

[博客园](#) [首页](#) [博问](#) [闪存](#) [新随笔](#) [订阅](#) [XML](#) [管理](#) [posts - 24, comments - 5, trackbacks - 0](#)

## logback的使用和logback.xml详解

### 一、logback的介绍

Logback是由log4j创始人设计的另一个开源日志组件,官方网站: <http://logback.qos.ch>。它当前分为下面下个模块:

**logback-core:** 其它两个模块的基础模块

**logback-classic:** 它是log4j的一个改良版本,同时它完整实现了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

### 二、logback取代log4j的理由:

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。

3、Logback-classic非常自然实现了SLF4j: Logback-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SLF4J API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、Lilith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

昵称: [行走在云端的愚公](#)

园龄: [2年1个月](#)

粉丝: [5](#)

关注: [0](#)

[+加关注](#)

≤	2017年8月							≥
日	一	二	三	四	五	六		
30	31	1	2	3	4	5		
6	7	8	9	10	11	12		
13	14	15	16	17	18	19		
20	21	22	23	24	25	26		
27	28	29	30	31	1	2		
3	4	5	6	7	8	9		

搜索

<input type="text"/>	<input type="button" value="找找看"/>
<input type="text"/>	<input type="button" value="谷歌搜索"/>

常用链接

[我的随笔](#)

[我的评论](#)

[我的参与](#)

[最新评论](#)

[我的标签](#)

我的标签

[python\(10\)](#)

[git\(5\)](#)

[memcached\(4\)](#)

[分布式缓存\(4\)](#)

[缓存\(4\)](#)

[详解\(1\)](#)

[DI\(1\)](#)

[python function\(1\)](#)

[spring\(1\)](#)

[IOC\(1\)](#)

[更多](#)

**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log: **RollingFileAppender**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maxHistory 12，那那些log文件超过12个月的都会被自动移除。

### 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、PostgreSQL、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

## 随笔分类

[git\(5\)](#)  
[Hadoop](#)  
[Java\(2\)](#)  
[Linux\(1\)](#)  
[Memcached\(4\)](#)  
[python\(11\)](#)  
[spring\(1\)](#)  
[Zookeeper](#)

## 随笔档案

[2017年8月 \(2\)](#)  
[2017年7月 \(13\)](#)  
[2017年5月 \(4\)](#)  
[2017年4月 \(2\)](#)  
[2016年7月 \(2\)](#)  
[2015年7月 \(1\)](#)

## 最新评论

[1. Re:logback的使用和logback.xml详解](#)

挺详细的，赞一个

--youzhibing2904

[2. Re:logback的使用和logback.xml详解](#)

可以的！

--行走在云端的愚公

[3. Re:logback的使用和logback.xml详解](#)

您好，我能转载你这篇文章吗？  
我会在文章开头说明并贴上链接注明出处。

--Gonjian

[4. Re:logback的使用和logback.xml详解](#)

@进击的饭饭是的，多谢提点，已经修改了。...

--行走在云端的愚公

[5. Re:logback的使用和logback.xml详解](#)

5.3、ConsoleAppender: 滚动记录文件

这里写错啦，博主，不过你的博文是我看到的最全的，不错哟

--进击的饭饭

## 阅读排行榜

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  $p \geq q$  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。root logger 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。

## 五、logback.xml常用配置详解

[1. logback的使用和logback.xml详解\(42344\)](#)

[2. 解决不同操作系统下git换行符一致性问题\(916\)](#)

[3. 最近公司用到了lombok，感觉不错的样子，所以上网搜了一些资料，总结了一下用法。\(194\)](#)

[4. Linux包管理器\(23\)](#)

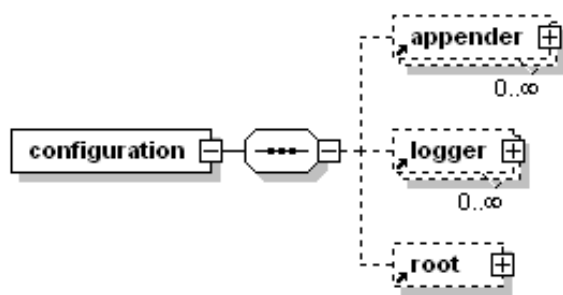
[5. IOC\(21\)](#)

### 评论排行榜

[1. logback的使用和logback.xml详解\(5\)](#)

### 推荐排行榜

[1. logback的使用和logback.xml详解\(3\)](#)



### 1、根节点<configuration>，包含下面三个属性：

**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：

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

2、子节点<contextName>：用来设置上下文名称，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。

例如：

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

3、子节点<property>：用来定义变量值，它有两个属性name和value，通过<property>定义的值会被插入到logger上下文中，可以使"\${}"来使用变量。

**name:** 变量的名称

**value:** 的值时变量定义的值

例如：

```
<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>
```

4、子节点<timestamp>：获取时间戳字符串，他有两个属性key和datePattern

**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:

```
<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>
```

5、子节点<appender>: 负责写日志的组件，它有两个必要属性name和class。name指定appender名称，class指定appender的全限定名

5.1、ConsoleAppender 把日志输出到控制台，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(默认)或者System.err（区别不多说了）

例如:

```
<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>
```

上述配置表示把>=DEBUG级别的日志都输出到控制台

5.2、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：

```
<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>
```

上述配置表示把>=DEBUG级别的日志都输出到testFile.log

**5.3、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**。

**<rollingPolicy>：**当发生滚动时，决定RollingFileAppender的行为，涉及文件移动和重命名。属性class定义具体的滚动策略类

**class="ch.qos.logback.core.rolling.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），活动文件的名称不会改变；

如果没设置file，活动文件的名称会根据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。

**class="ch.qos.logback.core.rolling.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。

**<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。

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

**class="ch.qos.logback.core.rolling.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

例如:

```
<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>

```

上述配置表示每天生成一个日志文件，保存30天的日志文件。

```

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

```

上述配置表示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。

**<encoder>**：对记录事件进行格式化。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。

**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。

5.4、还有SocketAppender、SMTPAppender、DBAppender、SyslogAppender、SiftingAppender，并不常用，这里就不详解了。大家可以参考官方文档



(<http://logback.qos.ch/documentation.html>)，还可以编写自己的Appender。

6、子节点<logger>：用来设置某一个包或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。

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

**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。同<logger>一样，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。

7、子节点<root>：它也是<logger>元素，但是它是根logger,是所有<logger>的上级。只有一个level属性，因为name已经被命名为"root",且已经是最上级了。

**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。

## 六、常用logger配置

```
<!-- show parameters for hibernate sql 专为 Hibernate 定制 -->
<logger name="org.hibernate.type.descriptor.sql.BasicBinder"
level="TRACE" />
<logger
name="org.hibernate.type.descriptor.sql.BasicExtractor"
level="DEBUG" />
<logger name="org.hibernate.SQL" level="DEBUG" />
<logger name="org.hibernate.engine.QueryParameters"
level="DEBUG" />
<logger name="org.hibernate.engine.query.HQLQueryPlan"
level="DEBUG" />

<!--myibatis log configure-->
<logger name="com.apache.ibatis" level="TRACE"/>
<logger name="java.sql.Connection" level="DEBUG"/>
<logger name="java.sql.Statement" level="DEBUG"/>
<logger name="java.sql.PreparedStatement" level="DEBUG"/>
```

## 七、Demo

1、添加依赖包logback使用需要和slf4j一起使用，所以总共需要添加依赖的包有slf4j-api

logback使用需要和slf4j一起使用，所以总共需要添加依赖的包有slf4j-api.jar, logback-core.jar, logback-

classic.jar, logback-access.jar这个暂时用不到所以不添加依赖了, maven配置

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <logback.version>1.1.7</logback.version>
  <slf4j.version>1.7.21</slf4j.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>${slf4j.version}</version>
    <scope>compile</scope>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>${logback.version}</version>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${logback.version}</version>
  </dependency>
</dependencies>
```

## 2、logback.xml配置

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false">
  <!--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="/home" />
  <!-- 控制台输出 -->
  <appender name="STDOUT"
    class="ch.qos.logback.core.ConsoleAppender">
    <encoder
      class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <!--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>
      </encoder>
    </appender>
  <!-- 按照每天生成日志文件 -->
  <appender name="FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy
      class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!--日志文件输出的文件名-->
        <FileNamePattern>${LOG_HOME}/TestWeb.log.%d{yyyy-MM-dd}.log</FileNamePattern>
```

```
<!--日志文件保留天数-->
<MaxHistory>30</MaxHistory>
</rollingPolicy>
<encoder
class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
<!--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>
</encoder>
<!--日志文件最大的大小-->
<triggeringPolicy
class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
<MaxFileSize>10MB</MaxFileSize>
</triggeringPolicy>
</appender>

<!-- 日志输出级别 -->
<root level="INFO">
<appender-ref ref="STDOUT" />
</root>
</configuration>
```

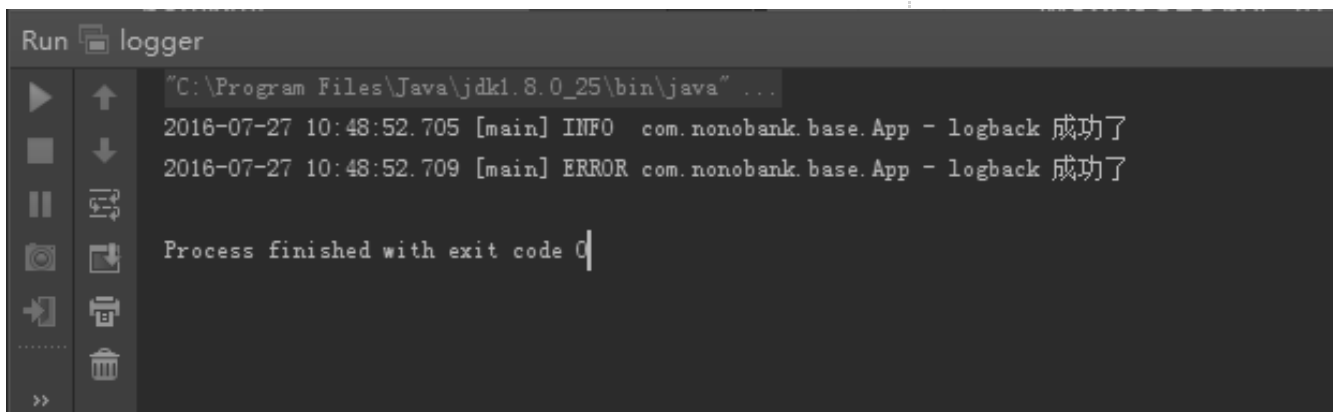
### 3、java代码

```
/**
 * Hello world!
 */
public class App {

    private final static Logger logger =
LoggerFactory.getLogger(App.class);

    public static void main(String[] args) {
        logger.info("logback 成功了");
        logger.error("logback 成功了");
        logger.debug("logback 成功了");
    }
}
```

### 4、输出

A screenshot of a Java IDE's console window. The title bar says "Run" and "logger". The console shows the command "C:\Program Files\Java\jdk1.8.0\_25\bin\java" followed by two log messages: "2016-07-27 10:48:52.705 [main] INFO com.nonobank.base.App - logback 成功了" and "2016-07-27 10:48:52.709 [main] ERROR com.nonobank.base.App - logback 成功了". At the bottom, it says "Process finished with exit code 0". The console has a dark background with light text and a vertical toolbar on the left with icons for running, stepping, and other debugging actions.

```
Run logger
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ...
2016-07-27 10:48:52.705 [main] INFO com.nonobank.base.App - logback 成功了
2016-07-27 10:48:52.709 [main] ERROR com.nonobank.base.App - logback 成功了
Process finished with exit code 0
```

## 八、总结

logback的配置，需要配置输出源appender，打日志的logger（子节点）和root（根节点），实际上，它输出日志是从子节点开始，子节点如果有输出源直接输入，如果无，判断配置的additivity，是否像上级传递，即是否向root传递，传递则采用root的输出源，否则不输出日志。

分类: [Java](#)

标签: [logback](#), [logback.xml](#), [详解](#)

好文要顶

关注我

收藏该文



行走在云端的愚公

关注 - 0

粉丝 - 5

[+加关注](#)

3

0

« 上一篇: [最近公司用到了lombok，感觉不错的样子，所以上网搜了一些资料，总结了一下用法。](#)

» 下一篇: [解决不同操作系统下git换行符一致性问题](#)

posted on 2016-07-27 11:05 [行走在云端的愚公](#) 阅读(42353) 评论(5) [编辑](#) [收藏](#)

## FeedBack:

### #1楼

2016-09-21 11:08 | [进击的饭饭](#)

#### 5.3、ConsoleAppender: 滚动记录文件

这里写错啦，博主，不过你的博文是我看到的最全的，不错哟

支持(0) 反对(0)

### #2楼[楼主]

2016-09-21 11:14 | [行走在云端的愚公](#)

@ [进击的饭饭](#)

是的，多谢提点，已经修改了。

支持(0) 反对(0)

### #3楼

2017-03-12 21:52 | [Gonjian](#)

您好，我能转载你这篇文章吗？我会在文章开头说明并贴上链接注明出处。

支持(0) 反对(0)

#4楼[楼主]

2017-03-12 21:52 | [行走在云端的愚公](#)

可以的！

支持(0) 反对(0)

#5楼

2017-05-13 14:51 | [youzhibing2904](#)

挺详细的，赞一个

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#)网站首页。

【推荐】50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

【推荐】极光开发者服务平台，五大功能一站集齐

【推荐】腾讯云域名+云解析 限时折扣抓紧抢购

【推荐】阿里云“全民云计算”优惠升级

【推荐】一小时搭建人工智能应用，让技术更容易入门



阿里云 | 奥运会全球指定云服务商

建网站 搭应用  
首选阿里云服务器

Intel Xeon E5V4 性能更优  
多节点部署全球

0.73元/日起

最新IT新闻：

- [Google地图想要帮助用户在25个新城市找到停车位](#)
  - [NASA将在明年上半年发射InSight火星探测器](#)
  - [阿里即将开源ApsaraCache，云数据库Redis版分支](#)
  - [知乎回应被今日头条挖走300大V：不会用肤浅手段拔苗助长](#)
  - [Google搜索上线Flight Insights：为你的出游省钱](#)
- » [更多新闻...](#)



JIGUANG | 极光

移动开发首选 极光

推送 IM 短信 统计 分享

最新知识库文章：

- [做到这一点，你也可以成为优秀的程序员](#)
- [写给立志做码农的大学生](#)
- [架构腐化之谜](#)
- [学会思考，而不只是编程](#)
- [编写Shell脚本的最佳实践](#)
- » [更多知识库文章...](#)

Copyright ©2017 行走在云端的愚公 Powered By[博客园](#) 模板提供: [沪江博客](#)