**一、logback的介绍**

Logback是由log4j创始人设计的又一个开源日志组件。logback当前分成三个模块：logback-core,logback-classic和logback-access。logback-core是其它两个模块的基础模块。logback-classic是log4j的一个改良版本。此外logback-classic完整实现SLF4J API使你可以很方便地更换成其它日志系统如log4j或JDK14 Logging。logback-access访问模块与Servlet容器集成提供通过Http来访问日志的功能。Logback是要与SLF4J结合起来用

两个组件的官方网站如下：

logback的官方网站：<http://logback.qos.ch>

SLF4J的官方网站：<http://www.slf4j.org>

**二、logback取代log4j的理由：**

Logback和log4j是非常相似的，如果你对log4j很熟悉，那对logback很快就会得心应手。下面列了logback相对于log4j的一些优点：

（1）更快的实现，Logback的内核重写了，在一些关键执行路径上性能提升10倍以上。

而且logback不仅性能提升了，初始化内存加载也更小了。

（2）非常充分的测试Logback经过了几年，数不清小时的测试。Logback的测试完全不同级别的。在作者的观点，这是简单重要的原因选择logback而不是log4j。

（3）Logback-classic非常自然实现了SLF4jLogback-classic实现了SLF4j。在使用SLF4j中，你都感觉不到logback-classic。而且因为logback-classic非常自然地实现了SLF4J，所以切换到log4j或者其他，非常容易，只需要提供成另一个jar包就OK，根本不需要去动那些通过SLF4JAPI实现的代码。

（4）非常充分的文档官方网站有两百多页的文档。

（5）自动重新加载配置文件当配置文件修改了，Logback-classic能自动重新加载配置文件。扫描过程快且安全，它并不需要另外创建一个扫描线程。这个技术充分保证了应用程序能跑得很欢在JEE环境里面。

（6）LilithLilith是log事件的观察者，和log4j的chainsaw类似。而lilith还能处理大数量的log数据。

（7）谨慎的模式和非常友好的恢复在谨慎模式下，多个FileAppender实例跑在多个JVM下，能够安全地写道同一个日志文件。RollingFileAppender会有些限制。Logback的FileAppender和它的子类包括RollingFileAppender能够非常友好地从I/O异常中恢复。

（8）配置文件可以处理不同的情况开发人员经常需要判断不同的Logback配置文件在不同的环境下（开发，测试，生产）。而这些配置文件仅仅只有一些很小的不同，可以通过,和来实现，这样一个配置文件就可以适应多个环境。

（9）Filters（过滤器）有些时候，需要诊断一个问题，需要打出日志。在log4j，只有降低日志级别，不过这样会打出大量的日志，会影响应用性能。在Logback，你可以继续保持那个日志级别而除掉某种特殊情况，如alice这个用户登录，她的日志将打在DEBUG级别而其他用户可以继续打在WARN级别。要实现这个功能只需加4行XML配置。可以参考MDCFIlter。

（10）SiftingAppender（一个非常多功能的Appender）它可以用来分割日志文件根据任何一个给定的运行参数。如，SiftingAppender能够区别日志事件跟进用户的Session，然后每个用户会有一个日志文件。

（11）自动压缩已经打出来的logRollingFileAppender在产生新文件的时候，会自动压缩已经打出来的日志文件。压缩是个异步过程，所以甚至对于大的日志文件，在压缩过程中应用不会受任何影响。

（12）堆栈树带有包版本Logback在打出堆栈树日志时，会带上包的数据。

（13）自动去除旧的日志文件通过设置TimeBasedRollingPolicy或者SizeAndTimeBasedFNATP的maxHistory属性，你可以控制已经产生日志文件的最大数量。如果设置maxHistory12，那那些log文件超过12个月的都会被自动移除。

总之，logback比log4j太优秀了，让我们的应用全部建立logback上吧！

**三、Logback的配置介绍**

（1）Logger、appender及layout

Logger作为日志的记录器，把它关联到应用的对应的context上后，主要用于存放日志对象，也可以定义日志类型、级别。

Appender主要用于指定日志输出的目的地，目的地可以是控制台、文件、远程套接字服务器、MySQL、PostreSQL、Oracle和其他数据库、JMS和远程UNIX Syslog守护进程等。

Layout负责把事件转换成字符串，格式化的日志信息的输出。

（2）logger context

各个logger都被关联到一个LoggerContext，LoggerContext负责制造logger，也负责以树结构排列各logger。其他所有logger也通过org.slf4j.LoggerFactory类的静态方法getLogger取得。getLogger方法以logger名称为参数。用同一名字调用LoggerFactory.getLogger方法所得到的永远都是同一个logger对象的引用。

（3）有效级别及级别的继承

Logger可以被分配级别。级别包括：TRACE、DEBUG、INFO、WARN和ERROR，定义于ch.qos.logback.classic.Level类。如果logger没有被分配级别，那么它将从有被分配级别的最近的祖先那里继承级别。root logger默认级别是DEBUG。

（4）打印方法与基本的选择规则

打印方法决定记录请求的级别。例如，如果L是一个logger实例，那么，语句L.info("..")是一条级别为INFO的记录语句。记录请求的级别在高于或等于其logger的有效级别时被称为被启用，否则，称为被禁用。记录请求级别为p，其logger的有效级别为q，只有则当p>=q时，该请求才会被执行。

该规则是logback的核心。级别排序为：TRACE<DEBUG<INFO<WARN<ERROR

**四、Logback的默认配置**

如果配置文件logback-test.xml和logback.xml都不存在，那么logback默认地会调用BasicConfigurator，创建一个最小化配置。最小化配置由一个关联到根logger的ConsoleAppender组成。输出用模式为

%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n的

PatternLayoutEncoder进行格式化。rootlogger默认级别是DEBUG。

1、Logback的配置文件

Logback配置文件的语法非常灵活。正因为灵活，所以无法用DTD或XML schema进行定义。尽管如此，可以这样描述配置文件的基本结构：以<configuration>开头，后面有零个或多个<appender>元素，有零个或多个<logger>元素，有最多一个<root>元素。

2、Logback默认配置的步骤

(1)尝试在classpath下查找文件logback-test.xml；

(2)如果文件不存在，则查找文件logback.xml；

(3)如果两个文件都不存在，logback用BasicConfigurator自动对自己进行配置，这会导致记录输出到控制台。

maven引入

<!--日志文件管理包-->

<!--log start-->

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-core</artifactId>

<version>${logback.version}</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>${logback.version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.7</version>

</dependency>

<dependency>

<groupId>org.logback-extensions</groupId>

<artifactId>logback-ext-spring</artifactId>

<version>0.1.4</version>

</dependency>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

<!--log end-->

**logback.xml配置，模板**



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

<!--属性描述scan：性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true scanPeriod:设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟。

debug:当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。-->

<configuration scanPeriod="2 seconds"debug="true">

<!--定义日志文件的存储地址勿在LogBack的配置中使用相对路径-->

<property name="LOG\_HOME"value="e:/log/SSM/"/>

<!--控制台输出-->

<appender name="STDOUT"class="ch.qos.logback.core.ConsoleAppender">

<!--日志输出编码-->

<Encoding>UTF-8</Encoding>

<layout class="ch.qos.logback.classic.PatternLayout">

<!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS}[%thread]%-5level%logger{50}-%msg%n

</pattern>

</layout>

</appender>

<!--按照每天生成日志文件-->

<appender name="FILE"class="ch.qos.logback.core.rolling.RollingFileAppender">

<Encoding>UTF-8</Encoding>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!--日志文件输出的文件名-->

<FileNamePattern>${LOG\_HOME}/myApp.log.%d{yyyy-MM-dd}.log</FileNamePattern>

<MaxHistory>30</MaxHistory>

</rollingPolicy>

<layout class="ch.qos.logback.classic.PatternLayout">

<!--格式化输出：%d表示日期，%thread表示线程名，%-5level：级别从左显示5个字符宽度%msg：日志消息，%n是换行符-->

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS}[%thread]%-5level%logger{50}-%msg%n

</pattern>

</layout>

<!--日志文件最大的大小-->

<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">

<MaxFileSize>10MB</MaxFileSize>

</triggeringPolicy>

</appender>

<!--show parameters for ibatis sql专为ibatis定制-->

<logger name="com.ibatis"level="DEBUG"/>

<logger name="com.ibatis.common.jdbc.SimpleDataSource"level="DEBUG"/>

<logger name="com.ibatis.common.jdbc.ScriptRunner"level="DEBUG"/>

<logger name="com.ibatis.sqlmap.engine.impl.SqlMapClientDelegate"level="DEBUG"/>

<logger name="java.sql.Connection"level="DEBUG"/>

<logger name="java.sql.Statement"level="DEBUG"/>

<logger name="java.sql.PreparedStatement"level="DEBUG"/>

<!--日志输出级别-->

<root level="INFO">

<appender-ref ref="STDOUT"/>

<appender-ref ref="FILE"/>

</root>

<!--日志异步到数据库-->

<!--<appender name="DB"class="ch.qos.logback.classic.db.DBAppender">-->

<!--日志异步到数据库-->

<!--<connectionSource class="ch.qos.logback.core.db.DriverManagerConnectionSource">-->

<!--连接池-->

<!--<dataSource class="com.mchange.v2.c3p0.ComboPooledDataSource">

<driverClass>com.mysql.jdbc.Driver</driverClass>

<url>jdbc:mysql://127.0.0.1:3306/databaseName</url>

<user>root</user>

<password>root</password>

</dataSource>

</connectionSource>

</appender>-->

</configuration>

===========================华丽分割线==============================

一、log4j的使用

注：使用到的架包log4j-1.2.17.jar、slf4j-log4j12-1.7.7.jar

1、web.xml配置增加内容

<!--加载log4j日志打印-->

<!--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>

1. log4j.properties配置

log4j.rootLogger=INFO,Console,stdout

#控制台输出

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

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

log4j.appender.Console.layout.ConversionPattern=[%p][%d{yyyy-MM-dd HH\:mm\:ss}][%c]-%m%n

#跳转地址日志

log4j.appender.stdout=org.apache.log4j.DailyRollingFileAppender

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

log4j.appender.stdout.layout.ConversionPattern=[%p][%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m%n

log4j.appender.stdout.File=e:/log/cyask\_wenzhen

#线上地址

#log4j.appender.stdout.file=/home/aduser/data/applog/cyask\_wenzhen\_${catalina.workerid}.log

log4j.appender.stdout.DatePattern='.'yyyy-MM-dd'.log'

log4j.appender.stdout.Encoding=UTF-8

#屏蔽框架日志输出

log4j.logger.org.springframework=OFF

log4j.logger.org.apache.struts2=OFF

log4j.logger.com.opensymphony.xwork2=OFF

log4j.logger.org.hibernate=OFF

log4j.logger.org.apache.commons=OFF

1. 自动以日志输出

action定义：

public static Logger logger=Logger.getLogger("WZLog");

log4j.properties配置

log4j.logger.WZLog=INFO,Console,WZLog

#additivity属性

#它是子Logger是否继承父Logger的输出源（appender）的标志位。具体说，默认情况下子Logger会继承父Logger的appender，

#也就是说子Logger会在父Logger的appender里输出。若是additivity设为false，

#则子Logger只会在自己的appender里输出，而不会在父Logger的appender里输出。

log4j.additivity=false

log4j.additivity.WZLog=false

#控制台输出

log4j.appender.Console=org.apache.log4j.ConsoleAppender

log4j.appender.Console.Target=System.out

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

log4j.appender.Console.layout.ConversionPattern=[%p][%d{yyyy-MM-dd HH\:mm\:ss}][%c]-%m%n

#跳转地址日志

log4j.appender.WZLog=org.apache.log4j.DailyRollingFileAppender

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

log4j.appender.WZLog.layout.ConversionPattern=[%p][%d{yyyy-MM-dd HH\:mm\:ss}][%c]%m%n

#服务器server1

log4j.appender.WZLog.File=${catalina.base}/log/wenzhen\_server1

#服务器server2

#log4j.appender.WZLog.File=${catalina.base}/log/wenzhen\_server2

#服务器server3

#log4j.appender.WZLog.File=${catalina.base}/log/wenzhen\_server3

log4j.appender.WZLog.DatePattern='\_'yyyy-MM-dd'.log'

log4j.appender.WZLog.Encoding=UTF-8

##屏蔽框架日志输出

#log4j.logger.org.springframework=OFF

#log4j.logger.org.apache.struts2=OFF

#log4j.logger.com.opensymphony.xwork2=OFF

#log4j.logger.org.hibernate=OFF

#log4j.logger.org.apache.commons=OFF