摩诘
我思故我在 常辨而常新

Log4Net使用指南

声明：本文内容主要译自Nauman Leghari的Using log4net，亦加入了个人的一点心得(节3.1.4)。
请在这里下载示例代码

1 简介

1.1 Log4net的优点：

几乎所有的大型应用都会有自己的用于跟踪调试的API。因为一旦程序被部署以后，就不太可能再利用专门的调试工具了。然而一个管理员可能需要有一套强大的日志系统来诊断和修复配置上的问题。

经验表明，日志记录往往是软件开发周期中的重要组成部分。它具有以下几个优点：它可以提供应用程序运行时的精确环境，可供开发人员尽快找到应用程序中的Bug；一旦在程序中加入了Log 输出代码，程序运行过程中就能生成并输出日志信息而无需人工干预。另外，日志信息可以输出到不同的地方（控制台，文件等）以备以后研究之用。

Log4net就是为这样一个目的设计的，用于.NET开发环境的日志记录包。

1.2 Log4net的安装：

用户可以从http://logging.apache.org/log4net/下载log4net的源代码。解压软件包后，在解压的src目录下将log4net.sln载入Visual Studio .NET，编译后可以得到log4net.dll。用户要在自己的程序里加入日志功能，只需将log4net.dll引入工程即可。

2 Log4net的结构

log4net 有四种主要的组件，分别是Logger（记录器），Repository（库），Appender（附着器）以及 Layout（布局）.

2.1 Logger

2.1.1 Logger接口

Logger是应用程序需要交互的主要组件，它用来产生日志消息。产生的日志消息并不直接显示，还要预先经过Layout的格式化处理后才会输出。

Logger提供了多种方式来记录一个日志消息，你可以在你的应用程序里创建多个Logger，每个实例化的Logger对象都被log4net框架作为命名实体(named entity)来维护。这意味着为了重用Logger对象，你不必将它在不同的类或对象间传递，只需要用它的名字为参数调用就可以了。log4net框架使用继承体系，继承体系类似于.NET中的名字空间。也就是说，如果有两个logger,分别被定义为a.b.c和a.b，那么我们说a.b是a.b.c的祖先。每一个logger都继承了祖先的属性

Log4net框架定义了一个ILog接口，所有的logger类都必须实现这个接口。如果你想实现一个自定义的logger，你必须首先实现这个接口。

你可以参考在/extension目录下的几个例子。

ILog接口的定义如下：

```
public interface ILog
{
  void Debug(object message);
  void Info(object message);
  void Warn(object message);
  void Error(object message);
  void Fatal(object message);

//以上的每一个方法都有一个重载的方法，用来支持异常处理。
//每一个重载方法都如下所示，有一个异常类型的附加参数。
  void Debug(object message, Exception ex);
  // ...

//Boolean 属性用来检查Logger的日志级别
//（我们马上会在后面看到日志级别）
  bool isDebugEnabled;
  bool isInfoEnabled;
  //... 其他方法对应的Boolean属性
}
```

Log4net框架定义了一个叫做LogManager的类，用来管理所有的logger对象。它有一个GetLogger()静态方法，用我们提供的名字参数来检索已经存在的Logger对象。如果框架里不存在该Logger对象，它也会为我们创建一个Logger对象。代码如下所示：

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

通常来说，我们会以类（class）的类型（type）为参数来调用GetLogger()，以便跟踪我们正在进行日志记录的类。传递的类(class)的类型(type)可以用typeof(Classname)方法来获得，或者可以用如下的反射方法来获得：

```
System.Reflection.MethodBase.GetCurrentMethod().DeclaringType
```

尽管符号长了一些，但是后者可以用于一些场合，比如获取调用方法的类(class)的类型(type)。

2.1.2 日志的级别

正如你在ILog的接口中看到的一样，有五种不同的方法可以跟踪一个应用程序。事实上，这五种方法是运作在Logger对象设置的不同日志优先级别上。这几种不同的级别是作为常量定义在log4net.spi.Level类中。你可以在程序中使用任何一种方法。但是在最后的发布中你也许不想

3. 深入浅出之正则表达式（二）(64774)
4. 自动向网页Post信息并提取返回的信息(31824)
5. 开发Visual Studio风格的用户界面 - - MagicLibrary使用指南(18712)

评论排行榜
1. Log4Net使用指南(123)
2. 深入浅出之正则表达式（一）(78)
3. 自动向网页Post信息并提取返回的信息(46)
4. 深入浅出之正则表达式（二）(40)
5. 在.NET环境中实现每日构建(Daily Build)--NAnt篇(38)

推荐排行榜
1. Log4Net使用指南(93)
2. 深入浅出之正则表达式（一）(45)
3. 深入浅出之正则表达式（二）(28)
4. Netron开发快速上手（一）：GraphControl，Shape，Connector和Connection(8)
5. FileSystemWatcher事件多次触发的解决方法(5)

让所有的代码来浪费你的CPU周期，因此，框架提供了7种级别和相应的Boolean属性来控制日志记录的类型。

Level有以下几种取值

级别	允许的方法	**Boolean属性**	优先级别
OFF			**Highest**
FATAL	**void Fatal(...);**	**bool IsFatalEnabled;**	
RROR	**void Error(...);**	**bool IsErrorEnabled;**	
WARN	**void Warn(...);**	**bool IsWarnEnabled;**	
INFO	**void Info(...);**	**bool IsInfoEnabled;**	
DEBUG	**void Debug(...);**	**bool IsDebugEnabled;**	
ALL			**Lowest**

表1 Logger的日志级别

在log4net框架里，通过设置配置文件，每个日志对象都被分配了一个日志优先级别。如果没有给一个日志对象显式地分配一个级别，那么该对象会试图从他的祖先继承一个级别值。

ILog接口的每个方法都有一个预先定义好了的级别值。正如你在表1看到的，ILog的Inof()方法具有INFO级别。同样的，以此类推，Error()方法具有ERROR级别。当我们使用以上的任何一种方法时，log4net框架会检查日志对象logger的级别和方法的级别。只有当方法的级别高于日志级别时，日志请求才会被接受并执行。

举例说明，当你创建了一个日志对象，并且把他的级别设置为INFO。于是框架会设置日志的每个Boolean属性。当你调用相应的日志方法时，框架会检查相应的Boolean属性，以决定该方法能不能执行。如下的代码：

```
Logger.Info("message");

Logger.Debug("message");

Logger.Warn("message");
```

对于第一种方法，Info()的级别等与日志的级别（INFO），因此日志请求会被传递，我们可以得到输出结果”message”。

对于第二种方法，Debug()的级别低于日志对象logger的日志级别(INFO)，因此，日志请求被拒绝了，我们得不到任何输出。同样的，针对第三行语句，我们可以很容易得出结论。

在表1中有两个特殊的级别：ALL和OFF。ALL表示允许所有的日志请求。OFF是拒绝所有的请求。

你也可以显式地检查Logger对象的Boolean属性，如下所示：

```
if (logger.IsDebugEnabled)
{
  Logger.Debug("message");
}
```

2.2 Repository

Repository主要用于负责日志对象组织结构的维护。在log4net的以前版本中，框架仅支持分等级的组织结构(hierarchical organization)。这种等级结构本质上是库的一个实现，并且定义在log4net.Repository.Hierarchy 名字空间中。要实现一个Repository，需要实现log4net.Repository.ILoggerRepository 接口。但是通常并不是直接实现该接口，而是以log4net.Repository.LoggerRepositorySkeleton为基类继承。体系库 (hierarchical repository)则由log4net.Repository.Hierarchy.Hierarchy类实现。

如果你是个log4net框架的使用者，而非扩展者，那么你几乎不会在你的代码里用到Repository的类。相反的，你需要用到LogManager类来自动管理库和日志对象。

2.3 Appender

一个好的日志框架应该能够产生多目的地的输出。比如说输出到控制台或保存到一个日志文件。log4net 能够很好的满足这些要求。它使用一个叫做Appender的组件来定义输出介质。正如名字所示，这些组件把它们附加到Logger日志组件上并将输出传递到输出流中。你可以把多个Appender组件附加到一个日志对象上。 Log4net框架提供了几个Appender组件。关于log4net提供的Appender组件的完整列表可以在log4net框架的帮助手册中找到。有了这些现成的Appender组件，一般来说你没有必要再自己编写了。但是如果你愿意，可以从log4net.Appender.AppenderSkeleton类继承。

2.4 Appender Filters

一个Appender 对象缺省地将所有的日志事件传递到输出流。Appender的过滤器(Appender Filters) 可以按照不同的标准过滤日志事件。在log4net.Filter的名字空间下已经有几个预定义的过滤器。使用这些过滤器，你可以按照日志级别范围过滤日志事件，或者按照某个特殊的字符串进行过滤。你可以在API的帮助文件中发现更多关于过滤器的信息。

2.5 Layout

Layout 组件用于向用户显示最后经过格式化的输出信息。输出信息可以以多种格式显示，主要依赖于我们采用的Layout组件类型。可以是线性的或一个XML文件。Layout组件和一个Appender组件一起工作。API帮助手册中有关于不同Layout组件的列表。一个Appender对象，只能对应一个Layout对象。要实现你自己的Layout类，你需要从log4net.Layout.LayoutSkeleton类继承，它实现了ILayout接口。

3 在程序中使用log4net

在开始对你的程序进行日志记录前，需要先启动log4net引擎。这意味着你需要先配置前面提到的三种组件。你可以用两种方法来设定配置：在单独的文件中设定配置或在代码中定义配置。

因为下面几种原因，推荐在一个单独的文件中定义配置：

- 你不需要重新编译源代码就能改变配置；
- 你可以在程序正运行的时候就改变配置。这一点在一些WEB程序和远程过程调用的程序中有时很重要；

考虑到第一种方法的重要性，我们先看看怎样在文件中设定配置信息。

3.1 定义配置文件

配置信息可以放在如下几种形式文件的一种中。

在程序的配置文件里，如AssemblyName.config 或web.config.

在你自己的文件里。文件名可以是任何你想要的名字，如AppName.exe.xyz等.

log4net框架会在相对于AppDomain.CurrentDomain.BaseDirectory属性定义的目录路径下查找配置文件。框架在配置文件里要查找的唯一标识是<log4net>标签。一个完整的配置文件的例子如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net"
      type="log4net.Config.Log4NetConfigurationSectionHandler,
          log4net-net-1.0"
    />
  </configSections>

  <log4net>

    <root>
      <level value="WARN" />
      <appender-ref ref="LogFileAppender" />
      <appender-ref ref="ConsoleAppender" />
    </root>

    <logger name="testApp.Logging">
      <level value="DEBUG"/>
```

```
    </logger>

    <appender name="LogFileAppender"
            type="log4net.Appender.FileAppender" >
      <param name="File" value="log-file.txt" />
      <param name="AppendToFile" value="true" />

      <layout type="log4net.Layout.PatternLayout">
        <param name="Header" value="[Header]\r\n"/>
        <param name="Footer" value="[Footer]\r\n"/>
        <param name="ConversionPattern"
          value="%d [%t] %-5p %c [%x]  - %m%n"
        />
      </layout>

      <filter type="log4net.Filter.LevelRangeFilter">
        <param name="LevelMin" value="DEBUG" />
        <param name="LevelMax" value="WARN" />
      </filter>
    </appender>

    <appender name="ConsoleAppender"
            type="log4net.Appender.ConsoleAppender" >
      <layout type="log4net.Layout.PatternLayout">
        <param name="ConversionPattern"
          value="%d [%t] %-5p %c [%x] - %m%n"
        />
      </layout>
    </appender>

  </log4net>
</configuration>
```

你可以直接将上面的文本拷贝到任何程序中使用，但是最好还是能够理解配置文件是怎样构成的。 只有当你需要在应用程序配置文件中使用log4net配置时，才需要在<configSection>标签中加入<section>配置节点入口。对于其他的单独文件，只有<log4net>标签内的文本才是必需的，这些标签的顺序并不是固定的。下面我们依次讲解各个标签内文本的含义：

3.1.1 <root>

```
<root>
  <level value="WARN" />
  <appender-ref ref="LogFileAppender" />
  <appender-ref ref="ConsoleAppender" />
</root>
```

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

```
<logger name="testApp.Logging" additivity="false">
</logger>
```

Additivity的值缺省是true.

3.1.2 <Logger>

```
<logger name="testApp.Logging">
  <level value="DEBUG"/>
</logger>
```

<logger> 元素预定义了一个具体日志对象的设置。然后通过调用LogManager.GetLogger(“testAPP.Logging”)函数，你可以检索具有该名字的日志。如果LogManager.GetLogger(…)打开的不是预定义的日志对象，则该日志对象会继承根日志对象的属性。知道了这一点，我们可以说，其实<logger>标签并不是必须的。

3.1.3 <appender>

```
<appender name="LogFileAppender"
        type="log4net.Appender.FileAppender" >
  <param name="File" value="log-file.txt" />
  <param name="AppendToFile" value="true" />
```

```
<layout type="log4net.Layout.PatternLayout">
  <param name="Header" value="[Header]\r\n" />
  <param name="Footer" value="[Footer]\r\n"/>
  <param name="ConversionPattern"
    value="%d [%t] %-5p %c - %m%n"
  />
</layout>
<filter type="log4net.Filter.LevelRangeFilter">
  <param name="LevelMin" value="DEBUG" />
  <param name="LevelMax" value="WARN" />
</filter>
</appender>
```

在<root>标签或单个的<logger>标签里的Appender对象可以用<appender>标签定义。<appender>标签的基本形式如上面所示。它定义了appender的名字和类型。 另外比较重要的是<appender>标签内部的其他标签。不同的appender有不同的<param>标签。在这里，为了使用FileAppender,你需要一个文件名作为参数。另外还需要一个在<appender>标签内部定义一个Layout对象。Layout对象定义在它自己的<layout>标签内。<layout>标签的type属性定义了Layout的类型(在本例里是PatternLayout)，同时也确定了需要提供的参数值。Header和Footer标签提供了一个日志会话(logging session)开始和结束时输出的文字。有关每种appender的具体配置的例子，可以在log4net\doc\manual\example-config-appender.html中得到。

3.1.4 log4net.Layout.PatternLayout中的转换模式 (ConversionPattern)

%m(message):输出的日志消息，如ILog.Debug(…)输出的一条消息

%n(new line):换行

%d(datetime):输出当前语句运行的时刻

%r(run time):输出程序从运行到执行到当前语句时消耗的毫秒数

%t(thread id):当前语句所在的线程ID

%p(priority): 日志的当前优先级别，即DEBUG、INFO、WARN…等

%c(class):当前日志对象的名称，例如：

模式字符串为：%-10c -%m%n

代码为：

```
ILog log=LogManager.GetLogger("Exam.Log");
log.Debug("Hello");
```

则输出为下面的形式：

Exam.Log - Hello

%L：输出语句所在的行号

%F：输出语句所在的文件名

%-数字：表示该项的最小长度，如果不够，则用空格填充

例如，转换模式为%r [%t]%-5p %c - %m%n 的 PatternLayout 将生成类似于以下内容的输出：

176 [main] INFO org.foo.Bar - Located nearest gas station.

3.1.5 <filter>

最后，让我们看看在Appender元素里的<filter>标签。它定义了应用到Appender对象的过滤器。本例中，我们使用了LevelRangeFilter过滤器,它可以只记录LevelMin和LevelMax参数指定的日志级别之间的日志事件。可以在一个Appender上定义多个过滤器（Filter）,这些过滤器将会按照它们定义的顺序对日志事件进行过滤。其他过滤器的有关信息可以在log4net的SDK文档中找到。

3.2 使用配置文件

3.2.1 关联配置文件

当我们创建了上面的配置文件后，我们接下来需要把它和我们的应用联系起来。缺省的，每个独立的可执行程序集都会定义它自己的配置。log4net框架使用 log4net.Config.DOMConfiguratorAttribute在程序集的级别上定义配置文件。

例如：可以在项目的AssemblyInfo.cs文件里添加以下的语句

[assembly:log4net.Config.DOMConfigurator(ConfigFile="filename",

ConfigFileExtension="ext",Watch=true/false)]

- **ConfigFile:**指出了我们的配置文件的路径及文件名，包括扩展名。
- **ConfigFileExtension:**如果我们对被编译程序的程序集使用了不同的文件扩展名，那么我们需要定义这个属性，缺省的，程序集的配置文件扩展名为”config”。
- **Watch (Boolean属性):** log4net框架用这个属性来确定是否需要在运行时监视文件的改变。如果这个属性为true,那么FileSystemWatcher将会被用来监视文件的改变，重命名，删除等事件。

其中：ConfigFile和ConfigFileExtension属性不能同时使用，ConfigFile指出了配置文件的名字，例如，ConfigFile=”Config.txt”

ConfigFileExtension则是指明了和可执行程序集同名的配置文件的扩展名，例如，应用程序的名称是”test.exe”,ConfigFileExtension=”txt”,则配置文件就应该是”test.exe.txt” ；

也可以不带参数应用DOMConfiguratio():

```
[assembly: log4net.Config.DOMConfigurator()]
```

也可以在程序代码中用DOMConfigurator类打开配置文件。类的构造函数需要一个FileInfo对象作参数，以指出要打开的配置文件名。 这个方法和前面在程序集里设置属性打开一个配置文件的效果是一样的。

```
log4net.Config.DOMConfigurator.Configure(
  new FileInfo("TestLogger.Exe.Config"));
```

DOMConfigurator 类还有一个方法ConfigureAndWatch(..), 用来配置框架并检测文件的变化。

以上的步骤总结了和配置相关的各个方面，下面我们将分两步来使用logger对象。

3.2.2 创建或获取日志对象

日志对象会使用在配置文件里定义的属性。如果某个日志对象没有事先在配置文件里定义，那么框架会根据继承结构获取祖先节点的属性，最终的，会从根日志获取属性。如下所示：

```
Log4net.ILog log =
Log4net.LogManager.GetLogger("MyLogger");
```

3.2.3 输出日志信息

可以使用ILog的几种方法输出日志信息。你也可以在调用某方法前先检查IsXXXEnabled布尔变量，再决定是否调用输出日志信息的函数，这样可以提高程序的性能。因为框架在调用如ILog.Debug(…)这样的函数时，也会先判断是否满足Level日志级别条件。

```
if (log.IsDebugEnabled) log.Debug("message");
if (log.IsInfoEnabled) log.Info("message);
```

3.3 在程序中配置log4net

除了前面讲的用一个配置文件来配置log4net以外，还可以在程序中用代码来配置log4net框架。如下面的例子:

```
// 和PatternLayout一起使用FileAppender
log4net.Config.BasicConfigurator.Configure(
  new log4net.Appender.FileAppender(
    new log4net.Layout.PatternLayout("%d
      [%t]%-5p %c [%x] - %m%n"),"testfile.log"));

// using a FileAppender with an XMLLayout
log4net.Config.BasicConfigurator.Configure(
  new log4net.Appender.FileAppender(
    new log4net.Layout.XMLLayout(),"testfile.xml"));
```

```
// using a ConsoleAppender with a PatternLayout
log4net.Config.BasicConfigurator.Configure(
  new log4net.Appender.ConsoleAppender(
    new log4net.Layout.PatternLayout("%d
      [%t] %-5p %c - %m%n")));

// using a ConsoleAppender with a SimpleLayout
log4net.Config.BasicConfigurator.Configure(
  new log4net.Appender.ConsoleAppender(new
    log4net.Layout.SimpleLayout()));
```

尽管这里用代码配置log4net也很方便，但是你却不能分别配置每个日志对象。所有的这些配置都是被应用到根日志上的。

log4net.Config.BasicConfigurator 类使用静态方法Configure 设置一个Appender 对象。而Appender的构造函数又会相应的要求Layout对象。你也可以不带参数直接调用BasicConfigurator.Configure()，它会使用一个缺省的PatternLayout对象，在一个ConsoleAppender中输出信息。如下所示：

```
log4net.Config.BasicConfigurator.Configure();
```

在输出时会显示如下格式的信息：

```
0 [1688] DEBUG log1 A B C - Test
20 [1688] INFO log1 A B C - Test
```

当log4net框架被配置好以后，就可以如前所述使用日志功能了。

4 总结

使用log4net可以很方便地为应用添加日志功能。应用Log4net，使用者可以很精确地控制日志信息的输出，减少了多余信息，提高了日志记录性能。同时，通过外部配置文件，用户可以不用重新编译程序就能改变应用的日志行为，使得用户可以根据情况灵活地选择要记录的信息。

分类: 技术研究

« 上一篇: 一些写英文简历的词汇吧
» 下一篇: 开发Visual Studio风格的用户界面 - - MagicLibrary使用指南

posted on 2005-03-24 08:17 sema 阅读(234407) 评论(123) 编辑 收藏

< Prev 1 2 3

评论

#101楼 2012-03-20 19:48 leilsn

我的程序采用跟楼主完全一样的配置，
如<param name="Header" value="[Header]\r\n"/>。
但输出结果\r\n不能换行，而是直接打印出来了，
如
2012-03-20 18:59:59,045 [1] FATAL testApp.Logging [(null)] - message
[Footer]\r\n[Header]\r\n2012-03-20 19:18:25,451 [1] INFO testApp.Logging [(null)] - message
2012-03-20 19:18:25,472 [1] DEBUG testApp.Logging [(null)] - message
这是为什么?

支持(0) 反对(0)

#102楼 2012-04-10 16:25 豆沙包y

@ dankye

引用

dankye：我的程序采用跟楼主完全一样的配置，
如<param name="Header" value="[Header]\r\n"/>。
但输出结果\r\n不能换行，而是直接打印出来了，
如
2012-03-20 18:59:59,045 [1] FATAL testApp.Logging [(null)] - message
[Footer]\r\n[Header]\r\n2012-03-20 19:18:25,451 [1] INFO testApp.Logging [(null)] - message
2012-03-20 19:18:25,472 [1] DEBUG tes...

换行使用 %n

支持(0) 反对(0)

#103楼 2013-01-17 20:21 2J

项目都是用代码管理器的，这样日志文件每次都是只读的，就写入不了，是否有相关设置呢?

支持(0) 反对(0)

#104楼 2013-05-22 11:38 德玛西亚冲锋

刚好看到公司项目中有这东西，还不知道是啥呢，这就给出了答案，帅啊，顶一下

支持(1) 反对(0)

#105楼 2013-11-06 17:03 ryanpark

mark

支持(0) 反对(0)

#106楼 2013-11-26 09:44 黄煜坤

转了哈谢谢

支持(0) 反对(0)

#107楼 2014-03-19 00:41 岁月已走远

@ 德玛西亚冲锋
日志不应该纳入源代码管理的，兄弟!
这个需要到源代码管理工具上去配置，svn，vss，git上都可以配的。

支持(0) 反对(0)

#108楼 2014-04-02 13:38 小菜? 大神?

很详细，谢谢分享

支持(0) 反对(0)

#109楼 2014-04-10 15:25 一 定 会 去 旅 行

这么长，瞬间就没有了看下去的兴趣。。。

支持(0) 反对(0)

#110楼 2014-04-25 09:15 劲抽风

用程序代码配置log4net的哪个部分能不能详细点?

支持(0) 反对(0)

#111楼 2014-08-04 10:43 夜の魔王

@ 劲抽风
其实你就按着config file那样的关键字写，是一样的。

支持(0) 反对(0)

#112楼 2014-08-25 16:10 看不见的颜色

刚好学习到这儿，转了

支持(0) 反对(0)

#113楼 2014-08-27 14:53 逍遥心

学习了，谢谢

支持(0) 反对(0)

#114楼 2014-09-25 09:49 jacketlin

受用啦，有空再深入研究下!

支持(0) 反对(0)

#115楼 2015-03-08 23:40 Greatcqi

写的很好，感谢博主。能否再给一个写入数据库的示例呢?

支持(0) 反对(0)

#116楼 2015-03-25 17:56 模拟人生

mark

支持(0) 反对(0)

#117楼 2015-04-16 10:07 linbin524

了解了，学习了~·

支持(0) 反对(0)

#118楼 2015-05-02 14:50 李可在江湖

请教 %logger property:[%property{NDC}] 和[%property{NDC}] 指的什么? 出错类 能不能说一下里面具体代表的意思呢?

支持(0) 反对(0)

#119楼 2015-07-24 09:21 AllanHao

转了哈，谢谢

支持(0) 反对(0)

#120楼 2016-04-28 15:08 查克拉的觉醒

mark

支持(0) 反对(0)

#121楼 2016-09-24 12:58 牛腩

fcrf f支持支持

支持(0) 反对(0)

#122楼 2017-01-07 17:44 杨国胜

博主您好，我现在有这样一个需求。我需要向多个文件写入日志。但是除了文件目录不一样外其他配置项都相同。所以我希望在配置文件中只写一个logger ，在代码中修改写文件目录。但是因为logger是命名实体的，如果我每次都用相同的logger名称获得logger对象的话，本质都是一个logger。那么其实我就是在不断地修改一个logger的写文件路径，我觉得这样不太好，我希望根据每一个文件路径，创建多个不同的logger。而且这样我还能使用缓存。
除了不用配置，完全代码建立logger以外，还有没有别的办法呢？？？

支持(0) 反对(0)

#123楼 2017-06-20 16:56 鸟鸡国国王

@ 杨国胜
哥们 这个问题你怎么解决的

支持(0) 反对(0)

< Prev 1 2 3

刷新评论 刷新页面 返回顶部

最新新闻:
· 蔚来：产品不够，服务来凑?
· NASA科学家：在火星上找古代生命痕迹要比地球还容易
· 苹果官方上线旧换新计划：小米华为OV都可以
· 滴滴又迎来一个新对手，上汽集团推出「享道出行」打车服务
· 互金行业进入最冷一年：减员两到三成 薪酬普遍腰斩
» 更多新闻...