# **java自定义日志输出文件(log4j日志文件输出多个自定义日志文件)**

<http://www.jb51.net/article/45525.htm>

**先看一个常见的log4j.properties文件，它是在控制台和test.log文件中记录日志：**

复制代码代码如下:

log4j.rootLogger=DEBUG, stdout, logfile

log4j.appender.stdout=org.apache.log4j.ConsoleAppender   
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout   
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - %m%n

log4j.appender.logfile=org.apache.log4j.RollingFileAppender  
log4j.appender.logfile.File=log/test.log  
log4j.appender.logfile.MaxFileSize=128MB  
log4j.appender.logfile.MaxBackupIndex=3  
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout  
log4j.appender.logfile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p [%t] %c.%M(%L) - %m%n

**如果在同一类中需要输出多个日志文件呢？其实道理是一样的，先在Test.java中定义:**

Test.java

private static Log logger1 = LogFactory.getLog("mylogger1");

private static Log logger2 = LogFactory.getLog("mylogger2");

log4j.properties

log4j.logger.mylogger1=DEBUG,test1  
log4j.appender.test1=org.apache.log4j.FileAppender  
log4j.appender.test1.File=log/test1.log  
log4j.appender.test1.layout=org.apache.log4j.PatternLayout  
log4j.appender.test1.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p [%t] %c.%M(%L) - %m%n

log4j.logger.mylogger2=DEBUG,test2  
log4j.appender.test2=org.apache.log4j.FileAppender  
log4j.appender.test2.File=log/test2.log  
log4j.appender.test2.layout=org.apache.log4j.PatternLayout  
log4j.appender.test2.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p [%t] %c.%M(%L) - %m%n

不同日志要使用不同的logger(如输出到test1.log的要用logger1.info("abc"))。

还有一个问题，就是这些自定义的日志默认是同时输出到log4j.rootLogger所配置的日志中的，如何能只让它们输出到自己指定的日志中呢？别急，这里有个开关：

log4j.additivity. mylogger1 = false

它用来设置是否同时输出到log4j.rootLogger所配置的日志中，设为false就不会输出到其它地方了。

但是这种方式有个小缺陷，那就是打印的日志中类名只能是mylogger或者mylogger2。

**结合‘common-logging.jar log4j.jar**

log4j.properties配置如下：

### set log levels ###  
log4j.rootLogger = debug , stdout , D , E  
  
### 输出到控制台 ###  
log4j.appender.stdout = org.apache.log4j.ConsoleAppender  
log4j.appender.stdout.Target = System.out  
## 输出INFO级别以上的日志  
log4j.appender.stdout.Threshold = INFO  
log4j.appender.stdout.layout = org.apache.log4j.PatternLayout  
log4j.appender.stdout.layout.ConversionPattern = %d{ABSOLUTE} %5p %c{1}:%L - %m%n  
  
### 输出到日志文件 ###  
log4j.appender.D = org.apache.log4j.DailyRollingFileAppender  
log4j.appender.D.File = D:/logs/log.log  
log4j.appender.D.Append = true  
## 输出DEBUG级别以上的日志  
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  
  
### 保存异常信息到单独文件 ###  
log4j.appender.E = org.apache.log4j.DailyRollingFileAppender  
## 异常日志文件名  
log4j.appender.E.File = D:/logs/error.log   
log4j.appender.E.Append = true  
 ## 只输出ERROR级别以上的日志!!!  
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

### **4.4.单独使用log4j实例：Log4jTest.java**

package log.sample;  
  
import org.apache.log4j.Logger;  
  
public class Log4jTest {  
 private static Logger log = Logger.getLogger(Log4jTest.class);  
 public void log() {  
 log.debug("Debug info.");  
 log.info("Info info");  
 log.warn("Warn info");  
 log.error("Error info");  
 log.fatal("Fatal info");  
 }  
  
 public static void main(String[] args) {  
 Log4jTest test = new Log4jTest();  
 test.log();  
 }  
}

### **4.5.common-logging结合log4j实例：JCLTest.java**

package log.sample;  
  
import org.apache.commons.logging.Log;  
import org.apache.commons.logging.LogFactory;  
   
public class JCLTest {  
 private static Log log = LogFactory.getLog(JCLTest.class);  
 public void log(){  
 log.debug("Debug info.");  
 log.info("Info info");  
 log.warn("Warn info");  
 log.error("Error info");  
 log.fatal("Fatal info");  
 }  
 public static void main(String[] args) {  
 JCLTest test = new JCLTest();  
 test.log();  
 }  
}

http://blog.csdn.net/memray/article/details/18193449

**由于spring也会加载log4j.properties，如果加载不到就往控制台打log信息。Spring提供了 一个Log4jConfigListener，本身就能通过web.xml中配置来指定位置加载log4j配置文件和log输出路径，注意该 listener需要放在spring的Listener之前。**

在web.xml 添加

1. **<context-param>**
2. **<param-name>**log4jConfigLocation**</param-name>**
3. **<param-value>**WEB-INF/log4j.properties**</param-value>**
4. **</context-param>**
5. **<context-param>**
6. **<param-name>**log4jRefreshInterval**</param-name>**
7. **<param-value>**60000**</param-value>**
8. **</context-param>**
9. **<listener>**
10. **<listener-class>**org.springframework.web.util.Log4jConfigListener**</listener-class>**
11. **</listener>**

**[log4j中配置日志文件相对路径方法](http://blog.csdn.net/longaohun/article/details/6233099)**

**<http://blog.csdn.net/longaohun/article/details/6233099>**

总结了上面的三个方法，可取的是方法三，但是还是繁琐，要写个servlet，写一大堆代码

下面这个方法比较实用我觉得，大家可以试下：

log4j.appender.AFile=org.apache.log4j.DailyRollingFileAppender  
log4j.appender.AFile.DatePattern='.'yyyy-MM-dd'.html'  
log4j.appender.AFile.File=..//webapps//项目//logs//d1cm\_log.html  
log4j.appender.AFile.layout=org.apache.log4j.HTMLLayout  
log4j.appender.AFile.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %p %c %m %n

默认生成的日志文件是在服务器的bin目录下（我使用的是Tomcat），这样通过相对路径  ..//webapps//项目名称//日志目录//日志文件(或../webapps/项目名称/日志目录/日志文件)，就能在你的项目中自动建立日志目录和日志文件。

**方法一：**用相对路径代替绝对路径

    主要是扩展了log4j的RollingFileAppender类，其他的FileAppender同样道理。扩展的方法，就是用一个子类去覆盖setFile方法，这个方法在log4j读取配置文件生成appender的时候调用，传入的就是配置文件中的路径，这样我就可以按照自己的想法在路径前面加上根路径了。这种方法可以在log4j.properties中用相对路径自由配置log4j.appender.A1.File属性来决定生成的日志相对web应用根目录的位置。

举例：ZxmDailyRollingFileAppender.java

log4j.properties文件

log4j.appender.AFile=com.common.config.ZxmDailyRollingFileAppender  
#log4j.appender.AFile=org.apache.log4j.DailyRollingFileAppender

public class ZxmDailyRollingFileAppender extends DailyRollingFileAppender {  
 public void setFile(String filetype, boolean arg2, boolean arg3, int arg4) throws IOException {  
  String path = getServletConfig().getServletContext().getRealPath("/logs/");  
  String fuhao = File.separator;  
  String filepath = path + fuhao + "log.html";  
   super.setFile(filepath, arg2, arg3, arg4);  
 }  
}

**方法二：**使用服务器环境变量

    是利用服务器vm中已经存在的环境变量如${catalina.home}或者${catalina.base}来设置相对于${catalina.home}或者${catalina.base}的日志路径，日志只能放到服务器子目录里，而且如果是用的其它服务器，则要改对应的环境变量。此方法平台移植不方便。

举例：log4j.appender.file.file=${catalina.base}/logs/psp\_error.log

catalina.home指向公用信息的位置，就是bin和lib的父目录。  
catalina.base指向每个Tomcat目录私有信息的位置，就是conf、logs、temp、webapps和work的父目录。  
仅运行一个Tomcat实例时，这两个属性指向的位置是相同的。

**方法三：**通过servlet初始化init()方法中加载file属性实现相对路径

    是扩展ActionServlet类，覆盖其init()方法，新方法中载入log4j.properties位置的参数，可以自由配置log4j的配置文件的名字和存放位置。也可自由配置log4j日志文件的相对于当前应用的路径。

举例：

web.xml

 <!-- 系统初始化配置信息  servlet -->  
 <servlet>  
  <servlet-name>configServlet</servlet-name>  
  <servlet-class>com.common.config.ConfigServlet</servlet-class>  
  <init-param>  
   <param-name>Log4jFile</param-name>  
   <param-value>/WEB-INF/config/log4j.properties</param-value>  
  </init-param>  
  <init-param>  
   <param-name>Log4jFileSavePath</param-name>  
   <param-value>/logs</param-value>  
  </init-param>  
  <load-on-startup>1</load-on-startup>  
 </servlet>

ConfigServlet.java文件

public class ConfigServlet extends HttpServlet {  
 private static final long serialVersionUID = 1L;  
 public void init(ServletConfig config) throws ServletException {  
  super.init(config);  
      //初始化log4j日志组件  
  initLogConfig(config);  
    
 }  
   
   
 private void initLogConfig(ServletConfig config) {   
  String prifix = getServletContext().getRealPath("/");  
    //System.out.println(prifix);  
      
     //获取log4j配置文件地址  
  String Log4jFile = config.getInitParameter("Log4jFile");  
  //System.out.println(Log4jFile);  
  String filePath = prifix + Log4jFile;  
  //System.out.println(filePath);  
  PropertyConfigurator.configure(filePath);  
  Properties props = new Properties();  
  try {  
   String Log4jFileSavePath = config.getInitParameter("Log4jFileSavePath");  
   //System.out.println(Log4jFileSavePath);  
   FileInputStream log4jStream = new FileInputStream(filePath);  
   props.load(log4jStream);  
   log4jStream.close();  
   //设置日志保存地址  
   String logFile = prifix + Log4jFileSavePath + File.separator + "d1cm\_log.html" ;  
   //System.out.println(logFile);  
   props.setProperty("log4j.appender.AFile.File", logFile);  
   PropertyConfigurator.configure(props); //装入log4j配置信息  
  } catch (IOException e) {  
   e.printStackTrace();  
  }  
  System.err.println("初始化log4j日志组件");  
 }  
}

**几种log工具**

**<http://shift8.iteye.com/blog/1316802>**

1、****Apache通用日志接口(commons-logging.jar)介绍****

Apache Commons包中的一个，包含了日志功能，必须使用的jar包。这个包本身包含了一个Simple Logger，但是功能很弱。在运行的时候它会先在CLASSPATH找log4j，如果有，就使用log4j，如果没有，就找JDK1.4带的 java.util.logging，如果也找不到就用Simple Logger。commons-logging.jar的出现是一个历史的的遗留的遗憾，当初Apache极力游说Sun把log4j加入JDK1.4，然而JDK1.4项目小组已经接近发布JDK1.4产品的时间了，因此拒绝了Apache的要求，使用自己的java.util.logging，这个包的功能比log4j差的很远，性能也一般。后来Apache就开发出来了commons-logging.jar用来兼容两个logger。因此用 commons-logging.jar写的log程序，底层的Logger是可以切换的，你可以选择log4j，java.util.logging或者它自带的Simple Logger。不过我仍然强烈建议使用log4j，因为log4j性能很高，log输出信息时间几乎等于System.out，而处理一条log平均只需要5us。

Apache通用日志包提供一组通用的日志接口,用户可以自由选择实现日志接口的第三方软件  
通用日志目前支持以下日志实现  
Log4j日志器  
JDK1.4Logging  
SimpleLog日志器  
NoOpLog日志器

****1.1、Log 接口****Common-logging的应用程序编程接口主要在org.apache.commons.logging.log接口中定义，这个接口主要定义了两类操作： 一类是级别判断，用于减少不必要的日志操作的参数计算从而提高性能。另一类是日志登记，按照级别登记日志信息。  
通用日志包把日志消息分为6种级别Fatal,Error,Warn,Info,Debug和Trace

org.apache.commons.logging.Log接口代表日志器,它提供了一组输出日志的方法，日志登记操作分又为两小类：一个参数的日志信息登记操作和两个参数的日志信息登记操作。前者对三类用户都适用，后者用于打印日志登记处的出错堆栈信息，所以更适用于开发人员调式与维护使用  
       fatal(Object message);  
       error(Object message);  
       warn (Object message);  
       info (Object message);  
       debug(Object message);  
       trace(Object message);

       debug(Object message, Throwable t);   
       trace(Object message, Throwable t);  
       ......  
      这里需要注意的是,只有当输出日志的级别大于等于日志器配置的日志级别时,这个方法才会真正被执行.例如日志器设置日志级别为Warn,那么程序中,它的  
      fatal(),error(),warn()方法会被执行,而info(),debug(),trace()  
      不会被执行.  
      Log接口还提供了一组判断是否允许输出特定级别日志消息的方法

       isFatalEnable();  
       isErrorEnable();  
       isWarnEnable();  
       isInfoEnable();  
       isDebugEnable();  
       isTraceEnable();  
****1.2、LogFactory接口****  
      org.apache.commons.logging.LogFactory接口提供了获得日志器实例的两个静态方法  
       public static Log getLog(String name)throws LogConfigurationException  
       public static Log getLog(Class class)throws LogConfigurationException  
       public static Log getLog(Class class)  
       {  
        getLog(class.getName());  
       }

****2、Log4J介绍****

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](http://baike.baidu.com/view/269351.htm)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。这些可以通过一个[配置文件](http://baike.baidu.com/view/2117618.htm)来灵活地进行配置，而不需要修改应用的代码。

2.1、Log4J配置说明：

log4j配置文件有三个主要的组件:Logger,Appender和Layout,分别为日志类型,日志输出目的地,日志输出格式.   
跟proxool类似,log4j支持两种类型的配置文件,xml和properties 。log4j.properties配置文件如下:(需要把log4j的配置文件放在classpath下)

****2.1.1****log4j.rootLogger =INFO, stdout , R

此句为将等级为INFO的日志信息输出到stdout和R这两个目的地，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或者自定义的级别。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](http://baike.baidu.com/view/330120.htm)中相应级别的日志信息的开关。比如在这里定义了INFO级别，只有等于及高于这个级别的才进行处理，则应用程序中所有DEBUG级别的日志信息将不被打印出来。ALL:打印所有的日志，OFF：关闭所有的日志输出。 appenderName就是指定日志信息输出到哪个地方。可同时指定多个输出目的地。

****2.1.2****log4j.appender.stdout=org.apache.log4j.ConsoleAppender

此句为定义名为stdout的输出端是哪种类型

Appender 负责控制日志记录操作的输出。

　　其语法为：

　　log4j.appender.appenderName = fully.qualified.name.of.appender.class

　　log4j.appender.appenderName.option1 = value1

　　…

　　log4j.appender.appenderName.optionN = valueN

　　这里的appenderName为在①里定义的，可任意起名。

　　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），可通过log4j.appender.R.MaxFileSize=100KB设置文件大小，还可通过 log4j.appender.R.MaxBackupIndex=1设置为保存一个[备份文件](http://baike.baidu.com/view/399279.htm)。

　　org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）

　　例如：log4j.appender.stdout=org.apache.log4j.ConsoleAppender

　　定义一个名为stdout的输出目的地，ConsoleAppender为控制台。

****2.1.3配置日志信息的格式（布局）Layout****

　　Layout 负责格式化Appender的输出。

　　其语法为：

　　log4j.appender.appenderName.layout = fully.qualified.name.of.layout.class

　　log4j.appender.appenderName.layout.option1 = value1

　　…

　　log4j.appender.appenderName.layout.optionN = 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）

##### 2.1.4格式化日志信息

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{yyyy 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

****2.2、关于Log4j比较全面的配置****

****2.2.1 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

****2.2.2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

****2.2.3 应用于回滚文件****

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

****2.2.4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

****2.2.5应用于邮件****

· 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

****2.2.6应用于数据库****

· 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.db.sql=INSERT INTO SS\_LOG4J\_LOG (PRIORITY,LOGDATE,CLASS,METHOD,MSG) VALUES('%p','%d{yyyy-MM-dd HH:mm:ss}','%C','%M','%m')

· log4j.appender.db.layout=org.apache.log4j.PatternLayout log4j.appender.DATABASE.layout=org.apache.log4j.PatternLayout

· log4j.appender.DATABASE.layout.ConversionPattern=[framework] %d - %c -%-4r [%t] %-5p %c %x - %m%n <pre name="code" class="java">

****3、Log4J的使用****

3.1、在application中的使用：

将log4j.jar和common-logging.jar加入到build path中即可。

3.2 在web应用中使用log4j

  3.2.1 创建一个集成httpservlet的servlet文件，在init中对log4j进行初始化

public class log4jInit extends HttpServlet {

    @Override  
    public void init() throws ServletException {  
        Log log = LogFactory.getLog(this.getClass().getName());  
        //Logger logger = Logger.getLogger(this.getClass());          
        String prefix = this.getServletContext().getRealPath("/");  
        System.out.println(prefix);  
        String file = this.getServletConfig().getInitParameter("log4j-init-file");  
        if(file!=null)  
            PropertyConfigurator.configure(prefix+file);  
    }

    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
            throws ServletException, IOException {  
    }

    @Override  
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
            throws ServletException, IOException {  
    }

}

   3.2.2 在web.xml中定义这个servlet，并在应用启动时启动

<servlet>  
    <servlet-name>log4jInit</servlet-name>  
    <servlet-class>com.highsoft.log4j.log4jInit</servlet-class>  
    <init-param>  
        <param-name>log4j-init-file</param-name>  
        <param-value>/WEB-INF/classes/log4j1.properties</param-value>  
    </init-param>  
    <load-on-startup>1</load-on-startup>  
</servlet>

3.3 在Spring中集成log4j

3.3.1 在web.xml中集成

<!--如果不设，默认为web.root,但最好设置，以免项目间冲突—>

<!--如果应用服务器下有不止一个的应用在使用spring的Log4jConfigListener，需要修改web环境中webAppRootKey值(这样log4j的配置文件里就可以用${myAppfuse.root}来表示刚刚设进去的系统变量，例如：log4j.appender.logfile.File=${myAppfuse.root}/logs/mylog.log)。否则两个默认值web.root在环境变量中就会有冲突导致第二个应用启动失败。—>

<context-param>       
    <param-name>webAppRootKey</param-name>        
    <param-value>myappfuse.root</param-value>   
</context-param>

<!--在这里定位配置文件，需要的是从root开始的绝对路径—>  
<context-param>     
    <param-name>log4jConfigLocation</param-name>     
    <param-value>/WEB-INF/classes/log4j.properties</param-value>  
</context-param>

<!--Spring默认刷新Log4j配置文件的间隔,单位为millisecond-->  
<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>

<!--这里用listener，也可以用下面的servlet—>

<servlet>  
   <servlet-name>log4j</servlet-name>  
   <servlet-class>org.springframework.web.util.Log4jConfigServlet</servlet-class>  
   <load-on-startup>1</load-on-startup>  
</servlet>

##### 3.3.2 在代码中使用Log4j

我们在需要输出日志信息的类中做如下的三个工作：

　　1、导入所有需的commongs-logging类：

　　import org.apache.commons.logging.Log;

　　import org.apache.commons.logging.LogFactory;

　　2、在自己的类中定义一个org.apache.commons.logging.Log类的私有静态类成员：

　　private final Log log = LogFactory.getLog(getClass());

　　LogFactory.getLog()方法的参数使用的是当前类的class。

　　3、使用org.apache.commons.logging.Log类的成员方法输出日志信息：

　　if (log.isDebugEnabled()){

　　log.debug("111");

　　}

　　if (log.isInfoEnabled()){

　　log.info("222");

　　}

　　if (log.isWarnEnabled()){

　　log.warn("333");

　　}

　　if (log.isErrorEnabled()){

　　log.error("444");

　　}

　　if (log.isFatalEnabled()){

　　log.fatal("555")

　　}

4、SLF4j介绍

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。

4.1、使用方式：  
       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）  
4.2、使用场景：  
        我们开发的是类库或者嵌入式组件，可以考虑使用slf4j，因为我们并不能决定用户选择哪种日志系统（不同软件开发公司会钟情于不同的日志系统）；但是如果我们开发独立应用，面向的是最终客户，则无需考虑slf4j，因为最终客户只关心功能实现，不会在意开发公司具体使用什么日志系统的。  
4.3、应用举例：    
import org.slf4j.Logger;     
import org.slf4j.LoggerFactory;     
/\*\*    
\* @author chb    
\*    
\*/    
public class TestSlf4j {     
        Logger log = LoggerFactory.getLogger(TestSlf4j.class);     
        public void testLog(){     
                log.info("this is a test log");     
        }     
        /\*\*    
         \* @param args    
         \*/    
        public static void main(String[] args) {     
                TestSlf4j slf = new TestSlf4j();     
                slf.testLog();     
        }     
}

1>JDK自带的log输出  
       首先，我们在classpath中加入slf4j-api-1.5.10.jar和slf4j-jdk14-1.5.10.jar两个包，然后运行main函数，输出信息如下：  
view plaincopy to clipboardprint?  
2010-1-5 21:44:47 chb.test.slf4j.TestSlf4j testLog     
信息: this is a test log    
2010-1-5 21:44:47 chb.test.slf4j.TestSlf4j testLog  
信息: this is a test log   
2>slg4j提供的simple log  
      然后，我们用slf4j-simple-1.5.10.jar替换slf4j-jdk14-1.5.10.jar，选择使用slf4j提供的simple log，输出信息如下：  
view plaincopy to clipboardprint?  
0 [main] INFO chb.test.slf4j.TestSlf4j - this is a test log    
0 [main] INFO chb.test.slf4j.TestSlf4j - this is a test log   
3>log4j日志输出  
    再然后，我们再用slf4j-log4j12-1.5.10.jar替换slf4j-simple-1.5.10.jar（记得classpath也需要增加log4j依赖jar包），同时增加一个log4j.properties文件，我们再稍微修改一下main函数，加载一下log4j.properties，如；  
view plaincopy to clipboardprint?  
public static void main(String[] args) {     
        PropertyConfigurator.configure("D:\\log4j.properties");    
        TestSlf4j slf = new TestSlf4j();     
        slf.testLog();     
}

4.4、原理介绍--静态绑定  
        大家看到要使用哪种日志系统，只需要将对应的日志系统所需要的jar包文件（包括slf4j提供的jar包和日志系统自身依赖的jar包，例如：slf4j-log4j12-1.5.10.jar和log4j.1.2.jar）放入classpath即可，slf4j可以自动探测具体使用哪种日志系统，这种技术被称为静态绑定。  
       在实际使用中，我们通过LoggerFactory.getLogger()获得logger，查看LoggerFactory的源代码会发现如下两点，  
LoggerFactory通过StaticLoggerBinder.getSingleton().getLoggerFactory()获得LogFactory，然后再通过该LogFactory来获取logger的   
但是StaticLoggerBinder类并不在slf4j-api-1.5.10.jar中，分析与具体日志系统相关的jar包，会发现每个jar包都有一个StaticLoggerBinder类的实现(如slf4j-log4j12-1.5.10.jar、slf4j-simple- 1.5.10.jar、slf4j-jdk14-1.5.10.jar均有StaticLoggerBinder类实现)，这就很明白了，slf4j在启动时会动态到classpath中查找StaticLoggerBinder类，找到之后就可以生成对应日志系统的日志文件了。   
    这里就有一个问题了，slf4j是如何将自己的通用日志格式转成不同的日志系统的格式的呢？  
    我们再分析每个日志系统相关的源代码，会发现不同日志系统包都会有一个Adapter，用来在slf4j和不同日志系统之间做转换。

4.5、与common-logging的区别

Apache Common-Logging是广泛使用的Java日志门面库。我以前一直都使用它和log4j编写日志。  
Apache Common-Logging通过****动态查找的机制****，在程序运行时自动找出真正使用的日志库。  
Apache Common-Logging一直都运作得很好。直到最近，我写OSGI插件时，它不能工作了。  
原因是****Apache Common-Logging使用了ClassLoader寻找和载入底层的日志库。而OSGI中，不同的插件使用自己的ClassLoader。****一个线程的ClassLoader在执行不同的插件时，其执行能力是不同的。  
OSGI的这种机制保证了插件互相独立，然而确使Apache Common-Logging无法工作！  
解决之道是使用新的日志门面库Slf4j。  
Slf4j库类似于Apache Common-Logging。但是，他****在编译时静态绑定真正的Log库****。使用Slf4j时，如果你需要使用某一种日志实现，那么你必须选择正确的Slf4j的jar包的集合。

-----------

# Properties for configuring Log4j

# Set logger level

log4j.rootLogger=CONSOLE

log4j.logger.org=CONSOLE

log4j.additivity.com.gm=false

log4j.additivity.com.eds=false

log4j.additivity.FrameworkLogging=false

log4j.additivity.com.gm.workbench.order.service.common.helper.mq=false

log4j.logger.com.gm=INFO,APP\_LOG

#log4j.logger.com.gm.workbench.order.service.offline=CONSOLE

#log4j.logger.com.gm.workbench.order.service.online.elc=CONSOLE

log4j.logger.com.eds=CONSOLE

#log4j.logger.FrameworkLogging=CONSOLE

#log4j.logger.com.gm.workbench.order.service.common.helper.mq=CONSOLE

#log4j.logger.com.gm.workbench.order.service.online.elc=CONSOLE

#log4j.logger.com.gm.workbench.order.service.online.manageinventory=CONSOLE

#log4j.logger.com.gm.workbench.order.service.common.dataaccess=CONSOLE

#log4j.logger.com.gm.workbench.order.service.online.bulkorder=CONSOLE

# CONSOLE is set to be a ConsoleAppender

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender

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

log4j.appender.CONSOLE.layout.ConversionPattern=%d{MM-dd@HH:mm:ss} %-5p %c{1}:(%13F:%L) %3x - %m%n

# APP\_LOG is set to be a RollingFileAppender

log4j.appender.APP\_LOG=org.apache.log4j.RollingFileAppender

log4j.appender.APP\_LOG.MaxFileSize=5000KB

log4j.appender.APP\_LOG.MaxBackupIndex=10

log4j.appender.APP\_LOG.File =c:/log4j/app-logger.log

log4j.appender.APP\_LOG.Append = true

log4j.appender.APP\_LOG.layout=org.apache.log4j.PatternLayout

log4j.appender.APP\_LOG.layout.ConversionPattern=%d{MM-dd@HH:mm:ss} %-5p %c{1}:(%13F:%L) %3x - %m%n

# AUD\_LOG is set to be a RollingFileAppender

log4j.appender.AUD\_LOG=org.apache.log4j.RollingFileAppender

log4j.appender.AUD\_LOG.MaxFileSize=50000KB

log4j.appender.AUD\_LOG.MaxBackupIndex=5

log4j.appender.AUD\_LOG.File =c:/log4j/audit-logger\_001.log

log4j.appender.AUD\_LOG.Append = true

log4j.appender.AUD\_LOG.layout=org.apache.log4j.PatternLayout

log4j.appender.AUD\_LOG.layout.ConversionPattern=%d{MM-dd@HH:mm:ss} %-5p %c{1}:(%13F:%L) %3x - %m%n