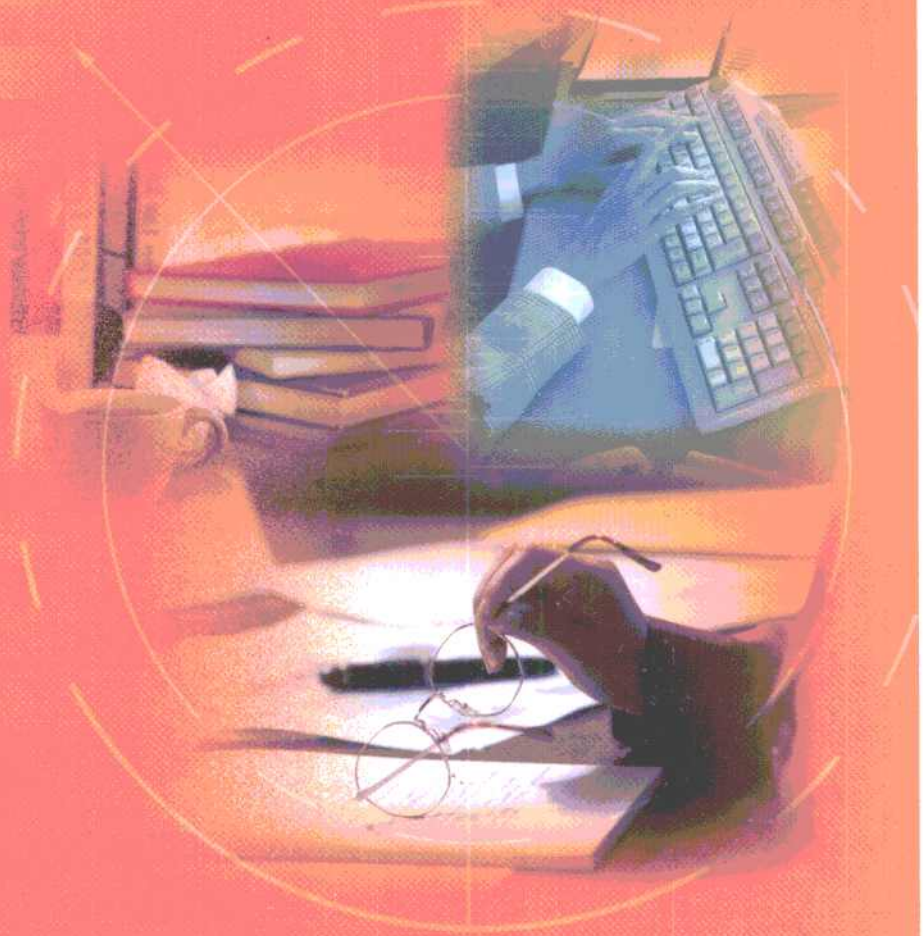
● 编程之路系列教材

Visual C++ 6

程序设计导学



马安鹏　编著

清华大学出版社
http://www.tup.tsinghua.edu.cn

编程之路系列教材

Visual C++ 6 程序设计导学

马安鹏　编著

清华大学出版社

（京）新登字 158 号

内 容 简 介

本书是作者在总结多年从事培训经验的基础上编写而成的，按照适合初学者学习的思路分 12 章组材，内容涉及面向对象的概念、Windows 编程基础、MFC 应用程序框架、消息映射、控制栏编程、MFC 和对话框编程、SDI 和 MDI 设计、动态链接库、绘图和打印、数据库访问等。

本书的特点是简洁的理论说明与大量的实例演示相结合，尽量避免枯燥空洞的理论罗列，易于上手。每章均附有练习题，并给出习题答案，便于读者复习掌握所学的内容。最后以一个综合实例的讲解，提高读者用所学知识解决实际问题的能力。

本书既是非计算机专业学生学习 Visual C++编程的自学教材，也可供各类软件培训班使用。

书　名：Visual C++ 6 程序设计导学
作　者：马安鹏
出版者：清华大学出版社（北京清华大学校内，邮编 100084）
印刷者：北京市耀华印刷有限公司（原门头沟胶印厂）
发行者：新华书店总店北京科技发行所
开　本：787×1092 1/16　　印张：28.25　　字数：687 千字
版　次：2002 年 2 月第 1 版　　2002 年 2 月第 1 次印刷
印　数：0001～5000
书　号：ISBN 7-302-05271-9/TP・3097
定　价：38.00 元

前 言

Visual C++（简称VC）是微软公司推出的一个面向对象的、功能丰富的可视化重量级的开发工具。利用Visual C++提供的高效Windows编程环境，我们可以编写各种各样的软件。与其他软件开发工具相比，Visual C++的强大功能表现在：

- 第一，它是C和C++的混合编译器，使得Visual C++开发的程序具备了C和C++的高效和简洁的特点。
- 第二，Visual C++是一个面向对象的语言，使得软件能够在源码级、类级、控件级等多个级别上重用，软件的开发效率大为加快。
- 第三，Visual C++借助于微软公司出色的MFC类库和应用程序框架，能够轻易开发出Windows标准界面的应用程序。

但是，Visual C++也有缺点，它没有Visual Basic、PowerBuilder等开发工具容易上手，不易入门。在我教过的众多Visual C++学员中，总有人问我："我没有学过C++，我能学好VC吗？"也常听到有人抱怨VC难学，说看了几个月的书，还是理不清头绪；有人则要求推荐能够真正指导Visual C++学习的书。

目前，市场上有关VC的书很多，但真正适合初学者的却很少，不是缺少对VC基本概念、思想的讲解，就是无法正确引导读者把这些基本思想应用到编程实践中。总结VC授课经验，笔者发现，要想学习VC这门深受专业人员称赞的开发工具，首先得了解面向对象的编程思想，这是我们学好VC的基础；然后，我们还需要理解Windows编程的一些基本概念，如窗口类、句柄、窗口句柄等；接下去再学习MFC等VC核心内容，并辅以相应的应用示例，就能起到事半功倍的效果。

本书就是按照上述思路编写而成的，共分为12章。按照学习Visual C++的特点组材，并结合教学的诸多经验，精心组织了每章的内容、实例和练习。

第 1 章　VC 编程第一关：学习面向对象的概念

主要介绍了面向对象的基本概念，包括类和对象以及需要重点掌握的面向对象的封装性、继承性、多态性三大特性思想和具体体现。这部分内容对学过C++和没有学过C++的人都有很好的指导意义。另外，还介绍了VC集成开发环境的部分用法和概念。

第 2 章　VC 编程第二关：学习 Windows 编程基础知识

主要介绍Windows程序的特点和进一步学习VC的一些核心知识：Win32 程序结构、窗口类、窗口C++类、窗口句柄、窗口等，这是理解VC关于MFC应用程序框架的一些必需的概念。

第 3 章 揭开 MFC 应用程序框架秘密

重点讨论了MFC应用程序框架与Win32程序的关系、如何对Win32程序主要逻辑的封装、MFC应用程序的启动流程和MFC应用程序框架的基本类等。还介绍了MFC应用程序框架的基本类对象之间的相互访问关系。

如果不理解应用程序的执行过程，就很难编写出好的VC应用程序，因而本章是学好Visual C++的MFC应用程序框架的基础。

第 4 章 MFC 应用程序的发动机：消息映射

对象的交流主要依靠消息，MFC应用程序框架则为应用程序消息的传递和处理提供了一个独一无二的出色的消息处理机制——消息映射。本章介绍MFC程序能够处理的消息类别：窗口消息、命令消息和控件消息，特别是消息传递和消息处理原理。并重点讨论了各种命令的定义和处理过程。

第 5 章 装饰应用程序的外观：控制栏编程

主要介绍应用程序常见的界面元素：浮动工具栏、自定义状态栏、DialogBar和ReBar，并用实例辅助说明各个元素的初始化与创建原理等方面的知识。

第 6 章 MFC 与对话框编程

对话框也是应用程序常见的交互形式。本章介绍了对话框的创建和数据交换、验证原理，重点对对话框的各种形式（模式对话框、无模式对话框、属性表和向导的创建和初始化）进行了讨论。

第 7 章 绘图与打印

本章重点介绍设备场境对象和图形设备接口对象，同时还对图形输出和打印的一些概念做了详细的阐述。

第 8 章 文档类对象持续性

做好的图形、编排好的文档以及计算好的数据都需要保存起来，如何保存呢？这就是本章要讲的内容。

本章首先对MFC类库的基类CObject的性质做了详细的阐述，然后对如何保存数据和文件以及如何从文件得到数据重建文档对象做了详细的讨论。

第 9 章 文档视图结构的高级形式：SDI 与 MDI

文档和视图结构是Visual C++ MFC应用程序框架提供的，一个文档可以作出多种形式的视图，这种文档和视图的不同组合形式决定了程序的多样性和生动性。本章通过实例介绍了各种常见的文档和视图结构形式的表达方法，借助它们不仅可以理解这种多样性的原理，还能轻而易举地开发出你自己的应用程序来。

第 10 章 动态链接库

对于面向对象应用程序的开发来说，应用程序或者中间代码的发布——动态链接库是常见的形式。本章重点介绍如何把函数、C++类以及资源打包到一个动态链接库中，如何使用动态链接库中的资源。

第 11 章 访问数据库

VC提供了多种访问数据库的方法，本章讨论了访问MFC ODBC和DAO数据库的方法的使用原理和过程。

第 12 章 综合应用——编写绘图程序

综合利用本书各章的主要内容，本章给出了一个比较完整的绘图程序的实例，借以说明程序开发的思路。

在本书各章的实例中，每次需要手工添加的代码都是以粗体显示的；每章的练习题答案见附录；所有实例和练习题涉及的程序代码请到http://www.KHP.com.cn/网站上免费下载，本书程序所有代码都在Windows NT平台下的Visual C++6.0编译链接并通过调试。

由于本书的目的是引导初学者快速掌握Visual C++程序设计方法，所以强调的是重要且常用的内容，对于本书未曾涉及到的知识，仍然需要我们去配合其他参考书籍，最好是VC的联机帮助。

由于笔者水平有限，时间仓促，书中难免存在错误，恳请各位读者同仁批评指正。

编 者

2002年1月

第 1 章　VC 编程第一关：学习面向对象的概念

1.1　为什么要学习VC

C++是当今功能最强大、最完善的计算机语言， Microsoft Visual C++ (简称VC)是微软公司提供的基于C/C++的应用程序集成开发工具。VC拥有丰富的功能和大量的扩展库，使用它能有效地创建高性能的Windows应用程序和Web应用程序，是软件开发人员不可多得的开发工具。几乎所有的世界级软件，从主流的Web浏览器到关键任务的企业应用程序，都是使用VC创建的。VC的优越性主要表现在以下4个方面：

- 开发分布式应用。VC经常用于开发需要广泛发布的应用程序，如Microsoft Word和Lotus Notes。
- 开发的应用运行效率高、具有健壮性。在开发支持成百上千并发用户的服务器程序方面，使用VC开发工具具有明显优势。例如Microsoft SQL Server和Microsoft Internet Information Server等数据库管理系统都是使用VC开发的。
- 能缩短软件升级周期。C++类的重用特性以及它对函数库、DLL库的支持能使程序更好地模块化，并缩短软件维护和升级时间。
- VC能够生成多线程应用，而多线程应用对于增加并发响应有实际意义。

VC除了提供高效的C/C++编译器外，还提供了大量的可重用类和组件，包括著名的微软基础类库（MFC）和活动模板类库（ATL），并且通过向导程序大大简化了库资源的使用和应用程序的开发。与其他开发工具相比，如果选择VC作为开发工具，那么开发的应用程序复杂性越大，则受益越多。

由于VC具有明显的优势，因而应用开发人员学习VC的兴趣有增无减。要想学好VC，首先必须理解VC的编程思想，即面向对象的思想；要使用VC开发Windows应用程序，还必须学习Windows编程基础。这是我们学习VC编程前必须闯过的两道基础关。

本章重点介绍面向对象的思想，有过这方面基础的人可以跳过。

1.2　什么是面向对象

面向对象（Object Oriented，简称OO）是一种新的软件设计思想。这种思想力求使软件系统直接模拟现实世界，尽量将现实世界中的事物直接映射到软件的解空间，从而使用户能够轻松地、最大程度地使用软件系统解决现实问题。

对象是面向对象思想的核心，正确地定义和理解它是掌握面向对象理论的前提。一般

认为，对象存在两方面的含义，即在现实世界中的含义和在计算机世界中的含义。在现实世界中，我们身边实实在在的人和物都是对象，不可触摸的时间、事件也是对象。这种情况下，对象是现实世界中的一个实体。只要我们继续研究，就会发现，这些对象都具备三个共同特征：有一个与其他事物区分开来的标识；具有可以描述自身的状态或属性；具有可以作用于自身和其他对象的操作。因此可以认为，对象是封装了实体状态属性和可以施加于该实体的操作的描述体。在计算机世界中，对象是存储器中一块可标识的区域，它保存了关于实体的属性和操作的数据值。

综合上述两方面的含义，可以给对象一个更加明确的定义：对象是问题域及其实现中的一些事物的抽象，它反映系统为之保存信息和与之交互的能力，是一些属性及其专用服务的一个封装体。

以对象的观点分析问题和解决问题，出现了面向对象分析（Object Oriented Analysis，简称OOA）、面向对象设计（Object Oriented Design，简称OOD）和面向对象编程（Object Oriented Programming，简称OOP）等概念。面向对象思想不但符合现实生活的实际状况，也符合人们的思维习惯。

一个符合面向对象思想的系统，必须具备以下三方面的特性：

- **封装性** 对象是属性数据和对属性数据进行的操作的集合体，而封装性把这些数据和操作屏蔽起来，使用户不必知道对象行为的实现细节，只需根据对象提供的外部特性接口访问对象即可。C++类的定义体现了封装性的基本特性。
- **继承性** 继承是一种表示对象类之间的相似性的机制，它使得某类对象可以具有另一类对象的特征和能力。C++语言的继承性表现在对类的单继承和多继承支持。
- **多态性** 不同的对象收到相同的消息时能产生不同的动作。C++语言支持两种多态性：静态多态性和动态多态性。静态多态性是通过重载实现的，到底执行函数的哪个版本在编译阶段确定；动态多态性是通过虚函数实现的，到底执行函数的哪个版本，需要在运行时找出发出消息的对象后才能确定。

基于上述标准，可以把使用对象的计算机语言分为面向对象和基于对象两种。基于对象的语言采纳了对象的思想，但是它可能不具备继承性或多态性带来的重用性和灵活性，如Visual Basic就缺乏对继承性的支持；而C++，Java和Small Talk等全面支持上述三个特性，属于面向对象语言。

1.3 面向对象的好处

面向对象思想是在面向过程的基础上发展起来的。面向过程的程序模式是数据+算法，程序的流程是固定的，并且程序中数据和对数据的操作是分开的。解决小型的复杂性不高的问题，面向过程的程序模式是十分合理的。但是，当问题变得复杂时，尤其是求解具有大量系统状态的问题时，开发工作面临着诸多实质性的挑战。复杂程序需要多人合作开发，如何划分任务、估计和分配资源、掌握每个程序员的进度、控制及检查每个阶段的设计标

准、如何在多人开发过程中保证程序的正确性，避免出现错误，并且增加程序的可维护性、可读性和可重用性等。当程序的规模变得很大时，开发会变得越来越难控制，当修改一个模块时，可能要改动多数模块，此所谓“牵一发而动全身”，这就是经常谈论的“软件危机”。

如何解决这场危机？人们找到了一个好的解决办法——面向对象方法。采用面向对象的分析和设计方法，可以将一个问题分解成若干小问题，每个小问题又可以分解成更小的问题。每个小问题都可以是一个独立的模块，并且具有一个清晰的抽象界面，它只说明做什么，不必说明如何去做。这种基于数据抽象的模块，又可以引入继承性、多态性等机制产生新的模块，最后再使用动态链接技术将这些模块组装成大型的程序。

维护面向对象的程序是比较方便的。当需求发生变化时，可以按照变化修改相关的对象描述，只要维持对象对外接口不变，其他对象完全可以不改变。例如，一个面向对象的车间管理应用程序的所有模块都依赖于一个成本核算的公共模块，如果成本核算办法变化了，那么，我们只需修改成本核算的模块并维护它的外部特性不变，经过测试，保证成本核算模块正确后，再把该模块插入原有程序，即可完成软件的修改和升级。

面向对象的思想使得开发软件的过程越来越像组装个人计算机。一台计算机是由多个标准的部件组成的，只要配置好各个部件，然后把多个厂家生产的标准部件按照彼此的接口连接组装在一个机箱里，就变成了一台可以工作的机器。为什么可以这么做？因为计算机能够分解成标准模块，而且模块与模块的接口和协议是标准的。同样道理，我们可以把软件分解为很多对象，只要定义清楚对象的外部接口，然后可把这些对象拼合成软件机器，就得到我们的程序。可以想象，有一天我们制作程序会像组装计算机一样容易。

面向对象思想给软件开发带来了深刻的变化。如果按照面向对象的思想分析设计和开发程序，人们会感受到许多实实在在的好处，综合起来，集中表现在以下几方面：

- 容易解决大型复杂问题。按照面向对象的思想，大型项目可继续进行分解，从而使得一个复杂问题的求解变成了许多小型问题的求解；
- 软件维护变得容易，可以在不破坏应用程序的情况下，用更新后的模块替代旧模块，就像把计算机机箱中不正常工作的声卡升级成新声卡一样；
- 软件的开发效率明显提高，因为可以重用很多标准商用功能的类、组件，从而减少开发新程序的时间；
- 当需求改变时，可以重新将对象透明地定位到新的平台，甚至跨越网络定位至其他计算机；
- 将逐步改变人们的编程模式，由全部设计性活动过渡到集成性组装式的工作。

1.4　类和对象关系

对象是现实世界中的一个实体。作为对实体的反映，它包含如下特性：

- 状态　由对象的物理属性决定，以一定的物理量来表示；

- 行为　指与其他对象之间的相互作用途径和方式；
- 标识　描述和标识该对象的惟一标识，由实体的名称来表示。

例如，有一个窗口，名称为MyWnd，位置坐标为(0，0)，大小为宽100、高200，对窗口的操作包括打开、关闭、移动等。

对这个对象的描述如下：

对象标识（或名称）：MyWnd

对象状态：

位置：（0，0）

宽：100

高：200

对象行为：

打开窗口

关闭窗口

移动窗口

类是一个或多个对象的描述，可用一组属性和服务的形式来描述；此外它还可以描述如何创建该类的对象。

类和对象的关系可以看作是抽象和具体的关系。类是多个对象（或者实例）的综合抽象，而实例是类的个体实物。以John为例，John是一个人，人是一个类的概念，包括所有像John一样的个体的特性；而John是人这个类的一个实例。对这个类的描述如下：

类名称：人

类状态描述：

姓名；

性别；

年龄；

肤色；

国籍；

类行为描述：

上班；

出差；

回家；

采用计算机语言形式可记为如下形式：

```
class CPerson
{
private:
char * m_szName;
   int   m_bSex;
   int   m_nAge;
   char* m_szColor;
   char *m_szNationality;
public:
```

```
    CPerson(char*,int,int,char*,char*);
    void OnWork(void);
    void OnBusiness(void);
    void OnGoHome(void);
};
```

在C++中，将上面描述的CPerson称为C++类，它是一个描述对象性质的数据类型，这里CPerson描述了人这一类对象的性质，它定义了一群具有相同属性集合（m_szName；m_bSex；m_nAge；m_szColor；m_szNationality）、相同行为集合（OnWork，OnBusiness，OnGoHome），但不同对象名的对象的集合。其中，类的数据变量如m_szName，m_bSex等，以及类中的函数OnWork等，都称作类的成员。相应地，类中的变量形式成员也可称为成员变量，而类中的函数OnWork等可称为成员函数或方法等。private和public是成员的访问权限，CPerson(char*,int,int,char*,char*)是创建对象的构造函数，在后面章节将详细介绍。

显然，类不能详细地说明某个具体对象的个性。要详细描述某个具体对象，需要将类描述的共性具体化，这个过程也叫作类的实例化过程。类提供了对对象的一种描述方法，但程序中操作的一般是类的实例。

因此，要具体说明个别人，需要从CPerson类生成实例。例如：

对象1（“张三”，男，18，“黄色”，中国）；
对象2（“Tess”，女，20，“白色”，英国）。

1.5　对象之间的关系

世界是由相互作用、相互影响的对象组成的。按照对象之间存在的服务和被服务的关系，可以把对象划分为两大类：客户机和服务器。为其他对象提供服务的对象叫做服务器，使用这些服务的对象叫做客户机。

按照客户机和服务器部署的位置，对象之间的关系大致分为3种情况：

- 进程内服务
- 跨进程服务
- 远程服务

其中，进程是计算机分配资源的基本单元。通俗地讲，它是一个正在执行的应用程序。由于系统以进程为单位，为其分配独立的地址、数据、代码及其他资源，因此进程可用于作为对象部署的边界之一。下面简要地介绍上述3种情况。

1. 进程内服务

当客户机和服务器都在计算机的同一个进程空间运行时，服务器提供的服务叫做进程内服务。进程内服务能够给客户机提供最快速的服务。它们通常是同一程序的对象，或者是从动态链接库（DLL）文件加载至同一进程空间的对象。进程内服务可用图1.1说明，圆圈表示接口。

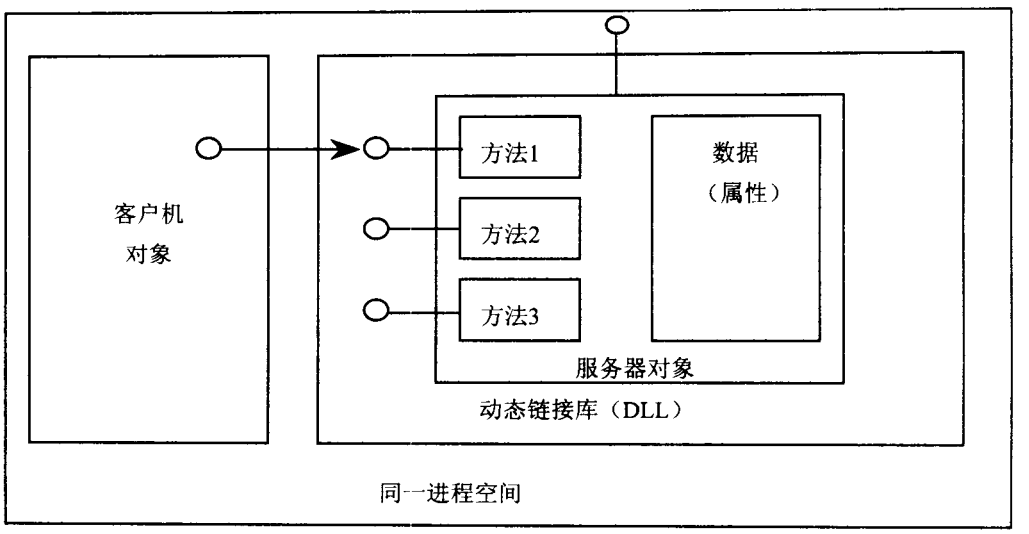


图 1.1 进程内服务

2. 跨进程服务

当客户机和服务器位于同一台计算机上，但是分别运行在相互独立的进程空间时，服务器提供的服务叫做跨进程服务。比较常见的例子是，从Excel拷贝电子表格数据，然后粘贴或链接到Word文档中，这就是著名的对象链接与嵌入服务（OLE）。自动化技术、COM技术等都体现了跨进程服务思想。跨进程服务可用图1.2说明。

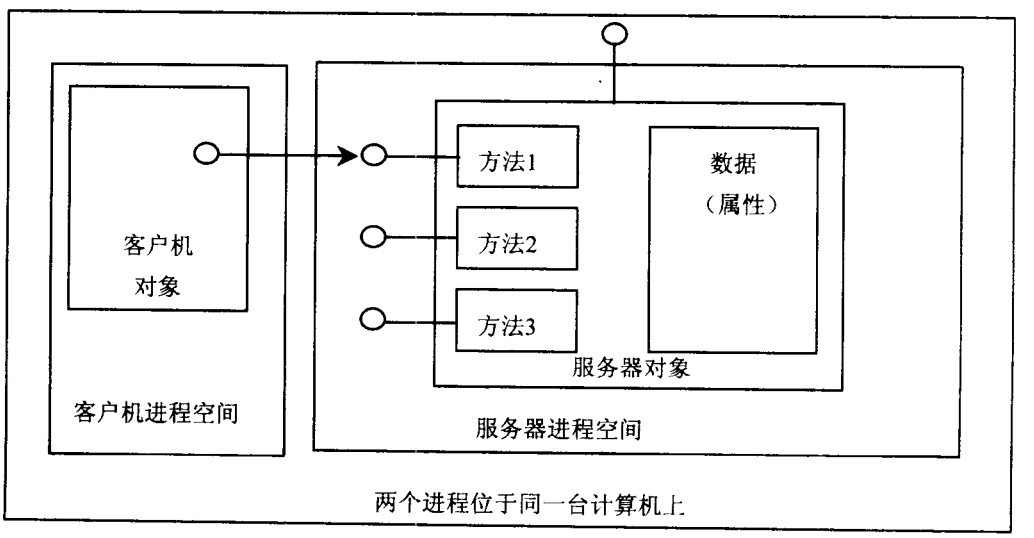


图 1.2 跨进程服务

3. 远程服务

如果客户机和服务器位于不同计算机上，则服务器提供的服务叫做远程服务。微软的DCOM技术以及SUN公司的CORBA技术，都体现了远程服务的特点。远程服务可用图1.3

说明。

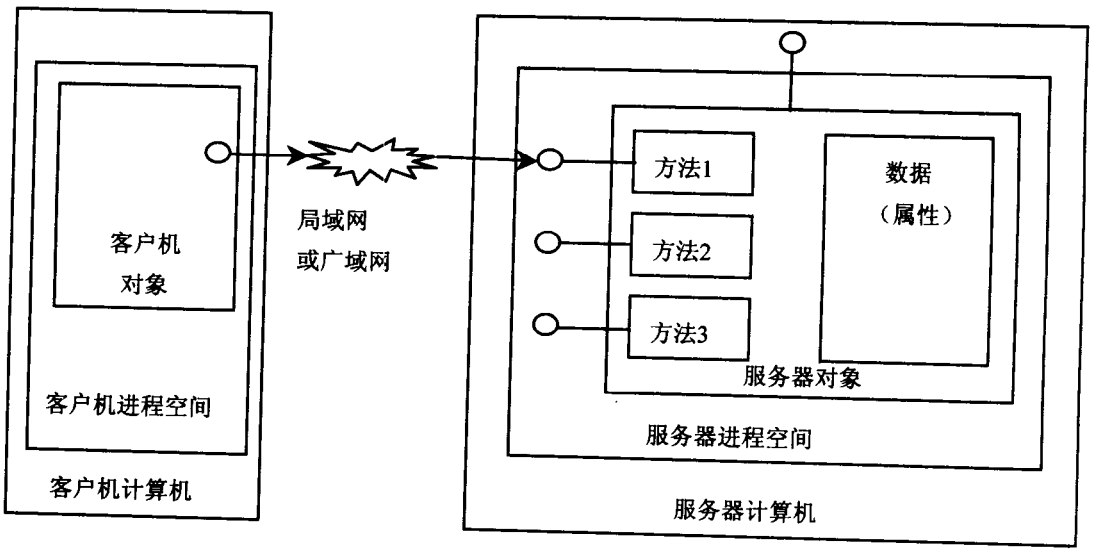


图 1.3　远程服务

1.6　封 装 性

封装性是面向对象的基础。面向对象系统的封装性其实是一种信息隐藏技术，它使系统设计人员能清楚地标注出他们所提供的服务界面，用户则只看见对象提供的操作功能，看不到其中的数据和操作代码细节。面向对象系统的封装单位是对象，对对象的封装是通过类的描述来实现的，它主要包含两个层面的含义：

- 类包含了对实体的属性和对属性操作的描述；
- 类规定了清晰的访问接口。类中具有公有权限（public）的属性和操作是对外公开的，所有外界均可对此进行访问；具有私有权限（private）的属性和操作只供自己使用；具有保护权限（protected）的属性和操作向部分外界公开，只向该类的派生类提供服务。另外，外界访问类中的这些服务时，勿需知道服务是如何实现的。

下面按照封装性的要求，定义一个C++类。

【例1】定义一个描述人的C++类，代码如下：

```
//Person.h文件
class CPerson
{
public:
    CPerson();
    CPerson(char *sl,int a,char *job);
    CPerson(CPerson&P);
```

```
        ~CPerson();
        void TellAbout(void);
private:
        char *m_szName;
        int m_nAge;
        char *m_szJob;
};

//Person.cpp文件
#include "string.h"
#include "iostream.h"
CPerson::CPerson()
{
        m_szName=NULL;
        m_nAge=0;
        m_szJob=NULL;
}
CPerson::CPerson(char *s1,int a,char *job)
{
        m_szName=new char[strlen (s1)+1];
        strcpy(m_szName ,s1);
        m_nAge=a;
        m_szJob= new char[strlen (job)+1];
        strcpy(m_szJob ,job);
}
CPerson::CPerson(CPerson&p)
{
       m_szName=new char[strlen (p. m_szName)+1];
        strcpy(m_szName , p. m_szName);
        m_nAge= p. m_nAge;
        m_szJob= new char[strlen (p. m_szJob)+1];
        strcpy(m_szJob , p. m_szJob);
}
CPerson::~CPerson()
{
        delete[] m_szName;
        delete[] m_szJob;
}

void CPerson:: TellAbout(void)
{
cout<<"Hi,I'm"<<m.szName<<",My job is" << m_szJob
<<",and I'm"<< m_nAge<<" years old."
}
```

上述代码中，Person.h文件与Person.cpp文件共同定义了一个C++类。CPerson类是一个

描述人的类，定义了三个属性（姓名、年龄和工作），还定义了四个方法，其中TellAbout用来供对象自我介绍使用。

使用CPerson类的程序代码如下：

```
#include "Person.h"
void main ()
{
CPerson p1;
CPerson p2("John",19,"computer");
CPerson p3(p2);
P2.TellAbout();
}
```

这里涉及到几个重要的概念，下面详细介绍。

1. 构造函数

构造函数是对象生命的开始。它承担对象的内存分配和初始化工作。构造函数的函数名与类名相同。一般由系统调用。

（1）默认构造函数

CPerson()是默认构造函数。当类中没有任何构造函数时，系统会自动提供一个类似的默认构造函数。但值得注意的是，一旦类中有了构造函数，系统就不再提供了，这时一般要求编程者自己提供一个默认构造函数。

例如，如果用以下代码定义一个A类：

```
class A{
private:
        int a;
};
```

则根据A类定义对象时可以采用如下代码形式：

```
A m; //正确。
```

这时系统会提供的默认构造函数。

然而，如果在定义A类时采用以下代码形式：

```
class A{
public:
        A(int x){a=x;}
private:
        int a;
};
```

则在定义对象时，就不能再使用以下代码：

```
A  m; //错误。
```

为使上面给出的代码正确运行，A类的正确形式应该为：

```
class A{
public:
        A(){a=0;}
        A(int x){a=x;}
private:
        int a;
};
```

（2）自定义构造函数

CPerson(char *s1，int a，char *job)是自定义的构造函数。系统区分自定义构造函数和默认构造函数的主要依据是被定义的对象是否带参数。

例如，用CPerson m("John",19, "computer")定义对象，系统将调用CPerson(char *s1，int a，char *job)函数初始化m；用CPerson n定义对象，系统将调用CPerson()函数初始化n。

（3）拷贝初始化构造函数

当用一个已经存在的对象来初始化另一个对象时，系统会自动调用拷贝初始化构造函数。CPerson(CPerson&P)是CPerson类的拷贝初始化构造函数。它的使用方式为：

```
CPerson m("John",19,"computer");
CPerson p(m);
```

其中CPerson p(m)会调用CPerson(CPerson&p)函数来使用m初始化p。

当类中没有定义拷贝初始化构造函数时，系统也能像提供默认构造函数一样自动提供默认拷贝初始化构造函数。但是，系统提供的默认拷贝初始化构造函数只能完成对象复制，这在类中含有动态分配的地址成员时，显然是有问题的。请考察下面一个例子。

【例2】默认拷贝初始化构造函数的局限性。

```
class A
{
public:
    A(){m=NULL;}
    A(int a){m=new int(a);}
    ~A(){delete m;}
private:
    int *m;
};
```

在main()函数中

```
#include "A.h"
void main()
{
A a(1);
A b(a);
```

```
}
```

使用VC编译并执行程序，会出现如图1.4所示的错误。

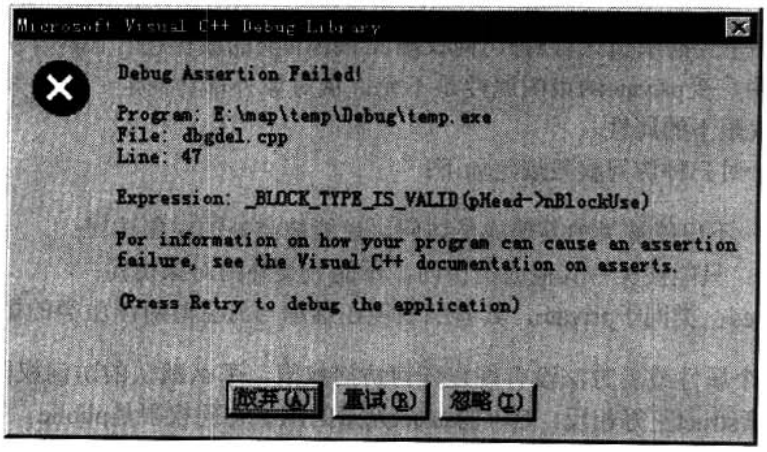


图 1.4　内存错误提示

错误原因是，当用对象a初始化对象b后，a和b的m指针指向同一块内存。结果，系统删除内存时，先删b，再删a时，那块m内存已经不存在。因为一块内存不允许删除二次，于是发生了上述错误。正确的做法应该使用自定义的拷贝初始化构造函数，该函数的定义如下：

```
class A
{
public:
    A(){m=NULL;}
    A(int a){m=new int(a);}
    A(A& p){m=new int(p.*m);}
    ~A(){delete m;}
private:
    int *m;
};
```

2. 析构函数

析构函数是对象寿命的终结。析构函数的函数名与类名相同，前面加“~”符号，一般由系统调用。当类中没有析构函数时，系统也会自动提供一个析构函数，主要用于清理系统分配的对象内存。C++中对内存进行分配和释放的new和delete必须成对使用；否则，会发生内存错误或者内存泄漏。因此，只要类的构造函数使用了new分配的内存，就必须自定义一个析构函数，以便使用delete来释放new分配的内存。

本例定义的~CPerson()函数很好地说明了这点：

```
CPerson::~CPerson()
{
        delete m_szName[];
```