动态建立(new)/释放(delete)

本文链接：https://knights.blog.csdn.net/article/details/107082273

1. 运算符 new/delete

> - ***new* 运算符**：动态分配堆内存， 如果分配失败，返回一个空指针**NULL**
> - ***delete* 运算符**：释放已分配的内存空间

```
int *p1 = new int;           // 开辟一个存放整数的存储空间，返回一个指向该存储空间的地址（即指针）
int *p2 = new int(100);      // 开辟一个存放整数的空间，并指定该整数的初值为100， 返回一个指向该存储空间的地址
char *p3 = new char[100];    // 开辟一个存放字符数组(array[10])的空间 返回首元素地址
int *p4 = new int[5][4];     // 开辟一个存放二维数组的空间(5*4)， 返回首元素地址       ？？？？

delete p1;                   // 释放内存空间
delete p2;
delete []p3;
delete []p4;    // ？？？？delete二维数组，需要深入探讨
```

2. 使用举例

2.1 基础类型

```
int *p = (int *)malloc(sizeof(int));
*p = 10;
free(p);

int *p2 = new int; // 分配内存空间
*p2 = 20;
delete p2;

int *p3 = new int(30);
cout << "*p3 = " << *p3 <<endl;
delete p3;
```

2.2 数组变量

```
int *p = (int *)malloc(sizeof(int) * 10);  //int array[10]
free(p);

int *pArray = new int[10] ;
pArray[1] = 2;
delete [] pArray; //数组不要把[] 忘记
```

2.3 类对象 !!!

```
Test *pT1 = (Test *)malloc(sizeof(Test));   // 分配空间时 malloc 不会自动调用 构造函数
free(pT1);                                  // 释放空间时 free   不会自动调用 析构函数

Test *pT2 = new Test(10);        // new 分配空间时 自动调用类的 构造函数
delete pT2;                      // delete 时    自动调用类的 析构函数
```

3. 与 malloc()/free() 区别

> 1. ***malloc()/free()*** 是标准库函数， 使用前需调用库头文件 **<stdlib.h>** 方可使用； 而 ***new/delete*** 是运算符，执行效率更高。
> 2. ***malloc()*** 需要手工计算字节数; 而 ***new*** 能够自动计算需要分配的内存空间。
> 3. ***malloc()*** 返回的指针是 void *类型*, *而* **new*** 返回的指针是它分配空间的类型。
> 4. ***new*** 时调用构造函数，而 ***malloc()*** 不能；***delete*** 时调用析构函数，而 ***free()*** 不能。
> 5. ***new*** 在申请单个类型变量时可以赋初值，而 ***malloc()*** 不具备。
