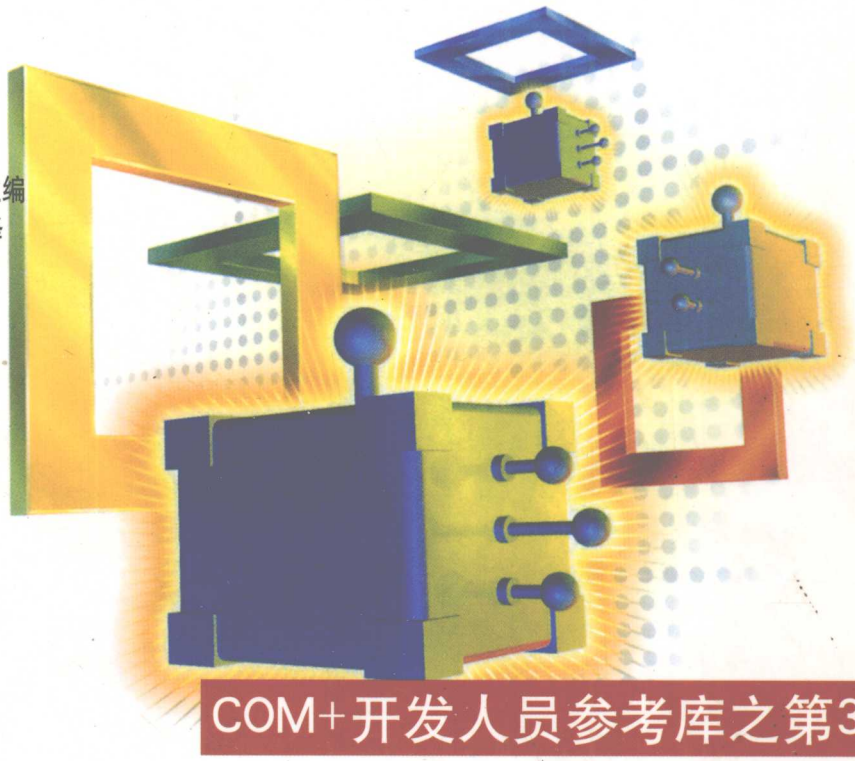
HZ BOOKS

微软公司核心技术书库

Microsoft

Windows

COM+ Developer's Reference Library, Volume 3:
COM Fundamentals



(美) David Iseminger 主编
宋丽红 叶小松 等译

COM+开发人员参考库之第3卷

COM技术

机械工业出版社
China Machine Press

微软公司核心技术书库

COM技术

COM+开发人员参考库之第3卷

（美） David Iseminger 主编

宋丽红 叶小松 等译

前导工作室 审校

机械工业出版社
China Machine Press

本书是“COM+开发人员参考库”的第3卷。全面提供了COM编程的指导和应用程序开发的资料。组件对象模型（COM）是平台无关的、分布式、面向对象系统，它使软件组织能跨进程和跨机器进行互操作。

本书适合Windows应用程序开发人员阅读。

本书版权登记号：图字：01-2000-4310

图书在版编目（CIP）数据

COM+开发人员参考库 第3卷 COM技术/（美）艾塞明格（Iseminger, D.）主编；宋丽红等译. －北京：机械工业出版社，2002.1

（微软公司核心技术书库）

书名原文：COM+ Developer's Reference Library, Volume 3 : COM Fundamentals

ISBN 7-111-09362-3

Ⅰ.C… Ⅱ.①艾…②宋… Ⅲ.软件接口，COM+－程序设计 Ⅳ.TP311.52

中国版本图书馆CIP数据核字（2001）第066120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：冯晓陆

北京第二外国语学院印刷厂印刷·新华书店北京发行所发行

2002年1月第1版第1次印刷

787mm × 1092mm 1/16 · 36.25印张

印数：0 001–4 000册

定价：68.00元（全套300元）

译 者 序

时下大概没有多少人不知道COM+技术。在热闹的BBS、纷飞的邮件和恬适的技术沙龙里充斥着COM这个字眼。有关COM/COM+的专业站点、专业刊物和探讨文章虽不能说多如牛毛，但形容为雨后春笋还是贴切的。如果你曾经将Microsoft Excel电子表格嵌入到Microsoft Word文档，使用过Web页面的ActiveX控件或Microsoft Visual Basic应用程序，或使用ADO检索过数据库中的数据；那么，你也一定使用过COM。这是因为：COM是Microsoft的基于组件的软件解决方案的基础。COM为基于组件的开发方式提供了简单、一致的模型，实践已经证明了这一点。

COM+实际上是COM的新版本，或者说是COM更高层次上的应用。COM+的底层结构仍然以COM为基础。但需要注意的是，COM+不再局限于COM的组件技术，它更加注重于分布式网络应用的设计和实现，已经成为Microsoft系统平台策略和软件发展策略的一部分。COM+继承了COM的几乎全部优势，同时避免了COM实现方面的不足。COM+紧密地与操作系统结合起来，通过系统服务为应用程序提供全面的服务。Windows 2000操作系统是COM+的第一个实现版本，在Windows 2000中，每个新的子系统都是作为一个COM+对象实现的。

本书是“COM+开发人员参考库”套书的第3卷。顾名思义，这套书解释的是COM/COM+技术。但重点并不是COM或COM+规范说明，而旨在为广大程序员提供这些领先技术的最可靠、最详细的编程参考。COM+继COM（组件对象模型）之后推出，但并不只是COM的升级。COM+以COM、DCOM（分布式组件对象模型）、MTS（微软事务服务器）和MSMQ（微软消息队列）为基础，继承了四者的关键特性，但在面向分布式企业应用的基础上，添加了许多新的服务。

在进行项目开发时，你会发现项目组的成员各怀绝技，有人擅长Visual C++，有人专攻Visual Basic，还有的人钟情于Delphi。为他们指定一门共同的开发语言显然是不可能的，那意味着开发周期的无限延长和人力资源的巨大浪费。所以，就有必要用一种办法，把他们各自的工作成果在项目中进行整合。这不仅仅是开发模块的分配、重组和联调，而是在二进制级别的兼容并蓄。

市场和用户需求的改变催促着软件疯狂地升级，而程序员越来越沉陷于无休止的升级、修补、扩展，此时COM+无疑给他们带来了新的希望。当然，我们可以采用DLL解决软件局部更新的问题，而只需要更新软件相应的DLL文件中的实现函数。遗憾的是，DLL是面向应用程序的。而COM为Windows操作系统提供了统一的、面向对象的、可扩充的数据交换协议。COM的一个至关重要的特征是，它规定的接口与编程语言无关，也就是定义了二进制代码的标准。这使得组件不论用什么语言编写，只要它们符合COM规范，就能够进行相互间的通信。DCOM规范提供了分布式计算条件下的组件重用可能性。

相信本套书的出版，可以在很大程度上弥补国内有关COM/COM+技术资料的空白。本套书编写体例独特，作者充分考虑了读者的实际需求，既可以作为COM/COM+的学习指南，也可以

作为编程时的参考。

本书由宋丽红、叶小松、钟大军、肖健、白云、蒋进生、俞涓、李楹、乔媛、龚立占、夏双新、宋玉双、尹昕、刘长岭、叶金锁、孙晓晖、陈飞、李香云、黄来增、龚振权、宋丽娜、刘桂清、华峰、童国林、李颖、刘小朋、张景、杨力、田利、刘风歧、于美、杨大伟、史克、王红明等进行翻译，并由宋丽红统稿。前导工作室全体工作人员为本书的翻译、审校、录排、校对付出了大量劳动。在此向他们表示诚挚的感谢！

由于时间仓促，且译者的水平有限，书中难免会出现一些错误或不当之处，恳请读者批评指正。如果你在阅读本书中遇到什么问题，或者对本书有什么意见和观点，请同前导工作室联系：qiandao@263.net。我们会尽力帮助你解决问题。

目　录

第三部分 COM 参 考

第一部分 概 述

本部分旨在介绍本卷其他各部分的内容并提供其他各卷的概述信息，以便于你查阅本卷及套书“COM+开发人员参考库”。

第1章 欢迎使用“COM+开发人员参考库”

如今，我们中的大部分人都已在“互联网时代”工作了至少几年。白天的时间从来都不够用，要做的工作从未完成，加快工作节奏所带来的利益从未如此显著。在互联网时代，努力更多、效率更高意味着进展更大、完成更快，而这些代表了更多——金钱、问世时间、知名度、市场份额或者悠长的舒适生活。有些事情不能一蹴而就，例如矫牙，通常不能仓促完成；然而在应用程序或者Web开发领域，由于前面提到的所有原因，互联网时代意味着可以而且必须迅速进行开发。在这个领域，COM+理解你所处的困境和所受的压力，创建COM+正是为了使你的开发工作更轻松、更迅速。

为帮助你了解有关COM+的必要信息及其COM基础，我编写了“COM+开发人员参考库”套书，它是有关COM和COM+编程指南、参考材料以及密切相关的或者新兴技术（如MIDL、ActiveX、自动化和结构化存储）的一套必备资料。

“COM+开发人员参考库”套书是关于COM+参考材料的完整参考集，包括COM+编程指南和全部参考材料。因此无论对使用COM+创建组件而言还是对创建工具而言，本套书都是“一站式”参考资料。由于COM+是对COM基础的补充，因此本套书也包括完整的传统COM编程指南和参考资料。

本套书的脉络很清晰。它完全围绕COM+主题编写。编写这套书旨在传达Windows编程中关于该主题的最完整、最权威以及最易读的可用参考信息，同时又不失侧重点。本套书中的每卷都论及一组逻辑相关的技术或者开发事项。这种编写方式经过特别安排，以使你能够快速、高效而直观地查到所需信息。

除了COM+开发信息，“COM+开发人员参考库”还包括提示，它们是为使开发工作更轻松而编写的。例如，其中包括微软开发者网络在线的完整解释和详细叙述，这很大程度上有助于你摆脱微软开发者网络预订。如果你没有预订微软开发者网络，或者不知道为什么应该预订，书中也有相关信息,包括三种级别微软开发者网络预订方式之间的不同、每一种订阅方式所提供的内容，以及可通过互联网访问微软开发者网络在线时为何需要预订。

1.1　“COM+开发人员参考库”的组织方式

“COM+开发人员参考库”套书由五卷组成，每一卷讲述的编程重点或者直接与COM+相关，或者与COM及其功能相关。这五卷指南和编程参考分述如下：

第1卷	COM+程序员指南
第2卷	COM+编程参考
第3卷	COM技术
第4卷	COM+自动化编程
第5卷	COM+结构化存储与ActiveX

将“COM+开发人员参考库”分为这五卷，使读者能够根据任务快速确定所需内容，并且有利于保持对该任务的侧重。

在第1卷中有一张微软开发者网络快照DVD。

1.2　本书的组织方式

在“COM+开发人员参考库”中，每卷都遵循通用规则划分章节，如下列标识：

第一部分：概述

第二部分：指南、示例及编程参考

本卷的格式与此稍微不同，这是为了以最清晰、最精确的方式表达其内容。

本卷最终分为如下几部分：

第一部分：概述

第二部分：COM编程基础

第三部分：COM参考

这里并没有很大的出入：第一部分提供概述，其余部分提供选定的参考或者概述，并对其巧妙分组。

本套书的编写方式

“COM+开发人员参考库”套书的编写方式是为了以最易读的方式传达最适当的信息。“COM+开发人员参考库”提供的外观和特性与以电子方式发布Microsoft参考信息的外观和特性一致，从而达到和微软开发者网络在线无缝集成的编写目的。换句话，本书中的给定函数参考的表示方式经过特别编写，以模仿MSDN和微软开发者网络在线中函数参考页的表示方式。

保持这种集成性的理由很简单：使你（Windows应用程序开发人员）更容易地使用这些工具，更方便地获取要创建高质量程序所需的进行时信息。通过在参考资料中提供“公用界面”，如果你熟悉“COM+开发人员参考库”参考材料，则可以立即将经验应用到微软开发者网络或者微软开发者网络在线，反之亦然。简言之，这意味着一致性。

第2章 充分利用微软开发人员资源之三

当你深入查询MSDN和MSDN在线帮助提供的接口，使用MSDN的搜索引擎进行搜索时，就开始了软件开发的工作，许多开发人员频繁查阅MSDN。但有些时候，有很多的图解、很多的选项，你却没有时间去深入理解帮助的信息。这样一来，得到你所需的信息几乎是不可能的。

很容易忘记那些能提供你所需信息的图解、入口和它在MSDN中的位置，所以本章旨在帮助你剥去其他的信息，并给你一个小例子，告诉你在该信息里面所有的内容。

接下来的一节来自于微软合作组的Brian Sabino，这一节的目的是为开发者提供如何在COM中编写异步调用的方法。

非阻断的方法(异步方法)调用

非阻断的方法调用易于使用，有许多的有趣的应用程序，它将尤其是在客户端得到普及，在客户端生成非阻断方法调用比一般的标准调用仅需要多做一点工作。开发人员必须注意到使用这种COM的特性能使你的服务器端程序的可伸缩性得到了改善。

2.1 介绍

随着非阻断的方法调用的引入，微软为开发者提供了一种新的强有力的工具，组件对象模型开发模式。客户程序使用非阻断方法调用来开发并行的程序，而不用为处理棘手的多线程的执行方式，在服务器端，服务可以异步处理这些调用，从而可大大提高系统的可伸缩性。

请考虑下面对于ISample ::Sun的标准调用：

```
int sum;
pISample->Sum(4,5,&sum);
```

一个客户程序以非阻断调用模式去调用异步接口AsynclSample的两个方法，去实现同样的调用：

```
int sum;
...
pAsyncISample->Begin_Sum(4,5);

//Client continues execution.

pAsyncISample->Finish_Sum(&sum);
```

Begin方法将外部参数传送到服务器端。当服务器端处理这个调用的时候，客户端不必等待，可以自由地去做其他工作。客户端调用Finish方法去结束本次调用，并取得结果。

服务器端程序可以通过异步调用接口的实现有选择地异步处理这些调用。用这种方法的目

的将在本章的第二部分深入的讨论，但从本质上讲，处理异步调用可以使服务器端具有更大的可伸缩性。

这种新的异步调用的优点在于客户程序和服务器在处理这些同步或异步调用的过程中完全独立。也就是说，客户程序能够使用异步的调用服务器端完全同步的方法。同样的，客户程序通过异步调用的方法去调用服务器端程序的异步处理也是有可能的。具体细节的问题将在本篇文档的第三部分“特性”的一节展开，但现在有一点值得一提的是客户端程序和服务器端程序对于同步调用要相对独立。

关于这篇文档

这篇文档旨是全面地介绍非阻断方法调用。在这里我们假定读者熟悉COM和分布式COM（DCOM）的技术。如果你不是一个开发人员，只是想找到关于这个主题的介绍，，你最好是先读一下第二部分和第三部分的开篇。

以下是本篇文档用到的术语：

client	应用程序或者是一个调用COM组件的COM对象。
asynchronous client	使用非阻断的调用客户程序。
server	接收来自客户程序调用的一个COM对象。
asynchronous server	异步地处理输入调用的服务器。
asynchronous calls	在这篇文档中仅指非阻断方法调用。

2.2　动机

由于客户端程序使用非阻断调用是独立于服务器端处理这些调用的，考虑一下使异步的客户程序和异步服务器的分开动机是很重要的。首先，这篇文章描述了许多的情况，在这些情况下客户端使用非阻断调用的动机是非常清楚的。其次，它也描述了许多适合调用异步服务器的特定情况。

2.2.1　异步客户程序

非阻断的方法调用为开发者在使用多线程时提供了许多的权力和灵活性，而无需考虑线程和同步的问题。许多有趣的应用程序使用异步方法调用，它可以设置信号量，为COM组件方法的调用定制超时。把这些特点综合起来考虑，包括轻松的定制和添加对非阻断方法的调用，你可以想象到非阻断的方法调用在客户端应用程序中将得到广泛的调用。

1. 低速/多重服务器

非阻断的方法调用在开发者传统上调用多线程的很多情况下是非常有用的。例如，在你使用标准调用方法去并行的调用多个服务器的方法时，你必须要创建多线程，每一个线程执行一次调用。这样做会使你在考虑参数传递、同步问题和设置信号量时非常麻烦。然而，在这种情况下，使用非阻断的调用是很理想的解决方法。使用非阻断方法调用，客户程序可以平行地调用许多方法，而不用担心多线程的问题。

另一个使用非阻断方法调用的明确动机是可以早些开始调用那些需要长时间才能完成的方法。客户程序使用非阻断方法调用可以尽可能早地调用那些需要长时间才能完成的方法，并且客户程序可以继续自己的工作而不用担心线程的问题。非阻断的方法调用在单线程的运行环境中能使许多操作并行地进行，这样能使应用程序的性能有很大的提高。

2. 发信号

非阻断方法调用也可被用作信号控制机制。例如这样一个情况：客户程序需要周期性检查服务器端程序中某个特定的事件是否发生。服务器提供一个Listen（）监听方法，它只有在事件发生的时候，才会返回。客户端程序可以在开始的时候调用异步方法：Begin_Listen()，然后去做其他的工作。当客户端程序需要知道那个特定事件的状态时，它去检查一下异步调用是否返回；检查异步方法调用是否返回的操作对于客户程序来说是完全在本地进行的，执行速度非常快。也就是说，当客户程序检查异步调用是否返回时，这种检查操作是瞬间完成的，没有数据的传送，是不需要等待的。使用这种非阻断的调用能使客户端从本质上可以自由地轮询服务器端事件的状态。

与前面提到的那种情况有些不同的情况是，当服务器的某个事件发生后客户程序需要一个回调信号。通常情况下，这需要两次往返的调用。首先，客户程序要调用服务器的某个方法去为这个事件注册，同时要提供一个回调接口的指针。其次，服务器在事件发生的时候，要调用这个接口。而使用非阻断方法的调用，只需要单程的调用。这将使用我们要在第三部分讨论的一项技术，客户程序聚合这个回调对象，以便方法执行结束时接收到一个回调信息。客户程序还是像前面一样调用Begin_Listen()方法。当服务器端事件发生时，服务器Listen（）方法将返回，客户程序就可以收到这个回调的信息。

发信号控制机制的另外一个用途被称作“激发并忘记”，假定这样一种情况，客户程序要快速地通知某一事件给多台服务器，而对任何的回应都不感兴趣。客户程序可以向服务器发送异步调用，然后“忘记”结束这些调用，就调用了该对象的Release()方法，这个问题我们将在本篇文档的第三部分讨论。因为调用对象在释放掉后没有办法去检查调用方法的错误，“激发并忘记”这种模式在调用失败后没有任何的错误回应。然而，这仍然是一种非常高效的传送大量信息的方法。同样，服务器也可用“激发并忘记”这种模式去将某事件的状态通报给许多的客户程序。

3. 定制超时

另外一个非阻断方法调用应用的强有力案例是实现定制超时。客户程序可以在任何时候取消未完成的调用，而不会造成资源的泄漏。客户程序可以开始一个异步调用然后定期的去检查该调用是否结束。当客户端程序觉得这次异步调用的时间太长时，可以取消这次调用操作，并调用Finish方法去释放所有资源。

2.2.2 其他应用

在许多的情况下，非阻断的方法调用对于客户程序非常有用。例如，非阻断方法调用可在COM管道的实现中应用，实现数据传送时，读数据在先，写数据在后。

1. 异步服务

当多线程服务器的可伸缩性作为一个问题要考虑的时候，应用异步服务的方式是适当的。当有许多的客户程序时，它可以节省线程的使用，它还可以使服务器对正在执行的线程进行细粒度的控制。

2. 标准的多线程服务器

当RPC（远程过程调用）到达一个多线程的服务器时，RPC实时地从RPC线程池中挑选一个线程。这个线程接着调用RPC通道，它可以直接访问服务器的占位对象。通过stub，被调用的方法被执行并返回占位对象，最后返回到RPC通道，如图2-1所示。

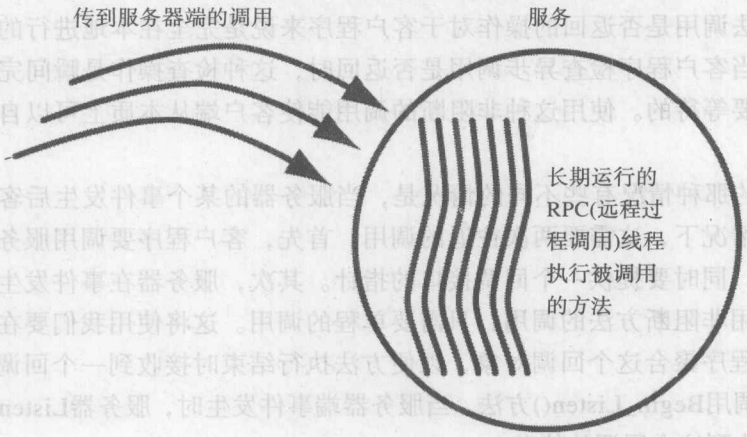


图2-1　标准的多线程服务

如果你不是同时运行的许多客户程序或进行由于复杂性和加锁机制而导致的长时间调用，这种体系结构的可伸缩性还可以。例如，假想一下，你有200个客户程序同时地执行调用。这意味着你有200个从RPC线程池中挑选的线程同时试图在服务器端执行。从性能的角度看，如果服务器大量的使用加锁并且调用的方法需要很长时间才能完成，这种情况将是灾难性的。

3. 异步多线程服务器

在一个异步多线程服务器中，在RPC线程执行之前线程就开始执行。这个线程访问占位对象，然后调用异步接口相应的Begin方法。继续第一部分中的例子，这将调用AsynclSample接口的Begin_Sum方法，该方法不是立刻开始执行调用，这种Begin方法是将客户的参数打包存储在某个结构中，并把这个数据结构放到一个队列中，至此，Begin方法返回，RPC线程也返回到RPC线程池。然后，服务器有少数的一些工作线程把这些调用从队列中取出来，并执行它们。当工作线程工作结束时，它将运行的结果放回结构体并为调用完毕设置标志位。线程继续执行相应的Finish方法。Finish方法将得到工作线程的运行结果，并且将这些结果送还给客户程序，如图2-2所示。

这种方法提供比标准的多线程服务有两个明显的优点。首先，RPC的线程个数守恒。线程不是执行整个的方法，而是通过Begin方法快速执行，剩下的时间大都在自由线程池内执行。第二

个优点是这种细粒度对线程控制的方法高于多道程序设计的层次。在许多的情况下，服务器有许多内部的同步问题，有少量的工作线程做一些固定的工作要比大量的线程把它们的时间浪费在阻塞状态要好。

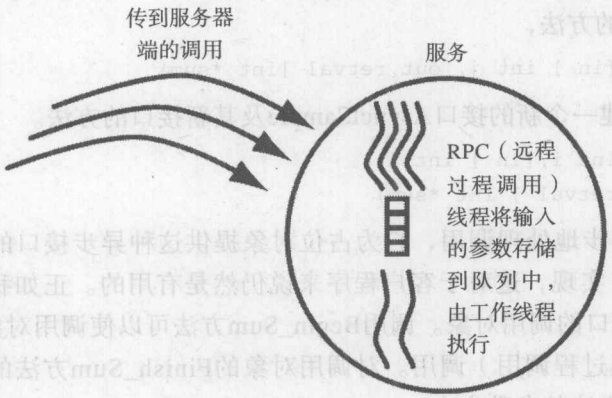


图2-2　异步的多线程服务器

2.3　体系结构

2.3.1　Async_uuid和具有非阻断方法调用能力的代理/占位对象

理解非阻断方法调用的关键是异步和同步的调用看上去都是同时执行的。换一种角度来看，服务器接收到一个调用，它没有办法判定在客户端程序是以同步方式还是异步方式初始化这个方法的。类似的，客户端程序也不知道它的调用在服务器中是同步还是异步地执行。为了提供这种新功能，组件对象模型（COM）给你机会去创建代理或占位对象，它们能进行非阻断的方法调用。这些代理或占位对象将提供能使服务器端和客户端完全独立处理它们的异步或同步调用所需必要的基础结构，如图2-3所示。

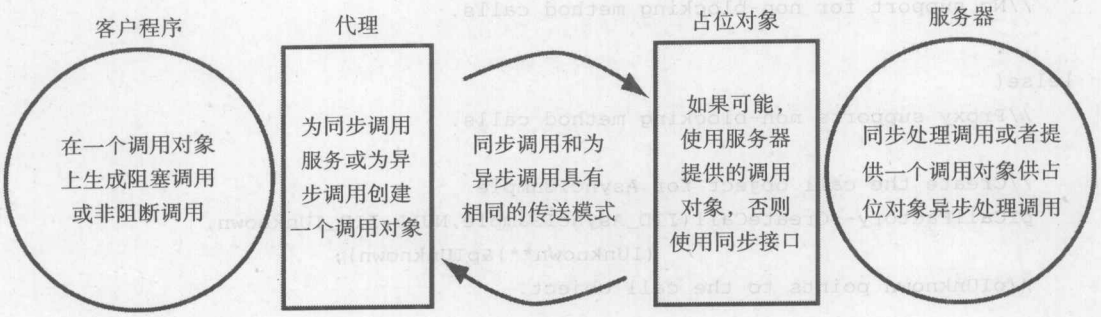


图2-3　代理或占位对象的基础结构，为了使服务器和客户程序在同步或异步地处理调用时互不影响

创建这些代理或占位对象就像在服务对象的IDL（接口描述语言）文件中增加一个async_uuid关键字一样简单。这需要在原有接口的基础上去做，所以一个支持多个接口的对象，如果有某个接口支持非阻断的调用，应指定是哪个接口。增加async_uuid关键字将告诉MIDL编译器编译时，为标识的接口创建一个接口的异步版本。例如，一个ISample接口提供一个带有async_uuid关键字标识的方法，

```
Sum([in ] int i,[in ] int j,[out,retval ]int *sum)
```

则MIDL将为它创建一个新的接口AsynclSample及其新接口的方法。

```
Begin_Sum([in ] int i,[in ] int j)
Finish_Sum([out,retval ] int *sum)
```

如果服务器希望异步地处理调用，它为占位对象提供这种异步接口的实现。不管服务器是否提供这种异步的接口实现，这对于客户程序来说仍然是有用的。正如我们所看到的，代理将提供一个实现了异步接口的调用对象。调用Begin_Sum方法可以使调用对象发送一个对Sum方法的标准异步RPC（远程过程调用）调用。对调用对象的Finish_Sum方法的调用可使远程过程调度的结果复制到Finish方法的参数中去。

2.3.2 客户端

1. 创建调用对象

从客户端开始执行异步调用时，先查询ICallFactory接口。通过查询，客户程序能够确定代理是否支持异步调用。如果在服务器的接口中，有一个接口有async_uuid关键字的标识，这次调用就可以成功执行，客户端程序将得到ICallFactory接口的指针。接下来，客户程序调用ICallFactory :: CreateCall方法，创建一个特定的异步接口，用来获得代理提供的调用对象。下面的代码将以实例说明调用对象的创建过程。

```
//Get ICallFactory from the Server
hr =pSimpleObj->QueryInterface(IID_ICallFactory,(void **)&pICallFactory);

if(FAILED(hr)){
   //No support for non-blocking method calls.
   ...
}else{
   //Proxy supports non-blocking method calls.

   //Create the call object for AsyncISample
   pICallFactory->CreateCall(IID_AsyncISample,NULL,IID_IUnknown,
                             (IUnknown**)&pIUnknown);
   //pIUnknown points to the call object.
...
```

调用的对象是客户端程序开始、监控、完成异步调用的参考点。调用对象为一个特定的异步接口而创建，每一个未完成的调用需要一个它自己的调用对象。CreateCall的第一个参数确定