C++内存管理

## 1 C++中内存分类

在C++中，内存分成5个区，他们分别是堆、栈、自由存储区、全局/静态存储区和常量存储区。

　　栈，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。

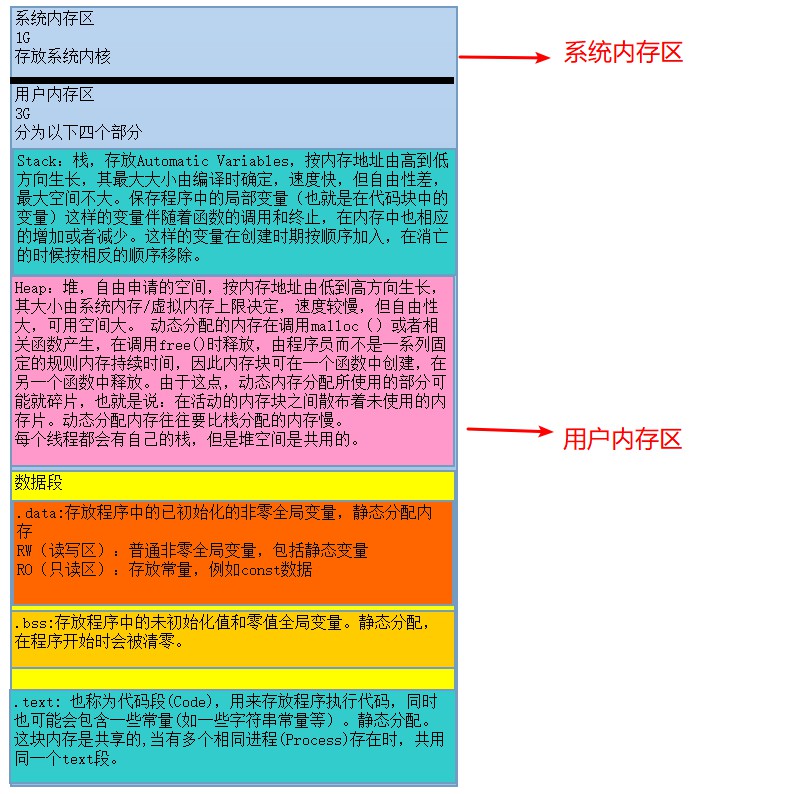
　　堆，就是那些由new分配的内存块，他们的释放编译器不去管，由我们的应用程序去控制，一般一个new就要对应一个delete。如果程序员没有释放掉，那么在程序结束后，操作系统会自动回收。

　　自由存储区，就是那些由malloc等分配的内存块，他和堆是十分相似的，不过它是用free来结束自己的生命的。

　　全局/静态存储区，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。

常量存储区，这是一块比较特殊的存储区，他们里面存放的是常量，不允许修改。

内存结构图 :



## 2 栈和堆的区别

### 管理方式不同；

栈是编译器自动管理的，无需我们手工控制。

堆的释放工作由程序员控制，容易产生memory leak。

### 空间大小不同；

一般来讲在32位系统下，堆内存可以达到4G的空间，栈每个编译器不同，可以自己设置。Vc6中1M左右。

### 能否产生碎片不同；

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，使程序效率降低

栈是先进后出形式，不会产生内存碎片。

### 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。

### 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。

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。

### 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，它的机制是很复杂的，效率较低。

## 3 常见的内存错误及其对策

\* 内存分配未成功，却使用了它。

通过assert(p!=NULL)进行判断。  
\* 内存分配虽然成功，但是尚未初始化就引用它。

在数组或者对象使用前，一定要对其进行初始化。

\*内存分配成功并且已经初始化，但操作越过了内存的边界。

\* 忘记了释放内存，造成内存泄露

\* 释放了内存却继续使用它。

有三种情况：

（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“野指针”。

规则：

【规则1】用malloc或new申请内存之后，应该立即检查指针值是否为NULL。防止使用指针值为NULL的内存。

　　【规则2】不要忘记为数组和动态内存赋初值。防止将未被初始化的内存作为右值使用。

　　【规则3】避免数组或指针的下标越界，特别要当心发生“多1”或者“少1”操作。

　　【规则4】动态内存的申请与释放必须配对，防止内存泄漏。

　　【规则5】用free或delete释放了内存之后，立即将指针设置为NULL，防止产生“野指针”。