vc 中 debug 和 release 的不同 收藏

在使用 VC 开发软件的过程中，正当要享受那种兴奋的时候突然发现：release 与 debug 运行结果不一致，甚至出错，而 release 又不方便调试，真的是当头一棒啊，可是疼归疼，问题总要解决，下面将讲述一下我的几点经验，看看是不是其中之一：

1. 变量。

大家都知道，debug 跟 release 在初始化变量时所做的操作是不同的，debug 是将每个字节位都赋成 0xcc(注 1)，而 release 的赋值近似于随机(我想是直接从内存中分配的，没有初始化过)。这样就明确了，如果你的程序中的某个变量没被初始化就被引用，就很有可能出现异常：用作控制变量将导致流程导向不一致；用作数组下标将会使程序崩溃；更加可能是造成其他变量的不准确而引起其他的错误。所以在声明变量后马上对其初始化一个默认的值是最简单有效的办法，否则项目大了你找都没地方找。代码存在错误在 debug 方式下可能会忽略而不被察觉到，如 debug 方式下数组越界也大多不会出错，在 release 中就暴露出来了，这个找起来就比较难了:(还是自己多加注意吧

2. 自定义消息的消息参数。

MFC 为我们提供了很好的消息机制，更增加了自定义消息，好处我就不用多说了。这也存在 debug 跟 release 的问题吗？答案是肯定的。在自定义消息的函数体声明时，时常会看到这样的写法：afx_msg LRESULT OnMessageOwn(); Debug 情况下一般不会有任何问题，而当你在 Release 下且多线程或进程间使用了消息传递时就会导致无效句柄之类的错误。导致这个错误直接原因是消息体的参数没有添加，即应该写成：afx_msg LRESULT OnMessageOwn(WPARAM wparam, LPARAM lparam); (注 2)

3. release 模式下不出错，但 debug 模式下报错。

这种情况下大多也是因为代码书写不正确引起的，查看 MFC 的源码，可以发现好多 ASSERT 的语句(断言)，这个宏只是在 debug 模式下才有效，那么就清楚了，release 版不报错是忽略了错误而不是没有错误，这可能存在很大的隐患，因为是 Debug 模式下，比较方便调试，好好的检查自己的代码，再此就不多说了。

4. ASSERT, VERIFY, TRACE.....调试宏

这种情况很容易解释。举个例子：请在 VC 下输入 ASSERT 然后选中按 F12 跳到宏定义的地方，这里你就能够发现 Debug 中 ASSERT 要执行 AfxAssertFailedLine，而 Release 下的宏定义却为"#define ASSERT(f) ((void)0)"。所以注意在这些调试宏的语句不要用程序相关变量如 i++写操作的语句。VERIFY 是个例外，"#define VERIFY(f) ((void)(f))"，即执行，这里的作用就不多追究了，有兴趣可自己研究:)。

总结：

Debug 与 Release 不同的问题在刚开始编写代码时会经常发生，99%是因为你的代码书写错误而导致的，所以不要动不动就说系统问题或编译器问题，努力找找自己的原因才是根本。我从前就常常遇到这情况，经历过一次次的教训后我就开始注意了，现在我所写过的代码我已经好久没遇到这种问题了。下面是几个避免的方面，即使没有这种问题也应注意一下：

1. 注意变量的初始化，尤其是指针变量，数组变量的初始化(很大的情况下另作考虑了)。
2. 自定义消息及其他声明的标准写法
3. 使用调试宏时使用后最好注释掉
4. 尽量使用 try - catch(...)

5. 尽量使用模块，不但表达清楚而且方便调试。

关于 Debug 和 Release 版本区别

关于 Debug 和 Release 之本质区别的讨论本文主要包含如下内容：
1. Debug 和 Release 编译方式的本质区别
2. 哪些情况下 Release 版会出错
2. 怎样“调试” Release 版的程序

一、Debug 和 Release 编译方式的本质区别

Debug 通常称为调试版本，它包含调试信息，并且不作任何优化，便于程序员调试程
序。Release 称为发布版本，它往往是进行了各种优化，使得程序在代码大小和运行速度
上都是最优的，以便用户很好地使用。
Debug 和 Release 的真正秘密，在于一组编译选项。下面列出了分别针对二者的选项
（当然除此之外还有其他一些，如/Fd /Fo，但区别并不重要，通常他们也不会引起 Rele
ase 版错误，在此不讨论）

Debug 版本：
/MDd /MLd 或 /MTd 使用 Debug runtime library(调试版本的运行时刻函数库)
/Od 关闭优化开关
/D "_DEBUG" 相当于 #define _DEBUG,打开编译调试代码开关(主要针对 assert 函数)
/ZI 创建 Edit and continue(编辑继续)数据库，这样在调试过
程中如果修改了源代码不需重新编译
/GZ 可以帮助捕获内存错误
/Gm 打开最小化重链接开关，减少链接时间

Release 版本：
/MD /ML 或 /MT 使用发布版本的运行时刻函数库
/O1 或 /O2 优化开关，使程序最小或最快
/D "NDEBUG" 关闭条件编译调试代码开关(即不编译 assert 函数)
/GF 合并重复的字符串，并将字符串常量放到只读内存，防止
被修改
实际上，Debug 和 Release 并没有本质的界限，他们只是一组编译选项的集合，编译
器只是按照预定的选项行动。事实上，我们甚至可以修改这些选项，从而得到优化过的调
试版本或是带跟踪语句的发布版本。

二、哪些情况下 Release 版会出错

有了上面的介绍，我们再来逐个对照这些选项看看 Release 版错误是怎样产生的

1. Runtime Library：

2. 优化：这类错误主要有以下几种：

(1) 帧指针(Frame Pointer)省略（简称 FPO ）：在函数调用过程中，所有调用信息
（返回地址、参数）以及自动变量都是放在栈中的。若函数的声明与实现不同（参数、返
回值、调用方式），就会产生错误————但 Debug 方式下，栈的访问通过 EBP 寄存器
保存的地址实现，如果没有发生数组越界之类的错误（或是越界"不多"），函数通常能
正常执行；Release 方式下，优化会省略 EBP 栈基址指针，这样通过一个全局指针访问栈
就会造成返回地址错误是程序崩溃。C++ 的强类型特性能检查出大多数这样的错误，但如
果用了强制类型转换，就不行了。你可以在 Release 版本中强制加入 /Oy- 编译选项来关
掉帧指针省略，以确定是否此类错误。
(2) volatile 型变量：volatile 告诉编译器该变量可能被程序之外的未知方式修改
（如系统、其他进程和线程）。

(3) 变量优化：优化程序会根据变量的使用情况优化变量。例如，函数中有一个未被
使用的变量，在 Debug 版中它有可能掩盖一个数组越界，而在 Release 版中，这个变量
很可能被优化调，此时数组越界会破坏栈中有用的数据。当然，实际的情况会比这复杂得
多。与此有关的错误有：

3. _DEBUG 与 NDEBUG ：当定义了 _DEBUG 时，assert() 函数会被编译，而 NDEBUG 时不
被编译。除此之外，VC++中还有一系列断言宏。这包括：

ANSI C 断言 void assert(int expression);
C Runtime Lib 断言 _ASSERT(booleanExpression);
_ASSERTE(booleanExpression);
MFC 断言 ASSERT(booleanExpression);
VERIFY(booleanExpression);
ASSERT_VALID(pObject);
ASSERT_KINDOF(classname, pobject);
ATL 断言 ATLASSERT(booleanExpression);
此外，TRACE() 宏的编译也受 _DEBUG 控制。

4. /GZ 选项：这个选项会做以下这些事

(1) 初始化内存和变量。
(2) 通过函数指针调用函数时，会通过检查栈指针验证函数调用的匹配性。（防止原
形不匹配）
(3) 函数返回前检查栈指针，确认未被修改.

三、怎样"调试" Release 版的程序

1. 前面已经提过，Debug 和 Release 只是一组编译选项的差别，实际上并没有什么
定义能区分二者。我们可以修改 Release 版的编译选项来缩小错误范围。如上所述，可以
把 Release 的选项逐个改为与之相对的 Debug 选项，如 /MD 改为 /MDd、/O1 改为 /Od
，或运行时间优化改为程序大小优化。注意，一次只改一个选项，看改哪个选项时错误消
失，再对应该选项相关的错误，针对性地查找。这些选项在 Project\Settings... 中都可
以直接通过列表选取，通常不要手动修改。由于以上的分析已相当全面，这个方法是最有
效的。
2.你也可以像 Debug 一样调试你的 Release 版，只要加入调试符号。在 Project/S
ettings... 中，选中 Settings for "Win32 Release"，选中 C/C++ 标签，Category 选
General，Debug Info 选 Program Database。再在 Link 标签 Project options 最后
加上 "/OPT:REF" (引号不要输)。

I. 内存分配问题

1. 变量未初始化。下面的程序在 debug 中运行的很好。

```
thing * search(thing * something)
  BOOL found;
  for(int i = 0; i < whatever.GetSize(); i++)
    {
    if(whatever[i]->field == something->field)
      { /* found it */
        found = TRUE;
        break;
      } /* found it */
    }
if(found)
      return whatever[i];
else
```

return NULL;

而在 release 中却不行，因为 debug 中会自动给变量初始化 found=FALSE, 而在 release 版中则不会。所以尽可能的给变量、类或结构初始化。

2. 数据溢出的问题

如：char buffer[10];
int counter;

lstrcpy(buffer, "abcdefghik");

在 debug 版中 buffer 的 NULL 覆盖了 counter 的高位，但是除非 counter>16M,什么问题也没有。但是在 release 版中， counter 可能被放在寄存器中，这样 NULL 就覆盖了 buffer 下面的空间，可能就是函数的返回地址，这将导致 ACCESS ERROR。

3. DEBUG 版和 RELEASE 版的内存分配方式是不同的 。如果你在 DEBUG 版中申请 ele 为 6*sizeof(DWORD)=24bytes,实际上分配给你的是 32bytes（debug 版以 32bytes 为单位分配），而在 release 版，分配给你的就是 24bytes（release 版以 8bytes 为单位），所以在 debug 版中如果你写 ele[6],可能不会有什么问题，而在 release 版中，就有 ACCESS VIOLATE。

II. ASSERT 和 VERIFY

1. ASSERT 在 Release 版本中是不会被编译的。

ASSERT 宏是这样定义的

```
#ifdef _DEBUG
#define ASSERT(x) if( (x) == 0) report_assert_failure()
#else
#define ASSERT(x)
#endif
```

实际上复杂一些，但无关紧要。假如你在这些语句中加了程序中必须要有的代码
比如

```
ASSERT(pNewObj = new CMyClass);
pNewObj->MyFunction();
```

这种时候 Release 版本中的 pNewObj 不会分配到空间

所以执行到下一个语句的时候程序会报该程序执行了非法操作的错误。这时可以用 VERIFY ：

```
#ifdef _DEBUG
#define VERIFY(x) if( (x) == 0) report_assert_failure()
#else
#define VERIFY(x) (x)
#endif
```

这样的话，代码在 release 版中就可以执行了。

III. 参数问题：

自定义消息的处理函数，必须定义如下：

afx_msg LRESULT OnMyMessage(WPARAM, LPARAM);

返回值必须是 HRESULT 型，否则 Debug 会过，而 Release 出错

IV. 内存分配

保证数据创建和清除的统一性：如果一个 DLL 提供一个能够创建数据的函数，那么这个 DLL 同时应该提供一个函数销毁这些数据。数据的创建和清除应该在同一个层次上。

V. DLL 的灾难

人们将不同版本 DLL 混合造成的不一致性形象的称为“动态连接库的地狱”(DLL Hell)，甚至微软自己也这么说(http://msdn.microsoft.com/library/techart/dlldanger1.htm)。

如果你的程序使用你自己的 DLL 时请注意：

1. 不能将 debug 和 release 版的 DLL 混合在一起使用。debug 都是 debug 版，release 版都是 release 版。

解决办法是将 debug 和 release 的程序分别放在主程序的 debug 和 release 目录下

2. 千万不要以为静态连接库会解决问题，那只会使情况更糟糕。

VI. RELEASE 板中的调试：

1. 将 ASSERT() 改为 VERIFY()。找出定义在"#ifdef _DEBUG"中的代码，如果在 RELEASE 版本中需要这些代码请将他们移到定义外。查找 TRACE(...)中代码，因为这些代码在 RELEASE 中也不被编译。请认真检查那些在 RELEASE 中需要的代码是否并没有被便宜。

2. 变量的初始化所带来的不同，在不同的系统，或是在 DEBUG/RELEASE 版本间都存在这样的差异，所以请对变量进行初始化。

3. 是否在编译时已经有了警告?请将警告级别设置为 3 或 4,然后保证在编译时没有警告出现.

VII. 将 Project Settings" 中 "C++/C " 项目下优化选项改为 Disbale（Debug）。编译器的优化可能导致许多意想不到的错误，请参考 http://www.pgh.net/~newcomer/debug_release.htm

1. 此外对 RELEASE 版本的软件也可以进行调试，请做如下改动：

在"Project Settings" 中 "C++/C " 项目下设置 "category" 为 "General" 并且将 "Debug Info"设置为 "Program Database"。

在"Link"项目下选中"Generate Debug Info"检查框。

"Rebuild All"

如此做法会产生的一些限制：

无法获得在 MFC DLL 中的变量的值。

必须对该软件所使用的所有 DLL 工程都进行改动。

另：

MS BUG：MS 的一份技术文档中表明，在 VC5 中对于 DLL 的"Maximize Speed"优化选项并未被完全支持，因此这将会引起内存错误并导致程序崩溃。

2. www.sysinternals.com 有一个程序 DebugView，用来捕捉 OutputDebugString 的输出，运行起来后（估计是自设为 system debugger）就可以观看所有程序的 OutputDebugString 的输出。此后，你可以脱离 VC 来运行你的程序并观看调试信息。

3. 有一个叫 Gimpel Lint 的静态代码检查工具，据说比较好用。
http://www.gimpel.com 不过要化$的。

参考文献：

1) http://www.cygnus-software.com/papers/release_debugging.html

2) http://www.pgh.net/~newcomer/debug_release.htm

在 VC 中当整个工程较大时，软件时常为出现在 DEBUG 状态下能运行而在 RELEASE 状态下无法运行的情况。由于开发者通常在 DEBUG 状态下开发软件，所以这种情况时常是在我们辛苦工作一两个月后，满怀信心的准备将软件发行时发生。为了避免无谓的损失，我们最好进行以下的检查:

1、时常测试软件的两种版本。

2、不要轻易将问题归结为 DEBUG/RELEASE 问题，除非你已经充分对两种版本进行了测试。

3、预处理的不同，也有可能引起这样的问题。
出现问题的一种可能性是在不同版本的编译间定义了不同的预处理标记。请对你的 DEBUG 版本的软件试一下以下改动：

在"Project Setting(ALT-F7)" 中的 C/C++项中设置目录(category)为"General"，并且改动"_DEBUG"定义为"NDEBUG".
设置目录为"Preprocessor"并且添加定义"_DEBUG 到"Undefined Symbols"输入框.
选择 Rebuild ALL,重新编译.
如果经过编译的程序产生了问题，请对代码进行如下改动：
将 ASSERT() 改为 VERIFY()。因为 ASSERT 中的内容在 Release 版本中不被编译。
找出定义在"#ifdef _DEBUG"中的代码，如果在 RELEASE 版本中需要这些代码请将他们移到定义外。
查找 TRACE(...)中代码，因为这些代码在 RELEASE 中也不被编译。
所以请认真检查那些在 RELEASE 中需要的代码是否并没有被编译。

4、变量的初始化所带来的不同，在不同的系统，或是在 DEBUG/RELEASE 版本间都存在这样的差异，所以请对变量进行初始化。

5、是否在编译时已经有了警告?请将警告级别设置为 3 或 4,然后保证在编译时没有警告出现.

6、是否改动了资源文件.

7、此外对 RELEASE 版本的软件也可以进行调试，请做如下改动：

在"Project Settings" 中 "C++/C " 项目下设置 "category" 为 "General" 并且将 "Debug Info"设置为 "Program Database".
在"Link"项目下选中"Generate Debug Info"检查框。
"Rebuild All"
如此做法会产生的一些限制：
无法获得在 MFC DLL 中的变量的值。
必须对该软件所使用的所有 DLL 工程都进行改动。

另：
MS BUG：MS 的一份技术文档中表明，在 VC5 中对于 DLL 的"Maximize Speed"优化选项并未被完全支持，因此这将会引起内存错误并导致程序崩溃。

VC++中 debug 跟 release 编译模式的区别总结

根据网络上的文章总结，以备查询，:)

Debug 与 Release 版本的区别

Debug 和 Release 并没有本质的区别，他们只是 VC 预定义提供的两组编译选项的集合，编译器只是按照预定的选项行动。如果我们愿意，我们完全可以把 Debug 和 Release 的行为完全颠倒过来。当然也可以提供其他的模式，例如自己定义一组编译选项，然后命名为 MY_ABC 等。习惯上，我们仍然更愿意使用 VC 已经定义好的名称。

Debug 版本包括调试信息，所以要比 Release 版本大很多（可能大数百 K 至数 M）。至于是否需要 DLL 支持，主要看你采用的编译选项。如果是基于 ATL 的，则 Debug 和 Release 版本对 DLL 的要求差不多。如果采用的编译选项为使用 MFC 动态库，则需要 MFC42D.DLL 等库支持，而 Release 版本需要 MFC42.DLL 支持。Release 不对源代码进行调试，不考虑 MFC 的诊断宏，使用的是 MFC Release 库，编译时对应用程序的速度进行优化，而 Debug 则正好相反，它允许对源代码进行调试，可以定义和使用 MFC 的 诊断宏，采用 MFC Debug 库，对速度没有优化。

既然 Debug 和 Release 仅仅是编译选项的不同，那么为什么要区分 Debug 和 Release 版本呢？

Debug 和 Release，在我看来主要是针对其面 向的目标不同的而进行区分的。Debug 通常称为调试版本，通过一系列编译选项的配合，编译的结果通常包含调试信息，而且不做任何优化，以为开发 人员提供强大的应用程序调试能力。而 Release 通常称为发布版本，是为用户使用的，一般客户不允许在发布版本上进行调试。所以不保存调试信 息，同时，它往往进行了各种优化，以期达到代码最小和速度最优。为用户的使用提供便利。

下面仅就默认的 Debug 和 Release 版本的选项进行 比较，详细的编译选项可以看 MSDN 的说明。
我们将默认的 Debug 和 Release 的选项设置进行比较，过滤掉相同设置，主要的不同如下：
编译选项：/Od /D "_DEBUG" /Gm /RTC1 /MDd /Fo"Debug“" /ZI
链接选项：/OUT："D：“MyProject“logging“Debug“OptionTest.dll" /INCREMENTAL

Release 设置：

编译选项：/O2 /GL /D "NDEBUG" /FD /MD /Fo"Release“”" /Zi

链 接选项：/OUT："D：“MyProject“logging“Release“OptionTest.dll" /INCREMENTAL：NO

Debug 版本：

/MDd /MLd 或 /MTd 使用 Debug runtime library(调试版本的运行 时刻函数库)

/Od 关闭优化开关

/D "_DEBUG" 相当于 #define _DEBUG,打开编译调试代码 开关(主要针对 assert 函数)

/ZI 创建 Edit and continue 数据库，在调试 过程中如果修改了源代码不需重新编译

/GZ 可以帮助捕获内存错误

/Gm 打开最小化重链接开关，减少链接时 间

Release 版本：

/MD /ML 或 /MT 使用发布版本的运行时刻函数库

/O1 或 /O2 优 化开关，使程序最小或最快

/D "NDEBUG" 关闭条件编译调试代码开关(即不编译 assert 函数)

/GF 合并重 复的字符串，并将字符串常量放到只读内存，防止被修改

MDd 与 MD

首 先，Debug 版本使用调试版本的运行时库（/MDd 选项），Relase 版本则使用的是发布版本的运行时库（vcrt.dll）。其区别主要在于运行时 的性能影响。调试版本的运行时库包含了调试信息，并采用了一些保护机制以帮助发现错误，也因此，其性能不如发布版本。编译器提供的 Runtime Library 很稳定，不会造成 Release 版本错误，倒是由于 Debug 版本的 Runtime Library 加强了对错误的检测，如堆内存分配检查等，反而会报告错误，应当指出，如果 Debug 有错误，而 Release 版本正常，程序肯定是有 Bug 的，只是我们还没有发现。

ZI 与 Zi

其次，/ZI 选项与/Zi 选项。通过使用/ZI 选项，可以在调试过程修改代码 而不需要重新编译。这是个调试的好帮手，可如果我们使用 Release 版本，这将变得不可行。

Od 与 O2

/O2 与/Od 选项：Od 是关闭编译器优化，普遍用于 Debug 版本。而 O2 选项是创建最快速代码，这当然是 Release 版本的不二选择。

RTCx 选 项

/RTCx 选项让编译器插入动态检测代码以帮助你检测程序中的错误。比如，它会将局部变量初始化为非零值。包括用 0xCC 初始化所有自动变量，0xCD 初始化堆中分配的内存（即 new 的内存），使用 0xDD 填充被释放的内存（即 delete 的内存），0xFD 初始化 受保护的内存（debug 版在动态分配内存的前后加入保护内存以防止越界访问）。这样做的好处是这些值都很大，一般不可能作为指针，考试，大提示作为数值 也很少用到，而且这些值很容易辩认，因此有利于在 Debug 版本中发现 Release 版才会遇到的错误。另外，通过函数指针调用函数时，会通过检 查栈指针验证函数调用的匹配性（防止原型不匹配）。使用/RTCx 选项会造成 Debug 版本出错，而 Release 版本正常的现象，因为 Release 版中未初始化的变量是随机的，很可能使指针指向了有效但是错误的地址，从而掩盖了错误。这个编译选项只能在/Od 选项下使用。

Gm，INCREMENTAL or NO

编译选项中的 Gm 和链接选项中的 INCREMENTAL 都只为一个目的，加快编译速度。我们经常遇上这样的问题，只修改了一个头文件，结果却造成所有动态库的重新编译。而这两个选项就是 为了解决这样的问题。如果启用了/Gm 开关，

编译器在项目中的.idb 文件中存储了源文件和类定义之间的依赖关系。之后的编译过程中使用.idb 文件中的信息确定是否需要编译某个源文件，哪怕是此源文件已经包含了已修改的.h 文件。

INCREMENTAL 开关默认是开启的。使用增量链接生 成的可执行文件或者动态链接库会大于非增量链接的程序，因为有代码和数据的填充。另外，增量链接的文件还包含跳转 trunk 以处理函数重定位到新地址。

MSDN 上明确指出：为确保最终发布版本不包含填充或者 trunk，请非增量链接程序。

/GZ 选项：做以下这些事

1.初 始化内存和变量。包括用 0xCC 初始化所有自动变量，0xCD (Cleared Data) 初始化堆中分配的内存（即动态分配 的内存，例如 new ），0xDD (Dead Data) 填充已被释放的堆内存（例 如 delete ），0xFD(deFencde Data) 初始化受保护的内存（debug 版在动态分配内存的前后加入保护内 存以防止越界访问），其中括号中的词是微软建议的助记词。这样做的好处是这些值都很大，作为指针是不可能的（而且 32 位系统中指针很少是奇数值， 在有些系统中奇数的指针会产生运行时错误），作为数值也很少遇到，而且这些值也很容易辨认，因此这很有利于在 Debug 版中发 现 Release 版才会遇到的错误。要特别注意的是，很多人认为编译器会用 0 来初始化变量，这是错误的（而且这样很不利于查找错 误）。

2. 通过函数指针调用函数时，会通过检查栈指针验证函数调用的匹配性。（防止原形不匹配）

3. 函数返回前检查 栈指针，确认未被修改。（防止越界访问和原形不匹配，与第二项合在一起可大致模拟帧指针省略 FPO ）

通常 /GZ 选 项会造成 Debug 版出错而 Release 版正常的现象，因为 Release 版中未初始化的变量是随机的，这有可能使指针指向一 个有效地址而掩盖了非法访问。

_DEBUG 与 NDEBUG

这是最重要的一个选项。这两个是编译器的预处理器定义，默认情况下_DEBUG 用于 Debug 版本，而 NDEBUG 用于 Release 版本。 它们可以说是重要的无以复加。因为，assert 系列的断言仅仅在_DEBUG 下生效！

下面是 assert.h 文件中摘出来的：

C++代码

```
#ifdef NDEBUG
#define assert(_Expression)     ((void)0)
#else  /* NDEBUG */
#ifdef __cplusplus
extern "C" {
#endif  /* __cplusplus */
 _CRTIMP void __cdecl _wassert(__in_z const wchar_t * _Message, __in_z const wchar_t *_File, __in unsigned _Line);
#ifdef __cplusplus
}
#endif  /* __cplusplus */
#define assert(_Expression) (void)( (!!(_Expression)) || (_wassert(_CRT_WIDE(#_Expression), _CRT_WIDE(__FILE__), __LINE__), 0) )
#endif  /* NDEBUG */
```

可以看出在未定义_DEBUG 时，assert 变成一条空语句不被执行。

也就是说，我们现在所有发布的版本无法使 用断言机制进行程序调试。

相关经验：

1. 变量。

大家都知道，debug 跟 release 在初始化变量时所做的操作是不同的，debug 是将每个字节位都赋成 0xcc，而 release 的赋值近似于随机。如果你的程序中的某个变量没被初始化就被引用，就很有可能出现异常：用作控制变量将导致流程导向不一致；用作数组下标将会使程序崩溃；更加可能是造成其他变量的不准确而引起其他的错误。所以在声明变量后马上对其 初始化一个默认的值是最简单有效的办法，否则项目大了你找都没地方找。代码存在错误在 debug 方式下可能会忽略而不被察觉到。debug 方式下数组越界也大多不会出错，在 release 中就暴露出来了，这个找起来就比较难了。

2. 自定义消息的消息参数。

MFC 为我们提供了很好的消息机制，更增加了自定义消息，好处我就不用多说了。这也存在 debug 跟 release 的问 题吗？答案是肯定的。在自定义消息的函数体声明时，时常会看到这样的写法：afx_msg LRESULT OnMessageOwn(); Debug 情况下一般不会有任何问题，而当你在 Release 下且多线程或进程间使用了消息传递时就会导致无效句柄之类的错误。导致这个错误直接原因是消息体的参数 没有添加，即应该写 成：afx_msg LRESULT OnMessageOwn(WPARAM wparam, LPARAM lparam); 3. release 模式下不出错，但 debug 模式下报错。

这种情况下大多也是因为代码书写不正确引起的，查看 MFC 的源码，可以发现好多 ASSERT 的语句(断言)，这个宏只是在 debug 模式下才有效，那么就清楚了，release 版不报错是忽略了错误而不是没有错误，这可能存在很大的隐患，因为是 Debug 模式下，比较方便调试，好好的检查自己的代码，再此就不多说了。

3. ASSERT, VERIFY, TRACE..... 调试宏

这种情况很容易解释。举个例子：请在 VC 下输入 ASSERT 然后选中按 F12 跳到宏定义的地方，这里你就能够发现 Debug 中 ASSERT 要执行 AfxAssertFailedLine，而 Release 下的宏定义却为\"#define ASSERT(f) ((void)0)\"。所以注意在这些调试宏的语句不要用程序相关变量如 i++写操作的语句。

VERIFY 是个例外，\"#define VERIFY(f) ((void)(f))\"，即执行。

哪些情况下 Release 版会出错？

1. Runtime Library：链接哪种运行时刻函数库通常只对程序的性能产生影响。调试版本 的 Runtime Library 包含了调试信息，并采用了一些保护机制以帮助发现错误，因此性能不如发布版本。编译器提供 的 Runtime Library 通常很稳定，不会造成 Release 版错误；倒是由 于 Debug 的 Runtime Library 加强了对错误的检测，如堆内存分配，有时会出现 Debug 有错 但 Release 正常的现象。应当指出的是，如果 Debug 有错，即使 Release 正常，程序肯定是有 Bug 的，只不 过可能是 Release 版的某次运行没有表现出来而已。

2. 优化：这是造成错误的主要原因，因为关闭优化时源程序基本上是直接翻译的，而打开优化后编译器会作出一系列假设。这类错误主要有以下几种：

(1) 帧指针 (Frame Pointer)省略（简称 FPO ）：在函数调用过程中，所有调用信息（返回地址、参数）以及自动变量都是放在栈中的。若函数的声明与实现不同（参数、返回值、调用方式），就会产生错误，但 Debug 方式下，栈的访问通过 EBP 寄存器保存的地址实现，如果没 有发生数组越界之类的错误（或是越界“不多”），函数通常能正常执行；Release 方式下，优化会省略 EBP 栈基址指针，这样通过一个全局 指针访问栈就会造成返回地址错误是程序崩溃。C++ 的强类型特性能检查出大多数这样的错误，但如果用了强制类型转换，就不行了。你可以在 Release 版本中强制加入 /Oy- 编译选项来关掉帧指针省略，以确定是否此类错误。此类错误通常有：

MFC 消息响应函数书写错误。正确的应为

afx_msg LRESULT OnMessageOwn(WPARAM wparam, LPARAM lparam);

ON_MESSAGE 宏 包含强制类型转换。防止这种错误的方法之一是重定义 ON_MESSAGE 宏，把下列代码加到 stdafx.h 中 （在#include "afxwin.h"之后）,函数原形错误时编译会报错

C++代码

```
#undef  ON_MESSAGE
#define  ON_MESSAGE(message,  memberFxn)  \
{  message,  0,  0,  0,  AfxSig_lwl,  \
(AFX_PMSG)(AFX_PMSGW)(static_cast<  LRESULT  (AFX_MSG_CALL  \
CWnd::*)(WPARAM,  LPARAM)  >  (&memberFxn)  },
```

(2) volatile 型变量：volatile 告诉编译器该变量可能被程序之外的未知方式修改（如系统、其他进程和线程）。优化程序为了使程序性能提高，常把一些变量放在寄存器中（类似于 register 关键字），而其他进程只能对该变量所在的内存进行修改，而寄存器中的 值没变。如果你的程序是多线程的，或者你发现某个变量的值与预期的不符而你确信已正确的设置了，则很可能遇到这样的问题。这种错误有时会表现为程序在最快 优化出错而最小优化正常。把你认为可疑的变量加上 volatile 试试。

(3) 变量优化：优化程序会根据变量的使用情况优化变量。例如，函数中有一个未被使用的变量，在 Debug 版中它有可能掩盖一个数组越界，而在 Release 版中，这个变量很可能被 优化调，此时数组越界会破坏栈中有用的数据。当然，实际的情况会比这复杂得多。与此有关的错误有：

非法访问，包括数组越界、指针错误 等。例如

C++代码

```
void  fn(void)
{
    int  i;
    i  =  1;
   int  a[4];
   {
      int  j;
      j  =  1;
   }
   a[-1]  =  1; //当然错误不会这么明显，例如下标是变量
   a[4]  =  1;
}
```

j 虽然在数组越界时已出了作用域，但其空间并未收回，因而 i 和 j 就会掩盖越界。而 Release 版由于 i、j 并未其很大作用可能会被优化掉，从而使栈被破坏。

3. _DEBUG 与 NDEBUG ：当定义了 _DEBUG 时，assert() 函数会被编译， 而 NDEBUG 时不被编译。除此之外，VC++中还有一系列断言宏。这包括：

ANSI C 断 言 void assert(int expression);
C Runtime Lib 断 言 _ASSERT(booleanExpression);
_ASSERTE(booleanExpression);
MFC 断 言 ASSERT(booleanExpression);

VERIFY(booleanExpression);
ASSERT_VALID(pObject);
ASSERT_KINDOF(classname, pobject);
ATL 断言 ATLASSERT(booleanExpression);
此外，TRACE() 宏的编译也受 _DEBUG 控制。

所有这些断言都只在 Debug 版中才被编译，而在 Release 版中被忽略。唯一的例外是 VERIFY() 。事实上，这些宏都是调用了 assert() 函数，只不过附加了一些与库有关的调试代码。如果你在这些宏中加入了任何 程序代码，而不只是布尔表达式（例如赋值、能改变变量值的函数调用 等），那么 Release 版都不会执行这些操作，从而造成错误。初学者很容 易犯这类错误，查找的方法也很简单，因为这些宏都已在上面列出，只要利
用 VC++ 的 Find in Files 功能在工程所有文件中 找到用这些宏的地方再一一检查即可。另外，有些高手可能还会加入 #ifdef _DEBUG 之类的条件编译，也要注意一下。
顺便值 得一提的是 VERIFY() 宏，这个宏允许你将程序代码放在布尔表达式里。这个宏通常用来检
查 Windows API 的返回值。有些人 可能为这个原因而滥用 VERIFY() ，事实上这是危险的，因为 VERIFY() 违反了断言的思想，不能使程序代码和调试代码完全分离， 最终可能会带来很多麻烦。因此，专家们建议尽量少用这个宏。

一、"Debug 是调试版本，包括的程序信息更多"
补充：只有 DEBUG 版的程序才能设置断点、单步执行、使用 TRACE/ASSERT 等调试输出语句。REALEASE 不包含任何调试信息，所以体积小、运行速度快。

I. 内存分配问题
1. 变量未初始化。下面的程序在 debug 中运行的很好。

C++代码

```
thing * search(thing * something)
BOOL found;
for(int i = 0; i < whatever.GetSize(); i++)
{
if(whatever[i]->field == something->field)
{ /* found it */
found = TRUE;
break;
} /* found it */
}
if(found)
return whatever[i];
else
return NULL;
```

而在 release 中却不行，因为 debug 中会自动给变量初始化 found=FALSE,而在 release 版中则不会。所以尽可能的给变量、类或结构 初始化。

2. 数据溢出的问题

如：

C++代码

```
1.  char    buffer[10];
2.  int    counter;
3.  lstrcpy(buffer,    "abcdefghik");
```

在 debug 版中 buffer 的 NULL 覆盖了 counter 的高位，但是除非 counter>16M,什么问题也 没有。但是在 release 版中，counter 可能被放在寄存器中，这样 NULL 就覆盖了 buffer 下面的空间，可能就是函数的返回地址，这将导致 ACCESS ERROR。

3. DEBUG 版和 RELEASE 版的内存分配方式是不同的 。如果你在 DEBUG 版中申请 ele 为 6*sizeof(DWORD)=24bytes,实际上分配给你的是 32bytes（debug 版以 32bytes 为单位分配）， 而在 release 版，分配给你的就是 24bytes（release 版以 8bytes 为单位），所以在 debug 版中如果你写 ele[6],可能不会有 什么问题，而在 release 版中，就有 ACCESS VIOLATE。

II. ASSERT 和 VERIFY

1. ASSERT 在 Release 版本中是不会被编译的。

ASSERT 宏是这样定义的

C++代码

```
1.
2.  #ifdef    _DEBUG
3.  #define    ASSERT(x)    if(    (x)    ==    0)    report_assert_f
    ailure()
4.  #else
5.  #define    ASSERT(x)
6.  #endif
```

实际上复杂一些，但无关紧要。假如你在这些语句中加了程序中必须要有的代码

比如

C++代码

```
1.
2.  ASSERT(pNewObj    =    new    CMyClass);
3.
4.  pNewObj->MyFunction();
```

这种时候 Release 版本中的 pNewObj 不会分配到空间

所以执行到下一个语句的时候程序会报该程序执行了非法操作的错误。这时可以用 VERIFY ：

C++代码

```
1.
2.  #ifdef  _DEBUG
3.  #define  VERIFY(x)  if(  (x)  ==  0)  report_assert_f
    ailure()
4.  #else
5.  #define  VERIFY(x)  (x)
6.  #endif
```

这样的话，代码在 release 版中就可以执行了。

III. 参数问题：

自定义消息的处理函数，必须定义如下：
afx_msg LRESULT OnMyMessage(WPARAM, LPARAM);
返回值必须是 HRESULT 型，否则 Debug 会过，而 Release 出错

IV. 内存分配
保证数据创建和清除的统一性：如果一个 DLL 提供一个能够创建数据的函数，那么这个 DLL 同时应该提供一个函数销毁这些数据。数据的创建和清除应该在同一 个层次上。

V. DLL 的灾难
人们将不同版本 DLL 混合造成的不一致性形象的称为 “动态连接库的地狱“(DLL Hell) ，甚至微软自己也这么说 http://msdn.microsoft.com/library/techart/dlldanger1.htm)。

如果你的程序使用你自己的 DLL 时请注意：

1. 不能将 debug 和 release 版的 DLL 混合在一起使用。debug 都是 debug 版，release 版都是 release 版。
解决办法是将 debug 和 release 的程序分别放在主程序的 debug 和 release 目录下

2. 千万不要以为静态连接库会解决问题，那只会使情况更糟糕。

VI. RELEASE 版中的调试 ：
1. 将 ASSERT() 改为 VERIFY() 。找出定义在"#ifdef _DEBUG"中的代码，如果在 RELEASE 版本中需要这些代码请将他们移到定义外。查找 TRACE(...)中代码，因为这些代码在 RELEASE 中 也不被编译。 请认真检查那些在 RELEASE 中需要的代码是否并没有被便宜。
2. 变量的初始化所带来的不同，在不同的系统，或是在 DEBUG/RELEASE 版本间都存在这样的差异，所以请对变量进行初始化。
3. 是否在编译时已经有了警告?请将警告级别设置为 3 或 4,然后保证在编译时没有警告出现.

VII. 将 Project Settings" 中 "C++/C " 项目下优化选项改为 Disbale（Debug）。编译器的优化可能导致

许多意想不到的错误，请参 http://www.pgh.net/~newcomer/debug_release.htm

1. 此外对 RELEASE 版本的软件也可以进行调试，请做如下改动：

在"Project Settings"中"C++/C"项目下设置"category"为"General"并且将"Debug Info"设置为"Program Database"。

在"Link"项目下选中"Generate Debug Info"检查框。

"Rebuild All"

如此做法会产生的一些限制：

无法获得在 MFC DLL 中的变量的值。

必须对该软件所使用的所有 DLL 工程都进行改动。

另：

1. MS BUG：MS 的一份技术文档中表明，在 VC5 中对于 DLL 的"Maximize Speed"优化选项 并未被完全支持，因此这将会引起内存错误并导致程序崩溃。

2. http://www.sysinternals.com/有 一个程序 DebugView，用来捕捉 OutputDebugString 的输出，运行起来后（估计是自设为 system debugger）就可以观看所有程序的 OutputDebugString 的输出。此后，你可以脱离 VC 来运行你的程序并观看调试信息。

3. 有一个叫 Gimpel Lint 的静态代码检查工具，据说比较好用 http://www.gimpel.com/ 不过要化$的。

Debug 与 Release 不同的问题在刚开始编写代码时会经常发生，99%是因为你的代码书写错误而导致的，所以不要动不动就说系统问 题或编译器问题，努力找找自己的原因才是根本。我从前就常常遇到这情况，经历过一次次的教训后我就开始注意了，现在我所写过的代码我已经好久没遇到这种问 题了。下面是几个避免的方面，即使没有这种问题也应注意一下：

1. 注意变量的初始化，尤其是指针变量，数组变量的初始化(很大的情况下另作考 虑了)。
2. 自定义消息及其他声明的标准写法
3. 使用调试宏时使用后最好注释掉
4. 尽量使用 try - catch(...)
5. 尽量使用模块，不但表达清楚而且方便调试。