

第16章 实例研究2——XML和分布式应用程序

XML是一种非常适于应用程序之间数据交换的格式。特别是对于松耦合的应用程序——如：基于Web的应用程序，这点非常重要。作为一种通信协议，HTTP具有跨平台性。对于应用程序数据来说，XML具有同等的功效。为了体现XML在这方面的优越性，我们将构造一个应用程序样例，并将它作为XML在Web应用程序领域的实例研究。

当然，这个例子完全是人为设置的。然而，其中的设计决策确实能够反映真正为企业编制软件时将做出的一些抉择。我们将使用 Active Server Pages，Internet Information Server和Internet Explorer 5.0构建应用程序。我们将介绍如何在脚本语言中使用XML和DOM。更重要的是，我们将说明XML能够解决哪些类型的问题。

我们的应用程序样例是一个编程小组管理工具。我们希望根据姓名搜索特定的程序员，并获得他所完成的项目的成绩报告清单。用户应该能够添加新的程序员记录，并给现有的程序员添加性能报告。我们会将XML作为浏览器和服务器之间数据通信的手段。同时利用Web页面中的JavaScript和服务端端的ASP JavaScript实现应用程序的功能。

之所以采取XML这类新技术，是因为它能够满足当前技术不能很好处理的一些需求。需要这样的暗示性问题，我们对应用程序特性的需求能满足当今计算环境。在分析实例之前，我们首先阐述一下Web开发人员面临的问题。为了帮助你解决这个问题，我将提出五条指导性原则。毫无疑问，XML是解决方案的核心。

本实例的大部分材料来自《Designing Distributed Applications》一书（Wrox Press，ISBN 1-861002-27-0）。该书详细说明了我在总结的弱点的根源和动机。本章即将出现的应用程序实例与该书中的应用程序略有不同，在本章中，该程序的作用是引出用于说明开发原则的扩展实例。这个实例是完全独立的。即使不参考以前的书，你也完全能够通过本例理解XML在ASP应用程序中的作用。

16.1 目前的弱点

近年来，Web应用程序非常流行。许多新的应用程序都是用Web协议和工具构建的。Web似乎成为一种出色的计算环境。对于这种赞誉，它自然当之无愧，然而它同时对程序员提出了一些新的挑战。如果我们不对此做好准备的话，它们将发展成为我们将面临的主要问题和弱点，并最终削弱构建功能强大的应用程序的能力。然而，到底有哪些弱点呢？

1. 处理不断增长的复杂性

功能强大的站点要解决的问题常常与独立的应用程序一样复杂。而且，分布式系统本身的特性进一步增加了它的复杂度。无法管理的复杂性是分布式系统的一个弱点。随着后期开发的客户端对于资源的含义和功能越来越不了解，资源的价值也变得越来越微弱。

由于基于 Web 的应用程序很容易实施，它不需要经过任何人的允许就能够添加到 Internet 上（除了需要获得域名之外），因此 Web 服务器和服务迅速发展和成熟。虽然有人可能理解他们能够执行的操作，但是多数本来能够使用某些功能的用户却不知道存在这些特殊的功能。功能将被一次次不必要地重复实现，仅仅因为没有定义用于说明可用功能的机制。这在公共的 Internet 中尚可忍受；但是在企业内部网中，这不仅是一种浪费，而且也是企业网存在的弊病。

2. 不灵活的应用程序

应用程序，甚至是 Web 应用程序，也常常是针对特定的客户端编写的。如果改变客户端的需求，你必须重写服务器端程序，或者创建极其类似的新服务。

3. 代码的重复

新的应用程序常常重复大量现有的代码，因为需求和规范的变化很小。代码的重复可以视为是一种极大的浪费。

4. 向自动化 Web 任务的转移

假设我们有一个 Web 应用程序，它通过基于浏览器的界面与用户交互，完成用户要求的任务。现在，我们希望将它变为自动的过程——消除手工干涉，直接将两个应用程序连接在一起。但是，存在这样一个问题。服务器以前产生的 HTML 是供人类使用的。然而，我们越来越期望将应用程序连接在一起，消除其中的手工干预。基于 Web 的 B2B 贸易就是实例之一。如果 B2B 过程的每一步都由用户读取结果并重新提供输入，就会使电子商务的许多优越性丧失殆尽。但是，在实现任务自动化的过程中，我们却发现 HTML 很难保存数据的含义。HTML 的主要用途是提供可视化的样式，而不需要人类干预的应用程序对此并不感兴趣。

5. 分布式开发和实施小组

当某些组织机构提供的服务不在预期的客户端控制之下时，就会遇到麻烦。

例如，在你建立的企业外部网供应链应用程序中，某些服务可能必须依赖于外部合作伙伴。当协作不正规或者不存在协作时，服务将转移、改变或消失。为了跟上环境的变化而产生的管理开销将迅速增长。为此，我们需要依靠新的技术和服务构建灵活、强健、能容错，且能够根据需求变化的分布式系统。更重要的是，在实现这一目标时，应该消除人类的干预。

16.2 构建网络应用程序的五条原则

只要开发小组的每个成员能够就实现应用程序所需的工具和技术达成一致，就能够利用众所周知的技术克服任何弱点。然而在 Web 上，要达成这种一致是不现实的。依我看来，可行的解决方案是建立一个较短的公共原则列表。这个列表必须足够短，保证每个人不必做大的改动或技术承诺就能够同意。我提出了五条核心原则。

这五条原则包含了开发应用程序所需获得的目标，它将促进应用程序代码的重用，提高应用程序在面对需求和程序变化时的适应能力。

构建协作网络应用程序的五条原则包括：

(1) 从粗粒度服务构建应用程序——应用程序应该协调来自基于服务器的模块的离散结果，这些模块使用各种部件解决特定的问题或完成特定的任务。

(2) 通过查询目录发现服务——应用程序应该在运行时通过查询目录寻找所需的服务的位置

和名称。它不应该寻找特定的实现，而是应该寻求能够解决特定问题或任务的任何服务实现。

(3) 将服务提供为自描述数据——应用程序应该通过交换结构化数据处理服务，这些数据符合事先约定的语法。而且，它们是根据专为服务所要解决的问题或者要执行的任务定义的词汇表编写的。

(4) 服务是短暂的——应用程序查找和使用服务只需要很少的几个（最好是一个）往返，而且在往返过程中不必保存状态信息。在交互过程中，应用程序和服务都不认为对方是长期可用的。

(5) 服务必须可扩展，且能够降低对外部的要求——服务必须考虑到将来的增强，比如：自身逻辑的增强，或者本身与其它应用程序和服务交换格式的增强。遇到新版本的交换数据时，服务应该尽可能利用这些数据。当应用程序和服务接收到不符合要求的格式时，都不能崩溃。

当然，必须根据当前的技术和网络中可能出现的异构平台判断以上原则是否现实。另外，合作伙伴和开发人员之间还要达成一定程度的共识和协作。我们希望侧重于利用现有的技术遵循以上原则实现应用程序。因此，我们将依次分析这些原则，并讨论如何实现这些目标。

16.2.1 从粗粒度服务构建应用程序

COM组件和控件等软件组件虽然是非常有用的工具，但是它们只能在支持相应组件技术的平台上运行，且只能在定位于这些平台的应用程序中重用。COM组件不能用于Unix平台，同样JavaBean不能在非Java服务器上使用（当然，某些Unix平台有COM端口，但是一般而言，COM是一种Windows技术）。在我们的例子中，开发小组只创建应用程序中的一小部分，为现有的服务器创建新的客户端，我们不能用组件跨越应用程序的各个层次。实际上，我们应该依靠服务。服务是任意应用程序代码的集合，它限制于一层，完成一项明确定义的任务。服务比组件更大；事实上，为了提高效率，我们经常用组件构建服务。服务比应用程序小。服务将问题域的一部分模型化。如果服务过小，数据格式转化产生的开销将威胁到程序的性能。如果服务过大，它将面临由单一模块构成的大型应用程序存在的问题。

将我们的定义与Windows DNA或Windows NT下服务的定义相比较。在Windows中，服务是指与操作系统挂钩的应用程序——Windows 32 API服务——或者用于获取数据的组件框架，如：ActiveX数据对象（ActiveX Data Object，ADO）。它们都比单一的组件大。ADO之所以不符合我们的定义，是因为它能够依靠专有的组件技术跨越应用程序的边界。然而，除此之外，它还是与我们的定义吻合的——它利用几个组件之间的交互回答特殊的问题：“根据SQL查询返回结果集合”。

通常，服务会将底层业务模型中的一些主要对象模型化——例如：销售应用程序中的客户，或者制造应用程序中的生产线。

我们的开发原则设想的服务类型通常实现为服务器端页面或者一小组相互协作的页面。它们使用已知的数据格式表示服务中商业对象的持续性状态。并且利用一组相关的对象提供各种功能。例如，除了纯数据格式之外，服务还将提供HTML，以便这些数据能够被手持设备等极瘦的客户使用。瘦客户虽然不能享受通过程序操作数据带来的益处，但是它也不必受由纯数据格

式产生的额外开销而带来的困扰。虽然它能够准确地解析和操作 HTML，但是HTML本身是一种标记数据表示的语言，它并不说明数据的语义。实际上，支持不同的交换格式比使用 HTML标记数据更加简单。简而言之，增加HTML支持是一种表面上的改变，它使得简单的平台能够加入这种有限的流行趋势，只要它们支持标准的 Web浏览器。我们总是可以通过开放的协议访问服务。通过将主要的对象抽象化，我们不仅能够保持面向对象模型的优势，而且能够减少对专有组件技术的依赖性。

由于一项服务是由一个组织开发并维护的，而且位于由该组织机构控制的站点上，因此可以在服务中使用更加专有的技术以提高效率。我们希望在服务中使用组件软件（通常，我们会给某些遗留的软件或数据库增加外包装，避免重写大量代码）。我们将充分利用宿主操作系统和Web服务器的各种功能，从站点中获得最大的价值。然而，当需要与另一个服务或客户交互时，我们应该认识到这有可能跨越组织机构的边界，因此必须使用开放的标准和格式（参见图 16-1）。

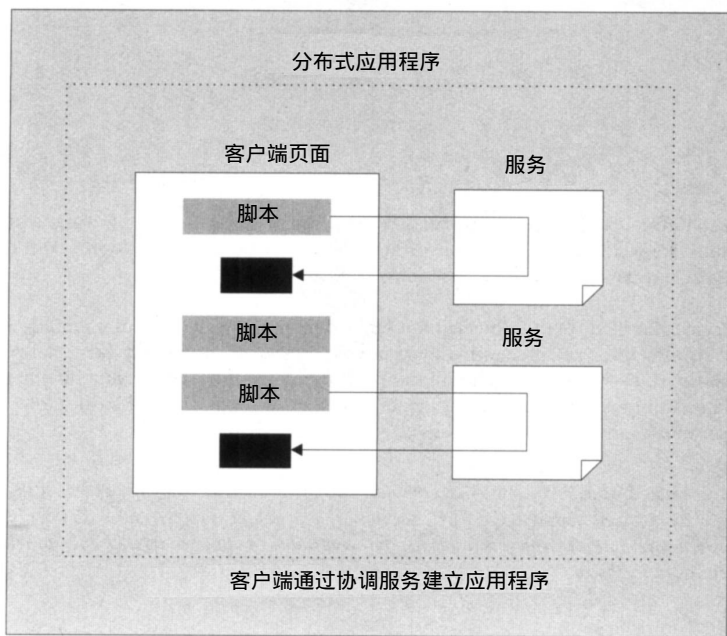


图 16-1

我们的应用程序将建立在可重用的服务基础之上。在服务的内部，我们将依赖于组件。应用程序通过精确定义的接口使用服务，服务通过接口使用内部的组件。因此，我们将受益于封装以及将任务委托给大的服务和小的组件。

16.2.2 通过查询目录发现服务

服务的网络位置和服务提供的格式定义了对服务的访问。我们必须将这些因素与任何本地约定相分离。虽然目录尚未成熟，也未被广泛应用，但是我们仍然要将它作为服务定位策略的基础。它们对于实现位置抽象和动态资源发现来说非常关键。仅当资源能够被网络中的所

有用户可见时，资源才有价值。因此，我们确信目录服务将受到信息系统组织最高层管理人员的关注。它的结构和服务器的位置将成为组织内讨论的主题，我们确信该领域能够达成一致的意见和理解。

目录不仅仅要保存服务的位置，而且要定义服务在它们所要解决的业务问题中的功能（参见图16-2）。

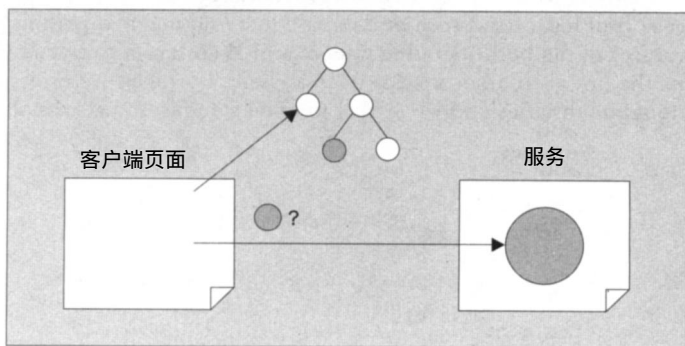


图 16-2

我们必须定义有价值的目录结构，并且利用工具对目录进行分析。每个应用程序都必须浏览目录，以便搜索与所需服务相关的信息，因此我们要尽可能将该任务标准化。

目录服务使得我们能够对应用程序隐藏服务实现的物理网络位置。应用程序可以在运行时动态发现所需的数据源——服务。在应用程序运行过程中，它可以从网络添加或删除网络服务。如果你小心翼翼地坚持这条原则，你的应用程序将变得非常灵活，能够适应资源位置和可用性的变化。

《Designing Distributed Applications》为实现这条原则提供了一种推荐的方案，它在服务表示中包含XML词汇表名称。由于目录服务目前尚未广泛采纳（大多数读者也不能使用它），因此在我们的应用程序样例中仅提供stub实现。

16.2.3 将服务提供为自描述数据

为什么我们对数据交换格式如此关注？毕竟，我们坚持了在组件软件中使用的面向对象方法，这些方法试图对数据的使用者尽量屏蔽数据的格式。但是我们需要例外，根据服务的定义，我们要分离服务中的组件。当跨越服务页面和客户之间的平台边界时，必须以一种所有客户都能够访问的形式表示服务实现中的对象。尽管近年来基于对象的分布式计算有了一定发展，但是这种形式仍然是静态数据结构。

按照元数据定义对象的特征，每个特定的对象实例用符合元数据定义的数据结构表示（参见图16-3）。由于需要特别的灵活性和应付错误的健壮性（参见原则5），我们将使用自描述数据，即：XML。这意味着每个离散的数据元素被标记为XML文档的元素或属性。数据的使用者总是能够知道它所处理的是什么元素（无论它期望的是什么），以及元素在何处结束，因为元素是用表示元素内容的标签标记的。因此，任何理解服务词汇表的使用者都能够理解数据提供者要表

达的含义。

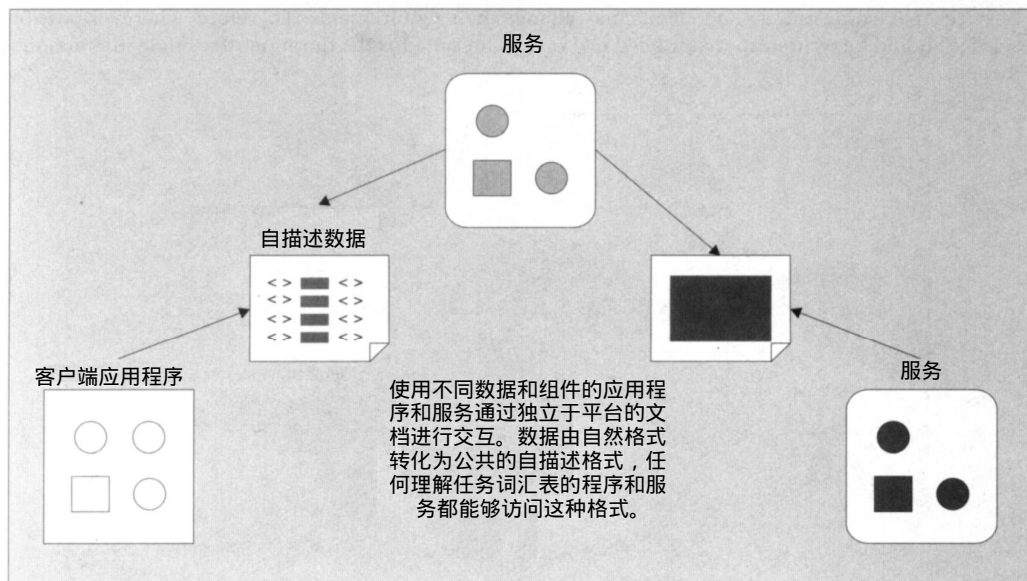


图 16-3

传统上，数据是以二进制格式传递的，它隐含着数据结构是共享的。如果通信双方有相同的消息或公共的操作，意味着它们能够理解数据的二进制格式。数据的大小、数据类型以及数据结构中各字段的顺序是在设计阶段严格定义的。服务器端自主地改变数据结构会导致客户端的崩溃。你可能参加过编程小组开发大型客户-服务器应用程序，其中部分程序分配给其他程序员完成。当你编写完处理特定数据结构的客户端程序后，却在某一天发现应用程序无法正常运转。经过调查，发现另一个程序员在未通知你的情况下，已经根据增强程序功能的请求或错误报告增加了一个字段，或者修改了现有字段的长度。你所负责的程序因为数据的随意变化而崩溃。如果服务器及其结构是由分布于不同地理位置和时区的许多编程小组共享的，相信你不难想象其中隐藏的麻烦。

利用XML对数据进行编码不仅减小了出现这种麻烦的可能性，而且尽量扩大数据共享的范围。使用数据的客户或服务可以忽略数据结构中无法理解的变化部分。由于我们有众所周知的词汇表，因此能够标记结构中的每个字段。每个字段以字段的名称作为明确的分界线。所以，我们总是能够逐个字段地验证数据的完整性，并使代码与真正接收的数据而不是设想要接收的数据相对应。

16.2.4 服务是短暂的

在应用程序设计方面，必须保证持续性状态是由在计算过程中需要该状态的层单独保留的。举例来说，购物代理对于购物者的标识和购物清单感兴趣，因此该信息应该由购物代理来维护，而不是由代理访问的销售商服务保存。这不仅符合 HTTP 无状态的特性，而且能够减少网络上机

器之间的依赖性。

服务有可能变化不定，客户端也许会改变它们的需求。因此，编写服务代码时，应该尽可能保证它只需要维护与客户端进行一次交互过程中所需的状态信息（参见图 16-4）。

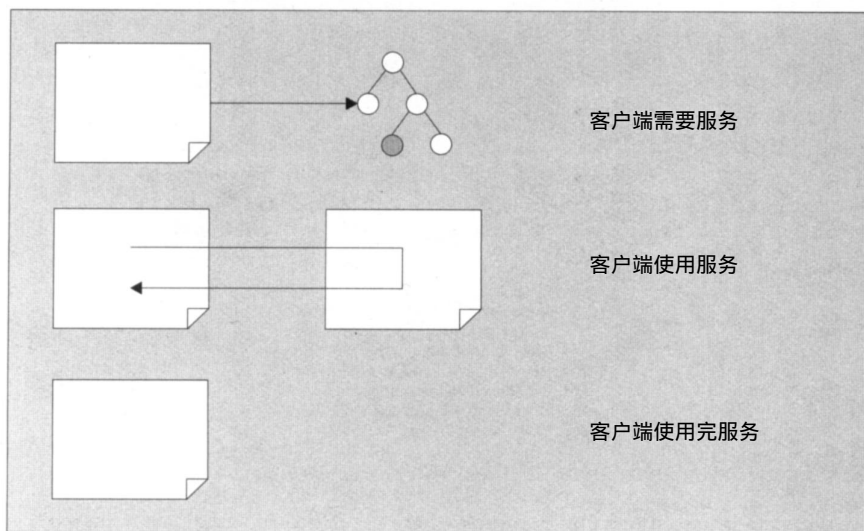


图 16-4

这并不总能实现，但是你应该将它作为自始至终追求的目标。应用程序是客户端的集合，这些客户端通过发出获取数据的 HTTP 请求使用服务。一旦服务端发送了数据，就认为客户端接受了服务端的服务。我们并不打算维护分布式系统中的状态信息，而是努力做到不需要维护信息。客户端将获取它所需要的所有数据。

从组织机构的角度考虑，这条原则充分认识到随着时间的推移不同的开发小组会改变他们的优先级。今天为了维护状态而达成的一致意见在将来可能发生变化。这条原则尽力消除了达成这种一致的必要性。我们将限制保存资源的时间，以减少在需要时丢失资源的可能性。以访问一个繁忙的 Web 站点为例，当你访问一个页面时，它可能出现故障或者崩溃。如果该页面包含一个表单，你正在为了采购而输入信息，如果提交表单后得不到应答，你会非常生气。HTTP 并不能对资源提供保证，但是特定站点的应用程序隐含着资源的长期可用性。

16.2.5 服务必须可扩展，且能够降低对外部的要求

相信你没有忘记，我们潜在的问题是要使用独立的开发小组。时差、距离和组织机构界限将这些小组分隔开来。我们完全能够想象在数据格式的实现上必然存在着一定的偏差。即使实现是准确无误的，当新的实现版本发布时，广域网上也常常存在着不同版本的数据。由于 XML 能够对数据进行标记，因此我们可以通过读取标记判断文档的内容，并做出正确的应答。当然，如果使用验证有效性的解析器，就能够消除这些错误，但这有可能不是我们所期望的。在某些情况下，我们宁愿关闭有效性验证，接收格式正规但有一些小错误的 XML（参见图 16-5）。

即使坚持使用有效的 XML 文档，我们仍然会对这条原则感兴趣。由于不同的组织机构有不同的公共对象视图，因此我们希望将数据表示为既能够描述数据的哪部分是公共的，又能够说明哪部分是组织特有的。类似地，随着时间的推移，数据格式会逐渐发展，所以我们将希望描述数据格式发生了哪些变化。这样，当客户端访问的服务所提供的数据与客户端的想象稍有差别时，它仍然能够从中提取部分信息。我们将利用 XML 的标记和自描述特征表示我们的问题词汇表。由于它完全是以文本方式书写的，因此我们有理由相信即将遇到的任何计算平台都能够读取我们的数据结构。我们可以设计一些用于指定 XML 词汇表的约定，以便表示集合、版本发展和通用格式规范。这些约定超出了编写 DTD 和模式的规则。根据这些约定编写的软件能够识别表示商业对象的数据结构集合，能够接受它所需要的特定版本的数据信息，当它接收到无法识别的特殊格式时，它能够提取其中的通用数据版本（参见图 16-6）。

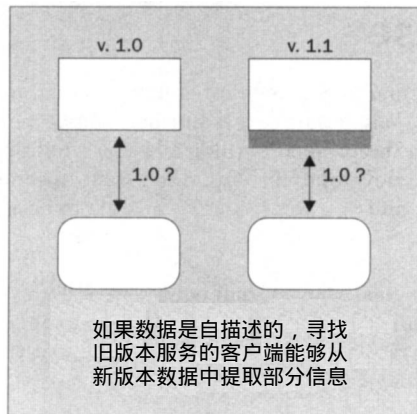


图 16-5

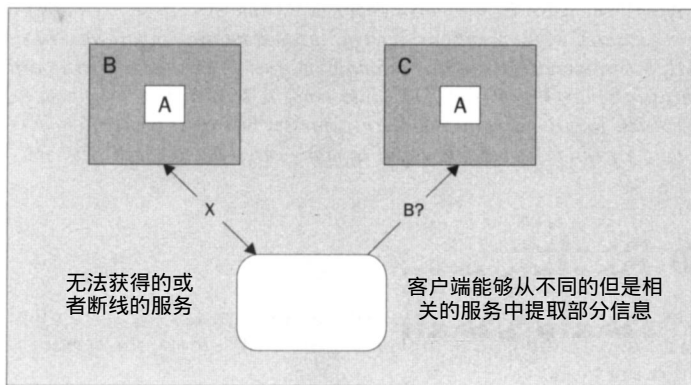


图 16-6

这些原则都是非常宏伟的目标。一个实例研究和一个基本的例子不足以说明这些原则的各个部分。我们即将讨论的这个实例将用于解释在 Web 应用程序中使用 XML 进行数据交换的基本技术。你应该将这五条原则牢记在心，并且经常考虑如何利用它们改进代码。在此过程中，我将为你提供一些建议。

16.3 商业实例

假设你负责管理的部分由若干计算机程序员组成。自从部门成立以来，每个程序员都被分配了多个项目。当一个项目完成时，你要记录一段简短的说明，描述程序员在项目中的表现。你会定期查看这些说明，对程序员的专业开发能力进行评价。你不打算花费大量时间和精力开发能够完成这项任务的应用程序，你只是希望从网络中的任何一台计算机都能够访问这些数据。

企业内部网的应用程序似乎是理想的选择。任何配备了公司标准浏览器（根据我们的设计，它应该是Internet Explorer 5.0）的系统都能够运行该应用程序。ASP和客户端脚本能够迅速构造出满足我们需求的系统原型。应用程序将使用 JavaScript作为它的脚本语言。

刚刚学习了五条开发原则，你一定渴望着将它们付诸实践。如果你从客户端通过 ADO远程访问数据，就会使数据中蕴含着技术依赖性。如果你在服务器上使用 ADO获取数据并产生HTML——该问题领域最通用的Web开发技术——你的服务器代码将受到使用浏览器的人类用户的限制。这意味着商业过程中要包含手工步骤。现在，你决定在服务器上使用 ADO，并将数据转化为XML传输到客户端。通过这种方式，就能够保持数据的原始面貌，仅仅在最后一刻才将它转变为可视化的样式。由于将XML作为数据交换格式，因此数据采用了一种独立于平台的格式。你的ASP将采取服务的形式。客户端将向服务器发送请求（XML文档），并接收来自服务器的应答（另一个XML文档）。当你要查询某项服务时，通过获得它的URL定位服务。从理论上讲，网络管理员在服务调用期间可以在客户端应用程序不知情的情况下移动或更换服务。客户端将通过指定要获得的XML词汇表的名称询问服务的位置。

Microsoft最近给ADO和SQL Server增加了XML支持。然而，除非关系模式恰好与我们的约定相匹配，否则这种自动化的支持不能为我们提供任何帮助。在本例中，为了使用表达集合的约定，我们将通过程序构造XML。XML数据是趋于层次化的，它能够巧妙地表达大多数商业对象。然而，关系数据不善于表达层次结构，它只能通过复杂的链接。Microsoft提供的XML支持在许多情况下都非常有价值，但是不适合我们的这个例子。

16.4 应用程序设计

对于这个应用程序来说，用户界面设计成功的关键在于减少服务代理需要查看的页面数。在客户使用系统的过程中，服务代理将执行以下任务：

- 在系统中查询程序员的信息（全名和职称）。
- 增加新的程序员，更新现有的程序员记录。
- 在系统中查询项目成果说明。
- 输入新的成果说明。

我们应该能够将所有内容放在一页中，并用水平线进行分隔。水平线之上是显示一个程序员信息的数据绑定表单。水平线之下是项目成果历史表，每个单元格对应一个项目成果说明，其中包括：项目名称、项目ID和成果说明文本框。图16-7显示了页面的内容。

页面上半部分左侧的四个按钮用于浏览记录，以便寻找与要查看的成果说明对应的程序员。用户可以显示集合中的第一个或最后一个程序员，可以每次向前或向后移动一条记录。程序员表单中的元素是与服务器返回的XML绑定的。

位于同一行的其余按钮用于执行以下功能：

- 输入新的程序员。
- 编辑现有的程序员信息。
- 获取特定程序员的项目成果历史，它将显示在页面的水平线之下。

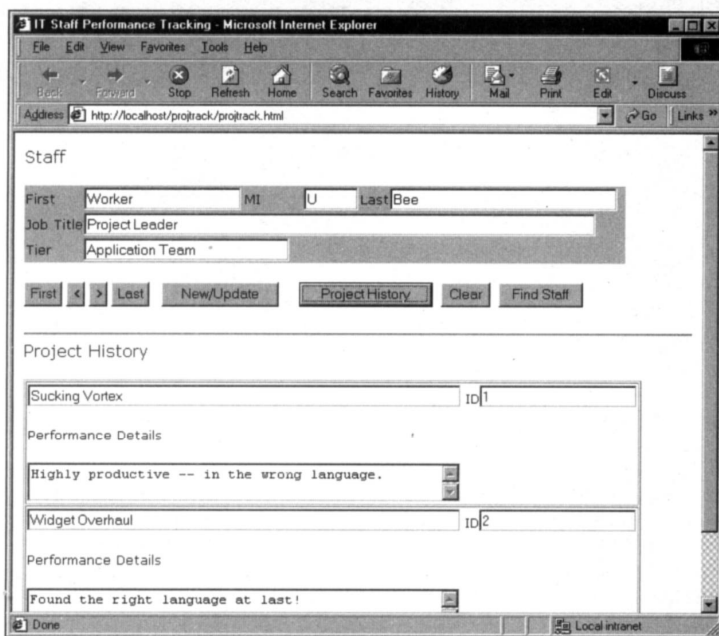


图 16-7

- 清除程序员表单，准备搜索特定的程序员或插入新的程序员数据。
- 搜索现有的程序员信息。

页面的下半部分用于显示程序员的成果历史，或者插入新的成果报告（参见图 16-8）。

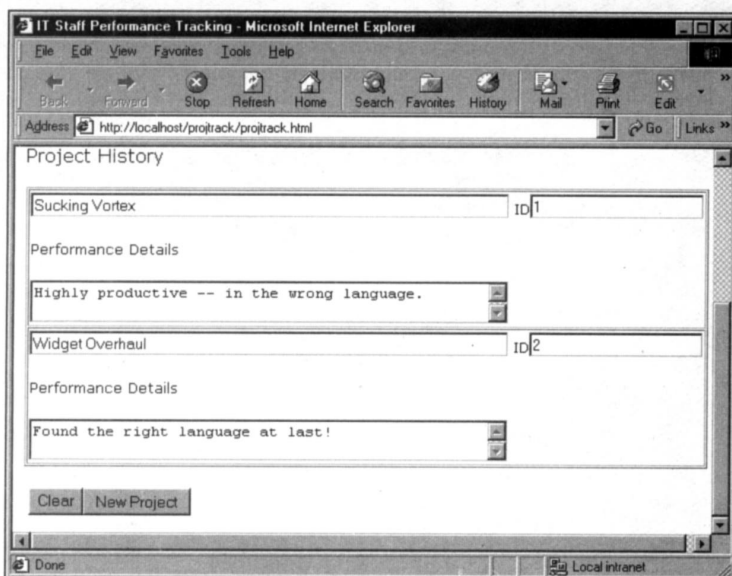


图 16-8

这个应用程序可以设计为直接与程序员信息数据库绑定。然而，我们提出的原则鼓励我们对应用程序隐藏数据存储的细节。我们应该将元素与程序员和程序员成果历史服务绑定。这些服务将直接对数据库进行操作，但是它们将以 XML 文档的形式与客户端交流信息。这样，客户端有足够的灵活性选择各种方法，服务器端也可以自由地改变数据库和数据库模式。

如果你希望亲自尝试这个应用程序，建议你从我们的Web站点<http://www.wrox.com/>下载代码和样例数据库。数据库中包含程序员，以及程序员 Genius，Authority，Doe，Smith，Typhoid，Calamity，Sullivan，Bee和Greeley的成果说明项。

16.4.1 应用程序的组织

下面我们将对应用程序进行整体概述（参见图 16-9）。页面 ProjTrack.html 将下载到客户端。当用户对页面中的元素进行操作时，脚本将在本地执行。这些脚本将确定它们所需的服務的位置。服务位于 Web 服务器上，它们实现为 ASP 页面 Staff.asp 和 History.asp。请求以 XML 文档的形式发送给服务，其中文档是根据适当的请求词汇表编写的。服务将解析请求，查询数据库 projects.mdb，并根据查询的结果利用应答词汇表创建新的 XML 文档。客户端接收应答，进行解析，并将结果显示在数据绑定的 DHTML 表单中。需要注意的是，服务器对客户端隐藏了数据存储和获取的细节，客户端对服务隐藏了使用 and 显示数据的问题。以独立于平台的方式交换数据的原则保证了独立性，因而也提高了重用的可能性。例如，你可以将客户的页面 ProjTrack.html 替换为报告编写和总结程序，这些改动不会对服务器端有任何影响。

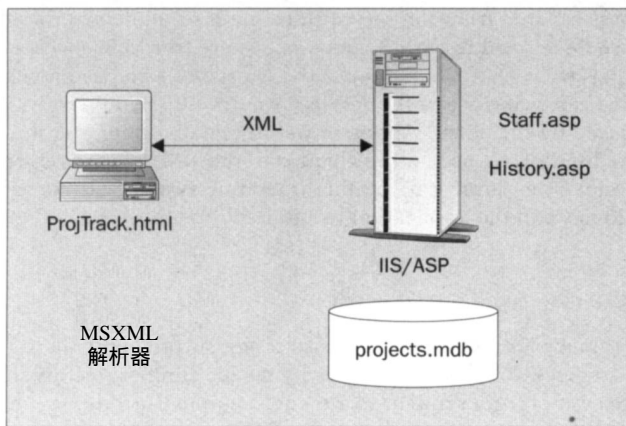


图 16-9

16.4.2 程序员服务客户

应用程序的客户只包含一个 HTML 页面。程序设计管理员永远都是通过该页面使用应用程序的。然而，客户和服务端之间存在着密切的交互和通信。客户通过查询两个服务获得与程序员和程序员成果历史相关的数据。该数据分别用两个 XML 数据词汇表进行编码。每个词汇表都是与一个查询词汇表对应的。客户读取表单中的值，编码为适当的查询，并提交给适当的服务。

当返回数据时，XML是与表单绑定的。当程序设计管理员输入新的程序员数据，或者为现有的程序员添加新的成果报告时，应用程序将该信息封装在数据词汇表中，并提交给服务器。

我们的系统使用两个对象类：ITStaffers和ProjReports。从程序设计管理员的角度看，ITStaffers是从父公司的通用Staffer类发展而来的，它增加了Tier属性。该属性反映了程序员所负责的模块在三层编程模型中所处的层次：客户、应用程序逻辑或者数据服务。理想情况下，负责开发管理ITStaffer对象的服务的程序员应该让服务器同时处理ITStaffer和Staffer词汇表。这样，服务就能够与任何能处理Staffer信息的应用程序共享ITStaffer信息的Staffer子集。这种设计方式符合原则5，降低服务对数据的要求。对于IT部门及其程序员来说，项目成果报告历史是唯一的。由于这些报告对于ITStaffers是唯一的，因此最好通过另一个服务管理数据，而不是使用相同的服务处理ITStaffers和Staffers。

当然，说实话，本章的整个实例研究都是我们设计的，因此根本不存在Staffer处理代码。而且，ITStaffers相对于Staffer而言特殊化之处很少。然而，考虑一下现实世界的员工。他们应该有详细的描述，而不仅仅是姓名和职称。私人信息，例如：美国的社会保障号，应该受到特殊代码的保护，防止未授权的访问。通过在词汇表中增加特殊的限制创建ITStaffers时，也要包含类似的代码。例如，我们可能要维护程序员所掌握的编程语言和熟悉程度的列表。在这种情况下，Staffers和ITStaffers的代码应该明确地区分开。

总之，客户是与管理数据的服务器端代码分离的。只要客户使用我们在前几章介绍的技术，在服务器端就可以随意选择任何数据库或文件管理技术。更重要的是，任何人只要知道我们定义的XML词汇表，就能够使用我们开发的服务。相反，服务器与客户端采用的技术无关。虽然我们使用了数据绑定，服务不一定要支持这种绑定。我们能够将服务移植到不支持Windows数据管理技术的另一种操作系统上，应用程序照样可以运行。

16.4.3 服务

我们开发的服务是用ASP实现的。这些ASP通过ADO访问Access数据库中的程序员数据，这是目前最简单的技术。由于整个系统是从头开始开发的，因此我们能够自由选择适于我们的平台并且能够有效满足系统需求的技术。然而，现实中的许多组织机构在主机系统上存有遗留的数据。他们只能通过消息或某些专有的API访问这些数据。

系统提供的两种服务具有类似的格式。它们从名为XMLRequest的表单元素中提取值，解析XML，并确定所提交的是哪个词汇表。根据我们的约定，程序员和项目成果报告应该封装为集合。即使特殊的客户每次只提交一条程序员记录或者一个成果报告，在客户和服务之间来回传递程序员和报告集合也是非常合理的（实际上，服务器常常会向客户传递这种集合）。一旦服务器确定了所用的词汇表，它产生SQL查询语句并执行它，然后以XML文档的形式将数据返回给客户。从查询的角度而言，它是一个包含零个或多个程序员或项目成果报告集合。从客户的提交而言，它是一个对客户的肯定或否定的确认。

ASP是纯粹的脚本，它只返回XML。其中不包含HTML、HEAD或BODY元素。它使用了MSXML和ADO对象，但是这些都是动态创建的。因此，ASP返回的XML不是完整的Web页面，

而更像是一小段程序片段。

16.4.4 交换词汇表

为了实现客户与服务器之间的数据交换，我们的应用程序需要四个 XML 词汇表。客户端需要向服务器请求符合某些标准的程序员，以及特定程序员的报告。相反，服务器需要返回匹配程序员查询的程序员集合，以及匹配成果报告查询的报告集合。另外，客户要使用程序员和报告集合词汇表向服务器传送新的程序员和报告。

1. 程序员查询

我们决定允许应用程序根据程序员的姓氏和员工 ID 查询 ITStaffer 服务。由于我们的用户界面不显示员工 ID，因此对于该字段，这个特殊的客户端将提交空元素。下面是 StaffQuery 文档的外壳：

```
<StaffQuery>
  <StaffID/>
  <LastName/>
</StaffQuery>
```

虽然我们不使用 DTD 验证文档的有效性，但是以上外壳可以更加正式地定义为：

```
<!ELEMENT   StaffQuery   (StaffID, LastName)>
<!ELEMENT   StaffID      #PCDATA>
<!ELEMENT   LastName     #PCDATA>
```

2. 项目成果查询

对于一个较大的部门，员工查询会导致向客户应用程序返回多条记录。每条记录都包含一个员工 ID，当我们需要搜索特定程序员的成果报告时，可以根据这个 ID 进行更精确的查找。实际上，我们之所以需要这种精确性，也是为了避免混淆多个程序员的报告记录。下面是 ProjPerformanceQuery 的外壳：

```
<ProjPerformanceQuery>
  <StaffID/>
</ProjPerformanceQuery>
```

它对应于以下 DTD：

```
<!ELEMENT   ProjPerformanceQuery   StaffID>
<!ELEMENT   StaffID                 #PCDATA>
```

3. 程序员应答

对于 StaffQuery 查询，应答中返回的文档由包含一个或多个 ITStaffer 元素的 Collection 构成。如前所述，ITStaffer 是 Staffer 的特例。Staffer 元素包含用于表示姓名、员工标识符和职称的元素。ITStaffer 元素除了包含 Staffer 元素之外，还包含用于表示程序员所负责的编程层次的元素。所有元素都是必须出现的。下面是包含一个 ITStaffer 元素的文档外壳：

程序清单 16-1

```
<Collection>
  <ITStaffer>
    <Staffer>
      <StaffID/>
```

```

    <FirstName/>
    <MI/>
    <LastName/>
    <JobTitle/>
  </Staffer>
  <Tier/>
</ITStaffer>
</Collection>

```

以上文档对应的 DTD 有些特殊。我们将 Collection 作为文档类型。严格意义上讲，它将转化为一个 DTD，但是我们希望将 Collection 作为能够包含各种对象的通用容器。有些人认为，每次使用 Collection 时，都需要声明命名空间，然而我们不希望如此严格：

程序清单 16-2

```

<!ELEMENT   Collection      (ITStaffer*)>
<!ELEMENT   ITStaffer      (Staffer, Tier)>
<!ELEMENT   Staffer        (StaffID, FirstName, MI, LastName, JobTitle)>
<!ELEMENT   StaffID        #PCDATA>
<!ELEMENT   FirstName      #PCDATA>
<!ELEMENT   MI              #PCDATA>
<!ELEMENT   LastName        #PCDATA>
<!ELEMENT   JobTitle        #PCDATA>
<!ELEMENT   Tier            #PCDATA>

```

4. 程序员成果报告历史应答

成果报告应答文档 ITStaffer 由包含零个或多个 ProjReport 元素的 Collection 构成，其中每个 ProjReport 元素必须包含用于表示项目名称和项目 ID 的元素，以及包含形式随意的文本描述的元素，其中的描述是由程序设计管理员输入的：

程序清单 16-3

```

<Collection>
  <ProjReport>
    <ProjName/>
    <ProjID/>
    <PerformanceDetails/>
  </ProjReport>
</Collection>

```

下面是我们使用的 Collection 变种：

程序清单 16-4

```

<!ELEMENT   Collection      (ProjReport*)>
<!ELEMENT   ProjReport      (ProjName, ProjID, PerformanceDetails)>
<!ELEMENT   ProjName        #PCDATA>
<!ELEMENT   ProjID          #PCDATA>
<!ELEMENT   PerformanceDetails #PCDATA>

```

16.5 实现

负责开发本章应用程序原型的编程小组在开始正式编程之前首先熟悉了我们的设计理念。

除了了解系统及 XML 词汇表中的类，他们还学习了如何使用组件。编程小组决定使用 MSXML DOM 接口解析和操作交换数据；使用 MSXML 的 IXMLHTTP 接口实现通过 HTTP 在页内传输 XML 文档；使用 MSXML 的 XML DSO 进行数据绑定。他们使用 ADO 获取、插入和更新关系数据库中的数据。

16.5.1 定位服务

根据前面介绍的五条原则，定位服务的正确方法是：每次当客户端需要向服务器发送查询时，通过搜索目录获得适当的 URL。网络管理员应该在目录中以相关的词汇表名称为条目列出所有服务。在我们的例子中，History.asp 应该与 ProjPerformanceQuery 和 PerformanceReport 相关。由于本书的核心内容不是 LDAP 目录，因此我不打算详细说明查询的过程。客户端函数 GetASP() 以词汇表名称为参数，返回不含 http:// 的 URL：

程序清单 16-5

```
function GetASP(svc)
{
    switch (svc)
    {
        case "Staff":
            return "localhost/ProjTrack/Staff.asp";
        case "StaffQuery":
            return "localhost/ProjTrack/Staff.asp";
        case "PerformanceReport":
            return "localhost/ProjTrack/History.asp";
        case "ProjPerformanceQuery":
            return "localhost/ProjTrack/History.asp";
    }
}
```

以上代码假设你已经创建了名为 ProjTrack 的虚拟目录，其中包含应用程序的资源，而且该目录与程序的页面位于同一 HTTP 服务器上。对于本例来说，这两个假设是很容易实现的。如果你打算将以上代码用于其他应用程序，应该对代码进行适当修改。

当我们需要知道服务的 URL 时，将调用函数 GetASP()，并以所需的词汇表名称作为参数。对于 GetASP() 的返回值，我们需要添加前缀 “http://”，并将结果作为 XMLHTTP 组件的 URL。因此，要发送对应于词汇表 StaffQuery 的请求，需要以下两行代码：

```
var sServer = "http://" + GetASP("StaffQuery");
xmlStaffd.Open("POST", sServer, false);
```

《Designing Distributed Applications》介绍了通用的 LDAP 和 Windows 2000 ActiveDirectory，因此我在将服务列在目录中，并通过两个组件实现 GetASP() 的功能。如果你打算尝试抽象定位，并且没有 ActiveDirectory 或其他 LDAP 目录服务，你可以将 URL 保存在数据库中。当然，这种方法的缺点是在实际应用中每个人都必须连接公共的数据库。如果你希望做进一步的改进，可以将对数据库的访问封装在众所周知的 ASP 页面中，这个 ASP 页面的功能就如同我们的 GetASP() 函数。

任何支持特定数据词汇表（如：Staff）的服务自然也能够理解相应的查询词汇表（如：StaffQuery）。如果不存在数据词汇表，查询词汇表就没有意义；同样，如果没有提交数据时的查询词汇表，数据词汇表就没有意义。理解了查询词汇表与数据词汇表之间的链接关系，在定位服务时就可以随意请求查询词汇表或者数据词汇表。有些程序员认为理所当然应该请求数据词汇表，因为这是我们真正感兴趣的内容。其他程序员认为应该请求要发送给服务器的词汇表，因为这是服务必须处理的词汇表。请求数据词汇表的优势在于双方都能够非常清楚地知道要返回给客户的内容。然而，请求查询词汇表可以确保服务器能够处理你即将发送给它的内容。如果你怀疑特定的查询词汇表是否与相应的数据词汇表配对，应该请求查询词汇表。否则，客户端应该请求数据词汇表。在服务定位的设计中，应该保持松散的耦合，以便新的应用程序与旧的服务集成。

16.5.2 管理数据绑定

页面ProjTrack.html中用到了数据绑定。我们利用该技术在页面的两个表单中显示程序员记录和成果报告记录。成果报告表单是一个表格，我们通过数据绑定多次重复一个模板行，显示一个程序员的所有成果历史。这不是一次能够完成的。一次典型的用户会话是由一个管理器管理的，它要执行多次对程序员和历史的搜索，其中还要插入程序员和成果报告。我们必须注意不要让绑定断开。

为此，ProjTrack.html包含两个数据岛，分别对应于两个表单。它们是位于页面 <BODY>元素中的空外壳：

程序清单 16-6

```
<XML id=xmlStaff>
<Collection>
  <ITStaffer>
    <Staffer>
      <StaffID></StaffID>
      <FirstName></FirstName>
      <MI></MI>
      <LastName></LastName>
      <JobTitle></JobTitle>
    </Staffer>
  </ITStaffer>
</Collection>
</XML>
<XML id=xmlProjs>
<Collection>
  <ProjReport>
    <ProjName></ProjName>
    <ProjID></ProjID>
    <PerformanceDetails></PerformanceDetails>
  </ProjReport>
</Collection>
</XML>
```

在设计时，表单元素是与适当的数据岛绑定的。以下是程序员表单的名字字段对应的

INPUT元素：

```
<INPUT id=firstNameText name=firstNameText dataFld=FirstName dataSrc=#xmlStaff>
```

随后你将看到我们必须手工操作数据岛的解析树，并且动态建立数据绑定。记住了这一点，让我们继续讨论程序功能的实现。

16.6 输入和编辑程序员信息

我们首先分析一下利用服务向数据库添加新程序员的过程。管理员在空白表单中添加信息，然后点击New/Update按钮。此时，客户端的按钮点击处理过程 OnInsertClick()要做出反应。客户根据用户提供的数据构建查询文档。

16.6.1 客户端

无论我们输入新的程序员记录还是修改旧的记录，客户端都将执行相同的操作——建立查询文档。我们要生成包含 ITStaffer元素的Collection文档，ITStaffer元素中包含表单字段的值。隐含的INPUT元素staffIDText用于区分插入和修改操作。如果我们输入新的程序员记录，该字段为空。如果服务器接收的 StaffID元素中不包含文本值，它认为该文档代表新的程序员，并给程序员分配ID。然而，在客户是没有差别的。OnInsertClick()首先获取表单元素的值，并将它们插入XML文档的适当位置：

程序清单 16-7

```
function OnInsertClick()
{
    var sReq = "";

    sReq = "<Collection><ITStaffer><Staffer><StaffID>"
        + staffIDText.value + "</StaffID>";
    sReq += "<FirstName>" + firstNameText.value +
        "</FirstName><MI>" + miText.value + "</MI>";
    sReq += "<LastName>" + lnameText.value +
        "</LastName><JobTitle>" + jobtitletext.value + "</JobTitle>";
    sReq += "</Staffer><Tier>" + tiertext.value +
        "</Tier></ITStaffer></Collection>";
}
```

然后，我们要创建MSXML IXMLHTTP接口的实例，并使用它将文档 POST到支持ITStaffer词汇表的服务上。然而，这里还存在着一个小问题。为了简单起见，我们为 HTML表单使用MIME编码。这意味着在传输之前，我们刚才构建的文档中的空格将转化为“ %20”。除字母和数字之外的其他字符也要受到影响。我们利用JavaScript的String对象的replace方法执行字符转换。方法中作为参数的查找字符和替换字符应该用正则表达式表示。我们将创建以下常量，并调用replace方法：

```
var regExp = / /g;
sReq = sReq.replace(regExp, "%20");
```

为了简化程序，我忽略了其他非典型字符的转换。表单编码能够简化服务器端程序，因

为ASP Request对象只要指定表单元素的名称就能够获取XML文档，在我们的例子中，表单元素的名称是XMLRequest，它的值就是程序员文档。在某些实际的问题中会频繁出现非字母和数字的字符。在这种情况下，很容易使用其他MIME编码。

到目前为止，我们的客户还不知道向何处发送信息。在前面，已经使用 GetASP()获得了URL。而且，曾经建立过两个全局范围的XMLHTTP组件实例。我们将使用其中的xmlStaffd处理程序员信息的上传。在GetASP()方法返回的URL之前增加协议标识符（http://），形成完整的URL，然后根据XMLHTTP传输的要求通过xmlStaffd对象POST程序员信息。为此，首先要指定POST操作和目的URL，然后设置适当的请求头Content-Type，最后发送请求：

```
var sServer = "http://" + GetASP("Staff");
xmlStaffd.Open("POST", sServer, false);
xmlStaffd.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
xmlStaffd.Send("XMLRequest=" + sReq);
```

经过服务器端的处理后，xmlStaffd将包含解析的XML文档。如果程序员信息被正常插入，该文档应该为<ACK/>（表示“确认”）。如果收到其他指示，我们将通知用户出现错误：

```
if (xmlStaffd.responseXML.documentElement.nodeName != "ACK")
    alert("Staff update was not accomplished due to a server problem.");
}
```

函数的突然结束隐藏了在我们上传信息之后服务器端执行的操作。

16.6.2 服务器端

与词汇表ITStaffer对应的服务是ASP页面staff.asp，它完全是由纯文本构成的。它首先获取XMLHTTP组件传递的文档的根节点。由于所有词汇表都是与数据库访问相关的，因此我们要创建ADO连接和记录集合对象：

程序清单 16-8

```
<%@ Language=JavaScript %>
<%

var dbConn, dbRecordSet;

/* Create a parser object and retrieve the XML search request
   from the server's Request object. */
var parser = Server.CreateObject("microsoft.xmlDOM");
parser.loadXML(Request.Form("XMLRequest").Item);

if (parser.readyState == 4 && parser.parseError == 0)
{
    // Establish a global connection
    dbConn = Server.CreateObject("ADODB.Connection");
    dbConn.Open("ProjTrack", "", "");
    dbRecordSet = Server.CreateObject("ADODB.RecordSet");
```

对于Windows 2000和最新版本的MSXML解析器，load()方法能够接受对任何支持IStream

接口的对象的引用。特别是MSXML中的XML文档对象。因此，以XMLRequest变量的值为参数loadXML()方法调用可以改写为以下更加有效的方式：

```
Parser.load(Request);
```

Windows和COM将负责执行适当的转换。最终，服务器端页面将接收 XML文档对象，并且能够在接收端加载它。

如果你从Wrox的Web站点下载代码，不要忘记创建名为 ProjTrack的系统DSN，它指向projects.mdb，没有用户名和口令。

1. 确定要做什么

服务staff.asp是准备响应StaffQuery和Collection词汇表文档的。对于Collection文档，我们需要进一步查看集合中是否包含ITStaffer对象。以下代码用于完成上述功能：

程序清单 16-9

```
switch (parser.documentElement.nodeName)
{
    case "StaffQuery":
        ProcessStaffQuery(parser.documentElement);
        break;
    case "Collection":
        // Receiving an update to SvcCustomer
        switch (parser.documentElement.childNodes(0).nodeName)
        {
            case "ITStaffer":
                ProcessStafferInsertion(parser.documentElement);
                break;
        }
        break;
    default:
        // Lack of ACK will signal db error,
        // empty collection doesn't spoil data binding
        Response.Write("<Collection/>");
}
```

2. 将数据放入数据库

无论添加新的程序员记录，还是更新现有的程序员信息，ASP都将接收与数据库的程序员表的各个字段对应的元素。一共有六个元素。首先，我们要获取这些元素，将它们存入字符串数组：

程序清单 16-10

```
function ProcessStafferInsertion(collectionNode)
{
    var rsFrag = new Array("", "", "", "", "", "");

    // Assemble the values to insert or update
    for (var ni = 0; ni < collectionNode.childNodes.length; ni++)
    {
        switch (collectionNode.childNodes(ni).nodeName)
        {
```

```

case "ITStaffer":
    // First child must be Staffer
    var stafferNode = collectionNode.childNodes(ni).childNodes(0);
    for (var nk = 0; nk < stafferNode.childNodes.length; nk++)
    {
        var sFragVal = stafferNode.childNodes(nk).text
        switch (stafferNode.childNodes(nk).nodeName)
        {
            case "StaffID":
                rsFrag[0] = sFragVal;
                break;
            case "FirstName":
                rsFrag[1] = sFragVal;
                break;
            case "MI":
                rsFrag[2] = sFragVal;
                break;
            case "LastName":
                rsFrag[3] = sFragVal;
                break;
            case "JobTitle":
                rsFrag[4] = sFragVal;
                break;
        }
    } // end of Staffer loop
    // second child will be the tier assignment
    var tierNode = collectionNode.childNodes(ni).childNodes(1);
    sFragVal = tierNode.childNodes(0).text;
    rsFrag[5] = sFragVal;
    break; // ITStaffer case

} // end of master switch statement
} // end of master for - next loop

```

在我们这个简单的例子中，程序员表的所有字段的值都是字符串。它能够保持程序的简单性，在获取数据库字段值之后，只需通过简单的字符串连接就能够生成 SQL 查询语句。如果你的数据包含各种数据类型，则要使用包含参数占位符的 SQL 查询语句，然后使用 ADO Parameter 对象根据数据类型做进一步调整。《ADO 2.0 Programmer's Reference》(Wrox Press, ISBN 1-861001-83-5) 介绍了有关内容。在本章的实例研究中，我们主要从 XML 角度分析程序。

程序员 ID 要遵循一定的商业规则。我们将为新的程序员设置唯一的编码。在 Microsoft Access 中，我们可以使用 Autonumber 数据类型。一旦设置了 ID，就不能再修改。因此，非空的 StaffID 元素说明要更新现有的程序员信息；空的 StaffID 元素说明这是新的程序员。对于以上两种情况，需要不同的 SQL 查询语句。INSERT 语句用于插入新的程序员，例如：

```
INSERT INTO members (FName, MI, LName, JobTitle, Tier) VALUES ("John", "A", "Doe",
"Senior Programmer", "Client");
```

UPDATE 语句用于改变现有的程序员信息：

```
UPDATE members SET FName="Jerome", MI="B", ..., Tier="Data" WHERE ID = "157";
```

现在回到 ProcessStafferInsertion()，我们将检查是否存在 StaffID 值，并据此产生适当的 SQL 命令：

程序清单 16-11

```
if (rsFrag[0] == "")
{
    sCore = "INSERT INTO members (FName, MI, LName, JobTitle, Tier) VALUES (";
    sQuery = sCore + MakeStaffValues(rsFrag) + "));";
    DoQuery(sQuery);
}
else
{
    sCore = "UPDATE members SET ";
    var sConstraint = " WHERE ID = " + rsFrag[0];
    sQuery = sCore + MakeStaffSets(rsFrag) + sConstraint + "));";
    DoQuery(sQuery);
}
```

数组的第一项 rsFrag[0] 如果非空，则包含 StaffID 元素的值。函数 MakeStaffValues() 和 MakeStaffSets() 使用数组 rsFrag 中的值构造分别用于插入和更新的 SQL 命令。函数 DoQuery() 负责执行 SQL 命令：

程序清单 16-12

```
function DoQuery(sQuery)
{
    try
    {
        dbConn.Execute(sQuery);
        if (dbConn.Errors.Count > 0)
            Response.Write("<NACK/>");
        else
            Response.Write("<ACK/>");
    }
    catch(e)
    {
        Response.Write("<NACK/>");
    }
}
```

在以上代码中，我们使用 try...catch 块避免由于 ADO 抛出的异常而导致的程序中断。连接对象的 Errors 集合说明 SQL 命令是否正确执行。如果一切正常，我们希望返回行集合，并且在传递给客户端的文档中写入 <ACK/>，说明操作成功。如果出现问题，我们将发送表示否定的 </NACK>。

到此为止，我们已经与客户端的代码衔接上。以下两节将显示客户端和服务端与添加程序员操作相关的完整代码。

16.6.3 用于插入程序员信息的完整客户端代码

用于插入程序员记录的代码能够充分代表应用程序的其他部分。因此，这些代码值得深入研究。为了便于分析，下面列出了执行该任务的完整的客户端源代码：

程序清单 16-13

```

function OnInsertClick()
{
    var sReq = "";

    sReq = "<Collection><ITStaffer><Staffer><StaffID>" +
        staffIDText.value + "</StaffID>";
    sReq += "<FirstName>" + firstNameText.value + "</FirstName><MI>" +
        miText.value + "</MI>";
    sReq += "<LastName>" + lnameText.value + "</LastName><JobTitle>" +
        jobtitleText.value + "</JobTitle>";
    sReq += "</Staffer><Tier>" + tiertext.value +
        "</Tier></ITStaffer></Collection>";

    var regExp = / /g;
    sReq = sReq.replace(regExp, "%20");
    var sServer = "http://" + GetASP("Staff");
    xmlStaffd.Open("POST", sServer, false);
    xmlStaffd.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");
    xmlStaffd.Send("XMLRequest=" + sReq);
    if (xmlStaffd.responseXML.documentElement.nodeName != "ACK")
        alert("Staff update was not accomplished due to a server problem.");
}

```

16.6.4 用于插入程序员信息的完整服务器端代码

staff.asp的主体是由使用该服务的所有任务共享的。用于实现程序员插入的代码能够很好地说明服务的其他部分。下面是用于插入程序员信息的完整的服务器端源代码：

程序清单 16-14

```

<%@ Language=JavaScript %>
<%

var dbConn, dbRecordSet;

/* Create a parser object and retrieve the XML search request
   from the server's Request object. */

var parser = Server.CreateObject("microsoft.xmlDOM");
parser.loadXML(Request.Form("XMLRequest").Item);

// Configure the mime type to reflect XML
Response.ContentType="text/xml";

if (parser.readyState == 4 && parser.parseError == 0)
{
    // Establish a global connection
    dbConn = Server.CreateObject("ADODB.Connection");
    dbConn.Open("ProjTrack", "", "");
    dbRecordSet = Server.CreateObject("ADODB.RecordSet");

    // Check for the primary and secondary supported vocabularies
    switch (parser.documentElement.nodeName)
    {
        case "StaffQuery":

```

```

        ProcessStaffQuery(parser.documentElement);
        break;
    case "Collection":
        // Receiving an update to SvcCustomer
        switch (parser.documentElement.childNodes(0).nodeName)
        {
            case "ITStaffer":
                ProcessStafferInsertion(parser.documentElement);
                break;
        }
        break;

    default:
        // Lack of ACK will signal db error,
        // empty collection doesn't spoil data binding
        Response.Write("<Collection></Collection>");
    }
    // Clean up resources
    try
    {
        dbRecordSet.Close();
        dbConn.Close();
        dbConn = null;
        dbRecordSet = null;
    }
    catch (e)
    {
    }
}
else
    Response.Write("<Collection></Collection>");

parser = null;
function ProcessStafferInsertion(collectionNode)
{
    var rsFrag = new Array("", "", "", "", "", "");

    for (var ni = 0; ni < collectionNode.childNodes.length; ni++)
    {
        switch (collectionNode.childNodes(ni).nodeName)
        {
            case "ITStaffer":
                var staffNode = collectionNode.childNodes(ni).childNodes(0);
                for (var nk = 0; nk < staffNode.childNodes.length; nk++)
                {
                    var sFragVal = staffNode.childNodes(nk).text;
                    switch (staffNode.childNodes(nk).nodeName)
                    {
                        case "StaffID":
                            rsFrag[0] = sFragVal;
                            break;
                        case "FirstName":
                            rsFrag[1] = sFragVal;
                            break;
                        case "MI":
                            rsFrag[2] = sFragVal;
                            break;
                        case "LastName":

```

```

        rsFrag[3] = sFragVal;
        break;
    case "JobTitle":
        rsFrag[4] = sFragVal;
        break;
    }
} // end of Staffer loop
// second child will be the tier assignment
var tierNode = collectionNode.childNodes(ni).childNodes(1);
sFragVal = tierNode.childNodes(0).text;
rsFrag[5] = sFragVal;
break; // ITStaffer case

} // end of master switch statement
} // end of master for - next loop

var sCore, sQuery;
if (rsFrag[0] == "")
{
    sCore = "INSERT INTO members (FName, MI, LName, JobTitle, Tier) VALUES (";
    sQuery = sCore + MakeStaffValues(rsFrag) + "));";
    DoQuery(sQuery);
}
else
{
    sCore = "UPDATE members SET ";
    var sConstraint = " WHERE ID = " + rsFrag[0];
    sQuery = sCore + MakeStaffSets(rsFrag) + sConstraint + "));";
    DoQuery(sQuery);
}
}

```

16.7 搜索程序员信息

当然，我们的程序设计管理员并不总是修改程序员记录。更常规的操作是查看现有程序员的成果报告。这意味着他们必须从 ITStaffer 服务获取记录。以上操作是由 ProjTrack.html 页面的按钮处理函数 OnFindClick() 开始的。

16.7.1 客户端

我们允许使用 StaffQuery 词汇表根据员工 ID 和姓氏的组合进行搜索。同样，我们要使用全局 XMLHTTP 组件 xmlStaffd 将查询文档 POST 到服务上：

程序清单 16-15

```

function OnFindClick()
{
    var sReq = "";
    var sServer = "http://" + GetASP("StaffQuery");
    xmlStaffd.Open("POST", sServer, false);

    sReq = "XMLRequest=<StaffQuery><StaffID></StaffID><LastName>";
    sReq += lnameText.value + "</LastName>";
    sReq += "</StaffQuery>";
}

```



```

var regExp = / /g;
sReq = sReq.replace(regExp, "%20");
xmlStaffd.setRequestHeader("Content-Type",
                             "application/x-www-form-urlencoded");
xmlStaffd.Send(sReq);
FixStaffBinding();
}

```

以上代码类似于向服务器发送程序员记录时所用的代码。我们生成 XML 文档，进行 MIME 编码，使用组件执行 POST 方法。然而，在搜索操作中，我们希望获得包含零个或多个 ITStaffer 对象的 Collection 文档。我们需要管理数据绑定，这是在 FixStaffBinding() 中实现的：

程序清单 16-16

```

function FixStaffBinding()
{
    while (xmlStaff.documentElement.childNodes.length > 1)
    {
        xmlStaff.documentElement.removeChild(xmlStaff.documentElement.childNodes(1));
    }
    for (var ni = 0;
         ni < xmlStaffd.responseXML.documentElement.childNodes.length;
         ni++)
        xmlStaff.documentElement.appendChild(
            xmlStaffd.responseXML.documentElement.childNodes(ni).cloneNode(true));

    if (xmlStaff.documentElement.childNodes.length > 1)
    {
        xmlStaff.documentElement.removeChild(
            xmlStaff.documentElement.childNodes(0));
        xmlStaff.recordset.moveFirst();

        stafftable.dataFld="Staffer";
        stafftable.dataSrc="#xmlStaff";
        firstNameText.dataFld="FirstName";
        firstNameText.dataSrc="#xmlStaff";
        miText.dataFld = "MI";
        miText.dataSrc = "#xmlStaff";
        lnameText.dataFld = "LastName";
        lnameText.dataSrc = "#xmlStaff";
        jobtitleText.dataFld = "JobTitle";
        jobtitleText.dataSrc = "#xmlStaff";
        staffIDText.dataFld = "StaffID";
        staffIDText.dataSrc = "#xmlStaff";
        tierText.dataFld = "Tier";
        tierText.dataSrc = "#xmlStaff";
    }
}

```

以上代码执行下述操作：删除除了数据岛根节点的第一个子节点之外的所有节点，添加应答返回的 ITStaffer 元素拷贝，然后删除数据岛中原来的树的其余节点（即：根节点的第一个子节点）。一旦在客户端生成了正确的文档，就可以开始建立数据绑定，为每个表单元素设置 dataFld 和 dataSrc 属性。此时，程序员表单将显示从服务接收到的新值。下面让我们看看 staff.asp 如何获得这些值。

16.7.2 服务器端

首先, 让我们回到 staff.asp 文件的主体, 看看如何响应接收到的文档:

程序清单 16-17

```
if (parser.readyState == 4 && parser.parseError == 0)
{
    // Establish a global connection
    dbConn = Server.CreateObject("ADODB.Connection");
    dbConn.Open("ProjTrack", "", "");
    dbRecordSet = Server.CreateObject("ADODB.RecordSet");

    switch (parser.documentElement.nodeName)
    {
        case "StaffQuery":
            ProcessStaffQuery(parser.documentElement);
            break;
        case "Collection":
            switch (parser.documentElement.childNodes(0).nodeName)
            {
                case "ITStaffer":
                    ProcessStafferInsertion(parser.documentElement);
                    break;
            }
            break;
        default:
            Response.Write("<Collection></Collection>");
    }
}
```

1. 获取程序员信息

ASP文件 staff.asp 接收到的文档是根据 StaffQuery 词汇表产生的。以下代码用于将 XML 查询文档转化为 SQL 查询:

程序清单 16-18

```
function ProcessStaffQuery(staffNode)
{
    var sSelectCore = "SELECT FName, MI, LName, ID, JobTitle, Tier FROM Members";

    var sConstraint = "";

    sConstraint = MakeStaffConstraint(staffNode);
    if (sConstraint != "")
        sConstraint = " WHERE " + sConstraint;

    var sQuery = sSelectCore + sConstraint + ';';

    dbRecordSet = dbConn.Execute(sQuery);

    if (!dbRecordSet.EOF)
        dbRecordSet.MoveFirst();

    if (dbRecordSet.EOF)
        Response.Write("<Collection></Collection>");

    Response.Write("<Collection>");
}
```

```
while (!dbRecordSet.EOF)
{
    WriteStafferBody(dbRecordSet);
    dbRecordSet.MoveNext();
}
Response.Write("</Collection>");
}
```

在以上代码中，我们生成查询语句，执行它，然后以 XML 的形式输出结果记录集合。与前面的例子一样，我们通过字符串连接生成查询，而没有使用 `Parameter` 对象。下面是根据 `StaffQuery` 文档产生的完整的 SQL 命令例子：

```
SELECT FName, MI, LNAME, ID, JobTitle, Tier FROM Members WHERE LName="Bee";
```

从通用性角度考虑，我们的查询词汇表允许包含程序员的 ID。在我们的应用程序中，始终没有显示或手工输入 ID，因此来自客户的查询文档也不包含 ID。然而，服务的建立是与特定的客户无关的，因此在构造 SQL 查询命令时，我们写入了用于处理该搜索参数的代码。

2. 返回给客户的 XML

既然我们已经获得了结果集合，就可以将这些值以 XML 的形式返回给客户，以便客户显示。这是由函数 `WriteStafferBody()` 完成的，它以我们刚才获得的结果集合为参数：

程序清单 16-19

```
function WriteStafferBody(rsStaffers)
{
    Response.Write("<ITStaffer><Staffer>");
    Response.Write("<StaffID>" + rsStaffers("ID") + "</StaffID>");
    Response.Write("<FirstName>" + rsStaffers("FName") + "</FirstName>");
    Response.Write("<MI>" + rsStaffers("MI") + "</MI>");
    Response.Write("<LastName>" + rsStaffers("LName") + "</LastName>");
    Response.Write("<JobTitle>" + rsStaffers("JobTitle") + "</JobTitle>");
    Response.Write("</Staffer><Tier>" + rsStaffers("Tier") +
"</Tier></ITStaffer>");
}
```

16.8 清除程序员表单

页面中有必要包含一个用于清除程序员表单所有项的按钮，特别是当使用应用程序的管理员需要执行一个完全不同的搜索或者输入新的程序员记录。从编程角度讲，要执行的任务是将数据绑定重新设置为 `ITStaffer` 文档。当用户点击 `Clear` 按钮时，我们要求 `MSXML` 加载完全由不含文本内容的元素构成的文档，然后执行常规的树操作——删除除第一个子节点之外的所有子节点，添加新的元素，然后从解析树中删除原来保留的第一个子节点。这将使得表单元素重新与文本元素为空的 XML 文档绑定。因此，表单中不显示任何内容：

程序清单 16-20

```
function OnClearClick()
{
    var parser = new ActiveXObject("microsoft.xmlDOM");
```

```

parser.loadXML("<Collection><ITStaffer><Staffer><StaffID/><FirstName/>
<MI/><LastName/><JobTitle/></Staffer><Tier/>
</ITStaffer></Collection>");
if (parser.readyState == 4 && parser.parseError == 0)
{
    while (xmlStaff.documentElement.childNodes.length > 1)
    {
        xmlStaff.documentElement.removeChild(
            xmlStaff.documentElement.childNodes(1));
    }
    for (var ni = 0; ni < parser.documentElement.childNodes.length; ni++)
        xmlStaff.documentElement.appendChild(
            parser.documentElement.childNodes(ni).cloneNode(true));
    if (xmlStaff.documentElement.childNodes.length > 1)
    {
        xmlStaff.documentElement.removeChild(
            xmlStaff.documentElement.childNodes(0));
        xmlStaff.recordset.moveFirst();
    }
}
parser = null;
}

```

16.9 输入程序员成果报告

我们需要输入有关程序员在项目中的表现的简短报告。该任务是从页面 ProjTrack.html的按钮点击处理函数 OnInsertProj()开始执行的。

16.9.1 客户端

我们首先强调一条重要的业务规则：成果报告不能独立于程序员而存在。在程序设计中，这条规则体现在我们要在名为 StaffIDText的隐含INPUT元素中寻找程序员 ID。如果找不到程序员 ID，说明表单为空，或者是尚未提交给服务的新程序员。无论属于哪种情况，我们都不能提交成果报告。如果找到程序员 ID，就可以根据 PerformanceReport 词汇表构造文档，并且采用与提交 ITStaffer 文档相同的方式提交该文档。该词汇表是由包含一个或多个 PerformanceReport 元素的 Collection 构成的。我们的客户端每次仅提交一个报告，但是有些客户端要进行批传输，因此词汇表要能够处理这种情况。下面列出了函数 OnInsertProj() 的代码：

程序清单 16-21

```

function OnInsertProj()
{
    var sRequest;

    if (staffIDText.value != "")
    {
        sRequest = "<Collection><PerformanceReport><StaffID>" +
            staffIDText.value + "</StaffID>";
        sRequest += "<ProjID>" + projIDText.value +
            "</ProjID><Comments>" + projDetail.value;
        sRequest += "</Comments></PerformanceReport></Collection>";
    }
}

```

如果存在程序员 ID，我们将从表单中搜集数据值，并构造 XML 文档。让我们继续来看 OnInsertProj() 的其余代码，对文档进行 MIME 编码，并执行 HTTP 上传操作：

程序清单 16-22

```
var regExp = / /g;
sRequest = sRequest.replace(regExp, "%20");
var sServer = "http://" + GetASP("PerformanceReport");
xmlProjd.Open("POST", sServer, false);
xmlProjd.setRequestHeader("Content-Type",
    "application/x-www-form-urlencoded");
xmlProjd.Send("XMLRequest=" + sRequest);
OnStaffHistory();
}
else
    alert("We cannot add a service incident report without a staff member
        search.");
}
```

我们在前面已经看到过类似的代码。它与我们在提交 ITStaffer 和 StaffQuery 文档时所用的代码大同小异。然而，为什么要调用 OnStaffHistory() 函数呢？提交成果报告后，刷新客户端最简单的方法是获取程序员的成果报告历史。获取成果报告是我们的下一项任务。然而，在此之前，我们暂且封装该任务，看看服务器端的操作。

16.9.2 服务器端

迅速查看一下 GetASP() 能够知道词汇表 PerformanceReport 是与名为 History.asp 的 ASP 相关的。该脚本的主体与 staff.asp 大同小异。程序首先获取传输的文档，建立某些数据库资源，然后根据文档的词汇表执行适当的操作。如前所述，PerformanceReport 文档是 Collection 的一部分。下面显示了 History.asp 主体部分与该文档相关的分支语句：

程序清单 16-23

```
...
case "Collection":
    switch (parser.documentElement.childNodes(0).nodeName)
    {
        case "PerformanceReport":
            ProcessPerformanceReport(parser.documentElement);
            break;
        default:
            Response.Write("<Collection></Collection>");
            break;
    }
    break;
...
```

1. 处理成果报告

ProcessPerformanceReport 函数构建 SQL 语句，并将它提交给数据库。同样，我们在程序设计上保留了一定的灵活性，它能够处理一个 Collection 文档中包含多个报告的情况：

程序清单 16-24

```
function ProcessPerformanceReport(collectionNode)
{
    var sDetails, sID, sProjID;

    for (var ni = 0; ni < collectionNode.childNodes.length; ni++)
    {
        switch (collectionNode.childNodes(ni).nodeName)
        {
            case "PerformanceReport":
                sDetails = collectionNode.childNodes(ni).childNodes(2).text;
                sProjID = collectionNode.childNodes(ni).childNodes(1).text;
                sID = collectionNode.childNodes(ni).childNodes(0).text;
                InsertPerfReport(sDetails, sID, sProjID);
                break;
        }
    }
}
```

我们处理根节点 Collection 的每个子元素。如果它是 PerformanceReport——谁知道任意一个客户会发送哪些信息？——从中提取详细的文本描述以及程序员 ID 和项目 ID，并将它们作为参数传递给 InsertPerfReport()。该函数构造用于插入信息的 SQL 语句，并执行它。

2. 插入成果报告的数据库操作

我们不允许管理员编辑现有的成果报告——任何人都不能掩盖有利或不利的项 目评价！同时它能够简化程序设计。我们不必担心是使用 INSERT 还是 UPDATE。所有与提交成果报告任务相关的命令都是 INSERT。例如，我们可能生成以下 SQL 语句：

```
INSERT INTO Details(ProjID, MemberID, Comments) VALUES ("6", "157", "Worked hard,
had no clue.");
```

因此，用于插入新的成果报告的代码如下：

程序清单 16-25

```
function InsertPerfReport(sComment, sStaffID, sProjID)
{
    var sCore = "INSERT INTO Details (ProjID, MemberID, Comments) VALUES ('";
    var sQuery = sCore + sProjID + "', '" + sStaffID + "', '" + sComment + "');";
    dbConn.Execute(sQuery);
    if (dbConn.Errors.Count > 0)
        Response.Write("<NACK/>");
    else
        Response.Write("<ACK/>");
}
```

生成 SQL 语句之后，我们要执行该命令并判断是否出现错误。如果一切正常，将返回简短的 XML 文档 <ACK>。

细心的读者会发现上面的 SQL INSERT 语句在 Details 表中插入了对一个项目的引用，却没有验证 Projects 表中是否存在相应的行。由于本例侧重于说明 XML，而不是数据库问题，因此我采取了简略的形式，忽略了维护项目信息的问题。如果你打算将该原型扩展为真

正的应用程序，应该执行一致性检查。我可能还会利用触发器和存储过程协助你维护数据库的完整性。

16.10 清除成果报告历史表单

当你在成果报告历史表单中点击Clear按钮时，将执行这项任务，它与清除程序员表单的过程类似。我们首先将名为xmlProjs的数据岛的解析树重新设置为包含一个ProjReport元素的Collection文档。ProjReport元素及其子元素不包含文本值，只包含元素。下面是用于完成上述任务的源代码：

程序清单 16-26

```
function OnClearProj()
{
    var parser = new ActiveXObject("microsoft.xmlDOM");
    parser.loadXML("<Collection><ProjReport><ProjName/>
                  <ProjID/><PerformanceDetails/></ProjReport></Collection>");
    if (parser.readyState == 4 && parser.parseError == 0)
    {
        while (xmlProjs.documentElement.childNodes.length > 1)
        {
            xmlProjs.documentElement.removeChild(
                xmlProjs.documentElement.childNodes(1));
        }
        for (var ni = 0; ni < parser.documentElement.childNodes.length; ni++)
            xmlProjs.documentElement.appendChild(
                parser.documentElement.childNodes(ni).cloneNode(true));
        if (xmlProjs.documentElement.childNodes.length > 1)
            xmlProjs.documentElement.removeChild(
                xmlProjs.documentElement.childNodes(0));
        xmlProjs.recordset.moveFirst();
    }
}
```

16.11 获取程序员的成果历史

现在，我们只剩下一项功能了，这就是获取页面上显示的程序员的全部成果报告。该过程由ProjTrack.html页面的OnStaffHistory()处理器开始。在该函数中，我们将ProjPerformanceQuery文档上传到服务器，并接收包含零个或多个ProjReport元素的Collection文档。接收到来自服务的文档后，我们将更新数据绑定。

16.11.1 客户端

构造查询文档的工作非常简单，因为我们只需要发送 StaffID字段的值：

程序清单 16-27

```
function OnStaffHistory()
{
    var sReq = "";
```

```

if (staffIDText.value != "")
{
    var sServer = "http://" + GetASP("ProjPerformanceQuery");
    xmlProjd.Open("POST", sServer, false);

    sReq = "XMLRequest=<ProjPerformanceQuery><StaffID>" +
        staffIDText.value + "</StaffID></ProjPerformanceQuery>";

    xmlProjd.setRequestHeader("Content-Type",
        "application/x-www-form-urlencoded");
    xmlProjd.Send(sReq);
    FixHistoryBinding();
}
}

```

FixHistoryBinding()函数执行动态更新操作，在本章的实例研究中，我们已经多次看到类似的操作。唯一的特殊之处在于我们绑定的是重复的 HTML 元素。根据 XML DSO，表格的每一行与一个 ProjReport 元素匹配。ProjTrack.html 只需要提供包含一行的模板。下面是程序设计阶段存在的表格：

程序清单 16-28

```

<TABLE border=1 cellPadding=1 cellSpacing=1 width="75%"
        id=projTable name = "projTable">
    <TR>
        <TD>
            <P>
                <INPUT id=projName name=projName style="HEIGHT: 22px; WIDTH: 422px">
                &nbsp;
                <FONT face=Verdana size=2>ID
                <INPUT id=projIDText name=projIDText>
                </FONT>
            </P>
            <P><FONT face=Verdana size=2 >Performance Details</FONT></P>
            <P>
                <TEXTAREA id=projDetail name=projDetail
                    style="HEIGHT: 38px; WIDTH: 422px">
                </TEXTAREA>
            </P>
        </TD>
    </TR>
</TABLE>

```

当我们获取数据时，INPUT 和 TEXTAREA 元素与 projName、projIDText 和 projDetail 动态绑定。我们首先删除当前文档中除第一个节点之外的所有节点，之所以保留第一个节点是为了防止绑定失败。然后，添加所有新节点，最后删除旧文档的其余节点：

程序清单 16-29

```

function FixHistoryBinding()
{
    while (xmlProjs.documentElement.childNodes.length > 1)
    {

```

```

xmlProjs.documentElement.removeChild(xmlProjs.documentElement.childNodes(1));
}
for (var ni = 0;
    ni < xmlProjd.responseXML.documentElement.childNodes.length;
    ni++)
    xmlProjs.documentElement.appendChild(
        xmlProjd.responseXML.documentElement.childNodes(ni).cloneNode(true));

if (xmlProjs.documentElement.childNodes.length > 1)
    xmlProjs.documentElement.removeChild(
        xmlProjs.documentElement.childNodes(0));
xmlProjs.recordset.moveFirst();

projTable.dataSrc="#xmlProjs";
projName.dataFld="ProjName";
projIDText.dataFld = "ProjID";
projDetail.dataFld="PerformanceDetails";
}

```

16.11.2 服务器端

一旦History.asp接收到我们的查询文档，它就从根节点开始处理：

```

case "ProjPerformanceQuery":
    ProcessHistoryQuery(parser.documentElement);
    break;

```

ProcessHistoryQuery()负责将XML转化为SQL SELECT语句。这类SQL语句类似于以下形式：

```

SELECT Projects.ProjectName, Details.ProjID, Details.Comments FROM Details INNER
JOIN Projects ON Details.ProjID = Projects.ProjID WHERE Details.MemberID="6";

```

在执行查询之前，ProcessHistoryQuery()利用函数MakeHistConstraint()构造SQL语句的查询条件：

程序清单 16-30

```

function ProcessHistoryQuery(histQNode)
{
    var sSelectCore = "SELECT Projects.ProjectName, Details.ProjID,
        Details.Comments FROM Details INNER JOIN Projects ON
        Details.ProjID = Projects.ProjID";

    var sConstraint = "";

    sConstraint = MakeHistConstraint(histQNode);
    if (sConstraint != "")
        sConstraint = " WHERE " + sConstraint;

    var sQuery = sSelectCore + sConstraint + ' ';
    dbRecordSet = dbConn.Execute(sQuery);
}

```

发出查询命令之后，程序必须依次处理结果记录集中的每条记录，并将每一行作为一个成员写入Collection文档：

程序清单 16-31

```

if (!dbRecordSet.EOF)
    dbRecordSet.MoveFirst();

if (dbRecordSet.EOF)
    Response.Write("<Collection></Collection>");

// Write the collection root node and then the rest of the contents
Response.Write("<Collection>");
while (!dbRecordSet.EOF)
{
    WriteHistBody(dbRecordSet);
    dbRecordSet.MoveNext();
}
Response.Write("</Collection>");
}

```

以下是两个辅助性函数：MakeHistConstraint()和WriteHistBody()。MakeHistConstraint()从查询文档中提取程序员ID，并构造SQL WHERE从句的核心部分。WriteHistBody()是在函数ProcessHistoryQuery()中调用的，它将数据库查询的结果转化为要发送给客户端的XML文档：

程序清单 16-32

```

function MakeHistConstraint(node)
{
    var sClause = "";
    var sID = "";
    for (var ni = 0; ni < node.childNodes.length; ni++)
    {
        switch (node.childNodes(ni).nodeName)
        {
            case "StaffID":
                if (node.childNodes(ni).text != "")
                    sID = "Details.MemberID = " + node.childNodes(ni).text;
                break;
        }
    }
    sClause = sID;
    return sClause;
}

function WriteHistBody(rsCalls)
{
    Response.Write("<ProjReport><ProjName>");
    Response.Write(rsCalls("ProjectName"));
    Response.Write("</ProjName><ProjID>");
    Response.Write(rsCalls("ProjID"));
    Response.Write("</ProjID><PerformanceDetails>");
    Response.Write(rsCalls("Comments"));
    Response.Write("</PerformanceDetails></ProjReport>");
}

```

16.12 经验教训

既然我们已经完成了本章开始提出的用于管理程序的项目，现在应该根据提出的开发原则对我们的工作进行评价。应用程序能够满足项目的功能需求，只需要极少的代码，就能够保证平台独立性。在本例中，采用了特定平台的技术——用于可视化表示的数据绑定——但是我们将它与服务器端的代码实现相分离。类似地，服务器端使用了 ADO 和关系型数据库，但是它对客户端是屏蔽的。这种设计方式提高了代码重用的可能性。特别是服务器页面可以“原封不动地”与其他客户端应用程序配合使用。

然而，我们的工作并非尽善尽美。我们犯的错误将用以说明我们提出的开发原则的价值。

16.12.1 违反的开发原则

数据绑定迫使我们的程序员将 ITStaffer 词汇表与程序员表单以及 ProjReport 词汇表与成果报告历史表单紧密结合。实际上，我们并不能将之称为违背开发原则，至少在缺少公认的元数据标准的情况下。使用数据绑定极大地简化了编程任务，而且使客户端能够具有简洁的用户界面。然而，词汇表的改变会严重影响到客户端应用程序。

由于我们在代码中没有区分 ITStaffer 和 Staffer 元素，因此丧失了绝好的代码重用机会。在本例中，这两个元素的差别极小，因此没有理由使用附加的代码。然而，在现实世界中，我们要建立用于处理普通 Staffer 词汇表的函数，然后用其他函数处理特殊的 ITStaffer 词汇表。专有化的函数将调用通用函数中已经实现的功能，以处理 ITStaffer 文档的通用部分（Staffer 元素）。

16.12.2 组件

程序因使用了 MSXML 及其相关接口、XML DSO 和 IXMLHTTP 而得到简化。虽然这使得我们的客户浏览器局限于 Microsoft Internet Explorer 5.0，但是要想通过自己的编程代替这些组件提供的主要功能非常困难。我们特定的应用程序可以完全在服务器上执行，它用 ASP 编写，返回 HTML。然而，这种方法使得其他应用程序无法重用我们实现的服务，特别是自动化的客户。我们鼓励的代码重用是以客户编程为代价的。然而，我们确实有所损失。我们没有创建任何自己的组件。特别是，我们的开发原则提倡在每一层都使用组件，并将组件数据转化为 XML 在平台之间传递。哪里有与 ITStaffer 和 ProjReport 相关的组件？

我们本应该使用 JavaScript 将 ITStaffer 类创建为 COM 组件。应该利用它检查是否违反了字段限制或业务规则。例如，可以根据企业职称分类数据库检查某个职称是否合法。在应用程序的服务器端，这类组件非常有价值。在我们这个仅仅用于说明 XML 应用的例子中，不可能也没有必要采取这种方式，然而在现实世界中，这是至关重要的。事实上，其他应用程序只能重用本例中的脚本，而无法重用完整的代码。如果不编辑源代码，本例没有任何可重用的组件。

16.12.3 重用的可能性

然而，本章介绍的应用程序中有两个资源是将来可重用的：分别由 History.asp 和 Staff.asp 表示的服务。这些服务是通用的，而且符合我们的开发原则，因此任何需要这些词汇表的应用程

序可以调用它们。这就是协作网络应用程序的目的。你开发服务，通过目录列表发布它。由于这些服务有已知的接口，因此客户应用程序可以使用它们。在本例中，接口包含 XML词汇表。如果这些词汇表能够忠实地反应商业运作中的某个有用部分，它们本身就是可重用的资源。

16.13 小结

在本章中，我们介绍了将 XML与ASP相结合的基本方法。看到 XML非常适于ASP和动态文档生成。使用了一些特定平台的工具，主要是 MSXML的IXMLHTTP接口和数据绑定。如果要保证应用程序的平台独立性，需要将所有处理移至服务器端，使用与平台无关的组件，或者针对不同的浏览器产生不同的客户页面版本。

我们看到了普通的编程方法在 Intranet和Internet应用程序设计中暴露出的问题。例如：

- 处理日益增长的复杂性。
- 不灵活的应用程序。
- 代码重复。
- 转向自动化 Web任务。
- 分布式开发和实施小组。

在Web开发中，为了克服以上弱点，我提出了五条原则：

- 从粗粒度服务构建应用程序。
- 通过查询目录发现服务。
- 将服务提供为自描述数据。
- 服务应该是短暂的。
- 服务必须可扩展，且能够降低对外部的要求。

最后，我将理论与实践相结合。事实证明，XML是一种能够实现这五条原则的出色技术。ASP也非常适于构建基于 Web的服务。总而言之，如果要建立功能强大的 Web应用程序，而且要满足松散连接的网络和应用程序的特殊需求，ASP和XML是最合适的选择。