已经不大了,它们几乎完全是按照信息模型构建的。

确定一个属性是可选的还是必需的一般非常简单。决定是否要为属性设定缺省值通常也要从实际角度出发——它主要取决于当文档的作者对属性值不关心时,你作为文档的设计者是否认为有必要确定所用的值。当然,当你扩展 DTD使之包含前一版本没有的属性时,缺省值非常重要。

从事数据库设计与开发的人一般不习惯 DTD中FIXED属性值的概念(对于在 DTD中声明为FIXED的属性,它的每个实例都有相同的值),但是它确实是一种功能非常强大的机制。它可能用于以下情况:

- 采用类似于Java中静态字段的用法,用它们表示元素类型的属性,使之区别于实例的属性。例如,可以使用固定属性标识元素的数据类型:因此 date-of-birth字段可能有固定的属性 datatype="date";或者可以利用固定属性命名用于验证元素实例值的 Java类。
- 使用它们标识超类。举例来说,如果使用许多不同的元素标记记录客户一生中的不同事件(例如:open-account,close-account,start-session,make-order,cancel-order或make-payment),你可能需要某些程序分析特定客户的所有事件。你不必对表示事件的元素类型列表进行硬编码,而是可以给每个元素类型设置一个固定的属性 is-event="yes",这样你的程序就能够使用该属性选择相关的属性。例如你可以在 DTD中写入以下代码:

```
<!ELEMENT CLOSE-ACCOUNT ...>
<!--ATTLIST CLOSE-ACCOUNT is-event FIXED "yes"-->
```

然后,在用XSLT编写的应用程序中,可以处理属于事件的所有元素类型:

```
<xsl:for-each select="*/@is-event='yes'">
```

- 用于目前是常量,将来可能变化的属性:例如版本号。
- 与条件部分或参数实体配合使用,提供仅在程序的特定运行中固定的属性。例如,可以在外部参数实体中定义名为 access-rights的FIXED属性,它根据用户取不同的值。通过使用以下条件部分,可以在XSLT应用程序中筛选出允许用户看到的数据:

```
<xsl:for-each select="data[@access-rights &gt;= @security-level]">
```

(10) 选择属性类型

CDATA有倾向成为所有属性的缺省类型,因为它允许任何字符串作为属性值,然而实际上多数属性很可能都符合NMTOKEN或NMTOKENS类型。NMTOKEN是由一个或多个字母、数字或某些标点符号(包括.和-)构成的序列,NMTOKENS是由空白分隔的NMTOKEN值列表。因此:

```
1999-10-31
```

是有效的NMTOKEN值,另外:

Defining attributes in the DTD

是有效的NMTOKENS值。通过使用这些类型而获得的额外的有效性是微乎其微的,但是它能够让解析器来规格化空白,这也是一项非常便利的特征,特别是对于用 XSLT编写的应用程序。

就元素结构而言，DTD中的参数实体是一种表述许多不同元素的属性或属性集合共性的有效工具。

4.3.3 XML模式建议

前一段时间，W3C已经意识到DTD的局限性，为了寻求解决方案，它花费了大量时间，部分原因在于已经形成的几个重要建议似乎都不能完全解决这个问题。其中颇具影响的一个建议是Microsoft专有的XML数据规范，Web站点<http://biztalk.org/Resources/schemasguide.asp>介绍了这个规范。

XML模式工作组在1999年11月5日同时提出了两个草案：

- 第一部分，结构（<http://www.w3.org/TR/xmlschema-1>）说明了如何控制和描述文档的结构化规则。
- 第二部分，数据类型（<http://www.w3.org/TR/xmlschema-2>）描述了内容项的数据类型的定义。

这些建议尚未完成，许多部分都处在制定过程中，我们强烈建议你不要使用当前形式的规范。然而，这些建议的确提出了一些将在最终规范中出现的概念，这些概念非常值得研究，即使没有XML模式软件，对于你这个文档设计者来说，这些概念也是非常有价值的。

XML模式中最基础的概念是：模式是一个描述（和约束）一组XML文档实例的文档。XML模式本身就是一个XML文档：这一点非常重要，因为这意味着XML应用程序（例如：XSLT样式表）很容易查询模式。SQL程序员对这个概念并不陌生：它类似于在系统表中放置可以通过SQL访问的SQL模式。它还允许你利用仅供应用程序使用的信息修饰模式——例如，你可以给每个元素类型添加相关的机密等级，并在应用程序中利用该信息决定应该向特定的用户显示哪些信息。

目前处于草案状态的XML模式分为两个规范。第一部分结构，侧重于对元素结构的约束。它介绍了原型（Archetype）的概念，从本质上讲，原型是个复合数据类型。元素类型可以声明为符合特定的原型，其中原型定义了所有的约束，例如：它能够包含哪些子元素和属性。将原型定义与元素定义相分离的优势在于几个元素类型能够使用相同的原型（Billing.Address和Service.Address可以都符合原型Address）。它对应于DTD中参数实体的主要用法之一。

XML模式工作组还打算提供一种机制，使得一个原型能够精炼另一个原型，这是实现类型层次所必需的。然而，在当前的草案中，满足这种需求的语法尚未定义。

规范的第二部分讨论了数据类型，它包含的基本数据类型有：字符串、数字、布尔值和日期，除此之外，它还规定了如何通过组合或限制现有的数据类型或者通过枚举值的方式定义新的数据类型。这些数据类型可以用于约束属性的值或元素的文本内容。内置的数据类型包括了现代编程语言或SQL中常见的类型，另外，它提供的用于定义最小值和最大值等约束的机制比其他大多数语言更加强大。规范还保留了“遗留的”XML数据类型，例如：NMTOKENS。它将数据类型的值的定义与XML文档中词法的表示严格区分开，因此相同的值可能有多种不同的词法表示（例如：3，3.0和03.00）。与第一部分类似，数据类型的定义与使用该类型的元素和属性的定义是分离的，因此许多不同的元素和属性可以共享相同的数据类型。

就XML模式目前的状态而言，仍然有许多完整性约束是无法用模式表达的，必须由应用程序来限制。例如，目前无法定义属性之间的约束（逝世日期必须在出生日期之后），它只能定义对直接子元素的约束（例如：“在字典条目中，词目或词目的至少一个变种必须有词源”）。实际上，有些有效性约束只能在应用程序级完成，正如关系数据库中更加成熟的约束定义。

我们将在第7章详细介绍XML模式。

4.4 小结

在本章中，我们首先讨论了一些基本的信息建模原则，它们是在XML项目中经常用到的，我们特别区分了静态信息模型（理解现实世界中的事物和关系）和动态信息建模（理解在业务过程中从A到B需要获取哪些信息）的作用。这两种模型都与XML相关，将XML文档分为保存静态永久性数据和保存暂时性消息这两种类型是非常意义的。

随后，我们介绍了如何将概念化信息模型转换为XML文档的设计。我们分析了一些必要的设计决策：如何表示类型层次，使用元素或属性，以及如何将二进制属性编码。

最后，我们讨论了如何在XML DTD或模式中表达设计，这两者都规定了正式的约束，以便对文档实例进行自动检测，并将文档的含义传达给要创建文档和处理信息内容的人。