Visual C++

高级编程技巧与实例

沈刚 魏爽 李旭东 房波 等编著

机械工业出版社

Visual C++高级编程技巧与实例

沈　刚　魏　爽
李旭东　房　波　等编著

机械工业出版社

前　言

作为C语言面向对象的扩展，C++语言不仅继承了C语言的灵活、方便、高效和移植性好等特点，而且还具有了封装性、继承性和多态性等面向对象的特性，因而它深受广大编程人员的欢迎，现已成为使用最为广泛的一种编程语言。

Microsoft公司推出的Visual C++系列产品，在C++语言的基础上集编辑、编译、调试、运行和剖视优化于一体，是目前最为成功的一种C++语言产品。

本书将详细介绍Visual C++的编程方法。为便于读者理解，本书将从C++语言编程基础讲起，然后对面向对象编程技术和常用C++类做一简要介绍，接下来介绍Visual C++的具体编程技巧，最后给出Visual C++的编程参考信息。

为使读者对在Microsoft最新32位操作系统Windows 95上开发Visual C++程序有所了解，本书还对Windows 95编程技术做了具体介绍，这样就使读者可以方便自如地在各种常用的系统环境（如Windows 3.1和Windows 95）下用好Visual C++。

本书由多人合作完成，其中参加编写工作的有沈刚、魏爽、李旭东、房波、舒林华、尤树民、常文东、王丽、张阳、王立芬、李俊红、吴小云、沈晴、刘志伟、熊静、江涛、郭允和赵强等。由于时间短，作者水平有限，难免出现疏漏，欢迎大家批评指正。

作者　1997.7

目　录

第1章　C++编程基础

• 本章介绍C++语言的一些基本功能，如它的基本数据类型的声明和表达类型、功能以及程序中的各部分特征等。其中大部分功能在以后的章节中都有更详尽的说明。为使以后各章容易理解，本章就此做一概述。

• 对于已有用C语言编程经验的读者来说，对这里的许多说明都是十分熟悉的。即使读者没有使用过C语言，也很容易读懂。学习写C++程序最好的方法是循序渐进地学习。

1.1　C++程序入门

为对C++程序有感性认识，先来看一个“Hello”程序。这个程序说明的最重要的几点是：

1. main（）函数
2. 命令
3. 说明
4. 预处理器指令

Hello程序的内容如下：

```
//Hello program
#include<iostream.h>
main ()
{
    cout≪"Hello, hello, already!";
    return 0;
}
```

1.1.1　Hello程序的运行

在进一步讲述之前先来看看Hello程序的运行。请按以下步骤来做：

（1）若尚未建立\MSVC\MYFILES目录，则应建立该目录。

（2）从文件菜单中选择“New”，并在“Workbench”下建立新文件。此时Workbench将打开一个空的文件窗口。

（3）键入Hello程序，检查键入结果是否有误。

（4）在\MSVC\MYFILES目录下用Hello.CPP保存该程序文件（扩展名“CPP”无须键入）。

（5）从Project菜单中选择“Build”，为程序建立.EXE文件。

（6）从Project菜单中选择“Execute”来运行程序。

1.1.2 main () 函数

如程序 Hello.CPP 所示，一个 C++ 程序的核心部分为 main 函数，即 main ()，一切程序行为都将从这里开始。不论一个程序有多大，也不论 main 函数具体出现在源代码中的什么地方，程序的具体执行将开始于 main () 函数的第一行，并结束于 main () 函数的最后一行。任何一个 C++ 程序都必须含有一个 main () 函数或与之等效的函数，如用于 Windows 程序的 WinMain () 函数等。

所谓 main () 函数，当然是指一个函数。但这是什么意思呢？从最基本的原理来说，一个函数是由一组为达到一个明确定义的目的而组合在一起的工作语句集组成的一个简单的名称。在 C++ 程序中，用户可根据自己的特定要求建立不同的函数。进一步来说，用户可以用 C++ 的标准函数，如 main ()、cout 等，来达到一般的工作目的，如从键盘获取信息或将信息显示到屏幕上等。

在设计程序时，必须时刻记住产生函数的规则：每个函数需要完成一个特定的、明确定义的任务，这一任务可以简单到给出一个数值，也可复杂到根据程序数据完成一系列的运算，但必须能够用一简短而有意义的句子表达其函数，并且一个函数不可超过一页长。main () 是一个函数，因为其名称后紧跟一对左右括号。这是 C++ 中特定函数名区别于其他变量或程序变元的标准表达方式。在以后将要给出的 C++ 函数中，我们将尤其注意这一点。

显然，单单有一个 main () 函数还不足以构成一个有用的 C++ 程序，main () 函数中必须有些其他东西完善其函数。如下列程序所示，Empty.CPP 是一个完全有效的 C++ 程序，但却完成不了任何任务。

```
//Empty. CPP: a program that does nothing
void main ()
{
}
```

为使程序完成一定的任务，程序中应包含语句行，如 cout≪"Hello"或其他类似的东西。在程序 Empty.CPP 中，main () 函数未包含任何语句，因此不完成任何任务。程序行前端的两条斜线为非程序部分而被 C++ 完全忽略，因此也不能完成任何函数。

上述程序与程序 Hello 相比，还有两处有趣的不同点。其一，用户也许已经注意到程序 Hello 中命令行 #include＜iostream.h＞紧接在起始命令行后，这一命令行称为预处理器指令。它告诉 C++ 在编译标准代码前在该点插入 iostream.h 文件。换句话说，程序 Hello 之所以要用该文件（被称为头文件）是因为它包含了 main () 函数中的 cout 语句中的编译信息。因为程序 Empty 在 main () 函数中没有任何语句，所以就不需要这样的预处理器指令（如用户对 IOSTREAM.H 程序内容有兴趣的话，可在硬盘的 \ MSVC \ INCLUDE 目录下找到）。

程序 Hello 与 Empty 间的另一显著特征是表达 main () 函数的方式不同。在程序 Hello 中，只简单地用了 main 一词，后面分别跟左右括号，在程序 Empty 中，用了 void main () 来表达该函数，这是为什么呢？

原因和大多数函数一样，main（）函数同样需要有一个返回值。当一个函数运行并停止后，它应在程序中调用它的地方输出一个数值。这一返回值可用于检查函数运行是否正常，在程序 Hello 中，main（）函数的返回值为零，这表明函数成功地完成了我们期望它做的任务。

但是，送回一个返回值意味着在代码中有返回语句。因此，为使程序 Empty 运行，main（）函数的返回值被预先标识为虚的，这是为了告诉 C++，该函数没有返回值。对于 Return 语句来说，还有其他的用途，但这一作用是最重要的。

从这一点来说，用户可以发现 main（）函数是非常典型的。多数 C++（或 C）函数都将根据程序的需要有返回值。有时，返回值是使用函数最重要的原因，否则，可忽略返回值。还有一些时候，尽管返回值不是使用函数的主要原因，但它却是一种非常有价值的检查函数运行正常与否的方法。

1.1.3 程序语句

语句是 C++ 程序的核心内容。在程序 Hello 中有两种语句。一种是 cout 语句，它将程序中的文本信息传送到 PC 屏幕；另一种是 Return 语句，它用以控制并传送给主程序零值，以表示程序运行成功而结束。

请注意两点。一是语句均以分号结束，这是 C++ 中结束一个简单语句的标准方法。同 C 与 Pascal 一样，C++ 是一种语句定位语言，而不是行定位语言。一般说来，其含义是在 C++ 编译程序源代码时，断行并不产生影响。用户既可每行写一个语句，也可一行写几个语句，只要中间用分号隔开。每个分号结束一个单一的语句。在 C++ 中，分号还有另一些值得讲述的作用，这一点如同它在 C 和 Pascal 中一样。举例来说，若定义了一个结构类型，则需用左右括号分别表示结构定义的首尾。定义之后，还需在类型定义右括号后加一个分号，如下所示：

```
struct employee
{
  char fname [10];
  char lname [10];
};
```

请记住，在 C++ 编程人员中，通常共同犯一个错误，那就是忘记了在结构定义的右括号之后加一个分号。

但也存在另一种情况，即在使用左右括号来包含某些语句时，不必在右括号后加分号，如下所示：

```
for (int i=0; i<10; i++)
{
  cout<<"This is number"<<i<<'\n';
  cout<<"----------------------";
```

```
}
```

因此，一个经验的说法是，分号标志着语句的结束。若一个复杂语句已由右括号结束，则不必加分号。一个数据定义，从另一个角度来说，其本身也可看作是一个简单语句，因此，其后需用分号表示结束。现在来看看这一原则的实例。仍然用以前的方法，在 struct 定义中，每一行只简单命名了一部分结构，并给出了其数据类型，却不执行任何语句，在 for 循环中，每行均有一个执行语句（用分号结束），在这种情况下，它将在屏幕上显示一些文本。于是现在对这一事实已有了足够的感性认识，即 struct 定义需用分号结束，而对于 for 循环则不必使用分号。不过，对大部分情况来说，现在就需建立一种概念，即使用分号。在使用 C++时应开始建立一种何时使用、何时不使用分号的思想。

main（）函数中另一值得加以注意的问题是，它由一个左大括号开始并由一个右大括号结束。与分号不同，这并不是必需的，只是在用户需要将两个或两个以上语句组合在一起时才使用，它的作用相当于 Pascal 程序中的 BEGIN 和 END。

因为函数需要将语句组合在一起，所以在定义函数时通常需要用到左右大括号，这一点正如我们在程序 Empty 中看到过的，即当 main（）函数为空时，里面没有任何语句。因此，虽然括号有时并不是必需的，但在简单语句外加括号也没有什么不可以的。例子如下：

```
for (int i=0; i<10; i++)
{
  cout<<"The current number is"<<i<<'h';
}
```

上例中，因为循环中有一个语句，所以大括号此时并不是必需的。另一种更简单而有效的方法是：

```
for (int i=0; i=<10; i++)
cout<<"The current number is"<<i<<'h';
```

注意到 C++是语句定位而不是行定位，因此也可将上述循环写成这样：

```
for (int i=0; i<10; i++)
cout<<"The current number is"<<i<<'\n';
```

1.1.4 注释语句

在程序 Hello 和 Empty 中的开始处都有一句解释程序是干什么用的和给出其硬盘文件名的注释语句。注释语句并不是必需的，但对优秀的程序来说，这是一个解释源代码的好方法。C++有两种产生注释语句的方法：一种是沿用 C 的习惯，用/*...*/；另一种是它本身的双斜线。

旧的C语言注释语句由/*符标志开始，*/符标志结束。其中，开始符/*和结束符*/之间的部分被C和C++编译程序忽略，如下所示：

```
/* prompt for user's name */
...
printf ("Enter your name:");
gets (name);
... /* and so on */
```

C++既可用C语言注释语句，也可用由双斜线//开始一行的注释语句。加入双斜线"//"后编译器将忽略斜线后该行的所有字符。使用单行注释的一个明显例子如下：

```
printf ("Enter your name:")      //prompt for the user's name
gets (name);      //get user's name from keyboard
//... and so on.
```

1.1.5 预处理指令

程序Hello中包含一个预处理指令#include<iostream.h>。在程序中，它并不是一个真正的语句。它的作用是告诉C++在编译前，必须先将iostream.h文件的内容插入到源代码中#include出现的地方。

预处理指令的函数并不仅限于向源代码中插入文件（称为包含文件），它们也可用于定义常数名或指明编译器在特殊情况下的处理方式，以及将用户代码中的错误部分提出删除。在以后的叙述中，将给出预处理程序语句的其他作用。

通常情况下，#include预处理程序语句的用法都是同它在Hello程序中的用法一样的，即将需插入的文件名写于两个尖括号中间。但有时，插入文件名也会出现在引号中间，如#include "myutils.h"。

上述不同的差别在于Workbench知道了它的标准头文件在何处，如iostream.h。当把它们调入#include预处理程序语句时，它们的文件名出现在尖括号中，用来告知Workbench这些作为头文件的文件名可在某些标准目录下找到，如\MSVC\INCLUDE或\MSVC\MFC\INCLUDE。

然而，当文件名出现在引号中时，标志着Workbench应在当前目录下寻找文件。这一函数对于一个专用项目的头文件很有用，用户可将所有与该项目有关的文件存于同一目录下。当然，上述两种引入文件的方法并不是互相排斥的。在同一程序中，既可用括号包含标准头文件，也可用引号包含其他头文件。

1.1.6 源代码格式化

对于Hello程序，最后要说明的一个问题是格式化C++源代码的标准方法。除了个别情况有所例外，源代码的格式方式与C++本身完全没有区别。因此，程序Hello可以这样

来写：

```
//Hello program
#include<iostream. h>
main () {cout≪"Hello, hello, already!"; return 0;}
```

对于格式化源代码唯一需要考虑的问题是格式化后的易读性和易理解性。由于这一点缘故，格式化源代码没有更强更快的方法，本书中所有举例程序的格式化方法都是为使其源代码更清楚。用户既可沿用本书中的方法进行标准格式化，也可用自己平时惯用的方法进行标准格式化。C++和C忽略间隔、空格、空行和其他源代码中的书写空格，所以用户可用自己认为最好看的方式来书写程序。

有几种情况下格式化问题是不同的。例如：在上面非格式化的程序中，在第一行有个说明，第二行有预处理指令，只是在第三行我们才书写了真正的C++源代码，上述源程序必须这么写。双斜线说明标志右端的一行字符都将被C++忽略，因此在同一行所书写的所有说明都不被认为是C++程序的一部分。同样，预处理指令不是真正程序的一部分，因此每个预处理指令必须写在它本身的一行上。当用户想建立一个函数似的宏而使用#define预处理指令时，格式化将是个更复杂的问题。

对于C++不讲求格式化方式的问题，此处还有一个例外，即用户不能将引号内的书写内容断行。下面的程序将被C++认为是错误的而拒绝：

```
cout≪"Hello,
hello already!";
```

为避免这一错误，用户可在第一行后加入单斜线（\），如下所示：

```
cout≪"Hello,  \
hello already!";
```

这样表明引号间所有的字符包括空格都是文本的一部分，它们将缩进并显示于屏幕上。

1.2 变量和常数

假如C++程序的所有函数仅仅是将一些文本显示于屏幕，那么C++也就不值得学习了。在C++和其他程序语言中，一个非常有用的函数是可在程序中定义变量和常数。

一般来说，变量是指在计算机内存中可以存储有效数据类型数据的命名地址。每一种变量都决定了它的里面可存储某一种信息。下面的程序给出了一个包含有变量的简单程序。变量在程序的第3行中给出定义。

```
// Var. CPP: A simple program with a variable declaration
#include<iostream. h>
int MyNum; //This is the variable declaration.
```

```
main ()
{
  cout≪"What' s your favorite number?";
  cin≫MyNum;
  cout≪"You said your favorite number is"≪MyNum≪".";
  return 0;
}
```

请注意，上面程序中首先给出的是变量类型：int 表示该变量可取整型值。程序的剩余部分很简单。main () 函数的第一行在显示屏上显示了这样一条信息，询问用户最喜欢的数字，当用户键入数字后，下一行即从键盘获取数字并把它装入 MyNum 变量，再下一行又显示于屏幕上：先是文本，然后是存储于 MyNum 的整型值，然后是一个句号。最后，由 return 0 行结束程序。

1.2.1 变量定义方法

建立一个 C++ 变量应告诉 C++ 如下内容：

1. 变量的数据类型，如整型、字符型或结构型；
2. 变量的名称、形式必须符合 C++ 规定；
3. 有时需要给出变量的初值。

如果用户未提供上述信息，C++ 将拒绝编译程序。举例来说，程序 VarL.CPP 与程序 Var.CPP 非常相似。只有一点不同，即只是在程序的最后定义了 MyNum 变量。先来看程序：

```
//VarL. CPP: Fails to declare the MyNum variable before its first use
#include<iostream. h>
main ()
{
  cout≪"What' s your favorite number?";
  cin≫MyNum;
  cout≪"Yor said your favorite number is"≪MyNum≪".";
  return 0;
}
int MyNum; //This is the variable declaration - - TOOLATE!
```

对于一般人来说，可以在程序中很容易看出 MyNum 是什么，但对 C++ 来说，却是一个大障碍，当它在第 6 行和第 7 行看到 MyMum 时，因为未定义，所以认为是一个错误并停止编译。

现在已经看到，未定义变量就出现了变量名，会给程序编译带来什么样的结果。尽管在使用变量、常数和程序其他元素之前都必须先定义它们，C++ 还是给出了一些比 C 更灵活

的定义手段。有三种定义变量的方法是用户所必须掌握的。

1. 在文件和代码块的开始处定义

这是 C 定义变量的标准方式。如在程序 Var.CPP 中所示变量在源代码文件的前端定义，或在函数开始前定义。

2. 在用的时候定义

这是 C++ 比 C 稍灵活一点的地方。例如，下述程序是为控制 for 循环而定义变量的 C 的标准方式。

```
int counter;
for (counter=0; counter<10; counter++)
{
  // do something
}
```

而 C++ 可允许用户使用变量时再定义，这给用户提供了更清楚、更有效的方法，如下所示：

```
for (int counter=0; counter<10; counter++)
{
  // do someting
}
```

3. 定义时初始化

在 C++ 和 C 中应用的一种重要情形是，用户可在定义变量的同时完成对变量的初始化。当然，是否给出变量的初值是可选的，但这样既可节省时间又可使代码读起来容易些。下面是分两步定义变量并确定初值的方法：

```
int iMyNum;
iMyNum=10;
```

接下来是用一步完成上述两步的方法：

```
int iMyNum=10;
```

当然，这些情况都很简单，因为整型变量就是很简单的。不过，也可用这种一步定义法来定义更复杂的 C++ 变量，如结构型数据。

1.2.2 变量和其他程序元素的命名

如果一个变量被命名了内存地址，那么我们将给它怎样的名字呢？C++ 对变量、函数

和其他程序元素的名称定义更加严格，这些名称叫做标识符。一些基本规则如下：

1．标识符是一个可包含字母、数字和下划线的序列。

2．标识符的第一位必须是字母。

3．标识符对字母的大小写敏感：与 Pascal 不同，C＋＋标识大小写字母。例如，MyNum、mynum 和 MYNUM 在 C＋＋中分别是三种不同的变量。

4．标识符的长度可随便定义，但 C＋＋只承认前 32 个字符。

5．标识符不能与 C＋＋关键字相同，如 if、char、switch 等。Visual C＋＋又增加了新的关键字，这些字同样不能成为程序的标识符。表 1-1 和表 1-2 列出了标准 C＋＋关键字和 Microsoft C＋＋增加的关键字。

C＋＋语言有它自己使用的词汇来完成其操作，这些词汇称为关键字，有时称作保留字。用户在定义变量和其他程序元素时不得使用 C＋＋关键字，Visual C＋＋增加了少量新的关键字，也是用户不能进行命名的。

表 1-1　标准 C＋＋关键字

asm	float	signed
auto	for	sizeof
break	friend	static
case	goto	struct
catch	if	switch
char	inline	template
class	int	this
const	long	throw
continue	new	try
defaut	operator	typedef
delete	private	union
do	protected	unsigned
double	public	virtual
else	register	void
enum	return	volatile
extern	short	while

表 1-2　Microsoft C＋＋关键字

asm	far	interrupt	saveregs
based	fastcall	loadds	segment
cdecl	fortran	near	segname
export	huge	pascal	self

还要说明的是，有些标识符，如 cout，用户可将其用来做自己的变量名，但最好还是不要使用它们，以免既迷惑了自己，又迷惑了 C＋＋。

现举一些合法的 C++ 标识符的例子：

MyNum

TheName

a_very_long_name_for_something

theNumber2

再举一些不合法的 C++ 标识符：

switch　　//the same as a C++ keyword

2Livenew //doesn't start with a letter: no talent

以上说明的是怎样定义 C++ 标识符的问题。但是，何时定义、何时不该定义这些标识符，又是另一问题了。一般说来，变量和函数名可根据使用情况任意用大小写字母。例如，可用 MyNum 表示整型变量，可用 GetUserInput () 表示通过键盘获取用户信息的函数，所有大写字母常被用来表示常数，不论在它们的前面是否是 const 或 # define 预处理指令。

匈牙利注释是流行得最快且最有用的命名变量的一个方法。不过，这与匈牙利并无丝毫关系，原籍匈牙利的微软程序家 Charles Sinoryi 发明了这一法则，就在 Sinoryi 命名了它之后，他利用变量名将系统分成若干部分；每一名称下只有一种变量，表 1-3 给出了命名法则。

1.2.3 C++ 数据类型

除非是一位很有经验的程序员，否则，表 1-3 中的某些数据类型看起来将是陌生的。事实上，即使是熟练的程序员也未必了解其中一些专用于 Windows 程序的数据类型。

C++ 同时支持简单和复杂的数据类型。简单类型是预先定义的，如整型或浮点型。复杂型数据类型如枚举、结构、数组等，留给程序员自己根据习惯来定义。

表 1-3　命名变量的匈牙利法则

前　缀	含　义
b	布尔型（0 = FALSE，1 = TRUE）*
by	字节型（无符号字符）*
c	字符型（通常是一个字母，但也可以是一个字母、数字和特殊字符）
dw	字类型（一个无符号长整数，占据四个字节内存）
fn	函数
h	句柄（Windows 用来跟踪程序元素的整数）*
i	整型
l	长整型（允许整数和指针拥有更多的有效数字）
n	整数或短整数（在 PC 上两者相同）
p	指针（指向一个程序元素的内存地址）
sz	字符串型（一个以空字符结尾的文本字符串）
w	字（两个字节）或者无符号整数

注：用 * 标志的类型用于 Windows 程序设计

在使用数据类型时，有一个需要十分注意的问题。如上所述，变量的数据类型决定了C++希望得到什么样的事物：整型数就必须读入整数，浮点型可读入小数等等，同强分类语言 Pascal 不同，C++和 C 对程序中的数据类型并不做严格检验。在 Pascal 中，如果想用整型数变量读入字符值，编译器就不会继续工作。

同样情况下，C++和 C 却不会这样对待用户。如果用户给出了与定义变量类型不同的值，C++和 C（属弱分类）便假设用户自己知道在做些什么。这一点给程序增加了很大的灵活性，不过同时也必须付出双倍的程序检查工作。

1.3 语句

C++有几类不同的语句，前面的程序曾用 cout 和≪控制符将文本传递到计算机屏幕。

1.3.1 I/O 语句

I/O 语句在用户希望将数据从一处传送到另一处时均可使用，如从硬盘传至显示屏幕、从键盘送入 PC 或从 PC 内存送入已标记的微处理器。

1.3.2 赋值语句

赋值语句是另一种经常使用的语句。它同 I/O 语句稍有不同：它不是将数据从一处移到另一处，而是将数据传至程序元素或从程序元素中取出，例如，语句 iMyNum = 10 是一个赋值语句，它将数据值 10 放入 iMyNum 变量中。

1.3.3 比较语句

比较语句有时不被视为语句而被视为表达式，不过从它们在 C++中担任的角色来看，当然还是应归入语句类。当 C++程序（或其他程序）需要决定从一个方向转入另一方向，那么通常是通过比较语句来完成的。if 语句就是一个很好的例子，在 C++中可这样写：

```
if（iMyNum = 10）    //do something
```

在上面的情况中，“something”只有在 if 比较语句 iMyNum = 10 的情况下才会进行。如果比较结果为真，则 if 语句中的任务将完成，如果比较结果为假，则不执行。在 Windows 程序中，比较语句同样担任重要的角色。它们用于决定程序将怎样响应外部事件。

第2章　面向对象编程与C++类

类是C++语言针对面向对象编程（OOP）新增加的一项函数，它是C++语言的重要组成部分。在这章中将介绍如何建立独立的类及类层。

2.1　面向对象编程基础

我们生活在由“对象”组成的世界中，每个对象都有自身的属性和操作方式，因此可把对象分为类。

面向对象编程（OOP）是利用真实世界对象的概念来开发应用程序，可以涉及同一类层中各独立的类。OOP的基础是类、对象、信息、方法、继承及多态性。

类被认为是对象的类型，每个对象都是类中的一个例子。

2.1.1　类与对象

在同一类中的所有对象都具有与其他类相同的属性和函数。通常，一个对象只具有由其属性的当前值定义的唯一的状态。类的功能决定了类中的对象可能具有的函数。C++语言既调用类中数据成员的属性，又调用类中的成员函数的操作。类中封装了数据成员及成员函数。

从OOP的观点来看，成员函数可通过改变类的数据成员而改变对象的状态。

2.1.2　信息与方法

对于面向对象编程，当信息传送给一个对象或在不同对象间传送时，它就模拟了不同对象的关系。对象接受一个信息后，可进行适当的操作。C++和其他OOP语言一样，不明确地支持信息和方法的概念。然而，用术语“信息”讨论成员函数很方便。术语“方法”和“成员函数”是等价的。

信息就是说明对象将进行何种操作。方法就是指接受到有关信息后，对象将被怎样操作。

2.1.3　继承

在面向对象设计的语言中，可以从一个类派生出其他类。利用继承，派生类继承了它的父辈或原始类的数据成员及成员函数。

通过增加新的属性和新的操作方式，可以改进原始类生成派生类。派生类定义了新的数据成员及成员函数。此外，派生类也能取消成员函数，这样那些函数的操作就不再适合派生类。

2.1.4　多态性

OOP设计的多态性特点允许采用同样的方法处理多种不同类型的数据，使同一符号名