目 录

第一章
OLE 现状与发展趋势

1

1.1 简介

Microsoft 的 OLE 规范提供了集成应用程序部件的多种途径，这些应用程序部件包括诸如可视编辑，应用程序间的拖放及 OLE 自动化和对象的结构化存储等。

OLE 功能极其强大。最近 OLE 2.0 已获得两项荣誉：PC Magazine 的 Technical Excellence 奖及 PC/Computing 的对于软件革新的 MVP 奖。目前，超过 30 种使用 OLE 2.0 开发的应用程序已经上市，并且有数百种将会在以后 6 到 12 个月中出现。

OLE 不仅是桌面应用程序集成。为支持集成，OLE 定义和实现了一种允许应用程序作为软件"对象"(数据集合和操作数据的函数)彼此进行"连接"的机制。这种连接机制和协议称为 Component Object Model(部件对象模型)。OLE 许多面向用户和以文件为中心的特性是建立在部件对象模型的简单而全面扩展对象结构上的，换句话说 OLE 是一种方案，这种方案进入了用于建立分布，衍生对象系统的新的强有力通用技术的桌面空间。

本文简述关于计算机工业由面向对象软件迅速转换的商业原因，然后概述部件对象模型的一些功能，着重点放在部件对象模型为网络上跨多机运行的软件间的无缝连接提供了坚实系统模型这一功能。本文还包括了关于 Microsoft 和 Digital Equipment Corporation 的产品方向信息。这些产品将允许基于 OLE 的应用程序和运行在各种基于 UNIX®和使用同样编程模型的服务器平台以及被基于 Microsoft® 系统间 OLE 使用的通信层的软件对象交互作用。

1.2 对象的商业利益

正如其名称所暗示的，OLE 的部件对象模型是基于部件的概念，一个部件是一种可重用的程序片段，它可以很容易就插入到从其他软件销售者得到的其他部件中。例如，部件可能是由软件销售者卖出的拼写检查器，它可以插入从许多销售者得到的字处理应用程序，或者它可能是一个特定的事务监视器，它可以控制许多数据库服务器的相互作用。与之相反，传统的应用程序都是单块的，这意味着它们被广泛的特性所预先封装，它们中的许多不能被移去或替换。

部件软件提供许多用于设计、创建、销售、使用和重用软件的生产方法。这对于软件

销售者、最终用户和公司具有重大意义。

- 对于销售者，部件软件提供了和其他应用程序及分布式操作系统交互作用的单一模块。它可以不经过基本性的重写就能加到已有的应用程序中，它还提供了使应用程序模块化的机制，并增加了在适当时替换系统的能力。部件软件的到来可帮助小型、中型或大型的销售商建立多样化的市场。
- 对于用户，部件软件意味着更广泛的软件选择。当用户看到使用部件软件的可能性时，就可能在当地软件零售处购买特定部件并插入应用程序。
- 对于公司，部件软件意味着低花费的公司计算，帮助 IS 部件更有效的工作，并使公司计算机用户更有效。IS 开发者将花费少量时间在开发通用软件部件上而大量时间是开发解决特定商业问题需要的部件。已有的应用程序要使用部件结构不需要进行重写。相反，公司开发者可创建基于对象的“绕接器”，它可以封装传统的应用程序并使其操作和数据获取作为网络中其他软件部件的对象。

遵从 Microsoft 部件对象模型的对象被称为部件对象(Component Objects)。作为长远的策略，部件对象的概念由当前和未来技术组成，它们设计成便于开发和使用部件软件。OLE 2.0 是开发部件对象的第一步。将来 OLE 的实现将设计成使用同样的部件对象模型的基本机制并和 OLE 2.0 向上兼容，也就是都能提供广泛的附加特性以及包含对象网络通信的能力。ISVs 和 IS 开发者在实现 OLE 技术上的投资将通过 Microsoft 在未来版本的 OLE 和 Windows 投资而得以长期的保护。

1.3 OLE 的现在与将来

OLE 及其部件对象模型(Component Object Model)是 Microsoft 体系改革主要途径的第一步。OLE 其重要性在于以下原因：

- 对象的二进制标准，OLE 定义了一整套用于对象的建立和对象彼此间联系的标准方式。与传统而向对象程序设计环境不同的是，这些机制独立于使用对象服务的应用程序和用于建立对象的编程语言。这种二进制标准将广泛用于将来 Microsoft Windows™操作系统版本并能形成部件软件的广阔市场。
- 接口的强制集合，当有产品按其结构销售时，软件结构就变得很有趣了。尽管其他销售者和一些集团已定义了高级对象结构和语言独立对象体系的约定，但仅有使用 OLE 的 ISVs，将基于对象技术的一套互操作的应用程序投放市场。
- 真正的系统对象模型，要成为真正的系统模型，对象结构必须有一个分布衍生系统用于支持数以万计的对象，而没有对象错误连接的风险和与强类型或对象定义有关的其他问题。OLE 部件对象模型满足所有这些要求。
- 分布能力，许多现有的单进程对象模型和编程语言，及一些分布式对象系统都是可以得到的。但是没有一个提供了对于小型内进程对象和大型交叉网络对象相同的编程模型。而且，安全是必要的。Microsoft 的 OLE 具有这些能力。

1.3.1 对象的二进制标准

OLE 允许由不同公司不同编程者编写的对象间的互操作。例如，从一个销售商那里得到的电子数据表能连接到由另一个销售商那里得到的数据库对象，并可将数据库记录到电子数据表单元中。由于两个对象均支持预定义的数据交换接口，所以电子数据表和数据库彼此间不必了解对方是如何实现的，只须知道如何通过由部件对象模型和定义的标准机制连接以及通过公用接口进行数据交换。

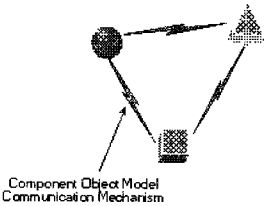


图 1.1 OLE 部件对象模型提供通信的标准方式

如没有对于交互对象通信的二进制标准及一套标准的通信接口，程序员就会面临书写大量子程序的繁琐工作，并且每一个子程序仅用于和一种类型应用程序通信，或重新编译依赖其他相互作用部件的应用程序。而且，如果用于对象交互作用的机制不是非常有效的话，PC 软件开发者如要考虑空间及运行要求的话就不会使用它们。最后，对象通信必须是独立的，因为编程者应该强制性的使用某特定编程语言以便和系统及其他应用程序相互作用。

OLE 部件对象模型遇到了这些挑战。在 OLE 中，应用程序通过使用称之为函数调用或方法集进行相互作用。所有 OLE 对象支持一种称为 QueryInterface(查询接口)的方法，这种方法允许部件间每一种有效的协议及通信去寻找它们共享哪种接口。定义在接口中的功能性的部分加上 OLE 固有的接口协议允许软件部件根据部件需要进行简单或复杂方式的相互作用。他们允许对象系统内部的改动与改进，因为新的接口只要不妨碍已有的部件间的相互作用模式，这种接口就可以被引进而且被认为是安全而有效的。

在最底层，OLE 对象的互相应用是相当迅速和简单的。软件部件间连接一旦建立起来后，在 OLE 对象上的函数调用就仅仅是通过两个内存指针的间接函数调用。在同样的地址空间并作为调用代码的 OLE 对象相互作用的运行花销是可忽略的——仅仅有少量的处理器指令比标准函数调用慢，并且这种花销很少考虑一个典型的调用应包括多种参数。因此，使用 OLE 对象是没有执行障碍的，即使在低档的 PC 机上。

OLE 模型的简明性还提供了语言的独立性，任何一种可建立指针结构并通过指针调用函数的语言——例如 C/C++，Pascal，Ada，Small Talk 和 Microsoft Visual Basic 编程系统——都可以建立并使用 OLE 对象。其他常用语言也正在扩展到能直接提供 OLE 支持，因为 OLE 格式编程将计划归入 Microsoft Windows 未来版本中。面向对象语言能提供他们自己的用在语言对象和 OLE 对象间的高层变换，并能提供分类库以使 OLE 编程变得简单。

未来版本的 Microsoft Windows 操作系统将计划在新服务被定义的地方广泛使用 OLE 格式的接口。可视控件的方便性、多媒体服务和分布式安全性将通过部件对象模型作为目标被定义并编程。已有的 Win32 应用程序编程接口将继续需要和被全面支持。与此同时，基于对象的功能将逐渐普及，应用程序和系统间的区别将会变得模糊并且 Windows 操作系统内部自身也提供简便可替换的部件。

1.4 OLE：接口的一个强制集合

对象模型对于搞理论的人和软件设计者来说是感兴趣的但对用户来说是不需要的。正如本文中所讨论的，OLE 是基于强有力、通用目的对象系统的。Microsoft 的部件对象模型对于对象的互相作用提供了一种语言独立的二进制标准和可缩放到由数以千计对象组成的分布式网络的强类型部件。但 OLE 之所以能在市场上战胜众多对象系统的原因是 OLE 的立即获利对用户来说是明显和必然的。

OLE 提供了可视编辑，应用程序间的拖放，OLE 自动化及对象的结构化存储。可视编辑在编译混合文件时，允许两个或更多文件协同工作并显示窗口以使用户只看到单一文件或窗口，多个编辑器动态装入依赖于文件哪一部分在使用。拖放允许用户使用鼠标选择诸如文件和图表这样的应用程序，并拖到另一个应用程序窗口中进行拷贝或移动。OLE 自动化提供了对于宏指令和描述性语言驱动一个或多个应用程序的标准方法，这种方法是通过查看一内部应用程序层的对象。例如，图片、单元、行、图表、表格并使用改变对象状态的方法实现的。最后，结构化存储允许在建立混合文件时应用程序协同工作，这种文件支持在标准文件格式中存储为嵌入对象的各种原有数据类型。

OLE 2.0 获得了工业界的奖励和好评，PC Magazine 主编 Michael J. Miller 谈到“OLE 2.0 提供了一种简便、有效的组合方法，它将从根本上改变我们对下一代软件的期望”(PC Magazine Dec. 7，1993，p. 78)。专栏作家 Jim Seymour 说“OLE 2.0 对于软件开发者和 PC 用户来说都是一个巨大的胜利，我深信它是 1993 年 PC 软件的一个最重要发展”(PC Magazine Dec. 7，1993 p. 98)。

1.5 一种真正的系统对象模型

对象技术迅速扩展并走出了面向对象语言的领域。一个令人感兴趣的领域是语言独立分类库技术的发展。这种技术解决了“C++in a DLL”问题 ——一个当类自生

改变时使用类的重编译所有代码的问题——并且对于应用程序开发是非常有用的。但它对系统对象模型是不适合的。

当用于建立分布衍生对象系统时，分类库技术有一些根本性的局限性。

- **强类型**：分布式对象系统具有潜在的成千上万个接口和需要唯一标识的软件部件，任何使用可读名称查询并结合到模块、对象、类或方法中的系统都是危险的。在复杂系统中可读名称间冲突的可能性是相当高的。这种可读名称标识就不可避免的造成两个或多个本不应该互相作用的软件部件的连接，这样就会导致出错或一些事故——尽管部件或系统本身无故障并且是按设计要求工作。

  经过对比，OLE 采用了独特的标识——128 位整型数，这样可以保证从时间和空间都是唯一的——用于标识每一个接口、类型和类。可读性的名称仅是为了方便而且只在局布范围内用。这种方法可以保证 OLE 部件不会意外的和另一个对象连接或经由一种接口或方法，即使在有成千上万个对象的网中。

- **不实现继承**：实现继承——是部件对于"子类"或继承另一个部件的一些功能性的能力——是建立应用程序时非常有用的技术。但越来越多专家认为这会在分布式、衍生对象系统中造成一些问题。这些问题收录在学术文章中，称之为"易碎的基类的问题"。实现继承产生问题是在实现类层次中部件间的"联系"或关系没有明确定义，它是模糊和多义性的。当父部件或子部件意外改变了其特性时，那么相关部件的特性就会变成无定义了。在实现类时，如在程序员的定义组控制下程序员同时改变了所有部件的属性就不会产生这种问题。但是它是这种同时控制和改变一组相关部件的能力，这些部件可在不同的应用程序中，甚至复杂的应用程序，从一个真正的分布对象系统。所以尽管实现继承对建立应用程序是一件好事，但在系统对象模型中是冒险的。

  OLE 提供了一种称之为"聚合"的代码重用机制。使用这种模型，一组对象可共同工作在一种良好定义的方式下，并作为单个对象出现于其他软件部件。当保持所有对象间清晰关系时，聚合提供了代码重用的，并可避免实现继承的危险。

- **单个编程模型**：与实现继承的一个相关问题是对于内对象和外进程/交叉网络对象的单个编程模型的问题。在前面情况中，分类库技术允许使用不工作在单个地址空间之外，更少跨网络的这种特性。例如，实现继承就是典型的不工作在单个地址空间之外。换句话说，程序员不能使一个远程对象成为子类。同样，象在类中的公用数据项这种特性，它们可在单地址空间被其他对象操纵，但不能跨过程和网络边界，OLE 的部件对象模型具有一种单接口联编模型，并精心设计成避免局部和远程编程模型的任何区别。

- **安全**：对于在实际中非常有用的分布对象系统，必须提供一种安全的访问所封装对象和数据的方式。尽管 OLE 2.0 没有实现安全特性，但其设计方式和将来全面实现安全性的 OLE 相兼容。对象服务器能被修改成利用这种安全对象调用，但未修改的 OLE 2.0 客户可分享这种全面安全的分布环境。

关于系统对象模型的问题对用户是复杂的，ISVs 在这方面作了计划决议。OLE 能迎接所有挑战，并且是企业界计算环境的坚实基础。

1.6 OLE 的分布式功能

当前版本的 OLE 支持用户桌面计算机集成信息的一整套特性。但可想象用户是否能轻易的集成不同计算上的对象就好像他们都是局部的一样。有了这种能力，例如，在旧金山的用户可以大范围的将其桌面计算机电子数据表格和纽约公司的数据库相连。只要数据库中数据更改了，用户电子表格也会自动地进行这种更改。为了跨不同计算机支持这种层次的对象集成，Microsoft 正在开发一种 OLE 的实现以全面利用部件对象模型的内在能力。分布式对象通信是 OLE 2.0 的下一步工作，它象 OLE 2.0 一样提供同样种类的标准交互对象协同，但允许这种协同发生在计算机的交叉网络。例如一个基于 Windows 计算机中地址手册应用程序可以和基于 UNIX 系统的不同地址手册对象连接并输出其地址信息，但用户不会知道通过网络不同平台间发生了什么样的相互作用。

1.6.1 不费力的远程服务

具有意义的是，OLE 的最新实现不要求对已有的程序进行改动。已有的 OLE 2.0 应用程序在不对其源代码作任何改变且不必重编译的情况下就可以立即和其他机器的应用程序连接。换句话说，应用程序在用户和程序员不费任何努力的情况下，就可得到这些远程能力。

应用程序不必进行任何更改的原因仅仅是低层通信机制（对应用程序是很明显的）正在被扩展。新版本的 OLE 对应用程序的编程接口（API）和对象接口未作改变，所以应用程序可以同样方式调用 OLE 2.0 的函数。如果支持这种函数调用的服务碰巧位于不同的计算机上，OLE 的基础结构会自动和无形的向远程服务发出请求。

1.6.2 分布式的 OLE 是如何工作的

分布式功能是 OLE 部件对象模型的一个自然扩展，当前 OLE 的实现和将来 OLE 实现的一个最重要区别是用于对象间传送操作和数据的远程过程调用。在 OLE 2.0 中，一种"轻便的"远程过程调用机制（LRPL）被用于在单个计算机内的交互对象通信。LRPC 允许对象跨过进程边界传送信息，这种边界是在操作系统中保护应用程序的。换句话说，LRPC 是一种交互进程通信设备，它允许进程（例如对象）和在同一机器中的其他进程对话。OLE 的最新实现通过使用 Microsoft RPC 完成的而不是 LRPC，通过增加了对象的跨网通信从而扩展了 OLE 2.0。

RPC 系统允许应用程序调用远程过程，就好像这些过程是位于同一地址空间，并像调用应用程序一样。有了 RPC，一个应用程序调用远程过程就好像调用局部（内进程）过程方式一样。但实际上，这种过程可能位于相同计算机的另一个进程中或位于跨网络的其他计算机中。因为调用应用程序和响应过程间的数据传送是透明

处理的，因此，就有可能建立起跨进程或计算机运行的应用程序，而不用改变被非分布式应用程序使用的编程模型。

Microsoft RPC 增加了许多功能给 LRPC，其中最值得注意的是 RPC 的跨网络调用的功能。Microsoft RPC 一个细微而重要的方面是它相应的一整套工具，这些工具能产生用于传送封装数据的"Marshaling"代码。这种功能允许程序员很轻易的定义新的对象接口，这种接口允许对象和外界单个进程通信。而且 Microsoft RPC 的 Open Sofware Foundation 与分布式计算环境(DCE)规范中所定义标准相兼容。由于 Microsoft RPC 是基于 DCE RPC 的，因此支持 Microsoft RPC 的系统可和大范围的和基于 DCE 的系统交换信息，包括 Open VMS™，MVS，AS/400 和超过 20 种的 UNIX。Microsoft RPC 对于用于定义对象的 DCE 接口定义语言(Interface Definition Language (IDL))增加了一些向上兼容，但没有改变连接层协议。

尽管 RPC 自身提供了许多分布式对象系统所需的关键能力，例如网络独立传送、安全和名称服务 APIs 以及在不相容处理器和操作系统结构间的数据转换，但是 RPC 还是不完善的。在 RPC 之上的 OLE 的对象层对分布式编程和对象的独特的多形能力提供了进一步的抽象。

在对象层，OLE 2.0 的 LRPC 结构已提供了局部对象样例和非局部样例(在调用程序地址空间外的对象)间的透明性。在建立分布对象系统中获得这种透明性是非常困难的。尽管在增加全部 RPC 机制(和相关能力例如分布式的安全性)中有大量的工作，但在新 OLE 的实现不会改变整体结构。

在下图中，一个客户应用程序连接到了一个运行在不同机器上的对象服务器。一个运行在客户机器上的 RPC Proxy 使客户能好似在同一机器上一样和一个对象服务器通信。Proxy 仅仅是将函数调用封装到了一个标准的 RPC 报文中，这些报文可使用运行客户机器上的网络传输而跨网络传送。在服务器上的一个相应的 RPC 从客户那里收到这些 RPC 报文并将其解开，传送到对象服务器。对于服务器来说，这一过程就好似和局部客户通信一样。

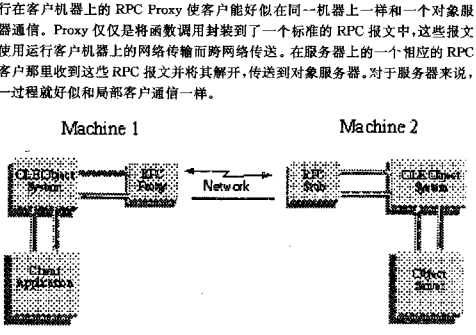


图 1.2 RPC 允许应用程序调用在远程系统的方法

应注意"客户"和"服务器"两者仅对于基于对象服务的用户和提供者来说的。并不是说客户就是桌面机器而服务器是大型的后端机器。事实上，在 OLE 中应用程序

间许多交互作用涉及到一种双向关系，在这过程中两个软件部件互相同时都是客户和服务器。

1.6.3 分布式对象将重定义计算

具有分布式对象支持的 OLE 允许单个应用程序分开成为许多不同的部件对象，其中每一个都可在不同计算机上运行。因为 OLE 提供了网络透明性，所以这些对象看起来并不是位于不同机器上。整个网络就像一台有巨大处理功能的大型计算机。例如，一个数据库应用程序能由一套部件创建：一个查询机，一个报告机，一个表格创建器和一个事务管理器。每一个部件都可运行在适合其大量处理能力，I/O 带宽和磁盘容量的机器上。因此，由于软件能更紧密和其所要求硬件相配，所以计算就会变得更有效。而且计算变得更加容易标度，因为整个网络无限的资源可被单个程序或一组应用程序所衡量。

1.7 公用对象模型(Common Object Model)

认识到允许不同操作系统对象相互作用的需要，Microsoft 和 Digital 已开发了一种允许 OLE 和 Digital 的多平台对象系统 Object Broker™协同操作的结构，这种结构称为公用对象模型(COM)，它定义了一个公用的基于 DCE RPC 的协议和一个将被 Digital 和其他软件公司支持的 OLE 内核函数子集。公用对象模型是直接由部件对象模型生长出的，并提供了和 OLE 完全的向上兼容。当 OLE 和 ObjectBroker 协同工作时，Windows、Windows NT™和 Windows NT™ Advanced Server 操作系统可和运行在包括 OSF/1™，HRUX，SUNOS™，IBM ® AIX ®，ULTRIX™及 OpenVms 平台上的对象相连接。

为了展示 OLE 如何使用 COM 跨平台操作的，Microsoft 和 Digital 已展示了一个样本应用程序，此程序将运行在 OSF/1 操作系统上的对象和在基于 Microsoft Windows NT 计算机上的对象相连接，这一样本程序支持在 Microsoft Excel 和运行在 OSF/1 服务器上的股票份额对象间的一个标准 OLE 2.0 兼容的“热链路”。

对用户的透明，在 ObjectBroker 中的 OLE 元素允许 Microsoft Excel 电子数据表连接到在 OSF/1 机器上的源数据。OLE 和 ObjectBroker 都隐藏了用于寻找 OSF/1 服务器和动态传送股票份额数据到 Microsoft Excel 电子数据单元中的机制。用户在屏幕上实时地看到股票数据更改，而不用了解关于不同数据源的任何事情。

对于用户，受益是巨大的。对象模型，用户可访问任何平台上的信息，而不用关心他们所连接应用程序的类型，或需要达到对象的通信机制。使用例如 OLE 2.0 的拖放这样简单技术，用户可操纵整个企业的对象而不用知道或关于他们位于何处或哪个应用程序用于建立他们。

Microsoft 和 Digital 都遵循关于公用对象模型的打开进程，这将广泛满足工业要求。对于系统集成者，公司应用程序开发者，独立的软件销售商和其他感兴趣的第

三者的设计审查将在 1994 年中进行。在审查之前，将出版审本和评审的草案。

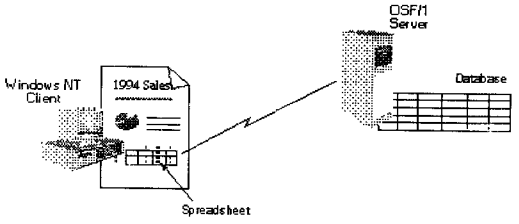


图 1.3 演示 OLE 允许对象链路跨不同的网络存在

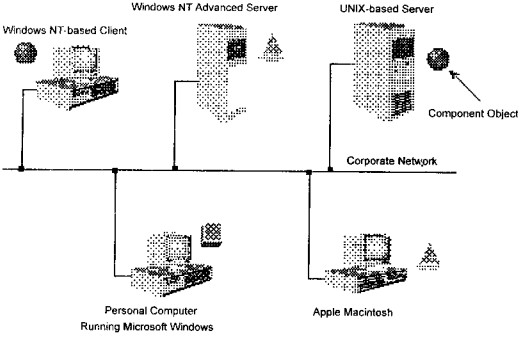


图 1.4

第二章
面向对象编程与 OLE

2

2.1 关于面向对象编程语言的讨论

面向对象编程(OOP)为用户应用程序开发提供了一种不同的方法,给开发策略信息系统的信息部门带来了受益。面向对象编程满足了不同于对象启动系统软件的另一种需求。比如,创建 OLE 应用程序和部件对象不要求使用面向对象编程语言,当然如果愿意的话可以使用。开发用户应用程序时,面向对象编程可在以下方面有所帮助:

- 缩短编程时间
- 提高程序员效率
- 使系统易于维护和适应不断变化的商业需要
- 提高新的应用程序的质量

因为 OOP 方法与标准的基于过程的系统开发很不相同,信息部门需要接受适当的面向对象设计和编程培训,才会从中受益。另外,用面向对象编程语言来实现面向对象系统时,花时间进行系统蓝图规划是很关键的。但未能预见到的类结构的变化(见下面的讨论),可以带走所有预期的受益。是否使用纯粹的面向对象编程语言,应针对不同情况仔细考虑后方可决定。

2.1.1 例子:面向对象的人力资源应用程序

设想一个公司想用面向对象编程创建一个人力资源应用程序对雇员工作记录进行跟踪。设计者开始时可定义一个雇员对象,对象包含了诸如名字、住址、免税额、工资水平等特定数据,然后就可引入特定的雇员对象处理方法,比如签发工资支票、更新福利信息、处理税条表格。应用程序中的所有子部件都可表示为对象。例如,工资支票可表示为工资支票对象。打印支票、记帐、审计则作为支票对象的处理方法。一旦设计好,对象自身就以源码对象类的方式被实现。这些类就是依赖于所用的特定面向对象语言的对象定义。

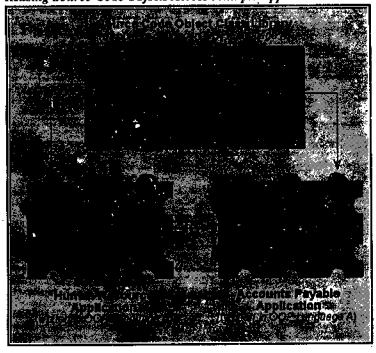
因为对象做到了模块化,所以它们比标准的过程和函数更易于在其他程序中重用。例如,为人力资源应用程序编写的支票对象,可在会计应用程序中重用。

如果设计合理,对象类可在许多不同的应用程序中重用,为此各公司可开发类库。一旦定义,这些类就可用来作为其他程序的源码模块,开发新系统节省所需的时间和金钱。起始的面向对象系统涉及较多编程,因为相对来说类库中类还较少。随着

时间推移，花在编制新对象的时间越来越少，程序员可浏览已有的类库，从中选择所需的类。

但这种方法也有缺点。类库高度依赖于所用的编程语言和实现细节，限制了它们在其他程序中重用的灵活性。如本章稍后所描述的，作二进制包装基于 COM（具有 OLE 服务），部件对象可减少许多重用源码类库的实际限制。例如，OLE 控件的库中提供套装二进制对象，可在任意 OLE 开发工具或包含程序中重新使用。

Reusing Source-Code Objects Across Multiple Applications



2.1.2 应用程序框架

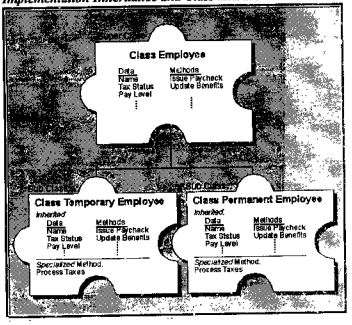
应用程序框架是使编写新程序更容易的商业性类库。它不仅提供减少编程量的类库，也常把复杂的计算资源接口抽象化以便于编程。例如，Microsoft 基类可用于以 C++ 创建 Windows 应用程序。这些类有比低级的应用程序编程接口（APIS）更易学和使用的高级接口。MFC 类提供 Windows 应用程序要求的数据处理和共同例程，大大减少了编程量。

像 MFC 那样的应用程序框架也可抽象其他计算服务的接口，包括打印服务 数据库服务和信息服务。例如，MFC 提供了许多代码，用于编写使用开放式数据库连接（ODBC）的应用程序。是开放式的、不依赖于软件商的 API，用于访问不同的数据库系统。MFC 也完全支持 OLE，允许各公司更容易创建自己的 OLE 部件对象，这些对象可和其他 OLE 套装软件集成。

2.1.3 实现的继承和类层次

实现的继承允许对象类的数据和处理过程定义一次后，可被其子类重新使用，子类对类的功能进行了增强或修改以满足更详细的需要。例如，在上面的人力资源例子中，应用程序可能需要区别固定雇员和临时雇员。虽然两类雇员的数据及其处理的部分可共享，但仍有一些处理可能是有针对性的，比如纳税报告。通过实现继承，所有公共数据和处理可定义为叫"雇员"的父类。然后子类可定义为"临时雇员"和"固定雇员"，每个子类自动继承父类"雇员"的所有数据和处理功能。接着可为临时和固定雇员分别定义纳税报告处理的特殊方法，用来满足各自的特殊需要。这导致称为类层次的结构。

Implementation Inheritance and Class Hierarchies



利用重载技术，两个特定的纳税处理方法可以有同样的名称，每个方法仍能根据处理对象(接受对象)的不同进行不同的工作。例如，为固定和临时雇员处理纳税报告的方法都可称为"纳税处理"。对象自身以其特有方式执行对"纳税处理"的调用(又叫信息)。重载可大大减少函数名的数目，免去程序员命名和记忆大量函数的麻烦。以便程序开发集中在繁随我们思维方式的语言上。这种把两种可选处理藏在一公共接口之后的方法称为多形，即许多形式的意思。利用多形，整个对象系列可共享同一方法名，大大简化应用程序开发。这使得应用程序更易于维护。并且允许其他应用程序重新使用大部分已开发的代码，只需修改对象以满足特殊需要即可。实现的继承和类层次可用于为个人开发项目以节省开发时间，只要这些项目满足一定的条件。这些条件包括 a)用一种面向对象开发所有对象类，b)设计和实现对象类定义时坚持特定的编程标准并做好文档工作。这些条件是有效使用实现继承和相应的类层次的必要前提。如果父类(也叫基类)可改变，程序员还需 c)能自

访问父类的源码对象定义。例如，父类被修改或升级时，负责开发子类的程序员必须有新的父类源代码，以弄明白这些改动如何影响子类。然后他们可以对他们的对象作适当修改，确保它们能继续工作。虽然许多个人软件开发项目满足上述三个条件，但对象启动系统软件一个条件都不满足。这是因为对象启动系统软件必须允许容易的集成事务应用程序，采用的成批生产部件可由许多不同的公司提供，可用许多不同的编程语言开发。

因为对象启动系统软件必须允许这些成批生产的部件对象（没有源码）自由的相互作用，无法控制的实现继承和类层次就不能在系统软件中使用。因此，OLE 部件对象的 COM 说明就不允许任何隐含的关系或相互依赖性存在于利用实现继承创建的对象之间。相互依赖性会限制对象和整个系统的耐用性。实现的依赖性会大大限制选择由不同公司以不同语言开发的对象的自由，因为不能保证由不同软件商或不同开发者所提供的对象能共同工作。进一步说，升级对象也得在不同公司之间协调，这是不现实的。

OLE 部件对象却可以通过部件继承很容易的在不同应用软件中重复使用。部件继承是一种简单的方法，通过它对象可容易的调用另一对象来提供服务。用这种方法，程序员不用重新实现已编制好的代码，他们只需简单的调用另一个部件的功能即可。同时，部件继承可完全控制，也不会出现使用无控制的实现继承可能产生的对象之间的任何隐含依赖性。简而言之，OLE 和 COM 用于创建可重复使用的二进制部件，而不是可重复使用的源代码。

同样的，在编制用户事务对象时，重要的是要明白，使用实现继承可做到代码的重复使用，虽然在一些情况下很有用，但也有局限之处。例如，如果某个父类要做根本改变，任何定义了其子类的应用程序都可能停止工作或需要做很多修改。当类层次由一小组可访问所有对象源代码的程序员集中控制时，它们是最有效的，但在以下情况时则没有用处：在分布式系统中实现时，系统集成由不同编程语言开发的对象时，或系统由不同软件商提供的多种对象组成时，正如在本章稍后讨论的对象编制为 OLE 部件时则可在所有这些情况下容易的重复使用，并且没有上面所说的缺点。

OLE 及其底层部件对象模型是满足对象启动系统软件要求的坚固定义，而一些提出的系统对象模型则试图突破真正的系统对象模型和面向对象编程技术之间的界限。例如，IBM SOM/DSOM 和 Apple OpenDoc（以 IBM SOM 为其对象模型）的定义。都允许无控件的实现继承，以便程序员重复使用已定义的其他对象的实现。这在编程方面可能是方便的，但将导致对象间的相互依赖，妨碍对象在不同软件商的实现之间互换以及在分布式对象系统中的自动升级。简而言之，这些技术不允许集成压缩软件和其他压缩客户部件。对象启动系统软件技术的其他许多方面在 IBM SOM/DSOM 技术或 Apple OpenDoc 定义中没实现。

2.2 面向对象编程：OLE 与 COM 的关系

在过去几年里，面向对象应用程序设计和编程语言越来越受到注意。对有些人来

说，这些工具就是对象技术的精华。OLE 和其底层部件对象模型已超越了编程语言和特定的面向对象开发技术的领域。不过这些工具和技术在 OLE 应用软件开发中仍可起重要作用。实际上，面向对象语言和编程技术可和 OLE 应用程序的开发紧密结合，使开发更迅速、更容易。例如，Microsoft 基类提供有封装的 OLE 功能的源代码对象类，可帮助程序员创建 OLE 应用程序。开发用户 OLE 部件对象和/或集成套装 OLE 应用软件时却不要求使用面向对象编程语言，但如果愿意的话可以使用。

2.2.1 运用 OLE 自动化集成部件软件

运用 OLE 自动化集成应用软件和编制 OLE 对象之间有区别，是很重要的。运用 OLE 自动化只需要任何一个支持 OLE 部件软件编程的编程工具(称为自动化控制器)。它不要求关于面向对象编程、OLE 编程接口，或是部件对象模型的技术知识。相反，开发者只需学会怎样使用基于对象的接口(命令)，以让自动化控制器访问套装软件的功能。关于这些接口的文件由应用软件商提供，就象为任一应用软件的特定宏语言提供文件一样。

简单的说，OLE 自动化，可创建"听"的应用程序，面编程工具用于对这些应用程序"说"，指导它们完成处理任务。例如，要在 Visual Basic 程序的电子报表中自动化图表的创建，开发者只需学会如何使用图表对象的基于对象的接口(命令)。然后，可使用 Visual Basic 开发用户程序，以电子报表应用软件支持的任一方式创建和操作电子报表图表。许多流行的高级开发工具和编程语言已经或将要具有内装的支持，以这种方式使用 OLE 自动化。

事实上，如果应用程序的宏语言支持对其他程序通过 OLE 自动操作"说"(即它是自动化控制器)，则宏语言本身就可用来编写对其他程序的调用。例如，开发人员和系统集成人员可使用字处理软件的宏语言直接调用电子报表的基于对象的接口，只要字处理软件的宏语言是自动化控制器。这样，用自动化集成 OLE 应用程序时，不用使用 C 或 Visual Basic 那样的完整编程语言，应用程序可编制得能够使用各自的服务面直接利用内装的宏语言即可。一个好例子是 Visual Basic for Application，它是现有的 Microsoft Excel 5 的宏语言，并将成为所有 Microsoft Office 应用程序的公共宏语言。

2.2.2 开发自己的 OLE 应用程序和部件对象

丰富的 OLE 部件软件(包括细粒部件对象，比如 OLE 控制器和粗粒的支持 OLE 自动化的应用程序)的可用性以及容易的利用客户应用程序集成软件的能力。使开发人员能够在大大减少编程投人的情况下，能够推出剪裁得当的、功能完整的商业软件。当然，公司和系统集成人员仍然需要开发一些特殊的事务处理程序。今天，联合开发人员和系统集成人员可选择传统语言如 COBOL 或面向对编程语言如 C++和 Smalltalk，来编制用户的处理逻辑。使用面向对象语言而不用传统语言的优点在于，有些情况下面向对象的源代码能在将来需要类似功能的应用程序中更