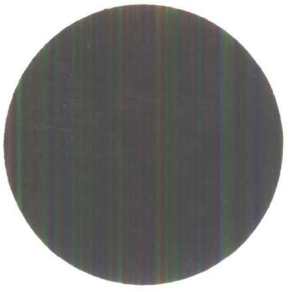
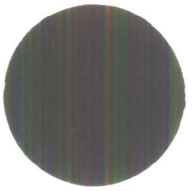
Computer





OBJECT-ORIENTED
PROGRAMMING
AN INTRODUCTION

面向对象程序设计导论

王申康　朱晓芸　编著

浙江大学出版社

面向对象程序设计导论

王申康　朱晓芸　编著

浙江大学出版社

面向对象程序设计导论

王申康　朱晓芸　编著

责任编辑　杜希武

*　　*　　*

浙江大学出版社出版发行

（杭州玉古路 20 号　邮政编码 310027）

（E-mail：zupress@mail.hz.zj.cn）

浙江大学出版社电脑排版中心排版

杭州金融管理干部学院印刷厂印刷

*　　*　　*

787×1092　16 开　20.75 印张　600 千字

1995 年 4 月第 1 版　2000 年 3 月第 6 次印刷

印数：7001—9000

ISBN 7-308-01409-6/TP・139　定价：26.00 元

序 言

面向对象的概念提出已有 20 年的历史，最初出现于挪威奥斯陆大学和挪威计算中心共同研制的 Simula 67 语言中，以后在 70 年代中期，Xerox Palo Alto 研究中心的研究人员设计了 Smalltalk-76 和 Smalltalk-80 语言，面向对象的程序设计方法得到了比较完善的实现，该语言的每个元素都被当作一个对象来实现。Smalltalk 的程序设计环境及其相应的各个方面都是面向对象的。即使在今天，Smalltalk 仍被认为是最纯的面向对象语言，Simula 67 和 Smalltalk 的开发为今天的研究开发工作奠定了基础。

面向对象程序设计(Object-Oriented Programming，简称 OOP)是基于一种很朴素的思想，通常计算机系统是在一定的对象上执行一定的行为操作，要获得灵活而且可再用的系统，最好的办法就是把软件结构建立在对象之上，而不是行为之上，使计算机求解问题更加类似于人类的活动。这也引发了一系列的问题，例如对象准确地说是什么？怎么发现和描述对象？程序怎样操作这些对象？对象之间有什么关系？怎样探索存在于不同类型对象之间的共性？等等。应该说面向对象程序设计的原理是独立于语言的，但面向对象程序设计的思想是可以在传统的程序设计语言如 C、Pascal、Fortran 的语言环境中实现，也可以在模块化而不是真正的面向对象程序设计语言如 Ada 和 Modula-2 中实现。当然纯面向对象程序设计语言 Simula 67 和 Smalltalk 更是这些思想的源泉。这些语言中体现面向对象程序设计思想的技术是依赖于一系列的新技术应用，即继承机制，包括单继承和多继承；动态联编和多态性；一种新的类型观点和类型检查机制；类属机制；信息隐藏；断言的使用；合约式程序设计；安全的异常处理机制。本书通过一系列的语言，如 C++，面向对象 Pascal、Actor 及 Smalltalk，介绍面向对象程序设计的概念和思想及这些语言在使用上述技术时的特征，这些特征使面向对象程序设计更方便、自然、安全。

当今对面向对象程序设计的不恰当解释导致了极度的混乱。人们极端地强调程序设计的语言特征，却忽略了面向对象系统的更大的结构问题。由此导致了一些程序设计者误认为结构化的程序设计原理已经过时并且已被面向对象程序设计所取代。然而，面向对象程序设计发展 20 年来，结构化程序设计的原理却比以往变得更稳固。因为面向对象程序设计补充了这些原理。有经验的程序员只要在他的专业工具箱中添加面向对象程序设计的工具，并且通过试验和训练就能了解如何为他手边的工作选择更恰当的工具。

本书中的示例程序提供了尽可能多的现实的、激动人心的面向对象应用程序，这将为你处理现实世界中出现的问题奠定求解基础。

本书首先介绍面向对象程序设计的概念，并且介绍了一些完整的示例程序，这些程序由于使用了面向对象程序设计技术而变得易于构造和理解。封装、继承、多态性及消息传递这些通用概念简化了程序结构，并且促进了对语言特性的研究，在程序中透彻地讨论了这些通用概念之后，本书才开始详细讨论不同语言所提供的支持程序设计的面向对象方式的特征。

在本书的第二部分，在C++、面向对象 Pascal、Actor 及 Smalltalk 中，对面向对象语言特征的实现进行了比较和对比。除了语言特征之外，对程序设计环境的重要特征也进行了讨论，这些特征中包括如 Smalltalk 那样的有效程序设计工具的表达，如 Smalltalk 中的浏览器(browser)、检测器(inspector)、调试器(debugger)等开发工具，如同对象所引入的新概念一样复杂。

本书的第三部分专门介绍一种特殊类型库——类库，亦称应用框架。应用框架使用户的代码重用成为可能。

我们尽量使示例程序实用，为此，起初示例的程序可能比通常的面向对象程序要大一些，但这些程序的大小及复杂度是精心设计的，它们为面向对象程序设计的原理提供了更好的例证。

如上所述，本书涉及了 4 种通用的面向对象程序设计语言：C++、面向对象 Pascal、Actor 及 Smalltalk。程序开发和程序执行的重要操作包括 MS-DOS 和 Windows。用 Smalltalk 开发的程序可以几乎不加修改地运行在 Macintosh 和 UNIX X-Windows 下，也可以在 MS-DOS 图形模式下运行，或在 Windows 及 OS/2 的图像管理模式下支持。

用 C++开发的程序可以运行在支持标准输入输出流(iostream)库的任何操作环境下，当然，某些较底层的图形输出需要有画线和圆的图形基元的支持。

用于开发面向对象程序设计的专用语言产品有：

- Borland C++及 Turbo C++(所有版本)
- Zortech C++
- Digitalk Smalltalk/V 286
- Digitalk Smalltalk/V Windows
- Turbo Pascal(5.5 及以后的版本)
- Turbo Pascal for Windows
- Microsoft Quick Pascal
- Whitewater Group's Actor

本书另外也使用了其他一些语言产品开发示例程序，这些包括用于 Microsoft Windows 的标准工具，如 Software Development Toolkit(SDK)软件开发工具包及用于扩充上述所列的基本产品而设计的附加类库，包括：

- C++ Views
- Tubro Vision
- Object Windows
- Whitewater Resource Toolkit
- Microsoft Windows 3.0
- Microsoft Windows SDK

C++ Views 是 CNS 公司用于开发 Microsoft Windows 应用程序的一个应用框架。C++ Views 可配合 Zortech C++和 Borland C++编程。Turbo Vision 是 Borland 开发的一个应用框架及类库，它可用于 Turbo C++、Borland C++和 Turbo Pascal 的程序开发。Turbo Vision 也支持在 MS-DOS 下的面向字符的窗口系统(COWS)。

Object Windows 是 Whitewater Group 和 Borland 国际公司联合开发的一个应用框架和类库。Object Windows 运行在 Microsoft Windows 下，并和 Borland C++，Windows 下的 Turbo Pascal 及 Actor 协同工作。Object Windows 使用几乎与 Turbo Vision 相同的类层次和类命名习惯。仅当你

使用 Zortech C++时才需用到 Microsoft Windows SDK。Borland C++提供 Microsoft Windows 标准库、与 Windows 相容的连接器、头文件和资源编译器。Actor 与 Smalltalk/V Windows 不需要 SDK。Whitewater 资源工具包能很方便地开发资源，比如菜单、对话框以及诸如图符和光标的位图映象。Whitewater 资源工具包与 Actor、Borland C++及 Windows 下的 Turbo Pascal 协同工作。

学习本书并不需要你掌握所有这些软件。事实上，如果你仅仅是想学习面向对象程序设计的基本概念，你就不需要拥有任何软件。如果你想实施一个面向对象程序的设计项目，那么本书为你提供了选购这些软件的咨询。你将会对上述所有的产品用于何处有深刻的感受。你可以通过本书知道这些产品在解决相似问题时所采用的不同途径，从而把你所能找到的任何程序改编为你所希望的目标语言。

另外，本书用到的术语是面向对象 Pascal，而非 Object Pascal。Object Pascal 是 Apple Computer 的一个专用产品，原来叫做 Clascal。当 Borland 和 Microsoft 在扩展标准 Pascal 用于支持面向对象程序设计语言特征时，都沿用了 Apple 的 Object Pascal 的基本实现方法。面向对象 Pascal 具有包含 Turbo Pascal 5.5(及其后版本)、Microsoft 的 Quick Pascal 及 Apple 的 Object Pascal 的最广泛含义。

了解上述开发面向对象程序设计所需的环境、语言和工具后，那么你应具备什么样的知识呢？

你必须熟悉至少一种过程性语言，如 C、Pascal 及 Basic 的新变种，另外，你还应该熟悉基本的程序结构，例如：条件语句、循环、可调用程序及结构化程序设计方法。

当前，人工智能、数据库、程序设计语言的研究有汇合之势，面向对象是一个有希望的汇聚点。由于用面向对象的语言开发的软件要比用传统语言开发的软件具有更强的优越性，因此，90 年代“面向对象＋消息”的程序设计模式将取代“数据结构＋算法”的程序设计模式。若称面向对象的程序设计将是 90 年代的一种成熟的计算模型这并不过分。当然，计算机软件工业是有些夸大其辞，在新技术还没有推广之前总喜欢先做许多的承诺，这种过分的热情并非总是不好的，关键是开发者、用户何时且如何受益于面向对象软件。作者希望通过本书的学习，将有更多的软件工作者从中受益。

目　录

第一篇　面向对象程序设计引论

第二篇 OOP 程序设计工具

第三篇 类库和应用框架

第一篇　面向对象程序设计引论

第一章　面向对象程序设计(OOP)的基本概念

1.1　为什么要学习 OOP

任何系统在构造之前都必须经过认真的设计和规划，否则它迟早会崩溃。在建筑领域中，这是一个很显然的道理。同样，在软件领域中也是如此。要构造一个复杂的软件，仅仅把一些机器指令序列、高级程序语言语句、或者过程和模块的集合堆积在一起是不够的。要构造一个强壮的复杂程序，需要有实现的结构技术和指导准则，这样才能建立易于扩充、理解和修改的程序结构。

1.1.1　为什么要学习 OOP

Smalltalk 的设计者 Alan Kay 曾说过："当复杂度上升的时候，结构决定了材料。"Kay 举了一个用砖头来架一条通道的例子。即使你可能知道许多砖头的材料性质，例如，如何把它们粘接在一起，怎样用它们来砌墙、地板以及一些普通结构，但是如果你不了解拱形的机理，那么当你要用砖头来架一个大的拱门时就会遇到麻烦。反之，如果你很熟悉拱形的原理，那么问题就好解决了。拱形原理使我们有可能利用较少的材料来横跨较大的距离，并且使建筑物的结构不仅可以做得很大，而且能支撑更重的负荷。

在软件领域内也是如此。现在我们主要是采用结构化的组织方式来组织软件，而并不仅仅是把原材料堆积在一起(我们可以把机器语言语句或者高级语言语句看作是软件的原材料)。虽然数据结构、过程和函数提供了包装原材料的方法，由这些组件构造的结构可以变得很复杂，但是归根到底，当系统变大并且复杂度上升的时候，简单堆积的方法不仅损坏了整个系统的完整性，而且导致了一个脆弱的系统。

结构化程序设计为程序设计者提供了强有力的工具。今天，大部分的程序设计者都学习过结构程序设计技术，并且认为由结构化程序设计所引入的结构原理是理所当然的。结构化程序设计的意义几乎等同于拱形的发现——它允许软件工程师和建筑师用较少的力气和材料去建造更强壮、更稳固的结构。

发现普遍原理并用它来构造系统，从而有可能利用较少的材料来完成较多的工作。首先要发现，然后是表达和测试。如果一个普遍原理经受了严格的测试，并且被证明是一个强有力的工具，那么就应该把它用到实践中去。经过多年来对程序结构技术的观察和研究，结构化程序设计已经为软件系统的结构建立了一整套原理。

拱形在工程和设计术语中已经引用很久了，以致于我们想当然地接受了它。虽然在大型建筑的构造中并不一定会直接用到拱形，但是拱形解决了一个实际的问题：在为建筑物提供结构支持的同时开设一条通道。在建筑学上，往往既要开通道，同时又要提供坚固的结构支持以支撑足够的重量。在图 1-1 中显示了这种观点的演变，先进的结构原理降低了用于解决特定问题的时间和材料。

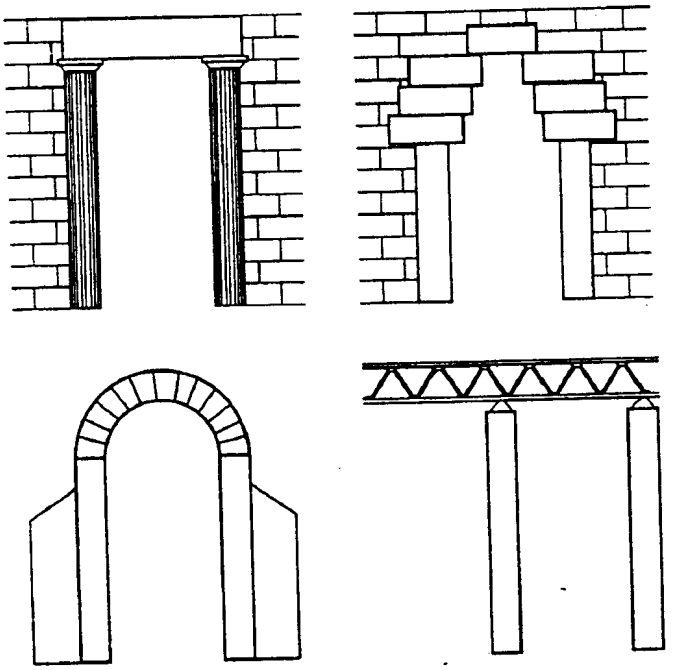


图 1-1　建筑学上结构的演变

拱形是一个以少建大的例子：较少的物理材料取得了更好的结构完整性。桁架和悬梁显示了在建筑学和建筑结构演变中的又一个进步。同样，面向对象程序设计是软件设计和构造中的又一个革命性的进步。

面向对象程序设计是最早承认程序的变化和演变不仅是必然的，而且也是人们所期望的一种程序设计系统。面向对象程序设计通过封装技术减少了程序变化所带来的冲击，并且不仅鼓励对现有代码的重用，而且使代码重用成为可能。

尽管代码重用是结构化语言的一个设计目标，但是它们不能提供易于修改的代码。在结构化语言中把库作为重用代码的一种形式，但是库所要求的稳定性使它们很难适用于特定的问题。而面向对象程序设计采用动态联编(dynamic binding)的方法克服了这一点。

习惯于用过程来思考问题的程序设计者往往觉得难于理解对象的目的和面向对象程序的结构。虽然有人说:"面向对象程序设计需要一种新的思维方法——忘掉你已知的每样东西。"然而,实际情况完全相反。你对结构化设计和编码的原理越熟悉,你就越容易转换到对象上去。

面向对象程序设计只是简单地把优秀程序员们在结构化语言中的某些习惯形式化。例如,一般人都知道应该限制使用全局变量,或者应该把共享数据的过程隔离开等习惯,那么这已经在练习使用面向对象程序设计中的很多原理了。从对象的观点来看,编码最终被归结为过程的构造,即方法。它只是对不同程序组件的包装。而面向对象程序设计要求你能显式地描述过程与共享数据之间的关系。

很多程序员已经把共享的数据和过程包装在单独编译的模块或文件中,并且把对那些模块的存取限制为一小组界面过程。定义这样一个模块与定义一个对象完全相同。面向对象语言仅仅是把这些结构形式化,并且为编译器提供一种方法以保证你的代码不违反你所描述的存取约束。

在面向对象程序设计中可以说没有什么新东西,但又可以说每样东西都是新的。今天,结构化程序设计的原理比以往更充实了,所谓新东西仅仅是人们思想方法的一点细微转变——人们对基于过程的程序结构的看法的变化。

面向对象程序设计和结构化程序设计的不同之处在于前者是同等地强调数据与动作(action)。后者强调动作,并且函数与过程被看作结构的基本单元。另一个区别是程序整体结构的组织观点,结构化程序设计提出了自顶向下的逐步精化方法(最初由 Pascal 设计者 N. Wirth 提出),抽象的动作被分解为许多具体的步骤,直到每一步都能被一句程序设计语言语句所描述。然而,许多问题更适宜用自底向上或双向求解方法,也可能用到第三种结构无顶的系统——事件驱动系统。最后,面向对象的类库提供了真正可重用的组件,而不是把用户限制于一组有限的数据类型之中。

面向对象程序设计的工具有助于缓解编程的复杂性。因为要使软件易于使用,必然使软件难以构造。引入高级用户界面、窗口系统和多任务程序使软件比以往更复杂了,而面向对象程序设计工具就是要帮助解决这些复杂性。而向对象程序设计存在的目的就是要提供一种方法以便系统结构化并处理大量系统组件之间的复杂关系。

面向对象结构减少了系统组件之间的关联数目,使对象通过狭窄的公有界面通讯,这样更易于隔离故障,便于判断哪个方法应对所发生的故障负责。

对象本身能够保护它的私有数据,使它们避免不必要的修改。显式定义通讯协议允许编译器和解释器对用户的非法存取提出警告,并且进一步地可以防止对系统组件的非法修改。

一个面向对象结构的最大好处是可以实现从问题空间对象到程序对象的直接映射。如果程序设计被看作是模拟,那么从被模拟世界中找出对象要比完全基于过程和动作的程序设计开发要容易多了。

1.1.2 如何学习 OOP

本书讲述面向对象程序设计的方法与传统的方法有所不同。本书把面向对象概念作为一个整体,用一个完整的示例程序来开始讨论,而不是逐个地介绍面向对象程序设计语言的语言特征和概念。如果你已经有了基于过程程序设计的一些知识,那么,当你遇到示例程序中一些新的面向对象语言特征时,你就会猜测到有关这些特征的含义。

如果你能阅读 C 或 Pascal 代码,并且了解面向对象程序的一般结构,那么你就已经能看懂

很多C++代码了。当然，可能会有一些新的面向对象特征表达式难住了你，但你常常能猜出这些新的特征和句法的含义。它们使原先由手工实现或者通过使用古怪的数据结构和控制结构来完成的任务自动实现或简化了。也许在看到上下文中的这些新特征时，你会觉得如果先介绍语言特征会使学习更方便，但我们这样做的目的是在讲述细节之前使你有进一步探索的兴趣。

大多数的面向对象程序设计书都是先集中讲述一门语言，通常的方法是讲述某一门语言的每个特征，随后才引入面向对象概念，并且用示例程序来表明如何把语言特征用到编程中去，最后才提出面向对象程序结构的理论。这种表达的次序起初看起来似乎是符合逻辑的，因为它遵循学习用对象开始编程的自然进展，首先是构造块，然后是结构，接着才显示系统和结构技术。假设我们用这种方法来学习说话，那么我们将首先要学习识别名词和动词，然后是正确构造词组和句子，接着我们再用句子来表达一系列的思想，最后，我们将看到其他人是如何使用这些语言的基本形式来与他人交谈的例子。在掌握这些基本概念之后，就会有人让我们坐下来，给我们解释通讯理论，以及我们所掌握的基本概念在那个理论中的地位，以及我们想要与人交谈的原因等等。这种做法显然是很不自然的。表达一种观点的动机是学习说话的第一步而不是最后一步。

用这种常规的方法学习OOP时人们常常会感到失望。他们往往需要花费很多时间和精力去学习面向对象语言的特征和语法。程序员经常会感到迷惑不解的是：为什么这些新的特征简化了我们的工作而它们本身却又难以掌握？当你刚开始试着学习有关语言特征及对象的基本结构来进行面向对象程序设计时，产生这种感觉是很自然的。要充满信心地通过这个阶段是很困难的，因为你可能会怀疑面向对象程序设计是否真的有意义。也许你可能会得出这样的结构，即面向对象程序设计只不过是某个公司为找到新东西来出售而塞给我们的昙花一现的东西罢了。

但是，面向对象程序设计并不真的是昙花一现的东西，它也不是象牙塔中不现实的想法——理论虽好，却毫无实际用处。发明及开发面向对象程序设计的人们并不是从基本词组和语言特征开始，然后再确立基于那些词组特征的程序设计风格的。他们是一些具有大量的程序设计经验的专家，他们利用自己的经验和知识来寻找简化程序设计过程的方法，并且一旦遇到挫折就用现有的工具和技术来改进。简而言之，他们出自于这样一个总的信念和动力：使程序更易于构造和理解。

假设在设计、构造系统时，面向对象程序设计的设计者不是从底向上开始的，那么你认为应该是怎么样的呢？应该试着把面向对象程序看作是一个整体。试着用你学说话的方法来用对象进行编程：从示例程序的上下文中找出完整、清晰的例子并向它学习。如果你采用这种方法，你将会发现面向对象程序设计并不那么困难。事实上，面向对象程序设计大大地简化了人们的工作。一开始，你不必掌握所有的细节，在你掌握了大致的轮廓以后，你就能很自然地理解这些细节了。

今天，人们很难想象，如果拱形从来没有被发现过，那么这个世界会怎么样。很可能在将来的某一天，要人们想象不用对象来构造程序也是同样的困难。

1.2 什么是 OOP

1.2.1 OOP 定义

在过程性程序设计中，仅仅了解什么是过程或如何构造过程并不能保证你将构造出一个易于使用、阅读和修改的强壮的无故障的程序。对面向对象方法而言，也是同样的道理。面向对象程序设计是一种结构技术。在面向对象程序设计中，对象是结构的主要元素，然而，仅仅了解什么是对象或在一个程序中使用对象并不表示你在用面向对象的方式进行程序设计，这只是按顺序把对象拼合在一起的方法而已。

面向对象程序设计的确切含义是一种把消息传递给未知类型对象的程序设计。对象是在一个数组中或一个像工作平台(desktop)这样的聚集中。集中的所有对象共享某种特性(诸如有关屏幕位置的知识以及被重定位、被激活或被释放的能力等)。程序员在集中选择一个对象时并不要求提供关于这个对象的确切类型。

由于当一个消息传递给对象时，该对象的类型是未知的，所以不能预测对象的确切反应。然而，发送消息给未知类型的对象是一种有效的程序设计技术，这也正是面向对象程序设计这个术语所包含的真正含义。

诸如 Microsoft Windows 和 Smalltalk 程序设计环境中的工作平台(desktop)这样的窗口环境中对象的确切类型并不总是已知的。例如，显示在屏幕上的对象可能是滚动条、文本、窗口、图符、菜单、按钮或其他类型的控制器。每个这样的对象对用户传来的消息(如按键盘或按鼠标器)作出反应，当消息发送时，指定了不同的对象就会有不同的反应。例如，按下鼠标器左按钮时使鼠标指向的窗口或应用程序窗口缩小成图标。如果鼠标指向字处理器菜单的文件存贮选择项时，那么正在被编辑的文件将被存入磁盘。

刚才所描述的工作平台(desktop)上的那些对象就是众所周知的多态对象，这意味着它们共享某种公共的特性，例如它们能被放入同一个集(如 desktop)中，而且对相同的消息都能作出反应。多态性与继承性直接相关，用继承进行程序设计可以通过描述与已有对象之间的区别来定义新对象。例如，在 Smalltalk 中，一个提示窗口表现得就像一个文本窗口一样，只是提示窗口只有一个行文本，因为对象“提示窗口”是对象“文本编辑窗口”的继承。

面向对象程序设计的书中常谈及封装、继承和多态性的术语。封装—继承—多态性之间存在着三角关系。继承与多态性是很复杂的，完整的讨论将放在下一章，而封装的概念将在本章讨论。

下面将介绍对象和类，并将解释如何从类定义中建立对象，以及对象是如何通讯的。尽管你必须由对象开始，但仍需要强调框架或程序结构的观点，一个面向对象程序最好被理解为一组通讯的对象。面向对象程序设计强调对象而不是强调行为，这也就导致了与典型的面向过程程序迥异的程序结构。

1.2.2 程序结构

过程性程序设计的结构技术是比较成熟的。在过程性程序设计中，过程是基本的构造块。一种广泛使用的技术是自顶向下逐步精化，即由总体程序结构的最一般的观点出发，然后把程

序分解成更小的可管理模块——过程。即先用一个句子来描述你想要用程序来做些什么，例如："我需要用一个程序来计算一个文件中的字数。"接着你通过描述解决该问题的大致所需步骤来表达一个初步抽象解答，例如：

1. 打开文件。

2. 从文件中读下一个字。

3. 递增字计数器。

4. 在文件末尾报告所读字数。

5. 关闭文件。

这个策略就是不断地分解每个子任务，直到最终你能用你所希望的程序设计语言的语句来表示给定任务的各个步骤。

大多数过程性程序远比上述例子复杂得多，但不管其复杂度如何，把复杂问题分解为更易解决的子任务的过程是相同的，即从一个最抽象的问题开始，把问题分解为越来越具体的步骤。源于这种分析，就扩展了抽象的层次。在最终的程序设计解答中，你将会使用比较一般的过程来调用更具体的过程(子程序)，直到最终达到最底层的抽象，即能直接用程序设计语言语句来表达。这种对一个问题的逐层分解所产生的结构如图 1-2 所示。

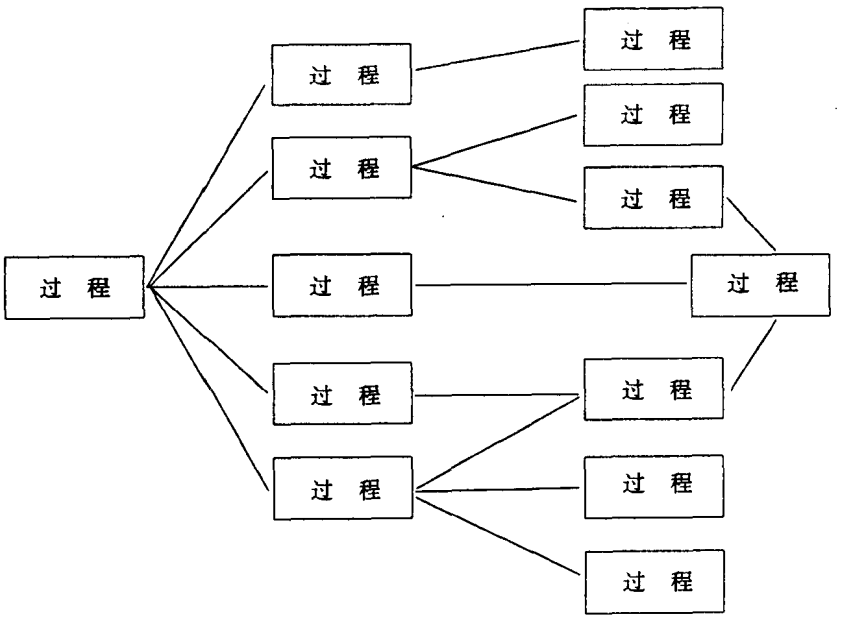


图1- 2 理想的过程性程序设计的程序结构。局部数据隐藏在过程的内部，共享数据由参数传递。

逐层分解和逐步精化的技术用得如此广泛，以至于大多数程序员都在不知不觉地用到它——他们在给定的问题中自动寻找过程。但是奠定这些技术的两个概念，遇到了面向对象方法的挑战。第一个概念是程序设计应自顶向下完成，第二个概念是程序是一序列一个接一个执行的过程或行为。

把图 1-2 中一个过程性程序的表示与图 1-3 中一个典型的面向对象程序图进行比较，你可以很容易地看到它们结构上的差异。在图 1-3 中还展示了有关面向对象程序设计的一些其

他概念。

首先，该图表明对象是程序中的基本结构单元。因为对象的确切结构还未谈到，所以图本身并不能说明其他更多的东西。但从中可以发现，对象一般用名词描述，而不用动词。问题空间中的对象更直接地映射到程序中的对象而非过程。面向对象程序通常被叫做模拟，因为程序中的对象通常摹拟它们在现实世界中相应对象的行为。

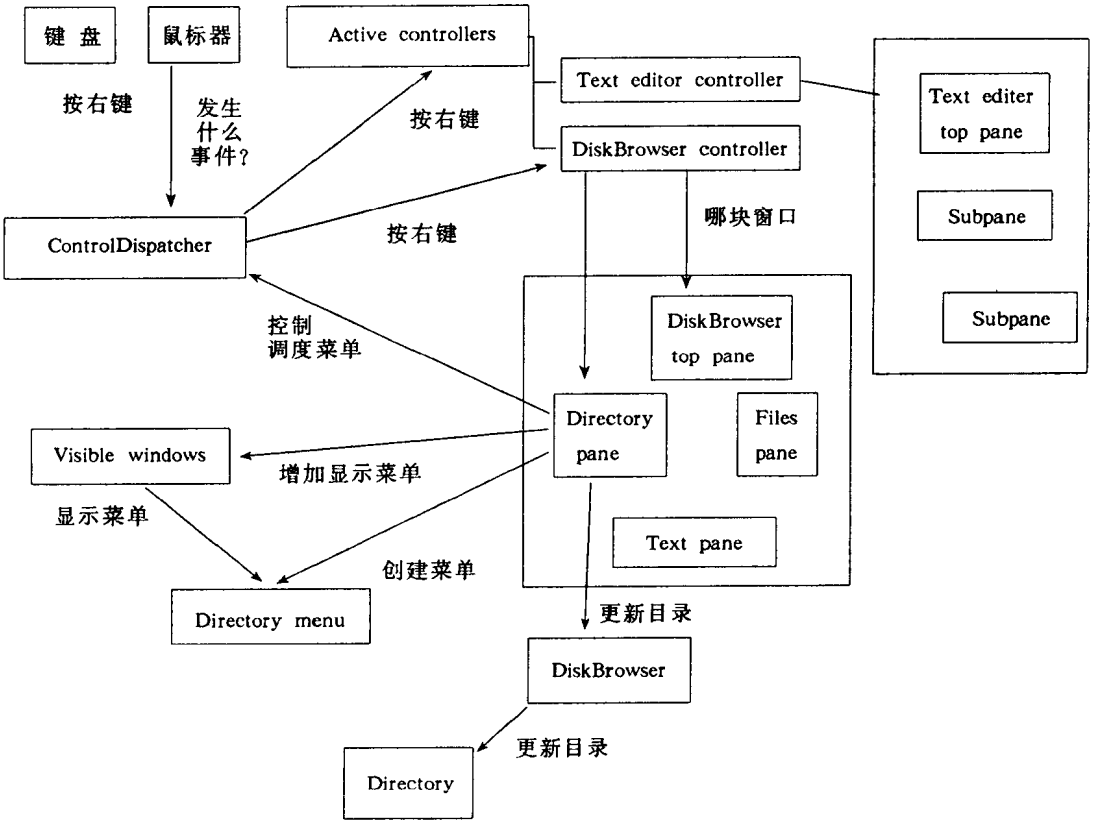


图1- 3　面向对象程序设计的理想程序结构。一个程序是通过消息来通讯的一组对象

从图 1-3 中看到对象也能被看作任务或进程。事实上，原先由 Smalltalk 开发者给予对象的两个名称就是任务(task)和动作(activity)。图 1-3 中对象的例子包括“键盘”、“鼠标”、“控制调度器”和“磁盘浏览器”等。

处理过程发生在对象内部，信息在对象之间相互传递。对象可以通过发送消息来启动其他对象中的处理过程。图 1-3 中的示例消息包括“按右键”、“更新目录”、“创建菜单”及“显示菜单”等。图 1-3 表明了一个面向对象程序是一组正在通讯的任务。

图 1-3 中的程序是一个 Smalltalk 磁盘浏览器(DiskBrowser)，这仅是该程序的部分框图。一个 DiskBrowser 由 5 个窗口组成，而在图中仅表示了 1 个。该图显示了当指在磁盘浏览器的目录表块时按下右鼠标键时发生在 Smalltalk 工作平台中对象之间所发生的通讯。图 1-4 中显示了一个浏览器窗口。

只需要经过 2 步就可以看出一个过程性程序结构是如何进化为一个面向对象的结构的。请看一下图 1-5，它表示了一个实际的程序设计的习惯做法，而图 1-2 表示的却是理想化的。理想化的结构提倡尽可能多地隐藏信息，以保证程序组件的模块化。局部数据被隐藏在过程中，