通常Release除错都是先通过SetUnhandledExceptionFilter捕获异常，然后生成报告文件，最后定位代码行，主要以下两种方法：

1. 通过遍历调用栈，将其调用栈信息输出到文件。然后查找出错地址。

查找方式有两种：

（1）通过编译器生成的包含行信息的map文件定位出错位置。

通过在“工程属性”-〉“link”-〉“Project Options”手工输入 /mapinfo:lines，生成包含行信息map文件。查找时首先根据出错地址范围找到obj文件名，查看obj文件对应的行信息，根据出错地址范围定位代码行。

（2）通过编译器生成的pdb文件定位出错位置。

debug版本会自动生成pdb文件，Release版本需要在“工程属性”-〉“link”面板中勾上选项“Generate debug info”，然后在“工程属性” -〉“C/C++”面板的“Debug Info”列表框选中“Program Database”。

在pdb文件中查找出错地址所在的代码行，需要通过dbghelp库（包含在windbg目录下），通过SymFromAddr函数可以获取符号信息，SymGetLineFromAddr64获取所在代码行。

遍历调用栈方法方法也有两种：

（1）自己遍历调用栈

这种方法的缺陷是Release版本通常会使用FPO（Frame-Pointer Omisstion) 优化，（注：在VC编译器中可以在“工程属性”—> “C/C++”—>“Project Options”中去掉选项Oy-关闭PFO优化），PFO优化主要是通过省略调用时栈指针的保存恢复等操作提高代码效率。下面自己遍历调用栈的方法对采用了FPO优化的模块可能会遍历不完全，遗漏掉一些函数。因此，即使自己的模块关闭了FPO，但第三方模块使用了FPO，如果报错的地址位于第三方dll内（例如mfc42.dll)，将有可能回溯不到自己模块内有问题的函数，从而很难定位bug。

自己遍历基于以下原理（这个原理只适用于没有采用FPO优化的函数）:

    1 函数调用时call指令将返回地址（通常是下一条指令的地址）压入堆栈 。

    2 函数运行第一行会将 ebp压入堆栈，保存它以使得当函数返回能恢复ebp。

    3 Copy当前栈位置esp到 ebp。

4.然后esp自减以空出栈空间容纳函数的局部变量

因此当前函数内的ebp即为第2步压入ebp后的栈顶位置，由此可推导出上一层函数的ebp为[ebp]，而上一层函数返回地址即为前一个压入栈的值，即[ebp+4]，由此可以一步步往上回溯调用栈。

（2）通过dbghelp库函数StackWalk64遍历堆栈。

这种方式可以选择是否加载pdb，对于做了那些被FPO优化的函数，pdb保存了相关数据来帮助遍历调用栈，如果不能加载到正确的pdb，StackWalk64将使用前面介绍的基于ebp的方式遍历调用栈，从而漏掉那些被FPO优化的函数。

（二）通过生成mini dump文件定位bug。

通过dbghelp库函数MiniDumpWriteDump将出错时信息写入文件，然后用windbg打开dump文件，配置好symbols路径，exe文件路径，source code 路径，输入.ecxr命令，就可以查看详细的调用栈，并能自动打开源文件定位到代码行。因此这种方法是简单和最可靠的方法。

下面是一个简单的dume类，只要添入到工程即可，出错时会自动生成dum文件。

#include <windows.h>

#include <tchar.h>

#include <assert.h>

//for VC6

#ifndef \_\_in\_bcount\_opt

#define \_\_in\_bcount\_opt(x)

#endif

#ifndef \_\_out\_bcount\_opt

#define \_\_out\_bcount\_opt(x)

#endif

//end (for VC6)

#include "dbghelp.h"

typedef BOOL (WINAPI \*MINIDUMPWRITEDUMP)(

IN HANDLE hProcess,

IN DWORD ProcessId,

IN HANDLE hFile,

IN MINIDUMP\_TYPE DumpType,

IN CONST PMINIDUMP\_EXCEPTION\_INFORMATION ExceptionParam, OPTIONAL

IN CONST PMINIDUMP\_USER\_STREAM\_INFORMATION UserStreamParam, OPTIONAL

IN CONST PMINIDUMP\_CALLBACK\_INFORMATION CallbackParam OPTIONAL

);

class CMiniDumper

{

public:

CMiniDumper();

private:

static LPTOP\_LEVEL\_EXCEPTION\_FILTER s\_pPrevFilter;

static long WINAPI UnhandledExceptionFilter( struct \_EXCEPTION\_POINTERS \*pExceptionInfo );

};

CMiniDumper g\_minObject;

LPTOP\_LEVEL\_EXCEPTION\_FILTER CMiniDumper::s\_pPrevFilter = 0;

CMiniDumper::CMiniDumper()

{

assert(!s\_pPrevFilter);

s\_pPrevFilter = ::SetUnhandledExceptionFilter(UnhandledExceptionFilter);

}

long CMiniDumper::UnhandledExceptionFilter( struct \_EXCEPTION\_POINTERS \*pExceptionInfo )

{

long ret = EXCEPTION\_CONTINUE\_SEARCH;

TCHAR szDbgHelpPath[\_MAX\_PATH] = {0};

TCHAR szDumpPath[\_MAX\_PATH] = {0};

TCHAR szPath[\_MAX\_PATH] = {0};

if (GetModuleFileName(NULL, szPath, \_MAX\_PATH))

{

TCHAR szDrive[\_MAX\_DRIVE] = {0};

TCHAR szDir[\_MAX\_DIR] = {0};

TCHAR szFileName[\_MAX\_FNAME] = {0};

\_tsplitpath(szPath, szDrive, szDir, szFileName, 0);

\_tcsncat(szDbgHelpPath, szDrive, \_MAX\_PATH);

\_tcsncat(szDbgHelpPath, szDir, \_MAX\_PATH - \_tcslen(szDbgHelpPath) - 1);

\_tcsncat(szDbgHelpPath, \_T("dbghelp.dll"), \_MAX\_PATH - \_tcslen(szDbgHelpPath) - 1);

\_tcsncat(szDumpPath, szDrive, \_MAX\_PATH);

\_tcsncat(szDumpPath, szDir, \_MAX\_PATH - \_tcslen(szDumpPath) - 1);

\_tcsncat(szDumpPath, szFileName, \_MAX\_PATH - \_tcslen(szDumpPath) - 1);

\_tcsncat(szDumpPath, \_T(".dmp"), \_MAX\_PATH - \_tcslen(szDumpPath) - 1);

}

HMODULE hDll = ::LoadLibrary(szDbgHelpPath);

if (hDll==NULL)

hDll = ::LoadLibrary(\_T("dbghelp.dll"));

assert(hDll);

if (hDll)

{

MINIDUMPWRITEDUMP pWriteDumpFun = (MINIDUMPWRITEDUMP)::GetProcAddress(hDll, "MiniDumpWriteDump");

if (pWriteDumpFun)

{

// create the file

HANDLE hFile = ::CreateFile(szDumpPath, GENERIC\_WRITE, FILE\_SHARE\_WRITE,

NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE){

\_MINIDUMP\_EXCEPTION\_INFORMATION ExInfo;

ExInfo.ThreadId = ::GetCurrentThreadId();

ExInfo.ExceptionPointers = pExceptionInfo;

ExInfo.ClientPointers = FALSE;

// write the dump

if (pWriteDumpFun(GetCurrentProcess(), GetCurrentProcessId(),

hFile, MiniDumpNormal, pExceptionInfo!=0? &ExInfo: 0, NULL, NULL))

ret = EXCEPTION\_EXECUTE\_HANDLER;

::CloseHandle(hFile);

}

}

}

if (s\_pPrevFilter)

ret = s\_pPrevFilter(pExceptionInfo);

return ret;

}