VS debug 入门指南

本教程仅为常用 debug 指令入门指南

1 导言

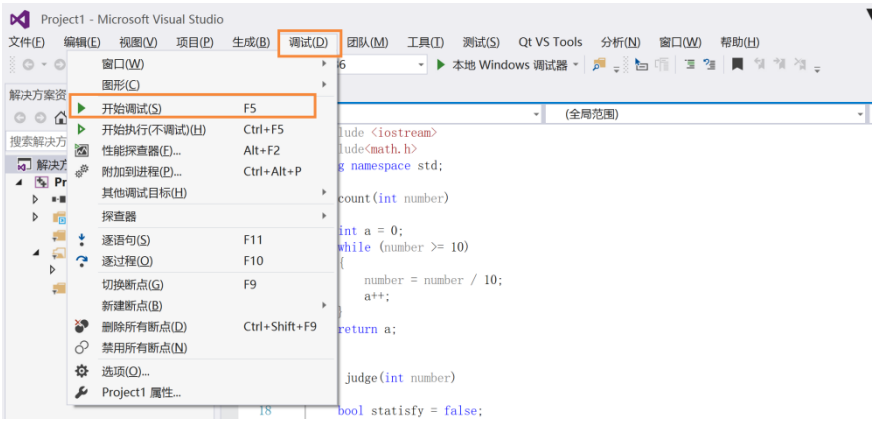
Debug 是指发现缺陷并改正的过程。如果代码中存在缺陷，我们首先要识别造成缺陷的根本原因（root cause），这个过程就称作调试（debugging）。找到根本原因后，就可以修正缺陷

2 如何启动调试？

可以通过 VS 的调试（Debug）菜单启动调试。

点击调试菜单下的**“启动调试”或者按 F5 键启动**

如果你已经在代码中加入了断点，那么会自动执行到断点处。



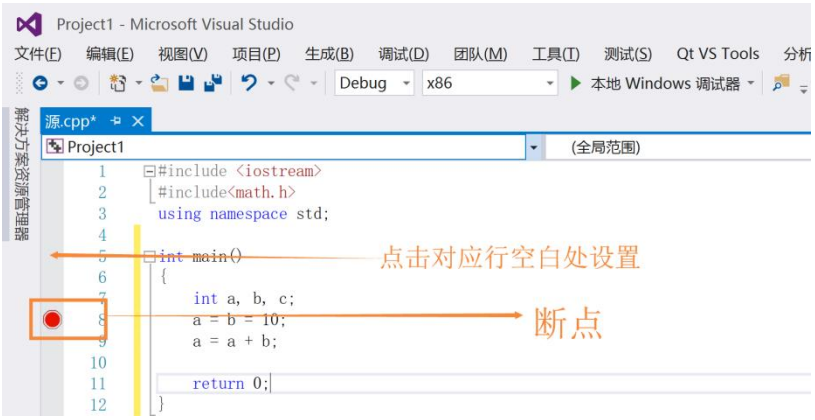
通常我们通过在可能存在问题代码处加断点来启动调试。因此，我们从断点开始讲起。

3 断点（Breakpoints）

断点用于通知调试器何时何处暂停程序的执行。

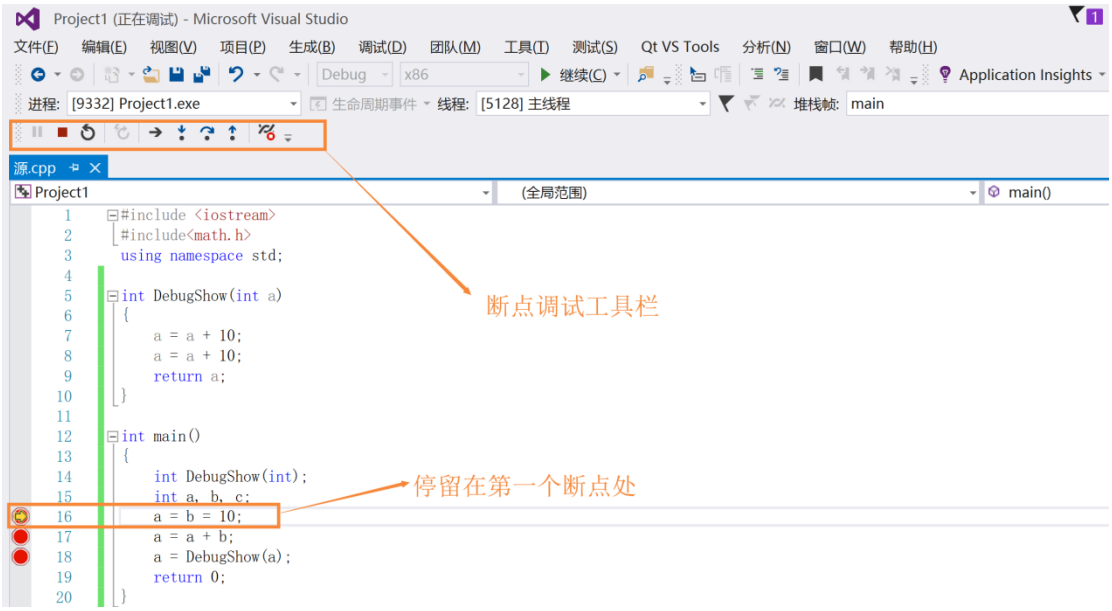
通过**点击左边栏或者按 F9 键**在当前行添加断点。

在加断点之前，你需要初步预判你的代码在哪里出错，并设置合理的断点位置以供调试。



3.1 使用断点进行调试

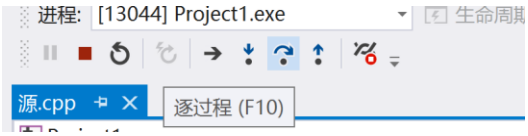
你已经在你想要暂停执行的地方设置了断点。现在按 F5 键启动调试，当程序执行到断点处时，自动暂停执行。此时你有多种方式来检查代码。命中断点（hit the breakpoint）后，加断点的行变为黄色，意指下一步将执行此行。



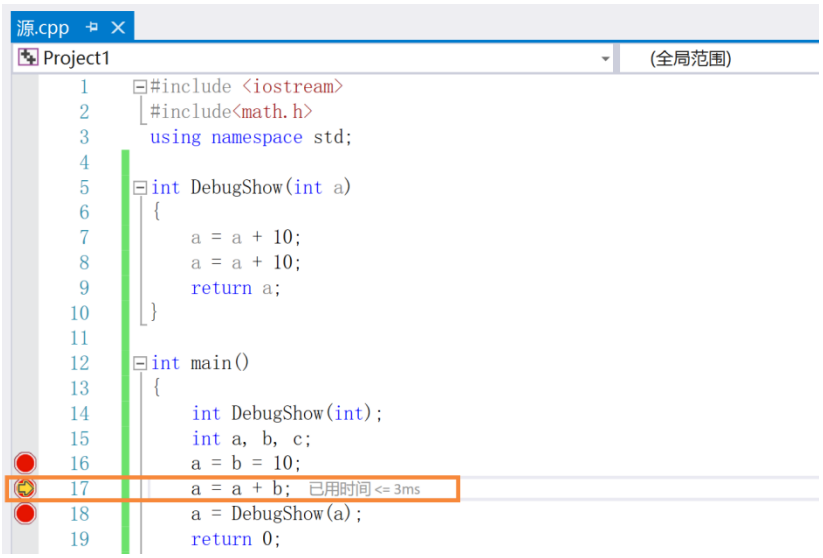
在中断模式下，你有多条可使用的命令，使用相应命令进行进一步的调试。

3.1.1 逐过程（Step Over）

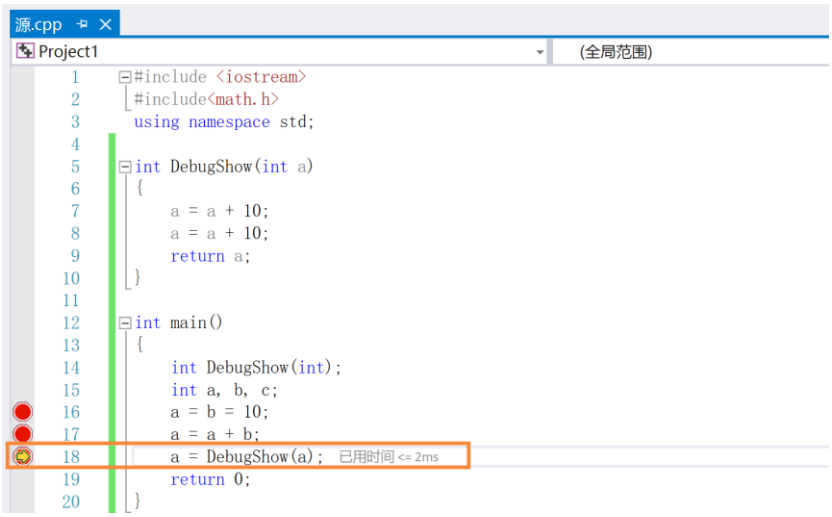
调试器执行到断点后，你可能需要一条一条的执行代码。
F10 命令用于一条一条的执行代码，这将执行当前高亮的行，然后暂停。
如果在一条方法调用语句高亮时按 F10，执行会停在调用语句的下一条语句上。Step Over 会一次运行完当前函数调用。



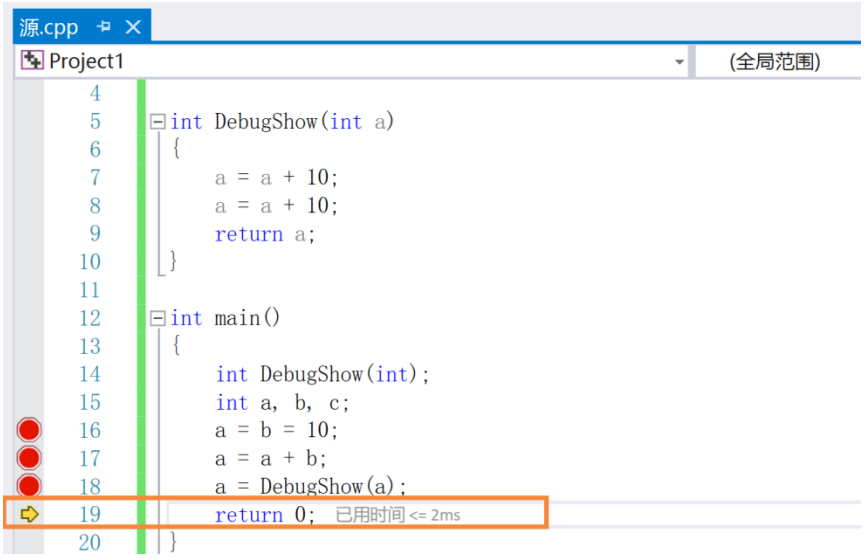
请看示例：
第一次 F10



第二次 F10

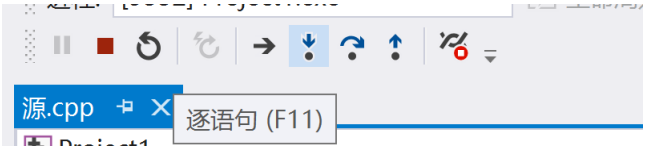


第三次 F10

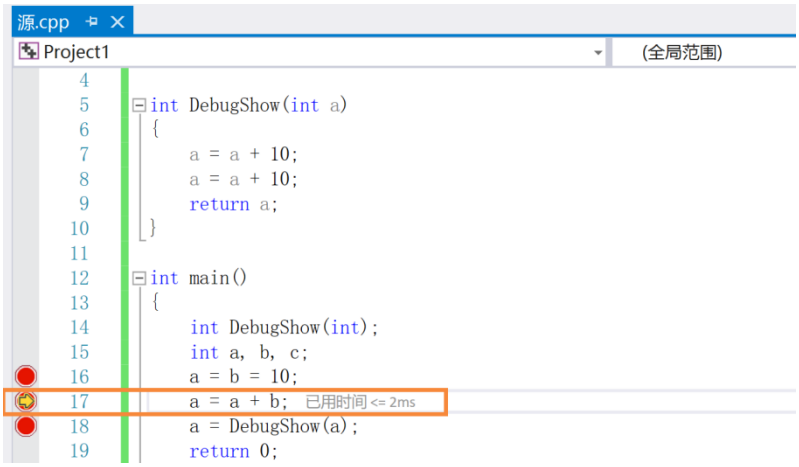


3.1.2 逐语句（Step Into）

它与 Step Over 相似。**唯一的不同是，如果当前高亮语句是方法调用，调试器会进入方法内部。**快捷键是”F11“。

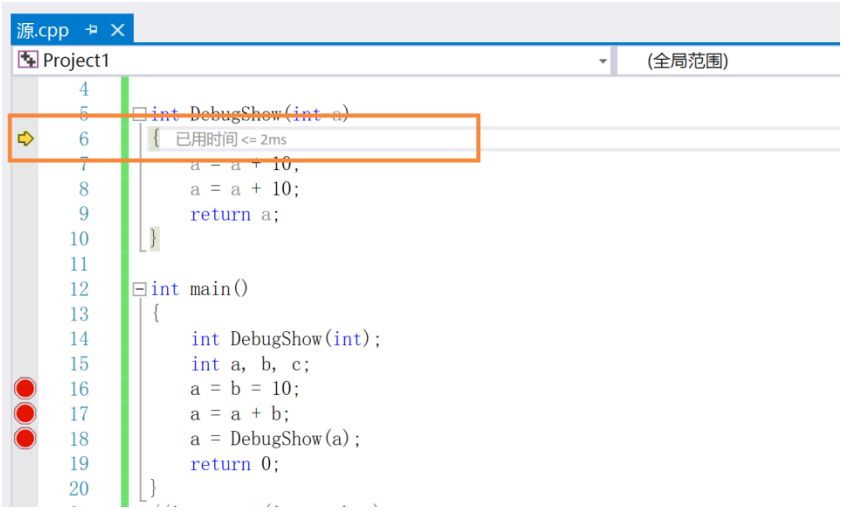


请看示例，注意 F11 与 F10 的不同：
第一次 F11



第二次 F11

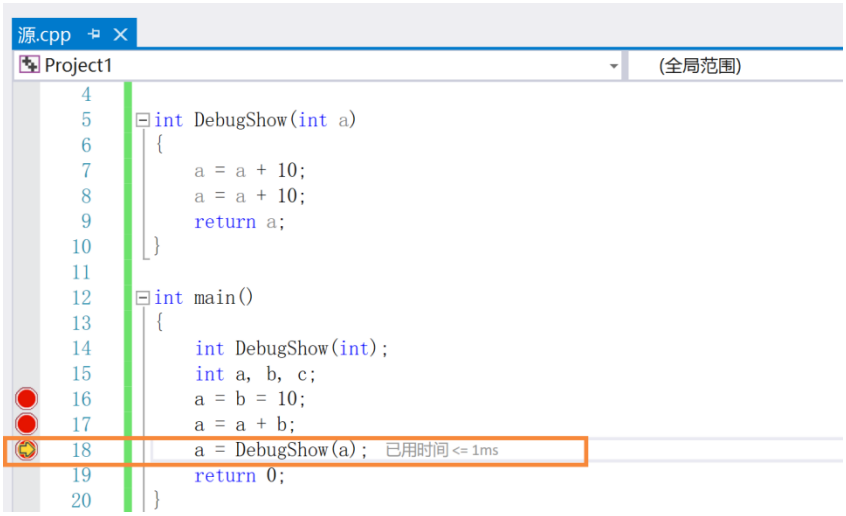
第三次 F11



3.1.3 跳出（Step Out）

当你在一个方法内部调试时会用到它。如果你在当前方法内按 **Shift + F11**，调试器会完成此函数调用的执行，之后在调用此函数的语句的下一条语句处暂停。

接上图



3.1.4 继续（Continue）

它像是重新执行你的程序。它会继续程序的执行直到遇到下一个断点。快捷键是“**F5**”。

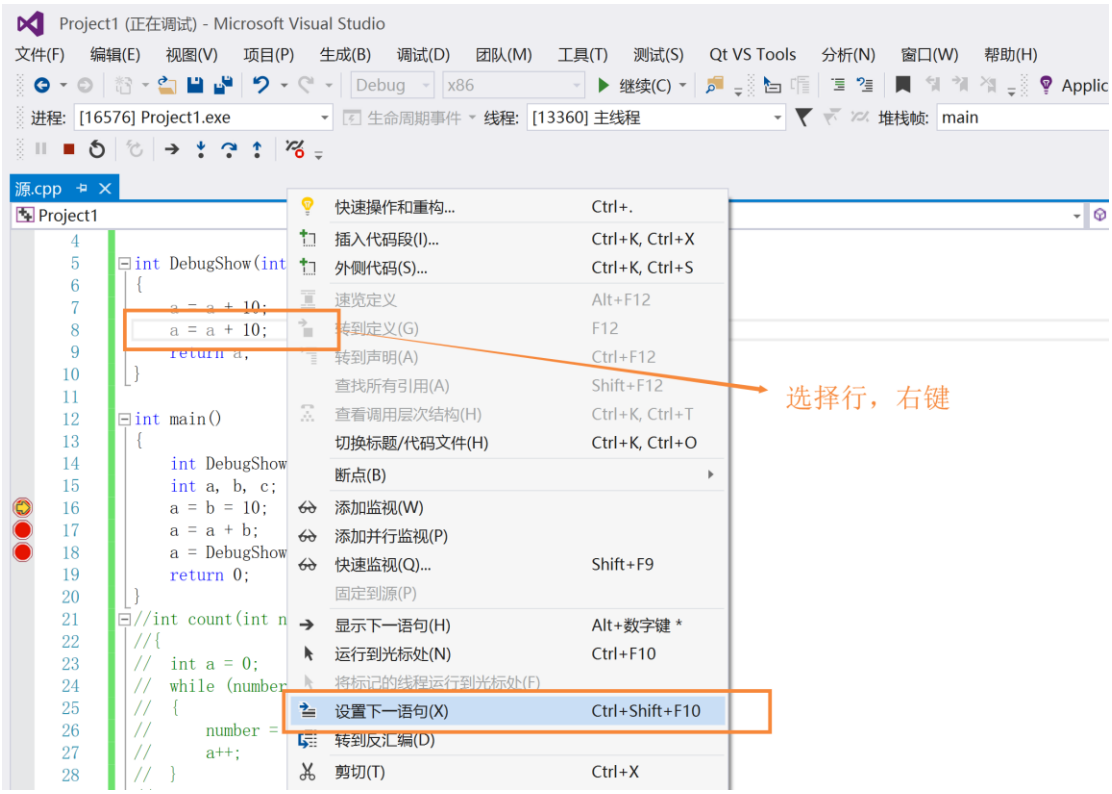
3.1.5 设置下一语句（Set Next Statement）

设置下一语句允许你在调试的时候改变程序的执行路径。

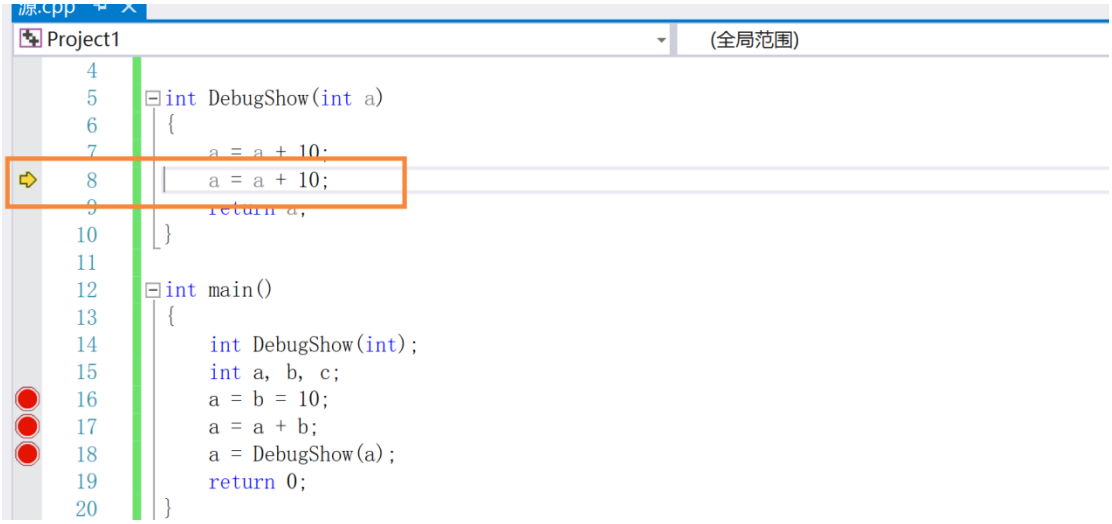
如果你的程序在某一行处暂停而且你想改变执行路径，跳到指定行，**在这一行上右击，在右击菜单中选择“设置下一语句”**。这样程序就会转到那一行而不执行先前的代码。

这在如下情况中非常有用：当你发现代码中某些行可能会导致程序的中断（break）而你不想让程序在那个时候中断。

快捷键是 Ctrl + Shift + F10。


Project1 (正在调试) - Microsoft Visual Studio
文件(F) 编辑(E) 视图(V) 项目(P) 生成(B) 调试(D) 团队(M) 工具(T) 测试(S) Qt VS Tools 分析(N) 窗口(W) 帮助(H)
Debug x86 继续(C)
进程: [16576] Project1.exe 生命周期事件 线程: [13360] 主线程 堆栈帧: main
源.cpp
Project1
int DebugShow(int
{
a = a + 10;
a = a + 10;
return a;
}
int main()
{
int DebugShow
int a, b, c;
a = b = 10;
a = a + b;
a = DebugShow
return 0;
}
//int count(int n
//{
// int a = 0;
// while (number
// {
// number =
// a++;
// }
快速操作和重构... Ctrl+.
插入代码段(I)... Ctrl+K, Ctrl+X
外侧代码(S)... Ctrl+K, Ctrl+S
速览定义 Alt+F12
转到定义(G) F12
转到声明(A) Ctrl+F12
查找所有引用(A) Shift+F12
查看调用层次结构(H) Ctrl+K, Ctrl+T
切换标题/代码文件(H) Ctrl+K, Ctrl+O
断点(B)
添加监视(W)
添加并行监视(P)
快速监视(Q)... Shift+F9
固定到源(P)
显示下一语句(H) Alt+数字键 *
运行到光标处(N) Ctrl+F10
将标记的线程运行到光标处(F)
设置下一语句(X) Ctrl+Shift+F10
转到反汇编(D)
剪切(T) Ctrl+X
选择行，右键

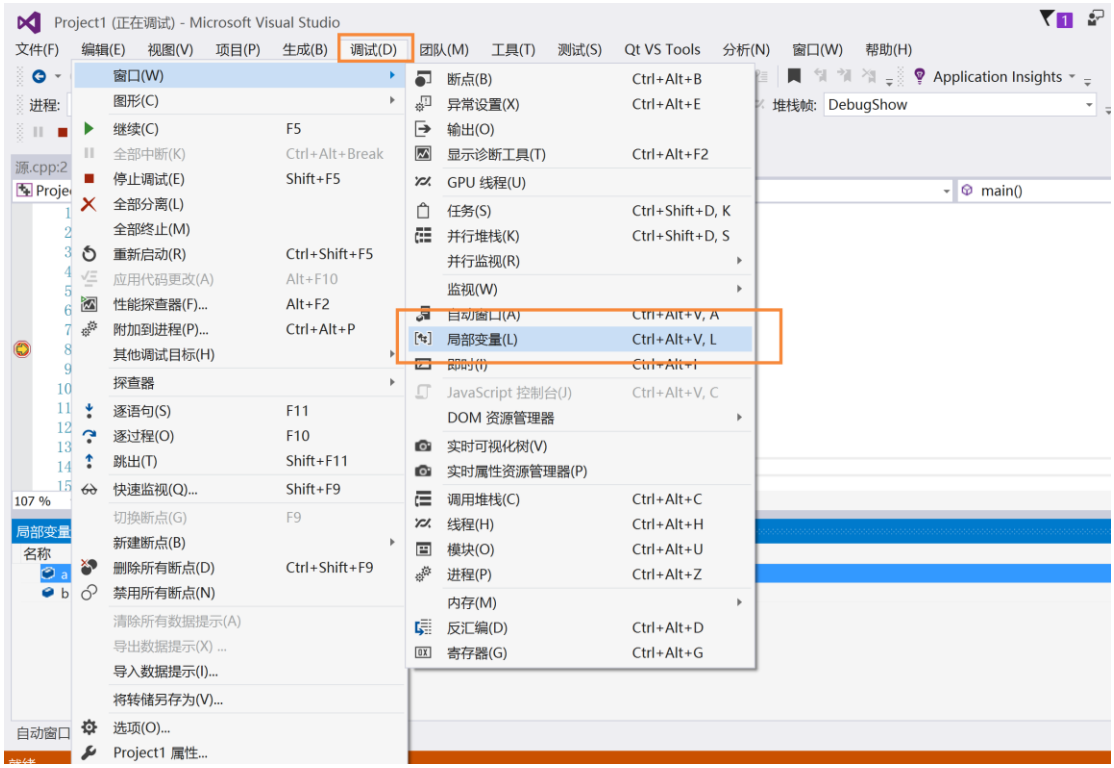
注意箭头

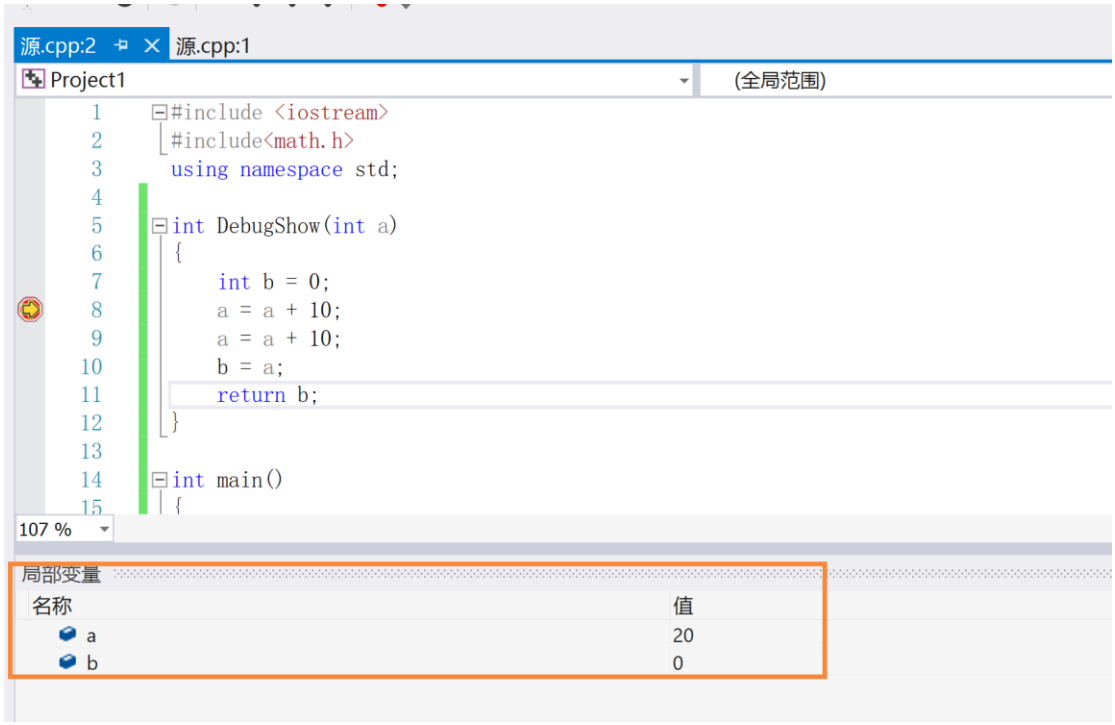

Project1
(全局范围)
int DebugShow(int a)
{
a = a + 10;
a = a + 10;
return a;
}
int main()
{
int DebugShow(int);
int a, b, c;
a = b = 10;
a = a + b;
a = DebugShow(a);
return 0;
}

4 监视窗口（Watch Windows）

4.1 局部变量（Locals）

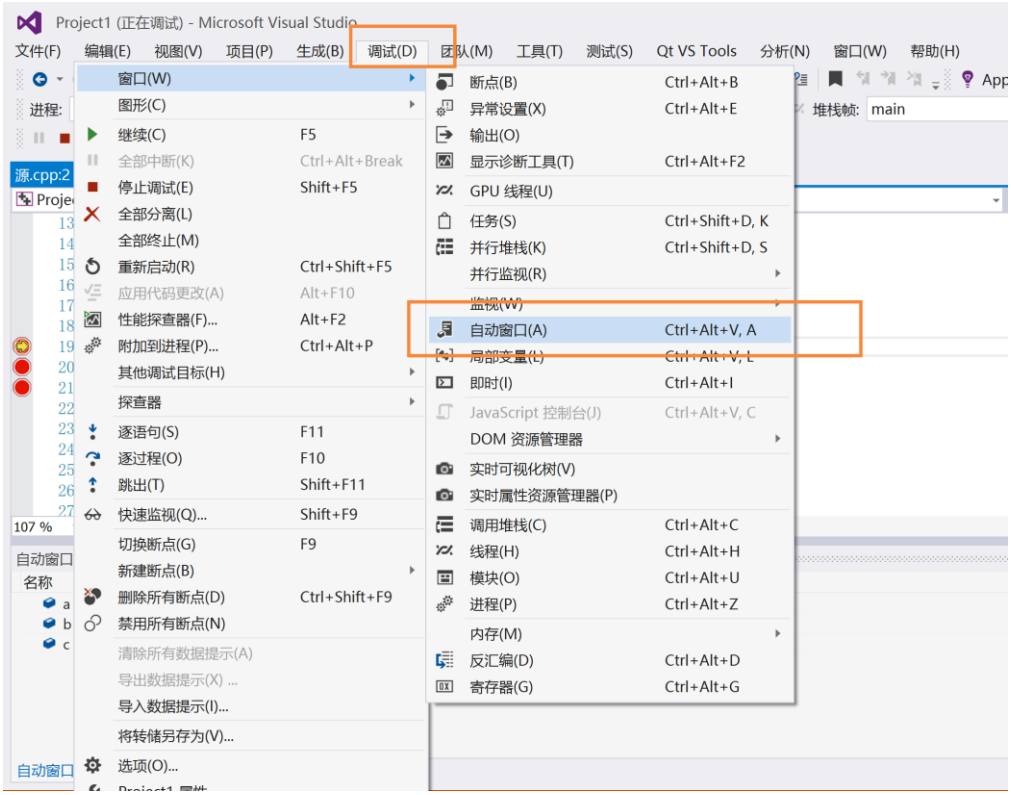
列出当前函数中的所有变量。当调试器停在某特定断点并打开 Autos 窗口时，将展示当前范围中与此值相关的变量。

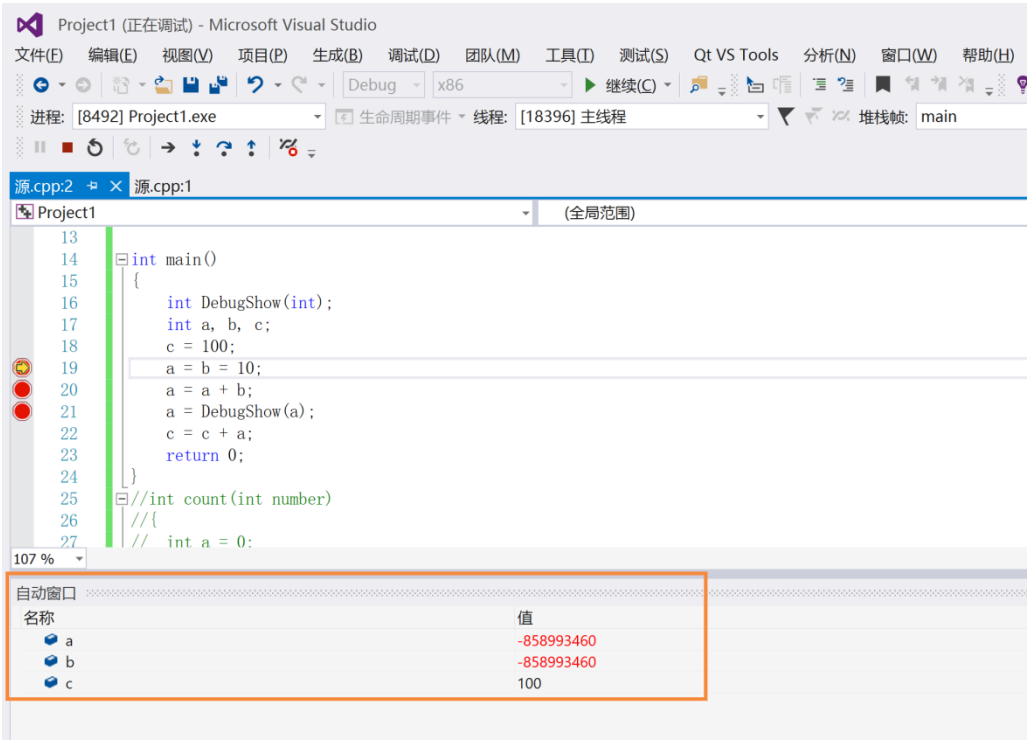




4.2 自动窗口（Autos）

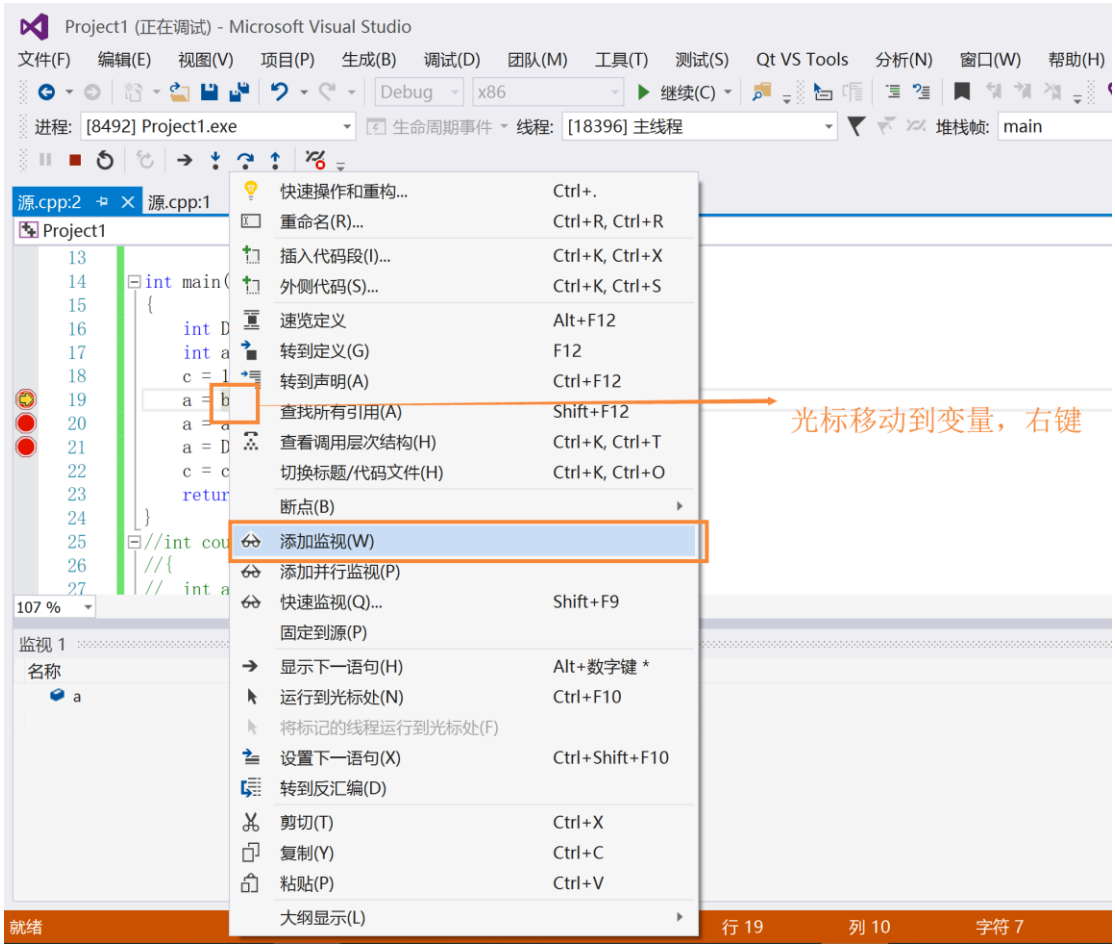
这些变量由 VS 调试器在调试的时候自动检测。VS 检测与当前语句相关的对象或变量，基于此列出 Autos 变量。

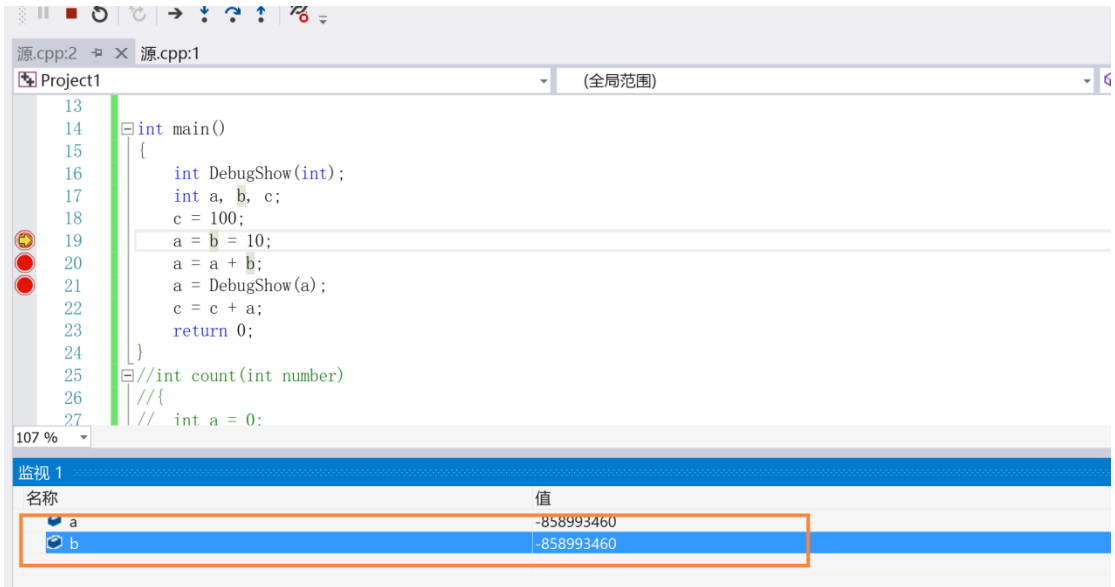




4.3 监视（Watch）

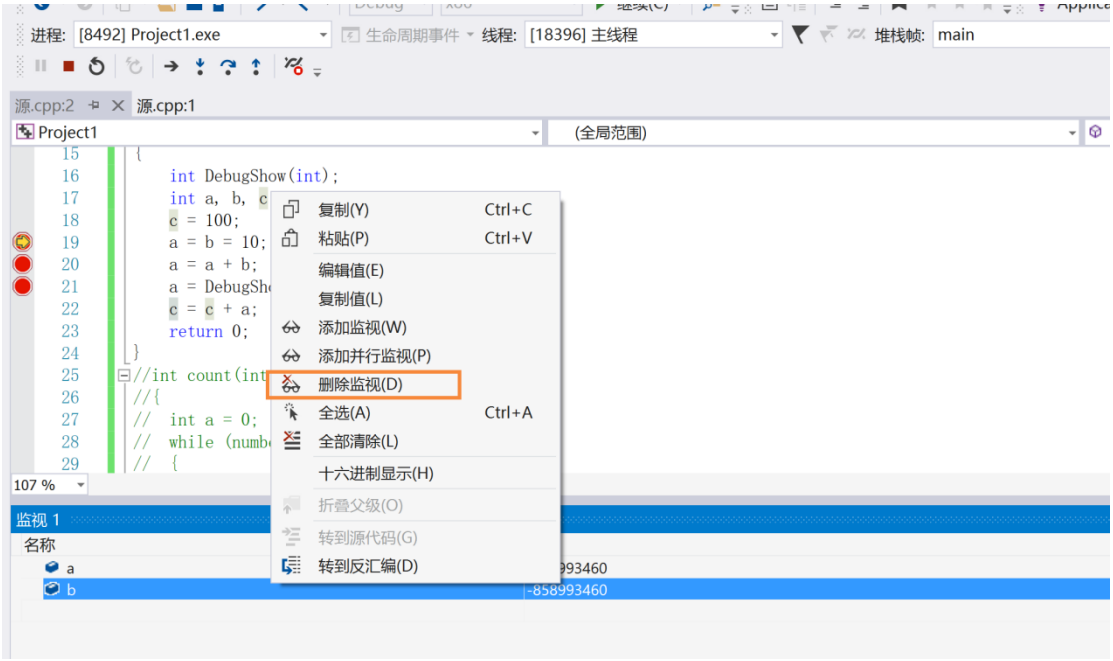
Watch 窗口用于添加变量。你可以添加任意多个变量。添加方法是，**右击变量并选择“Add to Watch”。**





也可以使用拖放（Drag and Drop）将变量添加到监视窗口中。

从监视窗口中删除变量的方法是，右击变量并选择“Delete Watch”。**通过调试窗口，也可以在运行时编辑这些变量值。**



附上示例程序代码：

```
#include <iostream>
using namespace std;

int DebugShow(int a)
{
	int b = 0;
	a = a + 10;
	a = a + 10;
	b = a;
	return b;
}

int main()
{
	int DebugShow(int);
	int a, b, c;
	c = 100;
	a = b = 10;
	a = a + b;
	a = DebugShow(a);
	c = c + a;
	return 0;
}
```