为应用程序请求 Run As "Administrator"

解决方法：

通过直接修改应用程序的 manifest 配置： Trace\_Demo.vshost.exe.manifest

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">

<assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">

<security>

<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">

<requestedExecutionLevel level="requireAdministrator" uiAccess="false"/>

</requestedPrivileges>

</security>

</trustInfo>

</assembly>

默认值是 ：　　　asInvoker

<trustInfo>

<security>

<applicationRequestMinimum>

<PermissionSet Unrestricted="true" ID="Custom" SameSite="site" />

<defaultAssemblyRequest permissionSetReference="Custom" />

</applicationRequestMinimum>

<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">

<!--

UAC Manifest Options

If you want to change the Windows User Account Control level replace the

requestedExecutionLevel node with one of the following.

<requestedExecutionLevel level="asInvoker" uiAccess="false" />

<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />

<requestedExecutionLevel level="highestAvailable" uiAccess="false" />

If you want to utilize File and Registry Virtualization for backward

compatibility then delete the requestedExecutionLevel node.

-->

<requestedExecutionLevel level="asInvoker" uiAccess="false" />

</requestedPrivileges>

</security>

</trustInfo>

**如果想在应用程序定义好：　则添加 app.manifest**

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">

<security>

<requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">

<!-- UAC Manifest Options

If you want to change the Windows User Account Control level replace the

requestedExecutionLevel node with one of the following.

<requestedExecutionLevel level="asInvoker" uiAccess="false" />

<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />

<requestedExecutionLevel level="highestAvailable" uiAccess="false" />

Specifying requestedExecutionLevel node will disable file and registry virtualization.

If you want to utilize File and Registry Virtualization for backward

compatibility then delete the requestedExecutionLevel node.

-->

<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />

</requestedPrivileges>

</security>

</trustInfo>

[**C#笔记30：Trace、Debug和TraceSource的使用以及日志设计**](http://www.cnblogs.com/luminji/archive/2010/10/26/1861316.html)

2010-10-26 11:28 by Luminji, 3119 阅读, 2 评论, [收藏](http://www.cnblogs.com/luminji/archive/2010/10/26/1861316.html), [编辑](http://i.cnblogs.com/EditPosts.aspx?postid=1861316)

C#笔记30：Trace、Debug和TraceSource的使用以及日志设计

本章概要：

1：Trace 和 Debug区别

2：什么是Listeners

3：跟踪开关

     3.1：使用[BooleanSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.booleanswitch%28VS.90%29.aspx)开关

     3.2：使用[TraceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.traceswitch%28VS.90%29.aspx)开关

4：使用TraceSource代替Trace和

5：设计一个日志系统

6：关于EventLog

      .NET Framework 命名空间 [System.Diagnostics](http://msdn.microsoft.com/zh-cn/library/system.diagnostics%28VS.90%29.aspx) 包含用于跟踪执行流程的 [Trace](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.trace%28VS.90%29.aspx)、[Debug](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.debug%28VS.90%29.aspx) 和 [TraceSource](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.tracesource%28VS.90%29.aspx) 类，以及用于分析代码的 [Process](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.process%28VS.90%29.aspx)、[EventLog](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.eventlog%28VS.90%29.aspx) 和 [PerformanceCounter](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.performancecounter%28VS.90%29.aspx) 类。

      跟踪是一种在应用程序运行时监视其执行情况的方式。当开发 .NET Framework 应用程序时，可以在其中添加跟踪和调试检测功能，并且在开发应用程序时和部署应用程序后，都可以使用该检测功能。利用 **Trace** 和 **Debug** 类，可以将有关错误和应用程序执行的信息记录到日志、文本文件或其他设备中，以便在随后进行分析。

      下面列出了六个写入跟踪信息的 Debug Members 和 Trace 方法。

**Assert：**指定的文本；如果未指定任何文本，则为“调用堆栈”。只有当 **Assert** 语句中以参数形式指定的条件为 **false** 时，才会写入输出。

**Fail：**指定的文本；如果未指定任何文本，则为“调用堆栈”。

**Write：**指定的文本。

**WriteIf：**如果满足 **WriteIf** 语句中以参数形式指定的条件，则为指定的文本。

**WriteLine：**指定的文本和一个回车。

**WriteLineIf：**如果满足 **WriteLineIf** 语句中以参数形式指定的条件，则为指定的文本和一个回车。

**1：Trace 和 Debug区别**

**Trace** 和 **Debug** 类基本相同，不同的只是 **Trace** 类的过程和函数默认为编译成发布版本。

**2：什么是Listeners**

      Listenters属性，它是TraceListenerCollection类型（TraceSource类和TraceListener类），给类 属性控制跟踪信息输出的方向，可以是控制台（add(TextWriterTraceListener(new Console.Out))），文件(add(TextWriterTraceListener(new IO.File.CreateText(“output.txt”))等。Listenters集合中的成员包括 TextWriterTraceListener，DefaultTraceListener，EventLogTraceListener，WebPageTraceListener 等。而TextWriterTraceListener的子类又有ConsoleTraceListener， DelimitedListTraceListener，XmlWriterTraceListener，EventSchemaTraceListener。

     您可以通过实现您自己的侦听器来生成自定义的结果。 所有自定义侦听器都应支持文章开头表中的六个方法。

     以下的例子说明输出的消息将会在控制台、TXT文件以及系统日志中均被记录。

TextWriterTraceListener tr1 = new TextWriterTraceListener(System.Console.Out);

Debug.Listeners.Add(tr1);

TextWriterTraceListener tr2 = new TextWriterTraceListener(System.IO.File.CreateText("output.txt"));

Debug.Listeners.Add(tr2);

EventLogTraceListener tr3 = new EventLogTraceListener();

Debug.Listeners.Add(tr3);

**3：跟踪开关**

      除了指定Listener外，要控制消息是否被输出，还需要指定跟踪开关。跟踪开关用于启用、禁用和筛选跟踪输出。

      Framework 中提供了三种类型的跟踪开关：[BooleanSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.booleanswitch%28VS.90%29.aspx) 类、[TraceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.traceswitch%28VS.90%29.aspx) 类和 [SourceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.sourceswitch%28VS.90%29.aspx) 类。[BooleanSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.booleanswitch%28VS.90%29.aspx)是最简单的跟踪开关，可以指定是否输出消息。[TraceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.traceswitch%28VS.90%29.aspx) 和 [SourceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.sourceswitch%28VS.90%29.aspx) 类用于为特定的跟踪级别启用跟踪开关，以便显示为该级别及其下的所有级别指定的 [Trace](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.trace%28VS.90%29.aspx) 或 [TraceSource](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.tracesource%28VS.90%29.aspx) 消息。

**3.1：使用[BooleanSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.booleanswitch%28VS.90%29.aspx)开关**

      以下是使用[BooleanSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.booleanswitch%28VS.90%29.aspx)的例子：

TextWriterTraceListener tr1 = new TextWriterTraceListener(System.Console.Out);

Debug.Listeners.Add(tr1);

TextWriterTraceListener tr2 = new TextWriterTraceListener(System.IO.File.CreateText("output.txt"));

Debug.Listeners.Add(tr2);

EventLogTraceListener tr3 = new EventLogTraceListener();

Debug.Listeners.Add(tr3);

bool someBizCondition = true;

BooleanSwitch bs = new BooleanSwitch("DataMessageSwitch", "DataMessageSwitch des");

bs.Enabled = true;

Debug.WriteLineIf(someBizCondition, "log....");

Debug.Flush();

      bs.Enabled设置为true或者false，并不会使程序自动决定是否输出信息。

      如果不使用代码方式，而是使用配置文件的方式，是在 <configuration> 标记之后，但在 </configuration> 标记之前添加相应的 XML 来配置您的开关。如下：

<system.diagnostics>

<switches>

<add name="DataMessagesSwitch" value="1" />

</switches>

</system.diagnostics>

**3.2：使用[TraceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.traceswitch%28VS.90%29.aspx)开关**

      TraceSwitch类可以通过使用跟踪开关来筛选消息，通过Level属性来获取或设置开关级别。0、1、2、3 和 4 分别对应于 **Off**、**Error**、**Warning**、**Info** 和 **Verbose**。任何大于 4 的数字都会被当作 **Verbose**，任何小于零的数字都会被当作 **Off**。

      以下的例子，用代码的方式来演示使用[TraceSwitch](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.traceswitch%28VS.90%29.aspx)来设置跟踪开关：

TextWriterTraceListener tr1 = new TextWriterTraceListener(System.Console.Out);

Debug.Listeners.Add(tr1);

TextWriterTraceListener tr2 = new TextWriterTraceListener(System.IO.File.CreateText("output.txt"));

Debug.Listeners.Add(tr2);

EventLogTraceListener tr3 = new EventLogTraceListener();

Debug.Listeners.Add(tr3);

bool someBizCondition = true;

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

ts.Level = TraceLevel.Verbose;

Debug.WriteLineIf(ts.TraceError && someBizCondition, "Error!!!");

Debug.WriteLineIf(ts.TraceWarning && someBizCondition, "Warning!!!");

Debug.WriteLineIf(ts.TraceInfo && someBizCondition, "Info!!!");

Debug.WriteLineIf(ts.TraceVerbose && someBizCondition, "Verbose!!!");

Debug.Flush();

       使用XML来配置，如下：

<system.diagnostics>

<switches>

<add name="mySwitch" value="1" />

</switches>

</system.diagnostics>

**4：使用TraceSource代替Trace和Debug**

       从FRAMEWORK2.0开始，就不建议使用Trace和Debug了，而改而用TraceSouce。[TraceSource](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.tracesource%28VS.90%29.aspx) 旨在用作增强的跟踪系统，并且可代替较旧的 [Trace](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.trace%28VS.90%29.aspx) 和 [Debug](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.debug%28VS.90%29.aspx) 跟踪类的静态方法使用。熟悉的 [Trace](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.trace%28VS.90%29.aspx) 和 [Debug](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.debug%28VS.90%29.aspx) 类仍然存在，不过建议的做法是使用 [TraceSource](http://msdn.microsoft.com/zh-cn/library/system.diagnostics.tracesource%28VS.90%29.aspx) 类进行跟踪。

       下面的例子演示使用代码来实现消息的输出：

private static TraceSource mySource = new TraceSource("TraceSourceApp");

static void Main(string[] args)

{

mySource.Switch = new SourceSwitch("sourceSwitch", "Error");

mySource.Listeners.Remove("Default");

TextWriterTraceListener textListener = new TextWriterTraceListener("myListener.log");

textListener.TraceOutputOptions = TraceOptions.DateTime | TraceOptions.Callstack;

textListener.Filter = new EventTypeFilter(SourceLevels.Error);

mySource.Listeners.Add(textListener);

ConsoleTraceListener console = new ConsoleTraceListener(false);

console.Filter = new EventTypeFilter(SourceLevels.Information);

console.Name = "console";

mySource.Listeners.Add(console);

Activity1();

// Set the filter settings for the

// console trace listener.

mySource.Listeners["console"].Filter = new EventTypeFilter(SourceLevels.Critical);

Activity2();

// Allow the trace source to send messages to

// listeners for all event types.

mySource.Switch.Level = SourceLevels.All;

// Change the filter settings for the console trace listener.

mySource.Listeners["console"].Filter = new EventTypeFilter(SourceLevels.Information);

Activity3();

mySource.Close();

return;

}

static void Activity1()

{

mySource.TraceEvent(TraceEventType.Error, 1, "Error message.");

mySource.TraceEvent(TraceEventType.Warning, 2, "Warning message.");

}

static void Activity2()

{

mySource.TraceEvent(TraceEventType.Critical, 3, "Critical message.");

mySource.TraceInformation("Informational message.");

}

static void Activity3()

{

mySource.TraceEvent(TraceEventType.Error, 4, "Error message.");

mySource.TraceInformation("Informational message.");

}

      以上代码，如果使用配置文件的方式实现，如下：

<system.diagnostics>

<sources>

<source name="TraceSourceApp" switchName="sourceSwitch" switchType="System.Diagnostics.SourceSwitch">

<listeners>

<add name="console" type="System.Diagnostics.ConsoleTraceListener">

<filter type="System.Diagnostics.EventTypeFilter" initializeData="Warning"/>

</add>

<add name="myListener" type="System.Diagnostics.TextWriterTraceListener" initializeData="myListener.log"

traceOutputOptions="Callstack">

<filter type="System.Diagnostics.EventTypeFilter" initializeData="Error"></filter>

</add>

<remove name="Default"/>

</listeners>

</source>

</sources>

<switches>

<add name="sourceSwitch" value="Warning"/>

</switches>

</system.diagnostics>

      配置文件实现的对应代码部分为：

private static TraceSource mySource = new TraceSource("TraceSourceApp");

static void Main(string[] args)

{

Activity1();

Activity2();

Activity3();

mySource.Close();

return;

}

static void Activity1()

{

mySource.TraceEvent(TraceEventType.Error, 1, "Error message.");

mySource.TraceEvent(TraceEventType.Warning, 2, "Warning message.");

}

static void Activity2()

{

mySource.TraceEvent(TraceEventType.Critical, 3, "Critical message.");

mySource.TraceInformation("Informational message.");

}

static void Activity3()

{

mySource.TraceEvent(TraceEventType.Error, 4, "Error message.");

mySource.TraceInformation("Informational message.");

}

**5：设计一个日志系统**

      有了以上的知识之后，我们就可以来设计一个应用程序的日志系统（是的，我们不在需要LOG4NET）。我们来假设这个日志系统最基础的功能：

      1、以日期来创建日志名，以免历史日志全部写入一个文件中去；

      2、可以配置想要输出的日志类别，如Warning或者Error等；

      3、可以配置想要的日志内容：如StackTrace或者错误信息等；

      思路：

      1、在应用程序启动代码中，设置文件名。因为文件名要根据日期动态生成，所以不能使用配置文件；

      2、其它配置可以配置文件实现；

      3、DO IT；

**6：关于EventLog**

     关于EventLog是一个我不喜欢的功能，更多内容，可参考：[http://msdn.microsoft.com/zh-cn/library/aaaxk5bx(VS.80).aspx](http://msdn.microsoft.com/zh-cn/library/aaaxk5bx%28VS.80%29.aspx" \o "http://msdn.microsoft.com/zh-cn/library/aaaxk5bx(VS.80).aspx)

练习：

1.You are using the Microsoft Visual Studio 2005 IDE to examine the output of a method that returns a string.    
You assign the output of the method to a string variable named fName. You need to write a code segment that   
prints the following on a single line     The message. "Test Failed. " The value of fName if the value of fName does    
not equal "John" You also need to ensure that the code segment simultaneously facilitates uninterrupted execution      
of the application. Which code segment should you use?   
A. Debug.Assert(fName == "John", "Test FaileD. ", fName);   
B. Debug.WriteLineIf(fName != "John", fName, "Test Failed");   
C. if (fName != "John") {Debug.Print("Test FaileD. "); Debug.Print(fName);      }   
D. if (fName != "John") {Debug.WriteLine("Test FaileD. "); Debug.WriteLine(fName);     }   
Answer: B

17.You are testing a newly developed method named PersistToDB. This method accepts a parameter of type    
EventLogEntry. This method does not return a value. You need to create a code segment that helps you to test the      
method. The code segment must read entries from the application log of local computers and then pass the entries   
on to the PersistToDB method. The code block must pass only events of type Error or Warning from the source   
MySource to the PersistToDB method. Which code segment should you use?    
A. EventLog myLog = new EventLog("Application", ".");          
foreach (EventLogEntry entry in myLog.Entries)      
{if (entry.Source == "MySource") {  PersistToDB(entry);     }  }   
B. EventLog myLog = new EventLog("Application", ".");           
myLog.Source = "MySource";        
foreach (EventLogEntry entry in myLog.Entries) {             
if (entry.EntryType == (EventLogEntryType.Error & EventLogEntryType.Warning))      
{ PersistToDB(entry);  }}

C. EventLog myLog = new EventLog("Application", ".");           
foreach (EventLogEntry entry in myLog.Entries)      
{ if (entry.Source == "MySource")     
{if (entry.EntryType == EventLogEntryType.Error || entry.EntryType == EventLogEntryType.Warning)    
  {PersistToDB(entry); }  } }   
D. EventLog myLog = new EventLog("Application", ".");          
myLog.Source = "MySource";        
foreach (EventLogEntry entry in myLog.Entries)      
{if (entry.EntryType == EventLogEntryType.Error || entry.EntryType == EventLogEntryType.Warning)      
{PersistToDB(entry); }   
Answer: C

[**TraceSource记录程序日志**](http://www.cnblogs.com/sirkevin/archive/2011/11/11/2245344.html)

1.配置文件

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

<system.diagnostics>  
 <sources>  
 <source name="TraceError" switchValue="Error,ActivityTracing,Information">  
 <listeners>  
 <add initializeData="./clog.txt" type="System.Diagnostics.TextWriterTraceListener,

System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"  
 name="txtListener"

traceOutputOptions="DateTime,ProcessId,ThreadId" >

<filter type="System.Diagnostics.EventTypeFilter"

initializeData="Error,Information" />  
 </add>  
 </listeners>  
 </source>  
 </sources>

<trace autoflush="true"></trace>

</system.diagnostics>

2.实例化TraceSource

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 public TraceSource \_ts = new TraceSource("TraceError");

3.在需要记录日志的地方这样写

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 \_ts.TraceData(TraceEventType.Information, 0, Message);