

# 阿凡卢

博客园 首页 新随笔 联系 订阅 管理

随笔- 124 文章- 0 评论- 165

## C++中的new、operator new与placement new

### C++中的new/delete与operator new/operator delete

new operator/delete operator就是new和delete操作符，而operator new/operator delete是函数。

new operator

- (1) 调用operator new分配足够的空间，并调用相关对象的构造函数
- (2) 不可以被重载

operator new

- (1) 只分配所要求的空间，不调用相关对象的构造函数。当无法满足所要求分配的空间时，则
  - >如果有new\_handler，则调用new\_handler，否则
  - >如果没要求不抛出异常（以nothrow参数表达），则执行bad\_alloc异常，否则
  - >返回0
- (2) 可以被重载
- (3) 重载时，返回类型必须声明为void\*
- (4) 重载时，第一个参数类型必须为表达要求分配空间的大小（字节），类型为size\_t
- (5) 重载时，可以带其它参数

delete 与 delete operator类似

按 Ctrl+C 复制代码

## 公告

昵称：阿凡卢

园龄：2年6个月

粉丝：187

关注：28

+加关注

<	2012年8月						>
日	一	二	三	四	五	六	
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	29	30	31	1	
2	3	4	5	6	7	8	

## 搜索

```
class X
{
public:
    X() { cout<<"constructor of X"<<endl; }
    ~X() { cout<<"destructor of X"<<endl;}

    void* operator new(size_t size,string str)
    {
        cout<<"operator new size "<<size<<" with string "<<str<<endl;
        return ::operator new(size);
    }

    void operator delete(void* pointee)
    {
        cout<<"operator delete"<<endl;
        ::operator delete(pointee);
    }
private:
    int num;
};

int main()
{
    X *px = new("A new class") X;
    delete px;

    return 0;
}
```

按 Ctrl+C 复制代码

X\* px = new X; //该行代码中的new为new operator，它将调用类X中的operator new，为该类的对象分配空间，然后调用当前实例的构造函数。

delete px; //该行代码中的delete为delete operator，它将调用该实例的析构函数，然后调用类X中的operator delete，以释放该实例占用的空间。

new operator与delete operator的行为是不能够也不应该被改变，这是C++标准作出的承诺。而operator

## 常用链接

[我的随笔](#)

[我的评论](#)

[我的参与](#)

[最新评论](#)

[我的标签](#)

[更多链接](#)

## 随笔分类(125)

[Algorithm\(25\)](#)

[Big Data\(4\)](#)

[C#\(11\)](#)

[C/C++\(19\)](#)

[Data Structure\(15\)](#)

[DataBase\(1\)](#)

[GIS\(11\)](#)

[Java\(8\)](#)

[Linux\(3\)](#)

[Network\(7\)](#)

[Program Design\(4\)](#)

[Python\(2\)](#)

[Research\(6\)](#)

[Web\(2\)](#)

[Windows\(7\)](#)

## 随笔档案(124)

[2015年2月 \(1\)](#)

[2015年1月 \(1\)](#)

[2014年12月 \(2\)](#)

new与operator delete和C语言中的malloc与free对应，只负责分配及释放空间。但使用operator new分配的空间必须使用operator delete来释放，而不能使用free，因为它们对内存使用的登记方式不同。反过来亦是一样。你可以重载operator new和operator delete以实现对内存管理的不同要求，但你不能重载new operator或delete operator以改变它们的行为。

为什么有必要写自己的operator new和operator delete？

答案通常是：为了效率。缺省的operator new和operator delete具有非常好的通用性，它的这种灵活性也使得在某些特定的场合下，可以进一步改善它的性能。尤其在那些需要动态分配大量的但很小的对象的应用程序里，情况更是如此。具体可参考《Effective C++》中的第二章内存管理。

### Placement new的含义

placement new 是重载operator new 的一个标准、全局的版本，它不能够被自定义的版本代替（不像普通版本的operator new和operator delete能够被替换）。

```
void *operator new( size_t, void * p ) throw() { return p; }
```

placement new的执行忽略了size\_t参数，只返还第二个参数。其结果是允许用户把一个对象放到一个特定的地方，达到调用构造函数的效果。和其他普通的new不同的是，它在括号里多了另外一个参数。比如：

```
Widget * p = new Widget;           //ordinary new
```

```
pi = new (ptr) int; pi = new (ptr) int; //placement new
```

括号里的参数ptr是一个指针，它指向一个内存缓冲器，placement new将在这个缓冲器上分配一个对象。Placement new的返回值是这个被构造对象的地址(比如括号中的传递参数)。placement new主要适用于：在对时间要求非常高的应用程序中，因为这些程序分配的时间是确定的；长时间运行而不被打断的程序；以及执行一个垃圾收集器 (garbage collector)。

### new 、operator new 和 placement new 区别

(1) new：不能被重载，其行为总是一致的。它先调用operator new分配内存，然后调用构造函数初始化那段内存。

new 操作符的执行过程：

1. 调用operator new分配内存；
2. 调用构造函数生成类对象；

2014年11月 (1)  
2014年10月 (3)  
2014年9月 (3)  
2014年8月 (1)  
2014年7月 (3)  
2014年6月 (3)  
2014年5月 (2)  
2014年4月 (1)  
2014年3月 (3)  
2014年1月 (1)  
2013年12月 (6)  
2013年11月 (2)  
2013年10月 (4)  
2013年9月 (4)  
2013年8月 (2)  
2013年7月 (2)  
2013年6月 (1)  
2013年5月 (4)  
2013年4月 (3)  
2012年12月 (4)  
2012年11月 (10)  
2012年10月 (14)  
2012年9月 (15)  
2012年8月 (28)

## 友情链接

IBM developerworks  
UC技术博客  
百度技术博客  
酷壳  
奇虎360技术博客  
淘宝核心系统团队博客

### 3. 返回相应指针。

(2) operator new: 要实现不同的内存分配行为, 应该重载operator new, 而不是new。

operator new就像operator + 一样, 是可以重载的。如果类中没有重载operator new, 那么调用的就是全局的::operator new来完成堆的分配。同理, operator new[], operator delete、operator delete[]也是可以重载的。

(3) placement new: 只是operator new重载的一个版本。它并不分配内存, 只是返回指向已经分配好的某段内存的一个指针。因此不能删除它, 但需要调用对象的析构函数。

如果你想在已经分配的内存中创建一个对象, 使用new时行不通的。也就是说placement new允许你在一个已经分配好的内存中(栈或者堆中)构造一个新的对象。原型中void\* p实际上就是指向一个已经分配好的内存缓冲区的首地址。

### Placement new 存在的理由

#### 1. 用placement new 解决buffer的问题

问题描述: 用new分配的数组缓冲时, 由于调用了默认构造函数, 因此执行效率上不佳。若没有默认构造函数则会发生编译时错误。如果你想在预分配的内存上创建对象, 用缺省的new操作符是行不通的。要解决这个问题, 你可以用placement new构造。它允许你构造一个新对象到预分配的内存上。

#### 2. 增大时空效率的问题

使用new操作符分配内存需要在堆中查找足够大的剩余空间, 显然这个操作速度是很慢的, 而且有可能出现无法分配内存的异常(空间不够)。placement new就可以解决这个问题。我们构造对象都是在一个预先准备好了的内存缓冲区中进行, 不需要查找内存, 内存分配的时间是常数; 而且不会出现在程序运行中途出现内存不足的异常。所以, placement new非常适合那些对时间要求比较高, 长时间运行不希望被打断的应用程序。

### Placement new使用步骤

在很多情况下, placement new的使用方法和其他普通的new有所不同。这里提供了它的使用步骤。

#### 第一步 缓存提前分配

有三种方式:

淘宝搜索技术博客

腾讯ISUX

## 最新评论

#### 1. Re:基于netty-socketio的w...

@高天乐socket.io会自动重连, 对于推送的消息, client处理后可以发送ack确认, 如果Server没有收到ack, 可以再次推送。...

-----  
--阿凡卢

#### 2. Re:基于netty-socketio的w...

如果在网络不稳定的情况下, 会不会出问题?

-----  
--高天乐

#### 3. Re:基于Netty与RabbitMQ...

@liaojia1因为设备是第三方的, 消息协议要解析, Netty是个中转, 图中的RabbitMQ Server是消息中心服务器, RabbitMQ是有中心服务器的, 而不像一般的MQ有的是点对点的。...

-----  
--阿凡卢

#### 4. Re:基于Netty与RabbitMQ...

好奇, 为什么不直接给MQ发消息, 要通过一层Netty?

-----  
--liaojia1

#### 5. Re:Dynamic Time Warp in...

博主厉害! 就是想请教一下, 你的C++代码实现dtwrecoge.cpp文件中, 有一部分不懂, 就是“DTWTemplate, 进行建立模板的工作”这部分, 感觉只需要上面一部分就够了, 为什么还有模板部分的工.....

-----  
--龙腾虎跃时

1.为了保证通过placement new使用的缓存区的memory alignment(内存队列)正确准备,使用普通的new来分配它:在堆上进行分配

```
class Task ;
```

```
char * buff = new [sizeof(Task)]; //分配内存
```

(请注意auto或者static内存并非都正确地为一个对象类型排列,所以,你将不能以placement new使用它们。)

2.在栈上进行分配

```
class Task ;
```

```
char buf[N*sizeof(Task)]; //分配内存
```

3.还有一种方式,就是直接通过地址来使用。(必须是有意义的地址)

```
void* buf = reinterpret_cast<void*> (0xF00F);
```

#### 第二步:对象的分配

在刚才已分配的缓存区调用placement new来构造一个对象。

```
Task *ptask = new (buf) Task
```

#### 第三步:使用

按照普通方式使用分配的对象:

```
ptask->memberfunction();
```

```
ptask-> member;
```

```
//...
```

#### 第四步:对象的析构

一旦你使用完这个对象,你必须调用它的析构函数来毁灭它。按照下面的方式调用析构函数:

```
ptask->~Task(); //调用外在的析构函数
```

#### 第五步:释放

你可以反复利用缓存并给它分配一个新的对象(重复步骤2, 3, 4)如果你不打算再次使用这个缓存,你可以象这样释放它: delete [] buf;

## 阅读排行榜

1. C++中数字与字符串之间...
2. NPOI读写Excel(8292)
3. hashmap的C++实现(6980)
4. 网络编程socket基本API详...
5. BP人工神经网络的介绍与...
6. C# byte数组与Image的相...
7. ArcGIS中的三种查询(4617)
8. Windows线程的创建与终...
9. windows下编译及使用libe...
10. 使用WinPcap抓包分析网...

## 评论排行榜

1. GMap.Net开发之自定义M...
2. GMap.Net开发之在WinFo...
3. GMap.Net开发之地址解...
4. windows下编译及使用libe...
5. 基于GMap.Net的地图解...
6. NPOI读写Excel(9)
7. hashmap的C++实现(7)
8. 内部排序法总结(6)
9. 字符串笔试题(5)
10. 高斯混合模型GMM的C+...

## 推荐排行榜

1. C#多线程编程(9)
2. C# byte数组与Image的相...
3. GMap.Net开发之在WinFo...
4. 基于GMap.Net的地图解...
5. 基于Libevent的HTTP Ser...

跳过任何步骤就可能导致运行时间的崩溃，内存泄露，以及其它意想不到的情况。如果你确实需要使用placement new，请认真遵循以上的步骤。

按 Ctrl+C 复制代码

```
class X
{
public:
    X() { cout<<"constructor of X"<<endl; }
    ~X() { cout<<"destructor of X"<<endl;}

    void SetNum(int n)
    {
        num = n;
    }

    int GetNum()
    {
        return num;
    }

private:
    int num;
};

int main()
{
    char* buf = new char[sizeof(X)];
    X *px = new(buf) X;
    px->SetNum(10);
    cout<<px->GetNum()<<endl;
    px->~X();
    delete []buf;

    return 0;
}
```

按 Ctrl+C 复制代码

- 6. 内部排序法总结(4)
- 7. 基于Netty与RabbitMQ的...
- 8. NPOI读写Excel(4)
- 9. C# IP地址与整数之间的转...
- 10. C++中的new、operator ...

作者: [阿凡卢](#)

出处: <http://www.cnblogs.com/luxiaoxun/>

本文版权归作者和博客园共有, 欢迎转载, 但未经作者同意必须保留此段声明, 且在文章页面明显位置给出原文连接, 否则保留追究法律责任的权利。

分类: [C/C++](#)

标签: [C++](#)

绿色通道:

[好文要顶](#)

[关注我](#)

[收藏该文](#)

[与我联系](#)



阿凡卢

关注 - 28

粉丝 - 187

[+加关注](#)

3

0

(请您对文章做出评价)

« 上一篇: [C++的重载 \( overload \) 与重写 \( override \)](#)

» 下一篇: [类的operator new与operator delete的重载](#)

posted @ 2012-08-10 14:26 阿凡卢 阅读(3246) 评论(1) 编辑 收藏

## 评论列表

#1楼

2013-08-03 14:18 果冻想

谢谢楼主的分享, 学习了。

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

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](#)。

**【免费课程】案例：PHP实现验证码制作**

**【推荐】**50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库  
融云，免费为你的App加入IM功能——让你的App“聊”起来！！



#### 最新IT新闻:

- 史玉柱发微博呼吁 公开招聘民生银行新行长
  - 苹果给开发者提供了OS X 10.10.3 Beta 2
  - 融资的艺术：用这6招在投资人面前保持吸引力
  - Microsoft Band重大更新：骑车模式、屏幕键盘、Band SDK等
  - 智能手表元年到来 苹果手表能否引领未来？
- » 更多新闻...



#### 最新知识库文章:

- HHVM 是如何提升 PHP 性能的？
- Web API设计方法论



2015年2月24日

C++中的new、operator new与placement new - 阿凡卢 - 博客园

- Bitmap的秘密
- 我该如何向非技术人解释SQL注入？
- 使用2-3法则设计分布式数据访问层
- » 更多知识库文章...

---

Copyright ©2015 阿凡卢