

个人资料



lwbeeyond



访问： 311537次

积分： 4306

等级： 

BLOG > 5

排名： 第2956名

原创： 165篇 转载： 29篇

译文： 1篇 评论： 214条

文章搜索

博客专栏



STL学习笔记  
文章： 16篇  
阅读： 16858

文章分类

C/C++ (44)

Linux Shell (4)

STL (17)

Linux (33)

vxWorsk (3)

ACE (4)

[Markdown](#)那么好，还不来试试 [中国云计算大会最新议题](#) [5月问答又送C币咯！](#) [Hadoop](#)实战高手速成宝典

## C++拷贝构造函数详解

分类： C/C++

2011-02-23 13:39

84758人阅读

评论(140)

收藏

举报

c++

fun

class

编译器

delete

c

### 一. 什么是拷贝构造函数

首先对于普通类型的对象来说，它们之间的复制是很简单的，例如：

```
[c-sharp]
01. int a = 100;
02. int b = a;
```

而类对象与普通对象不同，类对象内部结构一般较为复杂，存在各种成员变量。

下面看一个类对象拷贝的简单例子。

```
[c-sharp]
01. #include <iostream>
02. using namespace std;
03.
04. class CExample {
05. private:
06.     int a;
07. public:
08.     //构造函数
09.     CExample(int b)
10.     { a = b;}
11.
12.     //一般函数
13.     void Show ()
14.     {
15.         cout<<a<<endl;
16.     }
17. };
18.
19. int main()
20. {
21.     CExample A(100);
22.     CExample B = A; //注意这里的对象初始化要调用拷贝构造函数，而非赋值
23.     B.Show ();
24.     return 0;
```

- Eclipse (2)
- English (3)
- 设计模式 (25)
- 网络编程 (14)
- 汽车电子 (4)
- 加密算法 (1)
- 正则表达式 (2)
- 生活 (2)
- 思维改变生活 (4)
- 嵌入式linux开发 (8)
- QT (14)
- uC/OS-II (6)
- cocos2d (1)
- Java (6)

- 文章存档
- 2015年03月 (3)
- 2015年02月 (4)
- 2014年11月 (8)
- 2014年10月 (2)
- 2013年12月 (1)
- 展开

- 阅读排行
- C++拷贝构造函数详解 (84663)
- 如何正确的关闭 MFC 线程 (13943)
- C++类型转换详解--const (7777)
- 学习C++该看什么书? (5854)
- SPI 协议 (4056)
- 汽车诊断简介 (4024)
- VxWorks学习笔记 -- 信号 (3768)
- GDB调试精粹 (3526)
- QT 调用外部程序 (3399)
- 学习 effortless English 2 (3234)

- 评论排行
- C++拷贝构造函数详解 (140)
- 如何正确的关闭 MFC 线程 (7)
- C++类型转换详解--const (6)
- 详解C++ friend关键字 (6)
- strcpy 详解 (3)
- sizeof 精要 (3)
- VxWorks学习笔记 -- 信号 (3)
- STL编程轻松入门 (3)
- STL学习笔记----1.概述 (2)

```
25.     }

运行程序，屏幕输出100。从以上代码的运行结果可以看出，系统为对象 B 分配了内存并完成了与对象 A 的复制过程。就类对象而言，相同类型的类对象是通过拷贝构造函数来完成整个复制过程的。

下面举例说明拷贝构造函数的工作过程。

[c-sharp]
01. #include <iostream>
02. using namespace std;
03.
04. class CExample {
05. private:
06.     int a;
07. public:
08.     //构造函数
09.     CExample(int b)
10.     { a = b;}
11.
12.     //拷贝构造函数
13.     CExample(const CExample& C)
14.     {
15.         a = C.a;
16.     }
17.
18.     //一般函数
19.     void Show ()
20.     {
21.         cout<<a<<endl;
22.     }
23. };
24.
25. int main()
26. {
27.     CExample A(100);
28.     CExample B = A; // CExample B(A); 也是一样的
29.     B.Show ();
30.     return 0;
31. }
```

CExample(const CExample& C) 就是我们自定义的拷贝构造函数。可见，拷贝构造函数是一种特殊的构造函数，函数的名称必须和类名称一致，它必须的一个参数是本类型的一个引用变量。

## 二. 拷贝构造函数的调用时机

在C++中，下面三种对象需要调用拷贝构造函数！

- 对象以值传递的方式传入函数参数

```
[c-sharp]
01. class CExample
02. {
03. private:
04.     int a;
05.
06. public:
```

推荐文章

- \* 2015博文大赛
- \*CSDN Markdown简明教程-基本使用
- \*CSDN Markdown简明教程-快速上手
- \*CSDN Markdown如何绘制UML图
- \*CSDN Markdown使用LaTeX编写数学公式
- \*CSDN Markdown扩展语法

最新评论

- C++拷贝构造函数详解

SmilingSunrise: 不错，写的很详细具体！
- GDB调试精粹

菜鸟实验室: 帝仰大学的？
- C++拷贝构造函数详解

veryitman: 辛苦楼主了.赞一个!不过你的例子应该是在 vs 上面验证的,在 mac 平台下 xcode 编译运行...
- C++拷贝构造函数详解

Felix\_0920: 请教楼主一个问题，对于用户自定义的类，如果没有默认的空参构造函数，那么编译器是如果生成类的临时变量对...
- C++拷贝构造函数详解

sinat\_27624023: 博主，想请教你一个问题。拷贝构造函数能将一个对象的string类字符串拷贝给另一个对象吗？
- C++拷贝构造函数详解

CourageK: 楼主真是太有心了，让好多人都对复制构造函数有了很深入的理解，从此面试此类问题不再纠结了。谢谢你~
- C++拷贝构造函数详解

编程艺术家: mark
- strcpy 详解

Findxiaoxun: Good，学习了。已转载，且附上了原链接。  
http://www.findspace.name/eas...
- C++拷贝构造函数详解

qszchew: 讲得真好！以前一直搞不懂的！
- 设计模式C++描述----03.工厂(Fa

daemonwang: cool, 23中设计模式分析的很好。

```
07. //构造函数
08. CExample(int b)
09. {
10.     a = b;
11.     cout<<"creat: "<<a<<endl;
12. }
13.
14. //拷贝构造
15. CExample(const CExample& C)
16. {
17.     a = C.a;
18.     cout<<"copy"<<endl;
19. }
20.
21. //析构函数
22. ~CExample()
23. {
24.     cout<< "delete: "<<a<<endl;
25. }
26.
27. void Show ()
28. {
29.     cout<<a<<endl;
30. }
31. };
32.
33. //全局函数，传入的是对象
34. void g_Fun(CExample C)
35. {
36.     cout<<"test"<<endl;
37. }
38.
39. int main()
40. {
41.     CExample test(1);
42.     //传入对象
43.     g_Fun(test);
44.
45.     return 0;
46. }
```

调用g\_Fun()时，会产生以下几个重要步骤：

- (1). test对象传入形参时，会先会产生一个临时变量，就叫 C 吧。
- (2). 然后调用拷贝构造函数把test的值给C。 整个这两个步骤有点像：CExample C(test);
- (3). 等g\_Fun()执行完后，析构掉 C 对象。

2. 对象以值传递的方式从函数返回

```
[c-sharp]
01. class CExample
02. {
03. private:
04.     int a;
05.
06. public:
07.     //构造函数
08.     CExample(int b)
09.     {
10.         a = b;
```

```
11.     }
12.
13.     //拷贝构造
14.     CExample(const CExample& C)
15.     {
16.         a = C.a;
17.         cout<<"copy"<<endl;
18.     }
19.
20.     void Show ()
21.     {
22.         cout<<a<<endl;
23.     }
24. };
25.
26. //全局函数
27. CExample g_Fun()
28. {
29.     CExample temp(0);
30.     return temp;
31. }
32.
33. int main()
34. {
35.     g_Fun();
36.     return 0;
37. }
```

当g\_Fun()函数执行到return时，会产生以下几个重要步骤：

- (1). 先会产生一个临时变量，就叫XXXX吧。
- (2). 然后调用拷贝构造函数把temp的值给XXXX。整个这两个步骤有点像：`CExample XXXX(temp);`
- (3). 在函数执行到最后先析构temp局部变量。
- (4). 等g\_Fun()执行完后再析构掉XXXX对象。

3. 对象需要通过另外一个对象进行初始化：

```
[c-sharp]
01. CExample A(100);
02. CExample B = A;
03. // CExample B(A);
```

后两句都会调用拷贝构造函数。

### 三. 浅拷贝和深拷贝

#### 1. 默认拷贝构造函数

很多时候在我们都不知道拷贝构造函数的情况下，传递对象给函数参数或者函数返回对象都能很好的进行，这是因为编译器会给我们自动产生一个拷贝构造函数，这就是“默认拷贝构造函数”，这个构造函数很简单，仅仅使用“老对象”的数据成员的值对“新对象”的数据成员一一进行赋值，它一般具有以下形式：

```
[c-sharp]
```

```
01. Rect::Rect(const Rect& r)
02. {
03.     width = r.width;
04.     height = r.height;
05. }
```

当然，以上代码不用我们编写，编译器会为我们自动生成。但是如果认为这样就可以解决对象的复制问题，那就错了，让我们来考虑以下一段代码：

```
[c-sharp]
01. class Rect
02. {
03. public:
04.     Rect()          // 构造函数，计数器加1
05.     {
06.         count++;
07.     }
08.     ~Rect()         // 析构函数，计数器减1
09.     {
10.         count--;
11.     }
12.     static int getCount()    // 返回计数器的值
13.     {
14.         return count;
15.     }
16. private:
17.     int width;
18.     int height;
19.     static int count;        // 一静态成员做为计数器
20. };
21.
22. int Rect::count = 0;         // 初始化计数器
23.
24. int main()
25. {
26.     Rect rect1;
27.     cout<<"The count of Rect: "<<Rect::getCount()<<endl;
28.
29.     Rect rect2(rect1);      // 使用rect1复制rect2，此时应该有两个对象
30.     cout<<"The count of Rect: "<<Rect::getCount()<<endl;
31.
32.     return 0;
33. }
```

这段代码对前面的类，加入了一个静态成员，目的是进行计数。在主函数中，首先创建对象rect1，输出此时的对象个数，然后使用rect1复制出对象rect2，再输出此时的对象个数，按照理解，此时应该有两个对象存在，但实际程序运行时，输出的都是1，反应出只有1个对象。此外，在销毁对象时，由于会调用销毁两个对象，类的析构函数会调用两次，此时的计数器将变为负数。

说白了，就是拷贝构造函数没有处理静态数据成员。

出现这些问题最根本就在于在复制对象时，计数器没有递增，我们重新编写拷贝构造函数，如下：

```
[c-sharp]
01. class Rect
02. {
03. public:
```

```
04.     Rect()      // 构造函数，计数器加1
05.     {
06.         count++;
07.     }
08.     Rect(const Rect& r)  // 拷贝构造函数
09.     {
10.         width = r.width;
11.         height = r.height;
12.         count++;        // 计数器加1
13.     }
14.     ~Rect()      // 析构函数，计数器减1
15.     {
16.         count--;
17.     }
18.     static int getCount() // 返回计数器的值
19.     {
20.         return count;
21.     }
22. private:
23.     int width;
24.     int height;
25.     static int count;      // 一静态成员做为计数器
26. };
```

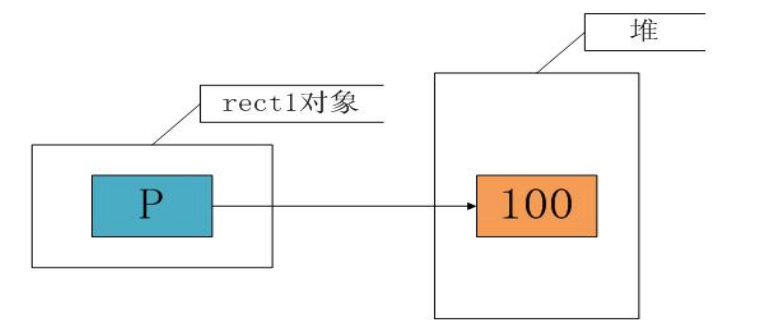
## 2. 浅拷贝

所谓浅拷贝，指的是在对象复制时，只对对象中的数据成员进行简单的赋值，默认拷贝构造函数执行的也是浅拷贝。大多情况下“浅拷贝”已经能很好地工作了，但是一旦对象存在了动态成员，那么浅拷贝就会出问题了，让我们考虑如下一段代码：

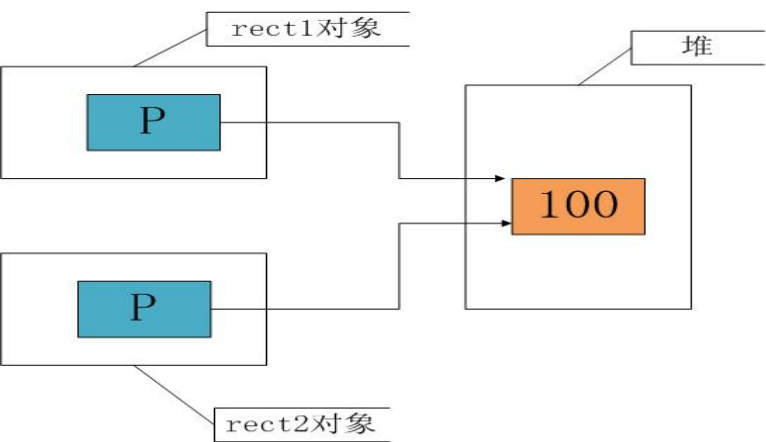
```
[c-sharp]
01. class Rect
02. {
03. public:
04.     Rect()      // 构造函数，p指向堆中分配的一空间
05.     {
06.         p = new int(100);
07.     }
08.     ~Rect()      // 析构函数，释放动态分配的空间
09.     {
10.         if(p != NULL)
11.         {
12.             delete p;
13.         }
14.     }
15. private:
16.     int width;
17.     int height;
18.     int *p;      // 一指针成员
19. };
20.
21. int main()
22. {
23.     Rect rect1;
24.     Rect rect2(rect1);  // 复制对象
25.     return 0;
26. }
```

在这段代码运行结束之前，会出现一个运行错误。原因就在于在进行对象复制时，对于动态分配的内容没有进行正确的操作。我们来分析一下：

在运行定义rect1对象后，由于在构造函数中有一个动态分配的语句，因此执行后的内存情况大致如下：



在使用rect1复制rect2时，由于执行的是浅拷贝，只是将成员的值进行赋值，这时 rect1.p = rect2.p，也即这两个指针指向了堆里的同一个空间，如下图所示：



当然，这不是我们所期望的结果，在销毁对象时，两个对象的析构函数将对同一个内存空间释放两次，这就是错误出现的原因。我们需要的不是两个p有相同的值，而是两个p指向的空间有相同的值，解决办法就是使用“深拷贝”。

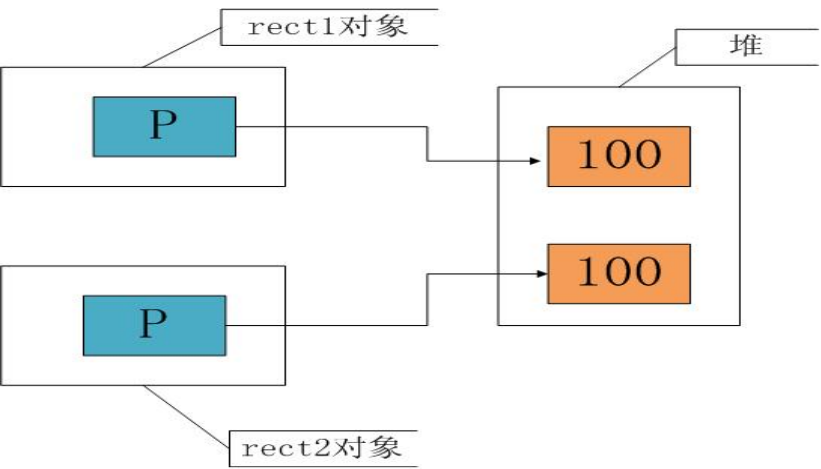
3. 深拷贝

在“深拷贝”的情况下，对于对象中动态成员，就不能仅仅简单地赋值了，而应该重新动态分配空间，如上面的例子就应该按照如下的方式进行处理：

```
[c-sharp]
01. class Rect
02. {
03. public:
04.     Rect()      // 构造函数，p指向堆中分配的一空间
05.     {
06.         p = new int(100);
07.     }
08.     Rect(const Rect& r)
09.     {
10.         width = r.width;
11.         height = r.height;
12.         p = new int;    // 为新对象重新动态分配空间
```

```
13.         *p = *(r.p);
14.     }
15.     ~Rect()    // 析构函数，释放动态分配的空间
16.     {
17.         if(p != NULL)
18.         {
19.             delete p;
20.         }
21.     }
22. private:
23.     int width;
24.     int height;
25.     int *p;    // 一指针成员
26. };
```

此时，在完成对象的复制后，内存的一个大致情况如下：



此时rect1的p和rect2的p各自指向一段内存空间，但它们指向的空间具有相同的内容，这就是所谓的“深拷贝”。

3. 防止默认拷贝发生

通过对对象复制的分析，我们发现对象的复制大多在进行“值传递”时发生，这里有一个小技巧可以防止按值传递——**声明一个私有拷贝构造函数**。甚至不必去定义这个拷贝构造函数，这样因为拷贝构造函数是私有的，如果用户试图按值传递或函数返回该类对象，将得到一个编译错误，从而可以避免按值传递或返回对象。

```
[c-sharp]
01. // 防止按值传递
02. class CExample
03. {
04. private:
05.     int a;
06.
07. public:
08.     //构造函数
09.     CExample(int b)
10.     {
11.         a = b;
12.         cout<<"creat: "<<a<<endl;
13.     }
```



```
14.
15. private:
16.     //拷贝构造, 只是声明
17.     CExample(const CExample& C);
18.
19. public:
20.     ~CExample()
21.     {
22.         cout<< "delete: "<<a<<endl;
23.     }
24.
25.     void Show ()
26.     {
27.         cout<<a<<endl;
28.     }
29. };
30.
31. //全局函数
32. void g_Fun(CExample C)
33. {
34.     cout<<"test"<<endl;
35. }
36.
37. int main()
38. {
39.     CExample test(1);
40.     //g_Fun(test); 按值传递将出错
41.
42.     return 0;
43. }
```

四. 拷贝构造函数的几个细节

1. 拷贝构造函数里能调用private成员变量吗?

解答：这个问题是在网上见的，当时一下子有点晕。其时从名子我们就知道拷贝构造函数其时就是一个特殊的构造函数，操作的还是自己类的成员变量，所以不受private的限制。

2. 以下函数哪个是拷贝构造函数, 为什么?

```
[c-sharp]
01. X::X(const X&);
02. X::X(X);
03. X::X(X&, int a=1);
04. X::X(X&, int a=1, int b=2);
```

解答：对于一个类X，如果一个构造函数的第一个参数是下列之一：

- a) X&
- b) const X&
- c) volatile X&
- d) const volatile X&

且没有其他参数或其他参数都有默认值, 那么这个函数是拷贝构造函数.

```
[c-sharp]
01. X::X(const X&); //是拷贝构造函数
```

```
02. X::X(X&, int=1); //是拷贝构造函数
03. X::X(X&, int a=1, int b=2); //当然也是拷贝构造函数
```

3. 一个类中可以存在多于一个的拷贝构造函数吗？  
解答：类中可以存在超过一个拷贝构造函数。

```
[c-sharp]
01. class X {
02. public:
03.     X(const X&);      // const 的拷贝构造
04.     X(X&);           // 非const的拷贝构造
05. };
```

注意, 如果一个类中只存在一个参数为 X& 的拷贝构造函数, 那么就不能使用const X或volatile X的对象实行拷贝初始化.

```
[c-sharp]
01. class X {
02. public:
03.     X();
04.     X(X&);
05. };
06.
07. const X cx;
08. X x = cx;    // error
```

如果一个类中没有定义拷贝构造函数, 那么编译器会自动产生一个默认的拷贝构造函数。  
这个默认的参数可能为 X::X(const X&)或 X::X(X&), 由编译器根据下文决定选择哪一个。

上一篇 [c/c++中typedef详解](#)

下一篇 [C++类型转换详解--const\\_cast](#)

主题推荐

[c++](#) [局部变量](#) [编译器](#) [namespace](#) [内存](#)

猜你在找

[C++学习之深入理解虚函数—虚函数表解析](#)

[cs硕士妹子找工作经历阿里人搜等互联网](#)

[我的2012-分享我的四个项目经验](#)

[割绳子的作者你如此歧视无视鄙视中国人这是何苦呢](#)

[手把手实现红黑树](#)

[【精品课程】JavaScript for Qt Quick\(QML\)](#)

[【精品课程】Qt基础与Qt on Android入门](#)

[【精品课程】深入浅出Java的反射](#)

[【精品课程】JavaScript for Qt Quick\(QML\)](#)

[【精品课程】C语言及程序设计初步](#)

更多相关资源：[函数](#) [c语言](#)

更多职位尽在 **CSDN JOB**

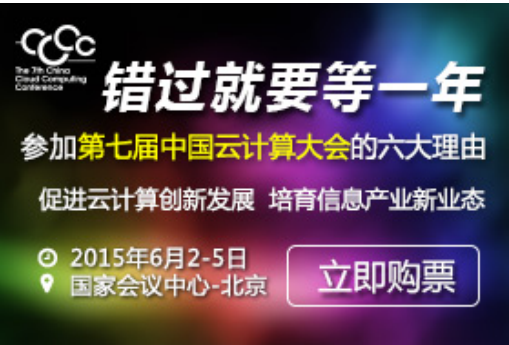
[iOS, Android, HTML5 程序员](#)

[我要跳槽](#)

[iOS, Android, HTML5 程序员](#)

[我要跳槽](#)

关闭



上海胜因软件技术有限公司		8-10K/月	上海普望企业管理有限公司		8-10K/月
delphi工程师		我要跳槽	iOS, Android, HTML5 储备程序员		我要跳槽
深圳硕软技术有限公司		10-20K/月	上海普望企业管理有限公司		4-6K/月



查看评论

118楼 [SmilingSunrise](#) 2015-05-13 16:36发表



不错，写的很详细具体！

117楼 [veryitman](#) 2015-05-09 22:10发表



辛苦楼主了.赞一个!  
不过你的例子应该是在 vs 上面验证的,在 mac 平台下 xcode 编译运行有点不一样.

116楼 [Felix\\_0920](#) 2015-05-08 17:30发表



请教楼主一个问题，对于用户自定义的类，如果没有默认的无参构造函数，那么编译器是如果生成类的临时变量对象的？

115楼 [sinat\\_27624023](#) 2015-04-22 16:36发表



博主，想请教你一个问题。拷贝构造函数能将一个对象的string类字符串拷贝给另一个对象吗？

114楼 [CourageK](#) 2015-04-02 09:35发表



楼主真是太有心了，让好多人都对复制构造函数有了很深入的理解，从此面试此类问题不再纠结了。谢谢你~

113楼 [编程艺术家](#) 2015-03-29 16:00发表



mark

112楼 [qszchew](#) 2015-03-17 19:36发表



讲得真好！以前一直搞不懂的！

111楼 [li1191863273](#) 2015-02-11 03:04发表



受教了

110楼 [mz490848173](#) 2015-02-09 13:45发表



想问一下 那个 复制构造函数 所复制的函数 是在 堆内存里 吗 还是 栈内存里

109楼 [小米mimica](#) 2015-01-15 20:21发表



写的非常详细，也很易于理解！受益匪浅！

108楼 [jinjinwu](#) 2015-01-14 15:45发表



看书一直没看懂，博主讲的很清楚啊，

107楼 [Sungyers](#) 2015-01-10 15:54发表



博主给力，狂赞！

106楼 [louObaichu](#) 2015-01-06 15:28发表



mark

105楼 [ColaWJY](#) 2014-12-16 21:54发表



给力！Mark了！

104楼 [Huai-yu](#) 2014-12-04 09:05发表



支持楼主，赞一个

103楼 [御弟叔叔](#) 2014-11-29 14:13发表



赞一个！

102楼 [\\_LebronLu\\_](#) 2014-11-28 11:16发表



很详细，佩服！

101楼 [jyf823691221](#) 2014-11-13 17:00发表



赞一个 写的太好了 豁然开朗呀

100楼 [eyeshot\\_yang33](#) 2014-11-06 10:16发表



谢谢，看完豁然开朗了！

99楼 [hityxhvp](#) 2014-11-05 21:48发表



通读全文，非常感谢！

98楼 [tandyx](#) 2014-11-03 17:10发表



很不错， 非常给力。特地登陆来点赞。我的密码都试了好多次才登上来。哈哈。

97楼 [zhaolianyun](#) 2014-10-26 20:28发表



真心不错，非常受教！！

96楼 [wjwizard](#) 2014-10-23 19:05发表



赞一个，好贴

95楼 [eternal\\_tune](#) 2014-10-14 12:41发表



CExample B = A; // CExample B(A); 也是一样的  
这段代码应该是赋值构造函数（操作符为'='）。如果将此代码放在private里面就编译不过：  
CExample & operator=(const CExample&);

CExample B(A);此段才是调用的拷贝构造函数

Re: [wobushixiaomi](#) 2014-12-31 14:31发表



回复u012844596：正解！

94楼 [Sylvernass](#) 2014-10-13 19:21发表

写的真心很不错，这部分对初学者来说会搞的有点混，但是看了这篇文章，顿时明白了很多，很清晰，不得不来留言点赞



93楼 [royliu1](#) 2014-10-11 12:54发表



写得好！ 顺便看了下前辈们的讨论，有收获！ 谢谢

92楼 [Edwin404](#) 2014-09-30 16:33发表



问个问题，为啥“2. 对象以值传递的方式从函数返回”这个标题下面那个例子什么都没有打印呢？

91楼 [lichangyu2011](#) 2014-09-29 10:05发表



转载了！学知识真应该这样，受教了！惭愧一个！！！

90楼 [puppyp1pg](#) 2014-09-18 20:09发表



太详细了！果断赞起！

89楼 [terry\\_o0o](#) 2014-09-17 16:37发表



一直不太理解，就找到lz这个文章了，豁然开朗！

88楼 [jiaqingmin1990](#) 2014-09-15 10:50发表



nice....

87楼 [wangxm1111](#) 2014-09-14 16:49发表



一下子思路很清晰，讲的很不错

86楼 [图像配准菜鸟](#) 2014-09-13 10:36发表



受教了！谢谢你耐心的讲解！

85楼 [他们叫我周周周](#) 2014-09-12 14:09发表



太棒了~

84楼 [忆之独秀](#) 2014-09-06 16:16发表



写的不能更好

83楼 [贪睡的萝卜](#) 2014-09-05 16:43发表



受益匪浅，感谢博主

82楼 [叶子399](#) 2014-09-02 11:03发表



大赞~

81楼 [旭梦1990](#) 2014-08-25 08:44发表




太好了，了解了拷贝构造函数

80楼 [uiong8163](#) 2014-08-09 00:38发表



大神，很感谢！


79楼 [panjunnn](#) 2014-08-07 14:47发表

 受教了，好啊


78楼 [坚决不做程序狗](#) 2014-08-07 08:48发表

 牛逼。。。。学习了


77楼 [PushFang](#) 2014-07-30 22:36发表

 大赞!!!!


76楼 [笃志近思](#) 2014-07-25 17:16发表

 受教了，写的非常棒！


75楼 [bin03](#) 2014-07-21 11:30发表

 很好。条理清晰。


74楼 [某种意境](#) 2014-07-09 23:16发表

 终于晓得这玩意儿了


73楼 [junjie58msp](#) 2014-07-02 11:08发表

 此文写的太好了。赞一个。


72楼 [HUASHUIXIAOHAI](#) 2014-06-27 14:49发表

 楼主写的真心不错，大赞！


71楼 [xanarry](#) 2014-06-19 09:12发表

 受益不浅，谢谢博主


70楼 [卿笃军](#) 2014-05-25 08:47发表

 拷贝构造函数：  
A(A a){}  
值传递传参为什么会编译通不过？


Re: [heipacker](#) 2014-06-04 23:11发表

 回复u012339743：这个不是复制构造函数

Re: [卿笃军](#) 2014-06-05 22:18发表


 回复fupacker：我只是想问这种传参方式 编译为何通不过。

Re: [heipacker](#) 2014-06-06 23:11发表

 回复u012339743：编译不通过是因为A a这里的定义问题，只有在看到一个类的定义的时候才能把一个数据成员声明为这个类型的对象，但是当 一个类的类头被看到时，它就被视为已经被声明了，所以一个类可以用指向自身类型的指针或引用作为数据成员

这里数据成员你也可以认为是一个变量的定义 看看C++ primer

Re: [卿笃军](#) 2014-06-07 00:25发表

 回复fupacker：这个我明白。就像结构体一样：  
struct node

```
{
struct node next; //错误
};
struct node
{
struct node *next; //正确
};
```

但是：  
class A  
{  
public:  
A(A a); //错误  
void Print(A a){}; //正确  
};  
这个是如何解释？

Re: [gdouse1093](#) 2014-09-28 22:02发表



回复u012339743： 如果不是引用会造成无限递归， （会无限的调用拷贝构造函数）

Re: [heipacker](#) 2014-06-07 13:45发表



回复u012339743： 拷贝构造函数一定要使用引用传递 你查一下就知道了

Re: [卿笃军](#) 2014-06-07 14:55发表



回复fupacker： 这个我查过，也明白A(A a)这样会带来的问题。  
就是上面那个问题有些不明白。

Re: [小王王小](#) 2014-08-04 16:37发表



回复u012339743： 拷贝构造函数如果允许值传递，那么实参对象传递给形参对象，  
会拥有两个存储空间，新创建的临时存储空间对象初始化时，同样需要调用拷贝构造函数，以此类推，会永无休止的调用拷贝构造函数，最终会导致栈溢出。

Re: [卿笃军](#) 2014-09-28 23:42发表



回复wangyong19881123： 我明白了。

69楼 [动感蚂蚁](#) 2014-05-15 13:03发表



学习了，谢谢

68楼 [tian15134549098](#) 2014-04-21 20:41发表



超级给力，赞

67楼 [lvjing2](#) 2014-04-20 10:52发表



必须赞一个！

66楼 [吉米芽菜](#) 2014-03-30 10:49发表



衷心地赞一个

65楼 [松木](#) 2014-03-23 18:06发表



楼主分析得太好了，让我对拷贝构造函数有了更深的认识，赞！


64楼 [雨中奔跑](#) 2014-02-17 11:55发表

 很详细nice

63楼 [loe](#) 2013-12-30 17:27发表

 mark!


62楼 [feierfeierfly](#) 2013-12-25 16:20发表

 受教了！谢谢楼主！

61楼 [无之念](#) 2013-12-23 17:13发表

 受教了


60楼 [chuyuanjie123](#) 2013-12-16 22:14发表

 很好。思路清晰


59楼 [曾是土木人](#) 2013-12-15 00:19发表

 总结的很好


58楼 [lcjwxd](#) 2013-12-05 15:39发表

 初学者也来赞一个，以后好好学习，天天向上！

57楼 [zzqq0103](#) 2013-12-04 17:20发表

 真的说的很好，受教了！！

56楼 [苍原狮啸](#) 2013-11-27 11:58发表

 后文的添加的内容很给力


55楼 [HXD974287503](#) 2013-11-21 16:17发表

 好好！

54楼 [virginia-qing](#) 2013-11-21 14:29发表

 good


53楼 [zhw\\_CZ](#) 2013-11-20 22:59发表

 写的真好，谢谢。


52楼 [yulu27](#) 2013-11-11 16:37发表

 写的真不错

51楼 [tyy\\_ing](#) 2013-11-10 20:58发表

 写的很棒 怒赞

50楼 [code\\_feng](#) 2013-11-05 15:29发表

 学习了.....谢谢楼主~

49楼 [黑白尤文](#) 2013-08-31 09:16发表





请问文章中的图是用的什么工具画的啊？

Re: 黑白尤文 2013-09-14 22:34发表



回复juvenfan: 这么强大。。。我用visio画出来的图都很丑.....

Re: faith19871023 2013-09-07 15:34发表



回复juvenfan: visio就可以啊

48楼 chz429 2013-08-25 16:06发表



先赞一个~

关于对象以值传递的方式从函数返回会调用拷贝构造函数，怎么在Devc++测试的时候从函数返回不会调用拷贝构造函数，而在Visual Stdio却调用了呢

```
[cpp]
01. #include<iostream>
02. using namespace std;
03. struct Node
04. {
05.     int a, b;
06.     Node(int i, int j):a(i), b(j){}
07.     Node(const Node& a)
08.     {
09.         cout << "copy" << endl;
10.     }
11. };
12. Node fun()
13. {
14.     Node n = Node(1,2);
15.     return n;
16. }
17. int main()
18. {
19.     fun();
20.     system("pause");
21. }
```

Re: whyisyoung 2014-11-25 23:05发表



回复chz429: g++ 3.2 以上版本对编译器做了优化，导致第三种值返回的情况不调用拷贝构造函数， mac 上也不会调用， vs2012 仍调用，

47楼 lnnlkm 2013-08-22 16:18发表



受教了，谢谢

46楼 鸭梨小乖宝宝 2013-08-15 18:35发表



非常好的一篇文章，清晰又有条理

45楼 夏雪冬日 2013-08-10 14:48发表



写的简单、清晰。还不错！


44楼 a429051366 2013-08-06 12:40发表



真心希望楼主能写一本关于c++的书，业界的书可谓汗牛充栋，讲的明白清楚的寥若晨星，大部分的书都是讲讲语法，再贴上几个例子，看的人似懂非懂，云里雾里，整本书好像条理清楚，脉络清晰其实一塌糊涂，外国人的书又好点，可是看外文的大部头真的要耐心的，能说的这么清楚明白太难得了，真心希望楼主写本书，不写语法什么的，就和论坛里面的这种一样，真心

的牛啊，要能给做几个c++的视频就更好了，赞一个


43楼 [wsb929](#) 2013-08-04 21:10发表

 顶 受教了！！ 谢谢！！！


42楼 [虚幻3](#) 2013-08-03 11:53发表

 很好,受教了


41楼 [云亦无心](#) 2013-07-26 15:58发表

 讲的很有条理，分析透彻~

40楼 [xiaominglang](#) 2013-07-24 16:33发表

 受教了！！！ 楼主大牛！！！


39楼 [hanfei69882](#) 2013-07-23 08:23发表

 学习啦，顶哥们，用的时候才能发现问题，有问题了再来跟哥们讨论


38楼 [4unreal](#) 2013-07-06 16:08发表

 很赞的文章


37楼 [秘制菜鸟](#) 2013-06-25 09:24发表

 本人菜鸟，想问一下：  
拷贝构造函数这样写行吗？  
//拷贝构造函数  
CExample(const CExample \*C)  
{  
a = C->a;  
}


36楼 [秘制菜鸟](#) 2013-06-25 09:03发表

 为什么要用引用，指针不行吗？

35楼 [MFCclassxiao](#) 2013-06-18 10:25发表

 学习了，思路清晰了很多，谢谢大牛

34楼 [wanhuatong1987](#) 2013-06-14 10:02发表



```
[cpp]
01. Rect(const Rect& r)
02. {
03.     width = r.width;
04.     height = r.height;
05.     p = new int;    // 为新对象重新动态分配空间
06.     *p = *(r.p);
07. }
```

这样写在程序结束的时候调用了两次析构函数，不存在33楼说的内存泄露问题，这是用VS2010调试的结果。

33楼 [MrSimp1e](#) 2013-06-03 15:08发表

深拷贝这里，



```
Rect(const Rect& r)
{
    width = r.width;
    height = r.height;
    p = new int; // 为新对象重新动态分配空间
    *p = *(r.p);
}
```

P重新分配了动态空间，但是原来的空间却没有释放掉。这样就造成了内存泄露吧。

```
Rect(const Rect& r)
{
    width = r.width;
    height = r.height;
    int *pTemp = p; // 将p指针指向的内存地址缓存起来
    p = new int; // 为新对象重新动态分配空间
    *p = *(r.p);
    delete pTemp; // 释放p原来的内存
}
```

只是探讨，没有其他意思。

Re: [Atlas](#) 2013-06-18 09:28发表



引用“bboyfeiyu”的评论：

深拷贝这里，

```
Rect(const Rect& r)
{
    ...
```

这样写的程序中，不会内存泄漏，但是另外一个问题，如果申请空间失败，但之前改变你实例中的部分，因为这里width = r.width; height = r.height; 已经执行了，是不是违反了异常安全性，这个怎么看？

Re: [wanhuatong1987](#) 2013-06-14 10:09发表



回复bboyfeiyu： 博友，你这样写的话会出现“0xC0000005: Access violation reading location 0xffffffff.”错误，这是VS2010调试的结果。其实，文中那样写在退出程序的时候会调用两次析构函数，不会存在内存泄漏的问题。如有不对的地方，请指教。

Re: [MrSimp1e](#) 2013-06-14 12:46发表



回复wanhuatong1987： gcc倒是不会报错，改成这样应该没有问题。  
P重新分配了动态空间，但是原来的空间却没有释放掉。这样就造成了内存泄露吧。

```
Rect(const Rect& r)
{
    width = r.width;
    height = r.height;
    int *pTemp = p; // 将p指针指向的内存地址缓存起来,局部指针变量
    p = new int; // 为新对象重新动态分配空间
    *p = *(r.p);
}
```

Re: [kkkwjx](#) 2014-09-06 12:35发表



回复bboyfeiyu： 我认为不会出现内存泄露。  
首先既然是构造函数，p最初就是一个野值，没有指向分配的内存，所以无需考虑释放原来的内存，直接进行内存分配即可。注意是直接分配不是重新分配。  
反而觉得您的代码有问题，delete了一个野值，可能会出现问题。

32楼 [LTheMiracle](#) 2013-05-10 10:10发表



挺有用的，谢谢！

31楼 蛋疼的 2013-04-22 15:20发表



除非返回值的临时变量是在函数刚进入时先入栈。。

Re: NBHH0329 2014-07-25 16:57发表



回复u010050719: 从汇编上来说确实如此。临时变量XXX生成的汇编语句是更靠前的。

30楼 蛋疼的 2013-04-22 15:16发表



- (1). 先会产生一个临时变量，就叫XXXX吧。
- (2). 然后调用拷贝构造函数把temp的值给XXXX。整个这两个步骤有点像：CExample XXXX(temp);
- (3). 在函数执行到最后先析构temp局部变量。
- (4). 等g\_Fun()执行完后再析构掉XXXX对象。

个人觉得先析构xxxx对象再析构temp局部变量

函数返回值如果大于8字节，就要创建一个临时内存来放临时变量，按照堆栈的先进后出原则，temp应该是先入栈，xxxx这个临时变量（其实只能叫做指针，指向一块存放该类型变量的内存）后入栈，这个指针指向的地址会存放在寄存器中，函数返回时寄存器EAX根据这个地址去找到临时变量的内容，你也许会说那函数返回，该函数的堆栈帧已经销毁，如何返回该临时变量的值，堆栈帧是被销毁了，但是程序不会清空其中的值，临时变量所在内存值依然存在。

Re: heipacker 2014-06-06 23:23发表



回复u010050719: 你自己分析的结论也是先析构的temp再析构XXX啊

Re: 江浙沪小团队移动外包 2013-10-11 15:34发表



回复u010050719: 本人亲测的确是先析构掉了函数内的局部变量，然后程序执行完后才析构函数返回时调用拷贝构造函数产生的临时对象。

Re: 夏雪冬日 2013-08-10 14:46发表



回复u010050719: 分析的到位。我的观点跟你一样！

29楼 蛋疼的 2013-04-22 15:03发表



- (1). 先会产生一个临时变量，就叫XXXX吧。
- (2). 然后调用拷贝构造函数把temp的值给XXXX。整个这两个步骤有点像：CExample XXXX(temp);
- (3). 在函数执行到最后先析构temp局部变量。
- (4). 等g\_Fun()执行完后再析构掉XXXX对象。

第三和第四是不是顺序反了？先析构xxxx对象再析构temp对象

28楼 chaoguo1234 2013-04-18 16:26发表



很好，不错

27楼 chm626905227 2013-04-12 17:28发表



很好的一篇学习拷贝构造函数的文章，顶一个~

26楼 曦花 2013-04-08 13:50发表



为什么要加const？  
为什么要用引用？ 指针行吗？

25楼 hulaquandequan 2013-04-07 16:15发表



真心不错~ 感谢~

24楼 hubi 2013-03-22 16:10发表



棒！！！！！！！！！！

23楼 [jackielzjn](#) 2013-03-14 14:42发表



谢谢楼主，高手就是不一样。顶一个。。。。。。。

22楼 [jouyouwen](#) 2013-03-10 16:51发表



写的很好，受教了，谢谢！

21楼 [FranklinLING](#) 2013-03-08 20:32发表



哈哈，和其它评论有同感：简单、透彻！多谢！！

20楼 [她大哥](#) 2013-03-04 14:51发表



写的太好了

19楼 [waldobear](#) 2013-03-02 18:14发表



很清楚，收获不小。

18楼 [na\\_nalove\\_huahua](#) 2013-02-22 18:25发表



文章写的很棒 解答了我很多疑惑

17楼 [fly2sky](#) 2013-02-21 23:11发表



2中的问题又验证了一下，在vc下就会调用拷贝构造函数，在linux下用g++编译则不会调用拷贝构造函数，看来和编译器有关，文章很给力，学习了

16楼 [fly2sky](#) 2013-02-21 22:48发表



2. 对象以值传递的方式从函数返回 这个给的例子验证了一下，并没有调用到拷贝构造函数，请博主试一下

15楼 [Ritter\\_Liu](#) 2013-02-19 20:03发表



此文写的太好了

14楼 [青茶柠檬](#) 2013-01-30 14:36发表



谢谢楼主，讲的很清楚。转走了

13楼 [skkks](#) 2013-01-20 10:21发表



条理很清晰，喜欢这样的风格，谢谢~~

12楼 [GrimRaider](#) 2013-01-09 11:22发表



I like it! Thank you!

11楼 [windyrain999](#) 2012-12-23 12:19发表



谢谢您，对我很有帮助！！

10楼 [nashouat](#) 2012-12-22 10:07发表



太好了，讲得很通俗易懂，很容易理解，谢谢楼主，真的感谢！现在对拷贝构造函数清晰多了。谢谢！

9楼 [\\_北方的雪\\_](#) 2012-12-19 11:05发表



对（四）中第一个问题也是有疑问：在拷贝构造函数中，形参是对象的引用，在函数体中怎么可以直接用对象调用私有成员呢？

Re: 夏雪冬日 2013-08-10 14:43发表



回复sgs1018: 拷贝构造函数是一种特殊的构造函数。类的成员函数是可以访问该类的private成员的。

8楼 yintangzengli 2012-12-05 09:52发表



写的蛮不错的，例子有对比，真的是让我看进去了的。

7楼 zhengtong0416 2012-11-17 11:40发表



受教！向您学习

6楼 lwbeyond 2012-10-26 13:06发表



共同进步啊

5楼 LinuxMan 2012-10-24 09:16发表



很容易理解，赞一个

4楼 骑着乌龟去看海 2012-10-24 09:00发表



MK

3楼 hanzhijun58 2012-10-15 20:25发表



受教了

2楼 MyLend 2012-10-05 22:01发表



牛啊。。。

1楼 aoshikang 2011-04-21 16:53发表




哥们，你这篇文章真给力！受教了！谢谢。

发表评论

用户名: zyp2524153

评论内容:



提交

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目												
全部主题	Hadoop	AWS	移动游戏	Java	Android	iOS	Swift	智能硬件	Docker	OpenStack		
VPN	Spark	ERP	IE10	Eclipse	CRM	JavaScript	数据库	Ubuntu	NFC	WAP	jQuery	
BI	HTML5	Spring	Apache	.NET	API	HTML	SDK	IIS	Fedora	XML	LBS	Unity
Splashtop	UML	components	Windows Mobile	Rails	QEMU	KDE	Cassandra	CloudStack				

FTC   coremail   OPhone   CouchBase   云计算   iOS6   Rackspace   Web App   SpringSide   Maemo  
Compuware   大数据   aptech   Perl   Tornado   Ruby   Hibernate   ThinkPHP   HBase   Pure   Solr  
Angular   Cloud Foundry   Redis   Scala   Django   Bootstrap

公司简介 | 招贤纳士 | 广告服务 | 银行汇款帐号 | 联系方式 | 版权声明 | 法律顾问 | 问题报告 | 合作伙伴 | 论坛反馈

网站客服   杂志客服   微博客服   webmaster@csdn.net   400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved

