今天看《Windows核心编程》无意间看到内核对象创建函数返回值的一段内容。以前写程序判断句柄无效的时候没有多想，经常用NULL，有时也用INVALID\_HANDLE\_VALUE。跟我同样犯迷糊的朋友们也要注意了，两者不能混淆的！

      如果调用一个函数以便创建内核对象，但是调用失败了，那么返回的句柄值通常是0（N U L L ）。发生这种情况是因为系统的内存非常短缺，或者 遇到了安全方面的问题。不过有少数函数在运行失败时返回的句柄值是-1 （I N VA L I D \_ H A N D L E \_ VA L U E ）。例如，如果C r e a t e F i l e未能打开指定的文件，那么它将返回I N VA L I D \_ H A N D L E \_ VA L U E ，而不是返回N U L L 。当查看创建内核对象的函数返回值 时，必须格外小心。特别要注意的是，只有当调用C r e a t e F i l e 函数时，才能将该值与I N VA L I D \_ H A N D L E \_ VA L U E 进行比较 。下面的代码是不正确的：

**[cpp]** [view plain](http://blog.csdn.net/anycell/article/details/7343125) [copy](http://blog.csdn.net/anycell/article/details/7343125) [print?](http://blog.csdn.net/anycell/article/details/7343125)

1. **HANDLE** hMutex = CreateMutex(...);
2. if (hMutex == lNVALID\_HANDLE\_VALUE)
3. {
4. //We will never execute  this code because
5. //CreateMutex returns NULL if it fails
6. }

[http://static.blog.csdn.net/images/save_snippets.png](javascript:;)

HANDLE hMutex = CreateMutex(...);

if (hMutex == lNVALID\_HANDLE\_VALUE)

{

//We will never execute this code because

//CreateMutex returns NULL if it fails

}

同样，下面的代码也不正确：

**[cpp]** [view plain](http://blog.csdn.net/anycell/article/details/7343125) [copy](http://blog.csdn.net/anycell/article/details/7343125) [print?](http://blog.csdn.net/anycell/article/details/7343125)

1. HANDIE hFile = CreateFile(.. );
2. if (hFile == NULL}
3. {
4. //We will never execute  this code because CreateFile
5. //returns lNVALID\_HANDLE\_VALUE (-1) if it fails.
6. }

# [HANDLE的无效值：NULL还是INVALID\_HANDLE\_VALUE? 以及对HANDLE的RAII封装](http://www.cnblogs.com/qinfengxiaoyue/archive/2013/05/20/3088795.html)

打开/创建一个HANDLE而忘记close的情况时有发生。利用RAII的思想，将HANDLE封装为一个类，在其析构函数中进行close，是一个不错

的方法。

ATL提供了一个CHandle类，但是提出了以下使用注意事项：

Some API functions will use NULL as an empty or invalid handle, while others use INVALID\_HANDLE\_VALUE. **CHandle** only uses NULL and will treat INVALID\_HANDLE\_VALUE as a real handle. If you call an API which can return INVALID\_HANDLE\_VALUE, you should check for this value before calling [CHandle::Attach](http://msdn.microsoft.com/en-us/library/hx7x6dz5(v=vs.80).aspx) or passing it to the **CHandle** constructor, and instead pass NULL.

即：有些API将NULL作为无效的HANLDE，但有些则将INVALID\_HANDLE\_VALUE作为无效值。CHandle只使用NULL作为无效HANDLE，

而将INVALID\_HANLDE\_VALUE视为一个真正的HANDLE.

看看相关定义：

HANDLE定义为：typedef void \*HANDLE;（在WinNt.h中定义的更详细一些）

NULL定义为：#define NULL 0 (在stddef.h中定义的更详细一些)

INVALID\_HANDLE\_VALUE定义为：#define INVALID\_HANDLE\_VALUE ((HANDLE)(LONG\_PTR)-1) . 其实就是补码表示的-1解释为

无符号数，就是0xFFFFFFFF.

比如，CreateThread返回的无效HANDLE是NULL，而CreateFile则以INVALID\_HANDLE\_VALUE表示无效HANDLE.因此使用返回

HANDLE的API需查看MSDN以保证安全.

下面这篇文章分析了HANDLE无效值不统一表示的历史原因以及相关注意：

[Why are HANDLE return values so inconsistent?](http://blogs.msdn.com/b/oldnewthing/archive/2004/03/02/82639.aspx)

那么，对HANDLE的封装怎么处理为好？

看看下例（<http://stackoverflow.com/questions/13028872/proper-way-close-winapi-handles-avoiding-of-repeated-closing>）：

基本类模板：

template< class traits >

class HandleWrapper

{

private:

traits::HandleType FHandle;

public:

HandleWrapper()：

FHandle(traits::InvalidValue)

{

}

HandleWrapper(const traits::HandleType value)：

FHandle(value)

{

}

~HandleWrapper()

{

Close();

}

void Close()

{

if (FHandle != traits::InvalidValue)

{

traits::Close(FHandle);

FHandle = traits::InvalidValue;

}

}

bool operator !() const {

return (FHandle == traits:::InvalidValue);

}

operator bool() const {

return (FHandle != traits:::InvalidValue);

}

operator traits::HandleType() {

return FHandle;

}

};

针对不同的Windows对象，提供不同的特化traits:

struct KernelHandleTraits

{

typedef HANDLE HandleType;

static const HANDLE InvalidValue = INVALID\_HANDLE\_VALUE;

static void Close(HANDLE value)

{

CloseHandle(value);

}

};

HandleWrapper<KernelHandleTraits> hFile(CreateFile(...));

struct NullKernelHandleTraits

{

typedef HANDLE HandleType;

static const HANDLE InvalidValue = NULL;

static void Close(HANDLE value)

{

CloseHandle(value);

}

};

HandleWrapper<NullKernelHandleTraits> hMapping(CreateFileMapping(...));

struct FileMapViewTraits

{

typedef void\* HandleType;

static const void\* InvalidValue = NULL;

static void Close(void \*value)

{

UnmapViewOfFile(value);

}

};

HandleWrapper<FileMapViewTraits> hView(MapViewOfFile(...));

struct GDIBitmapHandleTraits

{

typedef HBITMAP HandleType;

static const HBITMAP InvalidValue = NULL;

static void Close(HBITMAP value)

{

DeleteObject(value);

}

};

HandleWrapper<GDIBitmapTraits> hBmp(CreateBitmap(...));

妙哉！