# C++\_多态\_03\_虚析构函数

## 析构函数的重写

如果基类的析构函数为虚函数，此时派生类析构函数只要定义，无论是否加virtual关键字，都与基类的析构函数构成重写，虽然基类与派生类析构函数名字不同。虽然函数名不相同，看起来违背了重写的规则，其实不然，这里可以理解为编译器对析构函数的名称做了特殊处理，编译后析构函数的名称统一处理成destructor。

class Base

{

public:

virtual ~Base()

{ // 虚析构函数

cout << "Base destructor called!" << endl;

}

};

class Derived : public Base

{

public:

virtual ~Derived()

{

cout << "Derived destructor called!" << endl;

}

};

int main()

{

Base\* b = new Derived(); // 基类指针指向派生类对象

delete b; // 调用 Derived 的析构函数，然后调用 Base 的析构函数

return EXIT\_SUCCESS;

}

运行结果

Derived destructor called!

Base destructor called!

## 非虚析构函数

继承体系中，子类对象指向析构时候，本身就会现执行子类再执行父类

class Base

{

public:

~Base()

{ // 虚析构函数

cout << "Base destructor called!" << endl;

}

};

class Derived : public Base

{

public:

~Derived()

{

cout << "Derived destructor called!" << endl;

}

};

int main()

{

printf("--------------------begain-------------------\n");

Base\* a = new Derived(); // 基类指针指向派生类对象

Derived\* b = new Derived();

delete a; // 只调用 Base 的析构函数，无法调用子类虚构函数

delete b; // 调用 Derived 的析构函数，然后调用 Base 的析构函数

printf("--------------------end----------------------\n");

return EXIT\_SUCCESS;

}

运行结果

--------------------begain-------------------

Base destructor called!

Derived destructor called!

Base destructor called!

--------------------end----------------------

## 总结

所以说析构函数是否需要写成虚析构函数，首先得发生多态行为，才需要父类析构写成虚析构，这也才可以保证delete 父类指针时候，能够去调用子类的析构函数；

本质我们delete父类指针，就是为了析构子类对象的，因为父类指针指向了子类对象。为了析构子类对象，那么就需要多态，这个行为才会产生；需要多态就必须子类重写父类的虚析构函数；

以上就是今天要讲的内容，后续会有更多内容。

## 参考资料

版权声明：本文参考了其他资料和CSDN博主的文章，遵循CC 4.0 BY-SA版权协议，现附上原文出处链接及本声明。

1. https://blog.csdn.net/m0\_46606290/article/details/124017703