c++ 覆盖、重载、隐藏

“overload”翻译过来就是：超载，过载，重载，超出标准负荷；“override”翻译过来是：重置，覆盖,使原来的失去效果。

先来说说重载（Overload）的含义，在日常生活中我们经常要清洗一些东西，比如洗车、洗衣服。尽管我们说话的时候并没有明确地说用洗车的方式来洗车，或者用洗衣服 的方式来洗一件衣服，但是谁也不会用洗衣服的方式来洗一辆车，否则等洗完时车早就散架了。我们并不要那么明确地指出来就心知肚明，这就有重载的意思了。在 同一可访问区内被声名的几个具有不同参数列的（参数的类型、个数、顺序不同）同名函数，程序会根据不同的参数列来确定具体调用哪个函数，这种机制叫重载， 重载不关心函数的返回值类型。这里，“重载”的“重”的意思不同于“轻重”的“重”，它是“重复”、“重叠”的意思。例如在同一可访问区内有：

① double calculate(double);

② double calculate(double，double);

③ double calculate(double, int);

④ double calculate(int, double);

⑤ double calculate(int);

⑥ float calculate(float);

⑦ float calculate(double);

六个同名函数calculate，①②③④⑤⑥中任两个均构成重载，⑥和⑦也能构成重载，而①和⑦却不能构成重载，因为①和⑦的参数相同。

覆盖(Override)是指派生类中存在重新定义的函数，其函数名、参数列、返回值类型必须同父类中的相对应被覆盖的函数严格一致，覆盖函数和被覆盖函数只有函数体 （花括号中的部分）不同，当派生类对象调用子类中该同名函数时会自动调用子类中的覆盖版本，而不是父类中的被覆盖函数版本，这种机制就叫做覆盖。

下面我们从成员函数的角度来讲述重载和覆盖的区别。

成员函数被重载的特征有：

1) 相同的范围（在同一个类中）；

2) 函数名字相同；

3) 参数不同；

4) virtual关键字可有可无。

覆盖的特征有：

1) 不同的范围（分别位于派生类与基类）；

2) 函数名字相同；

3) 参数相同；

4) 基类函数必须有virtual关键字。

比如，在下面的程序中：

#include <iostream.h>

class Base

{

public:

void f(int x){ cout << "Base::f(int) " << x << endl; }

void f(float x){ cout << "Base::f(float) " << x << endl; }

virtual void g(void){ cout << "Base::g(void)" << endl;}

};

class Derived : public Base

{

public:

virtual void g(void){ cout << "Derived::g(void)" << endl;}

};

void main(void)

{

Derived d;

Base \*pb = &d;

pb->f(42); // 运行结果: Base::f(int) 42

pb->f(3.14f); // 运行结果: Base::f(float) 3.14

pb->g(); // 运行结果: Derived::g(void)

}

函数Base::f(int)与Base::f(float)相互重载，而Base::g(void)被Derived::g(void)覆盖。

隐藏是指派生类的函数屏蔽了与其同名的基类函数，规则如下：

1) 如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。

2) 如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）。

比如，在下面的程序中：

#include <iostream.h>

class Base

{

public:

virtual void f(float x){ cout << "Base::f(float) " << x << endl; }

void g(float x){ cout << "Base::g(float) " << x << endl; }

void h(float x){ cout << "Base::h(float) " << x << endl; }

};

class Derived : public Base

{

public:

virtual void f(float x){ cout << "Derived::f(float) " << x << endl; }//被继承之后,virtual 可有可无，但最好有。继承后，还是虚函数。

void g(int x){ cout << "Derived::g(int) " << x << endl; }

void h(float x){ cout << "Derived::h(float) " << x << endl; }

using Base::g;//这句话是用来引用父类中被隐藏的部分的。

};

通过分析可得：

1) 函数Derived::f(float)覆盖了Base::f(float)。

2) 函数Derived::g(int)隐藏了Base::g(float)，注意，不是重载。

3) 函数Derived::h(float)隐藏了Base::h(float)，而不是覆盖。

看完前面的示例，可能大家还没明白隐藏与覆盖到底有什么区别，因为我们前面都是讲的表面现象，怎样的实现方式，属于什么情况。下面我们就要分析覆盖与隐藏在应用中到底有什么不同之处。在下面的程序中bp和dp指向同一地址，按理说运行结果应该是相同的，可事实并非如此。

void main(void)

{

Derived d;

Base \*pb = &d;

Derived \*pd = &d;

// Good: behavior depends solely on type of the object

pb->f(3.14f); //运行结果: Derived::f(float) 3.14

pd->f(3.14f); //运行结果: Derived::f(float) 3.14

// Bad : behavior depends on type of the pointer

pb->g(3.14f); //运行结果: Base::g(float) 3.14

pd->g(3.14f); //运行结果: Derived::g(int) 3

// Bad : behavior depends on type of the pointer

pb->h(3.14f); //运行结果: Base::h(float) 3.14

pd->h(3.14f); //运行结果: Derived::h(float) 3.14

}

请大家注意，f()函数属于覆盖，而g()与h()属于隐藏。从上面的运行结果，我们可以注意到在覆盖中，用基类指针和派生类指针调用函数f() 时，系统都是执行的派生类函数f()，而非基类的f()，这样实际上就是完成的“接口”功能。而在隐藏方式中，用基类指针和派生类指针调用函数f()时， 系统会进行区分，基类指针调用时，系统执行基类的f()，而派生类指针调用时，系统“隐藏”了基类的f()，执行派生类的f()，这也就是“隐藏”的由 来。

===================================================================

-------------------------------------------------------------------

-------------------------------------------------------------------

重载(overload)

这个好理解,在同个space域同名的.

参数必须不同,有关virtual无关.

覆盖(override)

同名字,同参数,有virtual

覆盖好理解比如show()函数

A派生了B

如果B中的show()覆盖了A中的show()

但B中仍然有两个show(),而不管是A类指针也好,B类对象调用也好,都只能调用B类自己的那个show();

而从A类继承过来的show()函数真的就被覆盖了,没有了吗? 答应是不对的.这时可以在B类对象显示的调用A类继承过来的show();

程序代码：

#include <iostream>

using namespace std;

class A

{

public:

virtual void show()

{

cout << a << endl;

}

int a;

};

class B:public A

{

public:

void show()

{

A::show();

//显式地调用自己类中的 "由A类继承过来的show()函数"

//像这种直接显式指出某个类的某个函数时,

//编译器处理方式是这样的: 首先在自己类中找有没有A::show(),如果找到,调用.不在继续在A类中找

//如果找不到,则在显式指出的那个类中(即A类)调用那个函数. 这里当然是在B类中可以找到A::show()

//因为基类中指出了这个函数是virtual函数.

}

int b;

};

int main()

{

A a;

a.a = 3;

a.show();

B b;

b.b = 4;

b.show();

b.A::show(); //显示的调用自己类中的 "由A类继承过来的show()函数"

return 0;

}

总结:

virtual, 覆盖.

其实准确,通俗的讲B类还是有两个show(),只是调用由A继承过来的show()只能通过显式的调用方法 [类名::virtual函数名]

而不管是基类A的指针 (B b; A \*p = &b; p->show())或者派生类的对象(B b; b.show()),都只能调用B类的自己本身存在的show()函数

-----------------------------------------------------------------------------------------------------------------------

隐藏hide

这个有点麻烦了...

什么是隐藏情况?

1,同名同参无virtual

2,同名不同参不管有无virtual

程序代码：

class A

{

public:

void show() {}; //编号1

void rose(int a) {} //编号2

};

class B:public A

{

public:

void show() {}; //编号3

void rose(int a, int b) {}; //编号4

};

//

类B中的show()和rose()明显是隐藏了类A的show()和rose()

隐藏的理解:

B类中其实有两个show(),两个rose();

但为什么不叫重载呢?你会这样想,但我可以告诉你,因为类B中的两个show(),两个rose(),不是都可以被类B对象调用的.

//----1----//

编号1和编号2,在类B中哪怕存在,但只能通过类A的指针调用,而不能通过B类对象调用,如:

程序代码：

A \*p = new B;

p->show();

p->rose(3);

p->rose(3,5); //error

//----2----//

编号3和编程4,只能通过类B对象调用,而不能通过类A的指针调用,如:

程序代码：

B b;

b.show();

b.rose(3,5);

b.rose(4); //error