39 The NEW Keyword in C++

实际上你编写C++的时候，你需要关心内存、性能和优化等问题，不然用其他语言也许更好。

根据你所写的类型，以字节为单位决定了要分配的内存大小。

比如`new int`，它会请求分配4个字节的内存。
现在我们需要找到一个包含4个字节的连续内存块，虽然很好找很快就能分配，但是它仍然要在一行内存中找到一个4字节的内存地址。找到之后，它就会返回一个指向那个内存地址的指针，这样你就可以开始使用你的数据了，存储或者访问，或读或写都可以。

分配内存时不会像激光扫描一样连续地搜索内存，而是有一个叫*free list*(空闲列表)的东西，它会维护那些有空闲字节的地址。

```C++
int a = 2;
int* b = new int;
```

除了像平常创建整型一样，我们还可以通过*new*关键字在*heap*上创建来选择*dynamic memory allocatation*(动态分配内存)

```C++
int* b = new int[50]; // 200  bytes
Entity* e = new Entity();
```

这就是*new*关键字的基本使用方法。
在这里它不仅分配了空间，还调用了构造函数。

去查看`new`的定义：

```C
void* __CRTDECL operator new(
    size_t _Size
);
```

实际上只是个函数，`_Size`是它分配的内存大小，返回的是一个void指针。
可以去复习一下16 POINTERS in C++

通常，调用*new*关键字会调用底层的C函数*malloc*，它是用来分配内存的。
`malloc()`的实际作用是，传入一个`size`，也就是我们需要多少个字节，然后返回一个`void指针`

```C++
malloc(50);
```

这就是这个函数所做的，所以这里我们的代码就相当于

```C++
//  (1)  Entity* e = new Entity();
Entity* e = (Entity*)malloc(sizeof(Entity)); // (2)
```

这两行代码之间仅有的区别就是，(1) 调用了Entity的构造函数。而 (2) 仅仅是分配了内存，然后返回给我们一个指向那个内存的指针，没有调用构造函数。在C++里你不应该像 (2) 这样分配内存，在某些情况下你可能想要那么做，以后再说。

`delete`也被称作*destructor*，见26 Destructors in C++

`delete`也是一个操作符，它只是一个常规函数，调用了C函数`free()`,*free*可以释放*malloc*申请的内存。

```C++
delete e;
free(e);
```

这很重要，因为当我们使用*new*关键字的时候，内存没有被释放，没有被标记为*free*（空闲），它就不会被放回*free list*。因此我们再调用*new*时，这些内存也就不能再被分配，直到我们调用`delete`，我们必须手动释放它

注意事项

1. 如果我们用了`new[]`，那同样应该使用`delete[]`，因为这个操作符定义就是这样的。

```C++
int* b = new int[50];

delete[] b;
```

2. new支持一种叫*placement new*的用法，这决定了它的内存来自哪里。

```C++
int* b = new int[50];
Entity* e = new(b) Entity();
```

细节以后再讲，这里只是展示它的语法。

这里将 `Entity` 对象构造在已分配的内存地址 `b` 上，而不是使用默认的内存分配器。这样可以在指定的内存位置创建对象。这行代码在 `b` 指针指向的内存位置上构造了一个 `Entity` 对象，并返回指向该对象的指针，并将其赋给了 `e` 指针。