

Visual Studio 程序调试

准备调试

样例程序：

```
1  #include <string>
2  #include <vector>
3  #include <iostream>
4
5  void SendMessage(const std::wstring& name, int msg)
6  {
7      std::wcout << L"Hello, " << name << L"! Count to " << msg << std::endl;
8  }
9
10 int main()
11 {
12     std::vector<wchar_t> letters = { L'f', L'r', L'e', L'd', L' ', L's',
    L'm', L'i', L't', L'h' };
13     std::wstring name = L"";
14     std::vector<int> a(10);
15     std::wstring key = L"";
16
17     for (int i = 0; i < letters.size(); i++)
18     {
19         name += letters[i];
20         a[i] = i + 1;
21         SendMessage(name, a[i]);
22     }
23     std::wcin >> key;
24     return 0;
25 }
```

预期结果

```
1  Hello, f! Count to 1
2  Hello, fr! Count to 2
3  Hello, fre! Count to 3
4  Hello, fred! Count to 4
5  Hello, fred ! Count to 5
6  Hello, fred s! Count to 6
7  Hello, fred sm! Count to 7
8  Hello, fred smi! Count to 8
9  Hello, fred smit! Count to 9
10 Hello, fred smith! Count to 10
```

调试技巧

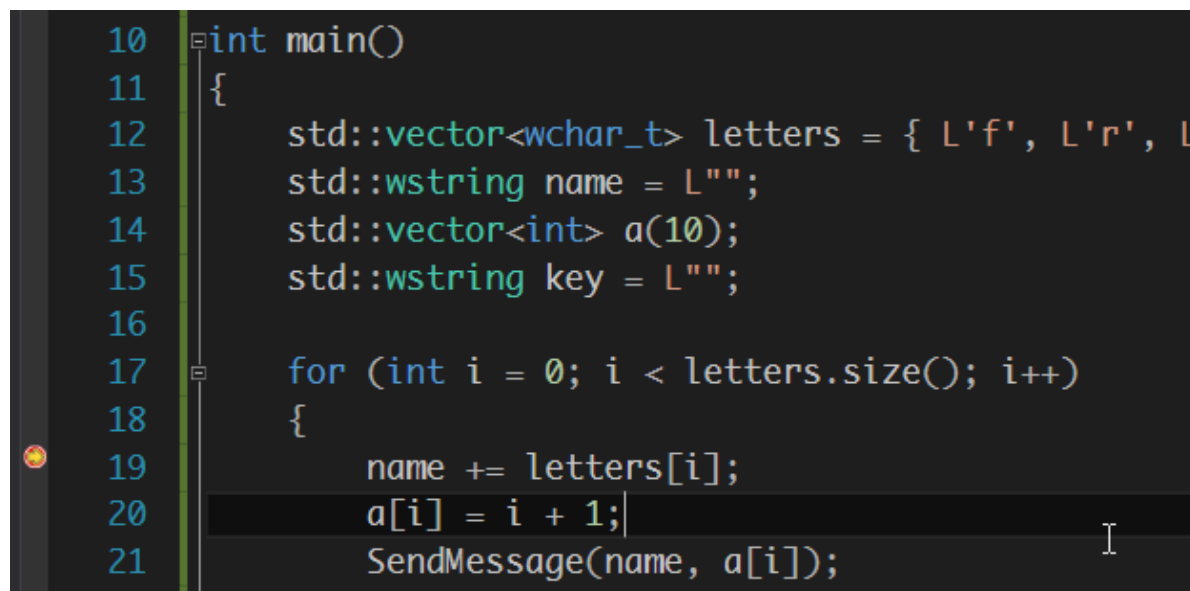
0. 开始调试

在项目中，按下 F5 (调试->开始调试) 开始调试。

1. 遍历语句

断点Breakpoint处暂停

单击 `name += letters[i]` 左边栏设置断点，按下 F5，程序会运行到断点处暂停。



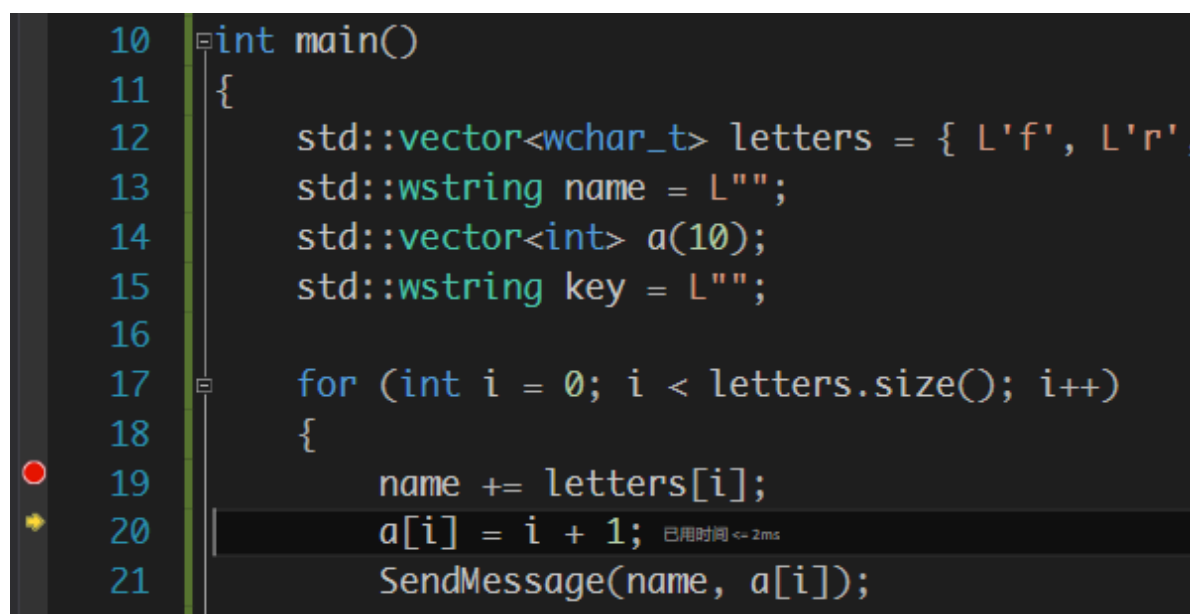
```
10 int main()
11 {
12     std::vector<wchar_t> letters = { L'f', L'r', L'k' };
13     std::wstring name = L"";
14     std::vector<int> a(10);
15     std::wstring key = L"";
16
17     for (int i = 0; i < letters.size(); i++)
18     {
19         name += letters[i];
20         a[i] = i + 1;
21         SendMessage(name, a[i]);
22     }
23 }
```

单步调试

当程序暂停在 `name += letters[i]`，按下 F10 (Step over)，程序会执行断点处的语句，并在下一条语句处暂停。接着按下 F11 (Step into)，也是相同的效果。

虽然都是单步调试，F10 不会进入函数体内部，而 F11 会进入函数体内部。如在 `SendMessage(name, a[i])` 语句处按下 F10，则程序会暂停在 `for` 语句处；若按下 F11，则程序会暂停在函数的第一条语句处（由于使用了 STL，程序会暂停在某个库函数中）。程序在函数体内部暂停时，可以按下 Shift+F11 跳出当前函数。

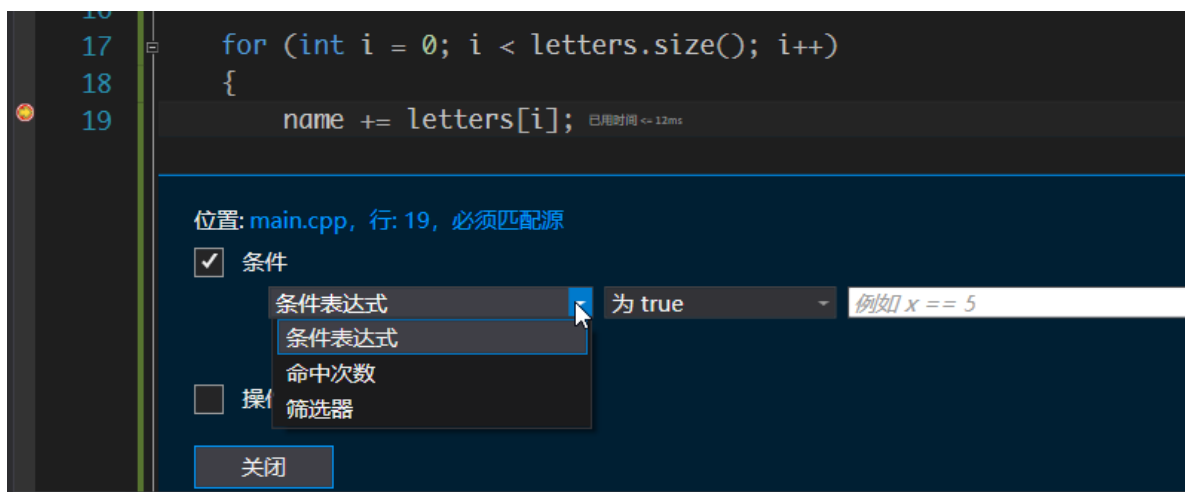
当程序在函数体内部暂停时，可以按下 Shift+F11 跳出当前函数。



```
10 int main()
11 {
12     std::vector<wchar_t> letters = { L'f', L'r', L'k' };
13     std::wstring name = L"";
14     std::vector<int> a(10);
15     std::wstring key = L"";
16
17     for (int i = 0; i < letters.size(); i++)
18     {
19         name += letters[i];
20         a[i] = i + 1;
21         SendMessage(name, a[i]);
22     }
23 }
```

条件断点

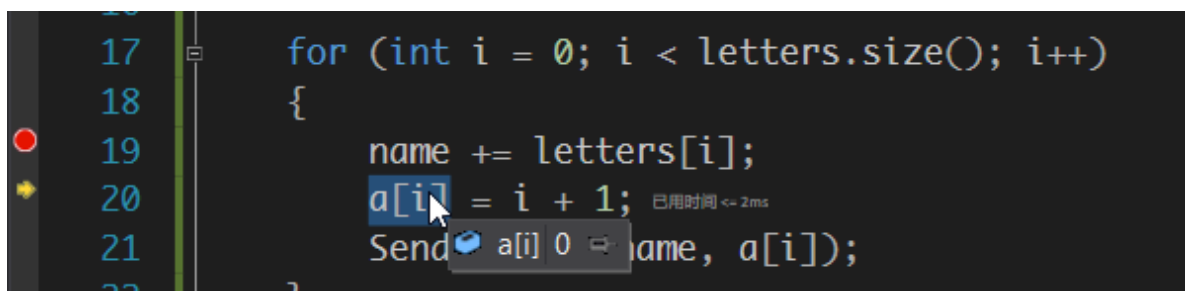
鼠标悬停在断点上，点击出现的齿轮按钮，可以进行断点条件的设置。Visual Studio提供了一些设置的方式，大家可以自行尝试。



2. 查看数据

悬停

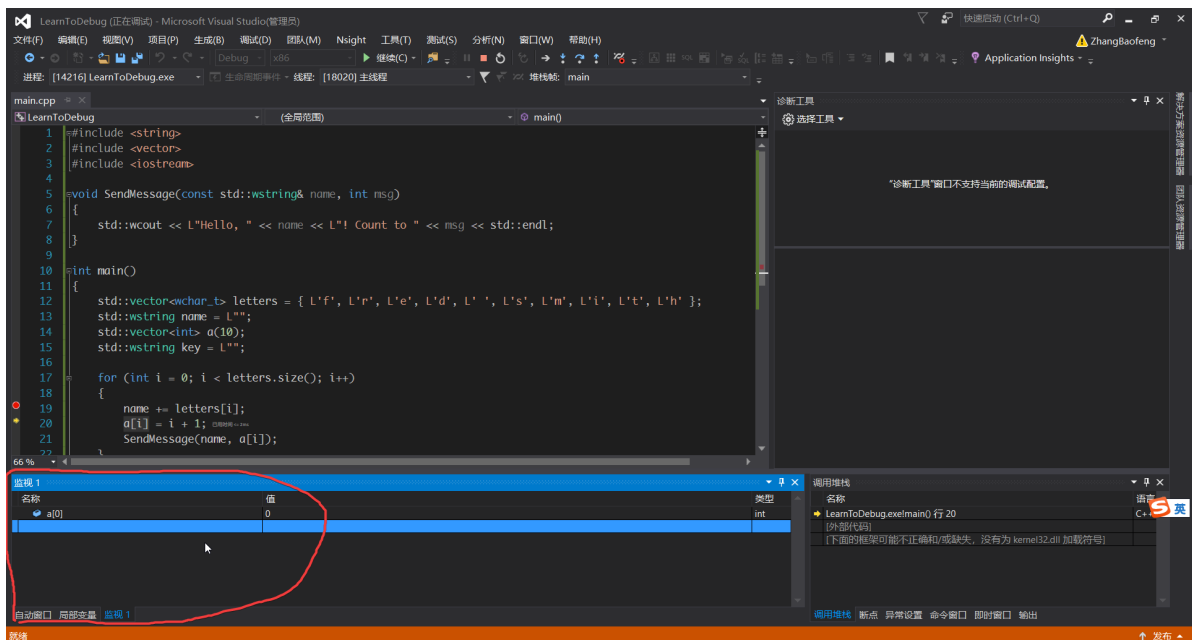
程序暂停时，可以将鼠标悬停在变量上查看它的值，若为结构体、对象等复杂的结构，会有扩展栏，点开可以看到更多的信息。



添加监视

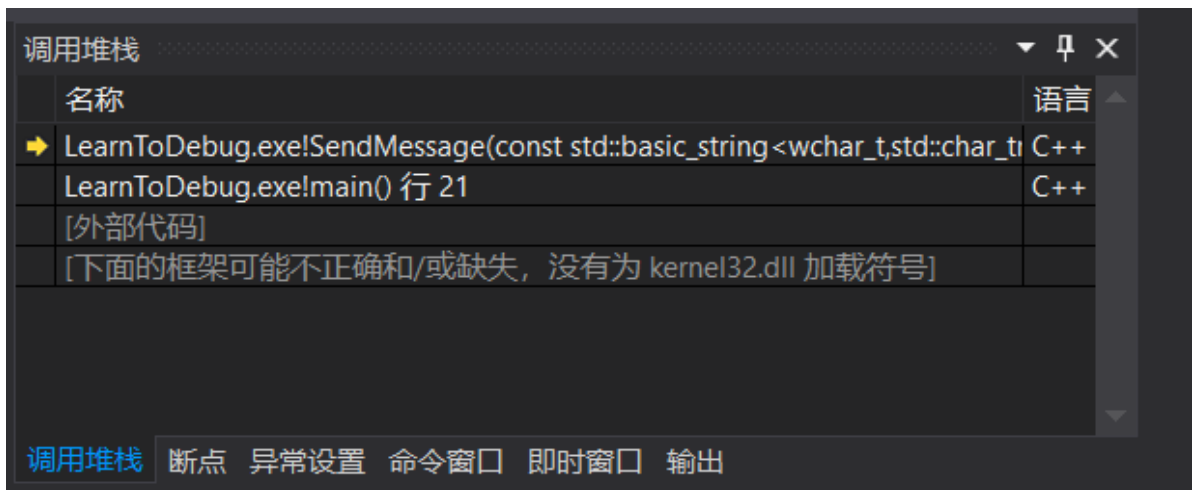
在调试界面的左下角会有 **监视**，**局部变量**，**自动窗口** 共3栏，自动窗口和局部变量两栏的功能可以顾名思义。

在 **监视** 窗口中，可以点击 **名称** 下方的空白区域添加新的监视（输入变量名即可），也可以选中代码中的变量名->右击->添加监视，这样我们就可以在程序运行中，随时查看我们感兴趣的变量值。



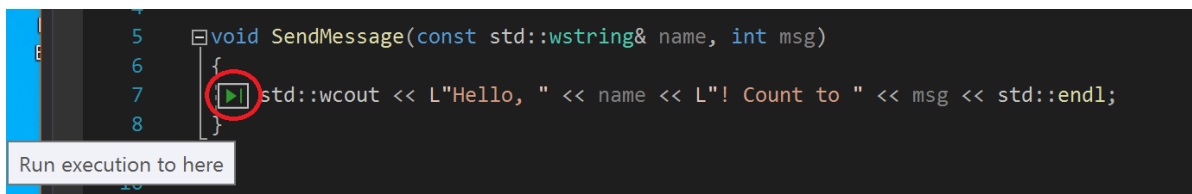
查看调用堆栈

下图为程序暂停在SendMessage函数内部时的调用堆栈，可以看到，程序首先调用 `main()`，接着调用了 `SendMessage()`。通过调用堆栈，我们在遇到bug时，逐层回溯，确定出现问题的位置。



运行到语句处（限Visual Studio 2019）

在某一语句上悬停，直到左侧出现绿色的运行到此处按钮，点击即可运行到语句处。



快捷键

开始调试： `F5`

结束调试： `Shift + F5`

单步调试，逐语句（进入函数内部）： `F10`

单步调试，逐过程（不进入函数内部）： `F11`

退出当前过程： `Shift+F11`

重新开始调试： `Ctrl+Shift+F5`

使用宏

使用 `#ifdef` 和 `#endif` 方便地输出中间结果

注解1：处于`#ifdef XXX`和`#endif`之间的语句，仅在XXX被`#define`的情形下才会被编译。如下例，输出语句 `std::cout<<msg;` 位于 `#ifdef MYDEBUG` 和 `#endif` 之间，仅在 `#define MYDEBUG` 的情形下才会被编译。如此，我们通过修改 `#define` 语句，就可以控制程序当中的所有中间debug输出语句。

注解2：程序中使用了C++模板，详情可以参考《C++ Primer》的第10章函数模板。

```
1 // 示例1：便于修改的中间结果输出
2 #include <iostream>
3
4 // 使用宏来控制输出语句的编译与否
5 #define MYDEBUG
6
7 template<class T>
8 void debugMsg(T msg) {
9     #ifdef MYDEBUG
10         std::cout << msg;
11     #endif // MYDEBUG
12 }
13
14 int main() {
15     debugMsg("This is debug msg1\n");
16     char a[10] = "msg2";
17     debugMsg(a);
18
19     system("pause");
20     return 0;
21 }
```

Reference

- [Tutorial: Learn to debug C++ code using Visual Studio](#)
- 《C++ Primer》