[**(转)非常完善的Log4net详细说明**](http://www.cnblogs.com/zhangchenliang/p/4546352.html)

**1、概述**

log4net是.Net下一个非常优秀的开源日志记录组件。log4net记录日志的功能非常强大。它可以将日志分不同的等级，以不同的格式，输出到不同的媒介。本文主要是介绍如何在Visual Studio2008中使用log4net快速创建系统日志，如何扩展以输出自定义字段。

**2、一个简单的使用实例**

**第一步：**在项目中添加对log4net.dll的引用，这里引用版本是1.2.10.0。

**第二步：**程序启动时读取log4net的配置文件。

如果是CS程序，在根目录的Program.cs中的Main方法中添加：

log4net.Config.XmlConfigurator.Configure();

如果是BS程序，在根目录的Global.asax.cs（没有新建一个）中的Application\_Start方法中添加：

log4net.Config.XmlConfigurator.Configure();

无论BS还是CS程序都可直接在项目的AssemblyInfo.cs文件里添加以下的语句：

[assembly: log4net.Config .XmlConfigurator()]

也可以使用自定义的配置文件，具体请参见4.4 关联配置文件。

**第三步：**修改配置文件。如果是CS程序，则在默认的App.config文件（没有新建一个）中添加内容；如果是BS程序，则添加到Web.config文件中，添加内容一样，这里不再列出。

App.config文件添加内容如下：

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

  <configSections>

<section name="log4net"

type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />

  </configSections>

  <log4net>

    <root>

      <level value="WARN" />

      <appender-ref ref="LogFileAppender" />

      <appender-ref ref="ConsoleAppender" />

    </root>

    <logger name="testApp.Logging">

      <level value="DEBUG"/>

    </logger>

    <appender name="LogFileAppender" type="log4net.Appender.FileAppender" >

      <param name="File" value="log-file.txt" />

      <param name="AppendToFile" value="true" />

      <layout type="log4net.Layout.**PatternLayout**">

        <param name="Header" value="[Header] "/>

        <param name="Footer" value="[Footer] "/>

        <param name="ConversionPattern" value="%d [%t] %-5p %c [%x]  - %m%n" />

      </layout>

      <filter type="log4net.Filter.LevelRangeFilter">

        <param name="LevelMin" value="DEBUG" />

        <param name="LevelMax" value="WARN" />

      </filter>

    </appender>

    <appender name="ConsoleAppender"  type="log4net.Appender.ConsoleAppender" >

      <layout type="log4net.Layout.**PatternLayout**">

        <param name="ConversionPattern"  value="%d [%t] %-5p %c [%x] - %m%n" />

      </layout>

    </appender>

  </log4net>

</configuration>

**第四步：**在程序使用。

log4net.ILog log = log4net.LogManager.GetLogger("testApp.Logging");//获取一个日志记录器

log.Info(DateTime.Now.ToString() + ": login success");//写入一条新log

这样就将信息同时输出到控制台和写入到文件名为“log-file.txt”的文件中，其中“log-file.txt”文件的路径是当前程序运行所在目录；也可以定义为绝对路径，配置如：

<param name="File" value="C:/log-file.txt" />就写入C盘根目录下log-file.txt文件中，具体使用技巧参见4.2.1。

本例的实现请参见8.6附件。

**3、Log4net的主要组成部分**

**3.1 Appenders**

Appenders用来定义日志的输出方式，即日志要写到那种介质上去。较常用的Log4net已经实现好了，直接在配置文件中调用即可，可参见上面配置文件例子；当然也可以自己写一个，需要从log4net.Appender.AppenderSkeleton类继承。它还可以通过配置Filters和Layout来实现日志的过滤和输出格式。

已经实现的输出方式有：

AdoNetAppender 将日志记录到数据库中。可以采用SQL和存储过程两种方式。

AnsiColorTerminalAppender 将日志高亮输出到ANSI终端。

AspNetTraceAppender  能用asp.net中Trace的方式查看记录的日志。

BufferingForwardingAppender 在输出到子Appenders之前先缓存日志事件。

ConsoleAppender 将日志输出到应用程序控制台。

EventLogAppender 将日志写到Windows Event Log。

FileAppender 将日志输出到文件。

ForwardingAppender 发送日志事件到子Appenders。

LocalSyslogAppender 将日志写到local syslog service (仅用于UNIX环境下)。

MemoryAppender 将日志存到内存缓冲区。

NetSendAppender 将日志输出到Windows Messenger service.这些日志信息将在用户终端的对话框中显示。

OutputDebugStringAppender 将日志输出到Debuger，如果程序没有Debuger，就输出到系统Debuger。如果系统Debuger也不可用，将忽略消息。

RemoteSyslogAppender 通过UDP网络协议将日志写到Remote syslog service。

RemotingAppender 通过.NET Remoting将日志写到远程接收端。

RollingFileAppender 将日志以回滚文件的形式写到文件中。

SmtpAppender 将日志写到邮件中。

SmtpPickupDirAppender 将消息以文件的方式放入一个目录中，像IIS SMTP agent这样的SMTP代理就可以阅读或发送它们。

TelnetAppender 客户端通过Telnet来接受日志事件。

TraceAppender 将日志写到.NET trace 系统。

UdpAppender 将日志以无连接UDP数据报的形式送到远程宿主或用UdpClient的形式广播。

**3.2 Filters**

使用过滤器可以过滤掉Appender输出的内容。过滤器通常有以下几种：

DenyAllFilter 阻止所有的日志事件被记录

LevelMatchFilter 只有指定等级的日志事件才被记录

LevelRangeFilter 日志等级在指定范围内的事件才被记录

LoggerMatchFilter 与Logger名称匹配，才记录

PropertyFilter 消息匹配指定的属性值时才被记录

StringMathFilter 消息匹配指定的字符串才被记录

**3.3 Layouts**

Layout用于控制Appender的输出格式，可以是线性的也可以是XML。

一个Appender只能有一个Layout。

最常用的Layout应该是经典格式的**PatternLayout**，其次是SimpleLayout，**RawTimeStampLayout**和ExceptionLayout。然后还有IRawLayout，XMLLayout等几个，使用较少。Layout可以自己实现，需要从log4net.Layout.LayoutSkeleton类继承，来输出一些特殊需要的格式，在后面扩展时就重新实现了一个Layout。

SimpleLayout简单输出格式，只输出日志级别与消息内容。

**RawTimeStampLayout** 用来格式化时间，在向数据库输出时会用到。

样式如“yyyy-MM-dd HH:mm:ss“

ExceptionLayout需要给Logger的方法传入Exception对象作为参数才起作用，否则就什么也不输出。输出的时候会包含Message和Trace。

PatterLayout使用最多的一个Layout，能输出的信息很多，使用方式可参见上面例子中的配置文件。PatterLayout的格式化字符串见文后附注8.1。

**3.4 Loggers**

Logger是直接和应用程序交互的组件。Logger只是产生日志，然后由它引用的Appender记录到指定的媒介，并由Layout控制输出格式。

Logger提供了多种方式来记录一个日志消息，也可以有多个Logger同时存在。每个实例化的Logger对象对被log4net作为命名实体（Named Entity）来维护。log4net使用继承体系，也就是说假如存在两个Logger，名字分别为a.b.c和a.b。那么a.b就是a.b.c的祖先。每个Logger都继承了它祖先的属性。所有的Logger都从Root继承,Root本身也是一个Logger。

日志的等级，它们由高到底分别为：

OFF > FATAL > ERROR > WARN > INFO > DEBUG  > ALL

高于等级设定值方法（如何设置参见“配置文件详解”）都能写入日志， Off所有的写入方法都不写到日志里，ALL则相反。例如当我们设成Info时，logger.Debug就会被忽略而不写入文件，但是FATAL,ERROR,WARN,INFO会被写入，因为他们等级高于INFO。

在具体写日志时，一般可以这样理解日志等级：

FATAL（致命错误）：记录系统中出现的能使用系统完全失去功能，服务停止，系统崩溃等使系统无法继续运行下去的错误。例如，数据库无法连接，系统出现死循环。

ERROR（一般错误）：记录系统中出现的导致系统不稳定，部分功能出现混乱或部分功能失效一类的错误。例如，数据字段为空，数据操作不可完成，操作出现异常等。

WARN（警告）：记录系统中不影响系统继续运行，但不符合系统运行正常条件，有可能引起系统错误的信息。例如，记录内容为空，数据内容不正确等。

INFO（一般信息）：记录系统运行中应该让用户知道的基本信息。例如，服务开始运行，功能已经开户等。

DEBUG （调试信息）：记录系统用于调试的一切信息，内容或者是一些关键数据内容的输出。

Logger实现的ILog接口，ILog定义了5个方法（Debug,Inof,Warn,Error,Fatal）分别对不同的日志等级记录日志。这5个方法还有5个重载。以Debug为例说明一下，其它的和它差不多。

ILog中对Debug方法的定义如下：

void Debug(object message);

void Debug(object message, Exception ex);

还有一个布尔属性：

bool IsDebugEnabled { get; }

如果使用Debug(object message, Exception ex)，则无论Layout中是否定义了%exception，默认配置下日志都会输出Exception。包括Exception的Message和Trace。如果使用Debug(object message)，则日志是不会输出Exception。

最后还要说一个LogManager类，它用来管理所有的Logger。它的GetLogger静态方法，可以获得配置文件中相应的Logger：

log4net.ILog log = log4net.LogManager.GetLogger("logger-name");

**3.5 Object Renders**

它将告诉logger如何把一个对象转化为一个字符串记录到日志里。（ILog中定义的接口接收的参数是Object，而不是String。）

例如你想把Orange对象记录到日志中，但此时logger只会调用Orange默认的ToString方法而已。所以要定义一个OrangeRender类实现log4net.ObjectRender.IObjectRender接口，然后注册它（我们在本文中的扩展不使用这种方法，而是直接实现一个自定义的Layout）。这时logger就会知道如何把Orange记录到日志中了。

**3.6 Repository**

Repository主要用于日志对象组织结构的维护。

**4、配置文件详解**

**4.1 配置文件构成**

主要有两大部分，一是申明一个名为“log4net“的自定义配置节，如下所示：

  <configSections>

<section name="log4net"

type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />

  </configSections>

二是<log4net>节的具体配置，这是下面要重点说明的。

**4.1.1<log4net>**

所有的配置都要在<log4net>元素里定义。

支持的属性：

|  |  |
| --- | --- |
| debug | 可选，取值是true或false，默认是false。设置为true，开启log4net的内部调试。 |
| update | 可选，取值是Merge(合并)或Overwrite(覆盖)，默认值是Merge。设置为Overwrite，在提交配置的时候会重置已经配置过的库。 |
| threshold | 可选，取值是repository（库）中注册的level，默认值是ALL。 |

支持的子元素：

|  |  |
| --- | --- |
| appender | 0或多个 |
| logger | 0或多个 |
| renderer | 0或多个 |
| root | 最多一个 |
| param | 0或多个 |

**4.1.2 <root>**

实际上就是一个根logger，所有其它logger都默认继承它，如果配置文件里没有显式定义，则框架使用根日志中定义的属性。root元素没有属性。

支持的子元素：

|  |  |
| --- | --- |
| appender-ref | 0个或多个，要引用的appender的名字。 |
| level | 最多一个。 只有在这个级别或之上的事件才会被记录。 |
| param | 0个或多个， 设置一些参数。 |

**4.1.3 <logger>**

支持的属性：

|  |  |
| --- | --- |
| name | 必须的，logger的名称 |
| additivity | 可选，取值是true或false，默认值是true。设置为false时将阻止父logger中的appender。 |

支持的子元素：

|  |  |
| --- | --- |
| appender-ref | 0个或多个，要引用的appender的名字。 |
| level | 最多一个。 只有在这个级别或之上的事件才会被记录。 |
| param | 0个或多个， 设置一些参数。 |

**4.1.4 <appender>**

定义日志的输出方式，只能作为 log4net 的子元素。name属性必须唯一，type属性必须指定。

支持的属性：

|  |  |
| --- | --- |
| name | 必须的，Appender对象的名称 |
| type | 必须的，Appender对象的输出类型 |

支持的子元素：

|  |  |
| --- | --- |
| appender-ref | 0个或多个，允许此appender引用其他appender，并不是所以appender类型都支持。 |
| filter | 0个或多个，定义此app使用的过滤器。 |
| layout | 最多一个。定义appender使用的输出格式。 |
| param | 0个或多个， 设置Appender类中对应的属性的值。 |

实际上<appender>所能包含的子元素远不止上面4个。

**4.1.5 <layout>**

布局，只能作为<appender>的子元素。

支持的属性：

|  |  |
| --- | --- |
| type | 必须的，Layout的类型 |

支持的子元素：

|  |  |
| --- | --- |
| param | 0个或多个， 设置一些参数。 |

**4.1.6 <filter>**

过滤器，只能作为<appender>的子元素。

支持的属性：

|  |  |
| --- | --- |
| type | 必须的，Filter的类型 |

支持的子元素：

|  |  |
| --- | --- |
| param | 0个或多个， 设置一些参数。 |

**4.1.7 <param>**

<param>元素可以是任何元素的子元素。

支持的属性：

|  |  |
| --- | --- |
| name | 必须的，取值是父对象的参数名。 |
| value | 可选的，value和type中，必须有一个属性被指定。value是一个能被转化为参数值的字符串。 |
| type | 可选的，value和type中，必须有一个属性被指定。type是一个类型名，如果type不是在log4net程序集中定义的，就需要使用全名。 |

支持的子元素：

|  |  |
| --- | --- |
| param | 0个或多个， 设置一些参数。 |

**4.2 <appender>配置**

<appender>在配置文件中至少有一个，也可以有多个，有些<appender>类型还可以引用其他<appender>类型，具体参数可参见上表。

下面只对写入回滚文件与输出到数据库（这里使用SQL数据库）配置体会说一下，其他配置可参考官方网站：<http://logging.apache.org/log4net/release/config-examples.html>

**4.2.1写入回滚文件**

    <appender name="ReflectionLayout" type="log4net.Appender.RollingFileAppender,log4net">

<!--日志文件路径，“/”与“/”作用相同，到达的目录相同，文件夹不存在则新建 -->

<!--按文件大小方式输出时在这里指定文件名，并且当天的日志在下一天时在文件名后自动追加当天日期形成新文件。-->

<!—按照日期形式输出时，直接连接元素DatePattern的value形成文件路径。此处使用这种方式 -->

<!--param的名称,可以直接查对应的appender类的属性名即可,这里要查的就是RollingFileAppender类的属性 -->

      <param name="File" value="D:/Log/" />

      <!--是否追加到文件-->

      <param name="AppendToFile" value="true" />

      <!--记录日志写入文件时，不锁定文本文件，防止多线程时不能写Log,官方说线程非安全-->

      <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />

      <!—使用Unicode编码-->

      <Encoding value="UTF-8" />

      <!--最多产生的日志文件数，超过则只保留最新的n个。设定值value="－1"为不限文件数-->

      <param name="MaxSizeRollBackups" value="10" />

      <!--是否只写到一个文件中-->

      <param name="StaticLogFileName" value="false" />

      <!--按照何种方式产生多个日志文件(日期[Date],文件大小[Size],混合[Composite])-->

      <param name="RollingStyle" value="Composite" />

      <!--按日期产生文件夹和文件名［在日期方式与混合方式下使用］-->

<!—此处按日期产生文件夹，文件名固定。注意&quot; 的位置-->

      <param name="DatePattern" value="yyyy-MM-dd/&quot;ReflectionLayout.log&quot;"  />

<!—这是按日期产生文件夹，并在文件名前也加上日期-->

      <param name="DatePattern" value="yyyyMMdd/yyyyMMdd&quot;-TimerServer.log&quot;"  />

<!—这是先按日期产生文件夹，再形成下一级固定的文件夹—>

      <param name="DatePattern" value="yyyyMMdd/&quot;TimerServer/TimerServer.log&quot;"  />

      <!--每个文件的大小。只在混合方式与文件大小方式下使用。

超出大小后在所有文件名后自动增加正整数重新命名，数字最大的最早写入。

可用的单位:KB|MB|GB。不要使用小数,否则会一直写入当前日志-->

      <param name="maximumFileSize" value="500KB" />

<!--计数类型为1，2，3…-->  
      <param name="CountDirection" value="1"/>

<!—过滤设置，LevelRangeFilter为使用的过滤器。 -->

      <filter type="log4net.Filter.LevelRangeFilter">

        <param name="LevelMin" value="DEBUG" />

        <param name="LevelMax" value="WARN" />

      </filter>

      <!--记录的格式。一般用log4net.Layout.**PatternLayout**布局-->

<!—此处用继承了log4net.Layout.**PatternLayout**的自定义布局，TGLog.ExpandLayout2

为命名空间。%property{Operator}、%property{Action}是自定义的输出-－>

      <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

        <param name="ConversionPattern"

 value="记录时间：%date 线程ID:[%thread] 日志级别：%-5level 记录类：%logger     操作者ID：%property{Operator} 操作类型：%property{Action}%n             当前机器名:%property%n当前机器名及登录用户：%username %n               记录位置：%location%n 消息描述：%property{Message}%n                    异常：%exception%n 消息：%message%newline%n%n" />

      </layout>

</appender>

注意这些配置属性有些是可选的，如果需要，一定要写正确，否则要么输出的不是自己想要的结果，要么干脆不输出任何信息。

**4.2.1写入SQL数据库**

需要在相应的数据库中准备好一张表，创建语句如下：

CREATE TABLE [Log] (

[ID] [int] IDENTITY (1, 1) NOT NULL ,

[Date] [datetime] NOT NULL ,

[Thread] [varchar] (100) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[Level] [varchar] (100) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[Logger] [varchar] (200) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[Operator] [int] NULL ,

[Message] [text] COLLATE Chinese\_PRC\_CI\_AS NULL ,

[ActionType] [int] NULL ,

[Operand] [varchar] (300) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[IP] [varchar] (20) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[MachineName] [varchar] (100) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[Browser] [varchar] (50) COLLATE Chinese\_PRC\_CI\_AS NULL ,

[Location] [text] COLLATE Chinese\_PRC\_CI\_AS NULL ,

[Exception] [text] COLLATE Chinese\_PRC\_CI\_AS NULL

)

<appender name="ADONetAppender" type="log4net.Appender.ADONetAppender,log4net">

<!--BufferSize为缓冲区大小，只有日志记录超设定值才会一块写入到数据库-->

<bufferSize value="10" /><!—或写为<param name="BufferSize" value="10" />-->

<!--引用-->

<connectionType value="System.Data.SqlClient.SqlConnection, System.Data, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />

<!--连接数据库字符串-->

<connectionString value="data source=.;initial catalog=Test;integrated security=false;persist security info=True;User ID=sa;Password=;" />

<!--插入到表Log-->

<commandText value="INSERT INTO Log ([Date],[Thread],[Level],[Logger],[Operator],[Message],[ActionType],[Operand],[IP],[MachineName],[Browser],[Location],[Exception]) VALUES (@log\_date, @thread, @log\_level, @logger,@operator, @message,@action\_type,@operand,@ip,@machineName,@browser,@location,@exception)" />

<!—日志记录时间，**RawTimeStampLayout**为默认的时间输出格式 -->

      <parameter>

        <parameterName value="@log\_date" />

        <dbType value="DateTime" />

        <layout type="log4net.Layout.**RawTimeStampLayout**" />

      </parameter>

      <!--线程号-->

      <parameter>

        <parameterName value="@thread" />

        <dbType value="String" />

<!—长度不可以省略，否则不会输出-->

        <size value="100" />

        <layout type="log4net.Layout.**PatternLayout**">

          <conversionPattern value="%thread" />

        </layout>

      </parameter>

      <!--日志等级-->

      <parameter>

        <parameterName value="@log\_level" />

        <dbType value="String" />

        <size value="100" />

        <layout type="log4net.Layout.**PatternLayout**">

          <conversionPattern value="%level" />

        </layout>

      </parameter>

      <!--日志记录类名称-->

      <parameter>

        <parameterName value="@logger" />

        <dbType value="String" />

        <size value="200" />

        <layout type="log4net.Layout.**PatternLayout**">

          <conversionPattern value="%logger" />

        </layout>

      </parameter>

      <!--操作者。这个是自定义的输出字段，使用重新实现的布局器ReflectionLayout -->

      <parameter>

        <parameterName value="@operator" />

<!—设置为Int32时只有bufferSize的 value<="1"才正确输出，没有找出原因。-->

        <dbType value="Int16" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{Operator}" />

        </layout>

      </parameter>

      <!--操作对象-->

      <parameter>

        <parameterName value="@operand" />

        <dbType value="String" />

        <size value="300" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{Operand}" />

        </layout>

      </parameter>

      <!—IP地址-->

      <parameter>

        <parameterName value="@ip" />

        <dbType value="String" />

        <size value="20" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{IP}" />

        </layout>

      </parameter>

      <!--机器名-->

      <parameter>

        <parameterName value="@machineName" />

        <dbType value="String" />

        <size value="100" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{MachineName}" />

        </layout>

      </parameter>

      <!--浏览器-->

      <parameter>

        <parameterName value="@browser" />

        <dbType value="String" />

        <size value="50" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{Browser}" />

        </layout>

      </parameter>

      <!—日志消息-->

      <parameter>

        <parameterName value="@message" />

        <dbType value="String" />

        <size value="3000" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{Message}" />

        </layout>

      </parameter>

      <!--动作类型-->

      <parameter>

        <parameterName value="@action\_type" />

        <dbType value="Int16" />

        <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

          <conversionPattern value="%property{ActionType}" />

        </layout>

      </parameter>

      <!—记录日志的位置-->

      <parameter>

        <parameterName value="@location" />

        <dbType value="String" />

        <size value="2000" />

        <layout type="log4net.Layout.**PatternLayout**">

          <conversionPattern value="%location" />

        </layout>

      </parameter>

      <!—异常信息。ExceptionLayout 为异常输出的默认格式-->

      <parameter>

        <parameterName value="@exception" />

        <dbType value="String" />

        <size value="4000" />

        <layout type="log4net.Layout.ExceptionLayout" />

      </parameter>

</appender>

注意：

向表中输出的字段不能多于数据表本身字段，而反之则可以，但这些多余字段一定使其可以为空，否则便写不到数据库；

输出字段的类型一定是对应数据表字段数据类型可以隐式转换的，而且长度也不能超过，否则也不能写入；

数据表字段设置尽量可以为空，这样可以避免一条日志记录存在空数据导致后面的日志都记录不了。

**4.3<logger>的配置**

在配置文件<appender>中的配置好了输出的介质，格式，过滤方式，还要定义日志对象<logger>。

在框架的体系里，所有的日志对象都是根日志(root logger)的后代。 因此如果一个日志对象没有在配置文件里显式定义，则框架使用根日志中定义的属性。在<root>标签里，可以定义level级别值和Appender的列表。如果没有定义LEVEL的值，则缺省为DEBUG。可以通过<appender-ref>标签定义日志对象使用的Appender对象。<appender-ref>声明了在其他地方定义的Appender对象的一个引用。在一个logger对象中的设置会覆盖根日志的设置。而对Appender属性来说，子日志对象则会继承父日志对象的Appender列表。这种缺省的行为方式也可以通过显式地设定<logger>标签的additivity属性为false而改变。

<root>不显式申明时使用默认的配置。我觉得在使用时不定义<root>,自定义多个<logger>,在程序中记录日志时直接使用<logger>的name来查找相应的<logger>，这样更灵活一些。例如：

<!--同时写两个文件和数据库-->

<logger name="ReflectionLayout">

      <level value="DEBUG"/>

      <appender-ref ref="HashtableLayout"/>

      <appender-ref ref="ReflectionLayout"/>

      <appender-ref ref="ADONetAppender"/>

</logger>

**4.4关联配置文件**

log4net默认关联的是应用程序的配置文件App.config(BS程序是Web.config)，可以使用程序集自定义属性来进行设置。下面来介绍一下这个自定义属性：

log4net.Config.XmlConifguratorAttribute。

XmlConfiguratorAttribute有3个属性：

ConfigFile： 配置文件的名字，文件路径相对于应用程序目录

(AppDomain.CurrentDomain.BaseDirectory)。ConfigFile属性不能和ConfigFileExtension属性一起使用。

ConfigFileExtension： 配置文件的扩展名，文件路径相对于应用程序的目录。ConfigFileExtension属性不能和ConfigFile属性一起使用。

Watch： 如果将Watch属性设置为true，就会监视配置文件。当配置文件发生变化的时候，就会重新加载。

如果ConfigFile和ConfigFileExtension都没有设置，则使用应用程序的配置文件App.config（Web.config）。

可以在项目的AssemblyInfo.cs文件里添加以下的语句：

 //监视默认的配置文件，App.exe.config

[assembly: log4net.Config.XmlConfigurator(Watch = true)]

//监视配置文件，App.exe.log4net。

[assembly: log4net. Config.XmlConfigurator(ConfigFileExtension = "log4net", Watch = true)]

//使用配置文件log4net.config，不监视改变。注意log4net.config文件的目录，BS程序在站点目录//下，CS则在应用程序启动目录下，如调试时在/bin/Debug下，一般将文件属性的文件输出目录调为//始终复制即可

[assembly: log4net. Config.XmlConfigurator(ConfigFile = "log4net.config")]

//使用配置文件log4net.config，不监视改变

[assembly: log4net. Config.XmlConfigurator()]

也可以在Global.asax的Application\_Start里或者是Program.cs中的Main方法中添加，注意这里一定是绝对路径，如下所示：

//这是在BS程序下，使用自定义的配置文件log4net.xml，使用Server.MapPath("~") + //@"/log4net.xml”来取得路径。/log4net.xml为相对于站点的路径

// ConfigureAndWatch()相当于Configure(Watch = true)

log4net.Config.XmlConfigurator.ConfigureAndWatch(

new System.IO.FileInfo(Server.MapPath("~") + @"/log4net.xml"));

//这是在CS程序下，可以用以下方法获得：

string assemblyFilePath = Assembly.GetExecutingAssembly().Location;

string assemblyDirPath = Path.GetDirectoryName(assemblyFilePath);

string configFilePath = assemblyDirPath + " [//log4net.xml](file:///\\log4net.xml\)";

log4net.Config.XmlConfigurator.ConfigureAndWatch(

new FileInfo(configFilePath));

或直接使用绝对路径：

//使用自定义的配置文件，直接绝对路径为：c:/log4net.config

log4net.Config.XmlConfigurator.Configure(new System.IO.FileInfo(@"c:/log4net.config"));

**5、如何记录日志**

Log4net使用很方便，先申明一个封装类ILog 的对象，如下：

log4net.ILog log = log4net.LogManager.GetLogger("ReflectionLayout");

其中"ReflectionLayout"便是我们自定义的日志对象<logger>的name的值。

对应5个日志输出级别，log有5 个方法，每个方法都有两个重载，使用如下：

try

            {

                log.Debug("这是一个测试！");

            }

            catch(Exception ec)

            {

                log.Error("出现错误！", ec);

         }

如果我们需要输出的消息是要区别开来，不按一个字符串全部输出，就需要进行一些扩展了。

**6、Log4net的简单扩展**

**6.1通过重写布局Layout输出传入的 message对象的属性**

**6.1.1重写Layout类**

通过继承log4net.Layout.**PatternLayout**类，使用log4net.Core.LoggingEvent类的方法得到了要输出的message类的名称，然后通过反射得到各个属性的值，使用**PatternLayout**类AddConverter方法传入得到的值。这里注意要引用用到的类的命名空间。

代码见附注8.2。

**6.1.2配置相应的配置文件**

配置文件其他地方不用改动，只是需要改动<appender>中的<layout>。例如：

<layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

        <param name="ConversionPattern"

 value="记录时间：%date    操作者ID：%property{Operator}

操作类型：%property{Action}%n  消息描述：%property{Message}%n                    异常：%exception%n " />

      </layout>

其中<layout>的type由原来的log4net.Layout.**PatternLayout**换为自定义的TGLog.ExpandLayout2.ReflectionLayout（TGLog.ExpandLayout2为命名空间）。%property{Operator}输出的即为message类对象的属性Operator的值。数据库配置同样，相应的字段如果是自定义的，则输出选用自定义的<layout>。例：

<!--动作类型-->

  <parameter>

      <parameterName value="@action\_type" />

      <dbType value="Int16" />

      <layout type="TGLog.ExpandLayout2.ReflectionLayout,TGLog">

         <conversionPattern value="%property{ActionType}" />

      </layout>

  </parameter>

**6.1.3程序中如何使用**

和一般使用方法基本相同，只是传入的参数是一个自定义的类，类的属性和配置文件中<layout>所有的%property{属性}是一致的，即%property{属性}在输出的时候就查找传入message类中有无对应的属性，如果有就输出值，没有则输出null。例：

log4net.ILog log = log4net.LogManager.GetLogger("ReflectionLayout");

try

            {

                log.Debug(new LogMessage(

1,

"操作对象：0",

 (int)TGLog.ActionType.Other,

 "这是四个参数测试")

);

            }

            catch(Exception ec)

            {

                log.Error(new LogMessage(

                                    1,

                                    "操作对象：0",

                                    (int)TGLog.ActionType.Other,

                                    "这是全部参数测试",

                                    "192.168.1.1",

                                    "MyComputer",

                                    "Maxthon(MyIE2)Fans"),

                         ec

);

      }

LogMessage的全部属性的构造方法如下：

public LogMessage(

            int operatorID,

            string operand,

            int ActionType,

            string message,

            string ip,

            string machineName,

            string browser

            )

     {

            this.ActionType = ActionType;

            this.Operator = operatorID;

            this.Message = message;

            this.Operand = operand;

            this.IP = ip;

            this.Browser = browser;

            this.MachineName = machineName;

}

**6.2通过重新实现ILog接口来增加输入的参数**

**6.2.1重写LogImpl，LogManager类及实现ILog接口**

这种方式是通过构造一个名为**IMyLog**接口，是继承**Ilog**接口而来，然后分别在**MyLogImpl，MyLogManager**重新实现**IMyLog**接口，增加了每种方法的参数。**MyLogImpl，MyLogManager**分别继承**LogImpl，LogManager**而来。

代码分别见8.3、8.4、8.5：

**6.2.2配置相应的配置文件**

配置文件其他地方不用改动，只是需要改动<appender>中的<layout>元素name为ConversionPattern的value中输出格式。例如：

<layout type=" log4net.Layout.**PatternLayout** ">

        <param name="ConversionPattern"

 value="记录时间：%date    操作者ID：%property{Operator}

操作类型：%property{Action}%n  消息描述：%property{Message}%n                    异常：%exception%n " />

      </layout>

%property{参数}中的参数在**MyLogImpl**类中定义，如语句：

loggingEvent.Properties["Operator"] = operatorID;

就定义了Operator输出参数，即%property{Operator}输出的即为**IMyLog**中的参数operatorID的值。

数据库配置同样。例：

<!--动作类型-->

  <parameter>

      <parameterName value="@action\_type" />

      <dbType value="Int16" />

      <layout type=" log4net.Layout.**PatternLayout** ">

         <conversionPattern value="%property{ActionType}" />

      </layout>

  </parameter>

**6.2.3程序中如何使用**

先引用IMyLog ，MyLogManager所在的命名空间，创建一个IMyLog对象，myLog的5 个方法，每个方法都有四个重载，增加了多参数的重载。例：

IMyLog myLog = MyLogManager.GetLogger("ExpandILog");

try

            {

myLog.Debug("这是一个参数重载测试!");

}

            catch(Exception ec)

            {

                log.Error(

                          1,

                          "操作对象：0",

                          (int)TGLog.ActionType.Other,

                          "这是全部参数测试",

                          "192.168.1.1",

                          "MyComputer",

                          "Maxthon(MyIE2)Fans",

                          ec

);

      }

**7、总结**

Log4net 功能很多，这里只是对已经尝试用过的功能总结一下，普通写日志已经足够。需要注意的是：

1.            Log4net本身也有一些缺陷，比如一个记录引起了log4net本身的异常，就会使后面的日志无法记录下来，尤其是在写入数据库时。例如使用6.1扩展后，int型的属性在<appender >的元素<bufferSize>设置不为1时，<dbType value="Int32" />时，就不能输出到数据库，而<dbType value="Int16" />则没任何问题。

2.            Log4net本身出现了异常，比如配置文件出现错误，有些日志输出方式会记录下这些异常，例如应用程序控制台；有些则不会输出这些错误，如数据库与文件。

3.            扩展时也会留下一些问题。例如在使用6.1扩展输出字段时就会出现，在log.debug(object message)中，如果message是一个自定义的类，属性与配置文件中输出设置也一致，构造函数时也只构造一个参数的实例，写文件与写数据库都成功，而将message按没有扩展的方式直接传入一个字符串，即log.debug(“信息内容”)使用则只能写入文件，而数据库则没写入。自定义的Layout 就是继承默认的**PatternLayout**，本来不应该出错，但出现了问题。原因分析是自定义的message类有类型为int的属性，作为一个对象传入时在默认值0，而直接使用字符串则int型的字段得不到默认值，引发异常。所以建议在有扩展存在时，最好多设几个<logger>,区分清楚，按照统一的形式记录日志，不要混合使用。

4.            配置文件的设置一定要准确，在一点不正确就会导致日志不能正常输出，所以在配置时先从最简单的开始，同时输出方式选择一种能输出log4net本身异常的方式，成功后一点一点加在新配置，这样出错了也容易找到那个地方配置有问题。

5.            log4net扩展性很强，几乎所有的组件都可以重写，在配置文件中配置好就可以使用。

**8、附注：**

**8.1PatterLayout格式化字符表**

|  |  |
| --- | --- |
| **转换字符** | **效果** |
| a | 等价于appdomain |
| appdomain | 引发日志事件的应用程序域的友好名称。（使用中一般是可执行文件的名字。） |
| c | 等价于 logger |
| C | 等价于 type |
| class | 等价于 type |
| d | 等价于 date |
| date | 发生日志事件的本地时间。 使用 DE>%utcdate 输出UTC时间。date后面还可以跟一个日期格式，用大括号括起来。DE>例如：%date{HH:mm:ss,fff}或者%date{dd MMM yyyy HH:mm:ss,fff}。如果date后面什么也不跟，将使用ISO8601 格式 。  日期格式和.Net中DateTime类的ToString方法中使用的格式是一样。  另外log4net还有3个自己的格式Formatter。 它们是 "ABSOLUTE", "DATE"和"ISO8601"分别代表 AbsoluteTimeDateFormatter, DateTimeDateFormatter和Iso8601DateFormatter。例如：%date{ISO8601}或%date{ABSOLUTE}。  它们的性能要好于ToString。 |
| exception | 异常信息  日志事件中必须存了一个异常对象，如果日志事件不包含没有异常对象，将什么也不输出。异常输出完毕后会跟一个换行。一般会在输出异常前加一个换行，并将异常放在最后。 |
| F | 等价于 file |
| file | 发生日志请求的源代码文件的名字。  警告：只在调试的时候有效。调用本地信息会影响性能。 |
| identity | 当前活动用户的名字(Principal.Identity.Name).  警告：会影响性能。（我测试的时候%identity返回都是空的。） |
| l | 等价于 location |
| L | 等价于 line |
| location | 引发日志事件的方法（包括命名空间和类名），以及所在的源文件和行号。  警告：会影响性能。没有pdb文件的话，只有方法名，没有源文件名和行号。 |
| level | 日志事件等级 |
| line | 引发日志事件的行号  警告：会影响性能。 |
| logger | 记录日志事件的Logger对象的名字。  可以使用精度说明符控制Logger的名字的输出层级，默认输出全名。  注意，精度符的控制是从右开始的。例如：logger 名为 "a.b.c"， 输出模型为%logger{2} ，将输出"b.c"。 |
| m | 等价于 message |
| M | 等价于 method |
| message | 由应用程序提供给日志事件的消息。 |
| mdc | MDC (旧为：ThreadContext.Properties) 现在是事件属性的一部分。 保留它是为了兼容性，它等价于 property。 |
| method | 发生日志请求的方法名（只有方法名而已）。  警告：会影响性能。 |
| n | 等价于 newline |
| newline | 换行符 |
| ndc | NDC (nested diagnostic context) |
| p | 等价于 level |
| P | 等价于 property |
| properties | 等价于 property |
| property | 输出事件的特殊属性。例如： %property{user} 输出user属性。属性是由loggers或appenders添加到时间中的。 有一个默认的属性"DE>log4net:HostName"总是会有。DE>  %property将输出所有的属性 。  （扩展后可以使用） |
| r | 等价于 timestamp |
| t | 等价于 thread |
| timestamp | 从程序启动到事件发生所经过的毫秒数。 |
| thread | 引发日志事件的线程，如果没有线程名就使用线程号。 |
| type | 引发日志请求的类的全名。.  可以使用精度控制符。例如： 类名是 "log4net.Layout.**PatternLayout**", 格式模型是%type{1} 将输出"**PatternLayout**"。（也是从右开始的。）  警告：会影响性能。 |
| u | 等价于 identity |
| username | 当前用户的WindowsIdentity。（类似：HostName/Username）  警告：会影响性能。 |
| utcdate | 发生日志事件的UTC时间。DE>后面还可以跟一个日期格式，用大括号括起来。DE>例如：%utcdate{HH:mm:ss,fff}或者%utcdate{dd MMM yyyy HH:mm:ss,fff}。如果utcdate后面什么也不跟，将使用ISO8601 格式 。  日期格式和.Net中DateTime类的ToString方法中使用的格式是一样。  另外log4net还有3个自己的格式Formatter。 它们是 "ABSOLUTE", "DATE"和"ISO8601"分别代表 AbsoluteTimeDateFormatter, DateTimeDateFormatter和Iso8601DateFormatter。例如：%date{ISO8601}或%date{ABSOLUTE}。  它们的性能要好于ToString。 |
| w | 等价于 username |
| x | 等价于 ndc |
| X | 等价于 mdc |
| % | %%输出一个百分号 |

关于调用本地信息（caller location information）的说明：

%type %file %line %method %location %class %C %F %L %l %M 都会调用本地信息。这样做会影响性能。本地信息使用System.Diagnostics.StackTrace得到。.Net 1.0 不支持System.Diagnostics.StackTrace 类。

本地信息在调试模式下可以正常获取，在非调试模式下可能获取不到，或只能获取一部分。（根据我的测试，其实是需要有一个程序数据库（.pdb）文件。）

%property属性要用代码来设置才能使用（也就是扩展一下），

默认属性log4net:HostName不用设置。

转义字符的修饰符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Format modifier | left justify | minimum width | maximum width | comment |
| %20logger | false | 20 | none | 如果logger名不足20个字符，就在左边补空格。 |
| %-20logger | true | 20 | none | 如果logger名不足20个字符，就在右边补空格。 |
| %.30logger | NA | none | 30 | 超过30个字符将截断。 |
| %20.30logger | false | 20 | 30 | logger名要在20到30之间，少了在左边补空格，多了截断。 |
| %-20.30logger | true | 20 | 30 | logger名要在20到30之间，少了在右边补空格，多了截断。 |

**8.2Layout类代码**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using log4net.Layout;

using log4net.Layout.Pattern;

using System.Reflection;

using System.Collections;

using FastReflectionLib;

namespace TGLog.ExpandLayout2

{

    public class ReflectionLayout : PatternLayout

    {

        public ReflectionLayout()

        {

            this.AddConverter("property", typeof(ReflectionPatternConverter));

        }

    }

    public class ReflectionPatternConverter : PatternLayoutConverter

    {

        protected override void Convert(

System.IO.TextWriter writer,

 log4net.Core.LoggingEvent loggingEvent

)

        {

            if (Option != null)

            {

                // 写入指定键的值

                WriteObject(

writer,

 loggingEvent.Repository,

 LookupProperty(Option,

 loggingEvent)

);

            }

            else

            {

                // 写入所有关键值对

                WriteDictionary(

writer,

loggingEvent.Repository,

 loggingEvent.GetProperties()

);

            }

        }

        /// <summary>

        /// 通过反射获取传入的日志对象的某个属性的值

        /// </summary>

        /// <param name="property"></param>

        /// <returns></returns>

        private object LookupProperty(

string property,

 log4net.Core.LoggingEvent loggingEvent)

        {

            object propertyValue = string.Empty;

            PropertyInfo propertyInfo =

loggingEvent.MessageObject.GetType().GetProperty(property);

            if (propertyInfo != null)

            {

                propertyValue =

propertyInfo.GetValue(loggingEvent.MessageObject, null);

            }

            return propertyValue;

        }

    }

}

**8.3 MyLogImpl类代码**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using log4net.Core;

namespace TGLog.ExpandILog

{

    public class MyLogImpl : LogImpl, IMyLog

    {

        /// <summary>

        /// The fully qualified name of this declaring type not the type of any subclass.

        /// </summary>

        private readonly static Type ThisDeclaringType = typeof(MyLogImpl);

        public MyLogImpl(ILogger logger)

            : base(logger)

        {

        }

        #region Implementation of IMyLog

        public void Debug(int operatorID, string operand, int actionType,object message,

 string ip, string browser, string machineName)

        {

            Debug(operatorID,  operand,  actionType, message,

  ip,  browser, machineName, null);

        }

        public void Debug(int operatorID, string operand, int actionType,object message,

string ip, string browser, string machineName, System.Exception t)

        {

            if (this.IsDebugEnabled)

            {

                LoggingEvent loggingEvent =

new LoggingEvent(ThisDeclaringType, Logger.Repository,

                                       Logger.Name, Level.Info, message, t);

                loggingEvent.Properties["Operator"] = operatorID;

                loggingEvent.Properties["Operand"] = operand;

                loggingEvent.Properties["ActionType"] = actionType;

                loggingEvent.Properties["IP"] = ip;

                loggingEvent.Properties["Browser"] = browser;

                loggingEvent.Properties["MachineName"] = machineName;

                Logger.Log(loggingEvent);

            }

        }

        public void Info(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName)

        {

            Info(operatorID, operand, actionType, message, ip, browser, machineName, null);

        }

        public void Info(int operatorID, string operand, int actionType, object message,

 string ip, string browser, string machineName, System.Exception t)

        {

            if (this.IsInfoEnabled)

            {

                LoggingEvent loggingEvent =

 new LoggingEvent(ThisDeclaringType, Logger.Repository,

 Logger.Name, Level.Info, message, t);

                loggingEvent.Properties["Operator"] = operatorID;

                loggingEvent.Properties["Operand"] = operand;

                loggingEvent.Properties["ActionType"] = actionType;

                loggingEvent.Properties["IP"] = ip;

                loggingEvent.Properties["Browser"] = browser;

                loggingEvent.Properties["MachineName"] = machineName;

                Logger.Log(loggingEvent);

            }

        }

        public void Warn(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName)

        {

            Warn(operatorID, operand, actionType, message, ip, browser, machineName, null);

        }

        public void Warn(int operatorID, string operand, int actionType, object message,

 string ip, string browser, string machineName, System.Exception t)

        {

            if (this.IsWarnEnabled)

            {

                LoggingEvent loggingEvent =

 new LoggingEvent(ThisDeclaringType, Logger.Repository,

Logger.Name, Level.Info, message, t);

                loggingEvent.Properties["Operator"] = operatorID;

                loggingEvent.Properties["Operand"] = operand;

                loggingEvent.Properties["ActionType"] = actionType;

                loggingEvent.Properties["IP"] = ip;

                loggingEvent.Properties["Browser"] = browser;

                loggingEvent.Properties["MachineName"] = machineName;

                Logger.Log(loggingEvent);

            }

        }

        public void Error(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName)

        {

            Error(operatorID, operand, actionType, message, ip, browser, machineName, null);

        }

        public void Error(int operatorID, string operand, int actionType, object message,

 string ip, string browser, string machineName, System.Exception t)

        {

            if (this.IsErrorEnabled)

            {

                LoggingEvent loggingEvent =

 new LoggingEvent(ThisDeclaringType, Logger.Repository,

 Logger.Name, Level.Info, message, t);

                loggingEvent.Properties["Operator"] = operatorID;

                loggingEvent.Properties["Operand"] = operand;

                loggingEvent.Properties["ActionType"] = actionType;

                loggingEvent.Properties["IP"] = ip;

                loggingEvent.Properties["Browser"] = browser;

                loggingEvent.Properties["MachineName"] = machineName;

                Logger.Log(loggingEvent);

            }

        }

        public void Fatal(int operatorID, string operand, int actionType, object message,

 string ip, string browser, string machineName)

        {

            Fatal(operatorID, operand, actionType, message, ip, browser, machineName, null);

        }

        public void Fatal(int operatorID, string operand, int actionType, object message,

 string ip, string browser, string machineName, System.Exception t)

        {

            if (this.IsFatalEnabled)

            {

                LoggingEvent loggingEvent =

 new LoggingEvent(ThisDeclaringType, Logger.Repository,

                                       Logger.Name, Level.Info, message, t);

                loggingEvent.Properties["Operator"] = operatorID;

                loggingEvent.Properties["Operand"] = operand;

                loggingEvent.Properties["ActionType"] = actionType;

                loggingEvent.Properties["IP"] = ip;

                loggingEvent.Properties["Browser"] = browser;

                loggingEvent.Properties["MachineName"] = machineName;

                Logger.Log(loggingEvent);

            }

        }

        #endregion Implementation of IMyLog

    }

}

**8.4 MyLogManager类代码**

#region Copyright & License

//

// Copyright 2001-2005 The Apache Software Foundation

//

// Licensed under the Apache License, Version 2.0 (the "License");

// you may not use this file except in compliance with the License.

// You may obtain a copy of the License at

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// Unless required by applicable law or agreed to in writing, software

// distributed under the License is distributed on an "AS IS" BASIS,

// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

// See the License for the specific language governing permissions and

// limitations under the License.

//

#endregion

using System;

using System.Reflection;

using System.Collections;

using log4net;

using log4net.Core;

using log4net.Repository;

using log4net.Repository.Hierarchy;

namespace TGLog.ExpandILog

{

    public class MyLogManager

    {

        #region Static Member Variables

        /// <summary>

        /// The wrapper map to use to hold the <see cref="EventIDLogImpl"/> objects

        /// </summary>

        private static readonly WrapperMap s\_wrapperMap = new WrapperMap(newWrapperCreationHandler(WrapperCreationHandler));

        #endregion

        #region Constructor

        /// <summary>

        /// Private constructor to prevent object creation

        /// </summary>

        private MyLogManager() { }

        #endregion

        #region Type Specific Manager Methods

        /// <summary>

        /// Returns the named logger if it exists

        /// </summary>

        /// <remarks>

        /// <para>If the named logger exists (in the default hierarchy) then it

        /// returns a reference to the logger, otherwise it returns

        /// <c>null</c>.</para>

        /// </remarks>

        /// <param name="name">The fully qualified logger name to look for</param>

        /// <returns>The logger found, or null</returns>

        public static IMyLog Exists(string name)

        {

            return Exists(Assembly.GetCallingAssembly(), name);

        }

        /// <summary>

        /// Returns the named logger if it exists

        /// </summary>

        /// <remarks>

        /// <para>If the named logger exists (in the specified domain) then it

        /// returns a reference to the logger, otherwise it returns

        /// <c>null</c>.</para>

        /// </remarks>

        /// <param name="domain">the domain to lookup in</param>

        /// <param name="name">The fully qualified logger name to look for</param>

        /// <returns>The logger found, or null</returns>

        public static IMyLog Exists(string domain, string name)

        {

            return WrapLogger(LoggerManager.Exists(domain, name));

        }

        /// <summary>

        /// Returns the named logger if it exists

        /// </summary>

        /// <remarks>

        /// <para>If the named logger exists (in the specified assembly's domain) then it

        /// returns a reference to the logger, otherwise it returns

        /// <c>null</c>.</para>

        /// </remarks>

        /// <param name="assembly">the assembly to use to lookup the domain</param>

        /// <param name="name">The fully qualified logger name to look for</param>

        /// <returns>The logger found, or null</returns>

        public static IMyLog Exists(Assembly assembly, string name)

        {

            return WrapLogger(LoggerManager.Exists(assembly, name));

        }

        /// <summary>

        /// Returns all the currently defined loggers in the default domain.

        /// </summary>

        /// <remarks>

        /// <para>The root logger is <b>not</b> included in the returned array.</para>

        /// </remarks>

        /// <returns>All the defined loggers</returns>

        public static IMyLog[] GetCurrentLoggers()

        {

            return GetCurrentLoggers(Assembly.GetCallingAssembly());

        }

        /// <summary>

        /// Returns all the currently defined loggers in the specified domain.

        /// </summary>

        /// <param name="domain">the domain to lookup in</param>

        /// <remarks>

        /// The root logger is <b>not</b> included in the returned array.

        /// </remarks>

        /// <returns>All the defined loggers</returns>

        public static IMyLog[] GetCurrentLoggers(string domain)

        {

            return WrapLoggers(LoggerManager.GetCurrentLoggers(domain));

        }

        /// <summary>

        /// Returns all the currently defined loggers in the specified assembly's domain.

        /// </summary>

        /// <param name="assembly">the assembly to use to lookup the domain</param>

        /// <remarks>

        /// The root logger is <b>not</b> included in the returned array.

        /// </remarks>

        /// <returns>All the defined loggers</returns>

        public static IMyLog[] GetCurrentLoggers(Assembly assembly)

        {

            return WrapLoggers(LoggerManager.GetCurrentLoggers(assembly));

        }

        /// <summary>

        /// Retrieve or create a named logger.

        /// </summary>

        /// <remarks>

        /// <para>Retrieve a logger named as the <paramref name="name"/>

        /// parameter. If the named logger already exists, then the

        /// existing instance will be returned. Otherwise, a new instance is

        /// created.</para>

        ///

        /// <para>By default, loggers do not have a set level but inherit

        /// it from the hierarchy. This is one of the central features of

        /// log4net.</para>

        /// </remarks>

        /// <param name="name">The name of the logger to retrieve.</param>

        /// <returns>the logger with the name specified</returns>

        public static IMyLog GetLogger(string name)

        {

            return GetLogger(Assembly.GetCallingAssembly(), name);

        }

        /// <summary>

        /// Retrieve or create a named logger.

        /// </summary>

        /// <remarks>

        /// <para>Retrieve a logger named as the <paramref name="name"/>

        /// parameter. If the named logger already exists, then the

        /// existing instance will be returned. Otherwise, a new instance is

        /// created.</para>

        ///

        /// <para>By default, loggers do not have a set level but inherit

        /// it from the hierarchy. This is one of the central features of

        /// log4net.</para>

        /// </remarks>

        /// <param name="domain">the domain to lookup in</param>

        /// <param name="name">The name of the logger to retrieve.</param>

        /// <returns>the logger with the name specified</returns>

        public static IMyLog GetLogger(string domain, string name)

        {

            return WrapLogger(LoggerManager.GetLogger(domain, name));

        }

        /// <summary>

        /// Retrieve or create a named logger.

        /// </summary>

        /// <remarks>

        /// <para>Retrieve a logger named as the <paramref name="name"/>

        /// parameter. If the named logger already exists, then the

        /// existing instance will be returned. Otherwise, a new instance is

        /// created.</para>

        ///

        /// <para>By default, loggers do not have a set level but inherit

        /// it from the hierarchy. This is one of the central features of

        /// log4net.</para>

        /// </remarks>

        /// <param name="assembly">the assembly to use to lookup the domain</param>

        /// <param name="name">The name of the logger to retrieve.</param>

        /// <returns>the logger with the name specified</returns>

        public static IMyLog GetLogger(Assembly assembly, string name)

        {

            return WrapLogger(LoggerManager.GetLogger(assembly, name));

        }

        /// <summary>

        /// Shorthand for <see cref="LogManager.GetLogger(string)"/>.

        /// </summary>

        /// <remarks>

        /// Get the logger for the fully qualified name of the type specified.

        /// </remarks>

        /// <param name="type">The full name of <paramref name="type"/> will

        /// be used as the name of the logger to retrieve.</param>

        /// <returns>the logger with the name specified</returns>

        public static IMyLog GetLogger(Type type)

        {

            return GetLogger(Assembly.GetCallingAssembly(), type.FullName);

        }

        /// <summary>

        /// Shorthand for <see cref="LogManager.GetLogger(string)"/>.

        /// </summary>

        /// <remarks>

        /// Get the logger for the fully qualified name of the type specified.

        /// </remarks>

        /// <param name="domain">the domain to lookup in</param>

        /// <param name="type">The full name of <paramref name="type"/> will

        /// be used as the name of the logger to retrieve.</param>

        /// <returns>the logger with the name specified</returns>

        public static IMyLog GetLogger(string domain, Type type)

        {

            return WrapLogger(LoggerManager.GetLogger(domain, type));

        }

        /// <summary>

        /// Shorthand for <see cref="LogManager.GetLogger(string)"/>.

        /// </summary>

        /// <remarks>

        /// Get the logger for the fully qualified name of the type specified.

        /// </remarks>

        /// <param name="assembly">the assembly to use to lookup the domain</param>

        /// <param name="type">The full name of <paramref name="type"/> will

        /// be used as the name of the logger to retrieve.</param>

        /// <returns>the logger with the name specified</returns>

        public static IMyLog GetLogger(Assembly assembly, Type type)

        {

            return WrapLogger(LoggerManager.GetLogger(assembly, type));

        }

        #endregion

        #region Extension Handlers

        /// <summary>

        /// Lookup the wrapper object for the logger specified

        /// </summary>

        /// <param name="logger">the logger to get the wrapper for</param>

        /// <returns>the wrapper for the logger specified</returns>

        private static IMyLog WrapLogger(ILogger logger)

        {

            return (IMyLog)s\_wrapperMap.GetWrapper(logger);

        }

        /// <summary>

        /// Lookup the wrapper objects for the loggers specified

        /// </summary>

        /// <param name="loggers">the loggers to get the wrappers for</param>

        /// <returns>Lookup the wrapper objects for the loggers specified</returns>

        private static IMyLog[] WrapLoggers(ILogger[] loggers)

        {

            IMyLog[] results = new IMyLog[loggers.Length];

            for (int i = 0; i < loggers.Length; i++)

            {

                results[i] = WrapLogger(loggers[i]);

            }

            return results;

        }

        /// <summary>

        /// Method to create the <see cref="ILoggerWrapper"/> objects used by

        /// this manager.

        /// </summary>

        /// <param name="logger">The logger to wrap</param>

        /// <returns>The wrapper for the logger specified</returns>

        private static ILoggerWrapper WrapperCreationHandler(ILogger logger)

        {

            return new MyLogImpl(logger);

        }

        #endregion

    }

}

**8.5 IMyLog类代码**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using log4net;

namespace TGLog.ExpandILog

{

    public interface IMyLog : ILog

    {

        void Debug(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName);

        void Debug(int operatorID, string operand, int actionType,object message,

string ip, string browser, string machineName, Exception t);

        void Info(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName);

        void Info(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName, Exception t);

        void Warn(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName);

        void Warn(int operatorID, string operand, int actionType, object message,

 string ip, string browser, string machineName, Exception t);

        void Error(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName);

        void Error(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName, Exception t);

        void Fatal(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName);

        void Fatal(int operatorID, string operand, int actionType, object message,

string ip, string browser, string machineName, Exception t);

    }

}

**8.6附件**

[使用log4net记录日志](http://files.cnblogs.com/longshizhong/%E4%BD%BF%E7%94%A8log4net%E8%AE%B0%E5%BD%95%E6%97%A5%E5%BF%97.rar)

**8.7参考**

1、<http://peibing211.blog.163.com/blog/static/37116360200992811595469/>

2、<http://www.cnblogs.com/qiangzi/archive/2009/09/10/1541023.html>

3、<http://blog.chinaunix.net/u/23701/showart_1414206.html>

4、<http://itrust.cnblogs.com/archive/2005/01/25/97225.html>

5、<http://www.cnitblog.com/seeyeah/archive/2009/09/20/61491.aspx>

6、<http://www.cnblogs.com/zhmore/archive/2009/03/19/1416707.html>

7、<http://blog.shinylife.net/blog/article.asp?id=948>

8、<http://www.cnblogs.com/manhoo/archive/2009/06/25/1511066.html>