第 5 章　数据的共享与保护

主教材要点导读

本章主要介绍与程序的结构、模块间的关系和数据共享相关的内容。读者学习这一章时的主要问题可能是感觉到与其他章节相比，这一章显得有些芜杂，语法规定很多。不过，只要循着程序结构和数据共享这两条主线，思路就会比较清晰。

标识符的作用域和对象的生存期问题，是研究程序模块之间数据传递、数据共享的基础。静态成员是类的对象之间共享数据和代码的手段。友元是不同的类之间、类与类外的函数之间共享数据的机制。而常引用、常对象、常成员为共享的数据提供了保护机制。使用多文件结构，有利于大型项目的分工合作、分别开发。如果要在一个项目的不同程序文件之间共享数据和代码，就要用到外部变量和外部函数。

本章内容语法规定较多，有的读者对这些语法规定不太理解，总想找个老师问问：能不能这样写？是不是会有那样的效果？如果周围没有人可以请教，常常感到束手无策。有的读者逐一理解这些语法规定倒也不难，但是会觉得记不住，还会混淆，其实根本原因还是没有将每一个问题理解透彻。我建议读者要自己验证每一个语法规定，用反证的方法更有助于理解和加深印象。比如，语法规定当程序流程离开了一个变量的作用域，就不能使用该变量，那么你可以编一段程序，尝试在变量的作用域之外使用这个变量，看看后果是什么。还可以尝试用普通的成员函数去处理常对象，看看会是什么情况。这样验证以后，很多疑问就解开了。

以后学习后续章节时也是这样的，如果有些问题反复看都不能理解、反复想都想不清楚，那就不是看和想能解决的了，这时就需要自己动手编一些程序来试验，效果往往不错。

实验 5　数据的共享与保护(2 学时)

一、实验目的

(1) 观察程序运行中变量的作用域、生存期和可见性。

(2) 学习类的静态成员的使用。

(3) 学习多文件结构在 C++ 程序中的使用。

二、实验任务

(1) 运行下面的程序，观察变量 x、y 的值。

```
//lab5_1.cpp
#include <iostream>
```

```
using namespace std;

void fn1();
int x=1, y=2;

int main()
{
    cout<<"Begin..."<<endl;
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
    cout<<"Evaluate x and y in main()..."<<endl;
    int x=10, y=20;
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
    cout<<"Step into fn1()..."<<endl;
    fn1();
    cout<<"Back in main"<<endl;
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
    return 0;
}
void fn1()
{
    int y=200;
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
}
```

(2) 实现客户机(CLIENT)类。声明字符型静态数据成员 ServerName,保存其服务器名称;整型静态数据成员 ClientNum,记录已定义的客户数量;定义静态函数 ChangeServerName( ) 改变服务器名称。在头文件 client.h 中声明类,在文件 client.cpp 中实现,在文件 test.cpp 中测试这个类,观察相应的成员变量取值的变化情况。

三、实验步骤

(1) 运行 lab5_1 程序,观察程序输出。全局变量的作用域为文件作用域,在整个程序运行期间有效,但如果在局部模块中声明了同名的变量,则在局部模块中,可见的是局部变量,此时,全局变量不可见;而局部变量的生存期只限于相应的程序模块中,离开相应的程序模块,局部变量 x、y 就不再存在,此时同名的全局变量重新可见。

(2) 新建一个空的项目 lab5_2,添加头文件 client.h,在其中声明类 CLIENT,注意使用编译预处理命令;再添加源程序文件 client.cpp,在其中实现 CLIENT 类,注意静态成员变量的使用方法;再添加文件 lab5_2.cpp,在其中定义 main( )函数,测试 CLIENT 类,观察相应的成员变量取值的变化情况。

习 题 解 答

5-1 什么叫作用域？有哪几种类型的作用域？

解：作用域讨论的是标识符的有效范围，作用域是一个标识符在程序正文中有效的区域。C++ 的作用域分为函数原型作用域、块作用域（局部作用域）、类作用域和文件作用域。

5-2 什么叫可见性？可见性的一般规则是什么？

解：可见性是标识符是否可以引用的问题。

可见性的一般规则是：标识符要声明在前，引用在后；在同一作用域中，不能声明同名的标识符。对于在不同的作用域声明的标识符，遵循的原则是：若有两个或多个具有包含关系的作用域，外层声明的标识符如果在内层没有声明同名标识符时仍可见，如果内层声明了同名标识符则外层标识符不可见。

5-3 下面程序的运行结果是什么？实际运行一下，看看与你的设想有何不同。

```
#include <iostream>
using namespace std;
int x=5, y=7;
void myFunction()
{
    int y=10;

    cout<<"x from myFunction: "<<x<<"\n";
    cout<<"y from myFunction: "<<y<<"\n\n";
}
int main()
{

    cout<<"x from main: "<<x<<"\n";
    cout<<"y from main: "<<y<<"\n\n";
    myFunction();
    cout<<"Back from myFunction!\n\n";
    cout<<"x from main: "<<x<<"\n";
    cout<<"y from main: "<<y<<"\n";
    return 0;
}
```

解：程序运行输出：

```
x from main: 5
y from main: 7

x from myFunction: 5
y from myFunction: 10
```

```
Back from myFunction!

x from main: 5
y from main: 7
```

5-4 假设有两个无关系的类 engine 和 fuel，使用时，怎样允许 fuel 成员访问 engine 中的私有和保护的成员？

解：源程序：

```
class fuel;
class engine
{
        friend class fuel;
    private;
        int powerlevel;
    public;
        engine(){ powerLevel=0;}
        void engine_fn(fuel &f);
};
class fuel
{
        friend class engine;
    private;
        int fuelLevel;
    public:
        fuel(){ fuelLevel=0;}
        void fuel_fn( engine &e);
};
```

5-5 什么叫作静态数据成员？它有何特点？

解：类的静态数据成员是类的数据成员的一种特例，采用 static 关键字来声明。对于类的普通数据成员，每一个类的对象都拥有一份存储，就是说每个对象的同名数据成员可以分别存储不同的数值，这也是保证对象拥有自身区别于其他对象的特征的需要，但是静态数据成员，每个类只要一份存储，由所有该类的对象共同维护和使用，这个共同维护、使用也就实现了同一类的不同对象之间的数据共享。

5-6 什么叫作静态函数成员？它有何特点？

解：使用 static 关键字声明的函数成员是静态的，静态函数成员属于整个类，同一类的所有对象共同维护，为这些对象所共享。静态函数成员具有以下两方面的好处，一是由于静态成员函数只能直接访问同一个类的静态数据成员，可以保证不会对该类的其余数据成员造成负面影响；二是同一个类只维护一个静态函数成员的拷贝，节约了系统的开销，提高程序的运行效率。

5-7 定义一个 Cat 类，拥有静态数据成员 numOfCats，记录 Cat 的个体数目；静态成员函数 getNumOfCats()，存取 numOfCats。设计程序测试这个类，体会静态数据成员和静态

成员函数的用法。

解：源程序：

```
#include <iostream>
using namespace std;

class Cat {
public:
    Cat(int age):
        itsAge(age) {
        numOfCats++;
    }
    virtual ~Cat() {
        numOfCats--;
    }
    virtual int getAge() {
        return itsAge;
    }
    virtual void setAge(int age) {
        itsAge=age;
    }
    static int getNumOfCats() {
        return numOfCats;
    }
private:
    int itsAge;
    static int numOfCats;
};

int Cat:: numOfCats=0;

void telepathicFunction();

int main() {
    const int maxCats=5;
    Cat * catHouse[maxCats];
    int i;
    for (i=0; i<maxCats; i++) {
        catHouse[i]=new Cat(i);
        telepathicFunction();
    }

    for (i=0; i<maxCats; i++) {
        delete catHouse[i];
        telepathicFunction();
```

```
    }
    return 0;
}

void telepathicFunction() {
    cout<<"There are "<<Cat:: getNumOfCats()<<" cats alive!\n";
}
```

程序运行输出：

```
There are 1 cats alive!
There are 2 cats alive!
There are 3 cats alive!
There are 4 cats alive!
There are 5 cats alive!
There are 4 cats alive!
There are 3 cats alive!
There are 2 cats alive!
There are 1 cats alive!
There are 0 cats alive!
```

5-8 什么叫作友元函数？什么叫作友元类？

解：友元函数是使用 friend 关键字声明的函数，它可以访问相应类的保护成员和私有成员。友元类是使用 friend 关键字声明的类，它的所有成员函数都是相应类的友元函数。

5-9 如果类 A 是类 B 的友元，类 B 是类 C 的友元，类 D 是类 A 的派生类，那么类 B 是类 A 的友元吗？类 C 是类 A 的友元吗？类 D 是类 B 的友元吗？

解：类 B 不是类 A 的友元，友元关系不具有交换性；

类 C 不是类 A 的友元，友元关系不具有传递性；

类 D 不是类 B 的友元，友元关系不能被继承。

5-10 静态成员变量可以为私有的吗？声明一个私有的静态整型成员变量。

解：可以，例如：

```
private:
    static int a;
```

5-11 在一个文件中定义一个全局变量 n，主函数 main( )，在另一个文件中定义函数 fn1( )，在 main( )中对 n 赋值，再调用 fn1( )，在 fn1()中也对 n 赋值，显示 n 最后的值。

解：

```
#include <iostream>
using namespace std;

#include "fn1.h"
```

```
int n;
int main()
{
    n=20;
    fn1();
    cout<<"n 的值为" <<n;
    return 0;
}

//fn1.h 文件
extern int n;

void fn1()
{
    n=30;
}
```

程序运行输出：

```
n 的值为 30
```

5-12 在函数 fn1( )中定义一个静态变量 n，fn1( )中对 n 的值加 1，在主函数中，调用 fn1( )10 次，显示 n 的值。

解：

```
#include <iostream>
using namespace std;

void fn1()
{
    static int n=0;
    n++;
    cout<<"n 的值为"<<n <<endl;
}

int main()
{
    for(int i=0; i<10; i++)
        fn1();
    return 0;
}
```

程序运行输出：

```
n 的值为 1
n 的值为 2
n 的值为 3
n 的值为 4
n 的值为 5
n 的值为 6
n 的值为 7
n 的值为 8
n 的值为 9
n 的值为 10
```

5-13 定义类 X、Y、Z，函数 h(X＊)，满足：类 X 有私有成员 i，Y 的成员函数 g(X＊)是 X 的友元函数，实现对 X 的成员 i 加 1；类 Z 是类 X 的友元类，其成员函数 f(X＊)实现对 X 的成员 i 加 5；函数 h(X＊)是 X 的友元函数，实现对 X 的成员 i 加 10。在一个文件中定义和实现类，在另一个文件中实现 main( )函数。

解：

```
#include "my_x_y_z.h"
int main()
{
    X x;
    Z z;
    z.f(&x);
    return 0;
}

//my_x_y_z.h 文件
#ifndef MY_X_Y_Z_H

class X;
class Y {
public:
    void g(X*);
};

class X
{
private:
    int i;
public:
    X(){i=0;}
    friend void h(X*);
    friend void Y:: g(X*);
    friend class Z;
};
```

```
void h(X * x) { x->i =+10; }

void Y:: g(X * x) { x->i ++; }

class Z {
public:
    void f(X * x) { x->i +=5; }
};

#endif        //MY_X_Y_Z_H
```

程序运行输出：

```
无
```

5-14 定义 Boat 与 Car 两个类，二者都有 weight 属性，定义二者的一个友元函数 getTotalWeight( )，计算二者的重量和。

解：源程序：

```
#include <iostream>
using namespace std;

class Boat;
class Car {
private:
    int weight;
public:
    Car(int j) {
        weight=j;
    }
    friend int getTotalWeight(Car &aCar, Boat &aBoat);
};

class Boat {
private:
    int weight;
public:
    Boat(int j) {
        weight=j;
    }
    friend int getTotalWeight(Car &aCar, Boat &aBoat);
};

int getTotalWeight(Car &aCar, Boat &aBoat) {
    return aCar.weight +aBoat.weight;
}
```

```
int main() {
    Car c1(4);
    Boat b1(5);

    cout<<getTotalWeight(c1, b1)<<endl;
    return 0;
}
```

程序运行输出：

```
9
```

5-15 在函数内部定义的普通局部变量和静态局部变量在功能上有何不同？计算机底层对这两类变量做了怎样的不同处理，导致了这种差异？

解：局部作用域中静态变量的特点是：它并不会随着每次函数调用而产生一个副本，也不会随着函数返回而失效，定义时未指定初值的基本类型静态生存期变量，会被以 0 值初始化；局部作用域中的全局变量诞生于声明点，结束于声明所在的块执行完毕之时，并且不指定初值意味着初值不确定。

普通局部变量存放于栈区超出作用域后，变量被撤销，其所占用的内存也被收回；静态局部变量存放于静态数据存储区，全局可见，但是作用域是局部作用域，超出作用域后变量仍然存在。

5-16 编译和连接这两个步骤的输入、输出分别是什么类型的文件？两个步骤的任务有什么不同？在以下几种情况下，在对程序进行编译、连接时是否会报错？会在哪个步骤报错？

(1) 定义了一个函数 void f(int x, int y)，以 f(1)的形式调用。

(2) 在源文件起始处声明了一个函数 void f(int x)，但未给出其定义，以 f(1)的形式调用。

(3) 在源文件起始处声明了一个函数 void f(int x)，但未给出其定义，也未对其进行调用。

(4) 在源文件 a.cpp 中定义了一个函数 void f(int x)，在源文件 b.cpp 中也定义了一个函数 void f(int x)，试图将两个源文件编译后连接在一起。

解：编译的输入文件是源文件，输出是目标文件；连接的输入文件是目标文件，输出是可执行文件。

编译器对源代码进行编译，是将以文本形式存在的源代码翻译为机器语言形式的目标文件的过程。连接是将各个编译单元的目标文件和运行库当中被调用过的单元加以合并后生成的可执行文件的过程。

(1) 编译时报错，函数参数不匹配。

(2) 连接错误，函数未定义。

(3) 不报错。

(4) 连接错误，函数重复定义。