## C++告诉我们

在回收用 new 分配的单个对象的内存空间的时候用 delete，回收用 new[] 分配的一组对象的内存空间的时候用 delete[]。  
关于 new[] 和 delete[]，其中又分为两种情况：(1) 为基本数据类型分配和回收空间；(2) 为自定义类型分配和回收空间。

## 请看下面的程序。

|  |  |
| --- | --- |
|  | #include <iostream>;  using namespace std;    class T {  public:    T() { cout << "constructor" << endl; }    ~T() { cout << "destructor" << endl; }  };    int main()  {    const int NUM = 3;      T\* p1 = new T[NUM];    cout << hex << p1 << endl;    //  delete[] p1;    delete p1;      T\* p2 = new T[NUM];    cout << p2 << endl;    delete[] p2;  } |

大家可以自己运行这个程序，看一看 delete p1 和 delete[] p1 的不同结果，我就不在这里贴运行结果了。

## 从运行结果中我们可以看出，

delete p1 在回收空间的过程中，只有 p1[0] 这个对象调用了析构函数，其它对象如 p1[1]、p1[2] 等都没有调用自身的析构函数，这就是问题的症结所在。

如果用 delete[]，则在回收空间之前所有对象都会首先调用自己的析构函数。  
基本类型的对象没有析构函数，所以回收基本类型组成的数组空间用 delete 和 delete[] 都是应该可以的；

但是对于类对象数组，只能用 delete[]。对于 new 的单个对象，只能用 delete 不能用 delete[] 回收空间。  
    所以一个简单的使用原则就是：new 和 delete、new[] 和 delete[] 对应使用。