華中科技大學

面向对象程序设计

第五章 静态成员与友元

许向阳

xuxy@hust.edu.cn

大纲

5.1 静态数据成员

5.2 静态函数成员

5.3 静态成员指针

5.4 友元


```
static  int  *p;    // 静态数据成员
static  int   s;    // 静态数据成员
static  ..  f(…) ;  // 静态函数成员
int  *q =&C.s;      // 静态成员指针
int C:: *mq;        // 数据成员指针
```

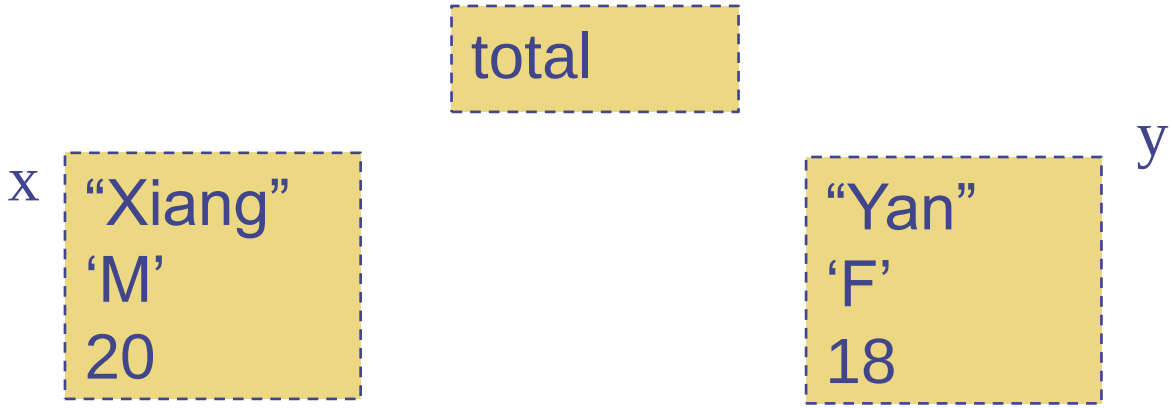
5.1 静态数据成员


```
#include <iostream.h>
class HUMAN{
 private:
   char name[11];
    char sex;
    int  age;
public:
   static int total;  //类体内声明，访问权限public
    HUMAN(char* n,char s,int a)     // 静态成员不能在函数体前初始化
   {     strncpy_s(name, n, 10);
          sex = s;     age = a;      total++;
    }
   ~HUMAN( ) {  total --; };
};

int  HUMAN::total=0;
    //类体外定义并初始化，有访问权限的独立变量
```

5.1 静态数据成员


```
class HUMAN{
public:
   static int total; ……
};
int  HUMAN::total=0;
void main(void){
    HUMAN x((char *) "Xiang", 'M', 20);//HUMAN::total=x.total=1
    HUMAN y((char *) "Yan", 'F', 18); //HUMAN::total=x.total=y.total=2
}
```



sizeof(HUMAN)=16，不包括静态成员total

5.1 静态数据成员

- 静态数据成员用 static 声明
- 静态数据成员的访问控制可以是公有、保护和私有
- 在类体内声明，在类体外定义并初始化
- 静态数据成员独立分配内存，不属于任何对象内存
- 所有对象共享静态数据成员内存，任何对象修改该成员的值，都会同时影响其他对象关于该成员的值。
- 用于描述类的总体信息，如对象总数、连接所有对象的链表表头等。

静态数据成员 VS 静态全局变量 VS 静态局部变量

作用域的差别？　生命周期：整个程序

5.1 静态数据成员

- 局部类不能定义静态数据成员

```
void f(void) {
    class T {          // T 是局部类
        int  c;
        static int d;  // 错误   此规定合理吗？
    };
}
```

- 联合体的数据成员必须共享存储空间，而静态数据成员各自独立分配存储单元，联合不能定义静态数据成员。

```
union UNTP{ static int c;  static long d; };        //错误
```

5.1 静态数据成员

- 静态且为常量数据成员的初始化（声明、定义）：

  static const int x=10;

  但不能在构造函数中初始化

- 静态数据成员必须在体外定义、初始化

  static int x;

  .....

  int 类名:: x = 10;

- 在构造函数中，可以给静态数据成员赋值，但不称为初始化。

常量数据成员初始化

方法1： **const int x=10;**

方法2： **const int x;**

类名（参数）：**x(10) { ..... }** // 构造函数

5.2 静态函数成员

- 在函数的最前面加 static
- 静态函数成员的访问权限及继承规则同普通函数成员没有区别，可以缺省参数、省略参数以及重载。
- 普通函数成员的第一个参数是隐含this指针，指向调用该函数的对象；
- 静态函数成员没有this指针，若无参通常只应访问类的静态数据成员和静态函数成员。

5.2 静态函数成员

- stu1.Modify(100);

```
lea         ecx,[ebp-5Ch]
call        Student::Modify (8E1195h)
```

- stu2.Modify(87);

```
lea         ecx,[ebp-9Ch]
call        Student::Modify (8E1195h)
```

对于一般的函数成员只存放一次；与对象无关，

ecx 中存放对象指针，调用的函数地址相同

5.2 静态函数成员

- 静态函数成员的调用和静态数据成员类似
- 类名::静态函数成员(...);
- 对象.静态函数成员(...) ;
- 对象.类名::静态函数成员(...)。

提倡第(1)种访问形式。

5.2 静态函数成员


```
class A{
    int x; //普通成员必须在对象(同this相关)存在时才能访问
    static int  I;
    static int  f( ){ return  x+ i; }; //错：static int  f( )无this则无x
public:
    static int  m( ) { return  i;  }; //静态函数成员无this
}a;
int A::i=0;        //私有的，体外定义并初始化
void main(void){
    int i=A::f( ); //错误，f私有的
    i=A::m( )+a.m( )+a.A::m( ); //正确，访问公有的静态函数成员
    i=a.f( );      //错误，f私有的不能访问
    i=a.A::f( );   //错误，f私有的不能访问
}
```

5.2 静态函数成员

- 不能使用**static**定义构造函数、析构函数以及虚函数。

```
class A {
   private:
        int x;
   public:
        static A( ) { x=0; };          //错误
        static virtual void f( ) { }  //错误
   };
```

构造函数、析构函数以及虚函数等必须有 **this** 参数

5.2 静态函数成员

- 常函数成员要说明隐含**this**参数。
- 常函数成员不能定义为没有**this**参数的静态函数成员

```
class A {
    static int f( )const      // 错误
        { return 1; }
};
```

- 联合union不能定义静态数据成员，但可以定义静态函数成员。

```
union A {
    int x,y;
    static void f( );
};
```

静态数据成员：

空间分配在对象之外；

多对象共享

类内申明、类外定义和初始化

静态函数成员

无this参数

5.3 静态成员指针

- **静态成员指针**指向类的静态数据（函数）成员。
- 变量、数据成员、普通函数和函数成员的参数和返回值都可以定义成静态成员指针类型。
- 静态成员相当于具有访问权限的变量和函数，同普通变量和函数相比，除访问权限外没有区别，因此，静态成员指针和普通指针形式上也没有区别。
- 静态成员指针存放静态成员地址，普通成员指针存放普通成员偏移；静态成员指针可以移动，普通成员指针不能移动；静态成员指针可以转换类型，普通成员指针不能转换类型。

5.3 静态成员指针


```
class  P{
    char   name[20];
public:
    static int n;
    static int  getn( )  { return n; }
    P(char *n) { strcpy_s(name, n);    n++; }
    ~P( ) { n--; }
}zan((char *)"zan");
int P::n=0;
void main(void){
    int  m;
    int  *d=&P::n;          // d 为静态成员指针；等价于d=&zan.n;
                            //静态成员指针 与 普通指针 没有差别
    int (*f)( )=&P::getn;  //普通函数指针指向静态函数成员
    m=*d + (*f)( );
}
```

5.3 静态成员指针


```
struct A{
    int a, *b, A::*u, A::*A::*x, A::**y, *A::*z;
    static int c, A::*d;
};z;
int A::c=0, A::*A::d=&A::a;  //静态数据成员初始化
void main(void){     //注意优先级高低：  ".">"*">".*"
    int i, A::**m;
    z.a=5;        z.u=&A::a;  i=z.*z.u; //i=z.*&A::a=z.a=5
    z.x=&A::u; i=z.*(z.*z.x);
    //i=z.*(z.*&A::u)=z.*z.u=z.a=5
    m=&A::d;  m=&z.u;  i=z.**m; //i=z.**&z.u=z.*z.u
    z.y=&z.u;    i=z.**z.y; // i=z.**&z.u=z.*z.u=z.a=5
    z.b=&z.a;    z.z=&A::b;  i=*(z.*z.z);
    //i=*(z.*&A::b)=*z.b=*&z.a
}
```

5.4 友元

```
class Student {
  private:
     int  number;
     char name[15];
     float score;
  public:
      Student(int number1, char* name1, float score1);
      Student() {  };
      Student(Student &a);
      void Print();   // 显示信息
      friend void display(Student &a); // 显示信息
};

void display(Student &a) { ……}
```

5.4 友元


```
void Student::Print()
{
    cout << " name : " << name << " score: " << score << "\n";
}
void display(Student &s)
{
   cout << " name : " << s.name << " score: " << s.score << "\n";
}                                                    // 访问私有成员
int  main()
{   Student  stu1(……);
    Stu1.Print();
    display(stu1);
}
```

5.4 友元

```
void Student::Print()
{ ……  }

void display(Student &s)
{ ……  }
```

普通友元：C语言普通函数

display 声明为类的友元

缺点：　破坏了类中信息隐藏的特性，可访问私有成员

为什么定义友元函数，而不直接将其定义为成员函数？

优点：提高程序的运行效率

减少类型检查和安全性检查的时间开销

5.4 友元

友元函数

- 友元函数不是声明该友元的当前类的成员
- 不受访问权限的限制，可以在任何访问权限下用**friend**
- 可以访问当前类的任何成员
- 一个函数可成为多个类的友员

- 若在类体定义友元函数体，友元函数自动成为内联函数
- 同其他内联函数一样，内联有可能失败。

5.4 友元

友元类：一个类是另外一个类的友元

```
class A {
        friend class B;   // B为类A的友元类
};
class B { …b1(…); …b2(…); };
```

- 类B 的所有函数成员都是类A的友元
- 关系不传递
- 友元关系不对称
- static、virtual、friend 只能单个独立使用。

5.4 友元

友元类：一个类是另外一个类的友元

```
#include <iostream>
using  namespace  std;
class  A {
private:
        int x;
public:
        A(int x) {  this->x = x; }
        friend class B;   // 类B 为类A的友元
};                  // 类B的所有函数成员都是类A的友元，
                    // 均可访问A的私有成员
```

5.4 友元


```
class B {
  public:
     void display(A &a)
       {  cout << "display A :" << a.x << endl;  }
};
// 若删除A类中的申明  friend class B;
// 则B类中的display函数不能访问A的私有成员
int main()
{        A  a(10);
         B  b;
         b.display(a);
         return 0;
}
```

5.4 友元

一个类的成员函数能否成为另一个类的友元？

```
#include <iostream>
using namespace std;
class B {
    public: void display();
};
class  A {
private:
    int x;
public:
     A(int x) { this->x = x; };
     friend void  B::display();
};
```

5.4 友元


```
void B::display() {
        A  a(10);
        cout << "display A :" << a.x << endl;
}

int main()
{
        B  b;
        b.display();
        return 0;
}
```

5.4 友元

类成员函数作为另一个类的友元

类定义的排列顺序如何？

能否将A 类排列在B的前面？　不能！

能否将 B类中的函数display的实现放在B类定义中？　不能！

如果函数改为 public: void display(A &a);
用 VS2019 能否写出相应的程序？

总结

数据成员、常数据成员、静态数据成员

函数成员、常函数成员、静态函数成员

const　　static

成员指针、静态成员指针

友元　friend

普通友元