

搞懂 XML，看清 SOAP

攻占 BizTalk

作者：**Brian E. Travis**
译者：**纪学勤**

出版日期：2001/02/21
书号：957-2085-20-4

推荐序

写这个序，最让我头痛的是该如何不要从因特网开始谈起。感觉上 Internet 或是因特网这些字眼已经用的泛滥了，再用这样的字眼，会让我的序显得平常无奇。

不过看来是无可避免，因为这本书是一本很 Internet 的书，而且是一本很先进的 Internet 书。如果您所从事的开发工作必须面对因特网，或者你开发的是因特网上的应用程序，您真的得好好看看这本书。因为这本书所谈的主题，绝对会在未来的几年内成为因特网上的主流（还是说现在已经是主流）？

XML，如果您说没听过，那我得怀疑您可能是远古时代的开发人员。

SOAP，您可能听过，而且还以为这是微软专有的技术。如果您真是这么想，您错了，而且错得离谱。如果您没听过，我建议你最好赶快研究一下，不然您很快就会变成远古时代的开发人员了。

至于 BizTalk，您可能真的没听过，承认没听过不用不好意思。就算您听过，也可能以为这是一个微软的产品。事实是：它不只是一个产品，而是一个技术，一个架构。所谓的产品，不过只是其中的一部份而已。

所以这三者之间有什么关系？

能不能看一本书就弄清楚这三者的重要性，并且充份掌握一切的来龙去脉？甚至直接进行开发的工作，在目前的解决方案中套用这些精彩的技术？

行！

就看这一本「搞懂 XML、弄清 SOAP、攻占 BizTalk」！

至少对我而言是如此！

这本书透过浅显的例子，清楚的说明这三项技术的重要性，并且对每一项技术的由来做了完整的交待。也许您的工作是开发团队的主管，或者是技术顾问人员，而您只是想了解这些技术有多重

要，并且知道怎么样去唬人，您可以直接跳过程序代码，只看各章说明的部份。如果您是专业的开发人员，很想知道如何善用这些技术在您的程序当中，那书中的范例绝对能让您快速的掌握其中的精神。

如果您确定未来几年您的开发工作不会和因特网有任何关连，建议您现在就把这本书放下，它不适合您。

如果您从来就不打算成为专业的开发人员，开发程序对您而言只是玩票而已，建议您省下买这本书的钱，它绝对不是您要的书。

如果您不是上述两种人之一，不要犹豫，这绝对是您案头必备的一本好书，它会影响您未来几年的职场生涯（这样说会不会太夸张了）！不向您大力推荐，我会很难过。

其它的我就不多说了，本书的作者和译者（我相信纪君在翻这本书时一定花了很多心力，兼顾译文的趣味和正确是件高难度的工作，我经历过这种过程）会让您明白我的感受。

彭靖灏

资深经理

图书出版部

学习事业群

华彩软件股份有限公司

译者序

身为信息图书出版社的技术编辑，对于「书」有着特别的情愫；对于「技术」有着执着的热爱；对于「新技术」更是显得雀跃不已。第一次看到这本书的时候，它显眼的书名立即吸引了我的目光－「XML、SOAP、BizTalk」，相信对于读者您来说，这三个要素一样也会是您所注意的焦点。

在翻译的过程中，着实遇着了不少的问题，原因是因为本书所涉及的技术公开标准现在仍然处于不断的更新状态，甚至包含涉及的相关软件也遇到了同样的问题。本书的原作者在书中对于版本的问题加上了许多相关的说明，您可以至相关的网站上找寻该技术的最新信息。如果您对于书中的某些技术层面存有质疑，或许是因为版本的差异性所致。本书分别针对书名中的三大主题作详细的探讨，分门别类的介绍其历史沿革、基础结构、运作机制等等。在书中的最后一章特别介绍了 Microsoft 所推出的 .NET Enterprise Servers 之一的 BizTalk Server 2000，您可以从该章的范例中清楚的了解到 BizTalk Server 2000 的强大功能性以及其便利的工具。

.NET 名词问世以来，许多人对于「.NET」是为何物存着许多的疑虑，希望本书可以为您开启 .NET 的大门。其实 .NET 的底层技术当然不止于本书所述，但是我由衷的希望读者可以藉由此书得到您想学的知识。毕竟，技术是辛勤的学习所获取的知识，不是吗？

当然本书也不是我独力一人可以完成，一本书的诞生绝对是来自许多人的心血结晶。在这里，我要感谢华彩谢慕萍副理对我的大力支持，提供我有翻译本书的机会，同时本书诞生的过程中，也给予我至多的辅助，让我可以专心致力于翻译的工作，同时也感谢参与本书诞生过程的每一个人！

纪学勤

2001.01.18

本书简介

延伸标记语言（eXtensible Markup Language，XML）－这个简单的技术在计算机世界带动了另一场革命。在 XML 被正式采纳之前，Microsoft 的总裁比尔盖兹曾经说过：「XML 是计算机技术上的一大突破」，因为他看到了愿景－以一种开放、跨平台的方式来定义数据交易的价值。自从他说过那句话之后，许多的数据交换标准都以 XML 技术为基础建立，每一种标准都有其特定目的，并且都使用 XML 的简单语法。

BizTalk 和简单对象存取协议（Simple Object Access Protocol，SOAP）都是基于 XML 的开放式标准，它们的设计是为了促进在 Web 上系统之间的电子商务开发，BizTalk 和 SOAP 都是使用 XML 为数据交换的语法（因为 XML 让您可以使用十分精确的方式指定数据）。

笔者之前的著作－《The SGML Implementation Guide》在 1990 年代中期出版，是一本专门供 SGML 使用者所参考的书籍。当 XML 问世后，笔者试图更新其著作以兼容于 XML 的规格，但那是一件相当困难的事。因为 XML 简单、具弹性，而且规模非常小，XML 甚至比 SGML 有更丰富的功能性。

后来 BizTalk 和 SOAP 又相继问世，这两种标准可以建立许多的 XML 应用程序，于是笔者撰写了本书。

BizTalk 和 SOAP 有许多相似之处，然而也有许多不同的地方。本书将会相继介绍许多实际的范例以让您了解其中的分野。

BizTalk

BizTalk 是由 Microsoft 创先提出，其目的是为了在电子商务合作伙伴之间移动商业文件，BizTalk 提供了一种简洁的方式以在 Internet 上移动电子文件，就像邮件服务在商业伙伴之间传送实体信封一样。

BizTalk 总共分为四个部分，第一部份就是 BizTalk Framework Independent Specification。BizTalk Framework 中定义了卷标的集合，您可以使用这些卷标来标记您的文件，藉由这个方式可以从某个位置传送文件至另一个位置，然后分别在不同的位置进行不同的文件处理方式。BizTalk Framework 的工作方式就像一个虚拟的信封，用以传送商业文件给合作伙伴。

第二部份是 BizTalk.org 网站，该网站专门提供相关电子商务的技术，例如 XML 和 BizTalk 等等。BizTalk.org 中有讨论区提供点对点的支持以辅助您开发自己的文件语法、架构和整合您的文件至 BizTalk Framework 中。BizTalk.org 同时也是架构（XML 文件的定义）的储存区，并且允许您上传架构以供其它人存取，该服务是免费的。

第三部份是处理您的 BizTalk 文件，您必须使用 BizTalk Framework Compliant（BFC）伺服程序以达成这个目的。BFC 伺服程序是读取 BizTalk 文件的软件，并且会对其读取的文件进行特定的处理，BFC 伺服程序的优点就是它会依照您所使用的系统不同而分别进行特定的处理。

第四个部份就是 Microsoft BizTalk Server 2000（需要执行于 Microsoft Windows 2000 Server），因为 BizTalk Framework 的开放式规格特性，您可以在 Linux 系统上以 Java 撰写 BizTalk 伺服

程序以建立 BizTalk 讯息，并且使用执行于 Microsoft Windows 2000 Advanced Server 上的 BizTalk Server 2000 读取该讯息。

BizTalk 看起来好像需要在 Microsoft 的平台上才能够执行，然而事实并非如此。刚刚所提到的前三点都不专属于 Microsoft，两个 Java 的应用程序能够来回的传送 BizTalk 讯息；执行 UNIX 系统的企业也可以发布架构至 BizTalk.org。BizTalk 伺服程序可以使用 Perl 或任何其它 Internet 相关的语言撰写。

SOAP

SOAP 允许您透过 Internte 通讯协议（HTTP 和 FTP 等等）在分布式的系统中呼叫远程的对象，SOAP 和 BizTalk 的相似之处在于都是用来交换信息，但是它们也分别为不同的交易类型进行最佳化。

SOAP 是 Web 服务（Web Services，Microsoft 首先倡导）的核心，Web 服务用另一种角度来观察 Web。在使用 HTML 浏览器的时代，使用者只是等待 Web 网站下载固定的信息，但随着越来越多的需求，网站内容的提供者必须提供更丰富的个人化信息以满足使用者。藉由移除数据的格式化需求，并将数据以原始的型态传送，然后 Web 服务就可以重新汇总和处理原始数据，这种方式可以提供无限的变化以满足使用者，增加信息的附加价值。

就像 BizTalk 一样，SOAP 也是可以跨平台的技术，您可以传送一个 SOAP 方法呼叫给执行于 Solaris 上的 Perl 指令码，然后 Perl 指令码会依次呼叫 Java 以从 Oracle 数据库中取得数据。响

应以 XML 文件对象方式传送，然后您的应用程序就可以使用这个对象，用这种方式您并不需要知道服务是执行于何种平台上。

要了解 BizTalk 和 SOAP，您首先需要了解底层的语法－XML。事实上，BizTalk 就是 SOAP 的延伸，您可以使用 SOAP 以透过 HTTP 传递商业文件。当您使用 BizTalk 时，您同时也在使用 SOAP 和 XML。

本书的第一篇是以商业和技术的观点来介绍 XML，目的是为了提供您使用 XML 结构建立和交换商业信息的未来展望。

这也是笔者一直很想撰写的 XML 书籍类型，纵使您根本不想学习 SOAP 和 BizTalk，也可以从第一篇中 XML 的说明得到一些宝贵的知识，这些知识对于程序开发师也是很有帮助的。

本书的第二篇会说明 BizTalk Framework，并且告诉您如何使用 XML 和 BizTalk 建立电子商务应用程序；在第三篇，我们将会使用 Microsoft 和非 Microsoft 的工具实际建立数个 BFC 伺服程序。

SOAP 和 Web 服务能够真正改变人们生活和工作方式的技术，而 BizTalk 将可以提供企业组织一种标准、有效率和低成本的方式进行交易，并且取代传统的纸上交易。

第 1 篇 XML 对电子商务的意义

1. XML 的商业前景

. 商业对商业（B2B）的电子商务

. 将处理过程和数据分离

. XML 和三层式的 Web 结构

. XML 对于内容管理的意义

. XML 对于内容收集的意义

. 电子文件交换的背景

. XML 对于电子文件交换的意义

2. XML 的背景资料

. HTML－优良的素材

. HTML：不好的素材

. SGML－优良的素材

. SGML：不好的素材

. 我们都需要 XML

3. XML 标准

. ISO

. OASIS

. W3C

. IETF

. 开发产业架构

4. XML 语法

. 格式正确的 XML 文件

. 有效合法的 XML 文件

. 名称空间

. 架构

5. XML 应用程序

. 文件对象模式

. 绝佳演练：在 Visual Basic 中使用 DOM

. 写入 DOM

. 事件驱动模式

6. XSL

. XSL 的演进背景

. XSL 如何工作

. XSLT 操作程序

. 绝佳演练：商业文件转换

1. XML 的商业前景

电子商务正在日益成长中，而且变得越来越重要。但是何谓电子商务？对某些人来说，它表示在 Web 网站上使用信用卡；对另一些人来说，它代表在 Web 网站上有购物车存在；但是对笔者来说，电子商务代表未来世界人类的互动方式。大部份的人都知道电子商务就是「电子式的商业交易」。当然，在商业行为中，交易是最有魅力的过程－人们各取所需。

人类已经交易了数千年。从文明世界的初期开始，人们面对面接触交易；而现在藉由科技的帮助进行交易，这表示商业交易过程已经从面对面的方式转变为利用其它手段完成。

旧时代的人们利用快递通讯，来进行距离遥远的双方之间的商业交易，长达数世纪之久，但是如果牵涉到重大的交易，还是需要面对面交易来达成。

十九世纪中期，美国藉由复杂的网络邮件传递系统（Pony Express）取得优势，使得人们可以进行远程的商业交易。这个改革变更了人们的消费方式，并且因为消费者有更多的选择，使得完成交易的成本更为便宜。

以土地开发来说，火车促进了美国西部的物流通畅，而密集的邮件传递网络也降低了成本，让更多的消费者和卖商之间有更多的互动。使用邮件方式的商务产生后，Sears、Roebuck 和 Company 便运用了邮购方式，以少量的成本完成了订购和交货过程。一旦人们信任某公司的商誉，面对面的互动交易就不再需要了。

然后是电话时代来临，人们拿起电话就可以订花，完全屏除了纸上作业。这种称为电话的电子式系统所发生的商业交易何尝不能称为电子商务呢？

然而现在，您可以联机至「www.1800flowers.com」（Flowers are just the beginning），并且在这个网站购物，您甚至已经不需要担心跟他人沟通的问题，过程十分简单，就像这样：在您的计算机显示的全彩的图片中选择您想要的货品，并将它们加入您的电子购物车中。当您完成购物后，您可以在包含送货信息和信用卡号码的窗体中选择您最喜欢的付费形式，在充满愉快的情况下结束您的交易过程。

大部份的人都会将电子商务纳入进行交易的方式之一，但是其它人呢？如果电子商务是消费者和卖商的互动方式，那么在网站中提供订购价格的内容对企业来说是不是一种冒险呢？在销售交易时并没有实质的产品，但是对于付费的人来说，这些网站会提供他们有价值的产品。而每次只要有人参观网站，就代表交易过程已经开始了。

那么何谓「免费」网站，强调「免费」这两个字是因为没有任何东西是真的免费－天下没有白吃的午餐，当然这是一种隐喻。无论在任何时候某人提供您免费的东西，他一样也会从你这边取得某些东西。以 Internet 来说，网页上有如此多的信息免费供您浏览，但是当您浏览这些信息时，您同时也花费了您的精神在那一些标语、赞助商讯息、甚至广告讯息等等。这些广告商和赞助商只需要您的眼睛停留在他们的广告上，他们就认为已经有足够的价值，并且付费给那些 Web 网站的管理员以便让他们继续登载最新的广告。这也造就了电子商务，不是吗？

商业对商业（B2B）的电子商务

到目前为止提及的都是商业对客户的电子商务，或称之为 B2C 电子商务。这种电子商务类型颇受好评，但是当我们听到有关 Amazon 虚拟库存的资料仓储问题时，B2C 电子商务的新问题就浮出台面了。

但是这却只是电子商务要突破的一小部份而已，当产品在 Web 上面贩售的时候，同样的在公司之间也会有潜在的商业往来。以一支简单的墨水笔为例好了，当您从在线零售商购买一支笔时，就牵涉了一笔交易，如果零售商想要节省交易的耗费，就要藉由使用适当的技术让交易更有效率。

换句话说，笔的贩售其中牵扯了数百笔交易。笔的生产商必须购买钢铁、墨水和弹簧，以及其它数种未加工的原料；然后笔的制造商必须购买产品服务、市场服务和管理服务，这些交易中的每一个都将是一笔耗费，如果每一个节点都可以节省一些耗费，那么所有的总节省数量将会使成本到达底线。

本书焦点将放在 B2B 的电子商务，以及您如何使用 XML 来降低您的交易成本，现今有不少产业发生了一些改革性的创新，本书将提及一些创新的技术，并且把焦点置于 BizTalk，因此您可以看见典型的 B2B 是如何作业的。

将处理过程和数据分离

XML 为许多人提供了降低成本和增加收入的机会，对技术人员来说，XML 提供他们可以撷取文件中数据意义的语法；对企业中的程序设计师来说，XML 提供了在对象之间取得数据的语法；对于在商业伙伴之间建立电子商务交易系统的程序设计师来说，XML 提供了一种可撷取交易中的信息的语法，使得交易更为精确和适时。

然而，XML 对这些人都提供了至少一种共同的利益：「XML 允许他们将数据与处理该数据的程序分离」。这是一个重点，您将会常常在本书中看到这个重点被不断地重复和说明。

将数据与处理该数据的程序分离并没有什么特别的新意，一个采购订单号码在数据库是储存在一个字段中，同时也表示了一种特别的意义。这个号码可以出现在许多不同的地方：在采购订单上、在发票记录上，和在发票上等等。然而并没有数据库设计者会考量到这个订单号码要以什么方式呈现、或是在某个特定的窗体上出现。

然而，当数据要以其它方式呈现时，我们常常需要加一些格式化数据的详细信息，Word 就是众所周知的将数据格式化再加以呈现的应用程序。试想如果您在一张纸或计算机画面中看到了一个置中加粗加大的文字，您一眼就会判断它是一个章节的标题，但是为了要让计算机将其视为「章节标题」，您必须指示计算机要如何呈现，因为计算机并不具有人类判断事务的理解力。举例来说，我们也许会在文字之中看见某些以斜体呈现的文字，因为我们习惯的语言中词组是以斜体撰写。我们知道为什么以斜体表示是因为我们有辨识的理解力，并且可以指出为什么在印刷上要有区别。而计算机并没有这种能力，因此我们必须明显的指定每一个词组为斜体。

XML 提供一种允许您以清晰明白的方式定义每一个信息对象的语法，藉由这种方式，您可以将信息当成对象加以撷取，然后许多应用程序就可以依照某时间点的需求来处理这些信息。

XML 和三层式的 Web 结构

XML 适用于 Web 网站中的许多部份，在本节中，我们将讨论适合使用 XML 的网站，并且显示 XML 如何胜过其它的技术。

一个现代化的 Web 网站会提供个人化的信息给使用者，过去使用静态 HTML 网页服务客户的时代已经落伍了，这些老时代的公司发现客户最初都会造访他们的网站，但是在下次造访时如果没有新鲜的信息，就不会再来了。因为这些客户会假设该网站不会再更新了。

现代化成功的网站会提供有用的信息给使用者，因此使用者会不断的再次造访。无论任何商业理由，这种回馈方式都会使您必须不断的提供新信息给您的客户，而 XML 让您可以使用许多不同的方式提供个人化的数据检视方法。

让我们来看看使用现代三层式 Web 结构的假想网站，如图 1-1 所示，并且使用 XML 提供完美的信息呈现方式。

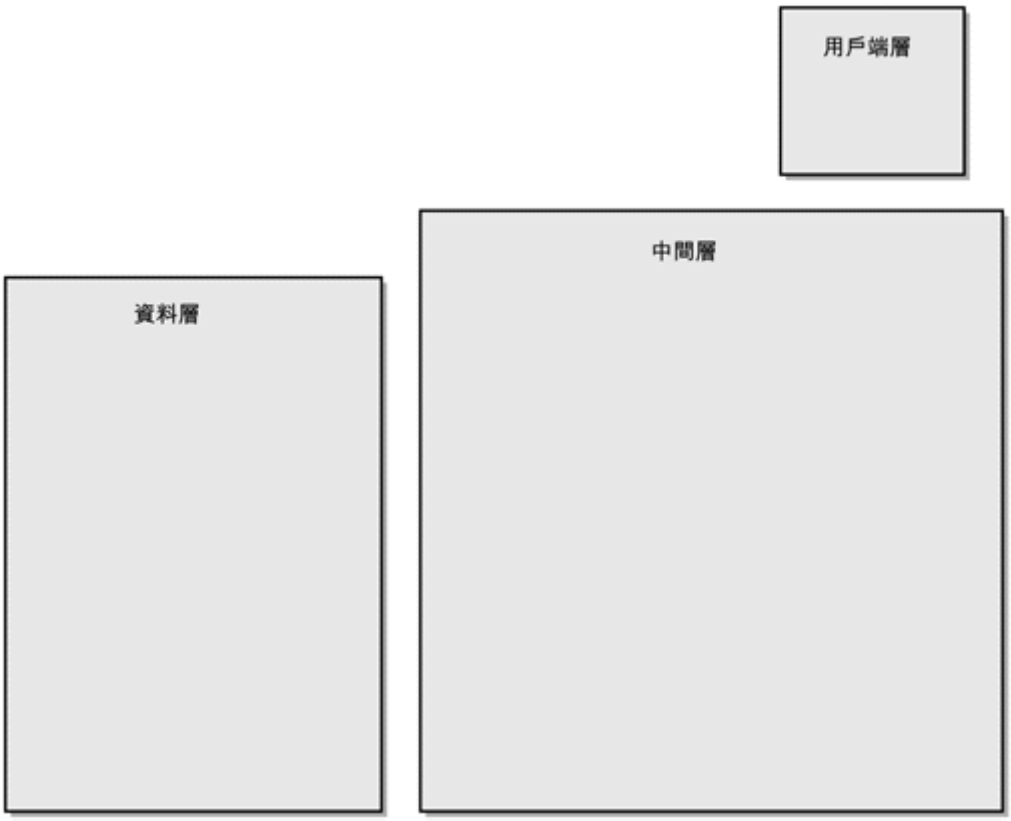


图 1-1 三层式的 Web 结构显示了数据层、中间层和用户层

三层式的结构通常也称为「n-tier」，因为在两个端点之间也许会有许多不同的层级，这完全取决于所需执行的工作。Microsoft 通常将此种类型的架构称为 Windows Distributed interNet Architecture 或 Windows DNA。

资料层

数据来源属于数据层，请注意并没有说是「数据库」，数据库是数据的来源之一，但是仍然有其它资料来源。在本范例中，我们将会把外部网站和商业伙伴当成资料来源，当笔者提及数据来源属于数据层的时候，并不一定表示数据层即为实体的计算机。它是一个概念性的观念－数据来源

是指数据可以供其它层进行处理的数据来源，数据层可以是一台计算机、多台计算机或连接至 Web 的虚拟 HTTP 联机等等。

用户层

在本范例中，我们将完全地仰赖 thin client－通常是 Web 浏览器。thin client 中的「thin」意指大部份的处理过程都于服务器上完成，并且将结果传送至浏览器输出。然而，现代的 Web 浏览器有能力作更多的处理以分担服务器的负载，当您使用 XML 传送数据至客户端时，将会看见这些额外的处理能力。

中间层

在数据层和用户层之间有一层中间层，在中间层内，来自数据层的数据会被套用商业规则，并且将结果传送至客户端。换句话说，中间层是工作完成的地方。

在中间层最普遍的组件通常是响应客户端请求以初始化处理过程的软件，该软件称为 HTTP 服务器或 Web 服务器。除了接收请求和起始中间层应用程序，Web 服务器通常不需要做什么事，这些应用程序以许多不同的计算机语言撰写，并且会套用您所需的商业逻辑以达成特定目的。

本范例使用数个伺服端的应用程序以连接数据层、取得未加工的资料、经由某些规则处理数据，并且呈现结果给使用者。

订阅者设定档

当 Web 浏览器发出请求，它会传送包含有用信息的数据串流（stream）给 Web 服务器，该 Web 服务器可以凭此信息找出何种浏览器发出请求、执行何种操作系统和该浏览器的处理能力为何。

在范例中的数据层，我们维护了一个订阅者设定文件的数据库，图 1-2 显示了这个数据库结构。

请注意有名称、电子邮件位置和邮递区码的字段，还有一些时间戳记（timestamp）字段指出该使用者是何时首次造访网站和他上次造访的时间。

另一个可以使用的是 cookie，我们在任何时间都可以传送一个 cookie 至使用者的计算机，如果使用者接收了该 cookie，该 cookie 会停留在使用者的系统一段时间，一旦我们将 cookie 放在使用者的计算机上，该计算机会在发出请求时也将 cookie 传回我们的服务器。举例来说，在图 1-2 中显示的 VisitorID 字段会在造访者的计算机上有一致对应的 cookie 值。当该服务器将 cookie 给一个新的造访者，就会有一笔新纪录会放在数据层的数据库，然后 VisitorID 变成了索引键，我们可以用以储存和取得订阅信息。图 1-3 显示了这三层是如何通讯的。

Visitor : Table

VisitorID	Email	FirstName	LastName	Zipcode	FirstVisit	LastVisit
205757				80112	3/24/2000 5:53:23 PM	4/5/2000 5:07:49
205758				94107	3/24/2000 8:38:13 PM	3/24/2000 8:38:13
205756				60601	3/11/2000 11:33:15 AM	3/23/2000 5:11:49
205755					3/11/2000 11:33:11 AM	3/11/2000 11:33:11
205633				95402	1/17/2000 11:33:31 AM	3/7/2000 10:36:41

Record: 1 of 32138

图 1-2 订阅者设定文件数据库显示了范例的记录，藉由唯一的 VisitorID 当索引键

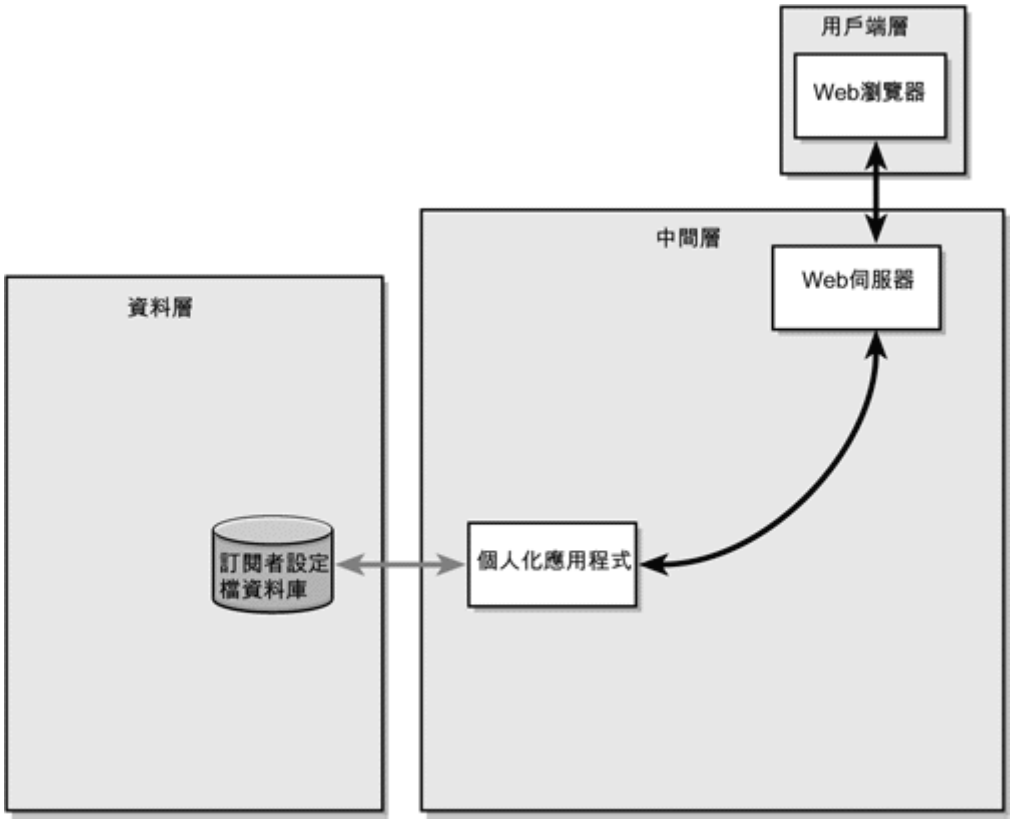


图 1-3 Web 服务器传送 HTTP 标头信息到个人化应用程序，该程序会查询包含订阅者信息的数

据库，VisitorID 字段是用于存取每一个特定使用者的信息

现在我们可以在伺服端读取 cookie，并且搜寻数据库以找出该使用者最后一次存取网站是什么时候。我们使用查到的信息以决定下一步要执行什么动作，这个过程中的所有动作都不需要使用者做任何辅助动作。

SQL 和 XML

结构化查询语言（SQL）是设计来取得关系型数据库中信息的语言，几乎所有的程序设计师和数据库管理师都使用 SQL、几乎每一种主要的关系型数据库产品都会支持 SQL－这一点告诉我们「标准」的力量有多强大：如果一个 Oracle 的数据库管理师的工作变成了 Microsoft SQL Server 的数据库管理师，藉由 SQL 的使用经验他仍然可以轻易的适应新产品。

然而，纵使 SQL 几乎是传送查询请求至任何关系型数据库的标准方式，每一个数据库产品生产商响应查询请求的方式都不同，响应是以结果集（recordset）的形式传回，但是每一种数据库传回结果集的二进制结构都不同。

虽然我们都是以开放式的标准（SQL）传送查询请求至数据库，但是传回结果集的方式却各有不同，如果响应的结果集也是开放式的标准不是更棒吗？（您一定猜到了－XML）数个数据库产品的生产商也互相合作要解决这个问题，现在让我们来看看 Microsoft 的努力成效吧！

Microsoft ADO（ActiveX Data Objects）纪录集对象可以将结果集变成有固定架构（ 第四章 将会说明架构）或规格的 XML 数据串流，下列为传送 SQL 查询至一个启用 ADO 的数据库的范例：

```
SELECT Name, Joke FROM Students
```

下列的范例就是使用 Microsoft Data Access Components（MDAC）传回查询结果集为 XML 文件。

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
    xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
    xmlns:rs='urn:schemas-microsoft-com:rowset'
    xmlns:z='#RowsetSchema'>

<s:Schema id='RowsetSchema'>
    <s:ElementType name='row' content='eltOnly'>
    <s:AttributeType name='Name'
    rs:number='1' re:nullable='true' rs:write='true'>
    <s:datatype dt:type='string' rs:dbtype='str'
                dt:maxLength='50'/>
    </s:AtrributeType>
    <s:AttributeType name='Joke'
```

```
    rs:number='2' re:nullable='true' rs:write='true'>
    <s:datatype dt:type='string' rs:dbtype='str'
                dt:maxLength='2147483647' rs:long='true'/>
    </s:AtrributeType>
    <s:extends type='rs:rowbase'/>
 </s:ElementType>
</s:Schema>
<rs:data>
    <z:row Name='Brain Travis' Joke='Two peanuts were walking down the
    road, and one was assaulted'/>
    <z:row Name='Ben Kinessey' Joke='A guy walks into a doctor's
    office.    The guy says, "Doc, everywhere I look I see
    spots".
    The doctor says, "Have you seen a doctor?"
    The guys says, "No, just spots."'/>
</rs:data>
</xml>
```

请注意该文件主要分为两个部份：一个是描述传回字段的结构部份，另一个包含传回的结果资料列部份。该架构是由数据库所定义的固定架构，但是您可以使用 XSLT（Extensible Stylesheet

Language Transformation）轻易的将这个固定架构转换为您所需要的架构（ 第六章 将会说明 XSLT）。

HTML 和 XML

当知道造访者最后一次存取网站的时间，我们搜寻储存在另一个数据库中的公司信息。这些信息会以静态的 HTML 网页呈现出版信息、新产品发表信息或其它过时的产品信息。同时我们也可以利用使用者最后一次造访网站时间这个有用的信息，向该使用者呈现上次造访时间以后的新产品信息，而不会呈现使用者上次已经看过的信息。

查询公司数据库时会以使用者上次造访网站的时间为依据，只查询该时间以后的新信息，XML 也可以在这种情况下当成从数据库传回信息的标准语法。图 1-4 显示了市场应用程序从数据层取得信息并传回 Web 服务器。

您将在稍后看见 HTML 其实就是 XML 的一种，如果 HTML 文件遵循 格式正确 （well-formed）条件限制的规则，那么它也可以被视为 XML 文件。这表示当我们使用 XML 工具处理信息时，仍然可以沿用 HTML 呈现信息。

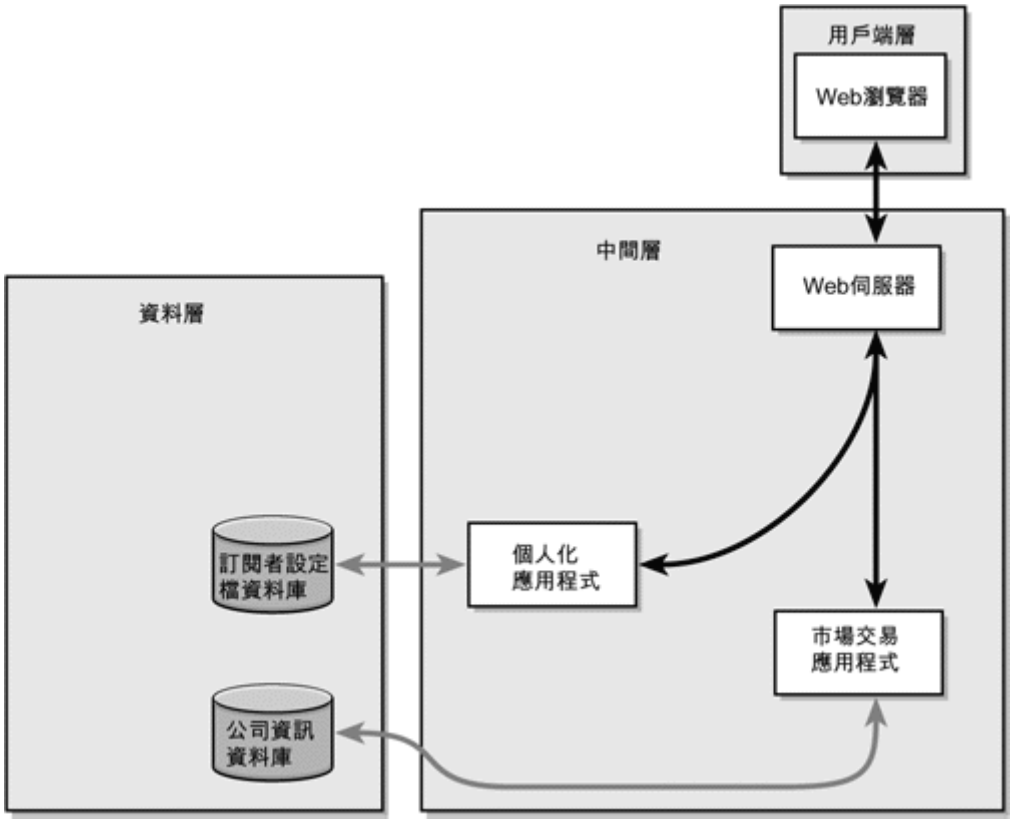


图 1-4 市场应用程序会使用关于使用者上次造访网站的时间信息以搜寻新的公司产品信息

电子商务服务

我们建置网站最主要的目的就是产生商业行为，因此我们再启动一个电子商务应用程序连接产品目录，并且为公司的产品提供服务。目录是 SQL 数据库的形式，如图 1-5 所示的数据层，在本范例中，在线目录是使用从订阅者设定文件数据库中撷取的信息以显示产品清单或提供联机使用者个人化的服务。

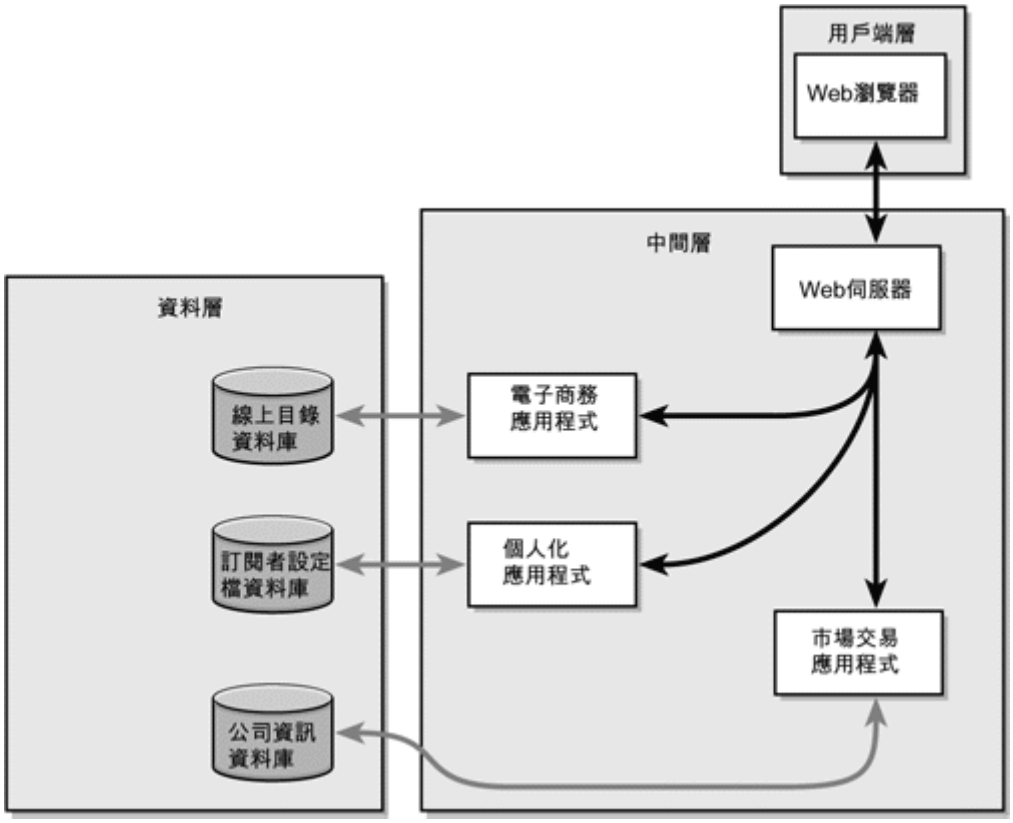


图 1-5 新增其它数据库至数据层，在线目录是 SQL 数据库，因此我们可以使用 SQL-to-XML 的技术

内容

现在我们开始提供信息给使用者，如果我们在网站上没有提供个人化信息的服务，使用者下次可能就不会再度造访我们的网站。

外部的网站也可以提供丰富的资料来源，这些网站并不是位于本机计算机上的数据库，但是其中包含丰富有价值的内容，我们可以使用 XML 请求信息，并且以 XML 对象的方式传回信息，以供未来更进一步的处理。

某些外部的服务，例如目前的天气状况或交通状况等等信息是需要付费的，利用这些外部的网站提供的信息，我们可以将其转换成对我们网站的造访者有用的信息。我们的工作是针对特定使用者，从数个网站集合特定类型的内容呈现在我们的网页上以符合特定使用者的需求。图 1-6 显示了在三层式结构中使用外部网站的服务。

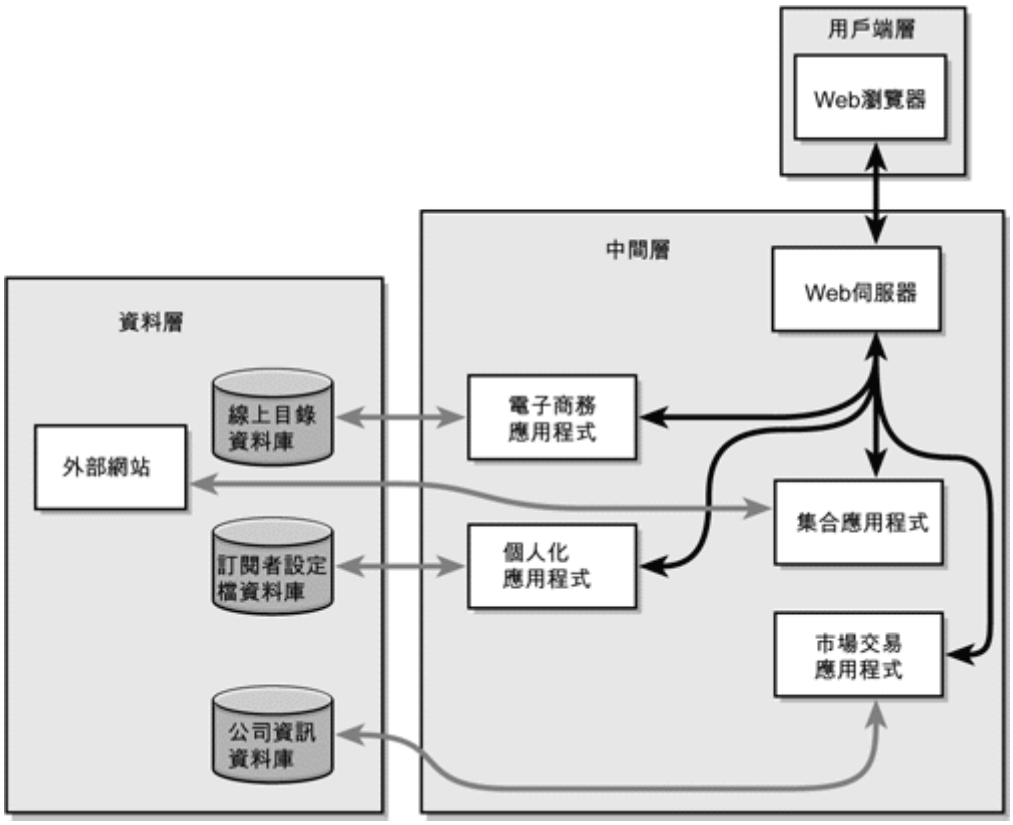


图 1-6 外部数据网站变成我们数据层中的一部份，并且集合应用程序用于整合这些新数据至中间层

假设我想要新增实时的天气信息至我的网站，目前的天气状况和预测的变更是以每个小时为基础，我可以自己开始收集天气信息，但是设定全国的天气网站超过我的付费范围，并且不在我的商业计划之内。

我也可以尝试从 National Oceanic and Atmospheric Administration（NOAA）取得免费的天气信息，该组织是透过全国的许多天气监测站撷取天气报告信息。然而，该数据是未经修饰并且需要解译以建立预测信息，这也超出了我的商业计划范围

有一个组织是专门撷取、预测和提供天气数据：The Weather Channel，我记得第一次听到关于 24 小时报告天气的频道是在 1980 年代早期的时候，当时有线电视才刚开始普及。当然，我觉得那是一个十分愚蠢的行为：一个只报导天气信息的全天候电视频道。

weather.com 提供了 The Weather Channel 内容的一部份，该网站提供了全美 25000 个地区的天气预测信息。如果我至 weather.com 浏览－如图 1-7 所示，就可以马上看见藉由邮政编码指定城市的目前天气信息。

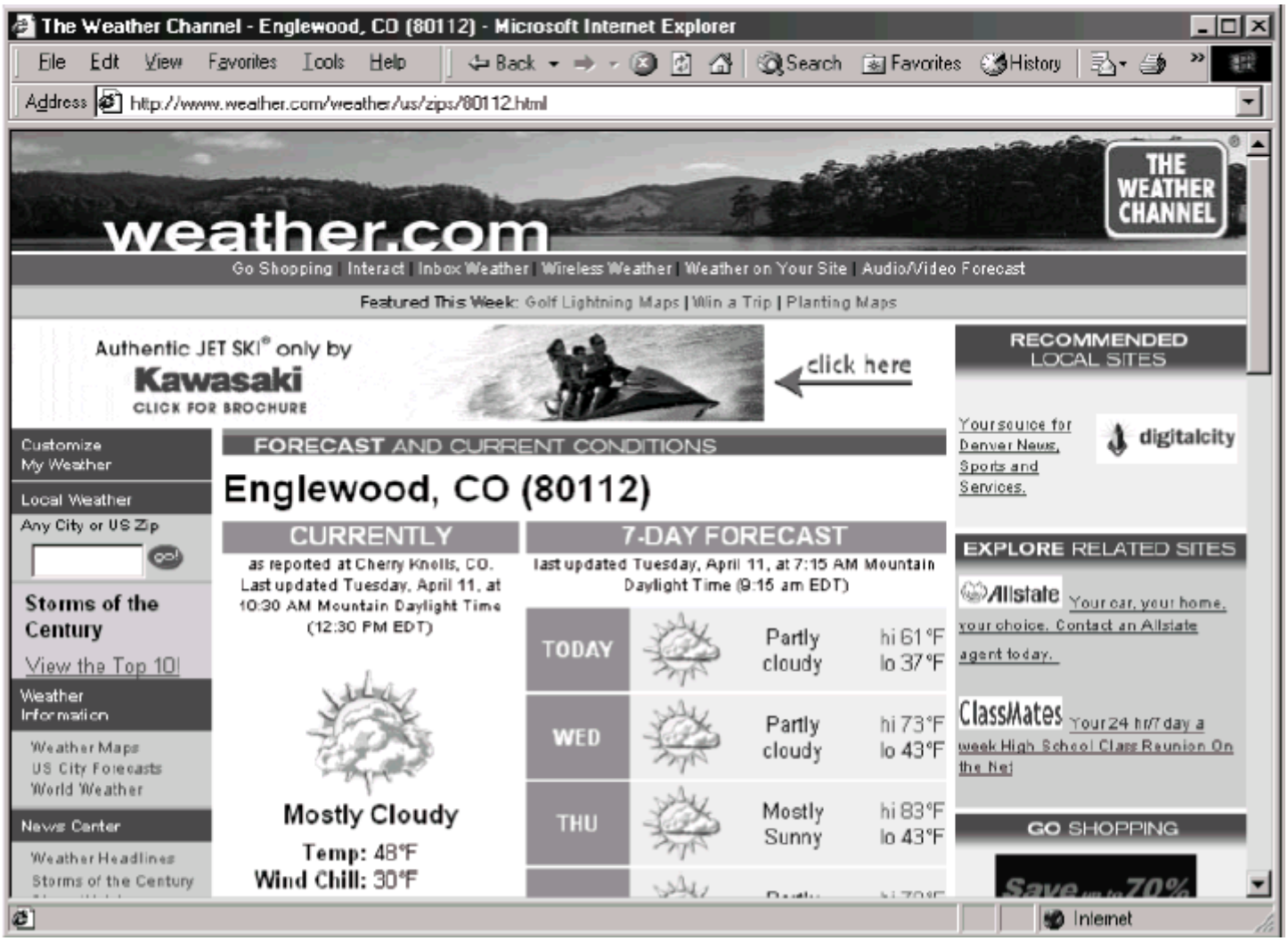


图 1-7 The Weather Channel 透过 weather.com 提供了未来几天的天气预报。

我想要提供这些信息给我的网站造访者。然而，如果他们连结至 weather.com 的话，同时也就离开了我的网站，我并不想要有这种结果。还有，我也不想在网站中使用 weather.com 的呈现方式、同时在某些时候我只想提供三天或七天的天气预测。最后，我想要提供给造访者的预设温度是摄氏温度。

因此，为了使用 weather.com 的天气数据，我决定要直接向该公司购买这些信息，我直接到 The Weather Channel 的办公室、敲办公室的门，并且询问职员有关我想要的数据呈现方式。

我告诉他们我的需求，并且建议无论何时我想要从他们公司取得信息时，我都会以下列的 XML 方式传送请求给他们：

```
<weather_request days="5" temp="Fahrenheit" wind="mph">
    <zipcode>80112</zipcode>
</weather_request>
```

然后我希望他们的响应讯息如下所示：

```
<weather_response zipcode='80012' Updated='2000-01-07  21:49"06'>
  <location>Aurora, CO</location>
    <current>
          <condition name='temp'>28</condition>
          <condition name='wind  chill'>12</condition>
               <condition name='wind'>from the Southease at 8
mph</condition>
          <condition name='dewpoint'>18</condition>
          <condition name='relative humidity'>66%</condition>
          <condition name='visibility'>25 miles</condition>
```

```
        <condition name='barometer'>30.02 inches</condition>
        <condition name='Sunrise'>7:02</condition>
        <condition name='Sunset'>4:50</condition>
    </current>
    <forecast>
        <day date='2000-01-08 21:49:06'
          high='49' Low='20' Sky='partly Cloudy'/>
        <day date='2000-01-09 21:49:06'
          high='49' Low='21' Sky='partly Cloudy' />
        <day date='2000-01-10  21:49:06'
          high='51' Low='25' Sky='partly Cloudy' />
        <day date='2000-01-11  21:49:06'
          high='51' Low='19' Sky='partly Cloudy' />
        <day date='2000-01-12  21:49:06'
          high='53' Low='21' Sky='partly Cloudy' />
    </forecast>
</weather_response>
```

在回到我们的例子之前，让我们花一些时间看看这些程序代码，这是一个由请求和响应组成的双向交易，这个结构基本上没有考量到错误发生的情况，那么有哪些潜在的错误呢？

假设我传送了下列的请求：

```
<weather_request day="5" temp="Fahrenheit" wind="mph">
    <zipcode>80000</zipcode>
</weather_request>
```

在本范例中，The Weather Channel 并不了解该邮政编码，所以它会传回的响应会如下所示：

```
<weather_request zipcode='80000'/>
```

这个响应是一个用来确认我输入的邮政编码的空元素，如果传回这一类型的文件，我需要执行某些动作以通知我的使用者邮政编码有一些错误，或是请使用者选择其它的邮政编码。

现在假设我传送了一个完全错误的请求：

```
<weather_request days="365" temperature"f">
        <zipcode>none of your beeswax</zipcode>
</weather_request>
```

而系统传回了下列响应：

```
<weather_request/>
```

我需要处理资料的传输和任何潜在的问题，这是在与 The Weather Channel 相关人员会议中的协议，然而这也是任何商业对商业交易的重要部份，而格式正确的 XML 文件提供了几乎可以掌控任何想象不到问题的方式。

最后，The Weather Channel 同意提供原始天气预侧数据的信息，但是这样做将会有开发和维护的耗费，因此他们要求我付费。

我们都同意在每一次我请求天气预测信息时，我方就必须付出半毛钱的代价，每月清算一次费用。因此 The Weather Channel 将会开发提供我所需信息的系统，并且我也基于他们所提供的内容付费。

现在，当我取得信息时，我应该做什么呢？The Weather Channel 所传送的 XML 文件中只包含我需要的资料，我可以对这些数据套用不同的处理程序，这完全取决于我需要何种类型的输出。换句话说，我可以任意让资料用我想要的方式呈现，并且套用我认为适合的处理程序，因为我存取的是原始数据。

SOAP

现在让我们来看看如何从外部网站取得数据，因为我们的数据提供伙伴可以使用数种不同的方式提供服务，我需要准备任何类型使用的联机。数据也许可以透过 FTP 联机、加密信道通讯协议，甚至电子邮件取得。然而，越来越多使用 HTTP 联机通讯协议取得数据，HTTP 会这么受欢迎是

因为它是最简单的通讯协议，但最重要的一点是，它几乎可以通过任何防火墙，这些因素使得 HTTP 可以在数种环境中轻松的建立系统。

我们如何请求外部信息并取得响应呢？答案就是 Simple Object Access Protocol（SOAP），Internet Engineering Task Force（ITEF）Internet 草案制定了 SOAP，它可以让您藉由使用标准的 XML 卷标集合呼叫远程系统上的程序，SOAP 使用 HTTP 为其数据请求和响应的通讯协议，因此在您与商业伙伴之间的系统可以快速和有效率的部署 SOAP 以取得数据。第八章 将会说明 SOAP。

XML 对内容提供者的意义

数个网站以联合的方式提供内容，这种联合方式的范例之一就是报纸，传统的联合组织例如 King Features 或 UPI 等等。

我花费了相当多的时间和 The Weather Channel 交涉以取得它的内容，然而 The Weather Channel 并不是唯一的天气数据提供者，还有 Accuweather、NOAA 和数不尽的其它提供者，同时也有位于其它国家的天气信息提供者。如果我不喜欢来自于 The Weather Channel 的服务，或者他们无法提供我所需要的所有信息呢？又或者，我想要在网站上提供国际性的天气预测呢？如果有这些因素存在，我可能就需要与其它的提供者交涉我所需要的内容了。

我们的网站不是唯一需要向提供者购买天气信息的网站，许多消费者的入口网站都会提供天气信息，但是这些天气的信息是以造访者的市场信息交换而来。在我们的情境模拟中，这些入口网站中的每一个都需要一或多个天气数据提供者，并且必须交涉它所需要的数据格式。

如果所有天气信息的提供者集合起来，并且提供消费者标准格式的天气数据以供信息的交换那不是很好吗？

在报纸的商业领域中真的达成了这个目的，首先让我们来看看报纸行业所面对的商业问题，以及他们所采纳的解决方案。图 1-8 显示了我们将用以存取报纸数据的三层式结构类型。

提供报纸内容的组织，例如 Newspaper Association of America（NAA），意识到产品的销售应该要以各种不同的格式－举例来说，New York Times 使用编辑管理和排版系统建立了他们的旗舰型产品。事实上，New York Times 有许多不同的系统，取决于它要建立何种型态的产品，每天的一般报纸使用一种系统，而 Sunday 杂志却又使用另一种系统。

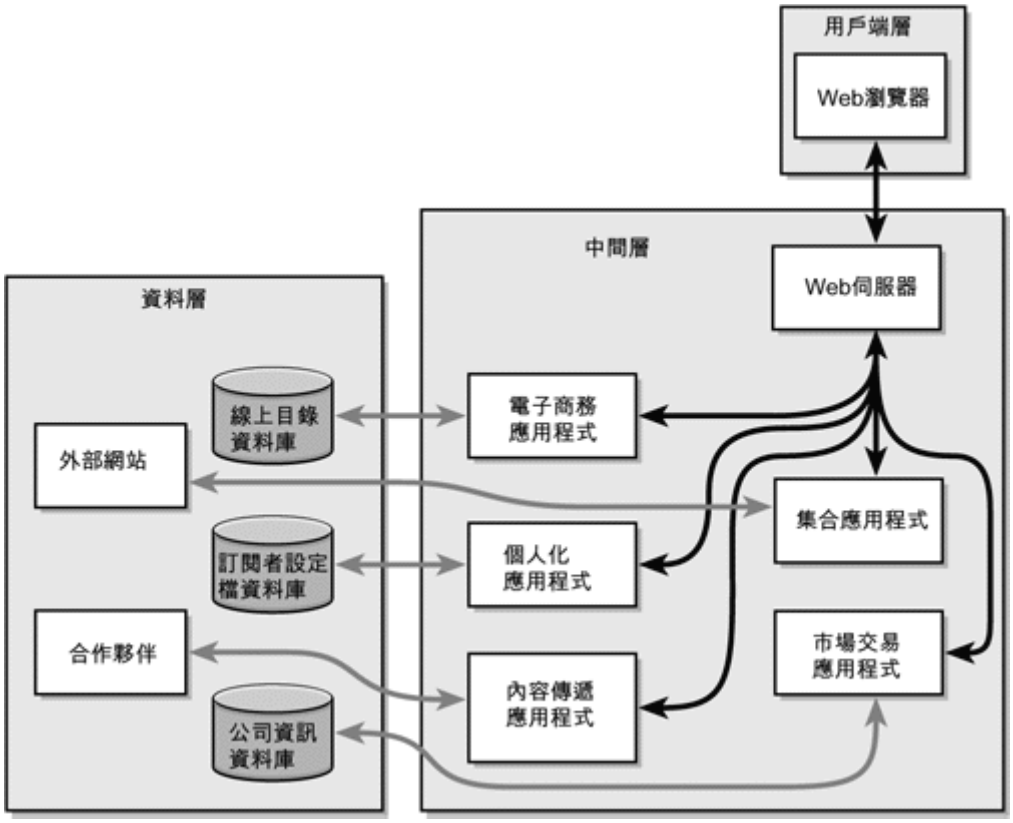


图 1-8 多个联合的商业伙伴基于标准提供内容，因此我们可以集合不同的来源为单一的文章清单

如果我想从 New York Times 购买内容，我就必须能够存取它所使用的档案格式，或者说服该报商转换它的信息以兼容于我的系统。我可以从 New York Times 取得国家或世界性的新闻，但是也许我只想提供给我的使用者地区性的新闻。

如果我的使用者只在纽约地区，New York Times 就足够满足我所有的需求了。然而，若要依照地域性来提供我的使用者所预期的信息，我将需要在许多不同的位置与不同的区域性报商订立合约。

我的小型 Web 网站只需要新闻提要，但是我却必须耗费许多时间去说服 New York Times 转换他们的格式以符合我的需求。还有，我也必须分别的和其它报商交涉以说服他们转换数据格式。如果我真的要向这些网站购买内容时，我将需要分别开发转换筛选器以解译他们的数据，来符合我的网站数据格式。

因为每一个提供者内容的编辑和管理方式都不同，我必须研究所有的数据格式、了解它们，并且撰写转换筛选器－这会耗费许多的时间，而且容易造成同步化的错误。该系统如图 1-9 所示，图中显示了两家报纸提供商（New York Times 和 Denver Post）、一家新的媒体内容提供商（MSNBC）和以每月杂志格式的报纸提供商（Reason）。

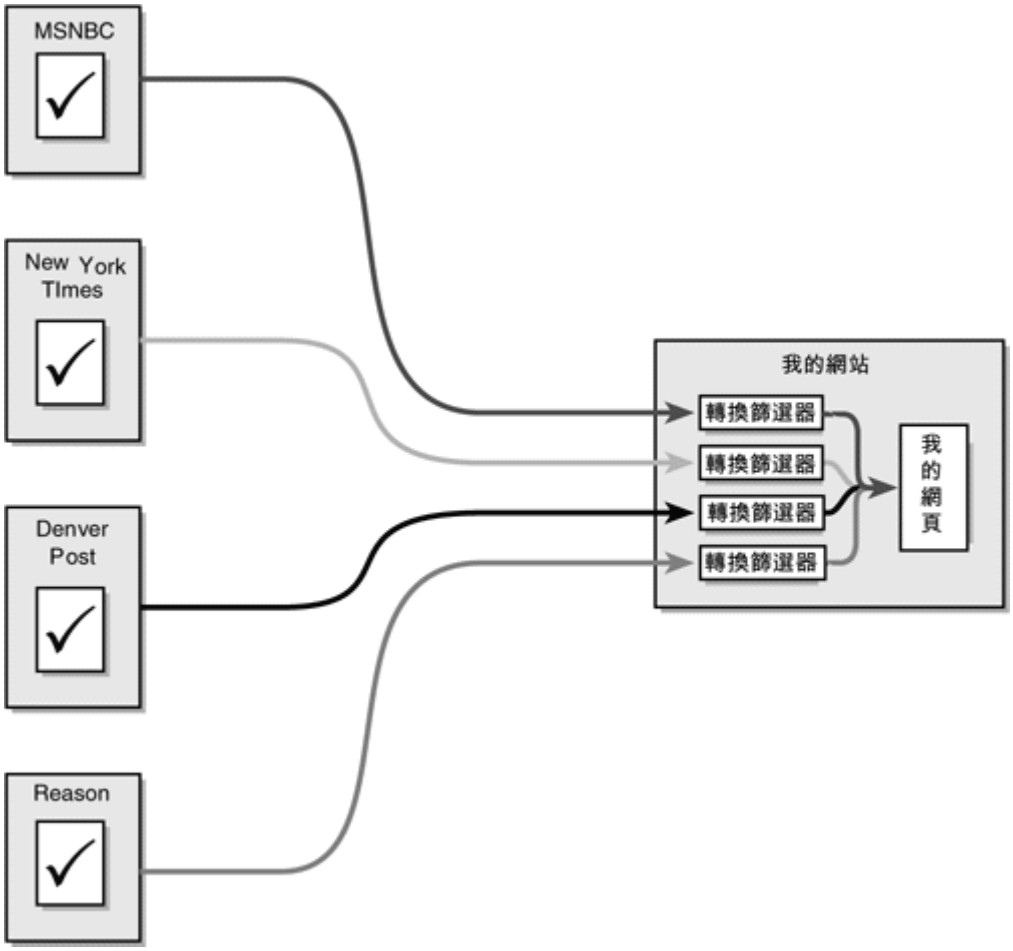


图 1-9 联合伙伴中的每一个都需要一种特定的转换器以转换它们的结构为我们网页中使用的格式

但是图 1-9 中的建议十分的复杂，假设有其它的网站也需要购买这些网站中的内容，同样的这些网站也需要撰写转换筛选器。因为我们都是竞争者，所以我并不会将我的转换程序与他们共享，当有其它的竞争者加入时，系统看起来就如 1-10 所示。

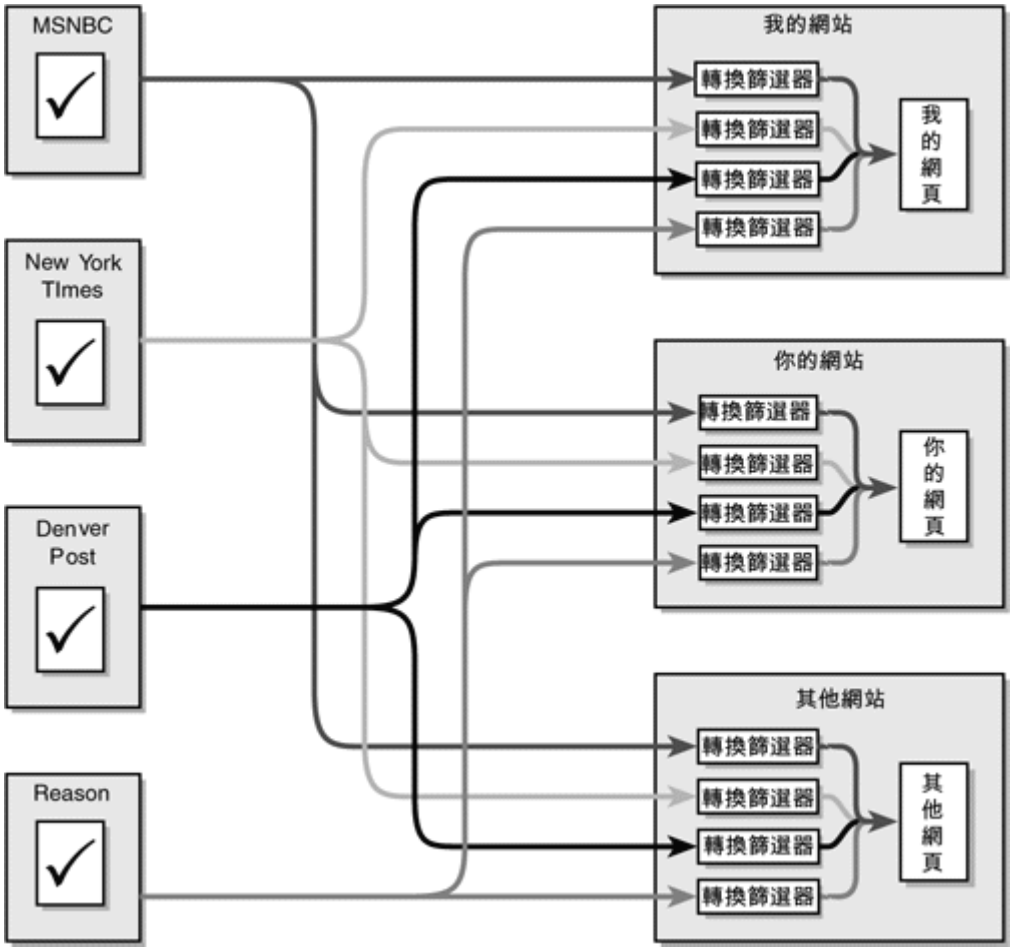


图 1-10 使用提供者内容的每一个客户都需要撰写自己的转换器

这还不是最糟糕的部份，所有这些信息来源会建立供打印报纸和 Web 网站使用的内容，而将这些信息整合在一起通常会产生过多的信息，报纸上并不需要打印这么多详细的信息。举例来说，请您看看下列的句子：

The Burning Man Festival is set for next June. The admission price is $150.

我们可以从这个句子中得到关于该节日的有用信息，但是这个句子的原始来源中却有许多对特定的读者来说是不必要的。举例来说，Burning Man Festival 是 Nevada 沙漠中每年都会举办的节

日，如果我们的读者想要更为详细的信息，您已经知道「六月」是一个很有用的信息，但是如果我们知道实际的日期不是更好吗？

然而，如果将下列形式的信息放在文章中就太过火了，因为这样就失去了简洁的目的：

The Burning Man Festival, an event happening in the Nevada desert, is set for the week of June 12-19, 2000. The admission price, in U.S. dollars, is 150.

相反的，这些信息应该以某种形式撷取，以建立适当的打印文件或提供使用者目标性的有用信息。换句话说，大部份的人都需要较切题的数据，越切题的数据对于我们的组织和投资者会越有价值。

标准

当 Web 在 1990 年中期产生重要的推动力时，许多传统的新闻提供者意识到市场上有许多网站会想要购买新闻内容，但是并无法负担转换排版数据的成本。如果这些新闻的销路市场可以利用某些形式提供低廉成本的信息，让小型的新闻提供者可以负担，那么或许可以建立新的收入来源。

许多的报纸都隶属于 NAA，在 1970 年代，NAA（后来被称为 American Newspaper Publishers Association）透过它的 Wire Service Committee 建立了一个众所周知的规格称为 ANPA 1312。委员会设计这个规格以提供网络服务的方式传送新闻材料，该规格也用以确保网络服务中会嵌入程序代码－粗体、字距和印刷标记等等。

几乎北美所有的网络服务和所有的网络软件供货商都采纳 ANPA 1312 标准，后来 ANPA 1312 变成了 News Industry Text Format（NITF）并且 Wire Service Committee 变成了 News Information Task Force（NITF）。

到了 1998 年，NITF 规格转变为 XML 架构，该架构新增了许多的规格－卷标用以描述数据，而不是描述印刷样式。

您可以比较图 1-11 和图 1-10，使用标准化的新闻传递方式需要每一个提供者建立一个转换器，并且消费者联合使用一个共享的转换器，因此所有的站台会使用相同的元素字汇进行通讯。图 1-11 显示了使用标准化方式如何从数据提供者取得数据。

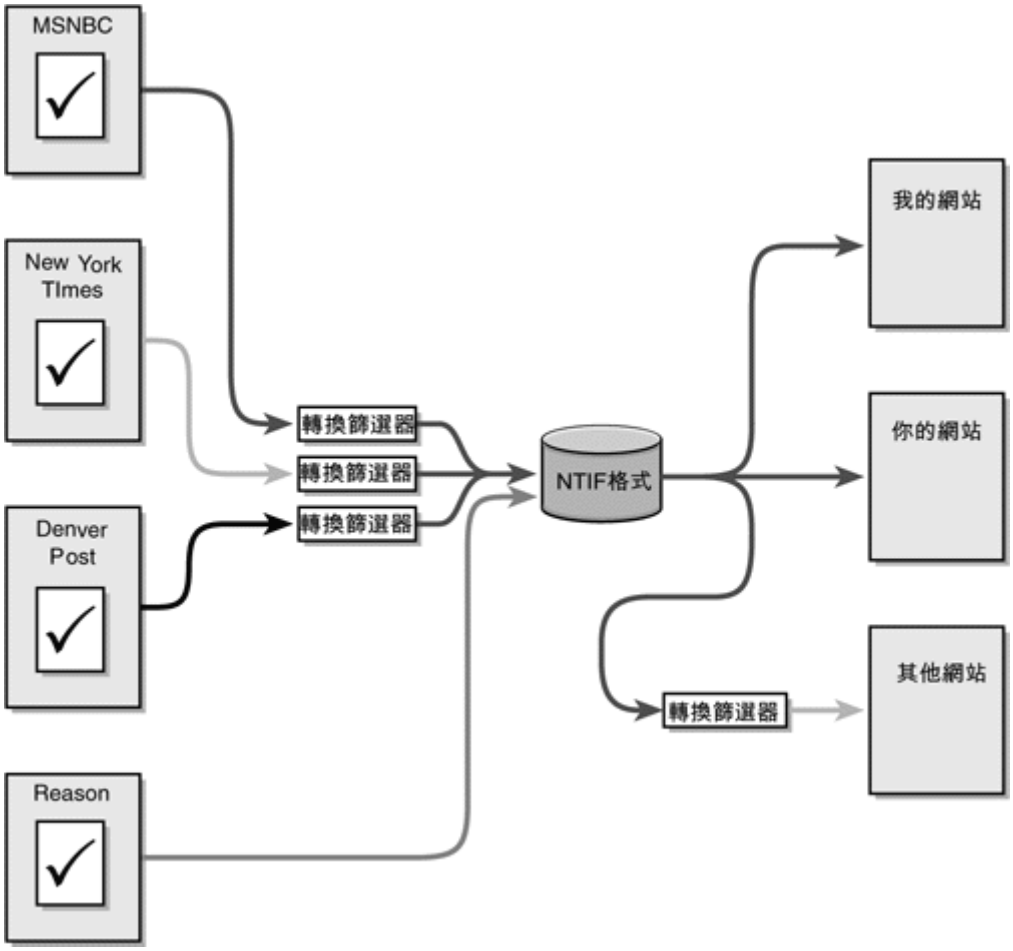


图 1-11 以单一格式提供资料可以减少经营者必须分别为每一个数据提供者建立转换软件的需求

每一个数据提供者只需要建立一个转换筛选器，将提供者的格式转换为 NITF 格式，然后这些网站就可以取得原始的格式，或是将其转换为网站系统所需的格式。

但是这也不是最佳的部份，试想一下本章稍早提到的 Burning Man 范例，藉由使用卷标的方法，基本信息的表现方式可以更为多样化且丰富。这些额外的信息让网站设计者可以设计出更多的内容。

```
The <event><location>Nevada desert</location>Burning Man Festival
```

```
</event> is set for <chron norm='20000612'>next June
</chron>. The admission price is <money unit="USD">150</money>
```

举例来说，假设按一下由 **<event>** 卷标指定的文字就会带出在 Nevada 沙漠中发生的所有事件；或者按一下由 **<next June>** 卷标指定的文字就会带出在该时间区段中发生的所有事件。事实上，所有的数据都是以标准的方式加上标签提供网站设计者以内容目标传送的方式传送信息给使用者。举例来说，在德国的使用者也许会要求所有的货币以德国马克的方式呈现，并且在澳洲的读者也许会希望所有的货币以他所想要的格式呈现。

NITF 格式提供了丰富的标签提供新闻题材的内容使用，而对想要以其它方式呈现额外信息的消费者也会十分有用。

图 1-12 显示了来自 MSNBC 中的新文章。

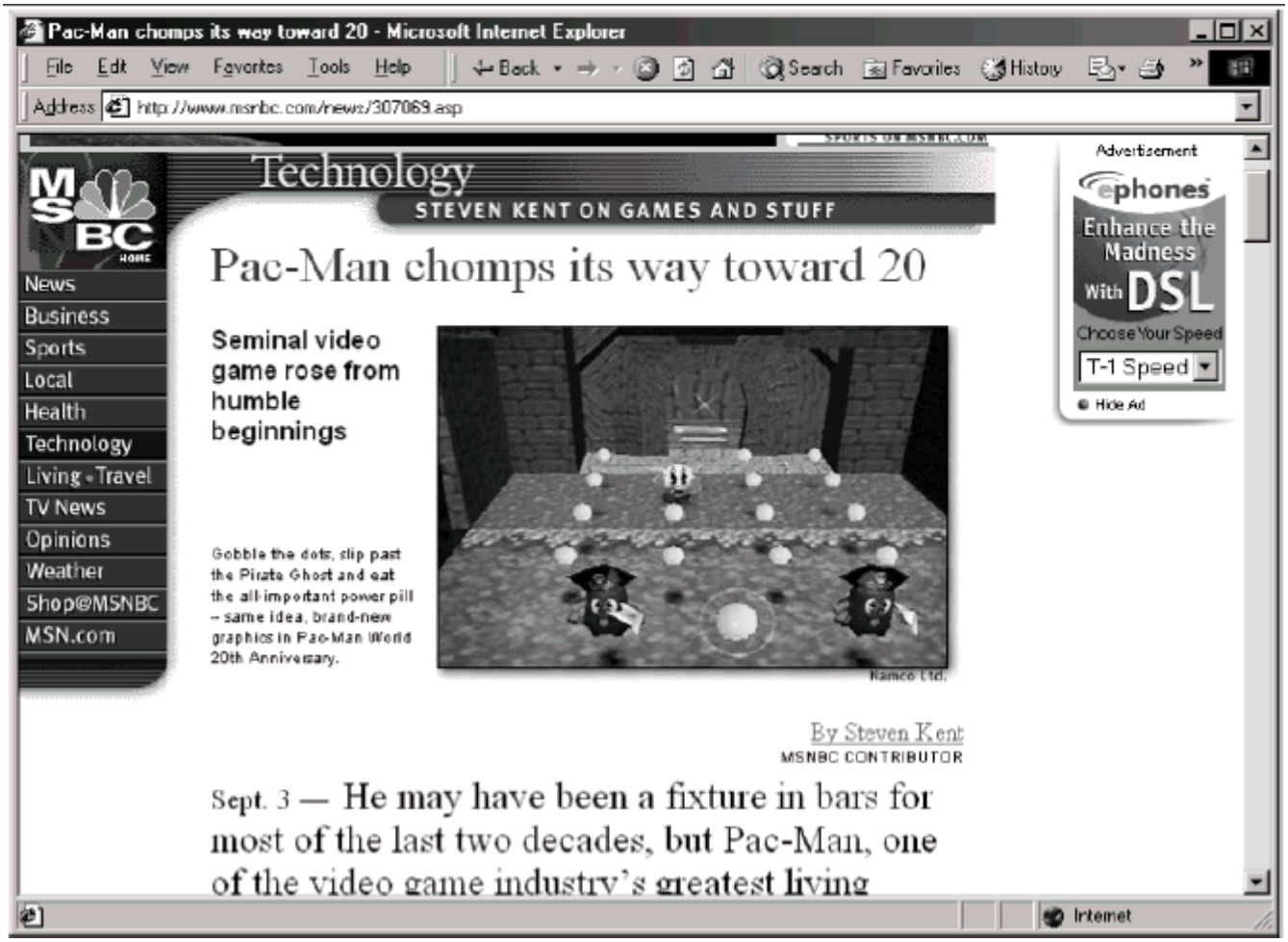


图 1-12 传统的报纸文章以 HTML 网页方式呈现

这个信息是以下列的卷标化 NITF 文章方式呈现：

```
<?xml  version="1.0?>
<nitf>
  <head>
    <title>Technology  Tools  and  Toys</title>
  </head>
```

```
<body>
  <body.head>
    <headline>
      <hl1>pac-man  chmps  its  way  toward  20<hl1>
      <hl2>seminal  video  game  rose  from  humble
      beginnings<hl2>
    </headline>
    <byline>
      By<person>Steven  Kent</person>
                    <bytag>MSNBC  CONTRIBUTOR</bytag>
    </byline>
    <dateline>
      <location></location>
      <story.dte>1999-09-03</story.date>
    </dateline>
  <body.head>
  <body.content>
    <p>He  may  have  been  a  fixture  in  bars  for
    most  of  the  last  two  decades,  but  Pac-Man,
    one  of  the  video  game  industry's  greatest
```

```
      living  legends,  is  still  a  bit  shy  of  legal
      age.</p>
      ...
    </body.content>
          <body.end></body.end>
  <body>
</nitf>
```

NITF 格式让新闻提供者可以使用标准的方式提供原始数据给消费者的网站，并且让消费者的网站可以简单的读取新闻提供者卖的内容。现在我不需要分开处理每一个提供者所卖的内容，而只要简单的问：「请问贵公司提供 NITF 兼容格式吗？」。如果答案是肯定的，那么我可以确定他们的内容符合我的需求；如果答案是否定的，他们也需就会失去我这个消费者并且应该考虑使用这个产业标准。

我们所建立的网站会针对每一个造访者传送个人化的信息（此技术称之为 内容传送目标，Targeting），这表示这是一个消费者导向的网站，使用 NITF 标准让我们可以针对造访者建立特别的信息，我们尝试吸引客户，并且提供我们的产品和服务给这些客户。这种以吸引客户为目标建置的网站，我们称之为 B2C 网站。

然而从提供者取得消费者所需的信息，需要和内容提供者协调两个网站之间的内容往来方式，从这个角度看来就是属于 B2B 的范畴了。

建立网页

我们讨论的所有信息都是由中间层应用程序撷取，这些信息用于建立内容、传送目标的网页给特定使用者。当藉由每一个使用者计算机上的组态和偏好请求传送信息时，这个网页就是为该使用者特别建立的。我们的 Web 网站建立网页、传送网页，并且等待更进一步的请求。XML 帮我们达成了这个梦想！

XML 对于内容管理的意义

本书的重点是介绍在 B2B 电子商务交易中使用 XML，因为 XML 的本质就是可以自己描述交易，所以它很自然的就相当适合电子商务。这表示您可以从交易的一方建立 XML 资料串流，然后将其传送至某处以供读取、解译和最后被删除。

XML 也是一项可以永久管理数据对象的技术，如同传统的文件。内容提供者就是发布信息让客户消费的公司，发行者例如 Microsoft Press、Prentice Hall 和 Macmillan 就是内容提供者，但同时也是报纸和杂志发行者。

内容提供者的商业压力迫使他们必须找到一种永久保存信息对象的方式，过去，以纸上文件提供内容的方式是可接受的，而且这种模式运作了数世纪之久。

纸的好处之一就是可以利用许多方式在上面表现您所需要的呈现方式，任何文字处理器、排字机或页面制作程序都可以达成相同的目的。您可以将某一页的版面排好，然后打印数千次，您也可以加以拷贝等等。纸变成了交换信息的通用媒体。

然而，纸在现代变成了 Word 档案、Adobe Acrobat PDF 档案等等，并且会有一些保存期限上的问题。一份打印的文件并不是 实际的 信息，该文件只是信息的呈现方式。纸这种媒体需要经过某些格式加以修饰，这是因为纸是散发内容的方便形式。

在 1990 年代早期出现了 CD-ROM，人们预期藉由电子媒体传递信息更为方便，所以内容提供者发现他们如果使用纸会有某些储存上的问题，因为他们无法将纸放在 CD-ROM 中。举例来说，如果想要搜寻特定字符串或关键词的文字，或是想要限制特定内容的存取，纸这种媒体就无法达成这些要求。

XML 对于内容收集的意义

Internet 增加了我们取得信息的方式，过去，消费者无法获得如此多的商业信息来帮他们下购买的决定。

试想 Web 蓬勃以前的汽车商业，要买一台汽车，您必须耗费许多时间到许多经销商处比较价钱、服务和您想要的功能，这同时耗费了买者和卖者的时间和金钱。

现在我只要到 cars.com、carpoint.msn.com 或其它数以百计的汽车网站就可以取得我想要的内容，我也可以从多种汽车制造商的网站取得丰富的内容－其中包含技术规格和经销位置等等。Internet 建立了购物的环境，让我可以免于在实际的世界中寻找我所想要的汽车。

XML 在这里有什么帮助呢？试想一个内容的收集者，例如图 1-13 的 carpoint.msn.com 所示，该网站从数百个位置取得信息，并且让这些信息集中在一个地方供我们浏览，现在就让我们进入购物的情境模拟并且看看哪里适合 XML 吧。

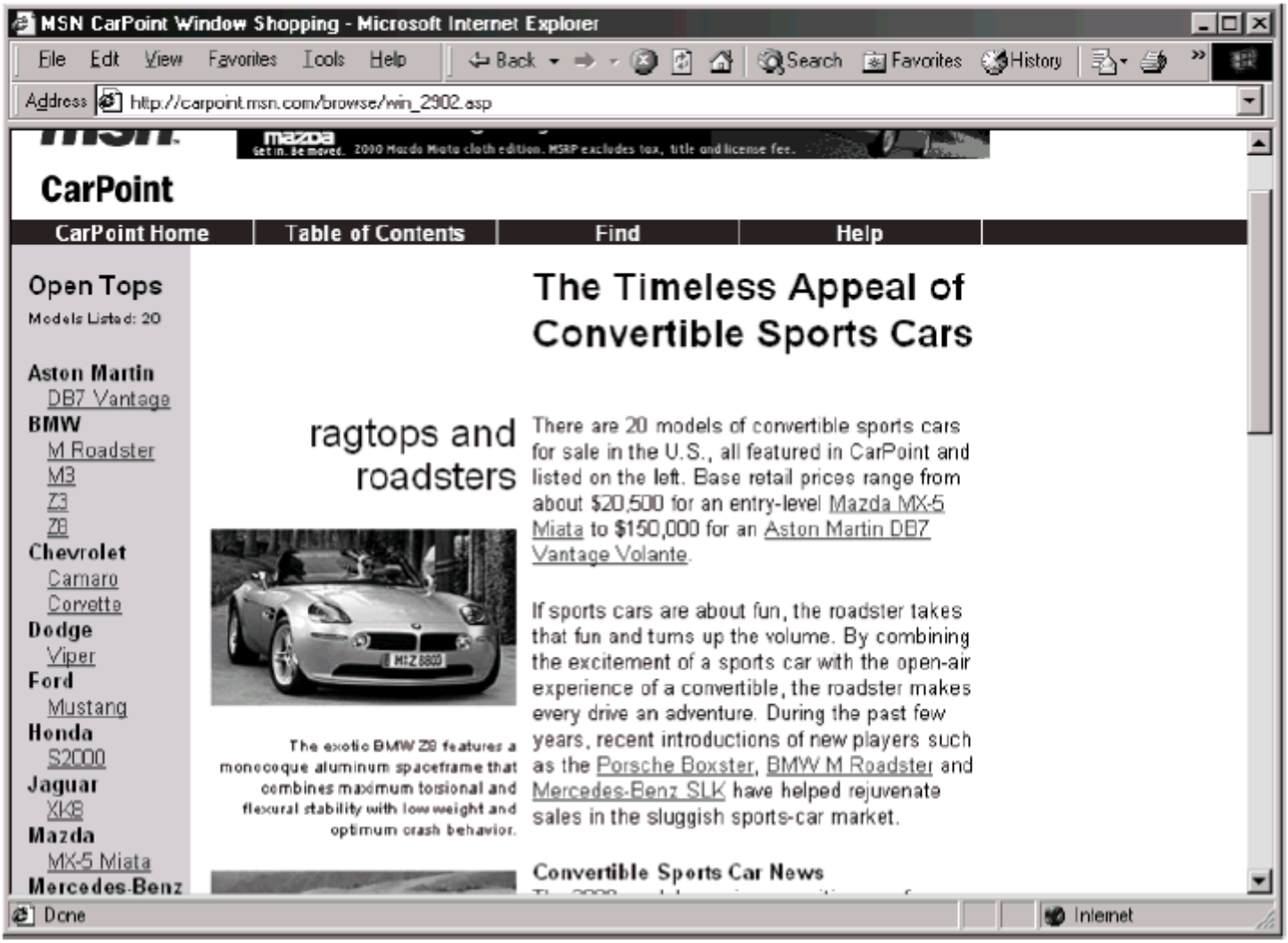


图 1-13 MSN CarPoint 的 B2C 网站有丰富的 B2B 组件，因为消费者所需的信息是来自于许多其它位置的公司

在本范例中，我要买一台敞篷车并且还想看看其它的信息。MSN CarPoint 网站在左侧窗格有每台车详细内容的连结，而在右侧窗格中会显示选取汽车的功能内容。

如您在图 1-14 所视，我选择了新型的 Jaguar XK8。

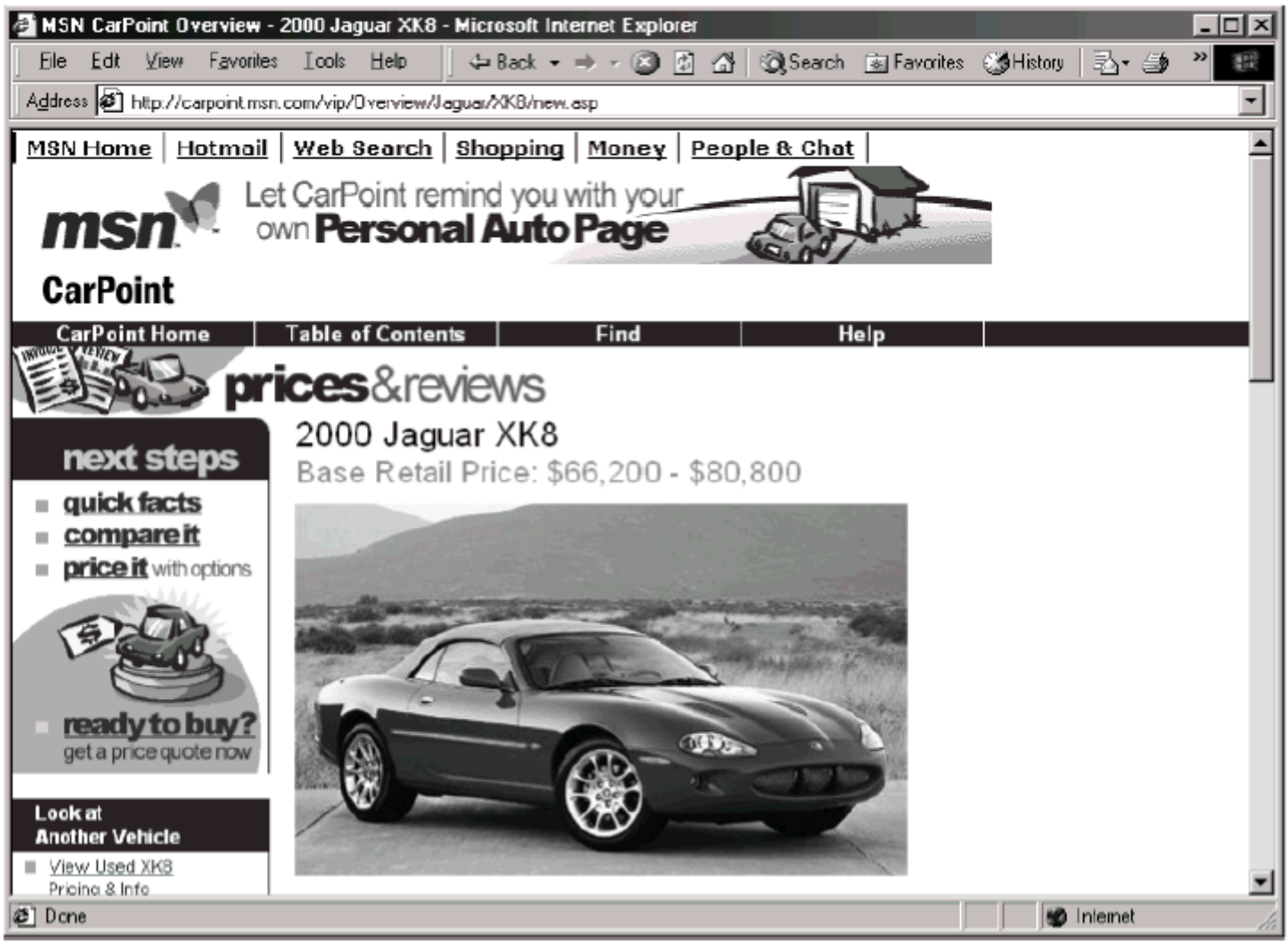


图 1-14 详细和多样化的信息用于建立优良的、数据丰富的网站。MSN CarPoint 从许多不同的地方集合信息在这个网页上呈现。

在图 1-15 所示的网页中，我可以看见关于该车的扼要信息与更详细信息的连结。我将会选取关于引擎的信息，我住在科罗拉多州，因为那里的空气稀薄，所以我需要较多的马力。

网页中显示 XKR 的引擎增压器为 370hp，我们设定适当的选项并且看看价格是多少，如图 1-16 所示，我选取导向系统以避免在长途旅行中迷路。

这只是几百台汽车中的其中一部，并且在该网站上有数以千计的组态选项，那么这些数据是如何取得的呢？基本上，那些选项信息是加载网页上的一个对象中，该对象和包含设定数据的网站通讯，当我在网页上输入数据时便执行动态的计算。

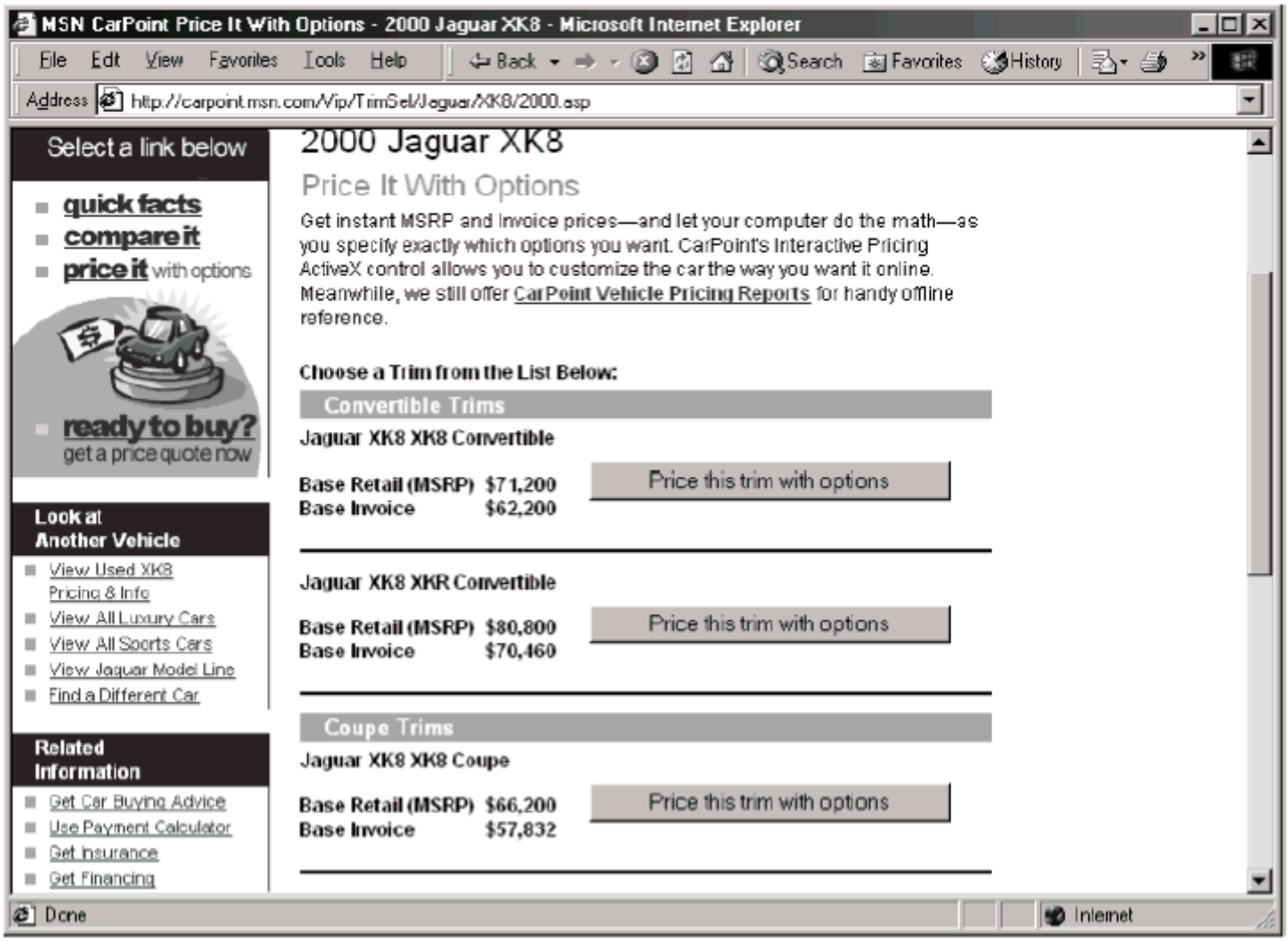


图 1-15 显示可用的设定信息，这些信息是来自制造商

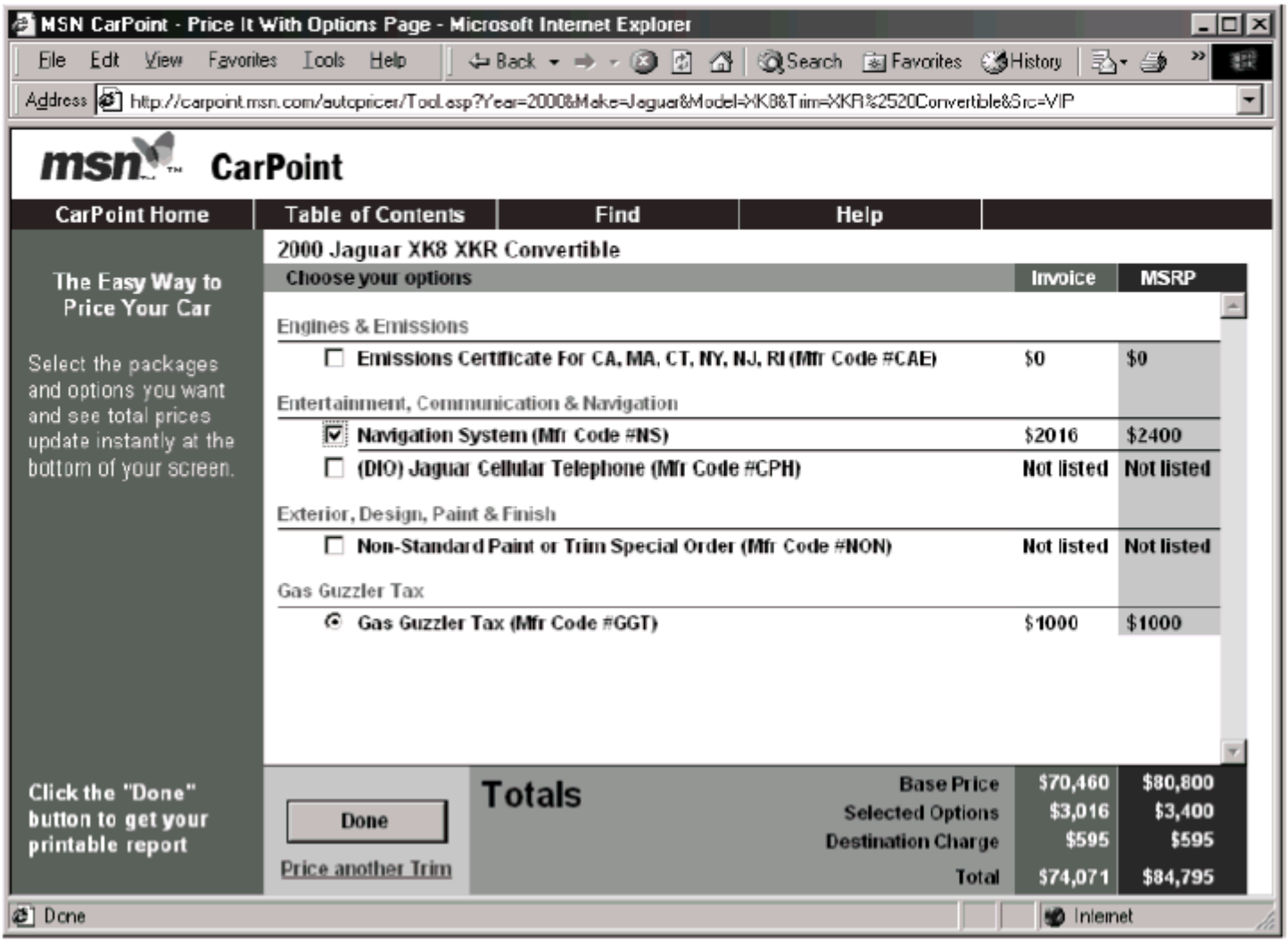


图 1-16 您所选取汽车的特定选项

假设我们想要建立这个网站，我们将需要重复上一个天气范例中的步骤－到 Jaguar 汽车公司、Honda 汽车公司，然后到所有其它的制造商。这将会是一个耗时的过程，而且对我们的收益来说可能是没有任何价值的。

然而，我们并不是唯一需要使用这些信息的网站，如同 NITF 范例一样，有许多相关汽车信息的提供者。

- 汽车制造商提供技术规格和所属汽车的配备选项。

- 汽车书籍出版社提供读者所关心的重要汽车评论。

- 狂热者例如区域性、国家性汽车俱乐部，或搜集者团体，有他们所关心的汽车信息。

- 保险公司有保险统计数据的相关信息，这些信息可以让您知道您的保险金额。

- Insurance Institute for Highway Safety 和多个国家的政府机关提供汽车的安全性测试信息。

- 汽车经销商提供关于价格和服务的信息。

一个网站如果要能够吸引潜在的客户，来取得最多的造访次数和最大的广告收益，它必须能够符合每一个造访者的要求并提供最多的信息－例如现在最热门的内容传送目标网站（B2C）。然而，

建立这样一个网站所需要的信息高度仰赖能够从其它的内容提供者取得内容的能力（B2B），要建构这种商业对商业的交易环境，您自然会选择 BizTalk。

电子文件交换的背景

电子文件交换的祖先就是电子数据交换规格（Electronic Data Interchange，EDI）。在 1970 年代初期，像 Sears 和 Kmart 等大型公司，是电子商业通讯的先锋，由于他们必须和数以千计的商店和供货商竞争，也造成了堆积如山的纸上文件。显而易见的，这些文件都是公司之间往来的媒体，由于造成了大量文件的问题，因此电子式的通讯脱颖而出。要在公司组织之间以电子式的方式通讯，供货商被强迫要针对每一个商业合作伙伴开发和维护自订的接口。

到了 1970 年代末期，由代表性的传输组织、政府机关和计算机制造商组成了一个委员会，着手开始开发改良的方法，并且将电子式商业通讯标准化。American National Standards Institute（ANSI）在 1979 年公布认可 Accredited Standards Committee（ASC）X12，它的目标是开发电子式商业交易数据交换的统一标准，X12 就成为最先开发数个标准格式，以便企业执行电子式商业活动的团体。

各种不同的 X12 委员会于特定文件的标准上做了许多努力－这些文件大部份都是发票和采购订单。每一个规格都必须在所有的状况下符合所有的使用者需求，也因此标准十分复杂并且难以建置。

因为 EDI 标准十分的复杂，某些分歧的团体开始出现，并且建立了更多的特定产业标准。举例来说，National Retail Merchants Association 着手开发了兼容 EDI 的采购订单讯息标准。然而，这些标准的定义并不完善，标准也十分模棱两可，所以零售商和供货商都没有使用这个标准，后来 National Retail Merchants Association 继续支持 X12 标准。

这个扼要的历史故事告诉我们标准必须符合两种需求：它需要被广泛的采纳；它必须够明确才会有用。

XML 对于电子文件交换的意义

XML 是由 World Wide Web Consortium（W3C）所建立的标准，XML 是一种强大的语法，允许公司之间为达成有效率的信息交换建立结构。本书将会在 第四章 说明 XML。

EDI 的问题（开发时间长、要达成跨组织的目的必须满足许多需求、难以建置）可以经由使用 XML 的交换标准解决，名为 EDI/XML 的一个团体在 EDI 的开放式版本上做了许多的努力，希望传统的 EDI 使用者可以使用 XML 为他们的数据交换语法。他们并不寂寞，因为有越来越多的团体朝着相同的目标努力－这表示他们都要建立一种通用的方式执行电子商业活动，并使用 XML 为数据定义的语法。

另一个努力的成果就是 ebXML Initiative，其中的 eb 是指「Electronic Business」，United Nations/CEFACT 和 Organization for the Advancement of Structured Information Standards（OASIS）－XML 的产业团体，他们都为 ebXML 工作。ebXML 的目标是让 XML 对象的标准集

合可以符合国际性的交易行为。ebXML Initiative 的目标是开发一种技术性的基础结构，提供一致性的使用 XML 方式来交换所有的电子式商业数据。这些参与团体的清单如下：Ariba、Amazon.com、Bank of America、Boeing、CommerceNet、Dunn and Bradstreet、IBM、Microsoft、Oracle、Sun 和全世界许多其它的团体。同时也有许多新的规格正由其它团体开发中：Ariba 的 cXML、CommerceOne 的 CBL、RosettaNet 的 RosettaNet Framework、CommerceNet 的 eCo Framework 等等。

其它的产业规格包含了 Financial Information Exchange（FIX）规格，该规格供交易安全性的信息使用、HL7 供交换医疗记录使用、ACORD 供资产和意外保险信息使用、J2008 供汽车产业使用等等，上述的所有规格都使用 XML 为基础的语法。

这些规格都可能成为一种称为 **架构** （Schema）的规格集合。架构描述特定的商业文件，例如发票、采购订单、医疗记录和安全性的交易信息。架构是电子商务解决方案的重要部份，本书 第四章 将会说明架构。

XML 和电子商务

那么 XML 对上述的各种情况有什么帮助呢？答案就是：任何地方都需要 XML。XML 只是一组规则的集合、一种标记数据的语法。同时也是一种小型、标准和简洁的语法，这也是它的价值所在。从事商业交易，您必须和商业伙伴之间有清晰的通讯方式，XML 提供直接的、精确的语法

以辨识通讯中要完成交易的所有信息－无论该交易是在 Internet 上以安全性的封包传送信用卡信息、或是从 Web 服务器上的免费 HTML 网页搜寻最新的 MP3 音乐。

XML 和电子商务的另一个观点是信任关系，如果您不信任某人，您也不会和他进行交易，无论是在商场中的面对面交易或是在 Web 上跨国的匿名交易，XML 都可以提供某种层级的验证。

XML 和应用程序解译方式

当日本航空（Japan Airline，JAL）的巨无霸客机飞进巴黎的 Charles de Caulle 机场时，驾驶员需要以共通的语言和塔台控制员通讯，该驾驶员的母语是日文；而塔台控制员的母语是法文。如果他们选择日文，塔台控制员必须负责了解日文，同理可证，塔台控制员可能需要了解德文、意大利文、英文和任何其它的语言（只要任何其它国籍的驾驶员飞进他们的机场）。

如果驾驶员和控制员以法文通讯，则解译语言的工作就是驾驶员的责任了，同理可证，驾驶员可能需要了解德文、意大利文、英文和任何其它的语言（只要他飞进任何其它国籍的机场）。

这些情境模拟中的任何一个都是耗费、耗时同时也包含了潜在的危险。因为安全性的理由和交易的效率，世界上所有的航空控制系统决定使用一种标准的语言通讯，这种语言就是英文。

当 JAL 的驾驶员要和塔台通讯时，他必须将母语转换成英文，所有的句子都必须以英文组成并且透过麦克风通知塔台，无线电会传送英文至塔台，塔台的控制员就会听到英文。

塔台控制员将英文解译成法文（他的母语），因此他可以使用最简单的方式处理这些信息。

说明

十分有趣的一点是，当同一个驾驶员飞回日本的时候，他仍然需要使用英文和塔台通讯，因为在相同机场中的其它飞机也许有非日文语系的驾驶员会在无意之中听到该对谈。

XML 文件通常是由某些应用程序执行中所建立的，这个文件中包含所有可能会影响交易的信息，最新建立的文件会以 XML 的呈现方式传送至商业合作伙伴，而 XML 语法和建立文件应用程序的语法不同；商业合作伙伴的接收应用程序会取得 XML 数据串流，而这些数据串流和接收应用程序所使用的语法也不同，因此这些信息必须被解译成应用程序了解的语言以供处理，在接收端处理完毕后，这些信息就不再是 XML 文件了。包含在文件中的数据会被加载系统，这些数据是由 XML 卷标所描述。

XML 和处理语言

XML 是一个开放式的标准，它并没有被某间公司所独占，但是它是由许多产业中的数家高度竞争的公司所制定，这些竞争对手都有共同的认知－那就是他们全部都同意这个交换信息的简单语法。

由于它的开放式性质，XML 可以在任何平台、使用任何语言上正常工作。XML 本身并不会做任何事－它只是一种描述和呈现数据的简单语法。XML 最重要的优点就是允许您可以将数据本身和处理数据的流程分开。

您可能曾经听过许多关于 XML 和 Java 的信息－XML 用以呈现数据；Java 用以处理数据。同时也因为他们的开放式性质，程序设计师可以使用 Java 建立跨平台的应用程序，并且使用 XML 在多种平台间交换数据。

然而 Java 并不是唯一可以和 XML 协同工作的语言，您可以使用 Microsoft Visual Basic、Microsoft Visual C++、Perl，甚至 COBOL 来读取 XML 数据。因为 XML 仅仅详述呈现数据的规则，任何程序语言都可以使用 XML 描述资料的能力。

不像 HTML 只有固定的卷标集合，XML 让您可以弹性的建立您自己的标签，您可以任意呼叫您想要的东西。举例来说，您也许会将它称为 **Title** 、 **Heading** 、 **Invoice Number** 等等，XML 卷标通常表现真实的数据。

如果您要移动文本文件的数据，您必须建立您自己的语法和您自己的规格以读取文本文件，并且从中取出数据。如果您使用 XML，您就不需要这么做，您只需要建立格式正确的 XML 文件，然后使用 **剖析器** （parser）读取 XML，并且从中萃取出数据即可。有许多 XML 剖析器可供您使用，在 Microsoft Internet Explorer 中就有一个，而它们通常是免费的。

2. XML 的背景资料

XML 是一种容易学会的语法，如果将 XML 规格打印出来，大概只有 30 页，但是在笔者演讲 XML 时，通常会将大部份的语法整理成两张投影片。当然，这并不包含全部，但是您只要了解一些基本的规则就可以做很多有用的工作了。

记得几年以前，笔者教儿子 Cooper 如何下西洋棋，当时他才七岁，但他很快的学会了。他甚至在即将败北的时候会有十分聪明的表现，他利用国王的垂直面保护自己以避免对方的主教攻击。

当然，他知道他自己的革新下棋法已经不符合西洋棋的规则了，但是他可以很快的了解规则。然而，Cooper 将会不断的学习，如果他终其一生不断的钻研这些简单的规则，终将精通西洋棋。要精通西洋棋不能只会一步一步的移动，而且必须能够预测所有可能的最佳移动方向。

XML 也是一样，本书可以使用少量的页数就教会您 XML 语法的规则，但是您需要累积经验以了解一个项目何时应该为元素、何时应该为属性，或是根本不应该存在。现在我们就来比较 XML 和您已经知道的技术之间的差异，当涉及 XML 应用程序的时候，本书也会教您一些相关的知识。

HTML－优良的素材

Hypertext Markup Language（HTML）是建立 Web 网页的标记语言，同时也是全球通用的标记语言，笔者估算 Web 上大约有 100 兆个 HTML 网页。估计 HTML 网页的实际数量是不可能的，因为一个 Perl 程序就可以依照不同的输入而建立数千个不同的网页。

HTML 小型、具可移植性、能执行于数百万台机器上，并且易于学习。许多销售极佳的书都保证可以很快的让您学会 HTML，而建立和检视 HTML 的工具都很便宜，甚至都是免费的。

HTML 使用二维空间（two-dimensional space，也就是平面）的结构格式化和处理文字，这些结构会呈现在您的 Web 浏览器中。Dynamic HTML（DHTML）模型，内建于 Microsoft 的浏览器中，并且其它卖商也支持该模型，提供了丰富的程序化接口以延伸 HTML 的功能和表现方式。然而，因为浏览器的差异性，许多的功能某些浏览器并不支持。HTML 大部份用于使用二维的方式发布信息：在浏览器中呈现 Web 网页。

Web 使用者从 HTML 中学到的最佳课题就是 超文字 （hypertext）模式真的成功了，任何存取 Internet 的使用者可以在浏览器网页中的连结上按一下，就可以从数千哩以外的 Web 服务器上下载一个网页。过去几年，我们学习了许多关于这种通用存取信息的方式，并且负责标准化的组织也发明了许多存取信息的方式。建立 HTML 网页的方式和连结这些网页的能力使得「Web」成功了！但是 HTML 却有一些问题。

HTML：不好的素材

关于 HTML 的主要抱怨就是它无法执行某种程度的功能，例如精确的空格键控制、定位排列（不使用复杂的 table 元素）等等－网页设计师所需要的功能。 串接样式表 （Cascading style sheets，CSS）的开发是为了用以解决 HTML 格式化的限制，但是这些功能是外挂于 HTML 的基本功能之上，而且每一种浏览器都掌控不同的样式属性。

HTML 是由非常不严谨的建议标准所管理，其中只有一点点的规则是用于处理网页数据，而且大部份的浏览器也不一定可以正确的执行这些处理网页数据的语法。举例来说，段落元素 **<P>** 的结束卷标 **</P>** 是选择性的，这暗示着您不一定要使用 **</P>** 这个结束标签。HTML 程序设计师不使用结束卷标在撰写程序上非常的方便，这样也造成了如果某元素真的需要结束卷标时，除错就非常的困难。例如在 HTML 网页中建立下列的标签：

```
<P>To find out more, <A HREF="more.htm">Go here<A>
```

<A> 卷标是用于建立超级链接的元素，而「Go here」默认值应该为蓝色并有蓝色的底线，按一下这个连结就会在浏览器中载入该档案。在「here」后面的卷标是结束卷标，因此应该要写成 **</A>** 才对。这是一个十分差劲的程序代码范例，其实 HTML 也是有官方的规格，但是如果让浏览器强制遵循该规格，则会引起浏览器发出错误，并且拒绝加载这类的文件。

但是事实并非如此，当笔者在 Microsoft Internet Explorer 4 中观看此网页时，连结看起来十分正常，因此将它发布到 Web 网站上。不久之后，笔者收到一封信，其内容说到任何在「Go here」之后的都变成有底线的连结，而发信人是使用 Netscape Navigator。在 Microsoft 浏览器中的剖析器（剖析器会读取文件并且决定其结构）会确认已经结束了一个连结，而 Netscape 的 HTML 剖析器却没有结束超级链接的标签。

由上述可知，某些浏览器可以接受不好的程序代码、其它的则不行，那么为什么浏览器必须去适应不好的程序代码呢？因为没有任何浏览器制造商会想要让消费者觉得他们的产品无法读取 Web 网页，然而这样却造成了所有的浏览器都在竞争谁能够读取最差劲程序代码的恶性循环。

在这种类型的环境中，程序设计师会继续撰写不好的程序代码，因为撰写的好坏与否并没有差别，因此坏的程序代码招致更坏的程序代码，造成永无止尽的延续。

固定的卷标集合定义了 HTML 以最佳化电子文件的传递，HTML 有元素名称，例如 **P**（段落）、**li**（清单）和 **table**（表格）都是格式化的指令。HTML 没有例如 **invoice number**、**policy type** 和 **blood pressure** 等等的元素名称。如果您想要表达信息的类型，HTML 可能就不是一个好的选择，它无法直接符合您的需求。

因为可用元素的数目十分有限，所以 HTML 的程序设计师会以某些技巧来使用卷标，但是这些卷标对描述数据来说却不是必要的。举例来说，程序设计师不需要任何的定义，却使用 **dl**（定义清单）元素，使用的原因只是因为他知道使用该元素会建立靠左缩排；或者他使用了 **author** 标签，但是并不是要指出网页的作者（这是该标签的目的），而只是为了使用 **author** 标签会建立一条中断线并且将包含的文字变成斜体。

Worldwide Web Consortium（W3C）是一个开发和维护核心 Internet 技术的团体，他们在 HTML 标准化以及在客户端软件（浏览器）严密的剖析 HTML 语法方面做了很多的努力。XHTML 规格将会解决不良程序代码的问题，因为剖析器会拒绝读取格式不正确的文件，这一点对新的文件来说是正面的，但是在整个 Web 都完全的正确结构化以前，有大量的旧程序代码必须要被修正。

SGML－优良的素材

International Organization for Standardization（ISO）建立和维护世界上所有 B2B 的标准。举例来说，您可以在卢森堡的供货商订购一支螺钉，并且在台湾的某公司订购一个螺帽，而标准让双方的订单可以兼容。

Standard Generalized Markup Language（SGML）是一项由 ISO 所拥有的标准，SGML 的建立是用于让不同系统的公司之间共享信息，当 SGML 规格在 1980 年代早期的开发期间，IBM、DEC 和 U.S. Internal Revenue Service（IRS）扮演最主要的参与者，当该标准开发之后，其它大型产业组织和联合的政府机关也都参与了这个规格，而在这当中最大的参与者就是 U.S. Department of Defense（DoD）。

DoD 想要将合约传送的成本降到最低，然而，将目前立约者系统之中的文件转换成新立约者系统之中的文件，有时候需要耗费相当高的成本。因此，DoD 意识到必须有一套标准的文件技术，SGML 就是这种技术。现在合约双方都必须支持 SGML，使用这种标准，旧项目的新立约者也可以轻易的读取这些支持 SGML 的文件。这个标准早期由 DoD 所采纳，而 IRS 派系的某些人也称其为「Standard Government Markup Language」。而后其它的产业纷纷跟进，使用 SGML 为产业之间的通讯方式。

SGML 是第一个允许使用者将数据和处理数据的程序分开的标准化技术。SGML 让使用者可以透过信息分析的程序以找出数据的结构和内容。后来又由该分析方法衍生出一个专有名词称为 文件类别定义（Document Type Definition），DTD 定义了信息的类别，因此每一个 DTD 都

有一套描述数据的自订方式。DTD 使用一套严格精确的语法称为 内容模式 （content model）以指出信息集合中的对象内容。

因为每一个信息集合都有不同的需求和不同的对象，所以描述每一个信息集合的 DTD 也都不同。举例来说，描述飞行器维护程序的 DTD 也许会有例如 **access cover** 、 **procedure** 和 **fuse** 等等的元素；描述训练课程的 DTD 也许会有例如 **objectives** 、 **question** 和 **answer** 等等元素。这两个 DTD 中都没有如同 HTML 一般的格式导向元素，例如 **center** 和 **bold** 。DTD 中并不包含关于元素如何显示的详细信息，因为这样就会限制了 DTD 的优点。当处理数据的软件知道输出方法为何的时候，它只需要加上格式并处理信息中的指令即可。

SGML 让公司组织可以将同一种信息资产套用不同的处理方式，以这种方式处理信息，您的公司组织就可以达到「建立一次、无限发布」的能力。举例来说，因为大部分的课程教材信息都是共享的，所以训练课程的设计者可以使用相同的 SGML 来源以建立学生和教师的课程教材。而在排版方面，建立学生课程教材时只须打印问题，而建立教师的课程教材时就要打印问题和答案。

SGML 文件是由标记和内容组成的简单 ASCII 数据串流，剖析器会读取该文件，并且藉由识别标记来决定这些信息的结构和内容，因为 SGML 文件是简单的 ASCII，所以它具有高可移植性并且可以执行于任何有剖析器的平台上，您将会学到 XML 有许多相同的特性。

因为 SGML 是 ISO 的标准，因此它具有稳定且难以修改的特性。建立每一个 ISO 标准的委员会都必须每五年复检和维护该标准一次，并决定该标准是否仍然需要。如果答案是肯定的，那么就

要检查该标准是否应该更新，但是关于商业信息管理的变更十分快速，可能在五年之内就要经历数次变更，因此这也带给了我们将在下节讨论的 SGML 问题。

SGML：不好的素材

SGML 赶不上 Web 的速度，它是在慢速、昂贵计算机的旧时代中设计的，旧型系统的 SGML 程序设计师设计了一套精简功能的复杂版本－尽可能的让档案越小越好，这个精简版的 SGML 处理过程复杂、建置昂贵，并且使剖析器变得很大而又速度缓慢。

要建立一份 SGML 文件，使用者必须利用 DTD 描述信息的结构，这表示使用者在建立第一份文件之前，必须先耗费大量的时间找出信息的结构，并且使用不是十分准确的语法来定义信息的结构，然后使用者必须在文件处理的过程不断的尝试修改 DTD。您在稍后将会发现除非变更底层结构描述（DTD），否则要验证一份文件的结构是否正确必须耗费大量的时间。

另一个问题是 SGML 由学院派系所拥有，这些学院派系的倡导者较感兴趣的是纯正性而非实用性。在 1991 年到 1996 年间，该团体有一个可以简化 SGML 以配合 Web 需求的机会，许多产业称之为「SGML Lite」－彻底的减少了繁杂的语法以符合浏览器中的需求，然而 ISO 委员会却拒绝进行该简化变更，因为这样会降低 SGML 的功能。

在 Web 上我们所需要的是具可移植性、便宜、快速和简单的新技术（如 HTML），而且这个新技术可以与现存的系统与工具兼容。

我们都需要 XML

SGML 委员会的本意并非将 SGML 的标准现代化，一个类似 W3C 的 Internet 专业组织（新加入的世界标准设定组织）提出了简化 SGML 的版本以达成可以在 Web 上使用的目标，这个精简的版本可以维持 SGML 和 HTML 的兼容性。在 1996 到 1997 年之间，该组织大部份都是经由电子邮件工作，并且最后发展出了一个称为 eXtensible Markup Language（XML）的规格，该组织以 10 个目标设计 XML：

1. XML 应该是于 Internet 上运作，并且简单易懂。
2. XML 应该可以广泛支持多样性的应用程序。
3. XML 应该要与 SGML 兼容。
4. 撰写处理 XML 文件的程序应该十分简单。
5. 在 XML 中的选择性功能应该尽可能的少，理想状况应该为零。
6. XML 文件应该易于读取，并且清楚合理。
7. XML 的设计应该能十分迅速地完成。
8. XML 的设计应该严谨并且简洁。
9. XML 文件应该易于建立。
10. XML 标记应该依重要性最小化。

XML 是一种可以让使用者建立标记语言的语法，一般用于建立标记语言的语言通称为 中继标记语言 （meta markup language）。

XML 是来自 W3C 的技术建议标准，XML 由 W3C 所拥有（不属于任何厂商），这就表示使用者不会受限于任何的平台或处理程序语言。

XML 易学易用，它是一种小型、简洁，并且专门为用于 Internet 上而设计。XML 也是连续性的文本文件，这些档案可以轻易通过防火墙，并且于现存的网络上传送。

XML 同时也是免费的，W3C 维护这个专用术语「XML」商标，并且在他们的网站（ http://www.w3c.org ）提供免费的规格。XML 只是由 W3C 规格所定义的标准语－实际上它本身并不会做任何动作。

XML 的迷思

笔者近几年来已经听过一些相关于 XML 的主张，但是要解释什么「是」XML 不如指出它「不是」什么反而比较有帮助－本节将指出一些错误的观念。

迷思 1：XML 是一种标记语言

实际上 XML 并不能够完全说是一种标记语言，即使「标记」是它名称的一部份。XML 是指定一种语法的标准可以让您建立您自己的标记语言，您所建立的标记语言将取决于您设法达成的工作为何。

假设路上有一块钢铁，如果它以现有的形式存在，那它只是一块钢铁，将不会有任何助益。但是，您可以对这块钢铁进行处理并制造成一辆汽车，然而制造每一辆不同类型的汽车所进行的处理也各自不同。

您可以对同一块钢铁进行不同的处理来制造成一辆脚踏车，使用相同的技术（钢铁），但是采用不同的处理就可以达成不同的结果。现在您就有两种交通工具可以使用了。

XML 就像上述范例中的钢铁一样，它本身并不会做任何动作；如果 HTML 是一辆脚踏车；航空安全维护手册标记语言就是一辆汽车，这些都是 XML 的应用－就如同汽车、脚踏车是钢铁的技术应用一样。

迷思 2：XML 只适用于 Web

XML 被认为是比 HTML 和 SGML 更能够在 Web 上有效率地传递内容的语法，在 XML 发展阶段初期，许多人认为像武器承包商和航空制造商等等组织都会继续沿用 SGML 管理他们的复杂文件资产。

事实上，XML 设计者 Tim Bray 描述 XML 为「on-ramp」（高速公路的驶进坡道），人们可以使用 XML 进入 SGML 这个高速公路，XML 完成文件所需的时间比 SGML 少，也因此能够吸引人们使用这种描述性的阶层式数据标记。

为了建立 XML，开发者是 SGML 为起点，并且去除所有的选择性功能，然后做了一些向后兼容于 SGML 的改进，因此 XML 的使用者也可以使用他们用来处理 SGML 的现存工具。虽然 XML 是 SGML 的子集，但是 XML 中也新增了一些延伸功能例如名称空间和架构等等，造成和 SGML 有某些程度的不兼容。

还有，自从采纳了 XML 之后，许多组织发现使用 XML 就可以完成大部份 SGML 所能完成的工作，大部份的 SGML 使用者根本不会用到在 SGML 中的某些选择性功能，因此在 XML 中没有这些功能并不会有什么重大影响。过去使用 SGML 建置的应用程序现在可以使用 XML 代替。

迷思 3：HTML 是 XML 的子集

车是钢铁的子集吗？错－车是钢铁的应用罢了。同样地，HTML 也不是 XML 的子集，如果 HTML 遵守 XML 中的格式正确条件限制的规则，则它也可以是在 XML 语法中表示的一种语言。

数年前，笔者收到一封电子邮件，信中请笔者寄 XML 卷标的清单给他。当然，这是不可能的，因为 XML 没有定义一套语言，它定义了一种语法让您可以建立您自己的标记语言，您所建立的标记语言可以描述您想要的信息，而 HTML 只是这些信息的集合之一。

3. XML 标准

标准对于信息交换的处理是很重要的，如果没有标准，我们就无法正确的完成商业交易程序。以货币为例，如果没有一个双方都同意的标准，货币的价值可能会专制且波动不定。除非货币建立固定的价值标准，就像一公尺或一公升的标准值一样，否则任何人都可以随意为货币订定价值。

但如果有太多又过于复杂的标准反而会像没有标准一样杂乱无章，在最近的一次会议中，某人对笔者抱怨：「交换信息的标准如此之多，真让我头大！」当然，如果我们全部都选择不同的标准，将会造成我们无法通讯－会陷入一场大混乱。试想：「您可以随意在道路的左侧或右侧开车，天啊，这真是一件可怕的事」。

本章所讨论的团体都是负责开发每一个人都同意的标准（世界通用），他们也发现了订定标准可以建立清楚明了的通讯方式。

ISO

国际标准组织（International Organization for Standardization，ISO）位于瑞士的日内瓦，为全世界的外交合作中心。ISO 由维护数千种规格的国际标准团体所组成，他们让许多事物可以正常工作，ISO 也是联合国中订定标准的一员。

American National Standards Institute（ANSI）是 ISO 在美国的成员之一，并且专门为美国市场设定标准，其它国家中也有相似的订定标准团体。ISO 中的每个成员组织都拥有 ISO 委员会

的投票权，不具备完整开发标准组织的国家在 ISO 中也会有相对的代表性成员，但是这些组织并没有投票权，只会在有相关标准要订定时被予以通知。申请标准的代表性成员是一些经济规模小的国家，他们付出少量的费用给 ISO 以维持和国际标准团体的合约，如您在前几章中所学的，ISO 建立和维护 XML 的前身－SGML。

OASIS

Organization for the Advancement of Structured Information System（OASIS）在 1993 年成立，当时名称为 SGML Open，1998 年该组织变更名称以反映他们支持 XML 和 CGM（Computer Graphics Metafile 标准）。

OASIS 的成立主要是为了两个目标。首先，该组织经由发布市场信息而致力于更进一步的采纳产品独立性的结构化信息系统，同时也扮演结构化信息系统的信息交换角色。OASIS 在许多国际性的会议中都有相当活跃的活动。

该组织的第二个目标是建立可以提供企业建立支持结构化信息系统产品的标准，每次只要有成员建议一种标准，OASIS 就会成立一个技术委员会。举例来说，一个 OASIS 的技术委员会开发了一种注册架构的标准，并将这些信息储存在一种储存机制中，OASIS 历年来已经开发了数种标准，并且都可以公开使用，OASIS 网站位于 http://www.oasis-open.org/ 。

W3C

World Wide Web Consortium（W3C）位于麻州的剑桥，是一个国际性的联合组织，其成员（包含企业）都致力于维护 World Wide Web 标准（您在 第二章 中已经学过一些相关于 W3C 的知识）。

W3C 标准都是全球最后一致通过的精确版本，W3C 规格的订定步骤如下：

1. W3C 的成员传送「提案（Submission）」给 WC3，然后 WC3 判别提案是否符合组织的宪章，接着判别是否应该改良为建议标准。某些企业成员提交提案给 W3C 只是为了在激烈的竞争中脱颖而出，或是为了宣称他们建置了「W3C 提案」。您必须了解到十分重要的一点是：只要是 W3C 的成员之一都可以提交规格，然而，单单提出规格并不表示 W3C 会承认该提案的可行性。
2. 有时候，提案会变成「备忘录（Note）」，备忘录就是载明日期的公开记录，其中包含构想、评论或文件。一旦 W3C 复审备忘录，备忘录就会变成「认可提案（Acknowledged Submission）」，备忘录并不代表 W3C 承诺会在相关备忘录的标准上继续努力，如果某些成员想要让他们的构想被视为备忘录，并且发布于 W3C 网站上，则必须遵守正式的提案程序。
3. 一旦提案变成认可提案，此时就会形成一个 Working Group。这个组织通常由提交草案的代表成员所组成，同时也可能会有其它感兴趣的团体参与。一旦到达了这个阶段，该 Working Group 就会发布 Working Draft，代表工作正在进行，而且 W3C 承诺将会在该领域上继续努力。但是 Working Draft 并不是暗示 W3C 已经一致通过，Working Draft 通常会张贴在 W3C 的 Web 网站，包含公开的技术说明。

这些 Working Draft 有时候是由热切的期盼该规格是否具有可实行性的软件供货商所建置，来自这些软件供货商的信息对 Working Group 来说是十分有价值的。

4. 一旦 Working Group 认为他们的工作已经接近最后完成阶段，他们将会发布「推荐标准（Proposed Recommendation）」，推荐标准代表该他们已经一致通过。Working Group 会传送推荐标准至 Advisory Committee 以供复审。
5. 「建议标准（Recommendation）」代表在 W3C 之中已经一致通过，并且具备 W3C 所有认可的特质，W3C 会慎重考量在建议标准之中指定的构想或技术是否适合于广泛的部署，以及该标准是否会提高 W3C 的声誉。
6. 复杂的规格也许基于最后的考量需要更多的输入（回馈）信息，如果遇到这种情况，委员会就会发布「候选标准（Candidate Recommendation）」，这种类型的标准是为了告知全世界：Working Group 认为其规格大部份都是完整和稳定的，并且使用该规格建置软件也没有风险。Working Group 会邀请使用这些规格的建置者提供一些回馈意见，以评量他们的建议标准应该如何建置或改进最为适宜。然而，候选标准也许会基于 Working Group 接受的回馈信息加以变更，然后再发布最终的建议标准。

W3C 标准

W3C 已经建立的许多建议标准中，HTML 和 CSS 就是最广泛使用的标准。1996 年，一个 SGML 团体提交了一个构想给 W3C，其中包含将 SGML 版本精简化的理念，好让 SGML 可以符合 Web 浏览器和伺服端应用程序的需求。

1998 年的二月，XML 1.0 正式被 W3C 的建议标准所采纳，为了建立 XML 建议标准，W3C XML Working Group 以 SGML 为出发点、移除所有选择性的功能，并且使语法合理化。这些努力的结果造就了小型、简洁，并且高速的剖析器可进行有效率的处理。如同您在前几章中所学的，XML 只提供了一种描述资料的方式，它并不包含处理数据的功能。XML 会一直维持其小型简洁的特色，然而许多使用者发现某种程度的一般性处理是有需要的，所以 W3C 同时也致力于开发使 XML 可以提供更多处理功能性的标准。

在本节中，我们将会看见由 W3C 所建立的 XML 标准，并且学习使用 XML 开发程序更有效率。

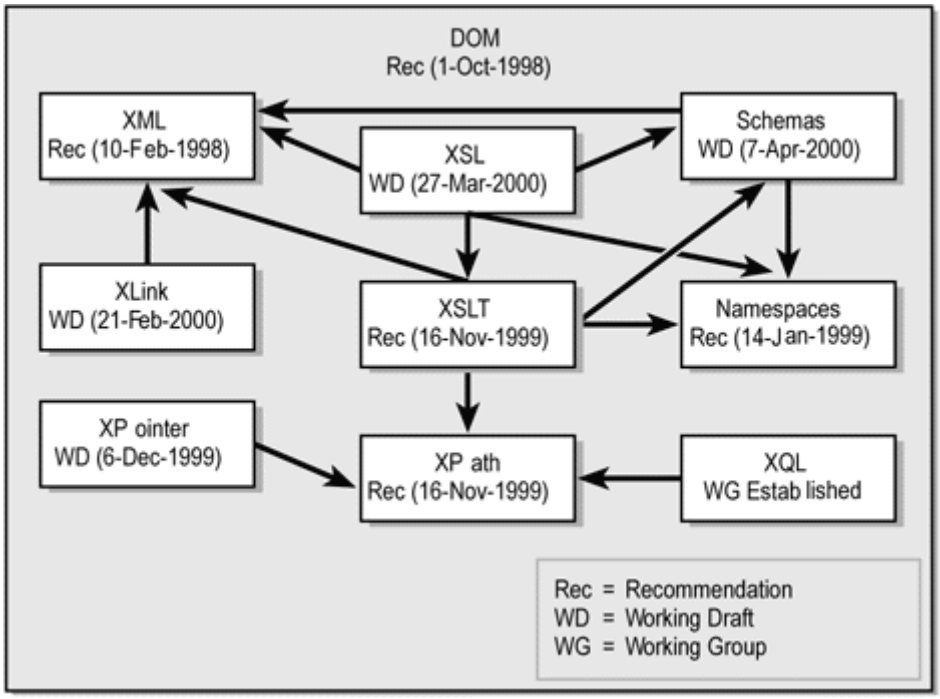


图 3-1　W3C 发布支持 XML 的丰富标准集合。藉由插入支持的规格（DOM），使用 XML 的开发者可以为特定的工作选取特定的技术。

XLink 和 XPointer

为了让 标准 有更佳的处理功能的范例之一就是处理 连结 （Link），XML 是利用阶层式的结构表达信息，一个元素可以包含子层（子元素）、而子层又可以包含另一个子层。一份 XML 的文件结构只会以这种模式描述，然而也会有某些数据结构是非阶层式的结构（一个对象也许会与在另一个位置的对象有关联性），这种类型的关联性必须以连结的方式表达，而不能够以父子式的关联性方式表达。

有两个 W3C Working Draft 提供了表达连结的标准方式：XLink 和 XPointer。以下的范例可以让您充分了解这些规格，试想 HTML 的 **<A>** 元素如下所示：

```
<A HREF=http://architag.com/solutions/senior.html#btravis>
Brian Travis</A>
```

这个元素分成两部分：

- **<A>标签指出连结的语意和处理性质** HTML 剖析器看见 **<A>** 元素就会萃取出 **HREF** 属性中的字符串，而在 **<A>** 卷标和 **</A>** 卷标之间的文字通常在浏览器中会加上底线，浏览器并不会显示位于 **HREF** 属性中的地址子串。

- **HREF 属性指出超级链接的位置** 在 **HREF** 属性中的地址子串由文件名称（通常为 HTML 档案）、选择性的网域名称和路径组成。在本范例中， **HREF** 指示 **A** 元素的处理器使用 HTTP 的 **get** 方法从 http://architag.com 网站取得位于 **solutions** 目录中的档案 **senior.html** ，然后跳至该档案中标记为 **btravis** 的地方。

那么如果我们想在连结中指定不同的语意呢？举例来说，如果要建立一个连结，而这个连结可以让使用者从数个不同的目标中选择一个目的地，在文件中参照 贝多芬 的连结可能会导向至不同

的地方，完全取决于使用者想要看到什么。它可能会导向到贝多芬的传记、贝多芬的交响乐乐谱等等。

如果想要在网页中包含类似的连结，您就需要建立会秀出快显窗口的指令码、以便让使用者选择其中一个选项，然后再选择目的地。XLink 就提供了这个功能，支持 XLink 的浏览器可以分别描述一对多的连结，不需要什么特别的指令码。

XLink 也支持内嵌式连结（inline link），如果想要在文件中包含许多其它文件呢？内嵌式连结让我们可以从许多不同的位置撷取不同的文件，并且将他们都呈现在同一网页中，XLink 就像为 **<A>** 标签打了一针类固醇一样。

同样的，XPointer 就像为 **HREF** 属性打了一针类固醇。在前面的范例中，URL 会指向文件中所引用的地址；而在文件加载后，指针（如范例中的#btravis）会指示浏览器指向文件中标记的位置。

如果您不想看见整个网页呢？如果您只想要网页中包含 btravis 的段落呢？XPointer 允许您将目标设定为只显现网页中的某一部份（这个功能比 **HREF** 属性细腻了许多）。举例来说，您可以使用 XPointer 字符串以指示 XLink 卷标只显示 第二章 中某小节的第三段至第六段。

架构

架构描述信息集合的结构，XML 1.0 指定的架构语法称为 文件类别定义 （Document Type Definition，DTD），DTD 是 SGML 所遗留的规格，其中并没有内建新式的文件处理能力。DTD 专门用来表达文字文件，例如技术手册，商业规划和训练指引等等。然而，当企业处于电子商务的环境中，并且使用 XML 传送实时的数据库数据和商业文件时，DTD 的架构语法就有某种程度上的限制。

在 W3C XML Working Group 发布 XML 1.0 建议标准后不久，即有数个团体提交了延伸 DTD 处理能力的正式提案给 W3C，其中包含 XML Data、Document Definition Markup Language （DDML）、Schema for Object-oriented XML（SOX）和 Document Content Description for XML （DCD）。 第四章 中将详细说明 XML 架构。

名称空间

名称空间会指示 XML 处理器哪里可以找到结构的规则（即为架构）以便套用到文件上，一个名称空间宣告会指向包含架构的资源，而该架构会描述文件。

现在就让我们来看看名称空间是如何工作的，在上面的 XLink 和 XPointer 说明中，曾经提及您可以使用更为精确的方式连结网页（只撷取网页中某个区段的数据），假设您从 Web 上的数个 XML 文件中取得数个不同的区段数据，而这些文件可能会产生名称冲突（使用相同的元素名称），但是这些相同的元素名称却又分别代表不同的意义。举例来说，笔者从 George 的文件中取得三

个段落的数据；从 Martha 的文件中取得四个段落的数据，并且使这些数据都出现在同一个网页上，然后撰写了一个样式表将完成的网页加以格式化，使其显现于浏览器中。

George 在文件中使用 **date** 元素代表一星期中的一天：

```
<date>Feb 13, 2000</date>
```

Martha 也有一个 **date** 元素，但是其代表的意义如下列：

```
<date>Dinner, then a movie</date>
```

如果在相同的文件中包含了这两个元素，文件将会发生名称空间冲突的情形。

名称空间提供了在单一份文件中分别将每一个元素都唯一化的方法，为了避免名称空间的冲突，必须宣告两个名称空间指向不同的架构，这两个架构分别描述两份文件的结构。然后分别在每一个元素之前加上适当的名称空间前置字：

```
<document
    xmlns:george="http://nolprez.com/schemas/schemal.xdr"
    xmlns:martha="http://schemas.martha.com/funstuff.xdr">
...
<george:date>Feb 13, 2000</george:date>
...
<martha:date>Dinner, then a movie</martha:date>
```

```
...
</document>
```

藉由这个方法就可以去除元素名称相同的问题，一旦在这些元素之前加上前置字，也就可以使用样式表建立两个规则分别去识别名称相同的元素，并且应用程序也可以分别处理两个元素， 第五章 将更进一步详细说明名称空间。

XSL、XSLT 和 XPath

在 XML 1.0 建议标准中并没有提供任何标准方式来格式化 XML 文件的内容，因此又发展出 **延伸样式语言** （eXtensible Stylesheet Language，XSL）。W3C XSL Working Group 是由某些想要建立 **文件样式语意规范语言**（Document Style Semantic Specification Language，DSSSL）的人组成，DSSSL 是一项 ISO 的标准，它提供一种可以对 SGML 文件加上样式的语法。然而因为必须对页面的配置加上定义的工作很复杂，所以 DSSSL 语法也是十分复杂的，XSL Working Group 以 DSSSL 为起点开发新的语法，但是修改 DSSSL 的语法需要变更许多部份，所以 XSL Working Group 转移了他们努力的焦点。

XSL 由两个主要功能组成：**XML 文件转换** 和 **格式化对象的解译** 。XSL 的主要目的是转换 XML 文件，使其能以某些格式呈现；另一个功能就是解译格式化对象的字符串－唯一需要这个功能就是当您要建立打印文件的时候（打印，意指印在纸上、张贴于 Web 网站或 CD-ROM 中）。

XSL 标准的功能十分的广泛，因此 XSL 委员会又从 XSL 标准中萃取出转换的部份，并且又建立了新的标准－XSL Transformation（XSLT）。XSLT 也是 W3C 的建议标准，同时许多软件卖商也建立了遵循该规格的 XSLT 处理器－XSLT 最主要的目的是转换引擎，适用于许多地方。

W3C 从 XSL 标准中萃取出 **对应语法** （pattern-matching syntax），并且建立了 XPath 规格。XPath 提供一种方式来描述来源文件的结构，所以您才可以转换文件，第六章 将详细说明 XSLT 和 XPath。

XQL

许多使用 XML 的人也意识到需要一种可以查询 XML 文件的标准方式，这种标准就像查询关系型数据库的 **结构化查询语言** （Structured Query Language，SQL）一样。

目前还没有一种标准的方式用来查询 XML 文件的内容，XML Query Language（XQL）Working Group 现在正在开发一种规格：这种规格提供一种标准化的语法来查询 XML 文件数据库的结构，并且也以一种标准的方式传回结果集。

DOM

以上所提及的所有规格都是为了处理 XML 文件，因此 W3C 成立了一个委员会，目的是要建立一种模式，该模式会描述 XML 文件以开发标准化的应用程序。文件对象模式（Document Object Model，DOM）规格即为该委员会努力的结果。

DOM 提供了一些标准的接口集合，您可以藉由这些接口使用 XML 和 HTML 文件的属性和方法， 第五章 将详细说明 DOM。

IETF

Internet Engineering Task Force（IETF）是一个大型、开放式的国际性团体，该团体是由关心 Internet 基础结构演进的网络设计师、卖商和研究者组成，其目的是为了让 Internet 的运作更平顺，该团体掌控 HTTP、FTP、NNTP 和传输通讯协议。

IETF 和上面所提及的许多标准订定团体的作业方式一样，IETF 的实际技术领域在许多团体中都是顶尖的（举例来说，路由、传输和安全性领域），IETF 每一年都会有三次的会议。

一项规格要变成 IETF 标准的话，必须经历一段开发的周期，并且经由 Internet 团体数次的反复审查，接着基于某些评论和回馈信息反复的修订。经由这些处理程序之后， IETF 中适当的团体就会采纳该规格为标准并发布它，IETF 所掌控的标准之一就是 SOAP， 第八章 将详细说明 SOAP。

开发产业架构

为了让您可以和商业上的合作伙伴顺利地交换数据，在您的组织系统和商业伙伴之间的系统就得使用共通的用语。而在利用 XML 的电子商务环境中，这些用语同时也形成了 XML 的架构。

那么这些架构是谁建立的呢？在最近的一次会议中，笔者提及了架构和商业之间建立交换信息用语的重要性，在会议的中场休息时间，有个人私下发问：「谁能够建立回收帐款产业中的架构？」，这个人正需要一种用语以便和他的客户（银行和金融组织）之间相互通讯，同时他也希望能够有一个委员会可以开发出正式的用语（符合他的产业需求），在告诉您答案之前，让我们先看看到底是什么状况可能会引发这种问题。

大部份的产业都会有他们所属的联合组织，这些联合组织一般都会有代表性的立法机关，标准订定团体等等。举例来说，大部份的商业航空公司和飞行器制造商都属于 Air Transport Association（ATA），甚至彼此之间相互竞争的组织也都会加入这个联合组织，而该组织就会订定一些共享的标准让其成员支持这些标准。

许多年前，ATA 发布了维护飞行器安全的规格－Appendix 100（ATA-100），大部份的飞行器，无论他们的外观看起来多么不同，都有相似功能的组件。来自该产业双方（机身的制造商和飞行器操作员）的 ATA 成员都会遵守称为 Airline Maintenance Manual（AMM）的严密规格。AMM 中包含了标准结构来描述如何维护任何飞行器中的共享组件，其中包含电子设备、引擎、燃料等

等。当一家航空公司购买飞行器时，航空公司的业务代表可以在合约中指定必须符合 ATA-100 的规格，同时这样也可以确保他们能从所有的制造商处取得包含相似结构的文件。

在 1980 年代到 1990 年代之间，ATA 成立了一个项目试图将原本以纸记载的文件规格转换成电子式的文件。最后订定了 ATA-2100 规格，是为描述飞行器的维护程序而制定的架构的集合（使用 DTD 语法）。波音 777（美国航空公司）就是第一台没有纸上文件的飞行器，它使用一系列的电子式媒体储存所有的维护信息。

说明

事实上 ATA-2100 规格是由 SGML 语法定义，但是如果该规格使用 XML 语法的话，概念上大致相同。

美国航空也买了波音系列以外的飞机，并且也要求所有的卖商都提供通用结构的文件，这样做最大的优点就是来自所有供货商的维护数据文件都是相同的格式，而美国航空就可以轻易的将这些文件整合为单一的维护系统。后来其它的航空公司随后跟进，现在大部份的机身制造商都会使用这个开放式标准格式传送文件给他们的客户（航空公司）。

这个程序在许多的产业中都顺利进行，例如汽车工业和 U.S. Environment Protection Administration（EPA）开发的 J2008 架构，该架构描述在制造商和政府规定之间的排放标准和

修复信息。汽车零件供货商和汽车制造商也使用 J2008 规格相互通讯，后来货车产业采纳 J2008 并且开发 T2008 规格，该规格也提供了相似的功能。

所以后来笔者回答提出问题的人：「你就可以啊！」。没有人会比某一产业之中的成员更了解该产业的信息结构和专业术语，那个提出问题的人可以雇用一个熟悉 XML 语法的专家，藉由他和组织中的成员会谈以找出该组织想要表达和传送的数据，并且最后再建立包含通用用语的一或多个架构。但是前提为该产业之中的每一个成员都必须派出各自的代表来决定有哪些需要共享的信息，才能够开发出通用的用语。

4. XML 语法

XML 的语法十分简单易学，您只需要了解其中一些简单的规则，就可以轻易上手。虽然它也有一些比较复杂的规则，但是这些规则并不会很难理解，比大部分的 Internet 技术简单多了。

XML 的文件可以分为两种： 格式正确的 （Well-Formed）和 有效合法的 （Valid）。

格式正确的 XML 文件表示该文件符合 XML 规范中所有格式相关的条件限制。然而，何谓有效合法的XML文件呢？引用XML规范中的定义，即为「如果XML文件由 文件类别宣告（Document Type Declaration，DTD）加以定义，并且遵从 DTD 所定义的规则来表现和限制数据，则该 XML 文件则为有效合法的文件」。您将会在本章的后续讨论中学到这些相关知识。

格式正确的 XML 文件

首先为您介绍格式正确的 XML 文件中有哪些较常用的规则。笔者将在后续的各节中为您详细说明这些规则。

- 每一个元素都必须有一个开始卷标和结束卷标。

- 一份文件中必须有单一、唯一的根元素。

- 元素必须是正确的巢状排列（也就是说，元素之间不得交叉）。

- 某些特定的保留字符不能使用（可以藉由使用其它字符来代替）。

- 属性值必须以引号包含。

- 空元素必须遵守特定的格式。

XML 宣告

格式正确的 XML 文件必须由选择性的 XML 宣告开始。该宣告会为 XML 剖析器定义该文件为 XML 文件，也提供了剖析器相关于文件的信息。XML 宣告如下所示：

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

虽然 XML 1.0 是目前唯一的版本，但是您仍然要宣告版本属性。请注意属性值 **1.0** 包含于引号中，您可以使用单引号或双引号。

文字编码宣告位于版本属性之后，如下所示：

```
encoding="SHIFT_JIS"
```

文字编码宣告指出文件是使用何种字符集建立。默认值 XML 文件是以 Unicode 编码（UTF-8 或 UTF-16）。如果您使用这两种编码方式中的任一个来建立文件，您就不需要指定文字编码属性。

独立文件宣告位于文字编码宣告之后，如下所示：

```
standalone="yes"
```

独立文件宣告使用的属性值可以为 **yes** 或 **no** 。属性值使用 **yes** 会指示剖析器所有相关于文件的信息都已经包含在文件中了，这表示文件中没有指定外部的实体（entity），也没有使用外部的架构（external schema）来指定文件中的预设属性值；属性值使用 **no** 会指示应用程序需要取得文件以外的信息以完成文件剖析。

在 XML 宣告之中必须包含文字编码宣告和独立文件宣告。联合使用这些属性之后，XML 宣告如下所示：

```
<?xml version="1.0" encoding="EBCDIC" standalone="no"?>
```

XML 宣告必须在格式正确文件的第一行，而且其中的文字是区分大小写的。

开始卷标和结束卷标

一份文件会包含文字和标记（markup），标记指出文件的结构以供应用程序处理该文件。标记是由用于描述文件中数据的卷标（tag）所组成。标签必须包含在左右各一个角括号中，成对的卷标中包含了文字数据，并成对使用以分隔数据。

元素（element）是使用来辨别的特殊数据对象，元素也会有属性，属性当中最重要的就是元素名称（元素名称是一个已知数据类型的描述性名称）。您必须使用卷标来指出元素的开始和结束，在 XML 文件中的每一个元素都必须以开始卷标的识别项开始，并且要有一个相对的结束标签结束。结束卷标看起来跟开始卷标十分相似，唯一的区别是结束卷标中的元素名称之前还包含了斜线（/）。在开始和结束卷标中的数据即为元素的内容，内容可以为文字或是其它元素（该元素具有自己的卷标）。范例如下所示：

```
<prologue>We the people</prologue>
```

开始卷标为 **<prologue>** 、内容为「We the people」、结束标签为 **</prologue>** 。三个部分加起来构成一个元素。

元素名称必须以字母或底线开始，在第一个字符后面可以是其它字符、数字、底线或连字号。

说明

许多 XML 1.0 剖析器可以让您使用冒号为元素名称，但是笔者建议您不要使用冒号－冒号是 W3C 建议标准（称为「Namespaces in XML」）的保留字，而 XML 1.0 建议标准采纳「Namespaces in XML」为补增标准。本章稍后将详细讨论名称空间。

根元素

XML 文件必须有单一、唯一的根元素：该元素在文件的顶端有开始标签，并且在文件的末端有结束标签。文件中遇到的第一个开始卷标就视为根元素，如果遇到了该元素的结束卷标，则文件就结束了。如果剖析器在这之后遇到了另一个开始标签，它将会显示错误讯息－因为在根层级只能有一个元素，如果根元素的开始卷标不在根层级，则会产生错误。

区分大小写

在 HTML 文件中，元素和属性名称是不区分大小写的。换句话说，您可以使用下列所有的卷标建立表格： **<TABLE>** 、 **<table>** 、甚至 **<TaBlE>** 。HTML 剖析器将它们都视为一样的标签。

所有的 XML 元素和属性都区分大小写。如果您使用开始标签 **<List-item>** ，则您必须使用 **</List-item>** 为结束标签，而并非 **<LIST-ITEM>** 或 **</list-item>** 为结束标签。

正确的巢状排列

在格式正确文件中的元素必须有正确的树状结构，子元素的结束卷标必须在父元素的结束卷标之前。大多数的浏览器都会忽略这个规则－举例来说，您可以在 HTML 的网页中撰写如下的程序代码：

```
<P>The rights <B>of the <I>individual person</B>
outweigh</I> the rights of the collective.
```

请注意 **B** 元素开始，然后接着 **I** 元素开始。但是 **B** 元素却在 **I** 元素之前结束了。程序代码被浏览器解译为下列结果：

```
The rights of the individual persons outweigh the rights of the
collective.
```

这就是不正确的巢状排列，这种格式的 XML 文件会被剖析器立即拒绝。要使用正确的巢状 XML 文件到达上述的相同效果，程序代码应该如下所示：

```
<P>The rights <B>of the <I>individual person</I></B>
<I>outweigh</I> the rights of the collective.</P>
```

特殊字符

剖析器基于少量的字符判断哪些属于 XML 文件的内容，哪些属于标记。藉由不断的搜寻特定的字符，剖析器可以判断标记和内容的差异。在本章中我们已经使用了三种字符－ 左角括号（<） 、 右角括号（>） 和 斜线（/） 。如果您想在 XML 文件的内容中使用特殊字符，又不想让剖析器将其视为标记的话，您必须使用实体参照（entity reference），实体参照其实是字符串，但是剖析器读取到实体参照字符串时会将其解译为其它的字符。举例来说，如果您在格式正确的 XML 文件数据的某处放了一个左角括号，剖析器会因为无法解译该符号而产生错误讯息。读到了该左角括号后，剖析器无法判断该符号为开始卷标或是结束卷标，请您看看下列的范例：

```
<P>Paul used the caculation A<B, but I don't agree.</P>
```

剖析器将会推测在 B 之前的左角括号为一标签的开始，在读至 B 的时候，判断其为元素名称，然后读到了逗点并传回错误，因为逗点并不是正确有效的名称字符。

如果您希望左角括号在您的文件中被视为一个字符（例如，代表小于的意思），那么您必须使用 &lt 实体参照：

```
<P>Paul used the caculation A<B, but I don't agree.</P>
```

另一个我们要探讨的字符是 & ，请注意实体参照使用 & 为开头的字符，但是如果我们要在 XML 文件中表达 & 这个字符的时候，必须使用 & 实体参照，如下所示：

```
<P>My AT&T mobile phone is working much better now that they put
the cell antenna on my neighbor's birdhouse</P>
```

剖析器可以在格式正确的文件中确认五种实体参照：

实体参照	意义
<	<（小于符号）
>	>（大于符号）
&	&（And 符号）
'	’(单引号)
"	″（双引号）

属性

有时候元素只有名称是不够的，您可以使用属性来描述 XML 文件。属性中包含关于元素的额外信息，属性的表现方式为－在元素的开始卷标中以名称和值的方式成对出现。

```
<insuranceClaim dateFiled="2000-06-24">
```

在上例中，属性名称为 **dateFiled** 、属性值为 **2000-06-24** 。请注意要使用等于符号（=）分开属性名称和属性值，您也可以在等于符号的一或两侧使用空格。

还有一点要注意的是，属性值必须包含在引号中。在 HTML 文件中，如果属性值只有一个字（没有空格）的话，并不需要包含在引号中。但是如果您使用 XML 文件，所有的属性值都要以引号

包含，纵使只有一个字也一样。您可以使用单引号或双引号，只要您是成对使用（不能一边使用单引号、一边使用双引号）。

同时您也必须避免在属性值中使用特定字符。如果您使用单引号包含您的属性值，您可以在属性值中使用双引号为字符；同样的，如果您使用双引号包含您的属性值，您也可以在其中使用单引号：

```
<driveway length="350'">
<gangster name='Wally "Fingers" Gambino'>
```

但是如果您必须在属性值中同时使用单引号和双引号呢？请使用 **'** 或 **"** 实体参照：

```
<irish-gangster name='Shawn "Leaft" O&apos;Doull'>
```

如果您必须在属性值中使用左角括号：

```
<math calculation="A&lt;B">
```

您不能在结束卷标中使用属性。

多年以来，许多 XML 程序设计师（和早期的 SGML 程序设计师）在指定特定的信息为元素的内容，或是为属性的值这个问题上争论不休。以下是当您决定哪一种格式才适合您的考量因素。

- 元素表示的对象如果是购物订单、产品线等等时，您可以将属性视为详细的内容，例如最后一次修改的日期、作者或者货币类型等等。换句话说，将属性想象成元

素的修饰方法，就像形容词修饰名词一样。

- 在语法上属性较有效率。最小的元素有七个字符的负载，然而最小的属性却只有五个字符的负载。当属性或元素有长的名称时，两者之间的差异就变的非常有趣。

- 在 W3C Document Object Model（DOM）规范中属性比元素易于存取；但是在 XSL 规范中两者存取方式一样简单。

- 属性无法有元素结构。如果一个对象有子对象，该对象必须为元素，不能为属性。

空元素

之前笔者曾经提到元素必须有开始卷标、结束卷标以及内容，可是在某些情况下您可能会使用没有任何内容的元素，为什么您会建立没有任何内容的元素呢？您记得在 HTML 中的水平线标签吗？您可以使用该标签在网页上画一条水平线，而水平线标签是没有任何内容的，同时也因为它没有内容，您就不需要使用结束标签，在 HTML 中的水平线标签如下所示：

```
<HR>
```

在格式正确的 XML 文件中，HTML 中的水平线标签仍然适用，但是唯一的例外是您必须加入结束标签，因为剖析器会搜寻 **HR** 元素的结束卷标，您可以使用下列的程序代码以建立格式正确的 **<HR>** 标签：

```
<HR></HR>
```

但是这个标签看起来十分笨拙，因为它并没有表达出元素的用途，为了要解决这个问题，XML 建议标准的程序开发师使用难以了解的 SGML 功能将开始卷标和结束卷标组合成一个卷标－空元素卷标，看起来就像结束卷标一样，除了斜线符号在元素名称的后面，如下所示：

```
<HR/>
```

当 XML 剖析器看到这种类型标签的时候，它就不会搜寻相对应的结束标签了。

空元素也可以有属性，您记得在 HTML 中的 **IMG** 标签吗？该卷标用于在网页中插入图像。纵使 **IMG** 元素使用 **SRC** 属性来指出图像的路径，它仍然为一个空元素。要成为格式正确的 XML 文件，该标签应该如下列方式表示：

```
<IMG SRC="/images/hookah.gif"/>
```

请注意该结束标签的右角括号前面有一条斜线（/ >），斜线和右角括号必须连在一起，它们中间不得有任何字符。

批注

在 XML 文件的任何地方都可以有批注，批注如下所示：

```
<!-- Better check these figures before sending the file -->
```

剖析器会忽略批注，而不会被传送到应用程序。批注提供了隐藏来源文件信息的最佳方式。

格式正确文件的范例

本节提供了一些格式正确文件的范例。下列的第一个范例中描述了计算机硬件中部分的设定项目，您可以从元素名称了解它的涵义，这也是描述式标记的威力！

```
<configuration type="printer">
     <parm name="port">/usr/lpr</parm>
     <parm name="driver">/usr/drivers/HP5SIPS</parm>
     <parm name="option">sheet feeder</parm>
     <configured/>
     <online/>
</configuration>
```

请注意该范例中并没有 XML 宣告，如同您在本章前面学到的，XML 宣告是选择性的。

下列的范例显示了有价值的信息：

```
<?xml version="1.0"?>
<Joke author="Groucho Marx">
     <Setup>Outside of a dog, a book is man's best
     friend.</Setup>
     <Punchline>Inside of a dog, it's too dark to
     read.</Punchline>
</Joke>
```

请注意范例中的 XML 宣告，以及每一个开始标签都有相对应的结束标签。

有效合法的 XML 文件

假设您要提供格式正确的 XML 文件来形容您最喜欢的笑话－您提供了两份文件，一份短的和一份长的文件。

```
<?xml version="1.0"?>
<favorite-joke author="Pate">
    <one-liner>A duck walks into a bar and says to the
    bartender, "Gimme a shot of whisky and put it on
    my bill."</one-liner>
```

```
</favorite-joke>
```

长的笑话比第一个笑话更复杂。

```
<?xml version="1.0"?>
<duck_joke>
    <scene number="1">
       A duck walks into a bar, goes up to the bartender,
       and says, "Don't you have any grapes?" The
       bartender says, "No, this is a bar, of course
       we don't have any grapes."
    </sence>
    <scene number="2">
       The next day, the duck walks into a bar, goes
       up to the bartender, and says, "Don't you have any
       grapes?" The bartender says, "No, like I told
       you yesterday, we don't have any grapes."
       If you come in here one more time asking for
       grapes, I'm going to nail your webbed feet to
       that bar!"
    </scene>
```

```
    <scene number="3">
        The next day, the duck walks into a bar, goes
        up to the bartender, and says, "Don't you have any
        nails?" The bartender says, "No, this is a bar,
        of course we don't have any nails." Then the
        duck says, "Don't you have any grapes?"
    </sence>
</duck_joke>
```

这两份文件都已经包含足够的信息指出每一个元素的意义，第一份文件清楚的描述了作者最喜欢的笑话；第二份文件是一份鸭子的笑话，总共有三种场景。两份文件都是格式正确，而且 XML 剖析器在读取文件时不会产生任何错误。

您可以想象，如果要将许多这一类型的文件集合在一起，然后撰写一支程序将这些笑话加载数据库的话，要怎么做呢？然而事实是：现有的这两份文件对您来说完全没有用处。为什么呢？因为没有一套标准的结构，没有方法可以预测资料将会如何排列，甚至预测标签是如何定义的。因此，您也无法撰写一支可以解译所有不同笑话的程序。

您必须了解，我们可以了解文件中的涵义是因为我们是人类，而计算机却不是，它无法如我们一般思考。在计算机处理数据之前，我们必须辅助它了解数据，如果没有预先定义的正式结构，XML 文件就只是一个纯文字文件而已。

要辅助计算机解译这些文件，您可以事先建立一套定义好的结构信息。计算机就可以利用这些结构信息为建立 XML 文件的准则。然后使用这个兼容于加载数据库程序的 XML 文件。因此我们可以将这个文件的定义信息想象为建立结构的样式准则。

样式准则并不是新的发明，就像记者也有必须遵循报业规则的样式准则一样。样式准则一般而言都有其必须遵循的原则，以记者来说，像是文法风格、避免毁谤和文章的论调等等。样式准则也会指出文章预期的结构，举例来说，指定文章的结构必须有标题、作者署名、日期、摘要和文章主体，还有其主体必须依次包含短评等等。

记者将原稿放在打字机中，并且开始输入文章的主体，她输入了标题、作者署名和摘要，然后输入短评。完成后将纸从打字机中移除，并传送给楼下的编辑。

编辑注意到记者忘记输入日期，她将原稿传回楼上。我们发现了什么？在这个错误中涉及了两个人一昂贵的错误。试想如果有一个守护天使在监控着记者的键盘，辅助她在文件中留下适当的空格，直到她遵守了所有规则才允许她将原稿从打字机中移除。对 XML 的建置来说，这个守护天使正是有效合法的 XML，而这些规则的集合即为 **架构** （Schema）。

文件类别宣告

就如同您在本章开始所学到的，XML 规格中定义的有效合法的文件如下：

「如果 XML 文件由 文件类别宣告 （Document Type Declaration，DTD）加以定义，并且遵从 DTD 所定义的规则来表现和限制数据，则此 XML 文件则为有效合法的文件。」

那么，何谓文件类别宣告？

文件类别宣告会指示 XML 剖析器到哪里搜寻规则的集合以断定文件中的哪些部分必须检查。这些规则的集合又在哪里呢？答案就是已经有某些人将文件的特定类别成员结构加以定义，并且描述这些结构以供剖析器使用。剖析器会读取该结构描述，然后再剖析文件，决定该文件是否格式正确。在 XML 1.0 中，可以藉由文件类别宣告（DOCTYPE）指向规则的集合，文件类别宣告又称为 文件类别定义 （Document Type Definition，DTD）。

请注意您现在应该了解格式正确的文件又有了额外的规则：该文件并不只需要遵守先前所列出格式正确文件的条件限制，同时它的结构还必须遵守使用者定义的结构描述。

该结构描述即为 架构 （Schema）。在 XML 1.0 中，唯一的架构类型即为 DTD。DTD 是直接取自于 SGML 的精简版，并且去除了某些选择性和难以建置的功能。

经过一番解释之后，我们回头看看第一个笑话的 DTD：

```
<!ELEMENT       Joke      (Setup, Punchline)          >
<!ATTLIST       Joke      author    CDATA    #REQUIRED
                          firstTold  CDATA  #IMPLIED  >
<!ELEMENT       Setup     (#PCDATA)                    >
```

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

为了建立有效合法的 XML 文件，而将文件类别宣告放在该笑话文件的顶端以指示包含 DTD 的实体档案－Joke.DTD。该档案应该先读取，并且 Joke 文件应该遵守 Joke.DTD 的结构定义而为一有效合法的文件。

```
<?xml version="1.0"?>
<!DOCTYPE Joke SYSTEM "Joke.dtd">
<Joke author="Groucho Marx">
    <Setup>Outside of a dog, a book is man's
    best friend</Setup>
    <Punchline>Inside of a dog, it's too dark
    to read.</Punchline>
</Joke>
```

本书主旨为叙述 XML，DTD 的详细内容已经超过本书的范围。许多相关 XML 的资源都会告诉您相关 DTD 各个部分的意义和语法。

DTD 主要是为了描述旧型遗留系统的文件信息，笔者在 XML 和 SGML 语法中协同 DTD 工作方面上具有 15 年的经验，然而，DTD 在电子商务交易的应用能力上却有某些因素上的限制。

首先，DTD 并非为 XML 语法使用而撰写，如同您在前面的范例中所见，DTD 使用简洁的语法，语法中奇怪的单字例如 **#PCDATA** 和 **ATTLIST**，这些语法和 XML 的设计目标并不一致。

另一个问题是 DTD 只有一种数据类型： **TEXT** 。我们可以定义包含其它元素或文字的元素。那么我们如何确定文件的内容是来自正确的号码或正确的日期呢？剖析器只确认元素中包含的字符，确认特定的字符串形成正确的日期是应用程序的工作。

但是 XML 文件有一个优点，那就是 XML 在处理数据时是将「数据」和「处理动作」分开的，换句话说，同一份 XML 文件可以同时让不同的应用程序进行处理。所有接触文件的应用程序只要负责确认特定的元素内容或包含特定数据型别的属性值即可。那么，如果在应用程序取得文件之前让剖析器确认数据型别的正确性不是更好吗？

DTD 最后一个主要问题是一份 XML 文件中只能有一个文件类型以描述整份文件，这样相当没有弹性。DTD 是为了官方的文件保值管理系统而建立的，当您需要文件遵守严密的标准时，DTD 是最佳的方式。举例来说，所有的飞机维护手册必须有相同的结构，因为安全性是主要的考量。安规工程师、维护人员、甚至法律部门都必须遵照这个设计，错误的文件设计结构会导致毁灭性的结果。

然而对于商业文件（例如发票或医疗纪录等等），最严谨的设计却无法符合其需求。以发票来说好了，如图 4-1 所示，典型的发票会有标题信息，例如发票号码、日期和客户编号等等，然后在描述订购项目之后的范围也会有买商住址。

我们对于世界上所有的客户都使用同样的发票，请您注意发票上的某些部分是无论我们送到哪里都一样的，客户编号、发票号码和项目等等，对所有的卖商来说都是共享的。然而，在美国、加

拿大、日本等等国家都有不同的住址结构标准，如果我们的发票格式中包含所有国家的住址结构，然后再以 DTD 订定一份文件类型，那么发票的格式可能会十分的交错复杂。

在这里有一个解决方案－发票使用单一的架构，然后每一个国家的住址标准分别使用一个架构，再将每一个国家的架构插入发票的单一架构中。这样不是更好吗？我们可以藉由新的 XML 语法以及名称空间达成这个目的。

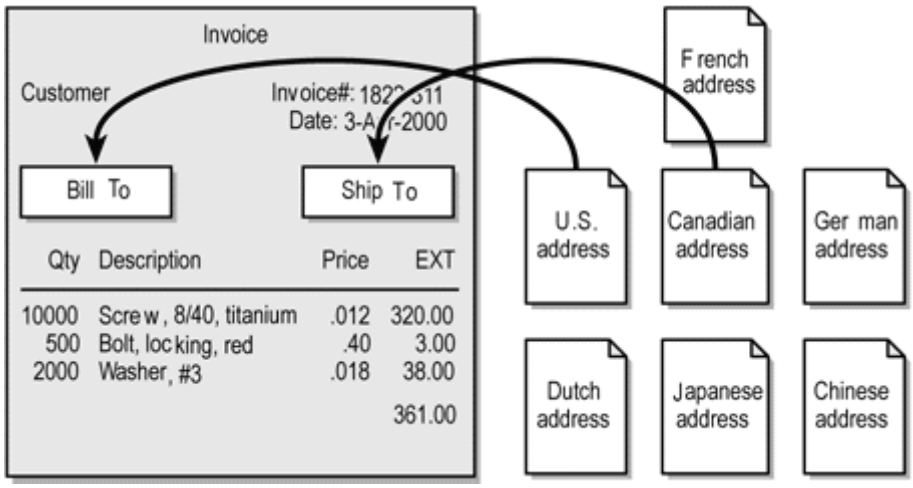


图 4-1 一份商业文件可以包含其它的文件，图中这份发票可以传送至许多不同的国家，其中传送的每一份发票可以使用不同的方式指定实体的住址信息。

名称空间

在说明架构之前，必须先说明 W3C 建议标准中的「Namespaces in XML」，您可以在 http://www.w3.org/TR/REC-xml-names/ 找到相关信息。

名称空间有两个基本目的：架构结构和数据型别的共享，以及在多重架构环境中可以识别元素名称为唯一的。名称空间是设计用于解决数据来源具有相同的名称，却代表不同意义的问题。

您可以藉由宣告名称空间来正确的指出在架构中您所使用文件信息的精确位置，以提供剖析器正确的资源所在位置。

您可以看看清单 4-1 中所举的发票范例，因为该发票是用于书，所以在 **invoice** 元素的 **body** 元素中有一个元素 **tilte** 。然而，在包含地址部分的架构中，也有一个元素名称为 **title** ，该元素的意义是表示欢迎订购该书的读者（MS., Mr., Dr.等等），此时就出现了两个 **title** 元素，却有分别不同的意义。我们想要使用不同的方式处理它们，此时就是使用名称空间的时机。

Invoice.xml

```
<?xml version="1.0"?>
<invoice>
  <number>A-99-1443</number>
  <customer>RA-81122</customer>
  <billto>
    <company/>
    <contact>
```

```
        <title>Mr.</title>
        <name>Barnes</name>
    </contact>
    <street1/>
    <street2/>
    <city/>
    <state/>
    <zip/>
</billto>
<body>
    <line>
        <title>The Fountainhead</title>
        <quantity>600</quantity>
        <price>4.12</price>
    </line>
    <line>
        <title>XML Programming for BizTalk Servers</title>
        <quantity>1000</quantity>
        <price>49.99</price>
    </line>
```

```
    <line>
        <title>1001 Duck-Bar Jokes</title>
        <quantity>1000</quantity>
        <price>6.34</price>
    </line>
  </body>
</invoice>
```

清单 4-1 名称空间的冲突会造成不明确结果的状况，因为两个元素具有相同的名称却表示不同的意义。本例中分别为书的 **title** 和欢迎读者的 **title** 。

我们需要某种方式来独立地指出每一个元素，并且使用分别的方式正确处理该元素，使用名称空间宣告正是这个可以避免名称空间冲突而产生不明确状况的方法。名称空间宣告必须出现在该使用名称空间的元素之前（或同时），名称空间的宣告与开始卷标的属性十分相似。所有的名称空间宣告必须以 **xmlns** 为开头，其后接着一个选择性的区域空间名称。

名称空间宣告属性的值是一个 URL，并且指向定义该文件的架构。

说明

W3C 的名称空间规格指出－「名称空间的名称必须符合其预期的目的，并具备唯一性和永久性的目的。直接取得可使用的架构信息（如果有的话）并不是唯一的目标。为达成这些目标所设计

的范例语法在 **Uniform Resource Names [RFC2141]** 中可以找到。然而，值得注意的是一般的 URL 可以使用这种方式管理以达成这些相同的目标」。

名称空间的范例 **invoice** 开始卷标如下：

```
<invoice xmlns="http://architag.com/schemas/invoice.xdr">
```

读取该发票文件的应用程序会至其列出的资源处取得架构，并且藉由其中包含的规则验证文件的架构。因为该名称空间的宣告位于文件的根元素，因此其作用的范围将会涵盖整份文件－除非又有其它的名称空间覆写原来取得的信息（稍后将会解释这部分）。

名称空间宣告会套用其宣告的元素以及该元素之下的所有子元素，我们可以在文件中任意使用名称空间，但是只有最新定义的名称空间会生效。

还有，请您必须注意一点，XML 是阶层式的语法。文件中所有的元素都在根元素之中，在根元素中的元素都藉由阶层式结构与其它的元素相互关联，因此在父层中的名称空间会套用至子层的所有元素之中，除非有其它的名称空间覆写原来的信息。覆写别人的名称空间宣告会一直保持其效力，直到它所宣告的元素关闭。

现在回到发票范例，我们定义了发票中可以插入住址信息，这种结构称为 开放内容模式（Open Content Model），因为它的结构是开放式的，随后可填入信息，下列的文件是受名称空间宣告影响范围的范例：

Nsinvoice.xml

```
<?xml version="1.0"?>
<invoice xmlns="http://architag.com/schemas/invoice.xdr">
    <number>21153</number>
    <customer>SY221-00</customer>
    <date>2000-01-24</date>
    <address xmlns="http://xml.org/schemas/us-address.xdr">
       <street>750 Davis Street</street>
       <city>San Francisco</city>
       <state>CA</state>
       <zipcode>94107</zipcode>
    </address>
    <body>
       <item num="YI-2289" qty="100" price="23.11"/>
       <item num="WD-7198" qty="35000" price=".137"/>
       <item num="ER-3211" qty="120" price="112.00"/>
    </body>
</invoice>
```

清单 4-2 在发票中的当位置使用名称空间技术插入住址信息。

请注意清单 4-2 的根元素 **invoice** 中包含了名称空间属性，并且其子元素 **address** 中也包含了自身的名称空间信息。如果您检查 **xmlns** 中所指的文件，您会发现在 **invoice** 名称空间中定义了如 **number** 、 **customer** 、 **date** 元素的信息等等；如果您检查 **address** 的名称空间，您会发现如 **street** 、 **city** 、 **state** 和 **zipcode** 等信息。

当您宣告名称空间的时候，在父层范围中的子元素会变成名称空间的一部份。这称为预设的名称空间（default namespace）。但是如果您想要在一个元素范围内混合名称空间时要怎么做呢？举例来说，如果您想要让 **invoice** 元素中的 **date** 元素有不同的数据型别宣告要怎么办？

您可以在 **invoice** 元素的开始卷标宣告其它的名称空间，因为您一次只能定义一个预设的名称空间，因此次要的名称空间必须指定一个名称空间前置字（prefix）。然后您可以在宣告次要名称空间元素的开始卷标中附加此前置字于元素名称之前，直到该元素结束卷标之前都属于次要名称空间的范围。

清单 4-3 显示相同的 **invoice** 元素中有多重名称空间前置字，两个名称空间指出到何处搜寻元素的描述（元素的描述即为架构）。

Mnsinvoice.xml

```
<?xml version="1.0"?>
<invoice xmlns="http://architag.com/schemas/invoice.xdr"
    xmlns:dt="http://schemas.com/schemas/datatypes.xdr">
    <number>21153</number>
```

```
    <customer>SY221-00</customer>
    <date dt:type="date">2000-01-24</date>
    <address xmlns="http://xml.org/schemas/us-address.xdr">
       <street>750 Davis Street</street>
       <city>San Francisco</city>
       <state>CA</state>
       <zipcode>94107</zipcode>
    </address>
    <body>
       <item num="YI-2289" dt:qty="100" dt:price="23.11"/>
       <item num="WD-7198" dt:qty="35000" dt:price=".137"/>
       <item num="ER-3211" dt:qty="120" dt:price="112.00"/>
    </body>
</invoice>
```

清单 4-3　一张发票使用多重名称空间。

请注意次要名称空间的描述在第 3 行，该宣告指定您可以在 **invoice** 元素范围中的任何地方使用该名称空间，但是您必须在欲宣告元素的开始卷标和属性中明确地指定该名称空间。在第 6 行 **date** 元素的属性使用 **dt** 名称空间，本范例中，在 **datatype** 架构中的 **type** 属性设定为 **date** 。请注意 **dt** 名称空间也用于 **body** 元素属性的 14-16 行中。

另一种名称空间的指定方式为在所有的元素中使用名称空间前置字，如清单 4-4 所示，这样可以消除预设名称空间都在一起的缺点。

Prefixns.xml

```
<?xml version="1.0"?>
<inv:invoice xmlns:inv="http://architag.com/schemas/invoice.xdr"
    xmlns:dt="http://schemas.com/schemas/datatypes.xdr">
    <inv:number>21153</inv:number>
    <inv:customer>SY221-00</inv:customer>
    <inv:date dt:type="date">2000-01-24</inv:date>
     <addr:address
xmlns:addr="http://xml.org/schemas/us-address.xdr">
       <addr:street>750 Davis Street</addr:street>
       <addr:city>San Francisco</addr:city>
       <addr:state>CA</addr:state>
       <addr:zipcode>94107</addr:zipcode>
    </addr:address>
    <inv:body>
       <inv:item inv:num="YI-2289"
         dt:qty="100" dt:price="23.11"/>
```

```
      <inv:item inv:num="WD-7198"
        dt:qty="35000" dt:price=".137"/>
      <inv:item inv:num="ER-3211"
        dt:qty="120" dt:price="112.00"/>
    </inv:body>
</inv:invoice>
```

清单 4-4 名称空间前置字可用于任何元素之前，这样会显示哪个元素属于哪个名称空间，但是会增加复杂度。

架构

自从 XML 1.0 建议标准在 1998 年 2 月公布后，W3C 一直致力于取代 DTD 描述架构的其它规格。于是推出了 **XML Schema** 为最后阶段的版本，该标准被视为 2000 年前半年的 候选标准 （Candidate Recommendation）。

W3C XML Schema Working Group 将焦点放在 3 个主要的改善重点：

- 以 XML 语法撰写架构，此目标可让使用者以相同的工具处理 XML 文件。
- 让新的架构标准支持一些共享的数据型别，例如 **number** 、 **date** 和 **currency** 等等，并且可以支持使用者自订的资料型别。这个改良可以使应用程序验证数据的

负担转嫁给 XML 剖析器，所有读取 XML 文件的应用程序可以存取数据。

- 让 XML Schema 具有开放内容模式，试想我们的发票需要有弹性的住址信息。XML Schema 将允许使用者定义发票信息的架构，该架构中有一个称为「hole」的区块可放置住址信息。然后使用者便可以使用描述住址信息的任何架构插入发票的架构中。

说明

W3C 遵守称为 Datatypes for DTDs（DT4DTD）的规格，该规格将新增支持 DTD 的数据型别，为 DTD 语法的补强，可以供想要继续利用 DTD 存取更多数据型别的使用者使用。当然，要以 XML 剖析器必须支持为前提。

W3C 架构也具有 DTD 不具备的其它功能。

XML Data Reduced

Microsoft 为 XML 早期的提倡者，并且已经参与 W3C 的 XML 规格制定一段时间，在此期间也参与许多 XML 相关的社群，并且提出了许多技术展望规格。1998 年 1 月，Microsoft、DataChannel、ArborText 和 Inso 提出代替一项架构的语法，称为 XML Data。该语法达成了上述的 3 大目标，并包含数个新的项目。

1998 年 12 月，Microsoft 公布了 Internet Explorer 5，其中内建 COM 对象剖析器称为 MSXML。MSXML 包含 XML Data 的精简版，称为 **XML Data Reduced** 或 **XDR** 。

纵使 W3C XML Schema Working Group 仍然致力于它们的官方架构，Microsoft 建议 XDR 为 BizTalk 的架构语法。然而，Microsoft 表明一旦 W3C Schema 的语法（Dubbed XSD）成熟就会配合该标准规格。

您应该使用数个月前的 XML Data 架构标准吗？如果您使用目前的 BizTalk 和 XML Data 撰写程序，您未来还需要重写程序吗？答案是「不用」。Microsoft 和其它的卖商将会提供将 XDR 转换为 XSD 语法的工具，因为两种语法有相同的目标，因此在计算机中将一种语法转换为另一种是微不足道的小事。因为所有的语法都是 XML 语法，XSL（Extensible Stylesheet Language）可以用于将一个架构语法转换为另一个。

Extensibility, Inc.有一套称为 XML Authority 的工具（ http://ww.extensibility.com ）也可以让您建立和维护架构。使用该产品，您可以使用图形化方式建立架构而不用担心最终的语法形式。它会强调重要的地方，并且定义逻辑阶层以呈现您的数据，也可以撰写任何架构语法，它甚至可以读取一种语法，并且储存成另一种语法，图 4-2 中显示该工具。

您也可以建立格式正确的文件范例，然后 XML Authority 将会使用该文件建立架构，它也可以从 COM 对象、Java Classes 和 ODBC 数据来源中建立架构。

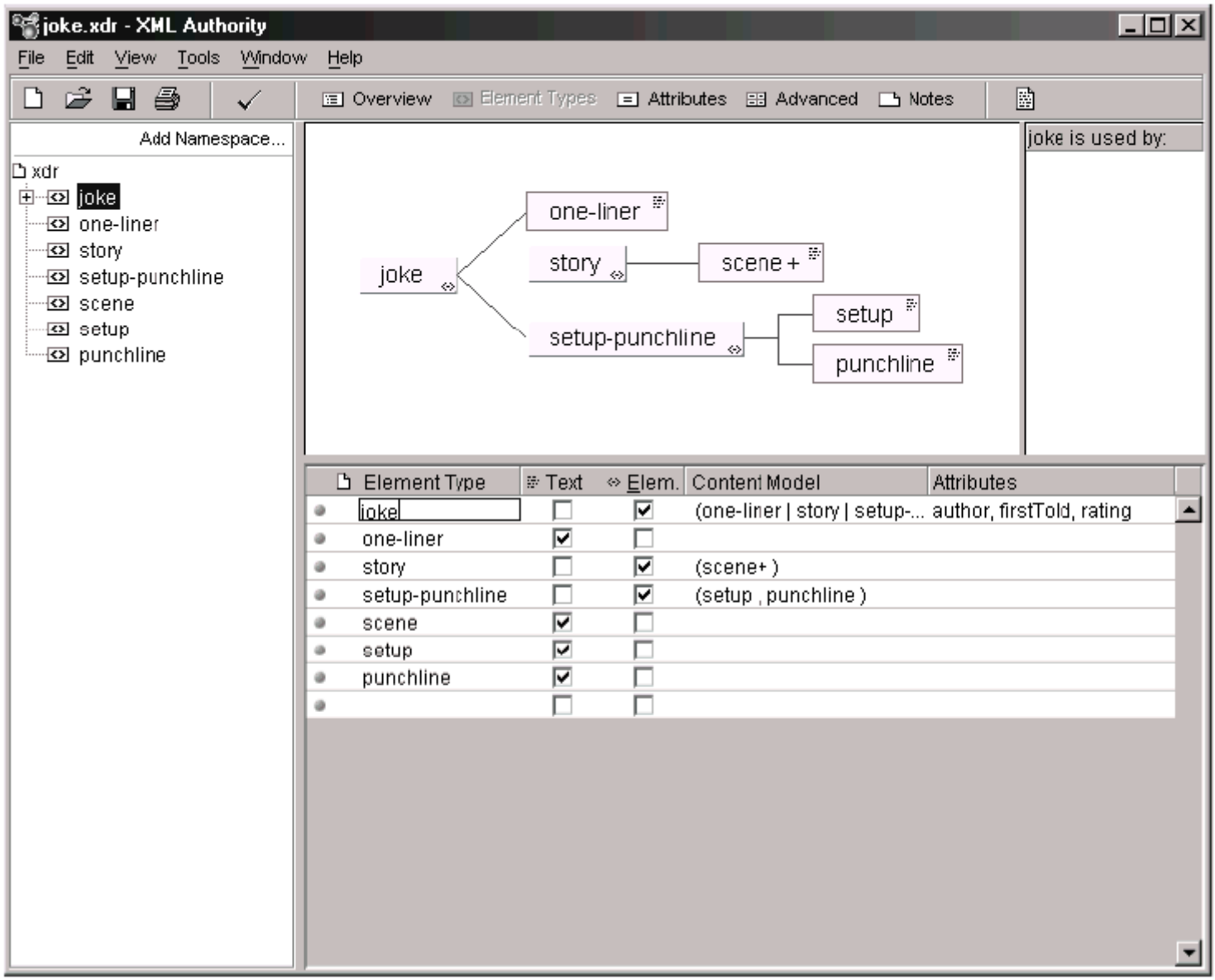


图 4-2　Extensibility, Inc.的 XML Authority 工具

在 Microsoft Developer Center（ http://msdn.microsoft.com/xml ）中有 XML Data Reduced 的语法。清单 4-5 显示了 XDR 架构的范例。

Joke.xdr

```
1 <?xml version ="1.0"?>
2 <Schema name="Joke.xdr"
3          xmlns="urn:schemas-microsoft-com:xml-data"
4         xmlns:dt="urn:schemas-microsoft-com:datatypes">
5         <ElementType name="joke" content="eltOnly" order="one">
6            <AttributeType name="author" dt:type="string"
7               required="yes"/>
8              <AttributeType name="firstTold"
dt:type="dateTime.tz"/>
9            <AttributeType name="rating" dt:type="enumeration"
10             dt:values="G PG R NC-17" required="yes"/>
11         <attribute type="author"/>
12         <attribute type="firstTold"/>
13         <attribute type="rating"/>
14         <element type="one-liner"/>
```

```
15          <element type="story"/>
16          <element type="setup-punchline"/>
17       </ElementType>
18
19       <ElementType name="one-liner" content="textOnly"/>
20       <ElementType name="story" content="eltOnly" order="seq">
21          <element type="scene" minOccurs="1" maxOccurs="*"/>
22       </ElementType>
23
24       <ElementType name="setup-punchline" content="eltOnly"
25          order="seq">
26          <element type="setup"/>
27          <element type="punchline"/>
28       </ElementType>
29
30       <ElementType name="scene" content="textOnly"/>
31       <ElementType name="setup" content="textOnly"/>
32       <ElementType name="punchline" content="textOnly"/>
33 </Schema>
```

清单 4-5　XML Data Reduced（XDR）架构的范例

行	说明
1	该架构为 XML 文件
2	**Schema** 为根元素的开始卷标，该架构的名称包含在 **name** 属性中。
3-4	XDR 架构规格包含两个名称空间，一个为结构名称空间，另一个为定义用于架构中的数据型别。
5-17	**ElementType** 元素宣告一个元素并指出它的内容。在本例中，我们宣告了 **joke** 元素，属性指出了 **joke** 元素只包含元素（ **content="eltOnly"** ），且该元素只能包含一个列出的子元素（ **order="one"** ）。
6-7	属性使用 **AttributeType** 元素加以宣告，此处的 **author** 属性宣告为 string 数据型别（ **dt:type="string"** ）。当 **joke** 元素开始后，**author** 属性为必须的（ **required="yes"** ）
8	**firstTold** 属性被宣告为包含有效日期，**datetime.tz** 数据型别是由 XDR 规格定义，为 ISO 8601 标准的一个子集，该标准用于指定日期、时区。语法看起来像这样：2000-04-25T09:00-08:00。 **firstTold** 属性不是必须的。
9-10	**rating** 属性的数据型别为 **enumerated** ，这表示它只能为在 **dt:values** 属性的值清单中指定的其中之一。在本范例中，该值只能为"G"，"PG"，"R"或"NC-17"中的其中一个， **rating** 属性是必须的。
11-13	每一个元素的属性宣告后，它们必须被置于 **attribute** 元素中。
14-16	**elemen** t 元素指定可以被置于 **joke** 元素中的元素，请记

	得 **ElementType** 的 **content** 属性指定在父层元素内只能包含这三个元素的其中之一。这些元素中的每一个必须使用它自己的 **ElementType** 元素加以宣告。
24-28	宣告 **setup-punchline** 元素，属性 **order="seq"** 指定在此列出的元素必须以它们宣告的顺序出现。列出的元素为 **setup** 和 **punchline** 元素，因此在文件中，**setup-punchline** 元素必须以一个 **setup** 接着一个 **punchline** 元素组成。
30-32	这三个元素只能为文字。

表 4-1　说明清单 4-5 中的架构

W3C XML Schema Working Group 依然不断致力于 XML Schema 的语法，2000 年 4 月，公布了 Working Drafts of XML Schema Parts 1、2 和 3。XML Schema 提供了 XML 1.0 DTDs 特性的超集（superset），BizTalk 率先使用 XDR 为描述 XML 文件的语法。

在 XSD 和 XDR 之间的主要差异在于架构文件的实际语法。然而，两者之概念是相同的。Microsoft 已经允诺当 XSD 成熟时会配合 XSD 各方面的建置工作，无论其为候选标准或最后的建议标准。

5. XML 应用程序

您已经从 前一章 了解了 XML 文件，以及描述其文件结构的架构，现在就让我们建立使用数据的应用程序。本章将带领您使用两种基本的方式处理 XML 文件：内存中的树状对象；当文件经由剖析器和任何附加应用程序处理时产生事件的对象。

文件对象模式

文件对象模式 （Document Object Model，DOM）在内存中将 XML 文件以树状结构方式呈现，并且提供程序设计师易于使用的环境。DOM 提供可存取的对象，使您可以像处理和操作现代化语言的对象一样使用 DOM 提供的对象。

DOM 定义了一组标准的对象集合和您可以操作 XML、存取文件、元素和属性的接口。DOM 可以让您将 XML 文件视为一个对象，藉由使用格式正确的应用程序接口和属性及方法，您可以像存取系统对象一样的存取该对象。

如同您在之前各章中所学到的，DOM 是 W3C 的建议标准。因为 DOM 是属于大型的项目，W3C DOM Working Group 仍然不断面对许多让人气馁的工作，要使用较佳的方式管理项目，该工作小组将工作分成许多小的部分，并且在 1998 年 10 月正式通过了第一部份。

W3C 建议标准当成一般对象模式的蓝图十分有用，但是在定义特定的 DOM 建置方面仍然不够完善。每一个 DOM 的建置都包含文件对象的不同检视方式组成。举例来说，Microsoft 认为应

该包含的某些重要关键性接口却被 DOM 的 W3C 版本排除了。笔者所用的两种方法， **selectNodes** 和 **selectSingleNode** 也不在 W3C 的规格之中。数个剖析器提供者在它们的产品中提供了 DOM 的建置方式，因为每一种剖析器有不同的需求，所以您必须建置不同的环境以符合各种不同的需求。

现在我将描述 Microsoft 建置 DOM 的方式，其建置方式有最完整的文件及最佳的支持。Microsoft DOM 是 Microsoft XML 剖析器对象的一部份，Microsoft Internet Explorer 5、Microsoft Office 2000 和 Microsoft Windows 2000 中都包含了 Microsoft XML DOM 对象，XML DOM 对象也是您可以封装在应用程序中重新部署的对象。DOM 的文件名称为 msxml2.dll，其注册 COM 组件的名称为 MSXML2.COMDocument。因为 DOM 是 COM 组件，您可以在任何启用 COM 的应用程序中呼叫它，同样也可以使用 ActiveX 控件的方式存取它。

XMLDOM 的版本

因为 XML DOM 的开发不断在进行中，或许您会在您的计算机中发现数个 xmldom.dll 的版本。如果您的程序是使用 **MSXML.DOMDocument** 撰写，那么您使用的是 DLL 2.5 版；如果您的程序是使用 **MSXML2.DOMDocument** 撰写，那么您使用的是 DLL 2.6 版；如果您的程序是使用 **MSXML.DOMDocument30** 撰写，那么您使用的是 DLL 3.0 版。在安装了 MSXML 的最新版

本后，您可以执行 **xmlinst.exe** 将您所有的注册项目指向最新的版本。在本章中，笔者将带领您撰写 **MSXML2.DOMDocument** 的程序。

使用 DOM

您可以将 DOM 想象为具有一组接口、属性和方法的动态阶层式对象。请注意一个重点，那就是以计算机的观点看来，我们所呼叫的 XML 文件只是一系列的字节集合。因为该字节集合以简单的文字形式呈现，所以它十分易于读取并可以简单地在网络中或透过因特网传送。

然而，对于操作 XML 文件中信息的计算机来说，您必须藉由高阶的语言将文件转载至内存中的对象以获取较佳的处理方式。您可以藉由建立 DOM 的执行个体（instance）来达成这个目的，该执行个体会呼叫剖析器将 XML 文件分成数个片段。图 5-1 显示建立 DOM 对象的剖析器作业。

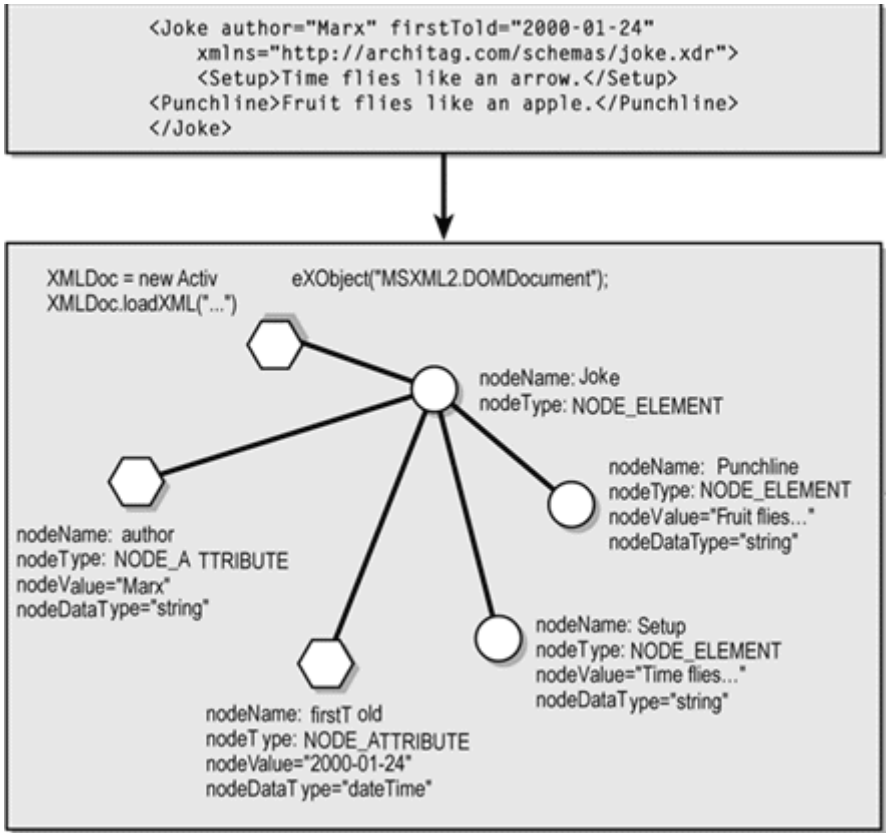


图 5-1　DOM 提供了一组标准的接口，允许程序设计师藉由 XML 数据串流（XML Stream）存取阶层式的对象。

图 5-1 的顶端是代表一个笑话的小型 XML 文件，您可以看到 **joke** 、**Setup** 和 **punchline** 元素，以及 **author** 和 **firstTold** 属性。XML 剖析器一次读取文件的一个字符以决定哪些字符是标记，哪些字符为内容，如果剖析器没有发现架构，即会使用格式正确的 XML 规则；如果剖析器发现了架构，则会读取该架构并且使用它来确认文件符合架构描述的文件结构。

一旦文件正确定义满足剖析器需求，剖析器会建立有正确内容的节点集合（ 第四章 已讨论过 XML 节点）。表 5-1 列出了 Microsoft DOM 建置的节点类型。

表 5-1　Microsoft W3C DOM 定义的节点类型

值	名称	说明
1	NODE_ELEMENT	该节点代表一个元素，一个元素节点可以有下列子节点类型：Element、Text、Comment、Processing Instruction、CDATA-Section 和 EntityReference。一个元素节点可以为 Document、Document-Fragment、EntityReference 和 Element 节点的子层。
2	NODE_ATTRIBUTE	该节点代表一个元素的属性，一个属性的节点可以有下列子节点类型：Text 和 EntityReference。属性不应该为任何其它节点类型的子节点，请注意它不应该为一个元素的子节点。
3	NODE_TEXT	该节点代表一个卷标的文字内容，一个文字节点不能有任何子节点，文字节点可以为 Attribute、Document Fragment、Element 和 EntityReference 的子节点。
4	NODE_CDATA_SECTION	该节点代表 XML 来源中的 CDATA 部分，CDATA 是用于避开由标记所确认的文字区块，CDATA 部分节点不能有任何子节点，CDATA 部分节点可以为 DocumentFragment、EntityReference 和 Element 的子节点。
5	NODE_ENTITY_REFERENCE	该节点代表参照 XML 文件中的一个实体，该节点类型可套用至所有的实体，包含字符实体参照。实体参照节点可以有下列子节点类型：Element、ProcessingInstruction、Comment、Text、CDATASection 和 EntityReference。实体参照节点可以为 Attribute、

		DocumentFragment、Element 和 EntityReference 的子节点。
6	NODE_ENTITY	该节点类型代表扩展的实体，一个实体节点可以有代表扩展实体的子节点（例如，Text 和 EntityReference 节点）。实体节点可以为 DocumentType 节点的子节点。
7	NODE_PROCESSING	该节点代表来自 XML 文件的处理指令_INSTRUCTION（processing instruction),PI 节点不能有任何子节点,PI 节点可以为 Document、DocumentFragment、Element 和 EntityReference 节点的子节点。
8	NODE_COMMENT	该节点代表 XML 文件中的批注，批注节点不能有任何子节点，批注节点可以为 Document、DocumentFragment、Element 和 EntityReference 节点的子节点。
9	NODE_DOCUMENT	该节点代表文件对象，为文件树状结构的根，提供整份 XML 文件的存取。它藉由使用 **ProgID "MSXML2.DOMDocument"** 或使用 **<SCRIPT LANGUAGE=XML>** 或 **<XML>** 数据岛建立的。文件节点可以有下列子节点类型：Element、Processing Instruction、Comment 和 DocumentType。文件节点不能有任何子节点。
10	NODE_DOCUMENT_TYPE	该节点代表文件类别宣告，可以藉由 **<!DOCTYPE>** 指定，文件类别节点可以有下列子节点类型：Notation 和 Entity，文件类别节点可以为 Document 节点的子节点。
11	NODE_DOCUMENT	该节点代表文件片段，文件片段节点_FRAGMENT 联合一个节点或子

		树状结构，此子树状结构中具有没有实际被包含在文件中的文件。文件片段节点可以有下列子节点类型：Element、Processing Instruction、Comment、Text、CDATASection 和 EntityReference。DocumentFragment 节点不能为任何节点类型的子节点。
12	NODE_NOTATION	该节点代表文件类别宣告的注释，注释节点不能有任何子节点。注释节点可以为 DocumentType 节点的子节点。

节点类型决定了每一个节点的属性和方法，举例来说，您可以使用 Serialized（raw text）XML 文件加载文件节点，但是您不能直接加载元素或属性节点。要存取元素节点，您必须首先将文件读入文件节点。

在 图 5-1 中，Joke 有四个节点，前两个节点为元素节点 **Setup** 和 **Punchline** ，后两个节点为属性节点 **author** 和 **firstTold** 。在 **<Joke>** 开始标签中，名称空间指向外部站台的架构，该架构被指定使用 XML Data Reduced（XDR）语法，请注意每一个节点的 **nodeDataType** 属性、 **firstTold** 属性字符串被宣告为 **datatime** 数据型别。如果您没有指定名称空间，所有的 **nodeDataType** 属性都为字符串，藉由存取 **nodeDataType** 的 **typedValue** 属性，该对象将传回的日期值为 date variant，因此您的应用程序不需要验证数据型别或在处理过程中转换值。

您可以在 http://msdn.microsoft.com 找到完整的 Microsoft DOM API，就像大部分的 API 一样，DOM API 十分的完整，允许您执行许多工作，例如加载和储存、建立元素和节点、剖析 XML

文件，但仍然只有数个您可以使用的方法和属性，让您执行每天的例行性工作。现在就让我们看看如何使用这些方法和属性。

建立 DOM 对象

使用 DOM 的主要工作是在您的应用程序中建立一个 **XML 剖析器/DOM 对象** 的执行个体，在 JavaScript 中，您可以使用 **ActiveXObject** 功能函数建立对象，如下列程序代码所示：

```
var objDocument = new ActiveXObject("MSXML2.DOMDocument");
objDocument.async = false;
```

第一行建立执行个体对象，并且建立一个 variant 数据型别的对象称为 **objDocument**，该对象将在您加载文件时包含文件节点，您可以藉由存取 **objDocument.nodeType** 属性值加以测试，本范例中会包含的属性值为 9，可以对应至表 5-1 中的 NODE_DOCUMENT 节点类型。

async 属性指定在对象可供程序设计师使用之前剖析器是否应该加载整份文件，设定 **async** 属性为 **false** 可以确定在文件完全加载之前不可以有任何动作，以便让程序的撰写安全并且简单，但缺点是使您的应用程序速度变慢。当您将该属性设定为 **true** 时（默认值），在下载完成之前呼叫程序就会取得控制权。您可以在之后使用 **readyState** 属性以检查下载的状态，您也可以附加一个 **onreadystatechange** handler 或使用 **onreadystatechange** 事件以通知您待命状态的变更和下载完成。

如果是在加载大型文件的情况下，您也许会想将 **async** 属性设定为 **true**，因为这样您可以在加载对象时继续执行其它处理程序，并且加载会使用分开的执行绪。如果您真的将 async 属性设定为 true，在您存取文件之前，您应该检查 **readyState** 的属性值，表 5-2 描述 **readyState** 的值。

表 5-3 描述两种方式将 XML 数据串流加载对象。

值	状态	说明
1	LOADING	该对象正在读取任何持续性的属性，并不剖析数据。
2	LOADED	该对象已经完成并且正在读取和剖析数据。
3	INTERATIVE	某些数据已经被读取和剖析，以及部分数据集合的对象模式已经可以使用。
4	COMPLETED	文件已经被顺利或不顺利的加载。

表 5-2 **readyState** 属性传回值

方法	说明
load（url）	该方法从 URL 指定的位置加载 XML 文件，如果 URL 无法被解析、被存取，或者无法参考到 XML 文件， **documentElement** 属性会设定为 **null** 和传回错误。
loadXML （xmlString）	该方法使用提供的字符串加载 XML 文件，**xmlString** 自变量可以为格式正确或合法有效的文件。如果在 **xmlString** 中的 XML 无法加载（因为剖析错误）， **documentElement** 属性会设定为 **null** 和传回错误。

表 5-3　将 XML 文件加载 DOM 对象的方法

下列的程序代码将 XML 字符串加载 DOM 对象：

```
objDocument.loadXML("<fact verified='2000-01-24'>Movies are " +
                    "better than books because you can't " +
                    "spill coffee on them.</fact>");
```

一旦该对象加载后， **parseError** 对象应该会被检查，一个被正确剖析过后的对象将会传回 **0** ，如下所示：

```
if (objDocument.parseError.errorcode !=0)
    {
    alert("Error: " + objDocument.parseError.reason +
      " on line " + objDocument.parseError.line);
    }
```

表 5-4 说明只读 parseError 对象的属性

属性	说明
errorCode	十进制的错误码。
url	包含该错误 XML 档案的 URL。
reason	human-readable 形式的错误理由。

srcText	一行包含该错误的完整文字
line	包含该错误那一行的号码（请注意该行的号码是相对于文件的顶端，因此如果您使用文件类别定义（DTD），该号码必须要由 DTD 下面的那一行开始算，而不是文件内容的第一行）
linepos	在错误发生那一行中的字符位置。
filepos	在错误发生档案中的绝对字符位置。

表 5-4　parseError 对象的属性

存取 documentElement

一旦文件正确加载，您就可以开始存取文件的内容了，有许多方式可以达成目的，其中最简单的方法是存取文件元素的属性。下列的程序代码在结果字符串中新

增 **documentElement.nodeName** 和 **documentElement.text** 属性：

```
result += "objDocument.parseError.errorCode: "
        + objDocument.parseError.errorCode + "\n";


result += "objDocument.documentElement.nodeName: "
        + objDocument.documentElement.nodeName + "\n";
```

```
result += "objDocument.documentElement.text: "
        + objDocument.documentElement.text + "\n";


alert (result);
```

任何元素节点都可以有 **nodeName** 和 **text** 属性， **objDocument** 对象是一个文件节点，该节点的 **documentElement** 属性提供了该元素节点，您可以将该元素储存至 varient 数据型别的变量以使程序代码较简单：

```
var rootElem = objDocument.documentElement;
```

包含在 **XMLDOMElement** 对象中的属性是已命名项目的集合。被指定给该元素的属性会包含在集合中。

您可以想象 **attributes** 集合为一数组－其为相像对象的集合，要存取 **verified** 属性值，您需要使用 **getNameItem** 方法：

```
result += "rootElem.attributes.getNamedItem('verified').nodeValue: "
        + rootElem.attributes.getNamedItem('verified').nodeValue +
"\n";
```

要存取文件元素的属性非常容易，但是要存取文件中的其它元素呢？难度只高了一点点，下段将为您详细的解释一些直接存取元素的方法。

取得文件中的项目

让我们加载较复杂的文件，例如 第四章 中的笑话。

```
<?xml version="1.0"?>
<joke type="story" keywords="duck bar grapes nails">
    <scene number="1">
            A duck walks into a bar, goes up to the bartender,
            and says, "Don't you have any grapes?" The
            bartender says, "No, this is a bar, of course
            we don't have any grapes."
    </sence>
    <scene number="2">
          The next day, the duck walks into a bar, goes
          up to the bartender, and says, "Don't you have any
          grapes?" The bartender says, "No, like I told
          you yesterday, we don't have any grapes."
          If you come in here one more time asking for
          grapes, I'm going to nail your webbed feet to
          that bar!"
```

```
    </scene>
    <scene number="3">
            The next day, the duck walks into a bar, goes
        up to the bartender, and says, "Don't you have any
        nails?" The bartender says, "No, this is a bar,
        of course we don't have any nails." Then the
        duck says, "Don't you have any grapes?"
    </sence>
</joke>
```

假设该文件加载我们的对象中并且剖析器传回的 **errorCode** 为 **0** ，您可以轻易的存取任何元素节点的元素和属性，如同您在前面章节所学。但是要如何存取 **scence** 元素呢？我们可以使用 **childNodes** 属性，就可以存取 **scence** 元素以取得我们所需的信息：

```
result += " rootElem.childNodes.item(1).text: "
       + rootElem.childNodes.item(1).text + "\n";
```

item 属性传回子节点，使用 **childNodes** 方法传回的节点集合为从 0 开始，所以此处的范例传回第二个 **scence** 元素中的文字。

您可以透过 **XMLDOMDocument** 对象存取子节点数组，表 5-5 列出可用的属性和方法。

属性	说明
length	传回节点清单中的节点数目，当从父层中新增或删除子层或属性时，该清单长度将会动态的变更。
方法	**说明**
item(index)	在节点清单中使用指定的索引值传回某节点，索引值从 0 开始。
nextNode	基于现有的节点传回节点清单中的下一个节点。
reset	传回 interator 至未建立执行个体的状态－即为在节点清单的第一个节点之前的状态。

表 5-5 XMLDOMNodeList 接口可以使用这些属性及方法

您可以使用 length 属性存取集合中的成员（在一次中）：

```
for (i=0; i < rootElem.childNode.length; i++)
    {
    result += " rootElem.childNodes.item(" + i + ").text: "
          + rootElem.childNodes.item(i).text + "\n";
    }
```

要让之前的程序代码更易于读取，您可以建立一个 varient 变量包含项目集合：

```
var colSences = rootElem.childNodes;
```

```
for (i=0; i < colSences.length; i++)
    {
    result += " colSences.item(" + i + ").text: "
          + colSences.item(i).text + "\n";
    }
```

colScences 变量是一个 DOM NodeList 对象，其中包含根元素 joke 中所有的子元素，使用该对象您可以直接存取 DOM 中的对象。

但如果您想存取 scence s 的其中之一，并且只在其属性包含特定值时才存取呢？这里提供一个方式。

```
var colSences = rootElem.childNodes;
for (i=0; i < colSences.length; i++)
    {
    if (colSences.item(i).attributes.getNamedItem("number").nodeValue
= = "1")
    {
    result += " colSences.item(" + i + ").text: "
                + colSences.item(i).text + "\n";
            }
```

```
    }
```

这个方式可以达成目的，但是很笨拙，让我们看看存取节点还有没有其它替代方式，Microsoft DOM 建置了两个方便的存取方式： **selectNodes(query)** 和 **selectSingleNode(query)** ，如表 5-6 所示。

方法	说明
selectNodes(query)	传回包含查询结果的节点清单，该查询结果是藉由 **query** 字符串指定，并使用现有节点为查询内容基准。如果没有节点符合该查询，会传回空节点清单；如果 **query** 字符串有一个错误，则会使用 DOM 报告错误。
selectSingleNode(query)	传回节点清单中的第一个节点，并使用现有节点为查询内容基准。如果没有节点符合该查询，会传回 **null** ；如果 **query** 字符串有一个错误，则会传回错误。

表 5-6 您可以藉由使用 Microsoft DOM 建置的 **selectNodes(query)** 和 **selectSingleNode(query)** 方法存取一个元素或是元素集合。 **query** 自变量包含由 W3C XPath 规格定义的对应式（pattern）。

被传送至表 5-6 中方法的查询字符串是 Extensible Stylesheet Language（XSL）对应式， 第六章 将讨论 XSL。现在让我们看看使用 **selectNodes** 方法代替 **childNodes** 方法。

```
var colSences = rootElem.selectNodes("sence")
for (i=0; i < colSences.length; i++)
    {
    result += " colSences.item(" + i + ").text: "
          + colSences.item(i).text + "\n";
    }
```

使用 **selectNodes** 方法比使用 **childNodes** 方法改善了许多，而且清楚易懂。 **selectNodes** 方法显然在复杂的文件中比较易于使用，举例来说，您可以藉由使用如下列的 XSL 查询对应式，以便在发票的文件中简单地存取项目集合：

```
selectNodes("/invoice/body/items/item")
```

使用 **childNodes** 方法存取一个项目需要使用相当长的程序代码，然而我们常常需要反复地存取我们想要的节点，要减少大部分的程序代码，您可以使用 **selectSingleNode** 方法：

```
result += rootElem.selectSingleNode("sence[@number='2']").text;
```

XSL 对应式会传回第二个 **scence** ，其属性（@）名称为 **number** ，值为 **2** 。

现在您已经有足够的 DOM 知识建立可以产生实际结果的应用程序。

绝佳演练：在 Visual Basic 中使用 DOM

DOM 提供了 XML 文件的接口，允许您存取文件的内容。因为 Microsoft XML 处理器是 COM 对象，并且支持 W3C DOM，您可以在任何使用 COM 的工具中建立执行个体，因此您可以在 Microsoft Visual Basic 中使用 XML 处理器，如同在本练习中一样，您也可以在 Web 浏览器中撰写于 JavaScript、C++程序中，甚至在 Java 应用程序中使用它。

在本节中，您将建立 Visual Basic 程序使用 DOM 以存取 XML 文件的内容。该程序建立 DOM 对象的执行个体、加载 XML 文件至该对象中、建立元素集合和反复使用该集合。您可以在随书光盘中找到所有的档案，Microsoft 建置的 W3C DOM 包含在名称为 msxml2.dll 的 DLL 档案中，该对象注册为 COM 对象 **MSXML2.DOMDocument**，您可以使用它建立 COM 对象。

1. 载入 mvp.xml 文件至 Microsoft Internet Explorer 5.5，您可以在随书光盘中的 **\Samples\Ch05** 目录找到该档案，加载后画面如图 5-2。

   请注意图 5-2 中显示的是一个格式正确的 XML 文件，其中包含超过 150 个项目。每一个曾经赢得「最有价值球员」的篮球选手都有一个项目，本范例中的每一个项目在 **entry** 元素中都有一个唯一的识别项，以及七个描述该球员的元素。我们的任务是计算每一个球员的年命中率和在窗口中显示年命中率中最好和最坏的球员。

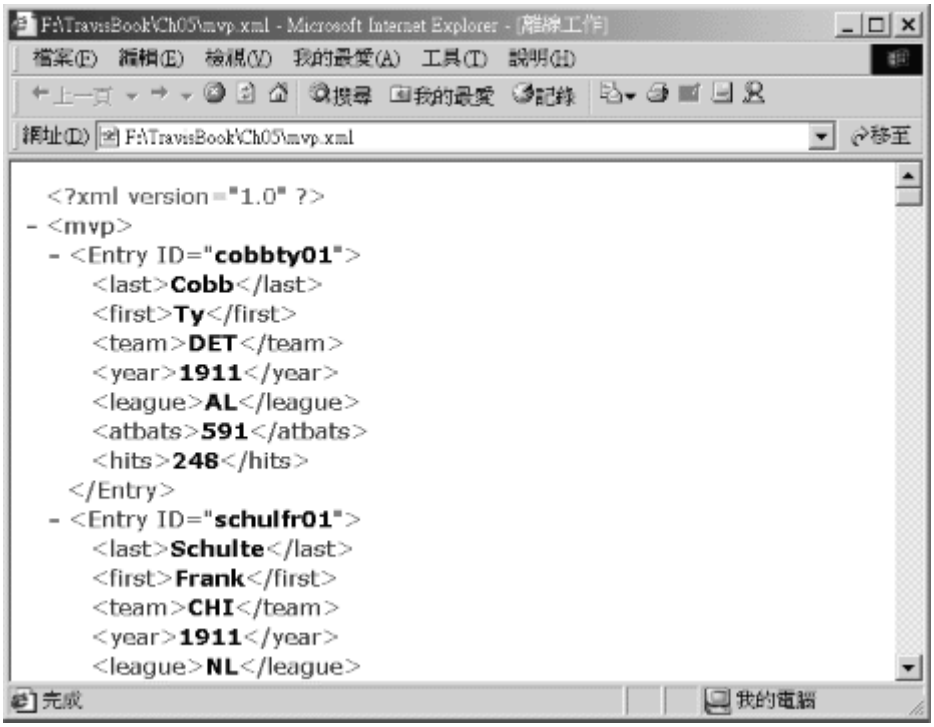


图 5-2　在 Microsoft Internet Explorer 5.5 中的 mvp.xml 档案

2. 启动 Visual Basic，如图 5-3 所示，按一下 **Standard.EXE** ，并且按一下 **Open** 。



图 5-3　Microsoft Visual Basic 6.0

3. 如图 5-4 所示，从 **Project Explorer** 窗口选取 **Form1** ，并且从 **Project** 菜单，选取 **Remove Form1** 。

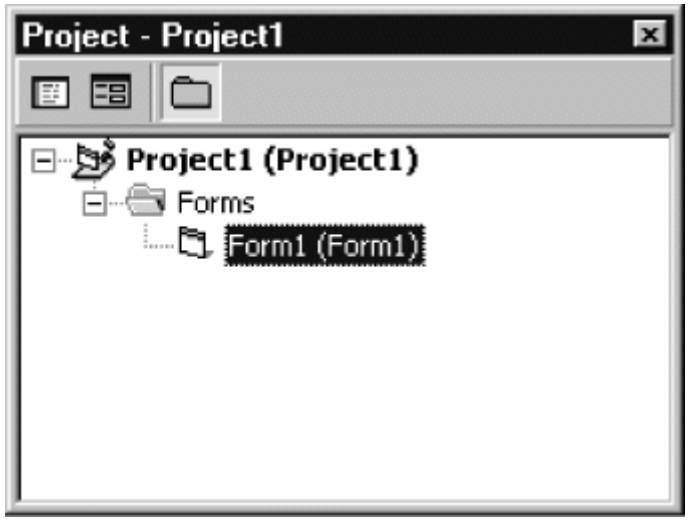


图 5-4　Project Explorer 窗口

4. 选取 **Project** 菜单，选取 **Add Form**，然后选取 **\Samples\Ch05\DOM.frm**。

5. 在 **Project** 窗口中按一下加号（+）以开启该窗体，按两下 **BaseballStats** 窗体，如图 5-5 所示。

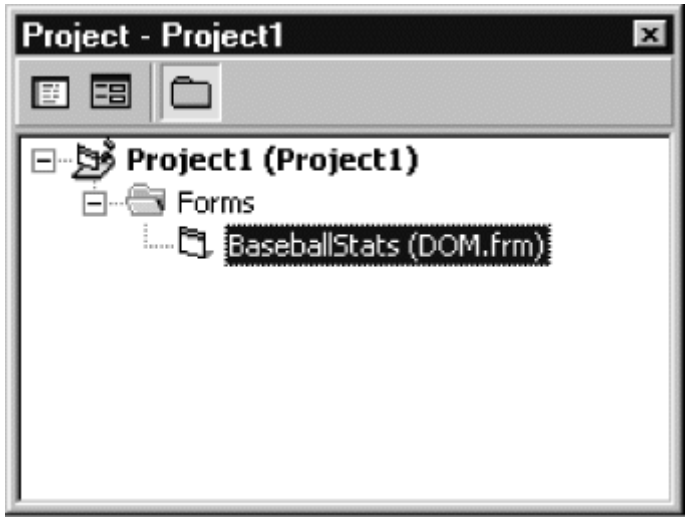


图 5-5　加载 **BaseballStats** 窗体

6. 从 **Project** 菜单中选取 **Project1 Properties**，从 **Startup Object** 下拉式菜单中选取 **BaseballStats** 使其成为启动窗体，如图 5-6 所示，按一下 **OK**。

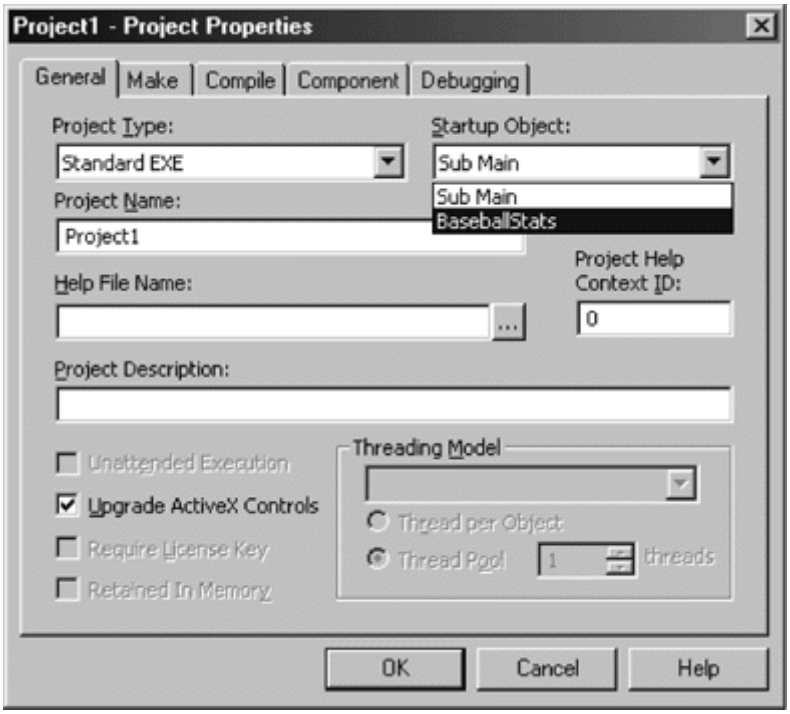


图 5-6　使 **BaseballStats** 窗体成为启动窗体

7. 从 **Project** 菜单中，选取 **Reference**。从 **Available References** 清单中选取 **Microsoft XML, v3.0** 以新增专案中的 **MSXML** 剖析器，如图 5-7 所示，按一下 **OK**。

译者注

如果您在 **Available References** 清单中没有找到 **Microsoft XML, v3.0**，您仍然可以选取 **Microsoft XML, v2.6** 完成本练习；或者您可以安装随书光盘中 **\Software\MSXML30\msxml3.exe** 以新增 **Microsoft XML, v3.0** 。

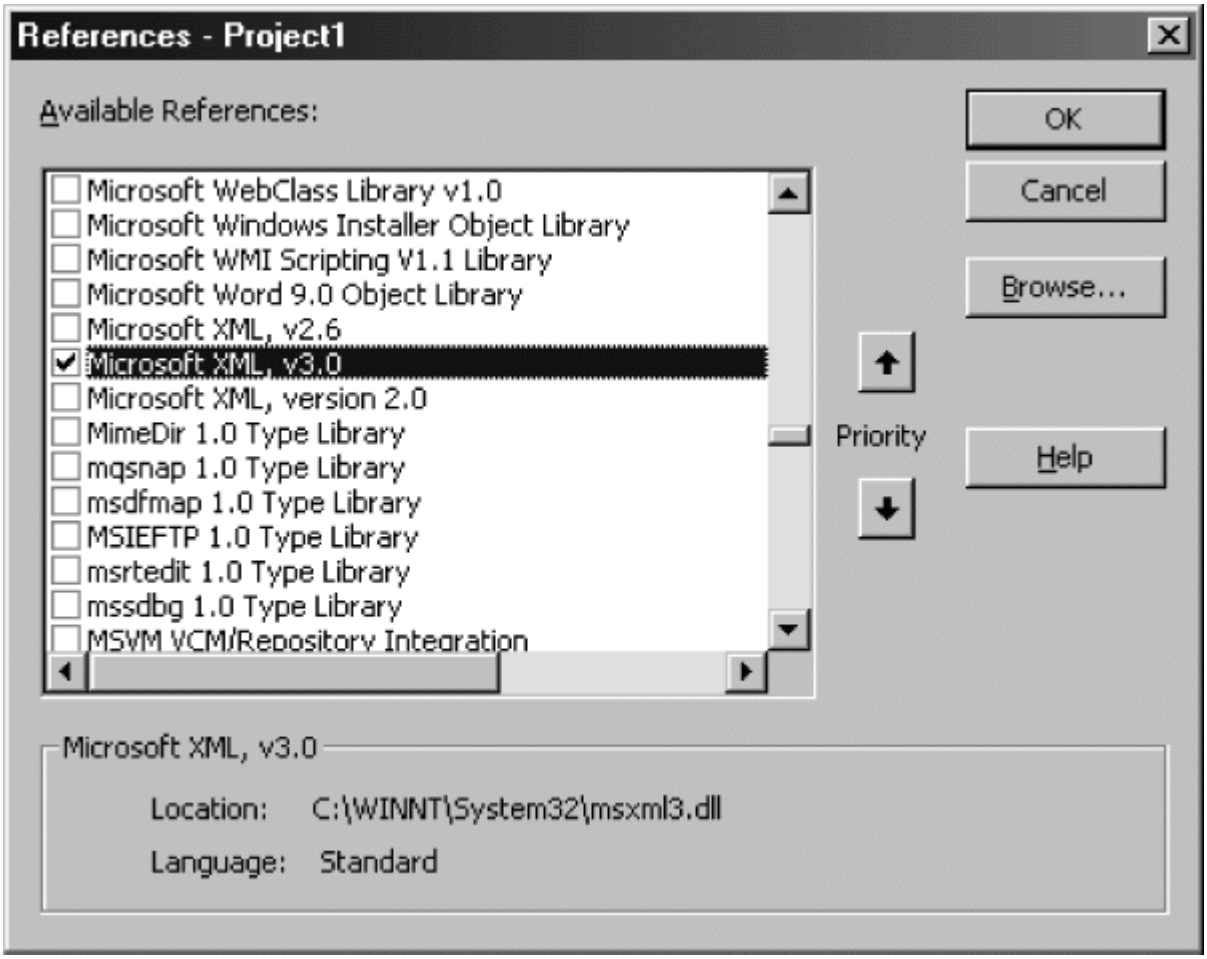


图 5-7　新增 **MSXML** 剖析器至您的专案中。

您可以看见如图 5-8 的环境。

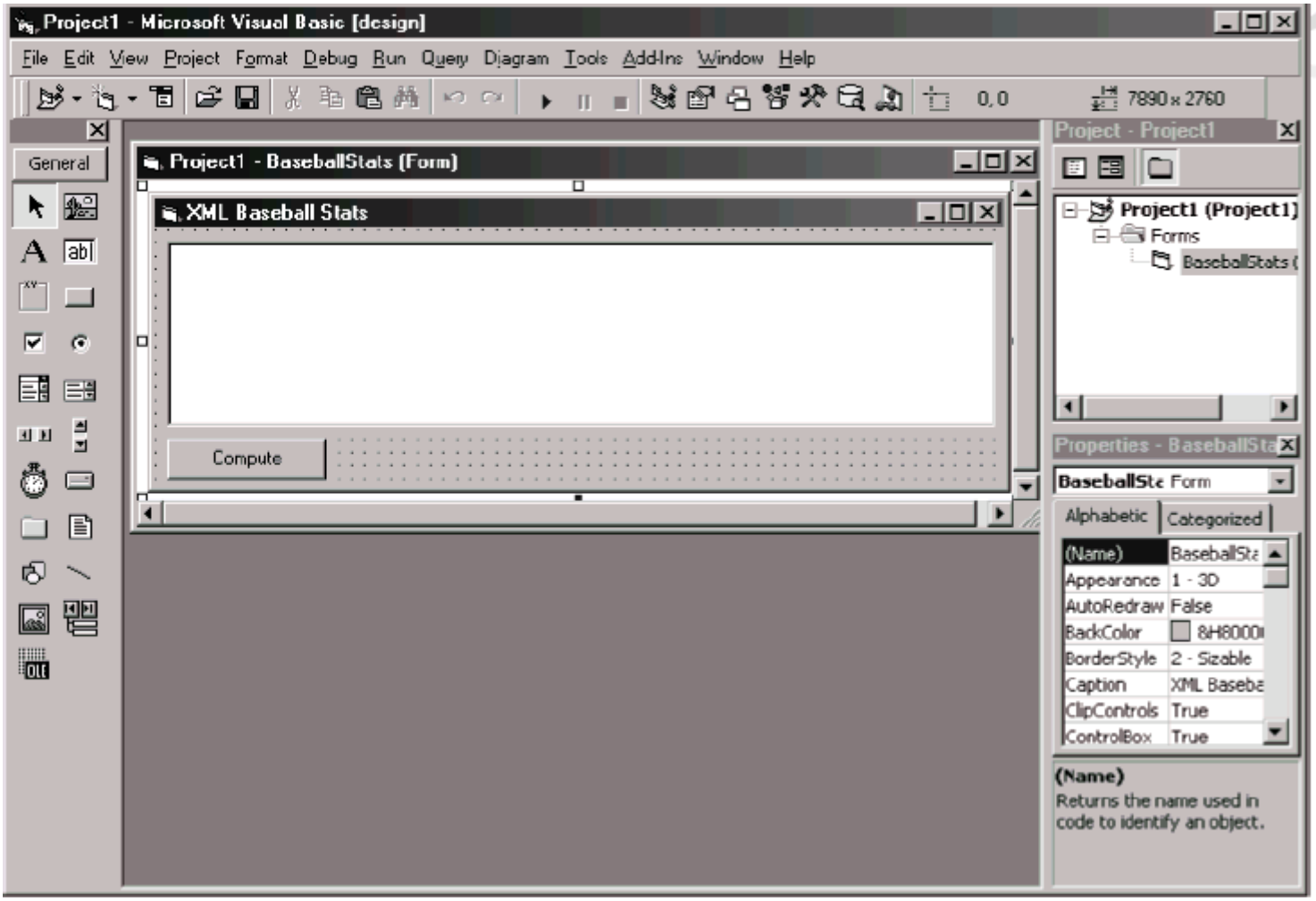


图 5-8 **BaseballStats** 窗体

8. 按两下 **Compute** 按钮旁边的窗体，这样您会看见 **Form Load** 子程序。您必须变更下列程序代码以将 mvp.xml 档案的路径指向您的机器中正确的位置：

```
9.      Set oXMLDoc = CreateObject("MSXML2.DOMDocument")
        oXMLDoc.Load ("C:\mvp.xml")
```

当您输入上列程序代码的第二行时，请注意 **IntelliSence** 处理器会显示所有可用的属性和方法，如图 5-9 所示：

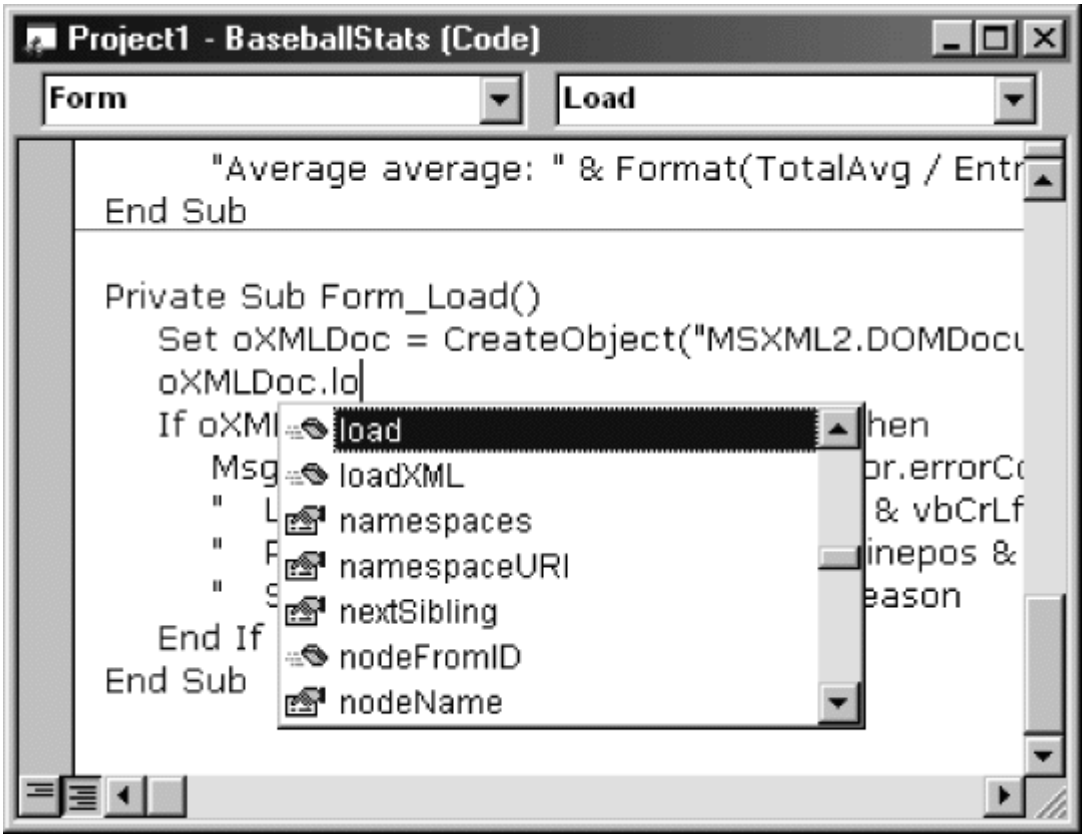


图 5-9 **IntelliSence** 处理器会在快捷菜单中显示所有可用的属性和方法

当您完成之后您的子程序会如图 5-10 中所示（档案路径视您的安装而定）。

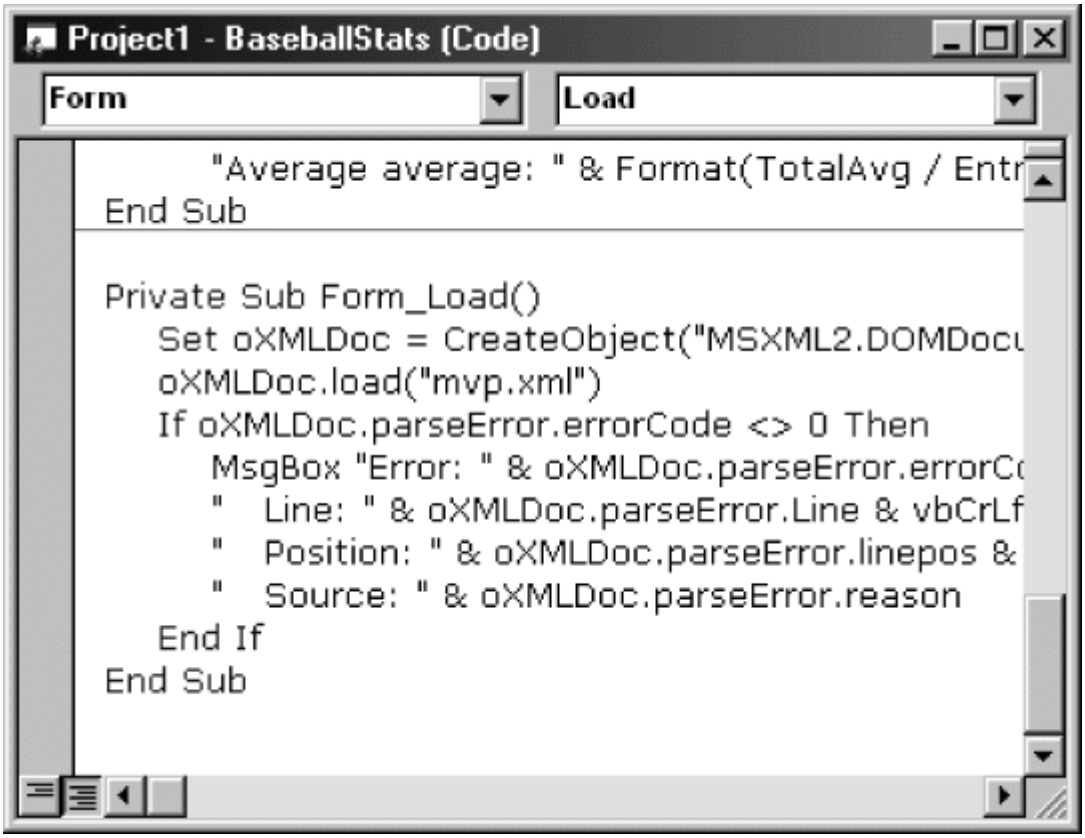


图 5-10 Form_Load 子程序

第一行建立了 Microsoft XML 剖析器的执行个体 Microsoft.XMLDOM，该剖析器为一 COM 对象，您可以在任何程序中使用，试着将 XML 的处理能力想象成操作系统的一部份。

第二行将 MVP 文件加载该对象，使其可供程序设计师用于指令码（script）。

10. 请注意在 **BaseballStats** 窗体的程序代码窗口顶端有一个下拉式清单方块显示 **Form**，按一下该下拉式方块的箭头，并且选取 **Compute**，您会看见如图 5-11 显示的子程序程序代码。该子程序会在使用者按一下 **Compute** 按钮时执行。



图 5-11 **Compute_Click** 子程序

11. 请注意在子程序中第一行是空白的，请输入下列程序代码：

```
Set Entries = oXMLDoc.selectNodes("//Entry")
```

当您输入该程序代码以后，您可以发现 IntelliSence 处理器会辅助您，如图 5-12 所示。

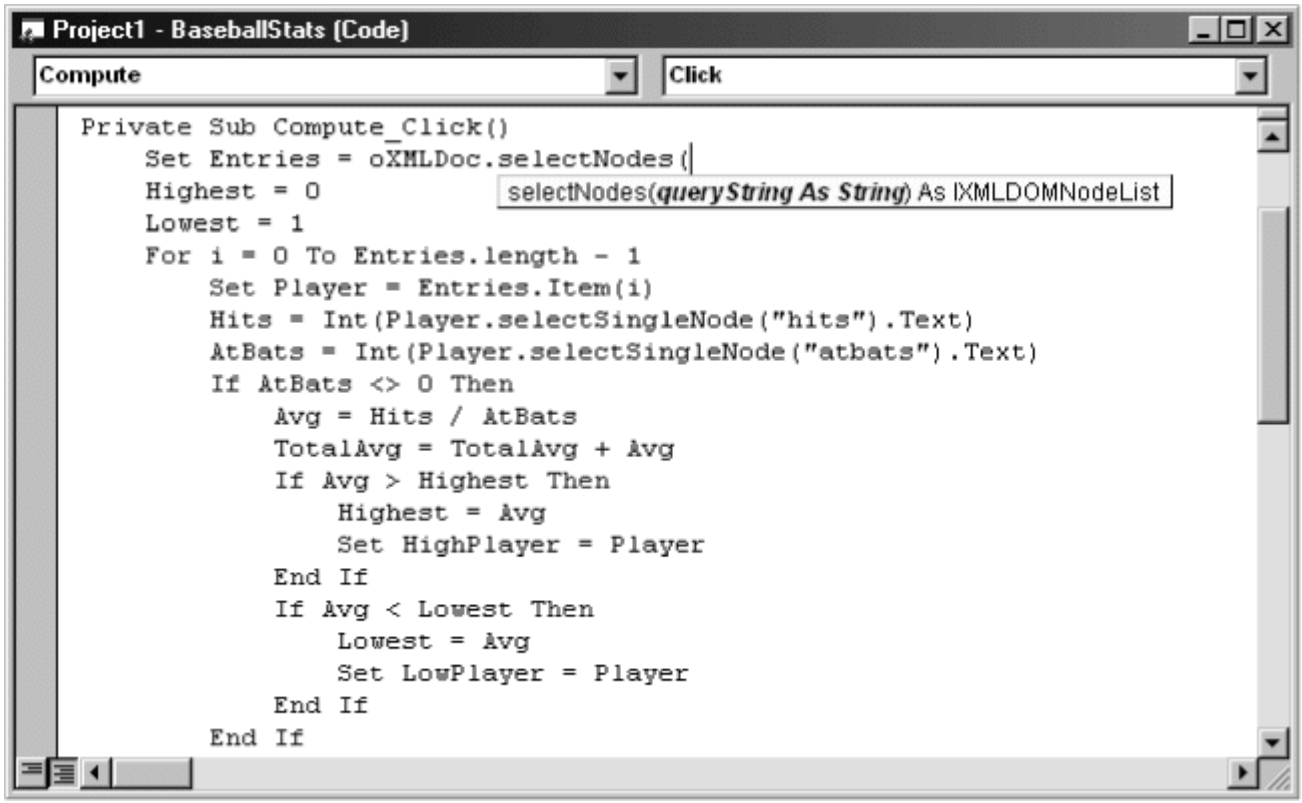


图 5-12 IntelliSence 处理器会辅助您选取 selectNodes 方法的参数。

Compute_Click 子程序建立元素节点的集合，其中包含所有 158 个篮球员的项目，现在我们存取该集合以搜寻哪些球员的命中率最高，哪些球员命中率最低。

12. 11 按 F5 键启动程序，您应该会看见下列如图 5-13 的窗口。



图 5-13　MVP 程序的主窗口

13.　按一下 **Compute** 按钮，您可以看见您的结果，如图 5-14 所示。

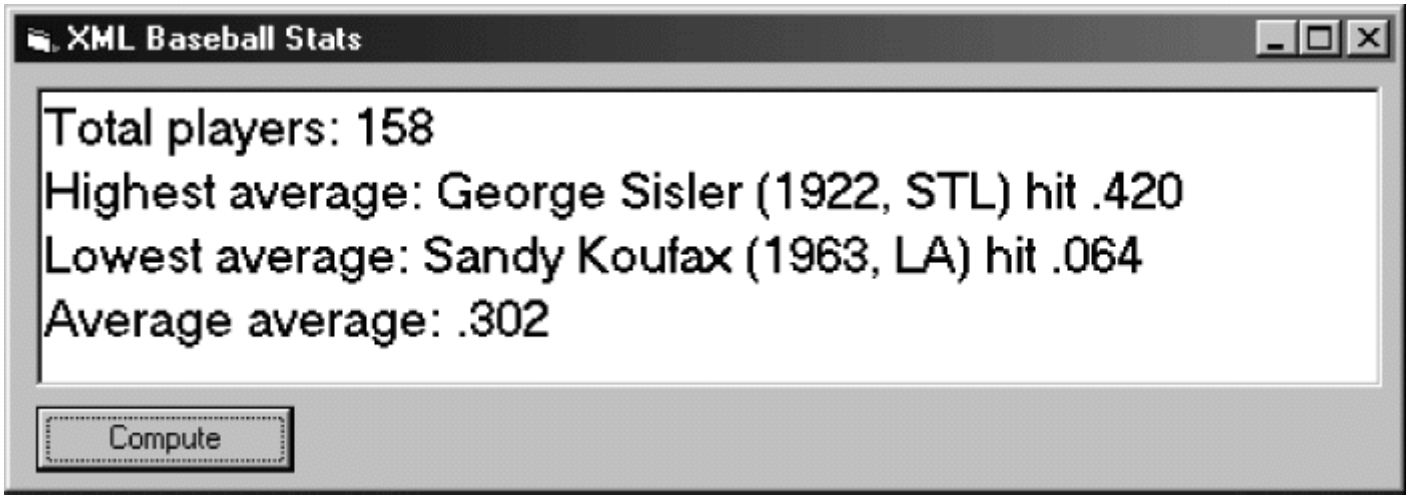


图 5-14　MVP 程序的结果

14.　关闭该窗口，并且将项目储存为 DOM.vbp。

写入 DOM

除了存取 DOM 中的特定属性值以外，您也可以在文件对象于内存中时，新增节点、去除节点和执行其它操作。假设您要建立包含下列结构的 XML 文件：

ToyCarPO.xml

```
<?xml version="1.0"?>
<ToyCarPO_1.0
    poNum="19982"
    poDate="2000-02-21">
    <header>
        <vendorNum>76671</vendorNum>
        <terms>1/10 net 30</terms>
        <shipTo>shipto</shipTo>
        <billTo>billto</billTo>
    </header>
    <body>
        <item needBefore="2000-02-29" needAfter="2000-02-22">
            <partNum>SY120</partNum>
            <qty>10</qty>
            <price>12.43</price>
        </item>
        <item needBefore="2000-03-01" needAfter="2000-02-23">
            <partNum>TQ331</partNum>
            <qty>100</qty>
            <price>0.443</price>
        </item>
```

```
    </body>
</ToyCarPO_1.0>
```

清单 5-1　XML 文件的结构

虽然您可以使用串连技术建立每一个元素，藉由您的程序语言掌控字符串的功能并在程序中连结字符串，但是这样会使得剖析 XML 文件的工作要由您来执行，非常耗时。然而，藉由写入 Microsoft DOM 对象的功能，您就可以省下许多的努力和时间。

在本范例中，您将会建立一份空的发票文件，并且在程序的控制下使用 DOM 以新增项目（我们使用 Visual Basic 建立文件）。这是一个十分简单的 Visual Basic 程序，当窗体加载时开启显示输出的对话框。您可以在随书光盘中的 **\Smaples\Ch05\Struc.vbp** 找到该项目。

首先您需要建立一份 XML 文件的执行个体对象，并且加载它与根元素，表 5-7 描述了清单 5-2 中的每一行。

一旦您建立对象的执行个体之后，您就可以开始新增元素和属性，首先让我们新增数个属性至根元素中，如表 5-8 所示。

buildXML.frm

```
1         Dim xXMLDoc as MSXML2.DOMDocument
2         Dim rootElement
3
4         Set oXMLDoc = CreateObject("MSXML2.DOMDocument")
5         XXMLDoc.loadXML ("<ToyCarPO_1.0></ ToyCarPO_1.0>")
6     Set rootElement = oXMLDoc.documentElement
```

清单 5-2　使用 **loadXML** 以加载 XML 程序代码

行	说明
1-2	宣告 **oXMLDoc** 和 **rootElement** 对象
4	建立 **oXMLDoc** 文件对象的执行个体
5	将根元素加载对象中，建立单一的节点为 **锚** （anchor）以新增其它的节点
6	将根元素参照为 **oXMLDoc.docuemntElement**

表 5-7　建立程序中需要的对象

buildXML.frm

```
1         Dim namedNodeMap
2         Set namedNodeMap = rootElement.Attributes
3            Dim newAtt
4            Set newAtt = oXMLDoc.createAttribute("poNum")
5            newAtt.Value = "19982"
6            namedNodeMap.setNamedItem newAtt
```

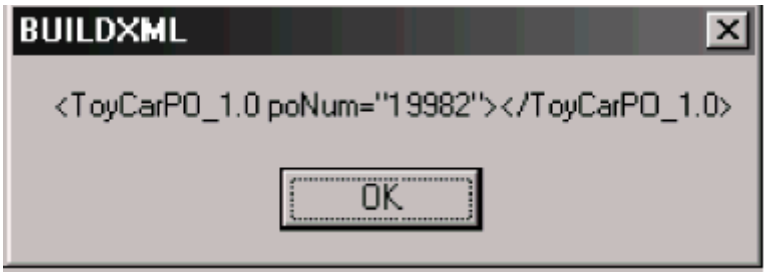
清单 5-3　使用 **createAttribute** 新增元素至文件对象

行	说明
1-2	建立将包含所有属性的节点， **namedNodeMap** 是一个将会包含所有属性对象的对象，其中每一个都会有名称和值。

3-5	建立一个属性对象，并加以命名，在本范例中，我们建立了一个属性名称为 **poNum** ，其值为 **19982** ，当建立一个属性时，名称和值需成对使用。
6	使用 **setNameItem** 方法以新增属性。

表 5-8　新增根元素的属性

如果现在输出我们的 XML 文件，它看起来会像下图：



我们将会建立许多属性，所以需要建立一个可以重新使用的功能函数，并将我们的建立逻辑放在该功能函数中：

```
Public Function addAttribute(oDoc, nodeMap, attrName, attrValue)
    Dim newAtt
    Set newAtt = oDoc.createAttribute(attrName)
    newAtt.Value = attrValue
    nodeMap.setNamedItem newAtt
```

```
End Function
```

当您想要建立属性时，您可以从应用程序中呼叫 **addAttribute** 功能函数：

```
addAttribute oXMLDoc, namedNodeMap, "poNum", "19982"
addAttribute oXMLDoc, namedNodeMap, "poDate", Now()
```

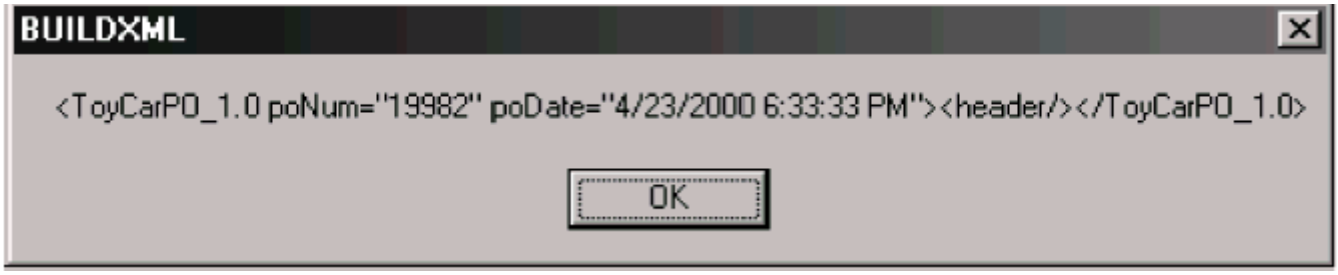
如果现在输出我们的 XML 文件，它看起来会像下图：



现在让我们新增元素至文件中，使用 **createElement** 方法建立元素，建立元素节点之后，然后使用 **appendChild** 方法将该节点附加至我们的文件中：

```
Dim headerNode
    Set headerNode = oXMLDoc.createElement("header")
    RootElement.appendChild (headerNode)
```

如果现在输出我们的 XML 文件，它看起来会像下图：



请注意一旦您建立了一个元素，就必须加入内容，否则它将是一个空元素。您需要新增一些子元素至 **header** 元素中并且加入文字内容。

```
Dim headerNode

Set headerNode = oXMLDoc.createElement("header")

    Dim newChild

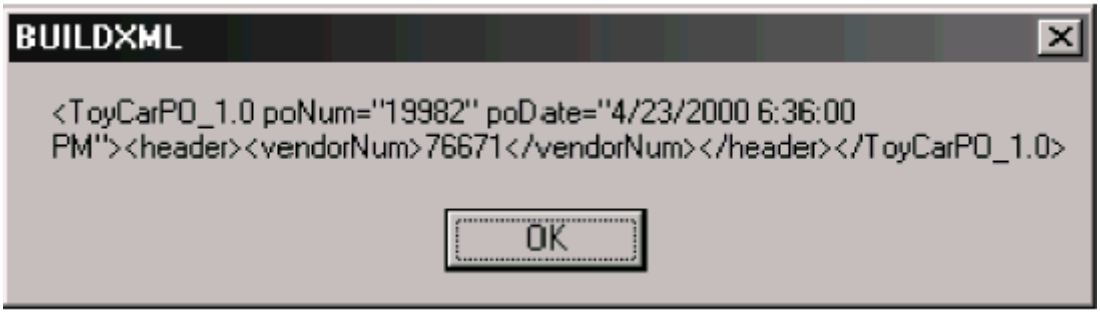
    Set newChild = oXMLDoc.createElement("vendorNum")

    newChild.Text = "76671"

    headerNode.appendChild (newChild)

    RootElement.appendChild (headerNode)
```

newChild 对象被建立的方式如同 headerNode 一样，它是一个元素节点，其名称为 vendorNum 、其值为 76671 。 appendChild 方法新增节点至 headerNode 对象中，然后 headerNode 对象再以另一个 appendChild 方法加入 rootElement 对象中。结果看起来如下图：



我们将会建立许多有内容的元素，所以需要建立一个可以重新使用的功能函数，并将我们的建立逻辑放在该功能函数中：

```
Public Function addElement(oDoc, node, elementName, elementValue)
    Dim newChild
    Set newChild = oDoc.createElement(elementName)
    newChild.Text = elementValue
    node.appendChild (newChild)
End Function
```

于是您可以每一次在 **header** 元素中建立元素时就呼叫该功能函数：

```
addElement oXMLDoc, headerNode, "vendorNum", "76671"
addElement oXMLDoc, headerNode, "terms", "1/10 net 30"
addElement oXMLDoc, headerNode, "shipTo", "Shipto address"
addElement oXMLDoc, headerNode, "billTo", "Billto address"
```

您可以使用相同的程序建立 **body** 元素和其子元素，表 **5-9** 会说明清单 **5-4** 中每一行的意义。

buildXML.frm

```
1       Dim bodyNode
2       Dim itemNode
3       Set bodyNode = oXMLDoc.createElement("body")
4           Set itemNode = oXMLDoc.createElement("item")
5               ' set attributes on item
6               Set namedNodeMap = itemNode.Attributes
7               addAttribute oXMLDoc, namedNodeMap, "needBefore",
                Now() + 8
8               addAttribute oXMLDoc, namedNodeMap, "needAfter", Now()
+ 14
9               ' set children of item
```

```
10                  addElement oXMLDoc, itemNode, "partNum", "SY120"
11                  addElement oXMLDoc, itemNode, "qty", "10"
12                  addElement oXMLDoc, itemNode, "price", "12.43"
13              bodyNode.appendChild (itemNode)
14
15              Set itemNode = oXMLDoc.createElement("item")
16                  ' set attributes on item
17                  Set namedNodeMap = itemNode.Attributes
18                  addAttribute oXMLDoc, namedNodeMap, "needBefore",


                    Now() + 7
19                  addAttribute oXMLDoc, namedNodeMap, "needAfter",

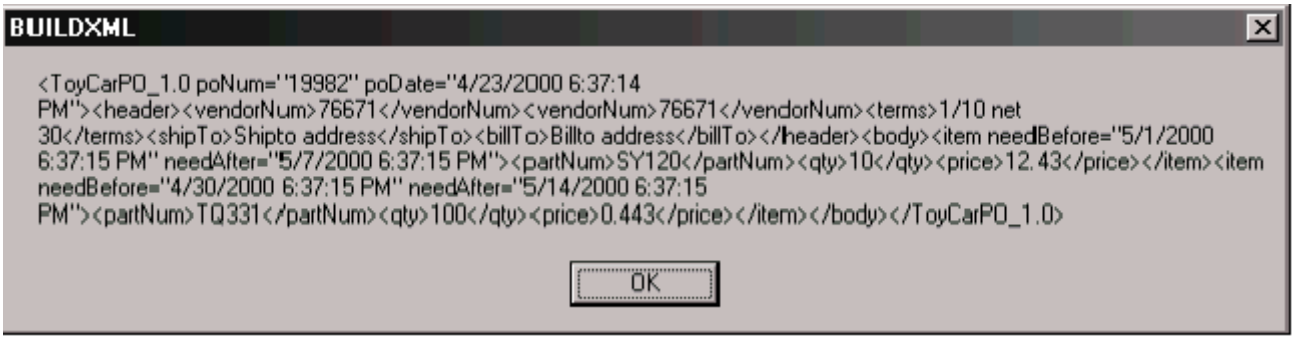

                    Now() + 21
20                  ' set children of item
21                  addElement oXMLDoc, itemNode, "partNum", "TQ331"
22                  addElement oXMLDoc, itemNode, "qty", "100"
23                  addElement oXMLDoc, itemNode, "price", "0.443"
24              bodyNode.appendChild (itemNode)
25          rootElement.appendChild (bodyNode)
```

清单 5-4　使用 createElement 方法建立 body 元素。

行	说明
1-2	宣告 bodyNode 和 itemNode 对象，这些对象将掌控您建立的元素对象
3-4	建立节点的执行个体，并且使用 createElement 方法加以命名
7-8	新增 item 元素的属性，请注意我们使用 Attributes 属性建立 namedNodeMap 为 itemNode 对象的一部份。我们藉由使用 addAttribute 功能函数以新增所有的属性。
10-12	addElement 功能函数新增 itemNode 元素的子元素，并且加以命名和提供其值。
13	最后，我们使用 appendChild 方法以新增 item 节点以及其 3 个子节点（ partNum 、 qty 和 price ）至 body 节点。
15-24	使用相同的技术新增其它的 item 节点。
25	使用 appendChild 方法新增 body 节点至根元素。

表 5-9 说明清单 5-4，建立 body 元素和其子元素

您现在可以看见完成的文件，如下图：



清单 5-5 显示整个 Visual Basic 程序。

buildXML.trm

```
Option Explicit

Public Function addAttribute(oDoc, nodeMap, attrName, attrValue)

    Dim newAtt

    Set newAtt = oDoc.createAttribute(attrName)

    newAtt.Value = attrValue

    nodeMap.setNamedItem newAtt

End Function

Public Function addElement(oDoc, node, elementName, elementValue)

    Dim newChild

    Set newChild = oDoc.createElement(elementName)

    newChild.Text = elementValue

    node.appendChild (newChild)
```

```
End Function
Private Sub Form_Load()
    Dim oXMLDoc As MSXML2.DOMDocument
    Dim rootElement


    Set oXMLDoc = CreateObject("MSXML2.DOMDocument")
    oXMLDoc.loadXML ("<ToyCarPO_1.0></ToyCarPO_1.0>")
    Set rootElement = oXMLDoc.documentElement
    Dim namedNodeMap
    Set namedNodeMap = rootElement.Attributes
        addAttribute oXMLDoc, namedNodeMap, "poNum", "19982"
        addAttribute oXMLDoc, namedNodeMap, "poDate", Now()
    Dim headerNode
    Set headerNode = oXMLDoc.createElement("header")
        addElement oXMLDoc, headerNode, "vendorNum", "76671"
        addElement oXMLDoc, headerNode, "terms", "1/10 net 30"
        addElement oXMLDoc, headerNode, "shipTo", "Shipto address"
        addElement oXMLDoc, headerNode, "billTo", "Billto address"
    rootElement.appendChild (headerNode)
        MsgBox (rootElement.xml)
```

```
Dim bodyNode
Dim itemNode
Set bodyNode = oXMLDoc.createElement("body")
    Set itemNode = oXMLDoc.createElement("item")
    ' set attributes on item
    Set namedNodeMap = itemNode.Attributes
    addAttribute oXMLDoc, namedNodeMap, "needBefore", Now() + 8
    addAttribute oXMLDoc, namedNodeMap, "needAfter", Now() + 14
    ' set children of item
    addElement oXMLDoc, itemNode, "partNum", "SY120"
    addElement oXMLDoc, itemNode, "qty", "10"
    addElement oXMLDoc, itemNode, "price", "12.43"
    bodyNode.appendChild (itemNode)
    Set itemNode = oXMLDoc.createElement("item")
    ' set attributes on item
    Set namedNodeMap = itemNode.Attributes
    addAttribute oXMLDoc, namedNodeMap, "needBefore", Now() + 7
    addAttribute oXMLDoc, namedNodeMap, "needAfter", Now() + 21
        ' set children of item
        addElement oXMLDoc, itemNode, "partNum", "TQ331"
```

```
            addElement oXMLDoc, itemNode, "qty", "100"
            addElement oXMLDoc, itemNode, "price", "0.443"
        bodyNode.appendChild (itemNode)
    rootElement.appendChild (bodyNode)
    MsgBox (rootElement.xml)
    End
End Sub
```

清单 5-5　使用 DOM 接口和 Visual Basic 建立的 XML 文件

事件驱动模式

DOM 提供了一套易于使用的界面以处理 XML 文件的内容。然而，有时候使用 DOM API 也不一定是最佳的方式。举例来说，假设您有大型的 XML 文件需要处理，因为 DOM 是属于树状结构的 API，您必须在整份文件都顺利读入对象之后才可以对文件进行处理动作。而事件驱动的 API 可以直接传回剖析事件结果至呼叫的应用程序，节省了大型文件的处理时间。

假设您只需要文件顶端的前五个元素，则加载整份文件再进行处理会浪费许多时间。而事件驱动模式允许您实时存取剖析器在处理时所遇到的元素，于是您就可以利用这种方式存取元素，无论是否有错误潜藏在文件中。请您记住很重要的一点，就是在内存中的 DOM API，即使错误是发生在文件中的最后一个元素也会造成整份文件的错误。

使用事件驱动模式这种方法的工具会产生事件，而事件是藉由规则产生的，然后依照这些规则处理在文件中遇到的元素。

有一个事件驱动模式已经完成了，称为 SAX，为一个 XML 使用的简单 API。SAX 是由 XML-DEV 中的成员所开发，并由 OASIS 主导。从某些方面来说，SAX 应该算是 DOM 的竞争对手，然而，在某些情况下，使用 DOM 比 SAX 适合，反之亦然。事实上，Microsoft 已经开始于 Microsoft XML Parser（MSXML）中建置一些 SAX 的功能，您可以在 http://www.megginson.com/SAX/ 找到更多相关 SAX 的信息。

6. XSL

Extensible Stylesheet Language，或称之为 XSL，是一种用于转换和格式化 XML 文件的语言。转换意指变更 XML 文件为其它的文件；格式化意指可视化或呈现 XML 文件。您可以使用 XSL 样式表转换 XML 文件为 HTML 格式来呈现数据，或是将 XML 结构转换至其它结构。如果您的公司使用内部的文件架构，而您的供货商使用另一种不同的架构，此时转换就十分有用了。您可以利用 XSL 转换一种架构的 XML 文件至另一种架构的 XML 文件，每一份文件中的资料仍然维持原样，并且您也可以使用指令码达成转换目的。

XSL 的演进背景

让我们来谈谈 XSL 的历史，如果您想直接进入主题，您可以跳过本节，但是在本节中您也许会发现一些关于 XSL 的有趣评价。

如同您在先前的章节中所学，1986 年国际标准组织（International Standard Organization，ISO）采纳了 Standard Generalized Markup Language（SGML）。就像 XML 一样，SGML 允许使用者描述结构－但是却不包含信息的格式。在 SGML 被采纳之后，很快的，SGML 委员会的成员又形成了一个新的委员会以建立另一种标准语言，该语言用于表达 SGML 文件的格式。该语言称为 Document Style Semantics and Specification Language（DSSSL），DSSSL 花费了 10 年的时间才完成，因为设计一种单一的语法来表达所有文件和装置的格式信息是十分困难的。

1998 年，DSSSL 委员会的某些成员成立了一个工作小组，该小组在 W3C 的赞助下建立了如何呈现 XML 文件的标准（与 DSSSL 要建立如何呈现 SGML 的标准文件有相同的目的），因为该组织致力于描述文件的呈现方式的工作已经有 10 年以上经验，这些成员很快的开始开发过程，最后开发了 XSL。

XSL 如何工作

W3C XSL 有两个主要的目的：格式化对象和转换 XML 文件。要格式化文件，XSL 会读取 XML 文件并套用一系列的转换程序以建立另一个 XML 文件，称为结果树（result tree）。结果树取决于格式化对象的名称空间，其中包含数百个描述 XML 文件呈现方式的元素和属性。举例来说，结果树指定文字对象应该为粗体或斜体、红色或橙红色，结果树中并没有包含任何特定排列版面语言的指令，指令是由 XSL 处理程序的下一个步骤才套用：格式化对象解译（formatting object interpretation）。

结果树会被读入格式化对象解译器，解译器会解译格式化对象元素和属性，并且为每一种特定目的输出其排列文档版式的程序代码，图 6-1 显示了这个过程。

在本范例中，如果一个设计师想要显示一个绿色的斜体文字，她只需要以一般的方式指定这些需求，您可以藉由格式化对象名称空间中的架构引用的字型样式和颜色属性，这些属性被设定为 **italic** 和 **green** 。这种指定输出的宣告方式优于任何特定的输出媒体，这表示设计师不需要担心如何使用各种不同的版面程序代码。

让我们使用 HTML 来显示 XSL 的工作方式，在 HTML 中，串接样式表（Cascading Style Sheet，CSS）指出字型的样式和颜色。HTML 的格式化对象解译器表现绿色斜体对象的方式为 **Style="font-sytle:italic;color:green"** ，该字符串可以由 Web 浏览器读取，最后的标签如下：

```
<P STYLE="font-style:italic;color:green;">December 3, 1997</P>
```

假设您想要以书面方式呈现一份文件（非 HTML 的文件），您可以使用了解 RTF 格式的格式化对象解译器（Microsoft 在 1980 年的中期建立了 RTF（rich text format）的语法，以 7-bit 的 ASCII 呈现丰富的格式化处理文件。因为 RTF 是简单易懂的文字，所以它可以很容易的透过电子邮件或其它简单的通讯协议传输）。本范例中的格式化对象解译器将 **font-style = "italic"** 命令转换为 **\i** ，它会将在该命令之后的文件变为斜体；将 **color="green"** 命令转换为 **\c6** ，它会指定使用 RTF 文件顶端的颜色表中第六个项目的颜色。RTF 文件的结果片段看起来就像下列：

```
\c6\i December 3, 1997\par
```

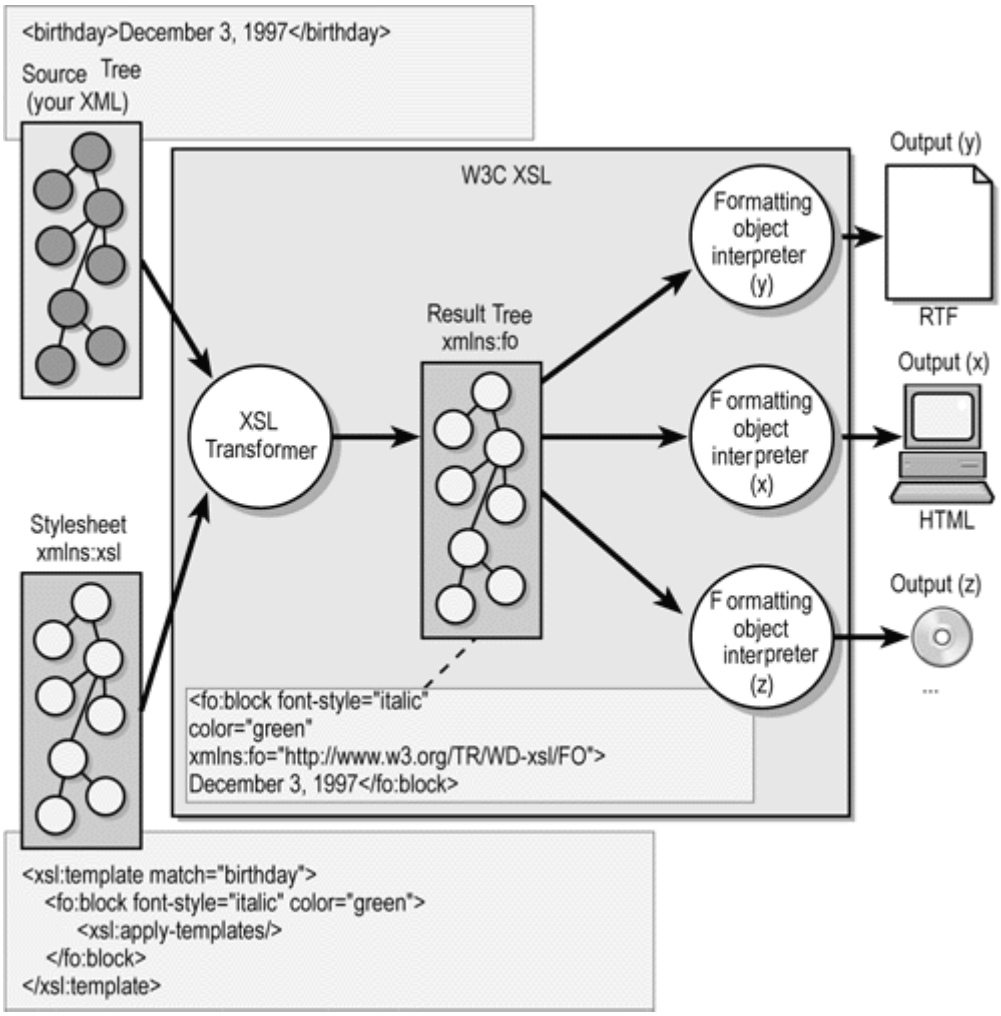


图 6-1 W3C XSL 处理过程的两个部分：首先 XML 文件被转换至结果树中，然后结果树会被格式化对象解译器解译并且为特定输出最佳化。

XSL 呈现方式的处理程序提供了一种强大的模式，因为它允许组织可以从相同的 XML 输入和样式表获得任何输出方式。当然，该模式只有在您所需要的输出方式支持格式化对象解译器才会取得其益处。

Microsoft 建置的 XSL

XSL 规格仍然在不断地开发中，格式化对象的链接库也十分复杂，并且仍有许多还未解决的问题。

1998 年，Microsoft 发觉 XSL 转换方式已经成熟，因此从 XSL 规格的草案建置部分的 XSL 转换方式。Microsoft 同样也将部分 XSL 规格引入了 MSXML 剖析器，内建于 Microsoft Internet Explorer 5。这允许开发者存取该机制－XML 对 XML 的转换。

XSLT 和 XPath 的突破

纵使有某部分人批评 Microsoft 建置的 XSL 是 W3C 规格中的未完成部分，但是仍然有许多人使用该建置标准以取得宣告式转换语言的强大功能。为了使这些规格能有更明确的分野，W3C XSL Working Group 萃取出 XSL 的转换部分（前面有提出 XSL 中有主要的两个部份）并且建立了新的 W3C 建议标准，称为 XSL Transformation（XSLT）Version 1.0。现在 XSL 以 XSLT 为它的转换引擎，而 XSL 仍然包含所有的格式化对象支持。

XSLT 需要一种对应语法来对应 XML 文件的各个部位，选择文件中的构成组件。举例来说，要以一种方式呈现章节标题，而以另一种方式呈现选单标题，两种规则都必须能够指定一条路径至特定语境（context）中的正确对象。因为 XSLT 并不是唯一需要该语法的 W3C 标准－因此 XML Path Language（XPath）从 XSLT 规格中萃取出来，W3C 在 1999 年 11 月采纳 XPath 和 XSLT 为建议标准。

XSLT 操作程序

XSLT 是一种以规则和事件驱动为原则的程序语言，它并不是一种程序语言，您可以将 XSLT 想象成一个负责填满方块的职员，其中每一个方块都负责特定的处理。图 6-2 显示了一个模板规则（template rule）－其为一独立的对象－并且等待由 **match** 属性指定的事件来驱动。一旦该规则接收到驱动事件，就可以将其内容提供给其它规则（apply-templates）使用，或者直接输出（value-of）。

当 XSLT 遇到元素时，它会基于特定基准选择一个方块来处理该元素（通常事件会发生在 XML 文件输入时），并且将事件的结果传入方块中。图 6-3 显示了这些步骤。

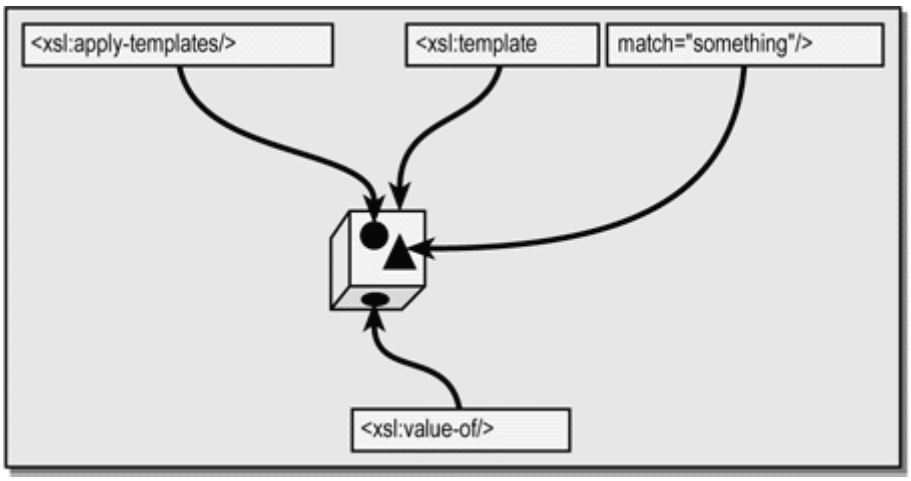


图 6-2 XML 模板规则会将处理明细压缩至方块中，允许程序设计师可以取得输入和输出。

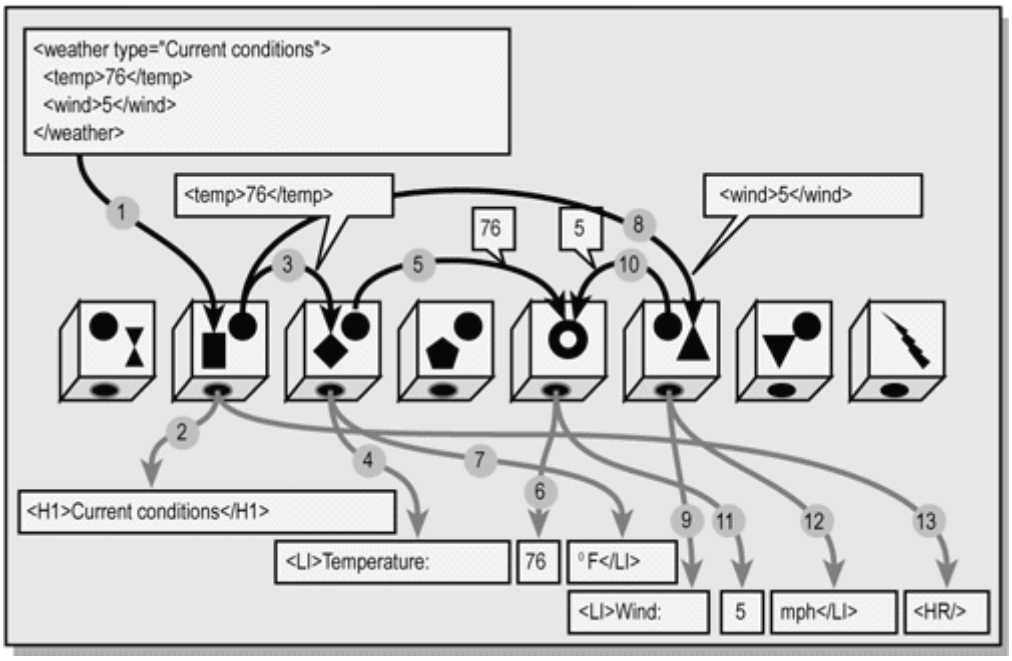


图 6-3 XSLT 是相似方块的集合，它们可以在方块中执行一些神秘的处理，某个方块可能输出某些信息供其它方块处理；而某个方块可能是直接输出，或者都不是。表 6-1 详细的描述了该过程。

处理过程发生在方块中，大部分的处理程序都会从方块中输出某些结果，完全取决于方块中执行了哪些处理，输出结果也许会是静态文字、计算结果或元素结构。而结果也有可能是直接输出或者再输入至其它方块，该方块会再进一步处理输入的结果。接下来让我们来了解这些处理程序是如何使用 XSL 转换引擎执行的。下列的范例使用范例的 XML 文件－Curcon.xml 来解释图 6-3 中的处理程序：

```
<?xml version="1.0"?>
<weather type="Current conditions">
```

```
    <temp>76</temp>
    <wing>5</wind>
</weather>
```

我们也将使用 XSL 样式表 Weathersheet.xsl，如清单 6-1 所示：

Weathersheet.xsl

```
1 <?xml version="1.0"?>
2       <xsl:stylesheet
3             version="1.0"
4             xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
5
6             <xsl:template match="weather">
7                   <H1><xsl:value-of select="@type"/></H1>
8             <xsl:apply-templates/>
9                         <HR/>
10      </xsl:template>
11
12      <xsl:template match="temp">
13            <LI>Temperature<xsl:apply-templates/>° F</LI>
14      </xsl:template>
```

```
15
16      <xsl:template match="wind">
17              <LI>Wind<xsl:apply-templates/>mph</LI>
18      </xsl:template>
19
20      <xsl:template match="text()">
21              <xsl:value-of select="."/>
22      </xsl:template>
23
24 </xsl:stylesheet>
```

清单 6-1　Weathersheet.xsl 是可以格式化上述天气数据的 XSL 样式表

表 6-1 说明了图 6-3 的处理程序，您可以在清单 6-1 中的相对应行找到表中的说明，读取这些说明以了解程序中的每一个步骤。

图中的步骤	说明
1	该方块用于处理 XML 文件中的 **weather** 元素，因为 **weather** 元素是根元素，所以它和它之下的所有元素都会被塞入方块中（请记住这个重点），这个具有方形的方块（如图 6-3）

	是由模板规则中的 6-10 行所定义。
2	首先该方块产生了一些输出，使用 **xsl:value-of/** 元素建立在 **select** 属性中的对象的文字内容。在本范例中， **type** 属性（@type）的内容会被输出到 HTML 的 **<H1>** 元素中。XSL 处理器会处理任何具有 **xsl:** 名称空间前置字的元素，任何不具有 **xsl:** 名称空间前置字的元素都会被逐字的输出。下一个命令， **<xsl:apply-templates/>** 会将 **weather** 元素的内容输出，因此其输出的内容可以供其它的方块处理。
3	包含在 **weather** 元素中的第一个项目为 **<temp>76</temp>** ，这是一个元素，并且已经有一个方块（具有钻石形的方块）等着处理该元素，该方块对应至模板规则中的 12-14 行。
4	HTML 清单项目（ **<LI>** ）元素的开始卷标和文字「Temperature」并不具备 **xsl:** 名称空间前置字，因此它们会被直接输出。
5	现在该方块注意到 **<xsl:apply-templates/>** 元素，就如同 **weather** 元素一样，此时会输出 **temp** 元素的内容「76」，该内容就可以由其它方块处理。
6	具有甜甜圈形的方块已经等很久了，该方块会处理所有其它方块输出的所有文字，可对应至模板规则中的 20-22 行，其中的 **<xsl:value-of/>** 元素将会输出文字内容至原来的元素（只能为文字元素）中直接输出。
7	在文字经过处理后，控制权会回到 **temp** 规则，然后输出清单项目（ **<LI>** ）元素的结束卷标。一旦 **temp** 元素的处理过程结束，控制权又会回到 **weather** 模板规则，并等待处理下一个子层。

8-12	**wind** 元素即为该子层，因此 **weather** 方块输出 **wind** 元素的内容，就如同 **temp** 元素一样，输出内容由其它方块处理。
13	在 **weather** 元素的所有子层都处理过后， **<xsl:apply-templates/>** 元素就完成了，规则会继续处理其后的命令。在本范例中，会输出一个 HTML 的水平线标签，然后处理过程就终止了。

表 6-1　一步步的分析图 6-3 中的 XSLT 处理程序

请注意在第九行中 HTML 水平线卷标的语法，您必须记得 XSL 样式表也要为格式正确的 XML 文件，这表示它必须遵守在 第四章 中所描述的格式正确文件，正常的标签看起来应该是 **<HR>** ，然而，水平线卷标是一个空元素，并且又包含在 XML 文件之中，因此它必须符合格式正确 — **<HR/>** 。

递归的处理

请您注意很重要的一点，就是当方块发现符合的条件时，就会将元素的内容输出，然后内容再供所有其它的模板规则使用，其中包含建立该内容的规则。以下将使用清单 6-2 中的 XSL 程序，Generic.xsl 为范例。

Generic.xsl

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet
```

```
3           version="1.0"
4           xml:space="default"
5           xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
6
7           <xsl:template match="node()">
8                   <DIV STYLE="margin-left:12pt;">
9                           [start <xsl:value-of select="name()"/>]
10                    <xsl:apply-templates select="@*"/>
11                    <xsl:apply-templates/>
12                    [end <xsl:value-of select="name()"/>]
13            </DIV>
14      </xsl:template>
15
16      <xsl:template match="@*">
17            [attr <xsl:value-of select="name()"/>:
18                  <xsl:value-of select="."/>]
19      </xsl:template>
20
21      <xsl:template match="text()">
22                  <xsl:value-of select="."/>
```

```
23      </xsl:template>
24
25 </xsl:stylesheet>
```

清单 6-2　XSL 样式表 Generic.xsl 用以处理 XML 文件以建立阶层式的结构

请注意在清单 6-2 中的样式表并没有特别指定元素或属性名称，在文件中所有位置的元素都会被处理，我们需要使用可以发现每一个节点的名称和类型的方式，因此我们必须在样式表中指定某些方式来处理文件，清单 6-3 显示了 XML 文件 80112.xml，我们将使用上述样式表处理这个 XML 文件。

80112.xml

```
<weather zipcode="80112">
    <location>Englewood, CO</location>
        <current>
            <condition name="temp">32</condition>
            <condition name="wind chill">20</condition>
            <condition name="wind">from the South at
                7 mph</condition>
            <condition name="dewpoint">16</condition>
            <condition name="relative humidity">51%</condition>
            <condition name="visibility">10 miles</condition>
```

```
            <condition name="barometer">30.01 inches</condition>
            <condition name="sunrise">7:20 am MST</condition>
            <condition name="sunset">4:51 pm MST</condition>
        </current>
    <forecast updated="2000-03-21T15:03:27-07:00">
        <day date="2000-03-22T15:03:28-07:00" high="46" low="21"
            sky="Partly Cloudy"/>
        <day date="2000-03-23T15:03:28-07:00" high="51" low="25"
            sky="Partly Cloudy"/>
        <day date="2000-03-24T15:03:28-07:00" high="51" low="19"
            sky="Partly Cloudy"/>
        <day date="2000-03-25T15:03:28-07:00" high="53" low="21"
            sky="Partly Cloudy"/>
        <day date="2000-03-26T15:03:28-07:00" high="49" low="20"
             sky="Partly Cloudy"/>
    </forecast>
</weather>
```

清单 6-3　XML 文件 80112.xml

表 6-2 描述清单 6-2 的样式表如何处理清单 6-3 的 XML 文件

行	说明
7	match="node()" 规则指定任何节点都要处理，我们最有兴趣的是处理元素节点，因此 node 是最佳的选择。
8	match 规则所遇到的第一行是一个 HTML <DIV> 元素， <DIV> 是一个区块容器标签，请注意 <DIV> 并没有 xsl: 名称空间的宣告，因此 <DIV> 元素会被直接输出，包含它的属性。 STYLE 属性是一个 CSS 样式，指定内容须靠左边。
9	字符串 start 并没有 xsl: 名称空间前置字，因此它会直接输出，然后 <xsl:value-of> 元素的 select 属性中的名称节点文字（"name()"）指定要输出元素名称，即为要求节点输出它的名称－就如同我们使用 select="." 要求其值一样，然后会输出元素名称后面的 []。 本行的输出结果会如下列形式： [start weather]
10	到现在为止，我们只使用 <xsl:apply-templates/> 元素本身，却没有使用过元素的属性 select 。默认值 select 属性将会选择所有的节点，并且将输出内容至样式表中所有的模板处理。然而，如果变更 select 属性，就可以限制处理的节点种类，在本范例中，我们只会处理属性节点，选择所有属性的对应式（pattern）为 "@*" 。
16	因为我们的第一个元素有一个属性（ zipcode ），本行中的模板规则就会撷取该属性。

17-18	**select="name()"** 节点测试传回的属性名称，该属性名称是由剖析器判定符合所传回的。在模板中，有一个属性符合条件，因此 **name** 会传回属性的名称。属性值会被处理，就像元素节点的文字值一样。 本行的输出结果会如下列形式： **[attr zipcode: 80112]**
11	当属性完成处理后，控制权回到 **apply-templates** 元素（输出属性的元素）的后面那一行（11 行），本行是未限定（unqualified）的 **apply-templates** 元素，会指定将所有的元素子层输出到所有的模板规则中处理。
7	很讶异吧！在 **<weather>** 元素之后的第一个元素是 **<location>** ，处理根元素（ **<weather>** ）的规则也会套用在 **<location>** 上。
8	就像父元素一样， **<DIV>** 会产生并且靠左缩排 12 个点。请您记得当 **<weather>** 元素被处理时，出现了靠左缩排 12 个点的 **<DIV>** ，但是该元素的 **<DIV>** 并没有结束。 **<location>** 元素的 **<DIV>** 会在 **<weather>** 元素的 **<DIV>** 之中。当然，串接样式表（Cascading Style Sheet，CSS）特性就是它们有连接性，这表示所有在 **<location>** 元素中的元素都会依次缩排 12 个点。
11	**<location>** 元素的内容会在本行被处理，单一的文字节点在本元素中，并且该文字节点会被 21 行的相符条件规则处理。
21	就像剖析器剖析任何元素节点一样，文字节点会使用本行规则处理。

22	被撷取的文字值会输出至浏览器，控制权再度传回 11 行，继续剖析过程。
12	当每一个元素的内容处理过后，我们就可以指定结束的程序（本行），本行将会在方括号中输出最近被处理的元素名称。
13	结束卷标 **</DIV>** 标记了这个 12 点缩排的结束，缩排的层级又会回到 **<location>** 元素处理之前。
7-14	这套模板规则将会因为在 XML 文件中的每一个元素被递归呼叫（不断呼叫自己），直到整份文件被处理完毕才告结束。

表 6-2　描述 Generic.xsl 如何处理 80122.xml 文件的 XSLT 处理过程

当我们在浏览器中检视 HTML 的结果输出时，我们可以看见信息是以巢状方式呈现，如图 6-4 所示。在图 6-4 中的档案是以 Microsoft XML DOM 的 **transformNode()** 方法建立，笔者稍后将带领您利用数种方式使用该方法。

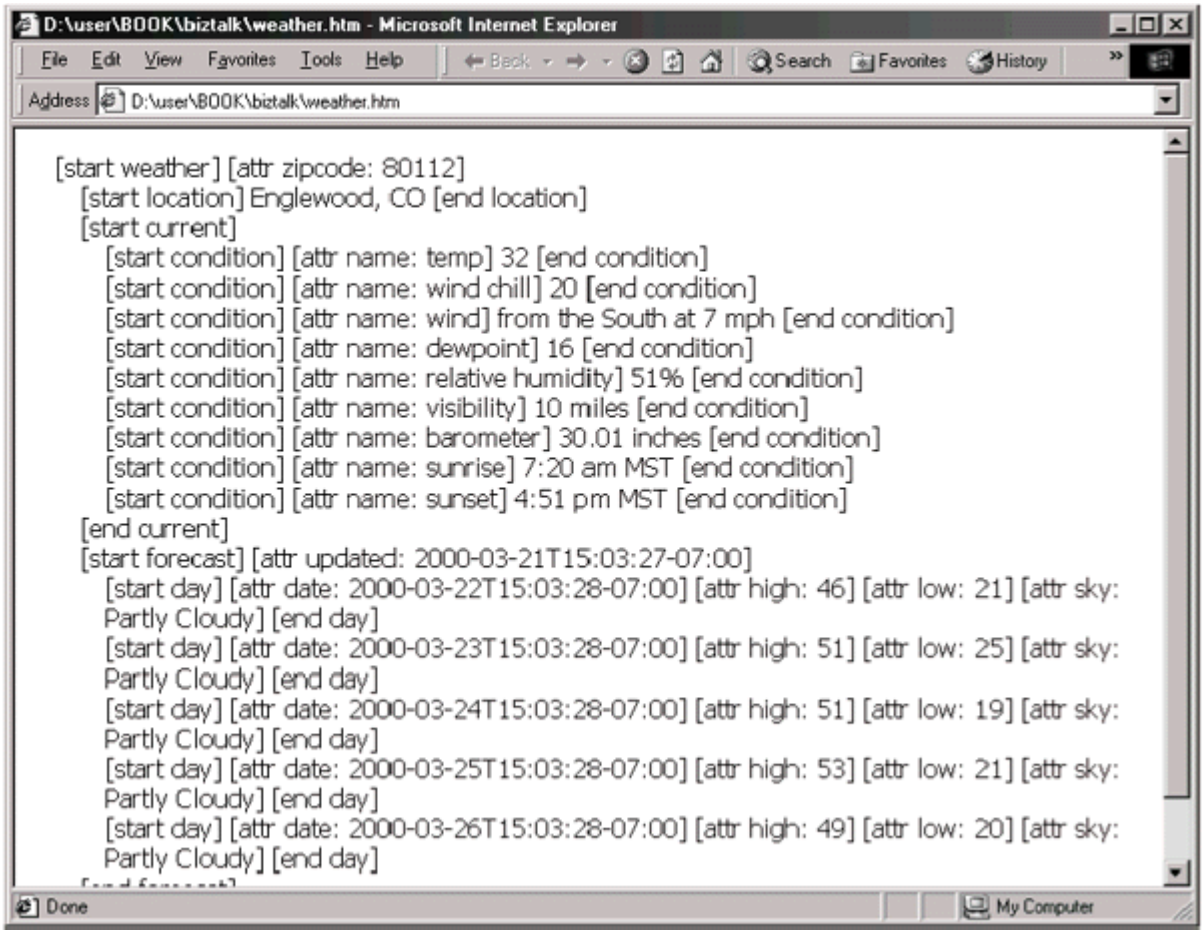


图 6-4　XML 文件经由一般样式表处理过后的巢状检视方式

XSLT 范例

让我们利用一些其它的 XSLT 元素，清单 6-4 的 XML 文件 Duckone.xml 如下：

Dockone.xml

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

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

```
<Jokes xml:space="default">
    <Joke type="one-liner" author="Brian">A duck
        walks into a bar and says,<quote>Gimme
        a beer and put it on my bill</quote>.
    </Joke>
    <Joke type="story" author="Monty">
        <Scene num="1">My dog has no nose</Scene>
        <Scene num="2">How does he smell?</Scene>
        <Scene num="3">Terrible!</Scene>
    </Joke>
    <Joke type="knock-knock" author="Miller">
        <Line>Knock-knock</Line>
        <Line>Who's there?</Line>
        <Line>Boo</Line>
        <Line>Boo who?</Line>
        <Line>It's just a joke, you don't have to cry about it.</Line>
    </Joke>
</Jokes>
```

清单 6-4　用于 XSLT 范例的 XML 文件 Duckone.xml

该文件包含三个 **Joke** 元素，其中每一个都有不同的 **type** 属性，在 XSLT 文件 Joke.xsl 中的规则将会分别处理每一个元素。清单 6-5 包含了本范例的 XSLT 样式表。

Joke.xsl

```
1   <?xml version="1.0"?>
2   <xsl:stylesheet
3      version="1.0"
4      xmlns:msxsl="urn:schemas-microsoft-com:xslt"
5      xmlns:scripts="http://architag.com/scripts"
6      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
7
8      <xsl:output method="html" version="4.0"/>
9
10     <msxsl:script implements-prefix="scripts">
11         <![CDATA[
12         function showXML(nodelist)
13            {
14            txtResult = "<SPAN "+
15                 "STYLE='background-color:lightyellow;" +
16                 "cursor:hand;font-size:8pt;' " +
```

```
17                "TITLE='"
18           replaceText = nodelist.nextNode().xml
19           replaceText = replaceText.replace(/&/g, '&amp;')
20           replaceText = replaceText.replace(/</g, '&lt;')
21           replaceText = replaceText.replace(/>/g, '&gt;')
22           replaceText = replaceText.replace(/'/g, '&rsquo;')
23           txtResult += replaceText
24           txtResult += "'>(show xml)</SPAN>"
25           return txtResult;
26           }
27        ]]>
28    </msxsl:script>
29
30    <xsl:template match="Jokes">
31        <HTML>
32            <HEAD>
33                <TITLE>XSL Demonstration</TITLE>
34                <STYLE>
35                P, UL {margin-top:2pt;margin-bottom;2pt;}
36                  H1, H2, H4 {margin-top:2pt;margin-bottom;2pt;}
```

```
37                </STYLE>
38            </HEAD>
39            <BODY STYLE="font-family:Verdana;">
40                <xsl:value-of select="scripts:showXML(.)"
41                    disable-output-escaping="yes"/>
42                <H1>World's Funniest Jokes</H1>
43                    <DIV STYLE=
44                        "border:1px solid black;padding:3pt;">
45                    <H4>Table of Contents</H4>
46                    <xsl:for-each select="Joke">
47                        <A>
48                            <xsl:attribute
49                                name="HREF">#<xsl:value-of
50                                select="generate-id()"/>
51                            </xsl:attribute>
52                            <xsl:value-of select="@type"/>
53                        </A><BR/>
54                    </xsl:for-each>
55                </DIV>
56                <xsl:apply-templates/>
```

```
57              </BODY>
58          </HTML>
59
60      </xsl:template>
61      <xsl:template name="email-author">
62          <xsl:param name="author-alias"/>
63          Author: <xsl:element name="A">
64              <xsl:attribute
65                  name="HREF">mailto: <xsl:value-of
66                   select="$author-alias"/>@jokes.architag.com
67              </xsl:attribute>
68              <xsl:value-of select="$author-alias"/>
69          </xsl:element><BR/>
70      </xsl:template>
71
72      <xsl:template name="set-name">
73          <A>
74              <xsl:attribute name="NAME">
75                  <xsl:value-of select="generate-id()"/>
76              </xsl:attribute>
```

```
77          </A>
78      </xsl:template>
79
80      <xsl:template match="Joke[@type='story']">
81          <xsl:call-template name="set-name"/>
82          <H2>Story Joke
83          <xsl:value-of select="scripts:showXML(.)"
84              disable-output-escaping="yes"/>
85          </H2>
86          <xsl:call-template name="email-author">
87              <xsl:with-param name="author-alias">
88                  <xsl:value-of select="@author"/>
89              </xsl:with-param>
90          </xsl:call-template>
91          <P>
92          <xsl:for-each select="Scene">
93              <B>Scene <xsl:value-of select="@num"/>: </B>
94              <xsl:value-of select="."/><BR/>
95          </xsl:for-each>
96          <I><xsl:value-of select="$laughtrack"/></I></P>
```

```
97        </xsl:template>
98        <xsl:template match="Joke[@type='knock-knock']">
99            <xsl:call-template name="set-name"/>
100           <H2>Knock-knock
101           <xsl:value-of select="scripts:showXML(.)"
102               disable-output-escaping="yes"/>
103           </H2>
104           <xsl:call-template name="email-author">
105               <xsl:with-param name="author-alias">
106                   <xsl:value-of select="@author"/>
107               </xsl:with-param>
108           </xsl:call-template>
109           <UL>
110               <xsl:apply-templates/>
111           </UL>
112           <I><xsl:value-of select="$laughtrack"/></I>
113       </xsl:template>
114
115       <xsl:template match="Joke[@type='one-liner']">
116           <xsl:call-template name="set-name"/>
```

```
117          <H2>One-liner
118          <xsl:value-of select="scripts:showXML(.)"
119              disable-output-escaping="yes"/>
120          </H2>
121          <xsl:call-template name="email-author">
122
123              <xsl:with-param name="author-alias">
124                  <xsl:value-of select="@author"/>
125              </xsl:with-param>
126          </xsl:call-template>
127          <P><xsl:apply-templates/><BR/>
128          <I><xsl:value-of select="$laughtrack"/></I></P>
129      </xsl:template>
130
131      <xsl:template match="Joke[@type='knock-knock']/Line">
132      <LI><xsl:apply-templates/></LI>
133      </xsl:template>
134
135      <xsl:template match="quote">
136          "<xsl:apply-templates/>"
```

```
137        </xsl:template>
138
139        <xsl:variable name="laughtrack">(laugh track)</xsl:variable>
140
141    </xsl:stylesheet>
```

清单 6-5　XSL 样式表 Joke.xsl 会转换 XML 文件至浏览器中的 HTML 输出。

表 6-3 提供清单 6-5 的 Joke.xsl 文件转换程序的说明。

行	说明
4-6	XSLT 文件需要三个名称空间，第一个名称空间包含 Microsoft XSLT 语言的延伸功能；第二个名称空间由 **script** 元素使用，稍后您将看见；XSLT 处理器需要第三个名称空间才能了解必须使用 XSLT 的哪一个版本。在本范例中，将使用 W3C 在 1999 年 11 月公布的 XSLT 建议标准。
8	**xsl:output** 元素宣告了预期的输出类型，本范例中，我们将输出指定为 HTML 4.0 版本规格。
10-28	此段程序代码为指令码（script）的范例，XSLT 规格并未包含内建的指令码接口，Microsoft 提供了数个指令码的选项，本范例中我们使用 Microsoft JScript。 我们将使用该指令码建立 HTML 的字符串，以呈现 XML 节点的结构。
30-59	当 XML 文件加载后，XML 剖析器将文件读入文件对象模式树状结构中，并且开始从根元素往下搜寻。根元素 **Jokes** 将会最先处理，然后处理剩下的。第一个符合的范本将会产生第一个事件，在本范例中，模板是根元素 **Joke** s。该规则定义了 HTML 文件的结构，包含 CSS 元素。

40	**xsl:value-of** 元素可以呼叫 XSLT 规格定义的指令码，在本范例中，在第 12 行定义的 **ShowXML** 函数会被呼叫，引起文字「（Show XML）」被插入 **<SPAN>** 中，还有目前元素的内容会被插入 **TITTLE** 属性中（「.」会被当成自变量传送至函数中）。于是建立的结果会是字符串后面还有其它字符串－意即当使用者使用鼠标指向该字符串（Show XML）时，元素的实际内容会出现在黄色的快显窗口中。
41	默认值， **value-of** 元素会将所有遇到的标记字符转成它们的实体参照。举例来说，如果遇到左角括号（<），会被实体参照 **<** 所取代。 在我们的范例中，使用 **disable-output-escaping** 属性覆写预设行为。
45-54	本段会基于文件的内容产生表格。
46	**xml:for-each** 元素会藉由 **select** 属性建立元素的集合，在本范例中，我们会建立 **Joke** 元素节点的三个成员集合。 **xsl:for-each** 元素的内容会处理这些成员，每次一个。
47-53	在本段中，HTML **A** 元素会包含每一个项目。
48	**xsl:attribute** 元素会新增元素至目前开启的元素，在本范例中，该元素为 **A** ， **HREF** 属性会被新增至 **A** 元素中，并且包含杂凑标记（#）和随机数值。
49	随机数值为集合成员的唯一识别值,Microsoft XML 剖析器会在文件中分配每一个元素一个唯一识别值。您可以使用 **generate-id** 函数存取该唯一识别值。
52	**Joke** 元素的 **type** 属性值在这里会开始处理程序，该值会被列在 **<A>** 标签中。
53	因为 XSLT 样式表也要求是一份格式正确的 XML 文件，而 HTML 的换行标签 ** ** 是一个空元素，只有 HTML 处理器会将其视为正确的，然而我们必须要建立格式正确的文件，所以必须要使用 ** ** 。

56	在 **xsl:apply-templates** 元素的 **match** 属性中的 **template** 元素，所撷取的全部节点内容会被输出到样式表中的所有规则中处理。
57-58	一旦子层都被处理后，就使用 HTML 的 **BODY** 和 **HTML** 结束标签。
61-70	请注意这个特定的模板并没有 **match** 属性，这种类型的模板称为命名模板（Named template），它将会从其它位置被呼叫。命名模板就像其它语言的子程序一样。当您有特定需要处理的部分，而且该部分的程序代码可以重复使用，您可以使用命名模板。
62	命名模板可以解译参数，因此您可以在呼叫时传递参数给命名模板。在本范例中，传递 **author-alias** 参数。
66	**author-alias** 参数会被插入至 **A** 元素的 **HREF** 属性中。
68	**author-alias** 值在本段中重复使用。
72-78	**set-name** 模板是另一个命名模板，但是这种命名模板并不需要参数。它会从其它位置被呼叫，而且所需值在模板内部计算。
80	**Joke** 元素中如果有 **type** 属性名称，并且其值为 **story** ，就会使用此模板。
81	这里会呼叫 72-78 行的命名模板，命名模板产生的任何值会在这一行被输出。
83	再次呼叫 **ShowXML** 函数，传送目前元素的内容。
86-90	使用 **xsl:call-template** 元素呼叫命名模板，该元素呼叫 **email-author** 命名模板并且传送 **author-alias** 参数，其值等于 **Joke** 元素的 **author** 属性值。
92-95	这个特定的 **Joke** 元素（type='story'）包含数个 **Scene** 元素，而 **xsl:for-each** 元素则会依序处理它

	们。
93	**Scene** 元素后面的 **num** 属性值和「Scene」这个字会以加粗方式显现。
94	**xsl:value-of** 元素会藉由使用 **select="."** 属性输出 **Scene** 元素的内容，然后再接着一个格式正确的 HTML 换行标签。
139	XSLT 允许您分配常数值给变量名称， **xsl:variable** 元素设定 **laughtrack** 变量值。
96	**xsl:value-of** 元素可以输出您预先定义变量的值，在本范例中， **laughtrack** 变量值以斜体输出于笑话之后。
99	**Joke** 元素类型为 **knock-knock** 在本行处理，跟 **story** 类型的处理相似。
116	**Joke** 元素类型为 **one-liner** 在本行处理，跟 **knock-knock** 类型的处理相似。
131-133	**knock-knock** 笑话包含数个 **Line** 元素－结构相似于 **story** 笑话。然而我们在 **story** 笑话中，使用 **xsl:for-each** 结构处理子层。但是在 **knock-knock** 笑话中，内容是由 109-111 行处理，在这个时间点，会呼叫 131 行的范本。
131	请您注意 **match** 属性会传回 **Line** 元素，但是只限于 **Line** 元素是 **Joke** 的直接子层（由 / 符号分隔），并且只限于 **Joke** 元素的 **type** 属性是 **knock-knock** 。如您所看到的，XPath 对应式可以非常的详细。
135-137	本段中包含至少一个模板以处理 **<quote>** 标记中的内容，并且将其内容放在引号（""）中。

表 6-3　XSLT Joke.xsl 转换定义的说明

使用清单 6-5 中的 XSL 转换清单 6-4 的 XML 文件，并且产生 HTML 结果，如清单 6-6 所示。

Joketrans.htm

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html;
charset=UTF-16">
<TITLE>XSL Demonstration</TITLE>
<STYLE>
     P, UL {margin-top:2pt;margin-bottom;2pt;}
     H1, H2, H4 {margin-top:2pt;margin-bottom;2pt;}
</STYLE>
</HEAD>
<BODY STYLE="font-family:Verdana;">
    <SPAN STYLE='background-color:lightyellow;cursor:hand;
               font-size:8pt;'
        TITLE='&lt;Jokes xml:space="default"&gt;
    &lt;Joke type="one-liner" author="Brian"&gt;A duck
               walks into a bar and says,&lt;quote&gt;Gimme
               a beer and put it on my bill&lt;/quote&gt;.
    &lt;/Joke&gt;
    &lt;Joke type="story" author="Monty"&gt;
```

```
            &lt;Scene num="1"&gt;My dog has no nose&lt;/Scene&gt;
            &lt;Scene num="2"&gt;How does he smell?&lt;/Scene&gt;
            &lt;Scene num="3"&gt;Terrible!&lt;/Scene&gt;
    &lt;/Joke&gt;
    &lt;Joke type="knock-knock" author="Miller"&gt;
            &lt;Line&gt;Knock-knock&lt;/Line&gt;
            &lt;Line&gt;Who&rsquo;s there?&lt;/Line&gt;
            &lt;Line&gt;Boo&lt;/Line&gt;
            &lt;Line&gt;Boo who?&lt;/Line&gt;
            &lt;Line&gt;It&rsquo;s just a joke, you don&rsquo;t
        have to cry about it.&lt;/Line&gt;
    &lt;/Joke&gt;
&lt;/Jokes&gt;'>(show xml)</SPAN>
<H1>World's Funniest Jokes</H1>
<DIV STYLE="border:1px solid black;padding:3pt;">
<H4>Table of Contents</H4>
<A HREF="#IDwuyA2">one-liner</A><BR>
<A HREF="#IDwxyA2">story</A><BR>
<A HREF="#IDwDyA2">knock-knock</A><BR>
</DIV>
```

```
<A NAME="IDwuyA2"></A>
<H2>One-liner
    <SPAN STYLE='background-color:lightyellow;cursor:
        hand;font-size:8pt;'
        TITLE='<Joke type="one-liner"
        author="Brian">A duck
    walks into a bar and says,<quote>Gimme
           a beer and put it on my bill</quote>.
        </Joke>'>(show xml)</SPAN></H2>
    Author: <A
               HREF="mailto:Brian@jokes.architag.com">Brian</A><BR>
<P>A duck
       walks into a bar and says,
       "Gimme
       a beer and put it on my bill"
   .
   <BR><I>(laugh track)</I></P>
<A NAME="IDwxyA2"></A>
<H2>Story Joke
       <SPAN STYLE='background-color:lightyellow;cursor:hand;
```

```
       font-size:8pt;'
       TITLE='<Joke type="story" author="Monty">
    <Scene num="1">My dog has no nose</Scene>
    <Scene num="2">How does he smell?</Scene>
    <Scene num="3">Terrible!</Scene>
</Joke>'>(show xml)</SPAN></H2>
    Author: <A
                HREF="mailto:Monty@jokes.architag.com">Monty</A><BR>
<P>
<B>Scene 1: </B>My dog has no nose<BR><B>Scene 2: </B>
How does he smell?<BR><B>Scene 3: </B>Terrible!<BR>
<I>(laugh track)</I></P>
<A NAME="IDwDyA2"></A>
<H2>Knock-knock
       <SPAN STYLE='background-color:lightyellow;cursor:hand;
font-size:8pt;'
TITLE='&lt;Joke type="knock-knock" author="Miller"&gt;
    &lt;Line&gt;Knock-knock&lt;/Line&gt;
    &lt;Line&gt;Who&rsquo;s there?&lt;/Line&gt;
    &lt;Line&gt;Boo&lt;/Line&gt;
```

```
    &lt;Line&gt;Boo who?&lt;/Line&gt;
    &lt;Line&gt;It&rsquo;s just a joke, you don&rsquo;t
have to cry about it.&lt;/Line&gt;
&lt;/Joke&gt;'>(show xml)</SPAN></H2>
       Author: <A


HREF="mailto:Miller@jokes.architag.com">Miller</A><BR>
<UL>
<LI>Knock-knock</LI>
<LI>Who's there?</LI>
<LI>Boo</LI>
<LI>Boo who?</LI>
<LI>It's just a joke, you don't have to
    cry about it.</LI>
</UL>
<I>(laugh track)</I>
</BODY>
</HTML>
```

清单 6-6　使用清单 6-5 中的 XSL 转换清单 6-4 的 XML 文件产生的

图 6-5 显示转换后所产生的 HTML 输出。

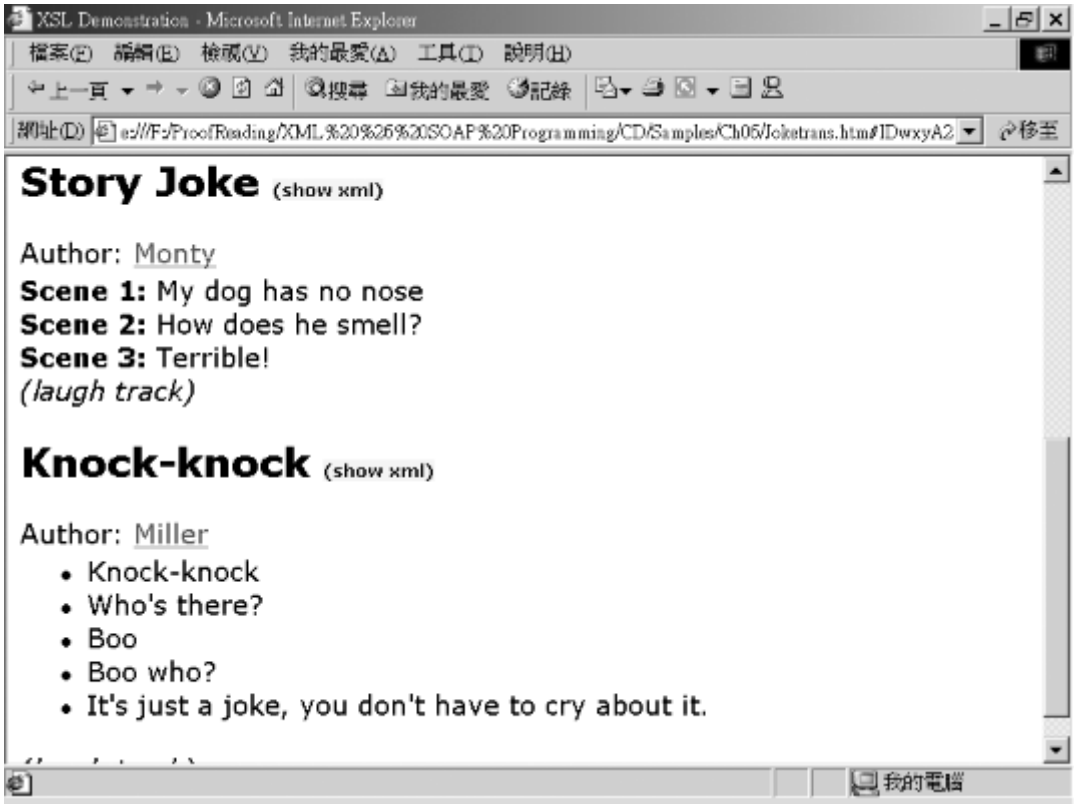


图 6-5　HTML 输出

绝佳演练：商业文件转换

在本练习中，您将建立一个 XSL 文件，将采购订单从一个架构转换成另一个架构。随书光盘包含本练习中所有的档案（\Samples\Ch06\Workshop），您将使用 Visual Basic 内建的 XSL 处理引擎执行转换。

我们在本练习中所执行的转换类型，其实在许多真实情况下都是必要的。举例来说，如果您的公司有 XML 的后端，您可以从后端存取特定架构的 XML 文件，因此公司要求您的商业合作伙伴提供给您的采购订单看起来如清单 6-7。

POReq.xml

```
<PurchaseOrder customer="H1553-7" PODate="12-Oct-1999">
    <LineItem>
        <Quantity>100</Quantity>
        <Item>SY120</Item>
    </LineItem>
    <LineItem>
        <Quantity>50</Quantity>
        <Item>SY100</Item>
    </LineItem>
    <LineItem>
        <Quantity>125</Quantity>
        <Item>AR238</Item>
    </LineItem>
    <LineItem>
        <Quantity>100</Quantity>
```

```
        <Item>JH877</Item>
    </LineItem>
</PurchaseOrder>
```

清单 6-7　采购订单 XML 文件

然而，您的最大客户之一同样也有 XML 的采购订单系统，虽然提供相同的信息，却是不同的结构。清单 6-8 显示出客户文件的范例。

POIn.xml

```
<PO date="12-Oct-1999" xml:space="preserve">
    <Customer>H1553-7</Customer>
    <Item qty="100">SY120</Item>
    <Item qty="50">SY100</Item>
    <Item qty="125">AR238</Item>
    <Item qty="100">JH877</Item>
</PO>
```

清单 6-8　来自于客户的 XML 文件

您可以撰写传统的程序，例如 C++、BASIC、Perl 或任何语言以处理文字信息，但是却有更好的方式－XSLT。清单 6-9 显示一份范例 XSL 样式表，可以将清单 6-8 的 XML 文件转换成清单 6-7 的 XML 文件。

POTrans.xsl

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet
3            version="1.0"
4            xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5               xml:space="default">
6
7               <xsl:template match="PO">
8                    <PurchaseOrder>
9                         <xsl:attribute name="customer">
10                             <xsl:value-of select="Customer"/>
11                   </xsl:attribute>
12                   <xsl:attribute name="PODate">
13                            <xsl:value-of select="@date"/>
14                   </xsl:attribute>
15                   <xsl:apply-templates/>
```

```
16              </PurchaseOrder>
17      </xsl:template>
18
19      <xsl:template match="Item">
20              <LineItem>
21                      <Quantity>
22                              <xsl:value-of select="@qty"/>
23                      </Quantity>
24                  <Item>
25                              <xsl:value-of select="text()"/>
26                  </Item>
27              </LineItem>
28      </xsl:template>
29
30      <xsl:template match="Customer"/>
31
32 </xsl:stylesheet>
```

清单 6-9　XSL 文件－将清单 6-8 的 XML 文件转换成清单 6-7 的 XML 文件

表 6-4 一步一步说明了上述程序代码。

行	说明
7	PO 元素由 match="PO" 规则撷取。
8	输入档案的 <PO> 元素被转换为 <PurchaseOrder> 元素输出。第一个 <PurchaseOrder> 开始标签输出，请您一定要记住 XSL 处理器只会处理 XSL 文件中具有 xsl: 名称空间前置字的元素，除此以外的元素会直接输出。在本范例中， <PurchaseOrder> 开始卷标并没有 xsl: 名称空间前置字，因此处理器直接将它输出。
9-11	PruchaseOrder 元素有两个属性必须被输出， xsl:attribute 元素会新增属性名称和值至目前开启的元素。在本范例中， customer 属性被新增至 Purchase 元素中，其值等于输入档案中 <Customer> 元素的内容。
12-14	另一个属性， PODate 会被新增至 <PurchaseOrder> 元素，其值等于输入档案中 <date> 属性的内容
15	藉由使用 xsl:apply-templates/ 元素，所有符合 match="PO" 规则的元素内容都会被输出，并且套用 XSL 文件中的其它模板规则，本范例中即指其输出内容还须经过 Item 模板规则处理。
16	在内容处理过后， PurchaseOrder 元素就处理完毕了。
19	match="Item" 元素在输入档案中找到 <Item> 元素，然后将其转换为 <LineItem> 元素。
20	<LineItem> 元素有两个子元素： Quantity 和 Item 。
21-23	Quantity 元素包含输入档案中的 qty 属性值。
24-26	Item 元素包含输入档案中 Item 元素的文字内容。

表 6-4　清单 6-9 中的转换过程说明

现在让我们来实作 XSL 转换的程序。

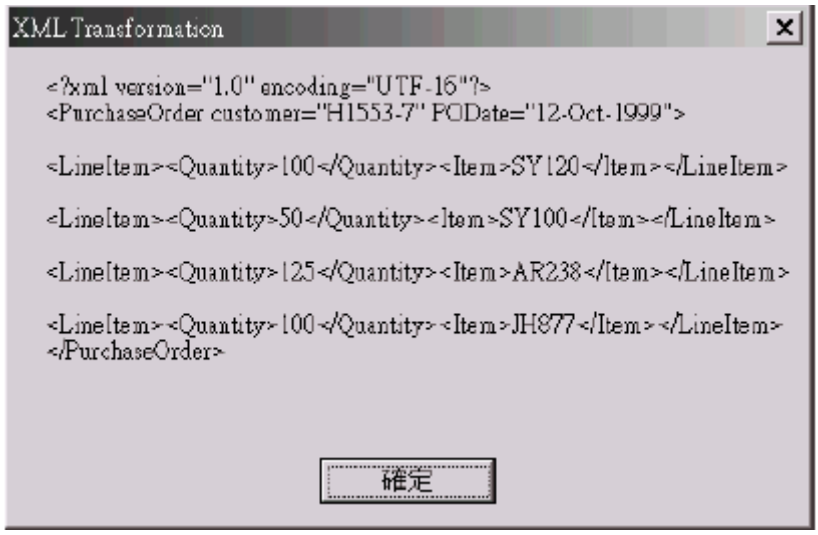
1. 加载 Visual Basic 范例项

   目 **XMLTransformation.vbp** （C:\TravisBook\Ch06\Workshop，实际位置视您的安装路径而定），您将会看见如下的程序代码：

```
2.      Dim Source As New MSXML2.DOMDocument
3.      Dim stylesheet As New MSXML2.DOMDocument
4.      Dim result As New MSXML2.DOMDocument
5.          'Load data.
6.              Source.async = False
7.              Source.Load ""
8.          'Load style sheet.
9.              stylesheet.async = False
10.             stylesheet.Load ""
11.         'Set up the resulting document.
12.             result.async = False
13.             result.validateOnParse = True
14.         'Parse results into a result DOM Document.
15.             Source.transformNodeToObject stylesheet, result
MsgBox result.xml
```

16. 在 **Source.Load** 后面的双引号中输入『C:\TravisBook\Ch06\Workshop\POIn.xml』；在 stylesheet.Load 后面的双引号中输入『C:\TravisBook\Ch06\Workshop\POTrans.xsl』。

17. 按一下 **F5** 键或工具列上的 **Start** 按钮，您将会看见屏幕中央出现名称为 **XML Transformation** 的按钮，按一下该按钮后您将会看见如下的对话框：



译者注

如果您执行时发生执行期间错误，可能是您没有新增 **MSXML** 剖析器，请从 **Project** 菜单中，选取 **Reference**，然后从 **Available References** 清单中选取 **Microsoft XML, v3.0** 以新增专案中的 **MSXML** 剖析器。

XSL 标准的转换功能是十分强大的工具，允许您可以简单的转换 XML 结构。Microsoft BizTalk Server 有一种工具－Microsoft BizTalk Mapper，当您从许多不同的商业合作伙伴接收文件，或者当您从公司系统传送文件给商业合作伙伴时，可以简单的建立 XSLT 样式表以自动执行转换工作。我们将会在 第 12 章 使用 BizTalk Mapper 以执行这些转换工作。

第 2 篇 XML 讯息

7. Web 服务

. 信息汇总的重要性

. 开放式的多个服务

8. 使用 SOAP 建立 Web 服务

. HTTP

. 通讯端口

. 防火墙

. SOAP 讯息结构

. XML Newsfeed 范例

9. BizTalk 基础结构

. BizTalk 文件和讯息规格

. 在 B2B 交易中的 BizTalk Framework

7. Web 服务

过去十年间，Web 已经经历了一系列重大的科技标准改革，如同图 7-1 所示，每一次的革命都是奠基于之前的基础而顺利推进，直到现在，我们已经有能力可以撰写取得数据的网络程序，并且透过网络进行处理程序。

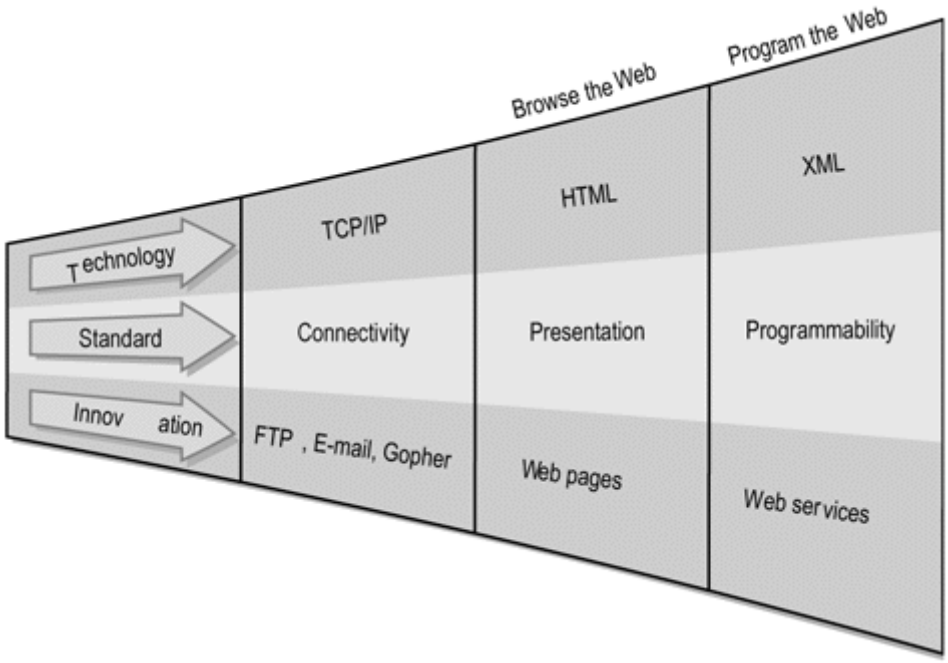


图 7-1 经历一系列技术革新的 Web

1980 年代，美国的 Department of Defense（DoD）开发了 TCP/IP（Transmission Control Protocol/Internet Protocol），TCP/IP 是一个简单的通讯协议，让我们可以在专有的私人网络之间实体上相互连接，成为公用网络并且传送封包。一旦网络上的计算机可以彼此互相连接，应用

程序便可以取得这个能力的优点，以第一个出现的技术来说－File Transfer Protocol（FTP）和电子邮件（E-Mail），就是架构于 TCP/IP 之上允许人们相互通讯，这样一来，同时也彻底的改革了我们的商业行为。

虽然 FTP 和电子邮件有许多优点，但是它们都没有提供一致的方式和接口来观察信息。于是 HTML 这个简单的卷标结构语言出现了，它被设计用来建立信息的检视方式，人们可以透过 Web 浏览器愉快的检视信息。HTML 卷标的其中一个「Link」标签，又为人们建立了新的网络利用方式－可以立即跳跃至网页中的某一点或者跳至其它的网页。

在公共网络上共享超级链接的文件又一次彻底的改革了我们检视信息的方式，因为使用者可以使用标准的方式共享信息，无论他们使用的是何种作业平台或软件版本。现在，任何人都可以使用一种应用程序浏览整个网络。

HTML 是一项整合性的科技，可以提供人们与计算机信息的相互通讯，但是当计算机之间的数据要互相沟通时，它的功能就显得非常短缺了。首先，HTML 是设计用来建立二维文件的呈现方式，适合于人类的眼球检视。然而计算机没有眼球，所以 HTML 对计算机来说并不怎么适合，HTML 强迫我们的信息平贴于屏幕上，同时也强迫我们将注意力集中于了解模拟两可的语意。

试着想象一下有一个旅游经纪人，她的工作是了解她的客户想做什么以及他们将参加哪种旅行套装行程。如果客户请她找一个定位在「便宜、温暖和适合家人同游」的假期，她就有足够的知识开始搜寻相关信息，因为她知道「温暖」代表的意义，这是一个非常标准的术语，纵使她可能会清楚的问客户：「需要像 Death Valley 一样干燥的温暖或是像 Orlando 一样湿热的温暖？」客

户也有可能会清楚的表示「便宜」的意义（好确定她至少会找一个有室内水管的地方！）；她也可能问客户「家人」包含谁－例如十几岁的青少年或幼童，然后她开始进行工作。

首先，她会开始搜寻当客户假期开始时世界上比较温暖的地方，然后她调查航空运费、旅馆、汽车租金，以及其它可能会影响价格的相关因素，她也许会使用旅馆提供的使用手册和假期包办的单位，或者搜寻相关她所需信息的 Web 网站以供参考。

如图 7-2 所示，旅游经纪人变成信息的汇总中心，联合符合客户需求的所有服务，她是信息的中心。

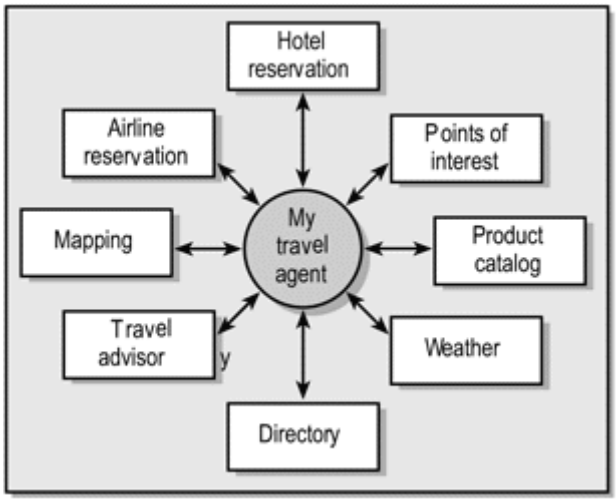


图 7-2 旅游经纪人是汇总中心，仰赖许多不同的信息来源以建立客户的旅游信息中心

在搜寻数种信息来源之后，旅游经纪人开始作决定，举例来说，在 Grand Gayman Island 的温度是 85 度，但是旅馆较贵并且从 Denver 出发的飞机班次有限。St. Thomas 只有 82 度，但是

便宜且容易到达，然而 St. Thomas 没有什么可供小孩游乐的措施。Bahamas 只稍微贵了一点点又不十分温暖，但是有许多提供给小孩的活动。换句话说，客户必须牺牲某些需求才能符合，最后，她带来了数个计划并且一一分析优缺点给客户听。

旅游经纪人是人类，她一次只能做一些事情，况且她还有别的客户，不可能将时间都花在一个案子上。同时也因为上列的限制，她只能建议少数的替代方案，可想而知一定有温暖、小型和适合家人的旅游景点，而且可能会削价以吸引更多的游客，如果旅游经纪人没有取得这些信息，这个客户也许就丧失了这些机会。

如果计算机可以辅助旅游经纪人，让她考量更多的代替方案以增广她的办事效能，这样不是很好吗？很棒的想法，但是没有用，问题就出现可用的数据格式。

如果旅游经纪人从 Web 上取得信息，信息很有可能是由 HTML 传递，并且藉由她的眼球从 Web 浏览器中检视。许多她所检视的 HTML 网页都是由网站的中间层应用程序在进行中临时建立的，当她造访一个旅馆网站并且寻找是否有房间时，旅馆的 Web 应用程序查询包含房间信息的数据库，然后为她建立一个网页。

如果我们建立一套计算机程序，该程序用和旅游经纪人相同的方式检视信息（试图以人类方式），则该计算机的问题就大啦！它会将 HTML 网页视为一维的标记字符串，而不是传达特定信息的二维文字或图形。举例来说，当您看见文字「76oF」，您会确认是华氏温度 76 度，而且许多人也都会认为那是一个愉快、温暖的温度。但是计算机会看见四个 ASCII 字符，也无法像人们这样解读。

回到旅馆网页的范例，我们都知道有价值的数据就在那里，伺服端的应用程序会「只傻傻的下载（dumb down）」数据，因此人类也得以解读。如果我们绕过将信息平贴于计算机上供眼球读取的步骤，并且直接使计算机与计算机对谈呢？

如果我们以某种方式指示计算机程序去旅馆的网站，该网站的计算机直接代替人类解译信息，然后传回我们的程序，程序将更易于处理：

```
<avg-temperature deg="f">76</avg-temperature>
```

现存的 Web，所有大大小小公司的数百万台计算机都她们的信息放在网络上供现存的客户或潜在客户存取，并且有大量的信息就在伺服端指令码（server-side script）的后面，等着 Web 浏览器存取。

如果您的公司有许多有价值的信息，难道您不会想将这些信息重新包装以不同的方式销售并获得更多的利益？如果您的公司需要信息，可以从某处依照您的需求购得信息，难道不会比您自己建立这些信息更方便或更便宜？

试着将 Web 服务想象成应用程序，它使您可以在 Web 上开发程序使用数据，就像您现在开发使用内部数据的程序一样。这个开发 Web 程序的能力将是未来几年我们如何使用 Internet 的重点。

信息汇总的重要性

信息汇总已经存在数个世纪了，旅游经纪人就是一个范例，但仍然有许多其它的范例。一个独立的保险经纪人会从许多不同的保险公司取得信息以便发现对她的客户来说最佳的原则。同样的，贷款经纪人也为了房贷做相似的事情，同理可证，如果任何人的工作描述中包含「经纪人」，她就是一个信息汇总中心。

像范例中的旅游经纪人一样，这些信息的汇总者一次可能只能考量少数的代替方案，简单的说就是因为人类收集信息较慢，Web 和 Web 服务的来临将会带来汇总信息的革新。

革新汇总方式的一个例子就是 反向拍卖 （reverse auction），在这个模式下，买家可以订立一个指标指定她要购买一特定的项目，而可以提供该项目的卖家互相出价，尽可能达到最低的价格。例如，如果您要买十磅的钉子，只要有人类介入，反向拍卖的价格可能会远超过您想节省的价值。

假使我想使用计算机自动化的过程，拍卖价格会低的惊人，那我们可以推想，如果我每买十磅的钉子就省下数分钱，那么所有省下的钱可能会十分惊人。所以我们要如何让拍卖价格压得如此低？简单的说，只是以智能型的方式利用开放的标准罢了。

开放式的多个服务

计算机的内部通讯是使用接口的集合，这些接口通常由操作系统平台或程序设计师撰写的语言定义。在现在的对象导向平台，您可以撰写可供其它程序重新使用的程序。Microsoft COM 就是一个范例，COM 组件通常储存在动态链接库（dynamic-link library，DLL）档案中，应用程序设计师可以使用这些组件建立 建构区块 （building blocks）。试想您在 Microsoft Windows 要打印

Word 文件时所看见的对话框，撰写 Word 处理器的程序设计师可能不会撰写对话框中所有的程序，而对话框的程序是由 COM 组件建立的，组件由 Windows 操作系统提供并且由 Word 处理器呼叫。事实上，这也是 Windows 操作系统的对话框看起来都一样的原因。

对象是程序代码的一部份，并执行特定的工作。对象是独立的，因此许多应用程序都可以共享它们。这个程序的独立性可以减少开发的耗费，因为程序设计师不用花时间做一些零散的工作。

对象通常会有属性和方法，属性是您可以查询值和存入值的地方。举例来说，您也许会想设定字型选择对话框的属性以让对话框开启时只显示 monospace 字型。

方法是当需要时对象就会执行的函数，方法也许是 **getStockPrice** 或 **multiplyTwoNumbers** 。有了方法，对象可以执行许多不同的动作以产生结果。举例来说，要取得股票价格，对象可能就要连接至包含目前股票信息的数据库，对象处理完成并建立结果的过程并非我们关心的目的，我们只需要结果。我们只需要记住很重要的一点，这个处理过程是可以跨平台的。

有时候方法会需要参数，您不会想要一些杂七杂八的股票价格，您可以指定您感兴趣公司的股票信息。在这种状况中，您也许会将股票名称当成参数传送给函数。同样的，方法也允许您传送日期或时间以取得历史数据。

在 UNIX 的平台，对象的重新使用是以共享对象的形式，就像 COM 一样。Java 平台有 Enterprise Java Bean（EJB），Java 程序都呼叫 EJB 以执行特定服务；Wondows 程序都呼叫 COM 组件以执行特定工作，但是因为这两种程序使用不同的接口语言，因此它们要互相呼叫十分困难。

依赖专有对象才能运作的系统虽然有完整的通讯定义，但是它们的通讯接口却十分没有弹性，如果在计算机和服务对象之间的通讯其中任何一部分中断了，或是呼叫方式不是十分正确，就会发生「非预期的错误」。

上段所述系统的另一个问题是跨计算机通讯，Windows 程序为了特定的服务可以呼叫 COM 对象（COM 对象要位于本机计算机上）－这表示执行程序的计算机呼叫该对象。举例来说，Web 服务器上也许会有一个服务（该服务以 COM 对象的方式存在），该服务连接至本机数据库以取得某公司股票的历史信息，只要对象和呼叫程序位于同一台计算机上，工作就可以正常执行。然而，如果取得股票信息的计算机是属于数台 Web 服务器负载平衡丛集之中的一台，则呼叫程序就可能不会在同一台计算机上。为了应付这个问题，Microsoft 开发了 Distributed Component Object Model（DCOM），利用 DCOM，使用者就可以透过网络存取服务所在的计算机。当然，其它的平台也有类似的分布式模式，然而我们虽然达成了跨计算机通讯的目的，却只限于相同的平台，在跨平台之间取得服务仍然有困难。举例来说，就像 Windows 程序无法呼叫执行于 Linux 之上的 EJB 一样。全世界的公司都有其它公司可以轻易存取的资料，然而，如果某家公司必须直接呼叫产生数据的对象，那么该公司就会遇到跨平台的兼容性问题。

了解上面的叙述之后，您应该知道从 Microsoft Visual Basic 无法直接呼叫 EJB，而 Java 程序也无法直接呼叫 COM 对象的方法。要使这些对象模式之间的通讯更有效率，我们需要利用某些方式以将建立数据的对象以及该对象需要特定平台的要求性分开（也就是说要将对象和平台分别管理）。解决方案就是 Simple Object Access Protocol（SOAP），SOAP 解决了多种平台在存取资料上不兼容的问题。

SOAP 是一种语法，它允许您可以建立从远程呼叫对象方法的应用程序，SOAP 解决了两个不同的系统必须执行相同平台或使用相同语言的问题，透过专有的二进制通讯协议代替呼叫方法，SOAP 使用开放式的标准语法以执行方法的呼叫，该语法就是 XML！

在应用程序之间所有的信息和接收的对象都是以 XML 结构将数据加上描述卷标传输。于是资料串流（stream）就是简单易懂的文字对象，可以经由 HTTP 传输，并透过防火墙，图 7-3 显示了该过程。

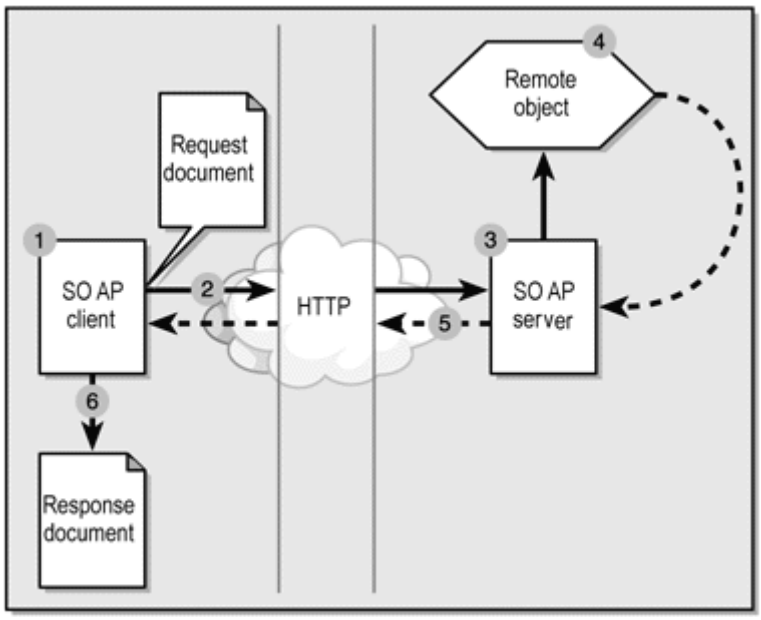


图 7-3 SOAP 提供了一种在 HTTP 联机呼叫对象的方式，本图中的说明在表 7-1 中

就如同您在图 7-3 中所视，SOAP 客户端以 HTTP 传送一要求文件至另一个站台的 SOAP 伺服程序，这个正在聆听（Listening，等着接收要求）的服务器会拦截讯息、解译要求，然后呼叫在

它的管辖范围的对象方法。该对象藉由 SOAP 伺服程序响应某些有用的信息至正在等待的客户端应用程序。表 7-1 说明了图 7-3 中的步骤。

图中的步骤	说明
1	一个程序扮演 SOAP 客户端，建立一份 XML 文件，该文件包含要在外部系统远程呼叫某方法所需的信息，不要将 SOAP 客户端视为传统的客户端。举例来说，SOAP 客户端可能是您的 Web 服务器或某些伺服端的应用程序，但是它也有可能为桌上型计算机客户端的某一部份（例如您的计算机中某个建立 XML 文件的应用程序）。简单的说，SOAP 客户端就是对 SOAP 伺服程序提出请求的客户端。 SOAP 客户端将 XML 文件包装在 SOAP 信封（SOAP envelope）内，并且转为 HTTP **POST** 请求。
2	藉由标准的 HTTP 联机传送该封装。
3	聆听的应用程序－定义是 SOAP 伺服程序－接收该讯息，该应用程序通常是 Web 服务器，负责聆听 SOAP 请求。SOAP 伺服程序会剖析 SOAP 封装并且呼叫适当的对象、传送包含在 SOAP 文件中的必要参数。

4	远程对象会执行请求中的功能函数，并且传回信息给 SOAP 伺服程序，SOAP 伺服程序再将响应放到 SOAP 信封中。
5	响应对象被传送回发出请求的计算机，SOAP 文件再次藉由标准的 HTTP 联机传送。
6	SOAP 客户端等待响应对象，当 SOAP 客户端接收该对象时，它会将 SOAP 信封解开，并且传送该响应的文件给需要的适当应用程序。

表 7-1　图 7-3 中的步骤说明

因为 SOAP 文件是透过 HTTP（通讯端口 80）传送，该文件几乎可以穿透任何防火墙－因为大部分的防火墙都没有关闭使用该通讯端口的通讯，因此在不同的平台之间取得信息不需要对防火墙的基础结构做任何变更。

注意很重要的一点，就是被呼叫的对象不需要任何修改。SOAP 伺服程序的工作是负责解译来自 HTTP 联机的 SOAP XML 文件，并且将其转成对象了解的语言－意指 SOAP 伺服程序扮演解译器，同时解读 SOAP 语言和被呼叫对象的语言，然后在两者之间互相沟通。因此，对象可以是任何在平台上以任何语言撰写，其中所有的通讯都是使用以 XML 语法撰写的文件。

8. 使用 SOAP 建立 Web 服务

Simple Object Access Protocol（SOAP）需要两种开放式的标准：HTTP 和 XML。SOAP XML 是直觉式、并只包含少数元素和属性的语法，SOAP 定义了您可以放数据的信封（envelope），而 **SOAP:** 名称空间前置字则定义了 SOAP 元素。

在本章中，我们将建立两个部份的范例－首先我们使用 **JScript** 和 **XMLHTTP** 对象以建立 SOAP 客户端，笔者将以 **XMLHTTP** 开始让您实际了解在 SOAP 客户端中包含什么， **XMLHTTP** 对象在您的应用程序之间提供一个接口，该接口是一个包含 JScript 的 Web 网页。我们将使用该网页在服务器上张贴 XML 文件，使用 HTTP 最大的好处就是张贴的数据穿过防火墙并不需要任何的修改。这表示您可以藉由使用 XML 从您的商业合作伙伴取得信息，就如同使用传统的 HTML 形式一样简单。

第二个范例，我们将使用非 Microsoft 产品－OmniMark 建立 SOAP 伺服程序和一个 Web 服务，OmniMark 可以执行于 Microsoft Windows、UNIX 和 Linux 平台。本章包含这个产品是为了让您了解，透过多种平台/语言也可以存取 Web 服务。

HTTP

要了解使用 HTTP 的 Web 服务，您需要了解 HTTP 是如何运作的，HTTP 是一个简单的通讯协议，并以简单的文字为基础进行传输，因此它无论透过何种传输媒体都可以运作的很好。HTTP

通讯由一系列的讯息在客户端和服务器之间传送，大致可以分成请求讯息和响应讯息。HTTP 标准定义了数个不同的请求方法，最常用的是 **GET** 和 **POST** 。

GET 方法

当使用者在 Web 浏览器的地址列输入 URL 时，同时也执行了 **GET** 请求，举例来说，如果使用者请求 http://www.myduckjokes.com/latest.htm 档案，Web 浏览器就会建立一个 HTTP 封包－包含请求的标头（header）－传送至由 URL 指定的服务器 IP 地址。标头看起来如下列所示：

```
GET /latest.htm HTTP/1.1
Host: www.myduckjokes.com
Content-Type: text/html
{blank line}
```

HTTP 封包由下列部分组成：

- 初始化行
- 标头行
- 空白行（例如，一个换行字符）

- 选择性的讯息区块（例如档案、查询数据或查询输出）

初始化行指定封包的目的，在本范例中，封包为 **GET** 请求指示服务器存取 **latest.htm** 档案，HTTP 版本是 1.1。

标头中的下一行指定请求的参数，这些参数必须成对出现－意指必须以参数名称开始该行，后面接着一个冒号，再来是一个空白，然后接着该参数的值，并以换行字符结束。

HTTP 请求封包可包含一打甚至上百个参数，因此浏览器可以传送浏览器类型、操作系统类型和版本，以及 cookie 信息给服务器。可想而知，如果您的 HTTP 封包越大，传送请求的时间就越长。标头以两个换行字符结束，所有在换行字符后的文字视为 HTTP 封包的数据主体（body）。

封包由 HTPP 服务器（也称之为 Web 服务器）接收，它会试图解析文件名称（/latest.htm）。而 HTTP 服务器应该会找到该网页、执行任何必要的处理，并且传回 HTML 的结果（HTTP 响应封包），清单 8-1 显示包含 HTTP 标头的响应封包，在标头的两个换行字符后面的信息即为数据区块（payload），它通常是 HTML 文件。

HTTP Response

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 283
```

```
<HTML>
   <HEAD>
                  <TITLE>Latest Duck-Bar Jokes</TITLE>
   </HEAD>
   <BODY>
                  <H1>Latest Duck-Bar Jokes</H1>
                  <P>A duck walks into a bar. The bartender says,
                  "We don't serve ducks here." The duck says,
                  "That's OK, I don't really like duck, anyway. How
         about a beer?"</P>
   <P></P>
   </BODY>
</HTML>
```

清单 8-1　HTTP 响应封包

请注意响应封包中标头的第一行有相关于程序本身的信息，然后接着一系列成对的值以辅助 Web 浏览器了解封包。

第一行包含状态信息，状态信息有很多种，最普遍的就是 200（表示封包已经顺利传递），其它还有一些状态信息，例如 404（表示某些资源并没有找到）、500（服务器产生错误），状态信息是供计算机读取的；而讯息信息是供人类读取的，状态信息分配了某些号码范围：

- 1xx 负责指出某些信息的讯息
- 2xx 负责指出某些程序顺利完成
- 3xx 重新导向客户端至其它的 URL
- 4xx 在客户端指出错误
- 5xx 在伺服端指出错误

在 1xx 层级状态的所有响应必须包含 **Date**： 的标头，在 HTTP 中的所有时间值使用格林威治时间（Greenwich Mean Time）。

POST 方法

当使用者在 Web 网页上按下 **Submit** 按钮时，信息可能会被张贴在 Web 服务器的某处，HTML 中有某些元素可以撷取使用者的信息，并且传送至 Web 服务器。在下列范例中，我们将使用 **Input** 元素从使用者取得信息，并且使用 [POST] 方法传送数据至服务器。清单 8-2 中的 1040.htm 档案显示一份典型的网页，这个网页会从使用者处取得信息，您可以在 **\Samples\Ch08** 资料夹中找到该档案。

1040.htm

```
<HTML>
        <HEAD>
                <TITLE>1040 Super E-Z Form</TITLE>
        </HEAD>
        <BODY STYLE="font-family: Verdana;">
                <H1>1040 Super E-Z Form</H1>
                <FORM METHOD="POST" ACTION="/cgi-bin/processForm.pl">
                        <TABLE>
                                <TR>
                                        <TD>Enter your social security
number:</TD>
                                        <TD><INPUT TYPE="TEXT"
NAME="socSecNum"></TD>
```

```
                                </TR>
                                <TR>
                                        <TD>How much did you make last
year?</TD>
                                        <TD><INPUT TYPE="TEXT"
NAME="income"></TD>
                                </TR>
                                <TR>
                                        <TD>Press "Calculate" for
verdict</TD>
                                        <TD><INPUT TYPE="SUBMIT"
VALUE="Calculate">
                                </TR>
                        </TABLE>
                </FORM>
        </BODY>
</HTML>
```

清单 8-2　包含 **FORM** 元素的 HTML 文件

请注意 FORM 元素，该元素是用于联合使用者接口的元素，例如 INPUT 方块、 TEXTAREA 字段和 SUBMIT 按钮。在本范例中， FORM 有两个属性－ METHOD 和 ACTION ， METHOD 属性指出我们将要使用 HTTP POST 方法以传送窗体中的数据至 Web 服务器； ACTION 属性包含指令码的路径名称，该指令码将会在 Web 服务器上处理这个窗体中的数据。文件名称中的 .pl 扩展名指出我们将会执行在 /cgi-bin 目录中的 Perl 指令码。

在 Web 浏览器中加载清单 8-2 中的程序代码如图 8-1 所示，使用者可以在字段中填入值。

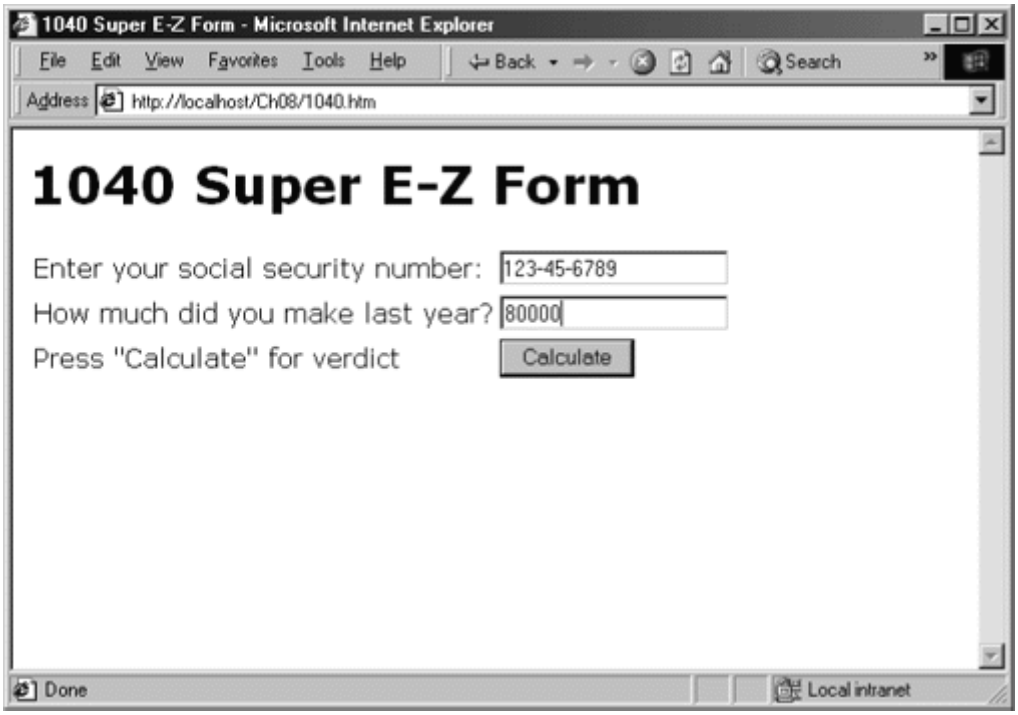


图 8-1 在 Web 浏览器中的 HTML 文件

当使用者按一下 Calculate 按钮，Web 浏览器会建立如清单 8-3 的 HTTP 封包。

```
HTTP Information

POST /cgi-bin/processForm.pl HTTP/1.1

Host: www.irs.gov

Content-Type: application/x-www-form-urlencoded

Content-Length: 35



SocSecNum=123-45-6789&income=8000
```

清单 8-3　内含 HTML 窗体信息的 HTTP 封包

Content-Type 标头指示服务器该封包是加密形式的对象，服务器要在将该封包传送至应用程序时必须先译码。封包的数据主体包含每一个输入字段的名称和值， **&** 符号将两对名称和值分开。

当服务器取得封包，它会剖析数据主体的值并且传送这些信息至指令码。然后指令码会处理这些信息并且透过 HTTP 服务器传回结果。

通讯端口

经由 IP 通讯的服务器会聆听各种 IP 通讯端口（Port）以决定要执行什么动作，默认值 FTP 服务器会聆听 21 通讯端口的请求、HTTP 服务器聆听 80 通讯端口的请求－这就表示 Web 服务器也会聆听这个通讯端口，等待请求的到来。虽然您可以变更这个通讯端口，但是那并不是一个好主意，因为应用程序通常会使用 80 通讯端口来传送 HTTP **GET** 和 **POST** 的请求。

另一种 HTTP 的形式，称为 HTTPS，它会在 Web 浏览器客户端和 Web 服务器之间提供安全性的加密。预设情况下 HTTPS 兼容的 Web 服务器会聆听 443 通讯端口。当使用者请求安全性联机时，具有安全性加密能力的 Web 服务器将会使用加密的程序代码和 Web 浏览器通讯，当每一次客户端和服务器交换数据时，传送端加密数据，接收端解密数据，它们都是使用加密金钥交换数据。

因为 HTTPS 比 HTTP 更为复杂，本书不会花太多时间解释，您只需要知道客户端和服务器可以使用某种方式交涉加密的封包，并且加密解密的程序对使用者来说并不会有任何影响。

其它的服务使用其它的预设通讯端口，举例来说，Telnet 使用 23 通讯端口，网络管理员可以使用防火墙或 Proxy 服务器限制特定信息类型的存取（原理即为将特定的通讯端口关闭，封锁封包的目标）。

防火墙

笔者花了一些时间解释 HTTP 的基础架构，因为 XML 可以透过和 HTTP 相同的通讯端口传输，现在大部分的公司都允许他们的员工存取 Web 网页，同时大部分的公共 Web 服务也会聆听 80 通讯端口，所以大部分的防火墙允许数据可以自由的通过这个通讯端口，而防火墙软件通常也不会检查通过该通讯端口的数据类型。使用 80 通讯端口可以在公司之间取得信息，也不会造成网络管理员的复杂管理问题。

SOAP 讯息结构

SOAP 文件是一种电子信封，您可以将您的数据区块（payload）放入该信封中，数据区块由 描述呼叫方法的标签 和 完成该方法所需的数据 组成。图 8-2 显示 SOAP 封装。

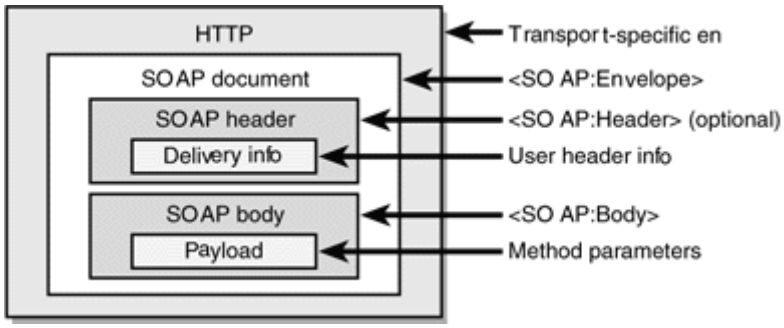


图 8-2　SOAP 文件是一个信封，其中包含传送至 SOAP 伺服程序的数据区块

SOAP 信封文件有两个子元素： SOAP:Header 和 SOAP:Body 。 SOAP:Header 元素包含相关交易的信息，该信息是使用者定义，其内容取决于您想达成的目的，举例来说， SOAP:Header 元素指定 Transaction 元素的值为 5，接收的应用程序可以使用该值来执行某些有用的事。请注意 Transaction 元素位于称为 trans 的使用者定义名称空间之中。

```
<SOAP:Header>
    <trans:Transaction
        xmlns:trans=http://schemas.architag.com/transaction.xsd
```

```
        SOAP:mustUnderstand="1">5</trans:Transaction>
</SOAP:Header >
```

SOAP 元素可以有 **SOAP:mustUnderstan** d 通用属性，该属性指定对于接收者来说，标头项目是强制性或选择性，这个属性值可以为 **1** 或 **0** ，它是设计用于使规格可以随时间而发展。举例来说，您可以传送一个新规格的 SOAP 文件至旧规格撰写的 SOAP 伺服程序；**SOAP:Header** 元素是选择性的。

说明

SOAP 可以于 HTTP 之上运作的很好，但是 SOAP 的规格指出它也可以使用其它的传输通讯协议。

SOAP:Body 元素是信封的第二个子元素，对于 SOAP 请求来说，数据主体中包含呼叫方法的定义卷标，这些卷标包含方法完成工作所需的信息；对于 SOAP 响应文件来说，**SOAP:Body** 元素包含被建立的数据（已经描述方法完成后会发生什么事，有一个选择性的 **SOAP:Fault** 元素可以指出 SOAP 伺服程序处理请求是否发生问题）。

SOAP 也提供丰富的描述语言，可以指定加密样式、数组结构和数据类型。所有的信息都在 SOAP 规格中。

XML Newsfeed 范例

在 XML Newsfeed 范例中，我们将会处理一个维护数据库的站台，该数据库中包含 XML 建置者感兴趣的文章。该站台位于 http://architag.com/xmlnews.html ，按一下 URL 会呼叫 Microsoft Active Server Pages（ASP），这个 ASP 网页会连接至新闻数据库，并且建立 HTML 输出传送给使用者。

假设我们想要存取的信息位于 newsfeed 数据库，但只需要某种数据，而不需要整个 HTML 网页。换句话说，我们只需要未经修饰的新闻数据，并且要将它格式化成符合我们所需的形式－使用 SOAP 可以达成这个目的。

SOAP 客户端应用程序

首先我们将会建立一个请求 Web 服务的客户端，我们将会使用 Jscript 建立 HTML 网页的客户端，图 8-3 显示我们范例中的客户端，会在表 8-1 中说明；图 8-4 显示出在浏览器中的客户端，清单 8-4 中的 HTML 程序代码显示我们将使用的 HTML 文件（getXMLNews.htm）的资料主体。您可以在随书光盘的 **\Samples\Ch08** 找到范例。

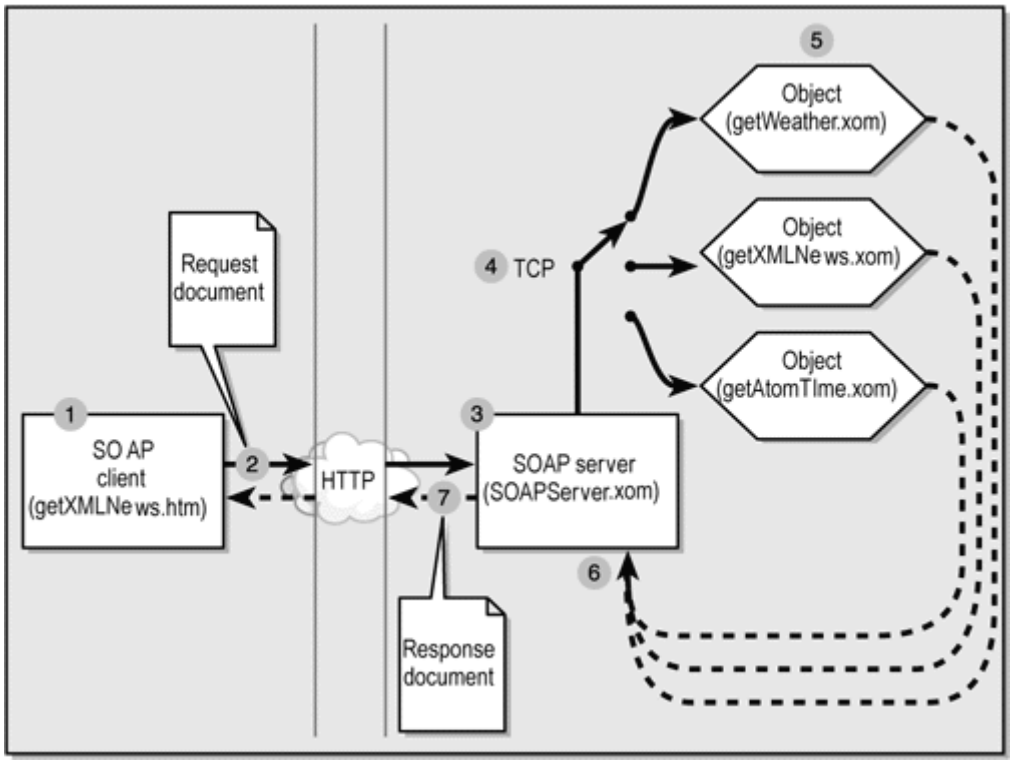


图 8-3 在浏览器中建立 SOAP 请求文件、张贴至 SOAP 伺服程序、处理该请求，然后传回一个 SOAP 响应文件，表 8-1 会说明该过程。

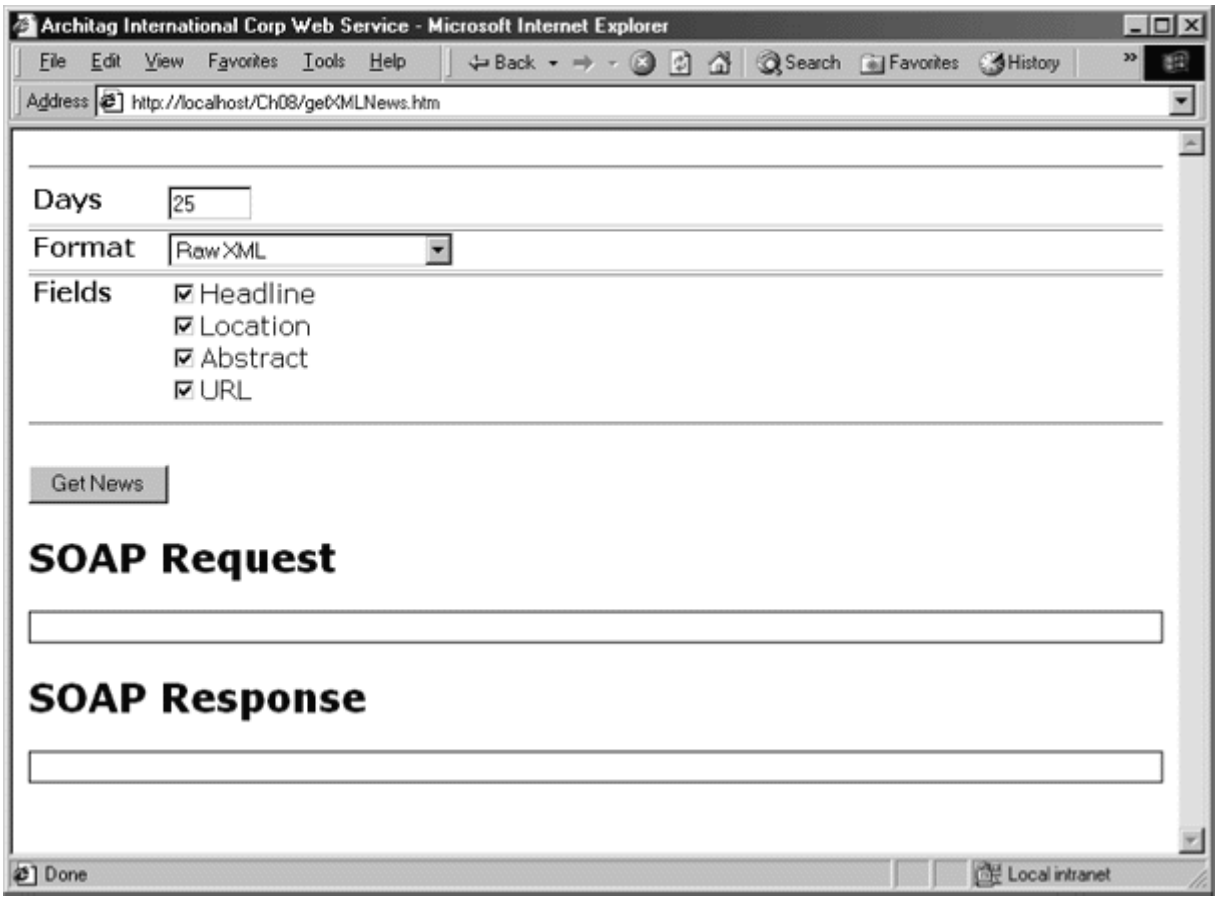


图 8-4 显示于 Web 浏览器的 getXMLNews.htm 文件

getXMLNews.htm

```
<HTML><HEAD>
</HEAD>
<BODY STYLE="font-family:Verdana;">
    <HR>
     <TABLE FRAME="VOID " BORDER=="1 " RULES=="ROWS " WIDTH="100%">
        <TR>
```

```
    <TD WIDTH="80 " VALIGN=="TOP"><B>Days</B></TD>
    <TD VALIGN="TOP"><INPUT TYPE="TEXT " SIZE="5 "
        ID="newsDays " VALUE=="23"></TD>
</TR>
<TR>
    <TD WIDTH="80 " VALIGN=="TOP"><B>Format</B></TD>
    <TD VALIGN="TOP"><SELECT ID="newsFormat">
      <OPTION VALUE="XML">Raw XML</OPTION>
      <OPTION SELECTED VALUE="HTML3">HTML 3</OPTION>
      <OPTION VALUE="HTML4">HTML 4 -Netscape</OPTION>
      <OPTION VALUE="HTML4">HTML 4 -Explorer</OPTION>
      <OPTION VALUE="xHTML">xHTML</OPTION>
      <OPTION VALUE="Text">Text</OPTION>
      </SELECT>
  </TD>
</TR>
<TR>
    <TD WIDTH="80 " VALIGN=="TOP"><B>Fields</B></TD>
    <TD VALIGN="TOP">
        <INPUT CHECKED TYPE="CHECKBOX " NAME=="newsFields"
```

```
                    ID="fldHeadline "
VALUE=="headline">Headline</INPUT>
            <BR><INPUT CHECKED TYPE="CHECKBOX " NAME=="newsFields "
                 ID="fldLocation " VALUE=="location">Location</INPUT>
            <BR><INPUT CHECKED TYPE="CHECKBOX " NAME=="newsFields"
                 ID="fldAbstract " VALUE=="abstract">Abstract</INPUT>
            <BR><INPUT CHECKED TYPE="CHECKBOX " NAME=="newsFields"
              ID="fldURL " VALUE=="url">URL</INPUT>
            </TD>
        </TR>
    </TABLE>
    <HR>
    <BR><INPUT TYPE="BUTTON " VALUE=="Get News"
        ONCLICK="ResultDiv.innerHTML=getXMLNews();">
    <H2>SOAP Request</H2>
    <DIV ID="SOAPDiv " STYLE=="border:1pt black solid;"> </DIV>
    <H2>SOAP Response</H2>
    <DIV ID="ResultDiv " STYLE=="border:1pt black solid;"> </DIV>
</BODY>
</HTML>
```

清单 8-4　用于建立 SOAP 客户端接口的最初 HTML 程序代码

步骤	说明
1	首先我们在 Web 网页中的窗体建立 SOAP 客户端，您将会从伺服端程序初始化 SOAP 请求（伺服端程序会从数个不同的数据来源取得信息）。
2	SOAP 文件透过 HTTP 联机以 **POST** 命令方式传送，在本范例中使用 **localhost**，它会指向本机的 Web 服务器。
3	SOAP 伺服程序（使用 OmniMark 撰写的 CGI 程序）接收该请求，伺服程序会读取 SOAP 请求，并且决定该请求使用的方法名称是否是它所了解的。
4	服务器开启适当服务的 TCP 联机，并且传送 SOAP 文件给该服务。
5	该服务是另一个 OmniMark 程序，它会聆听一个特定的通讯端口以等待启动。该服务会执行必要的工作和传回一个封装给该服务器。
6	服务器组成 SOAP 响应封装，包含来自于服务执行的结果。
7	回应文件传送回等待的客户端（以 HTTP 回应封装的形式）。

表 8-1　说明图 8-3 中的 SOAP 请求处理过程

要完成这个网页，我们需要请求 Web 服务以便收集来自 XML newsfeed 的项目，稍后我们将建立回应这个请求的 Web 服务。

getXMLNews.htm 档案让使用者可以请求我们储存在 Web 服务器上的信息，使用者可以指定新闻的格式、以及传回几天前的新闻等等。我们的 Jscript 将会处理使用者的请求、要接收几天前的项目、信息的格式和响应中要包含的字段。

Get News 按钮将会设定下列处理程序：

1. 执行建立 SOAP 文件的 JScript 函数，SOAP 文件中包含将会传送至 Web 服务的请求。
2. SOAP 文件经由 HTTP **POST** 命令方式传送至 Web 服务。
3. Web 服务执行下列工作，依次为：
    - 读取 SOAP 文件，并从中萃取出数据区块（payload）。
    - 剖析数据区块以决定呼叫方法的参数
    - 基于参数中的指示以查询数据库
    - 准备一个响应封包
    - 包装响应封包，使其成为 SOAP 响应对象

- 传送回应封包，使其成为 HTTP 回应到浏览器中

4. JScript 函数接收该响应，并且使用 XSLT 样式表转换该响应。

清单 8-5 显示传送 **POST** 命令至 SOAP 伺服程序的函数，表 8-2 提供了清单 8-5 中程序代码的说明，您可以在图 8-4 中看见完整的 Web 网页。

getXMLNews# POST Function

```
1  <SCRIPT LANGUAGE="JScript">
2       var xmlhttp =new ActiveXObject("Microsoft.XMLHTTP ");
3
4       var objStyle =new ActiveXObject("MSXML2.DOMDocument ");
5       objStyle.async =false;
6       objStyle.load("ShowXML.xsl ")
7
8       var SOAPRequest =new ActiveXObject("MSXML2.DOMDocument ");
9       SOAPRequest.async =false;
10
11      var SOAPResponse =new ActiveXObject("MSXML2.DOMDocument ");
12      SOAPResponse.async =false;
13
```

```
14     function getXMLNews()
15         {
16         xmlhttp.Open("POST",
17             "http://architag.com/scripts/SOAPServer.xar",false);
18         strXML =
19         "<SOAP:Envelope " +
20         "   xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1 '>" +
21         "   <SOAP:Body>" +
22         "     <getXMLNewsRequest " +
23         "     xmlns='urn:schemas-architag-com:getXMLNews '>" +
24         "       <days>" + document.all.newsDays.value +"</days>" +
25         "       <format>" +
document.all.newsFormat.value+"</format>" +
26         "       <fields>"
27         if (fldHeadline.checked)strXML +="headline "
28         if (fldLocation.checked)strXML +="location "
29         if (fldAbstract.checked)strXML +="abstract "
30         if (fldURL.checked)strXML +="url "
31         strXML +="         </fields>" +
32         "       </getXMLNewsRequest>" +
```

```
33          "    </SOAP:Body>" +
34          "</SOAP:Envelope>"
35
36          SOAPRequest.loadXML(strXML)
37          SOAPDiv.innerHTML =
38              SOAPRequest.transformNode(objStyle.documentElement)
39       xmlhttp.setRequestHeader("SOAPMethodName","getXMLNewsRequest
"")
40          xmlhttp.Send(SOAPRequest.xml);
41          SOAPResponse.loadXML(xmlhttp.responseXML.xml)
42          return
SOAPResponse.transformNode(objStyle.documentElement)
43          }
44  </SCRIPT>
```

清单 8-5 显示传送 **POST** 命令至 SOAP 伺服程序的函数，表 8-2 提供了清单 8-5 中程序代码的说明，您可以在图 8-4 中看见完整的 Web 网页。

行	说明
1	JScript 的程序区块的开始。
2	我们使用 **ActiveXObject** 以初始化 **XMLHTTP** 对象。

4	我们需要建立数个 XML 剖析器的执行个体，首先我们建立包含 XSL 样式表的执行个体。
5	**async** 属性控制剖析器如何加载文件，如果我们设定 **async** 属性为 **false** ，剖析器将会等待文件完全加载或发生错误；如果我们设定 **async** 属性为 **true** ，剖析器将会开始异步处理。如果我们要在背景加载文件，设定为 **true** 较佳，然而它会使程序变的复杂，因为我们必须指定当加载完成时，程序才能执行。
6	**load** 方法会从自变量中指定的 URL 加载文件，在本范例中，我们的 XSL 样式表位于目前的目录中。
8-9	XML 处理器的第二个执行个体将会在最后掌控 SOAP 请求文件。
11-12	XML 处理器的第三个执行个体将会掌控来自于 SOAP 伺服程序的响应文件。
14	Web 网页会呼叫 **getXMLNews** 函数。
16-17	**XMLHTTP** 对象的 **OPEN** 方法建立至伺服程序的联机。在本范例中，伺服程序是位于 Colorado 的 Architag 服务器上的 OmniMark 程序。第二个自变量是 **async** 属性，它的工作方式跟 XML 剖析器的对象的 **async** 属性一样。我们将其设定为 **false** 以强制该程序要到接收响应后才能执行。
18-34	**strXML =** 建立 XML 文件为连续的字符串、从 Web 网页的字段中插入值。该程序称为 **XML by concatenation** ，我们也会使用 DOM API，藉由插入元素和属性节点以建立 XML 文件。
36	一旦我们建立该 XML 文件，我们应该将它加载 XML 处理器，现在我们有可以操作的对象了。
37-38	这两行程序代码会产生使用者界面，使用 XSLT，我们可以转换刚刚加载的请求文件，使其成为 HTML 字符串并且显示在网页之上。在 HTML 的原始档中，您将会看见 HTML 的 **DIV** 元素中有 **SOAPDiv** 的 **ID** ，这一行会设定 **innerHTML** 属性为文字字符串， **SOAPRequest** 对象的 **transformNode** 方法会呼叫 XSLT 处理器，并且建立 HTML 对象。您可以藉由设定 **DIV** 的 **innerHTML** 属性以在网页上检视该对象。

39	SOAP 规格建议在 HTTP 请求标头中包含 **SOAPMethodName** 参数，这个参数值应该要是您要呼叫方法的名称。
40	**Send** 方法传送封包至 SOAP 伺服程序（SOAP 伺服程序在第 16 行被开启），请记住 **SOAPRequest** 对象是在内存中、二进制的 XML 对象。 **Send** 方法首先使用 **SOAPMethodName** 参数建立 HTTP 标头，然后建立空白行（为了将标头和数据主体分开），最后连结 XML 文件。
41	现在我们等待，因为我们将 **async** 属性设为 **false** ，该处理程序将会在第 40 行等待，直到 SOAP 伺服程序传回 HTTP 响应封包。该封包以 XML 文件形式（Unicode 字符的连续数据串流）传回。如果我们想要在指令码内存取该封包，必须将数据串流加载 XML DOM 中（ **LoadXML** 方法就执行了这个工作）。
42	我们想要处理传回值为 HTML 文件，然后传回该文件至呼叫的函数，在本范例中，该函数设定了另一个 **DIV** 的 **innerHTML** 属性。

表 8-2　清单 8-5 中 JScript 程序代码的说明

使用 OmniMark 建置 SAOP 服务器

SOAP 伺服程序在清单 8-5 中的第 16 行被开启，该服务器是 OmniMark 程序，并且也会呼叫其它的 OmniMark 程序－SOAPServer.xom。SOAPServer.xom 执行为一台 HTTP 服务器，现在就让我们来看看 OmniMark。

OmniMark 是免费的，您可以在随书光盘的 [\Samples\OmniMark] 目录中找到软件，其中包含了整合式的开发环境（IDE）。因为 OmniMark 有大量的自学文件，所以它非常好学，在随书光盘中也包含了相关的电子书（OmniMark at work, Volume 1: Getting Started）。

首先将介绍第一个 CGI 程序，称为 SOAPServer.xom（我们的指令码将会呼叫该程序）。该程序将会剖析 SOAP 讯息，并且决定是否加载掌控请求的服务。第二个程序 SOAP.xin 接着执行为一个服务，该服务会聆听一个已知的 IP 通讯端口，当我们的 CGI 程序传送讯息至该服务的通讯端口时就会启动该服务。由 CGI 程序传送的讯息是 XML 数据串流，该服务会处理请求、组织响应，然后传送结果回等待的 CGI 程序。

本范例中的 XML 文件会在 SOAP 伺服程序（以 OmniMark 撰写）和 Web 服务（以 OmniMark 撰写）之间传送。然而，请您特别注意很重要的一点是：提供 Web 服务环境的程序都不了解 XML。SOAP 伺服程序的工作是解译 SOAP 请求，并将其转换成服务对象所需要的语言或形式。这就表示即使您使用 20 年前的 COBOL 程序也能以服务方式正常的执行－换句话说，SOAP 伺服程序只需将 SOAP 请求解译成 COBOL 程序呼叫所需的环境，然后再将 COBOL 程序的响应转换成 SOAP 响应文件。

SOAPServer.xom

我们呼叫第一个程序－SOAPServer.xom，您可以透过 URL 直接呼叫 OmniMark 程序或从自变量档案中重新导向它－如同范例所做的，该自变量档案就是 SOAPServer.xar、清单 8-6 显示了该自变量档案。

OmniMark 也有丰富的命令列接口，SOAPServer.xar 包含被传送至该接口的选项。在第一行中呼叫 SOAPServer.xom 服务器程序为来源档案（-sb）；在第二行中的 **-i** 包含一些共享档案的路径；在第三行中的 **-x** 包含编译链接库的路径。任何程序产生的错误或是传送至 **#error** 的数据都会记录至第四行中的 **-log** 中所指定的档案。

SOAPServer.xar

```
-sb SOAPServer.xom
-i "c:\program files\omnimark\xin\"
-x "c:\program files\omnimark\lib\=L.dll"
-log SOAPServer.log
```

清单 8-6　呼叫 OmniMark SOAP 伺服程序的自变量档案

清单 8-7 包含完整的 SOAPServer.xom CGI 程序，在表 8-3 中会说明该程序。

SOAPServer.xom

```
1    include "SOAP.xin"
2
3    process
```

```
4       local stream methodName initial {""}
5       local stream SOAPResponse
6       local stream formVars variable initial-size 0
7       local stream remote-host initial {"localhost"}
8       local TCPConnection myConnection
9       local stream myRequest
10      local stream responseXML
11
12      cgiGetEnv into formVars
13      set methodName to UTIL_GetEnv ("HTTP_SOAPMETHODNAME ")
14
15      open SOAPResponse as buffer
16      do scan #main-input
17          match any{formVars{"CONTENT_LENGTH"}}
18              =>inputData
19          do when methodName =""
20              put SOAPResponse "%n<SOAP:Fault>"
21                  ||"%n <faultcode>101</faultcode>"
22                  ||"%n <faultstring>no method specified"
23                  ||"%n</faultstring>"
```

```
24                  ||"%n</SOAP:Fault>"
25      else
26          do when portSOAP has key methodName
27              set myConnection to TCPConnectionOpen
28                  on remote-host
29                  at portSOAP {methodName}
30                open myRequest
31                  as TCPConnectionGetOutput myConnection
32                put myRequest inputData ||crlf
33                close myRequest
34
35                do when TCPConnectionIsConnected myConnection
36                  set responseXML to
37                      TCPConnectionGetCharacters myConnection
38                  put SOAPResponse responseXML
39                done
40          else
41                put SOAPResponse "<SOAP:Fault>"
42                  ||" <<faultcode>101</faultcode>"
43                  ||" <<faultstring>method not supported:"
```

```
44                      ||" %g(methodName)</faultstring>"
45                      ||"</SOAP:Fault>"
46          done
47      done
48  done
49  close SOAPResponse
50
51  output "Content-Type:text/xml " ||crlf
52      ||"Cache-control:private "
53      ||crlf ||crlf
54      ||"<?xml version='1.0 '?>"
55      ||"%n<SOAP:Envelope "
56      ||"%n  xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1 '>"
57      ||"%n  <SOAP:Body>"
58      ||SOAPResponse
59      ||"%n  </SOAP:Body>%n</SOAP:Envelope>%n"
60
61  catch #external-exception
62          identity catch-id
63          message catch-msg
```

```
64          location catch-loc
65       output 'An external exception occurred in SOAPServer.%n '
66          ||'%g(catch-loc)%n '
67          ||'%g(catch-id):%g(catch-msg)%n '
```

清单 8-7　OmniMark SOAPServer.xom CGI 程序

行	说明
1	include 会执行您预期的工作：SOAP.xin 中包含用于该程序，以及该程序呼叫的 Web 服务所需的常数和函数集合。
3	OmniMark 有两种模式：它可以扮演在以规则为基础、事件驱动的环境中，如果有某些事件（通常是输入的数据串流）启动了规则，就会执行某些动作，以这种方式执行的话，OmniMark 就好像第四代的对象导向程序语言；另一种模式－会呼叫一个处理程序－就像第三代的语言，例如 C。我们的 CGI 程序是属于后者（一个处理程序），在这一种模式中，所有的动作将会以撰写的次序执行。
4-10	OmniMark 有四种基本的变数型别： counter （整数数据型别）、 stream （字符串）、 switch （布尔值）和 pattern 。除了这四种内建的数据析型别，OmniMark 的外部函数可以定义它们自己的数据型别，称为 opaque （未定）资料型别。当我们连接数据库并从查询取得记录时，我们将使用未定数据型别。 除了 pattern 变量以外，您必须宣告所有的变量为通用或区域。通用变量可以在程序的任何地方使用；区域变量只能在您定义该变量的范围之内使用（更多的相关信息请参阅随书光盘中的 OmniMark at work 第五

	章）。我们将会使用一个除错数据串流，如果程序没有正确的执行，该数据串流将会包含我们所需的信息。 我们必须宣告变量为一个范围中的第一个元素。在本范例中，范围就是该处理程序，这些变量在整个处理程序都可以使用。如果在次要的范围中有宣告相同的区域变量名称，内部的区域变量会覆写外部的区域变量（在内部范围期间），当内部变量结束时，外部变量又会重新取得掌控权。 在此欢迎您来到阶层式的程序设计。
4	**methodName** 变量中将会包含我们所呼叫的方法名称。
6	任何的 OmniMark 变量都可以以「shelf」的方式存在，shelf 是相似对象的集合，可以藉由他们的索引位移（index offset，例如 item1、item2）或描述性的索引键（descriptive key）存取 shelf。 这里我们以 **formVars** 名称建立 stream 的集合，其中关键词 **variable** 指出该集合是相似 item 的 shelf，而不是只有单一的变数。
8	TCPConnection 未定变量型别是经由包含在 SOAP.xin 中的外部链接库之一所定义。我们建立一个区域名称 **myConnection** ，它会扮演我们建立 TCP 联机的 handle。
12	**cgiGetEnv** 函数会取得所有的 CGI 相关的环境变量值，该函数会建立一个 stream 的 shelf，并且将值以成对方式（名称－值）储存。其中包含 item 的索引键即为该环境变量的名称，item 的值即为该变量的值。 **formVars** shelf 现在包含了 stream 的集合，并且每一个 stream 都使用相关的索引键指定，您可以在表 8-4 中看看这些服务器变量。

	要存取这些值的话，您需要指定集合中的成员。举例来说，要存取 **QUERY_STRING**，OmniMark 的程序代码如下： `set querystring to cgiGetEnv["QUERY_STRING"]`
13	**cgiGetEnv** 函数只能存取预先定义变量或预期的服务器变量。HTTP 1.1 允许我们建立自己的环境变量，我们用于 SOAP 的环境变量是 **SOAPMethodName** 参数。 **UTIL_GetEnv** 函数可以存取 HTTP 标头中的任何参数，要取得我们想要的参数，需要在参数名称之前增加 **HTTP_** 并且将任何字都转换为大写。
15	您可以使用 **set** 来设定 stream 为某值，或者您可以开启、附加和关闭一个 stream 为您想要的形式。在本范例中，我们开启 **SOAPResponse** stream 的方式和开启内存中档案的方式一样，因此我们可以随时放资料在里面。
16	OmniMark CGI 接口会显露（expose） **POST** 方法的内容为来自数据来源 **#main-input**，其为一只读 stream。 **do scan** 的控制结构会比对（使用 **match** 叙述）一系列的字符串，如果发现任何的对应相符，就会执行 **match** 叙述以下的动作。
17	本行比对所有被张贴的文字， **any** 对应式宣告自身将搜寻一个单一的文字。如果 **any** 后面的大括号中有整数，它将会搜寻该整数所指定数量的文字。在本范例中，我们查询在 HTTP 标头中被传送的 **CONTENT_LENGTH** 环境变量，并且实际撷取的文字数量存入称为 **input** 的对应式变量。

19	do when 的控制结构就如同许多语言中的 if 叙述一样。在本范例中，如果没有设定 methodName，我们就要指出一个错误，因为无法知道要请求什么方法。
20-24	SOAP 定义了一个元素供 SOAP 请求发生错误时使用， faultcode 和 faultstring 所设定的程序代码也许可以辅助 SOAP 客户端对问题可以有较多的了解。 OmniMark 连结字符 || 会将字符串连结为一个单位，我使用该字符以使程序易于读取；而 %n 字符串会插入换行字符。

26	portSOAP 变量是 item 的索引键数组（在 SOAP.xin 中初始化），该数组中包含服务的名称，以及每一个服务聆听的通讯端口。该宣告如下所示：
	
	`global stream portSOAP variable initial {`
	`"5432" with key "getWeather",`
	`"5433" with key "getXMLNewsRequest",`
	`"5434" with key "getAtomTimet"}`
	
	如果这个 stream shelf 有 item 的索引键等于请求的 methodName，我们就知道要传送封装至 shelf 的成员中所指定的通讯端口。

27-31	现在我们需要建立至通讯端口的联机，建立后我们就可以和该服务通讯。
32	一旦建立了联机并且开启了通讯端口， put 就可以传送数据。在本范例中，OmniMark 对象察觉到 XML 和 SOAP，所以我们传送整份 SOAP 文件。然而，如果对象不了解 XML 或 SOAP，我们也可以使用这个相同的技术，SOAP 伺服程序的工作只是负责在 输入的 SOAP 请

	求 和 **需要服务该请求的对象** 之间的解译。 同时正在聆听指定通讯端口的服务会接手，清单 8-9 显示了该服务（getXMLNews.xom），它将会接收该封装、执行它的工作并且传回封装给服务器。在本范例中，getXMLNews 程序会传回 XML 文件，其中包含该服务所产生的数据。
35	一个检查循环以确定联机是否仍然存在。
36-37	服务器在此等待，直到服务传回字符。当字符传回后， **responseXML** stream 就会被撷取。
40-45	如果我们没有确认 **methodName** ，我们需要建立一个包含有用信息的 **SOAP:Fault** 封装。
49	像档案 stream 一样，OmniMark stream 在被读取前必须关闭，现 在 **SOAPResponse** stream 中包含我们要传回客户端的数据区块。
51-59	输出的动作将传送某些信息至输出 stream，这是阶层式程序语言的巧妙之处。OmniMark 允许您以许多方式重新导向输出。在本范例中，输出 stream 是呼叫 CGI 程序的客户端。
51-52	我们需要设定一些 HTTP 标头变量。
53	在一列中有两个换行字符分开 HTTP 标头和数据主体。 因为我们建立 XML 文件，所以要指定 XML 宣告，其后接着 **SOAP:Envelope** 根元素。
61-67	OmniMark 有丰富的非预期错误掌控接口， **catch** 指定如果发生重大的程序错误事件，就会执行其指定动作。

表 8-3　说明清单 8-7 中的 OmniMark SOAP 伺服程序

变数	说明
AUTH_TYP	目前使用的验证通讯协议，该变量只能在服务器支持时和存取 CGI 程序需要验证时才能定义。
CONTENT_LENGTH	Web 服务器传送至 CGI 程序的信息长度，以字节为单位，当 CGI 程序使用 **POST** 方法以处理来自 HTML 窗体的输入时，您将使用该变量。
CONTENT_TYPE	Web 服务器传送至 CGI 程序的内容类型。
HTTP_CONNECTION	客户端和伺服端使用的联机类型，举例来说－ **HTTP_CONNECTION = Keep-Alive** 。
HTTP_HOST	被存取计算机的 IP 地址或主机名称。
HTTP_USER_AGENT	客户端系统执行的浏览器软件和操作系统。
QUERY_STRING	当该窗体是客户端使用 **GET** 方法传送时，包含来自窗体的加密数据；如果窗体是使用 **POST** 方法传送，这个环境变量就没有设定，您就需要透过标准输入方式传送加密数据至 CGI 程序（以 OmniMark 术语来说，就是 **#process-input** ）。

表 8-4　您可以由 **cgiGetEnv** 函数存取的服务器变量

getXMLNews.xom

清单 8-8 和 8-9 显示了等待 **getXMLNewsRequest** 方法的服务，服务的文件名称为 getXMLNews.xom。清单 8-8 显示了伺服端的循环、聆听适当的通讯端口、透过 TCP 读取数据，并且呼叫提供服务的函数，表 8-5 中有详细的说明；清单 8-9 显示服务的提供者：开启数据库、取得记录和建立 XML 数据区块的函数，表 8-6 中有详细的说明。

getXMLNews.xom

```
include "SOAP.xin"
declare catch server-die

process
    local TCPService myService
    local TCPConnection myConnection
    local stream caughtData
    local stream responseXML

    set myService to TCPServiceOpen
       at portSOAP{"getXMLNewsRequest"}
    put #error "getXMLNews listening on "
       ||portSOAP{"getXMLNewsRequest"}||"%n"
    throw server-die
```

```
15          unless TCPService-is-working myService
16
17      repeat
18          set daysWanted to 0
19
20          set myConnection
21              to TCPServiceAcceptConnection myService
22          throw server-die
23              unless TCPService-is-working myService
24          set caughtData to
25              TCPConnectionGetCharacters myConnection
26
27          put #error "getXMLNews caught and processed data on "
28              ||portSOAP{"getXMLNewsRequest"}
29              ||"%n----%n%g(caughtData)%n----%n"
30
31          clear fieldsWanted
32          using group processXML do
33              do xml-parse instance scan caughtData
34                  suppress
```

```
35              done
36          done
37
38          open responseXML as TCPConnectionGetOutput myConnection
39          put responseXML getXMLNews(daysWanted,formatType,
40              fieldsWanted)||crlf
41          close responseXML
42          catch #program-error
43              put #error "caught #program-error%n"
44
45          catch #external-exception
46                  identity catch-id
47                  message catch-msg
48                  location catch-loc
49              output 'An external exception occurred.%n '
50              output '%g(catch-loc)%n '
51              output '%g(catch-id):%g(catch-msg)%n '
52      again
53
54      catch server-die
```

```
55          open responseXML as
56              TCPConnectionGetOutput myConnection
57          put responseXML "<msg>getXMLNews Server killed</msg>"
58              ||crlf
59          close responseXML
60          halt
61  ;-----------------------------------------
62  group processXML
63  ;-----------------------------------------
64    element #implied
65        suppress
66
67  element format
68    set formatType to "%c"
69
70  element days
71  set daysWanted to "%c"
72
73  element fields
74    repeat scan "%c"
```

```
75        match [any except " "]+=>field
76            set new fieldsWanted key field to field
77                unless fieldsWanted has key field
78        match any
79    again
80
81  element die
82    do scan "%c"
83        match content-start POISON-PILL content-end
84            put #error "Shutting Down%13#%10#"
85            throw server-die
86  done
```

清单 8-8　等待 **getXMLNewsRequest** 方法的服务

行	说明
1	**include** 会执行您预期的工作：SOAP.xin 中包含用于该程序，以及该程序呼叫的 Web 服务所需的常数和函数集合。
2	OmniMark 的 **catch** 和 **throw** 能力让您可以宣告您自己的规则，该规则会撷取程控下的封装或非预期状况的结果。在本范例中，我们宣告了一个 **catch** ，如果某人意外的将服务器停机（非预期状况），我们就可以存取这个 **catch** 。

4	同表 8-3 所述。
5-6	我们需要宣告两个供 TCP 服务和联机使用的变量，这些变量将会透过我们所通讯的 SOAP 伺服程序导入。
10-11	该服务在 SOAP.xin 中宣告的 stream 所指定的连接埠被开启。
12-13	**#error** 只能写入，通常用于程序从命令列执行，当程序开始时这个讯息会显示在屏幕上。
14-15	如果 TCP 服务出现错误，一个 **非预期** （exception）讯息会传到 **catch** 中。
17	不像您在本章前面所看见的 CGI 程序，该程序会一直在内存中执行直到它被删除或 TCP 服务管理员发生联机中断情形。 **repeat again** 控制结构是一种循环，该程序每接收一个讯息该循环就会执行一次。
20-21	循环每执行一次，我们就要建立一次 TCP 服务的联机。
22-23	我们需要检查 TCP 服务是否正常工作，如果没有，就会传出 **非预期** （exception）讯息。
24-25	循环中的这两行是服务器花最多等待时间的地方， **TCPConnectionGetCharacters** 函数将会撷取 SOAPServer.xom 程序传送至本程序的讯息。
27-29	在命令列窗口显示封装。
31	SOAP 封装指定服务请求哪些字段，会有一个 shelf 包含这些字段，因为我们现在处于伺服端的情况下，可预期该程序会接收许多的请求，所以我们必须确认已经清除了 shelf，要清除这个 shelf，必须将其中 item 的数量设为 0，因此它们可以让其它程序加以初始化。
32-36	OmniMark 内建 XML 处理能力，因为我们接收的是 XML 文件，我们需要搜寻该文件以找到处理程序所需的参数。程序中有一个 group 称为 **processXML** ，该 group 中包含了遇到 XML 元素时如何处理的规则。这个程序代码藉由呼叫 XML 剖析器以在 **caughtData** 中处理 XML 文件。

62	**group** 宣告建立了一个我们可以使用名称加以呼叫的程序范围，这个 group 中包含了元素规则的集合，当剖析器在输入 stream 中遇到特定的元素就会启动这些规则。在本范例中，会有数据经由第 33 行的 **do xml-parse** 传送到这里。
	
	我们处理的 XML 文件如下：
	
	`<SOAP:Envelope`
	`xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">`
	`<SOAP:Body>`
	`<getXMLNewsRequest`
	`xmlns="urn:schemas-architag-com:`
	`getXMLNews">`
	`<days>10</days>`
	`<format>XML</formats>`
	`<fields>headline url abstract</fields>`
	`</getXMLNewsRequest >`
	`</SOAP:Body>`
	`</SOAP:Envelope>`

64-65	如果在特定事件发生时没有元素规则存在，**#implied** 元素规则将会启动，这个元素规则将会处理在 SOAP 名称空间中的所有元素。

67-71	当遇到 **format** 元素时就会启动这个元素规则以设定 format type 为适当的值， **%c** 是指该元素的内容，每一个元素规则都必须使用 **%c** 或 **suppress** 实际的处理该内容一次（请注意，除了 Raw XML 以外，getXMLNews.xom 不支援任何 format type）。 **days** 元素规则会撷取 **days** 元素，并且设定适当的变量。
73	当遇到 **fields** 元素时就会启动这个元素规则。
74-79	**repeat again** 控制结构相似于 **do scan** ，除了它会重复处理字符串，直到遇到不符合的字符才结束。每一个 **match** 叙述后面的对应式（pattern）会评估现在被扫描的内容是否符合。一旦在循环中发现符合就会执行下面的动作，并且继续处理下一个被扫描的字符。 在本范例中，被扫描的内容是为了搜寻想要的字段。当发现符合字段后，在 shelf 中就会建立一个新的 **fieldsWanted** item。
81	这个伺服端程序会永远执行，您也许会为了某种理由删除该程序，要删除该程序，您可以在 SOAP 请求中加入下列元素（该字符串设定在 SOAP.xin 中）： `<die>poison-pill string</die>`
82-83	poison pill 字符串必须为 **die** 元素中唯一的字符串－要执行这个动作可以使用 **content-start** 和 **content-end** 。
84-85	如果发现了该字符串，一个错误讯息会传送至命令列窗口，并且 非预期（exception）讯息会传到 **catch** 中
54-59	这个程序会传送一个 **msg** 元素、关闭联机和中止程序。
38	一旦我们有了所需的全部资料，我们就可以传送资料回客户端（客户端应该还在等待响

	应）。 **responseXML** 用于建立响应 stream，现在我们需要做的是开启输出联机、传送数据和关闭该联机。
39-40	**getXMLNews** 函数会在稍后说明，这个函数会传回包含新闻的 XML 文件，取决于所选取的参数。
42-43	OmniMark 有丰富的非预期错误掌控能力，第 42 行将会撷取任何程序错误（都是一些执行期间错误，例如试图将数据放进未开启的 stream 或从开启的 stream 中存取值所引发的结果）。
45-51	第 45 行的非预期 handler 将会撷取外部错误，例如试图存取未开启的数据库或试图写入只读文件。
52	结束从第 17 行开始的循环。

表 8-5　说明清单 8-8 中的服务，该服务等待 [getXMLNewsRequest] 方法

getXMLNews return # function

```
1  macro outField token tagName is
2    do when fields has key tagName
3        put newsXML "%n <%@(tagName)>"
4            ||cleanString
5                (dbFieldValue rsXMLNews key tagName)
6            ||"</%@(tagName)>"
7    done
8  macro-end
9
10 define stream function getXMLNews (
```

```
11        value counter daysDesired,
12        value stream format,
13        read-only stream fields)as
14    local stream newsXML initial {""}
15    local dbDatabase dbXMLNews
16    local dbField rsXMLNews variable
17    local stream strSQL
18    local stream minusDate
19    local stream finalDate
20
21    set dbXMLNews to dbOpenODBC "XMLNews"
22    set minusDate to add-to-ymdhms now-as-ymdhms days -daysDesired
23    set finalDate to format-ymdhms "=xM/=xD/=xY " with -date
      minusDate
24        set strSQL to "SELECT *FROM News "
25            ||"WHERE Date >=#%g(finalDate)#ORDER BY Date DESC"
26        dbQuery dbXMLNews sql strSQL record rsXMLNews
27
28        open newsXML as buffer
29        do when dbRecordExists rsXMLNews
```

```
30          put newsXML "%n<getXMLNewsResponse days='%d(daysWanted)'>"
31          repeat
32              exit unless dbRecordExists rsXMLNews
33              put newsXML "%n <item date='"
34                  ||dbFieldValue rsXMLNews key "date " |||"'>"
35
36              outField "url"
37              outField "location"
38              outField "headline"
39              outField "abstract"
40
41              put newsXML "%n </item>"
42              dbRecordMove rsXMLNews
43          again
44
45          put newsXML "%n</getXMLNewsResponse>"
46      else
47          put newsXML "%n<getXMLNewsResponse days='%d(daysWanted)'/>"
48      done
49
```

```
50    close newsXML
51    return newsXML
52
53    catch #external-exception
54            identity catch-id
55            message catch-msg
56            location catch-loc
57        output 'An external exception occurred.%n '
58        output '%g(catch-loc)%n '
59        output '%g(catch-id):%g(catch-msg)%n '
```

清单 8-9　OmniMark 函数会传回 XML 文件，该文件会包含在 SOAP 讯息中

行	说明
10-13	OmniMark 有两种建立可呼叫程序代码的方式：这几行包含了一个函数定义，您可以从程序中的任何地方呼叫该函数－getXMLNews 会传回 stream 变量，要使用三个自变量呼叫此函数：counter、stream、a shelf of streams。
14	我们建立 XML 文件于 **newsXML** stream 中，我们将该文件当成 SOAP 的数据区块（payload）传回。
15-16	OmniMark Database library（OMDB）提供了对象的集合和可以简单连接数据库的函数。在这里我们建立了两个数据库对象：一个是 ODBC 数据库联机，另一个是 SQL 查询传回的结果集。
21	**dbOpenODBC** 函数会使用 **XMLNews** 名称开启 ODBC 联机，并传回一个 handle。

22-23	为了建立 SQL 查询，我们需要建立一个日期，该日期是以目前日期为基准向前推 N 天。我们以 **daysDesired** 当成 getXMLNews 函数的自变量来传回时间的位移值（offset）。OmniMark 有数个函数用以执行时间的运算，以 **year-month-day-hour-minute-second** 格式储存的所有日期称为 **ymdhms** 。首先我们可以从目前日期减去 **daysWanted** 以决定想要的日期，然后我们格式化这个日期成为 SQL 处理器接收的形式。在本范例中，日期以 **4/26/2000** 形式出现。
24-25	我们藉由插入上述计算出的日期以建立 SQL 查询，然后使用 **dbQuery** 函数传送查询至连接的数据库。一个结果集会在 **rsXMLNews** 中被传回。
28	**newsXML** stream 被开启为 file-like buffer（在内存中），因此我们可以附加资料于其上。
29	**dbRecordExists** 函数会检查 SQL 查询是否传回任何结果集。
30	如果该查询有传回结果集，我们需要使用 **getXMLNewsResponse** 根元素开始建立 XML 文件。
31-43	**repeat again** 循环将会在每当查询传回一笔记录就执行一次。
32	一旦最后一笔记录进行处理， **dbRecordExists** 函数就会传回 **false** 以结束循环。
34	我们可以在结果集对象中存取字段－藉由其中的每一个项目都有一特定的索引键存取该项目的值，我们使用 **dbFieldValue** 函数来存取这些信息，本行将会输出字段名称为 **date** 的值。
36-39	**outField** marco 将会被呼叫四次，传送数据库中的字段名称。
1-8	这个 marco 会检查指定的字段是否为我们要的字段，如果是的话，该 marco 将会建立一个元素，并且加入字段的值。
42	**dbRecordMove** 函数会加载下一个记录至 **rsXMLNews** 中。

45	本行结束 **getXMLNewsResponse** 文件。
47	如果没有记录从 SQL 查询中传回，我们会建立空元素。
50-51	关闭 stream 并且将它传回呼叫程序。
53-59	如果某个外部接口发生问题（通常是找不到档案或某些数据库错误），这个 **catch** 将会执行。

表 8-6　在清单 8-9 中的伺服端程序说明

9. BizTalk 基础结构

您可曾经想过，如果数百家不同的公司都建立它们自己描述发票的架构（schema）会发生什么事？这些架构中会有彼此兼容的机会吗？这些架构中也许会包含相同的元素，但是每一种架构使用的元素名称却都不相同。举例来说，所有的发票都可能会有发票号码，但是发票号码的元素名称也许会是 **Invnum** 、 **InvNo** 、 **InvoiceNo** 、 **InvoiceNum** 等等。我们需要终结这数百种不同的 XML 文件，让它们全部都表示相同的数据，因为每一种文件都有不同的结构，会造成这些公司共享文档的困难，如果您所有商业合作伙伴都使用不同的架构，您便需要针对每一个伙伴撰写不同的程序才能符合其 XML 文件。很明显的，您不会想要这样，您会希望所有的企业伙伴使用相同的结构。

这个例子指出了我们需要一种基础结构（Framework）以开发标准的产业架构，这个基础结构会指定某些对象的共享元素集合，并且应有足够的弹性允许人们共同合作开发架构的环境，而 BizTalk 就试着达成这个目的。

BizTalk Framework 提供了一系列规则和卷标来建立商业对商业（business-to-business，B2B）的电子商务架构，建立 BizTalk 是为了藉由让 XML 更容易建立可供企业伙伴使用的架构并促进 XML 的使用，建立架构的共享规则集合将会促进更多架构的建立，同时这也会让商业伙伴认同要使用哪些架构，BizTalk 基础结构使广大的支持者采纳共享的方式使用 XML。

您需要 BizTalk Framework Compliant（BFC）伺服程序来处理 BizTalk 文件，BizTalk Framework 让独立软件卖商（independent software vendors，ISVs）和开发者可以更容易的以对应（map）

的方式，将一个商业程序对应至另一个，因此各式各样使用 XML 的产业可以更快速地采纳商业之间的电子交换方式。未来，藉由使用一致的格式建立大部分的架构，Biztalk Framework 同时也提供了清楚的工具设计目标，以及 ISVs 建立下一代电子商务和应用程序整合式产品的基础结构。

BizTalk 由下列四种元素组成：

- **The BizTalk Independent Document Specification** BizTalk Framework 是卷标的集合，这些卷标提供了从一个位置到另一个位置取得文件的地址定义（addressing definition），并且在每一端都会呼叫处理程序。BizTalk Framework 就像在商业合作伙伴中传送商业文件的虚拟信封（virtual envelope），您可以在附录 A 和随书光盘的 **\References** 数据夹中找到这个规格的副本。

- **BizTalk.org** BizTalk.org 是一个提供学习电子商务技术的 Web 网站。该网站中有讨论群组支持开发架构（schema）并辅助您将它们整合进 BizTalk Framework（附录 B 中将讨论更多关于 BizTalk.org）。BizTalk.org 也是架构的存放区，允许您张贴您的架构供其它人存取，该服务是免费的。

- **BizTalk 伺服程序** 您会需要 BizTalk 伺服程序以处理您的 BizTalk 文件，它是一套用以读取 BizTalk 文件的软件，然后以智能型的方式处理这些文件。而这些智能型的处理方式将取决于您的系统，以及处理文件时需要做些什么？试想其它服务器软件，例如邮件服务器，许多邮件服务

器都可供公司组织使用、执行于任何计算机平台，且价格从便宜到贵都有。相同的，BizTalk 伺服程序可以使用任何语言撰写并执行于任何计算机平台。许多公司都会使用处理 BizTalk 文件的伺服程序，并且会整合至它们的后端系统，您将会在 第十一章 看见基本 BizTalk 伺服程序的范例。

- **Microsoft BizTalk Server 2000** Microsoft BizTalk Server 2000 只可以执行在 Microsoft Windows 2000 Server 之上，因为 BizTalk Framework 开放式规格的特质，您也可以在 Linux 系统（其上执行以 Java 撰写的 BizTalk 伺服程序）上建立 BizTalk 讯息，并且在 Windows 2000 Advanced Server（其上执行 Microsoft BizTalk Server 2000）上读取讯息。Microsoft BizTalk Server 2000 不只是读取和撰写 BizTalk 文件的软件，该产品包含完整的 B2B 整合环境，允许您和您的商业伙伴使用 XML 通讯。

BizTalk 文件和讯息规格

BizTalk Framework 提倡建立 XML 的文件结构用在 B2B 的商业交易，在本节中，您将学习更多相关 BizTalk Framework 的知识。

有些人说使用 XML 建立数据就好像使用 ASCII 产生文字文件：格式十分有弹性，并增加了兼容性。让我们谈谈假设您使用 ASCII 建立文件，笔者应该能够读取它们，没错吧？那么如果您使

用法文写字，然后我用英文读取，这样的话，即使我们都使用 ASCII，双方仍然无法通讯。使用相同的技术即表示我可以在我的计算机上读取您的档案，但是前提是我必须了解您建立信息时的基础结构，才能完全了解您的讯息。同样的，在 XML 里我们也有相同的问题，这表示如果您建立 XML 文件，他人并不一定能够了解，而只能确定文件符合 XML 语法的规则，如果有架构可以使用的话，文件会经由架构中所定义的结构为合法有效的。

使用 XML 的问题在于：描述信息的范围是从决定要描述什么到决定每一个字段要命名什么，一旦使用者跨越了这些重要的因素，便需要将数据加上卷标，因此其它应用程序可以确认传送了什么。

BizTalk Framework 提供程序开发者一种方式来撰写更易于处理 XML 文件的应用程序，BizTalk Framework 定义了一套卷标的集合和设计指导，以及建立您自己的自订卷标集合提供您与您的商业伙伴交换信息。换句话说，BizTalk Framework 定义了使用 XML 的一致方式，您可以简单的决定您接收什么类型的 XML 文件和该文件中包含什么信息。

BizTalk Framework 将 XML 文件视为商业讯息，它只会注意文件如何建立，以及文件传送的路径是如何，BizTalk Framework 提供了一种机制－程序化的「邮递（mail）」XML 文件。

许多产业致力于「架构」来解决信息的交换问题，这些架构不需要在这些产业之间共享，举例来说，用来传送医疗记录而撰写的架构，就不适用于金融交易使用。这样造成这些产业只关心数据结构定义，而不关心两个位置之间的交易如何发生。

Microsoft 倡导 BizTalk，并且包含许多指导委员会，其中有 Ariba、Boeing Compaq、J.D.、Edwards、SAP、webMethods 和 UPS。因为许多产业彼此合作，BizTalk 努力的成果和全球组织以及特定工业订定的有一点点不同，BizTalk 提供了一个基础结构，专供指定如何从一个位置到另一个位置取得信息的整合。

如您所示，被传送的实际信息可以为使用任何架构的任何文件，使用这种方式，BizTalk 变成了商业文件的包装者，就像您所使用的信封一样。让我们看看典型的旧式商业交易，然后看看我们如何使用 XML 执行相同的交易。

在 B2B 交易中的 BizTalk Framework

让我们看看一个使用 BizTalk Framework 建立 XML 文件的 B2B 商业交易情境仿真。

假设我们是玩具汽车的制造商，为了制造玩具汽车，要购买零件、必需品，以及来自数个供货商的服务，这些供货商也有他们自己的供货商和其它的客户。

我们想要使用 Web 和这些供货商通讯，并且和客户、零售商店以及使用者保持联系。当然，我们尽可能重新使用现有的流程。

准备采购订单

事件会触发商业交易，这个事件可能起因于组织中的某人想要从卖商购买产品，这个特定的事件需要传送采购订单至该组织中。

假设我的公司想要从您的公司购买 10 个小引擎，在我的公司中的程序之一是建立订购需求窗体，这是一份适用于我的公司部门之间通讯的内部文件。

在填写订购需求之后，我将它拿给在我的公司中负责处理需求和建立采购订单的人，Jean，她了解处理这些文件的商业规则，并依照规则响应我的请购。

Jean 要做的第一件事情是确认我有使用公款的权限，Jean 看她的预算清单，并且注意到我的名字在清单上，确认我有足够的权利可以执行这个采购。

下一步 Jean 必须确认我们在库存中已经没有我请购的产品，订购库存中还有的产品是不明智的，这个商业规则是用来避免这个错误发生的可能。

一旦 Jean 确定我的订购是合法有效的，她会将我的采购要求放在她桌子上的角落，那个角落放了一大堆采购要求，因为同公司中的其它人也会想买您公司的产品。我们两个公司之间的合约有这样的规定：除非是紧急请求优先处理，否则一定要等到至少订购您公司三项产品时，才会开始处理采购订单。而 Jean 注意到之前已经有两份请求要订购您公司的产品，加上我一共三份，因此现在是建立这三份采购订单的时机了。

Jean 在她的文字处理程序中开启一个窗体，并且开始填写该窗体上的字段，这些字段包含住址信息、金融数据和传送期限等等，另外还有产品号码、说明和金额。

Jean 完成这个窗体并打印出来，然后她又开启另一份文件的窗体（一个信封），她在左上角输入我的公司名称和住址，在信封的中间输入您的公司名称和住址。她不需要特别注明在信封中间的住址是「To」；在左上角的住址是「From」，因为数个世纪以来，我们早就已经都知道在信封中间的住址是目的住址；而在左上角的住址是回邮住址。

Jean 特别在您的公司名称之上，注明了重要的信息：「Attn:Purchase Order Processing Department」，稍后我们将会看见这个特别的记号会引起您的公司执行特定的动作。

Jean 将这个信封打印出来，并且将采购订单放进信封中，加上邮资然后走到转角，将信封放入蓝色的箱子，透过 U.S. Postal Service，这个信封将会经过一趟神秘之旅然后辗转到您公司的邮件室中，这是故事的重要部分。

如果我们公司双方都设计了一个采购订单传递系统－意指我们不需要使用邮局所提供的服务－我们便需要花大量的时间在设计传递系统上才能应付直接的商业订购程序。公司双方都需要开发一套住址结构，雇用快递员或与快递服务签约，并且考虑保险、意外和数千个其它问题。然而，很幸运的，一个适当的基础架构已经存在了，我们只需要付出一点点小小的代价就可以使用。

很重要的一点是 Jean 从我们公司传递讯息到您公司的时候，可以使用许多不同的方法，她刚刚是使用 U.S. Postal Service，但是她也可以使用 FedEx、Airborne、Express 等等，她也可以使用传真订购或是使用电子邮件，甚至使用 Web 接口。

数天后，信封出现在您公司的邮件室中，邮件室的职员注意到「Attn:Purchase Order Processing Department」的标示，所以就将该信封交给公司内部负责处理采购订单的人－David。

处理采购订单

David 拆开信封并且读取其中的商业文件，他是你们公司中专司处理各种不同商业文件的人，例如采购订单、发票和提案请求（Requests for Proposals，RFP）等等。

David 注意到这份文件是属于采购订单，所以他进入「采购订单处理模式」，他了解您公司中处理采购订单的商业规则。首先，他会确认两家公司之间是否有执行这些交易的商业合约，这个商业合约中可能会订定期限和条件等等，例如传送时间、回件原则和付款明细。然后他会确认我们公司已经付款，同时这笔交易在两家公司之间没有任何争议，因为寄出产品至一家没有付款的公司是不明智的。

一旦 David 确认了两家公司之间有商业上的合约，他就开始进行「采购订单处理系统」，并且将信息从采购订单解译至他的系统中（使用您公司中适当的商业规则），而您公司的规则之一就是传送请求至仓储中看看是否有采购订单中要求项目的库存。

David 发现请求的三个项目中两个有库存，所以可以立即交货，但是有一个项目已经没有库存了，因此必须延期交货。而我们两家公司的合约中有这样的规则－您的公司必须传送给我们公司采购订单的确认单。因此 David 在他的文字处理程序中开启一个窗体，并且填入所需的信息，他注明了两个项目将会在今天传送，而另一个项目必须延期交货。

然后 David 又开启一个信封窗体，输入两个住址，他参考 Jean 传送给他的住址，将两个住址交换，我们公司的住址放在信封的中间，您公司的住址放在左上角。

他将这两个窗体印出来，把确认单放入信封中，并贴上邮资放入邮筒中。数天后，Jean 取得确认单并且使用我们公司的商业规则处理它，这些规则之一就是必须联络提出采购订单的三个人，先通知要求订购的其中两个人，告诉他们订单的状况；另一个是他所订购的产品必须延期交货的人，这个人也许会要求 Jean 取消到您公司的订单并且将订单传至其它供货商，这个事件也许会触发交易双方其它特定的商业规则。

这只是一个情境模拟，Jean 可以使用其它方式应付订单请求，举例来说，如果一笔订单十分紧急，Jean 也许会将该笔订单与其它两笔订单分开处理，Jean 也许会直接传真紧急订单给 David，甚至直接打电话给他以确认是否有收到订单。有时候 Jean 也会决定使用夜间快递传送采购订单和支持文件。这些都是传输的议题，完全取决于 Jean 如何使她的工作更有效率，而她所做的每一个决定都是基于公司中的商业规则。

其实重点在于商业规则如何配合公司之间的环境而不同，XML 的来临对于两家公司中任何一家的商业需求都不会有什么改变，但是我们可以预期的是 XML 和 BizTalk Framework 所能够提供的优点就是：让我们能更有效率的完成商业合作。

数据交换程序的进化

稍后范例中的公司已经开发出一套完成其商业程序的应用程序与商业流程，我们选择计算机程序、接口和员工使得所有的工作都能顺利执行。

您也可以选择一套可以符合您公司所需的应用程序和商业流程，然而我们之间的系统可能会完全不兼容。如果公司交换商业信息是使用纸为通用的交换格式，则纸上作业一定需要人类的介入。

人类需要了解商业规则，并且要能够执行解译，在我们的研究案例中，Jean 和 David 就是分别在两家公司中扮演这种角色。

范例所描述的交易并不是凭空捏造的，在传送任何采购订单之前，交易双方公司都会有商业上的合约协议，合约中会指定付款期限、传送期限，以及未详尽部分的解决方式等等，合约中的一部分也会注明交易所需的窗体信息。以采购订单为例，您应该告诉我们您需要何种信息才能够顺利的处理订单，并且我们也会告诉您有关我们所需确认单的信息。

这种类型的交易模式已经持续了数世纪，但是自动化的系统却可以使它更有效率，BizTalk Framework 减少了该模式中取得信息的复杂度。

使用 BizTalk Framework 下订单

BizTalk 文件是属于 SOAP 1.1 的讯息，该讯息的数据主体中包含一个以上的商业文件。SOAP 信封在 第八章 中已经说明过了，图 9-1 显示了 BizTalk 文件的不同部分。

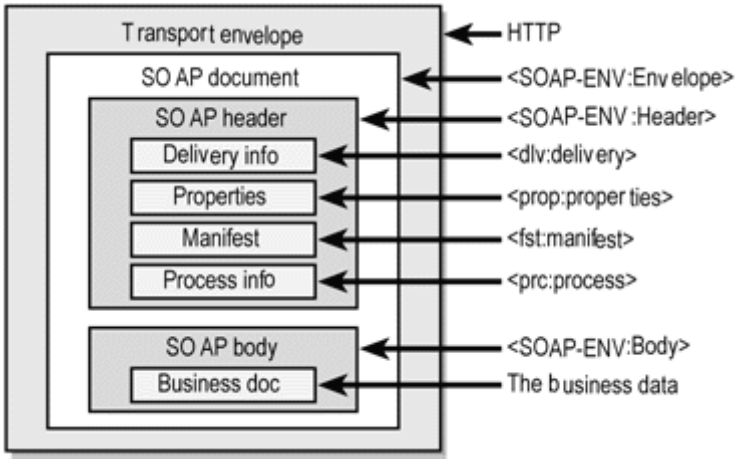


图 9-1 BizTalk 文件是属于 SOAP 1.1 的讯息，该讯息中有数个元素是设计用以传递和路由商业文件

BizTalk 文件可以视为 SOAP 文件的延伸，在 BizTalk 文件中有一套元素集合，这些元素都有名称空间，每一个都为了特定的目的而最佳化。这些名称空间中的标签称为 **BizTag** 。

整份文件都包含在 SOAP **Envelope** 元素中－这就表示， **Envelope** 是文件的根元素。SOAP 文件包含了一个选择性的 **Header** 元素和必要的 **Body** 元素。在 BizTalk 讯息中， **SOAP:Header** 元素是必要的，因为其中包含了一些必要的 **BizTag** 。 **SOAP:Body** 元素包含由一或多个商业文件组成的商业数据。

表 9-1 列出了在 BizTalk 标头的 **SOAP:Header** 元素中的 **BizTag** 。每一个元素都有自己的名称空间，您可以在 schemas.biztalk.org 网站中得到更多相关信息。

BizTag 名称	必要性	名称空间
delivery	必要	http://schemas.biztalk.org/btf-2-0/delivery

properties	必要	http://schemas.biztalk.org/btf-2-0/properties
manifest	非必要	http://schemas.biztalk.org/btf-2-0/manifes
process	非必要	http://schemas.biztalk.org/btf-2-0/process

表 9-1　包含在 BizTalk 标头的 **SOAP:Header** 元素中的 **BizTag**

BizTalk 文件本身并没有任何功能，除非您将它传送至某处。而传送文件的工作是由如图 9-1 的 **传输信封**（transport envelope）负责，它可以为任何电子传输通讯协议，例如 HTTP、SMTP 或 FTP。BizTalk 2.0 规格中只描述了 HTTP 连结（HTTP binding）。

让我们看看 BizTalk Framework 如何工作，清单 9-1 显示了一份 BizTalk 文件，该文件就是情境模拟中 Jean 和 David 所使用的信封和窗体。表 9-2 中详细说明清单 9-1 的每一行，附录 A 中包含了 **BizTalk Framework 2.0 草案：文件和讯息规格** 。

NewPO.xml

```
1  <?xml version="1.0 "?>
2  <SOAP-ENV:Envelope
3      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
6      <SOAP-ENV:Header>
7          <dlv:delivery SOAP-ENV:mustUnderstand="1"
```

```
8               xmlns:dlv="http://schemas.biztalk.org/btf-2-0/
                delivery"
9               xmlns:agr="http://www.trading-agreements.org/types/">
10               <dlv:to>
11  <dlv:address xsi:type="agr:department">Toy Car Parts</dlv:
    address>
12              </dlv:to>
13              <dlv:from>
14  <dlv:address xsi:type="agr:organization">Toi Carz</dlv:
    address>
15              </dlv:from>
16              <dlv:reliability>
17  <dlv:confirmTo>http://toicarz.com/biztalk/confirm.asp</dlv:
    confirmTo>
18  <dlv:receiptRequiredBy>2000-07-7T08:00:00+08:00</dlv:
    receiptRequiredBy>
19              </dlv:reliability>
20          </dlv:delivery>
21          <prop:properties SOAP-ENV:mustUnderstand="1"
22              xmlns:prop="http://schemas.biztalk.org/btf-2-0/
```

```
            properties">
23  <prop:identity>uuid:74b9f5d0 33fb 4a81 b02b 5b760641c1d6</
    prop:identity>
24              <prop:sentAt>2000-05-14T03:00:00+08:00</prop:
                sentAt>
25              <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:
                expiresAt>
26              <prop:topic>http://toycarparts.com/BTServer.xar</
                prop:topic>
27        </prop:properties>
28  <fst:manifest xmlns:fst="http://schemas.biztalk.org/btf-2-
    0/manifest">
29        <fst:reference fst:uri="#PurchaseOrderBody">
30              <fst:description>Purchase Order</fst:description>
31        </fst:reference>
32        <fst:reference fst:uri="toicarz.sig">
33              <fst:description>Digital signature</fst:description>
34        </fst:reference>
35     </fst:manifest>
36   </SOAP-ENV:Header>
```

```
37
38    <SOAP-ENV:Body>
39        <PurchaseOrder_1.0
40            xmlns="x-schema:http://toicarz.com/schemas/
              PurchaseOrder.xdr"
41            id="PurchaseOrderBody"
42            type="order"
43            PONumber="10-01-2118">
44            <Item number="122-11 " quantity= ="100"/>
45            <Item number="237-82 " quantity= ="10"/>
46            <Item number="811-91 " quantity= ="25"/>
47        </PurchaseOrder_1.0>
48    </SOAP-ENV:Body>
49 </SOAP-ENV:Envelope>
```

清单 9-1　包含采购订单的 BizTalk 文件

行	说明
1	此为一份 XML 文件，所以用 XML 宣告开头。
2	一份 XML 文件必须只能有一个根元素于文件的顶端，并且在底部结束。因为这是一份 SOAP 信封，所以 BizTalk 2.0 文件的根元素是 **SOAP-ENV:Envelope** 。

3-5	这份文件使用三个 XML 名称空间宣告指定架构将被 SOAP 信封使用。
6-39	**SOAP-ENV:Header** BizTag 包含您的 BizTalk 文件的识别和路由信息。
7-20	**dlv:delivery** BizTag 包含相关文件的传递和确认信息。
7	**mustUnderstand** 属性指示接收的应用程序必须了解自己接到了什么，如果该应用程序不了解，它必须传回 **SOAP:Fault** 错误元素，不能试着伪装处理程序。
8	藉由这个名称空间宣告可以找到 **delivery** BizTag。
9	宣告另一个商业伙伴合约的名称空间。
10	**dlv:to** BizTag 指定将接受您文件的实体名称。
11	**xsl:type** 属性指出数据型别为 **XML Schema Part 2:Datatypes Working Draft** 指定，该标准由 W3C 发布。
12	**to** 数据型别由商业伙伴合约架构中定义。
13-15	**from** 和 **to** 住址包含相同的属性类型和窗体。
16-19	BizTalk 伺服程序建立讯息文件，该文件可以确认讯息是否被接收。这个动作由新增选择性的 BizTag－ **dlv:reliability** 完成。
17	**dlv:confirm** BizTalk 提供一个住址，可以使用这个住址传送收据（已经接收到讯息的回条，情境模拟中的确认单）。
18	如果传送系统没有在 **dlv:receiptRequiredby** BizTag 指定的时间收到收据，该传送系统将会重传文件。

21-27	定义 BizTalk 文件的内容。
22	在 **prop:properites** 元素中的 BizTag 属于由名称空间宣告中所指定的名称空间。
23	**prop:identity** BizTag 包含 Universally Unique Identifier（UUID），大部分的平台都可以产生该识别码。
24	**prop:sentAt** BizTag 包含指出文件送出的时间戳记（timestampe），这是 **datetime.tz** 资料型别。
25	**prop:expiresAt** BizTag 包含文件过期的时间，在时间之后，相关的 BizTalk 文件都视为过期，并且无法藉由目的地的商业实体处理。
26	**prop:topic** BizTag 包含将处理文件的 BizTalk 处理器地址。
28-35	**fst:manifest** BizTag 是选择性的，它包含了 BizTalk 文件内容的目录。
29-31	**fst:reference** BizTag 会发生一或多次，它指定了 Uniform Resource Identifier（URI）和提供 BizTalk 文件中包含什么的描述。第一个 **fst:reference** 元素会指向在 SOAP **Envelope** 的 **Body** 元素。
32-34	这个文件也包含数字签名档案以保证文件的来源无误，它可能会用 MIME-encoded 附加档案的方式被包含在传输信封中。
38-48	**SOAP-ENV:Body** 元素会接在 **SOAP-ENV:Header** 之后， **Body** 元素就跟情境仿真里信封中的文件一样， **Body** 元素包含您的商业文件，并且使用您的架构以定义。
40	因为您的架构定义了您的文件，您需要使用名称空间宣告指定您的架构，在范例中使用的

	架构称为 [PurchaseOrder.xdr]，位于我们的伺服程序上，接收的应用程序将会发现该架构（如果该应用程序有存取 Web 的话）。
41-49	所有剩下的都是采购订单本身的信息，以及结束 **PruchaseOrder_1.0** 、 **SOAP-ENV:Body** 、 **SOAP-ENV:Envelope** 元素的卷标。

表 9-2　逐行说明清单 9-1 中的 BizTalk 文件

这个 BizTalk 文件实际上是两份文件，一份是信封；一份是信封中的窗体。名称空间和 XML 架构让我们建立虚拟的信封和虚拟的文件（信封中的窗体）。

说明

您应该还记得文件类别定义（Document Type Definition，DTD）的架构语法在一份文件中只允许单一种文件类别。BizTalk 文件至少有两个文件类别，它允许使用 XML Data Reduced（XDR）架构语法，因此如果您使用文件类别定义的话，这种类型的 BizTalk 交易处理是不可能的。

当您的公司接收清单 9-1 中的 XML 文件时，BizTalk 伺服程序会读取文件并且搜寻其中的信息。首先 BizTalk 伺服程序搜寻 **dlv:to** BizTag 以决定哪一个合作伙伴要接收这份文件，接着 BizTalk 伺服程序会剖析剩下的数据项，并且在订单追踪数据库中加入一笔记录，最后伺服程序将会取得实际订单－在 **SOAP-ENV:Body** 元素中的内容，然后将订单传送至处理订单的应用程序。

传递收据

因为我们使用了 **dlv:reliability** BizTag，所以会要求接收的 BizTalk 伺服程序一定要传送给我们收据，该收据如清单 9-2。

Receipt.xml

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
      <rct:receipt SOAP-ENV:mustUnderstand="1"
               xmlns:rct="http:/
      /schemas.biztalk.org/btf-2-0/receipt">
          <rct:receivedAt>2000-05-15T04:08:10-05:30</rct:
          receivedAt>
      </rct:receipt>
      <prop:properties SOAP-ENV:mustUnderstand="1"
      xmlns:prop="http://schemas.biztalk.org/btf-2-0/
      properties">
  <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b7606387dc1</prop:
  identity>
```

```
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:
            expiresAt>
            <prop:topic>http://toycarparts.com/POProc.asp</prop:
            topic>
        </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
```

清单 9-2　BizTalk 伺服程序产生的收据

文件的标头相似于被传送讯息的标头，不同之处在于 **prop:identity** BizTag 中的唯一识别码和时间戳记。

请注意清单 9-2 中的 **SOAP-ENV:Body** 元素是空的，即使该元素是必要的，也不需要任何内容，因为收据文件只是为了确定讯息已经被正确的接收。

确认单

一但您的后端系统确认了订单，将会建立出清存货的 BizTalk 文件，清单 9-3 显示了该文件；表 9-3 逐行说明该文件。

Confirm.xml

```
1  <?xml version="1.0 "?>
2  <SOAP-ENV:Envelope
3      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4      xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
6      <SOAP-ENV:Header>
7          <dlv:delivery SOAP-ENV:mustUnderstand="1"
8              xmlns:dlv="http://schemas.biztalk.org/btf-2-0/
               delivery"
9              xmlns:agr="http://www.trading-agreements.org/types/">
10             <dlv:to>
11     <dlv:address xsi:type="agr:department">Toi Carz</dlv:
       address>
12             </dlv:to>
13             <dlv:from>
14     <dlv:address xsi:type="agr:organization">Toy Car Parts</
       dlv:address>
15             </dlv:from>
16             <dlv:reliability>
```

```
17      <dlv:confirmTo>http://toycarparts.com/BTServer.xar</dlv:
        confirmTo>
18      <dlv:receiptRequiredBy>2000-07-7T08:00:00+08:00</dlv:
        receiptRequiredBy>
19          </dlv:reliability>
20      </dlv:delivery>
21      <prop:properties SOAP-ENV:mustUnderstand="1"
22          xmlns:prop="http://schemas.biztalk.org/btf-2-0/
            properties">
23      <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b7606434d3
        </prop:identity>
24          <prop:sentAt>2000-05-16T03:00:00+08:00</prop:sentAt>
25          <prop:expiresAt>2000-05-17T04:00:00+08:00</prop:
            expiresAt>
26          <prop:topic>http://toicarz.com/biztalk/confirm.asp</
            prop:topic>
27      </prop:properties>
28      </SOAP-ENV:Header>
29
30      <SOAP-ENV:Body>
```

```
31          <PurchaseOrder_1.0
32              xmlns="x-schema:http://toicarz.com/schemas/
                PurchaseOrder.xdr"
33              type="confirm"
34              PONumber="10-01-2118">
35              <Item number="122-11 " quantity="100 "
                shipped=="100"/>
36              <Item number="237-82 " quantity="10 " shipped ="0 "
37              backordered="10"/>
38                  <Item number="811-91 " quantity="25 "
                    shipped="25"/>
39          </PurchaseOrder_1.0>
40      </SOAP-ENV:Body>
41  </SOAP-ENV:Envelope>
```

清单 9-3　确认采购订单的 BizTalk 文件

行	说明
1	此为一份 XML 文件，所以会以 XML 宣告开头。
6-28	**SOAP-ENV:Header** 元素的结构比最初的文件中的 **SOAP-ENV:Header** 元素简单多了，但这个是包含关于响应文件的信息。

11, 14	**to** 和 **from** 的住址被交换了。
30	SOAP 文件的 **Body** 元素开始。
31-39	请注意确认文件使用和最初采购订单（NewPO.xml）中的架构相同，但这是不必要的，确认文件在 **Body** 元素中可以有完全不同的架构。在这个情境模拟中，我们藉由将第 33 行中的 **type** 属性改成 **confirm** 以使文件的功能不同。
40-41	**Body** 和 **Envelope** 元素的结束卷标。

表 9-3　清单 9-3 的文件说明

XML 是商业交易用来通讯的强大功能性语法，如同您在本章中所看到的，XML 藉由使用两种独立的架构定义了整个交易：BizTalk 讯息信封和采购订单。

因为 BizTalk 是根植于 XML，所以它兼容于任何商业文件规格，同时 BizTalk 也是任何想要建置 B2B 电子商务的开放式规格。

当您的公司系统接收了在 http://toycarparts.com 的信封后，**BTServer.xar** 指令码将开始处理采购订单。这个处理过程取代了在先前范例中 David 所做的工作，同时该过程也会套用 David 使用的商业规则，并且以 BizTalk 文件的形式传送确认单回我们公司。

引入自动化程序的潜在问题

这个情境模拟由 BizTalk 请求、确认和回应文件重复执行了 Jean 和 David 多年以来手动执行的程序。最大的成就是我们使用相依的计算机流程取代了人类，这样让我们可以更迅速、更有效率的完成商业活动。然而，您不必为 Jean 和 David 担心，因为有他们才会有 BizTalk 的出现，他们现在成为公司中的关键资源，因为他们具备内部系统的知识，并且在公司的其它领域中建立 B2B 应用程序也需要仰仗他们。

当然，即使计算机取代了人类，相对的如果有错误也会重复的更快，因此除错也是关键性的重点。有时候如果缺少了人类的认知力，计算机会使所有的流程 看起来 像是正确的。

然而无论在此所描述系统有多么强大，新的技术仍然会有缺点。BizTalk 伺服程序扮演 Jean 的角色，它仍然必须从买商那里取得目录（当然，您也可以使用 BizTalk 讯息使用不同的工作流程，新增公司能订购的产品清单）。然而公司的系统不知道卖商是否还有库存，仍然可能会在没有库存的公司下订单，因此增加了会订购延迟交货产品的机率。

如果使用旧的系统，Jean 会考量某特定项目的价格与交货时间最为适合的卖商，这样可以省下许多花费，如果她考量的范围越大的话，她可以为公司省下越多的开支。

但如果公司合作的每一个卖商都以 Web 服务的方式将他们的目录和仓储信息实时提供给我们，我们就可以清楚的知道是否还有特定项目的库存，以及在下订单之前就可以知道该项目的价格。我们可以使用 SOAP 在短时间内从数以千计的卖商处请求这些信息，一但我们决定了价格、数量和可用性的最佳组合，我们就可以使用 BizTalk 工作流程封装以建立正式的采购订单。透过

BizTalk，我们在购物和下订单方面可以节省许多潜在的开支，甚至连订购项目如果会延迟交货的话还会收到通知。

这个情境模拟已经描述了许多公司都会遇到的状况，现在我们要将订购流程反向进行，该过程称为 **反向拍卖** （reverse auction）：卖商互相竞争并提供他们的产品给请求的厂商，并且提供越来越低的价钱以响应购买商出的价钱，而实时的 Web 服务可以使这个愿景成真，就如同开放式的标准一样，SOAP 和 BizTalk 可以促进这些实际经验被广泛的采纳。

第 3 篇 BizTalk 侵服程序

10. BizTalk 系统需求

. 以任何传输方式都可以传递可信赖的信息

. 安全性

. 路由

. 工作流程

. 同步和异步通讯

. 队列

. 批处理支援

. 文件的追踪和活动状况

. 商业伙伴管理

. 扩展性

. 文件转换

. 协力厂商和 ISV 的延伸性

.EDI 的相互操作性

. 多重 API 接口

. 未来的适应性

. 重点在于以使用者为本位的商业驱动模式

11. 建立 BizTalk 伺服程序

. 程序和数据流

. 基础结构

. 数据库

. 系统安装

. 输入采购请求

. 处理采购订单

. 采购订单确认的处理

12. Microsoft BizTalk Server 2000

. BizTalk Editor

. BizTalk Mapper

. 其它 BizTalk 工具

. BizTalk Server 数据处理

10. BizTalk 系统需求

BizTalk 伺服程序是一套读取和处理 BizTalk 文件的软件，因为 BizTalk Framework 是使用 XML 语法自我描述的结构，所以扮演 BizTalk 伺服程序的应用程序必须了解 XML－换句话说，该伺服程序必须能够加载 XML 文件和存取其内容，这是 BizTalk 伺服程序最基本的需求。当然，只会加载和存取 XML 文件的伺服程序在企业的计算环境中并没有多大的实用性， **BizTalk Framework 2.0 文件和讯息规格** 将执行这些基本工作的软件称为 **BizTalk Framework Compliant Server** 或 **BFC Server** 。

本章将会谈及许多当您为公司环境中选择或建立 BizTalk 伺服程序时，您可能会想要了解的许多需求。如果您使用本书的定义范畴，您会感觉到 BizTalk 伺服程序就如同读取 BizTalk 文件的筛选器一样简单，它会搜寻封装中特定的信息并且传送该信息（整份或部分文件）至适当的应用程序；或者在整合性的连锁管理系统中，BizTalk 伺服程序也可以扮演极为重要的精密组件；BizTalk 伺服程序也可以是整个企业中与商业合作伙伴执行商业流程的应用程序。

您可以撰写 BizTalk 伺服程序（应用程序）于任何操作系统平台上执行，并以任何语言撰写 BizTalk 伺服程序来呼叫 XML 处理应用程序。BizTalk 伺服程序可以在任何操作系统上面执行，包含 Microsoft Windows 和 UNIX，同时您也可以使用独立于平台以外的语言，例如 Java、Python 或 OmniMark 等撰写 BizTalk 伺服程序。

您的 BizTalk 伺服程序应该有足够的弹性能够适应动态商业环境的变更需求、有精密的接口可以驱动旧型系统的应用程序并与其共存，并且可以由您的商业使用者日复一日的操作 BizTalk 伺服

程序。因为并不是每个公司都使用相同的系统，所以您的 BizTalk 伺服程序应该也要有跨语言和跨平台的互动性。在本章中，我们将会看到 BizTalk 伺服程序的必备功能。在 第十一章 中，我们将看看这些功能的范例。

以任何传输方式都可以传递可信赖的信息

BizTalk 文件本身十分的完美，然而，我们主要是需要其实用性，而不是一昧的赞赏而给予很高的评价。BizTalk 需要被传输才能够到达它的目的，将 BizTalk 文件从您的公司送到您的商业伙伴需要某种传递系统，也因为您所面临的是电子信息，您可以使用「位」的传送程序传递信息。

HTTP 是传输位最简单的方式了，它无所不在，并且全世界的防火墙都允许 HTTP 的呼叫封包传透。但是并非所有的卖商都有能力透过 HTTP 接受请求，还有某些人可能会觉得透过 HTTP 传送简单文字并非十分理想。File Transport Protocol（FTP）可以代替 HTTP，FTP 需要使用者 ID 和密码，这一层增加的登入控管减少了接收伪造讯息的风险。如果您的商业伙伴没有 FTP 伺服程序，他们可能会有电子邮件，您的 BizTalk 伺服程序也应该能够以 Simple Mail Transport Protocol（SMTP）传送讯息，SMTP 伺服程序是设计用来在 Internet 上转传讯息。

如果您的商业伙伴还活在二十世纪，可能没有电子邮件，因此您的 BizTalk 文件也许需要更原始的传输方式。如果要具有最大的弹性，您的 BizTalk 伺服程序应该要有能力传送讯息至传真机或甚至将讯息打印出来（再使用邮件快递传送）。上述的这些方式都是为了让人类读取信息所设计

的传递方式，因此您的 BizTalk 伺服程序至 3 少也要有能力提供一种商业文件的传递方式以符合商业伙伴的需求。

使用 HTTP、FTP 和 SMTP 的主要问题就出在它们预设是没有加密的，这个问题也因为这些传输通讯协议通常仰赖 Internet 而变得更糟，这样会将您的请求被任何可以撷取 TCP 封包的人拦截，因此让我们开始考量另一个需求－安全性。

安全性

透过 HTTP 传送 BizTalk 文件也代表负担了某些安全性上的风险，因为 HTTP 文件以纯文字的方式传输，在传输过程中可能会有某人读取该讯息，BizTalk Framework 2.0 规格使用 S/MIME 或 PKCS 安全性层保护您的 BizTalk 文件。

藉由使用 HTTPS 通讯协议应付开放式交易的问题也是一个简单的方式，HTTPS（「S」即为安全性）被 Web 广泛的采用，以保护敏感性交易的信息，例如账户收支或信用卡号码。HTTPS 使用公开金钥加密技术在客户端和伺服端之间保护封包，当客户端（通常为 Web 浏览器）请求一安全性文件时，Web 伺服程序就会传送给它一个加密金钥，在工作阶段中客户端会使用该金钥加密数据，并且只有伺服程序有能力解密数据。HTTPS 的加密程度取决于使用的加密金钥长度，如果您十分关切安全性的问题，您应该将伺服程序升级至使用最大长度的加密金钥以提供最佳的安全性。

BizTalk 伺服程序应该要能够提供安全性的传输通讯协议，使文件能够安全的到达目的地，您也许也需要应付未经授权文件传送至商业伙伴公司的问题，如果您的商业伙伴收到 HTTP 的纯文字 BizTalk 请求，可能就存在某人假冒您的名义传送讯息给您的商业伙伴，这种恶意的行为自从电子邮件盛行后就已经行之有年了。

BizTalk 伺服程序应该要能够让您在讯息中附加某些形式的数字签名（digital signature），您可以简单的使用 BizTalk 文件规格的 **attachment** 元素加上数字签名。

路由

BizTalk 伺服程序最重要的功能之一就是将文件从您的系统传输至其它系统，这个功能可以很简单，也可以很复杂－简单如在 Web 伺服程序上张贴请求；困难如复杂的路由步骤集合以穿透防火墙或其它安全性机制。还有，当您从商业伙伴接收到响应时，您也可能需要将该响应讯息在内部路由到适当的部门。

试想一下监理所职员，当您进入监理所大楼后，您必须在某个窗口提供适当的窗体，例如交通罚单等等，而在窗口后面的职员会将您的窗体路由（传送的路径）到适当的部门。

BizTalk 伺服程序就像这个职员一样：它必须能够确认进来哪一种文件，并且将文件路由到适当的地方加以处理，这个路由通常包含开始工作流程处理，并且针对每一种交易类型会对应至适当的工作流程。

工作流程

当您接收 BizTalk 文件后，就会发生一系列特定的事。首先文件会被拆解成数个部分，因此应用程序就能辨别它是哪一种文件。举例来说，应用程序需要存取 **address** 和 **state** 字段中的信息，它可以决定文件的来源处及需要何种类型的处理。 **address** 字段是决定当 BizTalk 文件被接收时要引发何种工作流程的一部份。

一但您识别这个文件和文件的目的，您就可以开始进行达成目的的处理程序，这就需要 工作流程 （workflow）。举例来说，当您要处理采购订单，首先可能会想要查询要求订购该产品的公司是否跟您的公司有商业上的往来，以及该公司是否有欠款。然后您会检查要求的产品是否有库存，或是必须采购、建立及制造产品才能满足这间公司的请求。

当接收文件时，您的 BizTalk 伺服程序应该以某些方式开始适当的工作流程，先进的 BizTalk 伺服程序应该可以建立和管理这些工作流程，以供适当的应用程序能够轻易的存取这些工作流程，而负责使系统工作更有效率的使用者也应该能够维护这些工作流程。

同步和异步通讯

第九章 中，您学到了 BizTalk 讯息的结构，您也可以使用数种方式传送 BizTalk 讯息给您的商业合作伙伴。一旦您建立了 BizTalk 讯息，您必须将它放在某种传输信封（envelope）中以传送至别的地方。

BizTalk 伺服程序也应该要有同步传送讯息的方式，在同步通讯中，伺服程序会传送文件给其它 BizTalk 伺服程序并等待其响应，这是最基本同时也最不复杂的要求。

然而在实际的生产环境中，您也许会从您的应用程序传送成千上百的 BizTalk 讯息，如果您担心每一个请求的响应会造成该应用程序的效能减低，您就可能要使用异步传输讯息的方式。

队列

您的 BizTalk 伺服程序应该要有将讯息放在队列中，稍后再传递的功能，使 BizTalk 伺服程序可以执行下一个工作，最适宜的方式就是远程队列讯息了，因为它将传输中的主要信息先卸载，但是远程队列讯息必须先以目前的状态信息传回伺服程序。举例来说，如果一个伺服程序传送了一个讯息到队列中，然后着手开始下一个工作而没有收到响应的状态信息，则伺服程序就无法知道是否顺利的传递了封包到队列中。

批处理支援

BizTalk 文件也许会在一天之中不断的产生，您与商业伙伴之间维持的交易关系也许是靠每天中特定几个小时，或是每个月中特定几天的交易。您的 BizTalk 伺服程序也应该设计为可以一次传递一份独立的 BizTalk 文件，或是一次传送一批 BizTalk 文件。

文件的追踪和活动状况

在队列环境中，如果发生了非预期的问题，必须传送某些信息给伺服程序是很重要的。举例来说，如果数据的传送未完成，伺服程序需要知道发生了问题以决定要重新传送数据或是取消交易。同时伺服程序也需要以某些方式让操作者可以查询特定讯息的状态或取得处理过程的统计信息。

您的 BizTalk 应该要有精密的交易信息追踪能力，其中包含下列功能：

- 追踪独立的文件或批处理的文件。
- 知道文件是否正确的传递。
- 基于文件、商业伙伴或活动状况以查询追踪数据库。
- 在每一个文件中附加时间的信息。
- 储存使用者定义的数据，例如采购订单的汇总信息。

所有的资料都必须随时可以使用，无论是在文件生命周期中、在文件处理完成之后建立交易报表时、或是需要支持任何交易需求时。

商业伙伴管理

每一次您与商业伙伴建立合作关系时，您必须要符合双方合约的需求，包含付款日期、折扣、传递模式等等。所有的这些因素建构了您与伙伴的商业关系，而不同的商业关系都有不同的需求。

您的 BizTalk 伺服程序应该要有能力追踪双方商业关系的所有层面，这些层面包含您的公司与商业伙伴之间交易的产生与处理过程。

BizTalk 伺服程序也应该能够随着您目前所交易的商业伙伴而有不同的处理方式，举例来说，如果您正与有潜力的伙伴交涉，您也许会想要在 BizTalk 伺服程序中设定 **proforma** 以测试商业伙伴合约中的参数是否符合您系统的工作流程管理模式。

扩展性

现代化的系统应该要能够有扩充性，您的 BizTalk 伺服程序也不例外。HTTP 就是一个最佳的起点（HTTP 为一具有扩展性的传输通讯协议），您可以在中间层放置商业逻辑并且藉由使用 Intranet 存取这些商业逻辑。

您的 BizTalk 伺服程序也应该要可以掌控特定时间的预期交易数量，然而企业的成长期间也会受到许多影响因素。然而，当您在规划未来的成长时，您的重点不应该放在目前合作的公司，而是要估计您未来会合作的潜在商业伙伴。使用 XML 和 BizTalk 将会让您降低许多成本，远比传统

的电子数据交换（Electric Data Interchange，EDI）的复杂性和高成本要划算许多。因此您会因为这些低成本的电子交易，而使得您的组织与更多公司建立商业合作关系。

文件转换

每一种产业都可能会有自己的架构以便在产业成员中相互通讯，举例来说，保险产业联盟称为ACORD（ http://www.acord.com ）开发了一套架构的集合，以供保险交易相关的文件使用。旅游产业也有一套标准，用来查询可用的饭店房间和班机航次等等。如果您是这些产业中的一员，您应该可以轻易地从 BizTalk 信封的资料主体直接读取商业文件。

本书的主题之一也包含了跨产业进行商业交易，您也许会接收来自汽车产业的采购订单，其中包含了该产业成员所定义的结构。采购订单的结构在各种产业中都完全不同，然而其中却包含了相同的基本信息。再次提醒您，唯一的不同只有结构。我们探讨的方法与传统的 EDI 不同，传统的 EDI 要求一种单一的结构，无论您和商业伙伴之间需要何种信息完成交易，您和商业伙伴都要遵循该结构。而 BizTalk 伺服程序有能力存取所有的文件信息，例如您可以撰写程序以转换汽车产业的采购订单成为您所使用的采购订单，或是将它变得容易并使用 XSLT。

如同您在 第六章 中所学，要达成文件转换的目的，建立 XSLT 样式表比撰写 C 程序更为直接简易，因为 XSLT 可以直接使用 XML 输入，并且建立您所需要的 XML 输出结构。您可以将 XSLT 想象成对应工具，它会将输入的特定元素对应至输出的元素。

BizTalk 伺服程序也应该能够达成对应的目的，将每一个输入的文件加以适当的对应转换。您可以在 BizTalk 伺服程序中指定这层对应关系以符合您的商业交易合约。

先进的 BizTalk 伺服程序也应该有能力以简单的方式建立 XSLT 转换，并在您需要的时候可以测试和更新。

协力厂商和 ISV 的延伸性

BizTalk 伺服程序不是空谈，您的 BizTalk 伺服程序应该要有能力接收外挂套件的延伸性，让它可以在各种产业中有轻易完成商业交易的能力。

想要在 BizTalk 伺服程序领域建立声誉的协力厂商和独立软件卖商也许会开发许多接口，您的 BizTalk 伺服程序应该要有能力可以接收这些接口，鼓励协力厂商开发外挂套件。

EDI 的相互操作性

建立传统的 EDI 系统需要高度的自订化和整合性，许多公司建立 EDI 系统和他们的商业伙伴交易，这些系统在部署和管理上都十分的痛苦。

要让一间公司非常愉快的建置 EDI 系统只有一个方法，就是去除建立 EDI 的念头，改用 XML 和 BizTalk。然而，您并不能完全放弃 EDI 系统，因为如果您不与使用 EDI 的商业伙伴交易，您可能遗失许多潜在的商业机会。因此，您的 BizTalk 伺服程序应该可以建立和接收 EDI 交易，要达

成这个目的很简单，建立转换 EDI 文件的对应，或是撰写可以了解 EDI 及可以与您的 BizTalk 文件通讯的子系统。

多重 API 接口

您应该了解 BizTalk 伺服程序只是简单的建立和追踪 BizTalk 文件的机制，BizTalk 伺服程序并不会完全取代您的公司中后端处理程序，因此您所选取的 BizTalk 伺服程序应该要有多重 API 接口，允许您从 BizTalk 伺服程序取得信息和传送信息至 BizTalk 伺服程序。

API 是您用来与后端系统通讯的方式，包含订单完成、帐务结算、生产流程、客户追踪、销售和其它数百个要与您的商业伙伴互动的作业。

未来的适应性

您的系统应该要有足够的弹性以符合未来的商业需求，同时也要能够适应环境的变更。举例来说，如果目前的客户以传送 EDI 文件的方式与您通讯，某天却忽然想开始使用工业标准的 XML 文件与 BizTalk，您应该可以在不用重建商业伙伴合约的前提之下，就变更您的系统来符合这个要求，所有的历史信息仍然维持一致，只是通讯方式变了。

您的系统应该也要有延伸产业标准架构的能力，也许您在一个特定的产业中与十间公司有商业交易。假设某个组织致力于为该产业设定标准，并建立架构以供该产业的成员共享信息，您也许会

选择该架构集合以便和所有的产业成员交易。然而，如果您与该产业中的某一个成员有特别密集的交易关系时该怎么办呢？

使用 XML，您可以在交易文件中插入特定的字段供特定的客户使用，并且不会破坏原有的系统。同样的，如果您在任何 BizTalk 文件的数据主体中包含特定商业伙伴不需要的字段，该商业伙伴可以忽略该字段。您的 BizTalk 伺服程序应该要可以依照每一个商业伙伴的不同需求，管理其数据和建立自订的数据检视。

您的系统应该能够掌控未来的商业需求、并且有足够的弹性维护两家公司合并或一家公司拆成两家的信息。您的系统也应该要能够掌控您自己公司的合并或统一，或是组织结构的变更。

重点在于以使用者为本位的商业驱动模式

因为 BizTalk 是设计用来维护商业之间的关系，您的 BizTalk 伺服程序和它的操作应该都由商业使用者掌控，而非 IT 职员。这表示商业之间关系的建立和文件对应都应该要能够让商业使用者掌控，甚至这些使用者并非程序设计师。

这并不表示 IT 职员就不需要安装和维护 BizTalk 系统，当建置 BizTalk 伺服程序或任何相关于旧型系统的应用程序都会涉及许多整合方面的论点。您的 IT 职员主要涉及建立与旧型系统的接口。

然而，无论什么时候，当您与商业客户建立交易关系时，都不会想接触 IT 职员。主要设计商业过程的人员才能够建立与维护商业关系、完成输入与输出文件的转换、对数据结构加以变更，以及产生系统状态和商业关系是否健全的报表。

像这样的一个系统应该也要能够让远程使用者可以使用安全的方式存取，您需要建置中间层的技术－包含加密程序，好让您的系统可以提供安全的存取方式。

11. 建立 BizTalk 伺服程序

本章将带领您建立 BizTalk 伺服程序，并且让您了解它的特性。为了显示 BizTalk 伺服程序建立的过程，以及对您的企业在数据层级的辅助性，笔者开发了可以建立 BizTalk 文件的程序，这些程序会将文件传送至 BizTalk 伺服程序，并且在 BizTalk 伺服程序上进行处理。

在本章的范例中，我们从 第九章 复制了 Jean 和 David 的采购订单交易，虽然这是一个简单的范例，却包含了您开始建立 BizTalk 解决方案的所有概念。还有，您也将学习到 BizTalk Framework 如何促进交易之间数据层级的通讯。

因为 BizTalk Framework 是 SOAP 的延伸，我们将使用 SOAP 在两间公司之间传送数据，同时也会介绍一种新的工具称为 ROAP（Remote Object Proxy Engine），它是 Microsoft SOAP Toolkit for Visual Studio 6.0 的一部份，提供了一种可以定位 Web 服务的方式，并且可以使用简单的 COM 接口存取这些 Web 服务。

在 第九章 中我们所遇到的两间公司也会包含在本章范例的交易中：Toi Carz, Inc.,（www.toicarz.com）和 Toy Car Parts Corporation（www.toycarparts.com）。如 第九章 所述，Toi Carz 是制造商；Toy Car Parts 是该制造商的供货商之一。Toi Carz 是建构于 Microsoft Windows

的环境，并且使用 Active Server Pages（ASP）和 Microsoft Visual Basic；Toy Car Parts 建构于 Linux 的环境，并且使用 Apache 为他的 Web 服务器，同时也使用 OmniMark 建立三层式的 Web 应用程序。

在本章范例中，笔者想要让您了解在两台执行不同操作系统和程序语言的计算机之间是如何传送 SOAP 和 BizTalk 讯息，这些范例可以在刚刚描述的环境之下正常工作，但是基于示范性质以及让您易于理解的目的，本书预设是在 Microsoft Windows 系统环境下执行 Toy Car Parts 的系统。

在本章中的所有范例都可以在同一台计算机上执行，使用这种方式建置它们有数个理由：第一，您不必使用两台机器工作；第二，本书的大部分范例都是经过笔者精心规划，并且可以在任何网络上运作。如果真的需要在不同的网络上执行，您需要做的就只有将这些程序移到不同的计算机上，并且分别在每一个系统上变更其 URL 就可以了。变更 OmniMark 程序让它们可以在 Unix 系统上执行并不简单，但却也不是很困难的大事，在 Unix 端程序您所需要做的最大变更就是数据库的存取形式。举例来说，在 Windows 中使用 ODBC（Open Database Connectivity），OmniMark 支持 ODBC，但是 OmniMark 在 Unix 计算机上也支持 OCI（Oracle Command Interface），OCI 接口跟 ODBC 十分相似，在程序代码方面只需要做一点点的修改。

本章中所有的程序代码都包含在随书光盘的 \Samples\Ch11 数据夹，您可以藉由本章稍后的〈系统安装〉一节中所述建立本章中所描述的系统，笔者假设您已经十分熟悉 Visual Basic 语言，因此并没有花许多时间解释 VB 程序代码。

然而，因为 OmniMark 对许多读者来说都十分的陌生，因此本书花了许多的时间解释 OmniMark 程序如何工作。如果您想要提升 OmniMark 的学习速度，随书光盘也包含了相关的电子书（OmniMark at work, Volume 1: Getting Started），OmniMark 实在是一套十分强大的开发语言，您如果肯花一些时间去学习它，您将感到受益匪浅。

程序和数据流

现在就让我们看看本章中的范例是如何工作的，图 11-1 和图 11-2 显示了在 Toi Carz 和 Toy Car Parts 之间的处理流程，如同 第九章 中两家公司的员工－Jean 和 David 都是负责建立和处理商业文件，我们将使用两套计算机系统取代他们的工作。

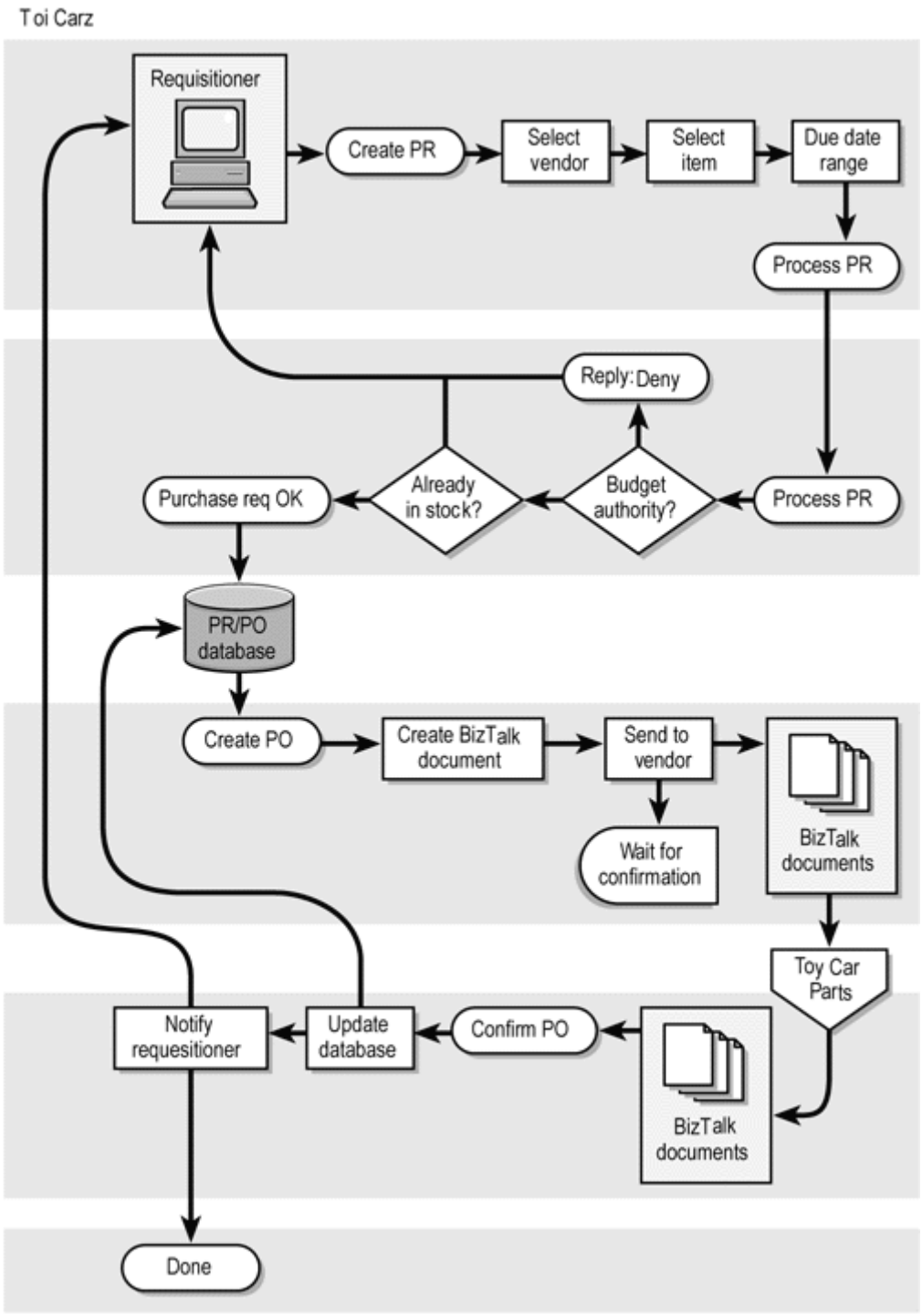


图 11-1 当采购请求开始后，进行中的工作流程

处理过程如下所述：一个事件启动了两家公司之间的处理程序。在图 11-1 中，Toi Carz 中的一个员工想要从 Toy Car Parts 采购某些产品，该员工

建立了一个采购请求（purchase requisition，PR），在采购请求中他会选择适当的供货商、产品项目和要求的交货日期范围。

这笔数据会由一套应用程序处理，该应用程序会套用 Toi Carz 的商业规则来处理数据。首先，该应用程序会检查员工的数据库以确认输入采购请求的员工是否有购买该项产品的授权，然后检查该产品是否有库存，如果还有的话，那么采购的总数将会做适度调整。如果授权和库存的查询都没有问题，则采购请求就会输入数据库中，该数据库中包含了所有经过相同程序验证的采购请求。

采购请求的数据库会被周期性的扫描，其目的是为了建立传送到 Toi Carz 的供货商的采购订单。扫描的程序会检查供货商的数据库以取得住址信息，并且建立 BizTalk 文件，每一份文件中都会包含有适当供货商的采购订单。这些 BizTalk 文件会被送到适当的供货商，其中之一就是 Toy Car Parts。Toy Car Parts 的程序和数据流如图 11-2。

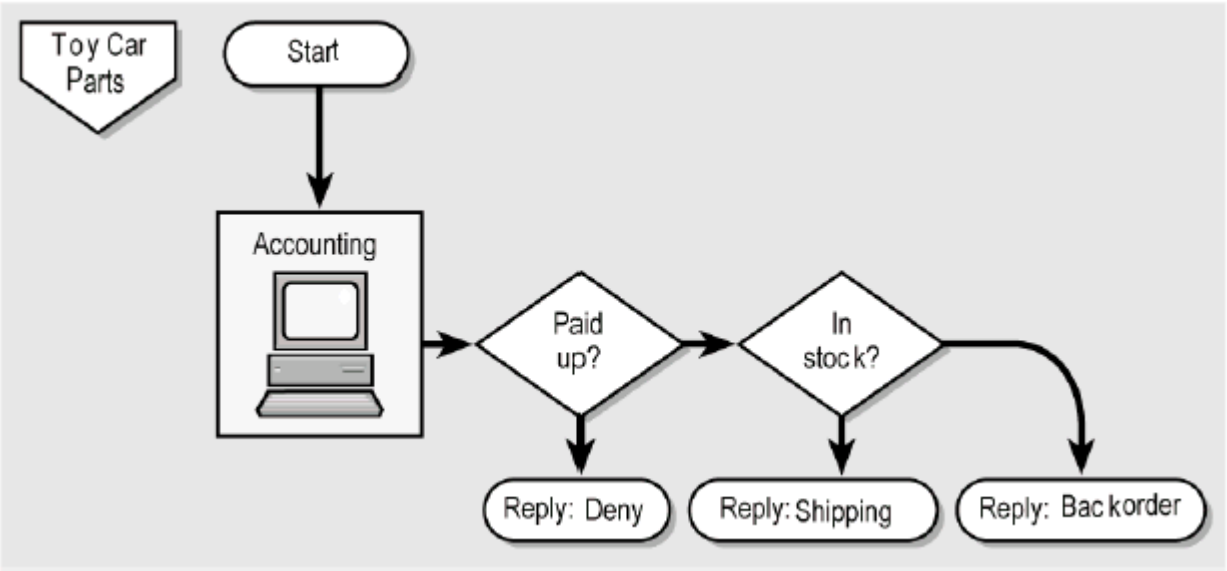


图 11-2　在 Toy Car Parts 中的数据流

当 Toy Car Parts 中的程序接收到 BizTalk 文件之后，就会剖析文件以取得必要的信息，其中包含采购订单的相关信息。客户的记录也会被质问以确认 Toi Carz 账户的信用良好－确认 Toy Car Parts 和 Toi Carz 目前有契约上的合作关系，并且 Toi Carz 已经支付款项了。

然后程序会检查该产品是否在库存中，如果该项产品并没有在库存中，一份回复文件会传送到 Toi Carz 以通知该项产品必须延期交货。如果在库存中有该项产品，则在订单数据库中就会加入这笔记录，并且在库存量方面必须适当的调整。最后，一份 BizTalk 文件会传送回 Toi Carz，其中包含采购订单的确认信息。

再让我们看看图 11-1，由 Toy Car Parts 建立的 BizTalk 回复文件被传送至 Toi Carz，该 BizTalk 文件是由 ASP 应用程序接收，它会撷取适当的信息并且执行两件工作以完成请求的处理：它会更新其状态是处于 未解决 的采购订单数据库，并且指出该采购订单已经被供货商接收和确认；它会通知初始化这个工作流程的每一个相关人员。

基础结构

Toi Carz 和 Toy Car Parts 使用数个程序完成采购请求的处理，表 11-1 和表 11-2 提供了程序的描述和它们的目的为何，在随书光盘中也包含了两家公司的所有程序（分别放在\Samples\Ch11\ToiCarz 和 \Samples\Ch11\ToyCarParts）中。

程序名称	程序类型	说明
prEnter.asp	ASP	该程序会建立采购请求、连接至员工和供货商的数据库来显示所有产品项目的清单以供订购，这些资料供 prProcess.asp 处理。
prProcess.asp	ASP	该程序会处理采购请求，首先检查员工是否有足够的职权订购该产品，并且确认订购的产品项目在库存中是否已经没有了。
poGen.exe	VB	采购订单产生程序会读取包含采购请求的数据库，并且会产生要传送

		至供货商的采购订单。而供货商的其中之一就是 Toy Car Parts，这间公司将会接收采购订单。poGen.exe 程序使用 ROPE 以封装和传送 SOAP 文件至 SOAP 伺服程序。
PoConfirm.asp	ASP	该程序接收包含采购订单确认的 BizTalk 文件，它会更新采购订单数据库，指出采购订单已经处理过了，然后通知采购请求的发起人。

表 11-1　用于采购请求处理的 Toi Carz 程序

程序名称	程序类型	说明
PlaceOrder.htm	HTML page	Toy Car Parts 有一个现存的旧型应用程序，它专门用来处理内部订单的输入。当接收采购订单后，会由一个职员使用 PlaceOrder.htm1 窗体输入信息，这份 HTML 文件是用于撷取订单信息。
PlaceOrder.xom	OmniMark	该程序为一 CGI 指令码，藉由使用在 PlaceOrder.htm 中 **FORM** 元素的 **ACTION** 方法呼叫。PlaceOrder.xom 包含确认订单遵守内部的查核和程序的商业逻辑，在通过确认后，程序会将订单插入订单数据库。Toy Car Parts 并不想重新撰写它们的系统，因为该系统已经正常工作数年之久了，所以该公司的 BizTalk 伺服程序必须能够使用现存的系统。

BTServer.xom	OmniMark	该程序为一 CGI 指令码，首先扮演为一个 SOAP 伺服程序，用于接收来自 poGen.exe 的 ROPE 接口传送的 SOAP 文件，然后该程序会转变成 BizTalk 伺服程序，转换 XML 的请求为 PlaceOrder.htm 文件。BTServer.xom 会接收来自于 PlaceOrder.xom 的响应，并且建立 BizTalk 确认讯息，该讯息会以 POST 方式直接传送至 Toi Carz 的 poConfirm.asp。该程序会处理采购订单确认并且结束循环。

表 11-2　用于采购完成程序的 Toy Car Parts 程序

完整的请求处理过程如图 11-3。

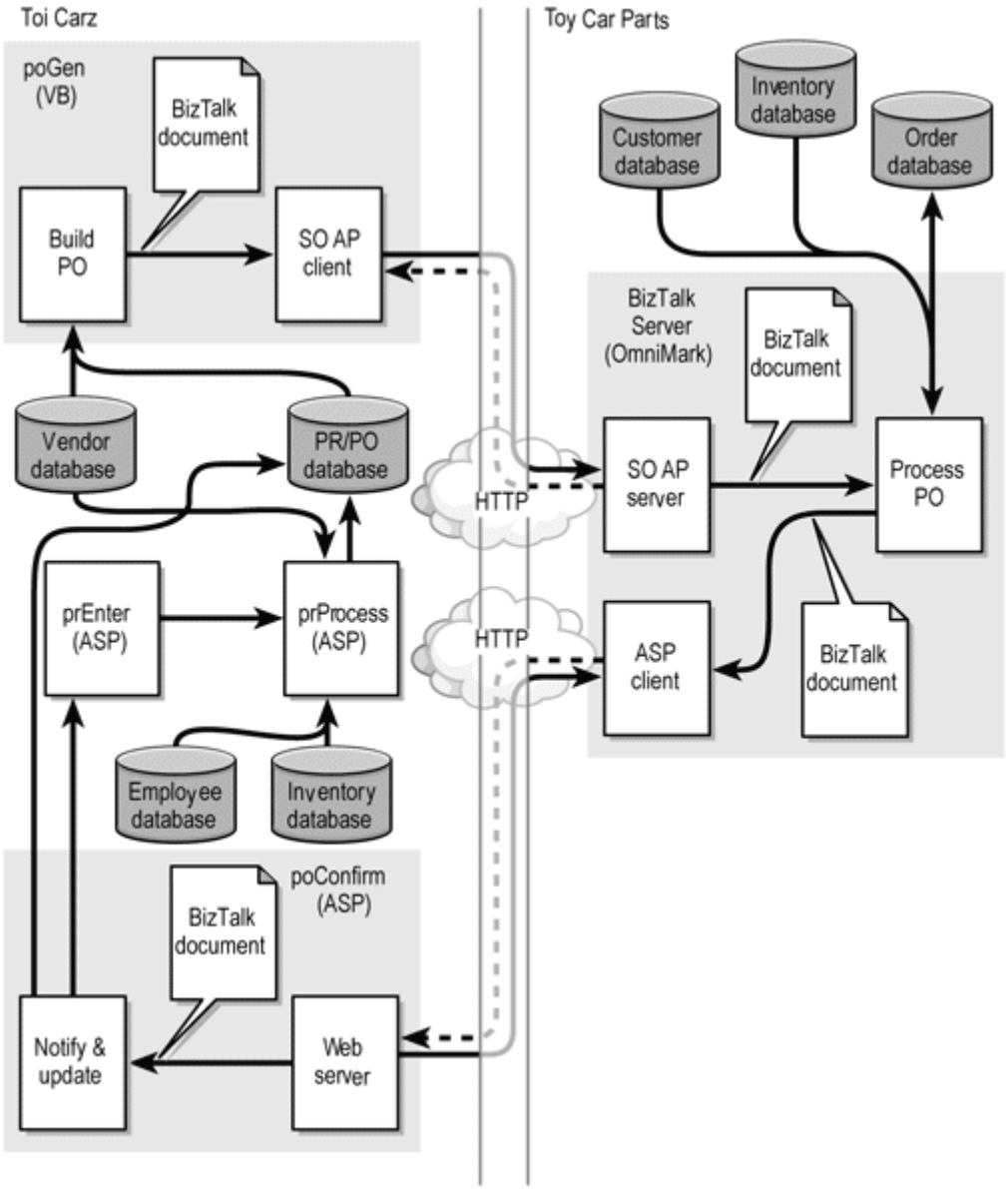


图 11-3　我们将在本章中建立的 BizTalk 伺服程序

数据库

每一家公司都使用数个数据库维持追踪信息，例如库存量、客户和订单的状态。在表 11-3 和表 11-4 中说明这些数据库，想要在本章描述的 BizTalk

解决方案中使用这些数据库，您将需要遵循本章稍后〈系统安装〉一节中的指示。这些数据库都是 Microsoft 数据库，因此您可以开启这些数据库以检视其数据结构和关联性。

数据库	说明
Employees.mdb	包含所有员工，以及他们的预算授权数据库。
Inventory.mdb	主要的库存量数据库，其中包含 Toi Carz 目前在库存中的所有相关产品项目信息。
Vendors.mdb	Toi Carz 所有供货商的所有产品项目清单，当 Toi Carz 从现存的供货商或新签约的供货商处取得新的目录时，该数据库就会更新。
PurchaseOrders.mdb	有两个数据表的数据库：PORequests，其中包含 Toi Carz 员工传送的采购请求清单；POSent，其中包含已经产生并且被传送至供货商的所有采购订单信息。

表 11-3　Toi Carz 数据库

数据库	说明
Customers.mdb	所有的客户和他们的账户状态清单，Toy Car Parts 将使用这些信息以判断在将产品传送给客户之前，该客户是否已经付款。

Inventory.mdb	主要的库存量数据库，包含在库存中所有的产品项目、数量和价格。
Orders.mdb	该数据库包含储存在 PlaceOrder 系统的订单，这个数据库整合 Toy Car Parts 系统并且将订单转变为实际出货，以及掌控正确的帐务。

表 11-4　Toy Car Parts 数据库

说明

PurchaseOrders.mdb 数据库包含两个数据表连结 Vendors.mdb 数据库，这些数据表名称为 catalog 和 vendors，如果您将数据库安装于预设的路径（C:\TravisBook\Ch11\ToiCarz\）以外，您将需要在 Access 中更新这些连结。

在本章范例中的采购请求系统在不同的供货商之间使用 ODBC 连接数据库，在表 11-3 和表 11-4 中的所有数据库都是 Access 数据库档案，它们也可以很容易的转换为 Microsoft SQL Server 或 Oracle 数据库，当然变更数据库档案为 SQL 或 Oracle 后，您可能会需要更改一些程序代码。

系统安装

要使采购请求系统可以实际执行，请遵循下列步骤：

- 执行随书光盘中的安装程序将范例档案从光盘拷贝到您的硬盘中，当然您也可以依照您的需求将光盘中 **\Samples\Ch11** 的档案拷贝至您硬盘中适当的位置。如果您使用随书光盘的安装程序，这些档案的预设位置是在 **C:\TravisBook\Ch11** 。
- 在 Microsoft Windows 2000 中建立 ODBC 资料来源名称（DSN）以存取数据库，您可以按两下 **开始/设定/控制台/系统管理工具** 中的 **数据来源 (ODBC)** 图标以开启 **ODBC 数据来源管理员** 对话框。如果您使用其它版本的 Windows，您可以在 **控制台** 中按两下 **数据来源(ODBC)** 图示。
- 您将需要建立 7 个系统 DSN 项目，列于表 11-5。按一下 **系统数据来源名称** 卷标页，再按一下 **新增** ，选取 **Microsoft Access Driver (*.mdb)** ，然后按一下 **完成** 。遵照表 11-5 中的指示输入 **数据来源名** 称，按一下 **选取** 按钮并找到该档案（这些档案已经在步骤 1 中拷贝到您的硬盘中了），按一下 **确定** ，重复此步骤直到完成表 11-5 的每一个数据来源名称为止。
- 建立两个 **Web 共享** ，使用 **Windows 档案总** 管 找到 **C:\TravisBook\Ch11** 数据夹，您将会看见两间公司的目录，在 **ToiCarz** 上按一下鼠标右键，并按一下 **共享** ，选取 **Web 共享** 标签页，按一下 **共享这个数据夹** 并接受在 **编辑别**

名 对话框中的默认值。针对 **ToyCarParts** 目录重复这个步骤，唯一的不同是要按一下 **执行(包含指令)** 选项按钮，这将会允许在 **ToyCarParts** 目录中执行 OmniMark 程序。

原始资料名称	位置
dbTC-Employees	C:\TravisBook\Ch11\ToiCarz\Employees.mdb
dbTC-Inventory	C:\TravisBook\Ch11\ToiCarz\Inventory.mdb
dbTC-PurchaseOrders	C:\TravisBook\Ch11\ToiCarz\PruchaseOrders.mdb
dbTC-Vendors	C:\TravisBook\Ch11\ToiCarz\Vendors.mdb
dbTCP-Customers	C:\TravisBook\Ch11\ToyCarParts\Customers.mdb
dbTCP-Inventory	C:\TravisBook\Ch11\ToyCarParts\Inventory.mdb
dbTCP-Orders	C:\TravisBook\Ch11\ToyCarParts\Orders.mdb

表 11-5　建立在步骤 3 中的 DSN 项目信息

输入采购请求

有两支程序用于输入采购请求：prEnter.asp 和 prProcess.asp，下节会说明这些程序。

prEnter.asp

prEnter.asp 程序是呈现 Vendors 数据库的 ASP 网页，其中包含所有承认供货商的目录，图 11-4 显示了该网页。

这个程序会读取 Employees 和 Vendors 数据库，并可以在下拉式清单方块中看见适当的信息。当选取一个供货商时，ASP 网页会呼叫自己本身以更新 Item 下拉式清单方块中的产品项目，在清单 11-1 中显示了该处理程序。

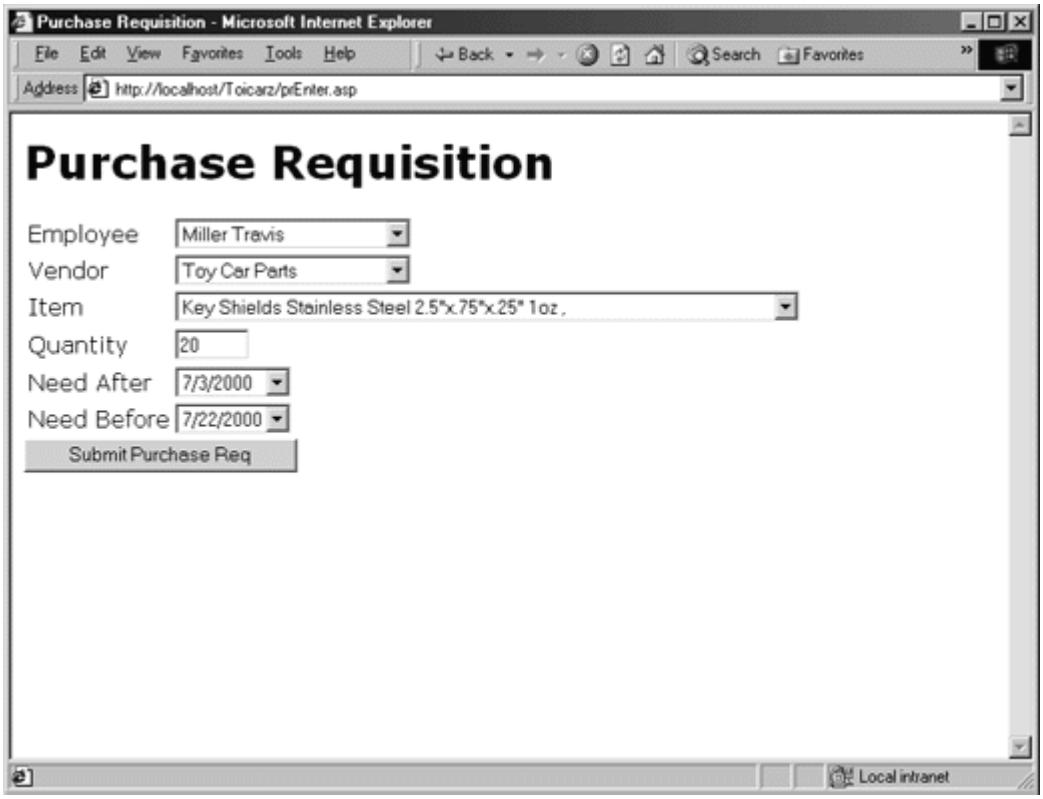


图 11-4 采购请求网页－prEnter.asp，允许员工开始产品订购程序

```
prEnter.asp

<%@language=vbscript%>

<%

Set oVendorConn =Server.CreateObject("ADODB.Connection ")

oVendorConn.Open("dbTC-Vendors ")



Function dateRange(fieldName,startDate,daysBackward,daysForward)

    If Request.Form(fieldName)="" Then

       selectDate =FormatDateTime(startDate,vbShortDate)
```

```
    Else
       selectDate =Request.Form(fieldName)
    End If
    dateRange =vbCrLf &"<SELECT NAME='" &&fieldName &"'>"
    For i =-daysBackward To daysForward
       thisDate =FormatDateTime(startDate +i,vbShortDate)
       dateRange =dateRange &vbCrLf &vbCrLf &"<OPTION "
       If thisDate =selectDate Then dateRange =dateRange &_
        "SELECTED"
       dateRange =dateRange &" VALUE='" &&thisDate &"'>" __
          &thisDate &"</OPTION>"
    Next
    dateRange =dateRange &vbCrLf &"</SELECT>"
End Function
%>
<HTML>
    <HEAD>
        <TITLE>Purchase Requisition</TITLE>
        <SCRIPT>
        function loadItems()
```

```
            {
            document.all.vendorForm.action="prEnter.asp"
            document.all.vendorForm.submit()
            }
        </SCRIPT>
    </HEAD>
    <BODY STYLE="font-family:Verdana;">
    <H1>Purchase Requisition</H1>
    <FORM METHOD="POST " id="vendorForm " ACTION="prProcess.asp">
        <TABLE>
            <TR>
                <TD>Employee</TD>
                <TD>
                <SELECT NAME="requestorID">
                <OPTION VALUE="0">--Select an Employee
--</OPTION>"
                <%
                Set oConn =Server.CreateObject("ADODB.Connection
")
                oConn.Open("dbTC-Employees ")
```

```
Set rsEmployee =oConn.Execute(_
    "SELECT *FROM Employees ORDER BY LName ")
Do Until rsEmployee.EOF
    Response.Write ("<OPTION ")
    If Int(Request.Form("requestorID "))=_
        Int(rsEmployee("empID "))Then
        Response.Write (" SELECTED ")
    End If
    Response.Write (" VALUE='" __
        &rsEmployee("empID ")&"'>"_
        &rsEmployee("FName ")&" " __
        &rsEmployee("LName ")_
        &"</OPTION>")
    rsEmployee.MoveNext
    Loop
    Set rsEmployee =Nothing
    oConn.Close
    Set oConn =Nothing
    %>
    </SELECT>
```

```
        </TD>
    </TR>
    <TR>
        <TD>Vendor</TD>
        <TD>
        <SELECT ID="vendorID " NAME="vendorID "
            ONCHANGE="javascript:loadItems();">
        <OPTION VALUE="0">--Select a Vendor --</
        OPTION>
        <%
        Set rsVendor =oVendorConn.Execute(_
        "SELECT *FROM Vendors ORDER BY vendorName ")
        Do Until rsVendor.EOF
            Response.Write ("<OPTION ")
            If Int(Request.Form("vendorID "))=_
               Int(rsVendor("vendorID "))Then
               Response.Write (" SELECTED ")
            End If
               Response.Write (" VALUE='" __
               &rsVendor("vendorID ")&"'>" __
```

```
                    &rsVendor("vendorName ")&"</OPTION>")
                rsVendor.MoveNext
        Loop
        Set rsVendor =Nothing
        %>
        </SELECT>
        </TD>
    </TR>
    <TR>
        <TD>Item</TD>
        <TD>
        <SELECT NAME="itemNum">
        <OPTION VALUE="0">--Select an Item --</OPTION>
    <%
    If Request.Form("vendorID ")<>0 Then
        Set rsItem =oVendorConn.Execute(_
             "SELECT *FROM catalog WHERE vendorID=" __
            &Request.Form("vendorID ")_
            &" ORDER BY description ")
            Do Until rsItem.EOF
```

```
                    Response.Write(vbCrLf &"<OPTION ")
                   If Request.Form("itemNum ")=rsItem("itemNum
                  ")Then
                        Response.Write(" SELECTED ")
                    End If
                    Response.Write(" VALUE=='" __
                        &rsItem("itemNum ")&"'>" __
                        &rsItem("description ")_
                        &"," &&rsItem("color ")&"</OPTION>")
                    rsItem.MoveNext
                Loop
                set rsItem =Nothing
            End If
            %>
            </SELECT>
            </TD>
        </TR>
        <TR>
            <TD>Quantity</TD>
            <TD><INPUT NAME="itemQty " VALUE="1 " SIZE="5">
```

```
                    </INPUT></TD>
            </TR>
            <TR>
                <TD>Need After</TD>
                <TD><%=dateRange("needAfter",Now(),0,30)%></TD>
            </TR>
            <TR>
                <TD>Need Before</TD>

<TD><%=dateRange("needBefore",Now()+21,20,30)%></TD>
            </TR>
        </TABLE>
        <INPUT TYPE="SUBMIT " VALUE="Submit PurchaseReq"></INPUT>
    </FORM>
    </BODY>
</HTML>
<%
oVendorConn.Close
Set oVendorConn =Nothing
%>
```

清单 11-1　输入采购请求的 prEnter.asp 程序

在 Vendor 的下拉式方块中有 ONCHANGE 事件，变更这个控件的值会执行 Jscript 函数 loadItems ，它会变更窗体的 ACTION 方法，引起网页再次加载自己一次。当网页呼叫它自己时，网页会从选取的供货商加载所有的产品项目到 Item 下拉式方块中。

当按一下 Submit Purchase Req 按钮后，prEnter.asp 会呼叫 prProcess.asp 确认该请求。

译者注

如果您发现有无法更新数据库的错误讯息，可能是数据库被设定为只读状态，请选取适当的数据库，并且将其只读状态取消。

---

prProcess.asp

prProcess.asp 程序包含确认请求者是否被授权采购该产品的商业逻辑，该程序也执行其它检查，包含确定库存中是否还有该产品项目。图 11-5 显示了确认画面。

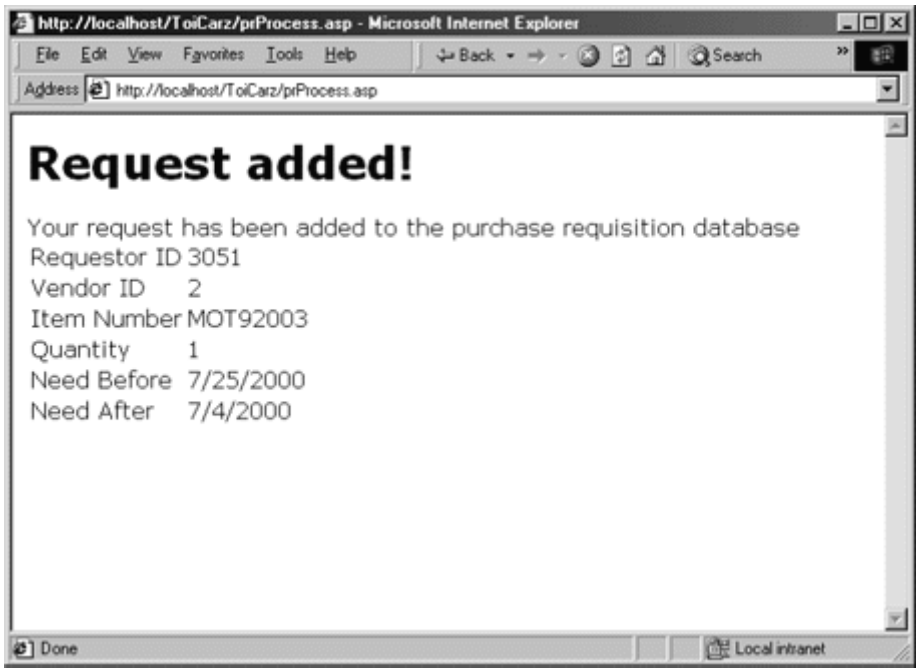


图 11-5　ASP 网页确认该采购请求已经被新增至采购请求数据库

清单 11-2 显示了 prProcess.asp 以确认采购请求是否正确有效。

```
prProcess.asp

<%@LANGUAGE ="VBScript " %%>

<HTML><BODY STYLE="font-family:Verdana">
```

```
<%
' Open Employee database
Set oEmpConn =Server.CreateObject("ADODB.Connection ")
oEmpConn.Open("dbTC-Employees ")

' Open Vendor database
Set oVendorsConn =Server.CreateObject("ADODB.Connection ")
oVendorsConn.Open("dbTC-Vendors ")
' Open PO database
Set oPOConn =Server.CreateObject("ADODB.Connection ")
oPOConn.Open("dbTC-PurchaseOrders ")
' Open Inventory database
Set oInventoryConn =Server.CreateObject("ADODB.Connection ")
oInventoryConn.Open("dbTC-Inventory ")

reqError =""
' Do some error checking
Set itemRS =oVendorsConn.Execute("SELECT price FROM Catalog "
    &"WHERE itemNum='" __
    &Request.Form("itemNum ")_
```

```
    &"'")
If itemRS.EOF Then
    reqError =reqError _
        &"<BR>Cannot find item " __
        &Request.Form("itemNum ")
End If
Set employeeRS =oEmpConn.Execute("SELECT purchaseLimit " __
    &"FROM Employees " __
    &"WHERE empID=" &&Request.Form("requestorID "))
If employeeRS.EOF Then
    reqError =reqError _
        &"<BR>Cannot find employee " __
        &Request.Form("requestorID ")
End If
' Check to see if our requestor has purchasing authority
qty =Request.Form("itemQty ")
price =itemRS("price ")
totalPrice =qty *price


If CDbl(totalPrice)>CDbl(employeeRS("purchaseLimit "))Then
```

```
    reqError =reqError _
        &"<BR>Your purchase authority of $" __
        &employeeRS("purchaseLimit ")_
        &" is less than your request total of $$" __
        &totalPrice _
        &".Try again."
End If
' Check to see if the item is already in stock
Set inventoryRS =oInventoryConn.Execute(_
     "SELECT *FROM inventory " __
    &"WHERE vendorID=" &&Request.Form("vendorID ")_
    &" AND itemNum=='" &&Request.Form("itemNum ")&"'")
qtyRequested =Request.Form("itemQty ")
If Not inventoryRS.EOF Then
    qtyOnHand =inventoryRS("quantity ")
    If CDbl(qtyOnHand)>=CDbl(qtyRequested)Then
        reqError =reqError _
            &"<BR>Inventory reports " __
            &qtyOnHand _
            &" units of item " __
```

```
            &Request.Form("itemNum ")_
            &" already in stock..Requisition denied."
    Else
        reqMessage =reqMessage _
            &"<BR>Inventory reports " __
            &qtyOnHand _
            &" units of item " __
            &Request.Form("itemNum ")_
            &" already in stock..Order quantity reduced to " __
            &qtyRequested -qtyOnHand _
            &" units.."
        qtyRequested =qtyRequested -qtyOnHand
    End If
End If
If reqError <>"" Then
    Response.Write("<H1>Error</H1>")
    Response.Write(reqError)
Else
    Response.Write(reqMessage)
    Set rsOrder =Server.CreateObject("ADODB.Recordset ")
```

```
rsOrder.CursorType =2

rsOrder.LockType =2



rsOrder.Open "PORequests",oPOConn

rsOrder.AddNew

rsOrder("timeStamp ")=Now()

rsOrder("requestorID ")=Request.Form("requestorID ")

rsOrder("vendorID ")=Request.Form("vendorID ")

rsOrder("itemNum ")=Request.Form("itemNum ")

rsOrder("itemQty ")=qtyRequested

rsOrder("needBefore ")=Request.Form("needBefore ")

rsOrder("needAfter ")=Request.Form("needAfter ")

rsOrder("itemPrice ")=price

rsOrder.Update



rsOrder.Close

Set rsOrder =Nothing



oPOConn.close

Set oPOConn =Nothing
```

```
    Response.Write("<H1>Request added!</H1>")
    Response.Write("<P>Your request has been added to the " __
        &"purchase requisition database ")
    Response.Write("<TABLE>")
    Response.Write("<TR><TD>Requestor ID</TD><TD>" __
        &Request.Form("requestorID ")&"</TD></TR>")
    Response.Write("<TR><TD>Vendor ID</TD><TD>" __
        &Request.Form("vendorID ")&"</TD></TR>")
    Response.Write("<TR><TD>Item Number</TD><TD>" __
        &Request.Form("itemNum ")&"</TD></TR>")
    Response.Write("<TR><TD>Quantity</TD><TD>" __
        &qtyRequested &"</TD></TR>")
    Response.Write("<TR><TD>Need Before</TD><TD>" __
        &Request.Form("needBefore ")&"</TD></TR>")
    Response.Write("<TR><TD>Need After</TD><TD>" __
        &Request.Form("needAfter ")&"</TD></TR>")
End If
%>
</BODY>
</HTML>
```

清单 11-2　该程序确认采购请求是否正确有效

ROPE 和 SDL

我们将在本节中使用 Visual Basic 撰写的程序是 poGen.exe，poGen 程序使用了 Microsoft 的新技术称为 ROPE（Remote Object Proxy Engine），在 第八章 ，我们使用了困难的方式建立 SOAP 客户端－我们连结字符串以形成 SOAP 信封，然后再传送该信封给服务器，服务器会读取这个封装和传送响应,连结字符串可以帮助我们了解 SOAP 封包和学习它们是如何工作的，但是连结字符串对使用者撰写程序而言并不具亲和力－首先，程序设计师必须了解 SOAP 语法以建立远程过程调用，接着他们也必须了解 XML 语法以便建立可供剖析器使用的文件。

那么如果在您在撰写程序与建立 SOAP 文件之间有现成的服务可以使用呢？这个服务会将您的服务器请求解译成 XML 程序代码，然后传回预期中的数据。换句话说，有一个代理程序可以帮助您达成此目的－ROPE，其由 SOAP Toolkit for Visual Studio 6.0 中的共享 COM 物件（rope.dll）组成，随书光盘中包含这些档案，图 11-6 显示了 ROPE 如何工作。

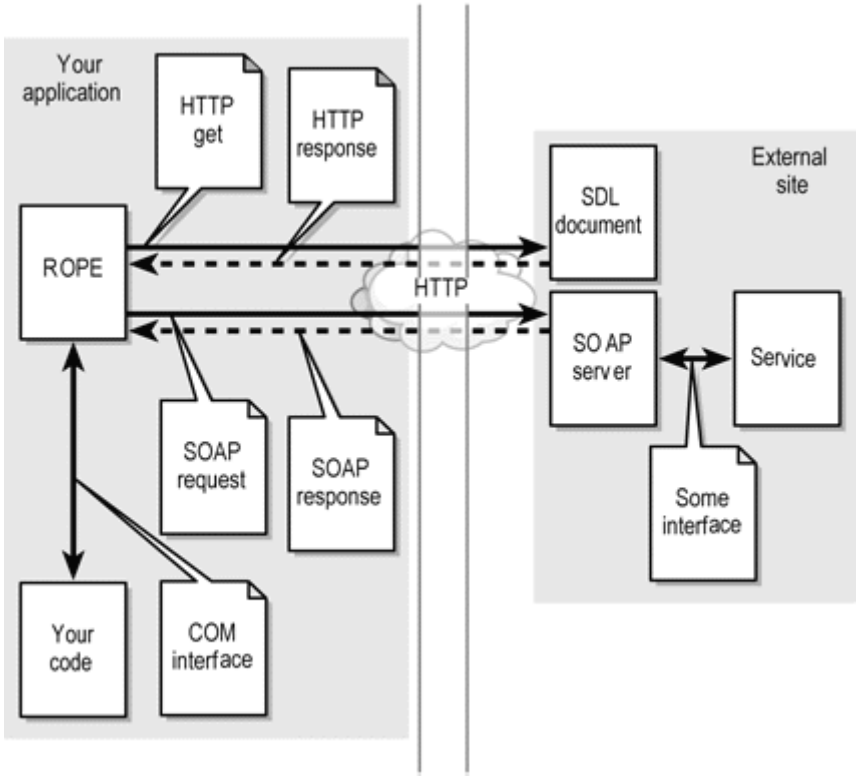


图 11-6　ROPE 将请求解译成 SOAP 文件，并且和远程对象通讯

ROPE 是藉由 Service Description Language（SDL），并依照 XML 档案的标签存在情况工作，这些标签档案会描述所有可用的服务以呼叫应用程序。您可以使用 SOAP Toolkit 中的 SOAP Toolkit Wizard 建立 SDL 文件，该精灵会读取 COM 对象并建立适当的 SDL，然而当您要应付例如清单 11-3 中的非 COM 应用程序时，您也可以自己撰写 SDL 文件。SDL 规格包含在 SOAP Toolkit for Visual Studio 6.0 中，清单 11-3 显示了我们使用的 SDL 文件范例。

```
Services.xml<?xml version="1.0 " ??>
<serviceDescription
```

```
name="SoapService"
xmlns="http://localhost/soapdemo/sdlschema.xml"
xmlns:dt="http://www.w3.org/1999/XMLSchema"
xmlns:ss ="#ServiceSchema">

<import namespace="#ServiceSchema " location ="#
ServiceSchema"/>
<soap xmlns="urn:schemas-xmlsoap-org:soap-sdl-2000-01-25">
    <interface name="BizTalkInterface"/>
    <service>
        <implements name="BizTalkInterface " //>
        <!--Send a BizTalk message for processing -->
            <requestResponse name="BizTalkMessage">
            <request ref="ss:BizTalkMessageRequest"/>
            <response ref="ss:BizTalkMessageResponse"/>
            <parameterorder>BizTalkDoc</parameterorder>
        </requestResponse>
        <!--Check Inventory -->
        <requestResponse name="checkInventory">
            <request ref="ss:checkInventoryRequest"/>
```

```
            <response ref="ss:checkInventoryResponse"/>
            <parameterorder>itemNum quantity</parameterorder>
        </requestResponse>
        <!--Check the server status ("are you OK?")-->
        <requestResponse name="RUOK">
            <request ref="ss:RUOKRequest"/>
            <response ref="ss:RUOKResponse"/>
        </requestResponse>
        <addresses>
        <location url="http://localhost/toycarparts/
        BTServer.xar"/>
        </addresses>
    </service>
</soap>
<ss:schema     id="ServiceSchema">
    <element name="BizTalkMessageRequest">
        <type>
            <element name="BizTalkDoc " type="dt:string"/>
        </type>
    </element>
```

```
<element name="BizTalkMessageResponse">
    <type>
        <element name="DeliveryReceipt " type=="dt:
        string"/>
    </type>
</element>
<element name="checkInventoryRequest">
    <type>
        <element name="itemNum " type="dt:string"/>
        <element name="quantity " type="dt:string"/>
    </type>
</element>
<element name="checkInventoryResponse">
    <type>
        <element name="itemPrice " type="dt:string"/>
        <element name="deliveryDate " type="dt:string"/>
    </type>
</element>
<element name="RUOKRequest">
    <type>
```

```
                <element name="Subsystem " type="dt:string"/>
            </type>
        </element>
        <element name="RUOKResponse">
            <type>
                <element name="Status " type="dt:string"/>
            </type>
        </element>
    </ss:schema>
</serviceDescription>
```

清单 11-3　SDL 定义了供 SOAP 和 ROPE 使用的 Web 服务

为了要提供 COM 接口供您的程序使用，ROPE 对象会读取 SDL 文件以便找到有哪些服务可以使用，然后这些服务会变成基于 SOAP 的 Web 服务。换句话说，对于 ROPE 程序设计师来说，整个世界就像 COM 对象一样，您将会在下一个范例看见 ROPE 接口在 Visual Basic sendMsg 函数中看起来像什么？

采购订单产生程序

采购订单产生程序－poGen 会读取 PurchaseOrders 数据库中的 PORequests 数据表、收集供货商的请求，并且以 Vendors 数据库中供货商的住址为基础建立数个订购单。这个程序可以每天执行，或者在有许多内部采购请求时每分钟执行，poGen 的使用者接口如图 11-7，在 poGen 程序的重点程序会在下一节中说明。

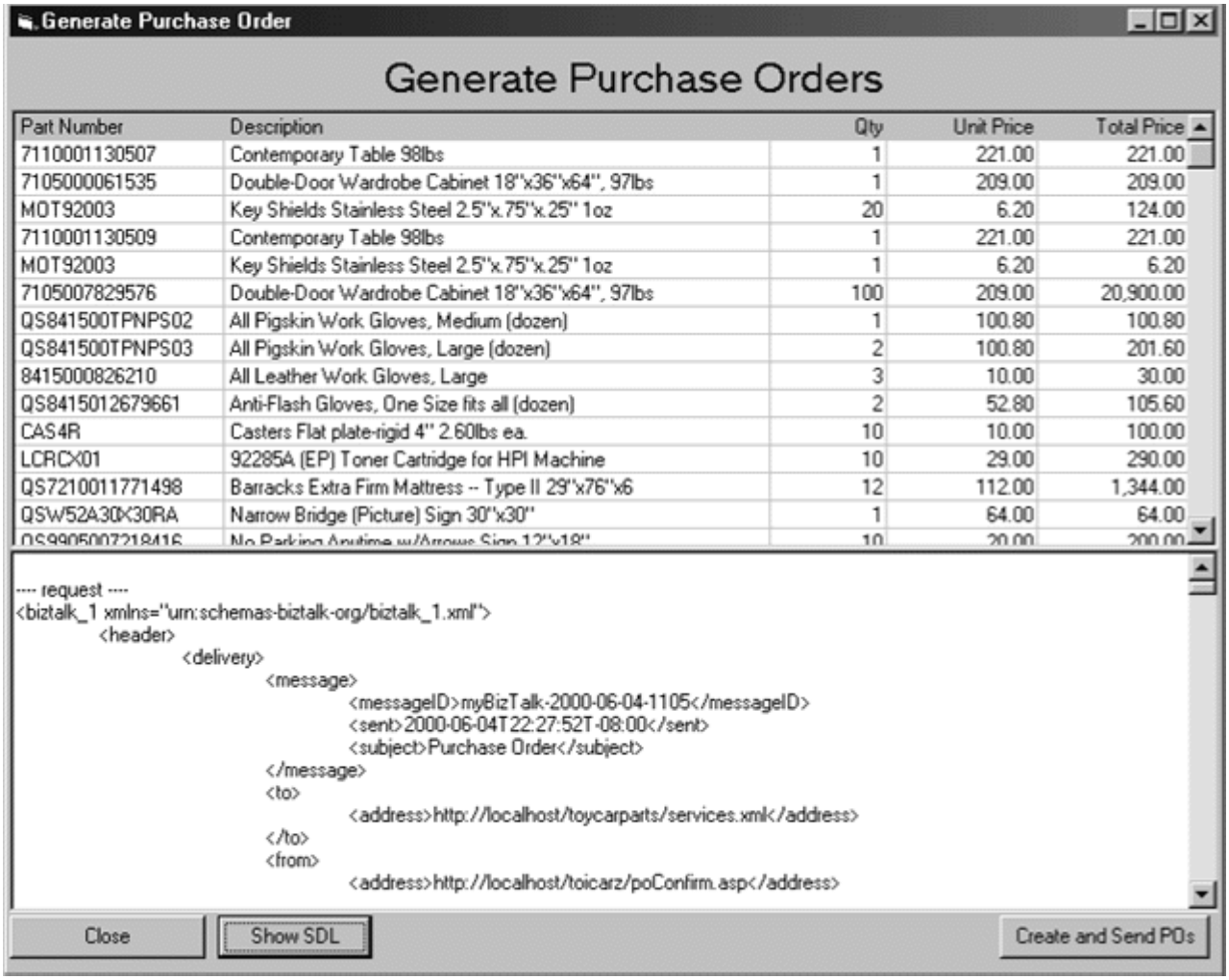


图 11-7　poGen 程序建立订购单，并且以 BizTalk 文件方式传送给供货商

Form.Load 子程序

Form.Load 子程序如清单 11-4 所示，从数据库中撷取所有的采购请求，并且将它们显示在窗口中。

```
Form.Load
Private Sub Form_Load()
    Randomize
        strSQL ="SELECT PORequests.*,catalog.*,vendors.*" __
        &"FROM vendors INNER JOIN (PORequests " __
        &"INNER JOIN [catalog ] ON [PORequests ].[itemNum ]=" __
        &"[catalog ].[itemNum ])ON ([PORequests ].[vendorID ]=" __
        &"[vendors ].[vendorID ])AND ([vendors ].[vendorID ]=" __
        &"[catalog ].[vendorID ])ORDER BY [vendors ].[vendorID ]"
    Set odbPO =New ADODB.Connection
    odbPO.Open ("dbTC-PurchaseOrders ")

    Set orsPO =odbPO.Execute(strSQL)
```

```
    Do Until orsPO.EOF
            gridPO.AddItem orsPO("itemNum ")_
            &vbTab &orsPO("description ")_
            &vbTab &orsPO("itemQty ")_
            &vbTab &Format(orsPO("itemPrice "),_
                 "###,##0.00 ")_
            &vbTab &Format(orsPO("itemPrice ")_
                *orsPO("itemQty "),"###,##0.00 ")
        orsPO.MoveNext
    Loop
    orsPO.Close
    Set orsPO =Nothing
End Sub
```

清单 11-4　poGen 程序的 Form_Load 子程序会显示采购请求

poGen 程序使用 MSFlexGrid 控件以显示信息字段，藉由查询 PurchaseOrder 数据库的数据表，并且将结果显示于 MSFlexGrid 控件。

createPO_Click 子程序

当使用者按一下 Create and Send POs 按钮时，createPO_Click 子程序会收集特定供货商的所有采购请求，并且建立该供货商的单一采购订单。然后往下移动到另一个供货商，直到所有的采购请求都传送出去为止，createPO_Click 子程序如清单 11-5 所示。

```
createPO_Click
Private Sub createPO_Click()
    Dim txtBizTalk As String
    Dim txtSOAP As String
    Dim txtXML As String
    Dim oXMLDoc As New msxml2.DOMDocument
    Dim oParmsDoc As New msxml2.DOMDocument
    Dim orsRequisitions As New ADODB.Recordset
    Dim orsPOSent As New ADODB.Recordset
    Dim strSQL As String
    Dim currentDate As Date
    Dim strSendTo As String
    Dim strReplyTo As String
    Dim currentVendor As Integer
    Dim poNum
```

```
    Dim strParms As String



    txtResponse.Text =""

    currentDate =Now()

    currentVendor =0

    strSQL ="SELECT PORequests.*,vendors.*,PORequests.vendorID "

__

        &"FROM vendors INNER JOIN " __

        &"PORequests ON vendors.vendorID =PORequests.vendorID " __

        &"ORDER BY PORequests.vendorID"

    orsRequisitions.CursorType =adOpenKeyset

    orsRequisitions.LockType =adLockOptimistic

    orsRequisitions.Open strSQL,odbPO,,,adCmdText



    orsPOSent.CursorType =adOpenKeyset

    orsPOSent.LockType =adLockOptimistic

    orsPOSent.Open "POSent",odbPO,,,adCmdTable



    Do Until orsRequisitions.EOF

        Screen.MousePointer =vbHourglass
```

```
        ’ Generate a new purchase order number
        orsPOSent.AddNew
        orsPOSent("poDate ")=currentDate
        orsPOSent("timeSent ")=Now()
        orsPOSent.Update

        poNum =orsPOSent("poNum ")
        ’ Generate the purchase order number
        orsPOSent("messageID ")="uuid:" &&genUUID()
        orsPOSent.Update
        txtXML ="<purchaseOrder_Chapter11_1.0 " __
            &vbCrLf &" xmlns
==’urn:purchaseOrder_Chapter12_1.0.xdr ’" __
            &vbCrLf &" custID ==’" &&CUST_ID &"’" __
            &vbCrLf &" poNumber ==’" &&poNum &"’" __
            &vbCrLf &" poDate ==’" &&iso8601(currentDate)&"’>"
        currentVendor =orsRequisitions("vendorID ")
nextItem:
        txtXML =txtXML &vbCrLf &" <<item " __
            &vbCrLf &" needAfter==’" __
```

```
            &iso8601(orsRequisitions("needAfter "))&"'" __
            &vbCrLf &" needBefore='" __
            &iso8601(orsRequisitions("needBefore "))&"'" __
            &vbCrLf &" reqID='" &&orsRequisitions("reqID ")&"'" __
            &">"
        txtXML =txtXML &vbCrLf &" <<number>" __
            &orsRequisitions("itemNum ")&"</number>"
        txtXML =txtXML &vbCrLf &" <<price>" __
            &orsRequisitions("itemPrice ")&"</price>"
        txtXML =txtXML &vbCrLf &" <<qty>" __
            &orsRequisitions("itemQty ")&"</qty>"
        txtXML =txtXML &vbCrLf &"</item>"
        strParms ="<?xml version='1.0 '?>" __
            &"<parms>" __
            &"   <toAddress type='agr:department '>" __
            &orsRequisitions("vendorToAddress ")&"</toAddress>"
__
            &"   <messageID>" &&orsPOSent("messageID ")_
            &"</messageID>" __
            &"   <sendTo>" &&orsRequisitions("vendorToAddress ")_
```

```
            &"</sendTo>" __
            &"   <confirmTo>"
&&orsRequisitions("vendorConfirmAddress ")_
            &"</confirmTo>" __
            &"</parms>"
        orsRequisitions("poNum ")=poNum
        orsRequisitions.Update
        orsRequisitions.MoveNext


        If orsRequisitions.EOF Then GoTo around
        If currentVendor =orsRequisitions("vendorID ")Then
            GoTo nextItem
        End If
around:
        txtXML =txtXML &"</purchaseOrder_Chapter11_1.0>"
        oXMLDoc.loadXML (txtXML)
        If oXMLDoc.parseError.errorCode <>0 Then
            MsgBox ("XML Error:" &&oXMLDoc.parseError.reason _
                &" on line " &&oXMLDoc.parseError.Line)
        End If
```

```
        oParmsDoc.loadXML (strParms)
        txtBizTalk =envelopeBizTalk(oXMLDoc,oParmsDoc)


        oXMLDoc.loadXML (txtBizTalk)
        If oXMLDoc.parseError.errorCode <>0 Then
            MsgBox ("XML Error:" &&oXMLDoc.parseError.reason _
                &" on line " &&oXMLDoc.parseError.Line)
        End If
        sendMsg oXMLDoc
    Loop
out:
    orsPOSent.Close
    Set orsPOSent =Nothing
    orsRequisitions.Close
    Set orsRequisitions =Nothing
    Screen.MousePointer =vbDefault
End Sub
```

清单 11-5　createPO_Click 子程序建构并且传送采购订单给供货商

该子程序会建立 PurchaseOrder_Chapter11_1.0 元素为 XML 字符串，信息将会放在 BizTalk 文件中传送至供货商。

sendMsg 函数

在 sendMsg 函数中可以和 ROPE 代理程序互动，该函数如清单 11-6 所示。

```
sendMsg
Function sendMsg(oXMLDoc As msxml2.DOMDocument)
    Dim retryCount
    Dim sendTo As String

     ' Discover service call
    sendTo =oXMLDoc.selectSingleNode("//dlv:to/dlv:address
").Text
    Dim oROPE As New ROPE.Proxy
    Dim oPackager As New ROPE.SOAPPackager

     ' Call ROPE proxy
    Dim bRetVal As Boolean
```

```
bRetVal =oROPE.LoadServicesDescription(icURI,sendTo)
If (bRetVal =False)Then
    MsgBox "LoadServicesDescription =" __
        &bRetVal,,"LoadServicesDescription FAILED"
    End
End If
Dim sBuffer As String
Dim sTransaction
sTransaction =vbCrLf &"----request ----" &&vbCrLf &oXMLDoc.xml
On Error GoTo badSOAP
sBuffer =oROPE.BizTalkMessage(oXMLDoc.xml)
sTransaction =sTransaction _
    &vbCrLf &"----response ----" &&vbCrLf &sBuffer
sTransaction =sTransaction _
    &vbCrLf &vbCrLf &"-----------------------"_
    &vbCrLf &vbCrLf
txtResponse.Text =sTransaction &txtResponse.Text
txtResponse.Refresh
On Error GoTo 0
Set oROPE =Nothing
```

```
    sendMsg =True
    Exit Function

badSOAP:
    Set oROPE =Nothing
    MsgBox ("SOAP server returned a fault ")
    On Error GoTo 0
    sendMsg =False
    Exit Function

End Function
```

清单 11-6 sendMsg 函数藉由读取远程站台的 SDL 档案建立至外部 Web 服务的联机，并且建立了 COM 接口

sendMsg 的主要部分是执行个体初始化和联机至 ROPE 引擎，请您看看下列的程序代码：

```
bRetVal = oROPE.LoadServicesDescription(icURI, senTo)
sBuffer = oROPE.BizTalkMessage(oXMLDoc.xml)
```

第一行藉由加载 SDL 文件建立至外部服务的联机，提供对象的延迟连结（late-binding）接口，延迟连结表示在设计时期对象的方法并无法使用，使用在执行时间对象才能够使用方法（您应该可以注意到 Microsoft IntelliSence 处理器并无法显示外部对象的方法）。

一旦 ROPE 的信息从 SDL 文件加载后，您就可以使用熟悉的 COM 接口呼叫方法来使用。如果您想要看看由 SDL 查询找到的方法，按一下 Show SDL 按钮。

处理采购订单

Toy Car Parts 有一套订单项目的旧型应用程序，其职员都使用该应用程序输入客户的采购订单，而 HTML 文件－PlaceOrder.htm 会呼叫以 OmniMark 撰写的 CGI 程序以处理订单，PlaceOrder.htm 的界面如图 11-8 所示。

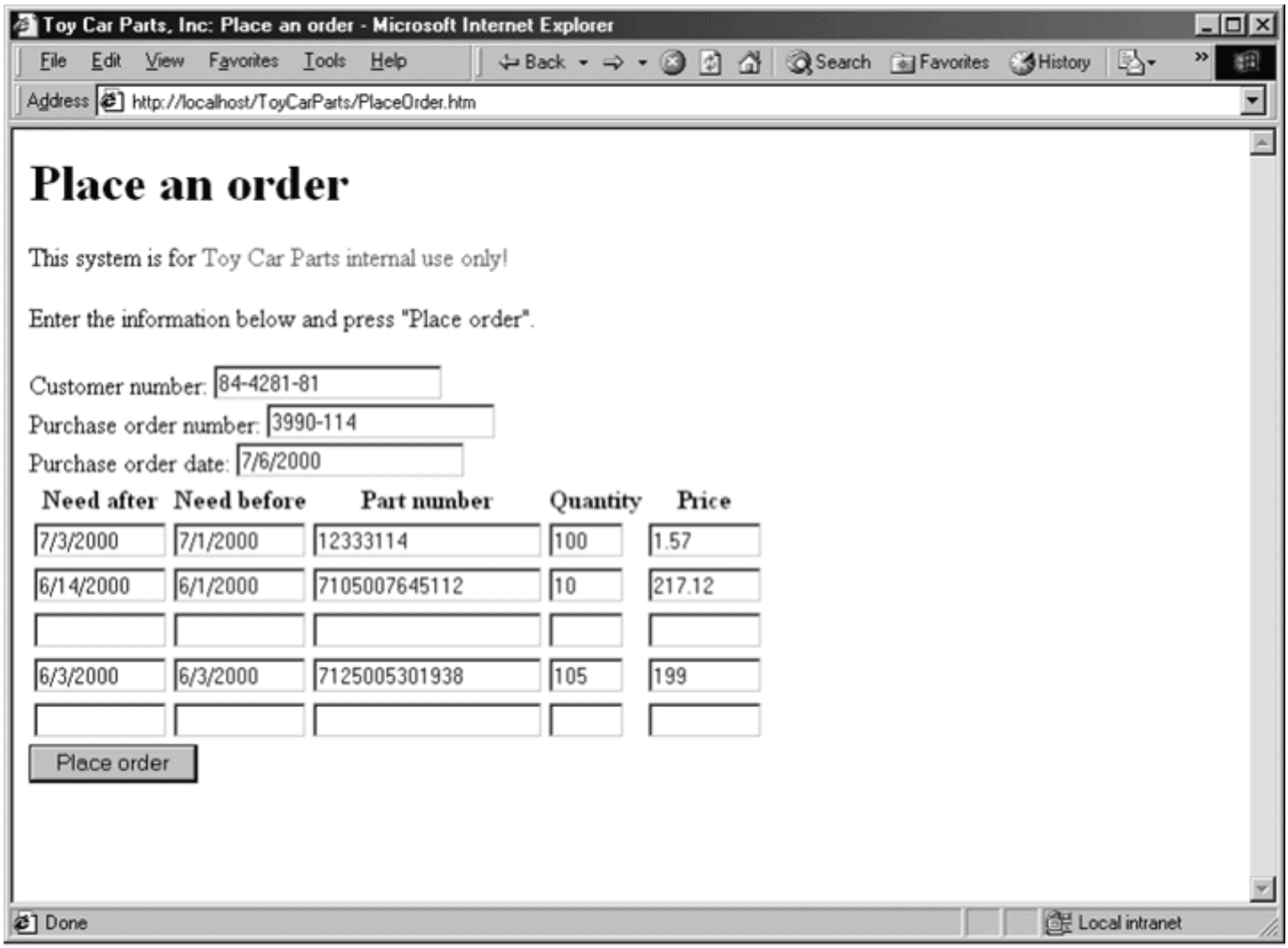


图 11-8　旧型的订单处理应用程序－PlaceOrder.htm

PlaceOrder.htm 是一个简单的 HTML 窗体，该窗体供职员输入采购订单的适当信息，而在 HTML 窗体中的数据会被传送至处理订单的 OmniMark 程序，该程序会套用适当的商业规则、更新适当的数据库，并且传送响应给使用者，响应画面如图 11-9。

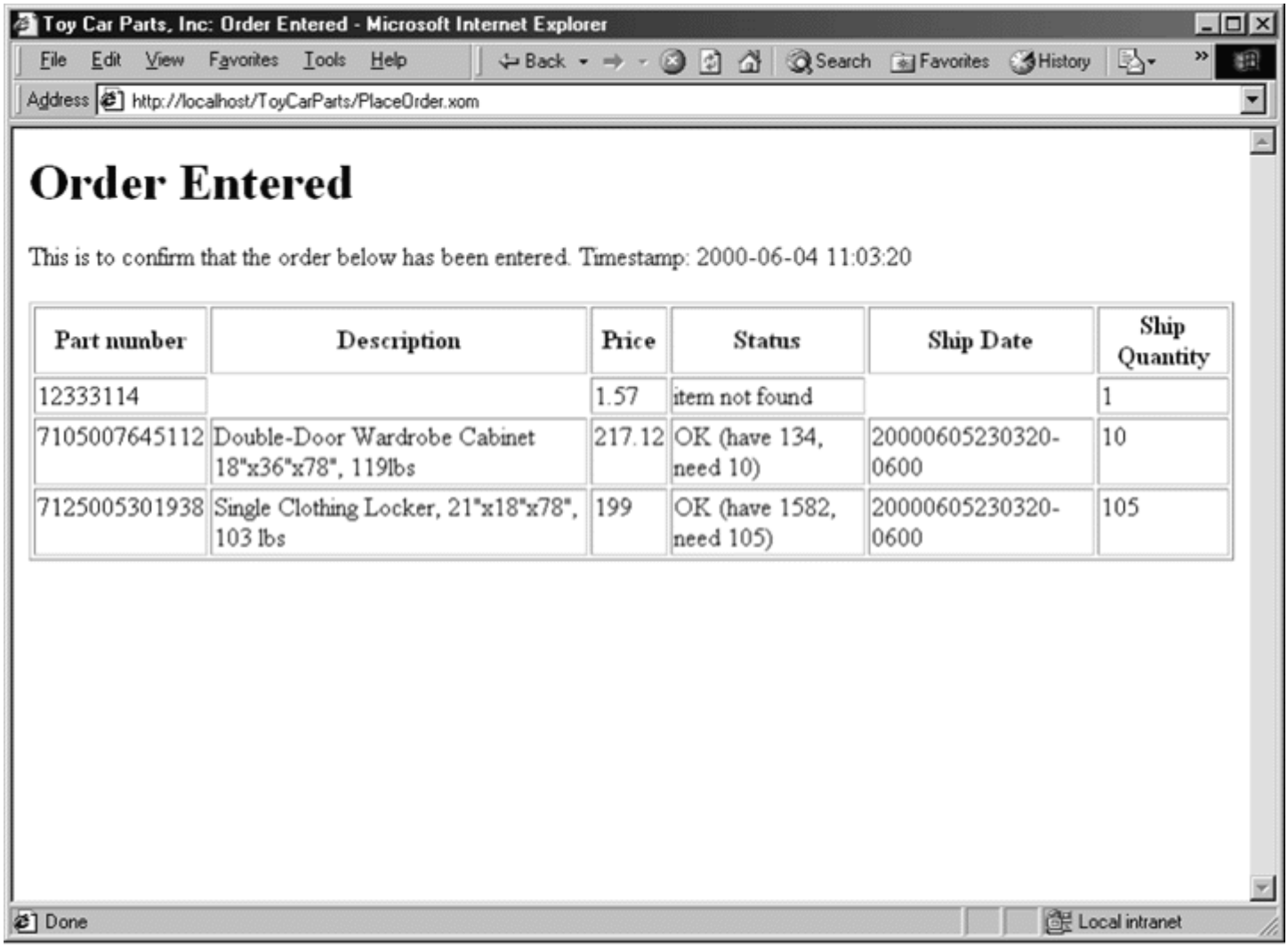


图 11-9　由 PlaceOrder.xom 产生的回应

BizTalk 自动化采购订单处理系统需要执行和旧型应用程序相同的工作，我们可以撰写和旧型系统处理规则一样的 BizTalk 文件处理应用程序，但是这样会浪费太多相同的工作，而且我们仍然想要使用手动输入的旧型应用

程序以应付纸上作业的采购订单。如果我们撰写了一个智能型的 BizTalk 应用程序，我们就可以在双方面都维护相同的商业规则（新旧系统都可以同时使用，保存既有的投资）。

现在我们就要进行这个工作，我们的 BizTalk 伺服程序建立的文件必须和 PlaceOrder.htm 产生的文件一样，并且还要将该文件张贴至 PlaceOrder.xom，这并不会很困难，较困难的地方只有在解译来自 HTML 的资料。我们将使用 BizTalk 伺服程序产生和解译 HTML 数据。

自动化采购订单处理

现在让我们重新回到处理流程，Toy Car Parts 接收到一个请求，该请求会呼叫 BizTalk 伺服程序对象的方法，这个方法是由 SDL 请求找到的－即为 BizTalkMessage ，它被扮演 BizTalk 伺服程序的 CGI 应用程序所处理，该程序由 OmniMark 撰写－BTServer.xom。

BizTalkMessage 需要单一的参数，即为 BizTalk 文件。在 poGen 程序中的 ROPE 对象建立了 SOAP 文件，同时也建立了由 SOAP 规格所定义的 HTTP 标头参数 SOAPAction ，该参数指定 SOAP 伺服程序将呼叫何种方法，OmniMark 使用 [UTIL_GetEnv 函数撷取该参数，如下所示：

```
set SOAPAction to UTIL_GetEnv("HTTP_SOAPACTION")
```

SOAPAction 参数以此形式表现：；

http://localhost/toycarparts/BTServer.xar#BizTalkMessage，我们需要孤立该动作和 URL， SOAPAction 变量使用清单 11-7 中的程序代码检查以确认它是否为支持的方法之一。

```
Checking SOAPAction
open SOAPResponse as buffer
do scan #main-input
  match any{formVars{"CONTENT_LENGTH"}}
        =>inputData
    put Debug "%ninputData:%n%x(inputData)%n "
    set SOAPAction to UTIL_GetEnv ("HTTP_SOAPACTION ")
    put Debug "%nSOAPAction (http):%g(SOAPAction)%n"
    ;http://localhost/toycarparts/BTServer.xar#BizTalkMessage
    do scan SOAPAction
        match [any except "#"]+"#" any**=>SOAPmethod
            set SOAPAction to SOAPMethod
    else
        set SOAPAction to ""
    done
```

```
    put Debug "%nSOAPAction (http):%g(SOAPAction)%n"



    do when SOAPAction =""

    put SOAPResponse "%n<SOAP-ENV:Fault>"

        || "%n  <faultcode>101</faultcode>"

        || "%n  <faultstring>no method specified"

        || "%n</faultstring>"

        || "%n</SOAP-ENV:Fault>"

    else

        do scan SOAPAction

            match "BizTalkMessage"

                using group processPO do

                    do xml-parse instance

                        scan inputData

                        suppress

                    done

                done

                do when errorMsg =""

                    put SOAPResponse "%n
<BizTalkMessageResponse>"
```

```
                        || "%n <DeliveryReceipt status='OK '>"
                        || "BizTalk Message received at "
                        || format-ymdhms
                             "=W,=xD =n =xY,=h:=m:=s =a.m.=t%n"
                             with-date now-as-ymdhms
                        || " From confirmation server::"
                        || escapeXML (fromPOConf)
                        || "</DeliveryReceipt>"
                        || "%n  </BizTalkMessageResponse>"
                else
                    put SOAPResponse "%n
<BizTalkMessageResponse>"
                        ||"%n  <DeliveryReceipt status='ERROR'>"
                        || "BizTalk Message Error:"
                        || errorMsg ||". Processed at "
                        || format-ymdhms
                             "=W,=xD =n =xY,=h:=m:=s =a.m.=t%n"
                            with-date now-as-ymdhms
                        || "</DeliveryReceipt>"
                        || "%n </BizTalkMessageResponse>"
```

```
            done
        match "checkInventory"
            put SOAPResponse "%n <checkInventoryResponse>"
                    || "%n    <DeliveryReceipt status='OK '>"
                    || "This method is not supported (2nd
                    edition...)."
                    || "Processed at "
                    || format-ymdhms
                        "=W,=xD =n =xY,=h:=m:=s =a.m.=t%n"
                        with-date now-as-ymdhms
                    || "</DeliveryReceipt>"
                    || "%n </checkInventoryResponse>"
        match "RUOK"
            put SOAPResponse "<RUOKResponse><status>"
                ||  "OK at "
                || format-ymdhms
                    "=W,=xD =n =xY,=h:=m:=s =a.m.=t%n"
                    with-date now-as-ymdhms
                || " </status></RUOKResponse>"
    else
```

```
            put SOAPResponse "%n<SOAP-ENV:Fault>"
                || "%n  <faultcode>100</faultcode>"
                || "%n  <faultstring>method not supported:"
                || "%n  %g(SOAPAction)</faultstring>"
                || "%n</SOAP-ENV:Fault>"
        done
    done
done
close SOAPResponse
```

清单 11-7　决定呼叫方法

使用下列六行呼叫该清单中最重要的部份：

```
using group processPO do
   do xml-parse instance
           scan inputData
           suppress
   done
done
```

这些程序代码会使用 OmniMark group 中的一系列规则处理文件，并呼叫 XML 剖析器，而 XML 文件也在这里被处理。

ProcessPO group 开始处理 BizTalk 文件中的元素，ProcessPO group 如清单 11-8 所示。

```
processPO

;-------------------------------------------

group processPO

;-------------------------------------------

global stream txtSendTo

global stream txtReplyTo

global stream itemNum

global stream itemQty

global stream itemPrice

global stream poDate

global stream poNumber

global stream custID

global stream msgBody

global counter itemCounter

global stream needAfter
```

```
global stream needBefore

global stream promiseDate

global counter promiseQty

global stream rowData variable initial-size 0



element purchaseOrder_Chapter11_1.0

    local stream HTMLResponse

    local counter stepper

    local stream payload



    set poDate to attribute poDate

    set poNumber to attribute poNumber

    set custID to attribute custID

    set itemCounter to 0



    open msgBody as buffer

    put msgBody "CustNum=%g(custID)"

    put msgBody "&PONum=%g(poNumber)"

    put msgBody "&PODate=%g(poDate)"

    output "%c"
```

```
close msgBody
set HTMLResponse to postMsg (msgBody,
     "http://localhost/ToyCarParts/PlaceOrder.xar ")
clear rowData
using group parseResponse do
    submit HTMLResponse
done
open payload as buffer
put payload "<purchaseOrder_Confirm_Chapter11_1.0 "
    ||"%n   xmlns='urn:purchaseOrder_Confirm_Chapter11_1.0'"
    ||"%n   CustNum='%g(custID)'"
    ||"%n   PONum='%g(poNumber)'>"

repeat
    set itemNum to rowData item stepper
    increment stepper by 4
    set promiseDate to rowData item stepper
    increment stepper by 1
    set promiseQty to rowData item stepper
```

```
    do unless promiseDate =""
        put payload "%n    <item reqID='"
        || "d " %  (reqIDMapper item (stepper /6))||"'>"
        || "%n     <datePromised>" |||promiseDate
        || "</datePromised>"
        || "%n     <qtyPromised>" |||"d " %%promiseQty
        || "</qtyPromised>"
        || "%n     </item>"
    done
    increment stepper by 1
    exit when stepper >number-of rowData
again
put payload "%n</purchaseOrder_Confirm_Chapter11_1.0>"
close payload
do
    local stream originalBizTalk
    set originalBizTalk to makeResponse(payload,
        txtReplyTo,txtSendTo)
    set fromPOConf to postMsg (originalBizTalk,txtReplyTo)
done
```

```
element #implied
    suppress
element "dlv:address " when parent is "dlv:to"
    set txtSendTo to "%c"
element "dlv:address " when parent is "dlv:from"
    set txtReplyTo to "%c"
element item
    output "%c"
    increment itemCounter
    put msgBody "&After%d(itemCounter)=%v(needAfter)"
    ut msgBody "&Before%d(itemCounter)=%v(needBefore)"
    put msgBody "&Part%d(itemCounter)=%g(itemNum)"
    put msgBody "&Qty%d(itemCounter)=%g(itemQty)"
    put msgBody "&Price%d(itemCounter)=%g(itemPrice)"
    set new reqIDMapper to attribute reqID


element number when parent is item
    set itemNum to "%c"
element price when parent is item
    set itemPrice to "%c"
```

```
element qty when parent is item
    set itemQty to "%c "
```

清单 11-8 ProcessPO group 中包含一系列的规则以处理 BizTalk 讯息中的 XML 元素

在清单 11-8 中，第一个被处理的元素是 purchaseOrder_Chapter11_1.0 ，该元素的主要目的就是对旧型订单处理程序（ProcessOrder.xom）执行足够的处理，当遇到 purchaseOrder_Chapter11_1.0 时，ProcessOrder.xom 开始建立 HTTP POST 文件的数据主体，HTTP POST 方法在 第八章 中有详细的解释，而该文件藉由使用 postMsg 函数张贴至 CGI 程序（PlaceOrder.xom）， postMsg 函数显示于清单 11-9。

```
postMsg
define stream function postMsg (
    value stream payload,
    value stream url
    )as
    local HttpRequest Request
    local HttpResponse Response
    local stream request-Headers variable
    local stream response-Headers variable
```

```
    HttpRequestSetFromURL Request from (url)
    set Request key "method " to "POST"
    set Request key "entity-body " to payload

    HttpRequestSend Request into Response
    HttpObjectGetHeaders Request into request-Headers
    HttpObjectGetHeaders Response into response-Headers

    do when not(HttpObjectIsInError Request)
        return Response key "entity-body"
    done
```

清单 11-9 **postMsg** 函数执行 HTTP POST 方法张贴至 CGI 程序

如清单 11-9 所示， postMsg 函数使用 OmniMark HTTP 函数链接库， HttpRequestSetFromURL 函数初始化 HTTP 封包，在该封包设定后，我们需要变更参数为 POST ，并且将信息放入文件的数据主体。然后我们使用 HttpRequestSend 函数传送封包，并且将响应储存至其它对象中。 HTTP POST 封装的数据主体范例如下：

```
CustNum=84-4281-81&PONum=1182&PODate=2000-0604&After1=2000-05-
```

```
24&Before1=2000-06-22&Part1=LSC91023&Qty1=1&Price1=340&After2=20
00-05-
21&Before2=2000-06-11&Part2=LSC91012&Qty2=12&Price2=35
```

我们唯一感兴趣的就是 HTTP 响应文件中的数据主体，它在张贴后会传回，该传回的文件是由 CGI 程序所建立的，范例如清单 11-10 所示。

```
Return.htm
<HTML>
   <HEAD>
     <TITLE>Toy Car Parts,Inc:Order Entered</TITLE>
   </HEAD>
   <BODY>
     <H1>Order Entered</H1>
     <P>This is to confirm that the order below has been entered.
         Timestamp:2000-06-04 11:54:52</P>
     <TABLE BORDER='1 '>
       <THEAD>
         <TR>
           <TH>Part number</TH>
           <TH>Description</TH>
```

```
            <TH>Price</TH>
            <TH>Status</TH>
            <TH>Ship Date</TH>
            <TH>Ship Quantity</TH>
          </TR>
        </THEAD>
<TR>
     <TD VALIGN='TOP '>LSC91023</TD>
     <TD VALIGN='TOP '>"Marine Corps " Locker,,265lbs </TD>
     <TD VALIGN='TOP '>340</TD>
     <TD VALIGN='TOP '>Manager approval (price too low)</TD>
     <TD VALIGN='TOP '>20000605235452-0600</TD>
     <TD VALIGN='TOP '>1</TD>
</TR>
<TR>
     <TD VALIGN='TOP '>LSC91012</TD>
     <TD VALIGN='TOP '>Single Clothing
Locker,21"x18"x78",117lbs</TD>
     <TD VALIGN='TOP '>35</TD>
     <TD VALIGN='TOP '>Manager approval (price too low)</TD>
```

```
      <TD VALIGN='TOP '>20000605235452-0600</TD>
      <TD VALIGN='TOP '>12</TD>
</TR>
      </TABLE>
   </BODY>
</HTML>
```

清单 11-10 由 PlaceOrder.xom 建立的 HTML 响应文件

一旦 HTML 文件传回，其它 group 中的规则－parseResponse，便由下列程序代码执行：

```
using group parseResponse do
   submit HTMLResponse
done
```

parseReponse group 中包含 find 规则会搜寻文字的样式和设定表格中每一个储存格的值。parseReponse group 如清单 11-11 所示。

```
parseResponse
;---------------------------
group parseResponse
;---------------------------
```

```
find ("<TD VALIGN='TOP '>" any***"</TD>" white--space*)+=>Cells
     repeat scan Cells
          match "<TD VALIGN='TOP '>" any***=>Content "</TD>"
               set new rowData to Content
          match any
     again
find any
    ;suppress the rest
```

清单 11-11　parseReponse group 包含 **find** 规则以处理从 PlacrOrder.xom 传回的 HTML 文件

rowData shelf 中包含储存格中所有的内容，现在就是完成采购订单确认文件的时候了，将该文件传回 Toi Carz，采购订单文件看起来应该是如下列所示：

```
<purchaseOrder_Confirm_Chapter11_1.0
    xmlns='urn:purchaseOrder_Confirm_Chapter11_1.0'
    CustNum='84-4281-81'
    PONum='1189'>
    <item reqID='62'>
          <datePromised>20000606000352-0600</datePromised>
          <qtyPromised>1</qtyPromised>
```

```
   </item>
    <item reqID='70'>
           <datePromised>20000606000352-0600</datePromised>
           <qtyPromised>12</qtyPromised>
   </item>
</purchaseOrder_Confirm_Chapter11_1.0>
```

要完成该文件结构，我们使用清单 11-12 的程序代码。

```
Creating line items
put payload "<purchaseOrder_Confirm_Chapter11_1.0 "
    ||"%n   xmlns='urn:purchaseOrder_Confirm_Chapter11_1.0 '"
    ||"%n   CustNum='%g(custID)'"
    ||"%n   PONum='%g(poNumber)'>"

repeat
    set itemNum to rowData item stepper
    increment stepper by 4
    set promiseDate to rowData item stepper
    increment stepper by 1
    set promiseQty to rowData item stepper
```

```
    do unless promiseDate =""
        put payload "%n <item reqID='"
        ||" d " %%(reqIDMapper item (stepper /6))||"'>"
        ||" %n <datePromised>" |||promiseDate
        ||" </datePromised>"
        ||" %n <qtyPromised>" |||"d " %%promiseQty
        ||" </qtyPromised>"
        ||" %n </item>"
    done
    increment stepper by 1
    exit when stepper >number-of rowData
again
put payload "%n</purchaseOrder_Confirm_Chapter11_1.0>"
close payload
```

清单 11-12　建立采购订单确认程序的产品项目

接着使用 makeReponse 函数将采购订单文件包装在 BizTalk 文件中，如清单 11-13 所示。

```
makeResponse

define stream function makeResponse(

    value stream strPayload,

    value stream strSendTo,

    value stream strReplyTo

    )as

    local stream response



    set response to "<?xml version='1.0'?>"

    ||"%n<SOAP-ENV:Envelope"

    ||"%n

xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'"

    ||"%n

xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'"

    ||"%n  xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance

'>"

    ||"%n  <SOAP-ENV:Header>"

    ||"%n    <dlv:delivery SOAP-ENV:mustUnderstand='1'"

    ||"%n xmlns:dlv='http://schemas.biztalk.org/btf-2-0/delivery

'"
```

```
    ||"%n
xmlns:agr='http://www.trading-agreements.org/types/'>"
    ||"%n    <dlv:to>"
    ||"%n      <dlv:address xsi:type='agr:department '>"
    ||escapeXML(strSendTo)||"</dlv:address>"
    ||"%n    </dlv:to>"
    ||"%n    <dlv:from>"
    ||"%n      <dlv:address xsi:type='agr:organization '>"
    ||escapeXML(strReplyTo)||"</dlv:address>"
    ||"%n      </dlv:from>"
    ||"%n    </dlv:delivery>"
    ||"%n    <prop:properties SOAP-ENV:mustUnderstand='1 '"
    ||"%n
xmlns:prop='http://schemas.biztalk.org/btf-2-0/properties '>"
    ||"%n      <prop:identity>uuid:" ||genUUID()
    ||"</prop:identity>"
    ||"%n      <prop:sentAt>"
    ||iso8601(now-as-ymdhms)||"</prop:sentAt>"
    ||"%n      <prop:expiresAt>"
    ||iso8601(add-to-ymdhms now-as-ymdhms hours 2)
```

```
    ||"</prop:expiresAt>"
    ||"%n      <prop:topic>" ||escapeXML(strReplyTo)
    ||"</prop:topic>"
    ||"%n    </prop:properties>"
    ||"%n  </SOAP-ENV:Header>"
    ||"%n  <SOAP-ENV:Body>"
    ||"%n " ||strPayload
    ||"%n  </SOAP-ENV:Body>"
    ||"%n</SOAP-ENV:Envelope>"
  return response
```

清单 11-13 **makeReponse** 函数建立 BizTalk 信封

在下一节中我们将看到采购订单确认处理程序，它是一个 ASP 网页，用于处理包含 BizTalk 讯息的 HTTP POST 文件。我们呼叫它的方式就如同呼叫订单处理程序（PlaceOrder.xom）一样。

采购订单确认的处理

回到 Toi Carz，一个 ASP 应用程序称为 poConfirm.asp 正在等待接收我们的采购订单确认讯息。这个程序扮演 BizTalk 伺服程序以便读取 BizTalk

文件，且基于数据主体元素类型和内容进行处理。采购订单确认程序如清单 11-14 所示。

```
poConfirm.asp
<%@language=vbscript%>
<%
Function ymdhms2mdy(ymdhms)
    strYear =mid(ymdhms,1,4)
    strMonth =mid(ymdhms,5,2)
    strDay =mid(ymdhms,7,2)
    ymdhms2mdy =strMonth &"/" &&strDay &"/" &&strYear
End Function
Function closeReq(reqID,datePromised,qtyPromised)
    Set dbPurchaseOrders =Server.CreateObject("ADODB.Connection")
    Set rsPORequests =Server.CreateObject ("ADODB.Recordset ")
    rsPORequests.CursorType =2
    rsPORequests.LockType =2

    dbPurchaseOrders.Open "dbTC-PurchaseOrders"
    txtSQL ="SELECT PORequests.reqID,vendors.vendorName," __
```

```
    &"PORequests.itemPrice,PORequests.needBefore," __
    &"PORequests.needAfter,PORequests.itemQty," __
    &"PORequests.qtyPromised,PORequests.datePromised," __
    &"PORequests.itemNum,PORequests.requestorID," __
    &"catalog.description,PORequests.itemPrice," __
    &"PORequests.itemQty FROM [catalog ] " __
    &"INNER JOIN (vendors INNER JOIN PORequests " __
    &"ON vendors.vendorID =PORequests.vendorID)" __
    &"ON catalog.itemNum =PORequests.itemNum " __
    &"WHERE (((PORequests.reqID)=" &&reqID &"));"
rsPORequests.Open txtSQL,dbPurchaseOrders
rsPORequests("datePromised ")=ymdhms2mdy(datePromised)
rsPORequests("qtyPromised ")=qtyPromised
rsPORequests.Update


Set dbEmployees =Server.CreateObject("ADODB.Connection ")
dbEmployees.Open "dbTC-Employees"
Set rsEmployees =dbEmployees.Execute(_
     "SELECT *FROM Employees where empID=" _
    & rsPORequests("requestorID "))
```

```
    If Not rsEmployees.EOF Then
        Set exchangeMail =CreateObject("CDONTS.NewMail ")
        strFrom ="biztalk_robot@toicarz.com"
        strTarget =rsEmployees("EMail ")
        strSubject ="Purchase requisition confirmation"
        strMsg ="The purchase requisition for your order has " __
               "been confirmed " __
            &vbCrLf &" Item::" &&rsPORequests("itemNum ")_
            &vbCrLf &" Description::" &&rsPORequests("description
")_
            &vbCrLf &" Vendor::" &&rsPORequests("vendorName ")_
            &vbCrLf &" Qty Ordered::" &&rsPORequests("itemQty ")_
            &vbCrLf &" Qty Promised::" &&rsPORequests("qtyPromised
")_0
            &vbCrLf &" Date Required::" &&rsPORequests("needBefore
")_
            &vbCrLf &" Date Promised::" &&rsPORequests
            ("datePromised ")
        Response.Write(vbCrLf &"Message to " __
```

```
            &rsEmployees("FName ")&" " &&rsEmployees("LName ")_
            &" <<" &&rsEmployees("Email ")&">" __
            &vbCrLf &vbCrLf &strMsg)
    exchangeMail.Send strFrom,strTarget,strSubject,strMessage
    Set exchangeMail =Nothing
End If
    rsPORequests.Close
    Set rsPORequest =Nothing
    dbEmployees.Close
    Set dbEmployees =Nothing
    dbPurchaseOrders.close
    Set dbPurchaseOrders =Nothing
End Function
Function closePO(poNum)
    Set dbPOSent =Server.CreateObject("ADODB.Connection ")
    dbPOSent.Open "dbTC-PurchaseOrders"
    txtSQL ="UPDATE POSent SET confReceived=#" &&CDate(date)_
        &"#WHERE poNum=" &&poNum
    dbPOSent.Execute txtSQL ',intrecs,adcmdtext
    dbPOSent.Close
```

```
    Set dbPOSent =Nothing
End Function
HTTPSize =Request.TotalBytes
HTTPBody =Request.BinaryRead(HTTPSize)
strXML =""
For i =1 To HTTPSize
    strXML =strXML &Chr(AscB(MidB(HTTPBody,i,1)))
Next
strXML =Replace (strXML,"%3C","<")
strXML =Replace (strXML,"%3E",">")
strXML =Replace (strXML,"%20"," ")
strXML =Replace (strXML,"%2F","/")
strXML =Replace (strXML,"%3D","=")
strXML =Replace (strXML,"%0A",vbCrLf)

Set oXMLDoc =Server.CreateObject("Microsoft.XMLDOM ")
oXMLDoc.loadXML(strXML)

If oXMLDoc.parseError.errorcode <>0 Then
    Response.Write("Error,Reason:" &&oXMLDoc.parseError.reason _
```

```
        &",Code:" &&oXMLDoc.parseError.errorcode _
        &",Line:" &&oXMLDoc.parseError.line)
End If
poNum =oXMLDoc.selectSingleNode_
("//purchaseOrder_Confirm_Chapter11_1.0
").attributes.getNamedItem_
("PONum ").nodeValue
Set collItem =oXMLDoc.selectNodes("//item ")
For i =0 To collItem.length -1
    reqID =collItem.item(i).attributes.getNamedItem("reqID
").nodeValue
    datePromised =collItem.item(i).selectSingleNode("datePromised
").text
    qtyPromised =collItem.item(i).selectSingleNode("qtyPromised
").text
    Response.Write("reqID:" &&reqID _
        &",datePromised:" &&datePromised _
        &",qtyPromised:" &&qtyPromised)
    closeReq reqID,datePromised,qtyPromised
Next
```

```
closePO(poNum)
%>
```

清单 11-14　poConfirm.asp 程序处理采购订单确认讯息

poConfirm.asp 程序必须执行三项工作，首先它必须关闭每一个采购请求，关闭的基准是使用 Toy Car Parts 承诺的交货日期和数量，才能更新 PORequests 数据表，它使用 closeReq 函数完成这个工作；同时这个函数也会执行第二个工作：通知发起采购请求的人（藉由使用 Collaboration Data Objects library CDONTS），CDONTS 提供了使用 Microsoft Exchange Server 的接口以传送电子邮件。

第三个工作是在 POSent 数据表中将某采购订单的记录加以标记好关闭采购订单，它使用 closePO 函数完成这个工作。

该程序会传送一响应至 OmniMark BizTalk 伺服程序，服务器会将讯息传送回起始该处理程序的采购订单产生程序。

这是一个非常基本的系统，它显示了使用两种操作系统、语言平台的 BizTalk 伺服程序环境最小的需求。当然，在拥有许多可能变因存在的环境下建置这种系统比较困难，但是像 BizTalk、SOAP 和 XML 这种开放式的标准使这种工作变得更简单了。

在我们面临建立系统时最有兴趣的挑战之一就是使用现存的应用程序，例如范例中的 PlaceOrder 系统已经完全没有问题并且已经工作正常了数年，因此我们并不需要去改变或放弃它。同时也因为我们使用的这些开放式标准和工具，我们延伸了旧型应用程序的生命周期。

在本章中所介绍的 SOAP Toolkit for Visual Studio 6.0 是一个十分好的工具，它允许开发者可以更进一步的使用 XML 语法以部署具有影响整个 Web 能力的系统。这真是一个功能强大的工具，而且其它供货商也已经宣称他们会支持 SOAP 相关的工具。

为了与 BizTalk 伺服程序协同工作，有许多的工具正在开发中。笔者将在 第十二章 教导您学习内建于 Microsoft BizTalk Server 2000 的某些强大工具，商业程序设计师可以使用例如 BizTalk Editor 的工具以更加容易的掌控 XML，BizTalk Editor 允许您建立架构（schema）；BizTalk Mapper 可以轻松建立在不同架构之间的转换。

12. Microsoft BizTalk Server 2000

第十章 已经提及一些当您在开发环境中撰写或选择 BizTalk Framework Compliant（BFC）服务器时应该加以考量的因素，但是有一点请您注意：您只需要让 BizTalk 伺服程序能够了解 BizTalk Framework 的卷标，并且基于这些标签以智能型的方式对文件做出应有的反应。

Microsoft BizTalk Server 2000 对 XML 的处理能力远超过 第十章 所描述的，这个服务器联合了数个服务，允许您整合商业文件至您目前的系统，这个产品也提供了您管理数据和交换信息的新组件，以及管理和维护服务器的组件，同时在伺服端的设计、整合商业伙伴和您的组织中的商业文件上也可以符合您的需求。

BizTalk Server 2000 是用于交换商业数据的 **数据解译** 和 **应用程序整合** 伺服程序，BizTalk Server 2000 使用 XML 技术以让您可以跨产业、平台、操作系统或底层技术的方式，与您的商业伙伴建立合作关系。服务器提供标准的网关（gateway）透过因特网传送和接收文件，并包含一些服务以确保数据的完整性、安全性和正确的传递。

BizTalk Server 2000 有许多的功能，其中包含收接输入文件、剖析文件以决定它们的格式、指定特定的处理规则，以及将文件分别传递至它们的目的地；同时也包含了数据对应（data mapping）的服务。

BizTalk Server 2000 提供了一系列的工具，您可以利用这些工具来定义商业文件的结构（schema）并使用 XSLT 在这些文件之间中转换数据。

虽然该系统中所有的数据传送都使用 XML，但是也可以接受其它的格式，一般大部分最常使用的是电子数据交换（Electronic Data Interchange，EDI），在 BizTalk Server 2000 中的工具可以为应用程序整合服务器提供数据解译的必要元素。

本章使用两个实际操作的范例，您可以使用下列这些工具建立和转换 XML 文件：Microsoft BizTalk Editor 和 Microsoft BizTalk Mapper。

BizTalk Editor

第四章 曾经提及一个建立 XML 架构的工具－XML Authority，为什么您需要建立架构的其它工具呢？虽然 XML Authority 已经是建立任何架构的强大工具了，但是 BizTalk Editor 却是建立定义商业文件架构的最佳工具。还有，经由 BizTalk Editor 建立的架构中，包含了额外 Microsoft 和 BizTalk 名称空间自订的信息，使用这些信息可以简化两种格式之间的对应。

绝佳演练：建立采购订单架构

现在就让我们来使用 BizTalk Editor 建立可以使用于 BizTalk Server 2000 的架构吧！首先建立我们最喜欢的商业文件－采购订单。这将会是我们用来和合作伙伴交换商业信息的标准采购订单架构，本范例是使用 BizTalk Server 2000 PDC Pre-Release 中的工具建立，您应该使用这个版本或者是 BizTalk Server 2000 的后续版本完成本练习。

1. 首先您需要加载 BizTalk Editor，按一下 **开始/程序集/Microsoft BizTalk Server 2000/BizTalk Editor**，您就可以看见如图 12-1 所示的画面。

   图 12-1 的范例只是要让您了解这个简单的工具，并且看看它们是如何工作的，大部分的文件都比范例要复杂许多。然而，您在此所学到的概念可以套用到所有的文件上。

2. 在 **File** 菜单中，按一下 **New** 建立一份新文件，您会看见 **New Document Specification** 对话框，按一下 **Blank Specification**，然后按一下 **OK**。

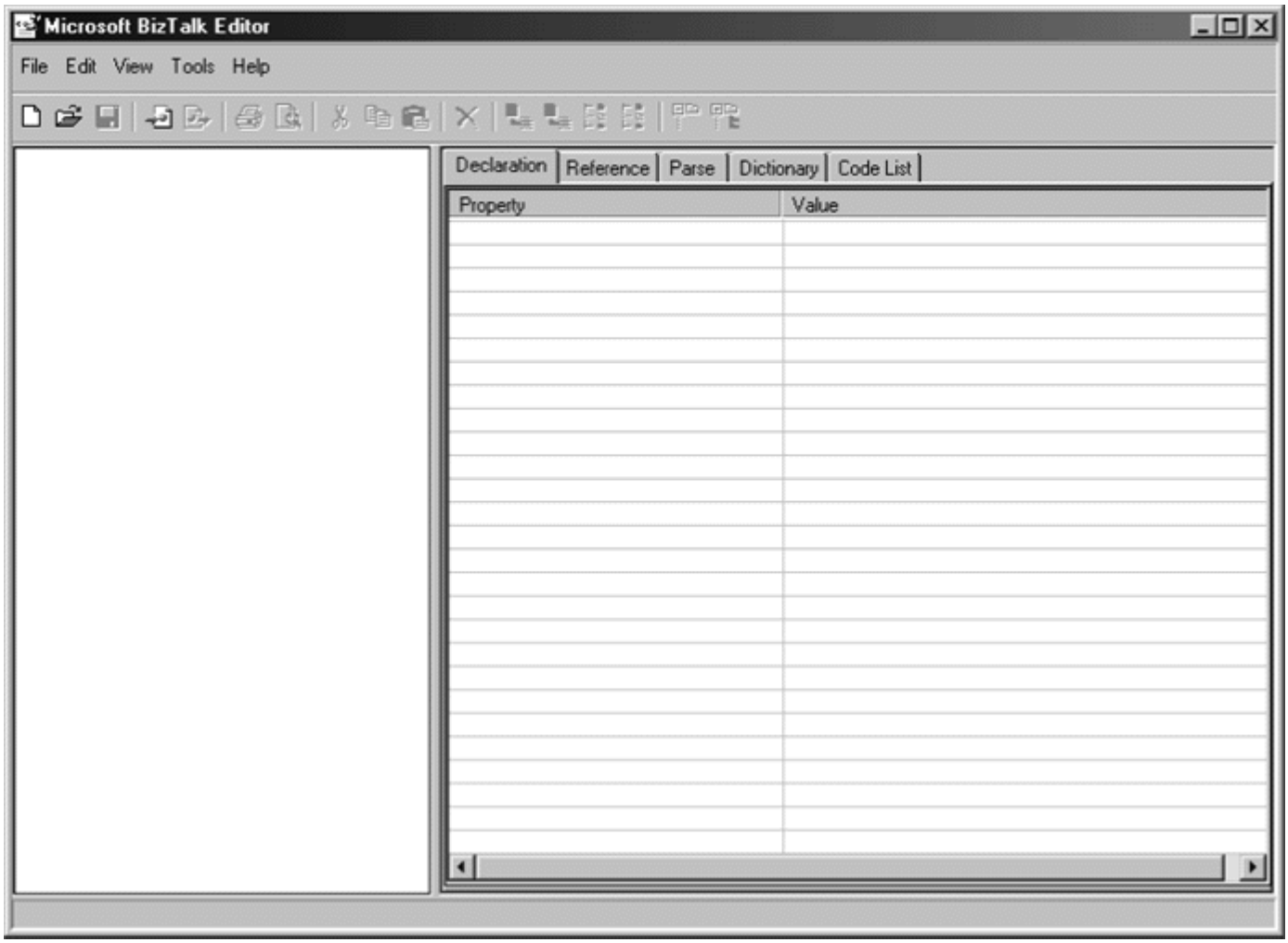


图 12-1　BizTalk Editor 让您可以建立描述商业文件结构的规格

3. 在编辑器的左侧窗格中，您会看见标示着 **Blank Specification** 的小图示，这是一个将会扮演我们文件中的根（root）的记录，在该记录上按一下鼠标右键，然后按一下 **Rename**，将该笔记录重新命名为 **PurchaseOrder**，然后按下 **Enter** 钮。

请注意 BizTalk Editor 不会使用您所熟悉的 XML 元素和属性这些字，而是使用记录（record）和字段（field），典型的数据库程序设计师来看这些纪录可说是十分熟悉。记录就好比数据库中所指的一样，它可以包含许多字段；而字段即为数据存放的位置。在 BizTalk Editor 中，一笔记录可以包含数据和字段。在宣告区中，您必须决定哪些为元素、哪些为属性，默认值记录为元素、字段为属性。

4. 在我们刚刚重新命名的新记录上按一下鼠标右钮，并且选取 **New Field** ，将其命名为 **PODate** 。
5. 重复步骤 4，并且将新字段命名为 **CustomerNumber** 。
6. 在 **PurchaseOrder** 记录上按一下鼠标右钮，并且按一下 **New Record** ，您将会为其它字段建立一个容器，将这笔新记录称为 **Item** ，请注意在右侧窗格中的字段更进一步的被定义了，所有您建立的字段被定义为 **Type Attribute** ，而记录预设会被定义为 **Type Element** 。按一下包含 **Empty** 值的 **Content** 储存格，该储存格会变成下拉式的清单方块，从下拉式清单方块中将该值变成 **Text Only** ，我们将使用这笔记录以掌控被订购的产品项目。
7. 在 **Item** 记录上按一下鼠标右钮，并且选取 **New Field** ，将该字段名称命名为 **Qty** ，在 **Declaration** 窗格中，按一下 Data Type 右边的储存格并且从下拉式清单中选取 integer (**int**)。
8. 重复步骤 7，并且将新字段命名为 **Price** ，其数据型别为 **Fixed Point (14.4)** 。

9. 从 **File** 菜单中，选取 **Save**，将文件名称命名为 **PurchaseOrder1.xml**，该画面看起来就像图 12-2 一样。

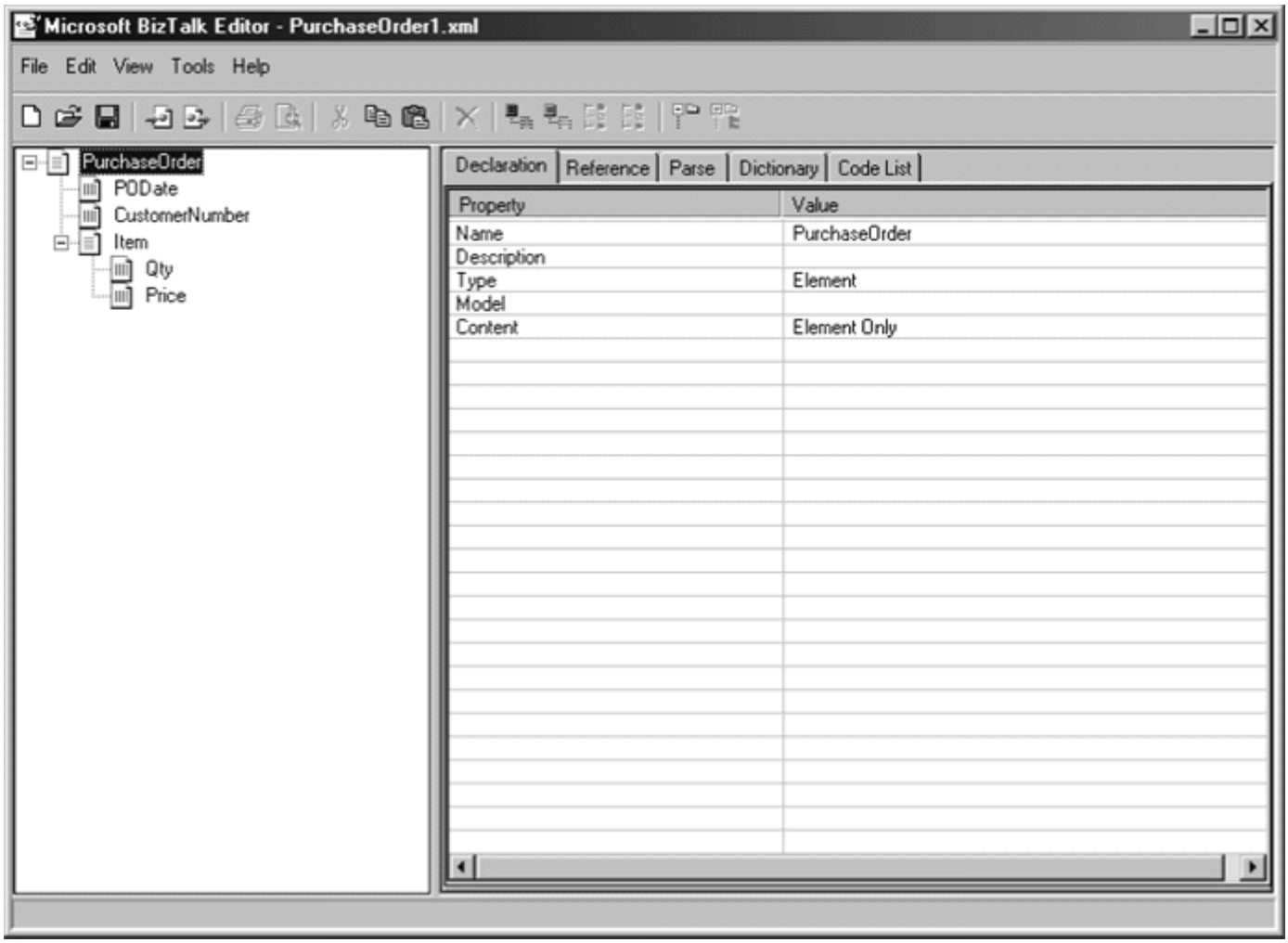


图 12-2　在 BizTalk Editor 中最后的采购订单结构

当您将该文件储存后，如清单 12-1 的 XML 架构就建立了。然而，这个架构和 第四章 中描述的架构有一点不同，因为 BizTalk Server 使用了基本架构语法没有提供的额外信息。

在本范例中所建立的 PurchaseOrder1.xml 文件在 XML Data Reduced（XDR）中有详细的说明。然而，当 W3C XML Schema Definition（XSD）语法最后公布的时候，Microsoft 同时也会支持该语法，只要简单的按一下 **Save As**，就能从一种语法转换到另一种语法，您也可以使用 XSLT 以使语法变更。

在清单 12-1 中，请注意在 **<Schema>** 开始标签中被定义的 **b:** 名称空间，该名称空间中包含了 BizTalk Server 2000 许多部分都会使用到卷标集合。

PurchaseOrder1.xml

```
<?xml version="1.0 "?>
<Schema name="PurchaseOrder"
    b:root_reference="PurchaseOrder "
    b:standard="XML "
    xmlns="urn:schemas-microsoft-com:xml-data "
    xmlns:d="urn:schemas-microsoft-com:datatypes "
    xmlns:b="urn:schemas-microsoft-com:BizTalkServer">
    <b:SelectionFields/>
    <ElementType name="PurchaseOrder " content=="eltOnly">
        <b:RecordInfo/>
        <AttributeType name="PODate">
            <b:FieldInfo/>
```

```
        </AttributeType>
        <AttributeType name="CustomerNumber">
            <b:FieldInfo/>
        </AttributeType>
        <attribute type="PODate"/>
        <attribute type="CustomerNumber"/>
        <element type="Item"/>
    </ElementType>
    <ElementType name="Item " content="textOnly " model="closed">
        <b:RecordInfo/>
        <AttributeType name="Qty " d::type="int">
            <b:FieldInfo/>
        </AttributeType>
        <AttributeType name="Price " d::type="fixed.14.4">
            <b:FieldInfo/>
        </AttributeType>
        <attribute type="Qty"/>
        <attribute type="Price"/>
    </ElementType>
</Schema>
```

清单 12-1　在 BizTalk Editor 中建立的架构

BizTalk Mapper

在 第六章 中，您撰写了 XSLT 程序转换 XML 文件至其它的格式，如果您的商业合作伙伴使用不同架构的话，这种类型的文件转换是 BizTalk 处理程序的重要部分。

即使笔者仍然是比较喜欢直接使用 XSLT 工作，但并不是每个人都会喜欢这样做，也不是每个公司都负担得起一个 XSLT 专家。使用 BizTalk Mapper，您可以对应不同架构之间的记录和字段。BizTalk Server 2000 使用对应的方式以处理和解译数据为一种格式，您的组织和商业合作伙伴就可以共享该格式以达成资料交换的目的。

最佳演练：对应两种不同的采购订单架构

为了让您了解 BizTalk Mapper 是如何工作的，现在让我们来看一看刚刚所建立的架构，并且将其对应到描述另外一个采购订单的架构。

1. 按一下 **开始/程序集/Microsoft BizTalk Server 2000/BizTalk Mapper** 以启动 BizTalk Mapper。
2. 为了要完成对应的工作，您必须找到两种架构－从 **File** 菜单，选取 **New**，在 **Select Source Specification Type** 对话框中，按一下 **Local Files**，然后按

一下 **OK**，此时会出现 **Select Source Specification** 对话框，然后选择您在第一个范例中所建立的 PurchaseOrder1.xml 架构。

您应该会看见 **Select Destination Specification Type** 对话框，重复上一个步骤并选取第二个架构－位于随书光盘的 **\Samples\Ch12** 中的 PurchaseOrder2.xml，您的画面看起来应该会像图 12-3 所示。

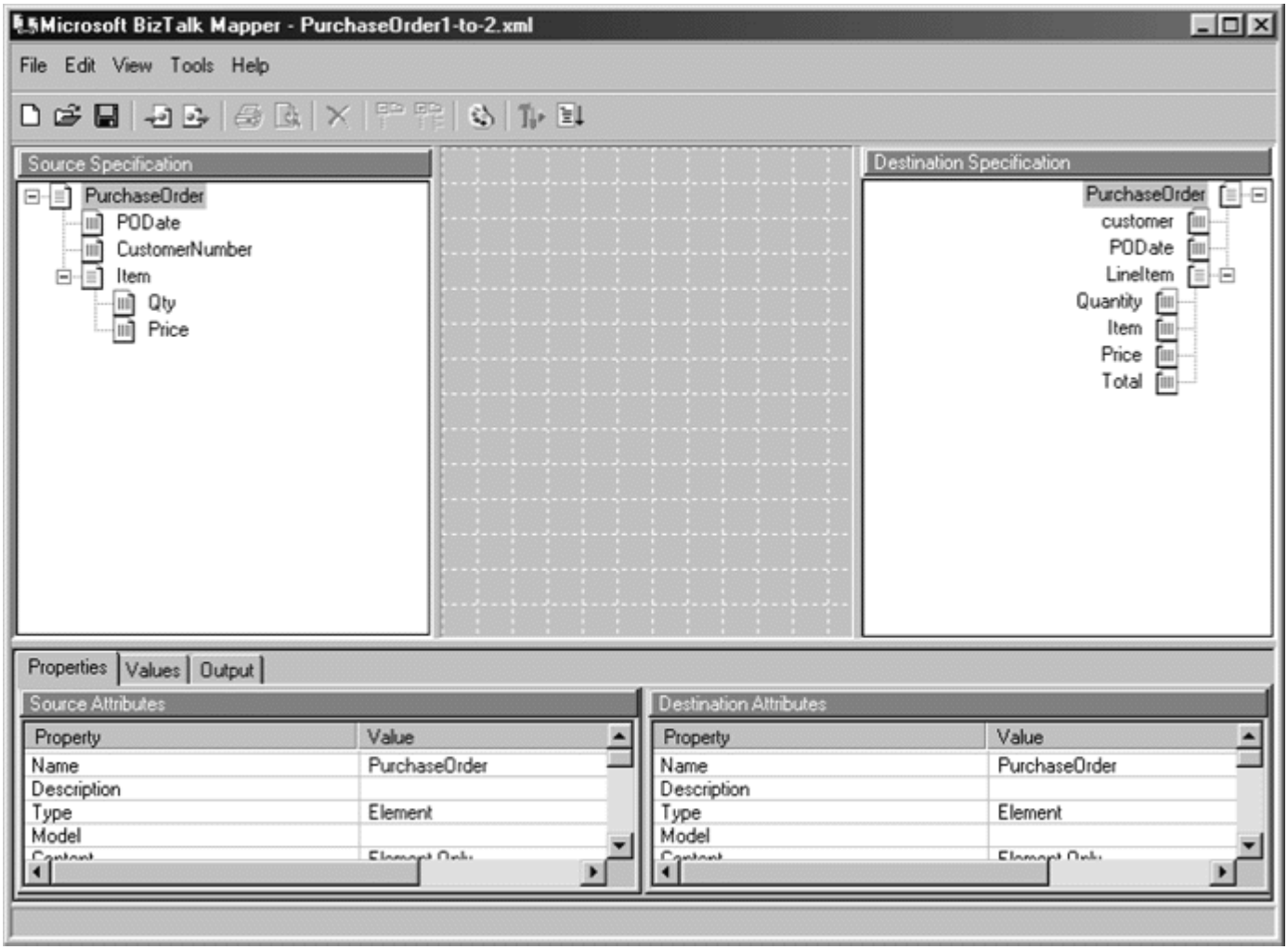


图 12-3　在 BizTalk Mapper 中对应两种不同的采购订单架构

您可以在左侧字段使用拖曳的方式至右侧字段以建立一个对应，处理过程就是这么简单，但是 BizTalk Mapper 增加了许多的重要新功能。

3. 从 **Tools** 菜单中，选取 **Options** 以开启 **BizTalk Mapper Options** 对话框，选取 **Allow record content links** 复选框，选取该选项允许您对应记录的文字内容，而不只是对应字段，按一下 **OK** 。

在 **Source Specification** 窗格中拖曳 **CustomerNumber** 字段至 **Destination Specification** 窗格的 **customer** 字段，您将会看见一条线连接两个字段，如图 12-4 所示。

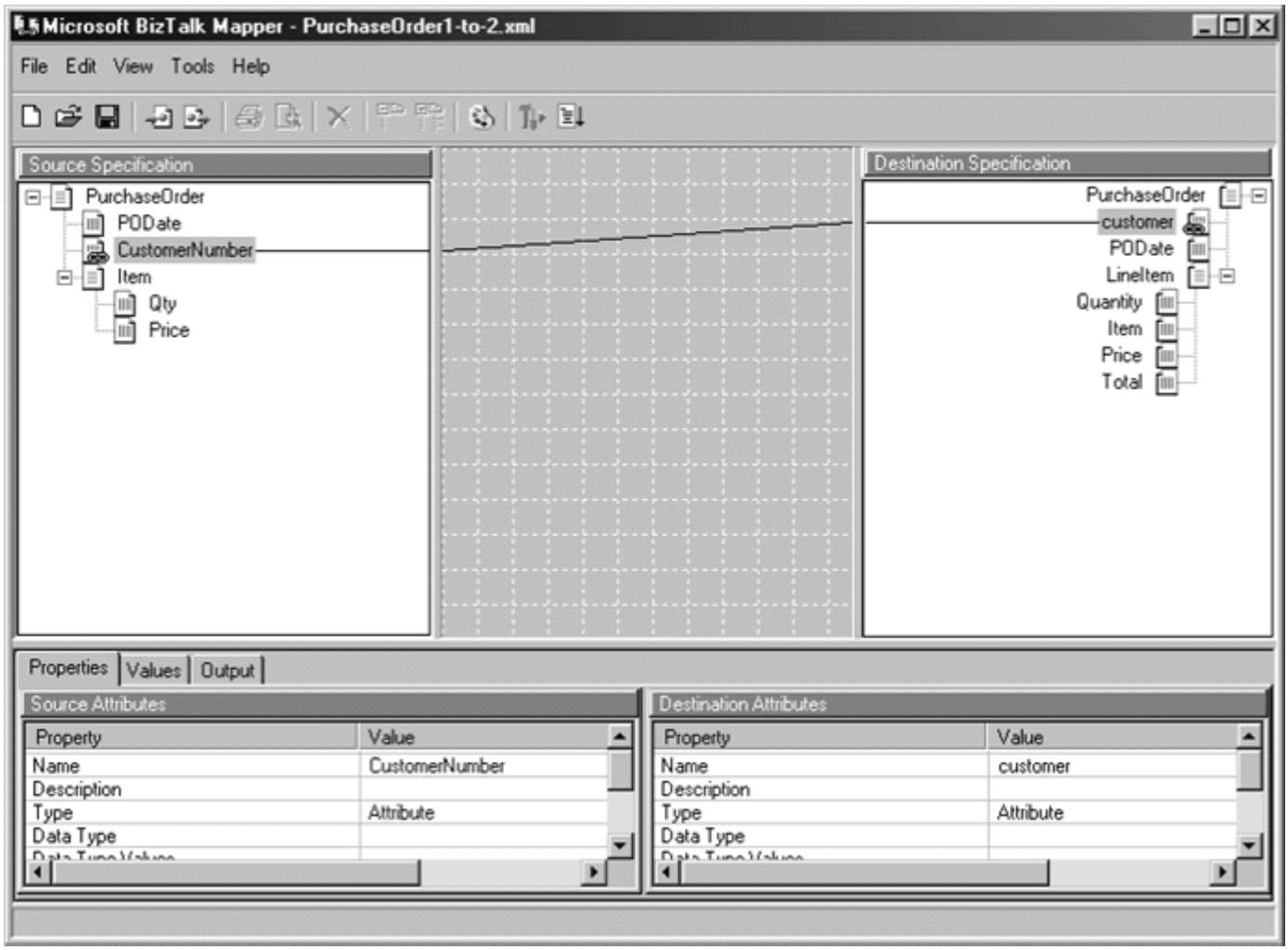


图 12-4 您可以藉由连接 **Source Specification** 窗格和 **Destination Specification** 窗

格中的字段以建立对应

4. 重复步骤 4 三次，建立 **Item-Item** 、 **Qty-Quantity** 和 **Price-Price** 的对应，十分简单。

5. 请注意在 **Destination Specification** 窗格中有一个并不包含在 **Source Specification** 窗格中的字段－ **Total** 。在这种情况下，您就无法直接对应两个窗格之中的文件，所以您将会做某些计算。从 **View** 菜单中，选取 **Functoid Palette** ，您将会看见 **Functoid Palette** 对话框，如图 12-5 所示。

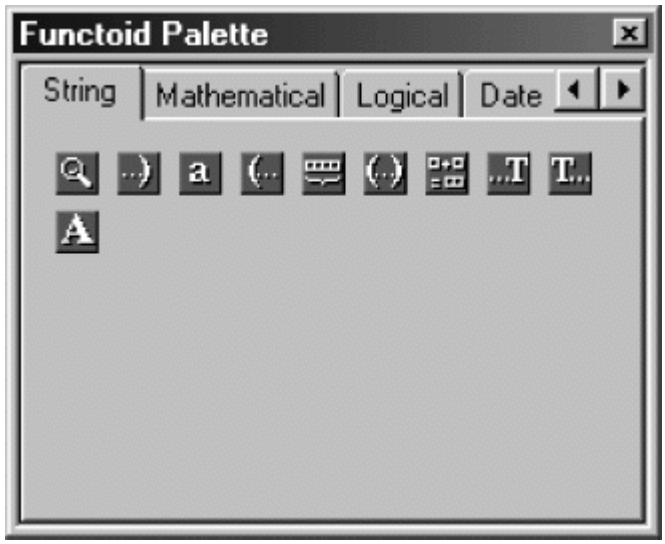


图 12-5 Functoid 在来源和目的档案之间提供了许多不同的计算辅助以供对应字段使用

6. 在 **Destination Specification** 窗格中的 **Total** 字段是 **Source Specification** 窗格中的 **Qty** 和 **Price** 字段计算而成，所以您必须做一些乘法，按一下 **Mathematical** 标签页。

7. 选取 Functoid 图标，该图标就像乘法符号一样，从 Functoid Palette 中将这个 Functoid 拖曳到 **Source Specification** 和 **Destination Specification** 窗格中的方格中，您的画面看起来会像图 12-6 一样。

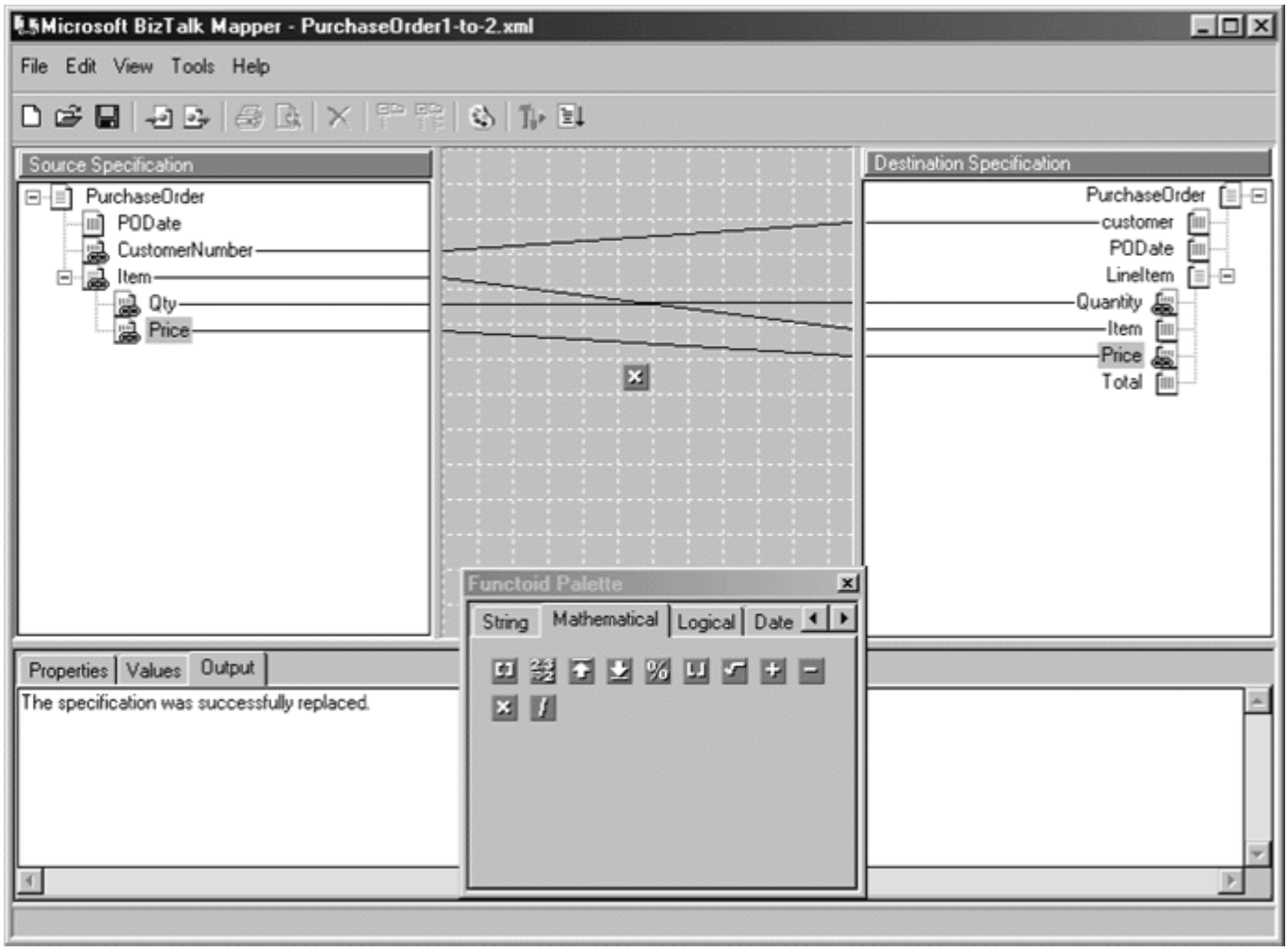


图 12-6 乘法 Functoid 允许您将 **Source Specification** 中的两个数字相乘的结果放到 **Destination Specification** 中。

8. 在 **Source Specification** 中的 **Qty** 元素拖曳一条线到乘法 Functoid 中。

9. 重复步骤 8 以拖曳 **Price** 元素。

10. 从乘法 Functoid 中拖曳一条线至 **Destination Specification** 窗格的 **Total** 字段。

    您已经完成了将两个数字相乘，并且将其结果输出。

11. 您仍然需要处理日期，在 **Source Specification** 文件的 **date** 字段中包含的是 U.S.格式的日期： **month/day/year** ；在 **Destination Specification** 文件中的日期规格是 ISO 8601 的格式： **YYYY-MM-DD** 。并没有存在此种形式转换的 Functoid，所以您需要建立一个特别的指令码以达成这个目的。按一下 **Functoid Palette** 对话框中的 **Advanced** 卷标页，该卷标页中包含一个 Functoid － **Custom Visual Basic Script Functoid** ，将这个 Functoid 拖曳至方格区域中。

12. 从 **Source Specification** 的 **PODate** 字段拖曳一条线至该 Scripting Functoid 中，然后再从该 Scripting Functoid 中拖曳一条线至 **Destination Specification** 的 **PODate** 字段。

13. 按两下该 Scripting Functoid，然后按一下 **Script** 标签页，您将会看见预设指令码如图 12-7 所示。

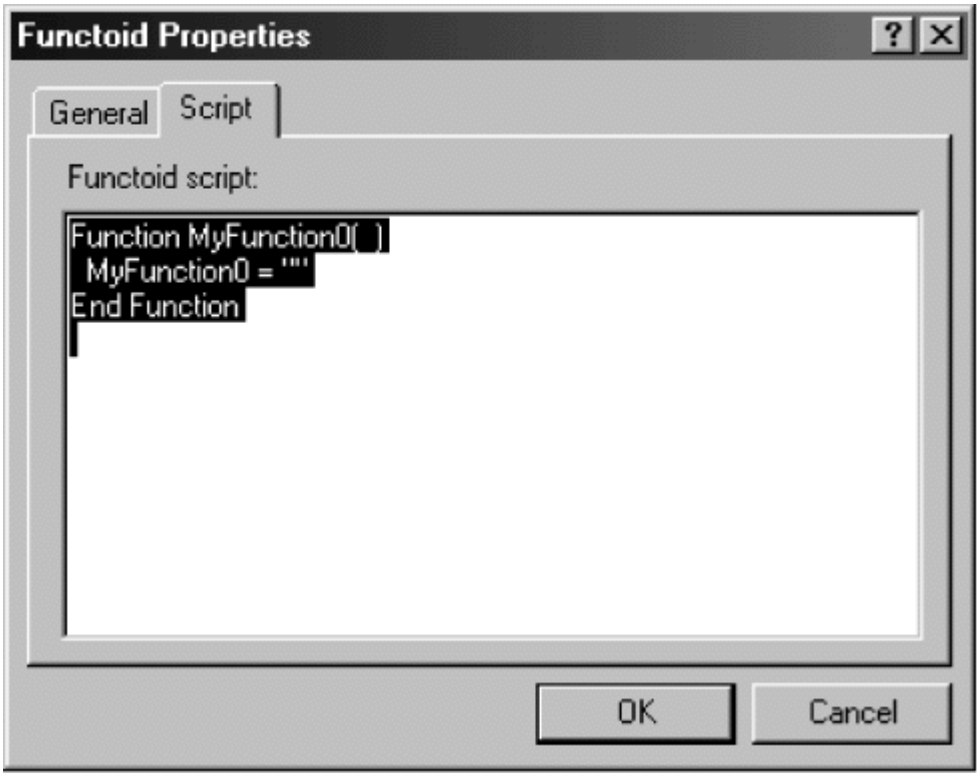


图 12-7 **Custom Visual Basic Script Functoid** 允许您建立指令码以辅助来源文件至目的文件的转换

14. 删除现存的函数，并且将它取代为下列的指令码：

```
15.     Function mdy2iso8601(dateIn
16.           iFirst = instr(dateIn, "/")
17.           iSecond = instr(iFirst + 1, dateIn, "/")
18.           iMonth = left(dateIn, iFirst - 1)
19.           iDay = mid(dateIn, iFirst + 1, iSecond - iFirst - 1)
20.           iYear = mid(dateIn, iSecond + 1, 4)
```

```
21.          mdy2iso8601 = iYear & "-" & right("0" & iMonth, 2)
   & _
22.                              "-" & right("0" & iDay, 2)
     End Function
```

这个 **mdy2iso8601** 函数会转换来源格式的日期（mm/dd/yyyy）至您需要的格式，按一下 **OK** 以关闭 **Functoid Properties** 对话框。您的 BizTalk Mapper 画面将会看起来如图 12-8。

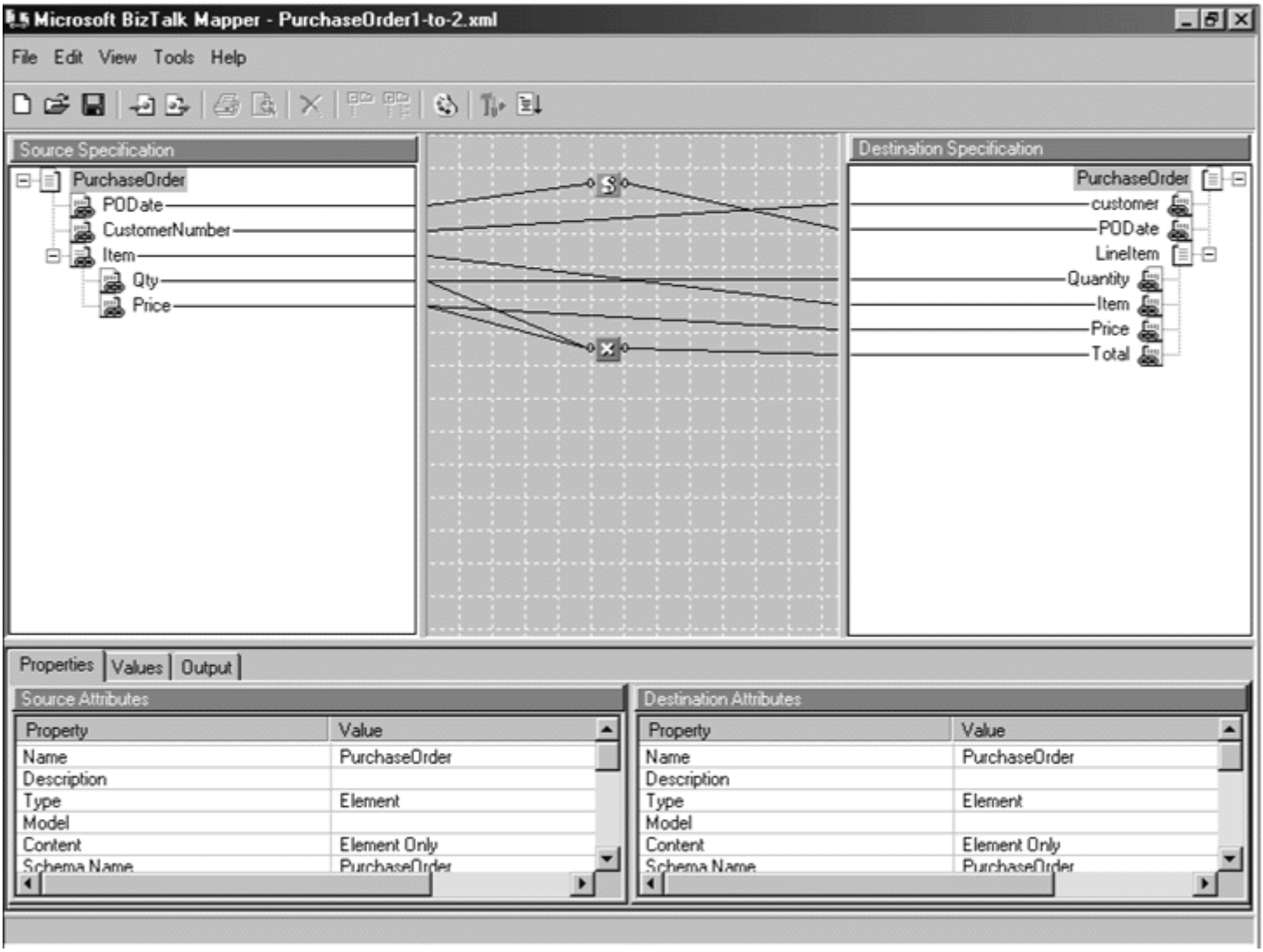


图 12-8　使用 Functoid 功能的 BizTalk Mapper

23. 现在您需要测试该指令码是否可以正确执行，关闭 **Functoid Palette** 对话框，按一下在 BizTalk Mapper 底部的 **Values** 标签页，确认 **Source Specification** 窗格中的 **PODate** 字段为反白状态，然后以 mm/dd/yyyy 格式输入来源测试值，例如 04/27/2000。您将会在稍后的步骤中测试该函数，但必须先输入某些其它字段的测试值。

24. 现在您需要确认乘法 Functoid 正常工作，在 **Source Specification** 窗格中选取 **Qty** 字段，并且在该字段中输入一个整数值－例如 95。

25. 在 **Source Specification** 窗格中选取 **Price** 字段，并且输入小数价格－例如 12.95。

26. 现在您已经准备好要测试了，从 **Tools** 菜单中，选取 **Test Map** ，请注意 **Output** 标签页会最先出现，并且处理的输出结果就会显示，输出结果会如下所示：

```
27.     <PurchaseOrder customer="CustomerNumber_1"
28.             PODate="2000-04-10">
29.         <LineItem>
30.         <Quantity>95</Quantity>
31.         <Item>Item_1</Item>
32.         <Price>12.95</Price>
33.         <Total>1230.25</Total>
34.         </LineItem>
    </PurchaseOrder>
```

请注意 **PODate** 的格式已经转变了，并且在 **<Total>** 标签中的值也是数量乘以价格的乘积。

35. 现在您需要建立将会执行转换的 XSLT 样式表，开启 **Tools** 菜单，并且按一下 **Compile Map**，请注意输出的结果是 XSLT 程序，如清单 12-2 所示。

当您的商业合作伙伴传送给您 XML 文件时，BizTalk Server 2000 将使用清单 12-2 的样式表将其文件转换为 PruchaseOrder1.xml 中所定义的架构。

PO1-2.xsl

```
<xsl:stylesheet
    xmlns:xsl='http://www.w3.org/1999/XSL/Transform '
    xmlns:msxsl='urn:schemas-microsoft-com:xslt '
    xmlns:var='urn:var' xmlns::user='urn:user'
version=='1.0 '>
    <xsl:output method='xml' indent=='yes' omit--xml-
    declaration='yes'/>
    <xsl:template match='/'>
        <xsl:apply-templates select='PurchaseOrder '/>
    </xsl:template>
    <xsl:template match='PurchaseOrder '>
        <PurchaseOrder>
            <xsl:attribute name='customer '>
                <xsl:value-of select='@CustomerNumber '/>
```

```
            </xsl:attribute>
            <xsl:variable name='var:v1'
select='user:mdy2iso8601(string(@PODate))'/>
            <xsl:attribute name='PODate'>
                <xsl:value-of select='$var:v1'/>
            </xsl:attribute>
        <xsl:for-each select='Item'>
            <LineItem>
                <Quantity>
                    <xsl:value-of select='@Qty'/>
                </Quantity>
                <Item>
                    <xsl:value-of select='./text()'/>
                </Item>
                <Price>
                    <xsl:value-of select='@Price'/>
                </Price>
                <xsl:variable name='var:v2'
```

```
select='user:fctmathmultiply2(string(@Qty),string
            (@Price))'/>
                <Total>
                    <xsl:value-of select='$var:v2 '/>
                </Total>
            </LineItem>
        </xsl:for-each>
        </PurchaseOrder>
    </xsl:template>
<msxsl:script language='VBScript ' implements--prefix='user
'>
    <![CDATA [
    Function FctMathMultiply2(p_strParm0,p_strParm1 )
      If (IsNumeric(p_strParm0 )And _
        IsNumeric(p_strParm1 ))Then
          FctMathMultiply2 =CStr (CDbl(p_strParm0 )_
          *CDbl(p_strParm1 ))
      Else
        FctMathMultiply2 =""
```

```
      End If
    End Function
    Function mdy2iso8601(dateIn)
        iFirst =instr(dateIn,"/")
        iSecond =instr(iFirst +1,dateIn,"/")
        iMonth =left(dateIn,iFirst -1)
        iDay =mid(dateIn,iFirst +1,iSecond -iFirst -1)
        iYear =mid(dateIn,iSecond +1,4)
        mdy2iso8601 =iYear &"-" &&_
        right("0 " &&iMonth,2)&_
         "-" &&right("0 " &&iDay,2)
      End Function
      ] ]>
    </msxsl:script>
</xsl:stylesheet>
```

清单 12-2　由 BizTalk Mapper 所建立的 XSLT 样式表

除了 XML，BizTalk Mapper 也可以读取和写入其它格式，其中包含广泛使用的 EDI 规格：ANSI X12 和 EDIFACT。BizTalk Mapper 也可以对应至一般档案、使用文件类别宣告（DTD）架构语法和 XDR 架构所定义的 XML 档案。然而即使

BizTalk Mapper 可以使用各种不同的标准，使用基于 XML 的 BizTalk Framework 2.0 可以达到最佳化目的。

其它 BizTalk 工具

Microsoft BizTalk Server 2000 还有数个其它功能让您可以管理与商业伙伴之间的合作关系：

- **BizTalk Management Desk** 您可以在服务器或远程客户端上使用这个图形化管理接口（GUI）以管理商业伙伴和应用程序之间的数据交换。

- **BizTalk Server Administration Console** 这是一个 Microsoft Management Console（MMC）嵌入式管理单元，您可以用来管理和维护您的服务器和服务器群组。这个管理主控台中的文件包含额外的疑难排解信息。

- **BizTalk Document Tracking** 当文件通过服务器时，这个使用者接口会追踪文件和监控文件活动，文件追踪对于两种人来说是十分重要的：商业分析师和系统管理员。商业分析必须仰赖于追踪文件和监控文件状况、分析并且开发内部的商业策略，藉由萃取和储存重要文件中的使用者定义数据，系统分析可以追踪和分析相关于商业作业的详细信息；系统管理员使用追踪文件和活动况状以定义数据追

踪的范围、建立活动记录的设定和疑难排解的分析。

- **BizTalk Workflow Designer** 透过使用 Microsoft Visio 的完美使用者界面，您可以建立工作流程，该接口是将您的商业处理程序以图形化流程的方式呈现。使用图形化接口的拖曳环境以建立商业分析工作流程，并且 Microsoft BizTalk Server 2000 在适当的处理程序中建立适当的插入点使工作流程可以达成目的。这个工作流程附加于您的商业伙伴合约中，并且有文件从商业伙伴处传来时就可以执行这个工作流程。

BizTalk Server 2000 数据处理

BizTalk Server 2000 在 Internet 上不仅可以移动 XML 文件，还可以做更多的应用，该产品最重要的设计理念是管理商业文件之间的协商，它为了达成这个目的，提供了非常丰富的工具以建立这些文件之间的描述和对应。

在下列的情境模拟中，让我们来看看 BizTalk 伺服程序是如何使用于三家公司之间整合订单流程，其中的角色描述如下：

- Toi Carz Corp.是一家生产玩具汽车的制造商，Toi Carz 执行 Microsoft Commerce Server 2000 管理它的 Internet 商业，它使用 Great Plains 软件为其账户系统，也

执行 BizTalk Server 2000 管理它的商业伙伴关系。

- Toy Car Parts, Inc.是玩具汽车零件的配销商和进口商，并且也是 Toi Carz 的供货商之一。因为逐年增加供需关系，Toy Car Parts 有许多大规模的旧型应用程序，某些更已经用了数十年。该公司执行 SAP R/3 系统以管理它的企业资源规划（ERP）环境，也执行 BizTalk Server 2000 管理它的商业伙伴关系。

- Mehtap Enterprise 是玩具汽车引擎的制造商，该公司同时也是和 Toy Car Parts, Inc. 有商业关系的制造商。Mehtap Enterprise 在 1980 年代就安装了 EDI 系统，并且从那个时候该开始就没有升级过。为了处理 Mehtap Enterprise 的问题，我们必须能够建立和读取 EDI 文件。

下订单

在本范例中，Toi Carz 中的一个员工将会下订单以订购 10 个 Mehtap Enterprise 的引擎，它的配销商－Toy Car Parts, Inc.有部份零件的库存，这个交易如图 12-9 所示并且在表 12-1 中有详细的说明。

步	说明

骤	
1	Toi Carz 中的一个员工下订单，该订单由 Commerce Server 2000 处理。
2	Commerce Server 2000 传送一个请求至 BizTalk Server 2000。
3	BizTalk Server 2000 传送一个讯息至 Great Plains 账户系统。
4	产生了一个 Great Plains 格式的采购订单，并且传送至 BizTalk Server 2000。
5	BizTalk Server 2000 解译 Great Plains 的采购订单为 BizTalk 文件，并且透过 Internet 使用 HTTP 传送该文件至 Toy Car Parts 的 BizTalk Server。
6	Toy Car Parts 使用 SAP R/3 系统，其所需的文件是 IDoc 格式，BizTalk Server 2000 解译 BizTalk 讯息为 IDoc 采购订单格式，并且传送至 SAP 系统。
7	该 SAP 系统会响应一个 IDoc 采购订单确认，并且传回 BizTalk Server。
8	Toy Car Parts 的 BizTalk Server 会转换该确认讯息为 BizTalk 文件，并且传回至 Toi Carz 的 BizTalk Server。
9	Toi Carz 的 BizTalk Server 会透过内部系统传送一个确认讯息（ack）至 Commerce Server 2000。
10	Commerce Server 2000 通知请求发出者配销商已经接收了该订单，并且配销商目前正在处理该订单。

表 12-1 图 12-9 中的说明，其中显示了使用 BizTalk Server 的两间公司之间的数据流

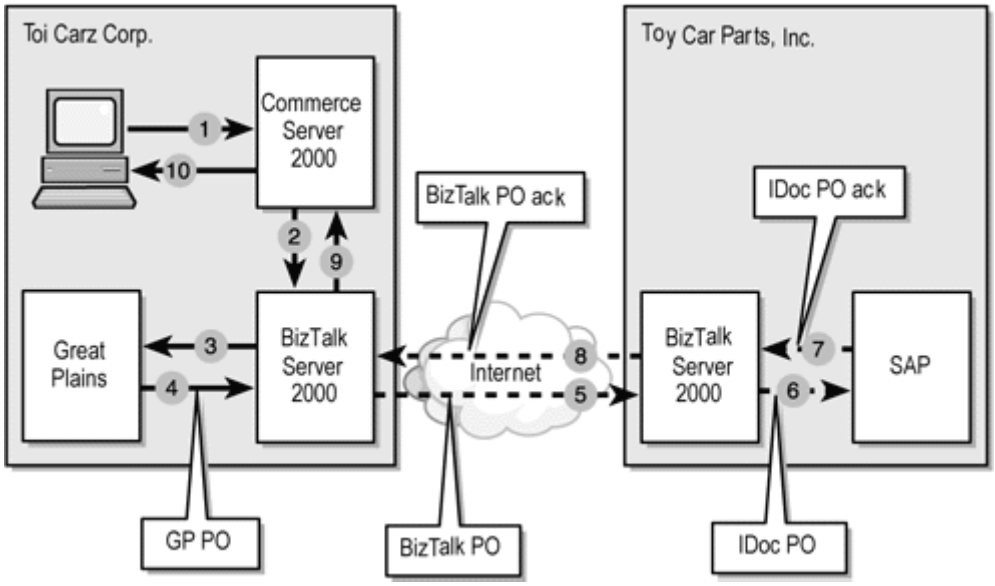


图 12-9　下订单给配销商

检查库存量与向制造商订货

Toy Car Parts 需要看看它是否有足够的库存量以符合订单的需求，图 12-10 显示了 Toy Car Parts 与它的供货商通讯的过程，表 12-2 说明了图 12-10 的步骤。

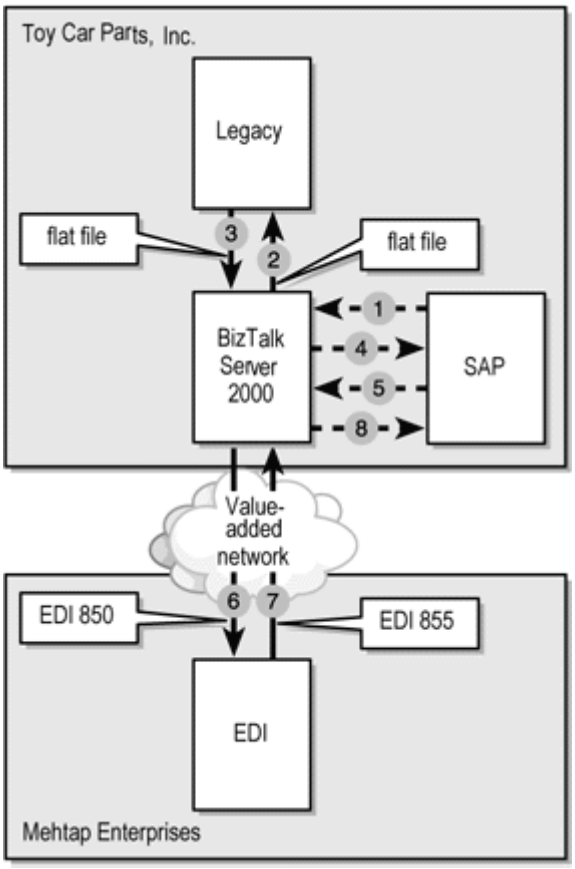


图 12-10　在配销商和制造商之间的通讯

步骤	说明
1	SAP 系统传送一个 IDoc 的请求文件至 BizTalk Server 2000，请求 BizTalk Server 2000 查询旧型库存系统的库存量。
2	BizTalk Server 2000 转换这个 IDoc 文件为一般档案（flat file），该档案用于和旧型库存系统通讯。请您注意很重要的一点是，SAP 系统并无法使用一般档案格式通讯；而库存系统也无法使用 IDoc 通讯。因此 BizTalk Server 2000 提供了两种格式之间的对应。
3	旧型的库存系统响应库存中只有三个零件。

4	BizTalk Server 2000 再次解译该响应，并且传送并且传送一份 IDoc 文件至 SAP 系统。
5	SAP 系统记录该请求最初是要订购十个零件，并且知道了库存还有三个零件，因此它只会建立订购七个零件的 IDoc 文件，该订单会传送至 BizTalk Server 2000。
6	因为制造商只接受 EDI 850 订单文件，BizTalk Server 2000 会将 SAP IDoc 请求转换为 EDI 850 文件，并且传送该文件至 Mehtap Enterprise 连接的增值（value-added）网络。
7	在 Mehtap Enterprise 的 EDI 系统会处理该请求，并且会传回一个 EDI 855 Acknowledge 文件，因为 BizTalk Server 2000 做了所有的转换，所以 Mehtap Enterprise 并不需要了解我们使用 BizTalk 方式通讯。
8	BizTalk Server 2000 会转换 EDI 850 Acknowledge 为 IDoc 格式，并且将其传送给 SAP 系统。

表 12-2　图 12-10 的说明

传送确认讯息给请求发出者

最后一个步骤是传送一个确认讯息给 Toi Carz 指出订单已经完成订购，并且提供了适当的交货日期。这个动作是藉由成交通知（advance ship notice，ASN）完成，图 12-11 显示了该过程，而表 12-3 说明图 12-11 中的步骤。

您应该仔细观察这个情境模拟过程中的某些重点－第一点，这个过程并不需要任何现存的应用程序（在本范例中，应用程序包含 Great Plains、SAP、旧型库存系统或制造商系统）变更它们原

来的工作方式。BizTalk Server 扮演一个通用的解译器－从一个来源处读取一个请求、并且藉由双方合约中的规则解译这些请求，并且将这些请求转换为适当的输出。

第二点，Internet 可以用于两个不同位置的交易，这表示任何连上 Internet 的公司都可以成为我们商业中的一部份（只要双方存在任何契约）。

第三点，如果您公司的增值网络已经上线并且已经正常工作很久，您可以使用增值网络代替 Internet。

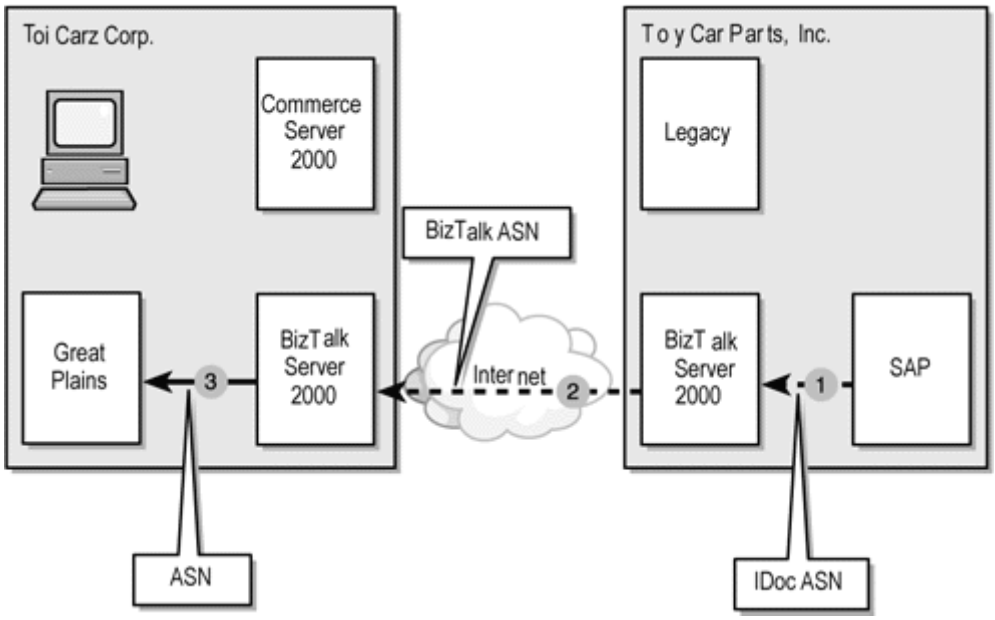


图 12-11　传送成交通知回到发出请求的公司

行	说明

1	SAP 系统产生了 IDoc 格式的 ASN，并且传送它至 BizTalk Server 2000。
2	在 Toy Part Cars 的 BizTalk Server 2000 转换这个 ASN 为 XML 文件，并且传送该文件至 Toi Carz BizTalk Server。
3	在 Toi Carz 的 BizTalk Server 2000 转换该 XML 文件为一 ASN，这个 ASN 兼容于 Great Plains 软件，此时会更新 Great Plains 系统，并且 Toi Carz 的员工可以查询更新完成的数据库。

表 12-3　说明图 12-11 中的步骤

附录 A BizTalk Framework 2.0 草案：文件和讯息规格

草案规格摘要

本草案规格提供了 BizTalk Framework 2.0 概念性基础结构的一般性概述，其中包含 BizTalk Document 和 BizTalk Message，同时也提供了建构 BizTalk Document 和 Message 的详细规格，以及 BizTalk Document 和 Message 在数种 Internet 标准传输通讯协议上相关于安全性传输的论点。

内容

1. 简介
2. 规格的范围和演进
3. 参考数据
4. BizTalk 概念
5. BizTalk 文件结构
6. BizTalk 文件数据主体
7. BizTalk 文件标头项目
8. BizTalk 文件的可信赖传递
9. 包含附件的 BizTalk 文件

10. 传输方式连结

11. BizTalk 文件架构

1 简介

现今的 Internet 安全性传输通讯协议已经非常成熟，而且这些通讯协议也跨网络、硬件和软件平台联合了各式各样的支持，这个成熟的技术让企业可以开发出新的方式来促进有效率和自动化的互动。这些互动可以发生于企业内部商业产品线之间；生产和知识管理应用程序之间；而您的客户和商业合作伙伴就可以使用这些应用程序和您的组织发生有效率的互动。

要在应用程序之间跨越商业的界限好让这么有效率、自动化的互动发生，是一个很大的问题，因为必须考量到在数个组织之间通讯的安全性和可信度问题。

这些跨越商业界限互动的问题包含下列各点（但是不受限于下列各点）：

- 缺乏足够的弹性和丰富的通用语言在应用程序或商业组织之间指定、封装、发布和交换结构化和非结构化的信息。

- 缺乏足够的弹性和丰富的通用语言能够在应用程序或商业组织之间指定、封装、发布和执行转换规则，好将信息格式从一种转换至另一种。

- 缺乏中间层的应用程序通讯协议以便在应用程序或商业组织之间可以自动化的互动。

延伸标记语言（eXtensible Markup Language，XML）和基于 XML 的架构语言提供了低阻碍的强大技术，这种语言提供了一种跨平台的方式让应用程序和商业合作伙伴之间描述和交换结构化的信息。

因此有许多的产业都开始采纳 XML 和基于 XML 的架构来指定产业中的语法和内容模式，这些架构也被广泛的发布和建置以促进应用程序和商业之间的顺利通讯，XML 的广泛支持让许多的协力厂商开始致力于开发 XML 架构的解决方案。然而，并没有一个单一的解决方案可以符合所有复杂环境的部署需求。

BizTalk Framework 让您可以不用担心平台和技术的因素，提供企业组织之间的相互操作性，它提供了一种可以在应用程序和商业组织之间设计和开发用 XML 通讯的解决方案。这个规格建立了标准并且融合了 Internet 技术，例如 HTTP、MIME、XML 和 SOAP，BizTalk Framework 的后续版本也会在 XML、Internet 技术和讯息标准方面的功能性上适当的增强。

有一点观念请您特别注意，BizTalk Framework 的定位并非要涵盖整个 B2B 电子商务的所有面像，BizTalk Framework 只会提供基本的 B2B 电子数据交换通讯机制。

2 规格的范围和演进

这个规格提供了 BizTalk Framework 概念性基础结构的概述，其中包含 BizTalk Document 和 BizTalk Message 的基本标记语法，然后提供了建构 BizTalk Document 和 Message 的详细规格，以及 BizTalk Document 和 Message 在数种 Internet 标准传输通讯协议上相关于安全性传输的论点，说明如下：

BizTalk Document 的结构和内容遵循数种规则以提供丰富的功能性和语法，本规格说明了 BizTalk Document 和语法的各个方面：

- BizTalk Document 的整体结构。

- BizTalk 标头以供文件的路由、属性、类别和程序管理使用。

- 需要可信赖传递的 BizTalk Document 的结构和控管。

当您使用 BizTalk Framework 建置解决方案的时候，您必须使用特定的传输通讯协议、程序代码撰写方式和安全性的机制以确保讯息能够安全且顺利的传递。本规格说明了下列的机制：

- Internet 通讯协议的传输连结（只有 HTTP，预计新增 SMTP、FTP）。

- MIME 传送的程序代码撰写方式和附件封装方式。

- 基于 S/MIME 和 Public-Key Cryptography System（PKCS）的签署和加密。

本规格试图定义讯息在 **BizTalk Framework 2.0 兼容伺服程序** （BizTalk Framework 2.0 Compliant Servers）之间的互动性，在本规格中称为 **BFC Servers** 。

与 BizTalk Framework 1.0 之间的相关性

BizTalk Framework 2.0 规格保留了 BizTalk Framework 1.0 规格的主要部分，BizTalk Framework 2.0 包含下列新功能：

- 传输连结（只有 HTTP，预计新增 SMTP、FTP）。

- 可信赖的讯息传递。

- MIME 附件的程序代码撰写方式。

- 基于 S/MIME 和 PKCS 的安全性（预计新增）。

还有，BizTalk Framework 2.0 也间接的影响了许多近期开发的新标准，例如下列：

- SOAP Version 1.1
- XML Schema Part I: Structures
- XML Schema Part II: Data types
- XML-Signature Syntax and Processing

因为 BizTalk Framework 2.0 是 SOAP 1.1 的延伸，所以 SOAP 1.1 几乎完全受到其影响，主要的修订部分是保留或新增有意义的语法、命名方式，以及许多 BizTag 的结构和本规格中的新功能。

在 BizTalk Framework 2.0 中有许多 BizTag 的语法定义要比 BizTalk Framework 1.0 中所定义的更具多功能性。在 BizTalk Framework 1.0 中描述的 BizTalk Document 结构和内容尽可能的被保留下来，但是标准规定的精确性和一致性语法比之前的语法有更高的优先权以提供更为正确的讯息结构。

版本模式

SOAP 1.1 规格中的 **SOAP ENV-:mustUnderstand** 属性会指定如果需要被解译的标头项目有错误的名称空间，则 BFC 伺服程序将会传回 **SOAP ENV-:mustUnderstand** 的错误。

在第 10 节中的 HTTP 连结内容清楚的说明这个错误讯息也许会在 HTTP 的响应中传回。然而，如果讯息是异步处理，HTTP 的响应也许会是 **202 accepted**，并且错误也是异步传回，请参阅 第 10 节 中更进一步的传输通讯协议连结说明。

3 参考数据

3.1 标准规范的规格

每一份 BizTalk Framework 文件都是以 Internet 的标准为其规范的参考数据，然而这些规范的参考数据也可能需要重新修订以符合 BizTalk Framework 目的的需求。一般而言，这些标准的参考数据都是经过授权的。每一分努力都是为了避免这些参考数据与它们在 BizTalk Framework 规格中的使用概念不一致，然而，如果确实发现了使用概念不一致的状况，这些参考数据必须提供正确的释义，而且 BizTalk Framework 的规格就需要校正。

下列各项即为本规格的标准规范：

- Extensible Markup Language（XML）1.0

- Simple Object Access Protocol（SOAP）Version 1.1

- Namespaces in XML

- Uniform Resource Identifier（URI）: Generic Syntax

- ISO 8601: Representations of dates and times

- Hypertext Transfer Protocol－HTTP/1.1

- XML Media Types

- MIME Part One: Format of Internet Message Bodies

- MIME Part Two: Media Types

- MIME Part Three: Message Header Extensible for Non-ASCII Text

- MIME Part Four: Registration Procedures

- Content-ID and Message-ID Uniform Resource Locators

3.2 非标准规范的规格

下列各项即为非标准规范的规格：

- XML-Data Reduced（XDR）
- XML Schema Part 1: Structures
- XML Schema Part 2: Data types
- XML-Signature Syntax and Processing

4 BizTalk 概念

4.1 术语

本文件使用特定的 BizTalk 术语如下：

- **BizTalk 基础结构兼容伺服程序**（BizTalk Framework Compliant（BFC）Server）

  BFC 伺服程序即为 BizTalk Framework 规格中定义用来提供讯息处理功能的服务集合。

- **应用程序** （Application） 应用程序即为 **商业数据和逻辑** 储存和执行的地方，并且会与 BFC Server 通讯。

- **商业文件** （Business Document） 商业文件即为格式正确的 XML 文件，其中包含了商业交易的数据。这些交易数据也许是采购订单、发票、销售预测或任何其它的商业信息。一个或多个商业文件就形成了 **BizTalk 文件** 的数据主体，BizTalk Framework 并没有规定商业文件的内容和架构（Schema）。

- **架构** （Schema） 架构即为用于描述 XML 文件类别的内容和结构的中继数据。这个正式的描述方式是应用程序开发人员建立处理商业文件的系统所使用；或是由剖析器在执行期间辨识商业文件是否合法有效所使用。

  商业组织也许会在 BizTalk Schemas Library 中发布它们的架构。

说明

商业文件的架构没有包含任何的 BizTag，架构只会在需要支持商业交易的时候才会包含 BizTag，在架构建置上的一般需求和指导方针定义于 BizTalk Schema Guidelines 中。

- **BizTalk 文件** （BizTalk Document） BizTalk 文件即为 SOAP 1.1 讯息，该讯息的数据主体包含商业文件，同时标头包含特定的 BizTalk 标头项目。

- **BizTag** BizTag 即为 XML 卷标的集合（包含强制性和选择性的），其功能为用于指定商业文件的控制。更精确的说，BizTag 即为本规格中定义的元素和属性，并且用于在 BizTalk 文件中建构 SOAP 标头项目，这些讯息会由 BFC 伺服程序处理，或者由其它辅助文件交换的应用程序所处理。

- **BizTalk 讯息** （BizTalk Message） BizTalk 讯息是在 BFC 伺服程序之间的交换单位，BizTalk 讯息用于传送 BizTalk 文件，以及任何相关的档案。BizTalk 讯息必须要包含主要的 BizTalk 文件，该文件必须以 BizTalk Framework 中的讯息语法定义。讯息中也许会包含一或多个附件、格式正确的 XML 文件等等。BizTalk

讯息的结构取决于讯息的传输方式，并且常常会包含传输方式所指定的标头。

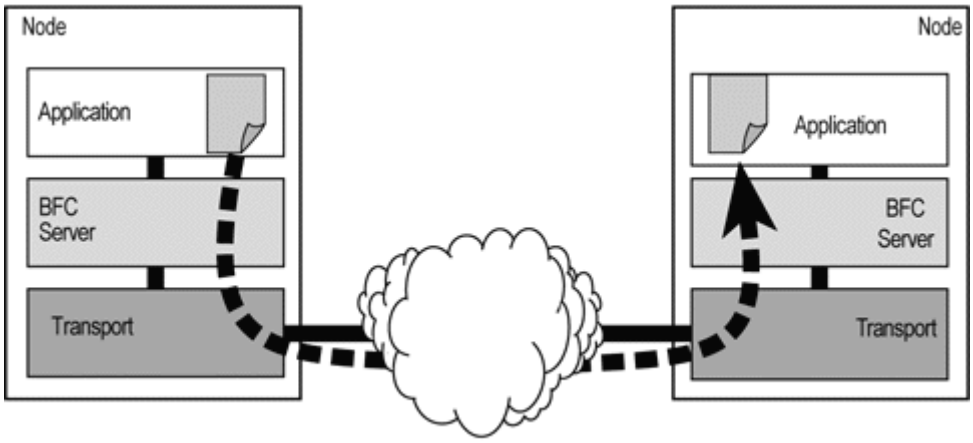
- **传输方式**（Transport） 在 BFC 伺服程序之间的 BizTalk 讯息实际交换方式必须以传送实体讯息的通讯机制为前提，我们使用 **传输方式** 这个术语来表示这个机制。传输方式可能会取决于它的特性、档案传输协议，例如 HTTP 和 SMTP 而有所不同。

- **附件**（Attachment） 附件通常不会是 XML 档案，也不会是 BizTalk 文件数据主体中的商业文件相关信息；附件通常是图形、大型的压缩档案或任何其它信息格式，或是不适用于商业文件的内容。

4.2 逻辑层

BizTalk Framework 的逻辑应用模式是以 **层（Layer）** 的方式建置，本节中所描述的 **层** 使用图示和解说就可以让您容易的了解。虽然 BizTalk Framework 规格定义用以传递讯息的 BizTalk 讯息规格和通讯协议，但是您也可以使用逻辑层的概念以达成相同的讯息传输目的。这些逻辑层中包含应用程序、BFC 伺服程序和传输方式，应用程序藉由 BFC 伺服程序来回传递商业文件以和其它应用程序通讯，而 BFC 伺服程序的通讯是建构于各种各样的传输通讯协议，例如 HTTP、

SMTP 和 Microsoft Message Queue（MSMQ），BizTalk Framework 并没有规定一定要使用哪一种通讯协议，因此每一个节点可以独立建置其使用的传输方式。



应用程序负责产生商业文件和任何要传输的附件，并且将它们传送给 BFC 伺服程序，而将商业文件封装入 BizTalk 文件的工作也许由剩下的应用程序或 BFC 伺服程序完成，这取决于 BFC 伺服程序的建置方式。伺服程序处理文件和任何附件，并建构适合于某种传输通讯协议的 BizTalk 讯息。BFC 伺服程序使用包含于 BizTag 中的信息以决定特定传输通讯协议的目标地址，然后伺服程序便会掌控目标 BFC 伺服程序的讯息传输层。在商业应用程序、BFC 伺服程序和传输层之间的接口都具备其明确的建置方式。

5 BizTalk 文件结构

下列的范例是一个简单的 BizTalk 文件：

```
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
                envelope/"

xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Header>
        <dlv:delivery SOAP-ENV:mustUnderstand="1"

xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"
                xmlns:agr="http://www.trading-agreements.org/types/">
            <dlv:to>
                <dlv:address xsi:type="agr:department">
                    Book Order Department</dlv:address>
            </dlv:to>
            <dlv:from>
                <dlv:address xsi:type="agr:organization">
                    Booklovers Anonymous</dlv:address>
            </dlv:from>
```

```
        </dlv:delivery>
        <prop:properties SOAP-ENV:mustUnderstand="1"
            xmlns:prop="http://schemas.biztalk.org/btf-2-0/
            properties">
            <prop:identity>
            uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</prop:identity>
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:
            expiresAt>
            <prop:topic>http://electrocommerce.org/purchase_order/</
            prop:topic>
        </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <po:PurchaseOrder
            xmlns:po="http://electrocommerce.org/purchase_order/">
            <po:Title>Essential BizTalk</po:Title>
        </po:PurchaseOrder>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

BizTalk 文件由包含下列各项的标准 SOAP 1.1 讯息组成：

- 特定的应用程序商业文件（本范例是采购订单），并且有应用程序定义的名称空间，包含于 SOAP 讯息的数据主体（body）中。

- 特定的 BizTalk **<delivery>** 和 **<properties>** SOAP 标头项目，其由标准的 BizTag 名称空间中定义的 BizTag 所建构，并且使用本规格中定义的架构和语法。

一般而言，SOAP 讯息的数据主体会构成包含数个相关商业文件的 BizTalk 文件，以及 SOAP 讯息的标头也会包含数个特定的 BizTalk 标头项目。 **SOAP-ENV: mustUnderstand** 属性值为「1」时，则表示接收该文件的目标商业实体必须了解并且正确的处理标头项目。如果标头项目无法了解（解译），文件的处理就会以错误中止。

标准的 BizTag 名称空间（http://schemas.biztalk.org/btf-2-0/）中定义了所有的 BizTag，BizTag 名称空间语法的有效范围永远受限于最外围 BizTalk 文件中的标头。换句话说，如果一份 BizTalk 文件完全包含于另一份 BizTalk 文件的数据主体中，内层文件的 BizTag 就会闲置并且不会发生任何作用，而会以外层文件的规则进行处理。

路由标签 **to** 和 **from** 会在下列各节中有详细的说明，它们通常会使用来源和目的地址的商业实体名称，而不会使用传输方式地址，例如 HTTP URL。地址内容的解译和形式是由 **xsi:type** 属性指定。

6 BizTalk 文件数据主体

SOAP 讯息的 **<Body>** 元素构成了包含商业文件的 BizTalk 文件，一般而言，BizTalk 文件也许会传递数个相关的商业文件（举例来说，采购订单、送货商名称，以及该订单的送货位置）。

相关的商业文件通常会共享内容，SOAP 有一个简单的机制可以参考多重资料－即为使用 XML ID 属性和相关的 URI。以采购订单的范例来说，采购订单和送货信息都会参考到书的信息，下列 BizTalk 文件的 **<Body>** 元素表达了如何使用 SOAP 规则来参考共享的信息。

```
<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
             xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Header>
        <!--headers omitted for brevity -->
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <po:PurchaseOrder
```

```
        xmlns:po="http://electrocommerce.org/purchase_order/">
        <po:item href="#theBook"/>
        <!--and other purchasing information -->
      </po:PurchaseOrder>
      <ship:shippingInfo xmlns:ship="http://electrocommerce.org/
      shippingInfo/">
        <ship:content href="#theBook"/>
        <!--and other shipping information -->
      </ship:shippingInfo>
      <book xmlns="http://electrocommerce.org/bookInfo/"
            id="theBook " SOAP--ENC:root="0">
          <Title>Essential BizTalk</Title>
          <!--and other book information -->
      </book>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

范例也显示了当您使用这个技术时会发生的问题，我们想要目标的商业实体会检视包含两个商业文件的 BizTalk 文件：采购订单和送货信息。然而，SOAP 讯息的 **<Body>** 包含了三个子元素，

因此我们需要一种方式从商业文件中辨识出采购订单和送货信息的元素，以排除掉被参考的元素。

SOAP 提供了 **SOAP-ENC:root** 属性，该属性可以指定某元素并非独立的实体。在 BizTalk 文件中 **<Body>** 元素的每一个直接子元素都是商业文件，除非该子元素中包含 **SOAP-ENC:root** 属性的值为「0」。

7 BizTalk 文件标头项目

本节说明 4 个 BizTalk-specific SOAP 标头项目，对于文件的路由和传递、文件的识别和内容等等来说，这些标头项目都扮演重要的角色。下列的表格列出了 4 个用以标记标头项目和内容的 BizTag，每一个标头项目会在下列各节更详细的说明。

标签名称	是否强制性	种类	型别	发生次数
delivery	是	元素	complexType	一次
Properties	是	元素	complexType	一次
Manifest	否	元素	complexType	一次
Process	否	元素	complexType	一次

7.1 文件路由和传递

文件的路由是藉由 SOAP 标头项目的 **<delivery>** BizTag 指定，这个项目包含了 BizTalk 文件的来源和目的相关信息，下列的范例是 **<delivery>** 标头项目的扩充版本，其中还增加了信赖传递的程序区段：

```
<dlv:delivery SOAP-ENV:mustUnderstand="1"


xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
            xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"
            xmlns:agr="http://www.trading-agreements.org/types/">
    <dlv:to>
        <dlv:address xsi:type="agr:department">
            Book Order Department</dlv:address>
    </dlv:to>
    <dlv:from>
        <dlv:address xsi:type="agr:organization">
            Booklovers Anonymous</dlv:address>
    </dlv:from>
    <dlv:reliability>
        <dlv:sendReceiptTo>
```

```
            www.we-love-books.org/po/confirmations
        </dlv:sendReceiptTo>
        <dlv:receiptRequiredBy>
            2000-05-14T08:00:00+08:00
        </dlv:receiptRequiredBy>
    </dlv:reliability>
</dlv:delivery>
```

在顺利的处理一份 BizTalk 文件过程中，BFC 伺服程序必须要解译和处理 **<delivery>** 标头项目及其所有的内容，同时这也是强制性的。 **<delivery>** 元素也一定要包含 **SOAP-ENV:mustUnderstand="1"** 属性。下列的表格列出了用以建构 **<delivery>** 标头项目的 BizTag 子元素和它们的内容。

标签名称	是否强制性	种类	型别	发生次数
to	是	元素	complexType	一次
from	是	元素	complexType	一次
reliability	否	元素	complexType	一次

<to> 卷标包含目标商业实体的 BizTalk 文件规格，该元素包含 **<address>** 子元素，并且只发生一次。

<from> 卷标包含来源商业实体的 BizTalk 文件规格，该元素包含 **<address>** 子元素，并且只发生一次。

<adderss> 卷标包含商业实体的识别（以字符串形式），**<adderss>** 元素必须使用 **xsi:type** 属性， **xsi:type** 属性的值表示地址的类别、也表示地址字符串形式的允许结构。本范例中包含数个类别，其中包含组织名称、URI 参考等等。

<reliability> 卷标是一个选择性的元素，其中包含必要的信息以便执行可信赖的传递（BizTalk 文件中由 **<reliability>** 标签包含的区段）。如果存在 **<reliability>** 元素，目标商业实体的 BFC 伺服程序就必须基于接收的讯息传送回条至已知的 URL。 **<reliability>** 元素包含下表中的两个子元素。

标签名称	是否强制性	种类	型别	发生次数
sendReceiptTo	是	元素	uriReference	一次
receiptRequiredBy	是	元素	timeInstant	一次

<sendReceiptTo> 卷标包含 URL，该 URL 会指定传送地址（一般都是来源商业实体）为接收者（BizTalk 文件必须传送至该地址）。

<receiptRequiredBy> 卷标包含一个实时时间信息，该信息会指定接收者接收讯息的绝对时间，并且该 BizTalk 文件一定要由来源商业实体的 BFC 伺服程序接收。

7.2 文件识别和内容

文件的识别信息和其它内容是藉由 SOAP 标头项目的 **<properties>** BizTag 指定，范例如下。

```
<prop:properties SOAP-ENV:mustUnderstand="1 "
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
        <prop:identity>
        uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6
        </prop:identity>
        <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
        <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
        <prop:topic>
        http://electrocommerce.org/purchase_order/
        </prop:topic>
    </prop:properties>
```

在顺利的处理一份 BizTalk 文件过程中，BFC 伺服程序必须要解译和处理 **<properties>** 标头项目及其所有的内容，同时这也是强制性的。 **<properties>** 元素也一定要包

含 SOAP-ENV:mustUnderstand="1" 属性。下列的表格列出了建构 <properties> 标头项目的 BizTag 子元素和它们的内容。

标签名称	是否强制性	种类	型别	发生次数
identity	是	元素	uriReference	一次
sendAt	是	元素	timeInstant	一次
expiresAt	是	元素	timeInstant	一次
topic	是	元素	uriReference	一次

<identity> 标签是一个 URI 参考，可以唯一的辨识 BizTalk 文件，目的是为了记录、追踪、错误控制或其它文件处理需求。 <identity> 标签必须全球唯一，这是可以达到的。举例来说，使用 Universally Unique Identifiers（UUIDs），如上例所示；或是使用加密杂凑算法，例如 MD5 套用到商业文件上。

<sentAt> 标签用于传送文件的时间戳记，这个时间戳记会反应出 <properties> 元素建立的时间。

<topic> 标签包含一个 URI 参考，用以唯一的识别所有 BizTalk 文件的目的。 <topic> 标签可以用于路由，而路由的路径是基于该卷标所指定的相关主题（藉由基于主题寻址的 出版/订阅 模式），并且用来辨识 BizTalk 文件的一致性。

expiresAt 标签是文件过期的时间戳记，超过此时间点的 BizTalk 文件视为过期，因此目标的商业实体就不会处理该文件。指定过期时间是很重要的，在分布式系统中就可以根据这个标签提供时间同步化时，错误发生的最低限度。

7.3 文件类别

文件的类别信息是藉由一个选择性的 SOAP 标头项目指定－ **<manifest>** BizTag；它会参考 BizTalk 讯息中主要 BizTalk 文件装载的商业文件信息和其它的附件，例如图形或二进制数据（这些数据也可视为 BizTalk 讯息的一部份）。BizTalk 讯息中附件的参考结构详细信息取决于 MIME 编码规则，范例如下：

```
<fst:manifest
        xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest">
            <fst:reference fst:uri="#insurance_claim_document_id">
                <fst:description>Insurance Claim</fst:description>
            </fst:reference>
            <fst:reference fst:uri="CID:claim.tiff@claiming-it.com">
            <fst:description>
        Facsimile of Signed Claim Document</fst:description>
        </fst:reference>
```

```
            <fst:reference fst:uri="CID:car.jpeg@claiming-it.com">
            <fst:description>Photo of Damaged Car
        </fst:description>
        </fst:reference>
</fst:manifest>
```

<manifest> 的目的是提供 BizTalk 讯息内容的完整类别以提供一致性的检查、完整性和验证机制，例如数字签署。到目前为止，类别的结构尽可能的相似于 XML 的数字签署结构。

<manifest> 标头项目并不是强制性的，当该标签存在时， **<manifest>** 标头项目必须将所有在 BizTalk 文件中的商业文件分类。在 BizTalk 讯息的主要 BizTalk 文件中，同时也必须将任何附件分离出来。

<manifest> 元素会接着 **reference** 元素，如下表所示：

标签名称	是否强制性	种类	型别	发生次数
reference	是	元素	complexType	一次

<reference >卷标包含 URI 属性，并包含描述文字如下表：

标签名称	是否强制性	种类	型别	发生次数
Uri	是	属性	uriReference	一次

Description	否	元素	字符串	一次

uri 属性值必须为一个 URI 参考用以解析 **reference** 元素中包含的资源，这个属性的一般值如下：

- **#id** 形式的区段识别项（fragment identifier）会解析 BizTalk 文件中的商业文件（其中包含 ID 属性值的区段）。

- **CID:content-id-value** 形式的 Content-ID URL 会解析 BizTalk 讯息中装载的 MIME 附件。

- 解析资源的 URL，例如视为附件的大型档案，但是内容并非装载于 BizTalk 讯息中。

- 请注意这个属性在 XML 数字签署中是选择性的，但是在本规格中是强制性的。

description 卷标是商业文件或附件的文字描述，内容是混合的（文字中包含或不包含 XML 标记）。

7.4 处理程序管理

处理程序管理的信息是藉由一个选择性的 SOAP 标头项目指定－ **<process>** BizTag，其中包含商业处理的信息，范例如下：

```
<prc:process SOAP-ENV:mustUnderstand="1"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:prc="http://schemas.biztalk.org/btf-2-0/process">
    <prc:type>purchasing:Book_Purchase_Process</prc:type>
    <prc:instance>purchasing:Book_Purchase_Process#12345</prc:
    instance>
    <prc:handle>port:po_receiver</prc:handle>
</prc:process>
```

当存在 **<process>** 时，为了顺利处理一份 BizTalk 文件的过程中，BFC 伺服程序必须要解译和处理 **<processs>** 标头项目及其所有内容，同时这也是强制性的。 **<process>** 元素也一定要包含 **SOAP-ENV:mustUnderstand="1"** 属性。下列的表格列出了用来建构 **<process>** 标头项目的 BizTag 子元素和它们的内容。

标签名称	是否强制性	种类	型别	发生次数
Type	是	元素	uriReference	一次
instance	是	元素	uriReference	一次
handle	否	元素	uriReference	一次

<type> 标签包含 URI 参考以便指出商业处理的类型。举例来说，采购一本书的处理程序，这是一个在多个商业合作伙伴中多个 BizTalk 文件交换的对应式（pattern），对应式中定义了「游戏规则」，并且可以重复使用多次。

<instance> 标签包含 URI 参考以唯一的识别特定商业处理的执行个体。

<handle> 卷标是提供更进一步信息的 URI 参考，这些信息也许会用以识别某步骤或商业处理执行个体中的项目点。

8 BizTalk 文件的可信赖传递

BizTalk 促进电子商务中的异步文件交换，以及企业应用程序整合（Enterprise Application Integration）来保证讯息一定会传达、错误侦测和报告必要信息等等。Internet 的高效能讯息中间层解决方案终于脱颖而出，并且用于达成可信赖传递的目的。然而，十分重要的一点就是：必须提供一个更简单的解决方案让 BizTalk 文件可以藉由轻易的建置 BFC 伺服程序达成信赖传递的目的，这个解决方案基于两个简单的概念：

- 文件收据（Document receipts）

- 幂级传递（Idempotent delivery）

整体目的是确保 BizTalk 文件传递的定义结果，下列各项说明了可信赖传递的摘要：

- 收据会提供一种方式让来源商业实体的 BFC 伺服程序可以确认它自己的 BizTalk 文件被接收并且被接受。

- 相同的 BizTalk 文件多次重传的可能性，目标商业实体的 BFC 伺服程序也许会套用幂级的传递规则（就像在队列中一样），并且删除重复的文件。

- 如果来源商业实体的 BFC 伺服程序在特定的时间周期中（由 **receiptRequiredBy** 元素所指定）没有接收到传递的收据，就会产生传递错误的报告。

本节最后描述收据文件的结构，以及在 BizTalk 文件进行可信赖传输时，如果网络拥挤会造成来源和目的商业实体有什么行为。

8.1 收据的结构和内容

BizTalk Framework Receipt 是一个 SOAP 1.1 讯息，其中至少包含 **receipt** 和 **properties** 标头项目，以及一个空数据主体（empty body），如下列范例所示。收据并非 BizTalk 文件，它是一个确认 BizTalk 文件收据的 SOAP 讯息。

清晰的收据内容会识别 BizTalk 文件已经被确认（藉由全球性的唯一识别码），文件在目标商业实体何时被接收的时间戳记是由 **<receipt>** 标头项目提供。

下列的收据是对应于本规格中最初的范例：

```
<SOAP-ENV:Envelope
              xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/
              envelope/">
  <SOAP-ENV:Header>
       <rct:receipt SOAP-ENV:mustUnderstand="1"
            xmlns:rct="http://schemas.biztalk.org/btf-2-0/receipt">
            <rct:receivedAt>2000-05-15T04:08:10-05:30</rct:
            receivedAt>
       </rct:receipt>
       <prop:properties SOAP-ENV:mustUnderstand="1"
            xmlns:prop="http://schemas.biztalk.org/btf-2-0/
            properties">
```

```
            <prop:identity>
            uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</prop:identity>
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:
            expiresAt>
            <prop:topic>http://electrocommerce.org/purchase_order/</
prop:topic>
        </prop:properties>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
```

除了 **<receipt>** 标头项目，收据也必须包含来自被确认文件的 **<properties>** 标头项目。

在顺利的处理一份 BizTalk 文件过程中，BFC 伺服程序必须要解译和处理 **<receipt>** 标头项目及其所有内容，同时这也是强制性的。 **<receipt>** 元素一定要包含 **SOAP-ENV:mustUnderstand="1"** 属性。下列的表格列出了用以建构 **<receipt>** 标头项目的 BizTag 子元素和它们的内容。

标签名称	是否强制性	种类	型别	发生次数
receiveAt	是	元素	timeInstant	一次

<receiveAt> 标签即为由该收据的文件被确认的接收时间戳记，在一份 BizTalk 文件的副本被多次接收的情况下，接收的时间戳记也许会反应出第一份副本被接收的时间或是被确认副本被接收的时间。

8.1.1 来源商业实体的行为

一般而言，将会被传送的 BizTalk 文件都会假设存在于来源商业实体的永久性储存区。在 BizTalk 文件被传送之前，强烈建议您先确认该文件存在于永久性储存区（尤其是当文件进行可信赖传输，并且以定义好的结果输出时）。

如果 BizTalk 文件要从来源商业实体进行可信赖传递时，必须新增 **<reliability>** 元素至 **<delivery>** 标头项目中，并且：

- 设定 **<sendReceiptTo>** 子元素的值至适当的传输 URL（收据传送的地址）。
- 设定 **<receiptRequiredBy>** 子元素的值（接收收据的时间界限）。

下列说明即为重新尝试传送的指导方针，这样可以增加顺利传递讯息的可能性。需要注意的是，当重新传送的时候，包含标头项目的 BizTalk 文件内容不能够修改，尤其 **<sentAt>** 属性的内容必须维持一致。

来源商业实体的重新传送行为一般取决于一个特定的参数：重新传送时间间隔。这个参数可以为固定的，或是根据商业合作关系来设定；同时也会有一个最大重新传送次数。重新传送时间间隔参数会不断的尝试，直到下列之一发生：

- 当接受了收据。
- **receiptRequiredBy** 界限值过期。
- 超过最大重新传送次数。

在本程序接近结束时，如果没有接收到收据，并且 **receiptRequiredBy** 界限值过期了，则 BizTalk 文件的传递就失败了。如果发生了失败的情况，一般正常的操作程序就会以适当的方式通知来源应用程序。举例来说，将文件的副本放在无法传递讯息的队列中。

有一点您必须注意，那就是如果重新传送使用的传输方式和第一次传送时的传输方式相同时，那么重新传送尝试就不需要发生。如果传输到目标商业实体（在 **delivery** 标头项目中的 **to** 地址指定）有多个传输的端点可以使用，重新传送时也许会试图使用其它的传输端点。

8.1.2 目标商业实体的行为

一般而言都会假设接收文件后，每一份 BizTalk 文件都会存在于目标商业实体的永久性储存区。在传送收据之前，强烈建议您先确认接收的文件存在于永久性储存区（在进行可信赖传输时）。

目标商业实体进行 BizTalk 文件的可信赖传递时，会根据每一个 **接收的文件** （该文件中的 **<delivery>** 标头项目会存在 **reliability** 元素）传送一个收据。而收据的传送地址即为 **sendReceiptTo** 元素所指定的传送地址，在本例中的「接收的文件」即表示该文件已经被确认为试图传递到目标实体。在 **<expiresAt>** 所指定时间之后所接收的文件会视为过期，并且 **不会** 被接受，当然也就不会传送收据回来源商业实体；在 **<receiptRequiredBy>** 所指定时间之后所接收的文件会视为过期，但是 会 被接受，并且必须传送收据回来源商业实体。

目标商业实体也许会将 BizTalk 文件的传递排入队列中，这个就要取决于商业之间的合作约定。队列传递意味着在目标商业实体接收的 BizTalk 文件一定会被传递至接收的应用程序。

8.2 传递和处理的时限

了解 **receiptRequiredBy** 和 **expriesAt** 元素所指定的时限，以及其分别代表的差异性是很重要的， **receiptRequiredBy** 的内容是 **传递时限** （delivery deadline）； **expriesAt** 的内容是 **处理时限** （processing deadline）。让您辨识两者之间差异性的原因是因为商业上的处理动作通常必须在一定的时间内完成，如果一份 BizTalk 文件所需执行的动作预期要在 4 个小时之内完成，并且文件的传送必须在 12 个小时之内完成，所以该文件的传递时限必须为 8 个小时（而不是 12 个小时），这种设计弹性是供设计实际的商业程序时使用。上述两种时限的语意如下：

- 传递时限即为 BizTalk 文件被目标商业实体所 接受 的时限，文件的接受在第 8.1.2 节中定义，接受的确认讯息必须被来源商业实体接收（根据传递时限指定）。

- 处理时限是 BizTalk 文件在某个时间点之前一定要被处理，如果没有被处理，该文件即视为空值或无效。

下列各项清楚的说明每一个时限过期后会发生什么：

- 当传递时限过期时：

    - 传送端的 BFC 伺服程序会通知传送应用程序，指出没有接收到收据，这个动作会让传送应用程序了解接收端可能没有接收到文件，或者是接收到文件之后没有足够的时间处理文件。

    - 对于接收端 BFC 伺服程序来说并没有任何特别的重要性，接收端的伺服程序会持续接受和确认超过传递时限的文件。这表示也许会产生和接收超过时限的收据。事实上，接收端并不是一定需要知道传递的时限。然而，接收端伺服程序如果不知道何时要传递收据，它也许会因为内部最佳化的理由而延迟传递收据，这样也会引起传送端的非必要复杂性；否则就是让收据有最高的优先权，但是这样会造成无法内部最佳化。传递时限对于接

收端传送收据来说是一个有用的优先权提示，而在中断性（有时联机、有时离线）的联机情况下又特别的有用。

- 当处理时限过期时：

  - 目前的规格对于传送端 BFC 伺服程序并没有特别的建议，在 BizTalk Framework 的未来版本中也许会加入处理完成的确认讯息。如果加入了这个概念之后，在处理时限过期之前，传送端的 BFC 伺服程序会通知传送端应用程序处理确认讯息是否没有接收。这个动作会让传送端应用程序知道接收端应用程序可能没有处理文件（甚至已经接收了收据）。

  - 接收端 BFC 伺服程序将会拒绝所有超过处理时限的文件，这些文件将不会被确认，并且不会被传递到任何的应用程序以执行正常的处理。

9 包含附件的 BizTalk 文件

商议处理常常需要商业文件和附件一起传输，附件通常是一些图形、大型的压缩文件等等，有些附件通常也是二进制格式，本节指出下列各项：

- 一种标准方式以联合主要 BizTalk 文件和一个或多个附件（传输方式为复合 MIME 结构）。

- 附件的 MIME 结构和主要 BizTalk 文件中 **<manifest>** 标头项目之间的关联性。

9.1 复合 MIME 结构

混合内容的 BizTalk 讯息是由主要 BizTalk 文件和一或多个附件组成，混合内容必须以 MIME 结构装载。

下列范例显示了 BizTalk 文件和两个附件以建构汽车保险的所有权，主要 BizTalk 文件包含所有权数据，并且与两个影像文件（Claim.tiff 和 Car.jpeg）一起传输。

```
MIME-Version:1.0
Content-Type:Multipart/Related;
    boundary=biztalk_2_0_related_boundary_example;
    type=text/xml;
    start="<claim.xml@claiming-it.com>"
Content-Description:This is the optional message description.
```

```
--biztalk_2_0_related_boundary_example
Content-Type:text/xml;charset=UTF-8
Content-Transfer-Encoding:8bit
Content-ID:<claim.xml@claiming-it.com>

<?xml version='1.0 ' ??>
<SOAP-ENV:Envelope

xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <!--delivery and properties header entries omitted for brevity
-->
        <manifest
xmlns="http://schemas.biztalk.org/btf-2-0/manifest">
            <reference uri="#insurance_claim_document_id">
                <description>Insurance Claim</description>
            </reference>
            <reference uri="CID:claim.tiff@claiming-it.com">
                <description>Facsimile of Signed Claim Document
                </description>
```

```
            </reference>
            <reference uri="CID:car.jpeg@claiming-it.com">
                <description>Photo of Damaged Car</description>
            </reference>
        </manifest>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <claim:Insurance_Claim_Auto id="insurance_claim_document_id"

xmlns:claim="http://schemas.risky-stuff.com/
Auto-Claim">
        <!--...claim details...-->
        </claim:Insurance_Claim_Auto>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--biztalk_2_0_related_boundary_example
Content-Type:image/tiff
Content-Transfer-Encoding:base64
Content-ID:<claim.tiff@claiming-it.com>
```

```
    ...Base64 encoded TIFF image...

--biztalk_2_0_related_boundary_example

Content-Type:image/jpeg

Content-Transfer-Encoding:binary

Content-ID:<car.jpeg@claiming-it.com>



    ...Raw JPEG image...

--biztalk_2_0_related_boundary_example--
```

<attachment> 元素结构的规则描述了在主要 BizTalk 文件中 **<menifest>** 标头项目的附件。**复合/相关** （multipart/related）媒体类型的使用规则在 RFC 2387 中有详细定义。在 BizTalk 讯息（包含附件）中使用 [复合/相关] 媒体类型的额外规则如下：

- 主要 BizTalk 文件装载于复合/相关结构的根（root）部份。
- 所有 BizTalk 文件的媒体类型是 **text/xml** ， 其定义在 RFC 2376 中。
- 每一个部份，包含根部份都必须有一个 **Content-ID MIME** 标头结构，与 RFC 2045 一致。

- 除了复合/相关媒体类型的必要参数之外，起始参数（在 RFC 2387 中是选择性）一定要存在。

9.2 附件的 Manifest 结构

manifest 标头项目中的 **<reference>** 元素要表示 MIME 附件十分简单， **reference** 元素的 **uri** 属性可以包含附件的位置，以 **Content-ID URL** 为表示方法，该 URL 与 RFC 2111 中所定义的一致。

10 传输方式连结

10.1 HTTP 连结

本节中描述使用 HTTP 通讯协议以传送 BizTalk 文件（包含或不包含附件）。

HTTP 是一种 **请求/响应** 通讯协议，然而 BizTalk Framework 结构却是基于异步传讯。HTTP 资源是 BizTalk 讯息存取的目标，因此资源并不一定是要 SOAP 1.1 规格中定义的 SOAP 处理器，或是 BFC 伺服程序。比较特别的是，顺利的响应（2xx 状态码）并不需要暗示下列各项：

- 讯息和它的主要 BizTalk 文件已经被目标商业实体接受。

或

- SOAP 信封的整合性和名称空间正确性已经验证。

本节中所描述的 HTTP 连结是 SOAP 1.1 HTTP 连结的特殊范例，并且稍后会描述在 SOAP 中传送附件（使用复合/相关 MIME 媒体类型）。请注意这里所描述的 HTTP 连结不使用 SOAP 远程过程调用（RPC）或同步的请求/回应对应式（request/response pattern），比较特别的是当可信赖传递机制所产生的收据讯息（第 8 节 所说明）都会视为 HTTP 请求传递，这些讯息永远不会当成 HTTP 响应传递。

10.1.1 简单 BizTalk 讯息的范例

下列的范例显示 BizTalk 文件，该文件使用 POST 方式的 HTTP 讯息传送：

```
POST /bookPurchase HTTP/1.1
Host:www.we-have-books.com
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn
```

```
SOAPAction:http://electrocommerce.org/purchase_order/



<?xml version='1.0 ' ??>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <SOAP-ENV:Header>
        <dlv:delivery SOAP-ENV:mustUnderstand="1"
            xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"
            xmlns:agr="http://www.trading-agreements.org/types/">
            <dlv:to>
                <dlv:address xsi:type="agr:department">
                Book Order Department</dlv:address>
            </dlv:to>
            <dlv:from>
                <dlv:address xsi:type="agr:organization">
                Booklovers Anonymous</dlv:address>
            </dlv:from>
        </dlv:delivery>
        <prop:properties SOAP-ENV:mustUnderstand="1"
```

```
            xmlns:prop="http://schemas.biztalk.org/btf-2-0/
            properties">
            <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6
            </prop:identity>
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:
            expiresAt>
            <prop:topic>http://electrocommerce.org/purchase_order/
            </prop:topic>
        </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <po:PurchaseOrder xmlns:po="http://electrocommerce.org/
purchase_order/">
            <po:Title>Essential BizTalk</po:Title>
        </po:PurchaseOrder>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

10.1.2 包含附件的 BizTalk 讯息范例

```
POST /insuranceClaims HTTP/1.1
Host:www.risky-stuff.com
Content-Type:Multipart/Related;
        boundary=biztalk_2_0_related_boundary_example;
        type=text/xml;
        start="<claim.xml@claiming-it.com>"
Content-Length:nnnn
SOAPAction:http://schemas.risky-stuff.com/Auto-Claim
Content-Description:This is the optional message description.


--biztalk_2_0_related_boundary_example
Content-Type:text/xml;charset=UTF-8
Content-Transfer-Encoding:8bit
Content-ID:<claim.xml@claiming-it.com>


<?xml version='1.0 ' ??>
<SOAP-ENV:Envelope
```

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"

xmlns:xsd="http://www.w3.org/1999/XMLSchema-datatypes">
     <SOAP-ENV:Header>
         <!--manifest header entry omitted for brevity -->
         <delivery SOAP-ENV:mustUnderstand="1"
                  xmlns="http://schemas.biztalk.org/btf-2-0/delivery">
           <to>
             <address xsi:type="xsd:uriReference">
             dept:insurance_claim_department</address>
           </to>
           <from>
             <address xsi:type="xsd:uriReference">
             agent:/WA/Issaquah#id=12345</address>
           </from>
         </delivery>
         <prop:properties SOAP-ENV:mustUnderstand="1"
```

```
xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
            <!--other elements omitted for brevity -->
            <topic>http://schemas.risky-stuff.com/Auto-Claim</topic>
        </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <Insurance_Claim_Auto
            xmlns="http://schemas.risky-stuff.com/Auto-Claim"
            id="insurance_claim_document_id">
            <!--...claim details...-->
        </Insurance_Claim_Auto>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--biztalk_2_0_related_boundary_example
Content-Type:image/tiff
Content-Transfer-Encoding:base64
Content-ID:<claim.tiff@claiming-it.com>
```

```
    ...Base 64 encoded TIFF image...

--biztalk_2_0_related_boundary_example

Content-Type:image/jpeg

Content-Transfer-Encoding:binary

Content-ID:<car.jpeg@claiming-it.com>



...Raw JPEG image...

--biztalk_2_0_related_boundary_example--
```

11 BizTalk 文件架构

11.1 XDR 架构

11.1.1 Delivery 标头项目

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

<!--

    BizTalk Framework 2.0

    BizTalk Document Schema:delivery header entry
```

```
    Copyright 2000 Microsoft Corporation
-->

<Schema
    name="biztalk_2_0_delivery.xml"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
    <!--delivery header entry element -->
    <ElementType name="delivery " content="eltOnly">
        <element type="to " minOccurs="1 " maxOccurs="1"/>
        <element type="from " minOccurs="1 " maxOccurs="1"/>
        <element type="reliability " minOccurs="0 " maxOccurs="1"/>
</ElementType>
    <ElementType name="to " content="eltOnly " >>
        <element type="address " minOccurs="1 " maxOccurs="1"/>
    </ElementType>
    <ElementType name="from " content="eltOnly">
        <element type="address " minOccurs="1 " maxOccurs="1"/>
    </ElementType>
```

```
    <ElementType name="reliability " content=="eltOnly">
        <element type="sendReceiptTo " minOccurs="1 " maxOccurs="1"/>
        <element type="receiptRequiredBy " minOccurs="1 "
        maxOccurs="1"/>
    </ElementType>
    <ElementType name="sendReceiptTo " content="textOnly " dt:
    type="uri"/>
    <ElementType name="receiptRequiredBy " content="textOnly " dt:
    type="dateTime.tz"/>
    <ElementType name="address " content="textOnly " dt::type="string">
    <attribute type="xsi:type " required="yes"/>
</ElementType>
</Schema>
```

11.1.2 Properties 标头项目

```
<?xml version="1.0 " ??>
<!--
    BizTalk Framework 2.0
    BizTalk Document Schema:properties header entry
```

```
    Copyright 2000 Microsoft Corporation
-->

<Schema
    name="biztalk_2_0_properties.xml"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<ElementType name="properties " content="eltOnly">
        <attribute type ="SOAP-
ENV:mustUnderstand " default="1 " required="yes"/>
        <element type="identity " minOccurs="1 " maxOccurs="1"/>
        <element type="sentAt " minOccurs="1 " maxOccurs="1"/>
        <element type="expiresAt " minOccurs="1 " maxOccurs="1"/>
        <element type="topic " minOccurs="1 " maxOccurs="1"/>
    </ElementType>
    <ElementType name="identity " content="textOnly " dt:type="uri"/>
    <ElementType name="sentAt " content="textOnly "
dt:type="dateTime.tz"/>
```

```
    <ElementType name="expiresAt " content="textOnly "
dt:type="dateTime.tz"/>
    <ElementType name="topic " content="textOnly " dt::type="uri"/>
</Schema>
```

11.1.3 Manifest 标头项目

```
<?xml version="1.0 " ??>
<!--
     BizTalk Framework 2.0
     BizTalk Document Schema:manifest header entry
     Copyright 2000 Microsoft Corporation
-->
<Schema
     name="biztalk_2_0_manifest.xml"
     xmlns="urn:schemas-microsoft-com:xml-data"
     xmlns:dt="urn:schemas-microsoft-com:datatypes">

     <ElementType name="manifest " content="eltOnly">
        <element type="reference " minOccurs="1 " maxOccurs="*"/>
```

```
    </ElementType>
    <ElementType name="reference " content="eltOnly">
        <attribute type="uri " required="yes"/>
        <element type="description " minOccurs="0 " maxOccurs="1"/>
    </ElementType>
    <AttributeType name="uri " dt::type="uri"/>
    <ElementType name="description " content="mixed"/>
</Schema>
```

11.1.4 Process 标头项目

```
<?xml version="1.0 " ??>
<!--
    BizTalk Framework 2.0
    BizTalk Document Schema:process header entry
    Copyright 2000 Microsoft Corporation
-->

<Schema
    name="biztalk_2_0_process.xml"
```

```
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <ElementType name="process " content="eltOnly">
        <attribute type="SOAP-ENV:mustUnderstand " default="1 "
        required="yes"/>
        <element type="type "          minOccurs="1 " maxOccurs="1"/>
        <element type="instance "       minOccurs="1 " maxOccurs="1"/>
        <element type="handle "        minOccurs="0 " maxOccurs="1"/>
    </ElementType>
    <ElementType name="type " content="textOnly " dt::type="uri"/>
    <ElementType name="instance " content="textOnly " dt::type="uri"/>
    <ElementType name="handle " content="textOnly " dt::type="uri"/>
</Schema>
```

11.1.5 Receipt 标头项目

```
<?xml version="1.0 " ??>
<!--
  BizTalk Framework 2.0
```

```
  BizTalk Document Schema:receipt header entry

  Copyright 2000 Microsoft Corporation

-->



<Schema

  name="biztalk_2_0_process_receipt_header.xml"

  xmlns="urn:schemas-microsoft-com:xml-data"

  xmlns:dt="urn:schemas-microsoft-com:datatypes"

  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

  <ElementType name="receipt " content="eltOnly">

    <attribute type="SOAP-ENV:mustUnderstand " default="1 "

            required="yes"/>

    <element type="receivedAt " minOccurs="1 " maxOccurs="1"/>

  </ElementType>

<ElementType name="receivedAt " content="textOnly "

            dt:type="dateTime.tz"/>

</Schema>
```

11.1.6 BizTalk 文件的 SOAP 1.1 信封

```
<?xml version="1.0 " ??>
<!--
    BizTalk Framework 2.0
    BizTalk Document Schema:envelope
    Copyright 2000 Microsoft Corporation

    This schema is based on the SOAP 1.1 schema
-->

<Schema
    name="biztalk_2_0_document_envelope.xml"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
<!--
    SOAP envelope,header and body
-->

    <ElementType name="Envelope " content="eltOnly">
        <element type="Header " minOccurs="1 " maxOccurs="1"/>
        <element type="Body " minOccurs="1 " maxOccurs="1"/>
```

```
    </ElementType>
    <ElementType name="Header " content="eltOnly " model="open"

xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"

xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties"

xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest"

xmlns:prc="http://schemas.biztalk.org/btf-2-0/process">
    <element type="dlv:delivery " minOccurs="1 " maxOccurs="1"/>
    <element type="prop:properties " minOccurs="1 " maxOccurs="1"/>
    <element type="fst:manifest " minOccurs="0 " maxOccurs="1"/>
    <element type="prc:process " minOccurs="0 " maxOccurs="1"/>
</ElementType>
<ElementType name="Body " content="eltOnly " model="open"/>
<!--
    Global Attributes.The following attributes are intended
    to be usable via qualified attribute names on any Element type
    referencing them.
```

```
-->

<AttributeType name="mustUnderstand " default="0 " dt::type="Boolean"/>
<AttributeType name="actor " dt::type="uri"/>
<!--
    'encodingStyle' indicates any canonicalization conventions followed
    in the contents of the containing element.For example,the value
    'http://schemas.xmlsoap.org/soap/encoding/' indicates
    the pattern described in SOAP specification.
-->
    <AttributeType name="encodingStyle " dt::type="string"/>
<!--
SOAP fault reporting structure
-->

<ElementType name="Fault " content="eltOnly">
    <element type="faultcode " minOccurs="1 " maxOccurs="1"/>
    <element type="faultstring " minOccurs="1 " maxOccurs="1"/>
    <element type="faultactor " minOccurs="0 " maxOccurs="1"/>
    <element type="detail " minOccurs="0 " maxOccurs="1"/>
```

```
</ElementType>

    <ElementType name="faultcode " content="textOnly "

dt::type="string"/>

    <ElementType name="faultstring " content="textOnly " dt:

    type="string"/>

    <ElementType name="faultactor " content="textOnly "

dt::type="uri"/>

    <ElementType name="detail " content="eltOnly " model="open"/>

</Schema>
```

11.1.7 BizTalk 收据的 SOAP 1.1 信封

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

<!--

    BizTalk Framework 2.0

    BizTalk Document Schema:receipt

    Copyright 2000 Microsoft Corporation

-->



<Schema
```

```
    name="biztalk_2_0_receipt_envelope.xml"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
<!--
    SOAP envelope,header and body
-->

    <ElementType name="Envelope " content="eltOnly">
        <element type="Header " minOccurs="1 " maxOccurs="1"/>
        <element type="Body " minOccurs="1 " maxOccurs="1"/>
    </ElementType>
    <ElementType name="Header " content="eltOnly " model="open"

xmlns:rct="http://schemas.biztalk.org/btf-2-0/receipt"

xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties"

xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest"

xmlns:prc="http://schemas.biztalk.org/btf-2-0/process">
```

```
        <element type="rct:receipt " minOccurs="1 " maxOccurs="1"/>
        <element type="prop:properties " minOccurs="1 " maxOccurs="1"/>
        <element type="fst:manifest " minOccurs="0 " maxOccurs="1"/>
        <element type="prc:process " minOccurs="0 " maxOccurs="1"/>
    </ElementType>
<ElementType name="Body " content="empty " model="closed"/>
</Schema>
```

附录 B XML 和 BizTalk 的 Web 网站

在此介绍许多致力于描述 XML、架构和 BizTalk 的网站：

- BizTalk 指导委员会维护的 http://biztalk.org

- Microsoft BizTalk 网站 http://www.microsoft.com/biztalk

笔者常常造访这些网站，本章也将说明一些其它相关网站。

BizTalk.ORG

BizTalk.org 包含了许多 XML 与 BizTalk 的相关信息，这个网站上有许多地方可以让新手学习相关的技术新知和最佳演练实例、可以与其它使用者交换心得，并且让使用者以最有效率的方式取得建置 B2B 环境和 B2C 应用程序的工具。

BizTalk.org 是您学习 B2B 电子商务解决方案的最佳起点，这个网站也可以帮助您了解商业伙伴和工业组织使用哪些架构和应用程序。

点对点的支援

许多涉及商业信息交换标准化的组织对商业处理过程的模式化技术比较熟悉，却对系统程序和 XML 并不了解。这些组织可以到 BizTalk.org 的 Web 网站寻找建置系统架构或目前应用程序所使用架构的最佳实例。

BizTalk.org 最重要的优点之一就是它可以让任何人存取，并且允许任何人发布 XML 的架构或者从网站中下载 XML 架构。这个网站的任务就是提升 XML 为商业上交易的最佳解决方案。到此为止，您应该了解 BizTalk.org 有两个主要的特色－首先，该网站是一个学习社群，人们可以在那里取得最新的信息并且交换心得，这个社群以开放式的 Q&A 形式讨论相关架构开发的议题。

其次，就是这个网站上有许多发布的架构，这些架构都经过预先的验证并且储存在链接库中提供任何人存取。程序开发师可以确信在 BizTalk.org 所取得的架构是经过技术上的测试，评估的数据指出在网络上约有 80%的架构没有经过验证，不正确的架构对验证 XML 文件方面十分的没有效率，传送至 BizTalk.org 网站的架构会经由技术验证以确认没有错误。

BizTalk.org 网站所掌控的架构是 24 小时都可以存取的，这个架构也可以供任何感兴趣的使用者下载。

由 BizTalk.org 所掌控的架构会控制版本，一旦架构发布后，发布的架构就不能够再加以变更。版本控制是一个重要的服务，这样该网站所提供的架构才有价值，因为企业进行商业交易不能够使用经常变更的架构。

Microsoft BizTalk Home Page

Microsoft BizTalk 网站： http://www.microsoft.com/biztalk/ 会提供关于 BizTalk Framework 和 Microsoft BizTalk Server 2000 的信息。在 BizTalk Server 2000 的开发期间，Microsoft 的 BizTalk 网站同时也提供了预先测试的版本，使用者可以先行测试该产品的功能性和实用性。该网站也提供成功的 XML 案例研究以指出一般的企业如何使用 BizTalk Framework 来建置 B2B 的交易，Microsoft 将利用这个网站介绍 BizTalk Server 2000 和提供支持。

XML.COM

XML.com（http://www.xml.com）是一个以广告维生的商业网站，该网站致力于报导 XML 的商业动态，XML.com 的任务是报导 XML 所发生的事件和新闻，并且也提供了 XML 标准的开发状态和供货商所研发的 XML 产品。

XML.com 并不是一个只发布技术信息的网站，他们也会报导相关产业或规格不兼容的文章。举例来说，该网站会报导 XML 剖析器受欢迎的程度、OASIS 和 NIST 所测试的技术兼容性文章，测试的设计理念是要检查剖析器是否符合 W3C 发布的 XML 规格。

OASIS

OASIS 是一个 XML 的产业组织，致力于相关于 XML 商业环境中所发生的供货商信息，同时它也是一个联合协会，包含数家对使用 XML 解决商业问题感兴趣的公司。大部份的公司也会提供在某些领域支持 XML 的产品，而 OASIS 中也包含 XML 的使用者。

在 OASIS 网站（ http://www.oasis-open.org ）的最佳资源就是 **The XML Cover Pages** （以本书的发行期间为准）， **The XML Cover Pages** 每天都会更新－有时候甚至一天供应数次，其中包含不同公司的新发布信息和 XML 标准的学术研究。

XML.ORG

XML.org 是设计用于提供独立、可信赖的信息来源，使用者可以利用这些信息建立兼容的系统。XML.ORG Steering Committee 和 OASIS Board of Directors 是专司建立有价值的产业架构，而且各个产业都可以在该网站注册和储存架构，他们也会处理某些产业的的棘手问题，例如智慧资产（Intellectual Property，IP）原则。

OASIS 形成 Registry and Repository Technical Committee，并且提供一种标准的方式在任何网站注册架构，使用者可以轻易存取这些架构。这是一个重要的研发工作，因为使用 XML 的产业需要一个可信赖的地方来储存架构（如果使用名称空间）。如果该研发组织完成了这个规格，它将会建立一个供您参考的应用程序以显示该规格如何工作。

因为 OASIS 开发的是跨产业的标准，因此他们的研发工作比小型的组织（公司领导的组织）更为复杂。纵使跨产业标准的研发过程需要比较长的时间，但是其研发的标准比单一供货商所研发的标准更容易让使用者采纳。

<TAG> NEWSLETTER

<TAG> Newsletter（http://tagnewsletter.com）在 1987 年就发布信息管理商业的技术信息，<TAG>的任务就是提供 XML 的使用者建置工具和信息，同时<TAG>也提供程序代码范例和技术提示、发布许多案例研究和相关书籍。

笔者曾经也是<TAG>的编辑之一，我们每天转换超过 1000 份文章为 XML 并且将它们储存在数据库中，使用该数据库，可以根据个别使用者的设定档提供每一份文章的呈现方式。举例来说，利用 Netscape Navigator 的使用者造访网站时与利用 Microsoft Internet Explorer 的使用者存取网站时所取得的网页结构不一样，这个网站是利用 Web 浏览器和存取方法的差异性以建立不同的内容呈现方式。

这种个人化系统的好处之一是，它可以让已经付费的订阅者实时存取网站上的所有文章（包含最新的版本）；非订阅者也可以存取所有的文章，但是他们只能取得标题、作者和文章的摘要。

我们使用 XML、XSLT 和中间层的指令码以达到个人化和使用不同浏览器的呈现方式，当我们建置该网站的时候，发现有许多呈现信息的方式。举例来说，信息可以利用许多方式输出，并且提供给不同的读者。我们的目标是利用少量的成本就可以建构智能型的内容管理和传递环境。

XMLSOFTWARE

如果您想了解有什么 XML 工具可以使用，XMLSOFTWARE（http://www.xmlsoftware.com）是您最佳的选择，这个网站会经常性的更新。

XMLSOFTWARE 也提供电子邮件通知的服务，该服务会在新信息加入时传送电子邮件给订阅者。该网站的作者也维护其它的 XML 相关网站，其中包含 XML 的建置工具和架构的储存处，这些网站分别是 XMLINFO（ http://www.xmlinfo.com）和 SCHEMA.NET（ http://www.schema.net ），XMLSOFTWARE 网站中包含这些网站的连结。

附录 C Simple Object Access Protocol（SOAP）1.1

W3C 备忘录, 2000/05/08

本章版本

http://www.w3.org/TR/2000/NOTE-SOAP-20000508

最新版本

http://www.w3.org/TR/SOAP

作者（依字母顺序排列）

Don Box, DevelopMentor

David Ehnebuske, IBM

Gopal Kakivaya, Microsoft

Andrew Layman, Microsoft

Noah Mendelsohn, Lotus Development Corp.

Henrik Frystyk Nielsen, Microsoft

Satish Thatte, Microsoft

Dave Winer, UserLand Software, Inc.

节录

SOAP 是一个在分布式环境中使用 XML 交换信息的通讯协议，其中包含了三个部份：一个信封中（envelope）定义如何处理讯息的基础结构；一组程序代码撰写规则的集合用以表达应用程序定义数据型别的执行个体；以及描述远程过程调用和回应的协议。SOAP 也可以和许多其它通讯协议联合使用。然而，本文件中定义的连结只会描述如何使用 SOAP 联合 HTTP 和 HTTP Extension Framework 协同工作。

内容列表

1 简介

1.1 设计目标

1.2 标记协议

1.3 SOAP 讯息的范例

2 SOAP 讯息交换模式

3 SOAP 和 XML 的关联性

4 SOAP Envelope

4.1.1 SOAP encodingStyle 属性

4.1.2 Envelope Versioning Model

4.2 SOAP Header

4.2.1 使用 Header 属性

4.2.2 SOAP actor 属性

4.2.3 SOAP mustUnderstand 属性

4.3 SOAP Body

4.3.1 SOAP Header 和 Body 的关联性

4.4 SOAP Fault

4.4.1 SOAP Fault Codes

5 SOAP Encoding

6 使用 SOAP 和 HTTP 协同工作

6.1 SOAP HTTP 请求

6.1.1 SOAPAction HTTP Header Field

6.2 SOAP HTTP 回应

6.3 HTTP Extension Framework

6.4 SOAP HTTP 范例

7 使用 SOAP 执行 RPC

7.1 PRC 和 SOAP Body

7.2 RPC 和 SOAP Header

8 Security 考量

9 参考数据

A SOAP Envelope 范例

A.1 呼叫请求的范例

A.2 响应的范例

1 简介

SOAP 使用 XML 在分布式的环境（点对点）中提供一种简单结构和型别信息的交换机制，SOAP 自身并不会定义任何应用程序的语法，SOAP 定义了表达应用程序语法的机制（利用模块化的封装模式和在模块中标记数据的机制）。这个功能让 SOAP 可以用于广泛的各类型系统之中。

SOAP 包含三个部份：

- SOAP 信封（请参阅 第 4 节 ）中定义了表达讯息中包含 什么 的基础结构；谁 应该处理这个讯息； 无论 讯息是选择性或强制性。

- SOAP 程序代码撰写规则（请参阅 第 5 节 ）定义了一个序列化的机制，这个机制用于交换应用程序定义数据型别的执行个体。

- SOAP RPC（请参阅 第 7 节 ）定义用于描述远程过程调用和响应的协议。

除了 SOAP 信封、SOAP 程序代码撰写规则和 SOAP RPC 协议以外，SOAP 也定义了两种通讯协议的连结方式，其中包含描述 SOAP 讯息如何在 HTTP[5]讯息中传送（协同或不协同 HTTP Extension Framework[6]工作）。

1.1 设计目标

SOAP 的主要设计目标就是要具备简单性和延伸性，这表示它包含了来自传统讯息系统的数个功能，并且分布式对象系统不是核心 SOAP 规格的一部份，这些功能包含：

- Distributed garbage collection
- Boxcarring or batching of messages
- Objects-by-reference（需要有 Distributed garbage collection）
- Activation（需要有 Objects-by-reference）

1.2 标记协议

本文件中的名称空间前置字「SOAP-ENV」和「SOAP-ENC」分别与 SOAP 名称空间「http://schemas.xmlsoap.org/soap/envelope 」和「schemas.xmlsoap.org/soap/encoding」联合使用。

本文件中的名称空间前置字「xsi」是与 XML Schemas 规格 [11] 中定义的 URI 「 http://www.w3.org/1999/XMLSchema-instance」联合使用；同样的，名称空间前置字「xsd」是与规格 [10] 中定义的 URI 「 http://www.w3.org/1999/XMLSchema 」联合使用；名称空间前置字「tns」用于指定目前文件的名称空间目标。

「Some-URI」的名称空间 URI 代表某些应用程序相依（application-dependent）或执行环境相依（context-dependent）的 URI [4]。

这个规格也使用在 RFC-2616 [5] 中描述的 Backus-Naur Form（BNF）。

1.3 SOAP 讯息的范例

在本范例中，GetLastTradePrice SOAP 请求传送至 StockQuote 服务。SOAP Envelope 元素为 XML 文件中的最顶层元素，以代表该文件为 SOAP 讯息，本范例显示了 第六节 中定义的 HTTP 连结。

范例 1 嵌在 HTTP 请求中的 SOAP 讯息

```
SOAP Message Embedded in HTTP Request



POST /StockQuote HTTP/1.1
```

```
Host:www.stockquoteserver.com

Content-Type:text/xml;charset="utf-8"

Content-Length:nnnn

SOAPAction:"Some-URI"



<SOAP-ENV:Envelope

   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"



SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

   <SOAP-ENV:Body>

         <m:GetLastTradePrice xmlns:m="Some-URI">

             <symbol>DIS</symbol>

         </m:GetLastTradePrice>

     </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

下列的响应讯息中包含一个在 HTTP 中嵌入 SOAP 的讯息。

范例 2　嵌在 HTTP 响应中的 SOAP 讯息

```
SOAP Message Embedded in HTTP Response



HTTP/1.1 200 OK

Content-Type:text/xml;charset="utf-8"

Content-Length:nnnn



<SOAP-ENV:Envelope

    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"



SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>

    <SOAP-ENV:Body>

        <m:GetLastTradePriceResponse xmlns:m="Some-URI">

            <Price>34.5</Price>

        </m:GetLastTradePriceResponse>

     </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

2 SOAP 讯息交换模式

SOAP 的原理基本上是从传送者至接收者的单方向传输，但是如同上述范例所示，SOAP 讯息交换通常会和商业上的伙伴以 **请求/响应** 的方式建置。

SOAP 是非常有弹性的技术，可以为特定网络系统所需要的特性以最佳化的方式建置。举例来说，在 第 6 节 中描述的 HTPP 连结提供了以 HTTP 响应方式传递的 SOAP 响应讯息，并使用同一个联机阶段为输入的请求。

接收 SOAP 讯息的 SOAP 应用程序必须依次执行下列的动作：

1. 识别供该应用程序处理的 SOAP 讯息每一个部份（请参阅 4.2.2 节）。
2. 确认该应用程序支持步骤 1 中识别的每一个部份（请参阅 4.2.3 节），某些不会影响输出的选择性部份会被忽略（请参阅 4.4 节）。
3. 如果 SOAP 应用程序不是该讯息的最终目的，那么在转传讯息之前先移除步骤 1 中所识别的部份。

3 SOAP 和 XML 的关联性

所有的 SOAP 讯息都使用 XML 程序代码撰写（请参阅 [7] 以取得相关于 XML 的更多信息）。

SOAP 应用程序应该要在所有的元素中都包含正确的 SOAP 名称空间，SOAP 应用程序必须能够在其接收的讯息中处理 SOAP 名称空间，并且必须能够忽略名称空间错误的讯息（请参阅

第 4.4 节）。同时它也许应该能够处理没有 SOAP 名称空间的 SOAP 讯息，并且将这些没有名称空间的讯息视为具备正确的 SOAP 名称空间。

SOAP 定义两种名称空间（请参阅 [8] 以取得相关于 XML 名称空间的更多信息）：

- SOAP Envelope 有名称空间识别项

  「http://schemas.xmlsoap.org/soap/envelope/」。

- SOAP Serialization 有名称空间识别项

  「http://schemas.xmlsoap.org/soap/encoding/」。

SOAP 讯息中绝对不能包含文件类别定义（DTD）和处理指令（PI）[7]。

SOAP 使用本机、未限定类别「ID」的「id」属性以指定一个程序代码撰写元素的唯一识别项；SOAP 使用本机、未限定类别「uri-reference」的「href」属性以指定该值的参照，这些方式都遵守 XML Specification [7]、XML Schema Specification [11] 和 XML Language Specification [9]。

除了使用 SOAP mustUnderstand 属性（请参阅第 4.2.3 节）和 SOAP actor 属性（请参阅第 4.2.2 节）以外，它通常也允许属性和属性值出现在 XML 执行个体或架构中－这就表示在 DTD 或架构中有默认值或固定值的话，在语法上是同于出现在执行个体中。

4 SOAP Envelope

SOAP 讯息是 XML 文件，该文件中一定要包含 SOAP Envelope 和 SOAP Body，而 SOAP Header 是选择性的。在本节定义中元素和属性的名称空间识别项是「http://schemas.xmlsoap.org/soap/envelope/」。SOAP 讯息包含下列各项：

1. Envelope 是 XML 文件的顶端元素，代表该文件为 SOAP 讯息。
2. Header 是一般用于在分布式环境中的 SOAP 讯息中新增功能的机制，不需要事先与商业伙伴建立合约。SOAP 定义了一些属性用于在 Header 中指定「谁」处理该功能，以及它是选择性或强制性（请参阅第 4.2 节）。
3. Body 就是接收者所要接收讯息的容器（请参阅第 4.3 节），SOAP 定义了一个元素供 Body 使用，即为 Fault 元素用以报告错误。

语法规则如下：

1. Envelope
    - 该元素名称为「Envelope」。
    - 该元素必须出现在 SOAP 讯息中。

- 该元素可以包含名称空间宣告，以及额外的属性。如果存在额外属性，属性必须使用名称空间限定（namespace-qualified）；一样的，该元素也许会包含子元素，如果存在子元素，这些子元素必须使用名称空间限定，并且位于 Body 元素之后。

1. Header（请参阅第 4.2 节）
    - 该元素名称为「Header」。
    - 该元素不一定要出现在 SOAP 讯息中，如果 SOAP 讯息中存在 Header 元素，那么 Header 元素必须直接为 SOAP Envelope 元素的第一个子元素。
    - 所有的 SOAP Header 元素的直接子元素必须名称空间限定。

1. Body（请参阅第 4.3 节）
    - 该元素名称为「Body」。
    - 该元素必须出现在 SOAP 讯息中，并且必须为 SOAP Envelope 元素的直接子元素。如果存在 SOAP Header 元素的话，那么它就必须直接位于其

后，否则 SOAP Body 元素就必须为 SOAP Envelope 元素的直接子元素。

- SOAP Body 元素的直接子元素不一定要名称空间限定，SOAP 定义了 SOAP Fault 元素用以报告错误（请参阅第 4.4 节）。

4.1.1 SOAP encodingStyle 属性

SOAP encodingStyle 通用属性可以指定用于 SOAP 讯息的序列规则，这个属性不一定要出现在任何元素中，并且有效范围从宣告该属性的元素内容开始及其所有的子元素。

该属性的值是一个依照顺序排列的清单，其中包含一或多个 URI 以识别要连续套用的规则，值的范例如下：

```
"http://schemas.xmlsoap.org/soap/encoding/"
"http://my.host/encoding/restricted http://my.host/encoding/"
""
```

在 第 5 节 中 SOAP 定义的序列规则是由 URI「http://schemas.xmlsoap.org/soap/encoding/」所识别，使用这个特定序列的讯息应该要使用 SOAP encodingStyle 属性。还有，所有的 URI 语法上都使用「http://schemas.xmlsoap.org/soap/encoding/」开始。

没有长度的 URI（zero-length URI，""）值是用于指定该元素的程序代码撰写样式并没有特定要求。

4.1.2 Envelope Versioning Model

SOAP 讯息必须有 Envelope 元素联合名称空间「http://schemas.xmlsoap.org/soap/envelope/」使用，如果一个 SOAP 应用程序接收了一个讯息，而该讯息的 SOAP Envelope 元素使用和上述不同的名称空间，该应用程序就必须将其视为版本错误并忽略该讯息。如果讯息使用例如 HTTP 的 请求/响应 通讯协议，应用程序就必须响应一个 SOAP VersionMismatch 错误码讯息（使用「http://schemas.xmlsoap.org/soap/envelope/」名称空间）。

4.2 SOAP Header

SOAP 提供了一个具弹性的方式延伸讯息，以及一种模块化的方式与商业伙伴之间通讯。延伸讯息的典型范例就是在标头项目中建置例如验证、交易管理、付款方式等等信息。

Header 元素必须为 SOAP Envelope XML 元素的第一个直接子元素，所有 Header 元素的直接子元素就称为标头项目（header entry）。

标头项目标程序码撰写规则如下：

1. 标头项目由它的完整元素名称所识别，其中包含名称空间 URI 和本机名称，SOAP Header 的直接子元素必须名称空间限定。
2. SOAP encodingStyle 属性不一定会被用于指定标头项目标程序码撰写样式（请参阅第 4.1.1 节）。
3. SOAP mustUnderstand 属性（请参阅第 4.2.3 节）和 SOAP actor 属性（请参阅第 4.2.2 节）不一定要用于指定如何处理项目或谁来处理（请参阅 第 4.2.1 节 ）。

4.2.1 使用 Header 属性

在本节定义的 SOAP Header 属性决定 SOAP 讯息的接收者应该如何利用第 2 节所描述的方式处理该讯息。产生 SOAP 讯息的 SOAP 应用程序应该只会在 SOAP Header 元素的直接子元素使用 SOAP Header 属性，如果 SOAP Header 属性没有套用于 SOAP Header 元素的直接子元素之中，SOAP 讯息的接收者必须忽略所有的 SOAP Header 属性。

标头使用元素识别项「Transaction」的范例一「mustUnderstand」属性的值为「1」；元素识别项「Transaction」值为「5」，程序代码撰写方式如下：

```
<SOAP-ENV:Header>
    <t:Transaction
        xmlns:t="some-URI " SOAP--ENV:mustUnderstand="1">
            5
```

```
    </t:Transaction>
</SOAP-ENV:Header>
```

4.2.2 SOAP actor 属性

SOAP 讯息从出发点至目的地之间会经过许多 SOAP 中间媒介，所谓 SOAP 中间媒介就是 SOAP 的应用程序，这些应用程序具备接收和转传 SOAP 讯息的能力，接收标头元素的接收者绝对不能将该标头元素转传到下一个 SOAP 应用程序，但是可以插入相似的标头元素。

SOAP actor 通用属性可以用于指定标头元素的接收者，SOAP actor 属性的值是 URI，特别的 URI「http://schemas.xmlsoap.org/soap/next/」可以指定该标头元素试图传递的下一个 SOAP 应用程序。

删除 SOAP actor 属性指出接收者即为 SOAP 讯息的最终点。

该属性一定要依次出现在 SOAP 讯息执行个体中（请参阅 第 3 节 和 第 4.2.1 节 ）。

4.2.3 SOAP mustUnderstand 属性

SOAP mustUnderstand 通用属性用于指出标头项目接收者是否一定要处理标头项目，标头项目的接收者是由 SOAP actor 属性所定义（请参阅第 4.2.2 节），mustUnderstand 属性的值可以为「1」或「0」，如果没有指定 SOAP mustUnderstand 属性值，预设为「0」。

如果该属性值为「1」，标头项目的接收者必须处理标头讯息，否则会传回错误（请参阅第 4.4 节）。

该属性一定要依次出现在 SOAP 讯息执行个体中（请参阅 第 3 节 和 第 4.2.1 节 ）。

SOAP Body

SOAP Body 元素必须为 SOAP Envelope XML 元素的第一个直接子元素，如果存在 SOAP Header 元素的话，那么它就必须直接位于其后，否则 SOAP Body 元素就必须为 SOAP Envelope 元素的直接子元素。

所有的 Body 元素的直接子元素都称为数据主体项目（body entry），并且每一个数据主体项目在 SOAP Body 元素中都是独立的元素。

数据主体的程序代码撰写规则如下：

1. 数据主体项目由它的完整元素名称所识别，其中包含名称空间 URI 和本机名称，SOAP Body 元素的直接子元素不一定要名称空间限定。
2. SOAP encodingStyle 属性不一定会被用于指定数据主体项目标程序码撰写样式（请参阅第 4.1.1 节）。

SOAP 定义了一个数据主体项目，即为 Fault 项目用于报告错误（请参阅 4.4 节）。

4.3.1 SOAP Header 和 Body 的关联性

虽然 Header 和 Body 被定义为独立的元素，事实上它们之间还是有关联性。在数据主体项目和标头项目之间的关联性如下：如果标头项目使用预设的 actor 属性值和 SOAP mustUnderstand 属性值为「1」时，基本上这种情况和数据主体项目在语法上是相同的，预设的 actor 属性是不使用 actor 属性（请参阅 4.2.2 节）。

4.4 SOAP Fault

SOAP Fault 元素用于传送 SOAP 讯息中的错误 和/或 状态信息。如果使用 SOAP Fault 元素，它一定要以数据主体的方式出现，并且只能够在 Body 元素中出现一次。

SOAP Fault 元素定义下列四个子元素：

- **faultcode** faultcode 元素由软件使用以提供算法的机制来识别错误，faultcode 必须存在于 SOAP Fault 元素中并且其值必须如 [8] 中所定义的完整名称。

- **faultstring** faultstring 元素试图提供可供人类读取的错误解释，并且不执行演算处理。Faultstring 元素相似于 HTTP （请参阅 [5] ）所定义的「Reason-Phrase」，faultstring 必须存在于 SOAP Fault 元素中并且应该至少要提供某些错误的解释信

息。

- **faultactor** faultactor 元素试图提供在讯息的路径中哪一个节点引起了错误（请参阅 第 2 节 ），它与 SOAP actor 属性相似（请参阅第 4.2.2 节），不同之处在于标头项目是指定目的地，而 faultactor 元素是指出错误的来源。faultactor 属性的值以 URI 来识别，但并非 SOAP 讯息终点的应用程序都必须在其 SOAP Fault 元素中包含 faultactor 元素。

- **detail** detail 元素试图指出特定应用程序的 Body 元素发生错误信息，如果 Body 元素的内容无法顺利处理，就必须存在 detail；请注意 detail 绝对不能够用于传递属于标头项目的错误信息。

在 Fault 元素中缺少 detail 元素指定错误跟 Body 元素的处理并没有相关性，这样可以用于区分是否 Body 元素被正确处理或并没有错误发生。

detail 元素的直接子元素称为明细项目（detail entry），而且每一个明细项目在 detail 元素中都是独立的元素。

detail 项目标程序码撰写规则如下（请参阅 范例 10 ）：

1. 明细项目由它的完整元素名称所识别，其中包含名称空间 URI 和本机名称，明细元素的直接子元素不一定要名称空间限定。
2. SOAP encodingStyle 属性不一定会被用于指定明细项目标程序码撰写样式（请参阅第 4.1.1 节）。

4.4.1 SOAP Fault Codes

在本节中定义的 faultcode 值必须用于 faultcode 元素中，这些 faultcode 值的名称空间识别项为「http://schemas.xmlsoap.org/soap/envelope/」，在这个规格中使用这个名称空间只是个建议（并不是必须）。

预设的 SOAP faultcode 值为了与现存的 faultcode 值向后兼容是以一种延伸的方式定义，这个机制非常相似于 HTTP（请参阅 [5] ）的基本状态类别 1xx、2xx 和 3xx 等等。SOAP faultcode 值不使用整数，并且使用 XML 完整名称定义（请参阅 [8] ），字符「.（点）」用于分开 faultcode 值，范例如下：

```
Client.Authentication
```

本文件中定义的 faultcode 值集合：

名称	意义
VersionMismatch	SOAP Envelope 元素的处理发现无效的名称空间（请参阅第 4.1.2 节）。

MustUnderstand	SOAP Header 元素的直接子元素无法解译或不遵从 SOAP mustUnderstand 属性值为「1」的处理程序。
Clien	错误的 Client 类别指出讯息的形成不正确或没有依次包含适当的信息。举例来说，讯息可能缺少正确的验证或付款方式信息，通常这个方式可以用来指定讯息不应该不经过修改过就重传。
Server	错误的 Server 类别指出讯息无法处理，原因是讯息本身的内容没有直接的属性，但是仍然需要处理该讯息。举例来说，处理时和服务器的通讯没有响应，讯息也许会在稍后的时间点就可以顺利处理。

5 SOAP Encoding

SOAP 的程序代码撰写样式是基于简单的型别系统，该型别系统具备程序语言、数据库和半结构数据中的一般性型别系统功能。型别可以为简单的型别，或是由数个部份组合而成的型别。首先，任何标记法的架构都会有一致的型别系统描述，例如 XML 语法的架构一样；次之，已知的型别系统架构会确认 XML 语法架构，就建立了 XML 执行个体。相反的，所产生的 XML 执行个体就会与这些规则一致，并且会建立最初的值图形。

在本节中所定义的元素和属性的名称空间识别项是

「http://schemas.xmlsoap.org/soap/encoding/」，之前所展示的程序代码撰写范例是假设所有的名称空间宣告都在较高的元素层级。

6 使用 SOAP 和 HTTP 协同工作

本节描述如何在 HTTP 中使用 SOAP 与（或不与）HTTP Extension Framework 协同工作，连结 SOAP 和 HTTP 可以取得许多优点，例如 SOAP 的弹性和 HTTP 的丰富功能。使用 HTTP 传送 SOAP 讯息并不表示这样就会完全取代 HTTP 的语法，因为 SOAP 的语法会对应 HTTP 语法。

SOAP 会遵守 HTTP 请求/响应 讯息模式，在 HTTP 的请求中提供 SOAP 请求的参数，并且在 HTTP 回应中包含 SOAP 回应参数。

当在 HTTP 讯息中包含 SOAP 数据主体时，HTTP 应用程序必须遵循 RFC 2376 [3] 的媒体型别「text/xml」。

6.1 SOAP HTTP 请求

纵使 SOAP 可以和多种 HTTP 请求方法联合使用，这个连结只能定义在 HTTP POST 中的 SOAP 请求（请参阅 第 7 节 以了解如何使用 SOAP for PRC 和 第 6.3 节 以了解如何使用 HTTP Extension Framework）。

6.1.1 SOAPAction HTTP Header Field

SOAPAction HTTP 请求标头字段可以用于指定 SOAP HTTP 请求的意图，该值为一 URI 以识别该请求的意图。当发布 SOAP HTTP 请求时，HTTP 客户端必须使用这个标头字段。

```
soapaction    = "SOAPAction" ":" [ <">URI-reference <"> ]
URI-reference = <as defined in RFC 2396 [4]>
```

SOAPAction 标头字段的内容由服务器使用，例如防火墙以在 HTTP 中适当的筛选 HTTP 请求讯息。使用空字符串（""）值的标头字段表示 SOAP 讯息是由 HTTP Request-URI 提供，没有值表示没有指定任何的意图。

举例来说：

```
SOAPAction:"http://electrocommerce.org/abc#MyMessage"
SOAPAction:"myapp.sdl"
SOAPAction:""
SOAPAction:
```

6.2 SOAP HTTP 回应

SOAP HTTP 遵循 HTTP 状态码（HTTP 的通讯状态信息）的语法，举例来说，2xx 状态码指出包含 SOAP 组件的客户端请求已经顺利接收、解译并且接受等等。

当处理请求时发生 SOAP 错误，SOAP HTTP 伺服程序必须发布 HTTP 500「内部服务器错误」响应，并且在响应中包含 SOAP 讯息（其中包含 SOAP Fault 元素）以指出 SOAP 处理错误。

6.3 HTTP Extension Framework

SOAP 讯息也可以和 HTTP Extension Framework [6] 一起使用以辨识 SOAP HTTP 请求的内容。

是否使用 HTTP Extension Framework 或是简单的 HTTP 是取决于通讯双方的原则和能力的问题，客户端可以藉由使用强制性的延伸宣告或「M-」HTTP 方法名称前置字以强制使用 HTTP Extension Framework；服务器可以藉由使用 510「Not Extended」HTTP 状态码以强制使用 HTTP Extension Framework。

用于指定 SOAP 使用 Extension Framework 的延伸识别项是「http://schemas.xmlsoap.org/soap/envelope/」。

6.4 SOAP HTTP 范例

范例 3 使用 POST 的 SOAP HTTP

```
SOAP HTTP Using POST
```

```
POST /StockQuote HTTP/1.1

Content-Type:text/xml;charset="utf-8"

Content-Length:nnnn

SOAPAction:"http://electrocommerce.org/abc#MyMessage"



<SOAP-ENV:Envelope...

HTTP/1.1 200 OK

Content-Type:text/xml;charset="utf-8"

Content-Length:nnnn



<SOAP-ENV:Envelope...
```

范例 4　使用 HTTP Extension Framework 的 SOAP

```
SOAP Using HTTP Extension Framework

M-POST /StockQuote HTTP/1.1

Man:"http://schemas.xmlsoap.org/soap/envelope/";ns=NNNN

Content-Type:text/xml;charset="utf-8"

Content-Length:nnnn

NNNN-SOAPAction:"http://electrocommerce.org/abc#MyMessage"
```

```
<SOAP-ENV:Envelope...
```

7 使用 SOAP 执行 RPC

SOAP 的设计目标之一是利用 XML 的弹性和延伸性来封装和交换 RPC 呼叫，本节定义远程过程调用和响应的统一表现方式。

使用 HTTP 为通讯协议连结的情况下，RPC 呼叫会自然对应 HTTP 请求，并且 PRC 回应会自然对应 HTTP 回应。然而，使用 SOAP 执行 PRC 并不只限于和 HTTP 通讯协议连结。

您需要下列的信息以使用一个方法呼叫：

- 目标对象的 URI。
- 方法的名称。
- 选择性的方法标记。
- 方法的参数。

- 选择性的标头数据。

7.1 RPC 和 SOAP Body

PRC 方法呼叫和响应都使用下列方式于 SOAP Body 元素中传送（请参阅 第 4.3 节 ）：

- 方法呼叫模式化为一结构。

- 方法呼叫视为单一结构，该结构中每一个 **in** 或 **in/out** 参数都包含存取器，该结构的命名和型别都和方法名称一致。

- 每一个 **in** 或 **in/out** 参数视为一个存取器，具有对应于参数名称的名称和对应于参数型别的型别。

- 方法响应是模式化为一结构。

- 方法响应视为单一结构，该结构中包含传回值和每一个 out 或 in/out 参数的存取器，第一个存取器就是参数后面传回的值，次序和方法标记中的次序一样。

- 每一个参数存取器都有一个相对于参数名称的名称和相对于参数型别的型别，传回值的存取器名称并不明显；同样地，结构名称也不明显。然而，方便之处在于方法名称后面会附加字符串「Response」。

7.2 RPC 和 SOAP Header

使用标头元素的范例就是在讯息中附带传送一个 TransactionID，因为 TransactionID 并非标记的一部份，且一般是由基础结构组件所掌控（而非由应用程序代码），所以没有直接的方式在这种呼叫（于标头中传递非标记信息）中传递必要的信息。藉由新增一个项目至标头中，并且为其命名一个固定名称，在接收端的交易管理员就可以萃取出 TransactionID。

8 安全性考量

在本文件中没有说明的论点即为某些专供整合性和私人信息保护的方法，本文件的较新版本将会说明这些考量论点。

9 参考数据

[1] S. Bradner, "The Internet Standards Process" - Revision 3, RFC2026, Harvard University, October 1996.

[2] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels", RFC2119, Harvard University, March 1997.

[3] E. Whitehead, M. Murata, "XML Media Types", RFC2376, UC Irvine, Fuji Xerox Info. Systems, July 1998

[4] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI):Generic Syntax", RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.

[5] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, U.C. Irvine, DEC W3C/MIT, DEC, W3C/MIT,W3C/MIT, January 1997

[6] H. Nielsen, P. Leach, S. Lawrence, "An HTTP Extension Framework", RFC 2774, Microsoft, Microsoft, Agranat Systems

[7] W3C 建议标准「The XML Specification」。

[8] W3C 建议标准「Namespaces in XML」。

[9] W3C Working Draft「XML Linking Language」，本草案还在修订中。

[10] W3C Working Draft「XML Schema Part 1: Structures」，本草案还在修订中。

[11] W3C Working Draft「XML Schema Part 1: Datatypes」，本草案还在修订中。

[12] Transfer Syntax NDR, in「DCE 1.1: Remote Procedure Call」。

[13] N. Freed, N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC2045, Innosoft, First Virtual, Novem-ber1996

A　SOAP Envelope 范例

A.1　呼叫请求的范例

范例 5　相似于范例 1，但是有强制性的标头

```
POST /StockQuote HTTP/1.1
Host:www.stockquoteserver.com
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn
SOAPAction:"Some-URI"


<SOAP-ENV:Envelope
```

```
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"


SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
      <t:Transaction
          xmlns:t="some-URI"
          SOAP-ENV:mustUnderstand="1">
             5
      </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
      <m:GetLastTradePrice xmlns:m="Some-URI">
          <symbol>DEF</symbol>
      </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

范例 6　相似于范例 1，但是有多个请求参数

```
POST /StockQuote HTTP/1.1
```

```
Host:www.stockquoteserver.com
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn
SOAPAction:"Some-URI"


<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"


SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
      <m:GetLastTradePriceDetailed
        xmlns:m="Some-URI">
          <Symbol>DEF</Symbol>
          <Company>DEF Corp</Company>
          <Price>34.1</Price>
      </m:GetLastTradePriceDetailed>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A.2　响应的范例

范例 7　相似于范例 2，但是有强制性的标头

```
HTTP/1.1 200 OK
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
    <t:Transaction
      xmlns:t="some-URI"
      xsi:type="xsd:int " mustUnderstand=="1">
        5
    </t:Transaction>
  </SOAP-ENV:Header>
```

```
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse
      xmlns:m="Some-URI">
        <Price>34.5</Price>
      </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

范例 8　相似于范例 2，但是有结构

```
HTTP/1.1 200 OK
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn


<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"


SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
     <m:GetLastTradePriceResponse
```

```
        xmlns:m="Some-URI">
          <PriceAndVolume>
              <LastTradePrice>
                  34.5
              </LastTradePrice>
              <DayVolume>
                  10000
              </DayVolume>
          </PriceAndVolume>
     </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

范例 9　相似于范例 2，但是强制性的标头发生错误

```
HTTP/1.1 500 Internal Server Error
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn

<SOAP-ENV:Envelope
```

```
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:MustUnderstand</faultcode>
            <faultstring>SOAP Must Understand Error</faultstring>
        </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

范例 10　相似于范例 2，但是掌控数据主体发生错误

```
HTTP/1.1 500 Internal Server Error
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <SOAP-ENV:Fault>
            <faultcode>SOAP-ENV:Server</faultcode>
```

```
            <faultstring>Server Error</faultstring>
            <detail>
              <e:myfaultdetails xmlns:e="Some-URI">
                <message>
                    My application didn't work
                </message>
                <errorcode>
                    1001
                </errorcode>
            </e:myfaultdetails>
        </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

< 全书完 >