# VS开发环境工具

## 开启安全相关选项

（1）GS（栈保护）：通过在栈中加入校验单元来防止出现缓冲区溢出错误。

（2）NXCOMPAT（与数据执行保护兼容）：DEP，也就是数据执行保护，可以有效降低堆或栈上的缓存溢出漏洞的危害性。采用NXCOMPAT选项后，应用程序的运行被DEP 机制保护。在考虑兼容性的前提下，建议开发人员采用NXCOMPAT链接选项。

（3）ASLR（地址空间分布随机）：是一种针对缓冲区溢出的安全保护技术，通过对堆、栈、共享库映射等线性区布局的随机化，通过增加攻击者预测目的地址的难度，防止攻击者直接定位攻击代码位置，达到阻止溢出攻击的目的。

为工程打开GF开关, 设置Runtime Library为Multi-threaded Debug (/MTd)

## 嵌入汇编

方法一:

\_\_asm 汇编语句1

\_\_asm 汇编语句2

方法二:

\_\_asm

{

汇编语句

}

## 快捷键

界面设置显示每个控件的Tab顺序:Ctrl + d

## 编码格式

\_T()根据当前的VS设定的编码格式来定义Unicode、Multibyte-character或SBCS编码

若当前编码为Unicode: TCHAR、\_stprintf\_s、LPCTSTR等，都会被替换成w\_char\_t之类的宽字符。 Sizeof(TCHAR)等于2。\_T等于L

若当前编码为其他:TCHAR、\_stprintf\_s、LPCTSTR等，都会被替换成char之类的窄字符。 Sizeof(TCHAR)等于1。\_T等于没有。

英文操作系统用Unicode编码， 否则中文会显示乱码。

若使用多字符集出现界面风格变化:

在stdafx.h文件中用替换则不会变化

#ifdef \_UNICODE

#if defined \_M\_IX86

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#elif defined \_M\_IA64

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='ia64' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#elif defined \_M\_X64

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='amd64' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#else

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='\*' publicKeyToken='6595b64144ccf1df' language='\*'\"")

#endif

#endif

#pragma comment(linker,"/manifestdependency:\"type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='x86' publicKeyToken='6595b64144ccf1df' language='\*'\"")

## VS工具命令行编译

1 编译器 CL.EXE 将源代码编译生成目标代码。

2 链接器 LINK.EXE 将目标代码，库文件链接生成最终文件。

3 资源编译器 RC.EXE将资源编译，最终通过链接器存入最终的文件。

## VS中相关格式文件

### 预编译头文件(.pch)

所谓头文件预编译，就是把一个工程(Project)中使用的一些MFC标准头文件(如Windows.H、Afxwin.H)预先编译，以后该工程编译时，不再编译这部分头文件，仅仅使用预编译的结果。这样可以加快编译速度，节省时间。

编译器通过一个头文件stdafx.h来使用预编译头文件。stdafx.h这个头文件名是可以在project的编译设置里指定的。编译器认为，所有在指令#include "stdafx.h"前的代码都是预编译的，它跳过#include "stdafx. h"指令，使用projectname.pch编译这条指令之后的所有代码。

因此，所有的CPP实现文件第一条语句都是：#include "stdafx.h"。

另外，每一个实现文件CPP都包含了如下语句：

#ifdef \_DEBUG

#undef THIS\_FILE

static char BASED\_CODE THIS\_FILE[] = \_\_FILE\_\_;

#endif

这是表示，如果生成调试版本，要指示当前文件的名称。\_\_FILE\_\_是一个宏，在编译器编译过程中给它赋值为当前正在编译的文件名称。

.vcxproy 相当于VC6中 .dsp

.vcxproj.filters 用于项目下文件的虚拟目录

.vcxproj.user 是用户的一些相关配置

### 预处理文件(.l)

.i文件将宏替换生成到文件中， 此选项默认关闭， 在”工程属性->C++->预处理”位置。

### 增量链接文件(.ilk)

在第一次链接时则生成.ilk文件，其后链接则更新.ilk文件， 若.ilk文件不存着或损坏则全量链接。 .ilk文件开关在”工程属性->链接器->常规”位置。

### 程序数据库文件(.pdb)

Pdb文件保存着程序的调试信息。(不保存断点位置信息，断点位置信息保存在.suo文件中)。

### 目标文件(.obj)

Obj文件是程序编译后的二进制文件；在通过链接器和资源文件链接生成exe文件了。

### 无编译浏览文件(.ncb)

自动完成功能出问题时，删掉该文件， build会自动生成。

### 解决方案用户选项记录文件(.suo)

.suo文件在VS关闭后生成， 保存了一些用户操作记录，如断点位置、监视标识等信息。

### 解决方案文件(.sln)

.sln 相当于VC6中 .dsw

### 二进制资源文件(.aps)

可以删除， 每次VS自动生成。

## Tool选项介绍

可以添加新Tool , 执行Dos命令。

## MFC自带宏定义

1) THIS\_FILE (\_\_FILE\_\_)

获取当前文件名,  是字符串类型

2) MAKELONG

DWORD = MAKELONG(LWORD, HWORD) //将两个WORD合并DWORD

3) LOWORD、HIWORD

WORD = LOWORD(DWORD)  //获取DWORD的低WORD

WORD = HIWORD(DWORD)  //获取DWORD的高WORD

4) \_\_FUNCTION\_\_

获取当前函数, 是字符串类型

5) \_\_LINE\_\_

获取当前行号, 是整型。

1. MAX\_PATH

等于260，全路径最大长度

## # pragma介绍

1)  编译器优化(optimize)

#pragma optimize("",off);

memset(pwd, 0, sizeof(pwd));

#pragma optimize("",on);

2)  加载动态库引入库(.lib)

#pragma comment(lib, "imagehlp.lib") //表示和当前工程(.vcxproj)在同一路径下。

3)  在Dll中设置共享节，多个进程访问同一个数据

(1) 创建节

#pragma data\_seg("MySec") //节名称:不超过8个，若超过则截断

HWND g\_hWnd = NULL;    //必须初始化， 不然系统会自动将该变量放入BBS段，导致多个进程共享数据失败。

#pragma data\_seg()

(2) 设置节

方法一:

在创建节后面直接添加:

#pragma comment(linker, "/section:MySec,RWS") //设置工程连接选项。

方法二:在.def文件中设置节

在函数之后：

SEGMENTS

MySec    READ WRITE SHARED    //读写方式不区分大小写

(3) 使用Dumpbin命令查看Dll中共享节。

4)  设置结构体字节对齐

#pragma pack(1):按1个字节对齐

5)  屏蔽告警

#pragma warning( disable : 4503)    4503:告警号

## 制作安装包

利用VS生成基本安装包流程:

"文件"-->"新建"-->"项目"-->"新建项目"-->选中"项目类型"-->"安装和部署"-->"安装项目"-->"安装向导"， 最后在里面添加安装包所需文件(如exe)。

## 获取某个应用程序的资源

选择“File”=>”Open”=>”指定的应用程序”。

## 批处理

### 命令

 %cd%:表示当前批处理的目录， 随cd命令而改变。如cd d:\BT则%cd%表示D:\BT。

在调用其他批处理之前最好使用set命令保存其路径， 不然%cd%很可能会改变。

 set name=name1   name并不是替换name1,而是保存name1。如set FILENAME\_DEVENV=%PATH\_VS2010%\Common7\IDE\devenv.exe, //%PATH\_VS2010%是环境变量。

 %~dp0:表示当前批处理的目录，不受cd命令的影响。

 %加数字表示第几个参数。 如%0表示批处理本身，%1表示输入批处理的第一个参数。

 %%等价于Cmd中% , 变量的意思。

call表示调用另一个批处理。 如CopyFile.bat

@echo on表示打开控制台显示相关信息。

@rem表示注释, 如@rem 这是注释

if exist Compile.log ( //删除已有的文件

del Compile.log /f /q /s

)

echo表示显示相关文本内容如echo "start build..."

### 编译生成exe

"%VS100COMNTOOLS%"..\..\ Common7\IDE\devenv.exe BT\_GUI.sln /Rebuild Release /Out Compile.log

## 常见编译链接错误

### Windows程序包含.c文件出现预编译错误

错误字符串：precompiled header file is from a previous version of the compiler, or the precompiled header is C++ and you are using it from C (or vice versa)

解决方法一: 点击指定的.c文件中的属性，设置不包含预编译头文件。

### LINK : fatal error LNK1000: Internal error during IncrBuildImage

解决方法一:

属性(Property pages)->配置属性(Configuration Properties) ->C/C++修改调试信息格式(Debug Information Format)程序数据库(Program Database(/Zi))。

## 调试相关

### Debug模式下快速检测内存泄露

void CheckOnExit()

{

\_CrtSetDbgFlag ( \_CRTDBG\_ALLOC\_MEM\_DF | \_CRTDBG\_LEAK\_CHECK\_DF );

assert( \_CrtDumpMemoryLeaks() == 0);

}

Atexit(CheckOnExit); //在程序开始时调用，关闭程序后，若有内存泄露则信息出现在调试窗口。 有文件，行号，第几次申请内存信息。

若只有第几次申请内存信息，则在之后调用下面函数。

\_CrtSetBreakAlloc(52); //52表示第几次申请内存， 查看调用堆栈。

### DeBug模式下的调试输出窗口显示文本

TRACE(“%s”,“文本”);

### Debug模式下的断言操作

void\* p = NULL;

ASSERT(p );

### 远程调试

远程机(要调试的程序):将Common7\IDE目录下的Remote Debugger目录拷到远程机任意位置， 运行msvsmon.exe程序并设置无需身份认证,再运行调试程序。

本机:

方法一：VS中选择调试->附加到进程->传输(选择远程),限定符输入远程机IP。 点击刷新可看见远程机所运行的线程了。 选择要调试的程序，并附加即可。

方法二：在工程属性->调试设置。

远程命令：要调试的程序。

工作目录：要调试的程序目录。

远程服务器名：填写IP或计算机名。

连接：选择无身份认证。

### 程序崩溃产生原因信息

第一步: 在程序开始出调用SetUnhandledExceptionFilter函数，该函数的参数是个函数指针，在程序崩溃时会调用该函数指针对应的函数(回调函数)。

第二步: 回调函数的参数存放的造成崩溃的信息， 可对此信息做处理，如写dmp文件等。

错误码:

EXCEPTION\_ACCESS\_VIOLATION   (0xC0000005)  非法访问虚拟内存

EXCEPTION\_INT\_DIVIDE\_BY\_ZERO  (0xC0000094)     除数不能为零

EXCEPTION\_STACK\_OVERFLOW     (0xC00000FD) 堆栈溢出或者说越界

EXCEPTION\_GUARD\_PAGE      (0x80000001)  由Virtual Alloc建立起来的属性页冲突。

EXCEPTION\_NONCONTINUABLE\_EXCEPTION  (0xC0000025h)  不可持续异常,程序无法恢复执行,异常处理例程不应处理这个异常。

EXCEPTION\_INVALID\_DISPOSITION    (0xC0000026h)  在异常处理过程中系统使用的代码,如果系统从某个例程莫名奇妙的返回,则出现此代码,例如调用RtlUnwind时没有Exception Record参数时产生的异常填入的就是这个代码

EXCEPTION\_BREAKPOINT              (0x80000003)  调试时因代码中int3中断

EXCEPTION\_SINGLE\_STEP              (0x80000004)  处于被单步调试状态

EXCEPTION\_ARRAY\_BOUNDS\_EXCEEDED   (0xC000008C)

EXCEPTION\_DATATYPE\_MISALIGNMENT    (0x80000002)

EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO     (0xC000008E)

EXCEPTION\_FLT\_DENORMAL\_OPERAND     (0xC000008D)

EXCEPTION\_FLT\_INEXACT\_RESULT          (0xC000008F)

EXCEPTION\_FLT\_INVALID\_OPERATION      (0xC0000090)

EXCEPTION\_FLT\_OVERFLOW         (0xC0000091)

EXCEPTION\_FLT\_STACK\_CHECK        (0xC0000092)

EXCEPTION\_FLT\_UNDERFLOW           (0xC0000093)

EXCEPTION\_ILLEGAL\_INSTRUCTION        (0xC000001D)

EXCEPTION\_IN\_PAGE\_ERROR    (0xC0000006)

EXCEPTION\_INT\_OVERFLOW    (0xC0000095)

EXCEPTION\_PRIV\_INSTRUCTION   (0xC0000096)

2)  完整实例

 HANDLE hProcess = GetCurrentProcess();

 HANDLE hThread = GetCurrentThread();

 bool   bOK    = false;

 vector<tagCrashDebugInfo> arrCallStackInfo;

SymInitialize(hProcess, NULL, TRUE);

 STACKFRAME64 sf = {0};

 DWORD  dwImageType = 0;

 sf.AddrPC.Mode = AddrModeFlat;

 sf.AddrStack.Mode = AddrModeFlat;

 sf.AddrFrame.Mode = AddrModeFlat;

 // 不同的CPU类型，具体信息可查询MSDN

#ifdef \_M\_IX86

 dwImageType = IMAGE\_FILE\_MACHINE\_I386;

 sf.AddrPC.Offset = c.Eip;    //c == 回调函数参数的CONTEXT结构体。

 sf.AddrStack.Offset = c.Esp;

 sf.AddrFrame.Offset = c.Ebp;

#elif \_M\_X64

 dwImageType = IMAGE\_FILE\_MACHINE\_AMD64;

 sf.AddrPC.Offset = c.Rip;

 sf.AddrFrame.Offset = c.Rsp;

 sf.AddrStack.Offset = c.Rsp;

#elif \_M\_IA64

 dwImageType = IMAGE\_FILE\_MACHINE\_IA64;

 sf.AddrPC.Offset = c.StIIP;

 sf.AddrFrame.Offset = c.IntSp;

 sf.AddrBStore.Offset = c.RsBSP;

 sf.AddrStack.Offset = c.IntSp;

#else

#error "Platform not supported!"

#endif

 while (true)

 {

  bOK = false;

  if (!StackWalk64(dwImageType, hProcess, hThread, &sf, &c, NULL, SymFunctionTableAccess64, SymGetModuleBase64, NULL))

  {  break;  }

  if (sf.AddrFrame.Offset == 0)

  {  break;  }

  tagCrashDebugInfo callstackinfo;

  memcpy\_s(callstackinfo.strFunctionName, MAX\_PATH, "N/A", 4);

  memcpy\_s(callstackinfo.strFileName, MAX\_PATH, "N/A", 4);

  memcpy\_s(callstackinfo.strModuleName, MAX\_PATH, "N/A", 4);

  memcpy\_s(callstackinfo.strLine, MAX\_PATH, "N/A", 4);

  // 得到函数名

  BYTE symbolBuffer[sizeof(IMAGEHLP\_SYMBOL64) + MAX\_PATH] = {0};

  IMAGEHLP\_SYMBOL64 \*pSymbol = (IMAGEHLP\_SYMBOL64\*)symbolBuffer;

  pSymbol->SizeOfStruct = sizeof(IMAGEHLP\_SYMBOL64);

  pSymbol->MaxNameLength = MAX\_PATH;

  if (SymGetSymFromAddr64(hProcess, sf.AddrPC.Offset, NULL, pSymbol))

  {

   memcpy\_s(callstackinfo.strFunctionName, MAX\_PATH, pSymbol->Name, MAX\_PATH);

   bOK = true;

  }

  // 得到文件名和所在的代码行

  IMAGEHLP\_LINE64 lineInfo = {0};

  DWORD dwLineDisplacement = 0;

  lineInfo.SizeOfStruct = sizeof(IMAGEHLP\_LINE64);

  if (SymGetLineFromAddr64(hProcess, sf.AddrPC.Offset, &dwLineDisplacement, &lineInfo))

  {

   memcpy\_s(callstackinfo.strFileName, MAX\_PATH, lineInfo.FileName, MAX\_PATH);

   sprintf\_s(callstackinfo.strLine, "%d", lineInfo.LineNumber);

   bOK = true;

  }

  // 得到模块名

  IMAGEHLP\_MODULE64 moduleInfo = {0};

  moduleInfo.SizeOfStruct = sizeof(IMAGEHLP\_MODULE64);

  if (SymGetModuleInfo64(hProcess, sf.AddrPC.Offset, &moduleInfo))

  {

   memcpy\_s(callstackinfo.strModuleName, MAX\_PATH, moduleInfo.ModuleName, MAX\_PATH);

   bOK = true;

  }

  if (bOK)

   arrCallStackInfo.push\_back(callstackinfo);

 }

 SymCleanup(hProcess);

回调函数编写:

 EnterCriticalSection(&CDebugMain::s\_Critical); //注意加锁，否则在多线程中出现异常会进入多次导致写dmp文件冲突。

 vector<tagCrashDebugInfo> vec = DebugInfoFunction(\*pException->ContextRecord);

 char  strInfo[1024] = {0};

 string strTotalInfo;

 sprintf\_s(strInfo, sizeof(strInfo), "-----------------------------------------[Time]:%s--------------------------------------\n", GetPresentTime().c\_str());

 strTotalInfo = strInfo;

 for (UINT i = 0; i < vec.size(); i++)

 {

  if (strcmp(vec[i].strLine, "N/A") || strcmp(vec[i].strFileName, "N/A"))

  {

   sprintf\_s(strInfo, sizeof(strInfo), "[ModuleName]:%s\n[FileName]:%s\n[FunctionName]:%s\n[LineNumber]:%s\n\r",

   vec[i].strModuleName,

   vec[i].strFileName,

   vec[i].strFunctionName,

   vec[i].strLine);

  strTotalInfo += strInfo;

  //break;

  }

 }

 sprintf\_s(strInfo, sizeof(strInfo), "[Reason]:%s\n\r" ,ErrorReason(pException->ExceptionRecord->ExceptionCode).c\_str());

 strTotalInfo +=  strInfo;

 CreateDumpFile(CDebugMain::s\_strDmpFileName.c\_str(), strTotalInfo.c\_str());

 MessageBox(NULL, "Program encountered serious problems, please contact the developer to see dmp log", "Error", MB\_OK);

 LeaveCriticalSection(&CDebugMain::s\_Critical);

注意:

使用VS2005时由于没有禁用CRT(C语言运行库)，则有些异常不能捕获， 解决方法如下:

#ifndef \_M\_IX86

#error "The following code only works for x86!"

#endif

void DisableSetUnhandledExceptionFilter()

{

  unsigned char code[16];

  int size = 0;

  code[size++] = 0x33;

  code[size++] = 0xC0;

  code[size++] = 0xC2;

  code[size++] = 0x04;

  code[size++] = 0x00;

  DWORD dwOldFlag, dwTempFlag;

  VirtualProtect(SetUnhandledExceptionFilter, size, PAGE\_READWRITE, &dwOldFlag);

  WriteProcessMemory(GetCurrentProcess(), SetUnhandledExceptionFilter, code, size, NULL);

  VirtualProtect(SetUnhandledExceptionFilter, size, dwOldFlag, &dwTempFlag);

}

DisableSetUnhandledExceptionFilter函数使用在SetUnhandledExceptionFilter之后。

### dmp文件

崩溃转储是创建一个应用程序崩溃瞬间的状态镜像。

初代的转储文件是记录了进程的虚拟空间中全部内容，但是这样的文件非常大，而且对于不太熟练使用的人员来说冗余信息太多。

xp之后，MicroSoft发明了“minidump”的转储技术，即：小型的，只是包含了必要的线程调用堆栈等信息文件。该文件很小，很容易通过网络发送，同时也可以定制该文件记录的信息，非常的灵活。

生成minidump的函数包含在DbgHelp.dll中。

调试dmp文件：

用vs打开dmp文件， 设置对应的符号路径(pdb)并且设置需要调试的文件(dll或exe)即可（可将pdb与exe或dll文件放在一个目录）。

注意：dmp文件版本要与pdb和调试文件(dll或exe)要同一次编译生成的才行，否则调试信息不会显示。

## 番茄插件

点击 “VAssistX”–>”Visual VAssistX Options”然后选择Suggestions,再点击”Edit VA Snippets”

注释:番茄插件自动添加

VAssistX->Insert VA Snippet->Eidt VA Snippets->Refactor Document Method  
函数模板  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Method: $SymbolName$  
// Description:   
// Author: K@yee   
// Date: $DATE$  
// Returns: $SymbolType$  
// Parameter: $MethodArgName$  
// History:  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
文件头模板  
/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Copyright (C), 2012-2013, CS&S. Co., Ltd.  
// File name: $FILE\_BASE$.$FILE\_EXT$  
// Author: K@yee   
// Version: 1.0   
// Date: $DATE$  
// Description:   
// Others:  
// History:  
// <author> K@yee   
// <time> $DATE$  
// <version> 1.0   
// <desc> build this moudle   
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/