C++内存分配

一个由 C/C++编译的程序所占用的内存可分为以下几个部分 ：

- **栈** stack，存放那些由编译器在需要的时候分配，在不需要的时候自动清除的变量，包括局部变量、函数参数、函数返回值等。它的内存分配是连续的，当声明变量时，编译器会自动接着当前栈区的结尾来分配内存，只要栈的剩余空间大于所申请空间，系统将提供内存，否则将给出异常提示。
- **堆** heap，由 new/malloc 申请的内存块，需要用户通过 delete/free 来手工释放，如果没有释放，那么在程序结束后，操作系统会自动回收。堆的分配方式类似于链表，在内存中的分布不是连续的，由不同区域的内存块通过指针链接起来的，一旦某一节点从链中断开，我们要人为的把所断开的节点从内存中释放。通常操作系统中有一个记录空闲内存地址的链表，当系统收到 new/malloc 的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的 delete/free 语句才能正确的释放本内存空间。
- **全局区/静态区** static，全局变量和静态变量被分配到同一块内存中，在以前的 C 语言中，全局变量又分为初始化的和未初始化的，在 C++里面没有这个区分了，他们共同占用同一块内存区。
- **常量区**，存放的是常量字符串，不允许修改。
- **代码区**，存放函数体的二进制代码。

堆与栈的区别：

1、管理方式：对于栈来讲，是由编译器自动管理，无需我们手工控制；对于堆来说，释放工作由用户控制，容易产生内存泄漏。

2、空间大小：一般来讲，在 32 位系统下，堆内存可以达到 4G 的空间，从这个角度来看堆内存几乎是没有什么限制的。但是对于栈来讲，一般都是有一定的大小限制的，比如 2M，当然，我们可以修改。

3、产生碎片：对于堆来讲，频繁的 new/delete 势必会造成内存空间的不连续，从而造成大量的碎片，降低程序效率。对于栈来讲，则不会存在这个问题，因为栈是先进后出的队列，他们是如此的一一对应，以至于永远都不可能有一个内存块从栈中间弹出，在他弹出之前，在他上面的后进的栈内容已经被弹出，详细的可以参考数据结构。

4、生长方向不同：对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方向是向下的，是向着内存地址减小的方向增长。

5、分配方式不同：堆都是动态分配的，没有静态分配的堆。栈有 2 种分配方式：静态分配和动态分配。静态分配是编译器完成的，比如局部变量的分配。动态分配由 alloca 函数进行

分配，但是栈的动态分配和堆是不同的，他的动态分配是由编译器进行释放，无需我们手工实现。

6、分配效率不同：栈是机器系统提供的数据结构，计算机会在底层对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行，这就决定了栈的效率比较高。堆则是 C/C++函数库提供的，它的机制较复杂，效率比栈要低。

从上面可以看到，我们尽量用栈，而不是用堆。但是由于栈和堆相比不是那么灵活，有时候分配大量的内存空间，还是用堆好一些。

无论是堆还是栈，都要防止越界现象的发生，因为越界的结果要么是程序崩溃，要么是摧毁程序的堆、栈结构，产生意想不到的结果。

一个例子：

```
int a = 0;  // 全局区
main()
{
    int b; // 栈
    char s[] = "abc"; // 栈
    char *p1; // 栈
    char *p2 = "123456"; // 123456 在常量区，p2 在栈上。
    static int c =0;  // 全局（静态）区

    p1 = (char *)malloc(10); // 堆区
    strcpy(p1, "123456"); // 123456 在常量区
}
```