<https://www.cnblogs.com/lidabo/p/4611411.html>

**(1)C语言跟内存分配方式**

<1>从静态存储区域分配.  
       内存在程序编译的时候就已经分配好，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.  
<2>在栈上创建  
       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.

<3>从堆上分配，亦称动态内存分配.  
       程序在运行的时候用malloc或new申请任意多少的内存，程序员自己负责在何时用free或delete释放内存.动态内存的生存期由用户决定，使用非常灵活，但问题也最多.

(2)C语言跟内存申请相关的函数主要有 alloca、calloc、malloc、free、realloc等.  
    <1>alloca是向栈申请内存,因此无需释放.  
    <2>malloc分配的内存是位于堆中的,并且没有初始化内存的内容,因此基本上malloc之后,调用函数memset来初始化这部分的内存空间.  
    <3>calloc则将初始化这部分的内存,设置为0.  
    <4>realloc则对malloc申请的内存进行大小的调整.  
    <5>申请的内存最终需要通过函数free来释放.  
    当程序运行过程中malloc了,但是没有free的话,会造成内存泄漏.一部分的内存没有被使用,但是由于没有free,因此系统认为这部分内存还在使用,造成不断的向系统申请内存,使得系统可用内存不断减少.但是内存泄漏仅仅指程序在运行时,程序退出时,OS将回收所有的资源.因此,适当的重起一下程序,有时候还是有点作用.  
【**attention**】  
    三个函数的申明分别是:  
        void\* malloc(unsigned size);  
        void\* realloc(void\* ptr, unsigned newsize);    
        void\* calloc(size\_t numElements, size\_t sizeOfElement);   
    都在stdlib.h函数库内，它们的返回值都是请求系统分配的地址,如果请求失败就返回NULL.  
    (1)函数malloc()  
        在内存的动态存储区中分配一块长度为size字节的连续区域，参数size为需要内存空间的长度，返回该区域的首地址.  
    (2)函数calloc()  
        与malloc相似,参数sizeOfElement为申请地址的单位元素长度,numElements为元素个数，即在内存中申请numElements\*sizeOfElement字节大小的连续地址空间.  
    (3)函数realloc()  
        给一个已经分配了地址的指针重新分配空间,参数ptr为原有的空间地址,newsize是重新申请的地址长度.  
    **区别**:  
    **(1)**函数malloc不能初始化所分配的内存空间,而函数calloc能.如果由malloc()函数分配的内存空间原来没有被使用过，则其中的每一位可能都是0;反之, 如果这部分内存曾经被分配过,则其中可能遗留有各种各样的数据.也就是说，使用malloc()函数的程序开始时(内存空间还没有被重新分配)能正常进行,但经过一段时间(内存空间还已经被重新分配)可能会出现问题.  
    **(2)**函数calloc() 会将所分配的内存空间中的每一位都初始化为零,也就是说,如果你是为字符类型或整数类型的元素分配内存,那么这些元素将保证会被初始化为0;如果你是为指针类型的元素分配内存,那么这些元素通常会被初始化为空指针;如果你为实型数据分配内存,则这些元素会被初始化为浮点型的零.  
    **(3)**函数malloc向系统申请分配指定size个字节的内存空间.返回类型是 void\*类型.void\*表示未确定类型的指针.C,C++规定，void\* 类型可以强制转换为任何其它类型的指针.  
    **(4)**realloc可以对给定的指针所指的空间进行扩大或者缩小，无论是扩张或是缩小，原有内存的中内容将保持不变.当然，对于缩小，则被缩小的那一部分的内容会丢失.realloc并不保证调整后的内存空间和原来的内存空间保持同一内存地址.相反，realloc返回的指针很可能指向一个新的地址.  
   **(5)**realloc是从堆上分配内存的.当扩大一块内存空间时，realloc()试图直接从堆上现存的数据后面的那些字节中获得附加的字节，如果能够满足，自然天下太平；如果数据后面的字节不够，问题就出来了，那么就使用堆上第一个有足够大小的自由块，现存的数据然后就被拷贝至新的位置，而老块则放回到堆上.这句话传递的一个重要的信息就是数据可能被移动.

**[c-sharp]** [view plaincopy](http://blog.csdn.net/shuaishuai80/article/details/6140979)

1. #include <stdio.h>
2. #include <malloc.h>
4. **int main(int argc, char\* argv[])**
5. {
6. **char \*p,\*q;**
7. p = (**char \*)malloc(10);**
8. q = p;
9. p = (**char \*)realloc(p,10);**
10. printf("p=0x%x/n",p);
11. printf("q=0x%x/n",q);
13. **return 0;**
14. }
15. 输出结果:realloc后，内存地址不变
16. p=0x431a70
17. q=0x431a70
19. 例2:
20. #include <stdio.h>
21. #include <malloc.h>
23. **int main(int argc, char\* argv[])**
24. {
25. **char \*p,\*q;**
26. p = (**char \*)malloc(10);**
27. q = p;
28. p = (**char \*)realloc(p,1000);**
29. printf("p=0x%x/n",p);
30. printf("q=0x%x/n",q);
32. **return 0;**
33. }
34. 输出结果:realloc后，内存地址发生了变化
35. p=0x351c0
36. q=0x431a70

分类: [C++/C](http://www.cnblogs.com/lidabo/category/338913.html)