<https://blog.csdn.net/u014170207/article/details/53126340>

对于这个说烂的问题 ，做一点感悟笔记。三者都是分配内存，都是stdlib.h库里的函数，但是也存在一些细微的差异。

首先，对于malloc函数。其原型void \*malloc(unsigned int num\_bytes)

num\_byte为要申请的空间大小，需要我们手动的去计算，如int \*p = (int \*)malloc(20\*sizeof(int)),如果编译器默认int为4字节存储的话，那么计算结果是80Byte，一次申请一

个80Byte的连续空间，并将空间基地址强制转换为int类型，赋值给指针p,此时申请的内存值是不确定的。

而对于calloc函数，其原型void \*calloc(size\_t n, size\_t size);

从直观的看，其比malloc函数多一个参数，并不需要人为的计算空间的大小，比如如果他要申请20个int类型空间，会int \*p = (int \*)calloc(20, sizeof(int)）,这样就省去了人为空

间计算的麻烦。但这并不是他们之间最重要的区别，malloc申请后空间的值是随机的，并没有进行初始化，而calloc却在申请后，对空间逐一进行初始化，并设置值为0;

如下

**[cpp]** [view plain](https://blog.csdn.net/u014170207/article/details/53126340) [copy](https://blog.csdn.net/u014170207/article/details/53126340)

1. **int** \*p = (**int** \*)malloc(20\***sizeof**(**int**));
2. **int** \*pp = (**int** \*)calloc(20, **sizeof**(**int**));
3. **int** i;
5. printf("malloc申请的空间值:\n\n");
6. **for** ( i=0 ; i < 20; i++)
7. {
8. printf("%d ", \*p++);
9. }
10. printf("\n\n");
11. printf("calloc申请的空间的值:\n\n");
12. **for** ( i=0 ; i < 20; i++)
13. {
14. printf("%d ", \*pp++);
15. }
16. printf("\n");

结果为



很多人会疑问：既然calloc不需要计算空间并且可以直接初始化内存避免错误，那为什么不直接使用calloc函数，那要malloc要什么用呢？

实际上，任何事物都有两面性，有好的一面，必然存在不好的地方。这就是效率。calloc函数由于给每一个空间都要初始化值，那必然效率较malloc要低，并且现实世界，很多情况的空间申请是不需要初始值的，这也就是为什么许多初学者更多的接触malloc函数的原因。

说完了malloc和calloc，接下来谈realloc函数。

realloc函数和上面两个有本质的区别，

其原型void realloc(void \*ptr, size\_t new\_Size) 用于对动态内存进行扩容(及已申请的动态空间不够使用，需要进行空间扩容操作)

ptr为指向原来空间基址的指针， new\_size为接下来需要扩充容量的大小

**[cpp]** [view plain](https://blog.csdn.net/u014170207/article/details/53126340) [copy](https://blog.csdn.net/u014170207/article/details/53126340)

1. **int** main(**void**)
2. {
3. **const** **int** size = 2000;
4. **int** \*p = (**int** \*)malloc(20\***sizeof**(**int**));
5. **int** \*pp = (**int** \*)realloc(p, size\***sizeof**(**int**));
7. printf("原来的p\_Address:%x   扩容后的pp\_Address:%x \n\n", p, pp);
9. **return** 0;
10. }

结果为：



可从图看出，扩容后地址和原先地址是不一样的，但是这仅仅取决于扩容的内存大小。

实际上：

        如果size较小，原来申请的动态内存后面还有空余内存，系统将直接在原内存空间后面扩容，并返回原动态空间基地址；

        如果size较大，原来申请的空间后面没有足够大的空间扩容，系统将重新申请一块(20+size)\*sizeof(int)的内存，并把原来空间的内容拷贝过去，原来空间free;

        如果size非常大，系统内存申请失败，返回NULL,原来的内存不会释放。

注意：如果扩容后的内存空间较原空间小，将会出现数据丢失，如果直接realloc(p, 0);相当于free(p)

完结。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

**[cpp]** [view plain](https://blog.csdn.net/u014170207/article/details/53126340) [copy](https://blog.csdn.net/u014170207/article/details/53126340)



版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/u014170207/article/details/53126340