

# WUDAIJUN的博客

懦弱的坚持 轻轻地笑 一路追寻..

CSDN学院讲师招募

Markdown编辑器 轻松写博文

PMBOK第五版精讲视频教程

读文章说感想 获好礼

企业高端研修班培训直通车

## C++ 内存分配(new , operator new)详解

分类：[C/C++](#) [汇编与底层](#)

2013-07-09 14:55

7047人阅读

[评论\(6\)](#)

[收藏](#)

[举报](#)

new

operator new

placement new

c++

STL

目录(?)

[+]

本文主要讲述C++ new运算符和operator new, placement new之间的种种关联，new的底层实现，以及operator new的重载和一些在内存池，STL中的应用。

### 一 new运算符和operator new()：

new：指我们在C++里通常用到的**运算符**，比如A\* a = new A; 对于new来说，有new和::new之分，前者位于std namespace，后者是全局的。operator new()：指对new的重载形式，它是一个**函数**，并不是运算符。对于operator new来说，分为全局重载和类重载，全局重载是void\* ::operator new(size\_t size)，在类中重载形式 void\* A::operator new(size\_t size)。还要注意的**是这里的operator new()完成的操作一般只是分配内存**，事实上系统默认的全局::operator new(size\_t size)也只是调用malloc分配内存，并且返回一个void\*指针。**而构造函数的调用(如果需要)是在new运算符中完成的。**

先简单解释一下new和operator new之间的关系：

关于这两者的关系，我找到一段比较经典的描述（来自于www.cplusplus.com 见参考文献）：

operator new can be called explicitly as a regular function, but in C++, new is an operator with a very specific behavior: An expression with the new operator, first calls function operator new (i.e., this function) with the size of its the object (if needed). Finally, the expression evaluates as a pointer to the appropriate type.

比如我们写如下代码：

A\* a = new A；

我们知道这里分为两步：1.分配内存，2.调用A()构造对象。事实上，分配内存这一操作就是由operator new(size\_t)来完成的，如果类A重载了operator new，那么将调用A::operator new(size\_t)，如果没有重载，就调用::operator new(size\_t)，全局new操作符由C++默认提供。因此前面的两步也就是：1.调用operator new 2.调用构造函数。这里再一次提出来是因为后面关于这两步会有一些变形，在关于placement new那里会讲到。先举个简单例子

**[cpp]** view plain copy print ?

```
01. //平台: Visual Stdio 2008
02. #include<iostream>
03. class A
04. {
05. public:
06.     A()
07.     {
08.         std::cout<<"call A constructor"<<std::endl;
09.     }
10.
11.     ~A()
12.     {
13.         std::cout<<"call A destructor"<<std::endl;
14.     }
15. }
16. int _tmain(int argc, _TCHAR* argv[])
17. {
18.
19.     A* a = new A;
20.     delete a;
21.
22.     system("pause");
```

个人资料



WUDAIJUN

访问：38267次

积分：916

等级：**BLOG > 3**

排名：千里之外

原创：53篇 转载：0篇

译文：0篇 评论：12条

文章搜索

文章分类

- 算法 (15)
- C/C++ (12)
- 汇编与底层 (8)
- Windows (3)
- 小项目 (13)
- STL (3)
- 开源POCO库 (3)
- 图像视频处理 (1)

文章存档

- 2013年11月 (1)
- 2013年10月 (1)
- 2013年09月 (1)
- 2013年08月 (2)
- 2013年07月 (3)

展开

阅读排行

- C++ 内存分配(new, operator new) (7046)
- POCO库 Foundation::Thread (1819)
- 实现pop push min操作 (1701)
- 星形密码探测器 (1024)
- POCO库 Foundation::Thread (986)
- 动态规划\_\_合唱队形问题 (971)

关于字符编码以及%ls w (851)

POCO库 Foundation::Str (849)

HSV色彩空间 (803)

C泛型\_\_数据结构 (714)

评论排行

C++ 内存分配(new, operator new) (6)

POCO库 Foundation::Thread (3)

线程安全几个重要概念 (1)

实现pop push min操作时间复杂 (1)

内存跟踪\_\_\_\_栈溢出 (1)

将邻接表表示的图转换为 (0)

找出无序数组中最小的前 (0)

找出二叉树中所有累加值 (0)

星形密码探测器 (0)

迭代器和迭代器适配器--- (0)

推荐文章

\* 【ShaderToy】开篇

\* FFmpeg源代码简单分析：avio\_open2()

\* 技能树之旅：从模块分离到测试

\* Qt5官方demo解析集36——Wiggly Example

\* Unity3d HDR和Bloom效果（高动态范围图像和泛光）

\* Android的Google官方设计指南

最新评论

POCO库 Foundation::Thread模块 WUDAIJUN: yuhaiyang457288: poco 线程池是不是没有 工作队列。

C++ 内存分配(new, operator new) wangzhe03091252: @WUDAIJUN:void\* ::operator new(size\_t size){ cout<...

C++ 内存分配(new, operator new) WUDAIJUN: 不知道报什么错呢？关于全局operator new 还可以参见陈硕的这篇博客：http://blog...

C++ 内存分配(new, operator new) wangzhe03091252: 全局重载 void\* ::operator new(size\_t size) 编译时出现错误。全局的vo...

C++ 内存分配(new, operator new) xcstyle: 讲的很深刻易懂，定位 new的运用真的可以很强大。

POCO库 Foundation::Thread模块 WUDAIJUN: @hesiyuan4:我并没有在实际开发中用过这个库，这是很久以前看的了。所以抱歉，不能帮上你哈。

内存跟踪\_\_\_\_栈溢出 贾大兵: 很不错！

POCO库 Foundation::Thread模块 苍原狮啸: 请问大神：1 如何在类内 使用poco 线程 调用类的成员函数？ 为什么我每次去调用的使用总...

C++ 内存分配(new, operator new) passion\_wu128: 讲的很好，不过有一点错了：new数组的时候不一定会多分配四个字节。只有存在析构函数的类型才会这样。

实现pop push min操作时间复杂 pcww: 赞一个

```
23.         return 0;
24.     }
```

下面我们跟踪一下A反汇编代码，由于Debug版本反汇编跳转太多，因此此处通过Release版本在A\* a = new A;处设断点反汇编：

在Release版本中，构造函数和析构函数都是直接展开的。

[cpp] view plain copy print ?

01. A\* a = new A;
02. 01301022 push 1 ; 不含数据成员的类占用一字节空间，此处压入sizeof(A)
03. 01301024 call operator new (13013C2h) ; 调用operator new(size\_t size)
04. 01301029 mov esi,eax ; 返回值保存到esi
05. 0130102B add esp,4 ; 平衡栈
06. 0130102E mov dword ptr [esp+8],esi ;
07. 01301032 mov dword ptr [esp+14h],0
08. 0130103A test esi,esi ; 在operator new之后，检查其返回值，如果为空(分配失败)，则不调用A()构造函数
09. 0130103C je wmain+62h (1301062h) ; 为空 跳过构造函数部分
10. 0130103E mov eax,dword ptr [\_\_imp\_std::endl (1302038h)] ; 构造函数内部，输出字符串
11. 01301043 mov ecx,dword ptr [\_\_imp\_std::cout (1302050h)]
12. 01301049 push eax
13. 0130104A push offset string "call A constructor" (1302134h)
14. 0130104F push ecx
15. 01301050 call std::operator<<<std::char\_traits<char> > (13011F0h)
16. 01301055 add esp,8
17. 01301058 mov ecx,eax
18. 0130105A call dword ptr [\_\_imp\_std::basic\_ostream<char,std::char\_traits<char> >::operator<<
19. 01301060 jmp wmain+64h (1301064h) ; 构造完成，跳过下一句
20. 01301062 xor esi,esi ; 将esi置空，这里的esi即为new A的返回值
21. 01301064 mov dword ptr [esp+14h],0FFFFFFFFh
22. delete a;
23. 0130106C test esi,esi ; 检查a是否为空
24. 0130106E je wmain+9Bh (130109Bh) ; 如果为空，跳过析构函数和operator delete
25. 01301070 mov edx,dword ptr [\_\_imp\_std::endl (1302038h)] ; 析构函数 输出字符串
26. 01301076 mov eax,dword ptr [\_\_imp\_std::cout (1302050h)]
27. 0130107B push edx
28. 0130107C push offset string "call A destructor" (1302148h)
29. 01301081 push eax
30. 01301082 call std::operator<<<std::char\_traits<char> > (13011F0h)
31. 01301087 add esp,8
32. 0130108A mov ecx,eax
33. 0130108C call dword ptr [\_\_imp\_std::basic\_ostream<char,std::char\_traits<char> >::operator<<
34. 01301092 push esi ; 压入a
35. 01301093 call operator delete (13013BCh) ; 调用operator delete
36. 01301098 add esp,4
37. 通过反汇编可以看出A\* = new A包含了operator new(sizeof(A))和A()两个步骤(当然，最后还要将值返回到a)
38. delete a包含了~A()和operator delete(a)两个步骤。

## 二 operator new的三种形式：

operator new有三种形式：

- throwing (1) void\* operator new (std::size\_t size) throw (std::bad\_alloc);
- nothrow (2) void\* operator new (std::size\_t size, const std::nothrow\_t& nothrow\_value) throw();
- placement (3) void\* operator new (std::size\_t size, void\* ptr) throw();

(1)(2)的区别仅是是否抛出异常，当分配失败时，前者会抛出bad\_alloc异常，后者返回null，不会抛出异常。它们都分配一个固定大小的连续内存。

用法示例：

A\* a = new A; //调用throwing(1)

A\* a = new(std::nothrow) A; //调用nothrow(2)

(3)是placement new，它也是对operator new的一个重载，定义于<new>中，它多接收一个ptr参数，但它只是简单地返回ptr。其在new.h下的源代码如下：

```
01. #ifndef __PLACEMENT_NEW_INLINE
02. #define __PLACEMENT_NEW_INLINE
03. inline void *__cdecl operator new(size_t, void *_P)
04. {return (_P); }
05. #if _MSC_VER >= 1200
06. inline void __cdecl operator delete(void *, void *)
07. {return; }
08. #endif
09. #endif
```

那么它究竟有什么用呢？事实上，它可以实现在ptr所指地址上构建一个对象(通过调用其构造函数)，这在内存池技术上有广泛应用。

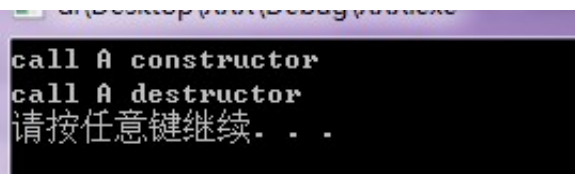
它的调用形式为：

new(p) A(); //也可用A(5)等有参构造函数。

前面说到，new运算符都会调用operator new，而这里的operator new(size\_t, void\*)并没有什么作用，真正起作用的是new运算符的第二个步骤：在p处调用A构造函数。这里的p可以是动态分配的内存，也可以是栈中缓冲，如char buf[100];  
new(buf) A();

我们仍然可以通过一个例子来验证：

```
[cpp] view plain copy print ? ❷
01. #include <iostream>
02. class A
03. {
04. public:
05.     A()
06.     {
07.         std::cout<<"call A constructor"<<std::endl;
08.     }
09.
10.     ~A()
11.     {
12.         std::cout<<"call A destructor"<<std::endl;
13.     }
14. };
15. int _tmain(int argc, _TCHAR* argv[])
16. {
17.
18.     A* p = (A*)::operator new(sizeof(A)); //分配
19.
20.     new(p) A(); //构造
21.
22.     p->~A(); //析构
23.
24.     ::operator delete(p); //释放
25.
26.     system("pause");
27.     return 0;
28. }
```



上面的代码将对象的分配，构造，析构和释放分离开来，这也是new和delete运算符两句就能完成的操作。

先直接运行可以看到程序输出：

再分别注释掉new(a) A();和a->~A();两句，可以看到对应的构造和析构函数将不会被调用。

然后查看反汇编：

平台: Visual Studio 2008 Debug版

```
[plain] view plain copy print ? ❷
01.     A* a = (A*)::operator new(sizeof(A)); //分配
02. 00F9151D push      1
03. 00F9151F call      operator new (0F91208h) ;调用::operator new(size_t size)也就是throwing(1)版本
04. 00F91524 add       esp,4
05. 00F91527 mov       dword ptr [ebp-14h],eax ;返回地址放入[ebp-14h] 即为p
06.
07.     new(a) A(); //构造
08. 00F9152A mov       eax,dword ptr [ebp-14h]
```

```
09.      00F9152D  push      eax
10.      00F9152E  push      1      ;压入p
11.      00F91530  call      operator new (0F91280h);调用operator new(size_t, void* p)即placement(3)版本 只是简单返回p
12.      00F91535  add       esp,8
13.      00F91538  mov       dword ptr [ebp-0E0h],eax ;将p放入[ebp-0E0h]
14.      00F9153E  mov       dword ptr [ebp-4],0
15.      00F91545  cmp       dword ptr [ebp-0E0h],0    ;判断p是否为空
16.      00F9154C  je        wmain+81h (0F91561h)      ;如果为空 跳过构造函数
17.      00F9154E  mov       ecx,dword ptr [ebp-0E0h] ;取出p到ecx
18.      00F91554  call      A::A (0F91285h)           ;调用构造函数 根据_thiscall调用约定 this指针通过ecx寄存器传递
19.      00F91559  mov       dword ptr [ebp-0F4h],eax ;将返回值(this指针)放入[ebp-0F4h]中
20.      00F9155F  jmp       wmain+8Bh (0F9156Bh)      ;跳过下一句
21.      00F91561  mov       dword ptr [ebp-0F4h],0    ;将[ebp-0F4h]置空 当前面判断p为空时执行此语句
22.      00F9156B  mov       ecx,dword ptr [ebp-0F4h] ;[ebp-0F4h]为最终构造完成后的this指针(或者为空) 放入ecx
23.      00F91571  mov       dword ptr [ebp-0ECh],ecx ;又将this放入[ebp-0ECh] 这些都是调试所用
24.      00F91577  mov       dword ptr [ebp-4],0FFFFFFFh
25.
26.      a->~A(); //析构
27.      00F9157E  push      0
28.      00F91580  mov       ecx,dword ptr [ebp-14h] ;从[ebp-14h]中取出p
29.      00F91583  call      A::~`scalar deleting destructor' (0F91041h) ;调用析构函数(跟踪进去比较复杂 如果在Release下, 构造析构函数都是直接展开的)
30.
31.      ::operator delete(a); //释放
32.      00F91588  mov       eax,dword ptr [ebp-14h]    ;将p放入eax
33.      00F9158B  push      eax                      ;压入p
34.      00F9158C  call      operator delete (0F910B9h);调用operator delete(void* )
35.      00F91591  add       esp,4 </span>
```

从反汇编中可以看出，其实operator new调用了两次，只不过每一次调用不同的重载函数，并且placement new的主要作用只是将p放入ecx，并且调用其构造函数。

事实上，在指定地址上构造对象还有另一种方法，即手动调用构造函数：p->A::A();这里要加上A::作用域，否则编译器会报错：

error C2273: “函数样式转换”：位于 “->” 运算符右边时非法

用p->A::A();替换掉new(p) A();仍然能达到同样的效果，反汇编：

```
[plain] view plain copy print ?  ⌂
01.      A* a = (A*)::operator new(sizeof(A)); //分配
02.      010614FE  push      1
03.      01061500  call      operator new (1061208h)
04.      01061505  add       esp,4
05.      01061508  mov       dword ptr [a],eax
06.
07.      //new(a) A();    //构造
08.      a->A::A();
09.      0106150B  mov       ecx,dword ptr [a]
10.      0106150E  call      operator new (1061285h)
11.
12.      a->~A(); //析构
13.      01061513  push      0
14.      01061515  mov       ecx,dword ptr [a]
15.      01061518  call      A::~`scalar deleting destructor' (1061041h)
16.
17.      ::operator delete(a); //释放
18.      0106151D  mov       eax,dword ptr [a]
19.      01061520  push      eax
20.      01061521  call      operator delete (10610B9h)
21.      01061526  add       esp,4
```

比之前的方法更加简洁高效(不需要调用placement new)。不知道手动调用构造函数是否有违C++标准或有什么隐晦，我在其他很多有名的内存池(包括SGI STL alloc)实现上看到都是用的placement new，而不是手动调用构造函数。

### 三 operator new重载：

前面简单提到过 A\* p = new A；所发生的事情：先调用operator new，如果类A重载了operator new，那么就使用该重载版本，否则使用全局版本::operatro new(size\_t size)。那么类中可以重载operator new的哪些版本？全局operator new可以重载吗？全局和类中重载分别会在什么时机调用？

#### 1.在类中重载operator new

上面提到的throwing(1)和nothrow(2)的operator new是可以被重载的，比如：

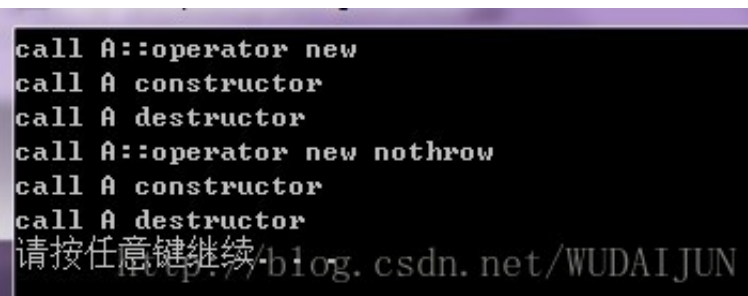
```
[cpp] view plain copy print ?  ⌂
01.      #include <iostream>
```



```

02.     class A
03.     {
04.     public:
05.         A()
06.         {
07.             std::cout<<"call A constructor"<<std::endl;
08.         }
09.
10.         ~A()
11.         {
12.             std::cout<<"call A destructor"<<std::endl;
13.         }
14.         void* operator new(size_t size)
15.         {
16.             std::cout<<"call A::operator new"<<std::endl;
17.             return malloc(size);
18.         }
19.
20.         void* operator new(size_t size, const std::nothrow_t& nothrow_value)
21.         {
22.             std::cout<<"call A::operator new nothrow"<<std::endl;
23.             return malloc(size);
24.         }
25.     };
26.     int _tmain(int argc, _TCHAR* argv[])
27.     {
28.         A* p1 = new A;
29.         delete p1;
30.
31.         A* p2 = new(std::nothrow) A;
32.         delete p2;
33.
34.         system("pause");
35.         return 0;
36.     }

```



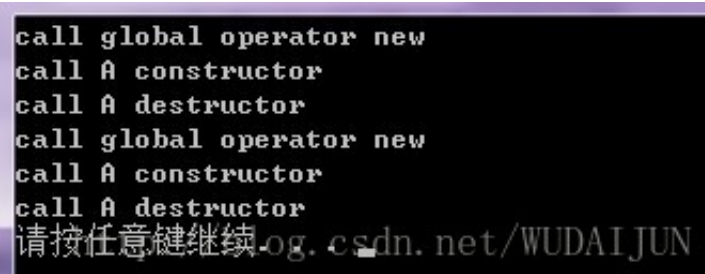
如果类A中没有对operator new的重载，那么new A和new(std::nothrow) A;都将会使用全局operator new(size\_t size)。可将A中两个operator new注释掉，并且在A外添加一个全局operator new重载：

```

[cpp] view plain copy print ?
01.     void* ::operator new(size_t size)
02.     {
03.         std::cout<<"call global operator new"<<std::endl;
04.         return malloc(size);
05.     }

```

程序输出：



注意，这里的重载遵循作用域覆盖原则，即在里向外寻找operator new的重载时，只要找到operator new()函数就不再向外查找，如果参数符合则通过，如果参数不符合则报错，而不管全局是否还有相匹配的函数原型。比如如果这里只将A中operator new(size\_t, const std::nothrow\_t&)删除掉，就会报错：

error C2660: "A::operator new" : 函数不接受 2 个参数。

至于placement new，它本身就是operator new的一个重载，不需也尽量不要对它进行改写，因为它一般是搭配 new(p) A(); 工作的，它的职责只需简单返回指针。

对operator new的重载还可以添加自定义参数，如在类A中添加

```

[cpp] view plain copy print ?
01.     void* operator new(size_t size, int x, int y, int z)
02.     {
03.         std::cout<<"X="<<x<<" Y="<<y<<" Z="<<z<<std::endl;
04.         return malloc(size);
05.     }

```

这种重载看起来没有什么大作用，因为它operator new需要完成的任务只是分配内存，但是通过对这类重载的巧妙应用，可以让它在动态分配内存调试和检测中大展身手。这将在后面operator new重载运用技巧中，展现。

## 2.重载全局operator new

全局operator new的重载和在类中重载并无太大区别，当new A;时，如果类A中没有重载operator new，那么将调用全局operator new函数，如果没有重载全局operator new，最后会调用默认的全局operator new。

## 3.类中operator new和全局operator new的调用时机

前面已经提到了在new时的调用顺序，但是这里提出来的原因是还存在一个全局的new运算符，也就是::new，这个运算符会直接调用全局operator new，并且也会调用构造函数。这可能让人很犯迷糊，只了解即可。这里提到的调用时机都是指通过new运算符调用，没有讨论其他情况，比如主动调用。

## 四 operator new运用技巧和一些实例探索

### 1.operator new重载运用于调试：

前面提到如何operator new的重载是可以有自定义参数的，那么我们如何利用自定义参数获取更多的信息呢，这里一个很有用的做法就是给operator new添加两个参数:char\* file, int line,这两个参数记录new运算符的位置，然后再在new时将文件名和行号传入，这样我们就能在分配内存失败时给出提示：输出文件名和行号。

那么如何获取当前语句所在文件名和行号呢，windows提供两个宏：\_\_FILE\_\_和\_\_LINE\_\_。利用它们可以直接获取到文件名和行号，也就是 new(\_\_FILE\_\_, \_\_LINE\_\_) 由于这些都是不变的，因此可以再定义一个宏：#define new new(\_\_FILE\_\_, \_\_LINE\_\_)。这样我们就只需要定义这个宏，然后重载operator new即可。

源代码如下，这里只是简单输出new的文件名和行号。

```
[cpp] view plain copy print ?
01. //A.h
02. class A
03. {
04. public:
05.     A()
06.     {
07.         std::cout<<"call A constructor"<<std::endl;
08.     }
09.
10.     ~A()
11.     {
12.         std::cout<<"call A destructor"<<std::endl;
13.     }
14.
15.     void* operator new(size_t size, const char* file, int line)
16.     {
17.         std::cout<<"call A::operator new on file:"<<file<<" line:"<<line<<std::endl;
18.         return malloc(size);
19.         return NULL;
20.     }
21.
22. };
23. //Test.cpp
24. #include <iostream>
25. #include "A.h"
26. #define new new(__FILE__, __LINE__)
27.
28. int _tmain(int argc, _TCHAR* argv[])
29. {
30.     A* p1 = new A;
31.     delete p1;
32.
33.     A* p2 = new A;
34.     delete p2;
35.
36.     system("pause");
37.     return 0;
38. }
```

输出：

```
call A::operator new on file:d:\desktop\xxx\xxx\xxx.cpp line:8
call A constructor
call A destructor
call A::operator new on file:d:\desktop\xxx\xxx\xxx.cpp line:11
call A constructor
call A destructor
请按任意键继续. . .
```

<http://blog.csdn.net/WUDAIJUN>

注意：需要将类的声明实现与new的使用隔离开来。并且将类头文件放在宏定义之前。否则在类A中的operator new重载中的new会被宏替换，整个函数就变成了： void\* operator new(\_\_FILE\_\_, \_\_LINE\_\_)(size\_t size, char\* file, int line) 编译器自然会报错。

### 2.内存池优化

operator new的另一个大用处就是内存池优化，内存池的一个常见策略就是分配一次性分配一块大的内存作为内存池(buffer或pool)，然后重复利用该内存块，每次分配都从内存池中取出，释放则将内存块放回内存池。在我们客户端调用的是new运算符，我们可以改写operator new函数，让它从内存池中取出(当内存池不够时，再从系统堆中一次性分配一块大的)，至于构造和析构则在取出的内存上进行，然后再重载operator delete，它将内存块放回内存池。关于内存池和operator new在参考文献中有一篇很好的文章。这里就不累述了。

### 3.STL中的new

在SGI STL源码中,defalloc.h和stl\_construct.h中提供了最简单的空间配置器(allocator)封装，见《STL源码剖析》P48。它将对象的空间分配和构造分离开来，虽然在defalloc.h中仅仅是对::operator new和::operator delete的一层封装，但是它仍然给STL容器提供了更加灵活的接口。SGI STL真正使用的并不是defalloc.h中的分配器，而是stl\_alloc.h中的SGI精心打造的"双层级配置器"，它将内存池技术演绎得淋漓尽致，值得细细琢磨。顺便提一下，在stl\_alloc.h中并没有使用::operator new/delete 而直接使用malloc和free。具体缘由均可参见《STL源码剖析》。

## 五 delete的使用

delete的使用基本和new一致，包括operator delete的重载方式这些都相似，只不过它的参数是void\*，返回值为void。但是有一点需要注意，operator delete的自定义参数重载并不能手动调用。比如

```
[cpp] view plain copy print ?
01. void* operator new(size_t size, int x)
02. {
03.     cout<<" x = "<<x<<endl;
04.     return malloc(size);
05. }
06. void operator delete(void* p, int x)
07. {
08.     cout<<" x = "<<x<<endl;
09.     free(p);
10. }
```

如下调用是无法通过的：

A\* p = new(3) A;//Ok

delete(3) p;//error C2541: “delete”：不能删除不是指针的对象

那么重载operator delete有什么作用？如何调用？事实上以上自定义参数operator delete 只在一种情况下被调用：当new运算符抛出异常时。

可以这样理解，只有在new运算符中，编译器才知道你调用的operator new形式，然后它会调用对应的operator delete。一旦出了new运算符，编译器对于你自定义的new将一无所知，因此它只会按照你指定的delete运算符形式来调用operator delete，而至于为什么不能指定调用自定义delete(也就是只能老老实实delete p)，这个就知道了。

细心观察的话，上面operator new用于调试的例子代码中，由于我们没有给出operator new对应的operator delete。在VS2008下会有如下警告：

warning C4291: “void \*A::operator new(size\_t,const char \*,int)”：未找到匹配的删除运算符；如果初始化引发异常，则不会释放内存

## 六 关于new和内存分配的其他

### 1.set\_new\_handler

还有一些零散的东西没有介绍到，比如set\_new\_handler可以在malloc(需要调用set\_new\_mode(1))或operator new内存分配失败时指定一个入口函数new\_handler，这个函数完成自定义处理(继续尝试分配，抛出异常，或终止程序)，如果new\_handler返回，那么系统将继续尝试分配内存，如果失败，将继续重复调用它，直到内存分配完毕或new\_handler不再返回(抛出异常，终止)。下面这段程序完成这个测试：

```
[cpp] view plain copy print ?
01. #include <iostream>
02. #include <new.h>// 使用_set_new_mode和set_new_handler
03. void nomem_handler()
04. {
05.     std::cout<<"call nomem_handler"<<std::endl;
06. }
07. int main()
08. {
09.     _set_new_mode(1); //使new_handler有效
10.     set_new_handler(nomem_handler);//指定入口函数 函数原型void f();
```

```

11.     std::cout<<"try to alloc 2GBmemory..."<<std::endl;
12.     char* a = (char*)malloc(2*1024*1024*1024);
13.     if(a)
14.         std::cout<<"ok...I got it"<<std::endl;
15.     free(a);
16.     system("pause");
17. }

```

程序运行后会一直输出call nomem\_handler 因为函数里面只是简单输出，返回，系统尝试分配失败后，调用 nomem\_handler函数，由于该函数并没有起到实际作用(让可分配内存增大)，因此返回后系统再次尝试分配失败，再调用 nomem\_handler，循环下去。

在SGI STL中的也有个仿new\_handler函数:oom\_malloc

## 2.new分配数组

A\* p = new A[3];中，会直接调用全局的operator new[](size\_t size)，而不管A中是否有operator new[]的重载。而 delete[]p却会优先调用A::operator delete[](void\*)(如果A中有重载)。另外还要注意的，在operator new[](size\_t size) 中传入的并不是sizeof(A)\*3。而要在对象数组的大小上加上一个额外数据，用于编译器区分对象数组指针和对象指针以及对象数组大小。在VS2008下这个额外数据占4个字节，一个int大小。测试代码如下

```

[cpp] view plain copy print ?
01. //A.h
02. class A
03. {
04. public:
05.     A()
06.     {
07.         std::cout<<"call A constructor"<<std::endl;
08.     }
09.
10.     ~A()
11.     {
12.         std::cout<<"call A destructor"<<std::endl;
13.     }
14.
15.     void* operator new(size_t size)
16.     {
17.         std::cout<<"call A::operator new[] size:"<<size<<std::endl;
18.         return malloc(size);
19.     }
20.     void operator delete[](void* p)
21.     {
22.         std::cout<<"call A::operator delete[]"<<std::endl;
23.         free(p);
24.     }
25.     void operator delete(void* p)
26.     {
27.         free(p);
28.     }
29. };

```

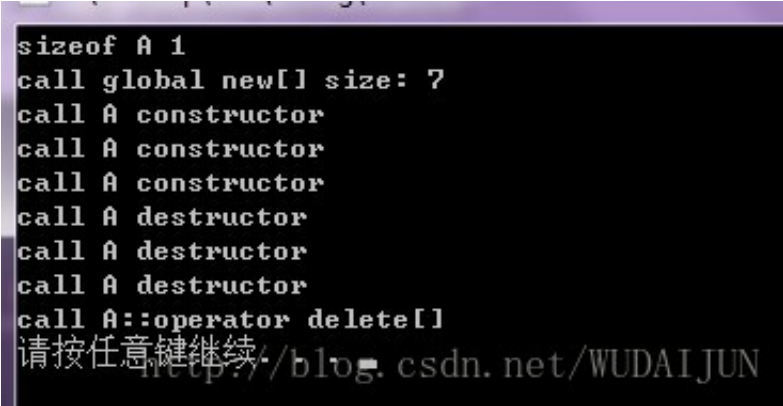
```

[cpp] view plain copy print ?
01. //Test.cpp
02. #include <iostream>
03. #include "A.h"
04.
05. void* operator new[](size_t size)
06. {
07.     std::cout<<"call global new[] size: "<<size<<std::endl;
08.     return malloc(size);
09. }
10.
11. void operator delete[](void* p)
12. {
13.     std::cout<<"call global delete[] "<<std::endl;
14. }
15. int _tmain(int argc, _TCHAR* argv[])
16. {
17.     std::cout<<"sizeof A "<<sizeof(A)<<std::endl;
18.     A* p1 = new A[3];
19.     delete []p1;
20.
21.     system("pause");
22.     return 0;
23. }

```



输出：



简单跟踪了一下：

operator new[]返回的是0x005b668 而最后new运算符返回给p的是0x005b66c。也就是说p就是数组的起始地址，这样程序看到的内存就是线性的，不包括前面的额外数据。

在内存中，可以看到前面的四个字节额外数据是0x00000003 也就是3，代表数组元素个数。后面三个cd是堆在Debug中的默认值(中文的cdcd就是"屯"，栈的初始值为cc，0xcccc中文"烫")。再后面的0xfdfdfdfd应该是堆块的结束标志，前面我有博客专门跟踪过。

注：其实在malloc源码中也有内存池的运用，而且也比较复杂。最近在参考dlmalloc版本和STL空间适配器，真没有想到一个内存分配能涉及这么多的东西。

参考文献：

- 1. <http://www.cplusplus.com/reference/new/operator%20new/?kw=operator> operator new的三种形式
- 2. <http://www.relisoft.com/book/tech/9new.html> c++ operator new重载和内存池技术
- 3. 《STL源码剖析》 空间配置器

转载请注明出处:<http://blog.csdn.net/wudaijun/article/details/9273339>

上一篇 Anagrams by Stack

下一篇 智能指针 auto\_ptr

顶 踩  
4 0

主题推荐

内存分配 c++ visual studio 2008 编译器 源代码

猜你在找

有效使用 Lambda 表达式和 stdfunction	C语言及程序设计提高
moto & google笔试题目-STLC++面试题	基于Unity的游戏开发（下）
C++学习之深入理解虚函数--虚函数表解析	零基础学HTML 5实战开发(第一季)
智能指针的实现及原理	[高安定PMP远程培训]PMP考试习题1000题精讲
手把手教你玩转网络编程模型之完成例程Completion	微信公众平台开发入门

准备好了么？跳吧！

更多职位尽在 CSDN JOB

c++开发工程师	我要跳槽	C/C++开发（中国移动融合通信）	我要跳槽
京品高科信息科技（北京）有限公司	7-15K/月	新媒传信(中国移动融合通信)	15-30K/月
c++高级软件工程师	我要跳槽	C++高级图形工程师	我要跳槽
北京金奔腾汽车科技有限公司	10-15K/月	百猎网	15-35K/月

查看评论

5楼 [WUDAJUN](#) 2015-01-13 23:49发表



引用“[wangzhe03091252](#)”的评论：  
全局重载void\* ::operator new(size\_t size)编译时出现错误。  
全局的v...

不知道报什么错呢？关于全局operator new 还可以参见陈硕的这篇博客：  
<http://blog.csdn.net/solstice/article/details/6198937>

Re: [wangzhe03091252](#) 2015-01-21 11:53发表



回复WUDAJUN： void\* ::operator new(size\_t size)  
{  
cout<<"global opertor new"<<endl;  
return malloc(size);  
}  
会出现 error: explicit qualification in declaration of ‘void\* operator new(size\_t)’  
void\* ::operator new(size\_t size)

但是如果把前面全局作用域的冒号去掉就好了。

4楼 [wangzhe03091252](#) 2015-01-05 22:20发表



全局重载void\* ::operator new(size\_t size)编译时出现错误。  
全局的void\* ::operator new(size\_t size)既然本身就存在，那么又重新定义一个一模一样的函数，就不是重载了吧！应该出错才对啊？还请赐教。

3楼 [xcstyle](#) 2014-12-05 11:33发表



讲的很深刻易懂，定位new的运用真的可以很强大。

2楼 [passion\\_wu128](#) 2014-09-01 23:41发表



讲得很好，不过有一点错了：**new**数组的时候不一定会多分配四个字节。只有存在析构函数的类型才会这样。

1楼 [zhang\\_int\\_int](#) 2014-07-08 17:19发表



讲的很详细，赞一个

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 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

