**多重继承**

一个类可以**继承于多个父类**

子类拥有所有父类的成员**变量和函数**

子类对象可以当做任意父类对象使用

例如:

class Derived : public BaseA,

public BaseB,

public BaseC

{

//... ...

}

**多重继承的问题1**

当**多个不同的父类指针**指向**同一个多重继承的子类**时,可能**拥有不同地址**

比如:

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

class BaseA

{

int ma;

public:

BaseA(int a)

{

ma = a;

}

int getA()

{

return ma;

}

};

class BaseB

{

int mb;

public:

BaseB(int b)

{

mb = b;

}

int getB()

{

return mb;

}

};

class Derived : public BaseA, public BaseB

{

int mc;

public:

Derived(int a, int b, int c) : BaseA(a), BaseB(b)

{

mc = c;

}

};

int main()

{

Derived d(1, 2, 3);

BaseA\* pa = &d;

BaseB\* pb = &d;

if((void \*)pa==(void \*)pb)

{

cout<<"true"<<endl;

}

else

{  
 cout<<"false"<<endl;

}

cout << "&d= " << &d << endl;

cout << "pa= " << pa << endl;

cout << "pb= " << pb << endl;

}

[复制代码](javascript:void(0);)

运行打印:

false

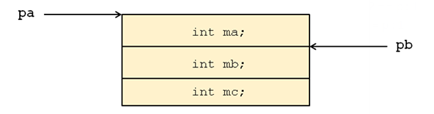
&d= 0x28fefc

pa= 0x28fefc

pb= 0x28ff00

**为什么,pa指针和pb指针都指向d对象,它们的地址却有所不同？**

这是因为**Derived d对象**地址里依次存了两个不同的父类成员变量值,如下图所示:



从上图看到,其实pa和pb还是位于d对象地址里,只是指向的位置不同而已.所以在多重继承里,最好**不要使用等号**直接判断**两个指针对象是否相等**.

**多重继承的问题2**

多重继承可能**产生冗余的成员**

**比如:**

**老师teacher类**,**学生student类**都继承于**people类**

而**有些老师**,为了工作还要考博士学位,**既是老师又是学生**,所以**同时继承于**老师teacher类,学生student类,则**该类的成员**便会**拥有两个people类成员**,从而产生冗余

**在工程中,如何正确使用多重继承**

* 只继承**一个父类**和**多个接口**
* 由于接口只有存虚函数,从而**避免了冗余的成员**
* 在父类中**提供equal()**成员函数,
* 通过**equal()**成员函数来判断指针是否指向当前对象,使用**dynamic\_cast**强制转换

例如:

[复制代码](javascript:void(0);)

#include <iostream>

using namespace std;

class Base

{

protected:

int mi;

public:

Base(int i)

{

mi = i;

}

int getI()

{

return mi;

}

bool equal(Base\* obj)

{

return (this == obj);

}

};

class Interface1

{

public:

virtual void add(int i) = 0;

virtual void minus(int i) = 0;

};

class Interface2

{

public:

virtual void multiply(int i) = 0;

virtual void divide(int i) = 0;

};

class Derived : public Base, public Interface1, public Interface2

{

public:

Derived(int i) : Base(i)

{

}

void add(int i)

{

mi += i;

}

void minus(int i)

{

mi -= i;

}

void multiply(int i)

{

mi \*= i;

}

void divide(int i)

{

if( i != 0 )

{

mi /= i;

}

}

};

int main()

{

Derived d(100);

Derived\* p = &d;

Interface1\* pInt1 = &d;

Interface2\* pInt2 = &d;

cout << "p->getI() = " << p->getI() << endl; // 100

pInt1->add(10);

pInt2->divide(11);

pInt1->minus(5);

pInt2->multiply(8);

cout << "p->getI() = " << p->getI() << endl; // 40

cout << "pInt1 == p : " << p->equal(dynamic\_cast<Base\*>(pInt1)) << endl;

cout << "pInt2 == p : " << p->equal(dynamic\_cast<Base\*>(pInt2)) << endl;

cout << "&d == p : " << p->equal(dynamic\_cast<Base\*>(&d)) << endl;

return 0;

}

[复制代码](javascript:void(0);)

运行打印:

p->getI() = 100

p->getI() = 40

pInt1 == p : 1

pInt2 == p : 1

&d== p : 1

可以发现,使用**dynamic\_cast转换**,判断出来的**地址就是相等的.**

以**p->equal(dynamic\_cast<Base\*>(pInt1))**为例,我们编译时,编译器就会去检查**pInt1**所在的地址,然后找到是**d对象**,通过d对象找到**Base父类**,从而去修正**pInt1指针**的地址.