new expression、operator new 和 placement new——三个“妞（new）”的故事（3）

placement operator new

placement operator new 用来在指定地址上构造对象，要注意的是，它并不分配内存，仅仅是 对指定地址调用构造函数。其调用方式如下：

```
point *pt=new(p) point3d;
```

观其名字可知，它是 operator new 的一个重载版本。它的实现方式异常简单，传回一个指针即 可：

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

不必要惊讶于它的简单，《深度探索 C++对象模型》中 Lippman 告诉我们，它有另一半重要的工 作是被扩充而来。我在想，扩充一个类中定义的 placement operator new 还好说，但是要如何 扩充一个库中提供的 placement operator new 呢？毕竟它要放之四海而皆准，我原以为这其中 有什么高超的技巧。后来我则坚信根本就没有什么扩充，placement operator new 也并不强 大。

我先明确调用了 placement operator new ：

```
point *pt=(point*)operator new(sizeof(point), p);
```

如我所料，输出结果显示（我在 `point` 的默认构造函数和 placement operator new 中间各输 出一句不同的话），此时 point 的默认构造函数并不会被调用。然后我通过 new expression 的方式来间接调用 placement operator new：

```
point *pt=new(p) point();
```

这个时候 point 的默认的构造函数被调用了。可见 placement operator new 并没有什么奇特 的地方，它与一般的 operator new 不同处在于，它不会申请内存。它也不会在指定的地址调用 构造函数，而调用构造函数的的全部原因在于 new expression 总是先调用

一个匹配参数的 operator new 然后再调用指定类型的匹配参数的构造函数，而说到底 placement operator new 也是一个 operator new。

通过一个 placement operator new 构建的一个对象，如果你使用 delete 来撤销对象，那么其内 存也被回收，如果想保存内存而析构对象，好的办法是显示调用其析构函数。

看一份代码：

```
struct Base { int j; virtual void f(); };struct Derived : Base { void f(); };void
fooBar() {
    Base b;
    b.f(); // Base::f() invoked
    b.~Base();
    new ( &b ) Derived; // 1
    b.f(); // which f() invoked?  }
```

上述两个类的大小相同，因此将 Derived 对象放在 Base 对象中是安全的，但是在最后一句代码 中 b.f()调用的是哪一个类的 f()。答案是 Base::f() 的。虽然此时 b 中存储的实际上是一个 Derived 对象，但是，通过一个对象来调用虚函数，将被静态决议出来，虚函数机制不会被启用。

参考：Lippman 的两本书《深度探索 C++对象模型》和《C++ Primer》。