Log4J

LOG4J 简介

在强调可重用组件开发的今天，除了自己从头到尾开发一个可重用的日志操作类外，Apache 为我们提供了一个强有力的日志操作包-Log4j。

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。

log4j 的作用

一、什么是 log4j
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。

二、日志简介
日志指在程序中插入语句以提供调试信息。使用日志能够监视程序的执行。例如，用户利用日志可以获得关于应用程序故障的完整信息。用户可以将调试语句（如 System.out.println）插入到程序中以获得详细的调试信息。

三、项目中为什么要用 log4j
大家在编程时经常不可避免地要使用到一些日志操作，比如开发阶段的调试信息、运行时的日志记录及审计。调查显示，日志代码占代码总量的 4%。通常大家可以简单地使用 System.out.println()语句输出日志信息，但是往往会有一些判断，比如：

```
if (someCondition)
{
  System.out.println("some information.");
}
```

这些判断造成正常的程序逻辑中混杂了大量的输出语句。而在开发阶段写下的这些判断仅为 了调试的语句，在开发完成时需要查找并移除。部署运行后，尤其是在一些企业应用系统中，还经常需要进一步调试，这时就遇到了更大的麻烦。所以，我们需要一 套完备的、灵活的、可配置的日志工具 log4J 就是优秀的选择。

四、log4j 组件
Log4j 由 logger、appender 和 layout 三个组件组成。可以通过同名的 Java 类访问 Log4j 的这三个组件。

Logger - 在执行应用程序时，接收日志语句生成的日志请求。它是一种重要的日志处理组件， 可以通过 log4j API 的 logger 类对其进行访问。它的方法有：debug、info、warn、error、fatal 和 log。这些方法用于记录消息。
Appender - 管理日志语句的输出结果。执行日志语句时，Logger 对象将接收来自日志语句的记录请求。此请求是通过 logger 发送至 appender 的。然后，Appender 将输出结果写入到用户选择的目的地。对于不同的日志目的地，提供不同的 appender 类型。这些 appender 包括：用于文件的 file appender、用于数据库的 JDBC appender 和用于 SMTP 服务器的 SMTP appender。
Layout - 用于指定 appender 将日志语句写入日志目的地所采用的格式。appender 可以用来格式化输出结果的各种布局包括：简单布局、模式布局和 HTML 布局。

log4j 常用配置过程

常用 log4j 配置，一般可以采用两种方式，.properties 和.xml,下面举两个简单的例子：

一、 log4j.properties

```
### 设置 org.zblog 域对应的级别 INFO,DEBUG,WARN,ERROR 和输出地 A1，A2 ##
log4j.category.org.zblog=ERROR,A1
log4j.category.org.zblog=INFO,A2
log4j.appender.A1=org.apache.log4j.ConsoleAppender
### 设置输出地 A1，为 ConsoleAppender(控制台) ##
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
### 设置 A1 的输出布局格式 PatterLayout,(可以灵活地指定布局模式）##
log4j.appender.A1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} [%c]-[%p] %m%n
### 配置日志输出的格式##
log4j.appender.A2=org.apache.log4j.RollingFileAppender
### 设置输出地 A2 到文件（文件大小到达指定尺寸的时候产生一个新的文件）##
log4j.appender.A2.File=E:/study/log4j/zhuwei.html
### 文件位置##
log4j.appender.A2.MaxFileSize=500KB
### 文件大小##
log4j.appender.A2.MaxBackupIndex=1
log4j.appender.A2.layout=org.apache.log4j.HTMLLayout
##指定采用 html 方式输出
```

二、 log4j.xml

```
<?xml version="1.0" encoding="GB2312" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
<appender name="org.zblog.all" class="org.apache.log4j.RollingFileAppender">
<!-- 设置通道 ID:org.zblog.all 和输出方式：org.apache.log4j.RollingFileAppender -->
   <param name="File" value="E:/study/log4j/all.output.log" /><!-- 设置 File 参数：日志输出文件名 -->
   <param name="Append" value="false" /><!-- 设置是否在重新启动服务时，在原有日志的基础添加新日志 -->
   <param name="MaxBackupIndex" value="10" />
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" /><!-- 设置输出文件项目和格式 -->
   </layout>
</appender>
<appender name="org.zblog.zcw" class="org.apache.log4j.RollingFileAppender">
   <param name="File" value="E:/study/log4j/zhuwei.output.log" />
   <param name="Append" value="true" />
   <param name="MaxFileSize" value="10240" /> <!-- 设置文件大小 -->
   <param name="MaxBackupIndex" value="10" />
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" />
   </layout>
</appender>
```

```
<logger name="zcw.log"> <!-- 设置域名限制，即 zcw.log 域及以下的日志均输出到下面对应的通道中 -->
   <level value="debug" /><!-- 设置级别 -->
   <appender-ref ref="org.zblog.zcw" /><!-- 与前面的通道 id 相对应 -->
</logger>
<root> <!-- 设置接收所有输出的通道 -->
   <appender-ref ref="org.zblog.all" /><!-- 与前面的通道 id 相对应 -->
</root>
</log4j:configuration>
```

三、 配置文件加载方法：

```
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.xml.DOMConfigurator;
public class Log4jApp {
   public static void main(String[] args) {
      DOMConfigurator.configure("E:/study/log4j/log4j.xml");//加载.xml 文件
      //PropertyConfigurator.configure("E:/study/log4j/log4j.properties");//加载.properties 文件
      Logger log=Logger.getLogger("org.zblog.test");
      log.info("测试");
   }
}
```

四、 项目使用 log4j

在 web 应用中，可以将配置文件的加载放在一个单独的 servlet 中，并在 web.xml 中配置该 servlet 在应用启动时候加载。对于在多人项目中，可以给 每一个人设置一个输出通道，这样在每个人在构建 Logger 时，用自己的域名称，让调试信息输出到自己的 log 文件中。

五、 常用输出格式

```
# -X 号:X 信息输出时左对齐；
# %p:日志信息级别
# %d{}:日志信息产生时间
# %c:日志信息所在地（类名）
# %m:产生的日志具体信息
# %n:输出日志信息换行
```

LOG4J 开发步骤

首先，需要去下载 LOG4J 这个软件并解压缩出其中的 log4j.jar.在你的应用程序的 classpath 中包含该 JAR 文件，你也可以简单地将这个文件拷贝到 JDK 的%java_home%\lib\ext 目录下。

第一步，

Properties 文件(first_log4j.properties) 放在系统的 src 包中。（eclipse 布置在\WEB-INF\classes）

```
log4j.rootLogger=INFO,stdout,logfile

log4j.category.com.sumit=DEBUG
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern= [%p][%c]- %m [%d] %n
# logfile set up.
log4j.appender.logfile=org.apache.log4j.RollingFileAppender
log4j.appender.logfile.File=set up in web.xml
log4j.appender.logfile.MaxFileSize=50MB
# Keep three backup files.
log4j.appender.logfile.MaxBackupIndex=10
# Pattern to output: date priority [category] - message
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
#log4j.appender.logfile.layout=org.apache.log4j.HTMLLayout
#log4j.appender.logfile.layout.ConversionPattern= 时间:%d%n 优先级:%p%n 源文件:%F%n 类:%c%n 方法:%M%n 行数%L%n 最全:%l%n 信息:%m%n----------------------------------------------%n%n
log4j.appender.logfile.layout.ConversionPattern=%n- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - - %n 时间:%d%n 优先级:%p%n 位置:%l%n 信息:%m%n%n
s
```

第二步，

Servlet 的编写

```
package com.royal.listener;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class LogListener extends HttpServlet {

 private static final long serialVersionUID = 1L;

 static Logger log = Logger.getLogger(LogListener.class);
```

```
    public LogListener() {
    }

    public void init(ServletConfig config) throws ServletException {
        String prefix = config.getServletContext().getRealPath("/");

        String properties = config.getInitParameter("log4j_properties");
        String logger=config.getInitParameter("log4j_outPutFile");

        String propertiesPath = prefix + properties;
        Properties props = new Properties();
        try {
            FileInputStream istream = new FileInputStream(propertiesPath);
            props.load(istream);
            istream.close();
            //toPrint(props.getProperty("log4j.appender.file.File"));
            String logFile = prefix + logger;//设置路径
            logFile=logFile.replace('\\', '/');

            props.setProperty("log4j.appender.logfile.File",logFile);


            PropertyConfigurator.configure(props);//装入 log4j 配置信息

            log.info("LOG4J 日志------");

            log.info("LOG 被加载  日志文件位置："+props.getProperty("log4j.appender.logfile.File"));

        } catch (IOException ioe) {
         log.error(ioe);
  ioe.printStackTrace();
            return;
        }
    }
}
```

Web.xml: (load-on-srartup=1 项目启动时加载)参数中有一个是 logs\log.log，自动生成日志文件，生成在项目下面的 logs directory 中，文件名为 log.log

```
<servlet>
        <servlet-name>LogListener</servlet-name>
        <servlet-class>com.royal.listener.LogListener</servlet-class>
        <init-param>
          <param-name>log4j_properties</param-name>
          <param-value>WEB-INF/classes/first_log4j.properties</param-value>
        </init-param>
        <init-param>
          <param-name>log4j_outPutFile</param-name>
          <param-value>logs\log.log</param-value>
        </init-param>
```

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

第三步
使用

如果是普通的 JAVA 类，就得初始化 log 变量 static Logger log = Logger.getLogger(Mail.class);在要记录日去的地方 log.error(ex+" 创建 MIME 邮件对象失败！"); //ex 是 Exception 对象或者 log.info/log.debug/log.warn 等。如果是 struts 中的 DispatchAction 的子类，或其它有 Log 的类的子类。就可以不用初始化 log 变量。

Log4j 比较完整的配置

LOG4J 的配置之简单使它遍及于越来越多的应用中了：Log4J 配置文件实现了输出到控制台、文件、回滚文件、发送日志邮件、输出到数据库日志表、自定义标签等全套功能。择其一二使用就够用了

```
log4j.rootLogger=DEBUG,CONSOLE,A1,im
log4j.addivity.org.apache=true

# 应用于控制台
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.Threshold=DEBUG
log4j.appender.CONSOLE.Target=System.out
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n
#log4j.appender.CONSOLE.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p[PRIORITY]%n%x[NDC]%n%t[THREAD] n%c[CATEGORY]%n%m[MESSAGE]%n%n

#应用于文件
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=file.log
log4j.appender.FILE.Append=false
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n
# Use this layout for LogFactor 5 analysis


# 应用于文件回滚
log4j.appender.ROLLING_FILE=org.apache.log4j.RollingFileAppender
log4j.appender.ROLLING_FILE.Threshold=ERROR
log4j.appender.ROLLING_FILE.File=rolling.log
log4j.appender.ROLLING_FILE.Append=true
log4j.appender.ROLLING_FILE.MaxFileSize=10KB
log4j.appender.ROLLING_FILE.MaxBackupIndex=1
log4j.appender.ROLLING_FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.ROLLING_FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n

#应用于 socket
log4j.appender.SOCKET=org.apache.log4j.RollingFileAppender
log4j.appender.SOCKET.RemoteHost=localhost
log4j.appender.SOCKET.Port=5001
log4j.appender.SOCKET.LocationInfo=true
# Set up for Log Facter 5
log4j.appender.SOCKET.layout=org.apache.log4j.PatternLayout
log4j.appender.SOCET.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p[PRIORITY]%n%x[NDC]%n%t[THREAD]%n%c[CATEGORY]%n%m[MESSAGE]%n%n


# Log Factor 5 Appender
```

```
log4j.appender.LF5_APPENDER=org.apache.log4j.lf5.LF5Appender
log4j.appender.LF5_APPENDER.MaxNumberOfRecords=2000

# 发送日志给邮件
log4j.appender.MAIL=org.apache.log4j.net.SMTPAppender
log4j.appender.MAIL.Threshold=FATAL
log4j.appender.MAIL.BufferSize=10
log4j.appender.MAIL.From=web@www.wuset.com
log4j.appender.MAIL.SMTPHost=www.wusetu.com
log4j.appender.MAIL.Subject=Log4J Message
log4j.appender.MAIL.To=web@www.wusetu.com
log4j.appender.MAIL.layout=org.apache.log4j.PatternLayout
log4j.appender.MAIL.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n


# 用于数据库
log4j.appender.DATABASE=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DATABASE.URL=jdbc:mysql://localhost:3306/test
log4j.appender.DATABASE.driver=com.mysql.jdbc.Driver
log4j.appender.DATABASE.user=root
log4j.appender.DATABASE.password=
log4j.appender.DATABASE.sql=INSERT INTO LOG4J (Message) VALUES ('[framework] %d - %c -%-4r
[%t] %-5p %c %x - %m%n')
log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout
log4j.appender.DATABASE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x -
%m%n


log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=SampleMessages.log4j
log4j.appender.A1.DatePattern=yyyyMMdd-HH'.log4j'
log4j.appender.A1.layout=org.apache.log4j.xml.XMLLayout

#自定义 Appender
log4j.appender.im = net.cybercorlin.util.logger.appender.IMAppender

log4j.appender.im.host = mail.cybercorlin.net
log4j.appender.im.username = username
log4j.appender.im.password = password
log4j.appender.im.recipient = corlin@cybercorlin.net

log4j.appender.im.layout=org.apache.log4j.PatternLayout
log4j.appender.im.layout.ConversionPattern =[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n
```

LOG4J 配置全接触

LOG4J 的配置之简单使它遍及于越来越多的应用中了：**Log4J** 配置文件实现了输出到控制台、文件、回滚文件、发送日志邮件、输出到数据库日志表、自定义标签等全套功能。择其一二使用就够用了，

```
log4j.rootLogger=DEBUG,CONSOLE,A1,im
log4j.addivity.org.apache=true

# 应用于控制台
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.Threshold=DEBUG
log4j.appender.CONSOLE.Target=System.out
log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
log4j.appender.CONSOLE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t]
%-5p %c %x - %m%n
#log4j.appender.CONSOLE.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p[P
RIORITY]%n%x[NDC]%n%t[THREAD] n%c[CATEGORY]%n%m[MESSAGE]%n%n

#应用于文件
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=file.log
log4j.appender.FILE.Append=false
log4j.appender.FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.FILE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p
%c %x - %m%n
# Use this layout for LogFactor 5 analysis

# 应用于文件回滚
log4j.appender.ROLLING_FILE=org.apache.log4j.RollingFileAppender
log4j.appender.ROLLING_FILE.Threshold=ERROR
log4j.appender.ROLLING_FILE.File=rolling.log
log4j.appender.ROLLING_FILE.Append=true
log4j.appender.ROLLING_FILE.MaxFileSize=10KB
log4j.appender.ROLLING_FILE.MaxBackupIndex=1
log4j.appender.ROLLING_FILE.layout=org.apache.log4j.PatternLayout
log4j.appender.ROLLING_FILE.layout.ConversionPattern=[framework] %d - %c -%-4r
[%t] %-5p %c %x - %m%n

#应用于 socket
log4j.appender.SOCKET=org.apache.log4j.RollingFileAppender
log4j.appender.SOCKET.RemoteHost=localhost
log4j.appender.SOCKET.Port=5001
log4j.appender.SOCKET.LocationInfo=true
# Set up for Log Facter 5
log4j.appender.SOCKET.layout=org.apache.log4j.PatternLayout
log4j.appender.SOCET.layout.ConversionPattern=[start]%d{DATE}[DATE]%n%p[PRIOR
ITY]%n%x[NDC]%n%t[THREAD]%n%c[CATEGORY]%n%m[MESSAGE]%n%n
```

```
# Log Factor 5 Appender
log4j.appender.LF5_APPENDER=org.apache.log4j.lf5.LF5Appender
log4j.appender.LF5_APPENDER.MaxNumberOfRecords=2000

# 发送日志给邮件
log4j.appender.MAIL=org.apache.log4j.net.SMTPAppender
log4j.appender.MAIL.Threshold=FATAL
log4j.appender.MAIL.BufferSize=10
log4j.appender.MAIL.From=web@www.wuset.com
log4j.appender.MAIL.SMTPHost=www.wusetu.com
log4j.appender.MAIL.Subject=Log4J Message
log4j.appender.MAIL.To=web@www.wusetu.com
log4j.appender.MAIL.layout=org.apache.log4j.PatternLayout
log4j.appender.MAIL.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p
%c %x - %m%n

# 用于数据库
log4j.appender.DATABASE=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DATABASE.URL=jdbc:mysql://localhost:3306/test
log4j.appender.DATABASE.driver=com.mysql.jdbc.Driver
log4j.appender.DATABASE.user=root
log4j.appender.DATABASE.password=
log4j.appender.DATABASE.sql=INSERT INTO LOG4J (Message) VALUES ('[framework]
%d - %c -%-4r [%t] %-5p %c %x - %m%n')
log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout
log4j.appender.DATABASE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t]
%-5p %c %x - %m%n

log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=SampleMessages.log4j
log4j.appender.A1.DatePattern=yyyyMMdd-HH'.log4j'
log4j.appender.A1.layout=org.apache.log4j.xml.XMLLayout

#自定义 Appender
log4j.appender.im = net.cybercorlin.util.logger.appender.IMAppender
log4j.appender.im.host = mail.cybercorlin.net
log4j.appender.im.username = username
log4j.appender.im.password = password
log4j.appender.im.recipient = corlin@cybercorlin.net
```

单实例模式数据库连接池+log4j

```
package cn.ifnic.db;
import java.io.*;
import java.sql.*;
import java.util.*;
import java.util.Date;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import cn.ifnic.util.Common;
import cn.ifnic.util.GetProperties;
/**
 *
 * <p>Title: IFNIC</p>
 *
 * <p>Description: 类 DBConnectionManager，连接池的管理类</p>
 *
 * <p>Copyright: Copyright (c) 2005 www.ifnic.cn</p>
 *
 * <p>Company: IFNIC.CN</p>
 *
 * @author IFNIC
 * @version 1.1.0
 */
/**
 * 管理类 DBConnectionManager 支持对一个或多个由属性文件定义的数据库连接
 * 池的访问.客户程序可以调用 getInstance()方法访问本类的唯一实例.
 */
public class DBConnectionManager {
    static private DBConnectionManager instance; // 唯一实例
    static private int clients;
    private Vector drivers = new Vector();
    private Logger logger;
    //private PrintWriter log;
    private Hashtable pools = new Hashtable();
    /**
     * 返回唯一实例.如果是第一次调用此方法,则创建实例
     *
     * @return DBConnectionManager 唯一实例
     */
    static synchronized public DBConnectionManager getInstance() {
        if (instance == null) {
            instance = new DBConnectionManager();
        }
        clients++;
        return instance;
    }
    /**
     * 建构函数私有以防止其它对象创建本类实例
```

```
 */
private DBConnectionManager() {
    init();
}
/**
 * 将连接对象返回给由名字指定的连接池
 *
 * @param name 在属性文件中定义的连接池名字
 * @param con 连接对象
 */
public void freeConnection(String name, Connection con) {
    DBConnectionPool pool = (DBConnectionPool) pools.get(name);
    if (pool != null) {
        pool.freeConnection(con);
    }
}
/**
 * 获得一个可用的(空闲的)连接.如果没有可用连接,且已有连接数小于最大连接数
 * 限制,则创建并返回新连接
 *
 * @param name 在属性文件中定义的连接池名字
 * @return Connection 可用连接或 null
 */
public Connection getConnection(String name) {
    DBConnectionPool pool = (DBConnectionPool) pools.get(name);
    if (pool != null) {
        return pool.getConnection();
    }
    return null;
}
/**
 * 获得一个可用连接.若没有可用连接,且已有连接数小于最大连接数限制,
 * 则创建并返回新连接.否则,在指定的时间内等待其它线程释放连接.
 *
 * @param name 连接池名字
 * @param time 以毫秒计的等待时间
 * @return Connection 可用连接或 null
 */
public Connection getConnection(String name, long time) {
    DBConnectionPool pool = (DBConnectionPool) pools.get(name);
    if (pool != null) {
        return pool.getConnection(time);
    }
    return null;
}
/**
 * 关闭所有连接,撤销驱动程序的注册
 */
public synchronized void release() {
    // 等待直到最后一个客户程序调用
```

```
        if (--clients != 0) {
            return;
        }
        Enumeration allPools = pools.elements();
        while (allPools.hasMoreElements()) {
            DBConnectionPool pool = (DBConnectionPool) allPools.nextElement();
            pool.release();
        }
        Enumeration allDrivers = drivers.elements();
        while (allDrivers.hasMoreElements()) {
            Driver driver = (Driver) allDrivers.nextElement();
            try {
                DriverManager.deregisterDriver(driver);
                logger.info("撤销 JDBC 驱动程序 " + driver.getClass().getName() + "的注册");
            } catch (SQLException e) {
                logger.error("无法撤销下列 JDBC 驱动程序的注册: " +
driver.getClass().getName(),
                        e.getNextException());
            }
        }
    }
    /**
     * 根据指定属性创建连接池实例.
     *
     * @param props 连接池属性
     */
    private void createPools(Properties props) {
        Enumeration propNames = props.propertyNames();
        while (propNames.hasMoreElements()) {
            String name = (String) propNames.nextElement();
            if (name.endsWith(".url")) {
                String poolName = name.substring(0, name.lastIndexOf("."));
                String url = props.getProperty(poolName + ".url");
                if (url == null) {
                    logger.error("没有为连接池" + poolName + "指定 URL");
                    continue;
                }
                String user = props.getProperty(poolName + ".user");
                String password = props.getProperty(poolName + ".password");
                String maxconn = props.getProperty(poolName + ".maxconn", "0");
                int max;
                try {
                    max = Integer.valueOf(maxconn).intValue();
                } catch (NumberFormatException e) {
                    logger.error("错误的最大连接数限制: " + maxconn + " .连接池: " +
                            poolName);
                    max = 0;
                }
                DBConnectionPool pool =
                        new DBConnectionPool(poolName, url, user, password, max);
```

```
            pools.put(poolName, pool);
            logger.info("成功创建连接池" + poolName);
        }
    }
}
/**
 * 读取属性完成初始化
 */
private void init() {
    GetProperties getProperties = new GetProperties();
    if (getProperties.getProperties("DBConnectionManager") != null) {
        PropertyConfigurator.configure(getProperties.getProperties(
                "DBConnectionManager"));
    }
    logger = Logger.getLogger(Common.class.getName());
    if (getProperties.getProperties("db") != null) {
        Properties pp=getProperties.getProperties("db");
        loadDrivers(pp);
        createPools(pp);
    } else {
        return;
    }
}
/**
 * 装载和注册所有 JDBC 驱动程序
 *
 * @param props 属性
 */
private void loadDrivers(Properties props) {
    String driverClasses = props.getProperty("drivers");
    StringTokenizer st = new StringTokenizer(driverClasses);
    while (st.hasMoreElements()) {
        String driverClassName = st.nextToken().trim();
        try {
            Driver driver = (Driver)
                        Class.forName(driverClassName).newInstance();
            DriverManager.registerDriver(driver);
            drivers.addElement(driver);
            logger.info("成功注册 JDBC 驱动程序" + driverClassName);
        } catch (Exception e) {
            logger.error("无法注册 JDBC 驱动程序: " +
                    driverClassName + ", 错误: " + e);
        }
    }
}

/**
 * 此内部类定义了一个连接池.它能够根据要求创建新连接,直到预定的最
 * 大连接数为止.在返回连接给客户程序之前,它能够验证连接的有效性.
 */
```

```
class DBConnectionPool {
    private int checkedOut;
    private Vector freeConnections = new Vector();
    private int maxConn;
    private String name;
    private String password;
    private String URL;
    private String user;
    private Logger logger;
    /**
     * 创建新的连接池
     *
     * @param name 连接池名字
     * @param URL 数据库的 JDBC URL
     * @param user 数据库帐号,或 null
     * @param password 密码,或 null
     * @param maxConn 此连接池允许建立的最大连接数
     */
    public DBConnectionPool(String name, String URL, String user,
                            String password,
                            int maxConn) {
        this.name = name;
        this.URL = URL;
        this.user = user;
        this.password = password;
        this.maxConn = maxConn;
        GetProperties getProperties = new GetProperties();
        if (getProperties.getProperties("DBConnectionManager") != null) {
            PropertyConfigurator.configure(getProperties.getProperties(
                    "DBConnectionManager"));
            logger = Logger.getLogger(Common.class.getName());
        } else {
            return;
        }
    }
    /**
     * 将不再使用的连接返回给连接池
     *
     * @param con 客户程序释放的连接
     */
    public synchronized void freeConnection(Connection con) {
        // 将指定连接加入到向量末尾
        freeConnections.addElement(con);
        checkedOut--;
        notifyAll();
    }
    /**
     * 从连接池获得一个可用连接.如没有空闲的连接且当前连接数小于最大连接
     * 数限制,则创建新连接.如原来登记为可用的连接不再有效,则从向量删除之,
     * 然后递归调用自己以尝试新的可用连接.
```

```
 */
public synchronized Connection getConnection() {
    Connection con = null;
    if (freeConnections.size() > 0) {
        // 获取向量中第一个可用连接
        con = (Connection) freeConnections.firstElement();
        freeConnections.removeElementAt(0);
        try {
            if (con.isClosed()) {
                logger.info("从连接池" + name + "删除一个无效连接");
                // 递归调用自己,尝试再次获取可用连接
                con = getConnection();
            }
        } catch (SQLException e) {
            logger.warn("从连接池" + name + "删除一个无效连接");
            // 递归调用自己,尝试再次获取可用连接
            con = getConnection();
        }
    } else if (maxConn == 0 || checkedOut < maxConn) {
        con = newConnection();
    }
    if (con != null) {
        checkedOut++;
    }
    return con;
}
/**
 * 从连接池获取可用连接.可以指定客户程序能够等待的最长时间
 * 参见前一个 getConnection()方法.
 *
 * @param timeout 以毫秒计的等待时间限制
 */
public synchronized Connection getConnection(long timeout) {
    long startTime = new Date().getTime();
    Connection con;
    while ((con = getConnection()) == null) {
        try {
            wait(timeout);
        } catch (InterruptedException e) {}
        if ((new Date().getTime() - startTime) >= timeout) {
            // wait()返回的原因是超时
            return null;
        }
    }
    return con;
}
/**
 * 关闭所有连接
 */
public synchronized void release() {
```

```
            Enumeration allConnections = freeConnections.elements();
            while (allConnections.hasMoreElements()) {
               Connection con = (Connection) allConnections.nextElement();
               try {
                  con.close();
                  logger.info("关闭连接池" + name + "中的一个连接");
               } catch (SQLException e) {
                  logger.error("无法关闭连接池" + name + "中的连接", e.getNextException());
               }
            }
            freeConnections.removeAllElements();
         }
         /**
          * 创建新的连接
          */
         private Connection newConnection() {
            Connection con = null;
            try {
               if (user == null) {
                  con = DriverManager.getConnection(URL);
               } else {
                  con = DriverManager.getConnection(URL, user, password);
               }
               logger.info("连接池" + name + "创建一个新的连接");
            } catch (SQLException e) {
               logger.error("无法创建下列 URL 的连接: " + URL, e.getNextException());
               return null;
            }
            return con;
         }
      }
   }
```

```
//PropertiesLoader.java 配置文件读取类

package cn.ifnic.util;
import java.util.Properties;
import java.io.InputStream;
import java.io.*;
/**
 *
 * <p>Title: IFNIC</p>
 *
 * <p>Description: 类 GetProperties，用来获取*.properties 配置文件的</p>
 *
 * <p>Copyright: Copyright (c) 2005 www.ifnic.cn</p>
 *
 * <p>Company: IFNIC.CN</p>
```

```
 *
 * @author IFNIC
 * @version 1.1.0
 */
public class PropertiesLoader {
    /**
     * 用法：getProperties（*.properties 配置文件名），返回一个 Properties 对象
     * @param propertiesName String
     * @return Properties
     */
    public Properties getProperties(String propertiesName) {
        InputStream is = getClass().getResourceAsStream("/" + propertiesName +
                ".properties");
        Properties dbProps = new Properties();
        try {
            dbProps.load(is);
            return dbProps;
        } catch (IOException ex) {
            System.out.println("不能读取属性文件. " +
                               "请确保" + propertiesName +
                               ".properties 在 CLASSPATH 指定的路径中");
            return null;
        }
    }
}
```

//db.properties 连接池配置文件

```
drivers=net.sourceforge.jtds.jdbc.Driver #为连接池指定一个数据库连接驱动程序

jtds_root.url=jdbc:jtds:sqlserver://localhost/root #数据库连接 URL
jtds_root.maxconn=0 #连接最大数 为 0 表示没有限制
jtds_root.user=sa #数据库用户名
jtds_root.password=sa #数据库密码
```

//DBConnectionManager.properties Log4j 配置文件

```
log4j.rootLogger=INFO, Common #定义被日志捕获的信息级别
#ERROR、WARN、INFO、DEBUG
log4j.appender.Common=org.apache.log4j.RollingFileAppender #定义日志的存储方式
log4j.appender.Common.File = logs/DBConnectionManagerLog.html #定义日志文件路径
log4j.appender.Common.Append=true #定义是否追加 否为覆盖
log4j.appender.Common.MaxFileSize = 1024kb #定义文件大小
log4j.appender.Common.MaxBackupIndex=10 #定义最大备份文件数
log4j.appender.Common.layout=org.apache.log4j.HTMLLayout #定义输出格式为 HTML 格式。
```

Log4j 的使用总结

1 关于 **log4j** 的文章**---**粗略看了一遍，不错！
出处给忘了好像是下面这个：
http://dev2dev.bea.com.cn/bbs/servlet/D2DServlet/download/124-14026-77822-650/
深入学习 **Log4J.pdf**
深入学习 **Log4J**
李翔

内容**:**
一**,Log4J** 配置文件的学习
二**,Log4J** 数据库
三**,Log4J** 封装
一**,Log4J** 配置文件学习**:**
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**).**
下面我们首先介绍使用 **Java** 特性文件做为配置文件的方法**:**
分析一个配置文件 **log4j.properties**

```
log4j.rootCategory=debug, stdout, R
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
# Pattern to output the caller's file name and line number.
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=example.log
log4j.appender.R.MaxFileSize=100KB
# Keep one backup file
log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%p %t %c - %m%n
```

说明**:**
①**log4j.rootCategory = [level] , appenderName, appenderName,**
其中**,level** 是日志记录的优先级**,**分为 **OFF,FATAL,ERROR,WA R N,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,**
WA R N,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** 就是指定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** 有以下几种**:**
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**)**
④**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,WA R N,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**.**
对上面 **log4j.properties** 配置文件的一个应用**;**

```
 package log4j;
import org.apache.log4j.*;
// How to use log4j
public class TestLogging {
// Initialize a logging category. Here, we get THE ROOT CATEGORY
//static Category cat = Category.getRoot();
// Or, get a custom category
static Category cat = Category.getInstance(TestLogging.class.getName());
// From here on, log away! Methods are: cat.debug(your_message_string),
// cat.info(...), cat.warn(...), cat.error(...), cat.fatal(...)
public static void main(String args[]) {
// Try a few logging methods
PropertyConfigurator.configure ( "log4j.properties" ) ;
cat.debug("Start of main()");
cat.info("Just testing a log message with priority set to INFO");
cat.warn("Just testing a log message with priority set to WARN");
cat.error("Just testing a log message with priority set to ERROR");
cat.fatal("Just testing a log message with priority set to FATAL");
// Alternate but INCONVENIENT form
cat.log(Priority.DEBUG, "Calling init()");
new TestLogging().init();
}
public void init() {
java.util.Properties prop = System.getProperties();
java.util.Enumeration enum = prop.propertyNames();
cat.info("***System Environment As Seen By Java***");
```

```
cat.debug("***Format: PROPERTY = VALUE***");
while (enum.hasMoreElements()) {
String key = (String) enum.nextElement();
cat.info(key + " = " + System.getProperty(key));
}
}
}
```

xml 格式的 **log4j** 配置文件概述

xml格式的**log4j**配置文件需要使用**org.apache.log4j.html.DOMConfigurator.configure()**方法来读入**.**对 **xml** 文件的语法定义可以在 **log4j** 的发布包中找到**:org/apache/log4j/xml/log4j.dtd.**

Xml 的一个配置文件**:sample1.xml**

说明**:**

①**xml** 配置文件的头部包括两个部分**:xml** 声明和 **dtd** 声明**.**头部的格式如下**:**

②**log4j:configuration (root element)**

xmlns:log4j [#FIXED attribute]: 定义 **log4j** 的名字空间**,**取定值 **"http://jakarta.apache.org/log4j/"**

appender [* child] : 一个 **appender** 子元素定义一个日志输出目的地

logger [* child] : 一个 **logger** 子元素定义一个日志写出器

root [child] : root 子元素定义了 **root logger**

源代码**:**

```
package exampleslog4j.xml;
import org.apache.log4j.xml.DOMConfigurator;
import org.apache.log4j.Category;
import java.net.*;
public class XMLSample {
static Category cat = Category.getInstance(XMLSample.class.getName());
public
static
void main(String argv[]) {
if(argv.length == 1)
init(argv[0]);
else
Usage("Wrong number of arguments.");
sample();
}
static
void Usage(String msg) {
System.err.println(msg);
System.err.println( "Usage: java " + XMLSample.class.getName() +
"configFile");
System.exit(1);
}
static
void init(String configFile) {
DOMConfigurator.configure(configFile);
}
static
void sample() {
```

```
int i = -1;
Category root = Category.getRoot();
cat.debug("Message " + ++i);
cat.warn ("Message " + ++i);
cat.error("Message " + ++i);
Exception e = new Exception("Just testing");
cat.debug("Message " + ++i, e);
}
}
```

执行后的效果:

```
2004-05-24 22:07:28,352 DEBUG [main] xml.XMLSample (XMLSample.java:55) -
Message 0
2004-05-24 22:07:28,352 WARN [main] xml.XMLSample (XMLSample.java:56) -
Message 1
```

二,Log4J 对数据库的操作:

其目的就是把日志信息写入数据库以方便开发人员和测试人员查询.

下面是写入数据库的配置文件:log4j.properties

```
log4j.appender.DATABASE=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.DATABASE.URL=jdbc:oracle:thin:@192.168.0.1:1521:siemen
log4j.appender.DATABASE.driver= oracle.jdbc.driver.OracleDriver
log4j.appender.DATABASE.user=system
log4j.appender.DATABASE.password=css12345
log4j.appender.DATABASE.sql=INSERT INTO LOG4J (Message) VALUES ('[framework]
%d
- %c -%-4r [%t] %-5p %c %x - %m%n')
log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout
log4j.appender.DATABASE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t]
%-5p %c %x - %m%n
log4j.appender.A1=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A1.File=SampleMessages.log4j
log4j.appender.A1.DatePattern=yyyyMMdd-HH'.log4j'
log4j.appender.A1.layout=org.apache.log4j.xml.XMLLayout
```

对其应用的源文件:

```
package database.servlet;
import java.io.File;
import java.io.LineNumberReader;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.Vector;
import java.sql.Driver;
import java.sql.DriverManager;
// import servlet packages
import javax.servlet.http.HttpServlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
// import log4j packages
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
public class SetupServlet extends HttpServlet{
public void init(ServletConfig config) throws ServletException{
```

```
super.init(config);
// first thing to do, is to set up the Driver that we might be using
// in case of JDBCAppender
try{
//Driver d =(Driver)(Class.forName("org.gjt.mm.mysql.Driver").newInstance());
Driver d = (Driver)(Class.forName("oracle.jdbc.driver.OracleDriver").newInstance());
DriverManager.registerDriver(d);
//加载 JDBC 驱动程序,当准备将日志记录到数据库的时候可以使用
}catch(Exception e){ System.err.println(e); }
// next load up the properties
//启动时从 web.xml 中获得配置文件的信息
String props = config.getInitParameter("props");
if(props == null || props.length() == 0 ||
!(new File(props)).isFile()){
System.err.println(
"ERROR: Cannot read the configuration file. " +
"Please check the path of the config init param in web.xml");
throw new ServletException();
}
}
public void destroy(){
super.destroy();
}
}
```

三,Log4J 的封装:

配置文件:log4j.properties

```
log4j.rootLogger=DEBUG, A2, A1
log4j.appender.A2=org.apache.log4j.RollingFileAppender
log4j.appender.A2.File=C:\develop\log\error.log
log4j.appender.A2.Append=true
log4j.appender.R.MaxFileSize=10000KB
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
```

4 关于 log4j 的文章---粗略看了一遍，不错!

```
 log4j.appender.A2.layout.ConversionPattern=[%-5p][%t] %d{yyyy-MM-dd
HH:mm:ss,SSS} message:%m%n
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
#Pattern to output the caller's file name and line number.
#log4j.appender.stdout.layout.ConversionPattern=%5p [%t] (%F:%L) - %m%n
# Print the date in ISO 8601 format
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p - %m%n
```

EncapsulationLog4J.java //Log4j 的实现类

```
package com.cn.lx;
/**
*
Title:
*
Description:
```

```
*
Copyright: Copyright © 2004 lixiang
*
Company:http://www.css.com.cn/
* @author lixiang
* @version 1.0
*/
import org.apache.log4j.*;
import java.io.*;
import java.util.*;
/**
* @author Administrator
*
* To change the template for this generated type comment go to
* Window>Preferences>Java>Code Generation>Code and Comments
*/
public class EncapsulationLog4J
{
public static final String PROFILE = "log4j.properties";
/**
* Holds singleton instance
*/
private static EncapsulationLog4J impl;
static
{
impl = new EncapsulationLog4J();
}
private Logger log4j;
/**
* prevents instantiation
*/
private EncapsulationLog4J()
{
log4j = LogManager.getLogger(EncapsulationLog4J.class);
try
{
Properties pro = new Properties();
InputStream is = getClass().getResourceAsStream(PROFILE);
pro.load(is);
PropertyConfigurator.configure(pro);
}
catch(IOException e)
{
BasicConfigurator.configure();
e.printStackTrace();
}
}
public void log(String level,Object msg)
{
log(level,msg,null);
```

```
}
public void log(String level,Throwable e)
{
log(level,null,e);
}
public void log(String level,Object msg,java.lang.Throwable e)
{
if(log4j != null)
{
log4j.log((Priority)Level.toLevel(level),msg,e);
}
}
/**
* Singleton Pattern
*/
static public EncapsulationLog4J getInstance()
{
return impl;
}
}
```

Log.java //记录 Log 使用类

```
package com.cn.lx;
/**
*
Title:
*
Description:
*
Copyright: Copyright © 2004 lixiang
*
Company: http://www.css.com.cn/
* @author lixiang
* @version 1.0
*/
public class Log
{
private static EncapsulationLog4J log = EncapsulationLog4J.getInstance();
/**
*
*/
public Log()
{
//super();
}
public static void logError(String msg)
{
log.log("ERROR",msg);
}
public static void logError(Throwable e)
```

```
{
log.log("ERROR",null,e);
}
public static void logWarn(String msg)
{
log.log("WARN",msg);
}
public static void logWarn(Throwable e)
{
log.log("WARN",null,e);
}
public static void logInfo(String msg)
{
log.log("INFO",msg);
}
public static void logInfo(Throwable e)
{
log.log("INFO",null,e);
}
public static void logDebug(String msg)
{
log.log("DEBUG",msg);
}
public static void logDebug(Throwable e)
{
log.log("DEBUG",null,e);
}
}
```

TestLog.java //调用 Log 类

```
package com.cn.lx;
public class TestLog{
public static void main(String[] args) {
Log test = new Log();
test.logDebug("DEBUG");
test.logInfo("INFO");
test.logWarn("WARN");
test.logError("ERROR");
try
{
int i = Integer.parseInt("lixiang");
}catch(Exception e)
{
test.logDebug(e.toString());
test.logInfo(e.toString());
test.logWarn(e.toString());
test.logError(e.toString());
}
}
}
```

执行后的日志信息:

2004-05-26 21:16:16,474 [main] DEBUG - DEBUG
2004-05-26 21:16:16,484 [main] INFO - INFO
2004-05-26 21:16:16,484 [main] WARN - WARN
2004-05-26 21:16:16,484 [main] ERROR - ERROR
2004-05-26 21:16:16,484 [main] DEBUG - java.lang.NumberFormatException: For input string:
"lixiang"
2004-05-26 21:16:16,484 [main] INFO - java.lang.NumberFormatException: For input string:
"lixiang"
2004-05-26 21:16:16,484 [main] WARN - java.lang.NumberFormatException: For input string:
"lixiang"
2004-05-26 21:16:16,484 [main] ERROR - java.lang.NumberFormatException: For input string:
"lixiang"
注**:**使用此方法封装 **Log4j** 的操作**,**可以使记录日志变得更方便**.**唯一不足的是无法返回当

Spring 对 log4j 的增强

Spring 最擅长的，就是在别家的蛋糕上再加些 cream，让你 J2EE without Spring 的时候心痒痒。

log4j，可以有如下的 cream：
1. 动态的改变记录级别和策略，不需要重启 Web 应用，如《Effective Enterprise Java》所说。
2. 把 log 文件定在 /WEB-INF/logs/ 而不需要写绝对路径。
3. 可以把 log4j.properties 和其他 properties 一起放在/WEB-INF/ ，而不是 Class-Path。

在 web.xml 添加

```
<context-param>
    <param-name>log4jConfigLocation</param-name>
    <param-value>WEB-INF/log4j.properties</param-value>
</context-param>

<context-param>
    <param-name>log4jRefreshInterval</param-name>
    <param-value>60000</param-value>
</context-param>

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

在上文的配置里，
Log4jConfigListener 会去 WEB-INF/log4j.propeties 读取配置文件；
开一条 watchdog 线程每 60 秒扫描一下配置文件的变化；
并把 web 目录的路径压入一个叫 webapp.root 的系统变量。

然后，在 log4j.properties 里就可以这样定义 logfile 位置

```
log4j.appender.logfile.File=${webapp.root}/WEB-INF/logs/myfuse.log
```

如果有多个 web 应用，怕 webapp.root 变量重复，可以在 context-param 里定义 webAppRootKey。

log4j 在 jbuilderX 中的配置

网络上关于此配置的文章有很多，可是大部分都不全面，也不能真正的用到实处。现将我昨天配置的步骤写下来，希望对你有所帮助。

1、 打开 **jbuilderX**，然后点击 **tools-**〉**configure libraries-**〉**user home->new->**然后选择 **log4j** 的 **jar** 文件，**jbuilder** 一般自带有 **log4j-1.2.8.jar** 版本，具体位置在 **/jbuilder/lib** 目录下，此时可以将此文件添加到 **user home** 中，也可以自己下载最新的版本，现在是 **log4j-1.3alpha-3.jar** 是最新的。同样可以将此文件用同样的步骤添加进来。

2、然后打开 **project->project properties->required libraries**，单击 **add**，将刚才添加的 **user home** 项选择进去即可。

3、测试，编写程序如下：

```
import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
 public class HelloWorld1{

 static Logger logger=Logger.getLogger("HelloWorld1");
   static public void main(String[] args){
 BasicConfigurator.configure();
logger.debug("Hello World.");
}
}
```

然后让此文件单独运行即可看到结果：

```
0 [main] DEBUG HelloWorld1 null-Hello World.
```

就是这么简单。

常用 log4j 配置

常用 **log4j** 配置，一般可以采用两种方式，**.properties** 和**.xml,**下面举两个简单的例子：

一、**log4j.properties**

```
### 设置 org.zblog 域对应的级别 INFO,DEBUG,WARN,ERROR 和输出地 A1，A2 ##
log4j.category.org.zblog=ERROR,A1
log4j.category.org.zblog=INFO,A2
log4j.appender.A1=org.apache.log4j.ConsoleAppender
### 设置输出地 A1，为 ConsoleAppender(控制台) ##
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
### 设置 A1 的输出布局格式 PatterLayout,(可以灵活地指定布局模式）##
log4j.appender.A1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS}
[%c]-[%p] %m%n
### 配置日志输出的格式##
log4j.appender.A2=org.apache.log4j.RollingFileAppender
### 设置输出地 A2 到文件（文件大小到达指定尺寸的时候产生一个新的文件）##
log4j.appender.A2.File=E:/study/log4j/zhuwei.html
### 文件位置##
log4j.appender.A2.MaxFileSize=500KB
### 文件大小##
log4j.appender.A2.MaxBackupIndex=1
log4j.appender.A2.layout=org.apache.log4j.HTMLLayout
##指定采用 html 方式输出
```

二、**log4j.xml**

```
<?xml version="1.0" encoding="GB2312" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
<appender name="org.zblog.all" class="org.apache.log4j.RollingFileAppender">
<!-- 设置通道 ID:org.zblog.all 和输出方式：org.apache.log4j.RollingFileAppender -->
   <param name="File" value="E:/study/log4j/all.output.log" /><!-- 设置 File 参数：日志
输出文件名 -->
   <param name="Append" value="false" /><!-- 设置是否在重新启动服务时，在原有日志的基
础添加新日志 -->
   <param name="MaxBackupIndex" value="10" />
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" /><!-- 设置输出
文件项目和格式 -->
   </layout>
</appender>
<appender name="org.zblog.zcw" class="org.apache.log4j.RollingFileAppender">
   <param name="File" value="E:/study/log4j/zhuwei.output.log" />
   <param name="Append" value="true" />
   <param name="MaxFileSize" value="10240" /> <!-- 设置文件大小 -->
   <param name="MaxBackupIndex" value="10" />
   <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p (%c:%L)- %m%n" />
   </layout>
</appender>
<logger name="zcw.log"> <!-- 设置域名限制，即 zcw.log 域及以下的日志均输出到下面对应的
```

通道中 -->
<level value="debug" /><!-- 设置级别 -->
<appender-ref ref="org.zblog.zcw" /><!-- 与前面的通道 id 相对应 -->
</logger>
<root> <!-- 设置接收所有输出的通道 -->
<appender-ref ref="org.zblog.all" /><!-- 与前面的通道 id 相对应 -->
</root>
</log4j:configuration>

三、配置文件加载方法：

```
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.apache.log4j.xml.DOMConfigurator;
public class Log4jApp {
   public static void main(String[] args) {
      DOMConfigurator.configure("E:/study/log4j/log4j.xml");//加载.xml 文件
      //PropertyConfigurator.configure("E:/study/log4j/log4j.properties");//加
载.properties 文件
      Logger log=Logger.getLogger("org.zblog.test");
      log.info("测试");
   }
}
```

四、项目使用 **log4j**

在 **web** 应用中，可以将配置文件的加载放在一个单独的 **servlet** 中，并在 **web.xml** 中配置该 **servlet** 在应用启动时候加载。对于在多人项目中，可以给 每一个人设置一个输出通道，这样在每个人在构建 **Logger** 时，用自己的域名称，让调试信息输出到自己的 **log** 文件中。

五、常用输出格式

 # -X 号**:X** 信息输出时左对齐；
 # %p:日志信息级别
 # %d{}:日志信息产生时间
 # %c:日志信息所在地（类名）
 # %m:产生的日志具体信息
 # %n:输出日志信息换行

log4j 与 tomcat 结合的简单配置

log4j 是 **apache** 和 **ibm** 联合弄得一个应用系统日 志管理工具，利用它的 **api** 可以方便的管理和操纵日志。在调试程序的时候，是一个非常不错的调试帮手。有关 **log4j** 的一些介绍，大家可以参考 **apache** 的网站（**http://jakarta.apache.org/log4j/docs/index.html**）

下面在开始介绍以前，首先推荐一点资料，大家看看，有助于了解。
(1)《**Log4j delivers control over logging**》
http://www-106.ibm.com/developerworks/java/library/jw-log4j/
(2)http://jakarta.apache.org/log4j/docs/api/org/apache/log4j/PatternLayout.html 这里面介绍了有关 **layout.ConversionPattern** 中的转意字符的含义。

（一）与 **tomcat** 结合的简单配置
首先到 **http://jakarta.apache.org/log4j/docs/download.html** 下载一个 **log4j**，目前版本是 **1.2.5**。下载的文件中有详细的介绍和实例、**apidoc**，可以参考一下。
将 **log4j-1.2.5.jar** 的放到系统 **classpath** 中。最好在**%tomca_home%/lib/**也方一份更好哦。（在此注意一个问题，据说 **log4j-1.2.5.jar** 这个文件的文件名**,weblogic6.1** 不能识别，需要改一个名字）

1.1 描写 **properties** 文件。
这个描述文件一般放在可以放在两个地方： （**1**）**WEB-INF/classes** 目录下，或者放在在 **/project_root/**，也就是你的 **web_app** 的根目录下。利用这个控制日志纪录的配 置，当然也可以通过 **xml** 文件配置，这里我们暂且说说 **properties** 的配置。
建立文件名 **log4j.properties** 文件。放在**%tomca_home%/web_app/fcxtest/**目录下。
文件内容如下：（**fcxtest** 为自己建立的 **web app** 目录）

```
#-------------------------------
# 设定 logger 的 root level 为 DEBUG，指定的输出目的地（appender）为 A1
log4j.rootLogger=DEBUG, A1

# 设定调试信息的输出位置，此处设定输出为控制台
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# 设定调试信息的输出位置，此处设定输出为 fcxtest.log 文件
# log4j.appender.A1=org.apache.log4j.RollingFileAppender
# log4j.appender.A1.File=fcxtest.log
# log4j.appender.A1.MaxFileSize=1000KB

# 设定制定的 A1 使用的 PatternLayout.
# 有关 ConversionPattern 中的转意字符的含义参考说明
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %-5p [%t] %C{2} (%F:%L) - %m%n
#-------------------------------
```

1.2 建立测试用的 **Servlet** 类
这个测试的 **com.fcxlog.LogShow** 类主要是显示如何使用 **log4j**。

```
package com.fcxlog;
```

```
import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;
import org.apache.log4j.PropertyConfigurator;

public class LogShow extends javax.servlet.http.HttpServlet{

protected String configfile = "log4j.properties";

public void init() throws ServletException{
ServletContext sct = getServletContext();
System.out.println("[Log4j]: The Root Path: " + sct.getRealPath("/"));
//指定自己的 properties 文件
//以前使用的是 BasicConfigurator.configure()，现在使用 PropertyConfigurator 替代
org.apache.log4j.PropertyConfigurator.configure(sct.getRealPath("/") + configfile);
}

public void service(javax.servlet.http.HttpServletRequest
req,javax.servlet.http.HttpServletResponse res){

//初始化 Logger，以前使用 Category，现在改用 Logger 替代
//org.apache.log4j.Category log =
org.apache.log4j.Category.getInstance(LogShow.class);
org.apache.log4j.Logger log = ora.apache.log4j.Logger.getLogger(LogShow.class);
log.info("调试信息");

}
}
```

1.3 测试了
至于如何测试，发布，就不说了。

在此说明：
（**1**）本篇可能错误不少，一方面是参考《**Short introduction to log4j**》着翻译了一点，有诸多言辞不妥之处，还望指正。一方面，凭借自己的理解，所以难免有不全的地方，还望各位补充。
（**2**）因为时间有限，每天只能写一点，本文主要是介绍有关 **Logger** 及 **Logger level** 相关概念的
（**3**）有关 **Log4j** 介绍（一），请参阅：
http://www.javaren.com/bbs/cgi-bin/topic.cgi?forum=1&topic=12766&show=0#lastviewpost

概述：
本文主要是简要的介绍了 **Log4j** 的 **api**，以及其一些特征和设计原理。它本身是一个开源的软件，允许开发者任意的操纵应用系统日志信息。**Log4j** 的使用通过外部配置文件进行配置。

任何大型应用系统都有其自己的系统日志管理或跟踪的 **API**，所以在 **1996** 年的时候，**E.U. SEMPER** 项目组（**http://www.semper.org/**）也开发其自己的日志管理 **API**，后来经过无数次的修改和补充，发展成了现在的 **log4j**，一个给予 **java** 的日志管理工具包。有关最新的版本信息和源码，请访问

http://jakarta.apache.org/log4j /docs/index.html
把纪录语句放在代码之中，是一种低端的调试方法，不过有时却不得不说是很有效，也是必需的。毕竟很多时候， 调试器并不见得很适用。特别是在多线程应用程序（**multithread application**）或分布式应用程序（**distributed application**）

经验告诉我们，在软件开发生命周期中，日志系统是一个非常重要的组件。当然，日志系统也有其自身的问题，过多的日志纪录会降低系统运行的速度。

（一）**Log4j** 的三个重要组件**?? Loggers, Appenders, Layouts**

这三个组件协同的工作，使得开发者能够依据信息类别和级别去纪录信息，并能够运行期间，控制信息记录的方式已经日志存放地点。

（二）记录器层次（**Logger hierarchy**）

几乎任何纪录日志的 **API** 得功能都超越了简单的 **System.out.print** 语句。允许有选择控制 的输出日志信息，也就是说，某的时候，一些日志信息允许输出，而另一些则不允许输出。这就假设日志纪录信息之间是有分别的，根据开发者自己定义的选择标 准，可以对日志信息加以分类。

纪录器的命名是依据实体的。下面有一段有点绕口的解释，我就直抄了，各位可以看看：（**Name Hierarchy**）**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.**
（形象的解释，比如存在记录器 **a.b.c**，记录器 **a.b**，记录器 **a**。那么 **a** 就是 **a.b** 的 **ancestor**，而 **a.b** 就是 **a.b.c** 的 **parent**，而 **a.b.c** 就是 **a.b** 的 **child**）

根纪录器（**root logger**）是记录器层次的顶端。它有两个独特点：（**1**）总是存在的（**2**）能够被重新找回。可以通过访问类的静态方法 **Logger.getRootLogger** 重新得到。其他的纪录器通过访问静态方法 **Logger.getLogger** 被实例话或被得到，这个方法将希望获得的记录器的名称作为参数。一些 **Logger** 类的方法描述如下：

```
public class Logger {
// Creation & retrieval methods:
public static Logger getRootLogger();
public static Logger getLogger(String name);
// printing methods:
public void debug(Object message);
public void info(Object message);
public void warn(Object message);
public void error(Object message);
public void fatal(Object message);

// generic printing method:
public void log(Level l, Object message);
}
```

记录器被赋予级别，这里有一套预定的级别标准：**DEBUG, INFO, WARN, ERROR and FATAL** ，这些是在 **org.apache.log4j.Level** 定义的。你可以通过继承 **Level** 类定义自己的级别标准，虽然并不鼓励这么做。
如 果给定的记录器没有被赋予级别，则其会从离其最近的拥有级别的 **ancestor** 处继承得到。如果 **ancestor** 也没有被赋予级别，那么就从根记录器继 承。所以通常情况下，为了让所有的记录器最终都

能够被赋予级别，跟记录器都会被预先设定级别的。比如我们在操作 **properties** 文件中，会写这么一 句：**log4j.rootLogger=DEBUG, A1** 。实际上就这就指定了 **root Logger** 和 **root Logger level**。

Appenders

Log4j 允许记录信息被打印到多个输出目的地，一个输出目的地叫做 **Appender**。目前的 **Log4j** 存在的输出目的地包括：控制台（**Console**）,文件（**File**），**GUI Componennt**，**Remote Socket Server**，**JMS**，**NT Event Logger**，**and Remote Unix Syslog daemons**。

多个 **Appender** 可以绑定到一个记录器上（**Logger**）。

通 过方法 **addAppender**（**Logger.addAppender**） 可以将一个 **Appender** 附加到一个记录器上。

每一个有效的发送到特定的记录器的记录请求都被转送到那个与当前记录器所绑定的 **Appender** 上。（**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**），换句话说，**Appender** 的继承层次是附加在记录器继承层次上的。举个例子：如果一个 **Console Appender** 被绑定到根记录器（**root Logger**），那么所有的记录请求都可以至少被打印到 **Console**。另外，把一个 **file Appender** 绑定到记录器 **C**，那么针对记录器 **C**（或 **C** 的子孙）的记录请求都可以至少发送到 **Console Appender** 和 **file Appender**。当然这种默认的行为方式可以跟改，通过设定记录器的 **additivity flag**（**Logger.setAdditivity**）为 **false**，从而可以使得 **Appender** 的不再具有可加性（**Additivity**）。

下面简要介绍一下 **Appender Additivity**。

Appender Additivity：记录器 **C** 所记录的日志信息将被发送到与记录器 **C** 以及其祖先（**ancestor**）所绑定的所有 **Appender**。

但是，如果记录器 **C** 的祖先，叫做 **P**，它的 **additivity flag** 被设定为 **false**。那么，记录信息仍然被发送到与记录器 **C** 及其祖先，但只到达 **P** 这一层次，包括 **P** 在内的记录器的所有 **Appender**。但不包括 **P** 祖先的。

通常，记录器的 **additivity flag** 的被设置为 **true**。

Layouts

这一块主要是介绍输出格式的。**PatternLayout**，**Log4j** 标准的分配器，可以让开发者依照 **conversion patterns** 去定义输出格式。**Conversion patterns** 有点像 **c** 语言的打印函数。

参看配置文件的 **java properties**，如下面的两行：

log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%d %-5p [%t] %C{2} (%F:%L) - %m%n

第一行就指定了分配器，第二行则指定了输出的格式。

有关输出格式的定义可以参考**/org/apache/log4j/PatternLayout.html**

更基础的知识，可以到田贯升门诊室，另有一篇偏重基础的介绍**~**

使用 Log4J 进行简单日志操作

1、配置 Log4J 的配置文件用于设置记录器的级别、存放器和布局，可以用 key=*value* 格 式的设置或 xml 格式的设置信息。通过配置，可以创建出 Log4J 的运行环境。 有几种方式可以配置 Log4J： 1）在程序中调用 BasicConfigurator.configure()方法； 2）配置放在文件里，通过命令行参数传递文件名字，通过 PropertyConfigurator.configure(args[x])解析并配置； 3）配置放在文件里，通过环境变量传递文件名等信息，利用 Log4J 默认的初始化过程解析并配置； 4）配置放在文件里，通过应用服务器配置传递文件名等信息，利用一个特殊的 servlet 来完成配置。 配置文件示例：
log4j.rootLogger=INFO,A1,R log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss} [%c]-[%p] %m%n
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=d:/tomcat4.1/webapps/ck1907/logs/log4j.log
log4j.appender.R.MaxFileSize=100KB log4j.appender.R.MaxBackupIndex=1
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=[%-d{yyyy-MM-dd HH:mm:ss}] %p %t %c - %m%n 该配置文件设置了两个存放器：A1 和 R，日志级别是 INFO（有 INFO、DEBUG、WARN、ERROR 等取值）。A1 用于工作台输出，即可以直接在 App server 的工作窗口或者 java 应用程序的调试窗口输出；R 用于可滚动的文件输出，设置了最大文件大小为 100K，当达到这个最大值的时候会自动删除 之前老的日志信息以存放新的日志。layout.ConversionPattern 定义了输出格式?~~ 2、应用举例 1）Java 类中使用 Log4J：
import org.apache.log4j.Logger; import org.apache.log4j.PropertyConfigurator; /** * Log4JTest.java * author:Ryan */ public class Log4JTest{ public static void main(String[] args){ String configFile = "d:/log4j.properties"; /*通过 PropertyConfigurator 解析配置文件并初始化 Log4J 配置*/ PropertyConfigurator.configure(configFile); Logger logger = Logger.getLogger(Log4JTest.class); logger.info("main() - init success..."); } } 工作台输出格式： 2004-12-16 15:10:59 [webSite.servlet.RyanConfig]-[INFO] main() - init success... log 文件输出格式： [2004-12-16 15:10:59] INFO main webSite.servlet.RyanConfig - main() - init success... 2）Web application 中使用 Log4J：在 web 应用程序中使用 Log4J，可以在一个随 app server 启动的 servlet 中完成初始化工作 web.xml 配置： …… RyanConfig webSite.servlet.RyanConfig log4j /WEB-INF/log4j.properties 1 …… </P>

log4j 属性含义

1）%r 输出程序开始执行之后的微秒数
2）%t 输出当前线程的名称
3）%-5p 输出消息的层次。
4）%c 输出 category 的名称
5)-%m 及 s 是日志消息本身,%n 是换行符。
当前在模式字符串中你可以嵌入任何想要输出的字符。
模式字符串中的模式如下：
%m:消息本身
%p:消息的层次
%r:从程序开始执行到当前日志产生时的时间间隔（微秒）
%c:输出当前日志动作所在的 category 名称。例如：如果 category 名称是"a.b.c","%c{2}"将会输出"b.c". {2}意谓着输出“以点分隔开的 category 名称的后两个组件”,如果 {n}没有，将会输出整个 category 名称.
%t:输出当前线程的名称
%x:输出和当前线程相关联的 NDC(具体解释见下文),尤其用到像 java servlets 这样的多客户多线程的应用中。
%n:输出平台相关的换行符。
%%:输出一个"%"字符
%d:输出日志产生时候的日期，当然可以对日期的格式进行定制。例如：%d{HH:mm:ss,SSSS}或者是%d{dd MMM yyyy HH:mm:ss,SSSS},如果没有指定后面的格式，将会输出 ISO8601 的格式。
%l:输出位置信息，相当于%C.%M(%F:%L)的组合。
%C:输出日志消息产生时所在的类名，如果类名是“test.page.Class1”%C{1}表示输出类名"Class1",%C{2}输出"page.Class1",而%C 则输出"test.page.Class1"。
%M:输出日志消息产生时的方法名称
%F:输出日志消息产生时所在的文件名称
%L:输出代码中的行号
可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：
1)%20c：指定输出 category 的名称，最小的宽度是 20，如果 category 的名称小于 20 的话，默认的情况下右对齐。
2) %-20c:指定输出 category 的名称，最小的宽度是 20，如果 category 的名称小于 20 的话，"-"号指定左对齐。
3) %.30c:指定输出 category 的名称，最大的宽度是 30，如果 category 的名称大于 30 的话，就会将左边多出的字符截掉，但小于 30 的话也不会有空格。
4)%20.30c:如果 category 的名称小于 20 就补空格，并且右对齐，如果其名称长于 30 字符，就从左边交远销出的字符截掉。
4)%20.30c: