美国微软出版社授权中文版系列书

Inside COM+ Base Services

组件编程技术内幕

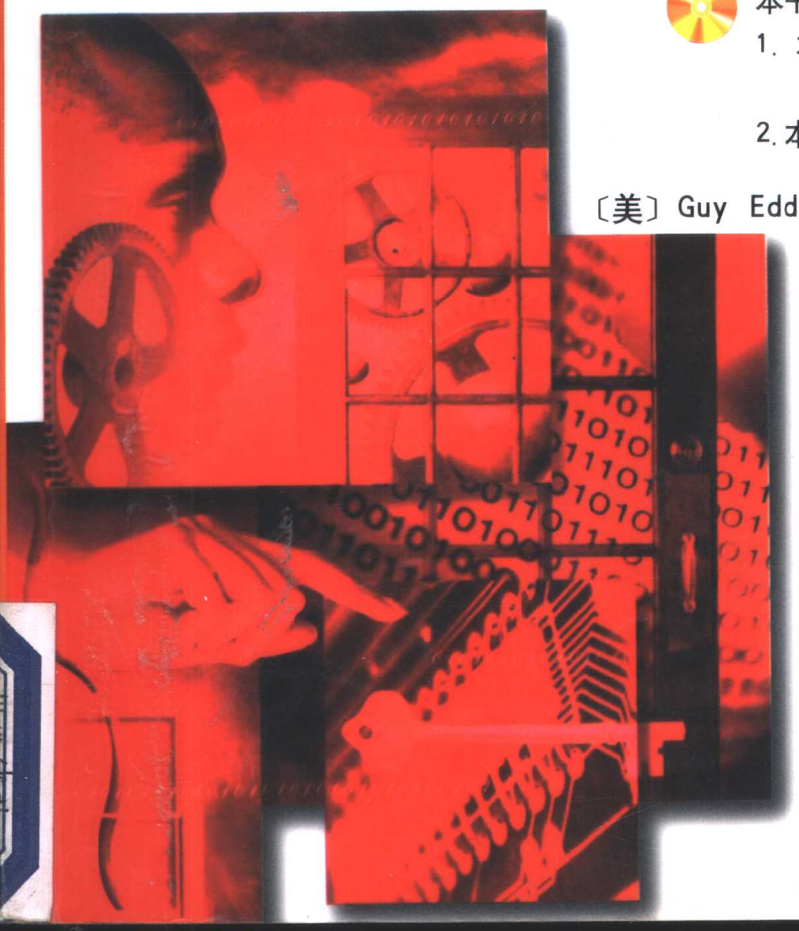
（最新版）

本书配套光盘内容包括：

1. 本书所讨论的C++、Visual Basic和Java范例
2. 本书英文版电子书

〔美〕Guy Eddon　Henry Eddon 著

希望图书创作室 译



北京希望电子出版社

Beijing Hope Electronic Press

www.bhp.com.cn

Microsoft Press

21 BEIJING HOPE 希望 Windows 2000 美国微软出版社授权中文版系列书

Inside COM+ Base Services (最新版)

组件编程技术内幕

本书配套光盘内容包括：

1. 本书所讨论的 C++、Visual Basic 和 Java 范例
2. 本书英文版电子书

〔美〕Guy Eddon Henry Eddon 著
希望图书创作室 译

北京希望电子出版社
Beijing Hope Electronic Press
www.bhp.com.cn

Microsoft® Press

内 容 简 介

本书是美国微软出版社授权的中文版系列书之一。COM+（Component Object Model 的缩写）是组件编程的最新版本。Windows 2000 操作系统提供了 COM+的第一个企业版本。在 Windows 2000 中，每个新的子系统都是作为一个 COM+对象来实现的。对于 Windows 程序开发人员来说，掌握 COM+的架构和各种基本服务是开发基于组件编程的必备知识。

本书主要介绍了以 COM+为核心的基本组件开发。全书的内容覆盖了 COM+所提供的涉及建立软件组件并且有效地使用软件组件的各种基本服务。内容涉及 COM+程序设计的最新概念。全书分为三个部分 COM+的基本编程架构、基本服务、远程架构和附录，内容包括组件软件、IUnknown 接口、语言集成、线程模式、自动化、异常处理、组件类别、连接点、类型信息、持续性、Moniker、代理、可执行组件、自定义调度、标准调度、接口定义语言、异步调用、安全性、网络协议等 19 章。

本书叙述清晰，实用性强，是有兴趣学习和掌握用 COM+进行开发的编程人员的重要参考书，同时也是计算机应用人员及大专院校师生不可多得的参考书。

本书配套光盘内容包括：1. 本书所讨论的 C++、Visual Basic 和 Java 范例；2.英文版电子书。

版 权 声 明

系　列　书：美国微软出版社授权中文版系列书
书　　名：COM+组件编程技术内幕（最新版）
文本著作者：〔美〕Guy Eddon　Henry Eddon 著　希望图书创作室 译
责任编辑：龙启铭
CD 制作者：希望多媒体创作中心
CD 测试者：希望多媒体测试部
出版、发行者：北京希望电子出版社
地　　址：北京海淀路 82 号，100080
网址：www.bhp.com.cn　E-mail：lwm@hope.com.cn
电话：010-62562329,62541992,62637101,62637102（图书发行,技术支持）
010-62633308,62633309（多媒体发行，技术支持）
010-62613322-215（门市）　010-62531267（编辑部）
经　　销：各地新华书店、软件连锁店
排　　版：希望图书输出中心
CD 生产者：文录激光科技有限公司
文本印刷者：北京双青印刷厂
开本 / 规格：787×1092 毫米　16 开本　31.5 印张　572 千字
版次 / 印次：2000 年 4 月第 1 版　2000 年 4 月第 1 次印刷
印　　数：0001~5000 册
本 版 号：ISBN7-900031-94-4/TP•94
定　　价：68.00 元（1CD，含配套书）

前　言

自从我开始为微软的 Windows 操作系统编写应用程序以来，已经有近十年了。以前有些人说要真正理解了 Windows 和它所有的子系统并不难。这些子系统中的每一个都只提供了几百个函数：在 Windows 2.11 中，Kernel 子系统中具有 283 个函数，User 子系统中具有 141 个函数，GDI 子系统中则具有 213 个函数，一共是 637 个函数。仅仅要理解这些函数都用来做些什么实在难不到哪里去。

经过了十年之后，微软公司已经极大地扩充了这些模块，并增加了大量的子系统到 Windows 操作系统中：电话通信、远程访问、打印假脱机、三维图形、Internet 访问、安全子系统、注册表、服务子系统、多媒体、网络连接等子系统。现在对于个人来说要完全了解整个操作系统几乎是不可能的。基于这个原因，我曾经建议开发人员从其中选出一些部分，研究它们并掌握它们的精髓，继而成为精通这些技术的专家。这并不是说将操作系统的其他部分忽略不顾——我本人不仅精通操作系统的内核子系统和用户子系统，同时对 GDT 子系统、网络连接子系统以及其他许多领域都有所涉猎。

在微软公司，这些子系统中的每一个都是由不同的开发小组所开发的，并且每一个开发小组都具有自己的一套开发“哲学”。例如，我知道注册表函数都是以“Reg”前缀开头并返回一个错误代码。而且我同时还知道许多内核函数没有特定的前缀，并且在调用失败时返回“FALSE”。在这种情况下，我不得不调用 GetLastError 函数来查看出现失败的原因。这种子系统之间不一致的表现是使 Windows 编程难以掌握的原因之一。

现在，很明显微软公司将继续坚持发展 Windows 操作系统以用于当前以及将来的应用。为实现这个目标，它需要制定一个计划，以允许新加入的子系统不会过多地增加开发人员学习的难度。换句话说，在使用不同的子系统（或组件）时必须具有一致性。这其中所应用的技术正是本书所要讨论的 COM+。

微软公司通过将每一个新的子系统作为一个 COM+对象来实现，来展示其新技术。当读者使用目录服务（活动目录）、事务服务（微软分布式事务协调器）、（DirectX）、外壳扩展、（ActiveX 控件）、数据库（OLE DB）、（ActiveX 脚本）时可以很容易地看到这种设计技术。COM+是与这些子系统进行交互的方式之一。实际上，为减少开发人员学习 Windows 编程的难度，微软公司正在将原有的一些子系统重新作为一个 COM+对象来实现。

对于 Window 程序开发人员来说，了解 COM+的核心结构是必备的。通过对其的理解，开发人员既可以很容易地使用这些子系统，也可以很容易地开发自己的子系统。分布式计算具有巨大的工作量，需要处理许多复杂的问题，例如数据转移、不兼容的计算机体系结构、异种网络体系结构、时间调度等问题。幸运的是，微软公司具有专门处理解决这些问题的许多开发小组，他们使我们要做的事更容易。Windows 2000 操作系统提供了第一个企业级的 COM+版本。

由于微软公司强有力的支持，COM+将毫无疑问地成为与子系统进行交互的最好的方式。这不仅是在单机上，而且在世界上任何地方的任何计算机上也是如此。本书将成为开发人员的指导手册。

Jeffrey　Richter

译 者 序

现代的编程模式应该达到这样的目的：向软件系统添加新的子系统时最好尽量减少开发者熟悉新系统的精力，换句话说，在使用不同的各种子系统或者组件时必须尽可能的关注其一致性。满足这一需要的先进编程技术就是 COM+。

实用 COM+实现了许多崭新的编程设计，比如，活动目录服务、DirectX、控件、数据访问和脚本技术等。COM+现在已经成为了以上这些技术之间互相交互的主要途径，事实上，大多数人在进行 Windows 编程的时候不免在其庞大的体系架构和内容面前望而却步，正是为了减少这方面的麻烦，微软公司开发了 COM+对象技术。

如果 COM+所带来的软件组件技术帮助，将会大大地缩小 Windows 编程所消耗的时间。现在采用 COM+架构的编写程序方式，结合本书中大量的具体实例，可以使读者内行地指出 Windows 程序的精妙所在，本书中的代码，好多可以直接引用到读者自己的应用程序中去，不啻是熟悉和掌握 COM+软件开发的一本优秀指南。

本书由廖铮、董建平、詹文军组织进行翻译，参加本书编译工作的同志还有吕汝元、张晋、高扬、孙耀辉、胡惠英、李志、曾安明、石小容、吕罡、杨书卷、廖钧、崔羽、王大军、李节、蒋华、郭祥雷、孙庆、周国庆、安洁等。在本书的编译过程中全体工作室同仁共同完成了本书的翻译、录排等工作。全书由廖铮进行审校。

由于时间仓促，且译者经验和水平有限，译文难免有不妥之处，恳请读者批评指正！

译者

1999 年 11 月

目　录

第一部分　基本编程架构

第二部分　基 本 服 务

第三部分 远 程 架 构

第一部分
基本编程架构

第1章 组件软件

只要 PC 还继续存在，信息产业就会继续为与 CPU 能力密切相关的软件提供日益增强的硬件。诸如 Intel 这样的公司已经将其处理器从 8 位和 16 位升级到了 32 位，处理器时钟频率则从 4.77MHz 增加到了 550MHz。现在正在开发的 64 位处理器将运行在更高的时钟频率上。但是，开发处理器的成本是如此高昂，以至于只有大公司才有实力负担得起升级的开销。或许有一天，回报减少规律将发生作用，昂贵的开发成本将超过可能的技术改进。

关于改善整体的处理速度已经提议了许多其他方法。其中最值得关注的概念就是并行处理（parallel processing）。并行处理就是在一个理想的环境中，如果 CPU 在某个特定时间内能处理 x 的工作量，那么 10 个 CPU 则可以在同样的时间内处理 10x 的工作量。并行处理可以用两种方法实现：对称多处理（SMP：symmetric multiprocessing）和分布式计算（distributed computing）。

SMP 方案是将几个（或者几百个）CPU 放在一台计算机里。主机操作系统必须能支持多线程以便不同的线程可以按照计划同时运行在不同的 CPU 上。实现 SMP 需要特殊的软件支持。比如微软公司的 Windows NT，就是最早具有支持 SMP 能力的 PC 操作系统之一。现在，微软的 Windows 2000 可以相当灵活地支持多达 16 个处理器。

本书将分布式计算定义为这样的系统：计算机为了共享 CPU 运算能力而互相连接起来，在用户面前则表现为单个的应用程序。该定义在解释计算机是如何连接起来是相当模糊的（根据该定义，向远程彩色打印机发送文档是这样完成的：首先将文档从自己的计算机拷贝到软盘，然后在连接了彩色打印机的计算机中插入软盘，打印文档。这一系列工作合起来就是分布式计算的示例）。理论上，计算机互相连接的方式是不相关的。最重要的问题或许是通过信息管道方式所能提供的吞吐量。由一个 100M bps、基于以太网的局域网所连接起来的分布式系统将产生完全不同于 56Kbps 调制解调器所连接的系统的性能。

在展开讨论分布系统定义之前，先研究一个实际的、带有一点假设意味的分布式系统。大型银行都拥有遍布世界的分支机构。在过去，银行通常具有一个基于主框架的计算中心作为其世界级的总部，所有的分支机构则使用终端连接到该主机。所有的帐目和客户信息也就存储在一个地方。银行后来决定让主框架计算机退役，取而代之采用分布式系统。每个分支机构现在就有了一台存储本地帐目的主计算机以处理本地事务。此外，每台计算机又通过全球区域网络与其他分支机构连接起来。如果系统不用考虑到客户实际所处的位置，或者不用考虑客户帐目实际存储的位置，甚至果用户没有意识到新系统和被取代的基于集中的主框架系统之间的差异，那么，这就是一个成功的分布式系统。

那么是什么原因使得银行用分布式系统取代中央化系统呢？成本就是驱动企业采用分布式系统的一个主要因素。购买和管理成百上千甚至成千上万的 PC 比起购买和管理一台主框架计算机要经济的多。但是，价格还不是唯一的原因。设计优良的分布式系统在花费同样金钱的情况下可以产生更好的性能。

另一个可能影响了银行决定的主要因素是分布式系统的性能可以超过任何主框架系统。当前的技术使得有可能建立一个由 5000 台基于 Intel 奔腾处理器的 PC 所组成的网络，其中每台计算机的运行速度可以达到 200 MIPS（million instructions per second：百万条指令每秒），整体性能达到 1 百万 MIPS。单个 CPU 要实现这样的性能需要在 10^{-12} 秒内执行一条指令。即便假设电流可以以光速（300,000 千米每秒）前进，在 10^{-12} 秒内也不过才走了 0.3 毫米的距离。如果能在 0.3 毫米立方内建造一个该性能的 CPU 确实是现代工程的壮举（CPU 产生大量的热量足以使其自燃）。

可靠性是分布式系统的主要关注点和目标。设计优良的分布式系统可以实现比相对应的集中化系统高得多的可靠性。比如，如果某集中化系统的失败率为 2%，也就是估计用户将遭遇 2%的停工期。在停工期间，所有的工作都会停顿下来。在分布式系统内，假如每个节点有 5%的失败率（失败率的差异可以由以下原因决定：多数主框架计算机都处于凉爽、无尘的环境内并由受过训练的操作员持续不断的维护，而 PC 则经常是谁也不搭理的扔在一边）。理想的状态下，每次少于 5%的机器停机，也就认为只是系统的性能降低 5%——而不是整个系统的失败。对任务攸关的应用程序，比如核反应堆或者宇宙飞船的导航系统的控制，这些任务的可靠性和冗余度都是非常重要的，而分布式系统就可以能达到特别高的可靠程度。

在以下的公式中，P 是两起不相关事件 A 和 B 同时发生的概率：

$$P(A \cap B) = PA \times PB$$

更一般的情况如下：

$$P_{c,n} = c^n \times 10^{-2n}$$

如果有 n 个不相关的事件，每个事件的失败概率为 c，那么所有事件同时失败的概率就是 $P_{c,n}$。这样，即使每个节点的失败概率为 5%，拥有 100 个节点的用户也最多只会遭遇 20^{-100}%的停机概率（这种情况相当罕见，几乎不可能 100 台计算机同时停机）。可以用以上的逻辑公式估计一下几台计算机同时失败的风险。

图 1-1 显示了整个 10 台计算机中从 1 到 5 台计算机同时失败的可能性。

分布式系统不需要离线升级或者维护。几台服务器可以同时离线或者同时增加几台服务器，一切都不会为用户造成不方便或者危及公司的主要业务。因为几乎没有必要购买超前需要的处理性能，所以对系统性能的计划也可以在更加健全和合理的方式下进行。

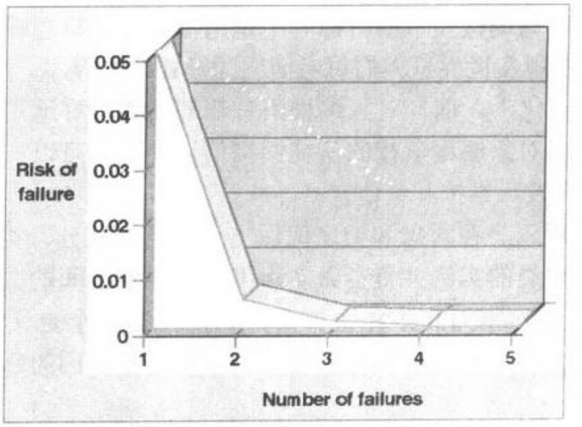


图 1-1　显示了全部 10 台计算机中从 1 到 5 台计算机同时失败的可能性

近年来，已经有许多关于将企业范围的系统从二层的客户机/服务器模式移植到三层架构的讨论。随着日益涌动的 Internet 潮流，以及 Java 编程语言和计算机网络的出现，人们表现出了对三层架构的兴趣。在本章中就将探讨一下三层解决方案以及 COM+是如何适应这一架构的。

1.1　从面向对象编程到组件软件

软件开发过程经过许多年其实并没有改变多少，多数软件开发的努力都受到了成本和时间超过限度的困扰，结果是经常被 bug 所驱使而几乎不可能进行维护。一段时间内，大量的范例和方法论，从流程图创建到面向对象等都一度在不同程度、不同时间范围内接受，成为解决这些难题的灵丹妙药。但最终，所有这些解决方案都遭到了失败，而且没有找到可靠的替代品，以实现小型开发组可以同时致力于一个工程的开发。这并不是说流程图是没有帮助或者面向对象设计是无效的，而是说，它们并没有将软件开发减少到可以保证结果的规范。或许这些都有点吹嘘过度，或者用户对它们的期望过高。现在已经意识到软件开发有其固有的困难、充满了许多没有单一解决方案的问题。

1.1.1　面向对象编程

面向对象编程某种程度上是软件工业乐于接受的最新的方案。这反应在面向对象编程语言（如 Ada、SmallTalk、Java 和 C++等）的大量流行上。这些语言都在不同程度（取决于它们的设计者的原有技术领域、他们所要着手解决的问题以及他们所面对的局限性）上带有面向对象的标志。SmallTalk 是面向对象思想的典型范例。而 C++则专注于面向对象编程。这可能要最终追溯到其来源——C 语言以及对其兼容性方面的迫切需要。尽管微软的 Visual Basic 并不是完全意义上的面向对象语言，但是面向对象的思想却随着它的发展而渗透到其设计中。

简单地说，面向对象编程技术的目标并不是用模仿计算机逻辑的程序方式以方便编程，而是让开发者按照现实世界里人们思考问题的模式编写程序。多数软件都尝试建模，或者将日常工作“可视化”。面向对象编程语言让开发者更好地用代码直接表达现实中存在的对象。这些面向对象编程学校的信徒们相信这种技术可以产生更富于表现力的代码，这些代码更易于开发、维护开销便宜。

多数面向对象编程语言在对象和类之间做出了严格的区分。类（class）是定义成员的模板。对象则是特定类的实例并负责做实际工作。他们之间的关系可以这样一个例子描述。试比较一下甜饼切割刀和小甜饼。甜饼切割刀是一个定义甜饼不同属性的模板（类），比如形状和大小。而甜饼则可以比喻为一个对象，因为甜饼是在甜饼切割刀的基础上创建的。

为了帮助程序员更好地使用面向对象方法，多数面向对象编程语言都提供对以下三个概念的支持：

- 封装（Encapsulation）——隐藏对象的实现细节
- 继承（Inheritance）——重用已有对象以创建新的、更专门对象的能力
- 多态性（Polymorphism）——根据所使用的对象，展现多种不同行为的编码能力

1.1.2　代码的共享和重用

面向对象编程得到大规模普及是因为它允许开发者可以在完全不同的工程之间共享代码。正如前面所提到的，进行相似代码和算法的重新开发将导致难以置信的时间、精力和金钱的浪费。虽然代码的共享和重用被认为是设计良好的面向对象设计最主要的益处，但是实际共享的代码还是太少了。直到最近，诸如微软的 Office 套件内的应用程序也为实现其工具条、状态栏和拼写检查等标准特性而采用了不同的代码。许多这样的标准图形用户接口（GUI：graphical user interface）控件都已经内置于最新的 Windows 版本中，并允许所有的应用程序共享它们。这样看来，那么操作系统就可谓一个规模相当大（但不是很灵活）的代码重用的范例。

代码重用是任何人都会自然想到的事。这也是很好想法，因为代码重用需要周密计划，其实现要考虑周到。如果你编写了一些代码然后将其送给朋友，以便他们也可以用到这些代码，那么这些代码可以重用吗？程序员如何将代码库链接到应用程序呢？虽然这些都是代码重用的实例，但每个都有其问题。如果将代码送给朋友而他们并不喜欢代码的某些方面，那么他们可能进入源代码内部并做出修改。这种修补与重用的思想是不一致的。修改别人的源代码就象是打碎瓷器店内的小雕像一样——不幸的浏览者成为了自豪的新主人。如果别人修改了代码后出现了问题，那么代码的作者也就没有义务去支持该代码了。此外，在以后升级代码并建立新版本时，别人可能不得不检查全部代码并将其集成到自己的变更中去。代码随后也就不能重用了。如果购买了一个类库并且不喜欢他们的工作方式，那真是很糟糕的事，除非是购买源代码以便可以修改它们，并随之

再度遇到前面的问题。

为了更好地理解代码重用，这里需要一个明确的定义。真正的代码重用意味着代码必须是以足够通用的方式为重建更大型的代码而编写，且还能够按照代码的工作方式得到定制。多数类型的代码重用存在另一个问题：它们一般要求原始的开发者和打算重用代码的人使用同样的编程语言。比如，如果某个类库是用 C++编写的，那么用其他语言编写的应用程序里就基本上不可能重用该代码。同样，Java 类就只能在 Java 程序中使用。因此，虽然使用面向对象编程语言比不使用它能获得更多的软件重用，但是得面对以上的局限。那么，我们能将代码共享和重用应用到实际的、现实的编程中去吗？虽然面向对象编程长期以来一直作为对这个问题的最直接解决方案，但是它还没有完成这个愿望。

1.1.3　组件软件

将工程分解为逻辑组件是面向组件分析和设计的基础。这也就是组件软件的基础。组件软件则由可重用的二进制形式的软件块组成（与源代码不同），这些软件块可以用相当小的精力插入到来自其他开发商的组件中。认识到对软件开发采取基于组件的方式而不规定软件的结构是很重要的。而且，它是一种使得二进制软件组件的编程、使用和独立发展成为可能的模式。组件独立于使用它们的应用程序和用于创建它们的编程语言之外。

软件组件可以划分为几个不同类别，包括可视化组件，比如按钮或者列表框；功能组件，比如增加打印或者拼写检查能力的组件等。例如，一个基于组件的架构可以将多个开发商的拼写检查组件插入到另一个开发商开发的字处理应用程序中去。这样就为最终用户带来了许多优点。某个用户可能喜欢 A 公司开发的字处理应用程序但讨厌其携带的拼写检查。如果该字处理程序设计为其拼写检查组件可以被替换，那么用户就可以购买 B 公司的拼写检查组件（该公司致力于开发拼写检查工具）。组件软件使得软件开发商可以专注于他们做得最出色的方面。

汽车工业可以作为组件软件的一个恰当的比喻。汽车工业的生产商经常购买汽车的各个部分，比如不同厂家的发动机和传送机构等，然后将他们组装成汽车。采用组件软件，软件块也象那样使用——它们不需要重新编译，也不需要源代码并且不局限于某一种编程语言。该过程的术语叫二进制重用（binary reuse），因为它是建立在接口而不是源代码级别的重用之上的。虽然软件组件必须遵守一致的接口，但是它们的内部实现是完全自动的。因此，可以用过程语言和面向对象语言创建组件。

基于组件编程模型的主要目的之一是发挥互操作性（interoperability）。互操作性是在计算机领域中对不同的人有不同含义的模糊术语之一。在组件软件的环境中，互操作性是组件协同工作的能力。如果工程的软件是很不自然地强性集成在一起，那么该应用程序就应该分成组件。

在过去的 Windows 编程岁月里，如果程序需要一个工具条，可以简单地直接在程序

的主体部分编写工具条代码。这样做有两个固有的问题。首先，你的目标也许不是为了创建很“酷”的工具条，而只是要创建一个有工具条的应用程序。其次，在耗费了很长时间开发工具条之后，如果公司内的其他开发者打算在他们的工程中也使用该工具条，而代码是分布在整个程序中响应 WM_CREATE、WM_PAINT 和 WM_LBUTTONDOWN 消息的，那么他们就不能很方便地重用它。

解决此问题的基于组件的方法是把工具条设计成为一个单独的组件。这样问题就成为工具条组件和应用程序应该如何交互——也就是互操作性。该组件软件方案要求所有的组件定义其接口以展示其可用的功能。只要组件实现了接口而且客户程序遵守它，互操作性就解决了。在上面的工具条范例中，应用程序开发者只要简单地从其他一些专业制作工具条的开发商那儿购买工具条组件即可，这样就节约了开发、调试和维护的时间。

在软件世界里，最好的紧凑组件实例就是 ActiveX 控件。ActiveX 控件就是一种 COM+组件，它们经常放置在窗体上（也就是和用户交互的地方）。用户将 ActiveX 控件和窗体看作一个单一应用程序，即便这两部分软件都是分别独立开发的（也许还使用了不同的编程语言），并且没有在一起编译。控件和窗体通过接口协同工作。窗体查询控件以了解其所支持的接口，控件也对窗体做同样的工作。将面向对象编程概念和基于组件的方法联合起来可以创建灵活的、功能强大的可供其他开发者重用的对象。

前面提到的一个面向对象编程的重要原则就是封装。封装对对象的用户隐藏了对象的实现细节。对象的用户只能访问对象的接口。在其工程中使用预先创建好的对象的开发者只对“契约”感兴趣——也就是对象所支持的预定行为。组件软件将对象和客户之间的契约形式化。每个对象通过实现某些接口而声明其功能。访问对象服务的唯一方式就是遍历它所支持的接口。这样的契约就是互操作性的基础。

1.1.4　接口

接口实际上是一种很简单的东西——即是无缝地被集合在同一个名称下的相关方法集。这也是本书对接口的定义。例如，Win32 API 就是 Windows 操作系统的功能接口。采用组件软件技术，不仅操作系统可以使接口可用，而且普通人创建的软件组件也可用。接口是软件组件和其客户之间严格类型化的契约；它的期望行为和期望响应也是相当清晰的；使用组件使得接口可以向程序员和设计师提供具体的实体以供使用。实现相同接口的两个对象也就被认为是多态的。虽然该模式没有严格的要求，但还是应当分解接口以便它们可以在不同的环境下使用。

1.2　COM+的发展

与其他大多数的重要技术一样，COM+也不是由迷恋咖啡的孤独开发者一夜就发明出来的。COM+是多种技术进步的结果，如图 1-2 所示。为了更好地理解 COM+的基础，

这里考察一下其起源。

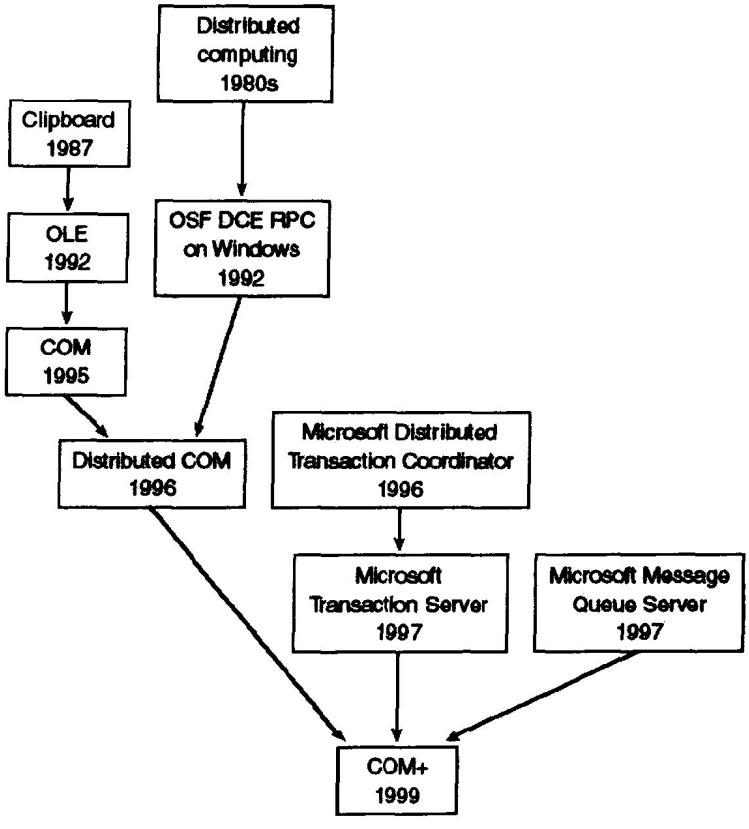


图 1-2　COM+的发展

1.2.1　从 OLE 到 COM+

微软公司的 MS-DOS 操作系统是为极其有限的计算机（按照今天的标准）开发的。随着 Intel 不断制造出功能更强大的 CPU，比如 80286 和 80386 等，微软公司也开始计划实现同时运行多个应用程序的可能性。后来，随着协同多任务 Windows 环境的出现，微软公司意识到用户需要在不同应用程序之间交换数据。剪切板和动态数据交换（DDE）就是微软公司最早集成在 Windows 中的进程间的通信工具。DDE 是开发者用来将协同数据交换工具嵌入其应用程序中的传送消息服务。然而，DDE 是一种相当复杂的协议，结果除了微软公司的 Excel 以外，很少有应用程序能成功地实现了它。

剪切板向用户提供了原始的但是很容易理解的剪切、复制和粘贴操作方案。可以用剪切板创建复合文档——也就是包含不同类型内容的文档（比如，字处理文档就可能包括文本、位图、图表和声音剪辑等）。假设要用微软早期版本的 Word for Windows 为公

司创建一份 1992 年度报告。你可能会首先输入关于描述公司上一年的状态和进度的文字，然后可能还要装载 Excel 来创建含有公司状况图表的电子表格。最后或许要选择电子表格中的某些单元，将其复制到剪切板，并将其粘贴到 Word 文档。如果没有剪切板就将不得不分别打印 Word 文档和 Excel 表单，然后用剪刀和胶水进行剪贴。

剪切板服务从开发者的角度而言很容易实现，也很容易为用户所控制，这使它立即获得了成功。直到现在剪切板都是 Windows 应用程序常用的功能。然而，剪切板的缺点也很快变得明显起来。基本的问题就是，剪切板作为创建复合文档的方式缺乏数据传送的智能化。例如，在使用剪切板创建公司的年度报告以后，可能会发现需要更新 Excel 电子表格上的某些图表。这就需要再次选择、复制并粘贴图表到 Word 文档中去。

最初的对象链接和嵌入（OLE）概念就在微软为解决以上问题的努力中产生了。OLE 首先是在 1992 年微软的 Windows 3.1 中发布的。OLE 的原始思想是为处理复合文档提供一种改进的机制。例如，智能的复合文档既可以嵌入 Excel 电子表格数据也可以将其链接到 Word 文档。只要电子表格中的链接图表更新，新数据也就自动地在 Word 文档中做相应更新。还有，通过双击 Word 中的图表，可以在 Excel 中打开并编辑它。虽然很少有人理解这个过程的运作机制，但是这种能力对用户确实是很大的实惠。实际上，DDE 就是 OLE 作为进程间通信协议的最早版本的内部应用。OLE 包含了一些很有意思的思想，其中最重要的就是由不同应用程序所支持的复合文档对象可以由“超文档”创建而协同工作的概念。OLE 设计者继续扩展并提炼这些想法，直到他们开始将复合文档对象看作软件组件——小的、自包含的软件对象，它们可以插入应用程序从而扩展其功能。

1993 年发布的 OLE 2，进一步地提炼了链接和嵌入能力并为其扩展了在线激活（in-place activation）能力。在线激活，有时也叫做可视化编辑（visual editing），它允许应用程序展现其他应用程序的外观，使得用户可以不离开单一窗口环境的情况之下编辑复合文档内包含的所有数据。虽然多数开发者那时或者很长一段时间后还没有意识到，OLE 2 架构是围绕作为基于组件软件的创造性思想而建立起来的。在经过长时期应用之后，已经证明这是 OLE 2 最重要的贡献。许多年后，OLE 退到了幕后而 COM 则走向了舞台的中央。

多数软件工程还继续围绕着满足所有用户所希望的各种特性的目标而进行设计。问题在于这种软件开发模式使得应用程序随着其扩展而变得愈加脆弱。如果要完全理解一个包含大约 100,000 行代码的应用程序很困难，那么理解 1,000,000 行代码的程序就更不可能了。即便是对程序进行一个小规模的修改也需要对其进行反复的测试。经常是对某段代码看起来无关紧要的修改却对其他位置的代码进行破坏了。

COM 在组件之间起到了“胶水”的作用，它使得不相关的软件组件按照有意义的方式连接起来并互相交互。然后，这些组件可以在客户机和服务器上的多种环境中重用。例如，某种特定事务组件可能是用于桌面应用程序以及在微软 Internet 信息服务器（IIS: Internet Information Server）上运行的 Web 应用程序。COM 允许将对象分成为不同的组