

第13章 组件和网络应用程序结构

前节介绍了如何使用 ASP 动态地创建网页。ASP 脚本能够查看由浏览器传送过来的信息，动态地创建一个 HTML(或 DHTML) 页面送回浏览器。创建这个页面时，我们能获得从数据库(如 SQL Server 或 MSDE) 中检索的信息，并且把这些信息加到 HTML 页面中送回浏览器。

但是，ASP 不仅能够创建动态网页，它还具有将页面组合进网站的能力，该网站具有应用程序的功能。Application 和 Session 对象使应用程序具有状态，允许在对网站的请求之间，甚至在网站的所有用户之间共享消息。这些应用软件包括传统应用程序的所有功能。

建立更大、更可扩展的应用程序时，ASP 技术能使建立过程更为简单。如果一个应用程序真正可扩展，那么就可以用同样的程序处理大量的用户，而且与处理少量用户时的性能相近。由于网站的建立能够给我们带来收益，所以能支持更多用户意味着有更多的收益。建立更可扩展的应用程序的关键是用组件构造它们，本章和下面几章介绍如何用组件设计应用程序。

本章主要内容有：

- 分布式应用程序的结构，包括 Windows DNA。
- 网络应用程序的构成。
- 组件的定义。
- ActiveX 和 COM 简介。
- 组件的三种类型和使用原因。
- 使用组件建立网络应用程序。
- 设计一个基于组件的网络应用程序。

下面首先介绍应用程序结构。

13.1 分布式应用程序的结构

应用程序设计的一个关键要素是系统结构，系统结构决定了应用程序的各个部分如何进行交互，同时也决定了每个部分实现的功能。在这个基于网络的世界中，通常在分布式环境中创建应用程序。实际上，本书中的 ASP 应用程序就是分布式应用程序。

分布式应用程序利用多台机器的资源及许多进程的空间，把应用程序分成更易管理的任务组，这些任务组能在各种不同的结构下进行部署。把应用程序分成任务组有许多好处，至少可以便于重复使用、扩充和管理。

最后，以这种方式下分解应用程序将导致一系列的应用程序的层或级出现，每层或级都各司其责。

13.1.1 分层应用程序

分层应用程序可以按层数进行划分，信息可以从数据层(通常存储在数据库中)传送到表现层(显示在客户端上)。通常每层相对于其他层来说都运行在不同的系统中，或者在同一系统中

的不同的进程空间里。

1. 两层应用程序(客户端 / 服务器)

先简单描述一下两层的客户端 / 服务器结构。典型的结构是一个客户端的用户 PC机(前端)和一个包含数据库的网络服务器(后端), 逻辑上根据两者的物理位置划分。通常, 客户端包含大部分业务逻辑, 随着存储过程的发展, SQL语言允许业务逻辑在数据库服务器中存储并执行。两层的客户端 / 服务器结构如图 13-1所示。

当一个小的业务仅仅使用或需要一个简单的数据源时, 这种两层方案会运行得很好。然而, 大多数业务目标在不断增长, 随着业务的不断增长, 数据库的性能必须提高。这种两层方案的性能不能成比例地提高。如果业务规则改变了, 那么应用程序就需要重建并重新配置。另外, 有些因素(例如并发的数据库连接的最大数量)将使这个结构在一个分布式的设置下不能发挥更大的效能。

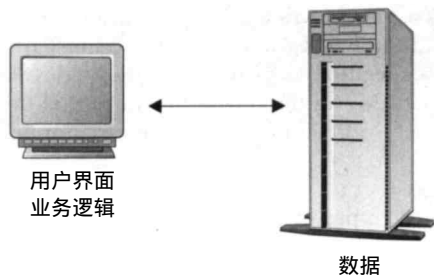


图13-1 两层的客户端 / 服务器结构

2. 三层和N层应用程序

由于二层客户端 / 服务器结构的限制, 分布式应用程序常常分为三层或者更多层。每层的组件都执行一个特定类型的处理, 在三层应用程序中包含一个用户服务(表现)层、一个业务服务层和一个数据服务层。

三层结构和传统的二层客户端 / 服务器结构的主要区别就是在三层体系内, 业务逻辑从用户界面和数据源中分离出来。

把应用程序划分成独立的层或部分能减小整个应用程序的复杂性, 并且使应用程序能够跟得上业务发展的需要。N层应用程序是三层应用程序的深化发展。在许多情况下, 一个二层以上的应用程序就可以看作是N层应用程序。三层应用程序的结构如图 13-2所示。

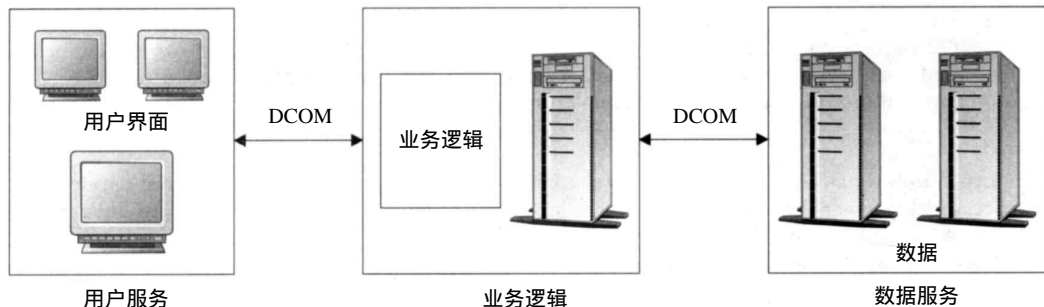


图13-2 三层应用程序的结构

这种应用程序的客户端不能直接访问数据存储系统, 如果这么做了, 那么将违反应用程序的业务规则, 并且不能确保客户端显示的数据是正确的。

应用程序的各个部分可分为N层, 这个应用程序的任何部分可以在不改变其他部分的情况下进行修改, 开发人员可以专门设计和开发一个特殊的层或许多层。同样, 开发人员还可采用专门用于层开发的开发工具, 比采用通用工具更为灵活方便, 虽然通用工具可用来建立应用程序, 但是功能却比专用开发工具逊色。

13.1.2 Windows DNA

Windows DNA是指微软的 Windows分布式Internet结构(Windows Distributed interNet Architecture),而不是脱氧核糖核酸,这是微软最新和最重要的 Windows平台的开发模型,它的目标是:

- 在Windows 服务器平台上,提供建立和传送稳固、可扩展的网络应用程序结构。
- 从传统的桌面应用程序到网络浏览器及网络应用程序,广泛的客户可获得新的和已有的应用程序和数据资源。

1. 具有基础结构的Windows DNA

Windows DNA使开发分布式应用容易得多,因为使用了已有的应用程序和操作系统服务,减少了创建分布式应用需要的基础结构代码的数量。

在应用程序中提供事务处理过程、信息排队、安全检查和自定义数据库访问时,必须编写几百行代码。如今已不需要所有这些分布式应用程序,微软重点考虑可能遇到的所有细节,提供了一系列有效和可编程的服务,这样只需将精力集中在应用程序的业务需要就行了。

用Windows DNA模型开发分布式应用程序,能够建立高质量、高可靠性、可扩展的分布式应用程序,传输更快,成本更低。

在Windows DNA出现之前,通常只有大规模企业采用分布式应用程序。但是,因为应用程序的基础结构,微软不断发展应用软件,使开发分布式应用程序更为容易。不久,许多更小的网络实用程序将可用 Windows DNA模型建立,从而取代传统的两层客户端/服务器模型。

2. 具有互操作能力的Windows DNA

Windows DNA能够与已有的传统系统相互操作和对话,因此可以使当前的技术投入物尽其用。

如今已不再需要为使所有使用 Oracle数据库系统的公司转为使用微软 SQL 7.0服务器数据库系统而耗费时间。微软已经意识到并不是所有人都愿意或能够将每一个传统的系统进行转换,以运行最新的微软数据平台。传统的系统仍将保留,但必须与新的应用相结合。所以,微软在Windows DNA中加入了互操作性和对工业标准的支持。例如,Windows DNA模式支持开放式Internet标准的HTML和XML。

系统之间的链接就是 DCOM,它允许应用程序在程序、各机器与提供服务的已编译组件之间交换信息,而且不用知道服务来自哪里或是如何实现的。DCOM可以让开发者创建以服务为基础的应用软件,而且使用组件式的应用程序开发方法。

DCOM主要是一个Windows操作平台服务,由Windows DNA模式支持的开放式Internet标准能够确保成功创建跨平台的分布式应用软件。

Windows DNA也使用通用数据访问(UDA)访问旧数据,例如允许使用相同的数据访问库,对SQL Server和旧的AS400数据库进行查询。通过多种技术能做到这点,这些技术都清楚如何与相同的库进行交互,并有能力从自己的系统访问关系数据和非关系数据。

3. Internet与Windows DNA

1997年9月,当微软引入Windows DNA结构时,为了创建新的Internet应用,把网络与现有的客户端/服务器和桌面系统相结合,引入了一个框架结构。Internet之所以成为主要的应

用程序平台是因为：

- Internet允许开发者创建独立于平台的应用程序，而不用另外学习一种新的语言。
- Internet为部署应用程序提供了一个灵活的结构，对许多组织来说，尤其是考虑对几个分散的位置部署一个大规模的企业应用程序，部署是一个主要问题。

13.1.3 Windows DNA服务

Windows DNA提供了一系列服务，这些服务提供了必要的通道和基本结构，利用这些服务就可以在相对生疏的情况下建立应用程序。

1. 简报服务

Windows DNA的简报服务功能处理应用程序如何同用户交互。用户可能被定位于同一台计算机、同一局域网的不同计算机或者与 Internet连接的另一计算机。Windows DNA的一个重要概念就是用户使用的客户端类型不影响对应用程序的访问能力。为了能获得这种灵活性，Windows DNA支持的客户端类型和工具非常广泛，利用它们就可以建立简报服务了。

Windows DNA应用程序能支持许多不同客户端类型，可分为四个类别，其中两个是传统的“胖客户端”，而另外两个则是“基于浏览器”的客户端。如果网络可用的话，传统的应用程序利用网络连接，当网络连接不存在时功能失效。基于浏览器的应用程序既可以是适用于一般浏览器，也可以是只适用于特殊浏览器。

(1) Internet增强型客户

增强型 Internet客户利用DNA的特点，能够使用 Internet连接，但不依赖 Internet连接，例如微软的 Office 2000和 Visual Studio。这些应用程序通过在它们中嵌入超级链接来支持统一浏览，拥有显示 DHTML文件的浏览器，并且能通过 Internet无缝地下载更新数据。

(2) Internet依赖型客户

Internet依赖型应用程序是作为传统的 Windows应用程序而编写的，它在运行时要求同服务器进行网络连接。当应用程序的用户界面超出了使用基于网络客户的有效开发范围时，通常使用这种程序。由现有的 Windows程序转换到 Windows DNA程序，与 Internet增强型应用程序不同，这种类型的程序的主要特点是运行时需要网络连接，Internet增强型应用程序能在独立的机器上和网络连接的机器上运行。

“胖客户端”应用程序可以使用任何 Windows开发工具建立，像 Visual Basic、Visual C++、Delphi或任何其他建立可执行应用程序的工具。网络连接的可靠性可以用一些不同的方法提高。例如，可将应用程序链接到 Windows套接字库，以便在本地网或 Internet上通过 TCP/IP与服务器连接。另外，它可以拥有一个嵌入式 IE版本，可对网上信息进行访问。在任何情况下，应用程序与网络上的服务器互相影响，并且在没有固定链路的情况下不能工作。

(3) 浏览器增强型客户

前面两种形式的应用程序是可执行程序的一部分，接下来的两种应用程序是基于浏览器的程序，因为它们使用 HTML作为描述工具。此 HTML采用 HTTP作为数据转换协议从网络服务器中实时检索得到。HTML提供给客户一个用户界面，允许用户使用网络浏览器同应用程序进行交互。

尽管 HTML是一种规范，但不同浏览器常常支持额外功能，这些功能都是非标准的。当用户界面开发需要对 HTML规范进行扩展时，它就被认为是增强的浏览器应用程序。对

Windows DNA应用程序来说,这通常意味着用户界面成为IE中的一个传递目标。用特定的浏览器作为传递目标,开发人员可利用ActiveX控件、动态HTML或者IE等技术来让用户界面更像传统Windows应用程序。甚至在这种情况下,使用HTTP协议将用户界面从网络服务器上上传递到网络浏览器上。这样的应用程序通常用于局域网,这样信息部门能控制目标浏览器平台。如果开发人员可以获得浏览器的特定版本,那么就能开发出浏览器增强型应用程序,并且这个应用程序能够成功地运行。

(4) 浏览器依赖型客户

浏览器增强型用户界面的一个缺点就是当用户使用应用程序时,通常需要特定的浏览器。不论什么时候,一种新技术可用时,如新的浏览器版本,都有一定的时间延迟,新版本浏览器的广泛使用需要长达数年的时间,在这段时间里,如果开发人员以这些新特点为目标,那么他们就要冒应用程序不能被广泛采用的风险。

为了避免这个问题,开发人员可以选择一个较早的浏览器版本作为目标,这样开发出来的程序就能被大多数人所采用。这种类型的客户端称为浏览器依赖型。这听起来有点不太真实,但浏览器依赖型就是指用户界面在浏览器上是可靠的,而不是对浏览器的特定版本可靠。对开发人员来说,开发一个面向所有浏览器的程序效率可能很低。通常广泛使用的程序的当前标准目标是网景和微软的3.x版本浏览器。

以这个浏览器版本作为目标后,开发人员就可以用JavaScript和HTML 3.2规范访问客户端脚本。这可与所有后来的浏览器兼容,不论是微软或网景。随着IE 4.0以上版本的使用,这已成为许多开发人员的基本发展目标。

2. 应用程序服务

Windows DNA应用软件靠可用的应用程序服务来存储和执行应用程序逻辑的核心,这个业务逻辑层中保持专用应用程序进程和业务规则。开发这些应用程序组件的Windows DNA服务包括Web服务、消息服务和组件服务。

(1) 组件服务

最近十年里,组件式应用软件已经成为创建应用软件的标准方法,最早是DDE,然后是OLE、OLE2及今天的ActiveX,Windows平台的组件应用软件已广泛传播。使用组件进行开发如此容易并不十分令人惊奇,现在整个过程是一个新的模型,这个模型包括用预先创建的组件建立应用程序。因为每一组件都有一些公用函数,其他组件可调用这些函数。这样组件可以重复使用,并允许这些软件互相合作。

为了使基于组件的应用程序成为企业级的应用程序,微软添加了微软事务处理服务器(MTS),为基于组件的分布式程序开发、部署和管理提供服务。将这个服务器封装在一个系统级的服务中,开发人员就没有必要明确地在应用程序中添加这种支持。

随着Windows 2000的推出,人们已逐渐了解COM+。COM+使COM和MTS程序模型成为一体,并且不再需要分别处理这两种服务,从而使开发分布式应用程序变得更加容易,因为减少了使用这些组件服务所需的代码数量,使得建立应用程序更快、更容易、成本更低。

(2) 消息服务

并不是Windows DNA应用程序的所有部分都必须定位于同样的计算机或者在同样的物理位置,因此,开发人员能够与不同系统进行交流是非常关键的。在网络应用程序模型中,需要一种通过应用程序与断开的客户进行交流的方法,消息是一种用于这种交流的方式。微软

消息排队服务器 (MSMQ) 通过在应用程序之间建立信息传送机构, 使得集成应用程序变得更为容易。这允许开发人员创建可靠的应用程序, 为连接的和不连接的客户工作。 MSMQ 可与其他消息排队产品进行无缝互操作, 如 IBM 的 MQSeries。

(3) 网络服务

Windows DNA 应用程序的应用程序服务需要一种与客户表现层交互的方法。这种方法就是超文本传输协议, 微软的网络服务器 (也就是 IIS) 已经提供了使用这个协议的服务。事实上, IIS 不仅支持传输静态网页到客户, 也可用于与 ASP 的集成动态地创建客户显示。

3. 数据服务

通用数据访问是微软为在企业之间传递信息而设计的, 能对不同种信息源提供高质量的访问, 包括关系数据、非关系数据, 并提供一个使用简便的既可单独作为工具又可单独作为语言的编程界面。UDA 存储数据费用低、耗时少, 也不需要特制一种程序。 UDA 建立在广泛工业支持的开放式工业规范上, 可运行于所有已建立的数据库平台上。提供数据服务的两个基本组件是 ADO 和 OLE DB, 这两者都已经在本书中介绍过。

13.1.4 网络结构

网络结构是 HTML 格式化页面的组合架构, 分布在网络服务器和客户计算机上。所使用查看用的工具称为浏览器。从本质上看, 包含几千页的服务器与只有几页的服务器没有什么本质不同, 把几千页以某种形式链接在一起就形成网站, 再加上一些附加逻辑, 网站就变成了一个网络应用程序。

1. 网页

网络交互的基本单元就是网页本身, 网页是用 HTML 编写的文本文件。根据浏览器的请求, 网页从网络服务器发送到浏览器, 浏览器先从语法上分析 HTML 文件中的信息, 所得到的用户界面表现在浏览器上。网络服务器的任务是等待客户端的请求, 分析并决定客户端需要的网页, 从服务器的存储区域中检索这个文件, 然后将文件传输到客户端。除非将信息放入日志文件中, 否则服务器会忽略所有关于传输文件到客户端的事情。就是这种“无连接”性使网络服务器有了可扩充性。但是, 在没有额外支持的情况下, 创建一个有意义的应用程序变得很困难。

2. 网站

从某种意义上说, 一个网站由一系列相关联的网页组成。通常一个网站是指存在于一个服务器上或者一个服务器文件夹里的所有网页。例如, 在 www.wrox.com 服务器上, 所有的网页都可以认为是这个网站的一部分。网站上网页相互联系主要依靠这个网站内的每个网页的链接来实现的。以这种方式组成网站的网页保持网站的分层结构。这些网页仍受到网络服务器结构的约束, 因为服务器不保留来自特定客户的一系列请求的信息, 所以这样构成网站的一系列相关网页看起来更像是应用程序, 但还是缺少一些组件。

3. 网络应用程序

Windows DNA 应用程序是一个传统意义上的应用程序, 因为它们向应用程序的用户提供服务。这两种应用程序的创建方式不同, 组件构成也不同。传统应用程序在开发过程中需要一套特殊的文件, 但是分布不同的输出。例如: Visual Basic 应用程序有一个 .vbproj 项目文件, 多个 .frm、.cls 和 .bas 文件和一系列构成应用程序项目的 OCX 组件。在程序分布之前, 这些文

件被编译成可执行文件，可执行文件的执行不需要开发过程中的源代码文件。

另一方面，基于脚本的应用程序是由一系列文件组成的，这些文件在开发和运行期间相同，没有编译成可执行文件。例如：网络项目中 .htm、.asp 和 .dll 文件是传输给网络服务器的相同文件。只有当浏览器向网页发出请求时，这些网络文件中的源代码或脚本才在客户端或服务器上执行。建立这些应用程序依赖于万维网 (WWW) 的结构，为使这些文件的功能像应用程序那样，需要增加一些复杂性和功能。

4. 网络应用程序设计

为了建立一个网络应用程序，开发人员必须考虑一些程序设计和开发的新知识，Windows DNA 的灵活性允许各种客户访问各种服务，我们将重点放在一个网络应用程序上。在一个网络应用程序内，将浏览器作为一个主要的用户界面来使用。在网络应用程序内的信息将使用 HTTP 协议从服务器传输到客户端。应用程序服务器将是微软的 ISS，既执行网络功能，也执行应用程序服务器功能。最后，我们的业务和数据访问逻辑将会被封装在 COM + 组件内，作为链接应用程序的通道。

(1) 浏览器作为用户界面

对一个网络应用程序来说，用户界面是表现在网络浏览器内的，这就意味着客户显示形式是浏览器增强型或者浏览器依赖型。所选应用程序的类型应该以用户安装的浏览器为依据，如果能保证大多数用户都使用某一特定的浏览器级别，那么就该考虑一下这一级别浏览器的特征，如果有许多不同种类的浏览器在同时使用，就只能支持一种浏览器依赖型客户显示形式。使用 ASP 传输应用程序的一个优点是能判断出当前正在访问应用程序的用户使用的是哪种类型的浏览器，这已经在第 6 章 6.2.5 节中介绍。通过了解当前用户所用的浏览器的类型，可动态改变客户的显示形式以支持浏览器的增强功能。

(2) HTTP 传输协议

客户与服务器之间的通信层对应用程序的设计和实现来说是至关重要的。超文本文件传输协议 (HTTP) 规定了服务器如何接受和处理客户的请求，以及信息如何发送回客户。根据所支持的客户显示类型的类型，使用这个协议可能有不同类型的信息传输。由于对浏览器依赖型客户的基本支持，HTTP 传输的信息限制为构成网页的 HTML、图像、信息载体 cookie 以及一些客户端脚本。一个增强型客户可以使用一些特殊特性，如远程数据服务 (Remote Data Services) 或在 HTTP 上的 COM，通过 HTTP 进行通信。但是这些支持依赖于客户端浏览器的功能。在任何一种情况下，HTTP 协议没有客户与服务器之间长期连接的概念。

(3) IIS 和 ASP 作为应用程序服务器

微软的 IIS (作为网络服务器) 与 ASP (作为应用程序服务器) 的组合能为网络应用程序提供应用程序组件。对于为显示静态网页服务的网页和网站，IIS 具有提供信息的功能。但当网络应用程序需要动态显示信息时，就必须把可生成动态页面的 IIS 网络服务功能与 ASP 对象集成功能结合起来，传输更稳固的动态网络程序。ASP 脚本功能可以支持脚本内的业务逻辑，链接业务逻辑组件，直接经 ADO 访问数据库，并且可使用数据组件检索信息。

(4) COM + 作为通道

组件服务器的 COM + 功能使 Windows DNA 应用程序能基于组件进行设计。然而一个功能齐全的应用程序可以在 ASP 脚本和 HTML 外构建，当业务逻辑和数据访问被分离并放进组件后，一个更稳固的可扩充的应用程序开始出现。将业务和数据访问逻辑放入执行前编译好的

组件里，应用程序的性能就会改善。并且，随着性能的改善，应用程序有效支持的客户数量就可增加。

13.2 组件

谈到组件和基于组件的应用程序时，了解什么是组件就显得非常重要。在本书中，我们已见过一些组件在ASP中使用，ADO就是一系列支持访问数据源的组件。ASP 3.0版的六个内置组件允许开发人员访问网络应用程序的某些部分。对可扩展的网络应用程序来说，组件的概念是非常重要的。

13.2.1 组件定义

组件是包含可改变数据形式的代码的对象，可通过一系列可用的特定公共服务对此代码进行访问。

以下是对组件的更直接的定义。

从一种更实际的意义上理解，组件是代码的封装，这些代码执行应用程序的一些功能。

这个功能可能是业务规则的处理，如销售税的计算，也可能是从应用程序的数据库检索某些信息。组件的关键特征是当创建组件时，组件的代码和与组件相关的信息一起打包。在这种方式下，如果同时使用多种版本的同一组件，每一个组件的信息都与其他组件的信息保持独立，不存在信息之间发生混乱的问题。

另外，根据组件的工作类型，组件可以由其接口来定义。为了了解关于接口的详细资料，必须先了解COM。

13.2.2 COM和COM+

COM指组件对象模型(Component Object Model)。

COM是面向对象范型的一个分支，是一个基于二进制标准的规范，这个标准规定如何通过接口进行重复使用。

COM是由微软开发的对象模型，可在所有的 Windows平台上执行。它构成所有 Windows应用程序的基础，因为几乎所有与基本操作系统的交互都通过 COM定义的接口来完成。COM定义了组件互操作性的标准，即组件互相之间交互的能力。组件不需要用指定的语言编写，只需指定组件之间如何通信和与操作系统如何通信。组件必须遵守的互操作性标准指定后，不同的开发人员在同一应用程序中创建组件时的协作变得非常简单。这就意味着，为 COM编写的组件可以重复使用，并不依赖于编写时所用的语言。一个应用程序包含的组件可以由 Visual Basic、C++、Java甚至COBOL编写，只要遵守COM规范就不会出现问题。

COM+是组合了COM和Windows 2000的微软事务服务器(MTS)的一系列服务。MTS提供措施管理组件的生存期，不用担心应用程序对象的创建和破坏，COM+有能力对此进行管理，所以可将重点放在实现业务逻辑上。随着COM+的进入，MTS的功能已经合并到操作系统中，被称为微软组件服务。我们将在下一节对它进行简单介绍，在15章将详细介绍。

13.2.3 组件服务

随着COM + 的发布, MTS在Windows 2000下作为一个单独的实体已不再存在。目前, 其功能已成为操作系统的基础部分(现在称为微软组件服务), 并且不只包括事务性服务和组件生存期的管理。

组件服务提供了一些新的功能, 在 MTS下组件可以应用这些功能。由于操作系统服务(而不是开发人员)能处理越来越多的特性, 开发稳固的可扩展组件变得更为简单。内容如下:

- 组件负载均衡(Component Load Balancing, CLB) CLB允许多个应用程序服务器为一个应用程序提供相同的 COM + 对象。如果需要对象, 创建请求首先发送到 CLB服务器, 然后再将请求发送给合适的应用程序服务器, 这根据一定的准则进行(例如运行应用程序的机器的繁忙程度和距离多远)。然后, 在组件生存期内客户端的应用程序与这个服务器交互, 在COM类级上实现负载均衡。但是, Windows 2000首次发布时不支持组件负载均衡, 或许在某个 Service Pack中或在未来的 Windows 200x版本中会支持组件负载均衡。
- 排队组件 将COM与MSMQ的特性相结合, 提供一种异步地调用和执行组件的方式。没有接收者或发送者可用或可访问, 处理也可发生。当客户调用一个排队组件时, 调用排队组件记录器, 它将调用打包成给服务器的消息的一部分, 并放入队列中。排序组件的侦听器者从队列中检索出消息, 并传输给排队组件的执行者, 执行者调用服务器组件并产生同样的方法调用。
- 内存数据库支持(In-Memory Database Support, IMDB) IMDB是瞬态的具有事务性数据库风格的高速缓存, 存在于 RAM存储器中, 可对数据所在的机器进行极快的访问。需要高速数据库查找能力或需要管理瞬态过程的 COM + 应用程序可以使用IMDB。可以从 COM + 对象中利用 ADO或OLE DB接口访问IMDB中的数据。与组件负载均衡相同, Windows 2000的首次发布不支持IMDB技术, 或许能够在未来的 Windows 200x中支持IMDB技术。
- 对象缓冲 对象缓冲是由COM + 提供的自动服务, 可使组件在缓冲池中保持有效, 可以由需要组件的客户使用。应用程序一旦运行, COM + 管理缓冲池, 处理激活对象的细节, 并按照所指定的标准重新使用。为了缓冲组件, 组件必须是无状态、没有线程约束、并且是可以聚集的。VB组件有线程约束, 这就意味着 VB组件不能被缓冲。

1. 事务管理器

微软组件服务具有组件事务管理器的功能, 详细内容可参阅第 19章。在管理事务的过程中, 组件服务首先检测组件是否参与事务处理, 了解组件的事务处理需求是什么。一些组件忽略事务处理的过程, 而一些组件应该或是必须参与事务处理。一个组件开发和部署时, 开发人员可对这个组件的事务处理参数进行设置。如果设置的话, 组件服务利用这个信息决定组件应该如何参与事务。

2. 组件管理器

除了管理组件与事务交互, 组件服务还可以管理组件自身。而这种功能是 MTS的一部分, 考虑MTS特点时通常没有注意它。组件服务利用 COM + 组件的环境帮助管理组件。

环境是用于维护一个对象集合的一系列运行期属性。每个对象在生存期间与一个环境相对应, 多个对象可以在相同的环境下运行, 多个环境可置于相同 COM单元内, 环境属性允许

组件服务提供运行期服务，这些属性保留决定执行环境如何为其中的对象执行服务的状态。

3. 安全管理器

组件服务为COM + 组件提供许多安全特征，其中自动安全特征在不添加代码的情况下就能使用，并且可以配置。另一些安全特征可以直接并入组件的开发中。基于规则的安全性可通过编程和管理实现，它是COM + 主要的安全特征，允许安全性的粒度降到特定组件的方法级，允许所有的用户访问组件，但对某些用户限制使用组件的某些方法。

13.2.4 以数据为中心的组件

三层结构的底层是数据访问层，这层负责与所需的数据源集成，这些数据源可能是SQL Server或Access数据库、Exchange消息库、MSMQ消息队列和UNIX传统应用程序，它们可能存在于服务器自身、LAN上的一些其他服务器或在互联网上的任何地方。数据组件不仅负责封装数据的存取，而且负责使数据的定位对于应用程序透明。应用程序需要做的就是实例化和使用组件，其他的工作由组件自己完成。

使用以数据为中心的组件的原因

Windows DNA应用程序中以数据为中心的组件对于三层结构是非常必要的。显然如果没有数据层的话，只剩下两层的应用程序。更为重要的是在组件中封装数据存取可以生成更为稳固的应用程序。使用以数据为中心的组件的原因是：

- 使开发人员不受数据库内部结构的影响 这种封装以面向对象设计为基本原则。如果组件的内部工作没有暴露给使用组件的开发人员，那么这些内部工作根据物理数据存储变化可以进行更改、更新、增强或替换。另外，开发人员无法通过以不正确的次序改变数据库的行，来绕过安全措施和存储过程。
- 对不同的数据源提供一致性的数据访问 开发人员通过在一个共用接口中封装对不同数据资源的访问，可以使用相似的方法访问数据库，而不必知道数据存在于何处。这意味着访问存储在SQL Server数据库中的数据，可以使用与访问存储于UNIX系统的平面文件中的数据相同的方法完成。
- 透明数据定位 访问数据库的物理方法封装在组件中，数据的位置与组件的用户没有任何关系。用户仅仅访问组件，组件完成到数据源的必要连接并检索数据。

13.2.5 业务组件

三层的中间是业务组件层。有时称为业务逻辑层或应用程序层。其工作就是为应用程序提供功能，可以为电子商务应用程序管理会员卡或计算地图应用程序中的两个位置之间的最佳路线。

业务组件设计隐藏了一系列业务规则需要的复杂交互，以方便处理和让用户界面设计人员不受基本数据的影响，这些数据简单地与业务组件提供的方法进行交互，表现用户输入的信息，解释组件处理的结果。

使用业务组件的原因

通常数据组件是N层结构的三种组件类型中使用最为广泛的一种，使用业务组件仍有许多不得已的原因，有些原因与使用数据组件类似或相同：

- 业务规则封装 通过在组件中封装业务规则，开发人员使用它时不必考虑如何处理规则。

组件有严格定义的接口，用于和软件开发人员交互。无论里面发生什么都不影响接口，组件能够传输正确的结果。

- 应用程序的重复使用 通过在组件内封装业务逻辑，在多个应用程序中重复使用业务功能变得非常容易。例如，一个企业可以用一个组件计算销售佣金，可以允许任何需要计算佣金的应用程序使用同一个组件。这个相同的组件专用于计算佣金，因此所有佣金都可用完全一样的方法计算。
- 性能。在开发网络应用程序时，使用脚本创建应用程序的业务规则是非常简单的。这使得获得一个应用程序很容易，并能很快投入运行。然而，脚本不仅违反业务规则封装和应用程序重复使用，而且会引起性能问题。由于每次运行脚本都要进行解释，而组件执行的是编译后的代码，脚本的性能不能与编译组件的性能相比。

13.2.6 用户界面组件

网络应用程序的最后一层是用户界面层。应用程序的中间层产生的信息在这一层传输到客户端。如何表现通常决定于 HTML 或 DHTML 模板。在一些场合中，业务逻辑可以选择用于显示的模板。

关于将用户界面创建代码封装到组件有许多争议。由于 ASP 脚本能够嵌入到 HTML 页中，直接利用 ASP 脚本将动态值插入 HTML 模板是很容易的。如果信息是编译组件的一部分，那么不重新编译组件而改变表现形式是很困难的。

使用用户界面组件的原因

对应用程序来说，在几种情况下用户界面组件是有意义的。通常与简化重复性任务有关，或者因为单独使用脚本不能满足性能要求。不管哪种原因，都存在其他一些非编译方法，可以不使用基于组件的用户界面而获得类似的功能。

- 重复表现 网页应用程序中某些部分在许多不同的页面中是重复的，这些页可能包含不同的信息。这种表现方式可以封装为一个组件，开发人员利用组件可以很快实现这种应用程序，并给出一个更标准的外观。然而，类似的功能也可以使用一个包含文件来完成，该文件含有能够将信息复制到浏览器的方法。
- 性能 要创建一个可扩展的网站，性能是一个重要的问题。应用程序任一方面的性能越好，这个应用程序作为整体运行得就越好。将复杂的表现逻辑封装到编译的组件内，网站可以运行得更快。但是，不重新编译组件，就不能改变表现方式，这限制了其灵活性。另外一种方法已在第 3 章介绍，将重复的 HTML 作为字符串存储在脚本中，然后将脚本存储在应用程序级的变量中。这在没有牺牲 ASP 和 HTML 的灵活性的情况下获得了编译代码的性能。

13.3 组件应用程序设计

在建立基于组件的应用程序时，首先必须了解如何使用组件设计应用程序。这包括把应用程序分割为组件的过程和如何设计组件本身。

明确这一点后，必须进一步学习有关创建基于组件的网络应用程序的问题。尤其必须考虑组件的设计、所选的组件接口、如何将组件集成到一起和用于实现上述工作的工具。

13.3.1 转换到组件

将应用程序从传统的整体式或客户 / 服务器应用程序转变过来之前, 首先必须了解如何将应用程序的功能分为组件, 这就是分解, 可用多种方式来完成。讲述各种对象设计方法的书很多, 所以我们不详细讨论。关键是选择或创建一种方法, 与处理业务的方式相适应, 并且能基本上不改变现有的开发过程, 除非对现有的方式不满意。

每种方法都有许多支持者, 作为应用程序设计者和开发者最好选择一种合适的方法。如果找不到喜欢的, 那么就采用现有方法的一部分, 创建自己的方法。只要设计的方案不仅能有效地创建应用程序, 而且为其他人对你的应用程序的理解提供指南, 那么这种方法就是一种有效的设计方法。

当开始将应用程序分解成组件时, 第一步就是将应用程序的功能分为三层, 这三层构成基于组件的应用程序。开始工作时, 一个好方法就是观察应用程序的每一部分, 判断出其属于哪一层。如果找出这种关系有困难, 应将应用程序划分得足够细。这样, 各个元素被分解为许多部分。目的就是使每一部分能很好地归于表现层、业务逻辑层和数据层中的一层。

当这些组件开始进入设计时, 在头脑中要保持应用程序的大体形式。

如果做不到这一点, 那么当最后将组件连在一起时, 就可能迷失整个航向。所以记住设计的最终目的, 就能使组件很容易地集成为一个有效的应用程序。

13.3.2 应用程序设计

在建立一个组件应用程序的过程中, 应用程序设计非常重要。如果没有指定应用程序的需求, 创建支持应用程序的组件将会非常困难。在设计整个应用程序的过程可以使用许多方法。例如: 在 Use Cases 方法中, 通过定义如何在指定的实例中使用应用程序来创建的应用程序设计。

设计一个基于网络的应用程序时, 必须克服在传统应用程序中不会碰到的难题。首先, 也是最重要的, 作为应用程序设计人员不能限制用户用来访问应用程序的工具。除非在受控制的局域网中, 否则人们用不同类型和版本的浏览器访问网站的概率是十分高的。这样, 应用程序必须设计成支持广泛的表现类型。

访问基于互联网的应用程序常常通过慢速的连接进行, 如使用 33.6k 或 56k 的调制解调器。应用程序设计中有两个重要的影响因素:

- 首先, 图形图像文件的大小, 包括那些组成用户表现层的图像, 必须能够快速传到客户端。如果应用程序的用户总是在等待应用程序下载图像, 那么这个程序将很快被放弃。基于网络的应用程序含有图像是比较好的, 而且非常有用, 但如果对使用性有负面的影响, 那么就必须找到另一种向用户传递图形信息的方法。
- 其次, 连接的速度同样可以解释为每次与服务器的交互所要耗费的时间长度。这种交互之间的暂停会对应用程序的使用性和性能产生影响。如果这种延迟太长或发生得过于频繁, 那么应用程序的性能将变得更差。因此, 作为基于互联网的应用程序设计人员, 必须注意应用程序在服务器与客户端之间“回合”的次数, 一个仅在高速局域网上运行的系统将会比那些依靠公共互联网工作的系统经受得住更多次数的“回合”。

13.3.3 设计网络组件

除了一些网络应用程序整体设计的概念之外,还有一些特殊的组件设计准则,遵循这些准则可以建立起高效的网络应用程序。由于网络应用程序必须能够处理大量用户而不引起性能下降,通常用于建立应用程序的组件应能支持这种需求。对于一个基于网络的应用程序来说,建立一个具有“用户会话”概念的应用程序也是有一定难度的。在这个应用中,一系列分散的用户请求都被当作与应用程序的单一交互。尽管有许多工具可用于保持用户会话,例如Session对象,但是应用程序设计人员还是需要了解这些工具是如何影响系统性能的。

1. 无状态组件

如果回过头去看看对象的基本定义,通常对象是以代码和数据作为一个整体的单元。把数据和对象代码组合起来成为一个包,同一对象可能同时存在许多实例,用于表示不同的信息。对客户/服务器应用程序来说,这是非常高效的设计,应用程序可以创建表示特定实体的对象,然后通过这个对象的接口,实现应用程序的功能。由于对象本身包含数据,所以能在不需要其他数据源的情况下实现这些功能。

这也意味着对象的特定实例是与其所包含的数据相关的。如果对象需要参与不同的交互,则没有办法用它自己表示不同的信息。唯一的方法就是创建一个新的对象实例,并加入所需数据来实现交互。基于对象的应用程序知道与之交互的对象类型,对象中包含的信息的细节却不是应用程序所关心的。缺点是必须用两个对象来表示两块不同的数据。

在支持单用户或有限用户的应用程序中,创建新的对象来表示新的数据项通常不会降低系统性能。但是处理网络应用程序时,同时支持成百上千个用户,为每个用户创建多个对象将会对系统产生严重的影响。幸好,网络应用程序N层模型提供了处理这个问题的独特方法。

由于客户的每个请求都被作为与网络服务器的一个单独连接,所以在请求之间不需要绑定服务器上的任何内容到客户端。当用户正在用浏览器下载一个应用程序页面时,即使正在该页面上进行信息交流,在请求之间服务器不必为这个用户进行任何处理。服务器可以自由地与其他用户交互。

但是如果当第一个用户创建一些对象与这个用户单独链接时,服务器必须确保这些对象被保留下来,以便在用户下次访问时能继续使用。这就是说服务器资源将会闲置而等待下一个请求。如果把这种情况扩展到成百上千的用户,许多对象就会闲置,等待下一个请求的到来,这就浪费了服务器资源。那么如何解决这个问题呢?

如果能够设计一个组件,这个组件能够执行相同类型的工作,但不需要在组件中保留任何信息,那么每次使用过后就可以丢弃。或者如果组件开发成像COM+那样能缓冲的组件,只要返回到组件缓冲池即可。当用户结束工作时,就不需要保持此对象的引用。由对象占用的资源就可以重新返回服务器。利用这种方式快速释放资源,应用程序在同一时间所支持的客户数量将会大幅度增加。

如果一个组件不携带与环境相关的任何信息,则称为无状态组件。因此一个状态组件在一个客户交互与另一个客户交互之间携带与环境相关的信息。在网络应用程序中使用下面两种方式创建状态组件。否则,创建的组件是无状态的:

- 在页面之间,通过在会话层变量中存储对组件的引用,使组件具有状态。
- 在组件里设置一系列属性,然后调用一个方法执行交互,将会使组件具有状态。将所有属性作为方法的参数传递,这将使组件的状态最小化。

2. 无会话应用程序

当设计网络应用程序时，另一个需要关心的问题是设计标准，来自于网络内在的无状态性。前面提到过，一旦服务器处理完客户的请求，将不再保存客户的信息。这就是说，在服务器应用程序上没有客户的历史记录。通常应用程序由一些必须连在一起的步骤组成。如果服务器在每一步之间忽略客户，那么设计应用程序就非常困难。

当应用程序平台只支持无会话应用程序时，就如同网络服务器一样，必须在其他地方保留会话状态。有许多工具可以使用，但每一种都对应用程序的可扩展性产生影响。第一步就是每次向应用程序发出请求时客户如何识别自己。

当向服务器发出请求时，可以通过从客户端传输到服务器的 cookie 进行识别，这个标识可以是包含在一个隐藏的 FORM 元素中的唯一标识符或者是添加到 URL 中的参数。在任何情况下，这个标识必须与服务器上的一些信息相一致，对客户来说是唯一的。这个标识可能是指向包含所有会话信息的对象的引用，可能是保留会话信息的数据库表的主键，也可能表示在客户的请求或者窗体字段、URL 参数、cookie 中还存在其他信息，这些信息包含当前的会话状态。

总之，保留会话信息的三种方式是：

- 在 Session 对象中存储数据。
- 在数据库中存储数据。
- 每次发出请求时，通过 Request 对象在网页之间传输信息。

每种方法都有优点和缺点，所选方法必须对应用程序的性能、可扩展性和使用性的影响最小。

信息存储在 Session 对象中时，可以快速检索，但这意味着信息对特定的服务器来说是局部的。如果应用程序在网阵 (Web Farm) 范围内使用，那么会话信息仅存在于网阵内的一个机器内，如果用户要访问他的会话信息，必须每次直接返回同一服务器。这妨碍了使用负载均衡服务器，将影响应用程序可扩展性。

如果在数据库内存储用户会话信息，网阵内的多个机器都可以访问。在这种方式下，客户端不受特定的服务器约束，网阵内的操作更为高效。这种方法的缺点是：从数据库中检索信息的速度远低于从服务器内存中检索信息的速度，在数据库中还没有一种方法存储实例化的对象，就像 Session 对象那样。另外，还没有一种像 Session 对象这样的易于使用的接口来访问信息。

使用客户端存储会话信息时，将服务器上保留此信息的负荷移动到客户端，通过减少服务器处理请求的数量，就能节约系统资源。也就是说，同样的服务器资源可以支持更多的用户。这个方法的缺点是：缺少处理以这种方式传来的信息的通用接口；同时要求信息在客户对服务器的每次请求时来回传输，还依赖于客户传输对应用程序的功能至关重要的信息，这对网络连接形成一定的压力。

在选择保留应用的会话状态的方法时，必须检查每种可能性，选择一种最合适的方法来满足设计要求。对于任何系统，可以使用三个选择的组合。例如：如果可以在两个页面内用状态组件解决一部分问题，那么可以把此组件的引用存储在 Session 对象中，从一页传送到另一页，而且当不再需要时，可以立即删除。注意必须遵守第 15 章中组件设计的要求。同时应用程序会话的所有其他状态信息可以存储在 cookie 和隐藏的窗体字段中。这种情况下，三种不

同的方式是互补的而不是互斥的。

13.3.4 组件设计

组件设计有两个关键问题。组件接口如何与外界进行交流，接口是外界与组件提供的服务一起工作的唯一途径；另外，还需要观察组件的物理定位，这决定组件如何与构成应用程序的其他组件协同工作。

1. 组件接口

组件接口是组件与外界交互的机制，它能够访问组件提供的信息和功能。接口是其他组件或应用程序与组件功能交互的唯一途径。接口是由组件的公共属性和公共方法构成的，组件的所有私有属性和私有方法只能从组件内部访问，不能通过组件接口使用。

下一章将详细讨论COM和接口。

设计组件接口时，必须支持组件设计的其他方面。即接口设计将直接导致组件是有状态的还是无状态的。如果创建的组件接口依赖于组件执行功能前的属性设置，那么此组件在某段时间内保持某个内部状态。这需要多次调用组件，其中几次设置属性，然后一次调用执行一些操作。通过下面的例子可以进一步理解上述问题：

```
<%  
Dim obj As Server.CreateObject("MyComponent.Test")  
obj.Property1 = 123  
obj.Property2 = 234  
obj.Property3 = "abcd"  
Call obj.Method1()  
Set obj = Nothing  
%>
```

如果想要设计这个组件的接口，调用该接口的一个方法并传送执行功能所需的值，那么这个组件将是无状态的。执行功能只需调用一次组件，没有内部信息需要存储在组件内。如果创建另一个方法Method2，可用下面的调用：

```
<%  
Dim obj As Server.CreateObject("MyComponent.Test")  
Call obj.Method2(123, 234, "abcd")  
Set obj = Nothing  
%>
```

在释放组件前，新的接口调用一次组件。而在前一例子中，在释放组件前需要调用四次组件。在可扩展性方面，越早释放一个组件并使资源返回系统，系统的可扩展性就越强。

2. 组件定位

除了组件支持的接口外，组件的物理定位对应用程序的可扩展性也有影响。应用程序与其组件交互时，将以最快和最高效的方法进行。当使用组件创建通讯通道时，其浪费的时间将比不使用组件时耗费的时间更多。要提高应用程序的扩展性，必须记住增加的任何处理时间或者资源占用，至少要乘以计划同时使用的用户数目。一旦这样做，将很容易看出资源占用和时间的微小变化将引起非常严重的问题。

要确保ASP脚本与组件及组件之间的交互，以最快并且最高效的方式进行。这意味着组件应该在与ASP应用程序相同的进程中运行。下一章将讨论COM+如何改变组件实际执行的位置。为了提高速度，在与ASP应用程序相同的进程中用运行组件，组件的速度越快，组件

就越可能影响ASP应用程序。当组件在与ASP应用程序相同的进程中运行时，所有对组件的调用都使用直接功能调用，这是访问组件的最快方式。

如果我们在不同进程中运行组件，那么组件与应用程序之间传输的信息都将通过组件和应用程序进程的边界。在Win32环境下，这是一种大量消耗资源和时间的操作方法。它将会花费更多的系统资源去调用，同时也需要更多时间。这也发生在对象与应用程序之间的双向会话中。任何信息(例如方法的参数)从应用程序传递到对象要通过进程的边界。在返回的过程中，对象的任何信息(如返回值)想要返回应用程序，也需要同样的过程。

也可以利用Win32环境的DCOM功能在另一个物理系统上运行组件。即使对应用程序这个连接是透明的，主要性能仍会下降。访问在不同进程中运行的组件的每件事物在DCOM组件中都能可靠保持，但客户与服务器的通讯时间也将剧增，特别是应用程序从局域网移植到广域网或互联网上时，传输时间会明显增加。如果要在ASP应用程序中使用DCOM组件，应该非常谨慎。

13.3.5 组件集成

创建完应用程序的所有组件后，需用一种方法将这些组件集成起来。如果没有一些“胶水”告诉组件彼此如何交互，就无法建立应用程序。可以使用其他组件作为“胶水”将组件集成在一起，实际上这将使建立应用程序像创建组件一样困难。基于组件的应用程序的优点之一就是可以快速地将一系列组件连结起来获得应用程序。所以为了方便，在应用程序中使用ASP脚本作为组件的“胶水”。

在容纳应用程序的系统中，自然需要安装ASP。安装ASP的系统就不仅仅是网络服务器，而且成为应用程序服务器。为使网络应用程序为客户服务，需要访问创建应用程序所需的组件。

能用来建立应用程序的组件有两种类型。一类是自定义并安装在服务器上的组件，另一类是服务器上现有的组件。组件可由微软、自己或其他第三方创建。

1. 使用现有的组件

使用现有的组件时，首先必须确保对组件的使用不会打断其他依赖于该组件的应用程序。例如，如果将组件更新为一个新的版本来支持你的应用程序，就有可能打断现有的依赖于组件的特定版本的应用程序。另外，如果组件依靠注册表设置来保持其配置，为了使其适合你的应用程序而修改了设置，那么其他使用此组件的应用程序可能不再起作用。千万注意，如果遵循基本的接口开发准则，那么就不会发生上述的任何一种情况。

查看哪些其他ASP应用程序正在使用组件的最简单方式是使用一个简单文本搜索例程，如GREP或Visual InterDev的Find in Files函数。所有ASP脚本代码都是文本格式的，因此可以在应用程序服务器中搜寻所有的.asp文件，寻找你想要使用的组件的ProgID。这可以直接指出使用此组件的应用程序。修改组件的配置，就可以为测试这些应用程序建立不同的版本。

2. 安装新组件

在应用程序服务器中安装新建的组件有许多方法。

一种方法就是使用一个包含组件的文件，一般是一个.dll文件，将其拷贝到服务器上并且注册到操作系统。这样，必要的信息就被置入系统注册表中，允许应用程序引用组件，并让系统知道引用哪一个组件。用这种方法安装组件，要确保组件需要的所有文件都存在于服务

器上。缺少文件，组件就不会正确地工作，甚至不能注册。

第二种方法是在开发机器上先建立一个安装程序，将其拷贝到应用程序服务器上，然后运行这个程序。这是一种极简单方法，能保证组件的文件都能正确地与组件安装在一起。这种方法的弱点就是需要同应用程序服务器的控制台进行物理交互。另外，如果安装程序不能正确识别组件的版本，就很可能覆盖已有版本中的 DLL，这可能使其他应用程序不能正确运行，甚至使服务器崩溃。

第三种方法就是通过使用一个 COM + 应用程序来完成。如前所述，COM + 应用程序由一系列组件构成，由微软组件服务作为一个单独的组管理。在第 15 章中，我们将介绍如何创建 COM + 应用程序，如何将一个应用程序从一台应用程序服务器移植到另一台应用程序服务器上。

13.3.6 建立组件

所有关于建立组件来封装业务逻辑或数据访问逻辑的讨论，为你建立自己的组件铺平了道路。最重要的是使用什么样的工具创建组件。COM 最大好处是没有语言限制。这就是说，几乎可用任何语言创建组件，而且可以选择各种工具来编写组件。

1. 选择工具

在市场上有许多可用于创建组件的开发工具。包括：

- Visual C++ 这种工具可以创建高性能、最轻量的组件。利用 Active Template Library 时应注意组件创建过程中的复杂性。要得到性能就需要付出代价。这是创建组件最难的一种方式，也是最难学习的。我们将在第 17 和 18 章中学习 Visual C++。
- Visual Basic (VB) 在一系列工具中，Visual Basic 是创建组件最容易的工具之一。这些组件并不一定具备 C++ 组件的最佳性能，也不一定具备 C++ 组件的灵活性。如果正确地设计和使用，大多数基准程序表明 VB 组件能够达到 C++ 组件速度的 90%~95%。大多数开发人员能够使用 VB 快速简单地创建组件，下一章将介绍如何使用 VB 创建组件。
- Windows 脚本组件 微软已经将脚本语言扩展到组件创建领域中。通过添加用 XML 编写的接口定义，能够采用与创建 ASP 脚本完全一样的工具和代码来创建组件。第 16 章将进一步介绍此内容。

2. 测试组件

创建好组件后，在使用前必须进行测试。被测组件不仅要确保能够完成设计的操作，而且要确保对系统的其他部分无不利影响。组件需要通过三种类型的测试。

- 功能测试 这种测试方式确保组件工作正常。通常这种性能的测试是通过向组件输入一系列已知正确输出的输入来进行的，将组件输出结果与已知结果相比较，任何错误可通过修改组件功能来解决。
- 强度测试 为了确保组件运行不影响系统的其他资源，要进行强度测试。通常由测试软件来成，如 WAS 或者 WCAT，此类工具软件模拟大量用户同时访问组件。在这种测试中，通过监控系统资源确保组件没有占用过多的资源。
- 组合测试 组合测试观察组件与构成应用程序的其他组件如何交互。这些测试确保没有争夺相同资源、依赖于同一 DLL 的不同版本等问题。

一旦测试完成，组件可以安装到应用程序服务器中，并可以集成为应用程序。

13.4 应用程序设计范例的研究

本章已经介绍了许多创建N层 Windows DNA应用程序的理论,这种理论覆盖了创建基于组件并可扩展的网络应用程序所需的基本知识。理论问题说到底还是理论,多数开发人员喜欢实际的例子,当然只有足够的理论才能指导实践。

本节用创建Windows DNA应用程序的理论实际创建一个应用程序。下面介绍如何设计一个基于网络的Windows DNA应用程序。

- 首先要明确建立的应用程序所要解决的问题,这是最重要的,但在设计过程中最容易忽视这一步骤。
- 明确问题以后,将考虑应用程序的设计,包括组件结构的设计和在应用程序中的组件接口的设计。
- 最后我们将考虑设计折衷问题。这将影响到应用程序的可扩展性和可扩充性。

13.4.1 明确问题

WhizzyBang.com公司,与所有具有成长性的小型网络公司一样,面临着持续不断增加新雇员的问题。雇员分布在世界各地,他们的职责和头衔总是在变化。为了维持人事资源支持系统的工作,他们决定通过Internet完成业务,并且创建基于网络的人事资源信息系统。

人事资源系统将首先支持以下功能:

- 招聘管理人员负责创建雇员的原始记录。
- 人事资源管理人员必须确保信息有效,并为雇员处理合适的部门工作文书。
- 雇员可用此系统选择他们的健康保险赔偿费,新雇员和现有雇员均可使用。

由于这个应用程序的信息可用于其他系统,因此必须将信息存储在一个集中式的系统中。所需的其他信息送到公司外的某一机构中。应用程序使用的数据存储在下述系统中:

- 雇员数据库存储在SQL Server 7.0 关系数据库中。
- 健康保险选择信息存储在Access数据库中。
- 部门工作文书通过XML格式的e-mail传输到一个处理器中。

应用程序用户的位置可以通过国家来定位,他们都可以对公司网络进行虚拟个人网络(VPN)访问。但是在数据中心和VPN提供者之间仅靠128k ISDN线路连接。VPN允许用户拨号进入公共互联网,然后通过公司的防火墙进入公司内的服务器,VPN被配置为仅仅允许HTTP协议通过。

运行应用程序的系统被放在一个集中的站点,这些系统专用于这个应用程序,但是应用程序所访问的数据库放置在分开的系统中,应用程序的用户使用Windows 98和Windows 2000工作站。

只有当工作站属于公司的网络时,才能进入应用程序。应用程序总是检查用户执行的程序,以确认为最新版本。

设计的影响

既然了解了应用程序的高层要求,我们要看一看这些要求对程序设计的影响,检查这一点是关键。如果要设计应用程序,而不注意要求的设计的影响,设计过程将是很困难的,设计出的应用程序也是不可能实现或使用的。

在考虑这些设计影响的过程中，我们应该理解影响的某些方面。首先，需要知道是否有某些彼此排斥的要求。例如：如果一项要求是系统必须允许不连接的膝上型计算机用户访问，而另一项要求是只允许 HTTP 协议访问，设计便陷入了困境。因为 HTTP 作为客户端与 Web 服务器之间的连接协议才能允许访问，这两项要求互相冲突，因此，必须改变要求以支持系统的现状。

为了解这些设计影响，同样要看一看那些非常精确的要求，例如雇员数据库使用数据库的类型。这些非常专业的内容必须明确，这直接影响最终设计。对于那些更一般的要求，例如客户端使用的工作站类型等，在应用程序设计中有更大灵活性。

要求初步通过后，作为设计影响应对下列进行验证：

- 工作站将支持 Win32 应用程序，但是要求应用程序能自更新，这意味着基于浏览器的应用程序很容易支持。在基于浏览器的应用程序中，用户总可以得到应用程序的最新版本。
- 系统所访问的数据库都具备 OLE DB 提供者，能够利用 ADO 与数据库交互。
- 人力资源管理员提供验证步骤来创建新雇员，所以需要有一个通知机制告诉这个用户有新雇员需要验证。
- 既然同一应用程序将用于输入和更新保险赔偿费信息，就应该支持对此数据的指导式交互(用于输入)和交互式交互(用于更新)。

通过认识这些影响，当创建应用程序时，我们就能指导设计向着问题最少的方向发展，远离那些不可能实现的选择，也使实际设计过程错误更少、更加简洁和有效。我们要经常回过头来看看是否实现了这些影响和要求，以确保设计是沿着正确的道路进行的。

13.4.2 应用程序的设计

既然已经有了应用程序的要求，也考虑了最终影响，现在可以开始着手设计了。现在我们的设计有许多方向，有相当多的应用程序设计方案，几乎每一本关于 ASP 的书都有一种方案。所选择的设计方案也许是雇主强加于我们的，或者是过去用过的，或者是曾经见过并对其有兴趣的，无论选择哪一种方案，最终都要达到同样的目的，即应用程序的设计。

选择设计方案时，应该考虑到应用程序开发效果的规模。你是否是应用程序的唯一开发者？你和其他开发者以后会不会扩充和增强该应用程序？应用程序是否必须与其应用程序在组件级进行集成？这些和其他一些问题需要提出，以决定所选择的方案是否适合你的应用程序工作环境。

对于这个例子，我们不使用已经发布的方法（如 UML 或 Use Cases）。正如许多开发者发现的那样，发布的方法在理论中虽然不错，但在实际创建应用程序设计时，最好的方法通常是满足应用程序的独特需求的方法。在我们的应用程序设计方法中，考虑下面五个方面：

- 组件结构：应用程序中的组件如何划分为逻辑层，组件之间如何协作。
- 数据层接口：描述数据组件逻辑上如何与物理的数据存储集成，以及组件结构中其他组件如何访问数据组件。
- 业务层接口：描述表现层如何访问业务组件和每一个组件使用的数据层接口。
- 表现逻辑：描述为了实现应用程序要求的功能，与用户交互的各种各样的可视化界面。
- 集成结构：集成结构是指如何将应用程序的组件集成在一起。这包括考虑组件的物理位置，将组件的封装到一个应用程序中，组件相互交流的机制等。

设计方法主要关心的是应用程序组件的接口，以及如何组合这些接口来创建应用程序。组件接口指定了组件将提供的信息，这些信息由调用者提供的输入决定。组件内部的工作由开发者完成。一旦在设计中指定了组件和组件接口的交互方式，就可以把组件看作是一个黑匣子。按这种方法设计组件，集成结构的主要任务就是将组件联系到一起，这是设计的最后步骤。

1. 组件结构

WhizzyBang.com公司的人力资源应用程序的组件结构将确定构成此应用程序的组件，以及如何将这些组件集成在一起。组件结构提供表现层和实际数据的“胶水”。从图13-3中可以看出组件结构在应用程序中的作用。

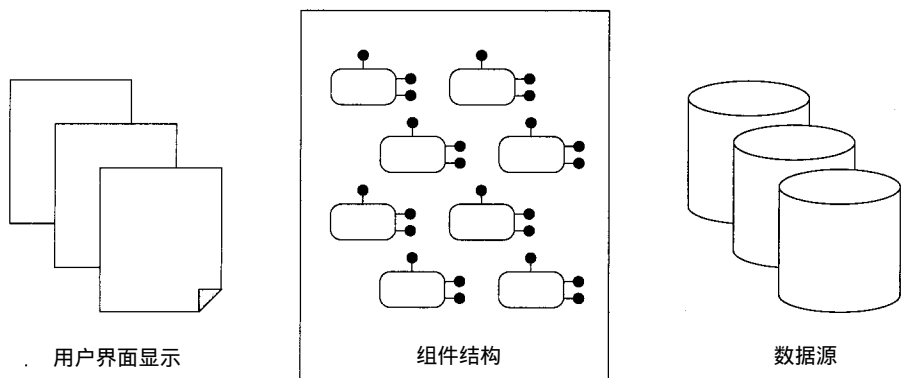


图13-3 组件结构在应用程序中的作用

在应用程序中的组件结构代表业务逻辑组件和数据访问组件。稍后将定义这些组件的接口，现在必须决定创建各种对象。通常对象分解允许先宽松地定义对象，并将此对象逐渐分解成组件对象，直到达到使各个对象尽量独立。首先让我们来看看数据对象。

在这个应用程序中，有三个基本的数据源，分别是 SQL Server雇员数据库、Access保险赔偿费信息数据库和发送税务信息的 e-mail信箱。SQL Server雇员数据库既可用于检索信息又可用于存储信息。雇员信息最初由应用程序创建并存放在数据库中，在应用程序中可以检索和使用这些信息。保险赔偿费信息数据库中的信息是只读方式的，在应用程序中不需要把信息写入保险赔偿费信息数据库中。税务信息的存储恰好相反，是一个只写数据源，即一旦将信息写入此数据存储，在应用程序中就不能从此数据存储中检索信息。

这三个原始数据资源与三个对象相对应，首先可指定为：

- Employee组件。
- BenefitsInfo对象。
- TaxInfo对象。

Employee组件负责与雇员数据库中的所有数据交互，因为这个数据库包含不同类型的数据，如名字、地址、赡养情况和保险赔偿费选择。可以将这个组件分解为多个对象，以访问单个的物理数据存储。将Employee组件分解更小的对象的过程就是对象分解。

在这个例子里，我们将把Employee组件分解成三个独立的对象。选择一个对象命名模式，这个模式将这三个小对象与原始Employee对象联系起来，EmpInfo对象可访问名字和地址信息，EmpDependent对象可访问赡养情况的信息，EmpBenefits对象可访问保险赔偿费信息。

BenefitsInfo对象和TaxInfo对象已经是小结构，这些对象就没必要分解为多个对象。

应用程序的业务层涉及创建和验证雇员记录和雇员保险赔偿费的选择或修改。雇员信息创建部分必须遵循在信息方面的某一业务规则，从而确保信息能正确输入数据库。在这种情况下，可以观察到功能被分离进只负责访问信息的数据对象和负责执行业务规则的业务逻辑对象。

我们将创建应用程序中用到的三个基本业务对象。NewEmployee对象处理与创建新雇员记录相关的业务规则。EmpValidation对象用于验证与新雇员有关的信息，然后处理合适的税务信息。Benefits对象执行有关保险赔偿费的业务规则。

可以看出，业务对象组织与应用程序的功能紧密联系在一起。这为业务规则与表现逻辑之间的交互创造了条件，减少了处理应用程序给定部分所需对象的数目。通过减少应用程序中包含的对象数目，可以增加应用程序的可扩展性，因为每个应用程序部分需要更少量的资源。组件结构的情况如图13-4所示。

业务组件结构包含八个对象。下面我们将接口应用到这些对象，以便这些对象彼此进行交互，并与应用程序的其他部分交互。

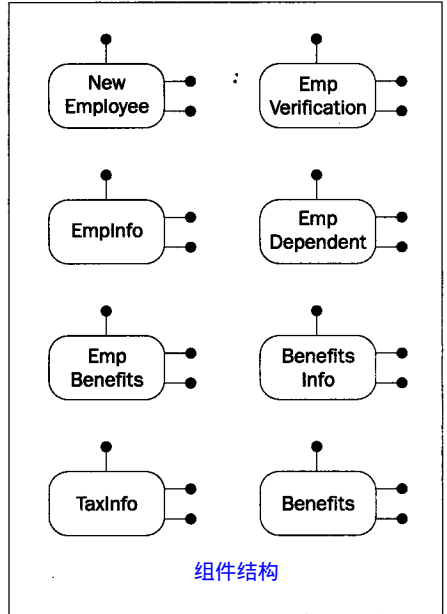


图13-4 组件结构中的对象

2. 数据层组件

数据层组件由五个数据访问组件的接口组成。这些组件用于访问应用程序使用的数据库。数据接口决定业务组件与数据组件的信息检索和设置的交互方法。

这些组件到物理数据本身也存在一个接口，但这个接口的定义并不是应用程序设计的关键。这种访问物理数据方法的约束和支持数据层接口所需的特定信息，一起决定这个接口如何与实际创建的物理数据存储交互。这个物理接口也由存储数据的系统驱动。对于存储在SQL Server雇员数据库中的数据库信息，使用SQL Server的OLE DB 提供者可以访问这个信息。存储在Access数据库中的保险赔偿费信息数据可通过 ODBC连接进行检索。TaxInfo组件所需要的数据传送请求将利用SMTP邮件对象(如协作数据对象，CDO)执行，因为这个数据是用e-mail传送的。

这些组件的外部接口非常重要，因为这是应用程序其他部分访问对象的方法。一旦定义了这个接口，那么业务对象可以通过这些接口与所需数据交互。这些业务对象将其接口暴露给表现逻辑层。

对于所需的五个数据对象，通过它们的公共接口可以定义各自的方法：

- BenefitsInfo对象 其方法有GetBenefitsForClass、GetBenefitInfo和 GetBenefitOptions。
- EmpBenefits对象 其方法有 AddBenefit、CheckBenefitStatus、GetBenefitList、RetrieveBenefit和UpdateBenefit。
- EmpDependents对象 其方法有 AddDependent、GetDependentList、RetrieveDependent和UpdateDependent。

- EmpInfo对象 其方法有 CreateEmployee、DeleteEmployee、EmployeeStatus、EmployeeClass、IsEmployee、RetrieveEmployee和UpdateEmployee。
- TaxInfo对象 其方法有 SendTaxData。

这些接口能够对存储在数据源中的信息进行访问。所有的访问都通过这些对象的方法进行。如果一个业务对象为了显示或更新信息需要与数据源交互来访问信息，那么必须通过这些方法进行。

3. 业务层组件

现在我们已经定义了应用程序如何访问数据。我们必须定义一些组件，这些组件允许表现逻辑按应用程序的业务规则访问数据。我们已经为这个应用程序定义了三个业务对象，NewEmployee、EmpValidation和Benefits，这些对象与应用程序所需的功能相对应。

这些组件的接口将由表现逻辑调用，并且从数据层检索用户界面所需的信息。如果用户可以修改信息的话，还需要定义一种更新信息的方式。这些组件的内部工作负责实现应用程序的业务规则，例如 Benefits组件负责保证雇员保险赔偿费选择是有效的。与数据对象一样，这些组件内部的实际工作对我们来说并不重要，只要能够正确处理业务逻辑就行了。

三个业务逻辑组件支持下列接口：

- Benefits对象。其支持的接口有 AddBenefit、AddDependent、DeleteBenefit、DeleteDependent、RetrieveBenefitList、SetBenefit和UpdateDependent。
- EmpValidation对象。其支持的接口有 GetEmployeeInfo、GetValidationList、LoginEmployee、SetEmpTaxInfo和ValidateEmployee。
- NewEmployee对象。其支持的接口有 AddEmployee和CheckData。

通过已定义的接口，应用程序的表现逻辑必须检索和设置功能所需的信息。定义的三层结构意味着表现逻辑与数据层或物理数据本身之间不能直接进行访问。从这方面来说，表现逻辑不受数据逻辑或数据结构本身变化的影响。只要业务逻辑不改变，那么表现逻辑就不需要改变。

4. 表现逻辑

表现逻辑是用户与应用程序联系的部分。这是应用程序中的一部分，用于生成用户界面，并根据与用户的交互接收用户界面输入。表现逻辑与应用程序的交互通过业务组件接口进行。应用程序的用户将仅能看到表现逻辑所表现的内容，在用户与数据库或业务对象之间不存在任何直接的关系。

应用程序要求此应用程序的表现层是网络应用程序。即使客户端的功能支持 Win32可执行程序，但仍然存在其他的需求。应用程序的需求之一是总支持最新的业务逻辑。只有在当每次应用程序需要业务功能时都从中心位置申请此功能的情况下，这才是可能的。当在 Win32应用程序中使用 DCOM与中央服务器上的组件交互时这是可行的，但另一种应用程序需求使这种方式不可行。

根据对所有需求的分析，可以看出应用程序的最佳结构是一个以网络为基础的应用程序，在那里用户界面通过一系列的动态网页表现出来。这些网页由一系列的 ASP脚本创建，ASP脚本使用COM与业务组件交互。通过结合用户输入与业务组件的信息，每一 ASP页将生成一个或更多的网页。这些 ASP脚本可以分为三类，大致与应用程序的三种基本功能相对应。

形成表现逻辑的ASP脚本的可分为如下三类：

- 新雇员的信息：AddEmployee.asp和AddEmployeeForm.asp。
- 雇员验证：DisplayValidationList.asp、EmployeeTaxInfoForm.asp、SetEmployeeTaxInfo.asp、ValidateEmployee.asp和ValidateEmployeeForm.asp。
- 保险赔偿费选择：ChangeBenefit.asp、DependentList.asp、DisplayBenefits.asp、EditDependent.asp、EmployeeLogin.asp和VerifyEmployee.asp。

现在已经定义了应用程序的三个层，下面研究一下这三个层是如何连在一起的。

5. 集成结构

集成结构将应用程序联接在一起。它详细说明了表现逻辑如何调用业务对象的接口，业务组件如何调用数据对象的接口，数据组件在何处获取所需的信息。一旦建立了集成结构，剩下的工作只是实现。利用应用程序的组件功能和已定义的接口，这个开发过程可以很容易地在多个开发人员中同时开展。

在设计练习中我们没有必要详细说明所有集成方法，但将给出应用程序的一个范例。创建集成结构的方法通常按照以下步骤进行：

- 对于表现逻辑，必须定义需要表现什么信息，或者返回什么信息给用户。
- 对于每一信息，决定需要哪个业务对象和哪个相应的接口提供这些信息，或者将这些数据送回应用程序。
- 利用每个业务逻辑方法，决定所需的物理数据交互，使用数据组件的接口提供此数据。
- 最后，对数据组件对象的每个方法，决定支持这个方法所需的物理数据，并将此方法链接到所需的物理数据上。

例如，setEmployeeTaxInfo.asp页将需要靠人事资源管理人员录入税务信息，并传输给相关的政府部门。执行这种功能的业务逻辑是 EmpValidation对象的SetEmpTaxInfo方法。此方法需要将由用户录入的税务信息转换为合适的格式，在本例中使用 XML格式。然后使用一个数据对象实际传输信息。传输此信息的方法是 TaxInfo对象的sendTaxData方法，这个方法获得提交的信息并打开一个连接到数据目标的物理连接以传送数据。在本例中，数据目标是一个电子邮件信箱，因此物理连接经过 SMTP服务器。如图13-5所示。

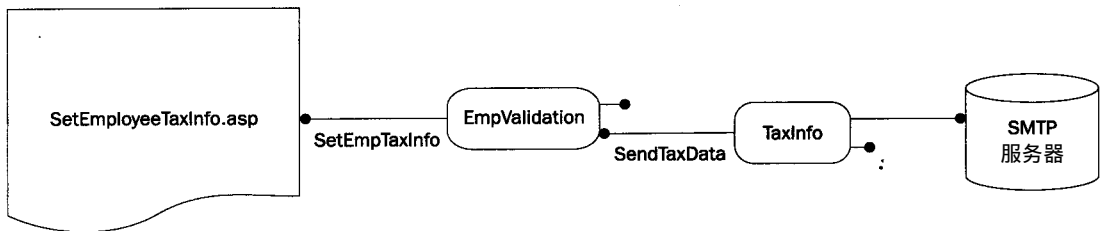


图13-5 传输的信息的过程

通过系统中的每个表现逻辑组件推进创建集成结构的方法。当越来越多的组件集成被定义，这种过程变成了已定义的组件的连接。一旦此步骤完成后，增加一个现有业务对象支持的新的表现功能就得非常简单。

13.4.3 设计权衡

当建立了应用程序的结构后，必须考虑设计对应用程序其他方面的影响。这些被认为是

设计的权衡。设计的权衡通常涉及在两个可能的设计方向之间做一个选择。设计权衡的因素包括应用程序功能的效果、开发应用程序的时间、应用程序的适用范围和应用程序增强和扩展的难易程度。

在创建上述的基于网络的人力资源应用程序时，通常考虑两个方面。由于整个公司都将使用这个应用程序，因此这个应用程序要有尽可能高的可扩展性，即应该使这个应用程序能够同时支持多个用户，并且通过增加额外的服务器处理能力可以提高应用程序的性能。另外一个方面，应该考虑这个应用程序的可扩充性。因为用户肯定会需要额外的功能，必须确保初始设计能够支持这些额外功能。这种支持应该不需要在多方面重复设计应用程序。

下面分析以下应用程序的设计，然后回答两个重要的问题，一个是可扩充性，另一个是可扩展性。

1. 可扩充性

应用程序的可扩充性可以追溯到对象模型的设计。先来看一下是否存在调用现有方法就能实现的其他功能。例如，用必要的业务和数据对象访问和编辑一个已有的雇员信息。这样使用现有的业务对象在应用程序中增加一些新的表现逻辑，就可在应用程序中增加雇员信息编辑功能。

考虑现有的数据对象接口，并寻找由它们支持的新业务功能，就能决定可以快速添加的其他应用程序功能。例如，数据对象接口已支持检索雇员保险赔偿费信息所需的接口。新的业务对象能够根据雇员选择的保险赔偿费，协助公司的健康申请负责人确认雇员的健康卡申请，雇员通过这个数据接口选择保险赔偿费。

面对新的应用程序需求时，应用程序设计者首先应查看构成应用程序的已有对象。与其立即着手创建新的对象，不如先搞清楚已有的对象。考虑这些对象时，可以寻找所需的接口来支持所需的业务功能。如果确实不存在，也有可能找到相似的接口，这些接口可以扩展和增强。只有查看了所有接口而没有找到时，才有必要设计一个新的接口。

2. 可扩展性

应用程序设计的另一个问题是，它是否可以扩展。本章前面讨论了网络组件的设计。利用这些设计原则，可以使应用程序扩展而支持更多的用户。组件应用程序的可扩展性与组件实现的细节有关，也和组件的无状态性有关。

为了创建能够由COM+缓冲的组件，必须确保应用程序中使用的组件是用 Visual C++ 编写的。这就造成了权衡的问题，因为用 Visual C++ 创建组件比用 Visual Basic 要耗费更长时间，并需要不同的开发技巧。我们必须确定为了获得更高的可扩展性，用 VC++ 花更多的时间和资金来开发组件是值得的。

应用程序的可扩展性也受到应用程序中组件是否具有状态的影响。如果组件设计为有较长的生存期，那么系统必须创建大量的组件来支持越来越多用户。如果组件的生存期很短并很快被丢弃，那么系统在任一时间都只需很少的组件。由于所有组件配备了所需信息来执行包含参数的每个方法，并且在方法调用之间，不需保留任何信息，我们就创建了无状态组件。这样提高了应用程序的可扩展性。

13.4.4 设计小结

我们已经介绍了用于人力资源管理的应用程序的设计方法。

- 应用程序应该成为基于组件的三层网络应用程序。
- 它应该使用ASP脚本生成的基于浏览器的表现形式。
- ASP脚本将与八个业务逻辑组件交互，这些组件支持雇员信息、保险赔偿费信息和税务信息的处理。
- 为了使这些业务组件与执行业务功能所需的数据能够交互，需要三个数据组件。
- 只有这些数据组件才能与实际的物理数据源进行交互，这些物理数据源可以是 SQL数据库或是SMTP邮件服务器。

整个应用程序使用组件创建，能支持未来的扩充，在不需要重新设计的情况下，能够支持更多的用户。

13.5 小结

在许多企业中网络应用程序正迅速发展成为业务应用程序的范型，网络应用程序正在代替客户/服务器或终端式应用程序。网络应用程序容易建立、稳固并且具有可扩展性，可同时支持大量的用户。Windows DNA对Windows平台的这些类型的应用程序来说是一种新的应用程序结构。

本章内容概括如下：

- 如何用Windows DNA建立可扩展的、灵活的应用程序。
- 网络应用程序的组成和如何进行设计。
- 组件和微软组件服务如何使建立网络应用程序更为简单。
- 如何使用网络应用程序结构的各个部分建立网络应用程序。
- 组件的三种类型和使用的原因。
- 一个基于组件的应用程序设计范例。

下一章介绍关于组件的新知识，学习如何用 Visual Basic建立Windows DNA组件。第15章将介绍如何将其放到COM+应用程序中，并进行测试以确保能够正常工作。