Log4Net

**使用手册**

www.bzMe.work

参考：

<http://svn.apache.org/repos/asf/logging/site/trunk/docs/log4net/download.html>

<https://github.com/apache/logging-log4net/>

<https://sourceforge.net/projects/log4net/files/>

**第一章 概述**

翻译自官网：<http://logging.apache.org/log4net/>

**一、log4nnet简介**

log4net是一个开源项目，官方对其描述如下：

”The Apache log4net library is a tool to help the programmer output log statements to a variety of output targets. log4net is a port of the excellent Apache log4j™ framework to the Microsoft® .NET runtime. We have kept the framework similar in spirit to the original log4j while taking advantage of new features in the .NET runtime. For more information on log4net see the [features](http://logging.apache.org/log4net/release/features.html) document.”

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

log4net在github上有一个映射，显示时间是2017年最后一次更新，这显示log4net基本上已经停止更新，说明其已经稳定成熟，暂时已经没有升级的需要，或者这个项目正式停止更新。

参见：<https://github.com/apache/logging-log4net/>

log4net的架构如下：



**二、log4net特性**

参考：<http://logging.apache.org/log4net/release/features.html>

log4net是一个帮助程序员将日志语句输出到各种输出目标的工具。在应用程序出现问题时，启用日志功能有助于定位问题。使用log4net，可以在运行时启用日志记录，而无需修改应用程序的二进制文件。log4net包的设计使日志语句可以保留在附带的代码中，而不会带来很高的性能成本。因此，日志记录(或者不记录日志)的速度是至关重要的。log4net is a tool to help the programmer output log statements to a variety of output targets. In case of problems with an application, it is helpful to enable logging so that the problem can be located. With log4net it is possible to enable logging at runtime without modifying the application binary. The log4net package is designed so that log statements can remain in shipped code without incurring a high performance cost. It follows that the speed of logging (or rather not logging) is crucial.

与此同时，日志输出可能会非常多，很快就会变得非常多。log4net的一个独特特性是分级日志记录器(hierarchical loggers)的概念。使用这些日志记录器可以有选择地控制以任意粒度输出哪些日志语句。At the same time, log output can be so voluminous that it quickly becomes overwhelming. One of the distinctive features of log4net is the notion of hierarchical loggers. Using these loggers it is possible to selectively control which log statements are output at arbitrary granularity.

log4net在设计时考虑到了两个不同的目标: 速度和灵活性。log4net is designed with two distinct goals in mind: speed and flexibility

**log4net提供的特性有：**

支持多种框架(Support for multiple frameworks)；

输出到多个日志目标(Output to multiple logging targets)；

分层日志架构(Hierarchical logging architecture)；

XML配置(XML Configuration)；

动态配置(Dynamic Configuration)；

日志上下文(Logging Context)；

经过证实的架构(Proven architecture)(翻译成经过实践检验的架构或许更好)；

模块化和可扩展设计(Modular and extensible design)；

具有灵活性的高性能(High performance with flexibility)。

**1.支持多种框架(Support for multiple frameworks)**

log4net在所有兼容ECMA CLI 1.0的运行时上运行。log4net为以下框架提供了特定的构建:

log4net runs on all ECMA CLI 1.0 compatible runtimes. log4net has specific builds for the following frameworks:

* .NET Core 1.0 - using netstandard-1.3
* Microsoft® .NET Framework 1.0[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)
* Microsoft .NET Framework 1.1[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)
* Microsoft .NET Framework 2.0
* Microsoft .NET Framework 3.5
* Microsoft .NET Framework 4.0
* Microsoft .NET Framework 4.5
* Microsoft .NET Framework 3.5 Client Profile
* Microsoft .NET Framework 4.0 Client Profile
* Microsoft .NET Compact Framework 1.0[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)
* Microsoft .NET Compact Framework 2.0[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)
* Mono 1.0[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)
* Mono 2.0
* Mono 3.5
* Mono 4.0
* Microsoft Shared Source CLI 1.0[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)
* CLI 1.0 Compatible[\*](http://logging.apache.org/log4net/release/features.html#no-bin-support)

说明：\*表示不支持二进制版本，但可以从源代码构建。

**2.输出到多个日志目标(Output to multiple logging targets)**

log4net内置了以下的输出器(appenders)(不是所有框架都支持):

|  |  |
| --- | --- |
| **Type** | **Description** |
| log4net.Appender.AdoNetAppender\* | 使用准备好的语句或存储过程将日志事件写入数据库。Writes logging events to a database using either prepared statements or stored procedures. |
| log4net.Appender.AnsiColorTerminalAppender | 将彩色突出显示的日志事件写入ANSI终端窗口。Writes color highlighted logging events to a an ANSI terminal window. |
| log4net.Appender.AspNetTraceAppender\* | 将日志事件写入ASP跟踪上下文。这些可以在ASP页面的末尾或在ASP跟踪页面上呈现。Writes logging events to the ASP trace context. These can then be rendered at the end of the ASP page or on the ASP trace page. |
| log4net.Appender.ColoredConsoleAppender\* | 将用颜色突出显示的日志事件写入应用程序的Windows控制台。Writes color highlighted logging events to the application's Windows Console. |
| log4net.Appender.ConsoleAppender | 将日志事件写入应用程序的控制台。这些事件可能会进入我们的标准流或标准错误流。Writes logging events to the application's Console. The events may go to either the standard our stream or the standard error stream. |
| log4net.Appender.DebugAppender | 将日志事件写入.NET系统。Writes logging events to the .NET system. |
| log4net.Appender.EventLogAppender\* | 将日志事件写入Windows事件日志。Writes logging events to the Windows Event Log. |
| log4net.Appender.FileAppender | 将日志事件写入文件系统中的文件。Writes logging events to a file in the file system. |
| log4net.Appender.LocalSyslogAppender | 将日志事件写入本地syslog服务(仅适用于UNIX)。Writes logging events to the local syslog service (UNIX only). |
| log4net.Appender.MemoryAppender | 将日志事件存储在内存缓冲区中。Stores logging events in an in memory buffer. |
| log4net.Appender.NetSendAppender\* | 将日志事件写入Windows Messenger服务。这些消息显示在用户终端的对话框中。Writes logging events to the Windows Messenger service. These messages are displayed in a dialog on a users terminal. |
| log4net.Appender.OutputDebugStringAppender | 将日志事件写入调试器。如果应用程序没有调试器，系统调试器将显示字符串。如果应用程序没有调试器，并且系统调试器不活动，则忽略该消息。Writes logging events to the debugger. If the application has no debugger, the system debugger displays the string. If the application has no debugger and the system debugger is not active, the message is ignored. |
| log4net.Appender.RemoteSyslogAppender | 使用UDP网络将日志事件写入远程syslog服务。Writes logging events to a remote syslog service using UDP networking. |
| log4net.Appender.RemotingAppender | 使用.NET remoting将日志事件写入remoting接收器。Writes logging events to a remoting sink using .NET remoting. |
| log4net.Appender.RollingFileAppender | 将日志事件写入文件系统中的文件。可以将RollingFileAppender配置为根据日期或文件大小约束登录多个文件。Writes logging events to a file in the file system. The RollingFileAppender can be configured to log to multiple files based upon date or file size constraints. |
| log4net.Appender.SmtpAppender\* | 将日志事件发送到电子邮件地址。Sends logging events to an email address. |
| log4net.Appender.SmtpPickupDirAppender | 将日志事件发送到电子邮件地址，但将电子邮件写入可配置目录，而不是通过SMTP直接发送。  Sends logging events to an email address but writes the emails to a configurable directory rather than sending them directly via SMTP. |
| log4net.Appender.TelnetAppender | 客户端通过Telnet连接以接收日志事件。  Clients connect via Telnet to receive logging events. |
| log4net.Appender.TraceAppender | 将日志事件写入.NET跟踪系统。  Writes logging events to the .NET trace system. |
| log4net.Appender.UdpAppender | 使用UdpClient将日志事件作为无连接UDP数据报发送到远程主机或多播组。  Sends logging events as connectionless UDP datagrams to a remote host or a multicast group using a UdpClient. |

说明：\*表示不支持.net core。

参见：<http://logging.apache.org/log4net/release/framework-support.html>

**3.分层日志架构(Hierarchical logging architecture)**

分层日志是基于组件(component)开发的设计思想。Hierarchical logging is an ideal fit with component based development.

每个组件都有自己的日志记录器(logger)。单独测试时，可以根据开发人员的需要设置这些记录器的属性。当与其他组件组合时，日志记录器(loggers)将继承组件累积器(integrator)确定的属性。

可以有选择地提高一个组件的日志记录优先级，而不影响其他组件。当您需要来自单个组件的详细跟踪，而不需要将来自其他组件的消息拥挤在跟踪文件中时，这是非常有用的。Each component has its own of logger. When individually tested, the properties of these loggers may be set as the developer requires. When combined with other components, the loggers inherit the properties determined by the integrator of the components. One can selectively elevate logging priorities on one component without affecting the other components. This is useful when you need a detailed trace from just a single component without crowding the trace file with messages from other components.

所有这些都可以通过配置文件完成;不需要更改代码。All this can be done through configuration files; no code changes are required.

**4.XML配置(XML Configuration)**

log4net是使用XML配置文件配置的。配置信息可以嵌入到其他XML配置文件(例如应用程序的.config文件)或单独的文件中。该配置易于阅读和更新，同时保留了表示所有配置的灵活性。另外，log4net也可以通过编程方式配置。log4net is configured using an XML configuration file. The configuration information can be embedded within other XML configuration files (such as the application's .config file) or in a separate file. The configuration is easily readable and updateable while retaining the flexibility to express all configurations. Alternatively log4net can be configured programmatically.

**5.动态配置(Dynamic Configuration)**

log4net可以监视配置文件的更改，并动态地应用配置器所做的更改。日志级别、附加程序、布局以及几乎所有其他内容都可以在运行时进行调整。在许多情况下，可以诊断应用程序问题而不终止相关的流程。在研究部署应用程序的问题时，这可能是一个非常有价值的工具。log4net can monitor its configuration file for changes and dynamically apply changes made by the configurator. The logging levels, appenders, layouts, and just about everything else can be adjusted at runtime. In many cases it is possible to diagnose application issues without terminating the process in question. This can a very valuable tool in investigating issues with deployed applications.

**6.日志上下文(Logging Context)**

可以使用log4net以一种在日志记录时对开发人员透明的方式收集日志上下文数据。GlobalContext和ThreadContext允许应用程序存储附加到日志消息的上下文数据。例如，在web服务中，一旦调用者通过身份验证，调用者的用户名就可以存储在ThreadContext属性中。然后，此属性将作为来自同一线程的每个后续日志消息的一部分自动记录。log4net can be used to collect logging context data in a way that is transparent to the developer at the point of logging. The GlobalContext and the ThreadContext allow the application to store contextual data that is attached to logging messages. For instance, in a web service, once the caller is authenticated the username of the caller could be stored in a ThreadContext property. This property would then be automatically logged as part of each subsequent logging message made from the same thread.

**7.经过证实的架构(Proven architecture)**

log4net基于非常成功的Apache log4j™日志库，该库从1996年开始开发。到目前为止，这种流行的、经过验证的体系结构已被移植到12种语言中。log4net is based on the highly successful Apache log4j™ logging library, in development since 1996. This popular and proven architecture has so far been ported to 12 languages.

**二、log4net设计概览**

本文基于Ceki Gulcu对log4j的简短介绍。This document is based on *Short introduction to log4j* by *Ceki Gülcü*.

log4net框架基于Apache log4j™，有关log4j的更多信息，请参见<http://logging.apache.org/log4j/> 。log4net框架(framework)、源代码(source code)、二进制文件(binaries)、文档(documentation)、示例(examples)和相关材料(related materials)都是在[Apache License, Version 2.0](http://www.apache.org/licenses/LICENSE-2.0)的条款下发布的，该版本的副本已经包含在LICENSE.txt文件中。The log4net framework is based on Apache log4j™, see <http://logging.apache.org/log4j/> for more information on log4j. The log4net framework, source code, binaries, documentation, examples and related materials are published under the terms of the [Apache License, Version 2.0](http://www.apache.org/licenses/LICENSE-2.0), a copy of which has been included with this distribution in the LICENSE.txt file.

本文档介绍了log4net API、它的独特特性和设计原理。log4net是一个基于许多作者工作的开源项目。它允许开发人员控制以任意粒度输出的日志语句。它可以在运行时使用外部配置文件进行完全配置。This document is an introduction to the log4net API, its unique features and design rationale. Log4net is an open source project based on the work of many authors. It allows the developer to control which log statements are output with arbitrary granularity. It is fully configurable at runtime using external configuration files.

几乎每个大型应用程序都包含自己的日志记录或跟踪API。将日志语句插入到代码中是一种低技术的调试方法。这也是唯一的方法，因为调试器(debuggers)并不总是可用(available)或适用的(applicable)。这通常是多线程应用程序(multithreaded applications)和分布式应用程序(distributed applications)面临的情况。Almost every large application includes its own logging or tracing API. Inserting log statements into code is a low-tech method for debugging it. It may also be the only way because debuggers are not always available or applicable. This is usually the case for multithreaded applications and distributed applications at large.

一旦部署了应用程序，就不可能利用开发和调试工具。管理员可以使用有效的日志系统来诊断和修复许多配置问题。Once an application has been deployed it may not be possible to utilize development and debugging tools. An administrator can use effective logging systems to diagnose and fix many configuration issues.

经验表明，日志记录是开发周期的重要组成部分。它有几个优点。它提供了关于应用程序执行的精确上下文。一旦插入到代码中，日志输出的生成就不需要人工干预。此外，日志输出可以保存在持久介质中，以供以后研究。除了在开发周期中使用它之外，一个足够丰富的日志记录包也可以看作是一个审计工具。Experience indicates that logging is an important component of the development cycle. It offers several advantages. It provides precise *context* about the execution of the application. Once inserted into the code, the generation of logging output requires no human intervention. Moreover, log output can be saved in persistent medium to be studied at a later time. In addition to its use in the development cycle, a sufficiently rich logging package can also be viewed as an auditing tool.

日志确实有其缺点。它会降低应用程序的速度。如果过于冗长，可能会导致屏幕闪动(scrolling blindness)。为了缓解这些问题，log4net被设计成可靠、快速和可扩展的。由于日志记录很少是应用程序的主要关注点，所以log4net API力求简单易懂和易于使用。Logging does have its drawbacks. It can slow down an application. If too verbose, it can cause scrolling blindness. To alleviate these concerns, log4net is designed to be reliable, fast and extensible. Since logging is rarely the main focus of an application, the log4net API strives to be simple to understand and to use.

**1.关于日志器和输出器(Loggers and Appenders)**

log4net有三个主要组件:日志器(loggers)、输出器(appenders)和布局(layouts)。这三种类型的组件协同工作，使开发人员能够根据消息类型和级别记录消息，并在运行时控制这些消息的格式和报告位置。这些组件通过控制将对象转换为字符串的appender和对象呈现器的操作的过滤器得到帮助。Log4net has three main components: *loggers*, *appenders* and *layouts*. These three types of components work together to enable developers to log messages according to message type and level, and to control at runtime how these messages are formatted and where they are reported. These components are helped by *filters* that control the actions of the appender and *object renderers* that turn objects into strings.

这些组件通过“将对象(objects)转换成字符串(strings)的输出器(appender)和对象渲染器(*object renderers*)的控制(control)操作(actions)”得到帮助。These components are helped by *filters* that control the actions of the appender and *object renderers* that turn objects into strings.

**2.关于日志器层级(Logger hierarchy)**

与System.Console.WriteLine相比，任何日志API的首要优势在于它能够禁用某些日志语句(log statements)，同时允许其他语句不受阻碍地打印。该功能假设日志空间(即所有可能的日志语句的空间)是根据一些开发人员选择的标准进行分类的。

日志记录器(Loggers)是命名的实体。日志记录器的名称是大小写敏感的，它们遵循以下层次命名规则(hierarchical naming rule): The first and foremost advantage of any logging API over plain System.Console.WriteLine resides in its ability to disable certain log statements while allowing others to print unhindered. This capability assumes that the logging space, that is, the space of all possible logging statements, is categorized according to some developer-chosen criteria. Loggers are named entities. Logger names are case-sensitive and they follow the following hierarchical naming rule:

**（1）日志名称层次规则(Named Hierarchy)**

如果一个日记录器的名称后跟一个点是其子记录器名称(*descendant* logger name)的前缀，则该记录器称为另一个记录器的祖先。如果记录器本身和子记录器之间没有祖先，则记录器称为子记录器的父记录器。A logger is said to be an *ancestor* of another logger if its name followed by a dot is a prefix of the *descendant* logger name. A logger is said to be a *parent* of a *child* logger if there are no ancestors between itself and the descendant logger.

层次结构的工作方式与.NET中的命名空间和类层次结构非常相似。我们很快就会知道这很方便。The hierarchy works very much in the same way as the namespace and class hierarchy in .NET. This is very convenient as we shall soon see.

例如，名为“Foo.Bar”的日志记录器是名为“Foo.Bar.Baz”的记录器的父级。同样，“System”是“System.Text”的父级和“System.Text.StringBuilder”的父级。大多数开发人员都应该熟悉这个命名方案。For example, the logger named "Foo.Bar" is a parent of the logger named "Foo.Bar.Baz". Similarly, "System" is a parent of "System.Text" and an ancestor of "System.Text.StringBuilder". This naming scheme should be familiar to most developers.

根日志记录器位于记录器层次结构的顶部。它在三个方面很特别: The root logger resides at the top of the logger hierarchy. It is exceptional in three ways:

1.它总是存在。It always exists

2.无法通过名称检索它。It cannot be retrieved by name

3.它总是有一个指定的级别。It always has an assigned level

log4net使用log4net.LogManager类中的静态方法检索日志记录器。GetLogger方法将所需记录器的名称作为参数。它们如下所示：Loggers are retrieved using the static method from the log4net.LogManager class. The GetLogger methods take the name of the desired logger as a parameter. They are listed below:

|  |
| --- |
| namespace log4net  {  public class LogManager  {  public static ILog GetLogger(string name);  public static ILog GetLogger(Type type);  }  } |

接受类型参数的GetLogger方法使用完全限定类型名作为要检索的日志记录器的名称。这些GetLogger方法返回一个ILog接口。这是传递回开发人员使用的记录器。ILog接口定义如下: The GetLogger methods that takes a Type parameter uses the fully qualified type name as the name of the logger to retrieve. These GetLogger methods return an ILog interface. That is the representation of the *Logger* passed back to the developer. The ILog interface is defined below:

|  |
| --- |
| namespace log4net  {  public interface ILog  {  /\* Test if a level is enabled for logging \*/  bool IsDebugEnabled { get; }  bool IsInfoEnabled { get; }  bool IsWarnEnabled { get; }  bool IsErrorEnabled { get; }  bool IsFatalEnabled { get; }  /\* Log a message object \*/  void Debug(object message);  void Info(object message);  void Warn(object message);  void Error(object message);  void Fatal(object message);  /\* Log a message object and exception \*/  void Debug(object message, Exception t);  void Info(object message, Exception t);  void Warn(object message, Exception t);  void Error(object message, Exception t);  void Fatal(object message, Exception t);  /\* Log a message string using the System.String.Format syntax \*/  void DebugFormat(string format, params object[] args);  void InfoFormat(string format, params object[] args);  void WarnFormat(string format, params object[] args);  void ErrorFormat(string format, params object[] args);  void FatalFormat(string format, params object[] args);  /\* Log a message string using the System.String.Format syntax \*/  void DebugFormat(IFormatProvider provider, string format, params object[] args);  void InfoFormat(IFormatProvider provider, string format, params object[] args);  void WarnFormat(IFormatProvider provider, string format, params object[] args);  void ErrorFormat(IFormatProvider provider, string format, params object[] args);  void FatalFormat(IFormatProvider provider, string format, params object[] args);  }  } |

日志记录器可能被分配到不同的级别。级别(Levels)是log4net.Core.Level类的的实例。为了提高优先级，定义了以下级别： Loggers *may* be assigned levels. Levels are instances of the log4net.Core.Level class. The following levels are defined in order of increasing priority:

ALL

DEBUG

INFO

WARN

ERROR

FATAL

OFF

如果一个给定的日志记录器没有被分配一个级别，那么它将从最近的祖先那里继承一个级别并分配一个级别。更正式地: If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. More formally:

**（2）级别继承(Level Inheritance)**

给定日志记录器X的继承级别等于日志记录器层次结构中的第一个非空级别，从X开始，在层次结构中向上一直到根记录器。The *inherited level* for a given logger *X*, is equal to the first non-null level in the logger hierarchy, starting at *X* and proceeding upwards in the hierarchy towards the *root* logger.

为了确保所有日志记录器最终都能继承一个级别，根日志记录器总是有一个指定的级别。根记录器的默认值是DEBUG。To ensure that all loggers can eventually inherit a level, the *root* logger always has an assigned level. The default value for the *root* logger is DEBUG.

下面是四个表，根据上面的规则，它们具有各种指定的级别值和结果继承的级别。Below are four tables with various assigned level values and the resulting inherited levels according to the above rule.

|  |  |  |
| --- | --- | --- |
| **日志器名称**  **Logger name** | **指派级别**  **Assigned level** | **继承级别**  **Inherited level** |
| *root* | Proot | Proot |
| X | none | Proot |
| X.Y | none | Proot |
| X.Y.Z | none | Proot |

在上面的示例1中，只有根记录器被分配了一个级别。此级别值Proot由其他记录器X、Y和X.Y.Z继承.In *Example 1* above, only the *root* logger is assigned a level. This level value, Proot, is inherited by the other loggers X, X.Y and X.Y.Z.

|  |  |  |
| --- | --- | --- |
| **Logger name** | **Assigned level** | **Inherited level** |
| *root* | Proot | Proot |
| X | Px | Px |
| X.Y | Pxy | Pxy |
| X.Y.Z | Pxyz | Pxyz |

在上面的示例2中，所有日志记录器都有一个指定的级别值。不需要级别继承。In *Example 2* above, all loggers have an assigned level value. There is no need for level inheritance.

|  |  |  |
| --- | --- | --- |
| **Logger name** | **Assigned level** | **Inherited level** |
| *root* | Proot | Proot |
| X | Px | Px |
| X.Y | none | Px |
| X.Y.Z | Pxyz | Pxyz |

在上面的示例3中，日志记录器的根是X和X.Y.Z分别被赋值为Proot、Px和Pxyz三个级别。日志记录器X.Y继承其父X的级别值。In *Example 3* above, the loggers *root*, X and X.Y.Z are assigned the levels Proot, Px and Pxyz respectively. The logger X.Y inherits its level value from its parent X.

|  |  |  |
| --- | --- | --- |
| **Logger name** | **Assigned level** | **Inherited level** |
| *root* | Proot | Proot |
| X | Px | Px |
| X.Y | none | Px |
| X.Y.Z | none | Px |

在上面的示例4中，日志器根和X分别被赋值为Proot和Px。日志器X.Y和X.Y.Z从最近的具有指定级别的父X继承它们的级别值。In *Example 4* above, the loggers *root* and X and are assigned the levels Proot and Px respectively. The loggers X.Y and X.Y.Z inherits their level value from their nearest parent X having an assigned level.

日志记录请求是通过调用logger实例的一个打印方法(printing methods)（通过log4net.ILog）发出的。这些打印方法有Debug、Info、Warn、Error和Fatal。Logging requests are made by invoking one of the printing methods of a logger instance (through the log4net.ILog). These printing methods are Debug, Info, Warn, Error, and Fatal.

根据定义，打印方法确定日志记录请求的级别。例如，如果log是一个logger实例，那么语句log.Info（“..”）是一个级别Info的日志请求。By definition, the printing method determines the level of a logging request. For example, if log is a logger instance, then the statement log.Info("..") is a logging request of level INFO.

如果日志记录请求(logging request)的级别高于或等于其日志记录器的级别，则称为启用日志记录请求。否则，请求将被禁用。没有指定级别的记录器将从层次结构(hierarchy)继承一个级别。这一规则概述如下。A logging request is said to be *enabled* if its level is higher than or equal to the level of its logger. Otherwise, the request is said to be *disabled*. A logger without an assigned level will inherit one from the hierarchy. This rule is summarized below.

**（3）基本的选择规则(Basic Selection Rule)**

如果L >= K，则启用日志记录器中级别L的日志请求(分配或继承级别K，无论哪种级别合适)。A log request of level *L* in a logger with (either assigned or inherited, whichever is appropriate) level *K*, is enabled if *L >= K*.

这条规则是log4net的核心。它假设级别是有序的。对于标准级别，我们有DEBUG < INFO < WARN < ERROR < FATAL。This rule is at the heart of log4net. It assumes that levels are ordered. For the standard levels, we have DEBUG < INFO < WARN < ERROR < FATAL.

使用相同的名称调用log4net.LogManager.GetLogger方法将始终返回对相同记录器对象的引用。Calling the log4net.LogManager.GetLogger method with the same name will always return a reference to the exact same logger object.

例如,在:

|  |
| --- |
| ILog x = LogManager.GetLogger("wombat");  ILog y = LogManager.GetLogger("wombat"); |

x和y指的是完全相同的logger对象。x and y refer to exactly the same logger object.

因此，可以配置一个日志程序，然后在代码的其他地方检索相同的实例，而不需要传递引用。与生物学上的父母关系(父母总是先于子女)相反，log4net日志记录器可以按任何顺序创建和配置。特别是，“父”日志记录器将查找并链接到它的后代，即使它是在它们之后实例化的。Thus, it is possible to configure a logger and then to retrieve the same instance somewhere else in the code without passing around references. In fundamental contradiction to biological parenthood, where parents always precede their children, log4net loggers can be created and configured in any order. In particular, a "parent" logger will find and link to its descendants even if it is instantiated after them.

log4net环境的配置通常在应用程序初始化时完成。首选的方法是读取配置文件。稍后将讨论这种方法。Configuration of the log4net environment is typically done at application initialization. The preferred way is by reading a configuration file. This approach will be discussed shortly.

Log4net使得通过软件组件命名日志记录器变得很容易。这可以通过在每个类中静态实例化一个日志程序来实现，日志程序的名称等于类的完全限定名。这是定义记录器的一种有用且简单的方法。由于日志输出使用生成日志记录器的名称，因此这种命名策略使确定日志消息的来源变得很容易。然而，这只是命名日志记录器的一种可能的(尽管很常见)策略。log4net不限制日志记录器的可能集合。开发人员可以根据需要自由地命名日志记录器。Log4net makes it easy to name loggers by *software component*. This can be accomplished by statically instantiating a logger in each class, with the logger name equal to the fully qualified name of the class. This is a useful and straightforward method of defining loggers. As the log output bears the name of the generating logger, this naming strategy makes it easy to identify the origin of a log message. However, this is only one possible, albeit common, strategy for naming loggers. Log4net does not restrict the possible set of loggers. The developer is free to name the loggers as desired.

然而，以它们所在的类来命名日志记录器似乎是迄今为止所知的最佳策略。对于开发人员来说，每个日志消息的来源很简单。最重要的是，它利用应用程序的设计来生成日志记录器层次结构的设计。希望在应用程序的设计中加入了一些思想。Nevertheless, naming loggers after the class where they are located seems to be the best strategy known so far. It is simple an obvious to the developers where each log message came from. Most importantly it leverages the design of the application to produce the design of the logger hierarchy. Hopefully some thought has gone into the design of the application.

**3.输出器(Appenders)**

输出器(Appenders)是一种记录方式，即你可以以文件、邮件等各种媒介来记录日志。

根据日志记录器有选择地启用或禁用日志记录请求的功能只是其中的一部分。log4net允许将日志请求打印输出到多个目的地。在log4net中，输出目的地称为输出器(*appender*)。输出器必须实现log4net.Appenders.IAppender接口。The ability to selectively enable or disable logging requests based on their logger is only part of the picture. Log4net allows logging requests to print to multiple destinations. In log4net speak, an output destination is called an *appender*. Appenders must implement the log4net.Appenders.IAppender interface.

log4net中定义了以下输出器:

|  |  |
| --- | --- |
| **Type** | **Description** |
| [log4net.Appender.AdoNetAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_AdoNetAppender.htm) | 使用准备好的语句或存储过程将日志事件写入数据库。Writes logging events to a database using either prepared statements or stored procedures. |
| [log4net.Appender.AnsiColorTerminalAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_AnsiColorTerminalAppender.htm) | 将彩色突出显示的日志事件写入ANSI终端窗口。Writes color highlighted logging events to a an ANSI terminal window. |
| [log4net.Appender.AspNetTraceAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_AspNetTraceAppender.htm) | 将日志事件写入ASP跟踪上下文。这些可以在ASP页面的末尾或在ASP跟踪页面上呈现。Writes logging events to the ASP trace context. These can then be rendered at the end of the ASP page or on the ASP trace page. |
| [log4net.Appender.BufferingForwardingAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_BufferingForwardingAppender.htm) | 在将事件转发给子输出器之前缓冲日志记录。Buffers logging events before forwarding them to child appenders. |
| [log4net.Appender.ColoredConsoleAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_ColoredConsoleAppender.htm) | 将日志事件写入应用程序的控制台。这些事件可能会进入我们的标准流或标准错误流。事件可以为每个级别定义可配置的文本和背景颜色。Writes logging events to the application's Console. The events may go to either the standard our stream or the standard error stream. The events may have configurable text and background colors defined for each level. |
| [log4net.Appender.ConsoleAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_ConsoleAppender.htm) | 将日志事件写入应用程序的控制台。这些事件可能会进入我们的标准流或标准错误流。  Writes logging events to the application's Console. The events may go to either the standard our stream or the standard error stream. |
| [log4net.Appender.DebugAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_DebugAppender.htm) | 将日志事件写入.NET系统。Writes logging events to the .NET system. |
| [log4net.Appender.EventLogAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_EventLogAppender.htm) | 将日志事件写入Windows事件日志。Writes logging events to the Windows Event Log. |
| [log4net.Appender.FileAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net.Appender_FileAppender.htm) | 将日志事件写入文件系统中的文件。Writes logging events to a file in the file system. |
| [log4net.Appender.ForwardingAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_ForwardingAppender.htm) | 将日志事件转发给子输出器。Forwards logging events to child appenders. |
| [log4net.Appender.LocalSyslogAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_LocalSyslogAppender.htm) | 将日志事件写入本地syslog服务(仅适用于UNIX)。Writes logging events to the local syslog service (UNIX only). |
| [log4net.Appender.MemoryAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_MemoryAppender.htm) | 将日志事件存储在内存缓冲区中。Stores logging events in an in memory buffer. |
| [log4net.Appender.NetSendAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_NetSendAppender.htm) | 将日志事件写入Windows Messenger服务。这些消息显示在用户终端的对话框中。  Writes logging events to the Windows Messenger service. These messages are displayed in a dialog on a users terminal. |
| [log4net.Appender.OutputDebugStringAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_OutputDebugStringAppender.htm) | 将日志事件写入调试器。如果应用程序没有调试器，系统调试器将显示字符串。如果应用程序没有调试器，并且系统调试器不活动，则忽略该消息。Writes logging events to the debugger. If the application has no debugger, the system debugger displays the string. If the application has no debugger and the system debugger is not active, the message is ignored. |
| [log4net.Appender.RemoteSyslogAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_RemoteSyslogAppender.htm) | 使用UDP网络将日志事件写入远程syslog服务。Writes logging events to a remote syslog service using UDP networking. |
| [log4net.Appender.RemotingAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_RemotingAppender.htm) | 使用.NET remoting将日志事件写入remoting接收器。Writes logging events to a remoting sink using .NET remoting. |
| [log4net.Appender.RollingFileAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_RollingFileAppender.htm) | 将日志事件写入文件系统中的文件。可以将RollingFileAppender配置为根据日期或文件大小约束登录多个文件。Writes logging events to a file in the file system. The RollingFileAppender can be configured to log to multiple files based upon date or file size constraints. |
| [log4net.Appender.SmtpAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_SmtpAppender.htm) | 将日志事件发送到电子邮件地址。Sends logging events to an email address. |
| [log4net.Appender.SmtpPickupDirAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_SmtpPickupDirAppender.htm) | 将SMTP消息作为文件写入拾取目录。然后可以通过诸如IIS SMTP代理之类的SMTP代理读取和发送这些文件。Writes SMTP messages as files into a pickup directory. These files can then be read and sent by an SMTP agent such as the IIS SMTP agent. |
| [log4net.Appender.TelnetAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_TelnetAppender.htm) | 客户端通过Telnet连接以接收日志事件。  Clients connect via Telnet to receive logging events. |
| [log4net.Appender.TraceAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_TraceAppender.htm) | 将日志事件写入.NET跟踪系统。Writes logging events to the .NET trace system. |
| [log4net.Appender.UdpAppender](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Appender_UdpAppender.htm) | 使用UdpClient将日志事件作为无连接UDP数据报发送到远程主机或多播组。Sends logging events as connectionless UDP datagrams to a remote host or a multicast group using a UdpClient. |

可以将多个输出器附加到日志记录器。More than one appender can be attached to a logger.

给定日志记录器的每个启用的日志记录请求将被转发给该日志记录器中的所有输出器以及层次结构中更高的输出器。换句话说，输出器额外地继承自记录器层次结构。例如，如果将控制台输出器添加到根日志记录器，那么所有启用的日志请求至少都将在控制台打印。如果将文件输出器添加到日志记录器(例如X)中，则X和X的子节点的启用日志记录请求将打印在文件和控制台上。通过将日志记录器上的additivity标记设置为false，可以覆盖此默认行为，使appender累积不再是可添加的。**Each enabled logging request for a given logger will be forwarded to all the appenders in that logger as well as the appenders higher in the hierarchy.** In other words, appenders are inherited additively from the logger hierarchy. For example, if a console appender is added to the *root* logger, then all enabled logging requests will at least print on the console. If in addition a file appender is added to a logger, say *X*, then enabled logging requests for *X* and *X*'s children will print on a file *and* on the console. It is possible to override this default behavior so that appender accumulation is no longer additive by setting the additivity flag on the logger to false.

有关输出器的规则概述如下。The rules governing appender additivity are summarized below.

**Appender可加性（Appender Additivity）**

logger X的日志语句的输出将发送到X及其祖先中的所有输出器。这就是“输出器可加性”一词的含义。The output of a log statement of logger *X* will go to all the appenders in *X* and its ancestors. This is the meaning of the term "appender additivity".

但是，如果日志记录器X的祖先，比如Y，将可加性标记设置为false，那么X的输出将定向到X中的所有输出器，并且它的祖先将一直到Y，并包括Y，但不包括Y的任何祖先中的输出器。However, if an ancestor of logger *X*, say *Y*, has the additivity flag set to false, then *X*'s output will be directed to all the appenders in *X* and it's ancestors up to and including *Y* but not the appenders in any of the ancestors of *Y*.

默认情况下，记录器的可加性标记设置为true。下表给出了一个例子:Loggers have their additivity flag set to true by default. The table below shows an example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Logger Name** | **Added Appenders** | **Additivity Flag** | **Output Targets** | **Comment** |
| *root* | A1 | not applicable | A1 | There is no default appender attached to *root*. |
| x | A-x1, A-x2 | true | A1, A-x1, A-x2 | Appenders of "x" and *root*. |
| x.y | none | true | A1, A-x1, A-x2 | Appenders of "x" and *root*. |
| x.y.z | A-xyz1 | true | A1, A-x1, A-x2, A-xyz1 | Appenders in "x.y.z", "x" and *root*. |
| security | A-sec | false | A-sec | No appender accumulation since the additivity flag is set to false. |
| security.access | none | true | A-sec | Only appenders of "security" because the additivity flag in "security" is set to false. |

**三、筛选器(Filters)**

输出器可以过滤传递给它们的事件。可以在配置中指定过滤器，以便对通过不同的输出器记录的事件进行精细控制。Appenders can filter the events that are delivered to them. The filters can be specified in the configuration to allow fine control of the events that are logged through different appenders.

最简单的控制形式是在输出器上指定阈值。它只记录级别大于或等于阈值的事件。The simplest form of control is to specify a Threshold on the appender. This works by logging only the events that have a level that is greater than or equal to the threshold.

可以使用在每个输出器上定义的过滤器链来执行更复杂的自定义事件过滤。过滤器必须实现log4net.Filter.IFilter接口。

More complex and custom event filtering can be done using the filter chain defined on each appender. Filters must implement the log4net.Filter.IFilter interface.

log4net包中定义了以下过滤器:

The following filters are defined in the log4net package:

|  |  |
| --- | --- |
| **Type** | **Description** |
| [log4net.Filter.DenyAllFilter](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Filter_DenyAllFilter.htm) | 删除所有日志事件。  Drops all logging events. |
| [log4net.Filter.LevelMatchFilter](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Filter_LevelMatchFilter.htm) | 与事件级别完全匹配。  An exact match to the event's level.  匹配一系列的级别。 |
| [log4net.Filter.LevelRangeFilter](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Filter_LevelRangeFilter.htm) | Matches against a range of levels.  与记录器名称的开头匹配。 |
| [log4net.Filter.LoggerMatchFilter](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Filter_LoggerMatchFilter.htm) | Matches against a the start of the logger name. |
|  |  |
|  |  |
| [log4net.Filter.PropertyFilter](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Filter_PropertyFilter.htm) | 匹配特定属性值的子字符串。  Matches a substring from a specific property value.  匹配事件消息中的子字符串。 |
| [log4net.Filter.StringMatchFilter](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Filter_StringMatchFilter.htm) | Matches a substring from the event's message. |

可以将筛选器配置为根据匹配接受或拒绝事件。The filters can be configured to either accept or reject the event based upon the match.

**四、布局(Layouts)**

通常，用户不仅希望自定义输出目的地，还希望自定义输出格式。这是通过将布局(layout)与输出器(appender)关联来实现的。布局负责根据用户的愿望格式化日志请求，而输出器负责将格式化的输出发送到目的地。PatternLayout是标准log4net分发版的一部分，它允许用户根据类似于C语言printf函数的转换模式指定输出格式。More often than not, users wish to customize not only the output destination but also the output format. This is accomplished by associating a *layout* with an appender. The layout is responsible for formatting the logging request according to the user's wishes, whereas an appender takes care of sending the formatted output to its destination. The PatternLayout, part of the standard log4net distribution, lets the user specify the output format according to conversion patterns similar to the C language printf function.

例如，转换模式“%timestamp [%thread] %-5level %logger - %message%newline”的模式布局(PatternLayout)将输出类似于: For example, the PatternLayout with the conversion pattern "%timestamp [%thread] %-5level %logger - %message%newline" will output something akin to:

|  |
| --- |
| [main] INFO Com.Foo.Bar - Located nearest gas station. |

第一个字段是程序开始后经过的毫秒数。第二个字段是发出日志请求的线程。第三个字段是log语句的级别。第四个字段是与日志请求关联的日志记录器的名称。“-”后面的文本是该语句的消息。log4net包中包含以下布局: The first field is the number of milliseconds elapsed since the start of the program. The second field is the thread making the log request. The third field is the level of the log statement. The fourth field is the name of the logger associated with the log request. The text after the '-' is the message of the statement. The following layouts are included in the log4net package:

|  |  |
| --- | --- |
| **Type** | **Description** |
| [log4net.Layout.ExceptionLayout](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_ExceptionLayout.htm) | 渲染来自日志事件的异常文本。Renders the exception text from the logging event. |
| [log4net.Layout.PatternLayout](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_PatternLayout.htm) | 根据一组灵活的格式化标记对日志事件进行格式化。Formats the logging event according to a flexible set of formatting flags. |
| [log4net.Layout.RawTimeStampLayout](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_RawTimeStampLayout.htm) | 从日志事件中提取时间戳。Extracts the timestamp from the logging event. |
| [log4net.Layout.RawUtcTimeStampLayout](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_RawUtcTimeStampLayout.htm) | 从通用时间的日志事件中提取时间戳。Extracts the timestamp from the logging event in Universal Time. |
| [log4net.Layout.SimpleLayout](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_SimpleLayout.htm) | 格式化日志事件非常简单:[level] - [message]  Formats the logging event very simply: [level] - [message] |
| [log4net.Layout.XmlLayout](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_XmlLayout.htm) | 将日志事件格式化为XML元素。Formats the logging event as an XML element. |
| [log4net.Layout.XmlLayoutSchemaLog4j](http://logging.apache.org/log4net/release/sdk/html/T_log4net_Layout_XmlLayoutSchemaLog4j.htm) | 将日志事件格式化为符合log4j事件dtd的XML元素。Formats the logging event as an XML element that complies with the log4j event dtd. |

**五、对象渲染(Object Renderers)**

同样重要的是，log4net将根据用户指定的条件呈现日志消息的内容。例如，如果您经常需要记录当前项目中使用的对象类型Oranges，那么您可以注册一个OrangeRenderer，该程序将在需要记录orange时被调用。Just as importantly, log4net will render the content of the log message according to user specified criteria. For example, if you frequently need to log Oranges, an object type used in your current project, then you can register an OrangeRenderer that will be invoked whenever an orange needs to be logged.

对象渲染遵循类层次结构。例如，假设桔子是水果，如果你注册了一个水果渲染器，所有的水果包括桔子都会被渲染，除非你注册了一个特定的桔子渲染器。Object rendering follows the class hierarchy. For example, assuming oranges are fruits, if you register an FruitRenderer, all fruits including oranges will be rendered by the FruitRenderer, unless of course you registered an orange specific OrangeRenderer.

对象渲染器必须实现log4net.ObjectRenderer.IObjectRenderer接口。Object renderers have to implement the log4net.ObjectRenderer.IObjectRenderer interface.

请注意，DebugFormat、InfoFormat、WarnFormat、ErrorFormat和FatalFormat方法不使用ObjectRenderers。Please note that ObjectRenderers are not used by the DebugFormat, InfoFormat, WarnFormat, ErrorFormat and FatalFormat methods.

**六、log4net配置 (Configuration)**

**1.默认配置**

将日志请求插入应用程序代码需要大量的计划和工作。观察表明，大约4%的代码用于日志记录。因此，即使是中等规模的应用程序也会在其代码中嵌入数千条日志语句。有了它们的数量，就必须管理这些日志语句，而不需要手动修改它们。Inserting log requests into the application code requires a fair amount of planning and effort. Observation shows that approximately 4 percent of code is dedicated to logging. Consequently, even moderately sized applications will have thousands of logging statements embedded within their code. Given their number, it becomes imperative to manage these log statements without the need to modify them manually.

log4net环境是完全可编程配置的。但是，使用配置文件配置log4net要灵活得多。目前，配置文件是用XML编写的。The log4net environment is fully configurable programmatically. However, it is far more flexible to configure log4net using configuration files. Currently, configuration files are written in XML.

让我们在一个使用log4net的假想应用程序MyApp的帮助下尝试一下这是如何实现的。Let us give a taste of how this is done with the help of an imaginary application MyApp that uses log4net.

|  |
| --- |
| using Com.Foo;  // Import log4net classes.  using log4net;  using log4net.Config;  public class MyApp  {  // Define a static logger variable so that it references the  // Logger instance named "MyApp".  private static readonly ILog log = LogManager.GetLogger(typeof(MyApp));  static void Main(string[] args)  {  // Set up a simple configuration that logs on the console.  BasicConfigurator.Configure();  log.Info("Entering application.");  Bar bar = new Bar();  bar.DoIt();  log.Info("Exiting application.");  }  } |

MyApp首先导入log4net相关的类。然后，它定义一个名为MyApp的静态日志程序变量，该变量恰好是类的完全限定名。MyApp使用以下Bar类: MyApp begins by importing log4net related classes. It then defines a static logger variable with the name MyApp which happens to be the fully qualified name of the class. MyApp uses the following Bar class:

|  |
| --- |
| // Import log4net classes.  using log4net;  namespace Com.Foo  {  public class Bar  {  private static readonly ILog log = LogManager.GetLogger(typeof(Bar));  public void DoIt()  {  log.Debug("Did it again!");  }  }  } |

调用BasicConfigurator.Configure()方法创建了一个相当简单的log4net设置。此方法被硬连接到根日志记录器中以添加一个ConsoleAppender。输出将使用模式“%-4timestamp [%thread] %-5level %logger %ndc - %message%newline”设置的模式进行格式化。

注意，默认情况下，根日志记录器被分配到Level.DEBUG。

MyApp的输出为:

The invocation of the BasicConfigurator.Configure() method creates a rather simple log4net setup. This method is hardwired to add to the *root* logger a ConsoleAppender. The output will be formatted using a PatternLayout set to the pattern "%-4timestamp [%thread] %-5level %logger %ndc - %message%newline".

Note that by default, the *root* logger is assigned to Level.DEBUG.

The output of MyApp is:

|  |
| --- |
| [main] INFO MyApp - Entering application.  [main] DEBUG Com.Foo.Bar - Did it again!  [main] INFO MyApp - Exiting application. |

作为补充说明，让我提一下，在log4net中，子日志记录器仅链接到它们现有的祖先。特别是名为Com.Foo的日志记录器。Bar直接链接到根日志记录器，从而绕过未使用的Com或Com.Foo伐木工。这将显著提高性能并减少log4net的内存占用。As a side note, let me mention that in log4net child loggers link only to their existing ancestors. In particular, the logger named Com.Foo.Bar is linked directly to the *root* logger, thereby circumventing the unused Com or Com.Foo loggers. This significantly increases performance and reduces log4net's memory footprint.

MyApp类通过调用BasicConfigurator.Configure()方法来配置log4net。其他类只需要导入log4net名称空间，检索它们希望使用的日志记录器，然后注销。The MyApp class configures log4net by invoking BasicConfigurator.Configure() method. Other classes only need to import the log4net namespace, retrieve the loggers they wish to use, and log away.

前面的示例总是输出相同的日志信息。幸运的是，修改MyApp很容易，因此可以在运行时控制日志输出。下面是一个稍微修改过的版本。The previous example always outputs the same log information. Fortunately, it is easy to modify MyApp so that the log output can be controlled at run-time. Here is a slightly modified version.

|  |
| --- |
| using Com.Foo;  // Import log4net classes.  using log4net;  using log4net.Config;  public class MyApp  {  private static readonly ILog log = LogManager.GetLogger(typeof(MyApp));  static void Main(string[] args)  {  // BasicConfigurator replaced with XmlConfigurator.  XmlConfigurator.Configure(new System.IO.FileInfo(args[0]));  log.Info("Entering application.");  Bar bar = new Bar();  bar.DoIt();  log.Info("Exiting application.");  }  } |

这个版本的MyApp指示XmlConfigurator解析配置文件并相应地设置日志记录。配置文件的路径在命令行中指定。下面是一个示例配置文件，它的输出与前面的基于BasicConfigurator的示例完全相同。This version of MyApp instructs the XmlConfigurator to parse a configuration file and set up logging accordingly. The path to the configuration file is specified on the command line. Here is a sample configuration file that results in exactly same output as the previous BasicConfigurator based example.

|  |
| --- |
| <log4net>  <!-- A1 is set to be a ConsoleAppender -->  <appender name = "A1" type="log4net.Appender.ConsoleAppender">    <!-- A1 uses PatternLayout -->  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%-4timestamp [%thread] %-5level %logger %ndc - %message%newline" />  </layout>  </appender>    <!-- Set root logger level to DEBUG and its only appender to A1 -->  <root>  <level value = "DEBUG" />  < appender - ref ref= "A1" />  </ root >  </ log4net > |

假设我们不再有兴趣查看属于Com.Foo包的任何组件的输出。下面的配置文件展示了一种可能的实现方法。Suppose we are no longer interested in seeing the output of any component belonging to the Com.Foo package. The following configuration file shows one possible way of achieving this.

|  |
| --- |
| <log4net>  <!-- A1 is set to be a ConsoleAppender -->  <appender name = "A1" type="log4net.Appender.ConsoleAppender">    <!-- A1 uses PatternLayout -->  <layout type = "log4net.Layout.PatternLayout" >  < !--Print the date in ISO 8601 format -->  <conversionPattern value = "%date [%thread] %-5level %logger %ndc - %message%newline" />  </ layout >  </ appender >  < !--Set root logger level to DEBUG and its only appender to A1 -->  <root>  <level value = "DEBUG" />  < appender - ref ref= "A1" />  </ root >  < !--Print only messages of level WARN or above in the package Com.Foo -->  <logger name = "Com.Foo" >  < level value="WARN" />  </logger>  </log4net> |

使用该文件配置的MyApp的输出如下所示。The output of MyApp configured with this file is shown below.

|  |
| --- |
| 2000-09-07 14:07:41,508 [main] INFO MyApp - Entering application.  2000-09-07 14:07:41,529 [main] INFO MyApp - Exiting application. |

作为日志程序Com.Foo.Bar没有指定的级别，它从Com继承它的级别。在配置文件中设置为WARN。

Bar中的log语句DoIt方法具有级别调试，低于记录器级别警告。因此，DoIt()方法的日志请求被抑制。下面是另一个使用多个输出器的配置文件。As the logger Com.Foo.Bar does not have an assigned level, it inherits its level from Com.Foo, which was set to WARN in the configuration file. The log statement from the Bar.DoIt method has the level DEBUG, lower than the logger level WARN. Consequently, DoIt() method's log request is suppressed. Here is another configuration file that uses multiple appenders.

|  |
| --- |
| <log4net>  <appender name = "Console" type="log4net.Appender.ConsoleAppender">  <layout type = "log4net.Layout.PatternLayout" >  < !--Pattern to output the caller's file name and line number -->  <conversionPattern value = "%5level [%thread] (%file:%line) - %message%newline" />  </ layout >  </ appender >  < appender name="RollingFile" type="log4net.Appender.RollingFileAppender">  <file value = "example.log" />  < appendToFile value="true" />  <maximumFileSize value = "100KB" />  < maxSizeRollBackups value="2" />    <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%level %thread %logger - %message%newline" />  </layout>  </appender>    <root>  <level value = "DEBUG" />  < appender - ref ref= "Console" />  < appender - ref ref= "RollingFile" />  </ root >  </log4net> |

使用此配置文件调用增强的MyApp将在控制台上输出以下内容。

Calling the enhanced MyApp with the this configuration file will output the following on the console.

|  |
| --- |
| INFO[main] (MyApp.cs:16) - Entering application.  DEBUG [main] (Bar.cs:12) - Doing it again!  INFO[main] (MyApp.cs:19) - Exiting application. |

此外，由于根日志记录器已经分配了第二个输出器，所以输出也将定向到example.log文件。当该文件达到100KB时将被滚动。当发生滚转时，旧版本的example.log将自动转移到example.log.1。In addition, as the *root* logger has been allocated a second appender, output will also be directed to the example.log file. This file will be rolled over when it reaches 100KB. When roll-over occurs, the old version of example.log is automatically moved to example.log.1.

注意，为了获得这些不同的日志记录行为，我们不需要重新编译代码。我们可以很容易地登录到一个电子邮件地址，重定向所有Com.Foo输出到NT事件日志记录器，或将日志事件转发到远程log4net服务器，后者将根据本地服务器策略进行日志记录。Note that to obtain these different logging behaviors we did not need to recompile code. We could just as easily have logged to an email address, redirected all Com.Foo output to an NT Event logger, or forwarded logging events to a remote log4net server, which would log according to local server policy.

有关使用XmlConfigurator配置Appender的更多示例，请参阅示例Appender配置文档（[Example Appender Configuration](http://logging.apache.org/log4net/release/config-examples.html)）。For more examples of configuring appenders using the XmlConfigurator see the [Example Appender Configuration](http://logging.apache.org/log4net/release/config-examples.html) document.

**2.通过XmlConfiguratorAttribute属性配置**

log4net配置可以使用程序集级别(assembly-level)的属性XmlConfiguratorAttribute

进行配置，而不是通过编程方式指定。log4net.Config.XmlConfiguratorAttribute允许使用以下属性配置XmlConfigurator:

(1)ConfigFile

如果指定，这是要与XmlConfigurator一起使用的配置文件的文件名。此文件路径相对于应用程序基目录(AppDomain.CurrentDomain.BaseDirectory)。此属性不能与ConfigFileExtension属性一起使用。If specified, this is the filename of the configuration file to use with the XmlConfigurator. This file path is relative to the application base directory (AppDomain.CurrentDomain.BaseDirectory). This property cannot be used in conjunction with the ConfigFileExtension property.

(2)ConfigFileExtension

如果指定，这是配置文件的扩展名。程序集文件名用作附加此扩展名的基名。例如，如果程序集是从文件TestApp.exe加载的，并且ConfigFileExtension属性设置为log4net，那么配置文件名就是TestApp.exe.log4net。这相当于将ConfigFile属性设置为TestApp.exe.log4net。

配置文件的路径是通过使用应用程序基本目录(AppDomain.CurrentDomain.BaseDirectory)、程序集文件名和配置文件扩展名来构建的。该属性不能与ConfigFile属性一起使用。If specified, this is the extension for the configuration file. The assembly file name is used as the base name with the this extension appended. For example if the assembly is loaded from the a file TestApp.exe and the ConfigFileExtension property is set to log4net then the configuration file name is TestApp.exe.log4net. This is equivalent to setting the ConfigFile property to TestApp.exe.log4net. The path to the configuration file is build by using the application base directory (AppDomain.CurrentDomain.BaseDirectory), the assembly file name and the configuration file extension. This property cannot be used in conjunction with the ConfigFile property.

(3)Watch

如果指定了此标志并将其设置为true，则log4net将监视配置文件，并在每次修改该文件时重新加载配置。If this flag is specified and set to true then the framework will watch the configuration file and will reload the config each time the file is modified.

如果没有指定ConfigFile或ConfigFileExtension属性，则应用程序配置文件(例如TestApp.exe.config)将用作log4net配置文件。示例如下: If neither of the ConfigFile or ConfigFileExtension properties are specified, the application configuration file (e.g. TestApp.exe.config) will be used as the log4net configuration file. Example usage:

|  |
| --- |
| // Configure log4net using the .config file  [assembly: log4net.Config.XmlConfigurator(Watch=true)]  // This will cause log4net to look for a configuration file  // called TestApp.exe.config in the application base  // directory (i.e. the directory containing TestApp.exe)  // The config file will be watched for changes. |

|  |
| --- |
| // Configure log4net using the .log4net file  [assembly: log4net.Config.XmlConfigurator(ConfigFileExtension="log4net",Watch=true)]  // This will cause log4net to look for a configuration file  // called TestApp.exe.log4net in the application base  // directory (i.e. the directory containing TestApp.exe)  // The config file will be watched for changes. |

每个程序集只能使用此属性一次。

This attribute may only be used once per assembly.

使用属性可以更清楚地定义从何处加载应用程序的配置。然而，值得注意的是，属性纯粹是被动的。它们只是信息。因此，如果使用配置属性，则必须调用log4net来允许它读取属性。一个对LogManager的简单调用。GetLogger将导致读取和处理调用程序集上的属性。因此，必须在应用程序启动期间(当然是在加载和调用任何外部程序集之前)尽早进行日志记录调用。Using attributes can be a clearer method for defining where the application's configuration will be loaded from. However it is worth noting that attributes are purely passive. They are information only. Therefore if you use configuration attributes you must invoke log4net to allow it to read the attributes. A simple call to LogManager.GetLogger will cause the attributes on the calling assembly to be read and processed. ***Therefore it is imperative to make a logging call as early as possible during the application start-up, and certainly before any external assemblies have been loaded and invoked.***

**3.在appSettings中配置**

首先你要关闭属性配置。If you use [attributes](http://logging.apache.org/log4net/release/manual/configuration.html#attributes) to configure log4net, two settings of the appSettings section of your application's configuration file can be used to override the values given in your assembly's attributes. The setting with the key "log4net.Config" overrides the configuration file name (and is considered to be relative to your application's base directory), the setting with the key "log4net.Config.Watch" determines whether the file should be monitored for changes. Even though the assembly attribute

|  |
| --- |
| [assembly: log4net.Config.XmlConfigurator(Watch=false)] |

将您的应用程序配置为使用一个配置文件“TestApp.exe.config“，然后增加appSettings段：

would configure your application to use a configuration file "TestApp.exe.config" and not monitor it for changes you can override this to use the file "log4net.config" and monitor it instead by adding

|  |
| --- |
| <appSettings>  <add key = "log4net.Config" value="log4net.config"/>  <add key = "log4net.Config.Watch" value="True"/>  </appSettings> |

**4.通过配置文件配置(Configuration Files)**

通常，log4net配置是使用文件指定的。该文件可以通过以下两种方式之一读取:

Typically the log4net configuration is specified using a file. This file can be read in one of two ways:

使用.NET系统配置API(Using the .NET System.Configuration API);

直接读取文件内容(Reading the file contents directly)。

**(1).config Files**

仅当配置数据位于应用程序的配置文件（名为MyApp.exe.config或Web.config）中时，System.Configuration API才可用。由于System.Configuration API不支持重新加载配置文件，因此无法使用log4net.config.XmlConfigurator.ConfigureAndWatch方法监视配置设置。使用System.Configuration API读取配置数据的主要优点是，它需要的权限比直接访问配置文件少。

The System.Configuration API is only available if the configuration data is in the application's config file; the file named *MyApp.exe.config* or *Web.config*. Because the System.Configuration API does not support reloading of the config file the configuration settings cannot be watched using the log4net.Config.XmlConfigurator.ConfigureAndWatch methods. The main advantage of using the System.Configuration APIs to read the configuration data is that it requires less permissions than accessing the configuration file directly.

使用System.Configuration APIs配置应用程序的惟一方法是调用log4net.Config.XmlConfigurator.Configure()方法或log4net.Config.XmlConfigurator.Configure(ILoggerRepository)方法。

The only way to configure an application using the System.Configuration APIs is to call the log4net.Config.XmlConfigurator.Configure() method or the log4net.Config.XmlConfigurator.Configure(ILoggerRepository) method.

为了将配置数据嵌入到.config文件中，必须使用configSections元素将段名标识到.NET配置文件解析器中。该部分必须指定log4net.Config.Log4NetConfigurationSectionHandler，用于解析配置部分。此类型必须是完全限定的程序集，因为它是由.NET配置文件解析器加载的，而不是由log4net加载的。必须指定log4net程序集的正确程序集名称。下面是一个简单的配置示例文件，它指定了log4net段使用的正确段处理程序。

In order to embed the configuration data in the .config file the section name must be identified to the .NET config file parser using a configSections element. The section must specify the log4net.Config.Log4NetConfigurationSectionHandler that will be used to parse the config section. This type must be fully assembly qualified because it is being loaded by the .NET config file parser not by log4net. The correct assembly name for the log4net assembly must be specified. The following is a simple example configuration file that specifies the correct section handler to use for the log4net section.

|  |
| --- |
| <? xml version="1.0" encoding="utf-8" ?>  <configuration>  <configSections>  <section name = "log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />  </configSections>  <log4net>  <appender name = "ConsoleAppender" type="log4net.Appender.ConsoleAppender" >  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />  </layout>  </appender>  <root>  <level value = "INFO" />  < appender - ref ref= "ConsoleAppender" />  </ root >  </ log4net >  </configuration> |

在上面的示例中，指定了log4net程序集。该程序集必须位于.NET运行时可以找到它的位置。例如，它可以与应用程序位于相同的目录中。如果log4net程序集存储在GAC中，则必须指定完全限定程序集名称，包括区域性、版本和公钥。在使用.config文件指定配置时，段名(section name)和XML元素名(element name)必须是log4net。In the above example the log4net assembly is specified. This assembly must be located where the .NET runtime can find it. For example it could be located in the same directory as the application. If the log4net assembly is stored in the GAC then the fully qualified assembly name must be specified including the culture, version and public key. When using the .config file to specify the configuration the section name and XML element name must be log4net.

**5.从文件目录读取配置(Reading Files Directly)**

XmlConfigurator可以直接读取任何XML文件并使用它配置log4net。这包括应用程序的.config文件，名为MyApp.exe.config或Web.config的文件。不直接读取配置文件的唯一原因是，如果应用程序没有足够的权限读取该文件，则必须使用.NET配置API加载配置（请参见上文）。The XmlConfigurator can directly read any XML file and use it to configure log4net. This includes the application's .config file; the file named *MyApp.exe.config* or *Web.config*. The only reason not to read the configuration file directly is if the application does not have sufficient permissions to read the file, then the configuration must be loaded using the .NET configuration APIs (see above).

可以使用接受System.IO.FileInfo对象的任何log4net.Config.XmlConfigurator方法指定要从中读取配置的文件。因为可以监视文件系统的文件更改通知，所以ConfigureAndWatch方法可用于监视配置文件的修改并自动重新配置log4net。The file to read the configuration from can be specified using any of the log4net.Config.XmlConfigurator methods that accept a System.IO.FileInfo object. Because the file system can be monitored for file change notifications the ConfigureAndWatch methods can be used to monitor the configuration file for modifications and automatically reconfigure log4net.

此外，log4net.Config.XmlConfigurator属性可用于指定文件从中读取配置。Additionally the log4net.Config.XmlConfiguratorAttribute can be used to specify the file to read the configuration from.

从文件中的log4net元素读取配置。文件中只能指定一个log4net元素，但它可能位于XML层次结构中的任何位置。例如，它可能是根元素：The configuration is read from the log4net element in the file. Only one log4net element can be specified in the file but it may be located anywhere in the XML hierarchy. For example it may be the root element:

|  |
| --- |
| <? xml version="1.0" encoding="utf-8" ?>  <log4net>  <appender name = "ConsoleAppender" type="log4net.Appender.ConsoleAppender" >  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />  </layout>  </appender>  <root>  <level value = "INFO" />  < appender - ref ref= "ConsoleAppender" />  </ root >  </log4net> |

也可以嵌套在其他元素中: Or it may be nested within other elements:

|  |
| --- |
| <? xml version="1.0" encoding="utf-8" ?>  <configuration>  <configSections>  <section name = "log4net" type="System.Configuration.IgnoreSectionHandler" />  </configSections>  <log4net>  <appender name = "ConsoleAppender" type="log4net.Appender.ConsoleAppender" >  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />  </layout>  </appender>  <root>  <level value = "INFO" />  < appender - ref ref= "ConsoleAppender" />  </ root >  </ log4net >  </configuration> |

上面的示例显示了如何将配置数据嵌入到.config文件中，即使log4net正在直接读取该文件。一个重要的注意事项是，如果.NET配置文件解析器发现一个未使用configSections元素注册的元素，它将抛出异常。因此，在上面的示例中，注册了log4net段名称(section name)，但指定用于处理该段的类型是System.Configuration.IgnoreSectionHandler。这是一个内置类，指示将使用另一个读取配置段的方法。The above example shows how the configuration data can be embedded inside a .config file even though the file is being read directly by log4net. An important note is that the .NET config file parser will throw an exception if it finds an element that has not been registered using the configSections element. Therefore in the above example the log4net section name is registered, but the type specified to handle the section is System.Configuration.IgnoreSectionHandler. This is a built-in class that indicates that another method for reading the config section will be employed.

**6.配置的语法(Configuration Syntax)**

log4net包括一个解析XML DOM的配置读取器log4net.Config.XmlConfigurator。本节定义配置程序接受的语法。log4net includes a configuration reader that parses an XML DOM, the log4net.Config.XmlConfigurator. This section defines the syntax accepted by the configurator.

这是一个有效的XML配置示例。根元素必须是<log4net>。注意，这并不意味着这个元素不能嵌入到另一个XML文档中。有关如何在配置文件中嵌入XmlConfigurator XML的更多信息，请参阅上面关于配置文件([Configuration Files](http://logging.apache.org/log4net/release/manual/configuration.html#configuration.files))的部分。This is an example of a valid XML configuration. The root element must be <log4net>. Note that this does not mean that this element cannot be embedded in another XML document. See the section above on [Configuration Files](http://logging.apache.org/log4net/release/manual/configuration.html#configuration.files) for more information on how to embed the XmlConfigurator XML in a configuration file.

|  |
| --- |
| <log4net>  <appender name = "ConsoleAppender" type="log4net.Appender.ConsoleAppender" >  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />  </layout>  </appender>  <root>  <level value = "INFO" />  < appender - ref ref= "ConsoleAppender" />  </ root >  </log4net> |

<log4net>元素支持以下属性(The <log4net> element supports the following attributes)：

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| debug | 可选属性。值必须为真或假。默认值为false。将此属性设置为true以启用此配置的内部log4net调试。Optional attribute. Value must be either true or false. The default value is false. Set this attribute to true to enable internal log4net debugging for this configuration. |
| update | 可选属性。值必须合并或覆盖。默认值是Merge。将此属性设置为Overwrite，以便在应用此配置之前重置正在配置的存储库配置。Optional attribute. Value must be either Merge or Overwrite. The default value is Merge. Set this attribute to Overwrite to reset the configuration of the repository being configured before applying this configuration. |
| threshold | 可选属性。值必须是在存储库上注册的级别的名称。默认值是ALL。设置此属性以限制整个存储库中记录的消息，而不管记录消息的日志记录器是什么。Optional attribute. Value must be the name of a level registered on the repository. The default value is ALL. Set this attribute to limit the messages that are logged across the whole repository, regardless of the logger that the message is logged to. |

<log4net>元素支持以下子元素(The <log4net> element supports the following child elements):

|  |  |
| --- | --- |
| **Element** | **Description** |
| appender | 允许零个或多个元素。定义输出器。  Zero or more elements allowed. Defines an appender. |
| logger | 允许零个或多个元素。定义记录器的配置。  Zero or more elements allowed. Defines the configuration of a logger. |
| renderer | 允许零个或多个元素。定义对象渲染器。  Zero or more elements allowed. Defines an object renderer. |
| root | 可选元素，最多允许一个。定义根记录器的配置。  Optional element, maximum of one allowed. Defines the configuration of the root logger. |
| param | 允许零个或多个元素。存储库特定参数。  Zero or more elements allowed. Repository specific parameters |

**7.配置输出器(Appenders)**

输出器只能定义为<log4net>元素的子元素。每个输出器必须具有唯一的名称。必须指定输出器的实现类型。下面示例显示实现类型为log4net.appender.ConsoleAppender的输出器，被命名为ConsoleAppender。Appenders may only be defined as child elements of the <log4net> element. Each appender must be uniquely named. The implementing type for the appender must be specified. This example shows an appender of type log4net.Appender.ConsoleAppender being defined. The appender will be known as *ConsoleAppender*.

|  |
| --- |
| <appender name = "ConsoleAppender" type="log4net.Appender.ConsoleAppender" >  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />  </layout>  </appender> |

<appender>元素支持以下属性(The <appender> element supports the following attributes):

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| name | 必需属性。值必须是此输出器的字符串名称。在该配置文件中定义的所有输出器中，名称必须是唯一的。记录器(Logger)的<appender ref>元素使用此名称来引用输出器。  Required attribute. Value must be a string name for this appender. The name must be unique among all the appenders defined in this configuration file. This name is used by the <appender-ref> element of a Logger to reference an appender. |
| type | 必需属性。值必须是此输出器的类型名。如果在log4net程序集中未定义输出器，则此类型名必须是完全程序集限定的。  Required attribute. Value must be the type name for this appender. If the appender is not defined in the log4net assembly this type name must be fully assembly qualified. |

<appender>元素支持以下子元素(The <appender> element supports the following child elements):

|  |  |
| --- | --- |
| **Element** | **Description** |
| appender-ref | 允许零个或多个元素。允许输出器引用其他输出器。不是所有输出器都支持。  Zero or more elements allowed. Allows the appender to reference other appenders. Not supported by all appenders. |
| filter | 允许零个或多个元素。定义此输出器使用的筛选器。  Zero or more elements allowed. Defines the filters used by this appender. |
| layout | 可选元素，最多允许一个。定义此输出器使用的布局。  Optional element, maximum of one allowed. Defines the layout used by this appender. |
| param | 允许零个或多个元素。输出器的特定参数。  Zero or more elements allowed. Appender specific parameters. |

有关配置Appender的示例，请参阅示例Appender配置文档。([Example Appender Configuration](http://logging.apache.org/log4net/release/config-examples.html))

**8.配置过滤器(Filters)**

过滤器元素只能定义为<appender>元素的子元素。

Filters elements may only be defined as children of <appender> elements.)。

<filter>元素支持以下属性：

The <filter> element supports the following attributes：

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| type | 必需属性。值必须是此筛选器的类型名。如果未在log4net程序集中定义筛选器，则此类型名必须是完全程序集限定的。  Required attribute. Value must be the type name for this filter. If the filter is not defined in the log4net assembly this type name must be fully assembly qualified. |

<filter>元素支持以下子元素：

The <filter> element supports the following child elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| param | 允许零个或多个元素。筛选特定参数。  Zero or more elements allowed. Filter specific parameters. |

过滤器形成事件必须经过的链。沿途的任何筛选器都可以接受事件并停止处理，拒绝事件并停止处理，或者允许事件转到下一个筛选器。如果事件到达筛选器链的末尾而未被拒绝，则会隐式接受该事件并将其记录下来。Filters form a chain that the event has to pass through. Any filter along the way can accept the event and stop processing, deny the event and stop processing, or allow the event on to the next filter. If the event gets to the end of the filter chain without being denied it is implicitly accepted and will be logged.

|  |
| --- |
| <filter type = "log4net.Filter.LevelRangeFilter" >  < levelMin value="INFO" />  <levelMax value = "FATAL" />  </filter> |

此筛选器将拒绝级别低于INFO或高于FATAL的事件。将记录“信息”和“致命”之间的所有事件。This filter will deny events that have a level that is lower than INFO or higher than FATAL. All events between INFO and FATAL will be logged.

如果我们只允许具有特定子字符串（例如“database”）的消息通过，则需要指定以下筛选器：If we want to only allow messages through that have a specific substring (e.g. 'database') then we need to specify the following filters:

|  |
| --- |
| <filter type = "log4net.Filter.StringMatchFilter" >  < stringToMatch value="database" />  </filter>  <filter type = "log4net.Filter.DenyAllFilter" /> |

第一个筛选器将在事件的消息文本中查找子字符串“database”。如果找到文本，筛选器将接受该消息，并且筛选器处理将停止，则将记录该消息。如果未找到子字符串，则事件将传递到下一个要处理的筛选器。如果没有下一个过滤器，事件将被隐式接受并被记录。但是因为我们不想记录不匹配的事件，所以我们需要使用log4net.Filter.DenyAllFilter来拒绝所有到达它的事件。此筛选器仅在筛选器链的末尾有用。The first filter will look for the substring 'database' in the message text of the event. If the text is found the filter will accept the message and filter processing will stop, the message will be logged. If the substring is not found the event will be passed to the next filter to process. If there is no next filter the event would be implicitly accepted and would be logged. But because we don't want the non matching events to be logged we need to use a log4net.Filter.DenyAllFilter that will just deny all events that reach it. This filter is only useful at the end of the filter chain.

如果要允许消息文本中包含“database”或“ldap”的事件，可以使用以下筛选器：If we want to allow events that have either 'database' or 'ldap' in the message text we can use the following filters:

|  |
| --- |
| <filter type = "log4net.Filter.StringMatchFilter" >  < stringToMatch value="database"/>  </filter>  <filter type = "log4net.Filter.StringMatchFilter" >  < stringToMatch value="ldap"/>  </filter>  <filter type = "log4net.Filter.DenyAllFilter" /> |

**9.配置布局 (Layouts)**

布局元素只能定义为<appender>元素的子元素。

Layout elements may only be defined as children of <appender> elements.

<layout>元素支持以下属性：

The <layout> element supports the following attributes:

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| type | 必需属性。值必须是此布局的类型名。如果未在log4net程序集中定义布局，则此类型名必须是完全程序集限定的。 Required attribute. Value must be the type name for this layout. If the layout is not defined in the log4net assembly this type name must be fully assembly qualified. |

<layout>元素支持以下子元素：

The <layout> element supports the following child elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| param | 允许零个或多个元素。布局特定参数。  Zero or more elements allowed. Layout specific parameters. |

此示例演示如何配置使用log4net.layout.PatternLayout的布局。

This example shows how to configure a layout that uses the log4net.Layout.PatternLayout.

|  |
| --- |
| <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="%date [%thread] %-5level %logger [%ndc] - %message%newline" />  </layout> |

**10.根日志器 (Root Logger)**

只能定义一个根记录器元素，并且它必须是<log4net>元素的子元素。根记录器是记录器层次结构的根。所有记录器最终都继承自此记录器。Only one root logger element may only be defined and it must be a child of <log4net> element. The root logger is the root of the logger hierarchy. All loggers ultimately inherit from this logger.

根记录器示例：

An example root logger:

|  |
| --- |
| <root>  <level value = "INFO" />  < appender - ref ref= "ConsoleAppender" />  </ root > |

<root>元素不支持任何属性。

The <root> element supports no attributes.

<root>元素支持以下子元素：

The <root> element supports the following child elements:

|  |  |
| --- | --- |
| Element | Description |
| appender-ref | 允许零个或多个元素。允许记录器按名称引用附加程序。  Zero or more elements allowed. Allows the logger to reference appenders by name. |
| level | 可选元素，最多允许一个。定义此记录器的日志记录级别。此记录器将只接受此级别或更高级别的事件。  Optional element, maximum of one allowed. Defines the logging level for this logger. This logger will only accept event that are at this level or above. |
| param | 允许零个或多个元素。记录器特定参数。  Zero or more elements allowed. Logger specific parameters. |

**11.日志器 (Loggers)**

Logger元素只能定义为<log4net>元素的子元素。日志记录器示例如下：

Logger elements may only be defined as children of the <log4net> element.

An example logger:

|  |
| --- |
| <logger name = "LoggerName" >  < level value="DEBUG" />  <appender-ref ref="ConsoleAppender" />  </logger> |

<logger>元素支持以下属性：

The <logger> element supports the following attributes：

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| name | 必需属性。值必须是记录器的名称。  Required attribute. Value must be the name of the logger. |
| additivity | 可选属性。值可以为true或false。默认值为true。将此属性设置为false可防止此记录器继承父记录器上定义的附加程序。  Optional attribute. Value may be either true or false. The default value is true. Set this attribute to false to prevent this logger from inheriting the appenders defined on parent loggers. |

<logger>元素支持以下子元素：

The <logger> element supports the following child elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| appender-ref | 允许零个或多个元素。允许记录器按名称引用输出器。  Zero or more elements allowed. Allows the logger to reference appenders by name. |
| level | 可选元素，最多允许一个。定义此记录器的日志记录级别。此记录器将只接受此级别或更高级别的事件。  Optional element, maximum of one allowed. Defines the logging level for this logger. This logger will only accept event that are at this level or above. |
| param | 允许零个或多个元素。记录器特定参数。  Zero or more elements allowed. Logger specific parameters. |

**12.渲染器 (Renderers)**

渲染器元素只能定义为<log4net>元素的子元素。渲染器示例：

Renderer elements may only be defined as children of the <log4net> element.

An example renderer:

|  |
| --- |
| <renderer renderingClass = "MyClass.MyRenderer" renderedClass="MyClass.MyFunkyObject" /> |

<renderer>元素支持以下属性：

The <renderer> element supports the following attributes：

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| renderingClass | 必需属性。值必须是此渲染器的类型名。如果在log4net程序集中未定义该类型，则该类型名称必须是完全程序集限定的。这是将负责呈现renderedClass的对象类型。  Required attribute. Value must be the type name for this renderer. If the type is not defined in the log4net assembly this type name must be fully assembly qualified. This is the type of the object that will take responsibility for rendering the *renderedClass*. |
| renderedClass | 必需属性。值必须是此呈现程序的目标类型的类型名。如果在log4net程序集中未定义该类型，则该类型名称必须是完全程序集限定的。这是此渲染器将呈现的类型的名称。  Required attribute. Value must be the type name for the target type for this renderer. If the type is not defined in the log4net assembly this type name must be fully assembly qualified. This is the name of the type that this renderer will render. |

<renderer>元素不支持子元素。

The <renderer> element supports no child elements.

**13.参数 (Parameters)**

参数元素可以是许多元素的子元素。请参阅上面的特定元素以了解详细信息。示例参数：

Parameter elements may be children of many elements. See the specific elements above for details. An example param:

|  |
| --- |
| <param name =“ConversionPattern”value=”%date[% thread]-5级%logger[% ndc]-%message%newline“/> |

<param>元素支持以下属性：

The <param> element supports the following attributes：

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| name | 必需属性。值必须是要在父对象上设置的参数的名称。  Required attribute. Value must be the name of the parameter to set on the parent object. |
| value | 可选属性。必须指定值或类型属性之一。此属性的值是可以转换为参数值的字符串。  Optional attribute. One of *value* or *type* attributes must be specified. The value of this attribute is a string that can be converted to the value of the parameter. |
| type | 可选属性。必须指定值或类型属性之一。此属性的值是要创建并设置为参数值的类型名。如果log4net程序集中未定义该类型，则该类型名称必须是完全程序集限定的。  Optional attribute. One of *value* or *type* attributes must be specified. The value of this attribute is a type name to create and set as the value of the parameter. If the type is not defined in the log4net assembly this type name must be fully assembly qualified. |

<param>元素支持以下子元素：

The <param> element supports the following child elements:

|  |  |
| --- | --- |
| **Element** | **Description** |
| param | 允许零个或多个元素。参数特定参数。  Zero or more elements allowed. Parameter specific parameters. |

一个使用嵌套参数元素的参数示例:

An example param that uses nested param elements:

|  |
| --- |
| <param name = "evaluator" type="log4net.spi.LevelEvaluator">  <param name = "Threshold" value="WARN"/>  <param> |

**14.扩展参数 (Extension Parameters)**

配置参数直接映射到对象的可写属性。可用的属性取决于正在配置的对象的实际类型。log4net SDK文档包含log4net程序集中包含的所有组件的API引用。

Configuration parameters map directly to writable properties on an object. The properties available depend on the actual type of the object being configured. The log4net SDK documentation contains the API reference for all the components included in the log4net assembly.

对于第三方组件，请参阅其相关的API参考，以了解可用属性的详细信息。

For 3rd party components please see their relevant API reference for details of the properties available.

**15.紧凑参数语法 (Compact Parameter Syntax)**

可以使用参数名作为元素名，而不是使用param element和name属性来交替指定所有参数。例如：

All parameters may alternately be specified using the parameter name as the element name rather than using the param element and name attribute.

For example a param:

|  |
| --- |
| <param name = "evaluator" type="log4net.spi.LevelEvaluator">  <param name = "Threshold" value="WARN"/>  <param> |

也可写成：

|  |
| --- |
| <evaluator type = "log4net.spi.LevelEvaluator" >  < threshold value="WARN"/>  <evaluator> |

**七、上下文 (Contexts)**

大多数现实系统必须同时处理多个客户端。在这种系统的典型多线程实现中，不同的线程将处理不同的客户端。日志记录特别适合跟踪和调试复杂的分布式应用程序。区分一个客户端和另一个客户端的日志输出的方法是为每个客户端实例化一个新的独立日志记录器。然而，这会促进记录器的激增，并增加日志的管理开销。Most real-world systems have to deal with multiple clients simultaneously. In a typical multithreaded implementation of such a system, different threads will handle different clients. Logging is especially well suited to trace and debug complex distributed applications. An approach to differentiate the logging output of one client from another is to instantiate a new separate logger for each client. However this promotes the proliferation of loggers and increases the management overhead of logging.

一种更简单的技术是对从同一个客户端交互启动的每个日志请求进行唯一的标记。

A lighter technique is to uniquely stamp each log request initiated from the same client interaction.

Log4net支持不同类型的上下文日志记录和具有不同作用域的上下文。

Log4net supports different types of contextual logging and contexts with different scopes.

**1.范围(Scopes)**

上下文数据可以在不同的范围内设置。这些上下文的可见性逐渐变小。在日志事件本身中，来自所有上下文的值被组合在一起，这样在范围较低的上下文中指定的值将隐藏来自较高上下文的值。

Contextual data can be set in different scopes. These contexts have progressively narrower visibility. In the logging event itself the values from all of the contexts are combined together such that values specified in a lower scoped context hide values from a higher context.

|  |  |  |
| --- | --- | --- |
| **Scope** | **Type** | **Description** |
| Global | log4net.GlobalContext | 全局上下文由当前AppDomain中的所有线程共享。此上下文是线程安全的，可供多个线程同时使用。  The global context is shared by all threads in the current AppDomain. This context is thread safe for use by multiple threads concurrently. |
| Thread | log4net.ThreadContext | 线程上下文仅对当前托管线程可见。  The thread context is visible only to the current managed thread. |
| Logical Thread | log4net.ThreadLogicalContext | 逻辑线程上下文对逻辑线程可见。逻辑线程可以从一个托管线程跳到另一个托管线程。有关详细信息，请参见.NET API System.Runtime.Remoting.Messaging.CallContext。  对于.NET标准，它使用AsyncLocal而不是CallContext。  The logical thread context is visible to a logical thread. Logical threads can jump from one managed thread to another. For more details see the .NET API System.Runtime.Remoting.Messaging  .CallContext. For .NET Standard this uses AsyncLocal rather than CallContext. |
| Event | log4net.Core.LoggingEvent | 每个事件都捕获生成事件时的当前上下文状态。可以在事件本身上设置上下文数据。此上下文仅对生成事件本身的代码可见。  Each event captures the current contextual state at the time the event is generated. Contextual data can be set on the event itself. This context is only visible to the code generating the event itself. |

**2.上下文属性(**Context Properties**)**

log4net上下文存储属性，即名称-值对。名称是一个字符串，值是任何对象。属性可以设置如下：

The log4net contexts store properties, i.e. name value pairs. The name is a string the value is any object. A property can be set as follows:

|  |
| --- |
| log4net.GlobalContext.Properties["name"] = value; |

如果在多个上下文作用域中设置了具有相同名称的属性，则最窄作用域中的值（在上面的列表中较低）将隐藏其他值。If properties with the same name are set in more than one context scope then the value in the narrowest scope (lower down in the list above) will hide the other values.

属性值作为对象存储在LoggingEvent中。PatternLayout支持使用%property{name}语法呈现命名属性的值。通过将该值传递给log4net.ObjectRenderer.RendererMap将其转换为字符串，该log4net.ObjectRenderer.RendererMap将定位该值类型的任何自定义呈现程序。自定义类型的默认行为是调用对象的ToString（）方法。The property values are stored as objects within the LoggingEvent. The PatternLayout supports rendering the value of a named property using the %property{name} syntax. The value is converted to a string by passing it to the log4net.ObjectRenderer.RendererMap which will locate any custom renderer for the value type. The default behavior for custom types is to call the object's ToString() method.

**3.活动属性值(**Active Property Value**)**

活动属性值是指其值随时间变化的值。

例如，假设一个自定义类型实现了ToString（）方法，以返回运行时垃圾收集器分配的字节数。

An active property value is one who's value changes over time.

For example, imagine a custom type that implemented the ToString() method to return the number of bytes allocated by the runtime garbage collector.

|  |
| --- |
| public class GCAllocatedBytesHelper  {  public override string ToString()  {  return GC.GetTotalMemory(true).ToString();  }  } |

在应用程序启动期间，可以将此类型的实例添加到log4net.GlobalContext中：

An instance of this type can be added to the log4net.GlobalContext during application startup:

|  |
| --- |
| log4net.GlobalContext.Properties["GCAllocatedBytes"] = new GCAllocatedBytesHelper(); |

在上下文中设置此属性后，所有后续日志记录事件都将具有名为GCAllocatedBytes的属性。属性的值将是GCAllocatedBytesHelper类型的实例。当通过调用ToString方法将此值呈现为字符串时，垃圾收集器分配的当前字节数将返回并包含在输出中。 Once this property is set in the context all subsequent logging events will have a property called *GCAllocatedBytes*. The value of the property will be an instance of the GCAllocatedBytesHelper type. When this value is rendered to a string by calling the ToString method the current number of bytes allocated by the garbage collector will be returned and included in the output.

**4.上下文堆栈(Context Stacks)**

有时，简单的键值对并不是获取上下文信息的最方便的方式。信息堆栈是存储数据的一种非常方便的方式，特别是当我们的应用程序趋向于基于堆栈时。Sometimes simple key value pairs are not the most convenient way of capturing contextual information. A stack of information is a very convenient way of storing data especially as our applications tend to be stack based.

ThreadContext和LogicalThreadContext还支持在堆栈中存储上下文数据。堆栈存储在context属性中，因此堆栈有名称，同一上下文中可以存在多个堆栈。在范围较窄的上下文中设置的属性值将重写在范围较广的上下文中设置了相同属性名的堆栈。The ThreadContext and LogicalThreadContext also support storing contextual data in a stack. The stack is stored in context property, therefore stacks have names and more than one stack can exist in the same context. A property value set in a narrower context would override a stack with the same property name set in a wider scoped context.

堆栈支持Push和Pop方法。随着更多的上下文数据被推送到堆栈上，堆栈会增长。在呈现堆栈时，所有推送到堆栈上的数据都将与最新的数据一起输出到字符串的右端。The stack supports Push and Pop methods. As more contextual data is pushed onto the stack the stack grows. When the stack is rendered all the data pushed onto the stack is output with the most recent data to the right hand end of the string.

由于堆栈只是存储在上下文属性中的一个对象，因此它也使用相同的PatternLayout语法进行呈现：%property{name}。其中name是堆栈的名称。As the stack is just an object stored in the context properties it is also rendered using the same PatternLayout syntax: %property{name}. Where name is the name of the stack.

调用堆栈的Push和Pop方法必须匹配，以便每个Push都有相应的Pop。Push方法还返回一个IDisposable对象，该对象在释放时将执行所需的pop操作。这允许C使用语法来自动化堆栈管理。Calls the the stack's Push and Pop methods must be matched up so that each push has a corresponding pop. The Push method also returns an IDisposable object that will perform the required pop operation when it is disposed. This allows the C# using syntax to be used to automate the stack management.

|  |
| --- |
| using(log4net.ThreadContext.Stacks["NDC"].Push("context"))  {  log.Info("Message");  } |

信息级日志在其NDC属性中存储了一个堆栈。堆栈中的顶部项是字符串上下文。using语法确保值上下文从块末尾的堆栈中弹出。The INFO level log has a stack stored in its NDC property. The top item in the stack is the string context. The using syntax ensures that the value context is popped off the stack at the end of the block.

建议使用using语法，因为它可以从开发人员中移除一些工作负载，并减少匹配推送和弹出调用时的错误，特别是在可能发生异常的情况下。

The using syntax is recommended because it removes some work load from the developer and reduces errors in matching up the Push and Pop calls, especially when exceptions can occur.

**5.嵌套诊断上下文(Nested Diagnostic Contexts)**

为了与旧版本的log4net兼容，存在NDC（嵌套诊断上下文）。这个helper类实现了一个堆栈，该堆栈存储在名为NDC的线程上下文属性中。The NDC (Nested Diagnostic Context) exists for compatibility with older versions of log4net. This helper class implements a stack which is stored in the thread context property named *NDC*.

**6.映射的诊断上下文(Mapped Diagnostic Contexts)**

MDC（MappedDiagnostic上下文）的存在是为了与旧版本的log4net兼容。这个helper类实现了一个直接映射到线程上下文属性的属性映射。The MDC (MappedDiagnostic Context) exists for compatibility with older versions of log4net. This helper class implements a properties map which is mapped directly through to the thread context properties.

为了说明这一点，让我们举一个web服务向众多客户机交付内容的例子。web服务可以在请求开始时构建NDC，然后再执行其他代码。上下文信息可以是客户机的主机名和请求固有的其他信息，通常是cookie中包含的信息。因此，即使web服务同时为多个客户机服务，由同一代码（即属于同一记录器）启动的日志仍然可以区分，因为每个客户机请求将具有不同的NDC堆栈。与此形成对比的是，将新实例化的记录器传递给客户端请求期间执行的所有代码的复杂性。To illustrate this point, let us take the example of a web service delivering content to numerous clients. The web service can build the NDC at the very beginning of the request before executing other code. The contextual information can be the client's host name and other information inherent to the request, typically information contained in cookies. Hence, even if the web service is serving multiple clients simultaneously, the logs initiated by the same code, i.e. belonging to the same logger, can still be distinguished because each client request will have a different NDC stack. Contrast this with the complexity of passing a freshly instantiated logger to all code exercised during the client's request.

然而，一些复杂的应用程序（如虚拟主机web服务器）必须根据虚拟主机上下文以及发出请求的软件组件进行不同的日志记录。Log4net支持多个日志存储库。这将允许每个虚拟主机拥有自己的记录器层次结构副本。配置多个记录器层次结构超出了本文档的范围。Nevertheless, some sophisticated applications, such as virtual hosting web servers, must log differently depending on the virtual host context and also depending on the software component issuing the request. Log4net supports multiple logger repositories. This would allow each virtual host to possess its own copy of the logger hierarchy. Configuring multiple logger hierarchies is beyond the scope of this document.

**八、仓库 (Repositories)**

**1.日志仓库(Logging Repositories)**

日志仓库被认为是高级功能。对于大多数用户来说，默认行为应该足够了。Logging repositories are considered advanced functionality. The default behavior should be sufficient for most users.

Log4net支持日志仓库。仓库的名称是唯一的。每个仓库都是一个（ILoggerRepository）。多个程序集可以链接到同一个仓库。Log4net supports logging repositories. A repository is uniquely named. Each repository is a (ILoggerRepository). Multiple assemblies can link to the same repository.

默认情况下，每个进程有一个日志仓库（更准确地说，每个AppDomain）。这将扩展加载到流程中的所有程序集，并允许它们共享单个配置。仓库的配置只需要完成一次，通常是在应用程序的入口点，可以通过编程方式完成，也可以使用配置属性。By default there is a single logging repository per process (more precisely per AppDomain). This extends across all assemblies loaded into the process and allows them to all share a single configuration. The configuration of the repository only needs to be done once, typically in the entry point to the application, either programmatically or using a configuration attribute.

可以使用LogManager.CreateRepository方法创建命名日志仓库。可以使用LogManager.GetRepository方法检索的仓库。以这种方式创建的仓库需要以编程方式配置。Named logging repositories can be created using the LogManager.CreateRepository method. The repository for can be retrieved using the LogManager.GetRepository method. A repository created in this way will need to be configured programmatically.

**2.特性(Attributes)**

程序集可以选择使用命名日志仓库，而不是默认仓库。这将程序集的日志记录与应用程序的其余部分完全分离。对于那些希望将log4net用于其组件但不希望要求使用其组件的所有应用程序都知道log4net的组件开发人员来说，这非常有用。这也意味着它们的调试配置与应用程序配置是分开的。程序集应指定RepositoryAttribute以设置其日志存储库。An assembly may choose to utilize a named logging repository rather than the default repository. This completely separates the logging for the assembly from the rest of the application. This can be very useful to component developers that wish to use log4net for their components but do not want to require that all the applications that use their component are aware of log4net. It also means that their debugging configuration is separated from the applications configuration. The assembly should specify the RepositoryAttribute to set its logging repository.

log4net日志存储库可以使用以下程序集级别属性进行配置：

The log4net logging repositories can be configured using the following assembly-level attributes:

**（1）AliasRepositoryAttribute**

指定要用作此程序集的存储库的命名仓库。Specifies a named repository to use as this assembly's repository.

程序集的记录器存储库由其RepositoryAttribute定义，但这可以由在目标程序集之前加载的程序集重写。An assembly's logger repository is defined by its RepositoryAttribute, however this can be overridden by an assembly loaded before the target assembly.

通过使用目标存储库的名称指定此属性，程序集可以为另一个程序集的存储库命名。An assembly can alias another assembly's repository by specifying this attribute with the name of the target repository.

可以根据需要多次使用此属性来为所有必需的仓库命名。This attribute may be used as many times as necessary to alias all the required repositories.

**（2）RepositoryAttribute**

指定程序集的日志仓库。

Specifies the logging repository for the assembly.

程序集映射到日志仓库。此属性控制仓库的配置。Name属性指定此程序集的仓库的名称。RepositoryType属性指定要为程序集创建的仓库对象的类型。如果未指定此属性且未指定名称，则程序集将成为默认共享日志存储库的一部分。Assemblies are mapped to logging repositories. This attribute controls the configuration of the repository. The Name property specifies the name of the repository for this assembly. The RepositoryType property specifies the type of the repository object to create for the assembly. If this attribute is not specified and a Name is not specified then the assembly will be part of the default shared logging repository.

每个程序集只能使用此属性一次。

This attribute may only be used once per assembly.

**第二章 log4net架构**



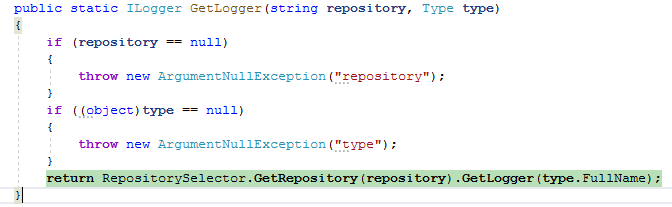
1、仓库(Repositories)是一个Hashtable，因此可以创建多个仓库(Repository)。





2、日志器(loggers)是一个Hashtable，因此一个仓库(Repository)中可以创建一个或多个日志器(logger)。

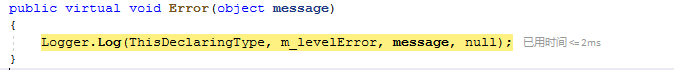




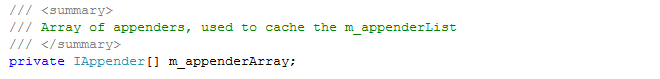
日志器是分层管理的：

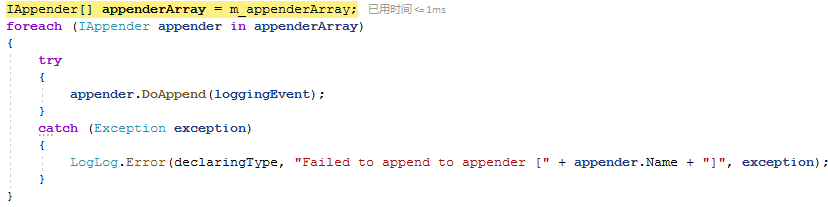


日志记录是按不同级别(Level)输出到输出器(Appender)中的：



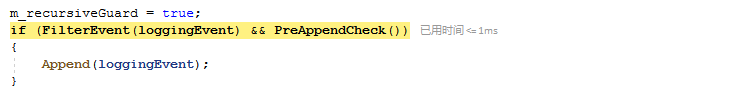
3、日志记录被输出到不同的输出器(Appender)中：



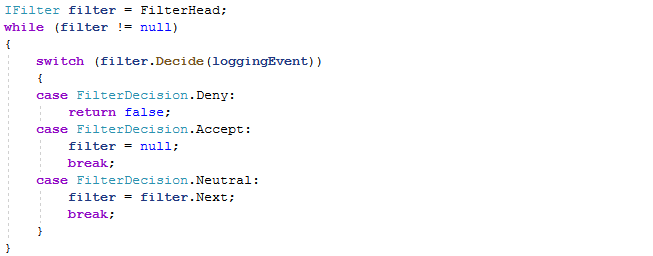


输出器是一个IAppender接口数组，因此每个输出器(Appender)必须实现这个接口。

4、记录在正式输出之前可以根据需要进行筛选(Filter)



筛选器(Filter)是一个链表，因此如果输出记录不满足要求你可以在任意筛选器中进行中断输出：



筛选以后就将记录正式输出到一个或多个的输出器(Appender)中，例如文件夹、控制台、邮件等等。

5、最后根据每个输出器设置的布局(layout)，格式化并输出想要的格式：



格式转换是通过模式转换器进行的：



日志记录完毕。

**第三章 在.net core项目中使用log4net**

**1.如何配置？**

官方对log4net的使用有详细的配置说明，参见：

<http://logging.apache.org/log4net/release/manual/configuration.html>

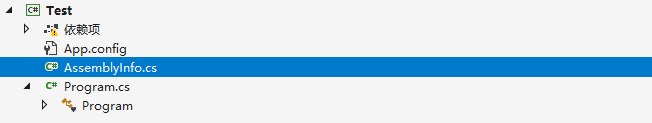
要使用log4net首先需要进行配置，有2种配置方式：嵌入式配置和手工配置。

**2.嵌入配置**

打开Visual Studio 2019新建一个项目：



完成以后，增加AssemblyInfo.cs：

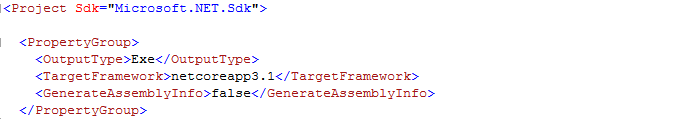


内容如下：

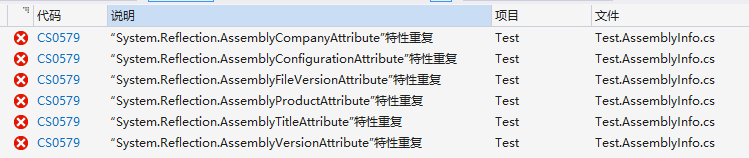
|  |
| --- |
| using System.Reflection;  using System.Runtime.CompilerServices;  using System.Runtime.InteropServices;  // 有关程序集的一般信息由以下  // 控制。更改这些特性值可修改  // 与程序集关联的信息。  [assembly: AssemblyTitle("Test")]  [assembly: AssemblyDescription("")]  [assembly: AssemblyConfiguration("")]  [assembly: AssemblyCompany("")]  [assembly: AssemblyProduct("Test")]  [assembly: AssemblyCopyright("Copyright © 2020")]  [assembly: AssemblyTrademark("")]  [assembly: AssemblyCulture("")]  // 将 ComVisible 设置为 false 会使此程序集中的类型  //对 COM 组件不可见。如果需要从 COM 访问此程序集中的类型  //请将此类型的 ComVisible 特性设置为 true。  [assembly: ComVisible(false)]  // 如果此项目向 COM 公开，则下列 GUID 用于类型库的 ID  [assembly: Guid("50516cd1-c887-47cf-9cd1-a9eb30c0046a")]  // 程序集的版本信息由下列四个值组成:  //  // 主版本  // 次版本  // 生成号  // 修订号  //  //可以指定所有这些值，也可以使用“生成号”和“修订号”的默认值  //通过使用 "\*"，如下所示:  // [assembly: AssemblyVersion("1.0.\*")]  [assembly: AssemblyVersion("1.0.0.0")]  [assembly: AssemblyFileVersion("1.0.0.0")]  [assembly: log4net.Config.XmlConfigurator(ConfigFile = "Configs\\log4net.config", Watch = true)] |

在修改项目文件Test.csproj，增加:

|  |
| --- |
| <GenerateAssemblyInfo>false</GenerateAssemblyInfo> |



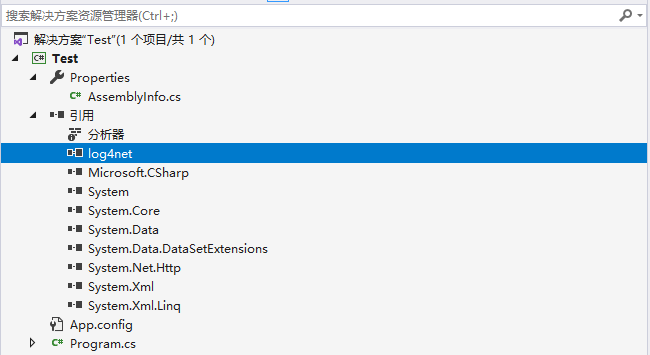
注意：由于.Net Core去掉了AssemblyInfo.cs文件，由编译器自动生成，因此要使用自己的AssemblyInfo.cs文件自定义程序集信息，必须在项目文件(csproj)中禁用自动生成AssemblyInfo.cs文件的功能，否则编译时会报错：



当程序集(dll/exe)启动的时候log4net就会自动查找配置文件”log4net.config”，并监视其变化，源码如下：

|  |
| --- |
| // 自动加载配置  public static void Configure()  {  Configure(LogManager.GetRepository(Assembly.GetCallingAssembly()));  }  // 监控配置文件变化并自动更新配置  private void ConfigureAndWatchHandler\_OnChanged(object source, FileSystemEventArgs e)  {  LogLog.Debug("ConfigureAndWatchHandler: " + e.ChangeType + " [" + m\_configFile.FullName + "]");  m\_timer.Change(500, -1);  } |

现在将”log4net.dll”引入进来，如果没有就用NuGet下载log4net包



并编写”log4net.config”，内容如下：

|  |
| --- |
| <? xml version="1.0"?>  <!--参见: http://logging.apache.org/log4net/release/manual/configuration.html -->  <configuration>  <!--配置段-->  <configSections>  <section name = "log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net"/>  </configSections>  <!--配置日志-->  <log4net>  <root>  <!--设置日志级别, 默认值为DEBUG,  级别由低到高依次为: ALL|DEBUG<INFO<WARN<ERROR<FATAL|OFF.其中:  ALL表示记录所有日志;  OFF表示不记录日志, 即关闭日志记录;  其它则按级别记录，例如级别设置成WARN，则低于WARN级别的INFO和DEBUG日志将不会被记录, 其它依次类推.  -->  <level value = "ALL" />  < !--定义输出源, 可以定义多个输出源,  例如你可以按日志级别或业务模块等定义多个输出源, 对日志进行分类记录.  下面仅定义了一个输出源 -->  <appender-ref ref="RollingFileAppender"/>  </root>  <!--输出源设置-->  <appender name = "RollingFileAppender" type="log4net.Appender.RollingFileAppender">  <!--日志文件名,  如果RollingStyle为Composite或Date, 则这里一般设置成目录名, 文件名在DatePattern里设置;  如果RollingStyle为其他滚动方式, 则必须设置成文件名 -->  <file value = "log\\" />  < !--是否是静态的日志文件名, 即固定的日志文件名称,  如果你想要动态生成日志文件名，例如将RollingStyle设置成Date或Composite, 以让系统自动生成日志文件的名称,  那么你必须将staticLogFileName设置成false -->  <staticLogFileName value = "false" />  < !--日志滚动方式，可设置成:  Size(按文件), 此时你应该将file的值设置成一个固定的名称, 例如: test.log 或 log\\test.log 或 c:\\log\\test.log;  Date(按日期), 此时你应该将file的值设置成一个目录或者空字符串, 如果设置成空字符串, 系统将把日志记录在当前应用程序所在的目录中;  Composite(按日期及文件)，默认为Composite -->  <RollingStyle value = "Composite" />  < !--日志文件名格式,  当RollingStyle为Composite或Date, 在此处设置日志文件名的格式时,  固定不变的部分用单引号括起来, 其它部分则设置成日期格式 -->  <datePattern value = "yyyyMMdd'.log'" />  < !--日志记录是否追加到文件,  默认为true, 表示将记录追加到文件尾部;  flase, 表示覆盖已有的日志文件记录 -->  <appendToFile value = "true" />  < !--单个日志文件的最大尺寸,  可用的单位有: KB|MB|GB, 默认为字节(不带单位) -->  <maximumFileSize value = "2MB" />  < !--每日最多记录的日志文件个数-- >  < maxSizeRollBackups value="10"/>    <!--单个日志文件超限后日志备份方式, 默认值为-1,  当日志文件超过MaximumFileSize大小时，系统根据CountDirection的值来备份日志文件:  (1)当此值设置成>-1时, 则根据file里指定的文件名依次按0,1,2...递增创建日志备份, 直到数量等于MaxSizeRollBackups参数值为止，  以后的日志记录则会根据maximumFileSize循环写入file, 当filesize>maximumFileSize, 进行一次新的循环写入时, 会将file记录写入备份日志, 并对备份日志进行重新编号;  (2)当此值设置成<=-1时, 则根据file里指定的文件名依次按0,1,2...递增创建日志备份, 直到数量等于MaxSizeRollBackups参数值为止，  以后的日志记录则会根据maximumFileSize循环写入file, 当filesize>maximumFileSize, 进行一次新的循环写入时, 不会将file记录写入备份日志, 即备份日志被固化不受影响 -->  <countDirection value = "-1" />  < !--日志样式:  %m(message) : 输出的日志消息;  %n(newline) : 换行;  %d(datetime) : 输出当前语句运行的时刻;  %r(runtime) : 输出程序从运行到执行到当前语句时消耗的毫秒数;  %t(threadid) : 当前语句所在的线程ID;  %p(priority) : 日志的当前日志级别;  %c(class): 当前日志对象的名称;  %L: 输出语句所在的行号;  %F: 输出语句所在的文件名;  %-10: 表示最小长度为10，如果不够，则用空格填充; -->  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="--------时间: %d, 耗时: %r(ms), 类型: %-5p --------%n对象: %c%n详情: %m%n%n %-1000"/>  </layout>    <!--日志锁定模式, 下面的设置是: 多线程时采用最小锁定 -->  <lockingModel type = "log4net.Appender.FileAppender+MinimalLock" />  </ appender >  </ log4net >  </ configuration > |

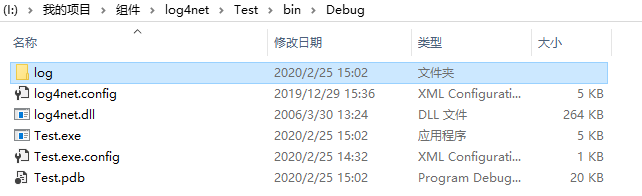
后面根据官方文档详细讲解各段的含义。编写以后放在程序集(dll/exe)所在目录，注意到前面的代码了吗？

Configure(LogManager.GetRepository(Assembly.GetCallingAssembly())); 所以要放在程序集所在目录。

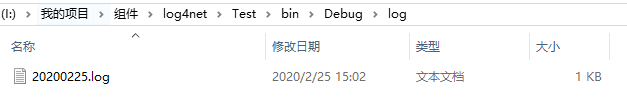
现在在Program.cs中编写如下代码：

|  |
| --- |
| using System;  using log4net;  namespace Test  {  class Program  {  // 获得一个日志器，捕捉当前Program类产生的错误  private static readonly ILog log = LogManager.GetLogger(typeof(Program));  static void Main(string[] args)  {  try  {  throw new Exception("你没有访问系统的权限"); // 模拟一个异常  }  catch(Exception e)  {  WriteLog(e);  }  }  /// <summary>  /// 写错误日志  /// </summary>  /// <param name="e"></param>  private static void WriteLog(Exception e)  {  if (log.IsErrorEnabled)  {  log.Error(e);  }  }  }  } |

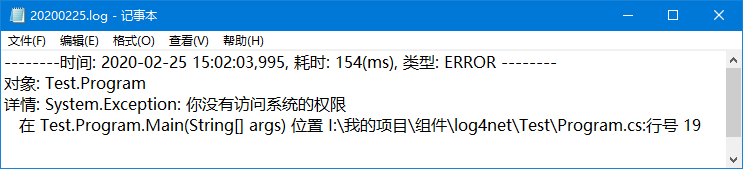
按F5运行，然后在当前程序集所在目录是否已经生成了日志。此时我们可以看到程序集(Text.exe)所在目录多了一个”log”文件夹。



打开里面按照配置文件的设置，生成了一个名为：20200225.log的日志文件：



内容为：



**3.手工配置**

手工配置就是不让log4net自动加载配置文件，因为配置文件不可能总是放在程序集所在目录。

首先删除掉前面在AssemblyInfo.cs文件中增加的配置：

|  |
| --- |
| ~~[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config", Watch = true)]~~ |

打开App.config配置文件，在其中增加：

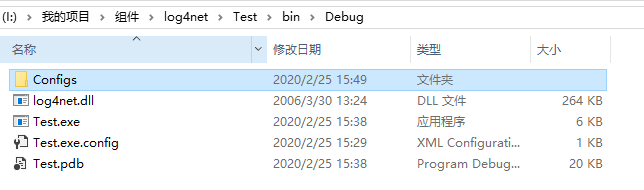
|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <configuration>  <startup>  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.7.2" />  </startup>  <appSettings>  <add key="log4net" value="Configs/log4net.config"/>  <add key="logEnabled" value="1"/>  </appSettings>  </configuration> |

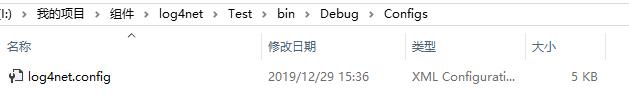
这里不是硬性配置，只是为了灵活方便，能从App配置文件中读取信息。log4net的示例代码可以参考官方：<http://logging.apache.org/log4net/release/manual/configuration.html>

修改Program.cs代码:

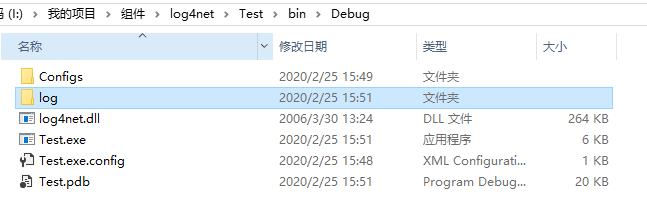
|  |
| --- |
| using System;  using System.Configuration;  using log4net;  namespace Test  {  class Program  {  // 获得一个日志器，捕捉当前Program类产生的错误  private static readonly ILog log = LogManager.GetLogger(typeof(Program));  static void Main(string[] args)  {  // 读取log4net配置文件  var logConfig = ConfigurationManager.AppSettings["log4net"];  // 读取appSettings段的logEnabled，检查是否允许开启日志  var logEnabled = ConfigurationManager.AppSettings["logEnabled"];  if (logEnabled == "1") // 开启日志  {  // 取得应用程序集(exe)所在的目录  var appPath = AppDomain.CurrentDomain.BaseDirectory;    // 日志文件  var logPath = appPath + logConfig;  // 获得日志配置文件  var configFile = new System.IO.FileInfo(logPath);  // 配置log4net  log4net.Config.XmlConfigurator.Configure(configFile);  }  try  {  throw new Exception("请重新登录系统"); // 模拟一个异常  }  catch(Exception e)  {  WriteLog(e);  }  }  /// <summary>  /// 写错误日志  /// </summary>  /// <param name="e"></param>  private static void WriteLog(Exception e)  {  if (log.IsErrorEnabled)  {  log.Error(e);  }  }  }  } |

现在，把log4net.config文件移动到Configs目录

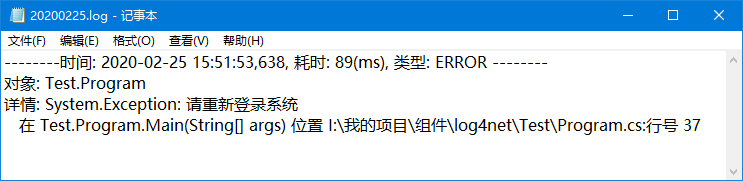




按F5运行，此时生成了新的日志：







log4的常规应用就这么简便。主要是要设置好配置文件，规划好日志的应用方式，总是能够捕捉到想要的错误信息。

**4.默认配置**

默认配置已经在BasicConfigurator.cs中进行了定义，参见测试源代码。使用默认的配置可以摆脱log4net.config配置文件的束缚。

**第四章 log4net配置参数详解**

下面根据前面的配置文件，结合官方文档讲解各个字段的含义:

|  |
| --- |
| <? xml version="1.0"?>  <!--参见: http://logging.apache.org/log4net/release/manual/configuration.html -->  <configuration>  <!--配置段-->  <configSections>  <section name = "log4net" type="log4net.Config.Log4NetConfigurationSectionHandler,log4net"/>  </configSections>  <!--配置日志-->  <log4net>  <root>  <!--设置日志级别, 默认值为DEBUG,  级别由低到高依次为: ALL|DEBUG<INFO<WARN<ERROR<FATAL|OFF.其中:  ALL表示记录所有日志;  OFF表示不记录日志, 即关闭日志记录;  其它则按级别记录，例如级别设置成WARN，则低于WARN级别的INFO和DEBUG日志将不会被记录, 其它依次类推.  -->  <level value = "ALL" />  < !--定义输出源, 可以定义多个输出源,  例如你可以按日志级别或业务模块等定义多个输出源, 对日志进行分类记录.  下面仅定义了一个输出源 -->  <appender-ref ref="RollingFileAppender"/>  </root>  <!--输出源设置-->  <appender name = "RollingFileAppender" type="log4net.Appender.RollingFileAppender">  <!--日志文件名,  如果RollingStyle为Composite或Date, 则这里一般设置成目录名, 文件名在DatePattern里设置;  如果RollingStyle为其他滚动方式, 则必须设置成文件名 -->  <file value = "log\\" />  < !--是否是静态的日志文件名, 即固定的日志文件名称,  如果你想要动态生成日志文件名，例如将RollingStyle设置成Date或Composite, 以让系统自动生成日志文件的名称,  那么你必须将staticLogFileName设置成false -->  <staticLogFileName value = "false" />  < !--日志滚动方式，可设置成:  Size(按文件), 此时你应该将file的值设置成一个固定的名称, 例如: test.log 或 log\\test.log 或 c:\\log\\test.log;  Date(按日期), 此时你应该将file的值设置成一个目录或者空字符串, 如果设置成空字符串, 系统将把日志记录在当前应用程序所在的目录中;  Composite(按日期及文件)，默认为Composite -->  <RollingStyle value = "Composite" />  < !--日志文件名格式,  当RollingStyle为Composite或Date, 在此处设置日志文件名的格式时,  固定不变的部分用单引号括起来, 其它部分则设置成日期格式 -->  <datePattern value = "yyyyMMdd'.log'" />  < !--日志记录是否追加到文件,  默认为true, 表示将记录追加到文件尾部;  flase, 表示覆盖已有的日志文件记录 -->  <appendToFile value = "true" />  < !--单个日志文件的最大尺寸,  可用的单位有: KB|MB|GB, 默认为字节(不带单位) -->  <maximumFileSize value = "2MB" />  < !--每日最多记录的日志文件个数-- >  < maxSizeRollBackups value="10"/>    <!--单个日志文件超限后日志备份方式, 默认值为-1,  当日志文件超过MaximumFileSize大小时，系统根据CountDirection的值来备份日志文件:  (1)当此值设置成>-1时, 则根据file里指定的文件名依次按0,1,2...递增创建日志备份, 直到数量等于MaxSizeRollBackups参数值为止，  以后的日志记录则会根据maximumFileSize循环写入file, 当filesize>maximumFileSize, 进行一次新的循环写入时, 会将file记录写入备份日志, 并对备份日志进行重新编号;  (2)当此值设置成<=-1时, 则根据file里指定的文件名依次按0,1,2...递增创建日志备份, 直到数量等于MaxSizeRollBackups参数值为止，  以后的日志记录则会根据maximumFileSize循环写入file, 当filesize>maximumFileSize, 进行一次新的循环写入时, 不会将file记录写入备份日志, 即备份日志被固化不受影响 -->  <countDirection value = "-1" />  < !--日志样式:  %m(message) : 输出的日志消息;  %n(newline) : 换行;  %d(datetime) : 输出当前语句运行的时刻;  %r(runtime) : 输出程序从运行到执行到当前语句时消耗的毫秒数;  %t(threadid) : 当前语句所在的线程ID;  %p(priority) : 日志的当前日志级别;  %c(class): 当前日志对象的名称;  %L: 输出语句所在的行号;  %F: 输出语句所在的文件名;  %-10: 表示最小长度为10，如果不够，则用空格填充; -->  <layout type = "log4net.Layout.PatternLayout" >  < conversionPattern value="--------时间: %d, 耗时: %r(ms), 类型: %-5p --------%n对象: %c%n详情: %m%n%n %-1000"/>  </layout>    <!--日志锁定模式, 下面的设置是: 多线程时采用最小锁定 -->  <lockingModel type = "log4net.Appender.FileAppender+MinimalLock" />  </ appender >  </ log4net >  </ configuration > |

log4net详细的配置可参考官方网站：

<http://logging.apache.org/log4net/release/manual/configuration.html>