# Java log4j详细教程

日志是应用软件中不可缺少的部分，Apache的开源项目log4j是一个功能强大的日志组件,提供方便的日志记录。在apache网站：jakarta.apache.org/log4j 可以免费下载到Log4j最新版本的软件包

**一：Log4j入门简介学习**

 Log4j是Apache的一个开放源代码项目，通过使用Log4j，我们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、UNIX Syslog守护进程等；我们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，我们能够更加细致地控制日志的生成过程。最令人感兴趣的就是，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。

此外，通过Log4j其他语言接口，您可以在C、C++、.Net、PL/SQL程序中使用Log4j，其语法和用法与在Java程序中一样，使得多语言分布式系统得到一个统一一致的日志组件模块。而且，通过使用各种第三方扩展，您可以很方便地将Log4j集成到J2EE、JINI甚至是SNMP应用中。

**二、入门实例**

1.新建一个JAva工程，导入包log4j-1.2.17.jar，整个工程最终目录如下

2、src同级创建并设置log4j.properties

 ### 设置###  
log4j.rootLogger = debug,stdout,D,E  
### 输出信息到控制抬 ###  
log4j.appender.stdout = org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target = System.out  
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n  
### 输出DEBUG 级别以上的日志到=E://logs/error.log ###  
log4j.appender.D = org.apache.log4j.DailyRollingFileAppender  
log4j.appender.D.File = E://logs/log.log  
log4j.appender.D.Append = true  
log4j.appender.D.Threshold = DEBUG   
log4j.appender.D.layout = org.apache.log4j.PatternLayout  
log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss}  [ %t:%r ] - [ %p ]  %m%n  
### 输出ERROR 级别以上的日志到=E://logs/error.log ###  
log4j.appender.E = org.apache.log4j.DailyRollingFileAppender  
log4j.appender.E.File =E://logs/error.log   
log4j.appender.E.Append = true  
log4j.appender.E.Threshold = ERROR   
log4j.appender.E.layout = org.apache.log4j.PatternLayout  
log4j.appender.E.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss}  [ %t:%r ] - [ %p ]  %m%n

3、设置日志内容

package com.mucfc;

import org.apache.log4j.Logger;

/\*\*

\*@author linbingwen

\*@2015年5月18日9:14:21

\*/

public class Test {

private static Logger logger = Logger.getLogger(Test.class);

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// System.out.println("This is println message.");

// 记录debug级别的信息

logger.debug("This is debug message.");

// 记录info级别的信息

logger.info("This is info message.");

// 记录error级别的信息

logger.error("This is error message.");

}

}

4、输出结果

（1）首先是控制台的信息

（2）再来看输出的文件

内容如下，发现已按照要求输出到对应的文档中去了。

三、Log4j基本使用方法

Log4j由三个重要的组件构成：日志信息的优先级，日志信息的输出目的地，日志信息的输出格式。日志信息的优先级从高到低有ERROR、WARN、 INFO、DEBUG，分别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。

2.1、定义配置文件

其实您也可以完全不使用配置文件，而是在代码中配置Log4j环境。但是，使用配置文件将使您的应用程序更加灵活。Log4j支持两种配置文件格式，一种是XML格式的文件，一种是Java特性文件（键=值）。下面我们介绍使用Java特性文件做为配置文件的方法：  
1.配置根Logger，其语法为：

log4j.rootLogger = [ level ] , appenderName, appenderName, …  
其中，level 是日志记录的优先级，分为OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者您定义的级别。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，您可以控制到应用程序中相应级别的日志信息的开关。比如在这里定 义了INFO级别，则应用程序中所有DEBUG级别的日志信息将不被打印出来。 appenderName就是指B日志信息输出到哪个地方。您可以同时指定多个输出目的地。

2.配置日志信息输出目的地Appender，其语法为：

log4j.appender.appenderName = fully.qualified.name.of.appender.class    
log4j.appender.appenderName.option1 = value1    
…    
log4j.appender.appenderName.option = valueN

其中，Log4j提供的appender有以下几种：

org.apache.log4j.ConsoleAppender（控制台），    
org.apache.log4j.FileAppender（文件），    
org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件），    
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），    
org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）

3.配置日志信息的格式（布局），其语法为：

log4j.appender.appenderName.layout = fully.qualified.name.of.layout.class    
log4j.appender.appenderName.layout.option1 = value1    
…    
log4j.appender.appenderName.layout.option = valueN

其中，Log4j提供的layout有以e几种：

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）  
Log4J采用类似C语言中的printf函数的打印格式格式化日志信息，打印参数如下： %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“rn”，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)

2.2、在代码中使用Log4j

1.得到记录器

使用Log4j，第一步就是获取日志记录器，这个记录器将负责控制日志信息。其语法为：

public static Logger getLogger( String name)

通过指定的名字获得记录器，如果必要的话，则为这个名字创建一个新的记录器。Name一般取本类的名字，比如：

static Logger logger = Logger.getLogger ( ServerWithLog4j.class.getName () )

2.读取配置文件

当获得了日志记录器之后，第二步将配置Log4j环境，其语法为：

BasicConfigurator.configure ()： 自动快速地使用缺省Log4j环境。    
PropertyConfigurator.configure ( String configFilename) ：读取使用Java的特性文件编写的配置文件。    
DOMConfigurator.configure ( String filename ) ：读取XML形式的配置文件。  
3.插入记录信息（格式化日志信息）

当上两个必要步骤执行完毕，您就可以轻松地使用不同优先级别的日志记录语句插入到您想记录日志的任何地方，其语法如下：

Logger.debug ( Object message ) ;    
Logger.info ( Object message ) ;    
Logger.warn ( Object message ) ;    
Logger.error ( Object message ) ;

2.3、日志级别

每个Logger都被了一个日志级别（log level），用来控制日志信息的输出。日志级别从高到低分为：

A：off 最高等级，用于关闭所有日志记录。  
B：fatal 指出每个严重的错误事件将会导致应用程序的退出。  
C：error 指出虽然发生错误事件，但仍然不影响系统的继续运行。  
D：warm 表明会出现潜在的错误情形。  
E：info 一般和在粗粒度级别上，强调应用程序的运行全程。  
F：debug 一般用于细粒度级别上，对调试应用程序非常有帮助。  
G：all 最低等级，用于打开所有日志记录。

上面这些级别是定义在org.apache.log4j.Level类中。Log4j只建议使用4个级别，优先级从高到低分别是error,warn,info和debug。通过使用日志级别，可以控制应用程序中相应级别日志信息的输出。例如，如果使用b了info级别，则应用程序中所有低于info级别的日志信息(如debug)将不会被打印出来。

四、Web项目中使用Log4j实例

上面代码描述了Log4j的简单应用，其实使用Log4j也就是这样简单方便。当然除了上面的配置方法，还有其它，比如做一个J2EE应用，在J2EE应用使用Log4j，必须先在启动服务时加载Log4j的配置文件进行初始化，可以在web.xml中进行。

1、web应用的log4j使用基本上都采用：新建一个servlet，这个servlet在init函数中为log4j执行配置。一般就是读入配置文件。所以需要在web.xml中为这个servlet配置，同时设定load-on-startup为1。

2、这个servlet配置log4j就是读出配置文件，然后调用configure函数。这里有两个问题：一、需要知道文件在哪里；二、需要正确的文件类型

3、配置文件位置在web.xml中配置一个param即可，路径一般是相对于web的root目录

4、文件类型一般有两种，一个是Java的property文件，另一种是xml文件

配置文件的大致内容：log4j可以指定输出的log级别的最低等级，以及log的输出配置格式，每个log可以指定多个输出方式

（1）创建Web工程，整个工程最后目录如下

（2）web.xml配置如下：

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>LogLearning</display-name>

<servlet>

<servlet-name>Log4JTestServlet</servlet-name>

<servlet-class>com.mucfc.Log4JTestServlet</servlet-class>

</servlet>

<!--用来启动 log4jConfigLocation的servlet -->

<servlet>

<servlet-name>Log4JInitServlet</servlet-name>

<servlet-class>com.mucfc.Log4JInitServlet</servlet-class>

<init-param>

<param-name>log4j-properties-location</param-name>

<param-value>/WEB-INF/classes/log4j.properties</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Log4JTestServlet</servlet-name>

<url-pattern>/test</url-pattern>

</servlet-mapping>

</web-app>

（3）配置文件log4j.properties

 ### set log levels ###    
log4j.rootLogger = debug,stdout,D,E    
log4j.appender.stdout = org.apache.log4j.ConsoleAppender    
log4j.appender.stdout.Target = System.out    
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout    
log4j.appender.stdout.layout.ConversionPattern = [%-5p] %d{yyyy-MM-dd HH:mm:ss,SSS} method:%l%n%m%n    
log4j.appender.D = org.apache.log4j.DailyRollingFileAppender    
log4j.appender.D.File = F://logs/log.log    
log4j.appender.D.Append = true    
log4j.appender.D.Threshold = DEBUG     
log4j.appender.D.layout = org.apache.log4j.PatternLayout    
log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss}  [ %t:%r ] - [ %p ]  %m%n    
log4j.appender.E = org.apache.log4j.DailyRollingFileAppender    
log4j.appender.E.File =F://logs/error.log     
log4j.appender.E.Append = true    
log4j.appender.E.Threshold = ERROR     
log4j.appender.E.layout = org.apache.log4j.PatternLayout    
log4j.appender.E.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss}  [ %t:%r ] - [ %p ]  %m%n

（4）web容器一来就初始化的servlet

Log4JInitServlet.java

package com.mucfc;

import java.io.File;

import java.io.IOException;

import javax.servlet.ServletConfig;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.BasicConfigurator;

import org.apache.log4j.PropertyConfigurator;

/\*\*

\* Servlet implementation class Log4JInitServlet

\*/

@WebServlet("/Log4JInitServlet")

public class Log4JInitServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public Log4JInitServlet() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see Servlet#init(ServletConfig)

\*/

public void init(ServletConfig config) throws ServletException {

System.out.println("Log4JInitServlet 正在初始化 log4j日志设置信息");

String log4jLocation = config.getInitParameter("log4j-properties-location");

ServletContext sc = config.getServletContext();

if (log4jLocation == null) {

System.err.println("\*\*\* 没有 log4j-properties-location 初始化的文件, 所以使用 BasicConfigurator初始化");

BasicConfigurator.configure();

} else {

String webAppPath = sc.getRealPath("/");

String log4jProp = webAppPath + log4jLocation;

File yoMamaYesThisSaysYoMama = new File(log4jProp);

if (yoMamaYesThisSaysYoMama.exists()) {

System.out.println("使用: " + log4jProp+"初始化日志设置信息");

PropertyConfigurator.configure(log4jProp);

} else {

System.err.println("\*\*\* " + log4jProp + " 文件没有找到， 所以使用 BasicConfigurator初始化");

BasicConfigurator.configure();

}

}

super.init(config);

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// TODO Auto-generated method stub

}

}

调用日志Log4JTestServlet,java

package com.mucfc;

import java.io.IOException;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.Logger;

/\*\*

\* Servlet implementation class Log4JTestServlet

\*/

@WebServlet("/Log4JTestServlet")

public class Log4JTestServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

private static Logger logger = Logger.getLogger(Log4JTestServlet.class);

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public Log4JTestServlet() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see Servlet#init(ServletConfig)

\*/

public void init(ServletConfig config) throws ServletException {

// TODO Auto-generated method stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// 记录debug级别的信息

logger.debug("This is debug message.");

// 记录info级别的信息

logger.info("This is info message.");

// 记录error级别的信息

logger.error("This is error message.");

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request,response);

}

}

接下来就是运行了，来看看结果：

输出结果：

五、Spring中使用Log4j

这里要实现web项目中利用Spring来使用Log4j

（1）接上面的工程，然后再导入Spring的包

（2）web.xml增加

<!-- 设置根目录 -->

<context-param>

<param-name>webAppRootKey</param-name>

<param-value>webapp.root</param-value>

</context-param>

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>/WEB-INF/classes/log4j.properties</param-value>

</context-param>

<!-- 3000表示 开一条watchdog线程每60秒扫描一下配置文件的变化;这样便于日志存放位置的改变 -->

<context-param>

<param-name>log4jRefreshInterval</param-name>

<param-value>3000</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

整个内容如下：

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"

id="WebApp\_ID" version="3.0">

<display-name>LogLearning</display-name>

<servlet>

<servlet-name>Log4JTestServlet</servlet-name>

<servlet-class>com.mucfc.Log4JTestServlet</servlet-class>

</servlet>

<!--用来启动 log4jConfigLocation的servlet -->

<!-- <servlet>

<servlet-name>Log4JInitServlet</servlet-name>

<servlet-class>com.mucfc.Log4JInitServlet</servlet-class>

<init-param>

<param-name>log4j-properties-location</param-name>

<param-value>/WEB-INF/classes/log4j.properties</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>-->

<servlet-mapping>

<servlet-name>Log4JTestServlet</servlet-name>

<url-pattern>/test</url-pattern>

</servlet-mapping>

<!-- Spring 容器加载 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:applicationContext.xml</param-value>

</context-param>

<!-- 设置根目录 -->

<context-param>

<param-name>webAppRootKey</param-name>

<param-value>webapp.root</param-value>

</context-param>

<context-param>

<param-name>log4jConfigLocation</param-name>

<param-value>/WEB-INF/classes/log4j.properties</param-value>

</context-param>

<!-- 3000表示 开一条watchdog线程每60秒扫描一下配置文件的变化;这样便于日志存放位置的改变 -->

<context-param>

<param-name>log4jRefreshInterval</param-name>

<param-value>3000</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>

</listener>

</web-app>

这里Log4JInitServlet.java就相当于没用到了。

（2）applicationContext.xml

没有内容：

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-3.2.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.2.xsd">

</beans>

（3）这样日志就跟随Spring窗口启动而启动了

程序一运行，就会自动把日志打印

log.log

error.log为空，因为它只打印error级别以上的信息

浏览器输入<http://localhost:8080/LogLearning2/test>

然后打开文件

# Log4J教程

log4j是一个用Java编写的可靠，快速和灵活的日志框架（API），它在Apache软件许可下发布。

Log4j已经被移植到了C，C++，C＃，Perl，[Python](http://www.yiibai.com/python/)和Ruby等语言中。

Log4j是高度可配置的，并可通过在运行时的外部文件配置。它根据记录的优先级别，并提供机制，以指示记录信息到许多的目的地，诸如：数据库，文件，控制台，UNIX系统日志等。

Log4j中有三个主要组成部分：

* **loggers:** 负责捕获记录信息。
* **appenders :** 负责发布日志信息，以不同的首选目的地。
* **layouts:** 负责格式化不同风格的日志信息。

## log4j的历史

* 始于1996年初的E.U. SEMPER （安全电子市场为欧洲）跟踪API的项目。
* 不计其数的改进，在几个大量的工作之后，API已经发展成为log4j，一个流行的日志记录包为Java。
* 该软件包是Apache软件许可证，由开源认证是一个不折不扣的开源许可证下发布。
* 最新log4j的版本，包括完整的源代码，类文件和文档可以在这里找到 <http://logging.apache.org/log4j/>.

## log4j 特性:

* log4j的是线程安全的
* log4j是经过优化速度的
* log4j是基于一个名为记录器的层次结构
* log4j的支持每个记录器多输出追加器（appender）
* log4j支持国际化。
* log4j并不限于一组预定义的设备
* 日志行为可以使用配置文件在运行时设置
* log4j设计从一开始就是处理Java异常
* log4j使用多个层次，即ALL，TRACE，DEBUG，INFO，WARN，ERROR和FATAL
* 日志输出的格式可以通过扩展Layout类容易地改变
* 日志输出的目标，以及在写入策略可通过实现Appender程序接口改变
* log4j 会故障停止。然而，尽管它肯定努力确保传递，log4j不保证每个日志语句将被传递到目的地。

## 日志记录N个缺点及优点：

日志是软件开发的重要组成部分。一个精心编写的日志代码提供快速的调试，维护方便，以及应用程序的运行时信息结构化存储。

日志记录确实也有它的缺点。它可以减缓的应用程序。如果太详细，它可能会导致滚动失明。为了减轻这些影响，log4j被设计为是可靠，快速和可扩展。

由于记录很少为应用的主要重点，但log4j API致力于成为易于理解和使用。

# log4j安装配置

Log4j的API包使用Apache软件许可证，由开源倡议认证一个完全成熟的开源许可证下发布。

最新log4j的版本，包括完整的源代码，类文件和文档可以在这里找到 <http://logging.apache.org/log4j/>.

下载 apache-log4j-x.x.x.tar.gz 做到以下几点：

## 步骤1:

将下载的文件解压缩和解压在 /usr/local/ 目录，如下所示：

$ gunzip apache-log4j-1.2.15.tar.gz

$ tar -xvf apache-log4j-1.2.15.tar

apache-log4j-1.2.15/tests/input/

apache-log4j-1.2.15/tests/input/xml/

apache-log4j-1.2.15/tests/src/

apache-log4j-1.2.15/tests/src/java/

apache-log4j-1.2.15/tests/src/java/org/

.......................................

当执行解压缩，这将创建一个名称 apache-log4j-x.x.x 的目录层次结构如下：

-rw-r--r-- 1 root root 3565 2007-08-25 00:09 BUILD-INFO.txt

-rw-r--r-- 1 root root 2607 2007-08-25 00:09 build.properties.sample

-rw-r--r-- 1 root root 32619 2007-08-25 00:09 build.xml

drwxr-xr-x 14 root root 4096 2010-02-04 14:09 contribs

drwxr-xr-x 5 root root 4096 2010-02-04 14:09 examples

-rw-r--r-- 1 root root 2752 2007-08-25 00:09 INSTALL

-rw-r--r-- 1 root root 4787 2007-08-25 00:09 KEYS

-rw-r--r-- 1 root root 11366 2007-08-25 00:09 LICENSE

-rw-r--r-- 1 root root 391834 2007-08-25 00:29 log4j-1.2.15.jar

-rw-r--r-- 1 root root 160 2007-08-25 00:09 NOTICE

-rwxr-xr-x 1 root root 10240 2007-08-25 00:27 NTEventLogAppender.dll

-rw-r--r-- 1 root root 17780 2007-08-25 00:09 pom.xml

drwxr-xr-x 7 root root 4096 2007-08-25 00:13 site

drwxr-xr-x 8 root root 4096 2010-02-04 14:08 src

drwxr-xr-x 6 root root 4096 2010-02-04 14:09 tests

## 步骤2:

这一步是可选的，取决于什么功能，要使用log4j 框架。如果已经有以下安装在您的机器，那么可以使用这些软件包，否则将需要安装它们，才能正常使log4j工作

* JavaMail API:  电子邮件。基于log4j日志记录功能需要Java邮件API（mail.jar）在计算机上安装 <https://glassfish.dev.java.net/javaee5/mail/>
* JavaBeans 活动框架： Java邮件API还需要JavaBeans激活框架（的activation.jar）从<http://java.sun.com/products/javabeans/jaf/index.jsp> 上下载安装在您的计算机上
* Java 信息服务:  log4j的JMS兼容的功能将需要两个JMS和JNDI（Java命名和Directory接口），可以从<http://java.sun.com/products/jms> 下载安装
* XML Parser: 需要一个JAXP兼容的XML解析器来使用log4j。请确保已经在机器上安装xerces.jar ，可从<http://xerces.apache.org/xerces-j/install.html> 下载安装

## 步骤3:

这一步是非常重要的，需要设置CLASSPATH和PATH变量正确。在这里要设置只是 log4j.x.x.x.jar 文件

$ pwd

/usr/local/apache-log4j-1.2.15

$ export CLASSPATH=

$CLASSPATH:/usr/local/apache-log4j-1.2.15/log4j-1.2.15.jar

$ export PATH=$PATH:/usr/local/apache-log4j-1.2.15/

注意：如果在Window上开发使用Eclipse的话，可以在Eclipse创建用户库并加入到构建路径中。

# log4j架构

Log4j API设计为分层结构，其中每一层提供了不同的对象，对象执行不同的任务。这使得设计灵活，根据将来需要来扩展。

有两种类型可用在Log4j的框架对象。

* **核心对象：** 框架的强制对象和框架的使用。
* **支持对象：** 框架和支持体核心对象，可选的对象执行另外重要的任务。

## 核心对象：

### Logger对象：

顶级层的Logger，它提供Logger对象。Logger对象负责捕获日志信息及它们存储在一个空间的层次结构。

### 布局对象：

该层提供其用于格式化不同风格的日志信息的对象。布局层提供支持Appender对象到发布日志信息之前。

布局对象的发布方式是人类可读的及可重复使用的记录信息的一个重要的角色。

### Appender对象：

下位层提供Appender对象。Appender对象负责发布日志信息，以不同的首选目的地，如数据库，文件，控制台，UNIX系统日志等。

以下是显示Log4J框架的不同组件的虚拟图：

## 支持对象：

log4j框架的其他重要的对象起到日志框架的一个重要作用：

### Level对象：

级别对象定义的任何记录信息的粒度和优先级。有记录的七个级别在API中定义：OFF, DEBUG, INFO, ERROR, WARN, FATAL 和 ALL

### Filter对象：

过滤对象用于分析日志信息及是否应记录或不用这些信息做出进一步的决定。

一个appender对象可以有与之关联的几个Filter对象。如果日志记录信息传递给特定Appender对象，都和特定Appender相关的Filter对象批准的日志信息，然后才能发布到所连接的目的地。

### 对象渲染器：

ObjectRenderer对象是一个指定提供传递到日志框架的不同对象的字符串表示。这个对象所使用的布局对象来准备最后的日志信息。

### 日志管理：

日志管理对象管理的日志框架。它负责从一个系统级的配置文件或配置类读取初始配置参数。

# log4j配置

上一章介绍log4j的核心组件。本章介绍如何使用配置文件来配置这些核心组件。配置log4j涉及分配级别，定义追加程序，并在配置文件中指定布局的对象。

log4j.properties文件是一个键 - 值对保存 log4j 配置属性文件。默认情况下，日志管理在CLASSPATH 查找一个名为 log4j.properties 的文件。

* 根日志记录器的级别定义为DEBUG并连接附加器命名为X到它
* 设置名为X的附加目的地是一个有效的appender
* 设置布局的附加器X

## log4j.properties 语法:

以下是 log4j.properties 文件的一个appender X的语法：

# Define the root logger with appender X

log4j.rootLogger = DEBUG, X

# Set the appender named X to be a File appender

log4j.appender.X=org.apache.log4j.FileAppender

# Define the layout for X appender

log4j.appender.X.layout=org.apache.log4j.PatternLayout

log4j.appender.X.layout.conversionPattern=%m%n

## log4j.properties 示例:

使用上面的语法，我们定义 log4j.properties 文件如下：

* 根日志记录器(logger)的级别定义为DEBUG并连接附加器命名为FILE
* 附加器(appender)File是定义为org.apache.log4j.FileAppender并写入到一个名为“log.out”位于日志log目录下
* 定义的布局模式是%m%n，这意味着每打印日志消息之后，将加上一个换行符

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

需要注意的是log4j支持UNIX风格的变量替换，如 ${variableName}.

## 调试级别：

使用DEBUG两个追加程序。所有可能的选项有：

* TRACE
* DEBUG
* INFO
* WARN
* ERROR
* FATAL
* ALL

这些级别将在 [Log4j调试级别](http://www.yiibai.com/log4j/log4j_logging_levels.html) 这一文章中解释

## Appenders:

Apache的log4j提供Appender对象主要负责打印日志消息到不同的目的地，如控制台，文件，sockets，NT事件日志等等。

每个Appender对象具有与之相关联的不同的属性，并且这些属性表明对象的行为

|  |  |
| --- | --- |
| **属性** | **描述** |
| layout | Appender使用布局Layout 对象和与之相关的格式化的日志记录信息转换模式 |
| target | 目标可以是一个控制台，一个文件，或根据附加器的另一个项目 |
| level | 级别是必需的，以控制日志消息的过滤 |
| threshold | Appender可以有与之独立的记录器级别相关联的级别阈值水平。Appender忽略具有级别低于阈级别的任何日志消息 |
| filter | Filter 对象可以分析超出级别的匹配记录信息，并决定是否记录的请求应该由一个特定 Appender 或忽略处理 |

可以通过包括以下方法的配置文件中的下面设置一个 Appender 对象添加到记录器：

log4j.logger.[logger-name]=level, appender1,appender..n

可以编写以XML格式相同的结构如下：

<logger name="com.apress.logging.log4j" additivity="false">

<appender-ref ref="appender1"/>

<appender-ref ref="appender2"/>

</logger>

如果想要添加Appender对象到程序，那么可以使用下面的方法：

public void addAppender(Appender appender);

addAppender()方法添加一个appender到Logger对象。作为示例配置演示，可以添加很多Appender对象到记录器在逗号分隔的列表，每个打印日志信息分离目的地。

我们仅使用一个附加目的地FileAppender在我们上面的例子。所有可能的附加目的地选项有：

* AppenderSkeleton
* AsyncAppender
* ConsoleAppender
* DailyRollingFileAppender
* ExternallyRolledFileAppender
* FileAppender
* JDBCAppender
* JMSAppender
* LF5Appender
* NTEventLogAppender
* NullAppender
* RollingFileAppender
* SMTPAppender
* SocketAppender
* SocketHubAppender
* SyslogAppender
* TelnetAppender
* WriterAppender

我们将涵盖 [FileAppender文件](http://www.yiibai.com/log4j/log4j_logging_files.html) 和 JDBCAppender 记录将被包括[记录在数据库](http://www.yiibai.com/log4j/log4j_logging_database.html)

## Layout:

我们使用的PatternLayout 使用 appender。所有可能的选项有：

* DateLayout
* HTMLLayout
* PatternLayout
* SimpleLayout
* XMLLayout

使用HTMLLayout和XMLLayout，可以在HTML和XML格式和生成日志。

## 布局格式：

如何在章节格式的日志信息：[Log格式](http://www.yiibai.com/log4j/log4j_log_formatting.html)

# log4j示例程序

前面我们已经看到了如何创建一个配置文件。本教程将讲解如何生成调试信息和日志在一个简单的文本文件。

下面是我们的例子中创建了一个简单的配置文件。这里再重复一次：

* 下载最新的Log4j库：<http://logging.apache.org/log4j/2.x/download.html>
* 根记录器的级别定义为DEBUG并连接appender命名为FILE。
* appender FILE文件被定义为 org.apache.log4j.FileAppender 并写入到一个名为“log.out”位于 log 目录下。
* 定义的布局模式是 %m%n，这意味着打印日志消息之后自动加上一个换行符。

所以 log4j.properties 文件的内容如下：

# Define the root logger with appender file

log = /usr/home/log4j

log4j.rootLogger = DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

log4j.appender.FILE.File=${log}/log.out

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%m%n

## 在Java程序中使用log4j：

下面的Java类是一个非常简单的例子，Java应用程序初始化，然后使用Log4J日志库。

import org.apache.log4j.Logger;

import java.io.\*;

import java.sql.SQLException;

import java.util.\*;

public class log4jExample{

/\* Get actual class name to be printed on \*/

static Logger log = Logger.getLogger(

log4jExample.class.getName());

public static void main(String[] args)

throws IOException,SQLException{

log.debug("Hello this is an debug message");

log.info("Hello this is an info message");

}

}

## 编译和运行：

下面是步骤编译并运行上述程序。确保在进行编译和执行之前，适当地设置PATH和CLASSPATH。

所有的库应该在 CLASSPATH 和 log4j.properties  文件应该在PATH可用。所以，做到以下几点：

* 创建log4j.properties如上图所示。
* 创建log4jExample.java如上图所示，并对其进行编译。
* 执行log4jExample二进制运行程序。

在里面 /usr/home/log4j/log.out 文件会得到下面的结果：

Hello this is an debug message

Hello this is an info message