第3章　类和对象

类(class)是面向对象程序设计的最基本的概念,是C++最强有力的特征,是进行封装和数据隐藏的工具,它将数据与操作紧密地结合起来。对象是类的实例,面向对象程序设计中的对象来源于现实世界,更接近人们的思维。本章介绍有关类和对象的基本概念和使用方法。

3.1　类与对象的基本概念

3.1.1　结构体与类

1. 结构体的扩充

结构体是C语言的一种自定义的数据类型,在结构体中可以含有各种不同类型的数据。C++语言对结构体类型进行了扩充,它不仅可以含有不同类型的数据,而且还可以含有函数。例如下面声明了一个扩充的结构体Complex:

```
struct Complex{                          //声明了一个名为 Complex 的结构体
  double real;                           //复数的实部
  double imag;                           //复数的虚部
  void init (double r,double i)          //定义函数 init,给 real 和 imag 赋初值
  { real=r;
    imag=i;
  }
  double abscomplex()                    //定义函数 abscomplex,求复数的绝对值
  { double t;
    t=real * real+imag * imag;
    return sqrt(t);
  }
};
```

在这个声明为Complex的结构体中,含有两个双精度数据real和imag,分别代表复数的实数部分和虚数部分,另外含有两个属于结构体的函数:init和abscomplex。函数init用于给real及imag赋初值,函数abscomplex用于计算和返回复数的绝对值。

结构体中的数据和函数都是结构体的成员,在C++中,通常把结构体中的数据称为数据成员,把函数称为成员函数。在这个结构体中,real和imag是数据成员,函数init和abscomplex是成员函数。为了访问这些成员函数,必须先定义该结构体类型的变量,然后

像访问结构体中的数据一样进行访问。

以下是这个例子的完整程序。

例 3.1 用扩充的结构体类型求复数的绝对值。

```
#include<iostream>
#include<cmath>
using namespace std;
struct Complex{                          //声明了一个名为 Complex 的结构体类型
 double real;                            //数据成员,复数的实部
 double imag;                            //数据成员, 复数的虚部
 void init(double r,double i)            //成员函数,给 real 和 imag 赋初值
 { real=r; imag=i; }
 double abscomplex()                     //成员函数,求复数的绝对值
 {
  double t;
  t=real * real+imag * imag;
  return sqrt(t);
 }
};
int main()
{ Complex  A;                            //定义结构体 Complex 的变量 A
  A.init(1.1,2.2);                       //调用成员函数 init,给 real 和 imag 赋初值
  cout<<"复数的绝对值是:"<<A.abscomplex()<<endl;  //调用成员函数 abscomplex
  return 0;
}
```

程序运行结果如下:

```
复数的绝对值是:2.45967
```

2. 类的声明

C++ 提供了一种比结构体类型更安全有效的数据类型——类。类与结构体的扩充形式十分相似,例如,上面的结构体类型 Complex 可以用类改写如下:

```
class Complex{                           //用关键字 class 取代结构体的 struct
                                         //声明了一个名为 Complex 的类
 double real;                            //数据成员,复数的实部
 double imag;                            //数据成员,复数的虚部
 void init(double r,double i)            //成员函数,给 real 和 imag 赋初值
 { real=r; imag=i;
 }
 double abscomplex()                     //成员函数,求复数的绝对值
 { double t;
   t=real * real+imag * imag;
   return sqrt(t);
 }
};
```

从本例可以看出声明类的方法与声明结构体类型的方法相似，第 1 行“class Complex”是类头，由关键字 class 和类名 Complex 组成。从第 1 行的左花括号起到最后一行的右花括号是类体。

在第 1 章中已说明，类是一种数据类型，它是一种用户定义的抽象的数据类型。类代表了一批对象的共性和特征。类是对象的抽象，而对象是类的实例。如同结构体类型和结构体变量的关系一样，在C++ 中也是先声明一个类类型，然后用它去定义若干的同类型的对象，对象是类类型的变量。如果使用类，则例 3.1 的程序可以改写如下。

例 3.2 用类类型替代例 3.1 中的结构体类型。

```
#include<iostream>
#include<cmath>
using namespace std;
class Complex{                                  //声明了一个名为 Complex 的类
 double real;                                   //数据成员,复数的实部
 double imag;                                   //数据成员,复数的虚部
 void init(double r,double i)                   //成员函数,给 real 和 imag 赋初值
 { real=r; imag=i; }
 double abscomplex()                            //成员函数,求复数的绝对值
 {
   double t;
   t=real*real+imag*imag;
   return sqrt(t);
 }
};
int main()
{ Complex  A;                                                  //定义类 Complex 的对象 A
  A.init(1.1,2.2);                                             //编译错误
  cout<<"复数的绝对值是:"<<A.abscomplex()<<endl;               //编译错误
  return 0;
}
```

但是，当我们对例 3.2 进行编译时，程序中标注出的两条语句将出现编译错误。那么，为什么在例 3.1 中使用结构体类型时程序能够正常运行，而在例 3.2 中将结构体类型改成类类型时出现了错误呢？原因是，为了保护类中数据的安全，在C++ 中将类中的成员分为两类：私有成员（用 private 声明）和公有成员（用 public 声明）。私有成员（包括数据成员和成员函数）只能被类内的成员函数访问，而不能被类外的对象访问；公有成员（包括数据成员和成员函数）既可被类内的成员函数访问，也可被类外的对象访问。

C++ 规定，在默认情况下（即没有指定属于私有或公有时），类中的成员是私有的。C++ 结构体中的成员同样可以分为私有成员和公有成员，但是在默认情况下，结构体中的成员是公有的。因此，在例 3.1 结构体 Complex 中的成员默认为公有的，即成员函数 init 和 abscomplex 是公有的，结构体外的变量 A 能够对它们直接进行访问，所以以下两条语句是可以正常运行的：

```
A.init(1.1,2.2);
cout<<"复数的绝对值是:"<<A.abscomplex()<<endl;
```

而在例 3.2 类 Complex 中的成员都是私有的，即成员函数 init 和 abscomplex 是私有的，类外的对象 A 不能对它们直接进行访问，所以以下两条语句编译时将出现错误：

```
A.init(1.1,2.2);                                    //编译错误
cout<<"复数的绝对值是:"<<A.abscomplex()<<endl;       //编译错误
```

声明为私有(private)的成员对外界是隐蔽的，在类外不能直接访问。如果一个类的所有成员都声明为私有的，即只有私有(private)部分，那么该类将完全与外界隔绝，这样，虽然数据“安全”的目的达到了，但是这样的类是没有实际意义的。在实际应用中，一般把需要保护的数据设置为私有的，把它隐蔽起来，而把成员函数设置为公有的，作为类与外界的接口。

为了帮助大家理解私有成员和公有成员的作用，我们用彩色电视机作一个例子。所有的彩色电视机都有图像、亮度、色彩、频道等数据，也都有相应的调节按钮(或在遥控器上)。正如第 1 章所述，可以把所有的彩色电视机的共性抽象为彩色电视机类。正常使用时，使用者只能通过屏幕观看图像，通过按钮来调整各项数据。为了保证彩色电视机数据的安全，除了专业人员可以拆开电视机调整图像等数据外，普通用户是不准拆开电视机的。这样，屏幕和各种按钮就是用户接触电视机的仅有途径，因此将它们设计成类的公有成员函数，作为类的外部接口。而电视机的图像、亮度等数据便是类的私有成员，使用者只能通过它的外部接口(公有成员函数)去访问这些私有成员。

按照这种思路，例 3.2 可以改写如下。

例 3.3 含有公有成员和私有成员的 Complex 的类。

```
#include<iostream>
#include<cmath>
using namespace std;
class Complex{                          //声明了一个名为 Complex 的类
 private:                               //声明以下部分为私有的
   double real;                         //私有数据成员,复数的实部
   double imag;                         //私有数据成员,复数的虚部
 public:                                //声明以下部分为公有的
   void init(double r,double i)         //公有成员函数,作为类的外部接口
   { real=r;
     imag=i;
   }
   double abscomplex()                  //公有成员函数,作为类的外部接口
   { double t;
     t=real*real+imag*imag;
     return sqrt(t);
   }
};
int main()
{ Complex  A;                           //定义类 Complex 的对象 A
  A.init(1.1,2.2);                      //类外的对象 A 可以访问公有成员函数 init
```

```
    cout<<"复数的绝对值是:"<<A.abscomplex()<<endl;
                                        //类外的对象 A 可以访问公有成员函数 abscomplex
    return 0;
}
```

由于在本程序中成员函数 init 和 abscomplex 都被设置成公有成员，所以类 Complex 的对象 A 可以访问这些函数。程序运行结果如下：

```
复数的绝对值是:2.45967
```

归纳以上对类类型的说明，类类型声明的一般形式如下：

```
class 类名{
  [private:]
    私有数据成员和成员函数
  public:
    公有数据成员和成员函数
};
```

其中，class 是声明类的关键字，类名是要声明的类类型的名字；后面的花括号表示出类的声明范围；最后的分号表示类声明结束。

在类中，封装了有关数据和对这些数据进行操作的函数，分别称为类的数据成员和成员函数。

private 和 public 称为成员访问限定符，用它们来声明各成员的访问属性。每个成员访问限定符下面又都可以有数据成员和成员函数。数据成员和成员函数一般统称为类的成员。

private 部分称为类的私有部分，这一部分的数据成员和成员函数称为类的私有成员。私有成员只能由本类的成员函数访问，而类外部的任何访问都是非法的。这样，私有成员就整个隐蔽在类中，在类的外部，对象无法直接访问它们，实现了访问权限的有效控制。

public 部分称为类的公有部分，这部分的数据成员和成员函数称为类的公有成员。公有成员对外是完全开放的。公有成员既可以被本类的成员函数访问，也可以在类外被该类的对象访问。公有成员函数是类与外界的接口，来自类外部的对私有成员的访问需要通过这些接口来进行。

从以上的分析可以看出，类和结构体的功能基本上相同。那么，何必多此一举，在C++中设两种功能一样的类型呢？这主要是考虑到设计C++语言的一条原则，即C++必须兼容C，要使得以前用 C 编写的已在广泛使用的 C 程序能够不加修改地在C++的环境下使用。所以，在C++中必须保留 C 结构体这种数据类型，并对其功能进行扩充。

另外，结构体和类是有区别的。考虑到 C 的传统原因，在结构体中，如果对其成员不作 private 或 public 声明，系统将其默认为公有的(public)，外界可以任意地访问其中的数据成员和成员函数，它不具有信息隐蔽的特性，除非将其中某些成员显式声明为 private。而在类声明中，如果对其成员不作 private 或 public 声明，系统将其默认为私有的(private)，外界不可以访问其中的数据成员和成员函数，它提供了默认的安全性。这一规定符合面向对象思想中数据隐藏的准则。数据隐藏使得类中的成员得到更好的保护。

虽然,类和结构体的功能基本上相同,但是建议读者尽量使用类与对象,写出完全体现C++风格的程序。

说明:

(1) 除了 private 和 public 之外,类中的成员还可以用另一个关键字 protected 来说明。被 protected 说明的数据成员和成员函数称为保护成员。保护成员可以由本类的成员函数访问,也可以由本类的派生类的成员函数访问,而类外的任何访问都是非法的,即它是半隐蔽的,关于保护成员将在第 4 章详细介绍。

(2) 对一个具体的类来讲,类声明格式中的三个部分并非一定要全有,但至少要有其中的一个部分。一般情况下,一个类的数据成员应该声明为私有成员,成员函数声明为公有成员。这样,内部的数据隐蔽在类中,在类前面的外部无法直接访问,使数据得到有效的保护,也不会对该类以外的其余部分造成影响,程序模块之间的相互作用就被降低到最小。

(3) 类声明中的 private、protected 和 public 三个关键字可以按任意顺序出现任意次。但是,如果把所有的私有成员和公有成员归类放在一起,程序将更加清晰。

(4) 有些程序员主张将所有的私有成员放在其他成员的前面,因为一旦用户忘记了使用说明符 private,由于默认值是 private,这将使用户的数据仍然得到保护。另一些程序员主张将公有成员放在最前面,这样可以使用户将注意力集中在能被外界调用的成员函数上,使用户思路更清晰一些。不论 private 部分放在前面,还是 public 部分放在前面,类的作用是完全相同的。

(5) 数据成员可以是任何数据类型,但是不能用自动(auto)、寄存器(register)或外部(extern)进行说明。

3.1.2 成员函数的定义

类的成员函数是函数的一种,它也有函数名、返回值类型和参数表,用法与普通函数基本上是一样的,只是它属于一个类的成员。成员函数可以访问本类中任何成员(包括公有的、保护的和私有的)。成员函数可以被指定为私有的(private)、公有的(public)和保护的(protected)。其中,私有的成员函数只能被本类中其他成员函数调用,不能被类外的对象调用;公有的成员函数既可以被本类的成员函数访问,也可以在类外被该类的对象访问。保护的成员函数将在第 5 章详细介绍。

在C++程序设计中,成员函数既可以定义成普通的成员函数(即非内联的成员函数),也可以定义成内联成员函数。以下介绍成员函数的三种定义方式中,第一种方式是定义成普通成员函数,而后两种方式是定义成内联成员函数。

成员函数的第一种定义方式是:在类声明中只给出成员函数的原型,而将成员函数的定义放在类的外部。这种成员函数在类外定义的一般形式是:

```
返回值类型 类名::成员函数名(参数表)
{
    函数体
}
```

例如，表示坐标点的类 Point 可声明如下：

```
class Point{
  public:
    void setpoint (int,int);              //设置坐标点的成员函数 setpoint 的函数原型
    int getx();                           //取 x 坐标点的成员函数 getx 的函数原型
    int gety();                           //取 y 坐标点的成员函数 gety 的函数原型
  private:
    int x,y;
};
void Point::setpoint(int a,int b)         //在类外定义成员函数 setpoint
{ x=a;
  y=b;
}
int Point::getx()                         //在类外定义成员函数 getx
{ return x;
}
int Point::gety()                         //在类外定义成员函数 gety
{ return y;
}
```

从这个例子可以看出，虽然函数 setpoint、getx 和 gety 在类外部定义，但它们属于类 Point 的成员函数，它们可以直接访问类 Point 中的私有数据成员 x 和 y。

说明：

(1) 在类外定义成员函数时，必须在成员函数名之前缀上类名，在类名和函数名之间应加上作用域运算符“::”，用于声明这个成员函数是属于哪个类的。例如上面例子中的“Point::”，说明这些成员函数是属于类 Point 的。如果在函数名前没有类名，或既无类名又无作用域运算符“::”，如

```
::getx() 或 getx()
```

则表示 getx 函数不属于任何类，这个函数不是成员函数，而是普通的函数。

(2) 在类声明中，成员函数的原型的参数表中可以不说明参数的名字，而只说明它们的类型。例如：

```
void setpoint (int,int);
```

但是，在类外定义成员函数时，不但要说明参数表中参数的类型，还必须要指出其参数名。

(3) 采用“在类声明中只给出成员函数的原型，而将成员函数的定义放在类的外部”的定义方式，是C++ 程序设计的良好习惯。这种方式不仅可以减少类体的长度，使类的声明简洁明了，便于阅读，而且有助于把类的接口和类的实现细节相分离，隐藏了执行的细节。

成员函数的第二种定义方式是：将成员函数直接定义在类的内部。例如：

```
class Point{
  public:
    void setpoint(int a,int b)            //成员函数 setpoint 直接定义在类的内部
```

```
    { x=a;
      y=b;
    }
    int getx()                              //成员函数 getx 直接定义在类的内部
    { return x;
    }
    int gety()                              //成员函数 gety 直接定义在类的内部
    { retrun y;
    }
  private:
    int x,y;
};
```

此时，C++ 编译器将函数 setpoint、getx 和 gety 作为内联函数进行处理，即将这些函数隐含地定义为内联成员函数。和第 2 章中介绍的普通内联函数相同，内联成员函数的函数体代码也会在编译时被插入到每一个调用它的地方。这种做法可以减少调用函数的开销，提高执行效率，但是却增加了编译后代码的长度，所以只有相当简短的成员函数才定义为内联函数。

这种定义内联成员函数的方法没有使用关键字 inline 进行声明，因此称为隐式定义。

成员函数的第三种定义方式是：为了书写清晰，在类声明中只给出成员函数的原型，而将成员函数的定义放在类的外部。但是在类内函数原型声明前或在类外定义成员函数前冠以关键字 inline，以此显式地说明这是一个内联函数。这种定义内联成员函数的方法称为显式定义。

例如，使用显式定义内联成员函数，上面表示坐标点的类 Point 可声明如下：

```
class Point{
  public:
    inline void setpoint(int,int);          //声明成员函数 setpoint 为内联函数
    inline int getx();                      //声明成员函数 getx 为内联函数
    inline int gety();                      //声明成员函数 gety 为内联函数
  private:
    int x,y;
};
inline void Point::setpoint(int a,int b)  //在类外定义此函数为内联函数
{ x=a;
  y=b;
}
inline int Point::getx()                    //在类外定义此函数为内联函数
{ return x;
}
inline int Point::gety()                    //在类外定义此函数为内联函数
{ return y;
}
```

说明：

可以在声明函数原型和定义函数时同时写 inline，也可以在其中一处声明 inline，效果是相同的，都能按内联函数处理。使用 inline 定义内联函数时，必须将类的声明和内联成员函数的定义都放在用一个文件（或同一个头文件）中，否则编译时无法进行代码置换。

3.1.3 对象的定义及使用

1. 类与对象的关系

通常我们把具有共同属性和行为的事物所构成的集合叫做类。在C++ 中，可以把具有相同数据和相同操作集的对象看成属于同一类。

一个类也就是用户声明的一个数据类型。每一种数据类型（包括基本数据类型和自定义类型）都是对一类数据的抽象，在程序中定义的每一个变量都是其所属数据类型的一个实例。类的对象可以看成是该类类型的一个实例，定义一个对象和定义一个一般变量相似。

在C++ 中，类与对象间的关系，可以用数据类型 int 和整型变量 i 之间的关系来类比。类类型和 int 类型均代表的是一般的概念，而对象和整型变量却是代表具体的东西。正像定义 int 类型的变量一样，也可以定义类的变量。C++ 把类的变量叫做类的对象，对象也称为类的实例。

2. 对象的定义

可以用以下两种方法定义对象。

（1）在声明类的同时，直接定义对象，即在声明类的右花括号“}”后，直接写出属于该类的对象名表。例如：

```
class Point {
  public:
    void setpoint(int,int);
    int getx();
    int gety();
  private:
    int x,y;
} op1,op2;
```

在声明类 Point 的同时，直接定义了对象 op1 和 op2。

（2）声明了类之后，在使用时再定义对象。定义对象的格式与定义基本数据类型变量的格式类似，其一般形式如下：

类名 对象名 1,对象名 2,……;

例如：

```
Point op1,op2;
```

此时定义了 op1 和 op2 为 Point 类的两个对象。

说明：

声明了一个类便声明了一种类型，它并不接收和存储具体的值，只作为生成具体对象的一种“样板”，只有定义了对象后，系统才为对象分配存储空间，以存放对象中的成员。

3. 对象中成员的访问

不论是数据成员，还是成员函数，只要是公有的，在类的外部可以通过类的对象进行访问。访问对象中的成员通常有以下三种方法。

1）通过对象名和对象选择符访问对象中的成员

其一般形式是：

对象名.数据成员名

或

对象名.成员函数名[(实参表)]

其中“.”叫做对象选择符，简称点运算符。

下面的例子中定义了 Point 类的两个对象 op1 和 op2，并对这两个对象的成员进行了一些操作。

例 3.4 通过对象名和对象选择符访问对象中的成员。

```
#include<iostream>
using namespace std;
class Point{
  public:
    void setpoint(int a,int b)
    { x=a;
      y=b;
    }
    int getx()
    { return x;
    }
    int gety()
    { return y;
    }
  private:
    int x,y;
};
int main()
{ Point op1,op2;            //定义对象 op1 和 op2
  int i,j;
  op1.setpoint(1,2);        //调用对象 op1 的成员函数 setpoint,给 op1 的数据成员赋值
  op2.setpoint(3,4);        //调用对象 op2 的成员函数 setpoint,给 op2 的数据成员赋值
  i=op1.getx();             //调用对象 op1 的成员函数 getx,取 op1 的 x 值
```