

## 问题

C++中释放内存的方式有 delete 和 delete[] 两种，它们的区别是什么呢？

## 区别

在C++ Primer的书中提到，delete用来释放new申请的动态内存，delete[]用来释放由new[]申请的动态内存，也就是说：

- delete释放new分配的**单个对象指针**指向的内存
- delete[]释放new分配的**对象数组指针**指向的内存

直接这样理解对吗？先看看下面这段代码：

```
1 int *a = new int[10];
2 delete a;           //方式1
3 delete [] a;       //方式2
```

事实上，上面两种释放内存的方式都是正确的，方式1并不会造成内存泄漏。这是为什么呢？因为在申请动态内存的时候，一般有两种情况：基本数据类型的分配和自定义数据类型的分配，两者在释放内存的时候略有不同。

## 基本数据类型

对于像 **int/char/long** 等等这些基本数据类型，使用new分配的不管是数组还是非数组形式的内存空间，delete和delete[]都可以正常释放，不会存在内存泄漏。**原因是：分配这些基本数据类型时，其内存大小已经确定，系统可以记忆并且进行管理，在析构时不会调用析构函数，它通过指针可以直接获取实际分配的内存空间。**

```
1 int *a = new int[10];
2 delete a;           //正确
3 delete [] a;       //正确
```

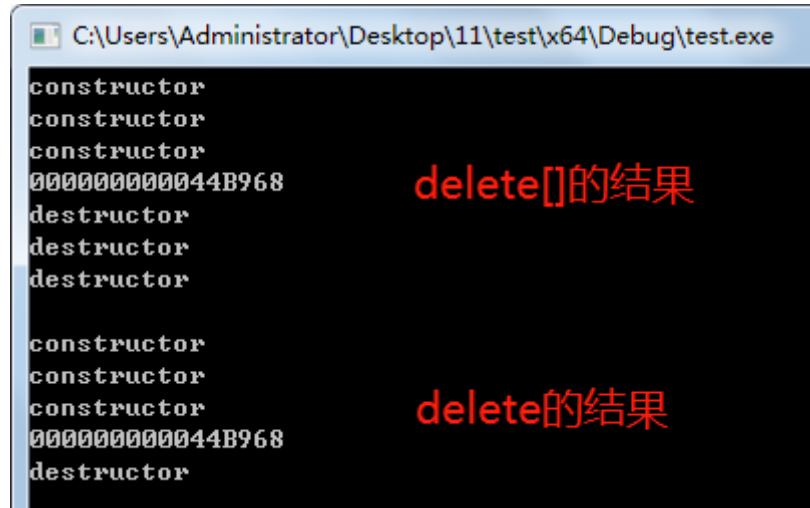
## 自定义数据类型

对于自定义的Class类，delete和delete[]会有一定的差异。先看一段代码示例：

```
1 class T {
2 public:
3     T() { cout << "constructor" << endl; }
4     ~T() { cout << "destructor" << endl; }
5 };
6
7 int main() {
8     T* p1 = new T[3];           //数组中包含了3个类对象
9     cout << hex << p1 << endl; //输出P1的地址
10    delete[] p1;
11
12    cout << endl;
13 }
```

```
14     T* p2 = new T[3];
15     cout << p2 << endl;           //输出P2的地址
16     delete p2;
17
18     return 0;
19 }
```

上面程序的运行结果：



```
C:\Users\Administrator\Desktop\11\test\x64\Debug\test.exe
constructor
constructor
constructor
000000000044B968
destructor
destructor
destructor

constructor
constructor
constructor
000000000044B968
destructor
```

可以看到，`delete[]`在释放内存的时候，调用了3次析构函数，也就是说，`delete[]`会调用析构函数对数组的所有对象进行析构；而`delete`只调用了一次析构函数，即数组中第一个对象的析构函数，而数组中剩下的对象的析构函数没有被调用，因此造成了内存泄漏。

## 总结

- 对于基本数据类型：

不管释放的是单个对象还是数组对象，使用`delete`和`delete[]`释放内存的效果相同。

- 对于自定义数据类型：

释放数组对象时，使用`delete`时只会调用第一个对象的析构函数，可能会造成内存泄漏；而使用`delete[]`时会逐个调用析构函数来释放数组的所有对象。

## 参考资料

[delete 和 delete\[\] 的真正区别](#)

[C++中的delete和delete\[\]的区别](#)