第五周 继承

第一节 继承和派生

1.继承

继承是在定义一个新类 B 时，如果该类与某个已有类 A 相似（指 B 至少拥有 A 的全部特点），那么就把 A 作为一个**基类**，B 作为基类的一个**派生类**。

派生类是通过对基类进行修改和扩充得到的。在派生类中，可以扩充新的成员变量和成员函数。派生类一经定义后，可以独立使用，不依赖于基类。

派生类拥有基类的全部成员函数和成员变量，不论是 private、protected、public。**但是在派生类的各个成员函数中，不能访问基类中的 private 成员。**

派生类的写法：

```
class 派生类名: public 基类名
{
};
```

例程：

```
class CStudent{
    private:
    string sName;
    int nAge;
    public:
    bool IsThreeGood(){};
    void SetName(const string & name)
    {sName = name;}
};
class CUndergraduateStudent: public CStudent{
    private:
    int nDepartment;
    public:
    bool IsThreeGood(){}//覆盖
    bool CanBaoYan(){};
};
class CGraduatedStudent: public CStudent{
    private:
    int nDepartment;
    char szMentorName[20];
    public:
    int CountSalary(){};
};
```

2.派生类对象的内存空间

派生类对象的体积，等于基类对象的体积再加上派生类对象自己的成员变量的体积。**在**

派生类对象中，包含着基类对象，而且基类对象的存储位置位于派生类对象新增的成员变量之前。

完整程序如下：

```
#include <iostream>
#include <string>
#include <cstdlib>
using namespace std;
class CStudent {
private:
    string name;
    string id;
    char gender;
    int age;
public:
    void PrintInfo();
    void SetInfo(const string & name_, const string & id_, int age_, char gender_);
    string GetName() { return name; }
};
class CUndergraduateStudent : public CStudent
{
private:
    string department;
public:
    void QualifiedForBaoyan() {
        cout << "qualified for baoyan" << endl;
    }
    void PrintInfo() {
        CStudent::PrintInfo();//因为我也要输出基类的信息，覆盖就没办法直接输出了，所以先调用下基类的该函数，如果是构造函数就可以直接省略，进行隐式方式调用
        cout << "Department:" << department << endl;
    }
    void SetInfo(const string & name_, const string & id_, int age_, char gender_, const string & department_) {
        CStudent::SetInfo(name_, id_, age_, gender_);
        department = department_;
    }
};
void CStudent::PrintInfo() {
    cout << "Name:" << name << endl;
    cout << "ID:" << id << endl;
    cout << "Age:" << age << endl;
    cout << "Gender:" << gender << endl;
}
void CStudent::SetInfo(const string & name_, const string & id_, int age_, char
```

```
gender_) {
    name = name_;
    id = id_;
    age = age_;
    gender = gender_;
}
int main() {
    CUndergraduateStudent s2;
    s2.SetInfo("Zhang Yushuai", "011210149", 23, 'M', "comunication");

    s2.QualifiedForBaoyan();
    s2.PrintInfo();
    system("pause");
    return 0;
}
```

第二节 继承关系和复合关系

类与类的关系有三种：①没有关系；②继承关系；③符合关系。

继承：“是”的关系。基类 A，B 是基类 A 的派生类。逻辑上的要求：“一个 B 对象也是一个 A 对象”。

复合：“有”关系。

类 C 中“有”成员变量 k，k 是类 D 的对象，则 C 和 D 是符合关系。一般逻辑上的要求：D 对象是 C 对象的固有属性或组成部分。

1.复合关系的使用

几何形体程序中，需要写“点”类，也需要写“圆类”，两者的关系就是复合关系——每一个圆对象里都包含一个点对象，即圆心。代码如下：

```
class CPoint{
    double x,y;
    friend class CCircle;
}

class CCicle{
    double r;
    CPoint center;//圆心
}
```

举例：写一个小区养狗管理程序，需要些一个业主类，还需要些一个狗来。而狗是由猪肉的，主人当然有业主。假定狗只有一个主人，但一个业主可以有最多 10 条狗。

写法：为“狗”类设一个“业主”类的对象指针；为“业主”类设一个“狗”类的对象指针数组。

```
class CMaster;
class CDog{
    CMaster *pm;
};
```

```
class CMaster{
    CDog *dogs[10];
};
```

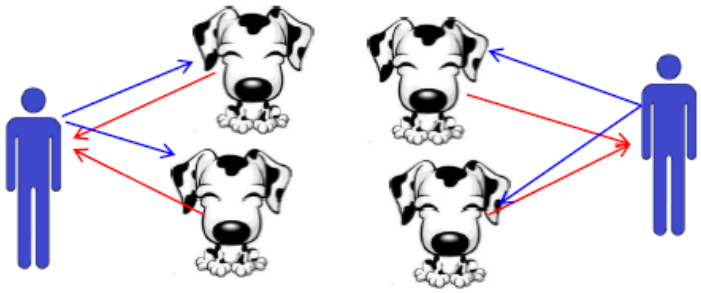

图 2.1 复合关系示意

第三节 覆盖和保护成员

1.覆盖

派生类可以定义一个和基类成员同名的成员，这叫覆盖。在派生类中访问这类成员时，缺省的情况是访问派生类定义的成员。要在派生类中访问由基类定义的同名成员时，要使用作用域符号::。

例如：

```
class base{
    int j;
    public:
    int i;
    void func();
};
class derived:public base{
    public:
    int i;
    void access();
    void func();
};

void derived::access(){
    j = 5;//error
    i = 5;//引用的是派生类的 i
    base::i = 6;//引用的是基类的 i
    func();//派生类的
    base::func();//基类的
}
int main()
{
    derived obj;
    obj.i = 1;
```

```
    base::obj.i = 2;
    return 0;
}
```

一般来说，基类和派生类不定义同名成员变量，但同名成员函数很常见。

2.类的保护成员

（1）protected

基类的 private 成员，可以被下列函数访问：

①基类的成员函数；

②基类的友元函数

基类的 public 成员可以被下列函数访问：

①基类的成员函数；

②基类的友元函数；

③派生类的成员函数；

④派生类的友元函数；

⑤其它的函数。

基类的 protected 成员可以被下列函数访问：

①基类的成员函数；

②基类的友元函数；

③派生类的成员函数可以访问当前对象的基类的保护成员。

例子：

```
class Father{
    private: int nPrivate;//私有成员
    public: int nPublic;//公有成员
    protected: int nProtected;//保护成员
};
class Son:public Father{
    void AccessFather(){
        nPublic = 1;//ok
        nPrivate = 1;//wrong
        nProtected = 1;//OK，访问从基类基础的 Protected 成员可以
        Son f;
        f.nProtected = 1;//wrong，f 不是当前对象
    }
};
int main(){
    Father f;
    Son s;
    f.nPublic = 1;//ok
    s.nPublic = 2;//ok
    f.nProtected = 3;//error 只能在本类的成员函数，派生类的成员函数，友元的成员函数访问
    f.nPrivate = 4;//error 基类的对象没办法访问基类的私有对象
    s.nProtected = 5;//error 派生类的对象无法访问基类的保护成员
    s.nPrivate = 6;//error 派生类的对象无法访问派生类从基类得到的私有成员
```

```
    return 0;
}
```

第四节 派生类的构造函数

1.派生类的构造函数

```
class Bug{
    private:
    int nLegs;
    int nColor;
    public:
    int nType;
    Bug(int legs, int color);
    void PrintBug(){};
};
class FlyBug:public Bug{
    int nWings;
    public:
    FlyBug(int legs, int color, int wings);
};
Bug::Bug(int legs, int color)
{
    nLegs = legs;
    nColor = color;
}
FlyBug::FlyBug(int legs, int color, int wings):Bug(legs,color){
    nWings = wings;
}
int main(){
    FlyBug fb(2,3,4);
    fb.PrintBug();
    fb.nType = 1;
    fb.nLegs = 2;//error 无法直接访问私有成员
    return 0;
}
```

其实派生类的构造函数大部分都很简单，和普通的构造函数一样，但是派生类的构造函数如何继承来自基类的私有成员呢？其实在第一节 2 的代码给了我们一个答案，就是如下，利用了初始化列表调用 Bug 类中的构造函数来实现，第一节 2 的代码是直接写在了代码块里面。

```
FlyBug::FlyBug(int legs, int color, int wings):Bug(legs,color){
    nWings = wings;
}
```

这两种方式总结如下：

（1）显式方式：在派生类的构造函数中，为基类的构造函数提供参数。格式如下：

derived::derived(arg_derived-list):base(arg_base-list)

（2）隐式方式：在派生类的构造函数中，省略基类构造函数时，派生类的构造函数则自动调用基类的默认构造函数（无参构造函数，若不存在则报错）。

在创建派生类的对象时，需要调用基类的构造函数：初始化派生类对象中从基类继承的成员。**在执行一个派生类的构造函数之前，总是先执行基类的构造函数。**

派生类的析构函数被执行时，执行完派生类的析构函数后，自动调用基类的析构函数。

2.包含成员对象的派生类（封闭类）的构造函数写法

代码如下：

```
class Bug{
    private:
    int nLegs;
    int nColor;
    public:
    int nType;
    Bug(int legs, int color);
    void PrintBug(){};
};
class Skill{
    public:
    Skill(int n){}
};
class FlyBug:public Bug{
    int nWings;
    Skill sk1,sk2;
    public:
    FlyBug(int legs, int color, int wings);
};
FlyBug::FlyBug(int legs, int color, int wings):Bug(legs,color),sk1(5),sk2(color),nWings(wings){}
```

（吐槽：都这么喜欢初始化列表，回头查查初始化列表仔细了解下）

3.封闭派生类对象的构造函数执行顺序

在创建派生类的对象时:

（1）先执行基类的构造函数，用以初始化派生类对象中从基类继承的成员；

（2）再执行成员对象类的构造函数，用以初始化派生类对象中成员对象；

（3）最后执行派生类自己的构造函数。

在派生类对象消亡时：

（1）先执行派生类自己的析构函数

（2）再依次执行各成员对象类的析构函数

（3）最后执行基类的析构函数

第五节 公有（public）继承的赋值兼容规则

1.public 继承的赋值兼容规则

```
class base{};
class derived:public base{};//公有派生
```

base b;
derived d;

如上情况下，可以写出以下兼容规则：

（1）派生类的对象可以赋值给基类对象

b =d;//OK

（2）派生类对象可以初始化基类引用

base & br = d;

（3）派生类对象的地址可以赋值给基类指针，但是不**能通过 pb 访问 d 对象中属于 Derived 类而不属于 Base 类的成员**：

base *pb = &d;

这一切都是基于派生类对象就是一个基类对象的原因。

但是，如果派生方式是 private 或 protected，则上述三条不可行。

2.protected 继承和 private 继承

class base{};
class derived:protected base{};//公有派生
base b;
derived d;

protected 继承时，基类的 public 成员和 protected 成员成为派生类的 protected 成员。

private 继承时，基类的 public 成员成为派生类的 private 成员，基类的 protected 成员成为派生类的不可访问成员。

protected 和 private 的继承不是“是”的关系。

3.基类与派生类的指针强制转换

通过强制指针类型转换，可以把 ptrBase 转换成 Derived 类的指针：

Base * ptrBase = &objDerived;
Derived *ptrDerived = (Derived *) ptrBase;

4.直接基类和间接基类

A 派生类 B，类 B 派生类 C，类 C 派生类 D，……

- 类 A 是类 B 的直接基类
- 类 B 是类 C 的直接基类，类 A 是类 C 的间接基类
- 类 C 是类 D 的直接基类，类 A、B 是类 D 的间接基类

在声明派生类时，只需要列出它的直接基类：

（1）派生类沿着类的层次自动向上继承它的间接基类。

（2）派生类的成员包括：

①派生类自己定义的成员；

②直接基类中的所有成员；

③所有间接基类的全部成员。