log4j&slf4j日志框架入门

Log4j是什么、有什么

介绍

Log4j是Apache的一个开放源代码项目，通过使用Log4j，可以控制日志信息输送的目的地是控制台、文件、GUI组件、甚至是套接口服务器、NT的事件记录器等；也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，能够更加细致地控制日志的生成过程。这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

是什么

一个开源的、轻量级的、用于日志管理的框架

有什么

Log4j由三个重要的组件构成： **日志信息的输出格式，日志信息的优先级， 日 志信息的输出目的地**。日志信息的优先级用来指定这条日志信息的重要程度；日志信息的输出目的地指定了日志将打印到控制台还是文件中；而输出格式则控制了日志信息的显示内容 。

Log4j能干什么

能干什么

主要用来进行日志记录的管理，包括对日志输出的目的地，输出的信息级别和输出的格式等。

日志类别的层次结构(Loggers)

Log4j首要的相对于简单的使用System.out.println()方法的优点是基于它的在禁止一些特定的信息输出的同时不妨碍其它信息的输出的能力。这个能力源自于日志命名空间，也就是说，所有日志声明的空间，它根据一些开发员选择的公式而分类。

Logger被指定为实体，Logger的名字是大小写敏感的，它们遵循以下的命名语法规则：

用“.”来划分层次级别，如：cn.javass.test

日志级别

Log4j有如下级别

OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL。Log4j建议只使用四个级别，优先级从高到低分别是ERROR、WARN、INFO、DEBUG。通过在这里定义的级别，可以控制到应用程序中相应级别的日志信息的开关。

几个重要规则

1：级别的控制，就是只要大于等于指定的控制级别，就可以输出

2：如果有多个logger，都可以匹配输出，则每个logger都产生输出，其中根logger匹配所有的输出；而级别控制来源于路径最详细的logger。

如何获得日志操作类

java代码：

查看复制到剪贴板打印

```
1. public class A {
2. private static Logger logger = Logger.getLogger(A.class);
3. }
```

输出源

输出源

Log4j允许日志请求被输出到多个输出源。用Log4j的话说，一个输出源被称做一个Appender。

一个logger可以设置超过一个的appender。

常见Appender

java代码：

查看复制到剪贴板打印

```
1. org.apache.log4j.ConsoleAppender（控制台）
2. org.apache.log4j.FileAppender（文件）
3. org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件）
4. org.apache.log4j.RollingFileAppender（文件大小到达指定尺寸的时候产生一个新的文件）
5. org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）
6. org.apache.log4j.jdbc.JDBCAppender（把日志用JDBC记录到数据库中）
```

布局

布局

布局就是指输出信息的格式。在Log4j中称作Layout

常见Layout

org.apache.log4j.HTMLLayout（以HTML表格形式布局），

org.apache.log4j.PatternLayout（可以灵活地指定布局模式），

org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串），

org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等等信息）

最常用的PatternLayout

%m 输出代码中指定的消息

%p 输出优先级，即DEBUG，INFO，WARN，ERROR，FATAL

%r 输出自应用启动到输出该log信息耗费的毫秒数

%c 输出所属的类目，通常就是所在类的全名

%t 输出产生该日志事件的线程名

%n 输出一个回车换行符，Windows平台为“\r\n”，Unix平台为“\n”

%d 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyy MMM dd HH:mm:ss,SSS}，输出类似：2002年10月18日 22：10：28，921

%l 输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.main(TestLog4.java:10)

配置示例

配置

Log4j有两种配置方式，一种是xml格式，一种是properties格式。都是放置到classpath下面。默认名称分别是：log4j.xml和log4j.properties

nproperties配置示例如下：

java代码：

查看复制到剪贴板打印

```
1. log4j.rootLogger=error,javass.Console,javass.File
2.
3. log4j.appender.javass.Console=org.apache.log4j.ConsoleAppender
4.
log4j.appender.javass.Console.layout=org.apache.log4j.PatternLayout
5.
log4j.appender.javass.Console.layout.ConversionPattern=%d{HH:mm:ss,SSS} %5p (%C{1}:%M) - %m%n
```

java代码：

查看复制到剪贴板打印

```
1.
log4j.appender.javass.File=org.apache.log4j.DailyRollingFileAppender
2. log4j.appender.javass.File.file=javass.log
3. log4j.appender.javass.File.DatePattern=.yyyy-MM-dd
4.
log4j.appender.javass.File.layout=org.apache.log4j.PatternLayout
5.
log4j.appender.javass.File.layout.ConversionPattern=%d{HH:mm:ss,SSS} %5p (%C{1}:%M) - %m%n
6.
7. log4j.logger.cn.javass=debug
```

Appender、Layout、Logger三者之间的关系

通过前面的讲解，我们可以在配置文件中看出以下关系：

每个Appender都要引用自己的Layout。

每个Logger都可以指定一个级别，同时引用多个Appender；而一个Appender也同时可以被多个Logger引用。

配置示例

xml配置示例如下：

java代码：

查看复制到剪贴板打印

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration
xmlns:log4j='http://jakarta.apache.org/log4j/'>
<appender name="test"
class="org.apache.log4j.ConsoleAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="[%d{dd HH:mm:ss,SSS\} %-5p] [%t] %c{2\} - %m%n" />
</layout>
</appender>
```

java代码：

查看复制到剪贴板打印

```
<appender name="file"
class="org.apache.log4j.DailyRollingFileAppender">
<param name="File" value="E:/test.log" />
<param name="DatePattern" value="'.'yyyy-MM-dd'.log'" />
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern"
value="[%d{MMdd HH:mm:ss SSS\} %-5p] [%t] %c{3\} - %m%n" />
</layout>
</appender>
<logger name="cn.javass" additivity="true">
<level value="debug" />
<appender-ref ref="file" />
</logger>
<root>
<level value="error"/>
<appender-ref ref="test" />
<appender-ref ref="file" />
</root>
</log4j:configuration>
```

log4j.xml比log4j.properties多的功能

首先要注意，log4j.xml优先于log4j.properties，如果同时存在log4j.xml和log4j.properites，以log4j.xml为准。

在log4j.properties里，控制级别的时候，只能打印出大于指定级别的所有信息；但是在log4j.xml中可以通过filter来完成过滤：典型的引用是只打印出某种级别的信息。

java代码：

查看复制到剪贴板打印

```
<appender name="log.file"
class="org.apache.log4j.DailyRollingFileAppender">
<filter class="org.apache.log4j.varia.LevelRangeFilter">
<param name="levelMin" value="info" />
<param name="levelMax" value="info" />
<param name="AcceptOnMatch" value="true" />
</filter>
</appender>
```

可以通过logger的additivity="false"属性，来设置多个logger是否重复输出同一条信息

java代码：

查看复制到剪贴板打印

```
1. <logger name="cn.javass1" additivity="false">
2. <level value="debug" />
3. <appender-ref ref="log.console" />
4. <appender-ref ref="log.file" />
5. </logger>
6.
```

在java中使用log4j中要注意的小问题

看似奇怪的重复级别判断：我们在看一些成熟框架的源代码中，经常看到如下代码：

java代码：

查看复制到剪贴板打印

```
1. if (logger.isDebugEnabled()){
2. logger.debug("debug: "+name);
3. }
```

为什么不是直接logger.debug("debug: "+name);呢?

在配置文件中虽然可以使用控制级别为比debug级别更高的级别，而不输出debug信息；但是，这里的字符串连接操作仍然会影响运行效率；如果先判断当前logger的级别，如果级别不合适的话，连这句字符串连接都可以不做了。

在java中使用log4j时，我们可以使用它的eclipse插件log4e。它的官方网址是：http://log4e.jayefem.de/，分为商业版本和免费版本。我们只需要使用其免费版本，就可以极大的帮我们提高开发效率。比如：在一个类中声明一个logger；帮我们写麻烦的logger.isDebugEnabled()；在一个方法开始的时候打印所有的参数；输出一个变量等等。

jdk内置的日志系统

除了使用log4j进行日志输出之外，我们还可以使用jdk内置的日志系统。它位于java.util.logging包下，因此不需要引用额外的jar包。其基本概念与log4j大同小异，我们可以类比的学习。

其配置文件位于jdk安装目录下的/jre/lib/logging.properties。

请同学们观察配置文件，自己找到我们在log4j里找的Appender、Layout、级别和Logger，以及它们的关系。

配置示例

java代码：

查看复制到剪贴板打印

```
1. handlers= java.util.logging.FileHandler,
java.util.logging.ConsoleHandler
2.
3. #.level= ALL
4.
5. # default file output is in user's home directory.
6. java.util.logging.FileHandler.pattern = %h/java%u.log
```

```
7. java.util.logging.FileHandler.limit = 50000
8. java.util.logging.FileHandler.count = 1
9. java.util.logging.FileHandler.formatter =
java.util.logging.XMLFormatter
10.
11. # Limit the message that are printed on the console to INFO
and above.
12. java.util.logging.ConsoleHandler.level = FINEST
13. java.util.logging.ConsoleHandler.formatter =
java.util.logging.SimpleFormatter
14.
```

java代码：

查看复制到剪贴板打印

```
1. # messages:
2. #cn.javass.level = SEVERE
3. cn.javass.level = WARNING
4. #cn.javass.level = INFO
5. #cn.javass.level = CONFIG
6. #cn.javass.level = FINE
7. #cn.javass.level = FINER
8. #cn.javass.level = FINEST
9.
```

在java中使用jdk内置的日志系统

声明一个logger：

java代码：

查看复制到剪贴板打印

```
1. private static final Logger logger =
LoggerFactory.getLogger(“cn.javass.Test”);
```

输出信息：

java代码：

查看复制到剪贴板打印

```
1. logger.finest("This is finest Level");
2.          logger.finer("This is finer Level");
3.          logger.fine("This is fine Level");
4.          logger.config("This is config Level");
5.          logger.info("This is info Level");
6.          logger.warning("This is warning Level");
7.          logger.severe("This is severe Level");
8.
```

使用classpath下的logging.properties

如果想使用自定义的logging.properties，只需要保证这段代码运行在getLogger之前即可：

java代码：

查看复制到剪贴板打印

```
1.  ClassLoader classLoader =
Thread.currentThread().getContextClassLoader();
2.      InputStream inputStream =
classLoader.getResourceAsStream("logging.properties");
3.      if (inputStream != null) {
4.          try {
5. LogManager.getLogManager().readConfiguration(inputStream);
6. } catch (SecurityException e) {
7. e.printStackTrace();
8. } catch (IOException e) {
9. e.printStackTrace();
10. }
11.      }
```

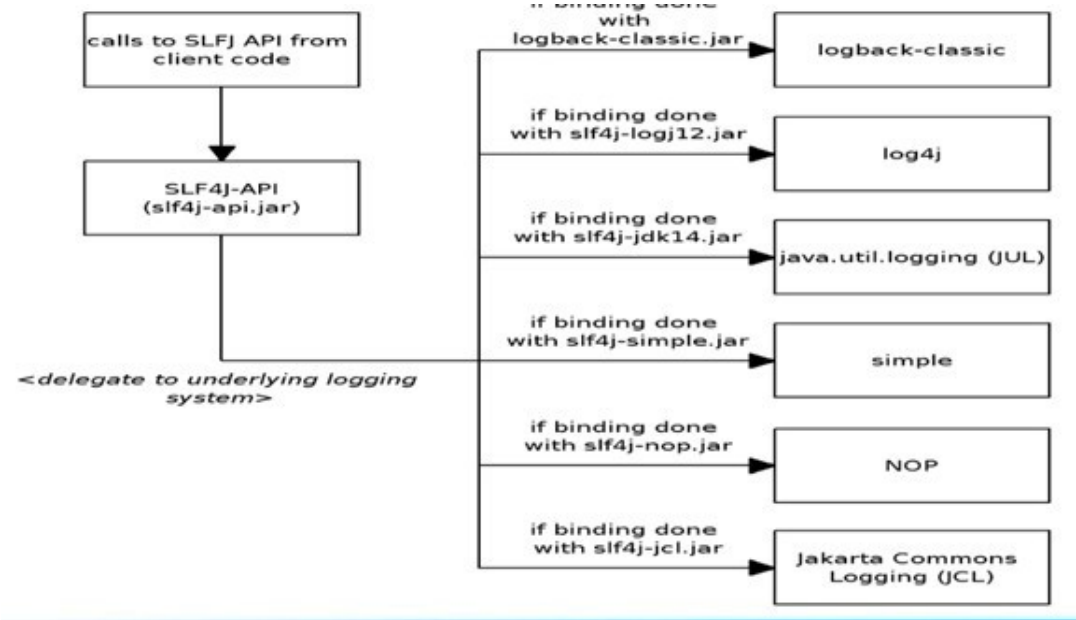
slf4j简介

简单日记门面(simple logging Facade for java)SLF4J是为各种loging APIs提供一个简单统一的接口，从而使得最终用户能够在部署的时候配置自己希望的loging APIs实现。

准确的说，slf4j并不是一种具体的日志系统，而是一个用户日志系统的facade，允许用户在部署最终应用时方便的变更其日志系统。

在系统开发中，统一按照slf4j的API进行开发，在部署时，选择不同的日志系统包，即可自动转换到不同的日志系统上。比如：选择JDK自带的日志系统，则只需要将slf4j-api-1.5.10.jar和slf4j-jdk14-1.5.10.jar放置到classpath中即可，如果中途无法忍受JDK自带的日志系统了，想换成log4j的日志系统，仅需要用slf4j-log4j12-1.5.10.jar替换slf4j-jdk14-1.5.10.jar即可（**当然也需要log4j的jar及 配置文件**）

slf4j门面原理



在java中使用slf4j

获得logger对象：

java代码：

查看复制到剪贴板打印

```
1. private static final Logger logger =
LoggerFactory.getLogger(Test.class);
2. 输出日志信息：
3. logger.debug("debug");
```

slf4j中的占位符一不再需要冗长的级别判断

大家应该还记得，在log4j中，为了提高运行效率，往往在输出信息之前，还要进行级别判断，以避免无效的字符串连接操作。如下：

```
if (logger.isDebugEnabled()){
logger.debug("debug："+name);
}
```

slf4j巧妙的解决了这个问题：先传入带有占位符的字符串，同时把其他参数传入，在slf4j的内容部实现中，如果级别合适再去用传入的参数去替换字符串中的占位符，否则不用执行。

*logger.info("{} is {}", **new String[]{"x","y"});***