**1. log4net简介**

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。Java平台下，它还有一个姐妹组件——log4j。

log4net的下载地址：<http://logging.apache.org/log4net/download.html>

**2. log4net的组成**

log4net主要由五部分组成，分别为Appenders、Filters、Layouts、Loggers和Object Renders。

**2.1 Appenders**

Appenders用来定义日志的输出方式。它还可以通过配置Filters和Layout来实现日志的过滤和输出格式。

它的输出方式有：

1. 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。
2. AnsiColorTerminalAppender 将日志高亮输出到ANSI终端。
3. AspNetTraceAppender 能用asp.net中Trace的方式查看记录的日志。
4. BufferingForwardingAppender 在输出到子Appenders之前先缓存日志事件。
5. ConsoleAppender 将日志输出到应用程序控制台。
6. EventLogAppender 将日志写到Windows Event Log。
7. FileAppender 将日志输出到文件。
8. ForwardingAppender 发送日志事件到子Appenders。
9. LocalSyslogAppender 将日志写到local syslog service (仅用于UNIX环境下)。
10. MemoryAppender 将日志存到内存缓冲区。
11. 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。
12. 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。
13. RemoteSyslogAppender 通过UDP网络协议将日志写到Remote syslog service。
14. RemotingAppender 通过.NET Remoting将日志写到远程接收端。
15. RollingFileAppender 将日志以回滚文件的形式写到文件中。
16. SmtpAppender 将日志写到邮件中。
17. 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。
18. TelnetAppender 客户端通过Telnet来接受日志事件。
19. TraceAppender 将日志写到.NET trace 系统。
20. 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。

**2.2 Filters**

使用过滤器可以过滤掉Appender输出的内容。过滤器有以下几种：

1. DenyAllFilter 阻止所有的日志事件被记录
2. LevelMatchFilter 只有指定等级的日志事件才被记录
3. LevelRangeFilter 日志等级在指定范围内的事件才被记录
4. LoggerMatchFilter Logger名称匹配，才记录
5. PropertyFilter 消息匹配指定的属性值时才被记录
6. StringMathFilter 消息匹配指定的字符串才被记录

**2.3 Layouts**

Layout用于控制Appender的输出格式，可以使线性的也可以使XML。一个Appender只能有一个Layout。

最常用的Layout应该是用户自定义格式的PatternLayout，其次是SimpleLayout和ExceptionLayout。然后还有4个Layout，其中有两个是输出Xml的Layout，但是中文会有问题。

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

最后说一下PatterLayout的格式化字符串：

|  |  |
| --- | --- |
| **Conversion Pattern Name** | **Effect** |
| a | 等价于**appdomain** |
| appdomain | 引发日志事件的应用程序域的友好名称。（我在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 %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。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"log4net:HostName"总是会有。  **%property**将输出所以的属性 。  （我除了知道可以用它获得主机名外，还不知道怎么用。） |
| 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。后面还可以跟一个日期格式，用大括号括起来。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 这个东西中的属性好像是要用代码来设置(除了默认属性log4net:HostName)。

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。 |

**2.4 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。

下面说一下日志的等级，它们由高到底分别为：

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

其中OFF表示停用所以日志记录，ALL表示所有日志都可以记录。

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。我们以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)，则无论如何日志是不会输出Exception的，因为没有啊。

最后还要说一个LogManager类，它用来管理所应得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");

**2.5 Object Renders**

我对Object Renders的理解是这样的。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。（你可能注意到了，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。这时logger就会知道如何把Orange记录到日志中了。

不过我没有测试过，具体怎么做还是看文档吧。

**2.6 Repository**

Repository主要用于日志对象组织结构的维护。如果你不想自己扩展log4Net的话，可以用不找它。但我还是觉得应该提一下。

**3. 在程序中使用log4net**

在使用log4net前要先做一些配置的工作。配置工作可以在配置文件中完成也可以再程序中用代码完成。我们主要讲在配置文件中如何配置log4net，因为这样更方便灵活，而且还不用重新编译代码。至于如何使用代码进行配置，请查看文档和后面参考中的连接。

**3.1 定义配置文件**

log4net的配置可以放在应用程序的默认配置文件中（app.config或web.config），也可以再你自己的配置文件中。

如果用Visual Studio来编辑配置文件，它对log4net的标签是不会智能提示和自动补全的。当然也不是不可能，看这里：

<http://www.cnblogs.com/didasoft/archive/2007/07/23/log4net_xsd.html>

这里要感谢Jerry同学，为我们提供的xsd schema。

废话少说，先看一个完整的配置文件的例子：

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

<configuration>

<configSections>   
<section name="log4net"   
type="log4net.Config.Log4NetConfigurationSectionHandler, log4net-net-1.2.10" />   
</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]&#13;&#10;"/>   
<param name="Footer" value="[Footer]&#13;&#10;"/>   
<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>

如果log4net的配置不是放在应用程序的配置文件里，而是在自己定义的文件里，<configSection>节点里的<section>节点是不需要的。

下面对其中的标签元素做一下说明。

**3.1.1 <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 |

**3.1.2 <root>**

实际上就是一个根logger，所有其它logger都默认继承它。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。 |

**3.1.3 <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。 |

**3.1.4 <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。

**3.1.5 <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。 |

**3.1.6 <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。 |

**3.1.7 <param>**

<param>元素可以是如何元素的子元素。

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。 |

**3.2 使用配置文件**

**3.2.1 关联配置文件**

log4net默认关联的是应用程序的配置文件(AppName.exe.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(AppName.exe.config)。

举例：

1. [assembly: log4net.config.XmlConfigurator(Watch=true)]
2. //监视默认的配置文件，AppName.exe.config
3. [assembly: log4net.config.XmlConfigurator(ConfigFileExtension="log4net",Watch=true)]
4. //监视配置文件，AppName.exe.log4net
5. [assembly: log4net.config.XmlConfigurator(ConfigFile="log4net.config")]
6. //使用配置文件log4net.config，不监视改变。



**3.2.2 获取日志对象**

log4net.ILog logger = log4net.LogManager.GetLogger("LoggerName");

LogManager还有其他多个方法，比如检查指定日志是否存在，返回所以日志对象等等。

特别的，如果日志名在配置文件中不存在，GetLogger方法会创建一个日志对象，它会继承它的父类的属性。例如"x.y.z"会继承"x.y"的属性，如果没有"x.y"就继承"x"的属性，如果连"x"也没有，会继承root的属性。

**3.2.3 使用日志对象**

获取日志对象后，使用它是很简单的，只要调用对应的Debug, Info等方法就可以了。不过有一件事要说一下，我们以Debug为例。

if (log.isDebugEnabled) {log.Debug(...)}

文档上是这么说的“如果Debug功能不被使用，就不会有参数构造上的开销。但是，另一方面，如果logger的Debug功能被起用，就会有俩倍的开销用于评估logger是否被起用：一次是判断debugEnabled，一次是判断debug是否被启用。但这不是极重的负担，因为评估logger的时间只有整个log语句执行时间的1%。”

也就是说Debug方法会先判断Debug是否启用，再记录日志。使用isDebugEnabled，是否会带来性能提升，是要看情况的。

**3.3 一些配置的例子**

配置文件的结构可以参照3.1节，这里只给出部分元素的使用实例。

**3.3.1 <filter>**

1) 日志等级过滤器

<filter type="log4net.Filter.LevelRangeFilter">  
<param name="levelMin" value="WARN" />  
<param name="levelMax" value="FATAL" />  
</filter>

其中，日志等级WARN,FATAL必须大写。

2) 字符串过滤器

<filter type="log4net.Filter.StringMatchFilter">  
<param name="StringToMatch" value="Warn"/>  
</filter>  
<filter type="log4net.Filter.StringMatchFilter">  
<param name="StringToMatch" value="Error"/>  
<param name="AcceptOnMatch" value="true"/>  
</filter>  
<filter type="log4net.Filter.DenyAllFilter" />

上面的例子，只输出日志信息中包含字符串"Warn"或"Error"的才输出。最后的DenyAllFilter会阻止所有的日志信息。

过滤器还有一个AcceptOnMatch属性，默认为true，表示匹配的时候提交日志事件。设置成false的时候，不同类型的过滤器是不一样的。StringMatchFilter会不提交日志事件，造成匹配的日志信息不被输出。

StringMatchFilter还有一个RegexToMatch属性，用来设置正则表达式。

**3.3.2 <appender>**

1) **AdoNetAppender** 将日志写入到Sql Server数据库

数据库表的Create语句：

CREATE TABLE [dbo].[Log] (  
[Id] [int] IDENTITY (1, 1) NOT NULL,  
[Date] [datetime] NOT NULL,  
[Thread] [varchar] (255) NOT NULL,  
[Level] [varchar] (50) NOT NULL,  
[Logger] [varchar] (255) NOT NULL,  
[Message] [varchar] (4000) NOT NULL,  
[Exception] [varchar] (2000) NULL  
)

配置文件：

<appender name="AdoNetAppender" type="log4net.Appender.AdoNetAppender">  
<param name="bufferSize" value="100" />  
<param name="connectionType" value="System.Data.SqlClient.SqlConnection" />  
<param name="connectionString" value="data source=DatabaseServer;initial catalog=DatabaseName;integrated security=false;persist security info=True;User ID=user;Password=password" />  
<param name="commandText" value="INSERT INTO Log ([Date],[Thread],[Level],[Logger],[Message],[Exception]) VALUES (@log\_date, @thread, @log\_level, @logger, @message, @exception)" />  
<param name="parameter" type="log4net.Appender.AdoNetAppenderParameter">  
<param name="parameterName" value="@log\_date" />  
<param name="dbType" value="DateTime" />  
<param name="layout" type="log4net.Layout.RawTimeStampLayout" />  
</param>  
<param name="parameter" type="log4net.Appender.AdoNetAppenderParameter">  
<param name="parameterName" value="@thread" />  
<param name="dbType" value="String" />  
<param name="size" value="255" />  
<param name="layout" type="log4net.Layout.PatternLayout">  
<param name="conversionPattern" value="%thread" />  
</param>  
</param>  
<parameter>  
<parameterName value="@log\_level" />  
<dbType value="String" />  
<size value="50" />  
<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%level" />  
</layout>  
</parameter>  
<parameter>  
<parameterName value="@logger" />  
<dbType value="String" />  
<size value="255" />  
<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%logger" />  
</layout>  
</parameter>  
<parameter>  
<parameterName value="@message" />  
<dbType value="String" />  
<size value="4000" />  
<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%message" />  
</layout>  
</parameter>  
<parameter>  
<parameterName value="@exception" />  
<dbType value="String" />  
<size value="2000" />  
<layout type="log4net.Layout.ExceptionLayout" />  
</parameter>  
</appender>

我经过测试，使用<param>元素，和<parameter>元素都可以，还可以混合使用。对于<parameter>等元素，log4net的文档上并没有说，但文档的例子中却在用。我个人感觉很多类属性都可以做标签用，并不限于文档中说的那几个。

上面只是Sql Server的配置，其他数据库见文档：<http://logging.apache.org/log4net/release/config-examples.html>

2) **Console** 输出到控制台

<appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender">  
<target value="Console.Error" />  
<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />  
</layout>  
</appender>

target默认是输出到标准输出流的(Console.out)，这里输出到标准错误输出流(Console.Error)。

3) **ColoredConsoleAppender** 高亮输出到控制台

<appender name="ColoredConsole" type="log4net.Appender.ColoredConsoleAppender">  
<mapping>  
<level value="ERROR" />  
<foreColor value="White" />  
<backColor value="Red, HighIntensity" />  
</mapping>  
<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />  
</layout>  
</appender>

上例中，高于等于ERROR的日志都会用指定的前景色和背景色显示。也可以有多个<mapping>

4) **FileAppender** 输出到文件

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

<param name="File" value="${Tmp}/log.txt"  
<param name="AppendToFile" value="false"/>  
<param name="Encoding" value="GB2312" />  
<param name="lockingModel" type="log4net.Appender.FileAppender+MinimalLock" />  
<layout type="log4net.Layout.PatternLayout">  
<param name="Header" type="log4net.Util.PatternString" value="[%date]%newline"/>  
<param name="Footer" value="--------------------------------------------------&#13;&#10;"/>  
<param name="ConversionPattern" value="logger: %.30logger Message:%message %newline"/>  
</layout>  
</appender>

上面是一个复杂的FileAppender，其实除了File属性，其他的属性都是可选的。这个属性值可以使用系统的环境变量，上面就用到了系统的临时文件夹${Tmp}。也可以像layout的Header属性那样使用转义字符串，像这样：

<param name="file" type="log4net.Util.PatternString" value="%date{HHmmss} - log.txt"/>

注意文件名要符合操作系统的命名规范。

layout的Footer属性，因为没有将type设置为PatternString，所以不能使用转义字符串。所以要想换行就只有用xml实体了，&#13;&#10;表示换行回车/n/r。layout的Header和Footer在很多Appender中是没有用的，比如ConsoleAppender和AdoAppender。

FileAppender的AppendToFile表示日志写入文件的方式是追加还是覆盖，默认是true，追加。Encoding用来设置文件编码，不知道问什么我测试的时候，好像不起作用。lockingModel没搞懂，大概是多进程操做同一个日志文件的时候用的吧。

5) **RollingFileAppender** 输出到可滚动的文件

RollingFileAppender继承自FileAppender，FileAppender的属性，它都可以用。一个简单的例子：

<appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">  
<file value="log.txt" />  
<appendToFile value="true" />  
<rollingStyle value="Size" />  
<maxSizeRollBackups value="10" />  
<maximumFileSize value="100KB" />  
<staticLogFileName value="true" />  
<layout type="log4net.Layout.PatternLayout">  
<conversionPattern value="%date [%thread] %-5level %logger [%property{NDC}] - %message%newline" />  
</layout>  
</appender>

上例中，如果log.txt的文件大小超过100KB，就会把log.txt做备份，备份文件名为log.txt.1,log.txt.2……。但是备份文件的最大数是10个。

**说明：**

a. 本文中使用的log4net版本是1.2.10。

**参考：**

<http://logging.apache.org/log4net/>

<http://log4j.jaxmao.org/log4j/docs/manual.html>

<http://www.cnblogs.com/liwei6797/archive/2007/04/27/729679.html>

<http://www.cnblogs.com/dragon/archive/2005/03/24/124254.aspx>

<http://heroman.cnblogs.com/archive/2006/01/09/314053.html>

<http://www.cnblogs.com/didasoft/archive/2007/07/23/log4net_xsd.html>