## VC调试入门

### 设置

为了调试一个程序，首先必须使程序中包含调试信息。一般情况下，一个从AppWizard创建的工程中包含的Debug Configuration自动包含调试信息，但是是不是Debug版本并不是程序包含调试信息的决定因素，程序设计者可以在任意的Configuration中增加调试信息，包括Release版本。

为了增加调试信息，可以按照下述步骤进行：

1. 打开Project settings对话框（可以通过快捷键ALT+F7打开，也可以通过IDE菜单Project/Settings打开)
2. 选择C/C++页，Category中选择general ，则出现一个Debug Info下拉列表框，可供选择的调试信息 方式包括：

|  |  |  |
| --- | --- | --- |
| 命令行 | Project settings | 说明 |
| 无 | None | 没有调试信息 |
| /Zd | Line Numbers Only | 目标文件或者可执行文件中只包含全局和导出符号以及代码行信息，不包含符号调试信息 |
| /Z7 | C 7.0- Compatible | 目标文件或者可执行文件中包含行号和所有符号调试信息，包括变量名及类型，函数及原型等 |
| /Zi | Program Database | 创建一个程序库(PDB)，包括类型信息和符号调试信息。 |
| /ZI | Program Database for Edit and Continue | 除了前面/Zi的功能外，这个选项允许对代码进行调试过程中的修改和继续执行。这个选项同时使#pragma设置的优化功能无效 |

1. 选择Link页，选中复选框"Generate Debug Info"，这个选项将使连接器把调试信息写进可执行文件和DLL
2. 如果C/C++页中设置了Program Database以上的选项，则Link incrementally可以选择。选中这个选项，将使程序可以在上一次编译的基础上被编译（即增量编译），而不必每次都从头开始编译。

### 断点

断点是调试器设置的一个代码位置。当程序运行到断点时，程序中断执行，回到调试器。断点是 最常用的技巧。调试时，只有设置了断点并使程序回到调试器，才能对程序进行在线调试。

设置断点：可以通过下述方法设置一个断点。首先把光标移动到需要设置断点的代码行上，然后

1. 按F9快捷键
2. 弹出Breakpoints对话框，方法是按快捷键CTRL+B或ALT+F9，或者通过菜单Edit/Breakpoints打开。打开后点击Break at编辑框的右侧的箭头，选择 合适的位置信息。一般情况下，直接选择line xxx就足够了，如果想设置不是当前位置的断点，可以选择Advanced，然后填写函数、行号和可执行文件信息。

去掉断点：把光标移动到给定断点所在的行，再次按F9就可以取消断点。同前面所述，打开Breakpoints对话框后，也可以按照界面提示去掉断点。

#### 条件断点

条件断点：可以为断点设置一个条件，这样的断点称为条件断点。对于新加的断点，可以单击Conditions按钮，为断点设置一个表达式。当这个表达式发生改变时，程序就被中断。底下设置包括“观察数组或者结构的元素个数”，似乎可以设置一个指针所指向的内存区的大小，但是我设置一个比较的值但是改动 范围之外的内存区似乎也导致断点起效。最后一个设置可以让程序先执行多少次然后才到达断点。

#### 数据断点

数据断点：数据断点只能在Breakpoints对话框中设置。选择“Data”页，就显示了设置数据断点的对话框。在编辑框中输入一个表达式，当这个 表达式的值发生变化时，数据断点就到达。一般情况下，这个表达式应该由运算符和全局变量构成，例如：在编辑框中输入 g\_bFlag这个全局变量的名字，那么当程序中有g\_bFlag= !g\_bFlag时，程序就将停在这个语句处。

#### 消息断点

消息断点：VC也支持对Windows消息进行截获。他有两种方式进行截获：窗口消息处理函数和特定消息中断。

在Breakpoints对话框中选择Messages页，就可以设置消息断点。如果在上面那个对话框中写入消息处理函数的名字，那么 每次消息被这个函数处理，断点就到达（我觉得如果采用普通断点在这个函数中截获，效果应该一样）。如果在底下的下拉 列表框选择一个消息，则每次这种消息到达，程序就中断。

### 值

#### Watch

VC支持查看变量、表达式和内存的值。所有这些观察都必须是在断点中断的情况下进行。

观看变量的值最简单，当断点到达时，把光标移动到这个变量上，停留一会就可以看到变量的值。

VC提供一种被成为Watch的机制来观看变量和表达式的值。在断点状态下，在变量上单击右键，选择Quick Watch， 就弹出一个对话框，显示这个变量的值。

单击Debug工具条上的Watch按钮，就出现一个Watch视图（Watch1,Watch2,Watch3,Watch4），在该视图中输入变量或者表达式，就可以观察 变量或者表达式的值。注意：这个表达式不能有副作用，例如++运算符绝对禁止用于这个表达式中，因为这个运算符将修改变量的值，导致 软件的逻辑被破坏。

#### Memory

由于指针指向的数组，Watch只能显示第一个元素的值。为了显示数组的后续内容，或者要显示一片内存的内容，可以使用memory功能。在 Debug工具条上点memory按钮，就弹出一个对话框，在其中输入地址，就可以显示该地址指向的内存的内容。

#### Varibles

Debug工具条上的Varibles按钮弹出一个框，显示所有当前执行上下文中可见的变量的值。特别是当前指令涉及的变量，以红色显示。

#### 寄存器

Debug工具条上的Reigsters按钮弹出一个框，显示当前的所有寄存器的值。

#### 进程控制

VC允许被中断的程序继续运行、单步运行和运行到指定光标处，分别对应快捷键F5、F10/F11和CTRL+F10。各个快捷键功能如下：

键功能如下：

|  |  |
| --- | --- |
| 快捷键 | 说明 |
| F5 | 继续运行 |
| F10 | 单步，如果涉及到子函数，不进入子函数内部 |
| F11 | 单步，如果涉及到子函数，进入子函数内部 |
| CTRL+F10 | 运行到当前光标处。 |

#### Call Stack

调用堆栈反映了当前断点处函数是被那些函数按照什么顺序调用的。单击Debug工具条上的Call stack就显示Call Stack对话框。在CallStack对话框中显示了一个调用系列，最上面的是当前函数，往下依次是调用函数的上级函数。单击这些函数名可以跳到对应的函数中去。  
  
其他调试手段  
系统提供一系列特殊的函数或者宏来处理Debug版本相关的信息，如下：

|  |  |
| --- | --- |
| 宏名/函数名 | 说明 |
| TRACE | 使用方法和printf完全一致，他在output框中输出调试信息 |
| ASSERT | 它接收一个表达式，如果这个表达式为TRUE，则无动作，否则中断当前程序执行。对于系统中出现这个宏 导致的中断，应该认为你的函数调用未能满足系统的调用此函数的前提条件。例如，对于一个还没有创建的窗口调用SetWindowText等。 |
| VERIFY | 和ASSERT功能类似，所不同的是，在Release版本中，ASSERT不计算输入的表达式的值，而VERIFY计算表达式的值。 |

### 关注

一个好的程序员不应该把所有的判断交给编译器和调试器，应该在程序中自己加以程序保护和错误定位，具体措施包括：

1. 对于所有有返回值的函数，都应该检查返回值，除非你确信这个函数调用绝对不会出错，或者不关心它是否出错。
2. 一些函数返回错误，需要用其他函数获得错误的具体信息。例如accept返回INVALID\_SOCKET表示accept失败，为了查明 具体的失败原因，应该立刻用WSAGetLastError获得错误码，并针对性的解决问题。
3. 有些函数通过异常机制抛出错误，应该用TRY-CATCH语句来检查错误
4. 程序员对于能处理的错误，应该自己在底层处理，对于不能处理的，应该报告给用户让他们决定怎么处理。如果程序出了异常， 却不对返回值和其他机制返回的错误信息进行判断，只能是加大了找错误的难度。

另外：VC中要编制程序不应该一开始就写cpp/h文件，而应该首先创建一个合适的工程。因为只有这样，VC才能选择合适的编译、连接 选项。对于加入到工程中的cpp文件，应该检查是否在第一行显式的包含stdafx.h头文件，这是Microsoft Visual Studio为了加快编译 速度而设置的预编译头文件。在这个#include "stdafx.h"行前面的所有代码将被忽略，所以其他头文件应该在这一行后面被包含。

对于.c文件，由于不能包含stdafx.h，因此可以通过Project settings把它的预编译头设置为“不使用”，方法是：

弹出Project settings对话框

选择C/C++

Category选择Precompilation Header

选择不使用预编译头。

## 调试输出说明

在调试程序的时候，我们可以使用下面这3个函数将调试信息输出到控制台窗口，这对我们调试程序非常有帮助：

echo(text);

warn(text);

error(text)

其中echo函数用于输出标准黑色字体的文本信息到控制台窗口中；warn函数用于输出标准灰色字体的文本信息到控制台窗口中；error函数用于输出标准红色字体的文本信息到控制台窗口中。使用上述3个函数的时候，文本可根据字符串规则进行格式化。

通过正确地使用恰当的输出消息，可以跟踪在脚本中发生的任何事件。一般情况下，人们都希望把最可能出现问题的地方的相关调试信息输出到控制台。您或许想将一个容易识别的标志符置于代码某处，以便在控制台的滚动窗口中找到它；或者输出一些与代码相关的活动的重要信息(这些信息可能是一些变量的输出)。

为了把控制台输出信息转储在console.log文件中，首先，您必须在您的代码中调用setLogMode函数，并确保该函数在程序运行至有问题的代码语句之前被执行。一个更为简单的办法就是：使用-log命令行选项，后接一个空格，然后输入数字0、1或2。其中0表示您不能记录日志；1表示您可以把每一个新会话的内容追加到日志文件的末尾；2表示您可以用新日志内容覆盖以前的日志文件。

## **使用trace函数**

Torque提供了一个很方便的跟踪函数trace。使用它，在执行脚本文件时就可以计算出当前正在执行的是哪一行代码。当您苦苦思索一些逻辑问题的时候，这个功能尤为有效，您只需在感兴趣的代码前面插入trace函数即可。使用trace(true)语句表示跟踪的开始，在您关注的代码段末端插入trace(off)语句，禁用跟踪功能。

您也可以通过打开控制台窗口，输入：

trace(true)；来使跟踪选项可用。

## VC中的TRACE宏：

TRACE宏对于VC下程序调试来说是很有用的东西，有着类似printf的功能；该宏仅仅在程序的DEBUG版本中出现，当RELEASE的时候该宏就完全消失了，从而帮助你调式也在RELEASE的时候减少代码量。

使用非常简单，格式如下：

TRACE("DDDDDDDDDDD");

TRACE("wewe%d",333);

## 在非MFC程序中使用调试宏ASSERT(),VERIFY()和 TRACE()

ASSERT（）被测试它的参数，若参数为0，则中断执行并打印一段说明消息。在 Release 版本的程序中它不起任何作用。

VERIFY（）和 ASSERT（）很相似，区别在于在 Release 版本中它仍然有效（译者注：原作者在这里没有讲清楚，VERIFY（）不会打印说明，只是会对参数表达式求值）。

ASSERT（）使用的时候必须保证参数表达式中不能有函数调用（译者注：ASSERT（）宏在 Release 版本中不对表达式求值），因此对于任何有函数调用的参数表达式，应该使用宏 VERIFY（），以保证表达式中的函数调用在 Release 版本中会被正确求值。

TRACE（）基本上就是函数 printf（）的一个复制品，唯一的区别是它把结果输出到调试窗口。在 Release 版本中，它也是无效的。

这三个宏在 Release 版本中都不会产生任何实质性的影响，它们是否起作用取决于是否定义了预定义了宏 \_DEBUG。这是对 Microsoft Visual C++ 而言，在其它的编译器中可能其它不同的宏。

这里是代码：

// file debug.h

#ifndef \_\_DEBUG\_H\_\_

#define \_\_DEBUG\_H\_\_

#ifdef \_DEBUG

void \_trace(char \*fmt, ...);

#define ASSERT(x) {if(!(x)) \_asm{int 0x03}}

#define VERIFY(x) {if(!(x)) \_asm{int 0x03}} // 译注：为调试版本时产生中断有效

#else

#define ASSERT(x)

#define VERIFY(x) x // 译注：为发行版本时不产生中断

#endif

#ifdef \_DEBUG

#define TRACE \_trace

#else

inline void \_trace(LPCTSTR fmt, ...) { }

#define TRACE 1 ? (void)0 : \_trace

#endif

#endif // \_\_DEBUG\_H\_\_

// file debug.cpp

#ifdef \_DEBUG

#include <stdio.h>

#include <stdarg.h>

#include <windows.h>

void \_trace(char \*fmt, ...)

{

char out[1024];

va\_list body;

va\_start(body, fmt);

vsprintf(out, fmt, body); // 译注：格式化输入的字符串 fmtt

va\_end(body); // 到输出字符串 ou

OutputDebugString(out); // 译注：输出格式化后的字符串到调试器

}

#endif

译者续：一点小扩展

大家可以看到宏 TRACE（）的最后，调用的是 OutPutDebugString（）函数，只能将信息输出到调试器窗口中，但我们同样也可以实现 MFC 中的弹出式窗口，只要用 MessageBox（）函数输出就可以了。（不过……好像样子也不一样哎！）