HZ BOOKS

Designed for Microsoft® Windows NT® Windows® 98

附 赠 CD-ROM

Designing for Scalability with Microsoft Windows DNA

微软公司核心技术书库

(瑞典) Sten Sundblad Per Sundblad 著 前导工作室 译

Windows DNA 可扩展设计

机械工业出版社 China Machine Press

Microsoft Press

微软公司核心技术书库

Windows DNA 可扩展设计

Sten Sundblad
(瑞典) Per Sundblad 著
前导工作室 译

机械工业出版社
China Machine Press

本书全面深入地介绍了微软的新一代体系结构技术——DNA，通过实例说明如何利用Windows DNA进行可扩展程序设计。书中以Visual Basic作为前台开发工具，以SQL Server作为后台数据库，详细地说明了示例程序——赛马应用程序的主要开发过程。还介绍了DNA中XML的应用。作者结合自己多年的实践经验，总结了使用微软 COM+技术的一般性原则，以及升级到COM+的必要性、过程和注意事项。本书附带光盘包括示例程序的代码，以及DNA XML资源工具包。

本书版权登记号：图字：01-2000-1516

图书在版编目(CIP)数据

Windows DNA 可扩展设计 /（美）桑得布莱德（Sundblad, S.），（美）桑得布莱德（Sundblad, P.）著；前导工作室译. - 北京：机械工业出版社，2001.1

（微软公司核心技术书库）

书名原文：Designing for Scalability with Microsoft Windows DNA

ISBN 7-111-08464-0

I. W… II. ①桑… ②桑… ③前… III. 窗口软件，Windows - 程序设计 IV. TP316.7

中国版本图书馆CIP数据核字(2000)第54496号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）

责任编辑：丁凌峰 吴 怡

北京昌平第二印刷厂印刷 · 新华书店北京发行所发行

2001年1月第1版第1次印刷

787mm × 1092mm 1/16 · 21.5印张

印数：0 001-5 000册

定价：49.00元（附光盘）

译 者 序

软件重用一直是软件技术的发展方向，以COM和CORBA为代表的组件技术给软件业的发展带来了革命性的突破，使得三层乃至多层的软件体系结构成为可能。Microsoft更是在原有COM和MTS的基础之上推出了新的COM+技术，Windows DNA就是基于该技术的应用程序体系结构。为了使读者能更好地利用新的平台，我们以最快的速度翻译了此书，希望对读者有所帮助。

书中作者以实际应用（赛马比赛）为背景，循序渐进地讲述了如何利用新的Windows DNA技术开发具有良好的可扩展性的应用程序。在具体的步骤过程中，还对可能采用的多种设计和实现方法进行了比较。由于作者多年从事这方面的研究，所以他们提出的建议很有参考价值，对实际工作有很强的指导意义。

本书主题明确，脉络清晰，读者很容易就能发现并掌握自己所需的信息。与此同时，书中还对各种相关的产品和技术做了一定的介绍，如XML、Rational Rose等，使读者不必频繁查阅其他书籍就能较好地理解书中的内容。

本书由前导工作室的虞万荣、刘锋组织进行翻译，前导工作室的全体工作人员参加了本书的翻译、审校、录入和排版工作。

由于译者水平有限，加上时间仓促，错误之处在所难免，恳请广大读者批评指正。

2000 年6月

前　言

1998年3月，我们向客户做了名为《En skalbar arkitektur》的技术报告。这是用我们的母语——瑞典语写的。该报告提供一种思想，这种思想用于设计和发展可扩展的Windows应用程序（使用诸如Microsoft Visual Basic 5.0和Microsoft Transaction Server(MTS)的工具）和数据库管理系统（使用诸如Microsoft SQL Server或Oracle的工具）。

在完善这个报告的同时，我们得到在Redmond的微软的朋友们的大力支持。以下是他们提供帮助的一些方面：

- 微软已经出版了一些关于可扩展Windows应用程序的白皮书。从全世界的开发者对这些文章的反馈中可以知道，在世界范围内，有许多人对开发模式有兴趣。为了对重复编程的问题提出解决方案，为了MTS的发展，这激励着我们花时间去写更深入的报告以及支持软件。
- Scott Swanson（当时在Visual Basic组中），几乎安排了我们和每一个程序管理者的会面，这些程序管理者在设计即将发布的Visual Basic 6.0和ActiveX Data Objects(ADO)2.0。这些人提供了一些重要而先进的有关最新版本的ADO的信息，强烈建议远离Remote Data Objects(RDO)。我们采纳了他们的建议并修改了设计模式、报告和所有的代码。
- James Utzschneider，在对Microsoft Transaction Server负责的时候，赞扬了我们的设计思想，并说（以我们的理解）该思想恰好符合他的尽量利用Transaction Server的理想化设计意见。在1998年2月在洛杉矶举行的SQL Server 7.0工作会议上，他也给我们在几百名参加者面前展示我们的设计的机会。当时，他的报告名为《Building Windows DNA Application for SQL Server 7.0》（建立为SQL Server 7.0服务的Windows DNA 应用程序）。在工作会议的末尾，James和参与者的反馈意见使我们感觉到应该翻译这篇报告，于是我们便进行了翻译。

事实上，是James第一个提醒我们，微软出版社可能对我们的材料感兴趣。在那时，我们感到在微软出版社出版本书的准备还不够充分。但这个想法留在了我们的脑海里。

现在，《En skalbar arkitektur》和它的英文版《A Scalable Architecture》已经出版两年多了。许多基于本书中所验证的思想的应用程序正大量出现，或者自从我们发表了第一个报告以来就已经出现了。正如期望的一样，其中有许多电子商务的应用程序。没有预料到的是，一家瑞典报纸用我们的设计模式来开发网站。一些开发者使用了这个设计模式和附带的向导（Wizard），而另一些则修改了我们的思想使之符合他们自己的战略和代码标准。这正是我们所期望的。

1999年仲夏，我们出版了第二版的技术报告。自从第一版出版以来，我们学到了许多东西，应该和读者们分享。有许多新的工具，或者说在第一版技术报告中使用的工具的新版本出现了。其中，几个重要的软件升级了，例如出现了Visual Basic 6.0、SQL Server 7.0和Rational Rose 98i和2000。自然，应该充分利用这些工具。它们在我们的第二版技术报告，即Scalable Visual

Basic and MTS Applications中是重要的成分。

我们想使用新的标题以显示1999年的报告并不是第一版报告的新版本，而完全是新版本。1999年的报告的设计模式比前一版高级许多。此外，例子应用程序的设计方式不同。首先创建用例模型，然后为之工作，这要优于仅仅分离技术设计细节的方法。这看起来是值得去做的，首先，因为这是一个很好的接近设计任务的方式；其次，因为许多开发者能从这种设计的过程中得到帮助。

我们同样要突出在第二版报告中的业务规则（business rule）的概念，给它一个切实的质量保证。随着n层应用程序环境的出现，我们发现许多开发者都在考虑把业务规则放在哪里最有效。中间层经常称为业务规则层，一些使模型更加容易理解的东西可能会导致混淆概念。毫无疑问，许多业务规则应该在中间层实现，但还有一些最好安排在数据库层。我们希望通过在第二版的报告中对这些问题进行深入讨论，从而把这一点说清楚。

本书的产生过程

当1999年的报告发表时，COM+只不过是个许诺。XML虽然已经出现，但是帮助开发者创建基于XML的解决方案的工具非常少。

我们在那时的决定是延迟对COM+和XML的讨论，到时机成熟时再讨论。当微软出版社和我们签定本书合同时，这个时机出现了，因此本书增加了COM+和XML的内容。

问题是，应该继续支持老的MTS和ADO环境吗？或者是应该完全使用COM+和XML这样的新鲜事物吗？在和微软出版社讨论后，得到如下决定：

- 当本书出现在书架上时，和COM+一致并且对XML开发者有更好的支持的Windows 2000将可能完全可用。新环境的早期采纳者肯定需要体系结构的指引，那么本书绝对应该全力帮助他们。
- 然而，并不是每一个开发者会成为Windows 2000的采纳者。许多开发者可能在一段时间内在Windows NT4.0和MTS平台上设计应用程序，即使新的开发系统和新的操作系统在市场上已经可用也会有这种现象。这些开发者同样需要体系结构的指引。在某种程度上，他们的需要更多。COM+已经简化了在MTS中需要的东西，而且，他们的应用程序要在MTS环境中工作顺利，并且设计为更容易移植到COM+中，使其能利用在MTS和Windows NT 4.0中没有的新鲜事物。

因此，我们把精力主要集中在Windows 2000和COM+的开发上，但又不遗弃在Windows NT 4.0和MTS平台上的开发者。我们给出使用无连接的ADO记录集的数据传送的例子，也给 出使用XML的例子。所以我们应该给Windows 2000体系结构指南的采纳者提供更早的信息，

MTS与COM+

微软发展COM+技术是首先通过分别改善COM和MTS，而后把改善后的COM版本和MTS合并。COM+不只是一个更好的COM和一个更好的MTS的组合，这是开发COM+的初衷。

本书不是MTS或COM+的教学资料

本书绝对不是MTS或COM+的教学资料。有许多书是那样的，本书中我们将介绍其中的一些。

本书的目标是三个字母的组合词NAG：Need Architecture Guidance（需要体系结构化的指南）。如果本书还包括一些怎样为COM+和MTS编码的例子（这我们认为是有的），那就应该把它们视为额外收获。

虽然本书不是教学资料，但我们将详细介绍示例程序的一部分设计和实现过程。当一步一步地走过设计和实现过程，开发者将看到我们所做的决定（这基于在那个阶段获得的信息），以及它们是怎样影响可扩展性的。我们希望开发者将发现这种方式是现实的和有帮助的，因为我们的步骤是那些现实软件项目的缩影。我们希望开发者喜欢本书，我们同样希望本书在设计可扩展的Windows DNA方案时为开发者指点迷津。

英文原书书号为：ISBN-0-7356-0968-3

目　录

第1章　设计新的体系结构

Windows DNA是微软为分布式应用程序的开发所提供的平台。图1-1显示了Windows DNA应用程序体系结构的概览图。我们能用Rational Rose 2000生成这种概览图。在图1-1中看起来像文件夹的符号叫包（package），它们是UML（Unified Modeling Language，统一建模语言）包，而不是Microsoft Transaction Server（MTS）包。MTS包是组件的容器。开发者将自己编译的MTS组件安装在MTS包中。所以MTS包是计算机中的物理事物，而UML包是UML类模型中的一个逻辑容器，它能帮助区分模型中的类（注意，在Microsoft Windows 2000中，术语包（packet）变为应用程序（application），而MTS被重命名为组件服务（Component Service））。

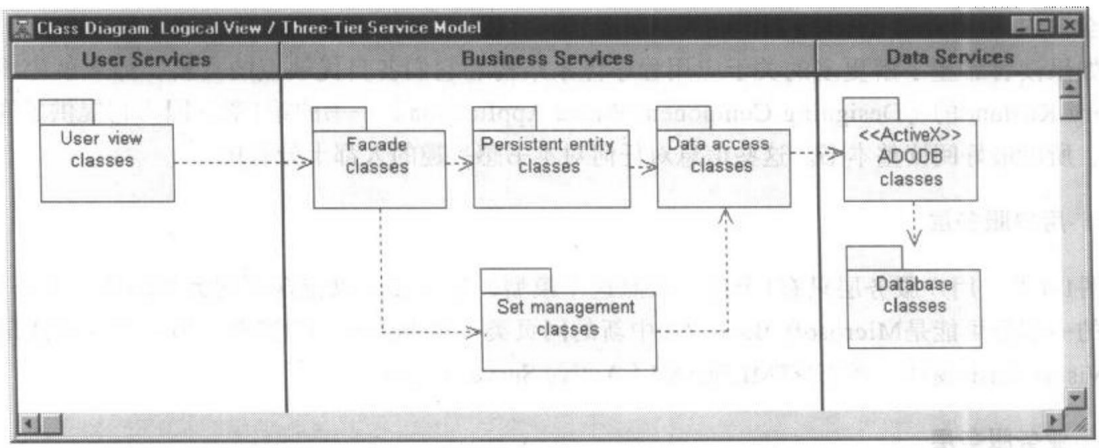

图1-1　可扩展的Windows DNA设计的起点

在UML包之间的虚线箭头代表包之间的依赖关系。图1-1说明User Views包中的对象在功能上依赖于Use Case Facades包中的对象。换句话说，前一个包中的对象会调用后一个包中的对象。从图1-1中还可以看到，在这样高的抽象水平之上，该体系结构中的对象是怎样定向的。模型包括了由Set management classes和另一个Persistent entity classes组成的UML包（这些包将在本章稍后的1.1.2节“业务服务层”中阐述）。图1-1显示的体系结构是任何面向对象的纯论化者都会遇到的一个好例子。关于更高的可扩展性设计的例子将在本书后面进行阐述。

下一节解释图1-1，我们将采用循序渐进的方式给出这个十分高效的并且更具可扩展性的体系结构的完整描述。这个体系结构是适合于Microsoft Windows NT和MTS或者Microsoft Windows 2000和COM+业务应用程序的。

1.1　可随意选择三层或五层

两条垂直线把图1-1的模型分为众所周知的三层：

- 用户服务层。
- 业务服务层。
- 数据服务层。

使用三层模型而不使用以前的二层模型，也就是通常说的客户/服务器模型，将会让开发者获得更大的益处。在二层模型中，客户持续地连接到数据库服务器，这将消耗诸如服务器内存和处理器处理时间等资源。考虑如下事实，用户通常花费许多时间处理本地数据而不是读取或更新数据库中的数据，那么二层模型将是十分浪费的，不应该允许客户在不需要数据时还占有资源。本书的很大部分，包括许多章节，阐述节约资源和怎样获得资源的内容。在三层模型中，许多工作变得比客户/服务器模型中更加容易。

在第3章“规则及其位置”中，我们将讨论到三层模型的另一个益处。简而言之，这个模型允许程序开发者使用自己喜欢的语言去编写业务规则的程序，而无需将这些规则伸展到大量的客户工作站。在三层模型中，将自己的业务规则放到中间的应用程序服务器上，就能很容易地维护和增强它们。而且，在许多应用程序中可重用同样的业务规则对象。这就是Internet应用程序的全部内容。

如果读者希望了解更多的关于应用程序体系结构和它们各自优缺点的资料，那么就应该阅读Mary Kirtland的《Designing Component Based Application》一书的第1章。因为它提供了许多信息，所以最好阅读整本书。这些信息对任何对本书感兴趣的人都十分有用。

1.1.1 用户服务层

图1-1中，用户服务层只有1个包。依照这个模型，这个包可以包含不同类型的用户服务类。其中的一部分可能是Microsoft Basic 6.0中新的网页类（Webclass）的实例，另一部分可能是普通的Visual Basic窗体、动态HTML或ASP（Active Server Pages）。

1.1.2 业务服务层

业务服务层包含四个包：

- 第一个是外观（facade）类。这个类是用户界面需要与之通信的“真实”业务对象前面的外观类。在第5章“对第一个业务层进行模型化”中将说明为什么外观类是重要的，及它们如何提供支持。
- 第二个是持久实体（persistent entity）类。它包含传统的面向对象的业务类。属于这个类的对象在允许自己变为无效之前必须先把数据存入数据库中。顾客、产品和订货单是这种类型的类的典型例子。
- 第三个是集管理（set management）类。它们对面向集的类负责，可以返回记录值，甚至可以处理面向集的更新。它们也同样可以状态化，处理持久实体类的集合，即使在很多情况下这只是可选择的。毕竟，状态化的对象并不能像无状态对象那样扩展。第8章将会讨论关于状态化对象和无状态对象的问题，其标题是“设计主业务层”。在本书的后面，将把第二个和第三个包结合成一个。我们将以其他方式修改它们，以增强其可扩

展性。

- 第四个是数据访问（data access）类。这个类建立SQL语句并存储过程调用。这个类单独与数据库通信（通过ADO或者是多维版本的ADO，即ADO MD (Multidimensional)）。这个包同样管理业务。除了数据访问类，通常不应该允许其他类支持或请求事务。在其他包中的类应该远离这项任务。

1.1.3 数据服务层

第三层是数据服务层，在图1-1所示的模型中，这一层包括两个包：

- 一个是ADODB包。它包含所有在ADO中的类。ADO是微软的战略性通用型数据库。
- 数据库应该具有低层无关性，能通过OLE DB或ODBC进行访问。本书的例子使用的是Microsoft SQL Server，但若使用Oracle、Informix或DB2数据库，它们也同样能工作得很好。

本书同样尝试着在ADO MD，以及由此而来的在线分析处理（Online analytical processing）服务器中使用SQL Server 7.0提供的数据服务。

开发者可能对访问其他类型的数据服务感兴趣，诸如Lotus Notes和Microsoft Exchange Server等。

1.1.4 五层而不是三层

许多开发者更喜欢把业务服务层分为三个子层，从而创建了五层体系结构：

- 三个新层中的第一个是外观层（facade tier），如果喜欢的话，可称为应用程序层或工作流层。
- 第二个是主业务层。
- 第三层是数据访问层。

一个可供选择的办法是，使用如下所示的五层：

- 用户服务层。
- 业务服务层。
  - 外观服务层。
  - 主业务服务层。
  - 数据访问服务层。
- 数据服务层。

把业务处理层分为外观层、主业务层和数据访问层的明显优点在于：外观层将更接近用户应用程序而不是业务逻辑，而主业务层包含了实际的业务逻辑。因此，应该依照特定用户的应用程序需求来设计外观层，从而能在这个应用程序的需求随着时间的流逝而发生改变时也能修改外观层的类。

因为主业务层的类实现业务逻辑，而不是任何特定的应用程序功能，所以它应该设计成更具有重用性。因此，它们应该很健壮，不能像外观层中的类一样允许自由地修改。否则，可能导致其他应用程序不能执行。

因此，外观层把用户界面对象从主业务层的复杂逻辑中分离出来。外观层中的类可能使用用户应用程序中的词汇，并且其程序接口可能会比主业务层中的类更加简单，这样就获得了更多的修改类的自由，以及易于开发、易于维护和随着时间的流逝而带来的系统强化。第8章将讨论这些问题。

同样地，数据访问层把主业务层从数据库的复杂逻辑中分离出来，此外，如果希望有效地使用COM+和MTS的业务服务，那么把数据访问从主业务对象中分离出来正是实际上需要的。

基于同样的原因，应该使用不同的类从数据库中取数据、添加或修改信息。如果由COM+和MTS来管理事务，那么把组件中的事务部分从非事务部分中分离出来是合理的。图1-2给出了数据访问类的包的一个例子，其中包含Customer持久实体类中的一个获取类（fetch class）和一个修改类（modifier class）。

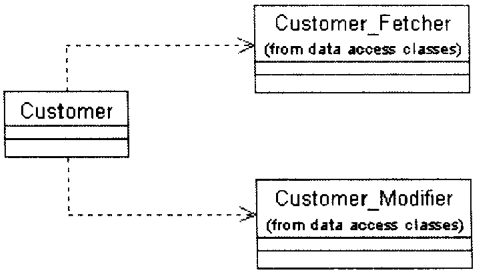

图1-2 依赖于相关的Customer_Fetcher和Customer_Modifier对象的Customer对象

在第12章“委托数据访问”中将给出适合于该解释的更多细节问题。

分区的对象

注意在图1-2中的数据访问类是持久实体类（persistent entity class）Customer划分的结果。在面向对象的设计中（而不是面向组件的设计），Customer类通常不应该以这种方式划分：在这样的设计中，可能不会存在获取类和修改类。

基于本书后面将会阐述的几个原因，这里我们如下所示水平区分持久实体类：

- 初始类——在本例中为Customer类。
- 用于直接操作而不是数据存储的单独类。

于是把数据操作分为两个单独的类：

- 一个类执行获取操作。也就是说，这个操作并不参与事务（例如：Customer_Fetcher）。
- 一个类执行对数据库的修改操作。也就是说，参与事务的操作（例如：Customer_Modifier）。

1.2 使用ADO记录集

当扩展应用程序使之符合用户需求时，通常需要把应用程序的各个部分分布到一定数量的计算机中去。这里有许多配置选项。在图1-3中的策略是其中之一，并且该策略在许多方面是合理的。依照这个策略，配置这个应用程序的用户已经选择将两个数据服务对象尽可能放在离数据库较近的地方。对象驻留在运行数据库管理系统的物理服务器上。

顾客的持久实体类已经找到了区分应用程序或业务服务器的方法。而网页类和外观类则运行在作为Internet服务器的同一服务器上，这个服务器当然是Microsoft Internet Information Server（IIS）。

1.2.1 在服务器之间传送数据

当开发者像图1-3所展示的方式那样配置分布式应用程序时，它的数据必须在服务器间传送。从数据库中取回需要的数据之后，Customer_Fetcher对象必须把它返回到用DCOM调用，请求数据的应用程序服务方。

此时数据必须移动到外观对象。后者是驻留在Internet服务器上的。这个外观对象和网页类对象紧密合作，在HTML页面中显示数据集中的信息。

如果用户要改变任何从服务器接收的数据，那么新版本的数据必须经过所有返回数据库服务器的路径。简而言之，在分布式的应用程序中，数据持续地在分布式的组件之间以及它们所在的机器之间流动。因此需要找到高效的、容易管理的和标准化的方式来移动数据。

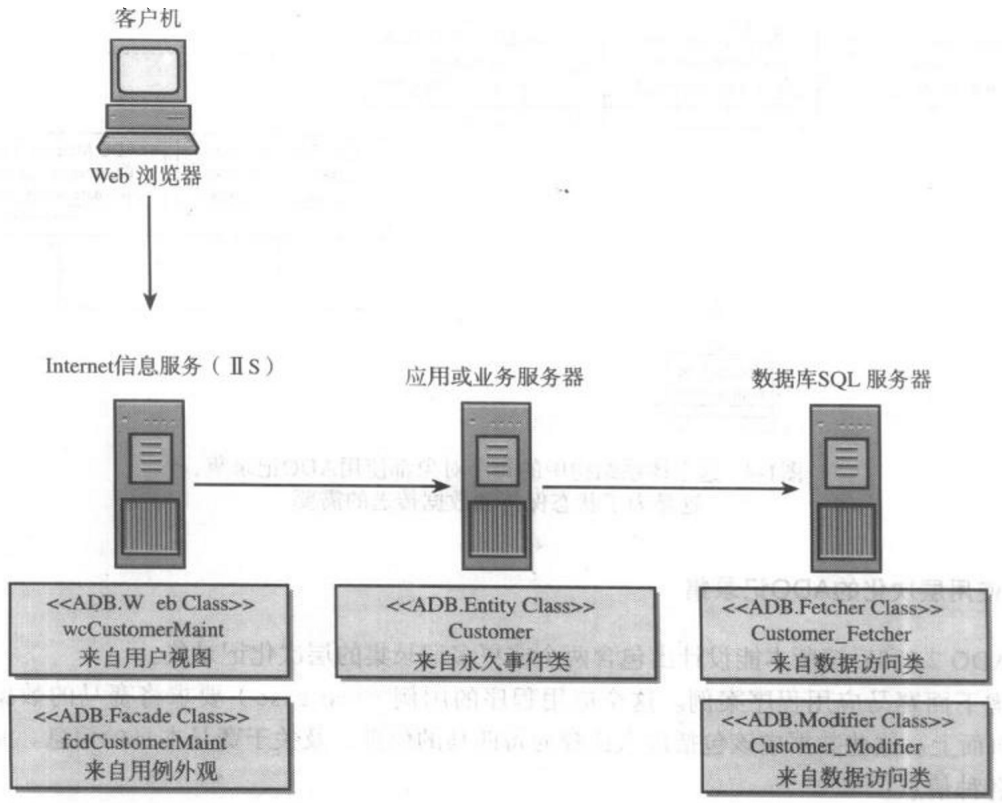

图1-3 部署在四台单独的计算机上的分布式应用程序

为断开的ADO记录集传送数据

不像其他COM对象，ADO记录集能在进程和机器之间移动。这意味着数据的确在移动，并不只是发送一个对它们的引用。

传统的COM对象不能像这样移动。而是发送一个指向自己的引用给其他机器。每当那台机器上的客户软件发送消息时，这个消息会通过网络传播到对象的所在地，对象的所在地应是绝不改变的。

但这与ADO记录集并不相同。如果遵循一些简单的规则，ADO记录集将使用调度方式来移动。这不仅移动数据也同时移动元数据。这些数据在进程之间，甚至是计算机之间移动。这里有一些规则：

- ADO记录集必须与数据库断开。可以通过将记录集的ActiveConnection属性设为Nothing来实现。
- ADO记录集必须使用客户方的游标库。可通过将记录集的CursorLocation属性设为adUseClient来指定客户方游标。否则，不能把记录集和数据库断开。如果记录集保持连接，它肯定使用服务方游标，那么记录集拒绝在体系结构中的机器之间移动。

图1-4说明每个对象是怎样利用这些记录集功能的，每个对象都使用ADO记录集，这不仅是为了数据传输的需要，也是为了状态保持的需要。

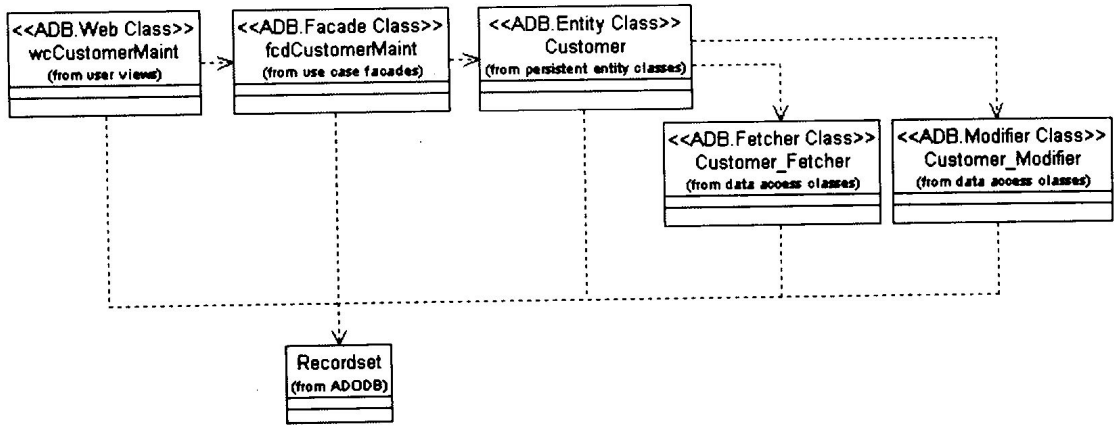

图1-4 这个体系结构中的每个对象都使用ADO记录集，这是为了状态保持和数据传送的需要

1.2.2 使用层次化的ADO记录集

用ADO 2.0和后续版本能设计出包含两个或更多记录集的层次化记录集。

考虑下面赛马应用程序案例。这个应用程序的用例（use case）要求将赛马的数据显示在HTML页面上。这些数据应该包括进入比赛的每匹马的信息，及关于赛马本身的信息。必须显示下面的各种信息：

Races

Id	TrackId	Date	Race	Distance	TrackType	RaceClass	Prize
11	Täb	1998-01-11	1	1600	DT	Åv	34000
12	Täb	1998-01-11	2	1350	DT	Åv	24000
13	Täb	1998-01-11	3	1200	DT	Åv	18000
14	Täb	1998-01-11	4	2000	DT	Hkp50	13000
15	Täb	1998-01-11	5	2400	DT	Hkp65	34000
16	Täb	1998-01-11	6	1600	DT	Hkp 2	40000
17	Täb	1998-01-11	7	1600	DT	AmHkp55	14000
18	Täb	1998-01-11	8	1350	DT	Åv	23000