公告

昵称：nkxyf
园龄：3年2个月
粉丝：4
关注：154
+加关注



< 2015年2月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 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 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

CentOS(1)
mysql(1)
MySQL UBUNTU(1)
Swing Eclipse 插件(1)
TinyMce(1)
工具操作(1)

随笔分类

C#(3)
C++学习笔记(15)
JAVA(16)
Javascript JQuery CSS Html(2)
Liux(1)
MFC学习笔记(1)
Python(3)
从零开始-数据结构
数据库(2)
算法(2)
习题练习(2)
转载(45)

随笔档案

2014年9月 (2)
2014年7月 (2)
2014年1月 (5)
2013年9月 (1)
2013年6月 (3)
2013年5月 (2)
2013年4月 (7)
2013年3月 (6)
2013年2月 (3)

# C++在单继承、多继承、虚继承时，构造函数、复制构造函数、赋值操作符、析构函数的执行顺序和执行内容 【转】 参考度4.6星

源地址：http://blog.csdn.net/daheiantian/article/details/6438782

---

## 一、本文目的与说明

1. 本文目的：理清在各种继承时，构造函数、复制构造函数、赋值操作符、析构函数的执行顺序和执行内容。

2. 说明：虽然复制构造函数属于构造函数的一种，有共同的地方，但是也具有一定的特殊性，所以在总结它的性质时将它单独列出来了。

3. 单继承、多继承、虚继承，既然都属于继承，那么虽然有一定的区别，但还是相同点比较多。如果放在一块讲，但为了将内容制作成递进的，就分开了，对相同点进行重复，（大量的复制粘贴哈），但在不同点进行了标注。

注意：三块内容是逐步递进的

如果你懂虚函数，那么单继承和多继承那块你就可以不看；

如果你懂多继承，那单继承你就不要看了，至于虚继承就等你懂虚继承再回来看吧；

如果你只懂单继承，那你就只看单继承就好。

## 二、基本知识

1. 对于一个空类，例如：

```
class EmptyClass{};
```

虽然你没有声明任何函数，但是编译器会自动为你提供上面这四个方法。

```
class EmptyClass {
public:
    EmptyClass();                           //  默认构造函数
    EmptyClass(const EmptyClass &rhs);    //  复制构造函数
    ~EmptyClass();                          // 析构函数
    EmptyClass& operator=(const EmptyClass &rhs);    //  赋值运算符
}
```

对于这四个方法的任何一个，你的类如果没有声明，那么编译器就会自动为你对应的提供一个默认的。（在《C++ primer》中，这个编译器自动提供的版本叫做“合成的***”，例如合成的复制构造函数）当然如果你显式声明了，编译器就不会再提供相应的方法。

2. 合成的默认构造函数执行内容：如果有父类，就先调用父类的默认构造函数。

2. 合成的复制构造函数执行内容：使用参数中的对象，构造出一个新的对象。

3. 合成的赋值操作符执行内容：使用参数中的对象，使用参数对象的非static成员 依次对 目标对象的成员赋值。注意：在赋值操作符执行之前，目标对象已经存在。

2012年12月 (2)
2012年11月 (5)
2012年10月 (1)
2012年9月 (1)
2012年8月 (1)
2012年7月 (15)
2012年6月 (12)
2012年5月 (16)
2012年4月 (5)
2012年3月 (6)
2012年2月 (1)

最新评论
1. Re:TinyMce 使用初探
@nkxyf谢了！...
--Zempty
2. Re:TinyMce 使用初探
@ZemptyTinyMce 官网API 搜索该方法...
--nkxyf
3. Re:TinyMce 使用初探
@nkxyf先谢谢，我去试下。我想知道这个方法是在官方哪里找到的？我想自己看文档...
--Zempty
4. Re:TinyMce 使用初探
@Zemptyvar tmp= tinymce.get('elm1').getContent();...
--nkxyf
5. Re:TinyMce 使用初探
楼主知道用js怎么获取tinymce所选textarea(以下直接称textarea)的内容吗？是这样的：我想在内容提交之前通过js判断textarea的内容是否为空再决定是否提交，但是我看了官方文档.....
--Zempty

阅读排行榜
1. linux下c++开发主要是做什么？(4322)
2. adobe reader添加pdf书签功能(2212)
3. c++ 如何定义未知元素个数的数组？【转】(2173)
4. C++-inserter(1914)
5. C++笔试题 String类的实现 三大复制控制函数(1656)

评论排行榜
1. TinyMce 使用初探(5)
2. adobe reader添加pdf书签功能(3)

推荐排行榜
1. C++-inserter(1)
2. 快排 【转】(1)

4. 在继承体系中，要将基类（或称为父类）的析构函数，声明为**virtual**方法（即虚函数）。

5. 子类中包含父类的成员。即子类有两个部分组成，父类部分和子类自己定义的部分。

6. 如果在子类中显式调用父类的构造函数，只能在构造函数的初始化列表中调用，并且只能调用其直接父类的。

7. 在多重继承时，按照基类继承列表中声明的顺序初始化父类。

8. 在虚继承中，虚基类的初始化 早于 非虚基类，并且子类来初始化虚基类（注意：虚基类不一定是子类的直接父类）。

## 三、单继承

核心：在构造子类之前一定要执行父类的一个构造函数。

1.构造函数（不包括复制构造函数）。

顺序：①直接父类；②自己

注意：若直接父类还有父类，那么“直接父类的父类”会在“直接父类”之前 构造。 可以理解为这是一个递归的过程，知道出现一个没有父类的类才停止。

2.1 如果没有显式定义构造函数，则“合成的默认构造函数”会自动调用直接父类的“默认构造函数”，然后调用编译器为自己自动生成的“合成的默认构造函数”。

2.2 如果显式定义了自己的构造函数

2.2.1 如果没有显式调用直接父类的任意一个构造函数，那么和“合成的默认构造函数”一样，会先自动调用直接父类的 默认构造函数，然后调用自己的构造函数。

2.2.2 如果显式调用了直接父类的任意一个构造函数，那么会先调用直接父类相应的构造函数，然后调用自己的构造函数。

2. 复制构造函数

顺序：①直接父类；②自己

注意：和构造函数一样，若直接父类还有父类，那么“直接父类的父类”会在“直接父类”之前 构造。 可以理解为这是一个递归的过程，知道出现一个没有父类的类才停止。

2.1 如果 没有显式定义复制构造函数，则“合成的复制构造函数”会自动调用直接父类的“复制构造函数”，然后调用编译器为自己自动生成的“合成的复制构造函数”（注意：不是默认构造函数）

2.2 如果显式定义了自己的复制构造函数 （和构造函数类似）

2.2.1 如果没有显式调用父类的任意一个构造函数，那么会先调用直接父类的 默认构造函数（注意：不是 复制构造函数）。

2.2.2 如果显式调用了直接父类的任意一个构造函数，那么会先调用直接父类相应的构造函数。

3.赋值操作符重载

3.1 如果没有显式定义，会自动调用直接父类的赋值操作符。（注意：不是 默认构造函数）

3.2 如果显式定义了，就只执行自己定义的版本，不再自动调用直接父类的赋值操作符，只执行自己的赋值操作符。

注意：如有需要对父类子部分进行赋值，应该在自己编写的代码中，显式调用父类的赋值操作符。

4. 析构函数

与构造函数 顺序相反。

## 四、多继承

和单继承的差别就是：需要考虑到多个直接父类。其它的都相同

1.构造函数（不包括复制构造函数）。

顺序：①所有直接父类；（按照基类继承列表中声明的顺序）②自己

注意：若直接父类还有父类，那么"直接父类的父类"会在"直接父类" 之前 构造。 可以理解为这是一个递归的过程，知道出现一个没有父类的类才停止。

2.1 如果 没有 显式定义构造函数，则"合成的默认构造函数"会自动依次调用所有直接父类的"默认构造函数"，然后调用编译器为自己自动生成的"合成的默认构造函数"。

2.2 如果显式定义了自己的构造函数

2.2.1 如果没有显式调用父类的任意一个构造函数，那么和"合成的默认构造函数"一样，会自动依次调用所有直接父类的 默认构造函数，然后调用自己的构造函数。

2.2.2 如果显式调用了父类的任意一个构造函数，那么按照基类列表的顺序，对于每一个父类依次判断：若显式调用了构造函数，那么会调用该父类相应的构造函数；如果没有显式调用，就调用默认构造函数。最后调用自己的构造函数。

2. 复制构造函数

顺序：①所有直接父类；（按照基类继承列表中声明的顺序）②自己

注意：和构造函数一样，若直接父类还有父类，那么"直接父类的父类"会在"直接父类" 之前 构造。 可以理解为这是一个递归的过程，知道出现一个没有父类的类才停止。

2.1 如果 没有显式定义复制构造函数，则"合成的复制构造函数"会自动依次调用所有直接父类的"复制构造函数"，然后调用编译器为自己自动生成的"合成的复制构造函数"（注意：不是默认构造函数）

2.2 如果显式定义了自己的复制构造函数 （和构造函数类似）

2.2.1 如果没有显式调用父类的任意一个构造函数，那么会先自动依次调用直接父类的 默认构造函数（注意：不是 复制构造函数）。

2.2.2 如果显式调用了直接父类的任意一个构造函数，那么按照基类列表的顺序，对于每一个父类依次判断：若显式调用了构造函数，那么会调用该父类相应的构造函数；如果没有显式调用，就调用默认构造函数。最后调用自己的复制构造函数。

3.赋值操作符重载

3.1 如果没有显式定义，会自动依次调用直接父类的赋值操作符。（注意：不是 默认构造函数）

3.2 如果显式定义了，就只执行自己定义的版本，不再自动调用直接父类的赋值操作符，只执行自己的赋值操作符。

注意：如有需要对父类子部分进行赋值，应该在自己编写的代码中，显式调用所有直接父类的赋值操作符。

4. 析构函数

与 构造函数 顺序相反。

## 五、虚继承

和多继承的差别就是：要考虑到虚基类，其它的都相同。（虚基类的初始化要早于非虚基类，并且只能由子类对其进行初始化）

1.构造函数（不包括复制构造函数）。

顺序：①所有虚基类（按照基类继承列表中声明的顺序进行查找）；②所有直接父类；（按照基类继承列表中声明的顺序）③自己

注意：若虚基类或者直接父类还有父类，那么"直接父类的父类"会在"直接父类" 之前 构造，"虚基类的父类"也会在"虚基类"之前构造。 可以理解为这是一个递归的过程，知道出现一个没有父类的类才停止。

2.1 如果 没有 显式定义构造函数，则"合成的默认构造函数"会先依次调用所有虚基类的默认构造函数，然后再自动依次调用所有直接父类的"默认构造函数"，最后调用编译器为自己自动生成的"合成的默认构造函数"。

2.2 如果显式定义了自己的构造函数 2.2.1 如果没有显式调用父类的任意一个构造函数，那么和"合成的默认构造函数"一样，会先依次调用所有虚基类的默认构造函数，然后再自动依次调用所有直接父类的 默认构造函数，最后调用自己的构造函数。

2.2.2 如果显式调用了父类的任意一个构造函数，那么按照基类列表的顺序，先初始化所有虚基类，再初始化所有直接父类。对于每一个父类依次判断：若显式调用了构造函数，那么会调用该父类相应的构造函数；如果没有显式调用，就调用默认构造函数。最后调用自己的构造函数。

2. 复制构造函数

顺序：①所有虚基类（按照基类继承列表中声明的顺序进行查找）；②所有直接父类；（按照基类继承列表中声明的顺序）③自己

注意：和构造函数一样，若虚基类或者直接父类还有父类，那么"直接父类的父类"会在"直接父类" 之前 构造，"虚基类的父类"也会在"虚基类"之前构造。 可以理解为这是一个递

归的过程，知道出现一个没有父类的类才停止。

2.1 如果 没有显式定义复制构造函数，则“合成的复制构造函数”会自动依次调用所有直接父类的“复制构造函数”，然后调用编译器为自己自动生成的“合成的复制构造函数”（注意：不是默认构造函数）

2.2 如果显式定义了自己的复制构造函数 （和构造函数类似）

2.2.1 如果没有显式调用父类的任意一个构造函数，那么会先依次调用所有虚基类的默认构造函数，然后再依次调用所有直接父类的 默认构造函数（注意：不是 复制构造函数）。

2.2.2 如果显式调用了直接父类的任意一个构造函数，那么按照基类列表的顺序，先初始化所有虚基类，再初始化所有直接父类。对于每一个父类依次判断：若显式调用了构造函数，那么会调用该父类相应的构造函数；如果没有显式调用，就调用默认构造函数。

3.赋值操作符重载

3.1 如果没有显式定义，会自动依次调用所有虚基类和所有直接父类的赋值操作符。（注意：不是 默认构造函数）

3.2 如果显式定义了，就只执行自己定义的版本，不再自动调用直接父类的赋值操作符，只执行自己的赋值操作符。

注意：如有需要对父类子部分进行赋值，应该在自己编写的代码中，显式调用所有虚基类和所有直接父类的赋值操作符。

4. 析构函数

与 构造函数 顺序相反。

## 六、总结：

1. 整体顺序：虚基类 --> 直接父类 -->自己

2. 在任何显式定义的构造函数中，如果没有显式调用父类的构造函数，那么就会调用父类的默认构造函数。

3. 合成的复制构造函数、合成的赋值操作符，（当没有显式定义时，编译器自动提供），会自动调用的是虚基类和直接父类的复制构造函数和赋值操作符，而不是默认构造函数；

4. 自己显式定义的复制构造函数，除非在初始化列表中显示调用，否则只会调用虚基类和父类的默认构造函数。

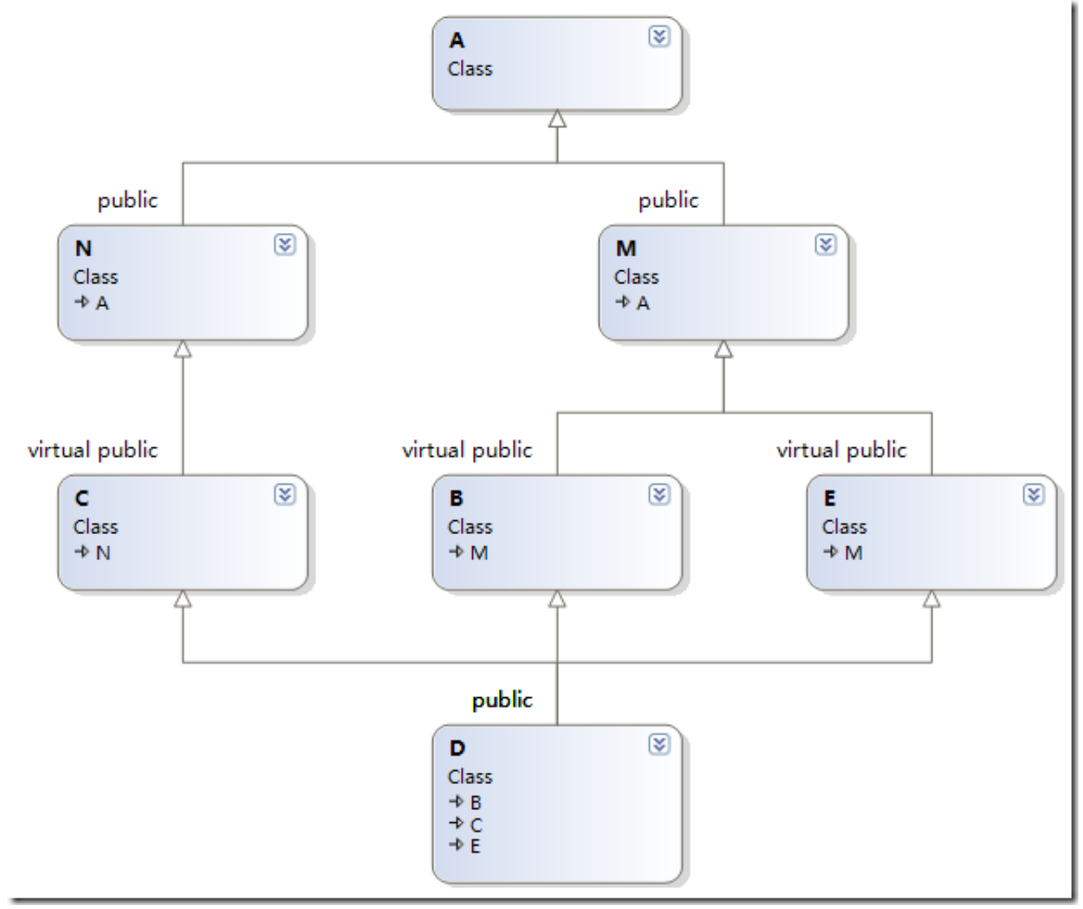
5. 自己显式定义的赋值操作符，除非显式调用，否则只执行自己的代码。

6. 析构函数的执行顺序与 构造函数 相反。

## 七、例子程序

话说只有自己写一个程序，然后研究运行结果，才会掌握的更好。所以下面就是个例子程序了。可以根据需要，注释掉某个类的相应函数，观察结果。

1. 该例子的继承层次图为：（M和N是虚基类）




2. 代码如下

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

class A {
public:
    A() {
        cout<<"int A::A()"<<endl;
    }
    A(A &a) {
        cout<<"int A::A(A &a)"<<endl;
    }
    A& operator=(A& a) {
        cout<<"int A::operator=(A &a)"<<endl;
        return a;
    }
    virtual ~A() {
        cout<<"int A::~A()"<<endl;
    }
};

class M :public A {
public:
    M() {
        cout<<"int M::M()"<<endl;
    }
```

```
    M(M &a) {
        cout<<"int M::M(M &a)"<<endl;
    }
    M& operator=(M& m) {
        cout<<"int M::operator=(M &a)"<<endl;
        return m;
    }
    virtual ~M() {
        cout<<"int M::~M()"<<endl;
    }
};

class B:virtual public M {
public:
    B() {
        cout<<"int B::B()"<<endl;
    }
    B(B &a) {
        cout<<"int B::B(B &a)"<<endl;
    }
    B& operator=(B& b) {
        cout<<"int B::operator=(B &a)"<<endl;
        return b;
    }
    virtual ~B() {
        cout<<"int B::~B()"<<endl;
    }

};

class N :public A {
public:
    N() {
        cout<<"int N::N()"<<endl;
    }
    N(N &a) {
        cout<<"int N::N(N &a)"<<endl;
    }
    N& operator=(N& n) {
        cout<<"int N::operator=(N &a)"<<endl;
        return n;
    }
    virtual ~N() {
        cout<<"int N::~N()"<<endl;
    }
};
class C:virtual public N {
public:
    C() {
        cout<<"int C::C()"<<endl;
    }
    C(C &a) {
        cout<<"int C::C(C &a)"<<endl;
    }
    C& operator=(C& c) {
```

```
        cout<<"int C::operator=(C &a)"<<endl;
        return c;
    }
    virtual ~C() {
        cout<<"int C::~C()"<<endl;
    }
};
class E:virtual public M{
public:
    E() {
        cout<<"int E::E()"<<endl;
    }
    E(E &a) {
        cout<<"int E::E(E &a)"<<endl;
    }
    E& operator=(E& e) {
        cout<<"int E::operator=(E &a)"<<endl;
        return e;
    }
    virtual ~E() {
        cout<<"int E::~E()"<<endl;
    }
};
class D:public B, public C, public E {
public:
    D() {
        cout<<"int D::D()"<<endl;
    }
    D(D &a) {
        cout<<"int D::D(D &a)"<<endl;
    }
    D& operator=(D& d) {
        cout<<"int D::operator=(D &a)"<<endl;
        return d;
    }
    virtual ~D() {
        cout<<"int D::~D()"<<endl;
    }
};


int main(int argc, char **argv) {
    cout<<"-------构造函数-------"<<endl;
    D d;
    cout<<"-------复制构造函数-------"<<endl;
    D d1(d);
    cout<<"-------赋值操作符-------"<<endl;
    d = d1;
    cout<<"-------析构函数-------"<<endl;


    return 0;
}
```

3. 运行结果与分析

分析：M和N是虚基类，但是A不是虚基类。B和E共享一个M，但是M和N都会含有类A的部分，因为A不是虚基类，所以M和N不共享A。下面的注释部分为添加的分析。

```
-------构造函数-------
int A::A()
int M::M()//构造虚基类M时，要先构造其父类A
int A::A()
int N::N()//和M一样，构造虚基类N时，也要先构造其父类A
int B::B()//构造完虚基类，开始构造直接父类，按照声明顺序为B、C、E
int C::C()
int E::E()
int D::D()//最后构造自己
-------复制构造函数-------
int A::A()
int M::M()
int A::A()
int N::N()
int B::B()
int C::C()
int E::E()
int D::D(D &a)//因为D中定义了复制构造函数，并且没有显式调用父类的构造函数，所以所有的“虚基类”和“直接父类”都调用默认构造函数
-------赋值操作符-------
int D::operator=(D &a) //因为显式调用了赋值操作符，那么就只调用自己的代码，不会隐式调用其它的函数
-------析构函数-------
int D::~D()
int E::~E()
int C::~C()
int B::~B()
int N::~N()
int A::~A()
int M::~M()
int A::~A()//因为main函数中定义了两个D对象，所以main函数结束时要进行析构两个D对象。析构的顺序与 构造函数相反。
int D::~D()
int E::~E()
int C::~C()
int B::~B()
int N::~N()
int A::~A()
int M::~M()
int A::~A()

Press any key to continue.
```



nkxyf
关注 - 154
粉丝 - 4
+加关注



« 上一篇：拷贝构造函数与赋值操作符的具体实现 【转】参考度4.5星
» 下一篇：Linux下C++开发工具介绍【转】 扫盲用 参考度4.5星

posted on 2012-07-18 19:51 nkxyf 阅读(875) 评论(0) 编辑 收藏

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。




最新**IT**新闻：
· 可否？Touch ID脱离Home键安装在屏幕上
· 科技改造了农业，可也改出了问题
· 专注不赚钱20年：贝索斯如何说服VC继续支持自己烧钱？
· Google地图看世界：人类文明如此震撼！
· 国家自然科学奖一等奖都颁给过谁？“透明计算”是否会自惭形秽？
» 更多新闻...


最新知识库文章：
· 通俗解释「为什么数据库难以拓展」
· 手机淘宝高质量持续交付探索之路
· 高效运维最佳实践（01）：七字诀，不再憋屈的运维
· 什么是工程师文化？
· 大数据架构和模式（五）对大数据问题应用解决方案模式并选择实现它的产品
» 更多知识库文章...