C++ 高级编程技术

陈 媛 张晓刚 刘 军 张继录 编

電子工業出版社

C++高级编程技术

陈　媛　张晓刚
刘　军　张继录　编

电子工业出版社

(京)新登字 055 号

内 容 提 要

本书分十二章全面介绍 C++编程技术，包括 C++工具库，串类及动态存储分配，范围和下标的程序实现，数组应用技巧，数组在统计方面的应用，怎样具体使用永久性对象，散列表应用实例，树结构应用技巧，C++与汇编语言接口，窗口技术应用实例，图形技术应用实例等。

本书适合软件开发人员、大专院校计算机专业师生阅读。

C++高级编程技术

陈媛 张晓刚 刘军 张继录 编

审校 魏彬

责任编辑 张丽华

*

电子工业出版社出版(北京市万寿路)

电子工业出版社发行 各地新华书店经销

山东电子工业印刷厂印刷

*

开本:787×1092 毫米 1/16 印张:24 字数:604 千字

1994 年 4 月第 1 版 1994 年 4 月第 1 次印刷

印数:1—10100 册 定价:19.80 元

ISBN 7-5053-2228-1/TP・598

前　言

作为C语言和面向对象程序设计的结合体，C++一经面世，就得到了广大编程人员的普遍关注，同时迅速地获得了广泛的应用。我们知道，C语言在八十年代获得了极大的普及，但也正是由于其广泛使用才使得人们逐步认识到其缺点和局限性，即不适于复杂的程序设计。C++通过扩展C语言而支持新的软件开发概念，从而达到了改变C语言的目的。

本书并不是C++程序设计的教科书，而是讲述利用C++强有力的特性来创建程序构件和实现算法。通过本书的讲述和其中的大量实例，读者可掌握C++程序设计的具体方法，其中包括用C++实现各种常用数组、通过散列表索引文件、实现随机存取文件和动态二叉树、进行统计和科学计算以及处理动态串等。此外，本书还介绍了一些其它C++高级编程技术，如：C++与汇编语言的接口、窗口技术和图形技术。在MS C/C++和Borland C++环境中，都可直接运行本书中的程序。对要掌握C++编程技术的广大读者而言，无疑会从本书获得收益。

在本书的编写过程中，得到了许多同志的大力协助，其中参加编写工作的有陈媛、张晓刚、刘军 、张继录、黄晓宇、占学文、张明明、文丹岩、占卫兵等。

由于时间仓促，加上作者水平有限，不足之处一定很多，欢迎广大读者批评指正。

编者

1993年10月

目　录

第一章　C++概述

1.1　C++的来源

C++是C的一个扩展版本，C++首先是由Bjarne Stroustrup于1980年在贝尔实验室完成的。起初，他称这一新的语言为“带类的C语言”，而在1988年，他又把名字改为C++。

尽管C是目前世界上最受欢迎和使用最为广泛的编程语言之一，由于编程中一个最重要因素——复杂性的需要，导致了C++的发明。在C中，一旦程序从25000行扩展到100000行，它就变得相当复杂，很难组合成为一个整体。设计C++的目的就是要突破这一障碍，使程序员能够充分理解和管理更大型、更复杂的程序。

Stroustrup添加到C中的绝大多数内容支持面向对象编程（有时简写为OOP）。据Stroustrup说，C++的一些面向对象特性来源于另一个称为Simula67的面向对象语言。因此，C++代表着C语言编程和面向对象编程两种编程方法的综合。

在发明C++时，Stroustrup深知，最为重要的是保留C的原始精华，即有效性、灵活性和基本原理，与此同时再添加上对面向对象编程的支持。令人高兴的是他的目标都达到了。C++给程序员提供了C语言的灵活和控制，同时也具有面向对象的功能。用Stroustrup的话讲就是C++的面向对象特性“使程序构造得清晰、可扩充、易维护且不失有效性”。

尽管C++最初是为管理非常大型的程序而设计的，但实际上无法限制其使用，因为C++的面向对象属性可以有效地适用于任意编程任务。人们常常会看到C++用于象编辑程序、数据库、个人文件系统和通信程序之类的项目。此外，由于C++共享C的有效性，所以高性能的软件也可用C++来构造。

1.2　从函数到对象

在程序员们讨论起哪种语言最完美时，会发现各种面向功能的编程语言之间并没有本质差别，用户从BASIC语言转到C、Fortran及Pascal语言并没有多大困难，if语句就是if语句，函数就是函数，这与所用语言无关。在设计面向功能的程序时，不需要考虑特定的语言，因为各种语言间的语法和功能基本上是等价的。

面向对象语言的编程不同于面向功能的语言的编程，这是因为面向对象的程序在结构上不同于面向功能的程序。面向功能的程序围绕着要执行的动作进行组织，而面向对象的程序按照被操作的对象进行安排。对于习惯了面向功能编程的人员来说，这个转变就比较困难。

有些人说，C++是扩充成对象的C语言，因为从C到C++的过渡只是加进了一些面向对象的特征，它使得面向对象的技术应用起来快速、小巧、方便。但是，大家千万不要被这种说法引入歧途，C++不只是比C新增加了一些关键字。虽然我们能够用C语言写出面向对象的程序，也能够用C++开发出面向功能的程序，但面向对象的编程和面向功能的编程有着根本区别，即程序员必须从围绕数据类型及其交互作用的编程框框中解脱出来，如果不能深刻地理解这一点，就会陷入困境。因此，读者必须要有耐心，并要积极地开动大脑，同时要改变编写程序所用的老路子，这样才是通向C++的成功之路。

1.3 术语

重新定义术语可以建立新的规则，但也会因此变得神秘和难于理解。定义良好的术语将有助于说明某些事情，而新的思想也需要用新的术语来将它同旧的一切区分开，只是这些术语会给初学者带来一些困难。

让我们先从一个简单的代码段开始讨论：

```
double a, b, c, s;
  b = 1.0;
  c = 2.0;
  a = b + c;
  s = sqrt(a);
```

其中，double 类型是对浮点数的抽象表示，程序员在使用它的时候不必关心其格式，而由编译程序处理与 double 类型有关的细节，这就简化了程序员的工作。

这种抽象并不只是停留在数据类型一层上，函数抽象(如上例中的 sqrt 函数)就为确定浮点类型值的平方根提供了“黑箱”机制，使得多数程序员不需要了解怎样求浮点类型值的平方根，只需要引用 sqrt 就可完成这一工作。

C++允许程序员创建新的抽象。类(class)定义了与一系列有关的程序成份的接口，通过这种接口(这里使用的是 public 函数)，用户就能同所定义的抽象进行交互，而不必关心这种抽象的内部情况。就象使用 double 和 sqrt 能够更为方便地处理浮点类型数一样，用户在设计类时，也应该考虑到由它来简化对抽象的使用。

关键字 double 用来标识一个抽象，当程序员在程序中声明一个 double 类型时，就创建了一个对象，该对象具有 double 类型所具有的特性。术语对象(object)是字面上的，其具体含义是：具有可识别特性的事物。当我们创建 double 对象时，凭经验就可以知道该对象能够干什么。

内部数据类型(如 double 和 int)是 C++本身所具有的，它们可在任何时候任何地方使用，其特性是预定义的并且不能改变。程序员可以使用类在 C++中创建新的抽象，这样创建的抽象就具有用户所定义的特性。同内部类型不同的是，我们可以扩充类所定义的类型。

不要将类和对象仅仅作为一种数据类型。例如：Pascal 允许用户在外层 shell 函数中定义几个有关的函数，shell 为一进程提供一个入口点，这样内部函数能够执行这一进程，并能引用其中的数据类型，这实际上是在一个程序内又创建了另一个程序。

在 C++中，程序员可以使用类来完成上述工作，即创建一个类，类中私用(private)函数供内部使用，公用(public)函数提供入口点，各种函数所分享的数据可以是类定义的一部分。同 Pascal 一样，C++可以用来产生自包含(self-contained)进程。

封装就是将形成单个概念或实体的不同成份约束在一起，并隐藏程序成份的内部结构。例如：类 complex 定义了复数类型的数据结构，同时也定义了能够对 complex 值进行操作的函数，并使这些函数成为 complex 定义的一部分。

在日常生活中，我们经常会用到封装，如：由多个部件构成的汽车通常被视为一个实体。实际上，汽车也包含几个封装，“发动机”和“车闸”是由几个部件构成的系统，这些部件共同完成同一任务。把封装与抽象结合在一起，程序员就可以将某一较大范围内的各种构件组合在一起，形成单一的、简化的模型。

继承用来根据现有类创建新类。例如：Sortable Array 类从 Array 类继承来特性，其中 Array 类定义了数组的核心特征，Sortable Array 类将它本身的特性加进去以支持对数组的排序。同样，Int Array 类由 Sortable Array 类派生而来(即继承后者)，它创建了 Int 型的可排序数组。Array 类定义了所有数组的公用特性，Sortable Array 加进本身的功能来对数组进行排序，Int Array 类定义了一种特殊类型的数组，这样，使用继承性就建立了相关类的层次结构。

继承还允许多态性(polymorphism)，其含义是可采用多种格式的能力。在面向对象编程中，多态数据类型就是可以有几种变体的类型，并可用同样的方式对待它们。例如：Array 类就具有多态性，它所派生的任何类都可看成是类属 Array 的，同时允许程序以类属的方式处理 Array 的各种类型。

在后面创建类的过程中，我们将指出在何地和为什么要使用封装、继承性和多态性。

1.4 注意事项

我们常会听到这样一种说法：C 语言本身的灵活性会给程序员套上致命的绳索。幸运的是，优秀的程序员知道怎样避免把绳索套到自己的脖子上，经过实践，他们知道了在 C 中可以做什么和不可以做什么。

C++所允许的不可预知的自由表达式要求我们必须小心谨慎并彻底了解它。因为 C++不只是提供了一根绳索，而是把绞索放到了头顶，这种说法有点夸张但并不言过其实。

当程序员熟悉了 C++之后，情况就不会这么糟了。我们早已知道，某些编程技术并不是一种良好的编程习惯，多数 C++程序员只用该语言编写了几个月，最多是几年的程序，一般人还不能够区分哪些技术是好的、哪些技术是坏的。作者将根据自己几年来的经验教训，为读者提供一些防范措施。

1.4.1 C++的前景

每隔几年，总会有一种新的编程技术问世，并有可能使软件开发变得更为容易。而且，提倡者总是向我们许诺该技术会有很好的发展前景，但实际情况是这样吗？

在推广面向对象的编程语言时，我们也会听到这样的许诺。如：程序的构造将变得更为容易，因为可以将现有成份快速、方便地链接在一起；使用对象会简化程序的调试过程，因为问题可以在给定的数据类型中单独解决，而且，从现有的构件中派生出的新的程序以后只需要进行一些修改即可。

上述观点在理论上是成立的，但经过多年实践后发现，面向对象编程所提供的独特优点只能建立在开发者对所设计和开发的软件进行精心构思的基础上，这就需要花费更多的时间来分析应用程序，识别其构件，找出各构件之间的关系，并创建类的网络。

许多 C 程序员只是将程序堆积在一起，主要原因是缺乏充足的设计时间和良好的编程习惯，C 程序员习惯于在屏幕上编写程序，而没有对设计进行充分的考虑。程度员需要清楚地知道，要使用什么样的数据类型，怎样使用它们，以及它们之间的关系如何。如果在日常的工作环境中很难得到程序的规格说明书，或者程序开发是在匆忙之中，就最好不要使用 C++语言。

但是，C++所提供的环境还是很诱人的，它有许多不容易察觉的特性，以致用户即使花费几年时间来开发 C++程序也可能没有完全弄懂它。这既是优点又是缺点，优点是 C++功能强大，缺点是复杂性增大了出错的概率。

上述内容并不是说 C++有什么毛病，它只是提醒我们要小心谨慎地使用它，但也不要因

为某些权威人士告诫大家哪些可以做，哪些不可以做而缩手缩脚。只有不断尝试，权衡利弊，才能真正地掌握C++。

1.4.2 继承中会出现的问题

在进行面向对象编程时，首先要考虑的问题是怎样组织对象类。如果组织方面的错误在后期才发现，那将是很致命的。比如：发现某处需要的只是一个分枝而不是一个类，就象一棵树一样，我们不能砍掉并移动分枝，也不能改变它在树上的附着点。由于这个原因，在补进新类时，这种类层次常常会蜕化成类的聚集(thicket)。要想解决这个问题，可以创建多条祖先路径，实现多重继承。

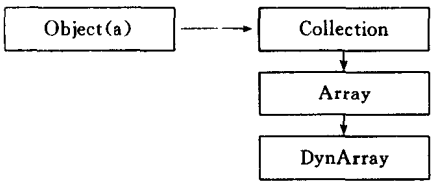

图1.1 厂家A的类层次

当我们用多重继承的方法创建一个新类的时候，一定要搞清楚那些被组合在一起的类的祖先是什么。例如：图1.1和图1.2分别展示了不同厂家所提供的库中类继承关系的一部分。厂家A的库所提供的是基本集成器(container)类，厂家B的层次则包含封装了数值数据类型的类(注意，两个名字同为Object的类是不同的)。

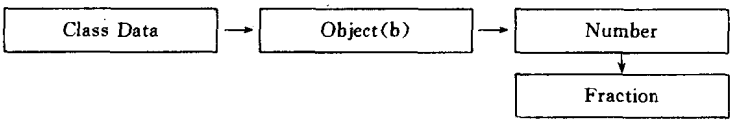

图1.2 厂家B的类层次

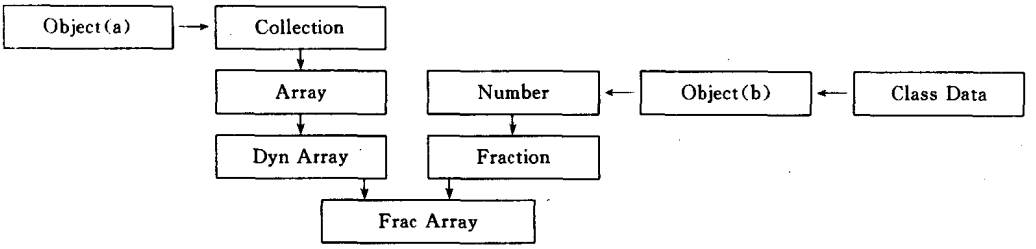

图1.3 试图将两种层次组合在一起

如果我们想把厂家A的Dyn Array类和厂家B的Fraction类组合在一起而创建一个新类，会出现怎样的情况呢？图1.3给出了FracArray类的继承关系。

表面看来一切正常，仔细研究就会发现，祖先中有两个名字为Object的不同类。如果这两个类是相同的，可以将Collection的Number的声明修改成虚基类。不幸的是，这两个Object类是不同的，虚机制无法起作用，编译程序会提示信息："基类具有二义性"；链接程序也会告诉我们："两个类具有相同的名字"。另外，假定在所讨论的例子中不包括头文件、源文件和目标文件命名相同的情况。

如果我们拥有厂家类库的源码，解决这个问题的最简捷办法就是将其中一个Object类换名，尽管这样做能立即凑效，但对类的重命名需要做大量的工作。因为Object很可能是许多其它类的基类，我们需要在引用Object的所有地方都做修改，即使使用的是功能很齐全的编辑器，这个工作量也是很大的。而且，每当该厂家将其类库升级到新的版本，我们还需要再做

同样的改名工作(否则只能忍痛割爱，不进行升级)。

在本书中，我们将假定不出现类名文件冲突的情况，如此处 Object 分别命名为 ObjectA 和 ObjectB，这样一切矛盾就都解决了。

如果只为眼前利益着想，使用单树形结构可以使对象的多态操作变得更为容易。多数 Object 类型的类都提供了虚函数来确定对象的类，以比较两个对象是否相等和获得对象的大小及对对象执行 I/O 操作。但不幸的是，尽管单树形结构的设计者可以对 Object 类的定义保持一致，但很少会对它的实现保持一致。

有些很简单的事情(如规定名字必须不同)也可能产生令人头疼的问题，如厂家 A 在 ObjectA 中定义了虚成员 函数 PrintOn，而厂家 B 的 ObjectB 定义了名为 PrintTo 的虚成员函数，在多数情况下，两者之间有着完全不同的语义。这样用户的 FracArray 类同时继承它们时要提供对 Frac Array 专用的实现，因为这两个厂家会对类标识、类等价和成员函数大小计算使用不同的约定和名字。这时，将会出现很大的混乱，同时继承两种以上的层次结构就更不可设想了。

1.4.3 单树形结构

一个大型层次结构就象一截弹簧。一小段弹簧掉在地上可以很容易地识别其首尾所在，若弹簧很长，要找到它的两端恐怕就不那么容易了。如果弹簧很长，沿着它的一头搜索下去直至它的另一头会很烦人，这就是复杂类结构不易理解的原因，尤其是在读别人写的源代码或者是在读自己很久以前写的程序的时候。

什么时候需要一数据项来标识其自身？是否每一对象都要支持 I/O 特性？类属数据结构对应用程序来说足够用了吗？使用层次化的虚函数需要多大的开销呢？要解决这些问题，面向对象的方法并不是唯一的解决途径。C++的最佳特性是允许使用以前所用的普通 C 语言，必要的时候使用 C++的对象。注意，只有必要的时候才加进 C++成份，而不是说要尽可能地加入。

1.4.4 至少要做两次

不知读者注意到没有，在编写软件的时候，过一段时间后我们会发现，现在比以前做得更好，尤其在进行一个大项目的时候，有时还会发现原来的设想是错误的。这时就会面临一种选择，或者进行大幅度修改，或者勉强进行下去，不管走哪条路，结果都不会十分令人满意。程序往往在第二次编写的时候才会工作得更好。

继承和多态性都不能掩盖掉错误的设计。权威人士总是说从已有类中派生出新的类会简化程序设计，这往往言过其实。试设想要建造 一座房屋，总体结构已设计好并已建完了一部分，突然发现地基有问题，这时不管怎样对地基进行修补，整个建筑还是有潜在缺陷的。同样的道 理也适用于类结构，如果基类有缺陷，派生类尽管能够隐藏它，却不能修复它。

程序常常是在第二次、第三次构思的时候才会更加全面和具体。在实际生活中，我们往往没有足够的时间来计划和预先设计好软件中遇到的所有可能情况。尽管由于种种原因(时间、经费以及程序本身的价值)一般不能重写现有的代码，但只要挪出一部分时间来重新建造(rebuild)，所得的程序就会好得多。

C++的程序设计风格不是随意性的，如果在自己的工作环境中很难得到程序的规格说明书，或者时间很紧迫，就不要采用 C++语言。

1.5 软件构件

计算机是由模块化元件构成的，一旦硬件被制造出来，软件就能操纵硬件并指示它如何

完成任务。有些热衷于面向对象程序设计的人提出，软件可以由一些软件构件构造出来，就象计算机能够由标准芯片和电路板构成一样。他们还说，这样做的结果将开辟软件生产的新纪元，程序员只需将各构件连接起来就可以形成新的应用程序。

这种假定当然是依据软件类似于硬件。给定的硬件元件是绝对存在的构件，一个机器有CPU、I/O 端口、指定数目的扩展槽。一旦一个硬件元件被选好，就几乎不能对它进行大的改动。例如：我们想在当前使用的 PC 机上加进一个新的扩展槽，就会发现改变主板是很困难的，加进扩展槽所需的工作量与其价值相比是不相称的。我们会面临两种选择：更换现有扩展槽，或者将就使用旧的槽。

要想改变硬件的工作方式是很困难的，对于 PC 用户来说，ROM BIOS 是固定的，更换成其它的 BIOS 也必须以类似的方式进行工作。一旦硬件做成或者软件固化进芯片中，再进行改变就很困难了。

如果离散的软件功能可以被封装进构件(component)中，我们就可以象制造汽车或计算机一样来制造软件了。编写程序的工作就成了开发固定的网络和不可改变的软件芯片(IC)。软件和硬件的唯一区别是：软件比硬件更容易重新组合。这样做的结果是减少了创造性，用软件芯片进行程序设计就象用积木进行建筑一样激动人心。坏的情况是：软件芯片的出现使程序员的首要位置从软件业中消失了，而没有人类的参与，程序就会失去生命力，缺少创造性和天才的火花，组装线式软件对有些人来说可能有用，但我们不提倡这样做。

一种更好的设计方法是创建灵活的、可扩充的对象。灵活性允许对对象进行修改。可扩充性是指将对象可以作为其它对象的基础。例如，本书中的 Array 类试图使用灵活的结构来提供灵活的扩充。

1.6 类之间的关系

两个类之间的关系可以为以下一种形式或多种形式的组合：

①is a 关系：类 B 定义为类 A 的变体，类 B 的主要特性继承于类 A，而且常常是通过指向类 A 对象的指针来引用类 B 对象，以此实现多态性。例如，定义 int 型数组的类可以从另外一类派生而来，该类指定了所有数组类型的公共特性。

②modifies 关系：类 B 扩充或定义了类 A 的功能。例如，本书中的 SortableArray 类将排序功能加进了 Array 类中。

③made of 关系：类 A 对象为类 B 的构件，这时，类 A 定义了传统的数据类型，比如：整数或复数，类 B 定义的对象类型包含类 A 的对象。

④uses 关系：类 A 对象是类 B 对象所使用的工具，一个对象使用另一文件对象来存贮数据。

程序员必须知道程序中类的关系以确定类的类型和交互，因此常常需要查询完整的规格说明书，如果对类的描述不清楚，就会使程序难于理解和维护。

第二章　工具库

每个类库都包含一组实用类和数据类型。许多公用的、简单的工具可以被反复使用，将它们加到标准库中可以免除重新创建的工作。在进行深入讨论之前，首先要说明一下影响类设计的内容，因为类不是独立存在的。

2.1　开关和布尔型

程序构件并不是越复杂越有用。例如，在本书库中出现的最简单数据类型(Switch)就可以用一行代码来实现，结果表明，它特别有用。

在实际工作中，人们常常需要用一个值来表明某个东西的开或关，这又同布尔表达式不完全一样，后者关心的是某件事是真还是假。尽管在技术上来说开/关值同真/假值是相同的，但二者的物理含义却不同。人们不会说一个灯是真还是假，也不会说一个逻辑变元是开还是关。

2.1.1　C 风格的实现

on/off 开关是怎样实现的呢？许多 C++ 程序员会写出类似下面内容的代码：

```
#define    Switch    int
#define    OFF       0
#define    ON        1
```

或者定义成更为复杂的形式：

```
typedef int Switch;
```

上述设计均缺乏保护机制，无法避免将非法值赋给 Switch 的情况。而且，#define 常量是绝对的，在 C++ 中效率不高，因此要用更好的工具。

2.1.2　使用枚举类型

为简化起见，我们可以使用枚举类型来更好地实现 Switch，如：

```
enum Switch
  {
    OFF = 0,
    ON = 1
  };
```

枚举类型是容易被人忽略的 C++ 特性，就是在 C 中也一样。大多数 C++ 程序员最初都是 C 程序员，他们常常在 C++ 中忽略这一完成任务的最有效方法。

使用 C++ 的类来定义 Switch 类型未免大材小用。利用枚举类型可以简洁地实现 Switch，不需要提供构造函数和成员函数，它就能够提供保护措施来防止非法赋值，而且易于使用，同时不需要源文件来定义 Switch 类所使用的 On 和 Off 常量。检查赋值合法性的工作是在编译期间而不是在运行期间进行的。使用类似的枚举类型还可以定义 Boolean (true/false)数据类型，如：

```
enum Boolean
  {
```

```
    BOOL _FALSE,
    BOOL _TRUE
  };
```

既然 C++支持同样的布尔值系统(非零值为真，零值为假)，为什么还要使用枚举类型呢？这只能说是一种喜好，Boolean 型和 Switch 型将其值分别限制在一对说明性标识符范围内。

在 Boolean 类型常量的前面为什么还要加上 BOOL _前缀呢？因为在最初使用常量名字 TRUE 和 FALSE 的时候，我们发现许多其它的头文件中都有命名为 TRUE 和 FALSE 的 # define 常量。处理程序的常量会重写 Boolean 型的常量定义，从而导致语法错误。

Switch 和 Boolean 类型实现在 switch.h 和 boolean.h 文件中，详见本章后面的代码清单。

2.2 错误报告

C++的最新定义支持程序员定义的异步错误处理，它是通过异常处理程序来实现的，一般来说，异常处理程序是一个函数，它在指定事件发生时被调用，例如，除数为零。

不幸的是，异常处理是新引进的思想，多数 C++编译器并不支持它，其中包括常用的 PC 版编译器。由于不管使用的是哪种编译器都需要进行错误处理，因此我们需要创建一个不同的系统。

一般而言，类应当以一种或几种方式进行错误处理。对于简单的类，错误可以报告给该类的任意对象，这也是内部类型的工作方式。例如，在被零除时，可以调用全局函数来显示该信息。而对于复杂的对象，比如链表，就应当将错误信息返回给遇到该错误的对象。

我们建议大家创建一个错误报告对象，在类需要使用类范围内的错误处理的情况时，可以类中包括静态错误报告程序，而在类需要使用特定于对象的错误处理情况下，可以为每一对象包含一成员函数。下面给出所创建的类的定义：

```
class ErrReporter
  {
  public:
    ErrReporter(const String * lead);
     ErrReporter();
    virtual void Warning(const String & msg);
    virtual void Fatal(const String & msg);
  protected:
    virtual void MsgOut(const String & msg);
    String * Leader;
  };
```

ErrReporter 有六个成员：构造函数、析构函数、三个虚函数和一个指向串的指针。

在决定怎样表达类的时候，我们遇到了一个棘手的问题：ErrReporter 类使用了 String 对象，而此处的 String 类又要用到 ErrReporter 对象，这种互相依赖关系在 C++中是很常见的，这使程序很难办。应当先说明哪一个类呢？这里我们首先给出的是 ErrReporter 类，因为

它比较简单。

不管怎样，读者已经对 ErrReporter 类有所了解了，这里还需要对 String 对象提供一些基本信息。String 类所定义的数据类型包含一串字符，下一章将给出完整的说明。我们定义 ErrReporter 类的目的是试图说明以下一些有关 String 的事情：

①String 对象可以由 const char *S 构造；

②String 可以用在 const char* 的位置上；

③String 类定义了复制构造函数；

④String 类定义了流 I/O 函数。

2.2.1 ErrReporter 的成员

ErrReporter 对象只有一个实例变量 Leader，Leader 是一个指向 String 对象的指针。当通知 ErrReporter 显示一写错误信息的时候，首先显示 Leader 所指向的 String。Leader String 通过给出一个公共的标题正文来标识相关的错误信息。

ErrReporter 的构造函数创建了由 lead 参数所指向的 String 副本，所复制的串由 new 进行分配，并将指针赋给 Leader。如果 lead 为 NULL，Leader 也被赋成 NULL。

```
ErrReporter::ErrReporter(const String * lead)
  {
  if( lead == NULL )
    Leader = NULL;
  else
    Leader = new String( * lead);
  }
```

为什么不是简单地将 lead 赋给 Leader 呢？因为如果指向对象的指针超出了类的范围却还要存贮它，我们认为这是一种不好的程序设计习惯。如果 lead 指向的 String 被删除了，那么 ErrReporter 对象中的指针就是不合法的了。

如果 Leader 不是 NULL，则结构函数删除 Leader 指向的 String，如：

```
ErrReporter::~ErrReporter( )
  {
  // delete leader if it was allocated
  if( Leader ! = NULL )
    delete Leader;
  }
```

ErrReporter 将其余三个函数定义成虚的，这三个函数都实现成空的嵌入(inline)shell。

```
void ErrReporter::Warning(const String & msg)
  {
  // does nothing!
  }
void ErrReporter::Fatal(const String & msg)
  {
  // does nothing!
  }
```

```
void ErrReporter::MsgOul(const String & msg)
  {
  // does nothing!
  }
```

在显示错误信息的时候，程序或者调用 Fatal 或者调用 Warning。Fatal 用来终止程序，而 Warning 只是用来显示一条信息。由于这两个函数都可以显示信息，所以我们可以创建第三个函数 MsgOut 来实现真正的信息显示。

在最初的实现中，Warning、Fatal 和 MsgOut 都是纯虚函数。纯虚函数使 ErrReporter 成为一抽象基类。但由于不能创建抽象对象，所以不可能创建对抽象类的引用。读者在下面几章中将会发现，对 ErrReporter 对象可以进行引用，因此，这里实现三个空的虚函数而删去纯虚函数指示符。

使用虚函数的目的是为了在特定的环境下构造 ErrReporter 类。环境不同，显示信息的方式也可能有所不同。例如，DOS 的命令行程序常常将错误信息显示成滚动的正文行，而窗口环境中的程序要在一个窗口或显示区中显示错误信息。大多数对象都不会知道它所工作的环境，因此用与当前环境相兼容的方式显示一条错误信息就成了 ErrReporter 对象的工作了。

如果需要对象在多种不同的环境中以不同的方式进行工作，则要使用多态性工具。此处将 ErrReporter 类的错误显示函数定义成虚的，然后由 ErrReporter 派生出适用于各种特定环境的派生类，最后，在程序中定义指向 ErrReporter 对象的指针，并将不同环境下 ErrReporter 的派生类地址赋给它们。对于所有由 ErrReporter 派生出的类对象的引用，可以使用指向 ErrReporter 的指针来实现公共接口。

2.2.2 C++流的错误报告

为说明清楚起见，我们用 C++流定义一个名字为 Dos ErrReporter 的类，如：

```
class DosErrReporter : public ErrReporter
{
public:
  DosErrReporter(const String * lead =NULL;
                 ostream * strm =NULL);
  virtual void Warning(const String & msg);
  virtual void Fatal(const String & msg);
protected:
  virtual void MsgOut(const String & msg);
private:
  ostream * Destination;
};
```

DosErrReporter 由 ErrReporter 派生而来。我们定义了四个函数来反映出 ErrReporter 所定义的特性，并且加进了一个新的数据元素 Destination，它是指向 ostream 的指针。Destination 没有它自己的构造函数。

DosErrReporter 的构造函数有两个参数：String 指针和 ostream 指针。String 指针传递给 ErrReporter 的构造函数，如果 strm 为 NULL，Destination 就被置成预先定义的 cerr 流，否则，Destination 就被赋成 strm。如：