

## 第15章 COM+应用程序

当我们伴随着Windows 2000和ASP 3.0进入21世纪时，发现微软在企业应用程序开发方面扩展和改进了对开发人员来说非常有用的工具和技术。在前面两章中讨论了如何使用 ASP和COM构建Web应用程序。通过使用功能强大的 IIS，使用所选择的程序语言开发 COM组件，使Web应用程序远远超越仅使用 ASP脚本创建互相链接的简单网页的水平。可以开发安全高效的基于组件的可扩展应用程序，以此来支持面向数以千计用户的业务处理。

Windows 2000中显著的变化是微软的事务服务器 (Microsoft Transaction Server，MTS)得以改善并融入COM运行期中，当然，除此以外还有许多其他新的特性和改进。每一项改进都可以使我们的应用程序比以前更为强大和可扩展。

使用COM+可从组件开发COM+应用程序，以提供基础服务和代码，而不是从头进行开发。当开发基于 DNA的多层应用程序时，这些服务给我们提供了起点。这些应用程序能发挥COM+所提供的服务与基础结构的优势，这些都是企业级应用程序所需要的，如事务处理、组件管理、安全和对象缓冲等。每一项服务对于完成任务都是举足轻重的，如果自己开发需要投入很大的精力。还好现在都由 Windows 2000提供。我们所要做的只是使用 Component Services Explorer来定义如何使用而已。

组件服务是微软对COM、COM+以及像微软消息队列 (Microsoft Message Queue，MSMQ)这样的相关技术的概括性名称。

本章内容包括：

- COM+ 如何为开发稳固且可扩展的基于组件的应用程序提供构件。
- 如何创建和管理应用程序。
- 如何用 Visual Basic创建用于COM+环境的组件。
- 线程和组件作用域如何影响网站的可扩展性。
- 如何调试COM+组件。

下面先从COM+概述开始，了解COM+作为Windows 2000完整的一个组成部分，对于基于组件的应用程序开发人员多么有用，并进一步研究 COM+构造的基本原理。

### 15.1 微软组件服务(COM+)

第13章介绍了微软组件服务 (Microsoft Component Services，MCS)和COM+作为配置Windows DNA的工具，研究了COM+作为一个“对象管理者”的功能。作为一个对象管理者，COM+在整个生存期负责管理COM+对象，这就意味着COM+实际上创建了对对象，管理应用程序对组件的使用，然后当其不再使用时就取消这些对象。

使用像COM+这样的“对象管理员”来管理对象的生存期，而不是让每个应用程序去单独管理，通过仔细监视和操纵应用程序和COM+对象之间的基本交互，能够更加有效地使用系统资源，为了完成这个目的，COM+需要以某种方式与应用程序/对象相关联。

### 15.1.1 拦截原理

COM+能够在应用程序和COM对象(组件)之间插入代码。在本章你将看到,这种拦截(interception)是如何通过COM+实现的,但现在把这个过程想象为一种COM+“粘性”代码,只要COM对象的方法或属性在应用程序中使用,就调用这段代码,完成一些初始的设置过程,然后调用真正的对象方法。

为了解拦截所能提供的益处和功能,先考虑一个典型的Web应用程序:有多个用户每天以较高的频率访问同一个ASP页面,基于组件的Web应用程序中每个ASP页将包含完成下面三个基本步骤的ASP脚本。

- 创建一个或多个COM对象。
- 使用COM对象生成页面。
- 取消这些COM对象。

假如一天之内收到50000个关于该页面的请求,如果该页面创建两个COM对象,那就需要创建和取消100000个COM对象。考虑到创建和取消COM对象需要的计算机资源的总开销,相比之下循环使用COM对象效率更高。当ASP完成调用COM对象以后,将其放入一个缓冲池内,并可从缓冲池中取出,由其他的ASP页面重复使用。我们将看到这会明显提高Web站点的性能。如果所使用的对象在每次创建使用之前(例如连接一个数据库或载入值),进行合理的初始化工作,可使Web站点的潜在性能进一步提高。使用缓冲,可以仅进行一次初始化工作。依据这种循环的思想,理想的ASP页面中的理想的ASP脚本应该依据下面的基本步骤:

- 在建立COM对象前,查看缓冲池内是否存在该对象可供重复使用,如果没有,则创建一个新的对象。
- 使用COM对象来生成一个页面。
- 将COM对象返回缓冲池,以供其他页面使用。

这个循环过程听起来是很理想。当然实际上,这不可能在ASP脚本中实现。最简单的原因是不能直接访问COM运行期(runtime),正如在第14章中讨论过的,活动脚本(Active Scripting)引擎调用运行期。即使能使用COM运行期,ASP脚本本身的限制和活动脚本引擎实际上也阻止这种缓冲方式。暂且把这些限制放在一边,假如能够完成这种缓冲,可以想象,在实现时下面的这些复杂问题是不得不考虑的:

- 谁来管理对象的缓冲池?
- 缓冲池何时创建和取消?
- 对象在缓冲池中何时创造和取消?
- 在什么阶段取消对象?
- 如何管理多个用户同时访问缓冲池?
- ASP脚本与缓冲池如何交互?程序员编写脚本时,能够记住在创建一个新对象之前必须检查缓冲池,用完对象之后将一个COM对象返回到缓冲池吗?

显然在ASP脚本中实现这种功能是不可能的,即使能实现这种功能,在实现过程中,也存在着两个非常重要的障碍需要解决:

- 必须自己实现对象缓冲池的代码,要充分考虑到代码、测试、调试和维护周期。
- 使用缓冲池的ASP页面代码必须要正确使用对象缓冲池,而让程序员考虑得太多是一种错误的倾向。程序员可能经常忘记使用缓冲池。

COM+减轻了这些负担，提供了一个一般性的基础结构来支持许多不同类型的预写服务，使组件可透明使用。不用更改一条代码就可使用它们中的一部分。

### 1. COM+的补救措施

COM+是通过下面的机制来弥补这两点的不足。

#### (1) COM+提供服务

首先，COM+提供一系列的服务，例如对象缓冲池，可在COM+应用程序中使用，不用编写任何代码就可实现这些服务，这意味着不需要调试和检测，也不用担心前面讨论的任何其他问题。简单地配置组件，通过Component Services Explorer就可使用请求的服务。我们把那些使用COM+服务的组件称为“配置的组件”，通过Component Services Explorer构建的COM+应用程序就是使用这种类型的组件。

#### (2) COM+提供了组件/服务之间的透明交互

第二，COM+提供一个基础结构，这意味可以在不改变任何代码的情况下自动地利用这些服务的优点，因此ASP脚本和其他语言(如VB)，能够屏蔽COM+运行期，可以在代码没有重大改变的情况下使用这些服务。ASP脚本或组件使用这些服务不需要重新编译或修改，在COM+运行期中也不需要调用低层次的C/C++ API函数。不像COM，大多数COM+都是通过COM接口使用的。

一些COM+服务对编程增加了一些规则，需要稍稍修改代码，才能使组件充分使用它们。正如讨论过的，一些服务(如对象缓冲池)当前在一些编程语言中是不可用的，这主要是由于这些语言与COM+服务的额外要求之间存在着内在的不兼容性问题，然而，目前微软正在改进这些服务使之能够用于其他的语言。

本章讨论的是COM+ 1.0。COM+ 2.0许诺增加更多的服务。

### 2. 对象缓冲池

COM+会管理所有的代码，根据请求实现一个服务(如缓冲池)。这意味着代码仅需如通常一样创建和释放一个COM对象；在后台，COM+将使用拦截来实现缓冲池。COM+确保在创建一个新的对象之前检测缓冲池，试图使用缓冲池中存在的对象；不再使用时，把对象放到缓冲池中，实现缓冲池的代码不必插入应用程序中，因为在组件中插入服务是透明完成的。

#### VB没有缓冲池

COM+提供了一个强大的通用框架，可提高Web应用程序性能，而不必改变任何代码。通过Component Services Explorer来选择组件所使用的服务，不需要重新编译这些组件。若管理员要增加组件安全性，只让特定的用户能够调用特定的方法，仅需使用组件服务，不需改变代码。不必担心所有可能使用此组件的应用程序的安全要求。我们把话题转回到COM+。

在幕后，COM+运行期确保已经为组件配置好的服务通过拦截被加入及使用。

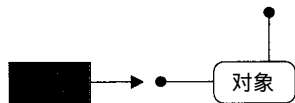
我们已经指出在COM+中“拦截”这个词是非常重要的。

不好的方面是，不是所有的服务能被那些用VB编写的组件所使用。大部分能，但缓冲池是一种不能使用的服务。原因将在后面讲述，但假如想使用特殊的服务，必须用C++编写自己的组件(在17章中讨论)。已经有大量的服务可供选择，如事务支持 and 安全性。但为了充分利用这些服务，应该理解我们所讨论的相关内容，如无状态组件。我们将在第19章中研究COM+在事务管理中充当的角色。

因此, 让我们先研究COM+是怎样工作的。

### 15.1.2 COM+结构

第14章讨论了COM的两个组成部分, 组件和接口。我们认为组件是个黑盒子, 通过一个或更多的接口为客户提供功能, 客户不必了解组件功能是如何编写的, 但它们在使用组件所提供的功能时, 需要知道所依赖的接口是一种协议。在两者之间最基本的交互如图 15-1所示。



客户使用组件的接口来完成一些任务。

图15-1 用户与组件的关系

图15-1中没有显示任何接口名, 这是一个适用于任何客户 /对象对的通用框图。

#### 1. 拦截

COM+为了能将服务透明地加到一个对象上, 将自己放入客户与组件实例之间, 例如为了实现缓冲池, COM+服务必须拦截对 IUnknown 的 Release 方法的调用。如在 14 章中讨论的, 每一个COM对象保存一个引用计数, 以便每个对象能够跟踪有多少个客户正在使用它。当引用计数为零时, 对象负责取消自己。很明显当我们想让别的客户重复使用此对象时, 由对象本身完成这个工作是不合适的, 因此 COM+需要管理这个基本的生存期循环。在理论上, 需要的改变是比较简单的。

- 当对象创建时, COM+对象的缓冲池服务额外增加对组件的引用计数, 表示进行缓冲。
- 监视对被缓冲对象的 Release 方法的调用, 当检测到对象只有一个外部引用时, 可将它放回缓冲池中, 因为其他用户不再使用该对象, 因此它属于这个引用。

实际上, 对一个特定的组件来说实现这些变换是比较复杂的, 但 COM+使之简单化。为完成拦截请求来实现服务, COM+使用了轻量代理(lightweight proxy), 如图 15-2所示。

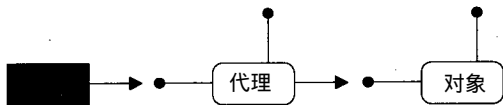


图15-2 用户、代理和组件的关系

之所以称之为轻量代理(也称为拦截器), 是因为其包含了正好足够的代码, 确保粘性代码能够放在客户与真实对象之间, 确保所有配置的组件需要使用的服务在运行期内正常使用。

当客户调用一种方法时, 粘性代码确保方法在实际执行时, 运行期环境是在合适的状态; 当方法完成后, 恢复原来环境。例如, 考虑下面这种情况, 一个对象客户在一个事务里, 但是所调用的组件的配置是请求一个新事务。拦截的粘性代码将确保把当前的事务放在一边, 创建一个新的事务, 然后调用方法。在调用的方法中完成的工作与新的事务相联系, 而不是与实际调用组件的客户端的事务相联系。当方法返回时, 将恢复原有的事务, 新创建的事务结束。

这就是拦截工作方式的最简单描述。可以想象一下, COM+运行期代表组件执行一个或多个前后的设置步骤, 就会明白 COM+运行期正在做些什么, 以及为什么轻量代理是如此重要。它们完成那些难以编写的代码, 这些代码完成了基本结构检测工作, 使我们从编写这些代码的工作中解脱出来。

拦截听起来像一个多余的过程, 因为客户调用对象的每一种方法都通过一个额外的间接过程(粘性代码), 但却能保证COM+运行期以一种非常有效的方式执行操作, 如果客户与它创建的对象使用相同的服务, 经常会发现轻量代理不是必需的。因此也不会有额外的开销。即使使用的服务不同, 增加的代码也是正好足以处理这些不同之处。

## 2. 环境

COM+通过环境跟踪运行期的请求以及对象需要的服务。

一个环境(context)是一系列运行期的属性，代表了一个或多个 COM+对象的执行请求。

例如，环境将告诉COM+，一个对象正在使用缓冲池或请求一个新的事务，这些请求是由声明性属性(declarative attribute)来定义的，声明性属性是使用Component Services Explorer设置给组件的，存放在COM+类别中。这些设置直接影响包含在环境中的属性，并给COM+运行期提供正确实现运行期对象(集)的服务的信息。稍后你将看到，能够在对象中访问这些属性中的一部分。

每一个COM对象当其创建或从缓冲池取出时，都与单个的环境相关联。与对象关联的环境一旦激活，将在对象的整个生存期中保持不变，直到对象返回到缓冲池或被取消。由COM+使用的环境在对象生存期是有效的，使对象与边界外(out-of-bound)信息相关联。

术语“边界外”(out of band)意思是数据由COM+在幕后管理。

这些数据需要将服务应用于组件，没有环境则必须自己来保持这些信息。

如果一些对象的运行期请求是“兼容”的，它们可共享同一个环境。这里使用了兼容一词，是因为在运行期共享一个环境的组件类型的配置不需要统一。COM+用于测试环境兼容性的算法目前没有记录在文档中。在一个对象内部，可以获得一个环境的引用，可使用GetObjectContext方法来得到与当前对象关联的环境。这将返回一个 COM对象，称为ObjectContext。在这个对象上通过函数返回的接口是IOBJECTContext。

```
Set objContext = GetObjectContext()
```

### (1) 激活

对象进入可供客户使用的状态的过程我们称之为“激活”，创建对象的客户称为“激活者”(activator)，并且激活过程在 ASP代码调用CreateObject时发生。在激活期间，环境与一个对象相关联。

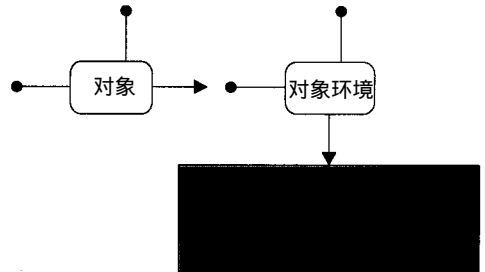


图15-3 对象、对象环境以及环境之间的关系

### (2) 对象与环境

对象、对象环境以及环境之间的关系如图15-3所示，一旦对象激活，这个关系将保持不变。

ObjectContext是一个COM对象，提供对运行期属性的访问。这些属性保存在与对象相关的环境中。

### (3) 环境协商

当一个COM+对象创建时，COM+类别用来确定组件所用的服务。假如现有的环境匹配新创建的对象的環境，将使用现有的环境，否则将创建一个新的环境，如图15-4所示。

如果组件请求的服务意味着激活者的环境与所需要的不兼容，则一个不同的环境(和ObjectContext)将用于该对

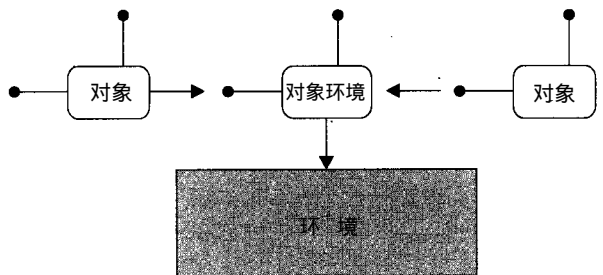


图15-4 对象环境匹配



象，如图 15-5 所示。

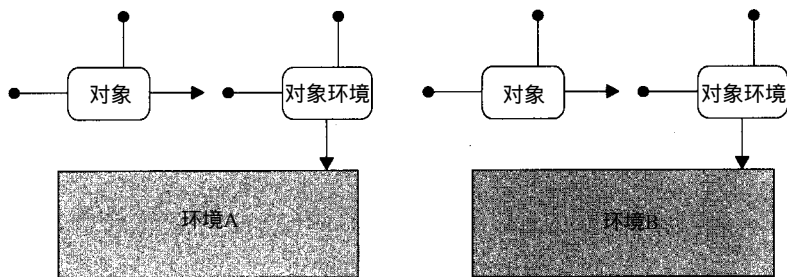


图15-5 创建新的环境

返回到客户的代理确保激活者与对象之间的环境差异在运行期由 COM+ 运行期透明管理。因此在代码中不必操心这种管理工作。

#### (4) IIS和环境

IIS使用环境向COM对象提供对ASP内置对象的访问，如图 15-6 所示。

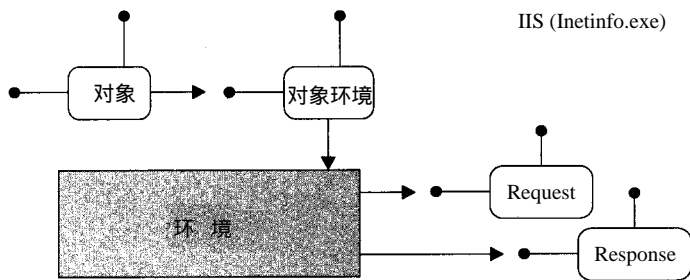


图15-6 ASP对象与环境的关系

对ASP所有的内置对象，例如 Request和Response的引用作为属性保存在环境中，该过程由IIS完成。任何与环境关联的 COM对象可通过与环境相联系的ObjectContext访问这些属性。记住，ObjectContext是一个COM对象，它提供从对象内部进入环境的接口。

这意味着一个COM对象能够访问 IIS(或任何其他使用环境暴露功能的应用程序)的功能，即使在IIS或对象之间没有“显式”的链接或联系。这就是说，这是一个 IIS通过使用COM+的基础结构提供给组件的COM+服务。

#### 3. COM+幕后的成员(拦截、环境和COM+类别)

迄今为止，我们已讨论了关于COM+的一些关键内容：

- 从配置的组件创建COM+应用程序。
- 在配置的组件中使用COM+服务。
- 使用Component Services Explorer 配置COM+应用程序和配置的组件。
- Component Services Explorer内的管理信息将存储在COM+类别中。
- COM+运行期使用环境和拦截来确保对象的运行期环境正确地使用为组件配置的服务。

所有这些对COM+来讲都是新内容的，不是 Windows 2000前的COM的组成部分。假如你曾经用过MTS，则下面两点需要注意：

- MTS包现在称为COM+应用程序。
- MTS提供的服务，如分布式事务和安全性，现在称为COM+服务。

### 15.1.3 组件 / 对象的生存期和状态

为了使用COM+提供的服务,需要重新考虑我们的编程风格。例如,COM+提供的一个服务恰好及时的(Just-In-Time,JIT)激活,它延迟使用服务器资源直到真的需要使用这些资源时,在客户不再使用对象的实例时确保释放它们。

#### 1. COM+的激活和释放

JIT激活指的是,直到调用第一个接口方法时,才激活对象的实例,这意味着一个对象的初始化代码和资源请求将被延迟,直到真正需要它们。也就是说,直到绝对需要时,才消耗服务器的资源,即消耗资源是客户调用对象方法的直接结果。这是很好的办法,因为这意味着一个客户能在应用程序生存期的早期创建一个组件,在较晚时使用,而不需担心过早地消耗资源。

基于相同的原则,COM+尽可能使不用的对象失效,尽可能早(AS-Soon-As-Possible,ASAP)失效使COM+能释放对象占用的内存和消耗的资源,同时允许客户确信仍然引用一个真实的对象,而不必了解此对象是否已被取消。假如能缓冲对象,可使对象失效(deactive)而不是取消(destroy)它。

但是一个客户怎样才能在应用程序生存期早期创建一个组件的实例,并仍然支持JIT激活和ASAP失效?答案是我们前面看到的拦截机制。当一个客户请求一个对象实例时,COM+拦截这个代码(并且假设JIT激活可用),它不创建对象实例,而是提供给客户一个虚拟的代理,使客户相信引用的是一个真实的对象。

然后,当客户调用对象的方法时,代理拦截该调用并由COM+创建一个真实的对象实例,这个实例完成必要的处理过程。当方法调用完成后,对象失效,但是代理继续“欺骗”客户。如图15-7所示。

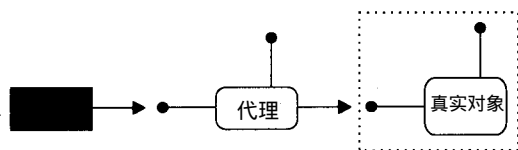


图15-7 客户调用对象时代理的拦截过程

正如我们已经解释过的,JIT激活的含义是非常清楚的,ASP页面(或其他客户)首次调用对象的方法时,激活对象。ASAP失效相对来说就不是很清楚了。当方法调用完毕后是否立即使该对象失效?作为一个好的可扩展的COM+组件,答案是肯定的。这听起来会让人感到奇怪,因为这样做,服务器必须重新创建对象,这会导致一定程度总开销的浪费。但是,需要记住的是,任何应用需要考虑的一个重要因素是服务器宝贵的资源(例如内存等),快速使用,快速释放以便其他客户能够使用这些资源。COM+保证能够做到这一点的唯一方法就是,使对象失效,如果不进行缓冲就取消对象。

COM+能够根据需要有效地创建一个组件的实例,不断创建和释放对象的事实依赖于它们的配置而不造成过高的系统开销。

#### (1) COM+应用程序中缓存类对象和DLL

为有效创建对象,COM+应用程序缓存COM服务器DLL和类对象。

类对象是一种创建COM+对象的机制。类对象是一个COM对象,支持COM接口

IClassFactory。该接口有一个方法 CreateInstance，创建并返回一个 COM+ 对象。每种类型的 COM+ 组件有一个类对象。VB 会自动创建这些类对象。

当客户创建一个包含在 COM+ 应用程序中的组件实例时，将会发生：

- 如果内存中没有的话，包含组件的 DLL 将被调入。
- 创建组件类型的类对象。
- COM+ 请求类对象创建新的 COM+ 对象实例。

当 COM+ 应用程序载入一个 COM 服务器 (DLL) 去创建某类型的 COM+ 对象时，在 COM+ 应用程序关闭前，DLL 一直保存在内存中。假如 COM+ 不能完成缓存工作，当最后一个由来自于该 DLL 的类对象创建的对象取消时，卸载该 DLL。一个对象的重复创建和取消会引起一个 DLL 重复载入和卸载，可能会引起硬盘不停地运转。COM+ 允许定义空闲时间段，经过这个空闲段后 COM+ 应用程序和所有载入的 DLL 才释放 (见图 15-8)。

当 COM+ 应用程序创建一个类对象时，将保持对类对象的一个引用直到其卸载。这意味着第一次创建对象以后仅需要执行下面一个步骤：

- COM+ 请求类对象创建一个新的 COM+ 对象的实例。

这种缓存能够改进对象创建的性能，这也意味着对于创建非缓冲对象，JIT 激活的开销不致于过高。

## (2) JIT 激活不需要改变 COM

需要清楚的是 COM 的基本规则没有为了对服务的支持而改变，如果没有外部引用，将取消 COM+ 对象的实例。然而，这会改变客户编程方式，使我们的组件成为无状态组件。

### 2. “无状态”编程范型

一个 COM+ 组件，像任何 COM 组件一样，能够在客户交互 (方法调用) 之间在内部存储信息。这一工作我们称之为保持状态，称这种组件为状态化组件。现在，假如我们标识该组件为使用 JIT 激活，并为使对象失效增加必要的调用，这个模型将发生较大的改变。

由对象存储和管理的信息，当客户调用时，对于后续的方法调用将不再可用。通过使组件使用 JIT 激活，告诉 COM+ 在方法调用之间可使对象失效，这其中的含义就是下一个方法调用将可能潜在地由不同的对象实例处理。

这里使用了“潜在地”一词，是因为如果对象被缓冲，那么使对象失效并放回缓冲池，则从缓冲池中取出供下一个调用使用的对象，可能会与原来的不相同。

### 3. 对于 JIT 组件忘掉面向对象编程

这个问题需要一些时间来理解，读者可能读过相关的许多文章，都在一定程度不正确地

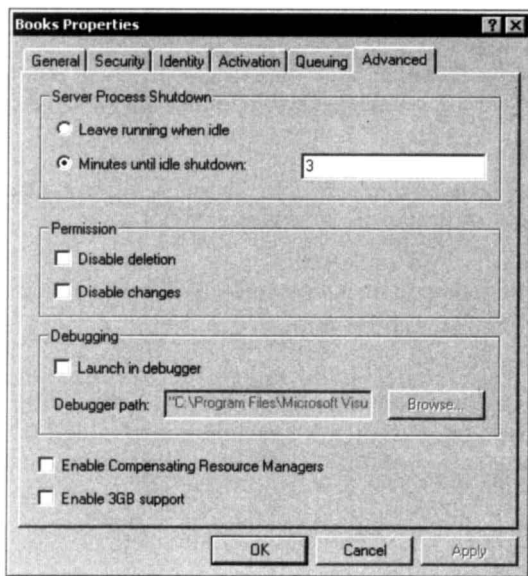


图 15-8 设定空闲时间段



定义无状态(stateless)编程模型这个概念,简单而言,无状态编程模型对于开发人员有两方面含义:

- 一个客户不能依靠同一个对象实例处理每一个方法的调用(对于同一个对象)。因为无法确定方法调用之间的一个对象的状态。
- 一个对象不能依赖于同一个客户调用其提供的每一个方法,如果对象被缓冲,许多客户将重复使用同一对象。

这两点实际上十分简单,但对创建组件和客户的方式有很大影响,面向对象编程(OOP)对JIT组件来说,不能正常工作。面向对象技术的基础是对象有同一性、行为以及一段时间的状态。恰好是最后一点不满足要求,因此如下代码不能正常工作:

```
Dim objPerson As Richard

Set objPerson = New Richard

objPerson.Name = "Richard"
objPerson.DOB = 1972
objPerson.Title = "Developer"

objPerson.UpdateDetails
```

这是一个非常典型的OOP,我们先设置对象的属性,然后要求对象通过一些存储媒介保存变化情况。你能看出问题吗?是的,每次属性的存取和方法的调用将引起COM+的运行期创建或取消COM对象,从代码中看不出这个过程,因为创建和取消发生在COM+运行期内,在我们调用UpdateDetails方法时使用的对象实例,可能不是设置属性时使用的那个。对象存储什么内容呢?存储的可能是在组件Class\_Initialize例程中初始化的缺省值。

作为传统的OOP不会使用COM+ JIT配置的组件。我们必须回过头来考虑其他的使用组件和客户的方式。不使用JIT的组件可使用OOP技术进行编程,但对于JIT组件必须把代码改变为“面向过程”。简单说,这意味着客户要把处理一个请求所需的所有状态传给对象(在每次方法调用中)。

#### (1) 面向过程的对象

对于上面的代码,我们重写为:

```
Dim objPerson As Richard

Set objPerson = New Richard

objPerson.UpdateDetails "Richard", 1972, "Developer"
```

我们现已知道依赖于方法调用间的状态不是一个可行办法,所以将所有需要的状态作为参数传给完成UpdateDetails的过程。

在ASP中,对于无状态组件有两个地方需要注意:

- 避免存储任何信息在组件的局部变量中。
- 避免页面-页面之间保持组件的引用

最简单的避免客户将信息存储在对象内的方法是,只提供组件的基于方法的接口,组件应没有公共属性,所有与组件的交互都通过方法调用完成,所有的组件所需的工作信息,均作为参数传递给方法调用。

#### (2) 对象状态可重复使用

我们知道包含相关信息的对象状态是由对象保存的，通常大多数对象保存的状态与某一特定客户有关。例如，在一个电子商务的应用程序中，如果用一个 COM对象表示某一特定用户采购的内容，那么该对象将唯一地对应这个用户，该对象的状态对于他人毫无用处，所以不能重复利用。

为什么要重复利用对象的状态？假设一个简单的对象可使一系列值有效，一些应用程序创建并使用此组件的实例。现在，假设每一实例从数据库中装入这些值，创建这些实例将花费很长的时间，如果有许多人用某一组件，将需要大量的数据库输入输出，给应用程序带来的开销很大。

通过缓冲这样一个组件，可以重复利用由组件实例创建的值列表，而无需每次重复装入。因为对象保存的状态并不特定对应于某指定的客户，如上述的采购示例，所以可以实现上述方法。因此，若 VB支持对象缓冲，需要记住的是状态可以重复使用。JIT是一个可加快对象初始化的好方法。一个具有代表性的例子就是 OLE DB，它能够缓冲数据库的连接，建立数据库的连接需要长时间，由于一般的启动过程和身份验证相当慢。所以，通过缓冲这些连接，连接开销大这一问题仅在第一次启动对象时出现，而后的用户可重复利用，当然这里假设安全和连接字符串是兼容的。

### (3) 在可扩展系统中保持对象的时间应最小化

前面提到的购物示例中，我们有一个代表购物的对象，但其可扩展性太差。要理解其原因，假设系统工作于一个大型的网站如 amazon.com，支持成千上万个同时访问的用户，每一用户有一相应的COM对象表示他们的采购行为，存储在 Session对象中。现在，假如我们知道组件需要 10KB内存来执行和维持采购行为，估计在高峰时有 5000个活动的Session，需要  $5000 \times 10\text{KB}$  (大约 50000KB 或 49MB) 来维持这一个组件。即使内存比较便宜，对于这个问题也要引起足够的重视。

选择用于创建组件的工具，有助于我们实现其他一些无状态组件的关键设计。通过 Visual Basic创造的组件的类型应该仅仅存在于一个应用程序的页面范围之内，通过 VB创造的组件不应存储在 Session中或者应用程序级的变量中，任何这样做的企图将导致 Web服务器性能严重地下降，可以配置 ASP从而产生一个错误信息，防止这种情况的发生。通过这种方式，在创建一个Web的应用程序时，ASP自身促进无状态组件的使用。这取决于 VB组件可以支持的线程模型。

## 15.2 单元和线程模型

作为VB ASP的开发人员，不需担心单元和线程模型，但应该意识到它们是 COM+的基本组成部分。理解它们之前首先需要理解线程。

### 15.2.1 线程

简单地说，可把线程看作“轻量进程”，不像通常的进程，线程并不是实际的系统资源，在Windows下启动一个运用程序，如 Notepad，操作系统将创建应用程序在其中执行的一个进程，这个进程将有其自己的地址空间，因而独立于其他进程，这样即使这个应用程序出现严重错误，将保持其他应用程序的完整性。

在一个进程中总是至少有一个线程，称为主线程。一个线程是操作分配 CPU时间最基本

实体, 因此, 如果有 10 个进程, 每一个进程有一个线程做一些密集性的工作, 操作系统将分给每个线程所有可获取的 CPU 时间的 10%, 这个时间的百分比要考虑到实际花费在管理线程的转换上的时间, 因而, 给予执行代码的进程的全部时间可能略少, 可能是 9.5% 左右。

一个进程能容纳多个并发运行的线程, 每个线程分享提供给这一进程的系统资源, 如虚拟地址空间, 这意味着这些线程可以执行应用程序的代码、访问共用内存和其他一些资源 (比如文件), 而不用消耗太多的额外资源。

因为相对于进程而言, 线程是“廉价”的, 真正的可扩展系统总是使用“每个客户一个线程”, 而不是“每个客户一个进程”, 因为节约资源十分重要。依赖于系统的划分, 对于“每个客户一个线程”的不同的应用程序, 系统也可很好地使用多进程。COM+ 使用了这种方式, 即每个 COM+ 应用程序在自己的进程中运行。

COM+ 也可缓冲线程, COM+ 做这项工作十分有效, 当线程缓冲池耗尽时, 组件将开始以一种环形的方式共享线程, 因而如果缓冲池的大小为 5, 则第 6 个创建的组件将共享第一个线程, 第 7 个共享第 2 个线程, 依次类推。使用 COM+ 的好处还在于无需在代码中编写线程管理方面的代码, COM+ 会自动处理线程, 不能创建和管理线程的语言可充分利用 COM+ 的这个功能。

### 15.2.2 单元

现在我们已经理解了什么是线程, 可以进一步研究单元 (apartment) 的角色,

一个单元是一个同步机制和 COM 对象的逻辑容器 (container), 从而提供一条线程的执行, 能在一个进程之中调用一个对象的方法。

编写一个 COM 组件时, 需要考虑线程如何访问组件的实例, 如果不止一条线程同时访问相同的对象, 需要在更新数据的方法中执行一些类型的同步操作, 否则会出现一些潜在的问题。

例如, 同时使用相同的全局变量的多线程有可能在同一时间企图操纵同一数据, 这会导致出现不一致的结果。

因为 COM 的设计目的是让组件开发人员集中精力编写组件, 它给出相应的选项去选择组件中需要的同步操作。如果不担心方法中的线程同步问题 (也就说不止一个线程同时执行相同的代码), 可以标记组件以单线程单元 (Single-Threaded Apartment, STA) 方式执行。

不像 COM+ 服务, 使用开发工具而不是 Component Services Explorer 创建组件。其机理是非常十分简单的, 仅仅编程人员知道这个组件是否是线程安全的, 因而除了编程人员外其他人员无法定义其属性。

在 COM+ 中, 在许多方面没有充分利用单元。现在环境是对象的主要容器, 但单元仍然是非常重要的, 我们可以把单元简单地认为是环境的容器, 如图 15-9 所示。

单元可被进一步分成一个或更多的环境, 在单元中环境的数量依赖于在其中创建的对象数量和这些对象的环境要求。缺省时每个单元都有一个缺省的环境, 缺省的环境用来包含不配置的传

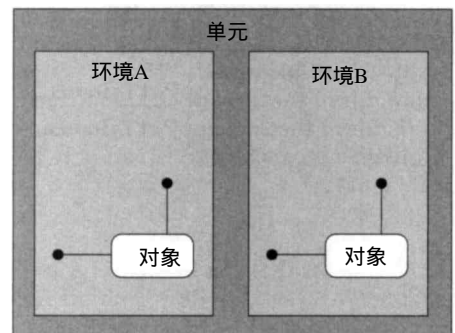


图15-9 单元和环境的关系

统组件，这些组件不能使用 COM+ 提供的服务。举例来说，这些组件不能通过组件服务装配到 COM+ 应用程序中。

一个进程能包含多个单元，单元能为其中的对象提供不同的相关线程服务。根据单元的类型，每个单元与一个或多个线程相关联。COM+ 定义三种单元的类型。

- 单线程单元 (Single-threaded Apartment)。
- 多线程单元 (Multi-threaded Apartment)。
- 中立线程单元 (Neutral-threaded Apartment)。

#### 1. 单线程单元 (STA)

一个单线程单元 (STA) 或单元线程 (Apartment-Threaded) 模型，在每个单元内基本上仅允许一个单线程运行，该线程在其生存期中与单元相联系。一个进程可包括任意数目的 STA。

STA 是为需要调用同步化和 / 或具有线程类似性 (affinity) 的组件而设计的，前者意味着不能并发调用对象的方法，必须一个完成后再调用另外的一个方法，后者确保在生存期中总由同一线程调用一个对象的每一个方法。

对于 COM+，如果一个组件没有线程类似性，但是要求调用同步化，则可通过使用中立线程模型，使性能有所提高。COM+ 提供调用同步化，而不必约束组件访问指定的线程。

目前，STA 是 VB 唯一支持的单元类型。

#### 2. 多线程单元 (MTA)

相比较而言，MTA 允许多个线程在同一个单元内运行，因此允许多个线程调用同一个对象实例的方法。不同于 STA，每一个进程只能有一个 MTA，如图 15-10 所示。

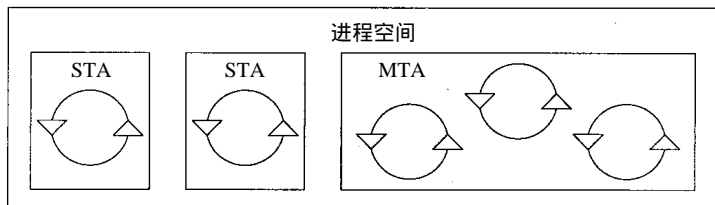


图15-10 线程、MTA与进程的关系

在 MTA 中运行的组件应考虑到它们能被多线程调用。每次调用可通过不同的线程进行，而且可能出现并发调用。

#### 3. 中立线程单元 (NTA)

STA 和 MTA 允许对象可在适合于线程特性的执行环境中运行，但是如果对象的请求是调用同步化的且不具有线程的类似性，则 STA 效率较低。这种情况下，对象不关心是什么线程调用它，假设仅有一个线程调用它。如果我们这样实现，就节省一个不必要的线程转换（如果调用者在一个不同的线程上）。事实上，调用者的线程总是可以调用对象的方法，只要保证没有其他的线程已经调用该方法。这就是新型的中立线程单元在 COM 中所起的作用。

NTA 确保一个 COM 对象的方法调用串行化，但是任何线程都可产生这些调用。这有两个主要的优点：

- 不要求跨线程的调度。
- 在 NTA 中的几个对象能同时激活。

结果是一种更快速的应用程序。

### 15.2.3 线程模型的属性

COM / COM+ 组件通过指定线程模型来表示所采用的线程的职责。这个线程模型向 COM 运行期表明组件所涉及的线程情况，同时直接影响组件在其中创建的单元的类型。

在COM+中有五种线程模型：

- 单线程(Single-threaded)：组件的实例总是在主 STA 中创建，主 STA 是在一个进程中产生的第一个的 STA。
- 单元线程(Apartment-threaded)：组件的实例必须在 STA 中创建。
- 自由线程(Free-threaded)：组件的实例必须在 MTA 中创建。
- 双线程(Both-threaded)：组件的实例可在 STA 或 MTA 中创建，由 COM+ 根据激活者的单元类型决定。
- 中立线程(Neutral-threaded)：组件的实例只能在 NTA 中创建。

#### 1. 线程模型

因而，COM 有一系列的线程模型，组件开发人员可决定为了处理从一个或多个线程访问组件而需担负的责任，组件负担的任务越重，就可能进一步优化组件的使用，提高使用它们的应用程序的性能和可扩展性。

VB 线程模型不是真正的问题，在 VB 6.0 中有两种选择：

- Single：对所有组件实例的所有访问通过单一线程串行进行。
- Apartment：对一个组件实例的所有访问通过单一线程串行进行。但不同的组件实例存活在不同的线程中，使用的线程在组件的生存期中是静态的。

单元线程模型对于大部分 ASP 组件类型的开发是适宜的。但是，如果在 ASP Session 或 Application 对象中存储组件(对象)的实例，从性能角度考虑，应使用自由的或双线程模型。这意味着不得不使用 VC++，这些线程模型不适合 VB，因为 VB 不能用于编写多线程应用程序，并对编程人员隐藏了线程概念。

#### (1) 单线程模型

单线程模型有其不足之处，它起源于 Windows 3.1 下早期的 COM，除非你正使用的环境指明是使用单线程模型，否则没有必要考虑使用它。IIS 明确建议不使用单线程模型。

假如在多线程应用程序(如 IIS)中使用了单线程组件，那么可能正在使此应用程序的较大部分单线程化。使用单线程组件意味着，对每一个组件实例的每个方法调用都不得从同一个线程进行。设想一下，假如有 50 个用户访问一些 ASP 网页，这些网页使用了单线程组件，每一个网页调用组件的方法。则第 50 个用户不得不在其调用执行之前等待第 49 个用户的调用执行完毕；而第 49 个用户也不得等待第 48 个用户的调用执行完毕；同理，第 48 个用户不得等待第 47 个用户的调用执行完毕，依次类推。可以设想第 50 个用户若想等待 ASP 页面显示出来，会感到非常的漫长。

单线程模型告诉 COM，组件不能够有效地使用线程；实际上，它告诉 COM 存在着严重的线程问题。若想让 COM 逐个调用组件的所有实例，那么组件可以访问全局或局部数据，而不必同步。假如一个线程以不同步的方式访问全局或局部数据，另一个线程同时也做同样的工作，其中一个的变化可能会丢失。



所以起码的要求是：永远不在 ASP中使用带有这种线程模型属性的组件，也永远不购买只支持这种线性模型的组件！

## (2) 单元线程模型

单元线程模型对大部分 ASP组件来说是一个正确的选择。由于 VB支持这种线程模型，所以VB非常适合于编写大部分类型的 ASP组件，条件是这些组件不需要快速执行或访问 VB不能访问的操作操作系统/SDK函数。当然如果你是一个技巧娴熟的 VC++程序员，使用VC++可能更好。那么，我们什么时候应使用 VC++来代替VB呢？

总的说来，VB和VC对编写使用单元线程的 ASP组件都是非常合适的，但 VB中一些有趣的实现可能曲解了使用 VC++所表达的意思。让我们回顾一下单元线程模型究竟意味着什么，并讨论如何选择相应的语言。

选择单元线程模型时，COM确保对组件的单个实例的所有访问通过一个线程串行进行。同一线程可用在对象生存期内调用组件的任何接口方法。由于调用的始终是同一个线程，对象就有着线程类似性，并对调用者可做多样的假设。这也意味着 VC++组件能够使用线程局部存储(Thread Local Storage, TLS)。

简单地说，TLS就是一种有效的为每个线程构造特有的全局变量的方法，我们不必管理对全局变量的访问，TLS可为我们管理全局变量。

单元线程模型对大部分环境来说是一个合适的线程模型，所以在 VC++和VB 6.0中该模型是缺省的线程模型选项。假如有 50个用户访问使用该模型的组件的 ASP网页，每一个调用都能并行处理，所以理论上每个用户的响应时间是相同的。

## (3) 自由线程模型

自由线程对象运行在一个进程的多线程单元 (MTA)中，如前面所看到的，在COM+的一个应用程序中只能有一个多线程单元。MTA中可以有一个或多个线程。任何标记为自由线程的对象都将在MTA中创建，而不考虑调用者的单元。现在，所有这一切听起来可能很好，多线程一定意味着可扩展，其实这不一定必要。

在同一环境或同一单元内，调用从一个组件转移到另一个组件时，此调用为直接函数调用。在Win32环境中，这一处理进程是很快的。但若调用从不同环境产生，或跨单元时，要产生这一调用就需要创建一个代理。跨单元 (cross-apartment)代理会由于发生线程选择而变得很慢。

代理的使用为每个函数的调用增加了开销并使得系统变得缓慢。由于自由线程组件将一直在MTA中创建，那么访问此组件的调用都将由代理创建。这意味着虽然自由线程组件听起来很好，但在ASP脚本中使用时，性能会有所下降。

## (4) 双线程模型

双线程模型是一类特殊的线程模型，因为它能够根据激活者的单元类型采用单元线程对象以及自由线程对象的一些特征。这一线程模型带来的优点是，无论组件在那里创建，将总是与激活者创建在同一单元 (NTA除外)中。若组件从单线程单元中创建，操作方式与单元线程组件类似并可在单元中执行，若组件从多线程单元中创建，操作方式与自由线程组件类似并在MTA执行。在这两种情况下，都不需要跨单元的代理调用。唯一需要跨单元的代理调用的情况是，对象在NTA中创建。

### (5) 中立线程模型

COM+和Windows 2000引入了一个新的线程模型，中立线程模型。它是一个比较适合于COM+服务器组件的线程模型。在出现MTS以前，对于灵活的ASP组件首选的线程模型是“多线程”，加上自由线程调度器而聚集。

这样的组件、快速、可扩展性强，但对于开发者来说难于创建，因为它们迫使开发者去应付例如锁定和调试多线程应用程序之类的问题，编写代码时不得不采用VB以外的语言。中立组件易创建，因为COM+承担了较困难的部分，而且已经作为COM+的一部分被实现。

每一个进程最多有一个NTA。不像MTA和STA，NTA没有自己的线程。当MTA或STA中的一个线程创建中立线程对象时，将向对象返回一个轻量代理。这一代理将转到被调用者的对象环境，并且不需要一个线程转换。实际上，进入NTA中的环境时，进程中的线程不需要执行线程的转换。这意味着所有对中立线程对象的调用都是直接的，或只需要一个轻级代理来完成跨环境的管理，这使得NTA对象运行的速度快并独立于调用者的单元。在Windows 2000中，对于没有用户界面的组件，这一线程模型是首选的设置。

一旦VB删除现在的运行期所具有的线程类似性，我们将看到中立线程模型为VB所支持。

### 2. ASP组件和单元线程模型

我们对单线程和单元线程模型的检测表明，对于用VB编写的ASP组件来说，唯一的选择是单元线程模型。如果以后VB支持中立线程模型，它将成为首选的选择。

若由于某些原因使用单线程模型，则应用程序的可扩展性和潜在性能存在比较严重的问题。对于每个组件实例所调用的每个方法，都必须通过一个线程串行进行，这就是瓶颈所在。

若在VB中编写ASP组件时使用了单元线程模型，则完全不必担心任何类型的多线程同步问题。你仅需记住全局状态是真正的“每个单元”状态，更不必担心组件不能工作，事实上VB的运行期使编程工作变得相对简单。

就IIS所涉及的内容，单元线程组件应只在一个ASP中使用，且只应存储到Session对象中，并且绝不能存储到Application对象中。那么如何在IIS中使用一个组件并定义其范围，即在IIS应用程序中它的生存期和可利用度。不同的范围适合于不同的线程模型（包括自由线程和多线程），所以下面我们来定义组件在IIS中使用的范围，并讨论不同的线程模型如何与不同的范围较好地搭配。

#### 15.2.4 线程模型和范围

我们刚刚进行了关于组件线程模型的讨论，讨论了各种线程模型的优缺点，也了解组件在ASP应用程序中的使用情况。我们关心的是如何使两方面结合起来，以及不同类型的组件如何在ASP应用程序中使用。

组件的用法与可扩展性和其在应用程序中何处使用有关，这里有三种使用域，或者说是组件可占用的范围。这些范围基于组件的生存期，以及哪些特定的用户能够访问此组件。这三个范围是：

- 页面范围(page scope)。
- 会话范围(session scope)。
- 应用程序范围(application scope)。

### 1. 页面范围

在单一的ASP网页中，创建、使用并释放一个组件实例的所有引用时，该组件可以说是具有页面范围。假如对象的生存期超过了一个页面，并在会话级或应用级的变量中储存对象的引用，则该对象就不是一个页面范围的对象。页面范围的对象仅能存在于一个页面的范围内，超出了页面的生存期，组件的生存期也就结束了。在网页结束时，包含在组件中的一些信息也将丢失。

在网页范围内创建一个组件时，意味着所有组件的调用将来自当前处理 ASP脚本的线程。这一过程发生在 ASP应用程序的单线程单元中。正如早先所看到的，希望确保这些调用是直接调用，因为这一方法提供了最好的性能。要使用直接调用来访问组件，你需要确保组件运行在与执行ASP脚本相同的单元中。

假如在页面中调度创建单线程组件，则仅能在 ASP应用程序的主STA线程上创建。这意味着组件的所有调用将不得不进行调度穿过单元边界，以便于访问运行在主线程上的组件。

应用程序的主线程是在一个 STA中，即任何时候都只能有一个激活的执行路径。当访问运行在此线程上的组件时，这意味着执行组件的时候，其他的组件不能在 ASP应用程序中执行。这意味着使用此线程时，不能处理其他进入的用户请求。可为多用户有效地创建单任务 Web服务器。一旦第二个用户试图去访问服务器，执行就会停止。

我们需要那些使用直接调用能够访问的且运行在非主线程中的组件。自由线程组件将运行于MTA中，能够自由地离开主线程继续其他处理过程。但是使用运行于 MTA中的这一组件，且页面的处理是在 STA中，就有了运行于不同单元中的对象。它们之间的调用的唯一方式是调度信息穿过单元的边界。虽然这不会把 Web服务器变成一个单任务服务器，但每次访问这一组件时都将耗费更多的时间和资源。

创建一个自由线程或中立线程组件时，也就创建了不具有线程类似性的组件。这意味着组件不关心在什么线程上执行。可以从任何线程中调用它。然而，仅中立线程组件能由任何线程调用。与一个 STA有关的线程不能调用自由线程组件，除非组件集成了自由线程调度器 (FTM)，以避免COM创建代理。

自由线程调度器使得对象的接口指针能够在同一进程中的单元间有效地调度。

当一个对象使用了 FTM来进行优化时，COM+基础结构将确保与对象在同一进程中的任何客户总是直接调用其方法，永远不会使用代理。

在组件集成了FTM的情况下，这两种类型的组件(自由和中立)都通过使用与一个 STA关联的线程来调用，即组件中的调用是直接的或者通过一个快速的代理实现的，因此不会因调度招致任何性能上的损失。

假如激活者与 STA一起使用(IIS 5.0中的情况)，单元线程组件将在创建它的同一线程上执行。当一个单元线程组件在 ASP脚本中创建时，将能够在处理 ASP脚本的线程上执行。这意味着对组件的任何调用都可以通过直接调用来完成。这种情况下，单元线程组件等同于自由线程或中立线程组件。这意味着 VB开发者可创建自己的组件，我们总有方法来创建同样能正常工作的组件，在这种情况下也可使用 C++来创建组件。

### 2. 会话范围

当在会话范围创建一个对象时，在用户的会话期内，此对象对于引用它的所有 ASP页面都是可用的。每一个在应用程序中创建的会话也将拥有自己的会话范围的对象的实例。为了

创建一个会话范围的对象，应在 global.asa 文件中为该应用程序加入相应的条目。

```
<OBJECT RUNAT=Server SCOPE=Session ID=myID PROGID="objectID"></OBJECT>
```

RUNAT = Server 参数表明，这个对象是一个服务器端组件。SCOPE = Session 表明这个对象具有会话级范围。此应用程序中的任何 ASP 脚本都可以通过 myID 对象的引用来访问它。

也可以通过在后续的 ASP 页面请求之间保持对象引用，来创建一个会话级范围效果。这可以通过把这些引用存储到会话级变量中的对象来实现，如下所示。

在 page1.asp 中：

```
<%  
Dim objMyObject  
Set objMyObject = Server.CreateObject("myObject.ProgID")  
...use the object on the page  
Set Session("myObjRef") = objMyObject  
%>
```

在 page2.asp 中：

```
<%  
Dim objMyObject  
Set objMyObject = Session("myObjRef")  
...use the object on the page  
%>
```

当在会话范围中存储一个对象时，就意味着同一用户提出的多个页面请求都可以访问这个对象。处理 ASP 请求的机制是，无论请求何时进入，ASP 脚本都能从线程缓冲池中为特定的 STA 指定页面的处理方法。所以，如果这一页面请求将信息存储在会话级变量中，并且这个信息包括一个对象的引用，那么新的页面也可以访问这一信息。

因为对象引用可以存储在会话状态中，负责处理这个新页面的环境也可以访问这些对象，方式与页面创建它们时相同。看一下这五个不同的线程模型对象存储在会话范围时是如何运行的。

- 可以在会话范围存储对单线程对象的一个引用，但使用这些类型的组件时性能会降低，这是无意义的。
- 如果在会话范围中存储一个自由线程对象，当在下一个页面中使用时，将仍然在 MAT 中运行。ASP 脚本不得使用代理来访问它，这样会造成性能的下降。
- 在会话范围中存储一个中立线程组件，或者一个集成了 FTM 的自由线程组件时，就是使用不含线程类似性的组件。无论运行在什么线程上，所有对该组件的调用都将使用直接调用或通过一个轻量代理进行。正因为如此，组件会在处理新页面的线程上执行。这将给出优化的性能。对于会话范围组件，这是推荐的线程模式。然而，这也意味着 VB 不适合于创建存储在会话范围的组件。
- 当一个单元线程组件或多线程组件存储在会话范围中时，这一组件与创建它的 STA 具有线程类似性，只能运行在这一线程上。这不像一个类似于自由线程加 FTM 和中立线程组件那样线程灵活，只能使用一个线程。在 ASP 线程缓冲池中有大量的线程（缺省值为 25 个）。所以当 ASP 为进入的页面请求指定线程时，就会有 1/25 的可能，使组件可能使用建立它的线程。但是 ASP 的工作不是这种方式。

实际上在将处理传递给一个特定的线程之前，ASP 检查会话范围的内容。它将查找任何可能存储在那里的组件引用，发现一个就检查该组件是否有线程类似性。如果那里存在一个单元线程组件，ASP 就会知道哪种线程适合这个组件。它只要将请求传送给合适的线程就可



以了。

需要记住的是，一个ASP应用程序在同一时刻可能有数以百计甚至成千上万的用户。对于一个用来服务于请求的有限的线程缓冲池，可能会忙于处理大量此类线程的请求。如果一个组件运行所需的那个线程恰好被占用，则组件不能运行，ASP就不能将请求传送给另外一个线程，只有等待。换句话说，直到所需的线程可用后，请求才会继续运行。

可以设想一个情形：大量的客户创建会话级对象，这些对象都被阻塞在一个特定的线程。Web服务器很快就会发现没有可用的线程，因为它们都被特定用户使用着。服务器将一直暂停而不能处理任何事情。

避免这种情况的唯一途径就是，只在会话范围中存储没有线程类似性的对象。这就限制为只使用中立或集成了FTM的双线程/自由线程对象。建立在IIS上的单线程、单元线程和双线程组件对一个特定的线程有类似性，所以它们不应该存储在会话范围中。

### 3. 应用程序范围

应用程序范围对象可以被应用程序中的任何一个ASP脚本访问。对于整个应用程序只有一个对象实例。这个对象的插入方式与会话范围对象相同，都是使用<OBJECT>标记。应用程序范围对象所需做的一个改动是：

```
<OBJECT RUNAT=Server SCOPE=Application ID=myID PROGID="objectID"></OBJECT>
```

通过改变SCOPE参数为Application，应用程序的所有用户将只有一个该对象的实例。为应用程序范围组件选择一个线程模式时，就出现一些有趣的问题。如果使用单元线程，那么进入服务器的任何一个请求都只能被一个线程处理。这就意味着任何时候，在所有访问这一站点的用户中只有一个用户可以使用这个对象，其他用户只能等待，直到轮到他们为止。

存储在应用程序范围中的对象的类型应该是自由线程、双线程或者中立线程组件。这些组件没有线程类似性，所以无论哪个线程正在处理请求，任何一个对应用程序的请求都可以访问它们。如果正在编写一个在应用程序范围中使用的双线程组件，必须在集成的FTM上创建。如果使用的组件有线程类似性，创建这个组件时，ASP就会产生一个错误。正因为如此，它们不应是COM+组件，并且当FTM导致环境被忽略时，不应向Component Services Explorer注册。

使用一个应用程序范围组件的情况很少。也许一个页面计数组件应该以应用程序级存储，但是也可以用其他方法存储信息，而不一定要用应用程序范围组件。

### 4. 范围的选择

根据对组件范围和组件的了解，在页面请求间或Web应用程序的用户间，使用组件存储状态，可选的方法很少。这些组件只能用C++来创建，并且必须有能力同时处理多个用户。这就意味着它们有多线程代码，例如关键部分和其他所有类型的高级的、复杂的、低层的代码。总的来说，创建它们很困难，而且调试将更加困难。

在许多情况下，有比建立应用程序或者会话范围对象更好的方法，就是使用会话或者范围的变量将信息传递给在页面级创建的对象。例如，不应该给ADO Connention对象提供会话或应用程序范围。因为它创建的连接在很长时间内一直是打开的，而且脚本也不再使用连接缓冲。然而，可以在Session或者Application对象中存储一个ODBC或者OLE DB连接字符串，并访问这个连接字符串，在单一页面中的Connention对象实例中设置一个属性。通过这一途径，可以在会话或者应用状态下存储常用的信息，但是仅仅在需要的时候才创建使用信息的



对象。

如果坚持用组件在页面间传递信息，就需要提供一个方法使得组件串行化。这就意味着不是在会话范围里为已创建的组件存储一个引用，而是在会话级变量中存储将组件描述为标量值的状态信息，而且在页面结束时会取消该组件。在下一页面，再次使用组件时，就要使用Server.CreateObject创建它，再把从会话级变量中获取的状态信息传递给它。

我们不必查看整个代码示例，下面有一些描述这一过程的伪代码：

```
<%  
Dim objMyComp  
Set objMyComp = Server.CreateObject("MyComp.ProgID")  
...use the component  
Dim vData  
vData = objMyComp.SerializeMe()  
Session("MyCompData") = vData  
Set objMyComp = Nothing  
%>
```

在另一页面中再次使用这个组件时：

```
<%  
Dim objMyComp  
Set objMyComp = Server.CreateObject("MyComp.ProgID")  
Dim vData  
vData = Session("MyCompData")  
objMyComp.LoadMe vData  
...use the component  
%>
```

组件中的SerializeMe方法负责把任意的内部状态转换为一种标量数据类型，这种类型能存储为一个会话级变量中的文本。可以使用相对应的LoadMe方法从这一文本中恢复变量状态，这样就能将状态恢复到调用SerializeMe之前的状态。

### 15.3 COM+应用程序

在这一章，我们已经了解了COM+和组件服务如何帮助开发者创建基于组件的应用程序。COM+所用的一种基本方法就是以MMC插件方式为这些组件提供管理机制。它允许把各个组件组成一个整体，也就是组成一个COM+应用程序。

一个COM+应用程序就是一组配置的COM+组件。通常，它们实现相关的功能。

实际上，ASP应用程序可以使用一个或多个COM+应用程序，通过把多个组件组合成一个COM+应用程序，把这些组件作为一个整体来管理。

对于熟悉NT4中的MTS包的人来说，COM+应用程序提供的是几乎完全相同的功能。

我们已经知道COM+如何把运行在Windows 2000中的COM+对象与一个环境联系起来，现在来研究如何在组件内使用这个环境，来扩展这些组件的功能及利用COM+环境。

#### 15.3.1ObjectContext接口

正像ObjectContext是ASP内置对象真正的“根”，可以从自定义组件中使用ObjectContext对象访问运行期环境。

在VB组件中使用ObjectContext之前，必须先在项目中添加一个引用，因为要通过类型库来定义代码中所要使用的组件和接口，如图 15-11所示。

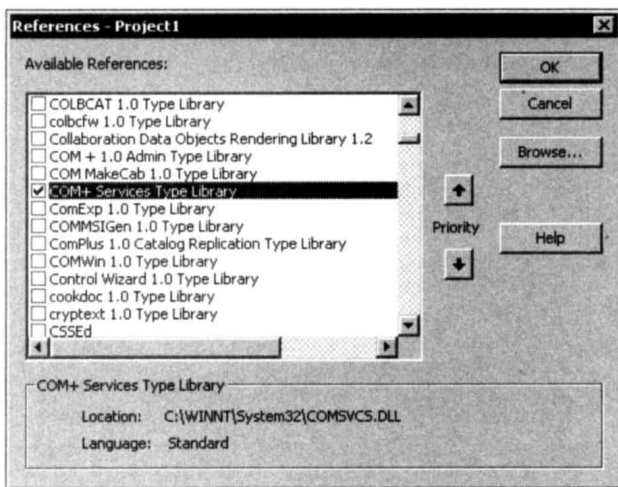


图15-11 在项目中添加引用

这允许我们通过 ObjectContext对象构造与对象相关的环境的引用，在运行期可通过使用 GetObjectContext函数来完成：

```
Dim objContext As COMSVCS.ObjectContext
```

```
Set objContext = GetObjectContext()
```

通过这个ObjectContext对象的引用，就可以就像其他 COM对象一样，使用环境的功能。

### 1. 控制COM+资源

组件通过ObjectContext对象的接口使COM+知道其正在执行的工作的状态。假如一个组件告诉COM+它已完成资源的使用，那么COM+将使该对象失效，并把资源交还给系统。

正如将在第 19章中详细介绍的事务一样，环境的一个运行期属性是所谓的“完成位”(Done Bit)。当一个对象完成工作并可失效时，就用这个布尔标志告知 COM+。通过设置此值为“TRUE”，告知COM+当前方法调用结束，可以继续工作并使此组件失效。

要将Done Bit设置为“TRUE”，可以调用ObjectContext对象的两种方法：

- SetComplete。
- SetAbort。

这两个方法看起来是相互对立的，但实际上它们都将 Done Bit设置为“TRUE”。不同之处在于对事务处理的结果的影响不同，这将在第 19章中说明。

使用这些方法，执行时可简单地把它们作为 ObjectContext引用的成员函数：

```
Dim objContext As COMSVCS.ObjectContext
Set objContext = GetObjectContext()
... Do some work
objContext.SetComplete ' This sets the Done Bit to TRUE
```

### 2. 引用ASP内置对象

COM+对象与一个环境相关联，ASP页面也有一个环境。这就意味着可以使用此环境从组件中访问ASP内置对象。正如所讨论的那样，此环境可作为一个两路通信信道。

为达到此目的，必须用两种方式扩展 VB 组件：

- 首先，必须通过ObjectContext对象访问环境。
- 其次，必须从所使用的ASP脚本中引用对象。

一旦有了这些引用，就可以像在页面上编写 ASP 脚本一样与这些对象进行交互。可在对象中编写脚本，使用与客户端页面相同的 ASP 内置对象。

为了从VB中引用ASP内置对象，首先必须告知 VB 项目我们将要使用 ASP 组件。这样VB能给程序员提供Intellisense和其他针对这些组件的支持。还能很容易使 VB 和此组件进行早期绑定，这有利于性能提高，尤其是和后期绑定的客户相比更是如此，如执行 ASP 脚本的活动脚本引擎。要获得这些优势只需添加一个对 Microsoft Active Server Pages Object Library 的引用即可，如图 15-12 所示。

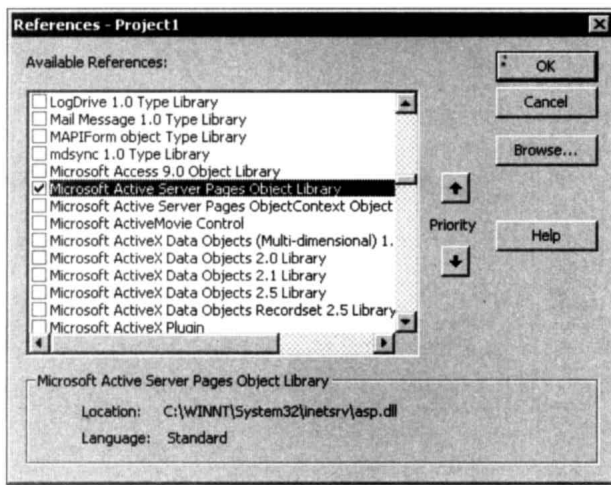


图15-12 添加对Microsoft Active Server Pages Object Library的引用

也需要用COM+ Services Type Library(服务类型库)来引用ObjectContext。

获得ASP对象的引用

既然已告知 Visual Basic 将在组件中使用 ASP 内置对象，那么下一步就是获得对这些对象的引用。我们已经看到如何引用 ObjectContext 对象，因此下面研究如何使用此对象获取 ASP 对象的引用。

ObjectContext 允许访问执行环境中的所有组件。为了访问我们感兴趣的组件，必须在 ObjectContext 中进行检索。例如想得到 ASP Response 对象的引用，必须从 ObjectContext 的对象集内捕获它：

```
Dim objResponse As ASPTypeLibrary.Response  
--  
Set objResponse = objContext("Response")
```

可以用这种方法检索每个 ASP 内置对象，所需做的只是替换对象的名称。使用存储在局部变量 objResponse 中的 Response 对象的引用，可以像在 ASP 文件中通过 VBScript 使用该对象那样使用它：

```
objResponse.Write "This was written by the component and not the ASP page"
```

### 15.3.2 Visual Basic COM+ 组件

回顾一下上一章介绍的组件，并进行扩展使之能够支持 COM+，以使用 COM+ 所提供的服务，这些服务可使应用程序更为可扩展。为达到这一目的，必须改变组件结构：

- 为了提高可扩展性，使用无状态组件。
- 数据访问将移入一个数据中心层。
- 更新表现逻辑，包括直接写入 ASP。

在此例中，我们会重现上一章例子的功能，但将重新构造以更好地支持本章讨论的概念。此应用程序的功能支持书的列表维护，这个应用程序可显示此列表，向列表中添加一些项或从列表中删除一些项。

该应用程序将使用三层结构：

- 数据层将负责与数据库交互。数据库交互信息就是用来连接数据库的信息，此层不包含这些信息。数据层的建立使得在系统之间移植应用程序变得相对简单。
- 业务逻辑层是一个真正以用户为中心的组件，因为重点放在数据和表现之间的交互，并在 ASP 脚本的窗体中包含了一些表现逻辑。由于我们的应用程序很简单，没有多少业务规则需要支持，但将保留此层，使应用程序具有分层结构。
- 表现层使用 ASP 脚本传递。

#### 1. 组件结构

应用程序中的数据组件支持与数据库的交互。正如上一章提到的，数据库是包含在 SQL Server 中的 pubs 数据库，数据组件需要从数据库中检索书名清单、向数据库中添加书名和从数据库中删除书名，有四个方法提供组件界面：

- GetTitleList：返回包含来自数据库中书名清单的一组信息。
- AddTitleToDB：向数据库书名表中添加一条记录。
- DeleteTitleFromDB：删除数据库中对应指定 ID 的记录。
- CheckTitleExists：检查数据库中指定 ID 的书名是否存在，并返回一个布尔值。

业务逻辑组件将作为数据层和表现层之间的中间层。我们将添加一条业务规则，计算每本书可能的最低售价。业务逻辑组件同样也支持增加新书名和删除已存在书名的方法。组件的接口如下：

- RetrieveTitleList：返回一个包含书名清单的数组。
- AddTitle：获得新书名的信息，添加到系统中。
- DeleteTitle：检查数据库中的指定 ID 是否存在，如果存在则删除对应此 ID 的记录。

虽然内部工作方式略有不同，但此应用程序的 ASP 页面与上一章例子中的 ASP 页面在名称和功能方面完全相同。我们将在此应用程序中添加一个 global.asa 文件，此文件包含数据库的连接字符串：

- list.asp：使用组件显示书名的完整清单。
- Add.asp：为录入新书的详细信息提供的屏幕。
- Add\_UpdateDB.asp：使用组件完成添加一本新书的功能，书的信息来自 Add.asp。
- Delete.asp：使用组件删除一本书。

#### 2. 数据组件

数据组件使用 ADO 检索存储 pubs 数据库中的信息。这些方法将检索信息，并向调用对象

返回数据或是一个指示是否成功的标志。所有的方法都使用数据库连接字符串作为参数。

创建一个新的名为BookData的ActiveX DLL项目，并命名缺省类为TitleData。添加一个对COM+ STL和ADO 2.5 库的引用。还要保证设置类的 MTSTransactionMode为1-No Transactions。缺省值为NotAnMTSObject，这可能意味着在运行期环境与组件无关。

### (1) GetTitleList 方法

GetTitleList方法在数据库中查询书名列表，再以数组形式将其返回到调用对象中。

```
Public Function GetTitleList(strConn As String, arTitles As Variant) As Boolean

    On Error GoTo errorGetTitleList

    Dim objConn As ADODB.Connection
    Dim rsTitles As ADODB.Recordset
    Dim strSQL As String

    Set objConn = CreateObject("ADODB.Connection")
    Set rsTitles = CreateObject("ADODB.Recordset")
    objConn.Open strConn

    strSQL = "SELECT title_id, title, price, notes FROM titles"
    rsTitles.Open strSQL, objConn, adOpenKeyset

    If Not rsTitles.EOF Then
        arTitles = rsTitles.GetRows()
    End If

    objConn.Close
    Set rsTitles = Nothing
    Set objConn = Nothing

    GetObjectContext.SetComplete
    GetTitleList = True

Exit Function

errorGetTitleList:

    GetObjectContext.SetAbort
    GetTitleList = False
    arTitles = Err.Description

End Function
```

这看起来有点像处理数据库的VBScript代码。主要区别在于ObjectContext引用的SetAbort和SetComplete方法的使用。它们被用来设置当前组件的 Done Bit，使COM+知道何时释放组件。在需要时才通过调用 GetObjectContext方法获取对ObjectContext的引用，而不是获取它并将其存到局部变量中去。

### (2) AddTitleToDB方法

这种方法读取用户输入的新书名信息，并加到数据库中。用户负责提供书名的 ID、名字和书的价格，以及一些关于此书的注释。

```
Public Function AddTitleToDB(Id As String, Title As String, _
    Price As Currency, Notes As String, _
    strConn As String) As Boolean

    On Error GoTo errorAddTitle
```



```

Dim objConn As ADODB.Connection
Dim strSQL As String

Set objConn = CreateObject("ADODB.Connection")
objConn.Open strConn

strSQL = "INSERT INTO titles (title_id, title, Price, Notes) VALUES ("
strSQL = strSQL + "'" + Id + "',"
strSQL = strSQL + "'" + Title + "',"
strSQL = strSQL + " " + CStr(Price) + ", "
strSQL = strSQL + "'" + Notes + "'"

objConn.Execute strSQL
objConn.Close
Set objConn = Nothing

AddTitleToDB = True
GetObjectContext.SetComplete
Exit Function

```

errorAddTitle:

```

GetObjectContext.SetAbort
AddTitleToDB = Err.Description

```

End Function

为了在数据库中插入信息，要构造一个 SQL INSERT 语句，然后用数据库连接对象的 Execute 方法，将信息插入数据库中。如果一切运行正常，就调用 SetComplete 方法。一旦出现错误，方法的错误处理程序会捕获错误，然后再调用 SetAbort 方法。

### (3) DeleteTitleFromDB 方法

要从数据库中删除一个书名时，可以把数据库连接字符串和书名的 ID 传递给这个方法。如果删除记录成功，则会返回 True；如果删除记录不成功，则返回一个错误消息的文本。

```
Public Function DeleteTitleFromDB(Id As String, strConn As String) As Boolean
```

```
On Error GoTo errorDeleteTitle
```

```

Dim objConn As ADODB.Connection
Set objConn = CreateObject("ADODB.Connection")
objConn.Open strConn

objConn.Execute "DELETE FROM roysched WHERE title_id = '" & Id & "'"
objConn.Execute "DELETE FROM sales WHERE title_id = '" & Id & "'"
objConn.Execute "DELETE FROM titleauthor WHERE title_id = '" & Id & "'"
objConn.Execute "DELETE FROM titles WHERE title_id = '" & Id & "'"

```

```

objConn.Close
Set objConn = Nothing

```

```

DeleteTitleFromDB = True
GetObjectContext.SetComplete
Exit Function

```

errorDeleteTitle:

```

GetObjectContext.SetAbort
DeleteTitleFromDB = Err.Description

```

End Function

如上章例子所示,在Pubs数据库中有四个包含书名信息的表。为使书名完全从数据库中移走,需要删除每个表中关于所选书名的信息。要注意的是,一旦使用完一个对象,例如数据库连接对象objConn,就马上释放对这一对象的引用,以确保资源在应用程序中能尽可能有效地利用。

#### (4) CheckTitleExists方法

为了适应对一些业务准则的处理,在执行某些函数以前需要检查一下数据库中某个书名ID是否存在。为此,创建了CheckTitleExists方法。这个方法获取一个书名ID,如果此ID在数据库中不存在就返回Ture。

```
Public Function CheckTitleExists(Id As String, strConn As String) As Variant

    On Error GoTo errorCheckTitleExists

    Dim objConn As ADODB.Connection
    Dim rsTitle As ADODB.Recordset
    Dim strSQL As String

    Set objConn = CreateObject("ADODB.Connection")
    objConn.Open strConn

    strSQL = "SELECT title_id FROM titles WHERE title_id = '" & Id & "';"

    Set rsTitle = objConn.Execute(strSQL)

    If rsTitle.EOF Then
        CheckTitleExists = False
        GetObjectContext.SetAbort
        rsTitle.Close
    Else
        CheckTitleExists = True
        GetObjectContext.SetComplete
    End If

    objConn.Close

    Set rsTitle = Nothing
    Set objConn = Nothing
    Exit Function

errorCheckTitleExists:

    GetObjectContext.SetAbort
    CheckTitleExists = Err.Description

End Function
```

这个方法查询数据库中的书名表,看是否存在与作为参数传入的书名ID匹配的记录。如果查询返回的记录集是空的话,就表明没有相对应的条目。这样就会就返回False,然后调用SetAhor。如果记录集不空,表明存在一个相对应的条目,就会返回Ture。

### 3. 业务逻辑组件

此应用程序中的业务逻辑组件不执行过多的处理。主要是在用户界面和数据库对象间提供一个中间层。如果以后要向此应用程序添加业务功能,这些组件可以扩展以支持新的功能,而不需对应用程序的结构做大的改动。

无论是修改现存的Book组件还是用相同的名字创建一个新的ActiveX DLL项目,都要确

保它含有对 COM+ Services Type Library 及 Active Server Pages Library 的引用(这个组件不需要 ADO 引用)。还要确保类的 MISTransactionMode 属性设置为 1-No Transactions。

### (1) RetrieveTitleList 方法

这一方法获得数据库中所有书名的列表,并将其以数组的形式返回给调用者。它还建立一个与其他信息相对应的一维数组,并包含对应每一个书名的折价信息。

```
Public Function RetrieveTitleList(strConn As Variant, arTitles As Variant, _
                                arDiscount As Variant) As Variant

    On Error GoTo errorRetrieveTitleList

    Dim objBookData As BookData.TitleData
    Dim blnTitlesOK As Boolean

    Set objBookData = CreateObject("BookData.TitleData")
    blnTitlesOK = objBookData.GetTitleList(CStr(strConn), arTitles)
    Set objBookData = Nothing

    If blnTitlesOK Then
        Dim intRows, intCount
        intRows = UBound(arTitles, 2)
        'arDiscount = Array(1)
        ReDim arDiscount(intRows)

        For intCount = 0 To intRows
            If IsNull(arTitles(2, intCount)) Then arTitles(2, intCount) = 0
            arDiscount(intCount) = Round((arTitles(2, intCount) * 0.65), 2)
        Next intCount

        RetrieveTitleList = "True"
        GetObjectContext.SetComplete
    Else
        RetrieveTitleList = "False - " & arTitles
        GetObjectContext.SetAbort
    End If

    Exit Function

errorRetrieveTitleList:

    RetrieveTitleList = Err.Description

End Function
```

这一方法使用数据库组件的 GetTitle 方法来检索书名列表。一旦检索完信息,就释放对数据组件的引用。然后计算数组的行数,建立一个有相同行数的数组,用来保存对应每个书名的折价情况。在遍历这一数组时,检查价格是否是一个 Null 值。如果是。就用 0 代替。然后计算这一折价,并将它存到新数组中。完成后,调用 SetComplete 方法,并从函数中返回。

这是一个使用 ASP 内置对象的好机会,使用 ASP 内置对象能使 ASP 脚本更简洁。虽然这一逻辑更适合于一个独立组件,但对于这个简单组件,我们把它添加到业务逻辑组件中。

### (2) RetrieveTitleListASP 方法

ASP 脚本把书名信息转换成了一个表,而不是只返回书名信息,我们从组件中直接取出该表的构造代码并写到浏览器中。这样就要创建一个新的方法,该方法调用 RetrieveTitleList 方法从数据组件中得到这个数组,再转换成 ASP 脚本。也就是说只要客户使用

RetrieveTitleList方法，组件就也能从一个非 ASP 客户使用：

```
Public Sub RetrieveTitleListASP(strConn As Variant)

    Dim arTitles()
    Dim arDiscount()
    Dim varResponse
    Dim intCount, intRows
    Dim objContext As COMSVCSLib.ObjectContext
    Dim objResponse As ASPTypelibrary.Response
    Dim res

    res = RetrieveTitleList(strConn, arTitles, arDiscount)
    Set objContext = GetObjectContext

    If res = True Then

        varResponse = "<P>The following books are currently defined in the " & _
            "pubs database:" & vbCrLf & "<TABLE CELLSPACING=""2"" & _
            "CELLPADDING=""0"">" & vbCrLf & "<TR>" & _
            "<TD BGCOLOR=""#3AC2EF""><STRONG>ID</STRONG></TD>" & _
            "<TD BGCOLOR=""#3AC2EF""><STRONG>Title</STRONG></TD>" & _
            "<TD BGCOLOR=""#3AC2EF""><STRONG>Price</STRONG></TD>" & _
            "<TD BGCOLOR=""#3AC2EF""><STRONG>Discount</STRONG></TD>" & _
            "<TD BGCOLOR=""#3AC2EF""><STRONG>Notes</STRONG></TD>" & _
            "<TD BGCOLOR=""#3AC2EF""></TD></TR><TR></TR>"

        intRows = UBound(arTitles, 2)
        For intCount = 0 To intRows

            varResponse = varResponse & "<TR>" & _
                "<TD BGCOLOR=""#3AC2EF"">" & arTitles(0, intCount) & _
                "<TD BGCOLOR=""#3AC2EF"">" & arTitles(1, intCount) & _
                "<TD BGCOLOR=""#3AC2EF"">" & arTitles(2, intCount) & _
                "<TD BGCOLOR=""#3AC2EF"">" & arDiscount(intCount) & _
                "<TD BGCOLOR=""#3AC2EF"">" & arTitles(3, intCount) & _
                "<TD BGCOLOR=""#3AC2EF"">" & _
                "<FORM NAME=""A"" METHOD=""POST"" ACTION=""Delete.asp?id=" & _
                & arTitles(0, intCount) & "">" & _
                "<INPUT TYPE=""SUBMIT"" VALUE=""Delete"" & _
                "NAME=""B1"" ></FORM></TD></TR>"

        Next

        varResponse = varResponse & "</TABLE>" & vbCrLf & _
            "<FORM NAME=""A"" METHOD=""POST"" ACTION=""Add.asp"">" & _
            "<INPUT TYPE=""SUBMIT"" VALUE=""Add a new Title"" & _
            "NAME=""B1"" >" & _
            "</FORM>"

        objContext.SetComplete
    Else
        varResponse = Right(CStr(arTitles(0)), 8)
        objContext.SetAbort
    End If

    Set objResponse = objContext("Response")
    objResponse.Write varResponse

    Set objResponse = Nothing
    Set objContext = Nothing

End Sub
```

当我们在组件中构造 HTML 字符串时，应该知道字符串的合成是一个相当占用资源的过程，所以要尽可能限制所需的字符串的个数。

### (3) AddTitle 方法

这个方法用来向数据库添加书名，它仅仅作为到数据组件的一个通路。

```
Public Function AddTitle(Id As Variant, Title As Variant, Price As Variant, _
    Notes As Variant, strConn As String) As Variant

    On Error GoTo errorAddTitle

    Dim objBookData As BookData.TitleData
    Set objBookData = CreateObject("BookData.TitleData")

    AddTitle = objBookData.AddTitleToDB(CStr(Id), CStr(Title), CCur(Price), _
        CStr(Notes), CStr(strConn))

    Set objBookData = Nothing

    If AddTitle = True Then
        GetObjectContext.SetComplete
    Else
        GetObjectContext.SetAbort
    End If

    Exit Function

errorAddTitle:

    AddTitle = "Error - " & Err.Description
    GetObjectContext.SetAbort

End Function
```

这一组件的基本步骤是创建数据库对象，调用它的方法，再根据这个方法的结果决定调用 SetComplete 或者调用 SetAbort 使 COM+ 释放对象。需要注意的是，一旦用完，就立即释放数据对象的引用。

### (4) DeleteTitle 方法

这个方法用来从数据库中删除书名。首先要确定将要删除的书名是否存在于数据库中，所以先调用 CheckTitleExists 方法。如果存在，就调用数据对象的 DeleteTitleFromDB 方法。虽然此方法在数据库中不存在此书名时也能工作，但这会造成不必要的资源占用：

```
Public Function DeleteTitle(Id As Variant, strConn As Variant) As Variant

    Dim objBookData As BookData.TitleData
    Set objBookData = CreateObject("BookData.TitleData")

    Dim ret As Variant
    ret = objBookData.CheckTitleExists(CStr(Id), CStr(strConn))

    If ret <> True Then
        GetObjectContext.SetAbort
        Set objBookData = Nothing
        DeleteTitle = False
        Exit Function
    End If

    DeleteTitle = objBookData.DeleteTitleFromDB(CStr(Id), CStr(strConn))
    GetObjectContext.SetComplete
End Function
```



```
Set objBookData = Nothing
```

```
End Function
```

#### 4. ASP页面

此应用程序的ASP页面与前一章中的几乎一样，实际上就用户而言，唯一可见的区别就是显示了书名对应的折价价格和正常价格。页面的变化发生在 ASP脚本内，在此处我们与一类新的对象模型交互。

##### (1) List.asp

这个脚本经过了很大的改动，由用交互式的调用改为用组件检索信息，仅仅调用能完成所有工作的RetrieveTitleListASP例程。页面也减少为如下所示：

```
<%
Dim objTitles 'Holds the book titles component
Set objTitles = Server.CreateObject("Book.BookTitles")
%>

<HTML>
<HEAD>
<TITLE>Titles List</TITLE>
<STYLE TYPE="text/css">
  BODY {font-family:Verdana,Tahoma,Arial,sans-serif; font-size:10pt}
  TD {font-family:Verdana,Tahoma,Arial,sans-serif; font-size:10pt}
</STYLE>
</HEAD>
<BODY BGCOLOR=WHITE>
<H1>Book Titles</H1>
<HR>
<%
'Write out the table
objTitles.RetrieveTitleListASP Application("ConnectionString")
Set objTitles = Nothing

%>
</BODY>
</HTML>
```

##### (2) global.asa

对此应用程序增加了一个 global.asa 文件，此文件用来初始化名为 ConnectionString 的应用程序级变量，它包含了应用程序中所有组件要用到的数据库连接字符串：

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>

Sub Application_OnStart()
  Application.Lock
  Application("ConnectionString") = "PROVIDER=SQLOLEDB; " & _
    "DATA SOURCE=<SERVER_NAME>;" & _
    "DATABASE=pubs;USER ID=sa; PASSWORD=;"

  Application.Unlock
End Sub

</SCRIPT>
```

在系统上安装此应用程序时，可能要改变 DATA SOURCE 参数以指向实际的数据库服务器。如果在此服务器上使用了安全措施，也同样要做相应的改变。

##### (3) 其他页面

对 Delete.asp 和 Add\_updateDB.asp 页面所做的唯一更改是对组件的相关方法的调用，它们

需要被更新以传入连接字符串信息：

```
objTitles.DeleteTitle Request.QueryString("id"), Application("ConnectionString")
```

和：

```
objTitles.AddTitle Request.Form("Id"), Request.Form("Title"), _  
Request.Form("Price"), Request.Form("Notes"), _  
Application("ConnectionString")
```

## 15.4 组件服务

管理COM+应用程序的工具是微软的 Component Services Explorer，可用此工具来创建新的COM+应用程序，向应用程序中添加组件，也可以为应用程序和它的组件设置声明性属性。

COM+应用程序为应用程序开发者提供了如下利益：

- 提供了一种将组件作为预打包集放置在 COM+应用程序内部的方法。
- 作为一个整体配置组件集。可以操作诸如组件安全性、负载平衡设置和事务等属性。
- COM+应用程序可以为开发时未设置属性的应用程序添加属性，例如组件同步属性。
- 具有在调用者的进程或独立的进程中运行 COM+应用程序的能力。
- COM+负责创建和管理组件使用的所有线程。
- COM+提供了对环境的访问。

在创建ASP应用程序时，要用到两种类型的 COM+应用程序。它们根据运行应用程序的组件的进程空间定义。

- 服务器应用程序 (server application)：服务器应用程序的组件运行在一个特定的进程（称为dll/tost.exe）中。所有COM+服务均可从服务器应用程序获得。
- 库应用程序 (library application)：库应用程序运行在创建他的客户的进程中。对一个 ASP 应用程序而言，这就意味着它在正在处理当前页面的 STA 中运行。库应用程序的性能更出色，但它不支持高级 COM+ 功能，如负载平衡、组件排队等。

库应用程序通常运行较快，因为它和 IIS 运行在同一进程中。服务器应用程序能提供更好的应用程序稳定性，因为即使它们崩溃，也不会破坏 IIS 进程。当我们调试和修改应用程序时，服务器应用程序更容易，因为它们可在不影响 Web 服务器本身的情况下卸载和重新载入。

### 创建COM+应用程序

创建并使用COM+应用程序有3个步骤：

- 组件的设计和创建。
- COM+应用程序的创建。
- COM+应用程序的管理。

我们已知道了如何进行第一步操作，现在来看看如何创建一个 COM+应用程序。我们要把已创建的组件组合起来建立一个COM+应用程序。总而言之，我们将所有将被特定的ASP应用程序(或其一部分)所使用的组件组合成一个COM+应用程序。

用以下步骤向COM+应用程序添加组件：

- 创建一个新的COM+应用程序或选中一个已经存在的COM+应用程序。可向这个COM+应用程序或新创建的空的 applications 中添加组件。
- 向应用程序中添加组件。

- 为每个组件设置声明性属性。
- 假如需要，导出应用程序使之能在其他系统上部署。

除了用 Component Services Explorer 来创建新的 COM+ 应用程序，也可用此工具管理已存在的应用程序。在管理 COM+ 应用程序时，系统管理员可改变组成这一应用程序的组件的属性和声明性属性，还可以在计算机之间传送应用程序。传送时它们的配置会有一些改变，然后改变后的应用程序会返回到其他的计算机上去。

下面我们将分步举例说明如何创建 COM+ 应用程序。

### 1. 创建新的 COM+ 应用程序

在为组件编写所有代码后，下一步就是用 VB 将组件编译成 DLL 文件。这是组件的可执行版本，并可在任何需要的时候由 COM+ 子系统载入。如果代码没有错误，VB 就会编译此组件并向操作系统注册。

如果已经有第 14 章中所讨论的组件的独立版本，建议先不要注册，以防止即潜在的冲突。

下一步是创建能为 Web 应用程序提供 COM+ 服务的 COM+ 应用程序。要创建它，第一步是启动 Microsoft Component Services Explorer，如图 15-13 所示。其快捷方式在 Windows 2000 的 Programs/ Administrative Tools 菜单中。



图 15-13 Microsoft Component Services Explorer

下面在计算机中创建一个新的 COM+ 应用程序。

要创建一个新的 COM+ 应用程序，展开文件夹直到看到 COM+ Application 文件夹，然后选中它并右击，最后在快捷菜单中选定 New|Application。这就启动了 COM Application Installation Wizard，通过它可以创建 COM+ 应用程序，如图 15-14 所示。

创建一个新的应用程序时，应该选择 Create an empty application 选项，它会在向导中显示下一步，即要求配置应用程序，如图 15-15 所示。如果事先已经创建并导出了一个 COM+ 应用程序，而且希望将它安装到机器上，就可以选择 Install pre-built application(s) 选项。

在这个例子中，可以将应用程序命名为“Books”。还要决定它是服务器应用程序还是库应用程序。缺省项是服务器应用程序，这里不做改变。

下一步是设置执行应用程序的用户标识，如图 15-16 所示。

我们可为此应用程序选择确定的用户，这样每个使用这一应用程序的客户都将让此应用程序代表此用户。在这个例子和多数由 ASP 使用的 COM+ 应用程序中，选择 Interactive user 选

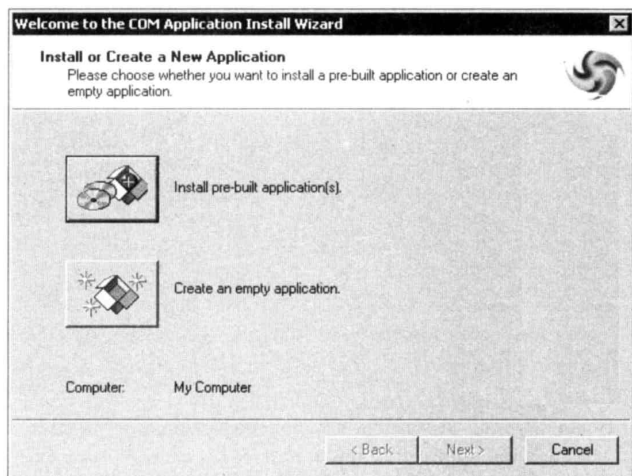


图15-14 COM Application Installation Wizard界面

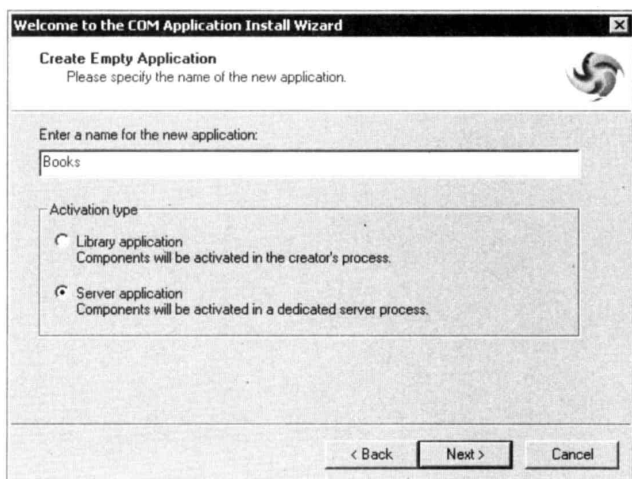


图15-15 选择Create an empty application选项后的界面

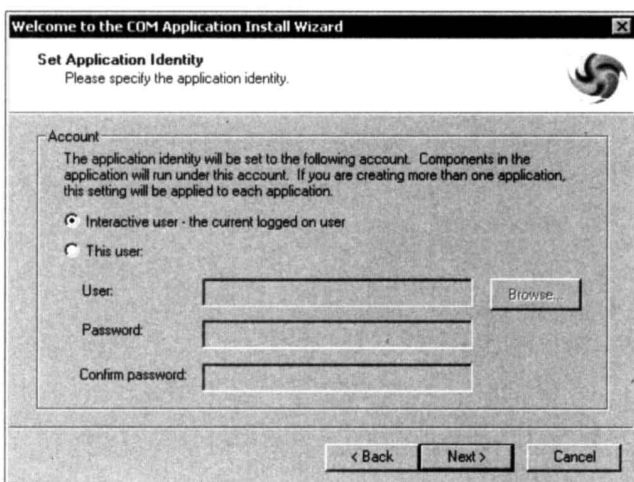


图15-16 设置执行应用程序的用户标识的界面

项。这就意味着进入的用户身份将为组件所使用。对于一个 ASP 应用程序而言,将使用 IUSR\_computername 用户。

完成此步骤后,一个新的 COM+ 应用程序就创建好了,如图 15-17 所示。

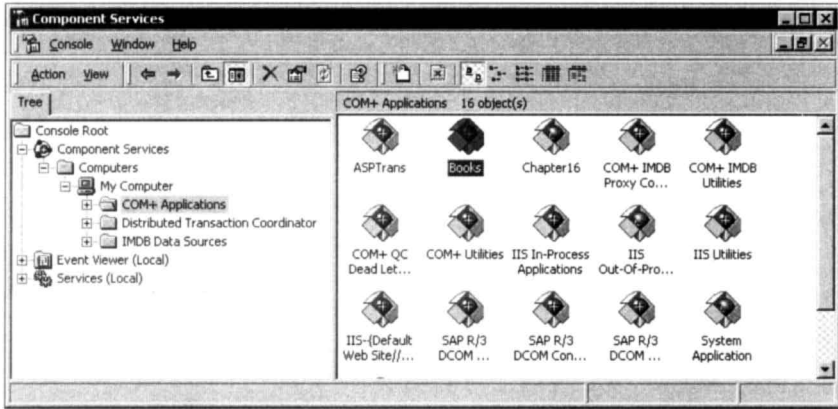


图15-17 创建Books COM+应用程序后的Component Services Explorer界面

## 2. 向COM+注册组件

可以继续向应用程序添加组件。为此可先打开此应用程序的文件夹以显示 Components 和 Roles 文件夹。选择 Components 文件夹,在这儿显示安装在此应用程序中的所有组件(目前,这里为空)。右击 Components 文件夹,并在快捷菜单中选择 New|component,就会显示 Component Install Wizard,如图 15-18 所示。

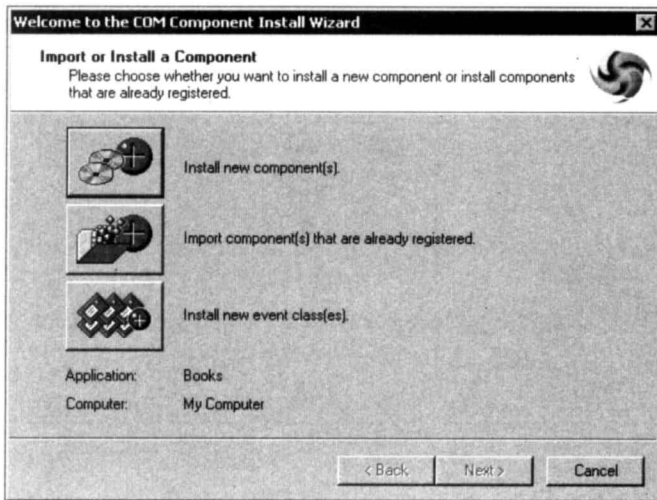


图15-18 注册组件界面1

这一向导允许在 COM+ 应用程序中安装组件或事件类。

COM+ 事件是对 COM+ 应用程序中支持后期绑定事件的 COM+ 编程模式的扩展。不必反复查询服务器,一旦有重大问题的发生,事件系统就会通知正在监听的应用程序。我们不研究与 ASP 协同工作的事件类。



因为在编译时VB已经注册了DLL，我们选择第二个选项，显示下一屏如图15-19所示。

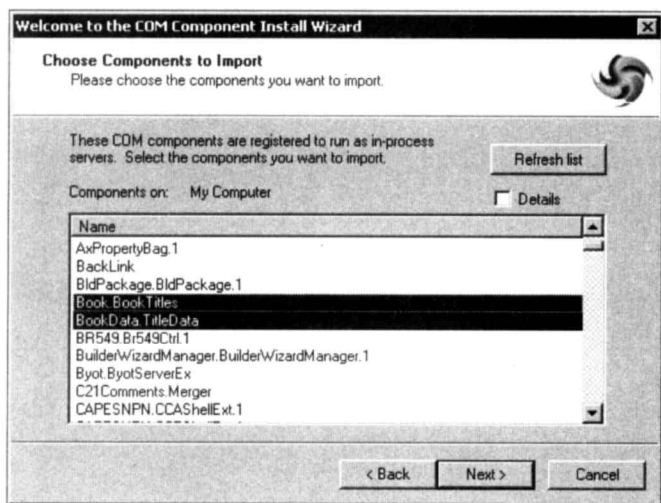


图15-19 注册组件界面2

这会显示已注册到系统但还没有成为 COM+应用程序的一部分的所有组件。通过选择 Details选项，还可看到组件的文件名和 CLSID。拖动此列表直到发现我们刚创建的组件。这个向导允许同时添加所需的多个组件。点击 Next就可以向COM+应用添加这两个组件，结果如图15-20所示。

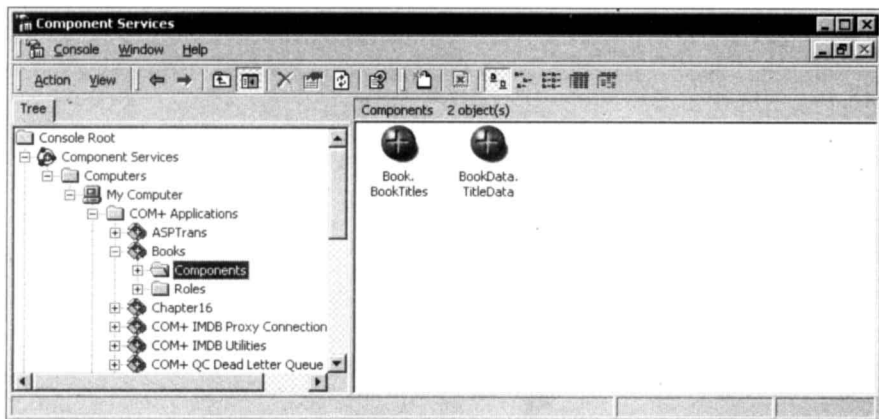


图15-20 注册组件界面3

### 3. 设置组件属性

只要在COM+应用程序中安装了组件，就可以修改这些组件的属性。要显示组件的属性页面，只需选中组件，然后右击，选择 Properties，如图15-21所示。

我们在这一章将看到的设置位于 Activation选项卡中。

在第19章中我们将使用 Transactions和Concurrency选项卡中的设置。

Activation选项卡包含如何创建和激活组件的信息，如图15-22所示。

因为我们是使用VB创建组件，因此无法对此组件使用组件缓冲。因此， Enable object

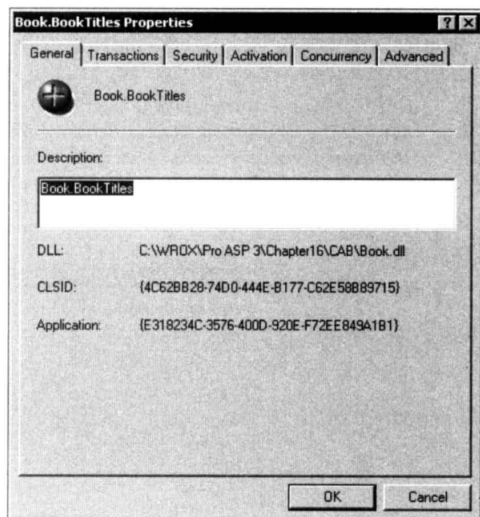


图15-21 设置组件属性的界面 1

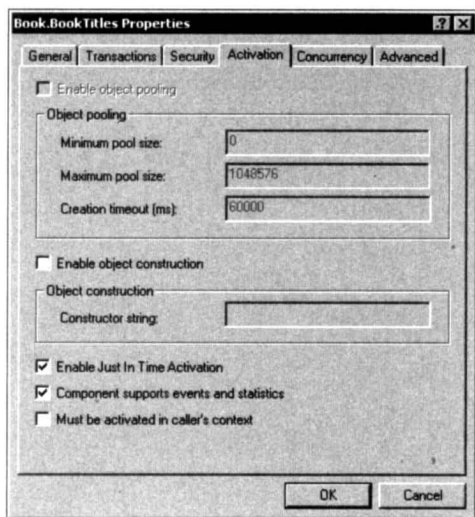


图15-22 设置组件属性的界面 2

pooling复选框是禁用的。Enable object construction允许设置一个将在对象构建时传递到对象的字符串。它可以用来传入一个数据库连接字符串，并成为对象的一部分。在我们的应用程序中将其传送到各个方法中去。因为对于所有对象这个值都是一样的，它就不会像传统的属性值那样使应用程序有状态。

通过选中Enable Just In Time Activation框，可以确保COM+可根据需要自动激活组件或使之失效。当然，对于一个VB组件来说，失效意味着取消。对于创建组件来说，JIT激活意味着对象只在需要时才开始消耗资源。

#### 4. 部署COM+应用程序

正如在第13章中提到的那样，可以将COM+应用程序在系统之间进行移动，这需要三个步骤：

- 第一，把它从当前位置以MSI文件形式导出。
- 第二，将此程序包移至目标机器。
- 第三，将此程序包安装到目标机器上。

可使用Application Export Wizard导出COM+应用程序。右击COM+应用程序，选择Export即可启动它，如图15-23所示。

我们可以导出服务器应用程序或应用程序代理。通过导出服务器应用程序，可在另一台机器上安装这个应用程序并在那里运行组件。反之，可以只导出一个应用程序代理，它在另一台机器上安装后将允许远程机器上的客户使用处于本地机上的组件。

#### 5. 安装导出的应用程序

我们已创建了MSI文件，可把它移到目标机器上，并将它安装在那里。应用程序的安装使用的是与创建新应用程序相同的方法。这次选择Install pre-built application(s)，而不是Create an empty application。从向导中选择这个选项时，就会出现一个对话框要求选定一个MSI文件进行安装，见图15-24。选定后，就会显示此文件中包含的COM+应用程序的列表。

在选择应用程序的身份之后，将会询问在何处安装此文件。MSI文件可能含有可使用的路径信息，如果没有就要指定一个明确的目录来安装此文件，如图15-25所示。

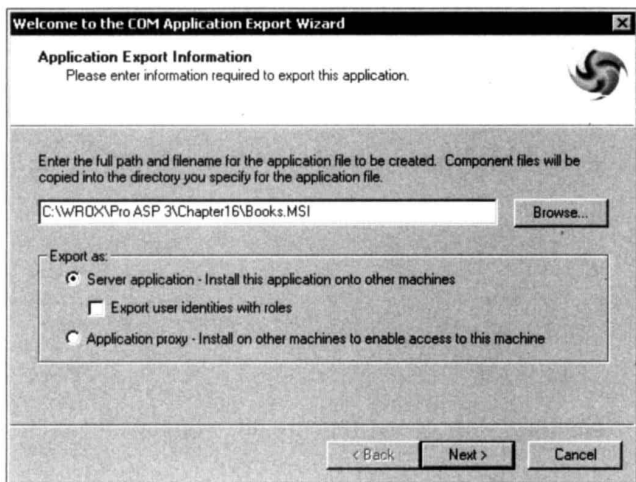


图15-23 Application Export Wizard 的界面

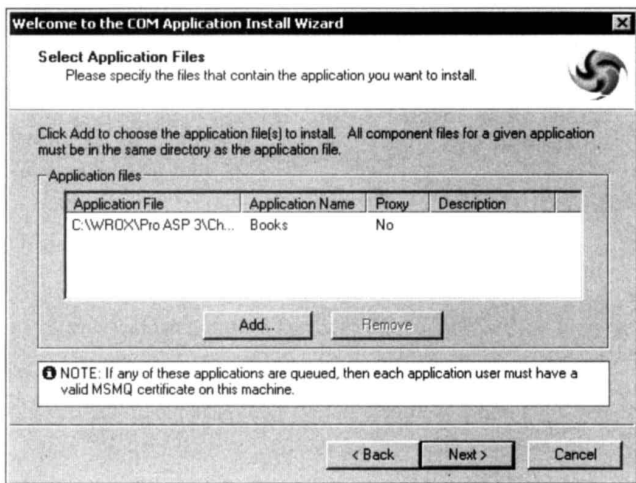


图15-24 选定MSI文件的对话框

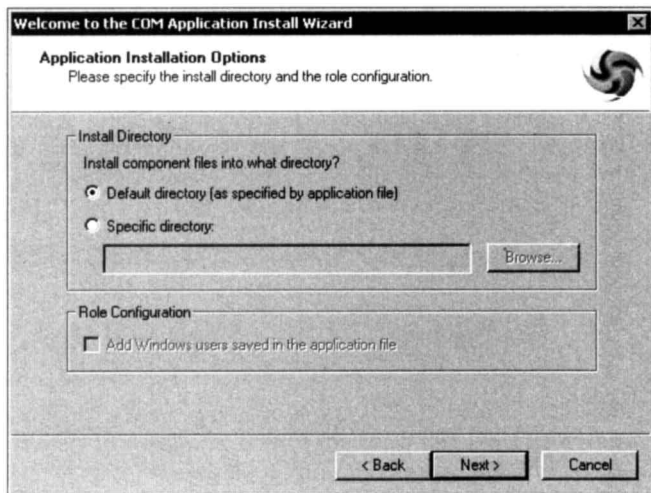


图15-25 选择MSI文件的安装路径的对话框

单击对话框中的Next, MSI文件就将安装到指定目录中去, COM+应用程序将创建在系统上。然后就可以像在原来的计算机上一样使用这个应用程序了。

## 15.5 组件的调试

和开发组件一样, 同样也要考虑到可能产生的错误。所以要通过一些方式找出错误所在的位置, 但这又不像 ASP脚本, 组件在编译后就很难查出错误是在哪儿出现的。

如果是使用VB 6.0进行开发, 就可以创建一个ASP脚本来测试组件。这样可轻松地在开发和测试环境间进行切换。

为了达到此目的, 打开组件的 Project Properties对话框, 选择Debugging选项卡, 如图15-26所示。

选择Start browser with URL并输入测试ASP文件的URL。如果想让VB在测试组件时启动一个新的浏览器实例, 那就不选择 Use existing browser选项。

现在可以在代码中设置断点并运行这个项目, 这就会以指定的URL启动浏览器, 但是当脚本对组件进行调用时, 可以用VB的调试器中断这些调用, 并正常地调试项目。

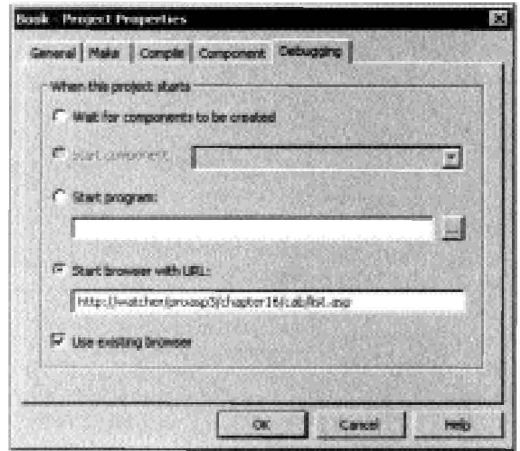


图15-26 Project Properties对话框的 Debugging选项卡

### 解锁DLL

当向内存中载入组件以便在 ASP脚本中执行时, 系统会锁定 DLL文件。这能防止在载入组件时删除或改变这个文件。测试过组件后, 需要确保组件实例确实已从内存中移出。如果在重新编译前不这么做, VB会给出错误信息, 组件也不会被编译, 如图15-27所示。

确保组件从内存移出的最简便的方法是确保应用程序运行在独立的内存空间内。这一步可通过在Application Protection下拉列表框中选择High(Isolated)实现。然后在IIS应用程序的Properties对话框的Virtual Directory选项卡单击Unload按钮, 如图15-28所示。

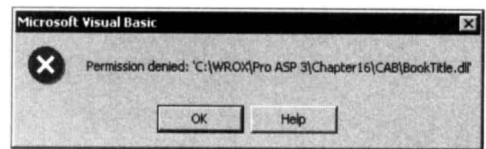


图15-27 VB给出的错误信息

此按钮仅在将COM+应用程序作为服务器应用程序创建时才有效。因为库应用程序将在IIS的进程空间内运行, 所以不能将其单独从内存中卸载。

如果用上述步骤卸载应用程序后仍不能重新编译, 下一步要做的就是使用 Component Services Explorer关闭COM+应用程序。这样可以使之从内存中释放, 如图15-29所示。

如果右击COM+应用程序名, 然后从快捷菜单中选择Shut down, 应用程序(dllhost.exe)就会从内存中卸载。

如果这样还不见效, 那么COM+应用程序中一定保存了其他内容。

Visual InterDev 6会保存对象的应用程序, 以提供Intellisense编辑能力, 这样就应关闭VI



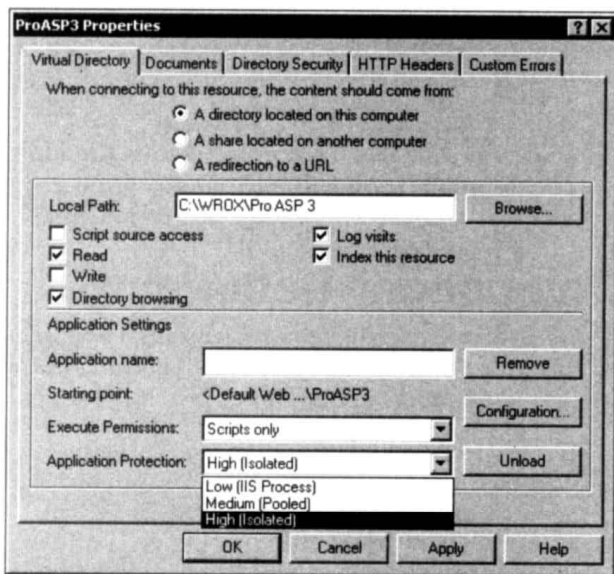


图15-28 在Application Protection下拉列表框中选择 High ( Isolated )

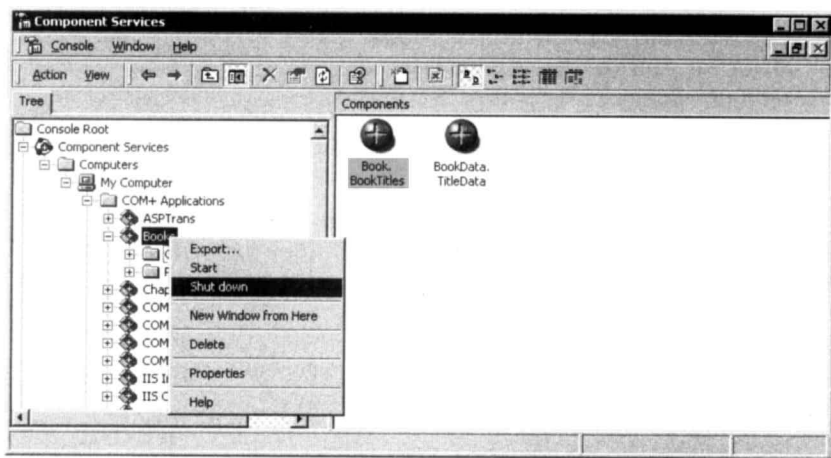


图15-29 关闭COM+应用程序

来释放此引用。

释放组件的最后一招是关闭并重新启动 Web 服务器，在命令提示下输入以下两条命令：

- 键入 `net stop iisadmin /y`。这将关闭 IIS Admin(即 IIS 的父服务)，同样也会关闭 FTP、SMTP 和其他 IIS Admin 的子服务。还会将 `inetinfo.exe` 进程从内存中卸载。假如只键入 `net stop w3svc` 来卸载 Web 服务器，将不卸载 `inetinfo.exe`。
- 然后，键入 `net start w3svc`。这样试着重启 Web 服务器，因为运行需要 IIS Admin，那么 IIS Admin 也会自动重新启动。注意如果要访问 FTP 或 SMTP 服务项，也要手动重启它们。

我还发现，有时停止 Protected Storage 服务也会起作用。

如果上述方法都不行，只能重新启动系统。



## 15.6 小结

本章研究了如何运用 COM+ 和 Microsoft Component Services Explorer 创建 Web 应用程序。它们使开发者不必担心组件如何与操作系统交互。这让我们能集中精力处理对应用程序很重要的业务逻辑支持，并确保 COM+ 一直在幕后支持这一应用程序。

运用 VB 之类的组件创建工具，可以很轻松地将应用程序的功能封装到可重复使用的组件中，包括业务、表现和数据访问逻辑等。然后将组件放置在 COM+ 应用程序的范围中，就可让操作系统在应用程序使用组件时进行相应的管理。

下一章将讨论如何用 C++ 改善组件的功能和性能。