**[.NET中Trace类和Debug类的应用举例](http://blog.csdn.net/yunhaic/article/details/4863322)**

.NET中的Trace类和Debug类是System.Diagnostics命名空间下的两个用于调试的类，在软件开发中使用得当，可以帮助我们调试开发程序，提高效率，很有用。其中，Debug类仅在调试版本中有效，而Trace类在调试版本和正式版本中均有效。

**（1）下面先介绍下Debug类的使用：**

Debug类的声明原型：public sealed class Debug

从声明中，我们可以看出，Debug类是无法继承的。它的成员情况，请参见：

<http://msdn2.microsoft.com/zh-cn/library/system.diagnostics.debug_members(VS.80).aspx>

先看一个例子：

TestDebug.cs：

using System;

using System.Collections.Generic;

using System.Text;

using System.Diagnostics; //引入Debug类所在命名空间

namespace traceanddebug

{

    class TestDebug

    {

        public static void TestDebugMethod()

        {

            Debug.Listeners.Add(new TextWriterTraceListener(Console.Out));

//将Debug类输出定向到控制台输出

            Debug.AutoFlush = true;

//设置Debug为自动输出，即每次写入后都调用Listeners上调用Flush

            Debug.Indent();

//设置缩进

            Debug.WriteLine("Debug WriteLine()");

//用Debug输出"Debug WriteLine()"

            Console.WriteLine("Console.WriteLine()");

//用Console输出"Console.WriteLine()"

            Debug.WriteLine("Debug WriteLine2()");

//用Debug输出"Debug WriteLine2()"

            Debug.Unindent();

//取消缩进

            Console.Read();

        }

    }

}

Program.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace traceanddebug

{

    class Program

    {

        static void Main(string[] args)

   {

            TestDebug.TestDebugMethod();

        }

    }

}

结果显示如下：

    Debug WriteLine()

Console.WriteLine()

    Debug WriteLine2()

从例子中，我们可以看出，Debug可以向控制台显示信息。需要注意的是，我们在使用Debug时候，需要引入System.Diagnostics命名空间。同样，我们也可以把信息写入文本中，例如，把TestDebug.cs中

Debug.Listeners.Add(new TextWriterTraceListener(Console.Out));

一句修改为：

Debug.Listeners.Add(new TextWriterTraceListener("c://testdebug2.log","testdebug"));

就可以向c盘的根目录下的testdebug2.log文件中写入信息了。

       给你留个小问题，你能让debug信息写入windows的事件查看器里面么？

       （看看下面的Trace类的例子，也许你就有思路了）

       值得注意的是，Debug类的代码在release版本中是无效的，也就是说，所有的Debug类输出语句在发布的时候，是不会输出的，你不用特意删除Debug类的代码，就能给用户一个没有调试语句的程序了。这个用起来是不是很方便呢？

       Debug类除了输出信息之外，还有一个静态成员函数很有用，就是Debug.Assert方法。它有三种重载形式：

Debug.Assert(Boolean)

Debug.Assert(Boolean, String)

Debug.Assert(Boolean, String)

它的作用为：检查条件，如果该条件为 false，则显示消息。此函数会显示相关堆栈信息。

例如，我们在上面的TestDebug.cs中TestDebugMethod函数中加入一下语句：

Debug.Assert(false,"test assert");

那么运行时，就会停在Debug.Assert调用的地方，弹出一个对话框，显示信息及堆栈信息。我们在实际应用的时候，需要讲例子中的false换成条件判断，如果不满足条件，就会停止。

**（2）下面介绍Trace类的使用：**

Trace类的声明原型为：public sealed class Trace，其成员情况，请参加网址：

<http://msdn2.microsoft.com/zh-cn/library/system.diagnostics.trace_members(VS.80).aspx>

同样，我们先从小例子来认识Trace类：

TestTrace.cs

using System;

using System.Diagnostics;

namespace traceanddebug

{

       public class TestTrace

       {

              public static void TheTrace()

              {

                     Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));

                     Trace.WriteLine("Trace !!!");

                     Console.ReadKey();

              }

       }

}

Program.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace traceanddebug

{

    class Program

    {

        static void Main(string[] args)

   {

            TestTrace.TheTrace();

        }

    }

}

编译运行程序，输出结果如下：

Trace !!!

从这个例子中，我们可以看出，Trace类可以像Debug类一样输出信息到控制台。同样，Trace类可以设置输出定向到文本文件或者日志中。可以添加一下代码测试：

Trace.Listeners.Add(new TextWriterTraceListener("testtrace.log"));

Trace.Listeners.Add(new EventLogTraceListener("testtrace"));

Trace.AutoFlush = true;

这样就同时把Trace!!这个字符串写入testtrace.log文件和事件查看器中应用程序里面了。

除了在程序中设定信息输出位置外，还可以在配置文件中设定跟踪调试信息的输出位置。默认是在控制台输出。我们可以通过添加配置参数，修改输出位置。例如：

假设我们的运行程序为traceanddebug.exe，我们在相同目录下建立traceanddebug.exe.config配置文件，输入如下内容：

<configuration>

  <system.diagnostics>

    <trace autoflush="true" indentsize="1">

    <listeners>

       <remove name="Default"></remove>

       <add type="System.Diagnostics.EventLogTraceListener" name="myEventLog" initializeData="application">

        </add>

    </listeners>

        </trace>

  </system.diagnostics>

</configuration>

这样我们可以修改程序TheTrace()为：

                     //Trace.Listeners.Add(new TextWriterTraceListener(Console.Out));

                     Trace.WriteLine("Trace !!!");

                     Console.ReadKey();

这样就可以把结果直接写入事件查看器中。

由于Trace类可以在发布版本中起作用，所以，在做Trace输出信息的时候，可能有些信息是调试的时候需要，发布的时候又不需要了，那怎么才能保证信息不在发布的时候输出呢？.NET为我们提供了五个控制Trace的跟踪级别，使用它们不必重新编译，他们跟别是：

|  |  |  |
| --- | --- | --- |
| Trace Level | Value | 显示（或写入指定输出目标）的消息类型 |
| Off | 0 | 无，即关闭显示信息 |
| Errors | 1 | 仅限错误信息 |
| Warnings | 2 | 警告消息和错误信息 |
| Info | 3 | 提示信息、警告消息和错误信息 |
| Verbose | 4 | 详细消息、提示性消息、警告消息和错误信息 |

是否显示消息，由TraceSwich开关类决定。即如果设定开关为4，那么所有的Trace消息都将显示；如果设定为3，那么设定为Verbose的Trace消息就不显示了，其余的都显示；如果设定为0，所有的消息都不显示了。这里设定的开关，可以在配置文件中设定，也可以在程序中设定，下面就举例说明。

TestTrace.cs

using System;

using System.Diagnostics;

namespace traceanddebug

{

       public class TestTrace

       {

              public static TraceSwitch ts = new TraceSwitch("mySwitch", "in the Config file");

              public static void TheTrace()

              {

                     Trace.WriteLineIf(ts.TraceError,"Error!!!");

                     Trace.WriteLineIf(ts.TraceWarning,"Warning!!!");

                     Trace.WriteLineIf(ts.TraceInfo,"Info!!!");

                     Trace.WriteLineIf(ts.TraceVerbose,"Verbose!!!");

                     Console.ReadKey();

              }

       }

}

Program.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace traceanddebug

{

    class Program

    {

        static void Main(string[] args)

   {

            TestTrace.TheTrace();

        }

    }

}

       编译出的可执行文件名为：traceanddebug.exe

       我们在相同目录下，添加配置文件traceanddebug.exe.config，并输入内容为：

traceanddebug.exe.config

<configuration>

  <system.diagnostics>

    <switches>

      <add name="mySwitch" value="0" />

    </switches>

  </system.diagnostics>

</configuration>

编译运行，没有显示信息。我们将traceanddebug.exe.config用记事本打开，修改mySwitch的值为1，保存，编译运行，结果显示”Error!!!”；同理，我们依次，将值修改为2，3，4查看结果，分别为Warning!!!，Info!!!，Verbose!!!。

有此，我们可以看出，通过控制开关级别，就可以显示不同的级别的信息，而不用重新编译，那么及时在软件已经发布的时候，我们也可以通过修改配置文件，而查看相关信息。

       通过以上的内容，我们可以对Trace和Debug类有个大概的了解，那么如果你需要调试程序，不妨用一下这两个类。

**第二种说法：**

**System.Diagnostics命名空间里的Debug类和Trace类的用途**

|  |
| --- |
| 在 .net 类库中有一个 system.diagnostics 命名空间，该命名空间提供了一些与系统进程、事件日志、和性能计数器进行交互的类库。当中包括了两个对开发人员而言十分有用的类——debug类和 trace类。本文介绍了这两个类的一些基本用途，旨在提高广大开发人员的开发效率。  使用debug类来帮助调试   调试程序对每个程序员来说是家常便饭。可是我们会经常遇到一些情况让我们头疼，例如：   当我们在开发一个界面控件的时候，简单的设断点会增加paint事件的响应次数，而造成的环境参数改变。  断点设多了，程序常常停在正常运行的地方；这样一来，调试一个错误要花费大量时间去寻找错误。  这时，我们就需要利用system.diagnostics.debug类来帮助我们调试。我们可以通过调用 debug.writeline(string message)函数，将我们所关心的信息打印在visual studio ide的output窗口中。也可以利用debug.assert(bool condition)来让程序停在错误的地方，并且显示call stack。   debug类中所有函数的调用都不会在release版本里有效。也就是说，我们通过这种方法所加的代码可以仅用于调试；在发布的时候无需删任何代码，就可以给用户一个没有调试指令的程序了。 |