# C++动态内存分配研究

王金玲[1]，柴万东[2]

(1.赤峰学院　计算机科学与技术系，内蒙古　赤峰　024000;2.赤峰学院　物理与电子信息工程系)

摘　要：本文介绍了 C++ 中内存的分配方式及动态内存分配中常见的内存错误及处理方法.

关键词：内存；new；delete；内存泄露

中图分类号：TP312　　文献标识码：A　　文章编号：1673-260X(2009)04-0019-02

C++ 是面向对象的程序设计语言，它继承了 C 语言的优点，又增加了面向对象的新特征，是受到广大程序设计者欢迎的开发工具.但在 C++ 的使用过程中，关于内存的使用和管理是程序设计者感到非常神秘而又容易出错的地方.本文对 C++ 的内存分配方式，new 和 delete 运算符的使用及动态内存分配中常见的错误和处理方法进行了分析和研究.

## 1　内存分配方式

C++ 中的内存分配方式有三种：

1.1　从静态存储区域分配.在程序编译的时候就已经分配好内存，这块内存在程序的整个运行期间都存在.例如全局变量，static 变量.

1.2　在栈上分配内存，也称为自动存储.在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放.栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限.

1.3　从堆上分配，亦称动态内存分配.这种堆对象被创建在内存一些空闲的存储单元中，程序在运行的时候用 malloc 或 new 申请任意大小的内存，程序设计者自己负责在何时用 free 或 delete 释放内存.动态内存的生存期由程序设计者决定，使用非常灵活，但问题也最多.

## 2　动态内存分配运算符

在 C 语言中使用 malloc/free 函数进行动态内存的分配与释放.C++ 为了兼容 C，也保留了这两个函数. 但对于非内部数据类型的对象而言，光用 malloc/free 无法满足动态对象的要求. 对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数. 由于 malloc/free 是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于 malloc/free.

因此 C++ 语言提供了一个能完成动态内存分配和初始化工作的运算符 new，以及一个能完成清理与释放内存工作的运算符 delete.

### 2.1　运算符 new 的用法

new< 类型说明符 >(< 初始值列表 >)　　(1)

new< 类型说明符 >[< 算术表达式 >]　　(2)

式 (1) 表明在堆中建立一个由 < 类型说明符 > 给定的类型的对象，并且由括号中的 < 初始值列表 > 给出被创建对象的初始值. 如果省去括号和括号中的初始值，则被创建的对象选用缺省值.式 (2) 表明创建一个由 < 类型说明符 > 给定的类型的对象数组.

使用 new 运算符创建对象时，它可以根据其参数来选择适当的构造函数，它不用 sizeof 来计算对象所占的字节数，可以自动计算其大小. 如果 new 运算符不能分配到所需要的内存，它将返回 0，这时的指针为空指针.使用式 (2) 创建数组时，不能为该数组指定初始值，其初始值为缺省值.

### 2.2　运算符 delete 的用法

delete< 指针名 >　　(3)

delete[]< 指针名 >　　(4)

式 (3) 表明释放使用 new 运算符所创建的堆对象的存储空间.式 (4) 用于释放对象数组的存储空间.此时指针名前只用一对方括号，并且不管所删除数组的维数，忽略方括号内的任何数字.

## 3　常见的内存错误及其对策

在进行内存分配时，发生内存错误是件非常麻烦的事情.编译器不能自动发现这些错误，通常是

基金项目：赤峰学院科学研究基金资助(ZRYB200603)

在程序运行时才能捕捉到.而这些错误大多没有明显的症状,时隐时现,增加了改错的难度.

常见的内存错误主要集中在以下几方面:

3.1 内存分配未成功,却使用了它

刚学习 C++ 的学生常犯这种错误,因为他们没有意识到内存分配会不成功. 解决办法很简单,那就是在使用内存之前检查指针是否为 NULL.如果是用 malloc 或 new 来申请内存,应该用 if(p==NULL)或 if(p!=NULL)进行防错处理.

3.2 内存分配虽然成功,但是尚未初始化就引用它

犯这种错误主要有两个原因:一是没有初始化的观念;二是误以为内存的缺省初值全为零,导致引用初值错误(例如数组).所以无论用何种方式创建数组,都别忘了赋初值,即便是赋零值也不可省略,不要嫌麻烦.

3.3 内存分配成功并且已经初始化,但操作越过了内存的边界

例如在使用数组时经常发生下标"多 1"或者"少 1"的操作.特别是在 for 循环语句中,循环次数很容易搞错,导致数组操作越界.

3.4 忘记了释放内存,造成内存泄露

内存泄露是指程序从堆中分配的内存块,使用完后,程序没有调用 delete 释放该内存块,或者没有正确的释放该内存块,这样,这块内存就不能被再次使用,称之为内存泄漏.含有这种错误的函数每被调用一次就丢失一块内存.刚开始时系统的内存充足,你看不到错误,接着运行速度会逐渐变慢.但最后程序会突然死掉,系统出现提示:内存耗尽.

3.5 释放了内存却继续使用它

产生这种情况的原因有以下几种:(1) 程序中的对象调用关系过于复杂,实在难以搞清楚某个对象究竟是否已经释放了内存,此时应该重新设计数据结构,从根本上解决对象管理的混乱局面. (2) 函数的 return 语句写错了,注意不要返回指向"栈内存"的"指针"或者"引用",因为该内存在函数体结束时被自动销毁. (3) 使用 free 或 delete 释放了内存后,没有将指针设置为 NULL. 导致产生"指针悬空".

4 合理使用 new 和 delete 的几点建议

针对上文介绍的 3.4 和 3.5 情况在使用 new 和 delete 运算符时有如下几点建议,可以大大减少内存泄露和指针悬空的情况.

4.1 动态内存的申请与释放必须配对

程序中 new 与 delete 的使用次数一定要相同,否则肯定有错误.

4.2 对应的 new 和 delete 要采用相同的形式

new 的执行过程是先分配内存,然后,为被分配的内存调用一个或多个构造函数.delete 的执行过程是先为将被释放的内存调用一个或多个析构函数,然后,释放内存.对于 delete 来说要分辨被释放的指针指向的是单个对象呢,还是对象数组?如果在用 delete 时没用括号,delete 就会认为释放的是单个对象,否则,它就会认为释放的是一个数组.

所以,解决这类问题的规则是:如果你调用 new 时用了[],调用 delete 时也要用[].如果调用 new 时没有用[],那调用 delete 时也不要用[].

4.3 在析构函数里对指针成员调用 delete

对于 delete 来说,释放空指针是安全的.所以,在写构造函数,赋值操作符,或其他成员函数时,类的每个指针成员要么指向有效的内存,要么就指向 NULL,那在析构函数里就可以使用 delete 释放空间,而不用担心他们是不是指向有效的存储空间.

4.4 杜绝"悬空指针"

"悬空指针"不是 NULL 指针,而是指向"垃圾"内存的指针.人们一般不会错用 NULL 指针,因为用 if 语句很容易判断.但是"悬空指针"是很危险的,if 语句对它不起作用.

形成"悬空指针"的成因主要有两种:

(1) 指针变量定义后没有被初始化.任何指针变量刚被创建时不会自动成为 NULL 指针,它的缺省值是随机的,它会乱指一气.所以,指针变量在创建的同时应当被初始化,要么将指针设置为 NULL,要么让它指向合法的内存.例如

```
char*p=NULL;
char*str=new  char(100);
```

(2) 指向某内存的指针被 delete 释放之后,没有置为 NULL,让人误以为该指针是个合法的指针,所以产生引用错误.因此,当指针被 delete 释放后,要将指针置为 NULL.

5 总结

内存管理是所有 C++ 程序开发中的重要内容,正确而有效的使用和管理内存是程序开发的关键步骤.在学习和使用 C++ 的过程中,应该认识到内存管理问题的重要性. 学习内存使用的正确模式,快速发现可能发生的错误,争取在一开始就养成内存管理的优良编程习惯,通过制订一些计划和实践,找到控制内存错误的方法.