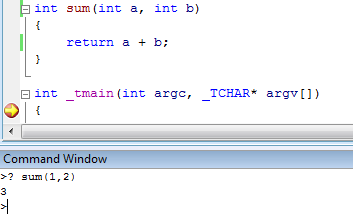
### 命令窗口

这个小技巧是由chaau建议的，它能节省你大量的时间。Visual studio支持一个命令窗口，你可以通过菜单View->Other Windows->Command Window 打开。你可以在窗口里输入不同命令使调试自动化。比如，可以通过很简单的命令去的测试MFC的COleDateTime变量。

[](http://jingyan.baidu.com/album/77b8dc7f225b466174eab604.html?picindex=11)

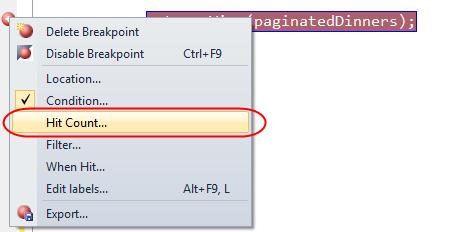
### 跳到当前光标处（Ctrl+F10）

我经常看到人们为了到达目标代码位置，而在程序中早早设定了断点，然后反复地按F10/F11，一步步走到目标代码处。当程序员的确需要仔细观察每一步的状态变化时， F10/F11是合理的选择。然而多数情况下，人们只想快速到达他们真正关心的代码处，这时候F10/F11就不是最佳选择了。

这时，你应该利用“跳到当前光标处”这个功能。先把光标定位在要测的目标代码行上，再同时按Ctrl和F10，被测程序将直接跳到该行停下。你再也不用按许多次F10/F11了。即使目标代码位于独立的类或方法中，你仍然可以从当前正在检查的地方跳过去。

### 记录到达断点次数

有时你希望，只有当第N次满足条件的运行到达断点时，才中断程序运行。例如：当第五次返回少于10份晚餐的查询结果时，中断程序运行。  
可以通过右击断点，然后在弹出菜单上选择“Hit count…”菜单命令实现。

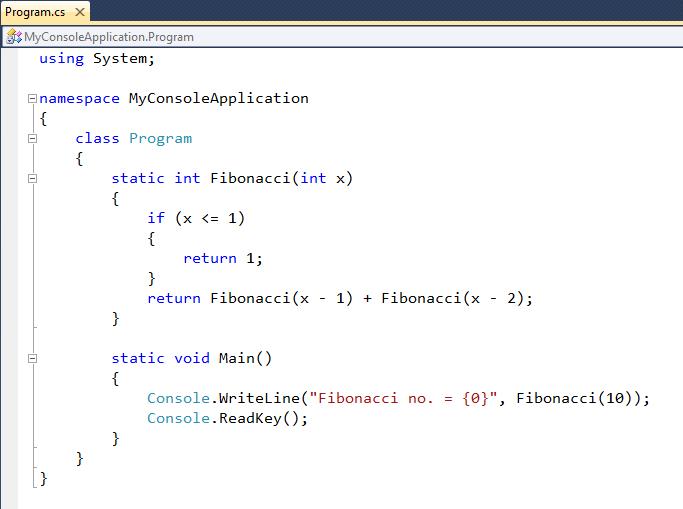


这时系统弹出一个对话框，它允许你指定：（1）当满足条件，而且进入断点的累计次数等于N时，断点命中一次。（2）当满足条件，而且进入断点的累计次数是N的倍数时，断点命中一次。（3）当满足条件，而且进入断点的累计次数大于N时，每次命中断点。

### 跟踪点—进入断点时的自定义操作

许多人不知道“跟踪点(TrackPoints)”这个调试功能。“跟踪点“是种特殊的断点，当它被命中时，它会触发一系列自定义操作。如果你想观察程序的行为，而又不想中断调试的时候，这个功能尤其有用。

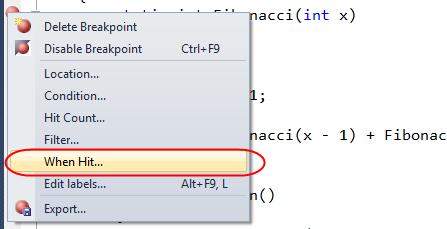
我将用一个简单的控制台程序来演示如何使用“跟踪点”。如下是斐波那契数列的一个递归实现：



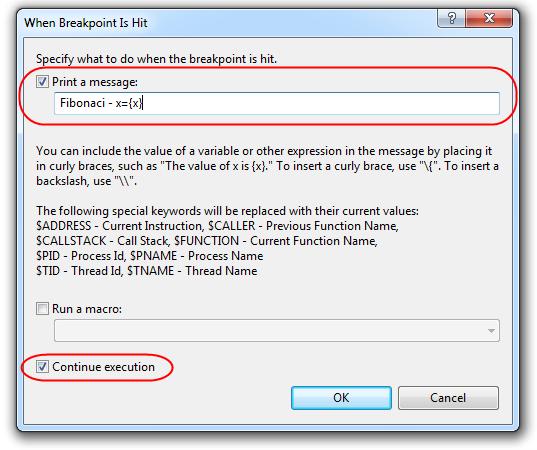
以上程序中，我们使用Console.WriteLine() 输出针对特定输入值生成的最终斐波那契数列。如果希望在调试器里观察操作中每一次递归运算后的数列而又不实际中断程序运行，该怎么办呢？“跟踪点”可以轻松实现。

1. **设置跟踪点**

你可以在特定的行上，按F9加跟踪点。然后  
　　右击断点，在上下文菜单中选择“When Hit…”：



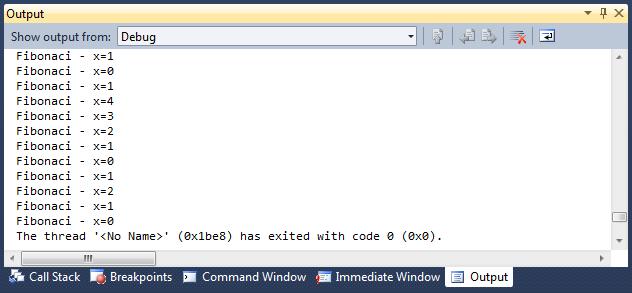
在弹出对话框上，你可以设置命中该断点时，所触发的事件。



在上面例子中，我们设定一旦命中断点时就打印追踪信息。注意，我们已经把局部变量“x”的值，作为追踪信息的一部分输出。局部变量可以通过{变量名}语法输出。你还可以利用系统内置的命令（$CALLER, $CALLSTACK, $FUNCTION等等），在追踪信息中输出常用的调试值。

在上例中，我们同时选中了底端的“continue execution“选项，这说明我们不希望程序中断调试状态，而是继续运行。唯一的不同是：每次断点条件满足时，我们的自定义追踪信息都将被输出。

现在当我们运行程序时，会发现自定义追踪信息自动显示在Visual Studio的“输出“窗口里。这让我们很容易看到程序的递归调用过程：



你也可以选择往应用程序中添加一个自定义追踪信息的监听器。这时追踪点的输出信息将通过它输出，而不是Visual Studio的“输出“窗口。

1. **跟踪点—运行自定义的宏**

上周，我在伦敦演讲时，有听众问道：当命中跟踪点时，能否自动输出所有的局部变量？

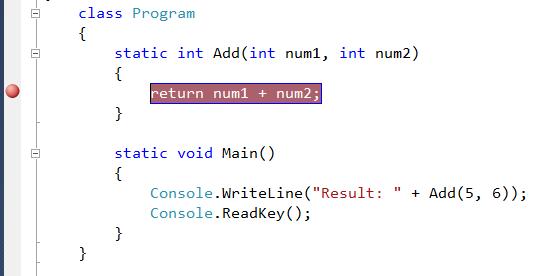
Visual Studio中并没有这样的内置功能，但我们可以写一个自定义宏来实现，然后在命中跟踪点时调用该宏。这个的实现需要先打开Visual Studio的宏编辑器（工具->宏->宏IDE菜单命令），然后在项目资源管理器的MyMacros节点下选择一个模块或创建新模块（如：加个名为“UsefulThings”的模块），再把下面的VB宏代码贴到模块中并保存。

Sub DumpLocals()  
Dim outputWindow As EnvDTE.OutputWindow  
outputWindow = DTE.Windows.Item(EnvDTE.Constants.vsWindowKindOutput).Object  
  
Dim currentStackFrame As EnvDTE.StackFrame  
currentStackFrame = DTE.Debugger.CurrentStackFrame  
  
outputWindow.ActivePane.OutputString(“\*Dumping Local Variables\*” + vbCrLf)  
For Each exp As EnvDTE.Expression In currentStackFrame.Locals  
outputWindow.ActivePane.OutputString(exp.Name + ” = ” + exp.Value.ToString() + vbCrLf)  
Next  
End Sub

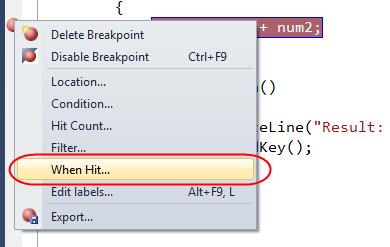
上述宏代码将循环当前的堆栈，把所有的局部变量输出到“输出”窗口。

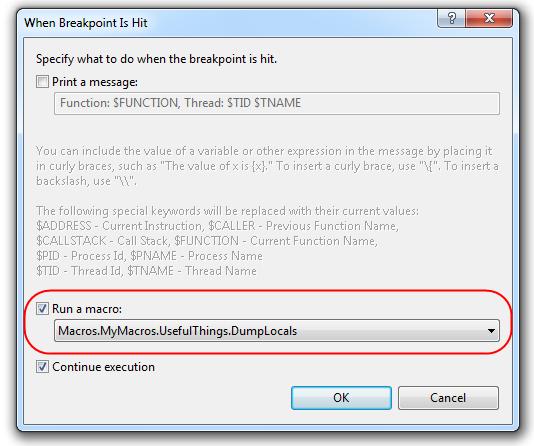
1. **使用自定义的“DumpLocals”宏**

然后，我们可以在如下的一个简单程序中使用刚定制的“DumpLocals”宏了：



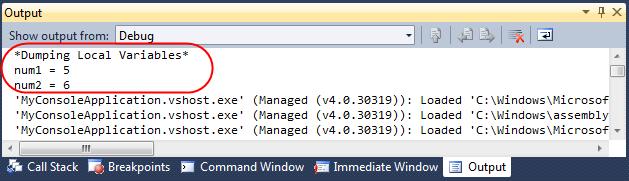
　　上述代码中，我们用F9在“Add”方法的返回值处加了个断点，然后右击断点，在弹出菜单上选择“When hit”。



将显示如下对话框。和之前不一样， 我们不选“Print a message”选项，也不手工设定需要输出的变量；而是选择“Run a marco”复选框，并指定到我们上面创建的UsefulThings.DumpLocals宏上：  


为了使程序能在命中跟踪点后仍继续运行，我们将继续选中“continue execution”复选框。

1. **运行程序**

现在按F5运行程序，当“Add”方法被调用时，我们会在Visual Studio的“输出”窗口中看到如下结果。注意命中跟踪点时，宏会自动列出每个局部变量的名称和值:  


### VS调试main的参数

在VS下断点调试,但是程序Main需要输入参数才能运行,怎样设置才能单步调试啊? 如:我在cmd窗口中键入 test.exe aa.bmp 才能运行。

调试方式：

VS"工程属性/Debug标签/启动项/命令行参数(Program arguments)，输入aa.bmp

《[Vistual Studio原生开发的10个调试技巧](http://blog.jobbole.com/45249/" \t "http://blog.csdn.net/Martin_chen2/article/details/_blank)》http://blog.jobbole.com/45249/

### Watch窗口中的伪变量

Watch窗口或QuickWatch对话框提供一些特定的（调试器可识别的）变量，被称为伪变量。文档包含以下：

$tid—–当前线程的线程ID

$pid——进程ID

$cmdline———-启动程序的命令行字符串

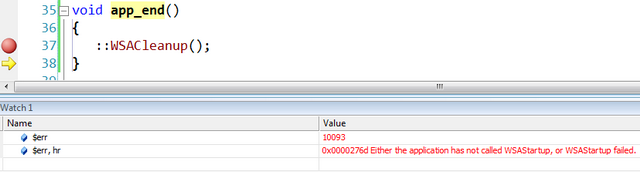
$user———-正在运行程序的账户信息

$registername—–显示寄存器registername 的内容

不管怎么样，关于最后一个错误的伪变量是非常有用的：

$err——–显示最后一个错误的错误码

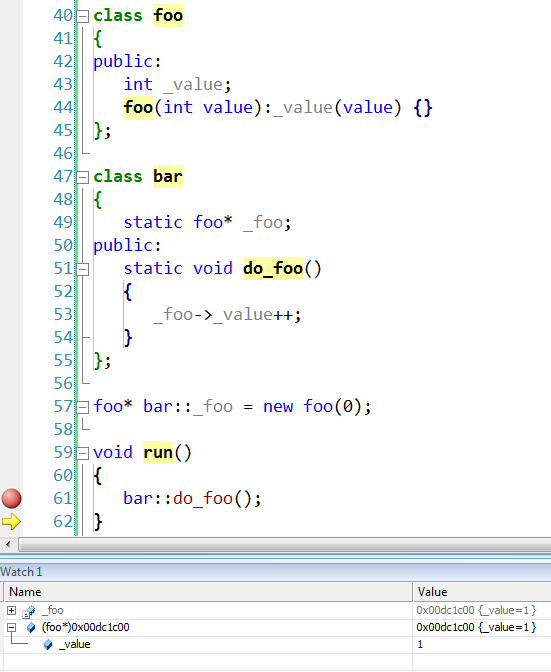
$err,hr—显示最后一个错误的错误信息

https://msdn.microsoft.com/en-us/library/ms164891.aspx

### 符合越界后查看堆对象

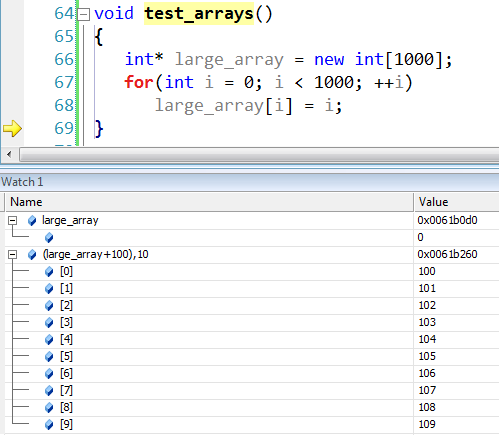
有时候，在调试符号越界后，你还想查看对象的值，这个时候，watch窗口中的变量是被禁用的，不能再查看（也不能更新），尽管对象仍然存在。你如果知道对象的地址，可以继续充分地观察它。你可以将地址转换为该对象类型的指针，放在watch窗中。

下面的例子中，当单步跳出do\_foo()之后，\_foo不能再被访问。但是，将它的地址转换为foo\*后，就可以继续观察这个对象。

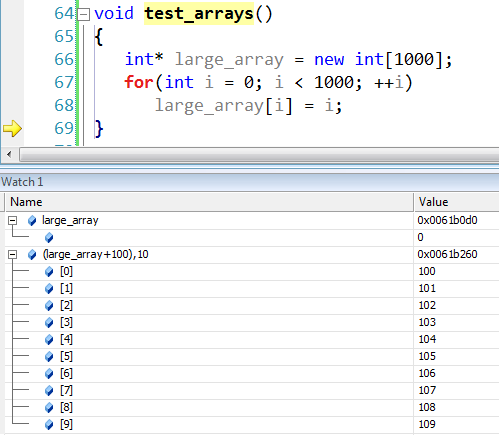


### 查看数组的值

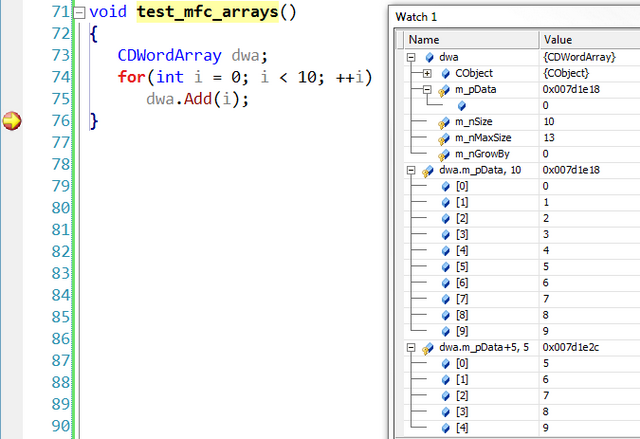
如果你在操作一个很大的数组(我们假设至少有几百个元素吧,但是可能更少),在Watch窗口中展开数组,查找一些特定范围内的元素很麻烦,因为你要不停地滚动.如果数组是分配在堆上的话,你甚至不能在watch窗口中展开数组元素.对此，有一个解决办法。你可以使用(array+ <offset>),<count> 去查看从<offset>位置开始的特定范围的<count>元素（当然，这儿的数组是你的实际对象）。如果想查看整个数组，可以简单使用array,<count>.

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq05.png)

如果你的数组是在堆上，你可以在watch窗口中将它展开，但是要查看某个特定范围的值，用法稍有不同：((T\*) array + <offset>),<count>(注意这种用法对于堆上的多维数组也有效)。但是这种情况下，T是指数组元素的类型。

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq06.png)

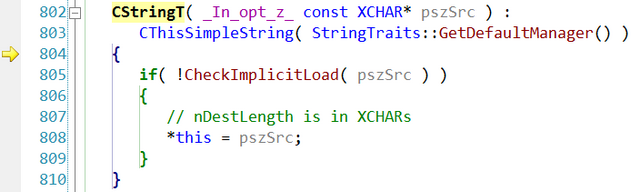
如果你在用MFC，并使用其中的’array’容器，像 CArray, CDWordArray,CStringArray等等。你当然可以使用同样的过滤方法。除此之外，你必须查看array的m\_pData成员，它是保存数据的真实缓存。

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq07.png)

### 避免进入不必要的函数

很多时候，你在调试代码时可能会进入到你想跳过的函数，像构造函数，赋值操作或者其他的。其中最困扰我的是CString构造函数。下面是一个例子，当你准备单步执行take\_a\_string()函数时，首先进入到CString的构造函数。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | void take\_a\_string(CString const &amp;text)  {  }    void test\_string()  {     take\_a\_string(\_T(&quot;sample&quot;));  } |

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq08.png)

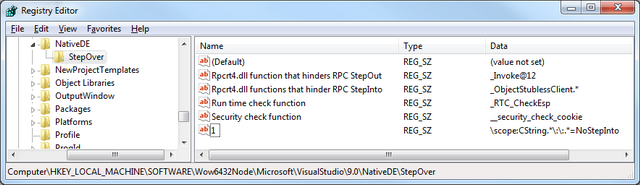
幸运的是可以告诉调试器去跳过哪些方法，类或者整个命名空间。实现它的方法也已经改变了,回到使用VS6的日子，通常是通过autoexp.dat文件来指定的。Vistual Studio 2002改成了使用注册表设置。想要跳过一些函数，你需要在注册表里添加一些值（详情如下）：

实际位置取决于你使用的Vistual Studio版本和操作系统平台（x86或x64，因为注册表只能在64位的Windows下浏览）

值的名字是数字，代表规则的优先级；数字越大，优先级越高。

值数据是一个正则表达式的REG\_SZ值，用于指定怎样过滤和执行。

为了避免进入任何CString方法，我添加了下面的规则：

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq09.png)

有了这个，即使你强制进入上例中的take\_a\_string()，调试器也会跳过CString的构造函数。

[使用Visual C++调试器怎样避免进入函数](https://blogs.msdn.microsoft.com/andypennell/2004/02/06/how-to-not-step-into-functions-using-the-visual-c-debugger/)

[使用AutoExp.dat调整调试器](https://www.codeguru.com/cpp/v-s/debug/article.php/c1281/Tune-the-debugger-using-AutoExpdat.htm)

### 从代码启动调试器 Launch the debugger from code

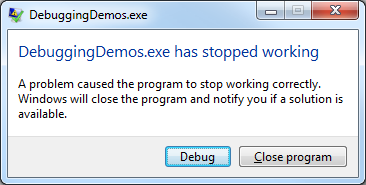
你可能很少需要将调试器附加到程序中，但你不能在Attach窗口这样做（可能因为中断发生太快而没有捕获到），你也不能一开始就在调试器中启动程序。你可以在程序中产生中断给调试器一个机会通过调用内部的\_degbugbreak()来附加。

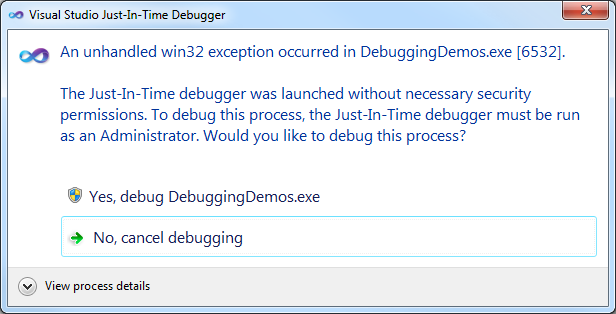
|  |  |
| --- | --- |
| 1  2  3  4 | void break\_for\_debugging()  {     \_\_debugbreak();  } |

实际上还有其他的方法来完成,例如触发中断3,但这仅仅适用于x86平台（C++64位不再支持ASM）。另外还有DebugBreak()函数，但它的使用不怎么简便，所以这里推荐使用内部方法。

|  |  |
| --- | --- |
| 1 | \_\_asm int 3; |

   程序运行内部方法时会停止运行，这时你就有机会将调试器附加到该进程。

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq10.png)

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq11.png)

[内部方法\_debugbreak](http://msdn.microsoft.com/en-us/library/f408b4et.aspx)

[任何时候都离不开设置断点和断言](http://www.codeproject.com/Articles/10022/DebugBreak-and-ASSERTs-that-work-always-anywhere)

[Visual Studio 20005/2008的调试，第四部分：为调试器设置代码](http://www.cprogramming.com/tutorial/visual_studio_debugging_code_setup.html)

### 隔离内存泄漏

内存泄漏是在原生开发中的一个很重要的问题，要检测内存泄漏是一个很严峻的挑战，尤其是在大型项目中。Vistual Studio可以提供检测内存泄漏的报告，还有其他的一些应用程序（免费的或商业的）也可以帮助你检测内存泄漏.有些情况下，在一些内存分配最终会导致泄漏时,可以使用调试器去中断。但是你必须找到可再现的分配编号（尽管没那么容易）。如果能做到这一点，执行程序时调试器才会中断。

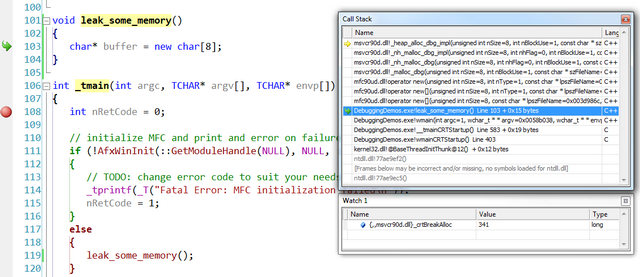
 我们来看下面的代码，分配了8个字节，却一直没释放分配的内存。Visual Studio提供了造成内存泄漏的对象的报告，多运行几次，会发现一直是同一个分配编号（341）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void leak\_some\_memory()  {     char\* buffer = new char[8];  }    Dumping objects -&gt;  d:\marius\vc++\debuggingdemos\debuggingdemos.cpp(103) : {341} normal block at 0x00F71F38, 8 bytes long.  Data: &lt;        &gt; CD CD CD CD CD CD CD CD  Object dump complete. |

在一个特定的（可复现的）位置中断的步骤如下：

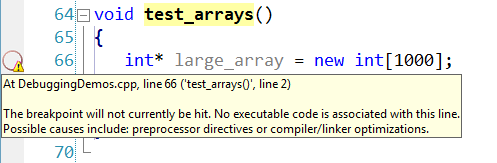
1. 确定你有足够的关于内存泄漏的报告模式（参考 [使用CRT库检测内存泄漏](https://msdn.microsoft.com/zh-cn/library/x98tx3cf.aspx)）
2. 多次运行程序直到你能在程序运行结束后的内存泄漏报告里找到一个可复现的分配编号，例如上个例子中的（341）
3. 在程序一开始的地方设置一个断点以便你能够尽早地进行中断。
4. 当最初的中断发生时，watch窗口的Name栏里会显示：{,,msvcr90d.dll}\_crtBreakAlloc，在Value栏里写入你想要查找的位置编号
5. 继续调试（F5）
6. 程序执行到指定位置会停止，你可以使用调用栈被指引找到被该位置触发的那段代码。

遵循这些步骤， 在上个例子中，使用分配的编号（341）就可以识别内存泄漏的起因。



### 调试发行版

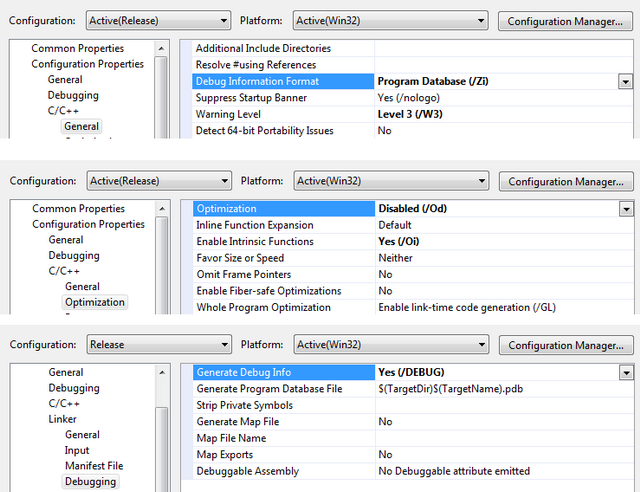
调试和发布是两个不同的目的。调试配置是用于开发的，而发布配置，顾名思义，是用来作为程序的最终版本，因为它必须严格遵循发布的质量要求，该配置包含优化部分和调试版本的中断调试的设置。而且，有时候，要像调试调试版本一样去调试发行版。要做到这一点，你需要在配置里做一些改变。但是这种情况下，你就不再是在调试发行版，而是调试和发行的混合版。

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq15.png)

你还应该做一些事儿，以下是必须要做的：

1. 配置C/C++ >General>Debug Information Format 应该为 “Program Database(/Zi)”
2. 配置C/C++ >Optimization>Optimization 应该为”Disabld(/Od)”
3. 配置Linker>Debugging>Generate Debug Info 应该为”Yes/(DEBUG)”

如图所示：

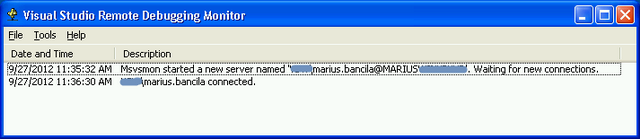


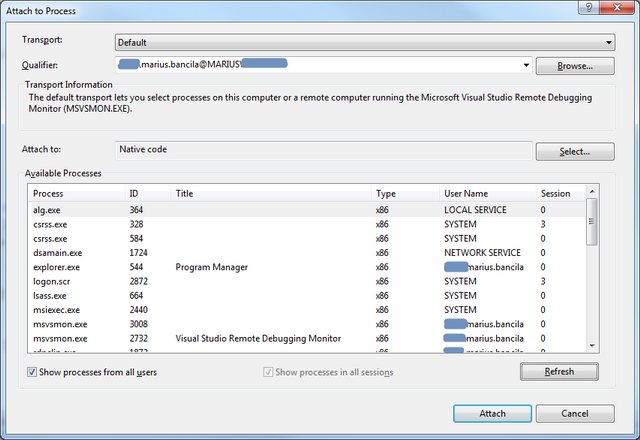
[怎样调试发行版](http://msdn.microsoft.com/en-us/library/fsk896zz.aspx)

### 远程调试

另一个重要的调试就是远程调试，这是一个更大的话题，多次被提到，这里我只做一下简单的概括：

1. 你需要在远程机器上安装远程调试监控
2. 远程调试监控必须以管理员身份运行，并且用户必须属于管理员组
3. 在你运行监控时，会开启一个新的服务，该服务的名字必须用Visual Studio的Attach to Progress窗口的Qualifier组合框的值。

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq17.png)

[](http://jbcdn2.b0.upaiyun.com/2013/08/10jq18.png)

1. 远程和本地机器上的防火墙必须允许Visual Studio和远程调试监控之间能够通信
2. 想要调试，PDB文件是关键；为了能够让VisualStudio自动加载它们，必须满足以下条件：

1）本地的PDB文件必须可用（在远程机器的相同路径下放置一个对应的模块）。

 2) 远程机器上的托管PDB文化必须可用。

远程调试监控下载：

* [Visual Studio 2008 Service Pack 1 Remote Debugger](http://www.microsoft.com/en-us/download/details.aspx?id=10647)
* [Microsoft Visual Studio 2010 Remote Debugger](http://www.microsoft.com/en-us/download/details.aspx?id=475)

Vs调试下 [http://blog.jobbole.com/45447/](http://blog.jobbole.com/45447/" \t "http://blog.csdn.net/Martin_chen2/article/details/_blank)