第3章

类和对象(一)

本章要点：

- 类的定义与类的成员；
- 对象的定义与使用；
- 类的构造函数；
- 重载构造函数；
- 析构函数；
- UML 及类图。

本书的第一部分讲解了 C++语言在面向过程方面对 C 语言的扩充与改进，从本章开始进入第二部分面向对象的程序设计，将介绍 C++语言在面向对象方面的应用。围绕类与对象的概念和特点，本章将介绍类的定义与使用方法、类的数据成员与成员函数、类的构造函数与析构函数，以及拷贝构造函数的定义、深拷贝与浅拷贝的区别。

3.1 类的构成

3.1.1 从结构到类

C++语言中的结构(structure 或 struct)类似于一种简单的类。结构是一种自定义的数据类型，它们把相关联的数据元素组成一个单独的统一体。

例如，下面声明了一个日期结构：

```
struct date
{
  int year;
  int month;
  int day;
};
```

结构 date 中包含了三个数据元素，即 year、month 和 day，分别表示年、月、日。在结构中可以进行设置日期、显示日期等操作。以下是完整的程序。

【例 3.1】 有关日期结构的例子。

```
/* 03_01.cpp */
#include <iostream>
using namespace std;
struct date
{
    int year;
    int month;
    int day;
};
int main()
{
    date date1;
    date1.year = 2009;
    date1.month = 5;
    date1.day = 26;
    cout << date1.year <<"." << date1.month <<"." << date1.day << endl;
    return 0;
}
```

程序的运行结果为：

```
2009.5.26
```

C 语言中的结构体存在一些缺点。例如，一旦建立了一个结构变量，就可以在结构体外直接修改数据，如在例 3.1 的 main()函数中，可以用赋值语句随意修改结构变量中的数据 year、month 和 day。但是在现实世界中有些数据不允许随意修改。换句话说，不同的使用者对数据修改的权限是不一样的。例如，一个人的出生日期是不能随意修改的。可见，在 C 语言结构中的数据很不安全的，C 语言结构无法对数据进行保护和权限控制。C 语言结构中的数据与对这些数据进行的操作是分离的，没有把这些相关的数据和操作构成一个整体进行封装，因此使程序的复杂性很难控制，维护数据和处理数据要花费很大的精力，使传统程序难以重用，严重影响了软件的生产效率。

在 C++语言中引入了类的概念，它能克服 C 语言结构的这些缺点，C++语言中的类将数据和与之相关的函数封装在一起，形成一个整体，具有良好的外部接口，可以防止数据未经授权的访问，提供了模块间的独立性。

3.1.2 类的构成

在面向对象程序设计中，类和对象是两个基本的概念。对象是客观事物在计算机中的抽象描述，而类则是对具有相似属性和行为的一组对象的统一描述。

类是把各种不同类型的数据和对数据的操作组织在一起而形成的用户自定义的数据类型。其中，把不同类型的数据称为数据成员，把对数据的操作称为成员函数。

类主要由 3 部分组成，分别是**类名**、**数据成员**和**成员函数**。按访问权限划分，数据成员和成员函数又可分为 3 种，分别是公有数据成员与成员函数、保护数据成员与成员函数，以

及私有数据成员与成员函数。类声明的一般格式如下：

```
class 类名
{
    [private: ]
        私有数据成员;
        私有成员函数;
    protected:
        保护数据成员;
        保护成员函数;
    public:
        公有数据成员;
        公有成员函数;
};
```

说明：

(1) class 是类定义的关键字。

(2) 类名由用户自定义，但必须是 C++语言的有效标识符，且一般首字母要大写。

(3) 花括号中是类体，最后以一个分号“；”结束。

(4) private、public、protected 这 3 个关键字是访问权限控制符，限制了类成员的访问权限。

例如，下面定义了一个描述日期的类。

```
class Date
{
    private:                                    //private 可以默认
      int year;                                 //定义数据成员
      int month;
      int day;
    public:
      void SetDate(int y,int m,int d);          //成员函数声明
     void ShowDate();                           //成员函数声明
};
```

在此，声明了一个类 Data，封装了有关数据和对这些数据的操作，分别称为类 Date 的数据成员和成员函数。

在一般情况下，类体中仅给出成员函数原型，而把函数体的定义放在类体外实现。成员函数的具体定义将在后续章节讨论。

从类的定义可看出，类是实现封装的工具。封装就是将类的成员按使用或存取的方式分类，从而有条件地限制对类成员的使用。

3.1.3　类成员的访问属性

类的任何成员都具有访问属性，类成员有 3 种访问属性：私有类型(private)、公有类型(public)和保护类型(protected)，并分别由 private、public、protected 这 3 个关键字后跟冒号“：”来指定。

这 3 种访问权限控制符可以以任何顺序出现，且在同一个类的定义中，这 3 个部分并非

必须同时出现。

(1) private 部分：这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，而类外部根本就无法访问，实现了访问权限的有效控制。在类 Date 中就声明了 3 个只能由内部函数访问的数据成员，即 year、month 和 day。

(2) public 部分：这部分的数据成员和成员函数称为类的公有成员。公有成员可以由程序中的函数访问，即它对外是完全开放的。公有成员函数是对类的动态特性的描述，是类与外界的接口，来自类外部的访问需要通过这种接口来进行。例如，在类 Date 中声明了设置日期成员函数 SetDate()和日期成员函数 ShowDate()，它们都是公有的成员函数，类外部若想对类 Date 的数据进行操作，只能通过这两个函数来实现。

(3) protected 部分：这部分的数据成员和成员函数称为类的保护成员。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，这个问题将在第 5 章详细介绍。

类的定义应注意以下几点：

(1) 对一个具体的类来讲，类声明格式中的 3 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，成员函数为公有成员。这样，内部的数据结构整个隐蔽在类中，在类的外部根本就无法看到，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。

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

(3) 若私有部分处于类体中第一部分，关键字 private 可以省略。这样，如果一个类体中没有一个访问权限关键字，则其中的数据成员和成员函数都默认为私有的。

(4) 数据成员可以是任何数据类型，但是不能用自动方式、寄存器方式或外部方式进行说明。

(5) 不能在类声明中给数据成员赋初值。例如：

```
class Date
{
   private:                              //定义私有数据成员
     int year = 2008;                    //错误
     int month = 10;                     //错误
     int day = 5;                        //错误
   public:
     void SetDate(int y,int m,int d);//公有成员函数声明
    void ShowDate();
};
```

C++语言规定，只有在类对象定义之后才能给数据成员赋初值。

C++语言增加了 class 类型后，仍然保留了结构体类型 struct，而且把它的功能也扩展了。

在 C++语言中通过对结构类型的扩充，使得它不仅可以包含不同类型的变量，还可以包

含对这些变量进行相关操作的函数。类 class 和结构 struct 很相似,都含有以数据成员表示的属性和以成员函数表示的行为。

但是,用 struct 声明的类和 class 声明的类是有区别的。C++语言中类 class 和结构 struct 的主要区别是默认访问属性不同。在类 class 中,成员的默认访问属性是 private,而在结构体 struct 中,成员的默认访问属性是 public,相比之下,类更好地体现了面向程序设计中封装与信息隐藏的特点。

3.2　类的成员函数

在面向对象程序设计中,类的成员函数是实现对封装的数据成员进行操作的主要途径,是类的行为。类中的所有成员都要在类的类体中进行说明,但成员函数的定义既可以在类体中给出,也可以在类体外给出。通常采用以下两种方式。

1. 将成员函数的定义直接写在类体中

在类中直接定义成员函数一般适合于成员函数规模较小的情况。

例如,定义表示坐标点的类 Point。

```
class Point
{
    private:
        int x,y;
    public:
        void SetPoint(int a,int b)
        {
          x = a;y = b;
        }
        int Getx()
        {
          return x;
        }
        int Gety()
        {
          return y;
        }
};
```

此时,成员函数 SetPoint()、Getx()和 Gety()就是隐含的内联成员函数。即使没有明确用 inline 关键字定义,它们也是内联函数。内联函数的调用类似宏指令的扩展,它直接在调用处扩展其代码,而不进行一般函数的调用操作。

2. 在类定义中只给出成员函数的原型,而成员函数体写在类的定义之外

这种方法比较适合于成员函数的函数体较大的情况,但要求在定义成员函数时,在函数的名称之前加上其所属的类名以及作用域运算符“::”,以此来表示该成员函数属于哪个类,未加类名及作用域运算符的函数是非成员函数。

这种成员函数在类外定义的一般形式如下：

```
函数返回值的类型 类名::成员函数名(形式参数表)
{
    //函数体
}
```

例如，对于前面的表示坐标点的类 Point 来说，给出下面的定义。

```
class Point
{
    private:
        int x,y;
    public:
        void SetPoint (int,int);
        int Getx();
        int Gety();
};
void Point::SetPoint(int a,int b)
{
    x = a;y = b;
}
int Point::Getx()
{
    return x;
}
int Point::Gety()
{
    return y;
}
```

在这个例子中，虽然函数 SetPoint()、Getx()和 Gety()的函数体写在类外部，但它们属于类 Point 的成员函数，它们可以直接使用类 Point 中的数据成员 x 和 y。

在 C++语言中，除了上面第一种方式的隐式声明外，还可以用下面格式将成员函数显式声明为类的内联函数。

```
inline 返回类型 类名::成员函数名(参数表)
{
  //函数体
}
```

在类声明中只给出成员函数的原型，而成员函数体写在类的外部。但为了使它起内联函数的作用，在成员函数返回类型冠以关键字 inline，以此显式地说明这是一个内联函数。

例如，上面的例子改为显式声明为内联函数可变成如下形式：

```
class Point
{
   private:
      int x,y;
   public:
      void SetPoint(int,int)
```

```
    int Getx();
    int Gety();
};
inline void Point::SetPoint(int a ,int b)
{
    x = a;y = b;
}
inline int Point::Getx()
{
    return x;
}
inline int Point::Gety()
{
    return y;
}
```

使用 inline 说明内联函数时，必须使函数体和 inline 说明结合在一起，否则编译器将它作为普通函数处理。例如函数原型写成：

```
inline void SetPoint(int, int);
```

不能说明这是一个内联函数。有效的声明应该如下：

```
inline void Point::SetPoint(int a,int b)
{
    x = a;y = b;
}
```

通常只有较短的成员函数才定义为内联函数，对于较长的成员函数最好作为一般函数处理。

3.3　对象的定义与使用

类描述了对象的共同属性和行为，是一个用户自定义的数据类型，实现了封装和数据隐藏功能。但是，类作为一种类型在程序中只有通过定义该类型的变量——对象，才能发挥作用。对象是类的实例或实体。下面来具体介绍对象的定义和使用。

3.3.1　类与对象的关系

一个类也就是用户声明的一个数据类型，而且是一个抽象数据类型。每一种数据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。

类是抽象的概念，而对象是具体的概念；类只是一种数据类型，而对象属于该类(数据类型)的一个变量，每个对象占用了各自的存储单元，都各自具有了该类的一套数据成员(静态数据成员除外)，而类的成员函数是所有对象共有的。每个对象的成员函数都通过指针指向同一个代码空间。

在 C++语言中，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++语言把类的变量叫做类的对象,对象也称为类的实例。

3.3.2 对象的定义

对象的定义也称对象的创建,在 C++语言中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

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
{
    private:
        int x,y;
    public:
        void SetPoint(int,int);
        int Getx();
        int Gety();
}op1,op2;
```

在声明类 Point 的同时,直接定义了对象 op1 和 op2。这时定义的对象是一个全局对象。

2. 声明了类之后在使用时再定义对象

定义的格式与一般变量的定义格式相同:

类名 对象名(参数表);

说明:

(1) 对象名可以是一个或多个对象的名字,多个对象名之间用逗号分隔;在对象名中,可以是一般的对象名,也可以是指向对象的指针变量名或引用,还可以是对象数组名。

(2) 参数表是初始化对象所需要的。创建对象时可以根据给定的参数调用相应的构造函数对对象进行初始化。没有参数时表示调用类的默认构造函数。关于构造函数的内容将在 3.4 节中介绍。

例如,上例中已定义了类 Point,则:

```
class Point
{
    //…
};
//…
int main()
{
    Point op1,op2;
    //…
}
```

在主函数中,为类 Point 定义了 op1 和 op2 的两个对象。

说明:

(1) 在声明类的同时定义的对象是一种全局对象,在它的生存期内任何函数都可以使用它。但有时使用它的函数只在极短的时间对它进行操作,而它却总是存在,直到整个程序运行结束,因此,容易导致错误和混乱。而采用使用时再定义对象的方法可以消除这种弊端,建议尽可能使用这种方法来定义对象。

(2) 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。

3.3.3　对象中成员的访问

对象中的成员就是该类对象所属的类所定义的成员,包括数据成员和成员函数。定义了类及其对象之后,就可以通过对象来访问其中的成员了。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,访问的方式包括圆点访问形式和指针访问形式。对象只能用前一种方式访问成员,而指向对象的指针用两种方式都可以访问。

1. 圆点访问形式

圆点访问形式就是使用成员运算符"."来访问类的成员,一般格式如下:

对象名.成员名

或

(* 指向对象的指针).成员名

在类定义内部,所有成员之间可以互相直接访问;在类的外部,只能以上述格式访问类的公有成员。主函数 main()也在类的外部,所以,在主函数中定义的类对象,在操作时只能访问其公有成员。

下面的例 3.2 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 3.2】 使用类 Point 的完整程序。

```
/* 03_02.cpp */
#include<iostream>
using namespace std;
class Point
{
    private:
        int x,y;
    public:
        void SetPoint(int a,int b)          //对数据成员赋值函数
        {
        x=a; y=b;
        }
        int Getx()                          //提取 x 变量值
```

```
    {
        return x;
    }
int Gety()                                  //提取 y 变量值
    {
        return y;
    }
};
int main()
{
    Point op1,op2;
    int i, j;
    op1.SetPoint(10,20);                    //通过对象以圆点形式访问成员函数
    op2.SetPoint(30,40);
    i = op1.Getx();
    j = op1.Gety();
    cout <<"op1 i = "<< i <<"   op1 j = "<< j << endl;
    i = op2.Getx();
    j = op2.Gety();
    cout <<"op2 i = "<< i <<"   op2 j = "<< j << endl;
    return 0;
}
```

程序的运行结果为：

```
op1 i = 10   op2 j = 20
op2 i = 30   op2 j = 40
```

说明：

(1) 例 3.2 中的 op1.SetPoint(10,20)实际上是一种缩写形式，它表达的意义是 op1。Point::SetPoint(10,20)，这两种表达式是等价的。

(2) 在类的内部所有成员之间都可以通过成员函数直接访问，但是类的外部不能访问对象的私有成员。

下面例 3.3 就是一个存在错误的程序。

【例 3.3】 一个存在错误的程序。

```
/* 03_03.cpp */
#include <iostream>
using namespace std;
class Date
{
    private:
        int year;
        int month;
        int day;
    public:
        void SetDate(int y,int m,int d);      //声明对数据成员赋值函数
        void ShowDate();                      //声明显示数据成员值的成员函数
};
void Date::SetDate(int y,int m,int d)
```

```
{
    year = y;
    month = m;
    day = d;
}
void Date::ShowDate()
{
    cout << year <<"."<< month <<"."day << endl;
}
int main()
{
    Date date1,date2;
    cout <<"date1 set and output:"<< endl;
    date1.SetDate(1998,4,28);
    cout << date1.year <<"."<< date1.month <<"."<< date1,day << endl;      //错误
    cout <<"date2 set and output:"<< endl;
    cout << date2.year <<"."<< date2.month <<"."<< date2.day << endl;     //错误
    return 0;
}
```

编译这个程序时,编译器将标示出两条错误的语句,因为类的外部不能访问对象的私有成员。因此,应该将这两条错误语句改成调用公有的成员函数 ShowDate()来显示私有数据成员 year、month 和 day 的值,修改后的程序如例 3.4 所示。

【例 3.4】 使用 Date 类的正确程序。

```
/* 03_04.cpp */
#include<iostream>
using namespace std;
class Date
{
    private:
        int year;
        int month;
        int day;
    public:
        void SetDate(int y,int m,int d);
        void ShowDate();
};
void Date::SetDate(int y,int m,int d)
{
    year = y;
    month = m;
    day = d;
}
inline void Date::ShowDate()
{
    cout << year <<"."<< month <<"."day << endl;
}
int main()
{
```