第五章 数据的共享与保护

信息学院-沙金

目录

this指针

- 一个类的多个对象各自拥有类的数据成员的一个副本
- 类的成员函数则被其所有对象共享
- 一个类的不同对象响应相同的消息时，调用的是同一个函数
- 问题：成员函数如何知道它要操作哪个对象的数据？

this指针

- 每个对象都有一个指向自身的this指针
- this的值是当前对象的起始地址。
- 对象调用成员函数时会将自己的this指针传递给成员函数（隐含参数）

5.3.1 静态数据成员

- 用关键字static声明
- 为该类的所有对象共享
- 必须在类外定义和初始化

原理

- 同一类的不同对象，**其成员数据之间是互相独立的。**
- 当我们将类的某一个**数据成员的声明为static**，则由该类所产生的所有对象，其静态成员均共享**一个存储空间**

static规定

1. static成员必须在类外定义并初始化
2. static数据成员和函数成员可以通过对象名引用也可以通过类名引用
3. static成员函数只能访问static数据成员，不能访问非static成员
4. 普通成员函数可以访问static数据成员.

例 静态数据成员举例

```
class Point {//类定义
public:    //外部接口
  Point(int x=0,int y=0):x(x),y(y){
      count++; //所有对象共同维护同一个count
  }
  Point(Point &p) {   //复制构造函数
      x = p.x;
      y = p.y;
      count++;
  }
  ~Point() {  count--; }
  int getX() { return x; }
  int getY() { return y; }
```

```
  void showCount() {              //输出静态数据成员
      cout << "  Object count = " << count << endl;
  }
private:     //私有数据成员
  int x, y;
  static int count;//静态数据成员声明，用于记录点的个数
};
int Point::count = 0;//必须定义和初始化
int main() {//主函数
  Point a(4, 5);   //定义对象a，其构造函数回使count增1
  cout << "Point A: " << a.getX() << ", " << a.getY();
  a.showCount();   //输出对象个数

  Point b(a);      //定义对象b，其构造函数回使count增1
  cout << "Point B: " << b.getX() << ", " << b.getY();
  b.showCount();   //输出对象个数
  return 0;
}
```

例5-4（续）

- 运行结果：

```
Point A: 4, 5  Object count=1
Point B: 4, 5  Object count=2
```

5.3.2 静态函数成员

- 类外代码可以使用类名和作用域操作符来调用静态成员函数。
- 静态成员函数只能引用属于该类的静态数据成员或静态成员函数。

例 静态函数成员举例

```
class Point {
public:
  Point(int x=0,int y=0):x(x),y(y){
      //所有对象共同维护同一个count
      count++;
  }
  Point(Point &p) {
      x = p.x;
      y = p.y;
      count++;
  }
  ~Point() {  count--; }
  int getX() { return x; }
  int getY() { return y; }
```

```
private:  //私有数据成员
  int x, y;
  static int count;  //静态数据成员声明
};

int Point::count = 0;//定义和初始化

int main() {
  Point a(4, 5);     //构造函数使count增1
  cout<<"Point A:"<<a.getX()<<","<<a.getY();
  Point::showCount();

  Point b(a);   //构造函数使count增1
  cout<<"Point B:"<<b.getX()<<","<<b.getY();
  Point::showCount();
  return 0;
}
```

5.4 类的友元

- 友元是C++提供的一种破坏数据封装和数据隐藏的机制。
- 通过将一个模块声明为另一个模块的友元，一个模块能够引用到另一个模块中本是被隐藏的信息。
- 可以使用友元函数和友元类。

友元

- 类的数据成员都设为私有的——良好的程序设计风格
- 如果某个（些）外部函数需要直接访问类的数据成员，怎么办？
  - 方法一：数据公有
  - 方法二：友元
- 友元函数,友元类,friend

5.4.1 友元函数

- 友元函数是在类声明中由friend修饰说明的非成员函数，在它的函数体中能够通过对象名访问private和protected成员
- 作用：增加灵活性，使程序员可以在封装和隐藏性方面做合理选择。

友元函数

- **类中私有和保护的成员在类外不能被访问**
- 友元函数是一种定义在类外部的普通函数，其特点是**能够访问类中私有成员和保护成员**，即类的访问权限的限制对其不起作用。
- 友元函数不是成员函数，用法也与普通的函数完全一致，只不过它能访问类中所有的数据。

友元函数的特点

- 友元函数近似于普通的函数，它不带有this指针，因此必须将对象名或对象的引用作为友元函数的参数，这样才能访问到对象的成员。
- 在类中对友元函数指定访问权限是不起作用的。

友元函数与一般函数的区别

1. **友元函数必须在类中声明**，其函数体可在类内定义，也可在类外定义；
2. **它可以访问该类中的所有成员（公有的、私有的和保护的）**，而一般函数只能访问类中的公有成员。

例5-6 使用友元函数计算两点间的距离

```
#include <iostream>
#include <cmath>
class Point { //Point类声明
public:  //外部接口
  Point(int x=0,int y=0):x(x),y(y){ }
  int getX() { return x; }
  int getY() { return y; }
  friend float dist(Point &a, Point &b);
private: //私有数据成员
  int x, y;
};
```

例5-6（续）

```
float dist( Point& a, Point& b) {
  double x=a.x-b.x;
  double y=a.y-b.y;
  return sqrt(x*x+y*y);
}
int main() {
  Point p1(1,1), p2(4,5);
  cout<<"The distance is:";
  cout<<dist(p1,p2)<<endl;
  return 0;
}
```

运行结果：

```
The distance is: 5
```

5.5 共享数据的保护

- 对于既需要共享、又需要防止改变的数据应该声明为**常类型**（用const进行修饰）。
- 对于不改变对象状态的成员函数应该声明为**常函数**。

5.5 共享数据的保护

- 常对象：必须进行初始化，不能被更新。

  const 类名 对象名

- 常成员用const进行修饰的类成员：

  常数据成员和常函数成员

- 常引用：被引用的对象不能被更新。

  const 类型说明符 &引用名

5.5.1 常对象

```
class A
{
  public:
    A(int i,int j) {x=i; y=j;}
                          ...
  private:
    int x,y;
};
A const a(3,4); //a是常对象不能被更新
```

5.5.2用const修饰的对象成员

- 常成员函数
  - 使用const关键字说明的函数。
  - 常成员函数不更新对象的数据成员。
  - 格式：类型说明符　函数名（参数表）const;
    在实现部分也要带const关键字。
  - const关键字可以被用于参与对重载函数的区分
- 通过常对象只能调用它的常成员函数。
- 常数据成员，用const说明的数据成员。

例5-7 常成员函数举例

```
#include<iostream>
using namespace std;
class R{
public:
   R(int r1,int r2):r1(r1),r2(r2){ }
   void print();
   void print() const;
private:
   int r1,r2;
};
```

例5-7（续）

```
void R::print(){
  cout<<r1<<":"<<r2<<endl;
}
void R::print() const{
  cout<<r1<<";"<<r2<<endl;
}
int main() {
  R a(5,4);
  a.print(); //调用void print()
  const R b(20,52);
  b.print(); //调用void print() const
   return 0;
}
```

运行结果：

```
5:4
20;52
```

例5-8 常数据成员举例

```
#include <iostream>
using namespace std;
class A {
public:
  A(int i);
  void print();
private:
  const int a;
  static const int b;   //静态常数据成员
};
```

例5-8（续）

```
const int A::b=10;
A::A(int i) : a(i) { }
void A::print() {
  cout << a << ":" << b <<endl;
}
int main() {
  A a1(100), a2(0);
  a1.print();
  a2.print();
  return 0;
}
```

运行结果：

100:10

0:10

小结

- 主要内容
  - 作用域与可见性、对象的生存期、数据的共享与保护、友元、编译预处理命令、多文件结构和工程
- 达到的目标
  - 理解程序的结构、模块间的关系、数据共享。