**效能分析**

  VSTS 效能分析工具

效能分析，Performance！这是每一个程序员都梦想的事儿，让自己的程序跑得又快又好，最好是比别的同学快一个数量级，别人的程序是O(N^2)，而我的程序是O(n\*logN)，或者是O(N)，这是多爽的一项成就呀！VSTS提供了方便的效能分析工具，让我们能很快地找到程序的效能瓶颈，从而能有的放矢，改进程序。下面我们看一个具体的例子。

和同学们的作业类似，有这样一道题：

写一个程序，分析一个文本文件中各个词出现的频率，并且把频率最高的10个词打印出来。

 下面用C#写好了程序，命名为WordFreq.exe，然后运行了一下，验证了正确性，程序的基本框架如代码清单2-3所示（全部程序可以在移山社区网站下载）：

代码清单2-3  WordFreq程序，程序框架

DoIt()

{

    ProcessFile()  //store all words in a big buffer

    ProcessBuffer()  //calculate and store the frequency of each word

    OutputResult()   //output top 10

}

 ProcessBuffer()

{    GetOneWord()   //get one word from buffer

    FreqOneWord()

}

 FreqOneWord(word)

{Find the word in the array list,

If (found)

    Update the frequency

If (not found)

    Add the word in the array list with frequency = 1

}

OutputResult()

{

ArrayList.Sort()   //sort the array

Output Top 10 entry;

}

文本文件大约是30KB~300KB大小。在运行效能分析之前，大家预计占用时间最多的是什么函数，或者哪些语句。处理文件？因为I/O很花时间。排序？处理每个词？有人建议应该把排序和处理每一个词同时进行，这样就能加快速度。

我们看看到底会是什么情况。第一步，要确保编译的程序是Release版本。然后在VS界面中选中Tools | Performance Tools | Performance Wizard（如图2-1所示）。

[](http://images.cnblogs.com/cnblogs_com/xinz/201111/201111201033366124.jpg)

图2-1  效能分析，选择分析方法

我们看到可以选择两种分析方法：

（1）        抽样（Sampling）

（2）        代码注入（Instrumentation）

通俗地解释，**抽样**就是当程序运行时，Visual Studio时不时看一看这个程序运行在哪一个函数内，并记录下来，程序结束后，Visual Studio就会得出一个关于程序运行时间分布的大致的印象。这种方法的优点是不需要改动程序，运行较快，可以很快地找到瓶颈。但是不能得出精确的数据，代码中的调用关系（CallTree）也不能准确表示。

另一方面，**代码注入**就是将检测的代码加入到每一个函数中，这样程序的一举一动都被记录在案，程序的各个效能数据都可以被精准地测量。这一方法的缺点是程序的运行时间会大大加长，还会产生很大的数据文件，数据分析的时间也相应增加。同时，注入的代码也影响了程序真实的运行情况（这有点像量子物理学中的“测试的光线干扰了测试物体本身”的现象）。

我们一般的做法是，先用抽样的方法找到效能瓶颈所在，然后对特定的模块用代码注入的方法进行详细分析。

对程序进行效能分析，我们先要弄清下面这几个名词，如表2-1所示：

表2-1  效能分析的名词解释

|  |  |
| --- | --- |
| 名  词 | 含  义 |
| 调用者Caller | 函数Foo()中调用了Bar()，Foo()就是调用者 |
| 被调用函数Callee | 见上，Bar()就是被调用函数 |
| 调用关系树Call Tree | 从程序的Main()函数开始，调用者和被调用函数就形成了一个树形关系——调用树 |
| 消逝时间Elapsed Time | 从用户的角度来看程序运行所花的时间。当用户看到一个程序没有反应，用户并不知道程序此时是在运行自己的代码，还是被调度出去了，或者操作系统此时正在忙别的事情 |
| 应用程序时间Application Time | 应用程序占用CPU的时间，不包括CPU在核心态时花费的时间 |

续表

|  |  |
| --- | --- |
| 名  词 | 含  义 |
| 本函数时间Exclusive Time | 所有在本函数花费的时间，不包括被调用者使用的时间 |
| 所有时间Inclusive Time | 包含本函数和所有调用者使用的时间 |

理解了上面的各种概念后，我们就不难理解“消逝的本函数时间（Elapsed Exclusive Time）”等其他组合名词所代表的概念了。

我们先进行抽样分析，在效能浏览器（Performance Explorer）中开始效能分析即可。

图2-2是WordFreq程序处理一个30KB的文本文件时的情况：

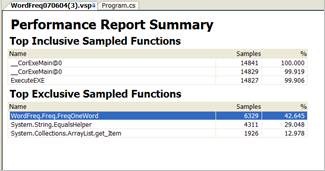
[](http://images.cnblogs.com/cnblogs_com/xinz/201111/201111201033449443.jpg)

图2-2  用抽样的方法分析效能

大家可以看到最花时间的三个函数是：

WordFreq.Freq.FreqOneWord(string)

System.String.EqualsHelper(string,string)

System.Collections.ArrayList.get\_Item(int32)

三个函数加起来占用了整个程序84%的时间。看来我们得分析为什么这三个函数会被调用得这么频繁，开销这么大了。

我们现在可以进行代码注入的分析，同样运行程序后，我们看看图2-3的调用树（Call Tree）报告。

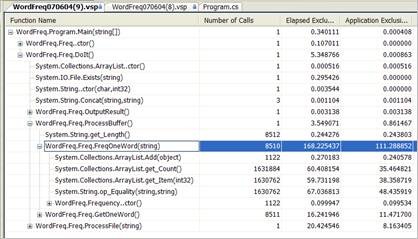
[](http://images.cnblogs.com/cnblogs_com/xinz/201111/20111120103400517.jpg)

图2-3  代码注入方法产生的效能报告

结合实际的代码（见代码清单2-4），可以看到在WordFreq. FreqOneWord函数中，究竟发生了什么：

代码清单2-4  FreqOneWord()

        private void FreqOneWord(string w)

        {

            // see if we have a match, if not, add it to the end,

            // then assign it initial frequency 1;

            // if yes, inc the frequency by 1

            for (int i = 0; i < m\_wordList.Count; i++)

            {

                Frequency fi = (Frequency)m\_wordList[i];

                if (fi.str == w)

                {

                    fi.n++;

                    return;

                }

            }

            //now we have to append it to the end.

            Frequency f = new Frequency();

            f.str = w;

            f.n = 1;

            m\_wordList.Add(f);

        }

我们可以清楚地看到：

WordFreq.Freq.FreqOneWord(string)被调用了8150次，说明有8150个词被处理了。但是，System.Collections.ArrayList.Add(object)被调用了1112次，说明有1112个不同的词被加入到ArrayList中。下面三个函数被调用的次数相似，它们都花了很多的时间。

System.Collections.ArrayList.get\_Count()

System.Collections.ArrayList.get\_Item(int32)

System.String.op\_Equality(string,string)

写这一个语句的时候

      for (int i = 0; i < m\_wordList.Count; i++)

没想到m\_wordList.Count，也就是ArrayList.GetCount()，会花这么多时间，累计被调用了1631884次！

马上把代码改成：

            int count = m\_wordList.Count;

for (int i = 0; i < count; i++)

这样会如何？大家等了一会儿，代码分析的结果出来了（如图2-4所示）：

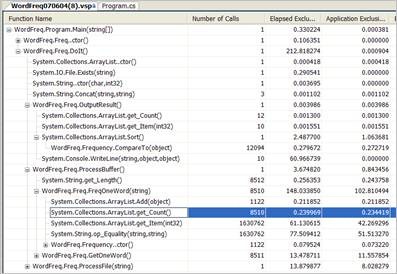
[](http://images.cnblogs.com/cnblogs_com/xinz/201111/201111201034095689.jpg)

图2-4  改进过的程序效能分析图

可以看到System.Collections.ArrayList.get\_Count()的调用次数和时间都大幅度地下降了。

我们可以继续进行“效能测试，分析，改进，再效能测试”的流程，逐渐提高程序的效能和我们的编程水平。

大家也要注意避免没有做分析就过早地进行“效能提高”，刚才有人提到我们可能要提高排序的性能，但是从图2-4来看，System.Collections. ArrayList.Sort()只占了FreqOneWord()不到1/50的时间，如果我们不经分析就盲目优化，也许会事倍功半。

5  讨论

甲：噫吁唏，危乎高哉！我以前一直害怕做效能分析，看来是否会用效能分析工具来提高程序质量是一个优秀程序员的标志之一。我在今天之前都是盲人骑瞎马。

乙：改成盲人摸象更恰当。我听说，如果一个程序员从来没有用过效能分析工具，那他就不是一个程序员，只是一个编程爱好者罢了。

我的WC 程序事实上是最快的。但是在今天的程序评比之前，我想更进一步，就再优化了一下，估计能把速度提高2%。没想到出了一个小错误，导致报告的结果（行数、词数、字数）仅仅差了1。由于程序不正确，因此不能参加速度评比。我才是最需要大喊“噫吁唏”的人。

丙：这有两个教训：

（1）先保证正确性，再提高效能。一个“仅仅差了1”的错误可能会导致缓冲区溢出（Buffer Overflow）的严重漏洞。

（2）如果效能的提高效果在5%以下，用户不会注意到程序效能的区别。所以要考虑那些微小的提高是否值得。

另外，WordFreq算法的时间复杂度是多少，能否再优化？请写实际程序加以验证。

丁：怎么才能快捷地得到有适量重复文字的文件，来帮助我们进行测试和效能分析？

九条：我是这样做的，在电脑的根目录下，运行“dir /s > c:\temp\test.txt”命令。

阿超：好主意，大家还有没有别的办法？