C++ Programming

Chapter 8 Objects and Classes

Zheng Guibin
（郑贵滨）

哈爾濱工業大學
Harbin Institute of Technology

目录

CONTENT

Objects and Classes

对象和类

- 什么是对象、类？
- 类的定义
- 类的使用
- 构造函数
- 类的接口、类的实现
- 析构函数（补充）
- const成员函数（补充）
- 友元(补充)
- static与const(进一步补充)

静态成员

- 问题需求
  - 某些成员不依赖于具体的类对象

- 静态成员
  - 使用static关键字
  - 静态类成员数据
  - 静态类成员函数

静态数据成员

- 静态数据成员的定义
  - static 类型 数据成员名;
- 静态数据成员的初始化
  - 必须在类外进行初始化
  - <类型> <类名>::<静态数据成员>=<值>;
- 静态数据成员使用
  - <类名>::<静态数据成员>
  - <对象名>.<静态数据成员>

静态数据成员

```
class  A
   {…
     static  int var;   //在类的内部声明
      …
   };
 int  A::var = 1;        //在类的外部定义
```

声明的时候，并没有分配内存。

在类的外面进行定义的时候才分配。

//例 静态数据成员的说明和初始化（私有静态）

```
#include<iostream.h>
class  Counter
{     static  int  num ;
  public :
      void  setnum ( int i ) { num = i ; }
      void  shownum() { cout << num << '\t' ; }
} ;
int  Counter :: num = 0 ;//公有、私有均需要初始化否则连接错误
void main ()
{  Counter  a , b ;
    a.shownum() ;      b.shownum() ;
    a.setnum(10) ;
    a.shownum() ;      b.shownum() ;
}
```

声明私有
静态数据成员

成员函数访问
静态数据成员

访问同一个
静态数据成员

调用成员函数访问
私有静态数据成员

// 例　使用公有静态数据成员 （公有静态）

```
#include<iostream.h>
class  counter
{ public :
      void  setnum ( int  i ) { num = i ; }
      void  shownum ( ) { cout << num << '\t' ; }
      static  int  num ;
} ;
int  counter :: num = 1 ;      // 初始值为1
void main()
{  int i ;
   for ( i = 0 ;  i < 5 ;  i ++ )
     { counter :: num += i ;
        cout << counter :: num << '\t' ;
     }
   cout << endl ;
}
```

```
1       2       4       7       11
Press any key to continue
```

静态数据成员

静态数据成员 vs 全局变量

- 有了静态数据成员，各对象之间的数据有了沟通的渠道，实现数据共享 。
- 全局变量破坏了封装的原则，不符合面向对象程序的要求。
- 公用静态数据成员与全局变量的作用域不同，静态数据成员的作用域只限于定义该类的作用域内

静态成员函数

- 静态成员函数定义
  - 在函数声明前加上static 关键字

- 静态成员函数的调用：
  - <对象名>.<静态成员函数名>(<参数表>);
  - <类名>::<静态成员函数名>(<参数表>);

例子：统计对象的数目

```
#include<iostream.h>
class  Counter
{       static  int  num ;
  public :
        Counter( ){ num++; }
       ~ Counter(){ num--; }
    void  setnum ( int i ) { num = i ; }
    void  shownum() { cout << num << '\t' ; }
    static int get(){ return num ; }
} ;
int  Counter :: num = 0 ;
void main ()
{       Counter  a ;     a.shownum() ;
        Counter  b ;     b.shownum() ;
        cout<<Counter::get()<<endl;
}
```

调用
公有静态成员函数

静态成员函数

- 静态成员函数只能访问静态数据成员、静态成员函数和类以外的函数和数据，不能访问类中的非静态数据成员（因为非静态数据成员只有对象存在时才有意义）。
- 任意访问权限许可的成员函数均可访问静态数据成员或静态成员函数。
- 和一般成员函数类似，静态成员函数也有访问限制，私有静态成员函数不能由外界访问。
- 静态成员函数没有this指针，因此，静态成员函数只能直接访问类中的静态成员，若要访问类中的非静态成员时，必须借助对象名或指向对象的指针。

代码模拟

```
#include <iostream.h>
class Student
{ public:
      Student(int n,int a,float s):num(n),score(s){ }
      void total( ) {    sum+=score;    count++; }
      static float average( );          //声明静态成员函数
private:
      int num;
      float score;
      static float sum;                 //静态数据成员
      static int count;                 //静态数据成员
};
float  Student::average( )              //定义静态成员函数
{     return(sum/count);          }
```

```
float Student::sum=0;                    //对静态数据成员初始化
int Student::count=0;                    //对静态数据成员初始化

int main( )
{
      Student stud[3]={   Student(1001,18,70),
                          Student(1002,19,78),
                          Student(1005,20,98)   };
      for(int i=0;i<3;i++)
            stud[i].total( );

      cout<<"the average score of is "
            <<  Student::average( ) <<endl;
      return 0;
}
```

const成员

需求:保护数据,防止随意修改

- const数据成员
- const成员函数
- const对象

const数据成员

- const 类型 数据成员名;
- const数据成员只能由构造函数通过自动初始化列表进行初始化。
- 成员函数不能修改const数据成员的值

const数据成员

```
class Time {
  public:
     Time(int h = 0,int  m= 0,int s = 0) ;
     void setTime( int, int, int );
     int setHour(int );
     int getHour();
     int getMinute();
     void print ();
private:
     const  int hour;
    int minute;          // 0 - 59
    int second;          // 0 - 59
  };
```

```
Time wakeUp( 6, 45, 0 );
wakeUp.setHour( 20 );
wakeUp.getHour();
wakeUp.print ();
```

const数据成员

```
Time :: Time(int h = 0,int  m= 0,int s = 0)
                : hour(h), minute(m), second(s){ }

Time :: Time(int h = 0,int  m= 0,int s = 0)
 {          hour=h;   minute= m; second =s;       }

Time :: Time(int h = 0,int  m= 0,int s = 0): hour(h)
 {        minute= m; second =s;        }
```

const成员函数

- **声明格式:**
  <类型> <函数名>(<参数表>) const;

- **定义格式如下:**
  <类型> <类名::><函数名>(<参数表>) const
  { ……..}

- **若成员函数不修改对象,则声明为const.**

const 成员函数

- 对于const 成员函数需要注意以下几点:

1）const是函数类型的一个组成部分，因此在实现部分也要带const关键词。

2）const成员函数不更新对象的数据成员，也不能调用该类中非const成员函数。

3）const关键词可以参与区分重载函数。

如果在类中有说明:

```
void print();
void print() const;
```

则这是对print的有效重载,编译器根据对象是否为const自动选择所用的重载版本.

const 对象

- 用const 声明的对象称为常量对象。

- const 对象只能调用它的const 成员函数，而不能调用其他成员函数。

代码模拟 - const

```
class Time
{public:
        Time(int h = 0,int  m= 0,int s = 0)
        {  hour=h;  minute= m; second =s;     }
        void setTime( int h, int m, int s)
        {  hour=h;  minute= m; second =s;     }
 private:
        int hour;
        int minute;
        int second;
};
 main()
{       const Time t(20,20,20);
        t.setTime(30,39,39);          //error
}
```

```
#include <iostream.h>
class A
{
public:
        A(int aa=0, int bb=0):a(aa),b(bb){}
        void f() const{cout<<"1234"<<endl;};
        void g(){ cout<<"asdf"<<endl;}
private:
        int a; int b;
};
main()
{       const A t;
        t.f();  //t.g();        error
}
```

const的其他应用

- const引用
  - 不能通过该引用修改相应变量或对象

```
Date   d1;
Date:: Date(const Date & d);
Date dd(d1),
```

const的其他应用

const修饰指针变量

- const放在指针的类型前

  const 数据类型* 指针名;

  例如： const Time* ptr1;

  意义: 不能通过该指针修改所指的数据

- const放在指针名前

  数据类型* const 指针名;

  例如： Time* const ptr2;

  意义: 不能修改指针的值

练习

根据要求，写出类gamma.

```
int main()
{
   gamma g1;
   gamma::showtotal();

   gamma g2, g3;
   gamma::showtotal();

   g1.showid();
   g2.showid();
   g3.showid();
   cout << "----------end of program----------\n";
   return 0;
}
```

练习

1.以下关于静态成员函数的叙述不正确的是（ ）

A. 静态成员函数属于整个类

B. 对静态成员函数的引用允许使用对象名

C. 在静态成员函数中不能直接引用类的非静态成员

D. 使用类名可以直接调用其静态成员函数

2.如果一个成员函数只存取一个类的静态数据成员,可将该成员函数说明为静态成员函数.

3.在类的构造函数中,可以对类的静态数据进行初始化.

4.假定类AB中有一个公用属性的静态数据成员bb，在类外不通过对象名访问该成员bb的写法为_____。

练习

5.对const与类，理解不正确的为：（）

A．可以用const限制对象、数据成员或成员函数；

B．常成员函数不能修改对象的数据成员，但能调用该类中非常成员函数；

C．const数据成员只能由构造函数通过初始化列表对其进行初始化；

D．const 对象只能调用它的const 成员函数，而不能调用非常成员函数；