## logback 简介

Ceki Gülcü在Java日志领域世界知名。他创造了Log4J ，这个最早的Java日志框架即便在JRE内置日志功能的竞争下仍然非常流行。随后他又着手实现SLF4J 这个“简单的日志前端接口（Façade）”来替代Jakarta Commons-Logging 。

LOGBack，一个“可靠、通用、快速而又灵活的Java日志框架”。

要在工程里面使用logback需要以下jar文件：

slf4j-api-1.6.1.jar

官网网址：<http://www.slf4j.org/>

下载地址：<http://www.slf4j.org/dist/slf4j-1.6.1.zip>

logback-access-0.9.29.jar

logback-classic-0.9.29.jar

logback-core-0.9.29.jar

官网网址：<http://logback.qos.ch/>

下载地址：<http://logback.qos.ch/dist/logback-0.9.29.zip>

在工程src目录下建立logback.xml

注：

1.logback首先会试着查找logback.groovy文件;

2.当没有找到时，继续试着查找logback-test.xml文件;

3.当没有找到时，继续试着查找logback.xml文件;

4.如果仍然没有找到，则使用默认配置（打印到控制台）。

在工程src目录下建立logback包，然后在包下建立LogbackDemo.java用于测试，代码如

Java代码 [复制代码](http://aub.iteye.com/blog/1101222) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. package logback;
3. import org.slf4j.Logger;
4. import org.slf4j.LoggerFactory;
6. public class LogbackDemo {
7. private static Logger log = LoggerFactory.getLogger(LogbackDemo.class);
8. public static void main(String[] args) {
9. log.trace("======trace");
10. log.debug("======debug");
11. log.info("======info");
12. log.warn("======warn");
13. log.error("======error");
14. String name = "Aub";
15. String message = "3Q";
16. String[] fruits = { "apple", "banana" };
17. // logback提供的可以使用变量的打印方式，结果为"Hello,Aub!"
18. log.info("Hello,{}!", name);
19. // 可以有多个参数,结果为“Hello,Aub! 3Q!”
20. log.info("Hello,{}!   {}!", name, message);
21. // 可以传入一个数组，结果为"Fruit:  apple,banana"
22. log.info("Fruit:  {},{}", fruits);
23. }
24. }

## logback 配置详解（一）

**<configuration> and <logger>**

**一：根节点<configuration>包含的属性：**

scan:

当此属性设置为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。

例如：

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

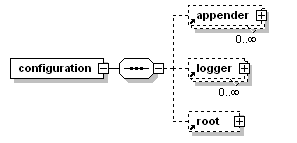
1. **<configuration** scan="true" scanPeriod="60 seconds" debug="false"**>**
2. <!-- 其他配置省略-->
3. **</configuration>**

<configuration scan="true" scanPeriod="60 seconds" debug="false">

<!-- 其他配置省略-->

</configuration>

**二：根节点<configuration>的子节点：**



**2.1设置上下文名称：<contextName>**

每个logger都关联到logger上下文，默认上下文名称为“default”。但可以使用<contextName>设置成其他名字，用于区分不同应用程序的记录。一旦设置，不能修改。

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration** scan="true" scanPeriod="60 seconds" debug="false"**>**
2. **<contextName>**myAppName**</contextName>**
3. <!-- 其他配置省略-->
4. **</configuration>**

<configuration scan="true" scanPeriod="60 seconds" debug="false">

<contextName>myAppName</contextName>

<!-- 其他配置省略-->

</configuration>

**2.2设置变量：** **<property>**

用来定义变量值的标签，<property> 有两个属性，name和value；其中name的值是变量的名称，value的值时变量定义的值。通过<property>定义的值会被插入到logger上下文中。定义变量后，可以使“${}”来使用变量。

例如使用<property>定义上下文名称，然后在<contentName>设置logger上下文时使用。

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration** scan="true" scanPeriod="60 seconds" debug="false"**>**
2. **<property** name="APP\_Name" value="myAppName" **/>**
3. **<contextName>**${APP\_Name}**</contextName>**
4. <!-- 其他配置省略-->
5. **</configuration>**

<configuration scan="true" scanPeriod="60 seconds" debug="false">

<property name="APP\_Name" value="myAppName" />

<contextName>${APP\_Name}</contextName>

<!-- 其他配置省略-->

</configuration>

**2.3获取时间戳字符串：<timestamp>**

两个属性 key:标识此<timestamp> 的名字；datePattern：设置将当前时间（解析配置文件的时间）转换为字符串的模式，遵循java.txt.SimpleDateFormat的格式。

例如将解析配置文件的时间作为上下文名称：

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration** scan="true" scanPeriod="60 seconds" debug="false"**>**
2. **<timestamp** key="bySecond" datePattern="yyyyMMdd'T'HHmmss"**/>**
3. **<contextName>**${bySecond}**</contextName>**
4. <!-- 其他配置省略-->
5. **</configuration>**

<configuration scan="true" scanPeriod="60 seconds" debug="false">

<timestamp key="bySecond" datePattern="yyyyMMdd'T'HHmmss"/>

<contextName>${bySecond}</contextName>

<!-- 其他配置省略-->

</configuration>

**2.4设置loger：**

**<loger>**

用来设置某一个包或者具体的某一个类的日志打印级别、以及指定<appender>。<loger>仅有一个name属性，一个可选的level和一个可选的addtivity属性。

name:

用来指定受此loger约束的某一个包或者具体的某一个类。

level:

用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，还有一个特俗值INHERITED或者同义词NULL，代表强制执行上级的级别。

如果未设置此属性，那么当前loger将会继承上级的级别。

addtivity:

是否向上级loger传递打印信息。默认是true。

<loger>可以包含零个或多个<appender-ref>元素，标识这个appender将会添加到这个loger。

**<root>**

也是<loger>元素，但是它是根loger。只有一个level属性，应为已经被命名为"root".

level:

用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，不能设置为INHERITED或者同义词NULL。

默认是DEBUG。

<root>可以包含零个或多个<appender-ref>元素，标识这个appender将会添加到这个loger。

例如：

LogbackDemo.java类

Java代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. package logback;
3. import org.slf4j.Logger;
4. import org.slf4j.LoggerFactory;
6. public class LogbackDemo {
7. private static Logger log = LoggerFactory.getLogger(LogbackDemo.class);
8. public static void main(String[] args) {
9. log.trace("======trace");
10. log.debug("======debug");
11. log.info("======info");
12. log.warn("======warn");
13. log.error("======error");
14. }
15. }

package logback;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LogbackDemo {

private static Logger log = LoggerFactory.getLogger(LogbackDemo.class);

public static void main(String[] args) {

log.trace("======trace");

log.debug("======debug");

log.info("======info");

log.warn("======warn");

log.error("======error");

}

}

logback.xml配置文件

**第1种：只配置root**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
2. **<appender** name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"**>**
3. <!-- encoder 默认配置为PatternLayoutEncoder -->
4. **<encoder>**
5. **<pattern>**%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n**</pattern>**
6. **</encoder>**
7. **</appender>**
8. **<root** level="INFO"**>**
9. **<appender-ref** ref="STDOUT" **/>**
10. **</root>**
11. **</configuration>**

<configuration>

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

<!-- encoder 默认配置为PatternLayoutEncoder -->

<encoder>

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

</encoder>

</appender>

<root level="INFO">

<appender-ref ref="STDOUT" />

</root>

</configuration>

 其中appender的配置表示打印到控制台(稍后详细讲解appender )；

<root level="INFO">将root的打印级别设置为“INFO”，指定了名字为“STDOUT”的appender。

当执行logback.LogbackDemo类的main方法时，root将级别为“INFO”及大于“INFO”的日志信息交给已经配置好的名为“STDOUT”的appender处理，“STDOUT”appender将信息打印到控制台；

**打印结果如下：**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. 13:30:38.484 [main] INFO  logback.LogbackDemo - ======info
2. 13:30:38.500 [main] WARN  logback.LogbackDemo - ======warn
3. 13:30:38.500 [main] ERROR logback.LogbackDemo - ======error

13:30:38.484 [main] INFO logback.LogbackDemo - ======info

13:30:38.500 [main] WARN logback.LogbackDemo - ======warn

13:30:38.500 [main] ERROR logback.LogbackDemo - ======error

**第2种：带有loger的配置，不指定级别，不指定appender，**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <configuration>
3. <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
4. <!-- encoder 默认配置为PatternLayoutEncoder -->
5. <encoder>
6. <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
7. </encoder>
8. </appender>
10. <!-- logback为java中的包 -->
11. <logger name="logback"/>
13. <root level="DEBUG">
14. <appender-ref ref="STDOUT" />
15. </root>
17. </configuration>

<configuration>

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

<!-- encoder 默认配置为PatternLayoutEncoder -->

<encoder>

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

</encoder>

</appender>

<!-- logback为java中的包 -->

<logger name="logback"/>

<root level="DEBUG">

<appender-ref ref="STDOUT" />

</root>

</configuration>

 其中appender的配置表示打印到控制台(稍后详细讲解appender )；

<logger name="logback" />将控制logback包下的所有类的日志的打印，但是并没用设置打印级别，所以继承他的上级<root>的日志级别“DEBUG”；

没有设置addtivity，默认为true，将此loger的打印信息向上级传递；

没有设置appender，此loger本身不打印任何信息。

<root level="DEBUG">将root的打印级别设置为“DEBUG”，指定了名字为“STDOUT”的appender。

当执行logback.LogbackDemo类的main方法时，因为LogbackDemo 在包logback中，所以首先执行<logger name="logback" />，将级别为“DEBUG”及大于“DEBUG”的日志信息传递给root，本身并不打印；

root接到下级传递的信息，交给已经配置好的名为“STDOUT”的appender处理，“STDOUT”appender将信息打印到控制台；

**打印结果如下：**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. 13:19:15.406 [main] DEBUG logback.LogbackDemo - ======debug
2. 13:19:15.406 [main] INFO  logback.LogbackDemo - ======info
3. 13:19:15.406 [main] WARN  logback.LogbackDemo - ======warn
4. 13:19:15.406 [main] ERROR logback.LogbackDemo - ======error

13:19:15.406 [main] DEBUG logback.LogbackDemo - ======debug

13:19:15.406 [main] INFO logback.LogbackDemo - ======info

13:19:15.406 [main] WARN logback.LogbackDemo - ======warn

13:19:15.406 [main] ERROR logback.LogbackDemo - ======error

**第3种：带有多个loger的配置，指定级别，指定appender**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
2. **<appender** name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"**>**
3. <!-- encoder 默认配置为PatternLayoutEncoder -->
4. **<encoder>**
5. **<pattern>**%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n**</pattern>**
6. **</encoder>**
7. **</appender>**
8. <!-- logback为java中的包 -->
9. **<logger** name="logback"**/>**
10. <!--logback.LogbackDemo：类的全路径 -->
11. **<logger** name="logback.LogbackDemo" level="INFO" additivity="false"**>**
12. **<appender-ref** ref="STDOUT"**/>**
13. **</logger>**
14. **<root** level="ERROR"**>**
15. **<appender-ref** ref="STDOUT"**/>**
16. **</root>**
17. **</configuration>**

<configuration>

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

<!-- encoder 默认配置为PatternLayoutEncoder -->

<encoder>

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

</encoder>

</appender>

<!-- logback为java中的包 -->

<logger name="logback"/>

<!--logback.LogbackDemo：类的全路径 -->

<logger name="logback.LogbackDemo" level="INFO" additivity="false">

<appender-ref ref="STDOUT"/>

</logger>

<root level="ERROR">

<appender-ref ref="STDOUT" />

</root>

</configuration>

其中appender的配置表示打印到控制台(稍后详细讲解appender )；

<logger name="logback" />将控制logback包下的所有类的日志的打印，但是并没用设置打印级别，所以继承他的上级<root>的日志级别“DEBUG”；

没有设置addtivity，默认为true，将此loger的打印信息向上级传递；

没有设置appender，此loger本身不打印任何信息。

 <logger name="logback.LogbackDemo" level="INFO" additivity="false">控制logback.LogbackDemo类的日志打印，打印级别为“INFO”；

additivity属性为false，表示此loger的打印信息不再向上级传递，

指定了名字为“STDOUT”的appender。

<root level="DEBUG">将root的打印级别设置为“ERROR”，指定了名字为“STDOUT”的appender。

 当执行logback.LogbackDemo类的main方法时，先执行<logger name="logback.LogbackDemo" level="INFO" additivity="false">，将级别为“INFO”及大于“INFO”的日志信息交给此loger指定的名为“STDOUT”的appender处理，在控制台中打出日志，不再向次loger的上级 <logger name="logback"/> 传递打印信息；

<logger name="logback"/>未接到任何打印信息，当然也不会给它的上级root传递任何打印信息；

**打印结果如下：**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. 14:05:35.937 [main] INFO  logback.LogbackDemo - ======info
2. 14:05:35.937 [main] WARN  logback.LogbackDemo - ======warn
3. 14:05:35.937 [main] ERROR logback.LogbackDemo - ======error

14:05:35.937 [main] INFO logback.LogbackDemo - ======info

14:05:35.937 [main] WARN logback.LogbackDemo - ======warn

14:05:35.937 [main] ERROR logback.LogbackDemo - ======error

 如果将<logger name="logback.LogbackDemo" level="INFO" additivity="false">修改为 <logger name="logback.LogbackDemo" level="INFO" additivity="true">那打印结果将是什么呢？

没错，日志打印了两次，想必大家都知道原因了，因为打印信息向上级传递，logger本身打印一次，root接到后又打印一次

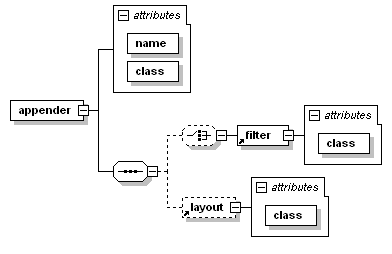
**打印结果如下：**

Xml代码 [复制代码](http://aub.iteye.com/blog/1101260) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. 14:09:01.531 [main] INFO  logback.LogbackDemo - ======info
2. 14:09:01.531 [main] INFO  logback.LogbackDemo - ======info
3. 14:09:01.531 [main] WARN  logback.LogbackDemo - ======warn
4. 14:09:01.531 [main] WARN  logback.LogbackDemo - ======warn
5. 14:09:01.531 [main] ERROR logback.LogbackDemo - ======error
6. 14:09:01.531 [main] ERROR logback.LogbackDemo - ======error

## logback 常用配置详解（二）

**<appender>**



**<appender>：**

<appender>是<configuration>的子节点，是负责写日志的组件。

<appender>有两个必要属性name和class。name指定appender名称，class指定appender的全限定名。

**1.ConsoleAppender:**

把日志添加到控制台，有以下子节点：

<encoder>：对日志进行格式化。（具体参数稍后讲解 ）

<target>：字符串 *System.out* 或者 *System.err* ，默认 *System.out* ；

例如：

Xml代码 [复制代码](http://aub.iteye.com/blog/1103685) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
3. **<appender** name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"**>**
4. **<encoder>**
5. **<pattern>**%-4relative [%thread] %-5level %logger{35} - %msg %n**</pattern>**
6. **</encoder>**
7. **</appender>**
9. **<root** level="DEBUG"**>**
10. **<appender-ref** ref="STDOUT" **/>**
11. **</root>**
12. **</configuration>**

<configuration>

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

<encoder>

<pattern>%-4relative [%thread] %-5level %logger{35} - %msg %n</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="STDOUT" />

</root>

</configuration>

**2.FileAppender:**

把日志添加到文件，有以下子节点：

<file>：被写入的文件名，可以是相对目录，也可以是绝对目录，如果上级目录不存在会自动创建，没有默认值。

<append>：如果是 true，日志被追加到文件结尾，如果是 false，清空现存文件，默认是true。

<encoder>：对记录事件进行格式化。（具体参数稍后讲解 ）

<prudent>：如果是 true，日志会被安全的写入文件，即使其他的FileAppender也在向此文件做写入操作，效率低，默认是 false。

例如：

Xml代码 [复制代码](http://aub.iteye.com/blog/1103685) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
2. **<appender** name="FILE" class="ch.qos.logback.core.FileAppender"**>**
3. **<file>**testFile.log**</file>**
4. **<append>**true**</append>**
5. **<encoder>**
6. **<pattern>**%-4relative [%thread] %-5level %logger{35} - %msg%n**</pattern>**
7. **</encoder>**
8. **</appender>**
9. **<root** level="DEBUG"**>**
10. **<appender-ref** ref="FILE" **/>**
11. **</root>**
12. **</configuration>**

<configuration>

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

<file>testFile.log</file>

<append>true</append>

<encoder>

<pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="FILE" />

</root>

</configuration>

**3.RollingFileAppender:**

滚动记录文件，先将日志记录到指定文件，当符合某个条件时，将日志记录到其他文件。有以下子节点：

<file>：被写入的文件名，可以是相对目录，也可以是绝对目录，如果上级目录不存在会自动创建，没有默认值。

<append>：如果是 true，日志被追加到文件结尾，如果是 false，清空现存文件，默认是true。

<encoder>：对记录事件进行格式化。（具体参数稍后讲解 ）

<rollingPolicy>:当发生滚动时，决定 **RollingFileAppender** 的行为，涉及文件移动和重命名。

<triggeringPolicy **>:** 告知 **RollingFileAppender** 合适激活滚动。

<prudent>：当为true时，不支持FixedWindowRollingPolicy。支持TimeBasedRollingPolicy，但是有两个限制，1不支持也不允许文件压缩，2不能设置file属性，必须留空。

**rollingPolicy：**

**TimeBasedRollingPolicy：** 最常用的滚动策略，它根据时间来制定滚动策略，既负责滚动也负责出发滚动。有以下子节点：

<fileNamePattern>:

必要节点，包含文件名及“%d”转换符， “%d”可以包含一个 java.text.SimpleDateFormat指定的时间格式，如：%d{yyyy-MM}。如果直接使用 %d，默认格式是 yyyy-MM-dd。 **RollingFileAppender** 的file字节点可有可无，通过设置file，可以为活动文件和归档文件指定不同位置，当前日志总是记录到file指定的文件（活动文件），活动文件的名字不会改变；如果没设置file，活动文件的名字会根据**fileNamePattern** 的值，每隔一段时间改变一次。“/”或者“\”会被当做目录分隔符。

<maxHistory>:

可选节点，控制保留的归档文件的最大数量，超出数量就删除旧文件。假设设置每个月滚动，且 <maxHistory>是6，则只保存最近6个月的文件，删除之前的旧文件。注意，删除旧文件是，那些为了归档而创建的目录也会被删除。

**FixedWindowRollingPolicy：** 根据固定窗口算法重命名文件的滚动策略。有以下子节点：

<minIndex>:窗口索引最小值

<maxIndex>:窗口索引最大值，当用户指定的窗口过大时，会自动将窗口设置为12。

<fileNamePattern >:

必须包含“%i”例如，假设最小值和最大值分别为1和2，命名模式为 mylog%i.log,会产生归档文件mylog1.log和mylog2.log。还可以指定文件压缩选项，例如，mylog%i.log.gz 或者 没有log%i.log.zip

**triggeringPolicy:**

**SizeBasedTriggeringPolicy：** 查看当前活动文件的大小，如果超过指定大小会告知 **RollingFileAppender** 触发当前活动文件滚动。只有一个节点**:**

<maxFileSize>:这是活动文件的大小，默认值是10MB。

例如：每天生成一个日志文件，保存30天的日志文件。

Java代码 [复制代码](http://aub.iteye.com/blog/1103685) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. <configuration>
2. <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
3. <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
4. <fileNamePattern>logFile.%d{yyyy-MM-dd}.log</fileNamePattern>
5. <maxHistory>30</maxHistory>
6. </rollingPolicy>
7. <encoder>
8. <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>
9. </encoder>
10. </appender>
11. <root level="DEBUG">
12. <appender-ref ref="FILE" />
13. </root>
14. </configuration>

<configuration>

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

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

<fileNamePattern>logFile.%d{yyyy-MM-dd}.log</fileNamePattern>

<maxHistory>30</maxHistory>

</rollingPolicy>

<encoder>

<pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="FILE" />

</root>

</configuration>

  例如：按照固定窗口模式生成日志文件，当文件大于20MB时，生成新的日志文件。窗口大小是1到3，当保存了3个归档文件后，将覆盖最早的日志。

Xml代码 [复制代码](http://aub.iteye.com/blog/1103685) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
2. **<appender** name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender"**>**
3. **<file>**test.log**</file>**
4. **<rollingPolicy** class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy"**>**
5. **<fileNamePattern>**tests.%i.log.zip**</fileNamePattern>**
6. **<minIndex>**1**</minIndex>**
7. **<maxIndex>**3**</maxIndex>**
8. **</rollingPolicy>**
9. **<triggeringPolicy** class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy"**>**
10. **<maxFileSize>**5MB**</maxFileSize>**
11. **</triggeringPolicy>**
12. **<encoder>**
13. **<pattern>**%-4relative [%thread] %-5level %logger{35} - %msg%n**</pattern>**
14. **</encoder>**
15. **</appender>**
16. **<root** level="DEBUG"**>**
17. **<appender-ref** ref="FILE" **/>**
18. **</root>**
19. **</configuration>**

<configuration>

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

<file>test.log</file>

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

<fileNamePattern>tests.%i.log.zip</fileNamePattern>

<minIndex>1</minIndex>

<maxIndex>3</maxIndex>

</rollingPolicy>

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

<maxFileSize>5MB</maxFileSize>

</triggeringPolicy>

<encoder>

<pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="FILE" />

</root>

</configuration>

4.另外还有SocketAppender、SMTPAppender、DBAppender、SyslogAppender、SiftingAppender，并不常用，这些就不在这里讲解了，大家可以参考官方文档。当然大家可以编写自己的Appender。

**<encoder>：**

负责两件事，一是把日志信息转换成字节数组，二是把字节数组写入到输出流。

目前**PatternLayoutEncoder** 是唯一有用的且默认的**encoder** ，有一个<pattern>节点，用来设置日志的输入格式。使用“%”加“转换符”方式，如果要输出“%”，则必须用“\”对“\%”进行转义。

例如：

Xml代码 [复制代码](http://aub.iteye.com/blog/1103685) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<encoder>**
2. **<pattern>**%-4relative [%thread] %-5level %logger{35} - %msg%n**</pattern>**
3. **</encoder**

<encoder>

<pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>

</encoder

**<pattern>里面的转换符说明：**

|  |  |
| --- | --- |
| **转换符** | **作用** |
| **c** {*length* }  **lo** {*length* }  **logger** {*length* } | 输出日志的logger名，可有一个整形参数，功能是缩短logger名，设置为0表示只输入logger最右边点符号之后的字符串。   |  |  |  | | --- | --- | --- | | **Conversion specifier** | **Logger name** | **Result** | | %logger | mainPackage.sub.sample.Bar | mainPackage.sub.sample.Bar | | %logger{0} | mainPackage.sub.sample.Bar | Bar | | %logger{5} | mainPackage.sub.sample.Bar | m.s.s.Bar | | %logger{10} | mainPackage.sub.sample.Bar | m.s.s.Bar | | %logger{15} | mainPackage.sub.sample.Bar | m.s.sample.Bar | | %logger{16} | mainPackage.sub.sample.Bar | m.sub.sample.Bar | | %logger{26} | mainPackage.sub.sample.Bar | mainPackage.sub.sample.Bar | |
| **C** {*length* }  **class** {*length* } | 输出执行记录请求的调用者的全限定名。参数与上面的一样。尽量避免使用，除非执行速度不造成任何问题。 |
| **contextName**  **cn** | 输出上下文名称。 |
| **d** {*pattern* }  **date** {*pattern* } | 输出日志的打印日志，模式语法与java.text.SimpleDateFormat 兼容。   |  |  | | --- | --- | | **Conversion Pattern** | **Result** | | %d | 2006-10-20 14:06:49,812 | | %date | 2006-10-20 14:06:49,812 | | %date{ISO8601} | 2006-10-20 14:06:49,812 | | %date{HH:mm:ss.SSS} | 14:06:49.812 | | %date{dd MMM yyyy ;HH:mm:ss.SSS} | 20 oct. 2006;14:06:49.812 | |
| **F / file** | 输出执行记录请求的java源文件名。尽量避免使用，除非执行速度不造成任何问题。 |
| **caller{depth}** **caller{depth, evaluator-1, ... evaluator-n}** | 输出生成日志的调用者的位置信息，整数选项表示输出信息深度。  例如， **%caller{2}**   输出为：  0 [main] DEBUG - logging statement  Caller+0 at mainPackage.sub.sample.Bar.sampleMethodName(Bar.java:22)  Caller+1 at mainPackage.sub.sample.Bar.createLoggingRequest(Bar.java:17)  例如， **%caller{3}**   输出为：  16 [main] DEBUG - logging statement  Caller+0 at mainPackage.sub.sample.Bar.sampleMethodName(Bar.java:22)  Caller+1 at mainPackage.sub.sample.Bar.createLoggingRequest(Bar.java:17)  Caller+2 at mainPackage.ConfigTester.main(ConfigTester.java:38) |
| **L / line** | 输出执行日志请求的行号。尽量避免使用，除非执行速度不造成任何问题。 |
| **m / msg / message** | 输出应用程序提供的信息。 |
| **M / method** | 输出执行日志请求的方法名。尽量避免使用，除非执行速度不造成任何问题。 |
| **n** | 输出平台先关的分行符“\n”或者“\r\n”。 |
| **p / le / level** | 输出日志级别。 |
| **r / relative** | 输出从程序启动到创建日志记录的时间，单位是毫秒 |
| **t / thread** | 输出产生日志的线程名。 |
| **replace(*p* ){r, t}** | **p** 为日志内容，**r** 是正则表达式，将**p** 中符合**r** 的内容替换为**t** 。  例如， "%replace(%msg){'\s', ''}" |

**格式修饰符，与转换符共同使用：**

可选的格式修饰符位于“%”和转换符之间。

第一个可选修饰符是

**左对齐** 标志，符号是减号“-”；接着是可选的

**最小宽度** 修饰符，用十进制数表示。如果字符小于最小宽度，则左填充或右填充，默认是左填充（即右对齐），填充符为空格。如果字符大于最小宽度，字符永远不会被截断。

**最大宽度** 修饰符，符号是点号"."后面加十进制数。如果字符大于最大宽度，则从前面截断。点符号“.”后面加减号“-”在加数字，表示从尾部截断。

例如：%-4relative 表示，将输出从程序启动到创建日志记录的时间 进行左对齐 且最小宽度为4。

## logback 常用配置详解（三）

**<filter>:**

过滤器，执行一个过滤器会有返回个枚举值，即DENY，NEUTRAL，ACCEPT其中之一。返回DENY，日志将立即被抛弃不再经过其他过滤器；返回NEUTRAL，有序列表里的下个过滤器过接着处理日志；返回ACCEPT，日志会被立即处理，不再经过剩余过滤器。

过滤器被添加到**<Appender>** 中，为**<Appender>** 添加一个或多个过滤器后，可以用任意条件对日志进行过滤。**<Appender>** 有多个过滤器时，按照配置顺序执行。

下面是几个常用的过滤器：

**LevelFilter：** 级别过滤器，根据日志级别进行过滤。如果日志级别等于配置级别，过滤器会根据onMath 和 onMismatch接收或拒绝日志。有以下子节点：

<level>:设置过滤级别

<onMatch>:用于配置符合过滤条件的操作

<onMismatch>:用于配置不符合过滤条件的操作

例如：将过滤器的日志级别配置为INFO，所有INFO级别的日志交给appender处理，非INFO级别的日志，被过滤掉。

Xml代码 [复制代码](http://aub.iteye.com/blog/1110008) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
2. **<appender** name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender"**>**
3. **<filter** class="ch.qos.logback.classic.filter.LevelFilter"**>**
4. **<level>**INFO**</level>**
5. **<onMatch>**ACCEPT**</onMatch>**
6. **<onMismatch>**DENY**</onMismatch>**
7. **</filter>**
8. **<encoder>**
9. **<pattern>**
10. %-4relative [%thread] %-5level %logger{30} - %msg%n
11. **</pattern>**
12. **</encoder>**
13. **</appender>**
14. **<root** level="DEBUG"**>**
15. **<appender-ref** ref="CONSOLE" **/>**
16. **</root>**
17. **</configuration>**

<configuration>

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

<filter class="ch.qos.logback.classic.filter.LevelFilter">

<level>INFO</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

<encoder>

<pattern>

%-4relative [%thread] %-5level %logger{30} - %msg%n

</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="CONSOLE" />

</root>

</configuration>

**ThresholdFilter：** 临界值过滤器，过滤掉低于指定临界值的日志。当日志级别等于或高于临界值时，过滤器返回NEUTRAL；当日志级别低于临界值时，日志会被拒绝。

例如：过滤掉所有低于INFO级别的日志。

Xml代码 [复制代码](http://aub.iteye.com/blog/1110008) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
2. **<appender** name="CONSOLE"
3. class="ch.qos.logback.core.ConsoleAppender"**>**
4. <!-- 过滤掉 TRACE 和 DEBUG 级别的日志-->
5. **<filter** class="ch.qos.logback.classic.filter.ThresholdFilter"**>**
6. **<level>**INFO**</level>**
7. **</filter>**
8. **<encoder>**
9. **<pattern>**
10. %-4relative [%thread] %-5level %logger{30} - %msg%n
11. **</pattern>**
12. **</encoder>**
13. **</appender>**
14. **<root** level="DEBUG"**>**
15. **<appender-ref** ref="CONSOLE" **/>**
16. **</root>**
17. **</configuration>**

<configuration>

<appender name="CONSOLE"

class="ch.qos.logback.core.ConsoleAppender">

<!-- 过滤掉 TRACE 和 DEBUG 级别的日志-->

<filter class="ch.qos.logback.classic.filter.ThresholdFilter">

<level>INFO</level>

</filter>

<encoder>

<pattern>

%-4relative [%thread] %-5level %logger{30} - %msg%n

</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="CONSOLE" />

</root>

</configuration>

**EvaluatorFilter：** 求值过滤器，评估、鉴别日志是否符合指定条件。有一下子节点：

<evaluator>:

鉴别器，常用的鉴别器是JaninoEventEvaluato，也是默认的鉴别器，它以任意的java布尔值表达式作为求值条件，求值条件在配置文件解释过成功被动态编译，布尔值表达式返回true就表示符合过滤条件。evaluator有个子标签<expression>，用于配置求值条件。

求值表达式作用于当前日志，logback向求值表达式暴露日志的各种字段：

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| event | LoggingEvent | 与记录请求相关联的原始记录事件，下面所有变量都来自event，例如，event.getMessage()返回下面"message"相同的字符串 |
| message | String | 日志的原始消息，例如，设有logger mylogger，"name"的值是"AUB"，对于 mylogger.info("Hello {}",name); "Hello {}"就是原始消息。 |
| formatedMessage | String | 日志被各式话的消息，例如，设有logger mylogger，"name"的值是"AUB"，对于 mylogger.info("Hello {}",name); "Hello Aub"就是格式化后的消息。 |
| logger | String | logger 名。 |
| loggerContext | [LoggerContextVO](http://aub.iteye.com/admin/blogs/xref/ch/qos/logback/classic/spi/LoggerContextVO.html) | 日志所属的logger上下文。 |
| level | int | 级别对应的整数值，所以 *level > INFO* 是正确的表达式。 |
| timeStamp | long | 创建日志的时间戳。 |
| marker | Marker | 与日志请求相关联的Marker对象，注意“Marker”有可能为null，所以你要确保它不能是null。 |
| mdc | Map | 包含创建日志期间的MDC所有值得map。访问方法是： *mdc.get("myKey")* 。mdc.get()返回的是Object不是String，要想调用String的方法就要强转，例如，  ((String) mdc.get("k")).contains("val") .MDC可能为null，调用时注意。 |
| throwable | java.lang.Throwable | 如果没有异常与日志关联"throwable" 变量为 null. 不幸的是， "throwable" 不能被序列化。在远程系统上永远为null，对于与位置无关的表达式请使用下面的变量throwableProxy |
| throwableProxy | [IThrowableProxy](http://aub.iteye.com/admin/blogs/xref/ch/qos/logback/classic/spi/IThrowableProxy.html) | 与日志事件关联的异常代理。如果没有异常与日志事件关联，则变量"throwableProxy" 为 null. 当异常被关联到日志事件时，"throwableProxy" 在远程系统上不会为null |

<onMatch>:用于配置符合过滤条件的操作

<onMismatch>:用于配置不符合过滤条件的操作

例如：过滤掉所有日志消息中不包含“billing”字符串的日志。

Xml代码 [复制代码](http://aub.iteye.com/blog/1110008) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration>**
3. **<appender** name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"**>**
4. **<filter** class="ch.qos.logback.core.filter.EvaluatorFilter"**>**
5. **<evaluator>** <!-- 默认为 ch.qos.logback.classic.boolex.JaninoEventEvaluator -->
6. **<expression>**return message.contains("billing");**</expression>**
7. **</evaluator>**
8. **<OnMatch>**ACCEPT **</OnMatch>**
9. **<OnMismatch>**DENY**</OnMismatch>**
10. **</filter>**
11. **<encoder>**
12. **<pattern>**
13. %-4relative [%thread] %-5level %logger - %msg%n
14. **</pattern>**
15. **</encoder>**
16. **</appender>**
17. **<root** level="INFO"**>**
18. **<appender-ref** ref="STDOUT" **/>**
19. **</root>**
20. **</configuration>**

<configuration>

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

<filter class="ch.qos.logback.core.filter.EvaluatorFilter">

<evaluator> <!-- 默认为 ch.qos.logback.classic.boolex.JaninoEventEvaluator -->

<expression>return message.contains("billing");</expression>

</evaluator>

<OnMatch>ACCEPT </OnMatch>

<OnMismatch>DENY</OnMismatch>

</filter>

<encoder>

<pattern>

%-4relative [%thread] %-5level %logger - %msg%n

</pattern>

</encoder>

</appender>

<root level="INFO">

<appender-ref ref="STDOUT" />

</root>

</configuration>

**<matcher> ：**

匹配器，尽管可以使用String类的matches()方法进行模式匹配，但会导致每次调用过滤器时都会创建一个新的Pattern对象，为了消除这种开销，可以预定义一个或多个matcher对象，定以后就可以在求值表达式中重复引用。<matcher>是<evaluator>的子标签。

<matcher>中包含两个子标签，一个是<name>，用于定义matcher的名字，求值表达式中使用这个名字来引用matcher；另一个是<regex>，用于配置匹配条件。

例如：

Xml代码 [复制代码](http://aub.iteye.com/blog/1110008) [[收藏代码http://aub.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. **<configuration** debug="true"**>**
2. **<appender** name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"**>**
3. **<filter** class="ch.qos.logback.core.filter.EvaluatorFilter"**>**
4. **<evaluator>**
5. **<matcher>**
6. **<Name>**odd**</Name>**
7. <!-- filter out odd numbered statements -->
8. **<regex>**statement [13579]**</regex>**
9. **</matcher>**
10. **<expression>**odd.matches(formattedMessage)**</expression>**
11. **</evaluator>**
12. **<OnMismatch>**NEUTRAL**</OnMismatch>**
13. **<OnMatch>**DENY**</OnMatch>**
14. **</filter>**
15. **<encoder>**
16. **<pattern>**%-4relative [%thread] %-5level %logger - %msg%n**</pattern>**
17. **</encoder>**
18. **</appender>**
19. **<root** level="DEBUG"**>**
20. **<appender-ref** ref="STDOUT" **/>**
21. **</root>**
22. **</configuration>**

<configuration debug="true">

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

<filter class="ch.qos.logback.core.filter.EvaluatorFilter">

<evaluator>

<matcher>

<Name>odd</Name>

<!-- filter out odd numbered statements -->

<regex>statement [13579]</regex>

</matcher>

<expression>odd.matches(formattedMessage)</expression>

</evaluator>

<OnMismatch>NEUTRAL</OnMismatch>

<OnMatch>DENY</OnMatch>

</filter>

<encoder>

<pattern>%-4relative [%thread] %-5level %logger - %msg%n</pattern>

</encoder>

</appender>

<root level="DEBUG">

<appender-ref ref="STDOUT" />

</root>

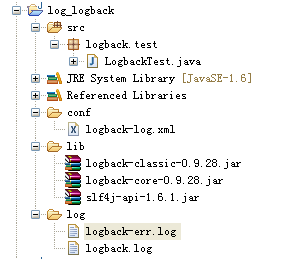
</configuration>

其他Filter不太常用我这里就不讲了，大家可以参见官网。14:09:01.531 [main] INFO logback.LogbackDemo - ======info14:09:01.531 [main] INFO logback.LogbackDemo - ======info

## logback+slf4j日志实例

**实例：**

1. **结构图**

[](http://photo.blog.sina.com.cn/showpic.html#blogid=3fc7e2810100vio9&url=http://s6.sinaimg.cn/orignal/3fc7e281ga9f43f593b25)

**2、 Jar包**

logback-classic-0.9.28.jar

logback-core-0.9.28.jar

slf4j-api-1.6.1.jar

**3、 TestLogback.java**

package logback.test;

import ch.qos.logback.classic.LoggerContext;

import ch.qos.logback.classic.joran.JoranConfigurator;

import ch.qos.logback.core.joran.spi.JoranException;

import ch.qos.logback.core.util.StatusPrinter;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LogbackTest {

private static final Logger log = LoggerFactory.getLogger(LogbackTest.class);

public static void main(String[] args) {

// TODO Auto-generated method stub

log.info("logback + slf4j starting up ...");

LoggerContext lc = (LoggerContext)LoggerFactory.getILoggerFactory();

try {

JoranConfigurator configurator = new JoranConfigurator();

configurator.setContext(lc);

lc.reset();

configurator.doConfigure("conf/logback-log.xml");

}

catch (JoranException e) {

log.error(e.getMessage());

}

//打印logback加载过程

StatusPrinter.printInCaseOfErrorsOrWarnings(lc);

log.info("flyfox server daemon run.");

log.info("test info");

try {

Thread.sleep(100);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

log.debug("test debug");

log.error("test error");

}

}

**4、 logback-log.xml**

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

<configuration>

<!-- 控制台输出日志 -->

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

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

<pattern>%d{yyyy-MM-dd HH:mm:ss}, %p, %c, %t, %r, %ex, %F, %L, %C{1}, %M : %m%n</pattern>

</layout>

</appender>

<!-- 文件输出日志 (文件大小策略进行文件输出，超过指定大小对文件备份)-->

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

<File>log/logback.log</File>

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

<FileNamePattern>log/sshd.log.%i.bak</FileNamePattern>

<MinIndex>1</MinIndex>

<MaxIndex>12</MaxIndex>

</rollingPolicy>

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

<MaxFileSize>5MB</MaxFileSize>

</triggeringPolicy>

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

<Pattern>%d{yyyy-MM-dd HH:mm:ss}, %p, %c, %t, %r, %ex, %F, %L, %C{1}, %M %m%n</Pattern>

</layout>

</appender>

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

<filter class=*"ch.qos.logback.classic.filter.LevelFilter"*>

<level>ERROR</level>

<OnMismatch>DENY</OnMismatch>

<OnMatch>ACCEPT</OnMatch>

</filter>

<File>log/logback-err.log</File>

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

<FileNamePattern>log/sshd-err.log.%i.bak</FileNamePattern>

<MinIndex>1</MinIndex>

<MaxIndex>3</MaxIndex>

</rollingPolicy>

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

<MaxFileSize>5MB</MaxFileSize>

</triggeringPolicy>

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

<Pattern>%d{yyyy-MM-dd HH:mm:ss}, %p, %c, %t, %r, %ex, %F, %L, %C{1}, %M %m%n</Pattern>

</layout>

</appender>

<root level=*"ALL"*>

<appender-ref ref=*"STDOUT"*/>

<appender-ref ref=*"FILE"*/>

<appender-ref ref=*"FILE-ERROR"*/>

</root>

</configuration>

**5、 logback.log和logback-err.log**

logback.log

2011-07-22 21:32:03, INFO, logback.test.LogbackTest, main, 328, , LogbackTest.java, 33, l.t.LogbackTest, main flyfox server daemon run.

2011-07-22 21:32:03, INFO, logback.test.LogbackTest, main, 328, , LogbackTest.java, 34, l.t.LogbackTest, main test info

2011-07-22 21:32:03, DEBUG, logback.test.LogbackTest, main, 438, , LogbackTest.java, 41, l.t.LogbackTest, main test debug

2011-07-22 21:32:03, ERROR, logback.test.LogbackTest, main, 438, , LogbackTest.java, 42, l.t.LogbackTest, main test error

logback-err.log

2011-07-22 21:32:03, ERROR, logback.test.LogbackTest, main, 438, , LogbackTest.java, 42, l.t.LogbackTest, main test error

**6、 控制台输出**

21:32:03.562 [main] INFO logback.test.LogbackTest - logback + slf4j starting up ...

21:32:03,468 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.groovy]

21:32:03,468 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback-test.xml]

21:32:03,468 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Could NOT find resource [logback.xml]

21:32:03,468 |-INFO in ch.qos.logback.classic.LoggerContext[default] - Setting up default configuration.

21:32:03,671 |-INFO in ch.qos.logback.classic.joran.action.ConfigurationAction - debug attribute not set

21:32:03,687 |-INFO in ch.qos.logback.core.joran.action.AppenderAction - About to instantiate appender of type [ch.qos.logback.core.ConsoleAppender]

………………………………………………………………

………………………………………………………………

………………………………………………………………

21:32:03,765 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appender named [FILE] to Logger[ROOT]

21:32:03,765 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction - Attaching appender named [FILE-ERROR] to Logger[ROOT]

2011-07-22 21:32:03, INFO, logback.test.LogbackTest, main, 328, , LogbackTest.java, 33, l.t.LogbackTest, main : flyfox server daemon run.

2011-07-22 21:32:03, INFO, logback.test.LogbackTest, main, 328, , LogbackTest.java, 34, l.t.LogbackTest, main : test info

2011-07-22 21:32:03, DEBUG, logback.test.LogbackTest, main, 438, , LogbackTest.java, 41, l.t.LogbackTest, main : test debug

2011-07-22 21:32:03, ERROR, logback.test.LogbackTest, main, 438, , LogbackTest.java, 42, l.t.LogbackTest, main : test error

14:09:01.531 [main] WARN logback.LogbackDemo - ======warn

14:09:01.531 [main] WARN logback.LogbackDemo - ======warn

14:09:01.531 [main] ERROR logback.LogbackDemo - ======error

14:09:01.531 [main] ERROR logback.LogbackDemo - ======error