从零开始玩转logback

2016/11/08 | 分类： 基础技术 | 3 条评论 | 标签： logback, 日志

分享到: 33 原文出处： 朱小厮

概述

LogBack是一个日志框架，它与Log4j可以说是同出一源，都出自Ceki Gülcü之手。（log4j的原型是早前由Ceki Gülcü贡献给Apache基金会的）下载地址：http://logback.qos.ch/download.html

LogBack、Slf4j和Log4j之间的关系

Slf4j是The Simple Logging Facade for Java的简称，是一个简单日志门面抽象框架，它本身只提供了日志Facade API和一个简单的日志类实现，一般常配合Log4j，LogBack，java.util.logging使用。Slf4j作为应用层的Log接入时，程序可以根据实际应用场景动态调整底层的日志实现框架(Log4j/LogBack/JdkLog...)。

LogBack和Log4j都是开源日记工具库，LogBack是Log4j的改良版本，比Log4j拥有更多的特性，同时也带来很大性能提升。详细数据可参照下面地址：Reasons to prefer logback over log4j。

LogBack官方建议配合Slf4j使用，这样可以灵活地替换底层日志框架。

> TIPS：为了优化log4j，以及更大性能的提升，Apache基金会已经着手开发了log4j 2.0, 其中也借鉴和吸收了logback的一些先进特性，目前log4j2还处于beta阶段。

LogBack的结构

LogBack被分为3个组件，logback-core, logback-classic 和 logback-access。

其中logback-core提供了LogBack的核心功能，是另外两个组件的基础。

logback-classic则实现了Slf4j的API，所以当想配合Slf4j使用时，需要将logback-classic加入classpath。

logback-access是为了集成Servlet环境而准备的，可提供HTTP-access的日志接口。

配置详解

根节点<configuration>包含的属性

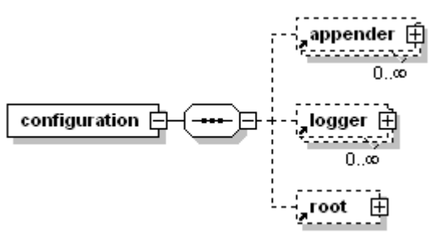
- scan：当此属性设置为true时，配置文件如果发生改变，将会被重新加载，默认值为true.
- scanPeriod：设置监测配置文件是否有修改的时间间隔，如果没有给出时间单位，默认单位是毫秒。当scan为true时，此属性生效。默认的时间间隔为1分钟.
- debug：当此属性设置为true时，将打印出logback内部日志信息，实时查看logback运行状态。默认值为false。

XML代码：

```
<configuration scan="true" scanPeriod="60 second" debug="false">
      <!-- 其他配置省略-->
</configuration>
```

根节点<configuration>的子节点

LogBack的配置大概包括3部分：appender, logger和root。



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：

```
<configuration scan="true" scanPeriod="60 second" debug="false">
      <contextName>myAppName</contextName>
      <!-- 其他配置省略-->
</configuration>
```

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。

```
<configuration scan="true" scanPeriod="60 second" debug="false">
      <property name="APP_Name" value="myAppName" />
      <contextName>${APP_Name}</contextName>
      <!-- 其他配置省略-->
</configuration>
```

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：

```
<configuration scan="true" scanPeriod="60 second" debug="false">
      <timestamp key="bySecond" datePattern="yyyyMMdd'T'HHmmss"/>
      <contextName>${bySecond}</contextName>
```

```
    <!-- 其他配置省略-->
</configuration>
```

设置logger和root

<logger>
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>。<logger>仅有一个name属性，一个可选的level和一个可选的additivity属性。

- name：用来指定受此logger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，还有一个特殊值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logger将会继承上级的级别。
- additivity：是否向上级logger传递打印信息。默认是true。

<logger>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logger。

<root>
也是<logger>元素，但是它是根logger。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logger。

案例介绍

首先，Java类如下：

```java
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

只配置root

```
<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；

输出结果：

```
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
```

带有logger的配置，不指定级别，不指定appender

```
<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。
输出结果：

```
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
```

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

没有设置additivity，默认为true，将此logger的打印信息向上级传递。

没有设置appender，此logger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。

带有多个logger的配置，指定级别，指定appender

```
<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>
```

输出结果：

```
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
```

<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additivity，默认为true，将此logger的打印信息向上级传递。

没有设置appender，此logger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logger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logger指定的名为“STDOUT”的appender处理，在控制台中打出日志，不再向次logger的上级 <logger name=”logback” /> 传递打印信息。

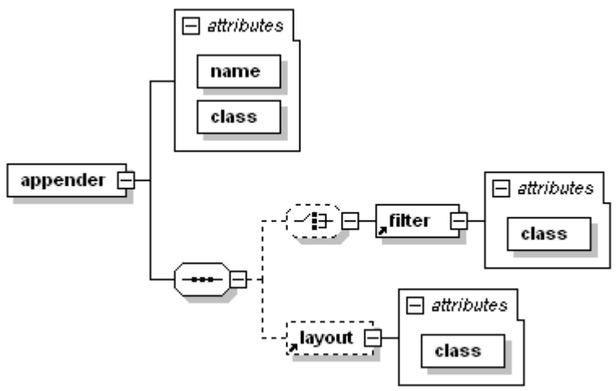
<logger name=”logback” />未接到任何打印信息，当然也不会给它的上级root传递任何打印信息。

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：

```
14:09:01.531 [main] INFO  logback.LogbackDemo - ======info
14:09:01.531 [main] INFO  logback.LogbackDemo - ======info
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
```

<appender>详解

<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。



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 .

```
<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>
```

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。

```
<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>
```

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 何时激活滚动。
- <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，既负责滚动也负责触发滚动。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：每天生产一个日志文件，保存30天的日志文件

```
<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>
```

又例如：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。

```
<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>
```

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：

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

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。

完整配置案例

最后附上相对比较完整的，涵盖大部分配置的案例，案例中有解析。

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
-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。
-
- configuration 子节点为 appender、logger、root
-->
<configuration scan="true" scanPeriod="60 second" debug="false">

    <!-- 负责写日志,控制台日志 -->
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

        <!-- 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 -->
        <encoder>
            <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%5level] [%thread] %logger{0} %msg%n</Pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!-- 文件日志 -->
    <appender name="DEBUG" class="ch.qos.logback.core.FileAppender">
        <file>debug.log</file>
        <!-- append: true,日志被追加到文件结尾; false,清空现存文件;默认是true -->
        <append>true</append>
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <!-- LevelFilter: 级别过滤器，根据日志级别进行过滤 -->
            <level>DEBUG</level>
            <onMatch>ACCEPT</onMatch>
            <onMismatch>DENY</onMismatch>
        </filter>
        <encoder>
            <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%5level] [%thread] %logger{0} %msg%n</Pattern>
            <charset>UTF-8</charset>
        </encoder>
    </appender>

    <!-- 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 -->
    <appender name="INFO" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <File>info.log</File>

        <!-- ThresholdFilter:临界值过滤器，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>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%5level] [%thread] %logger{0} %msg%n</Pattern>
            <charset>UTF-8</charset>
```

```
        </encoder>

        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <!-- 每天生成一个日志文件，保存30天的日志文件
            - 如果隔一段时间没有输出日志，前面过期的日志不会被删除，只有再重新打印日志的时候，会触发删除过期日志的操作。
            -->
            <fileNamePattern>info.%d{yyyy-MM-dd}.log</fileNamePattern>
            <maxHistory>30</maxHistory>
            <TimeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </TimeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
    </appender >

    <!--<!- 异常日志输出 ->-->
    <!--<appender name="EXCEPTION" class="ch.qos.logback.core.rolling.RollingFileAppender">-->
        <!--<file>exception.log</file>-->
        <!--<!- 求值过滤器，评估、鉴别日志是否符合指定条件. 需要额外的两个JAR包，commons-compiler.jar和janino.jar ->-->
        <!--<filter class="ch.qos.logback.core.filter.EvaluatorFilter">-->
            <!--<!- 默认为 ch.qos.logback.classic.boolex.JaninoEventEvaluator ->-->
            <!--<evaluator>-->
                <!--<!- 过滤掉所有日志消息中不包含"Exception"字符串的日志 ->-->
                <!--<expression>return message.contains("Exception");</expression>-->
            <!--</evaluator>-->
            <!--<OnMatch>ACCEPT</OnMatch>-->
            <!--<OnMismatch>DENY</OnMismatch>-->
        <!--</filter>-->

        <!--<triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">-->
            <!--<!- 触发节点，按固定文件大小生成，超过5M，生成新的日志文件 ->-->
            <!--<maxFileSize>5MB</maxFileSize>-->
        <!--</triggeringPolicy>-->
    <!--</appender>-->

    <appender name="ERROR" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>error.log</file>

        <encoder>
            <Pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%5level] [%thread] %logger{0} %msg%n</Pattern>
            <charset>UTF-8</charset>
        </encoder>

        <!-- 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。
        -    可以指定文件压缩选项
        -->
        <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
            <fileNamePattern>error.%d{yyyy-MM}(%i).log.zip</fileNamePattern>
            <minIndex>1</minIndex>
            <maxIndex>3</maxIndex>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>100MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <maxHistory>30</maxHistory>
        </rollingPolicy>
    </appender>

    <!-- 异步输出 -->
    <appender name ="ASYNC" class= "ch.qos.logback.classic.AsyncAppender">
        <!-- 不丢失日志.默认的,如果队列的80%已满,则会丢弃TRACT、DEBUG、INFO级别的日志 -->
        <discardingThreshold >0</discardingThreshold>
        <!-- 更改默认的队列的深度,该值会影响性能.默认值为256 -->
        <queueSize>512</queueSize>
        <!-- 添加附加的appender,最多只能添加一个 -->
        <appender-ref ref ="ERROR"/>
    </appender>

    <!--
    - 1.name：包名或类名，用来指定受此logger约束的某一个包或者具体的某一个类
    - 2.未设置打印级别，所以继承他的上级<root>的日志级别“DEBUG”
    - 3.未设置additivity，默认为true，将此logger的打印信息向上级传递；
    - 4.未设置appender，此logger本身不打印任何信息，级别为“DEBUG”及大于“DEBUG”的日志信息传递给root，
    -  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；
    -->
    <logger name="ch.qos.logback" />

    <!--
    - 1.将级别为“INFO”及大于“INFO”的日志信息交给此logger指定的名为“STDOUT”的appender处理，在控制台中打出日志，
    -   不再向次logger的上级 <logger name="logback"/> 传递打印信息
    - 2.level：设置打印级别（TRACE，DEBUG，INFO，WARN，ERROR，ALL 和 OFF），还有一个特殊值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logger将会继承上级的级别。
    - 3.additivity：为false，表示此logger的打印信息不再向上级传递,如果设置为true，会打印两次
    - 4.appender-ref：指定了名字为"STDOUT"的appender。
    -->
    <logger name="com.weizhi.common.LogMain" level="INFO" additivity="false">
        <appender-ref ref="STDOUT"/>
        <!--<appender-ref ref="DEBUG"/>-->
        <!--<appender-ref ref="EXCEPTION"/>-->
        <!--<appender-ref ref="INFO"/>-->
        <!--<appender-ref ref="ERROR"/>-->
        <appender-ref ref="ASYNC"/>
    </logger>

    <!--
    - 根logger
    - level: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。
    -appender-ref: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logger
    -->
    <root level="DEBUG">
        <appender-ref ref="STDOUT"/>
        <!--<appender-ref ref="DEBUG"/>-->
        <!--<appender-ref ref="EXCEPTION"/>-->
        <!--<appender-ref ref="INFO"/>-->
        <appender-ref ref="ASYNC"/>
    </root>
</configuration>
```

参考资料

1. http://logback.qos.ch/
2. logback 配置详解（一）
3. logback 常用配置详解（二） <appender>
4. Logback配置解析
5. LogBack简易教程

相关文章

- Java日志框架：logback详解
- Java日志框架：slf4j作用及其实现原理
- MySQL中的重做日志（redo log），回滚日志（undo log），以及二进制日志（binlog）的简单总结
- 正确的打日志姿势
- MySQL 死锁与日志二三事
- 异步打印日志的一点事
- Log4j1 升级 Log4j2 实战
- 关于日志记录的一些感想
- 日志打印的5点建议
- Logstash实践: 分布式系统的日志监控

发表评论

Comment form

Name*

姓名

邮箱*

请填写邮箱

网站 (请以 http://开头)

请填写网站地址

评论内容*

请填写评论内容

(*) 表示必填项

提交评论

3 条评论

1. *kamyam* 说道：
   2016/12/01 下午 1:19

   为什么我创建了一个Maven项目只引入了这几个包就启动报错?

   （当想配合Slf4j使用时，需要将logback-classic加入classpath） 可能我这一部没做好？因为删了这个包能正常启动，请问classpath指的是哪里?

   1 0

   回复
2. *paul* 说道：
   2017/07/04 上午 10:08

   addtivity拼写错了，少了i

   0 0

   回复
   - *唐尤华* 说道：
     2017/07/05 上午 7:02

     感谢指正,已更新

     0 0

回复

« 关于Java并发编程的总结和思考
数据库相关中间件全家桶 »

关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的，这是一个很特别的时刻 :)

ImportNew 由两个 Java 关键字 import 和 new 组成，意指：Java 开发者学习新知识的网站。 import 可认为是学习和吸收， new 则可认为是新知识、新技术圈子和新朋友……

-

联系我们

Email：ImportNew.com@gmail.com
新浪微博：@ImportNew
推荐微信号

ImportNew 安卓应用频道 Linux爱好者

反馈建议：ImportNew.com@gmail.com
广告与商务合作QQ：2302462408

推荐关注

小组 – 好的话题、有启发的回复、值得信赖的圈子
头条 – 写了文章？看干货？去头条！
相亲 – 为IT单身男女服务的征婚传播平台
资源 – 优秀的工具资源导航
翻译 – 活跃 & 专业的翻译小组
博客 – 国内外的精选博客文章
设计 – UI,网页，交互和用户体验
前端 – JavaScript, HTML5, CSS
安卓 – 专注Android技术分享
iOS – 专注iOS技术分享
Java – 专注Java技术分享
Python – 专注Python技术分享