C++中通过基类指针调用派生类中定义的方法&&C++继承中的名称遮掩_selfsongs的博客-CSDN博客

成就一亿技术人!

C++中通过基类指针调用派生类中定义的方法&&C++继承中的名称遮掩

动态绑定

动态类型指“目前所指对象的类型”;
动态绑定是指在执行期间（非编译期）判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

```
#include <iostream>
using namespace std;
class BaseA
{
public:
    virtual void display(){
        cout << "BaseA::display()" << endl;
    }
    virtual void display(int a){
        cout<<"BaseA::display(a)" <<endl;
    }
};
class DerivedA : public BaseA
{
public:
   virtual void display(){
        cout << "DerivedA::display()" << endl;
    }
};

int main()
{
    DerivedA t;

    BaseA* pt1=&t;
    pt1->display();

    DerivedA* pt2=&t;
    pt2->display();
    //pt2->display(2);
    //错误，DerivedA中的函数display()在“名称遮掩规则”下遮掩了BasedA中的函数display(int a)
    system("pause");
    return 0;
}
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18

结果：

```
DerivedA::display()
DerivedA::display()
请按任意键继续. . .
```

virtual函数是动态绑定，动态绑定时通过指针调用的是该指针指向的对象中的成员函数。

virtual函数是动态绑定，而成员函数缺省参数却是静态绑定

静态绑定

静态类型指在程序声明时所采取的类型，
静态绑定是指在程序编译过程中，把函数（方法或者过程）调用与响应调用所需的代码结合的过程称之为静态绑定。

```
#include <iostream>
using namespace std;
class BaseA
{
public:
    void display(){
        cout << "BaseA::display()" << endl;
    }
    void display(int a){
        cout<<"BaseA::display(a)" <<endl;
    }
};
class DerivedA : public BaseA
{
public:
   void display(){
        cout << "DerivedA::display()" << endl;
    }
};

int main()
{
    DerivedA t;

    BaseA* pt1=&t;
    pt1->display();//BasedA::display()

    DerivedA* pt2=&t;
    pt2->display();//DerivedA::display()
    //pt2->display(2);
    //错误，DerivedA中的函数display()在“名称遮掩规则”下遮掩了BasedA中的函数display(int a)
    system("pause");
    return 0;
}
```

9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

结果：

这个示例只是为了说明静态绑定时，通过指针调用的是该指针声明时的类型对应的类对象中的成员函数。

但是必须强调一下：**不要在派生类中重新定义基类的非虚函数**

从面向对象的角度来说，公有继承是一种Is-a的关系。其表明每一个派生类对象都可以被当做基类对象来处理。基类的每一个接口和成员变量派生类也有。如果重定义了非虚函数，会导致派生类对象将不再是基类对象（当通过指向派生类对象的指针访问时）

很少情况下，需要重定义非虚函数。一个特例是为了解决私有继承中的名称遮掩问题。在私有继承中，基类的公有函数在派生类中都是私有，如果派生类想要继承基类的某个接口，可以使用所谓的转交函数（forwarding function）。即定义一个public的与基类中那个接口同名的函数，这个函数一般是inline,函数体实现部分仅调用基类的对应函数即可。

关于C++继承中的名称遮掩问题，下面细讲一下：

C++继承中的名称遮掩

示例：

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

class BaseA
{
public:
        void display() {
                cout << "BaseA::display()" << endl;
        }
        void display(int a) {
                cout << "BaseA::display(a)" << endl;
        }
};
class DerivedA : public BaseA
{
public:
        void display() {
                cout << "DerivedA::display()" << endl;
        }
};

int main()
{
        DerivedA t;

        BaseA* pt1 = &t;
        pt1->display();//BasedA::display()
        pt1->display(2);//静态绑定，所以还能通过基类指针去调用派生类中的display(int a)函数
```

```
        DerivedA* pt2 = &t;
        pt2->display();//DerivedA::display()
        //pt2->display(2);
        //错误，DerivedA中的函数display()在“名称遮掩规则”下遮掩了BasedA中的函数display(int a)
        system("pause");
        return 0;
}
```

结果：

```
BaseA::display()
BaseA::display(a)
DerivedA::display()
请按任意键继续. . .
```

上面的代码展示了，DerivedA类“没有能够成功继承”BaseA中的display(int a)函数。打上引号的原因是，我们使用基类指针指向派生类时，又能够成功调用。

从名称查找的观点出发，BaseA::display(int a)被DerivedA::display()函数遮蔽了。

编译器拒绝这样的继承是为了防止在程序库或应用框架内建立新的derived class 时附带地从疏远的base classes 继承重载函数。

如果你想从基类BasedA中继承重载了的display()和display(int a)函数，同时又想在派生类DerivedA中重新定义display()函数时不遮掩display(int a)函数名，

可以通过在派生类DerivedA中用using声明基类BasedA中的函数名：

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

class BaseA
{
public:
        void display() {
                cout << "BaseA::display()" << endl;
```

```
        }
        void display(int a) {
                cout << "BaseA::display(a)" << endl;
        }
};
class DerivedA : public BaseA
{
public:
        using BaseA::display;//让BaseA class内名为dislay的所有东西在DerivedA中都可见(并且public)

        void display() {
                cout << "DerivedA::display()" << endl;
        }
};

int main()
{
        DerivedA t;

        BaseA* pt1 = &t;
        pt1->display();//BasedA::display()
        pt1->display(2);//静态绑定，所以还能通过基类指针去调用派生类中的display(int a)函数

        DerivedA* pt2 = &t;
        pt2->display();//DerivedA::display()
        pt2->display(2);//正确， DerivedA类中使用using BaseA::display，避免了名称遮掩
        system("pause");
        return 0;
}
```

36

37

结果：

```
BaseA::display()
BaseA::display(a)
DerivedA::display()
BaseA::display(a)
请按任意键继续. . .
```

总结：**derived classes 内的名称会遮掩base classes 内的名称；如果继承base class并加上重载函数，而又希望在derive中重新定义或覆写其中一部分，那必须手动使用using 声明那些覆写的函数，否则某些希望继承的名称会被遮掩。**

但有时候，比如在private继承时(表示的是一种根据基类实现派生类的关系，is-implemented-in-terms-of)，基类的公有函数在派生类中都是私有。如果派生类不想继承base class的所有函数接口。而只想要继承基类的某一个接口，可以使用所谓的转交函数（forwarding function）。

以上面的例子为例，DerviedA只想继承BaseA::display()函数，

```
#include <iostream>
using namespace std;
class BaseA
{
public:
        virtual void display() {
                cout << "BaseA::display()" << endl;
        }
        virtual void display(int a) {
                cout << "BaseA::display(a)" << endl;
        }
};
class DerivedA : private BaseA
{
public:
        //using BaseA::display;//让BaseA class内名为dislay的所有东西在DerivedA中都可见(并且public)

        virtual void display() {
                BaseA::display();
        }
};

int main()
{
        DerivedA t;
        t.display();
        //t.display(2);//错误
        system("pause");
        return 0;
}
```

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26
27
28
29
30

结果：

```
BaseA::display()
请按任意键继续. . .
```

dynamic_cast<>动态转型的作用

dynamic_cast:将指向base class objects的pointers或references转型为指向derived(或sibling base)clasee objects的pointers或references，

```
#include <iostream>
using namespace std;
class BaseA {
public:
        virtual void display() = 0;
};

class DerivedA : public BaseA {
public:
        virtual void display() {
                cout << "DerivedA::display()" << endl;
        }
        void displaymore() {
                cout << "DerivedA::displaymore()" << endl;
        }
};

int main() {
        BaseA *pt = new DerivedA;
        pt->display();
        //pt->displaymore();//error: ‘class BaseA’ has no member named ‘display()’
        dynamic_cast<DerivedA*>(pt)->displaymore();
        system("pause");
        return 0;
}
```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

结果：

```
DerivedA::display()
DerivedA::displaymore()
请按任意键继续. . .
```

区分接口继承和实现继承

在设计类的继承关系时，

有时候你希望derived classes只继承成员函数的接口（即声明）；

有时候你希望derived classes同时继承成员函数的接口和实现，但又希望能够覆写它们所继承的实现；

有时候你希望derived classes同时继承成员函数的接口和实现，并且不允许覆写任何东西；

声明一个pure virtual 函数的目的是为了让derived classes只继承成员函数的接口

声明impure virtual函数(非纯虚函数）是为了让derived classes同时继承成员函数的接口和缺省实现

声明non-virtual 函数的目的是为了让derived classes同时继承成员函数的接口和一份强制性实现；