**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：

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

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

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>

**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：

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>

**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

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. }

logback.xml配置文件

**第1种：只配置root**

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. <root level="INFO">
11. <appender-ref ref="STDOUT" />
12. </root>
14. </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：**

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

**第2种：带有loger的配置，不指定级别，不指定appender，**

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>

 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：**

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

**第3种：带有多个loger的配置，指定级别，指定appender**

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>
9. <!-- logback为java中的包 -->
10. <logger name="logback"/>
11. <!--logback.LogbackDemo：类的全路径 -->
12. <logger name="logback.LogbackDemo" level="INFO" additivity="false">
13. <appender-ref ref="STDOUT"/>
14. </logger>
16. <root level="ERROR">
17. <appender-ref ref="STDOUT" />
18. </root>
19. </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="ERROR">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：**

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

 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：**

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