我们先来看内存中的几大区：  **内存到底分几个区？**

下面有几种网上的理解，我整理一下：

**一：**

 1、栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

 2、堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由os回收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。

 3、全局区（静态区）（static）—，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。 - 程序结束后有系统释放。

 4、文字常量区 —常量字符串就是放在这里的。 程序结束后由系统释放。

 5、程序代码区—存放函数体的二进制代码。

**二：**

 1、栈，就是那些由编译器在需要的时候分配，在不需要的时候自动清楚的变量的存储区。里面的变量通常是局部变量、函数参数等。

 2、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。

 3、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。

 4、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。

 5、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。

**三：**

 1、bss是英文block started by symbol的简称，通常是指用来存放程序中未初始化的全局变量的一块内存区域，在程序载入时由内核清0。bss段属于静态内存分配。它的初始值也是由用户自己定义的连接定位文件所确定，用户应该将它定义在可读写的ram区内，源程序中使用malloc分配的内存就是这一块，它不是根据data大小确定，主要由程序中同时分配内存最大值所确定，不过如果超出了范围，也就是分配失败，可以等空间释放之后再分配。

 2、text段是程序代码段，在at91库中是表示程序段的大小，它是由编译器在编译连接时自动计算的，当你在链接定位文件中将该符号放置在代码段后，那么该符号表示的值就是代码段大小，编译连接时，该符号所代表的值会自动代入到源程序中。

 3、data包含静态初始化的数据，所以有初值的全局变量和static变量在data区。段的起始位置也是由连接定位文件所确定，大小在编译连接时自动分配，它和你的程序大小没有关系，但和程序使用到的全局变量，常量数量相关。

 4、stack保存函数的局部变量和参数。是一种“后进先出”（last in first out，lifo）的数据结构，这意味着最后放到栈上的数据，将会是第一个从栈上移走的数据。对于哪些暂时存贮的信息，和不需要长时间保存的信息来说，lifo这种数据结构非常理想。在调用函数或过程后，系统通常会清除栈上保存的局部变量、函数调用信息及其它的信息。栈另外一个重要的特征是，它的地址空间“向下减少”，即当栈上保存的数据越多，栈的地址就越低。栈（stack）的顶部在可读写的ram区的最后。

 5、heap保存函数内部动态分配内存，是另外一种用来保存程序信息的数据结构，更准确的说是保存程序的动态变量。堆是“先进先出”（first in first out，fifo）数据结构。它只允许在堆的一端插入数据，在另一端移走数据。堆的地址空间“向上增加”，即当堆上保存的数据越多，堆的地址就越高。

 总结(不确定！！！)：

 研究这个意义不大，不同编译器，可能行为不同，如果是vc的话，基本上如下：

 1、代码区，是编译器生成的一个exe区段，拥有可读和可执行属性，但是实际上如果不开dep数据执行保护，所有的区段都是可执行的。

 2、所谓的栈区，低地址（小于exe基地址），拥有可读写属性，exe中没有对应的区段，系统加载dll时自动生成，由于内存地址使用方式从大往小减，所以数量有限，尽量不要定义过大的数组变量。 const的局部变量也是放在栈里的，而不是放在常量区。

 3、所谓的堆区，就是malloc和new之类的内存所在区段，拥有可读写属性，exe中没有对应的区段，系统加载dll时自动生成，首先是利用栈区地址下面的区段，也是低地址，当用完了，会自动分配稍微高一点地址（大于exe基地址）。 malloc和new都在这里分配内存。

 4、全局数据区，是编译器生成的一个exe区段，拥有可读写属性，初始和未初始化的全局和静态变量都放在这里。

 5、常量区，是编译器生成的一个exe区段，只有可读属性，比如char s = " hello world" ，这时候" hello world" 就在常量区，由于没有可写属性，所以修改内容会出错，另外全局的const变量也放在常量区里，这和c++程序设计语言里对const变量存放位置是不符合的，因为存储器各有各的差异。

**局部变量，局部静态变量，全局变量，全局静态变量区别:**

局部变量：    栈区  
局部静态变量：静态区  
全局变量：    静态区的常量区  
全局静态变量：静态区

在进行C/C++[编程](http://www.2cto.com/kf" \t "http://www.cnblogs.com/bakari/archive/2012/08/05/_blank)时，需要程序员对内存的了解比较精准。经常需要操作的内存可分为以下几个类别：   
      1、栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。   
       2、堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。   
      3、全局区（静态区）（static）—，全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域， 未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。 - 程序结束后有系统释放   
      4、文字常量区 —常量字符串就是放在这里的。 程序结束后由系统释放   
      5、程序代码区—存放函数体的二进制代码。   
以下是一段实际说明的程序代码：

这是一个前辈写的，非常详细   
//main.cpp   
int a = 0; 全局初始化区   
char \*p1; 全局未初始化区   
main()   
{   
int b; 栈   
char s[] = "abc"; 栈   
char \*p2; 栈   
char \*p3 = "123456"; 123456在常量区，p3在栈上。   
static int c =0； 全局（静态）初始化区   
p1 = (char \*)malloc(10);   
p2 = (char \*)malloc(20);   
分配得来得10和20字节的区域就在堆区。   
strcpy(p1, "123456"); 123456放在常量区，编译器可能会将它与p3所指向的"123456"优化成一个地方。   
}

二、堆和栈的理论知识

2.1申请方式   
stack:   
由系统自动分配。 例如，声明在函数中一个局部变量 int b; 系统自动在栈中为b开辟空间   
heap:   
需要程序员自己申请，并指明大小，在c中malloc函数   
如p1 = (char \*)malloc(10);   
在C++中用new运算符   
如p2 = (char \*)malloc(10);   
但是注意p1、p2本身是在栈中的。   
2.2   
申请后系统的响应

栈：只要栈的剩余空间大于所申请空间，系统将为程序提供内存，否则将报异常提示栈溢出。

堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，   
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语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

2.3申请大小的限制   
栈：在Windows下,栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在 WINDOWS下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。   
堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储的空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

2.4申请效率的比较：  www.2cto.com  
栈由系统自动分配，速度较快。但程序员是无法控制的。   
堆是由new分配的内存，一般速度比较慢，而且容易产生内存碎片,不过用起来最方便.   
另外，在WINDOWS下，最好的方式是用VirtualAlloc分配内存，他不是在堆，也不是在栈是直接在进程的地址空间中保留一快内存，虽然用起来最不方便。但是速度， 也最灵活

2.5堆和栈中的存储内容   
栈： 在函数调用时，第一个进栈的是主函数中后的下一条指令（函数调用语句的下一条可执行语句）的地址，然后是函数的各个参数，在大多数的C编译器中，参数是由右往左入栈的，然后是函数中的局部变量。注意静态变量是不入栈的。   
当本次函数调用结束后，局部变量先出栈，然后是参数，最后栈顶指针指向最开始存的地址，也就是主函数中的下一条指令，程序由该点继续运行。   
堆：一般是在堆的头部用一个字节存放堆的大小。堆中的具体内容有程序员安排。

2.6存取效率的比较

char s1[] = "aaaaaaaaaaaaaaa";   
char \*s2 = "bbbbbbbbbbbbbbbbb";   
aaaaaaaaaaa是在运行时刻赋值的；   
而bbbbbbbbbbb是在编译时就确定的；   
但是，在以后的存取中，在栈上的数组比指针所指向的字符串(例如堆)快。   
比如：   
#include   
void main()   
{   
char a = 1;   
char c[] = "1234567890";   
char \*p ="1234567890";   
a = c[1];   
a = p[1];   
return;   
}   
对应的汇编代码   
10: a = c[1];   
00401067 8A 4D F1 mov cl,byte ptr [ebp-0Fh]   
0040106A 88 4D FC mov byte ptr [ebp-4],cl   
11: a = p[1];   
0040106D 8B 55 EC mov edx,dword ptr [ebp-14h]   
00401070 8A 42 01 mov al,byte ptr [edx+1]   
00401073 88 45 FC mov byte ptr [ebp-4],al   
第一种在读取时直接就把字符串中的元素读到寄存器cl中，而第二种则要先把指edx中，在根据edx读取字符，显然慢了

小结：   
堆和栈的区别可以用如下的比喻来看出：   
使用栈就象我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。   
使用堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。   
  
堆和栈的区别主要分：   
操作系统方面的堆和栈，如上面说的那些，不多说了。   
还有就是数据结构方面的堆和栈，这些都是不同的概念。这里的堆实际上指的就是（满足堆性质的）优先队列的一种数据结构，第1个元素有最高的优先权；栈实际上就是满足先进后出的性质的数学或数据结构。   
虽然堆栈，堆栈的说法是连起来叫，但是他们还是有很大区别的，连着叫只是由于历史的原因。