移动平台应用软件开发
C++面向对象基础

类和对象

主讲：张齐勋

zhangqx@ss.pku.edu.cn

《移动平台应用软件开发》课程建设小组

北京大学

二零一五年

北京大学

提纲

一、类的定义

二、对象的定义和成员表示

三、构造函数与析构函数

四、成员函数的特征

五、静态成员

一、类的定义

- 类是具有相同属性和行为的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述，其内部包括属性和行为两个主要部分。
- 利用类可以实现数据的封装、隐藏、继承与派生。
- 利用类易于编写大型复杂程序，其模块化程度比C中采用函数更高。

- 类的定义分为说明和实现两个部分。

- 类说明部分是用来声明该类中的成员。类的成员包括数据成员和函数成员。其中，函数成员又称成员函数或“方法”，用于对数据成员进行各种操作。

- 类实现部分用来对成员函数进行定义。
- 即说明部分告诉类要“干什么”，实现部分告诉类“怎么干”。

类说明部分

类说明部分一般格式如下:

```
class  类名
{
        private:(或缺省时)
                数据成员声明或函数成员的原型;
        protected:
                数据成员声明或函数成员的原型;
        public:
                数据成员声明或函数成员的原型;
};
```

类实现部分

类的实现部分，包括所有在类体中说明的成员函数的定义。成员函数的定义通常在类定义体之外给出，其中每个成员函数定义格式为:

返回值类型 类名::成员函数名（<参数表>）

{

…　　//函数体

}

类的成员函数在类外部定义时，前面必须加上“类名::”，以说明所定义的函数是哪一个类的成员。“::”称为作用域运算符。

类封装和数据隐藏

封装性来自对类成员的访问控制权限。

在C++中，类的成员从访问权限上分为私有（private）、公有（public）和保护（protected）三类。

私有成员通常是一些数据成员。private权限为类带来了封装性，它使私有成员隐藏起来，不能从类的外部对它们进行访问，或者说它们从类外部是不可见的，只有类自己的成员函数才可以访问它们。

公有成员往往是一些操作（即成员函数），可在程序中类的外部访问它们，它们是类的对外接口。

例：点类Point的定义

类说明部分：

```
class  Point{
private:
    double  x , y ;
public:
    void  SetPoint(double x , double y) ;
    double  GetX ( ) ;
    double  GetY ( ) ;
    void  Print ( ) ;
};
```

类实现部分：

```
void  Point :: SetPoint (double a , double b)   //定义成员函数SetPoint( )
{ x = a ; y = b; }
double  Point :: GetX ( )          //定义成员函数GetX ( )
{ return x ;}
double  Point :: GetY ( )          //定义成员函数GetY ( )
{ return y ;}
void  Point :: Print( )            //定义成员函数Print( )
{ cout<<” X=” << x <<” ,” <<  “Y=” << y <<endl ;}
```

定义类时说明部分中的关键字**public**、**private**、**protected**从它们出现的位置起开始生效，直到出现另一个访问权限关键字为止。

访问权限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，能增强程序的可读性。

如果把所有的私有成员放在公有成员前面，可以自动获得缺省访问控制权限**private**。

类中的数据成员可以是任何数据类型。例如整型、浮点型、字符型、数组、指针和引用等。

不能在类的说明部分给类的数据成员赋初值，例如在点类的定义中，下面的定义是错的：

```
class  Point{
        int  x=0 , y=0;
    public:
            ┆
};
```

一般习惯性地将类定义的说明部分或者整个定义部分（包括实现部分）放到一个头文件中，例如，将前面定义的点类Point的定义部分放在point.h文件中。

二、对象的定义和成员表示

类描述了对象的数据存储和操作特性，对象是类的实例。正像定义 int 类型的变量一样，创建类类型 Point 的对象也被看作定义Point 类型的变量。

对象在它的类确定了以后，定义格式为：

类名 <对象名表>

例如，定义(或者说创建)两个点类Point 的对象：

```
Point  p1 , p2 (2.0 , 3.0) ;
```

<对象名表>中，可以是一般的对象名，也可以是指向对象的指针或对象的引用，还可以是对象数组名。例如，一个复数类Complex的对象可以如下定义：

```
Complex  c1，c2，*pc，c[10];
```

对象成员的表示方法

一个对象的成员就是该对象的类所定义的成员。对象成员有数据成员和成员函数。

一般对象的成员表示如下：

<对象名>.<成员名>

或者

<对象名>.<成员名>(<参数表>)

前者用来表示数据成员，后者用来表示成员函数。这里的“.”是一个运算符，该运算符的功能是表示对象的成员。

指向对象的指针的成员表示如下：

<对象指针名>-><成员名>

或者

<对象指针名>-><成员名>(<参数表>)

这里的“->”是一个表示成员的运算符，它与“.”运算符的区别是“->”用来表示对象的指针的成员，而“.”用来表示一般对象的成员。同样，前者表示数据成员，而后者表示成员函数。

例10.1 分析该程序的输出结果

```
class CDate
{
   public:
      void SetDate(int y, int m, int d);
      int IsLeapYear();
      void Print();
   private:
      int year, month, day;
};
void CDate::SetDate(int y, int m, int d)
{
  year=y;
  month=m;
  day=d;
}
int CDate::IsLeapYear()
{
  return (year%4==0&&year%100!=0)||(year
      %400==0);
}
void CDate::Print()
{
  cout<<year<<"."<<month<<"."<<day<<endl;
}
```

```
#include <iostream.h>
#include "cdate.h"
void main()
{
    CDate date1, date2;
    date1.SetDate(1996,5,4);
    date2.SetDate(1998,4,9);
    int
    leap=date1.IsLeapYear();
    cout<<leap<<endl;
    date1.Print();
    date2.Print();
}
```

例10.2 分析该程序的输出结果

```
class CPoint
{
   public:
      void SetPoint(int x, int y);
      int Xcoord()
        {return X;}
      int Ycoord()
         {return Y;}
      void Move(int xOffset, int
     yOffset);
   private:
      int X, Y;
};
void CPoint::SetPoint(int x, int y)
{
  X=x;
  Y=y;
}
void CPoint::Move(int xOffset, int
     yOffset)
{
  X+=xOffset;
  Y+=yOffset;
}
```

```
#include <iostream.h>
#include "cpoint.h"
void main()
{
   CPoint p1, p2;
   p1.SetPoint(3,5);
   p2.SetPoint(8,10);
   p1.Move(2,1);
   p2.Move(1,-2);

     cout<<"x1="<<p1.Xcoord()<<",y1="<<p1.Ycoo
     rd()<<endl;

     cout<<"x2="<<p2.Xcoord()<<",y2="<<p2.Ycoo
     rd()<<endl;
}
```

三、构造函数与析构函数

- 构造函数和析构函数是在类体中说明的两种特殊的成员函数。
- 构造函数的功能是在创建对象时，用给定的对象对对象进行初始化。
- 析构函数的功能是用来释放一个对象，它与构造函数的功能正好相反。

构造函数的特点如下：

(1) 构造函数是成员函数，函数体可写在类体内，也可写在类体外。

(2) 构造函数是一个特殊的成员函数，该函数的名字与类名相同，该函数不指定类型说明。该函数可以没有参数，也可有参数。

(3) 构造函数可以重载，即可定义多个参数个数不同的函数。

(4) 程序中一般不直接调用构造函数，在创建对象时系统自动调用构造函数。

默认构造函数

没有参数的构造函数称为默认构造函数。默认构造函数有两种：一种是系统自动提供的，另一种是程序员定义的。

在程序中，程序员可以根据需要定义默认构造函数。在一个类中，如果没有定义任何构造函数，则系统自动生成一个默认构造函数。

使用系统提供的默认构造函数给创建的对象初始化时，外部类对象和静态类对象的所有数据成员为默认值，自动类对象的所有数据成员为无意义值。

构造函数举例

```
class Clock
{
public:
    Clock (int NewH, int NewM, int NewS);//构造函数
    void SetTime(int NewH, int NewM, int NewS);
    void ShowTime();
private:
    int Hour,Minute,Second;
};
```

构造函数的实现：

```
Clock::Clock(int NewH, int NewM, int NewS)
{
    Hour= NewH;
    Minute= NewM;
    Second= NewS;
}
```

建立对象时构造函数的作用：

```
void main()
{
    Clock  c (0,0,0); //隐含调用构造函数，将初始值作为实参。
    c.ShowTime();
}
```

拷贝构造函数

拷贝构造函数是用一个已知对象来创造一个新对象，而新创建的对象与已知对象的数据成员的值可以相同，也可以不同。

拷贝构造函数除了具有前述的带参数构造函数相同的特性外，它只有一个参数，并且是对象引用。拷贝构造函数的格式如下：

```
<类名>::<拷贝构造函数名>(<类名> & < 引用名>)
{
    <函数体>
}
```

如果一个类中没有定义拷贝构造函数，则系统会自动提供一个默认拷贝构造函数，作为该类的公有成员，用来根据已知对象创建与其相同的对象。

拷贝构造函数

```
class 类名
{ public :
      类名（形参）；//构造函数
      类名（类名 &对象名）；//拷贝构造函数
         ...
};
类名:: 类名（类名 &对象名）//拷贝构造函数的实现
{   函数体   }
```

拷贝构造函数举例【例10_3】

```
class Point
{
  public:
     Point(int xx=0,int yy=0){X=xx; Y=yy;}
     Point(Point&  p);
     int GetX() {return X;}
     int GetY() {return Y;}
  private:
     int  X,Y;
};
```

```
Point::Point (Point&  p)
{
    X=p.X;
    Y=p.Y;
    cout<<"拷贝构造函数被调用"<<endl;
}
```

拷贝构造函数举例

- 当用类的一个对象去初始化该类的另一个对象时系统自动调用拷贝构造函数实现拷贝赋值。

```
void main(void)
{   Point A(1,2);
    Point B(A); //拷贝构造函数被调用
    cout<<B.GetX()<<endl;
}
```

拷贝构造函数举例

- 若函数的形参为类对象，调用函数时，实参赋值给形参，系统自动调用拷贝构造函数。例如：

```
void fun1(Point p)
{    cout<<p.GetX()<<endl;
}
void main()
{    Point A(1,2);
     fun1(A); //调用拷贝构造函数
}
```

拷贝构造函数

- 当函数的返回值是类对象时，系统自动调用拷贝构造函数。例如：

```
Point fun2()
{       Point A(1,2);
        return A; //调用拷贝构造函数
}
void main()
{
        Point B;
        B=fun2();
}
```

拷贝构造函数

如果程序员没有为类声明拷贝初始化构造函数，则编译器自己生成一个拷贝构造函数。

这个构造函数执行的功能是：将已知对象的所有数据成员的值拷贝给未知对象的所有对应的成员。

析构函数

- 完成对象被删除前的一些清理工作。
- 在对象的生存期结束的时刻系统自动调用它，然后再释放此对象所属的空间。
- 如果程序中未声明析构函数，编译器将自动产生一个默认的析构函数。

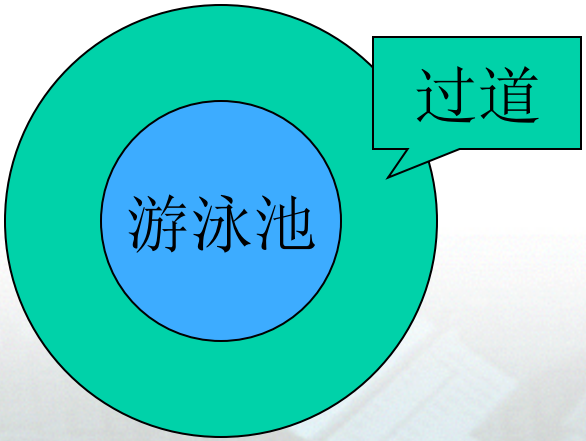
构造函数和析构函数举例

```
#include<iostream>
using namespace std;
class Point
{
  public:
    Point(int xx,int yy);
    ~Point();
    //...其它函数原形
  private:
    int X,int Y;
};
```

```
Point::Point(int xx,int yy)
{      X=xx;    Y=yy;
}
Point::~Point()
{
}
//...其它函数的实现略
```

类的应用举例【10_4】

一圆型游泳池如图所示，现在需在其周围建一圆型过道，并在其四周围上栅栏。栅栏价格为35元/米，过道造价为20元/平方米。过道宽度为3米，游泳池半径由键盘输入。要求编程计算并输出过道和栅栏的造价。



```
#include <iostream>
using namespace std;
const float PI = 3.14159;
const float FencePrice = 35;
const float ConcretePrice = 20;

//声明类Circle 及其数据和方法
class Circle
{
  private:
    float   radius;

  public:
    Circle(float r);  //构造函数

    float Circumference() const; //圆周长
    float Area() const;  //圆面积
};
```

```
// 类的实现
// 构造函数初始化数据成员radius
Circle::Circle(float r)
{radius=r}

// 计算圆的周长
float Circle::Circumference() const
{
    return 2 * PI * radius;
}

// 计算圆的面积
float Circle::Area() const
{
    return PI * radius * radius;
}
```

```
void main ()
{
   float radius;
   float FenceCost, ConcreteCost;

   // 提示用户输入半径
   cout<<"Enter the radius of the pool:
  ";
   cin>>radius;

   // 声明 Circle 对象
   Circle Pool(radius);
   Circle PoolRim(radius + 3);
```

```
// 计算栅栏造价并输出
    FenceCost = PoolRim.Circumference() * FencePrice;
    cout << "Fencing Cost is ￥" << FenceCost << endl;

    // 计算过道造价并输出
    ConcreteCost = (PoolRim.Area() -
     Pool.Area())*ConcretePrice;
    cout << "Concrete Cost is ￥" << ConcreteCost << endl;
}
```

运行结果

```
Enter the radius of the pool: 10
Fencing Cost is ￥2858.85
Concrete Cost is ￥4335.39
```

四、成员函数的特征

在类的定义中规定在类体中说明的函数作为类的成员，称为成员函数。

内联函数和外联函数

类的成员函数可以分为内联函数和外联函数。内联函数是指那些定义在类体内的成员函数，即该函数的函数体放在类体内。说明在类体内、定义在类体外的成员函数叫外联函数。外联函数的函数体位于类的实现部分。

内联函数在调用时不是像一般函数那样要转去执行被调用函数的函数体，执行完成后再转回调用函数中，执行其后的语句；而是在调用函数处用内联函数体的代码来替换，这样将会节省调用开销，提高运行速度。

外联函数变成内联函数的方法只需要在函数头的前面加上关键字**inline**就可以了。

重载性

一般的成员函数都可以进行重载，构造函数可以重载，但是析构函数不能重载。

设置参数的默认值

成员函数可以被设置参数的默认值。一般的成员函数和构造函数都可以被设置参数的默认值。

五、静态成员

静态成员的提出是为了解决数据共享的问题。实现共享有许多方法，例如设置全局性的变量或对象，但是全局变量或对象是有局限性的。因为全局量在程序的任何地方都可以更新，对它的可见范围没法控制。为了安全起见，在程序中很少使用全局量。

实现多个对象之间的数据共享，可以不使用全局对象，而使用静态的数据成员。静态成员包含静态数据成员和静态成员函数。它们都属于类，也属于类的所有对象。

使用静态成员时可以用对象引用，也可以用类来引用。静态数据成员和静态成员函数在没有对象时就已存在，并可以用类来引用。

静态数据成员

静态数据成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态数据成员是类的所有对象共享的成员，而不是某个对象的成员。

使用静态数据成员可以节省内存，静态数据成员只存储一处，供所有对象共用。

静态数据成员的值对每个对象都是一样的，但它的值是可以更新的。

某个对象对静态数据成员的值更新一次，就可保证所有对象都可存取更新后的相同的值。

(1)静态数据成员在定义或说明时前面加关键字static；

static int a;

(2)静态数据成员的初始化与一般数据成员不同；

<数据类型> <类名>::<静态数据成员名>=<初始值>;

(3)引用静态数据成员的格式

<类名>::<静态数据成员名>

静态成员函数

静态成员函数和静态数据成员一样，都属于类，而不属于某个对象。因此，对静态成员的引用可以用类名。

在静态成员函数的实现中可以直接引用类中说明的静态成员，而不可以直接引用类中说明的非静态成员。静态成员函数中要引用非静态成员时，可通过对象来引用。

在main()函数中，调用静态成员函数使用如下格式:

<类名>::<静态成员函数名>(<参数表>)

静态成员函数可以在没有定义 对象之前调用。

Q&A

本讲结束

北京大学