# 来源：http://www.cnblogs.com/linyilong/p/5165731.html

关于空指针NULL、野指针、通用指针，首先说一下什么是指针，只要明白了指针的含义，你就明白null的含义了。

假设 有语句 int a=10;  
那么编译器就在内存中开辟1个整型单元存放变量a，我们假设这个整型单元在内存中的地址是 0x1000；那么内存0x1000单元中存放了数据10，每次我们访问a的时候，实际上都是访问的0x1000单元中的10.  
现在定义：int \*p；  
                 p=&a；  
当编译器遇到语句int \*p时，它也会在内存中给指针变量p分配一个内存单元，假设这个单元在内存的编址为0x1003；此时，0x1003中的值是不确定的，（因为我们没有给指针赋值），当编译器遇到了p=&a时，就会在0x1003单元中保存0x1000，请看，这就是说：（指针变量p代表的）内存单元0x1003存放了变量a的内存地址！用通俗的话说就是p指向了变量a。  
p=NULL，就是说：内存单元0x1003不存放任何变量的内存地址。

删除一个new了的数组。有必要的话。比如非标准的类( new CMyClass)，在Type \*p = new Type[N]; delete []p;的最后最好再加一句:   p = NULL

空指针是一个特殊的指针值，也是唯一一个对任何指针类型都合法的指针值。指针变量具有空指针值，表示它当时处于闲置状态，没有指向有意义的东西。空指针用0表示，C语言保证这个值不会是任何对象的地址。给指针值赋零则使它不再指向任何有意义的东西。为了提高程序的可读性，标准库定义了一个与0等价的符号常量NULL.    程序里可以写 p = 0;     或者 p = NULL; 两种写法都把p置为空指针值。相对而言，前一种写法更容易使读程序的人意识到这里是一个指针赋值。

我们印象中C语言的指针都有类型，实际上也存在一种例外。这里涉及到通用指针，它可以指向任何类型的变量。通用指针的类型用（void \*）表示，因此也称为void 指针。  
int n=3, \*p;  
void \*gp;  
gp = &n;  
p=(int \*)gp1;

野指针，也就是指向不可用内存区域的指针。通常对这种指针进行操作的话，将会使程序发生不可预知的错误。   
“野指针”不是NULL指针，是指向“垃圾”内存的指针。人们一般不会错用NULL指针，因为用if语句很容易判断。但是“野指针”是很危险的，if语句对它不起作用。野指针的成因主要有两种：

一、指针变量没有被初始化。任何指针变量刚被创建时不会自动成为NULL指针，它的缺省值是随机的，它会乱指一气。所以，指针变量在创建的同时应当被初始化，要么将指针设置为NULL，要么让它指向合法的内存。

二、指针p被free或者delete之后，没有置为NULL，让人误以为p是个合法的指针。别看free和delete的名字恶狠狠的（尤其是delete），它们只是把指针所指的内存给释放掉，但并没有把指针本身干掉。通常会用语句if (p != NULL)进行防错处理。很遗憾，此时if语句起不到防错作用，因为即便p不是NULL指针，它也不指向合法的内存块。例：

char \*p = (char \*) malloc(100);

strcpy(p, “hello”);

free(p); // p 所指的内存被释放，但是p所指的地址仍然不变

if(p != NULL) // 没有起到防错作用

strcpy(p, “world”); // 出错

另外一个要注意的问题：不要返回指向栈内存的指针或引用，因为栈内存在函数结束时会被释放。  
指针是个很强大的工具，可是正因为它太强大，所以要操作它不是件易事。操作不当造成的野指针，甚至会引起系统死机等比较严重的后果。 　如果程序定义了一个指针，就必须要立即让它指向一个我们设定的空间或者把它设为NULL，如果没有这么做，那么这个指针里的内容是不可预知的，即不知道它指向内存中的哪个空间（即野指针），它有可能指向的是一个空白的内存区域，可能指向的是已经受保护的区域，甚至可能指向系统的关键内存，如果是那样就糟了，也许我们后面不小心对指针进行操作就有可能让系统出现紊乱，死机了。所以我们必须设定一个空间让指针指向它，或者把指针设为NULL，这是怎么样的一个原理呢，如果是建立一个与指针相同类型的空间，实际上是在内存中的空白区域中开辟了这么一个受保护的内存空间，然后用指针来指向它，那么指针里的地址就是这个受保护空间的地址了，而不是不可预知的啦，然后我们就可以通过指针对这个空间进行相应的操作了；如果我们把指针设为NULL，我们在头文件定义中的 #define NULL 0 可以知道，其实NULL就是表示0，那么我们让指针＝NULL，实际上就是让指针＝0，如此，指针里的地址（机器数）就被初始化为0了，而内存中地址为0 的内存空间……不用多说也能想象吧，这个地址是特定的，那么也就不是不可预知的在内存中乱指一气的野指针了。 　　还应该注意的是，free和delete只是把指针所指的内存给释放掉，但并没有把指针本身干掉。指针p被free以后其地址仍然不变（非NULL），只是该地址对应的内存是垃圾，p成了“野指针”。如果此时不把p设置为NULL，会让人误以为p是个合法的指针。用free或delete释放了内存之后，就应立即将指针设置为NULL，防止产生“野指针”。内存被释放了，并不表示指针会消亡或者成了NULL指针。（而且，指针消亡了，也并不表示它所指的内存会被自动释放。） 　　最后，总结一下野指针的的成因吧： 1、指针变量没有被初始化。任何指针变量刚被创建时不会自动成为NULL指针，它的默认值是随机的，它会乱指一气。 2、指针p被free或者delete之后，没有置为NULL，让人误以为p是个合法的指针。 3、指针操作超越了变量的作用范围。这种情况让人防不胜防。