HOPE

C++ 技术和应用

中国科学院希望高级电脑技术公司

C++技术和应用

孟文辉
尹马彤　　编译
王同峰

中国科学院希望高级电脑公司
一九九一年五月

■北京市新闻出版局
准印证号：3321—90321
■订购单位：北京8721信箱资料部
■邮　　码：100080
■电　　话：2562329
■乘　　车：320、332、302路车至海淀黄庄下车
■办公地点：希望公司大楼101房间

编 者 序

本书是为帮助专业的 C 程序员转到 C++世界中来的一本综合性的书．它包括使用和优化 C++的深入观察，并覆盖了 C++增加到 C 的全部功能的细节．主题包括:

- ☐ C 和 C++之间的比较和区别
- ☐ 函数和运算符重载
- ☐ 类设计和类之间通讯
- ☐ C++内存分配

书中也给出了面向对象程序设计的术语和概念，以及它们与 C++之间关系的讨论．另外，还给出了可以用于建立应用程序的基本类的构造过程的实践，有包容、串、复数和显示窗口，而且由于每一个类都有不同的要求，因此采用不同的途径，这样你可以对 C++能做哪些工作获得一个比较全面的印象．

面向对象程序设计正在很快成为当今的热点．C++是其中比较成功的例子．通过把面向对象概念移植到流行的 C 程序设计语言中，C++为当今软件工程提供了一个动态工具．手头拥有此书，你就可以迅速地转入 C++程序设计世界，并把其威力加入到你的工作中．

为了帮助读者学习和了解 C++程序设计，根据我们近年来学习和使用 C++的经验教训，特编译了这本书．由于时间较为仓促，其中肯定会有一些错误之处，敬请广大读友见谅指正．

前　言

本书是为希望了解 C++的有经验的 C 程序员而写的。C++是在贝尔实验室开发的，是 C 语言的面向对象的后继者。通过提供对数据抽象、继承和多态的支持，与 C 相比，C++提供了更多的软件工程优点。在几乎与 ANSI C 百分之百兼容的同时，C++用非面向对象的功能，如强类型检查、直接插入函数和函数重载，增强了 C 的功能。

本书适用于想知道 C++程序设计的方法、内容和原因的程序员。重点放在通过进行和演示实践 C++程序设计的实际例子来学习 C++程序设计上。所有的源程序都已被 MS-DOS 下主要的 C++编译程序，包括 Zortech C++和最新的 Turbo C++，以及 UNIX 和 MS-DOS 下的 AT&T 的 cfront C++翻译程序的部分测试。无论你有什么样的环境，相信此书总会对你有用。

本书的主要内容包括:

- ☐ 面向对象编程的设计
- ☐ C++程序设计技巧
- ☐ C++如何工作的内部秘密
- ☐ 有关 C 和 C++之间区别的示例
- ☐ 完整的面向对象实例，如窗口和视频显示器；表，堆栈和队列；动态串；以及人造生命模拟。

目　录

第一章　面向对象程序设计导论

1.1 投入 C++

我是在 1988 年夏天通过 Micro Cornucopia 杂志接触到 C++程序设计语言的．在此之前，我也听说过 C++，但是从来没有真正注意过它．

当开始调查 C++时，我还是非常小心的．我总是对"最新和最伟大"的革新持怀疑态度，无论是在计算机世界之内还是之外．今天的新闻热点十有八九到明天只是被遗忘的好奇心．当好几个人跟我提到 C++是如何之美妙精彩时，在我作个人决定之前还想对其有稍微多一点的了解．C、Fortran 和 Modula-2 给我提供了作为顾问和作家所需的一切．另外，那时候 C++还只是被实现为一个翻译程序；它把 C++转换为 C，然后需要用通常的 C 编译程序编译．这使得用 C++开发工作实在是太慢了，而用这些时间可以做更多的工作．

在 1988 年年中，Zortech 公司推出了 MS-DOS 计算机的 C++编译程序．这实在是一个十分重大的事件，有以下几个原因：首先，它是第一个真正的 C++编译程序——把 C++源代码直接翻译成目标形式，而不必首先翻译为 C．其次，该编辑程序不太昂贵(对象我这样之类"挨饿的作者"通常是重要的考虑)．

我对于 C++的第一个印象是：它是"修正"C 的一个尝试．这正是许多书和期刊对 C++的描绘——作为"较好"的 C．强类型检查和数据抽象的增加让我可以在 C++中做以前用象 Modula-2 之类强结构化语言做的事情．而且也正如许多程序员所做的那样，通过用 C++的高级性能写 C 程序而了解 C++．

开始用 C++进行工作之后，我的观点发生了改变．以前有些重大的工作都是用 Smalltalk(第一个面向对象的程序设计语言)做的．但是由于 Smalltalk 的解释性质以及图形接口而使性能大受影响．因此，当我发现 C++可以让我做以前在 Smalltalk 和 C 中的所有事情时，就完全支持它了．

不再用 C 编写较多的程序；C++已经取代了它．面向对象程序设计的威力在于使人上瘾；由于已经变得习惯于使用封装、继承和多态机制，而不再用 C 之类的非面向对旬程序设计语言来编程．仍然在 Smalltalk 中做些工作，C++具有许多其他"更纯"的面向对象程序设计语言所缺乏的性能．

大多数 C 程序员喜欢 C 假设他们知道现在正在做什么的事实．他们在象 Pascal 和 Modula-2 之类没有直接限制以防止偶然错误的程序设计语言前犹豫不决．直接语言是可以防止简单的错误，但是也使得创造性更为困难．我总是对象 Modula-2 之类的语言感到受限制，因为总是不能违反规则．C 从不对程序员有任何限制，因此它倾向于吸引持异议者；而那些具有高度创造性的人创造程序．

C++继续 C 的传统．它没有丢弃 C 的任何东西，而是增加了一套全新的功能．从我的观点来讲，C++是已有的最强大的通用程序设计语言．它把已经潜在的 C 语言推进到面向对象程序设计的世界．

但是，这种威力并不保险．如同 C 那样，C++也假设你知道现在你正在做什么．另一个令人头疼的问题是 C++是一种年轻的程序，其定义经常不是很严格的．学习 C++经常如同在不知方向的密林中的冒险，在突然出现之前是不知道路线的．

目前有关 C++的书采用两种途径中的一种．大多数 C++ 的大部头给出语言的简单介绍，勾画出 C++程序设计机理的提纲，而没有掌握该语言所需的细节．其他书把 C++当作 C 的改进，而不给你面向对象程序设计对你的工作意味什么的完整理解．在许多方法中，我把本书写成使你避免我在学习 C++过程中所遇到的一些挫折和头疼的问题．当我开始学习时，有关该语言的参考书少得可怜．它们中有的已经过时，有的写得很糟．因此，我是通过用它"玩"、设计项目并在 C++中构造它们来学习 C++的．结果是我发现了许多该语言的优点和秘密．在本书出版之前，这些秘密一直隐藏在每一个自己发现的程序员的头脑中．

你会发现本书包含一定数目的个人注解，一些轶事，甚至还有一些笑话．有些人认为程序设计是一个值得冷(并是令人厌烦的)学术处理的严肃主题．我个人喜欢读起来好玩的书，这也是我希望我在本书中所创立的程序设计例子都是有教育意义并且是有趣的．而且，也正因为认为程序设计是十分有趣的我才成为一个程序员．这里有一些前提条件．本书假设你已经是精通 C 的程序员．如果你还不习惯 ANSI C，则我建议你先不要读本书；你不需要是 C 程序设计专家，但是你应明白原理．例如，本书并不给出指针是如何工作的说明；而假设你在以前使用 C 时已经知道．对于所有的学习，在明白原理之前先不要进入任何复杂的事情．

1.2 什么是面向对象程序设计?

计算机工业对于术语(terminology)有着特殊的兴趣．我们可以很容易地记住象"人类工程学"(ergonomics)和"用户友好"(user-friendly)之类的术语，而它们以前曾因象是魔术词(magic word)而受批评．不幸的是，那些术语也曾被用在不适当的地方而变成恬聒之词(buzzword)．一旦术语被用得太多而成为恬聒之词，则它被描绘成无意义的．短语"面向对象"好象也被过多使用．

在某些方法中，可以在面向对象和燕麦麸(oat bran)之间画一条平等线．燕麦麸被证明可以减少血液中胆固醇的某些形式，正因为这个原因每天都可以看到新推出的含有燕麦麸的食品．类似地，因为已经"证明"面向对象技术使计算机程序更好，所以什么人都往其硬件和软件产品上贴"面向对象"的标签．结果是相同的：燕麦麸和面向对象成了笑话中的主要词．

面向对象程序设计并不是一个笑话(虽然燕麦麸可能是)．它是可以用于较大地提高程序开发效率并且提高最终计算机程序的可靠性的．但是，为了理解面向对象程序设计到底是什么，很需要理解其根源．因此让我们从看程序设计过程的历史开始，观察面向对象程序设计是如何发展的，并分析它为什么重要．

1.3 线性程序设计

计算机程序设计是一个很年轻的学科．第一台可编程的计算机是在距今四十年前开发出来的．在过去的四十年中，程序设计的发展显示了一个清楚的趋势；计算机程序设计已极大地面向人类程序员．

第一台计算机是用二进制编程的；机械开关用于加载程序．随着大容量存储设备和较大计算机内存的出现，第一代高级计算机程序设计语言投入了使用．不再考虑位和字节之类的术语，程序可以写一系列类似英语的指令，然后可以由编译程序将它翻译成计算机二进制语言．

这些第一代程序设计语言被设计为用于开发执行相对简单任务的程序．这些早期程序主要是进行计算，对程序设计语言没有太多的要求．因此，大多数程序都很短小，通常少于 100 行代码．

随着计算机性能的提高，开发更加复杂的程序的能力也提高了．早期的程序设计语言对于包含更多的编程任务就显得远远不够了．设备需要重复使用已有程序代码，而这在线性程序设计语言中是不可能的．事实上，总有一小段程序被在许多程序中反复使用．程序由于以较长的序列运行，而使它们的逻辑难以理解．程序控制是通过在程序中跳转来进行的，经常没有程序如何以及为什么跳转的明显标志．而且线性语言没有能力控制数据项的可见性．程序中的所有数据项都是全局的，也即它们可以被程序的任何部分修改．在较长的、错综复杂的代码中跟踪对全局变量的错误修改使许多程序员整天忙得不可开交，而效率非常低．

1.4 结构化程序设计

不久就明显要开发带有新的语言特点的新语言，以创建更加复杂的应用程序．伴随着结构化程序设计的引人，革新在六十年代末七十年代初出现．结构化程序按它们所执行的操作来组织．从本质上讲，程序被分成执行较大、非常复杂的过程中离散任务的单独过程(也称作函数)．这些过程之间尽可能地保持相互独立，每一个都有其自己的数据和逻辑．通过使用参数在过程之间传递信息，过程可以有不能被在过程范围以外存取的局部数据．从一个角度来讲，函数可以被看作放在一起可以构成一个应用程序的微型程序．

目标是使软件开发对程序员来讲简单，同时提高程序的可靠性和可维护性．结构化程序是通过把程序的主要功能分开然后变成程序中函数的基本片段来建立的．通过孤立函数中的过程，结构化程序使一个过程可以影响另一个的机会最少．这也使得容易隔离问题．分隔使你可以编写更加清楚的代码并维持对每一个函数的控制．全局变量消失，代之以具有较小的、更容易控制的范围的参数和局部变量．这种较好的结构意味着当你明白了结构化程序的逻辑之后就有了可以成功的机会，从而使得开发和维护更为迅速和更为有效．一个有用的概念也随着结构化程序引人——抽象(abstraction)．抽象可以被定义为不必关心其内部细节而观察某些东西的能力．在结构化程序中，知道一个给定过程执行特定的任务已经足够了．该任务如何被执行并不重要；而该过程是可靠的，可以在不必知道它如何正

确地完成其功能的情况下使用．这被称作功能性抽象，并且是结构化程序设计的基石．

今天，结构化程序设计和设计技术已经是无处不在了．几乎每一种程序设计语言都具有支持结构化程序设计所需的手段．甚至象 Basic 那样传统的非结构化语言也已开始利用结构化程序设计结构．原因是很简单的．结构化程序已被证明比非结构化程序容易编写和维护．

这些优点在程序员编写的种种应用程序中继续．随着程序复杂性的增长，对其处理的基本数据类型的依赖也增加．很显然，程序中的数据结构与在其上进行的操作同样重要．而且随着程序大小的增长，这一点变得更为明显．数据类型被在一个结构化程序中的许多过程处理，当那些数据类型发生变化时，则程序中对那些数据类型起作用的每一个地方都必须加以修改．这在包含成千上万行程序、几百个函数的程序中可能是十分化费时间的，而且可能经常遇到挫折．

当多个程序员组成一队进行一个应用程序的开发时，结构化程序设计的进一步弱点就暴露出来了．在一个结构化程序中，每一个程序员都被赋以建立一套特定的函数和数据类型的任务．因为不同的程序员处理相对于共享数据类型的不同函数，一个程序员对数据项作的修改必定在其他人的工作中反映出来．当结构化程序易于在一组的情况下使用时，组中成员之间的错误交流可能导致很化费时间的重写．

1.5 数据抽象

数据抽象是对数据，而功能抽象是对操作．用数据抽象，就可以在不必关心具体实现细节的情况下使用数据结构和数据项．例如，浮点数在所有程序设计语言中都被抽象．当给浮点数赋值时，不必关心其确切的二进制表示．为了把浮点值相乘不必知道二进制乘法是如何进行的．重要的是浮点数以正确的和可理解的方式进行工作．

数据抽象使你可以不必再关心不重要的细节了．如果程序员必须知晓程序功能的每一个细小方面，则只能写出很少一点程序．幸运的是，在所有程序设计语言中象浮点数这样复杂的组成元素都有数据抽象．不过，只是最近都有了可以让你定义你自己的抽象数据类型的语言．

1.6 面向对象程序设计

面向对象的变化是建立在结构化程序设计概念和数据抽象的基础上构造起来的．基本变化是面向对象的程序是围绕被操作的数据而设计的，而还是围绕操作本身．这当你是旦理解了程序的目的是操作数据时就十分自然了．而且，计算机执行的工作被称为数据处理．数据和动作在一个基本的水平上连接；每一个都要求其他有目的．面向对象程序使这个关系更为明显．

面向对象程序设计用操作联合数据结构，而这正是我们如何考虑世界的方法．我们用给定的对象类型联合一套特定的动作，而且是基于这些联合假设的基础上．例如，我们知道汽车有轮子，可以移动，可以通过方向盘来改变方向．类似地，我们知道树是有木质茎和叶子的植物．汽车不是树，树也不是汽车，因此我们可以假设可以用汽车做的事情不能

用树来做．例如，不能调树的方向，而给汽车浇水，也不会使汽车长大．

面向对象程序设计使得我们可以用在计算机程序中使用的抽象概念使用同样的心理过程．一个人员记录可以被读取、修改和保存，或者一个复数可以被用在计算中．一个复数也不能被作为一个人员记录而写到文件中，而两个人员记录也不能相加．面向对象程序指定其数据类型的性质和行为，这使我们可以确切知道应从各种数据类型预计什么．

在面向对象程序中可以在既相似又有区别的数据类型之间建立关系．人们自然地分类事物．我们常常把新概念与已有概念相联系，而且可以使演绎基于事物之间的关系．我们喜欢用一个树状结构来概念化世界，后继的细节层建立在前面的一般原则之上．这是组织我们周围世界的一种有效方法．面向对象程序以同样自然的方法工作，使新的数据／操作框架建立在已有框架的基础上，在增加新功能的同时联合基础框架的功能．

让我们从非程序设计水平来进一步看这个问题．当遇到单词"植物"，可以立即直观地看到该类型的一个一般项．不需要被告知植物的确切例子，而可以基于所有植物共有的我来一般化．每一种植物都把太阳能转换成食物，而且大多数植物是绿色的，因为在光合作用中它们使用叶绿素．这些一般化是十分有用的，因为当我们一般讨论植物时，不需要知道任何特定的植物实例．

我相信大家都已经见过显示不同种类植物之间关系的分类图．例如，蓝绿色的海藻是生活在水中的较小、单细胞植物．而橡树是具有木质干和树叶的多细胞植物．两者都是植物的，但是它们之间的区别很大．生物学家把海藻和橡树都定义为同一族的成员，但是属于该族的不同子组．

1.7 面向对象术语

面向对象程序设计使你可以用与生物学家组织不同种类植物的相同方法组织程序中的数据．用面向对象程序设计的一般说法来讲，汽车、树、清单记录、复数和书都被称作类(class)．类是描述数据结构和数据项的合法动作的模板．当数据项被声明为类的成员时，它被称作对象(object)．那些被定义为对于类是合法的函数被称作方法(method)，而且它们是操作类的对象数据的仅有函数．

每一个对象都有其自己被称作实例(instance)变量的类数据元素副本．给类定义的方法可以被该类的对象调用．这被称作发送一个消息(message)给对象．消息是专用于对象的；只有接收到消息的那个对象才对该消息起作用．对象相互之间都是独立的，因此对一个对象的实例变量的修改不影响其他对象的实例变量，而且发送消息给一个对象对其他对象毫无作用．

构造到程序设计语言中的标准数据类型可以被当作类．当声明一个整型变量时，你知道它可以存储位于确定值祖不带有分数的数．更进一步，修改一个整数不会影响任何已经声明的整数．面向对象程序设计语言使你可以在严格控制类型的情况下创建自己的数据，恰好构造到语言中的那些类型．

对于内部数据类型，类可以使用其他类作为构造块．新类可以通过继承从老类构造．新类(即派生类)可以继承源或基础类的数据类和方法．新类可以把数据元素和方法增加到那些从其基础类继承来的数据元素和方法中去．任何类(包括派生的)可以有任意数目的派

生类．类的等级是通过继承机制而构造的．类等级通常看上去类似家族树，这正是基础集合被称作父类、派生类被称作子类的原因．

例如，我们要建立一套描述出版物库的类．主要有两种类型的出版物：期刊和书．我们可以通过定义页数、库目录号、版权日期和出版商的数据项来创建一个一般的出版物类．出版物可以被检索、存储和阅读．这些是出版物的方法．

接下来，我们定义两个名为期刊和书的派生类．期刊有卷号和期号，并按不同的作者有多个．这些的数据项都将被包括到我们的期刊定义中．另外期刊还需要一个独特的方法：订阅．专用于书的数据项包括作者姓名、封皮类型(硬皮还是软皮)及其ISBN号．正如你所见，书和期刊在共享出版物的性质的同时，还具有各自的特性．

我们的基础类、出版物、定义存储和检索数据的方法．但是，期刊可以被存放在活页中，而书被放在书架上．进一步来讲，找到要被检索的特定期刊的方法不同于找到书的方法．期刊是通过期刊清单指导来定位的，而书是用卡片目录系统找到的．我们可以为期刊创建一个"通过期刊指导查找"的方法．但这是处理这种情况的最好方法吗？

面向对象程序设计提供一种被称为多态机制(polymorphism)的有效手段来处理这些情况．当类被从基础类派生出来时，可能需要修改基础类方法工作的方式．在我们的例子中，我们发现期刊和书都可以被检索．但是期刊的检索方法与书的检索方法不同，虽然最终结果是相同的．多态机制让你可以定义适用于期刊和书的出版物检索方法．当期刊被检索时，专门用于期刊的检索方法被使用，而当书被检索时，相关于书的检索方法被调用．最终结果是单个方法名可以用于在相关的派生类上进行的同样操作，虽然该方法的实现在各个类之间是完全不同的．

多态机制与连接(binding)——方法与实际函数相联的过程——有关．当多态方法被使用时，编译程序无法确定调用哪一个方法函数．被调用的特定函数与函数要被发送到的类项有关，因此被调用的函数在运行时被确定．这被称作后期连接(late binding)，因为它是在程序正在执行时出现．早期连接(early binding)出现在某些面向对象程序设计语言的非多态(也称作静态)方法中．编译程序确切知道哪一个函数要被调用，因此在编译时就可以建立一个直接调用．虽然早期连接效率很高，但是大多数面向对象语言都为所有方法使用后期连接．

正如你可以看到的，面向对象程序设计给计算机程序设计引入了几个新的术语和概念．看上去都很有趣，但是这些东西对你真正意味什么呢？面向对象程序可以兑现其在提高软件开发速度同时提高可靠性的诺言吗？回答是肯定的．

面向对象程序是由可以重复使用的软件成分建立起来的．一旦一个类被完成并已被测试，它就可以被作为一个建筑块一样来构造程序．如果需要新的功能或修改，新的类可以从已有类派生；基础类的已证明可靠的性能无需再重新开发．程序员只要写新的代码而不必再写一遍原有的东西．软件变得易于测试．因为程序可以被分离到派生类的新代码中．

当一堆类被创建之后，则从这些类开发新的软件变得十分容易了．在一个组中工作的程序员可以把应用程序构造为一系列的类，最后把这些类合在一起形成最终程序．你也可以使用和改进其他项目成员开发的类，而不必对这些类的实际实现作修改，甚至都不必看它们．

类的另一个重要用途是隐藏实现细节．例如，对于不同的视屏显示类型图形类库可以

有不同的实现．使用图形类的程序可以用这样一个方法来写，即可以不作修改地在库实现已被移植到的任何计算机上可以编译的方式来写．

从现在开始，我们要深入去看 C++如何实现我们这里讨论的面向对象程序设计概念．如果由于新概念太多而还有点不清楚，则实际使用象 C++那样的面向对象语言而使你彻底理解．