### **下文转自 http://www.cnblogs.com/porter/p/3611718.html**

### **C的NULL**

在C语言中，我们使用NULL表示空指针，也就是我们可以写如下代码：

int \*i = NULL;

foo\_t \*f = NULL;

实际上在C语言中，NULL通常被定义为如下：

#define NULL ((void \*)0)

也就是说NULL实际上是一个void \*的指针，然后吧void \*指针赋值给int \*和foo\_t \*的指针的时候，隐式转换成相应的类型。而如果换做一个C++编译器来编译的话是要出错的，因为C++是强类型的，void \*是不能隐式转换成其他指针类型的，所以通常情况下，编译器提供的头文件会这样定义NULL：

#ifdef \_\_cplusplus ---简称：cpp c++ 文件#define NULL 0#else#define NULL ((void \*)0)#endif

### **C++的0**

因为C++中不能将void \*类型的指针隐式转换成其他指针类型，而又为了解决空指针的问题，所以C++中引入0来表示空指针（注：0表示，还是有缺陷不完美），这样就有了类似上面的代码来定义NULL。实际上C++的书都会推荐说C++中更习惯使用0来表示空指针而不是NULL，尽管NULL在C++编译器下就是0。为什么C++的书都推荐使用0而不是NULL来表示空指针呢？我们看一个例子：

在foo.h文件中声明了一个函数：

void bar(sometype1 a, sometype2 \*b);

这个函数在a.cpp、b.cpp中调用了，分别是：

a.cpp:

bar(a, b);

b.cpp:

bar(a, 0);

好的，这些代码都是正常完美的编译运行。但是突然在某个时候我们功能扩展，需要对bar函数进行扩展，我们使用了重载，现在foo.h的声明如下：

void bar(sometype1 a, sometype2 \*b);void bar(sometype1 a, int i);

这个时候危险了，a.cpp和b.cpp中的调用代码这个时候就不能按照期望的运行了。但是我们很快就会发现b.cpp中的0是整数，也就是在overload resolution的时候，我们知道它调用的是void bar(sometype1 a, int i)这个重载函数，于是我们可以做出如下修改让代码按照期望运行：

bar(a, static\_cast<sometype2 \*>(0)); --- 我们的游戏项目就遇到这个问题，这样用开起来别扭

我知道，如果我们一开始就有bar的这两个重载函数的话，我们会在一开始就想办法避免这个问题（不使用重载）或者我们写出正确的调用代码，然而后面的这个重载函数或许是我们几个月或者很长一段时间后加上的话，那我们出错的可能性就会加大了不少。貌似我们现在说道的这些跟C++通常使用0来表示空指针没什么关系，好吧，假设我们的调用代码是这样的：

foo.h

void bar(sometype1 a, sometype2 \*b);

a.cpp

bar(a, b);

b.cpp

bar(a, NULL);

当bar的重载函数在后面加上来了之后，我们会发现出错了，但是出错的时候，我们找到b.cpp中的调用代码也很快可能忽略过去了，因为我们用的是NULL空指针啊，应该是调用的void bar(sometype1 a, sometype2 \*b)这个重载函数啊。实际上NULL在C++中就是0，写NULL这个反而会让你没那么警觉，因为NULL不够“明显”，而这里如果是使用0来表示空指针，那就会够“明显”，因为0是空指针，它更是一个整形常量。

在C++中，使用0来做为空指针会比使用NULL来做空指针会让你更加警觉。

### **C++ 11的nullptr**

虽然上面我们说明了0比NULL可以让我们更加警觉，但是我们并没有避免这个问题。这个时候C++ 11的nullptr就很好的解决了这个问题，我们在C++ 11中使用nullptr来表示空指针，这样最早的代码是这样的，

foo.h

void bar(sometype1 a, sometype2 \*b);

a.cpp

bar(a, b);

b.cpp

bar(a, nullptr);

在我们后来把bar的重载加上了之后，代码是这样：

foo.h

void bar(sometype1 a, sometype2 \*b);void bar(sometype1 a, int i);

a.cpp

bar(a, b);

b.cpp

bar(a, nullptr);

这时候，我们的代码还是能够如预期的一样正确运行。

在没有C++ 11的nullptr的时候，我们怎么解决避免这个问题呢？我们可以自己实现一个（《Imperfect C++》上面有一个实现）：

[IMG_256](http://www.cnblogs.com/porter/p/javascript:void(0);" \o "复制代码)

constclass nullptr\_t

{public:

template<class T>

inline operator T\*() const

{ return 0; }

template<class C, class T>

inline operator T C::\*() const

{ return 0; }

private:

void operator&() const;

} nullptr = {};

[IMG_257](http://www.cnblogs.com/porter/p/javascript:void(0);" \o "复制代码)

虽然这个东西被大家讨论过很多次了，但是我觉得还是有必要再讨论一下，毕竟在C++ 11还没有普及之前，我们还是应该知道怎么去避免问题，怎么很快的找到问题。

------  结论：  如果使用 nullptr 初始化对象，就能避免 0 指针的二义性的问题。  cocos2d-x3.0 采用 c++11的新特性了... c++ 二春来了...