第五讲 继承（一）

一、继承

二、继承的使用

一、继承

1.1 继承的概述

一个类A可以继承另一个类B，那么我们称类B为基类(父类)，类A为派生类(子类)。

子类从父类继承了所有成员，除了构造函数、析构函数、赋值运算符重载函数。

子类继承父类后，子类的成员分为两部分：1、继承自父类的部分（base part）；2、子类自己扩展的成员（appdent part）。

虽然父类的私有成员被子类继承，但子类依然不能直接访问这些私有成员，子类只能通过继承自父类的公有的成员函数来访问。

子类可以自己实现与父类成员函数原型相同（函数名、参数列表）的成员函数，称为覆盖或重写（overwrite）。

当通过父类对象调用被覆盖的函数时，父类版本的函数被调用，当通过子类对象调用覆盖父类的函数时，子类版本的函数被调用。

在子类中调用被覆盖的父类版本的函数时，在函数名前加Base::，如 Derived d; d.Base::print();。

【例1-1】继承示例

```
/** Base.h **/
#ifndef Base_h
#define Base_h

#include <iostream>
using namespace std;

class Base{
public:
    Base(int i);

    int get_number();

    void print();

private:
    int b_number;
};

#endif
```

```cpp
/** Base.cpp **/
#include "Base.hpp"

Base::Base(int i) : b_number (i) {
}

int Base::get_number(){
    return b_number;
}

void Base::print(){
    cout << b_number<<endl;
}
```

```
/** Derived.h **/
#ifndef Derived_hpp
#define Derived_hpp

#include <iostream>
using namespace std;
#include "Base.hpp"

class Derived:public Base{
public:
    Derived(int i, int j);
    void print();

private:
    int d_number;     //子类扩展的成员变量
    //int b_number;   //子类自动继承自父类的成员变量
};

#endif
```

```
/** Derived.cpp **/
#include "Derived.hpp"

Derived::Derived( int i, int j ): Base(i), d_number(j){
}

void Derived::print( ){ //和父类函数相同(覆盖)
    cout<<get_number()<<endl;
    cout << d_number << endl;

    //不能访问继承自父类的私有成员
    //cout<<b_number<<endl;
}
```

```cpp
/** main.cpp **/
#include "Base.hpp"
#include “Derived.hpp"

int main(int argc, const char * argv[]) {

    Base a(2); //创建父类对象
    Derived b(3, 4); //创建子类对象

    cout << "a is ";
    a.print( );

    cout << "b is ";
    b.print( );

    cout << "base part of b is ";
    b.Base::print( );//调用覆盖函数中父类的函数

    return 0;
}
```

程序运行结果如下:

```
a is 2
b is 3
4
base part of b is 3
```

1.2 protected成员

如果父类里有protected类型的成员，在子类中可以直接访问，不需要借助父类的公有函数；但是，protected类型的成员，对外界依然是隐藏的，对外就像private类型一样。

【例1-2】父类中的protected成员

```
/** Base.h **/
#ifndef Base_h
#define Base_h

#include <iostream>
using namespace std;

class Base{
public:
    Base(int i);
    int get_number();
    void print();

protected:
    int b_number;
};

#endif
```

```cpp
/** Base.cpp **/
#include "Base.hpp"

Base::Base(int i) : b_number (i) {
}

int Base::get_number(){
    return b_number;
}

void Base::print(){
    cout << b_number<<endl;
}
```

```
/** Derived.h **/
#ifndef Derived_hpp
#define Derived_hpp

#include <iostream>
using namespace std;
#include "Base.hpp"

class Derived:public Base{
public:
    Derived(int i, int j);
    void print();

private:
    int d_number;    //子类扩展的成员变量
};

#endif
```

```
/** Derived.cpp **/
#include "Derived.hpp"

Derived::Derived( int i, int j ): Base(i), d_number(j){
}

void Derived::print( ){ //和父类函数相同(覆盖)
    cout << b_number <<endl;
    cout << d_number << endl;
}
```

```
/** main.cpp **/
#include "Base.hpp"
#include "Derived.hpp"

int main(int argc, const char * argv[]){
    Base a(2);
    Derived b(3, 4);

    //a.b_number = 10; //外部不能直接访问protected成员
    b.print();
    return 0;
}
```

程序运行结果如下：

3
4

1.3 继承方式

子类继承父类后，父类中的private成员在子类中是private成员，父类中的protected成员在子类中可以是protected、private，父类中的public成员在子类中可以是public、protected 、private。

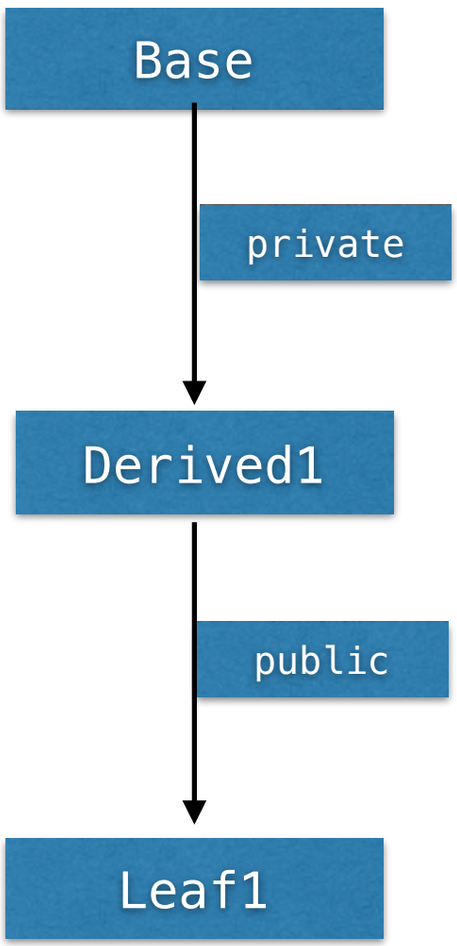
子类中被继承的成员访问级别由三种继承说明符决定:

（1）private；所有被继承的成员在子类中都是private。

（2）protected；所有被继承的public成员在子类中都是protected,所有被继承的protected、private成员在子类中访问级别不变。

（3）public；所有被继承的成员在子类中访问级别不变。

【例1-3】private继承方式



【例1-3】private继承方式示例代码

```
/** Base.h **/
#ifndef Base_h
#define Base_h

#include <iostream>
using namespace std;

class Base{
protected://受保护成员
    int get_priv();
public://公有成员
    Base();
    Base(int a, int b, int c);
    int get_prot( );
    int get_publ( );

private://私有成员
    int priv;
protected://受保护成员
    int prot;
public://公有成员
    int publ;
};
#endif
```

```cpp
/** Base.cpp **/

#include "Base.h"
Base::Base(){

}

Base::Base(int a, int b, int c):priv(a),prot(b),publ(c){

}

int Base::get_priv(){
    return priv;
}

int Base::get_prot(){
    return prot;
}

int Base::get_publ(){
    return publ;
}
```

```
/** Derived1.h **/

#ifndef Derived1_h
#define Derived1_h

#include <iostream>
using namespace std;
#include "Base.hpp"

class Derived1 : private Base{

public:
    Derived1 ();
    Derived1 (int a, int b, int c);

    int get1_priv();
    int get1_prot();
    int get1_publ();
};
#endif
```

```cpp
/** Derived1.cpp **/
#include "Derived1.hpp"

Derived1::Derived1 () : Base() {
}

Derived1::Derived1 (int a, int b, int c) : Base(a, b, c) {
}

int Derived1::get1_priv(){
    return get_priv();
}
int Derived1::get1_prot(){
    return prot;
}
int Derived1::get1_publ(){
    return publ;
}
```

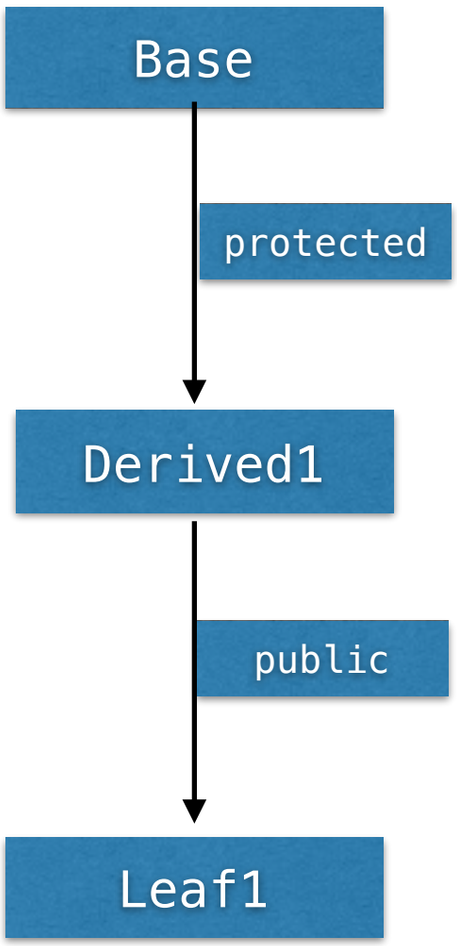
```
/** Leaf1.h **/
#ifndef Leaf1_hpp
#define Leaf1_hpp

#include <iostream>
using namespace std;
#include "Derived1.hpp"

class Leaf1 : public Derived1{
public:
    Leaf1(int a, int b, int c);
    void print( );
};

#endif
```

```
/** Leaf1.cpp **/
#include "Leaf.hpp"

Leaf1::Leaf1(int a, int b, int c) : Derived1(a, b, c) {
}

void Leaf1::print( ){
    cout << "Leaf1 members: " << get1_priv( ) << endl;
    cout<< get1_prot( ) << endl;
    cout<< get1_publ( ) << endl;

    //cout<< get_priv( )<<endl;   //父类中函数为private
    //cout<< get_prot( )<<endl;  //父类中函数为private
    //cout<< publ <<endl;  //父类中publ为private
}
```

```cpp
/** main.cpp **/
#include "Base.hpp"
#include "Derived1.hpp"
#include "Leaf1.cpp"
int main(int argc, const char * argv[]){

    Derived1 d1(1,2,3);
    //cout<<d1.publ<<endl; //私有成员不能直接访问
    //cout<<d1.get_priv()<<endl; //私有成员不能直接访问
    cout<<d1.get1_priv()<<endl;
    cout<<d1.get1_prot()<<endl;
    cout<<d1.get1_publ()<<endl;

    Leaf1 lf1(1, 2, 3);
    //cout << lf1.publ << endl; //私有成员不能直接访问
    lf1.print();

    return 0;
}
```

```
程序运行结果如下:
    1
    2
    3
    Leaf1 members: 1
    2
    3
```

【例1-4】protected继承方式



【例1-4】protected继承方式示例代码

```
/** Derived2.h **/

#ifndef Derived2_h
#define Derived2_h

#include <iostream>
using namespace std;
#include "Base.hpp"

class Derived2 : protected Base{

public:
    Derived2 (int a, int b, int c) : Base(a, b, c) {
    }
};
```

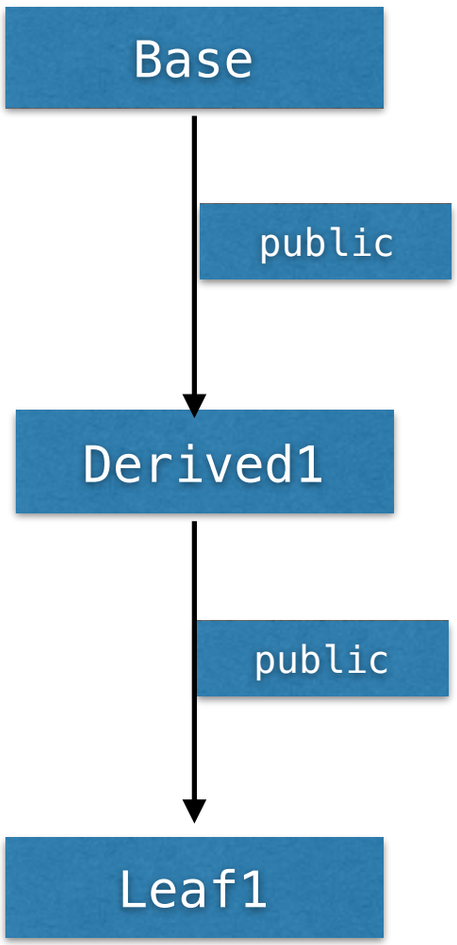
```
/** Leaf2.h **/
#ifndef Leaf2_hpp
#define Leaf2_hpp

#include <iostream>
using namespace std;
#include "Derived2.hpp"

class Leaf2 : public Derived2{
public:
    Leaf2(int a,int b,int c):Derived2(a,b,c){

    }

    void print(){
        cout<<get_priv()<<endl;
        cout<<"Leaf2 members: "<<get_priv()<<" "<<prot<<"
"<<publ<<endl;
    }
};
```

```
/** main.cpp **/
#include "Base.hpp"
#include "Derived2.hpp"
#include “Leaf2.cpp"

int main(int argc, const char * argv[]){

    Derived2 d2(4, 5, 6);
    //cout << d2.publ;          //受保护
    //cout << d2.get_priv( );  //受保护


    Leaf2 lf2(4, 5, 6);
    //cout << lf2.publ << endl; //受保护
    lf2.print();

    return 0;
}
```

程序运行结果如下：

```
    4
  Leaf2 members: 4 5 6
```

【例1-5】public继承方式



【例1-5】public继承方式示例代码

```
/** Derived3.h **/

#ifndef Derived3_h
#define Derived3_h

#include <iostream>
using namespace std;
#include "Base.hpp"

class Derived3 : public Base  {
public:
    Derived3 (int a, int b, int c) : Base(a, b, c) {
    }
};
```

```
/** Leaf3.h **/
#ifndef Leaf3_hpp
#define Leaf3_hpp

#include <iostream>
using namespace std;
#include "Derived3.hpp"

class Leaf3 : public Derived3
{
public:
    Leaf3(int a, int b, int c) : Derived3(a, b, c) { }

    void print( ){
        cout << "Leaf3 members: " << get_priv( ) << " "
        //<< priv<< " "
        << prot << " "
        << publ << endl;
    }
};
```

```
/** main.cpp **/
#include "Base.hpp"
#include "Derived3.hpp"
#include “Leaf3.cpp"

int main(int argc, const char * argv[]){

    Derived3 d3(7, 8, 9);
    cout << d3.publ<<endl;
    cout << d3.get_prot()<<endl;

    Leaf3 lf3(7, 8, 9);
    cout << lf3.publ << endl;
    return 0;
}
```

程序运行结果如下:

9
8
9

二、继承的使用

继承主要是在以下情况下使用：

（1）两个类之间有自然的继承关系，即一个类是另一类的特例，如graduatestudent is a student，student is a person，等等。

（2）实现代码重用（代码复用）。一个类可能需要使用其他类中定义的成员，此时可以定义一个派生类继承自该类， 这样我们就不必重复编写代码。

类之间最主要的两种交互关系：

（1）组合:一个类是另外一个类的成员变量，一个类拥有另一个类,是 Has -A ，有一个的关系。

eg: 自行车有一个轮子。

（2） 继承:一个类是另一个类的特例，IS-A,是一个的关系。

eg:一个学生是一个人。

【例2-1】组合关系使用继承实现

```
/** Point.h **/
#ifndef Point_hpp
#define Point_hpp

#include <iostream>
using namespace std;

class Point{
    friend ostream & operator<<(ostream &,Point &);

public:
    Point ();
    Point (double xval, double yval);

protected:
    double  x;
    double  y;
};

#endif
```

```
/** Point.cpp**/
#include "Point.hpp"

ostream & operator << (ostream & os, Point &  apoint){
    os <<"Point:X:Y: " << apoint.x << "," << apoint.y<< "\n";
    return os;
}

Point::Point(){
    x = 0;
    y = 0;
}

Point::Point(double xval, double yval){//构造函数
    x = xval;
    y = yval;
}
```

```
/** Circle.h**/
#ifndef Circle_hpp
#define Circle_hpp

#include <iostream>
using namespace std;
#include "Point.hpp"

class Circle : public Point{ //继承
    friend ostream & operator<<(ostream &,Circle&);

public:
    Circle ();

    Circle (double r,double xval,double yval);

    double area();

protected:
    double radius;
};
```

```
/** Circle.cpp**/
#include "Circle.hpp"

ostream & operator <<(ostream & os, Circle & aCircle){
    os<< "Circle:radius:" << aCircle.radius; //是Circle的成员
    os<< aCircle.x << "\n"; //x, y是Circle从Point继承而来成员。
    os<< aCircle.y << "\n";
    return os;
}

Circle::Circle (): Point(), radius(0){
}

Circle::Circle (double r,double xval,double yval):
Point(xval,yval), radius(r){
}

double Circle::area(){
    return (3.14159* radius *radius);
}
```

```
/** Cylinder.h**/
#ifndef Cylinder_hpp
#define Cylinder_hpp

#include <iostream>
using namespace std;
#include "Circle.hpp"

class  Cylinder : public Circle{
    friend ostream & operator << (ostream & ,Cylinder &);

public:
    Cylinder ();

    Cylinder (double hv, double rv, double xv, double yv);

    double area();

protected:
    double height;
};
```

```
/** Cylinder.cpp**/
#include "Cylinder.hpp"

ostream & operator << (ostream & os,Cylinder & acylinder){
    os << "cylinder dimensions: ";
    os << "x: " <<acylinder.x;
    os << "  y: " <<acylinder.y ;
    os << "  radius: " <<acylinder.radius ;
    os << "  height: " <<acylinder.height
    << endl;
    return os;
}

Cylinder::Cylinder():Circle(){
    height = 0;
}

Cylinder::Cylinder (double hv,double rv, double xv,double
yv):Circle( xv,yv,rv){
    height = hv;
}

double Cylinder :: area ( ){
    return  2.0* Circle::area() + 2.0*3.14159* radius*height;
}
```

```
/** main.cpp **/
#include "Point.hpp"
#include "Circle.hpp"
#include “Cylinder.cpp”

int main(int argc, const char * argv[]){
    Point p(2,3);
    Circle c(7,6,5);
    Cylinder cyl(10,11,12,13);

    cout << p;
    cout << c;
    cout << “area circle:" << c.area() << endl;

    cout<< cyl;
    cout<<"area cylinder:"<< cyl.area()<<endl ;

    cout<<"area cylinder base is "
    << cyl.Circle::area() << endl;

    return 0;
}
```

程序运行结果如下：

```
    Point:X:Y: 2,3
    Circle:radius:76 5
    area cirle:153.938
    cylinder x:13 y:11 radius:12 height:10
    area cylinder:1658.76
    area cylinder base is 452.389
```