# 10.5 对象数组

和Stock示例一样，用户通常要创建同一个类的多个对象。可以创建独立对象变量，就像本章前面的示例所做的，但创建对象数组将更合适。这似乎是在介绍一个未知领域，但实际上，声明对象数组的方法与声明标准类型数组相同：

Stock mystuff[4]; // 创建一个包含4个Stock对象的数组

前面讲过，当程序创建未被显式初始化的类对象时，总是调用默认构造函数。上述声明要求，这个类要么没有显式地定义任何构造函数（在这种情况下，将使用不执行任何操作的隐式默认构造函数），要么定义了一个显式默认构造函数（就像这个例子那样）。每个元素（mystuff[0]、mystuff[1]等）都是Stock对象，可以使用Stock方法：

mystuff[0].update(); // 调用第一个元素的update()方法

mystuff[3].show(); // 调用第4个元素的show方法

const Stock \*tops = mystuff[2].topval(mystuff[1]);

 // 比较第2个和第3个元素，获取有较大的topval值的对象的指针

可以用构造函数来初始化数组元素。在这种情况下，必须为每个元素调用构造函数：

const int STKS = 4;

Stock stocks[STKS] = {

 Stock(“NanoSmart" ,12.5 ,20),

 Stock(“Boffo Object” , 200 ,2.0),

 Stock(“Monolithic Obelisks” ,130 ,3.25),

 Stock（“Fleep Enterprises” , 60 ,.5）

 };

这里的代码使用标准格式对数组进行初始化：用括号括起的、以逗号分隔的值列表。其中，每次构造函数调用表示一个值。如果类包含多个构造函数，则可以对不同元素使用不同的构造函数：

const int STKS = 10;

Stock stocks[STKS]{

 Stock(“NanoSmart” ,12.5 ,20),

 Stock(),

 Stock(“Monolithic Obelisks” ,130 , 3.25),

}

上述代码使用Stock(const string & co,long n,double pr)初始化stock[0]和stock[2]，使用构造函数Stock()初始化stock[1]。由于该声明只初始化了数组的部分元素，因此余下的7个元素将使用默认构造函数进行初始化。

初始化对象数组的方案是，首先使用默认构造函数创建数组元素，然后花括号中的构造函数将创建临时对象，然后将临时对象的内容复制到相应的元素中。因此，要创建类对象数组，则这个类必须有默认构造函数。

程序清单10.9在一个小程序中使用了这些原理，该程序对4个数组元素进行初始化，显示它们的内容，并找出这些元素中总值最高的一个。由于topval()每次只检查两个对象，因此程序使用for循环来检查整个数组。另外，它使用stock指针来跟踪值最高的元素。该程序使用程序清单10.7中的头文件和程序清单10.8中的方法文件。

**程序清单10.9 usestock2.cpp**

// usestok2.cpp -- using the Stock class

// compile with stock20.cpp

#include <iostream>

#include "stock20.h"

const int STKS = 4;

int main()

{

 // 创建已初始化的对象数组

 Stock stocks[STKS] = {

 Stock("NanoSmart", 12, 20.0),

 Stock("Boffo Objects", 200, 2.0),

 Stock("Monolithic Obelisks", 130, 3.25),

 Stock("Fleep Enterprises", 60, 6.5)

 };

 std::cout << "Stock holdings:\n";

 int st;

 for (st = 0; st < STKS; st++)

 stocks[st].show();

 // 将指针指向第一个元素

 const Stock \* top = &stocks[0];

 for (st = 1; st < STKS; st++)

 top = &top->topval(stocks[st]);

 // 现在的指针指向价值最高的所持股票

 std::cout << "\nMost valuable holding:\n";

 top->show();

 std::cin.get();

 return 0;

}

下面是该程序的输出：

Stock holdings:

Company: NanoSmart Shares: 12

 Share Price; $20.000 Total Worth: $240.00

Company: Boffo Objects Shares: 200

 Share Price; $2.000 Total Worth: $400.00

Company: Monolithic Obelisks Shares: 130

 Share Price: $3.250 Total Worth: $422.50

Company: Fleep Enterprises Shares: 60

 Share Price: $6.500 Total Worth: $390.00

Most valuable holding:

Company: Monolithic Obelisks Shares: 130

 Share Price: $3.250 Total Worth: $422.50

有关程序清单10.9，需要注意的一点是，大部分工作是在类设计中完成的。完成类设计后，编写程序的工作本身便相当简单。

顺便说一句，知道this指针就可以更深入了解C++的工作方式。例如，最初的UNIX实现使用C++前端cfront将C++程序转换为C程序。处理方法的定义时，只需将下面这样的C++方法定义：

void Stock::show() const

{

 cout <<” Company: ” << company

 <<” Shares: “ << shares << ‘\n’

 <<” Share Price:$” << share\_val

 <<” Total Worth: $” << total\_val << ‘\n’;

}

转换为下面这样的C-风格定义：

void show(const Stock \*this)

{

 cout <<” Company: “ << this->company

 <<” Shares: “ << this->shares << ‘\n’

 <<” Share Price:$” << this->share\_val

 <<” Total Worth: $” << this->total\_val << ‘\n’;

}

即将Stock::限定符转换为函数参数（指向Stock的指针），然后用这个指针来访问类成员。

同样，该前端将下面的函数调用：

top.show();

转换为：

show(&top);

这样，将调用对象的地址赋给了this指针（实际情况可能更复杂些）。