

## 原创 Spring Boot 日志配置(超详细)

2017-07-12 11:37:13 Inke 阅读数 149669 更多

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。  
本文链接：<https://blog.csdn.net/Inke88/article/details/75007649>

## Spring Boot-日志配置(超详细)

更新日志：

20170810 更新通过 application.yml传递参数到 logback 中。

简书不支持目录，截图一张。

### • Spring Boot 日志配置

- 默认日志 Logback :
- 添加日志依赖
- 控制台输出
- 文件输出
- 级别控制
- 自定义日志配置
  - 根节点包含的属性
    - 子节点一 <root>
    - 子节点二: <contextName> 设置上下文名称
    - 子节点三: <property> 设置变量
    - 子节点四: <appender>
      - 控制台输出 ConsoleAppender :
      - 输出到文件 RollingFileAppender
    - 子节点五 <logger>
      - 第一种: 带有 logger 的配置, 不指定级别, 不指定 appender
      - 第二种: 带有多个 logger 的配置, 指定级别, 指定 appender
    - 多环境日志输出

##默认日志 Logback :

默认情况下, Spring Boot会用Logback来记录日志, 并用INFO级别输出到控制台。在运行应用程序和其他例子时, 你应该已经看到很多INFO级别的日志输出。

85

<

21

Q

Q

<

>

赏

脉

《Python从小白到大牛》  
图书视频源代码讲师专属答疑群  
关闭



举报

```
C:\Program Files\Java\jdk1.8.0_101\bin\java" ...

:: Spring Boot :: (v1.5.2.RELEASE)

2017-04-01 17:19:38.780 INFO 13692 --- [main] com.dudu.Application : Starting Application on DESKTOP-FG8EEPC with PID 13692 (E:\
2017-04-01 17:19:38.784 INFO 13692 --- [main] com.dudu.Application : No active profile set, falling back to default profiles: de
2017-04-01 17:19:39.145 INFO 13692 --- [main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext
2017-04-01 17:19:41.123 INFO 13692 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 8080 (http)
2017-04-01 17:19:41.143 INFO 13692 --- [main] o.apache.catalina.core.StandardService : Starting service Tomcat
2017-04-01 17:19:41.144 INFO 13692 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.11
2017-04-01 17:19:41.315 INFO 13692 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2017-04-01 17:19:41.315 INFO 13692 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 217
2017-04-01 17:19:41.798 INFO 13692 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/]
2017-04-01 17:19:41.811 INFO 13692 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [/]
2017-04-01 17:19:41.812 INFO 13692 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/]
2017-04-01 17:19:41.813 INFO 13692 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'httpPutFormContentFilter' to: [/]
2017-04-01 17:19:41.813 INFO 13692 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'requestContextFilter' to: [/]
```

从上图可以看到，日志输出内容元素具体如下：

- 时间日期：精确到毫秒
- 日志级别：ERROR, WARN, INFO, DEBUG or TRACE
- 进程ID
- 分隔符：— 标识实际日志的开始
- 线程名：方括号括起来（可能会截断控制台输出）
- Logger名：通常使用源代码的类名
- 日志内容

##添加日志依赖

假如maven依赖中添加了 `spring-boot-starter-logging`：

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-logging</artifactId>
4 </dependency>
```

但是呢，实际开发中我们不需要直接添加该依赖。

你会发现 `spring-boot-starter` 其中包含了 `spring-boot-starter-logging`，该依赖内容就是 `Spring Boot` 默认的日志框架 `logback`。工程中有 `Thymeleaf`，而 `Thymeleaf` 依赖包含了 `spring-boot-starter`，最终我只要引入 `Thymeleaf` 即可。

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-thymeleaf</artifactId>
4 </dependency>
```

##控制台输出

日志级别从低到高分为：

TRACE < DEBUG < INFO < WARN < ERROR < FATAL。

如果设置为 `WARN`，则低于 `WARN` 的信息都不会输出。

`Spring Boot` 中默认配置 `ERROR`、`WARN` 和 `INFO` 级别的日志输出到控制台。

您还可以通过启动您的应用程序 `--debug` 标志来启用“调试”模式（开发的时候推荐开启），以下两种方式皆可：

- 在运行命令后加入 `--debug` 标志，如：`$ java -jar springTest.jar --debug`
- 在 `application.properties` 中配置 `debug=true`，该属性置为 `true` 的时候，核心 `Logger`（包含嵌入式容器、hibernate、spring）会输出自己应用的日志并不会输出为 `DEBUG` 级别。

```
14 @RunWith(SpringRunner.class)
15 @SpringBootTest
16 public class LoggerTest {
17
18     private final Logger logger = LoggerFactory.getLogger(LoggerTest2);
19
20     @Test
21     public void test1() {
22         logger.debug("debug...");
23         logger.info("info...");
24         logger.error("error...");
25     }
26 }
```

日志的类名必须是当前类，  
如果不是当前类，那么输出日志  
的类名也是错的

1 test passed - 246ms

17:55:45.837 INFO 31544 --- [main] com.imooc.LoggerTest2  
17:55:45.839 ERROR 31544 --- [main] com.imooc.LoggerTest2

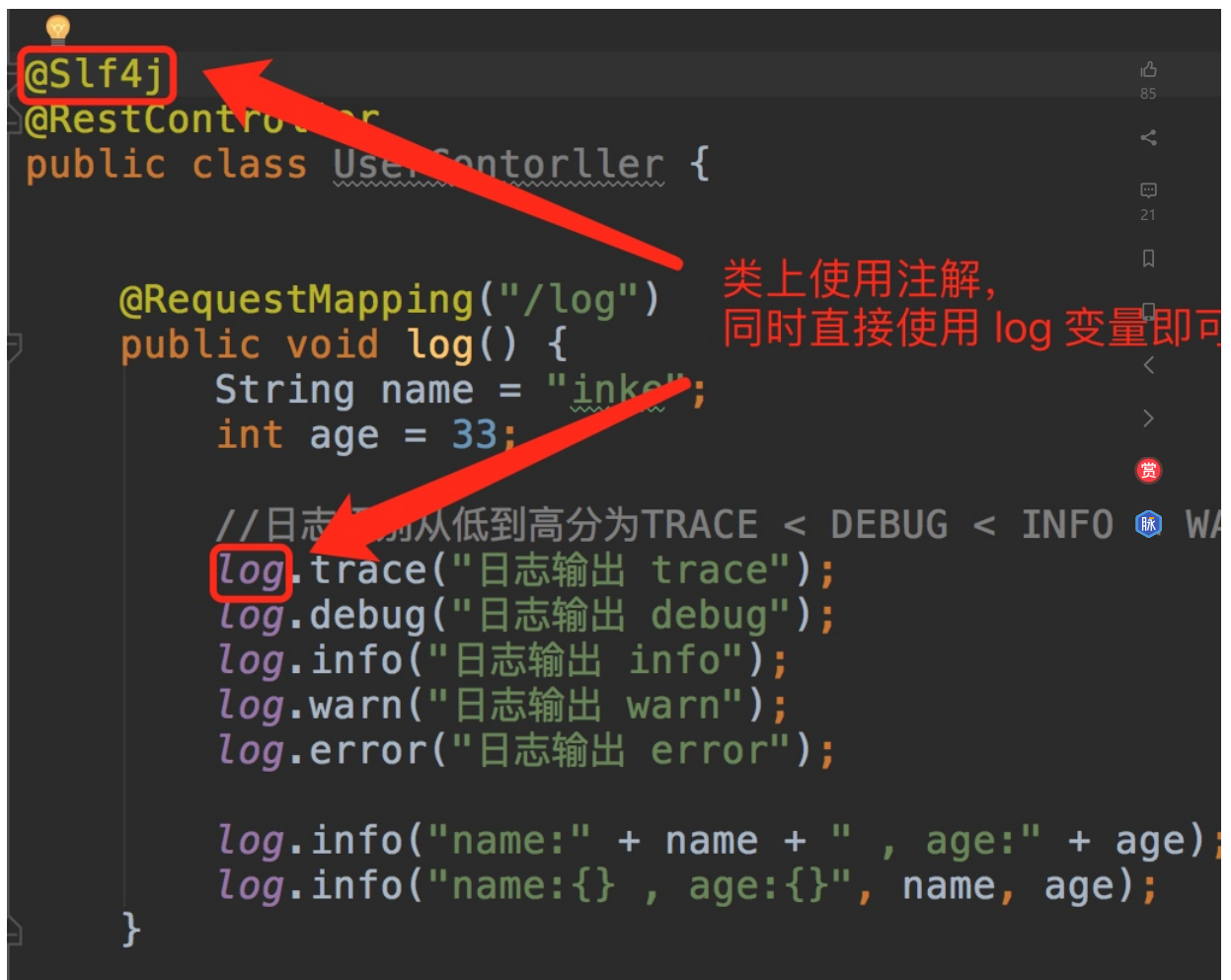
如果每次都写这行代码会很麻烦，可以使用注解，但是需要使用 `Lombok`：

• 添加依赖：

```
1 //注解
2 compile 'org.projectlombok:lombok:1.16.18'
```

• 安装 `Lombok` 的插件：

- Go to File > Settings > Plugins
- Click on Browse repositories...
- Search for Lombok Plugin
- Click on Install plugin
- Restart Android Studio
- 允许注解处理，Settings -> Compiler -> Annotation Processors

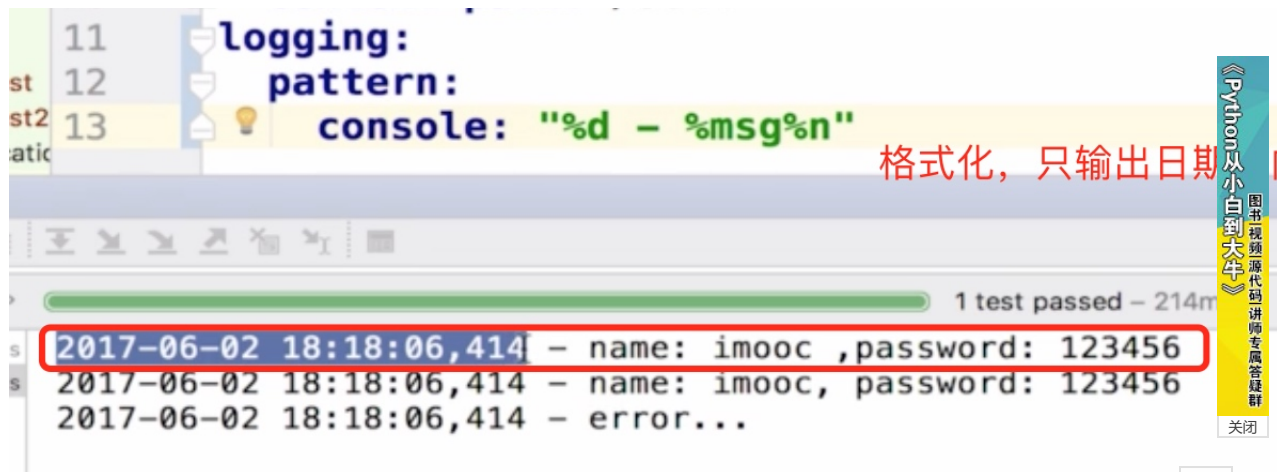


可以使用 {} 占位符来拼接字符串，而不需要使用 “+” 来连接字符串。

##文件输出

默认情况下，Spring Boot将日志输出到控制台，不会写到日志文件。

使用 Spring Boot 喜欢在 application.properties 或 application.yml 配置，这样只能配置简单的场景，保存路径、日志格式等，复杂的场景（区分 error 的日志、每天产生一个日志文件等）满足不了，只能自定义配置，下面会演示。



```
logging:
  pattern:
    console: "%d - %msg%n"
    path: /var/log/tomcat/
```

日志输出路径

```
→ tomcat cd /var/log/tomcat/
→ tomcat
→ tomcat
→ tomcat ll
total 0
drwxrwxrwx    2 root    wheel    68    6    2  18:21  .
drwxr-xr-x   61 root    wheel  2074    6    2  10:46  ..
→ tomcat ll
total 16
drwxrwxrwx    3 root    wheel   102    6    2  18:21  .
drwxr-xr-x   61 root    wheel  2074    6    2  10:46  ..
-rw-r--r--    1 admin   wheel  4489    6    2  18:21  spring.log
```

默认会在设置的 `path` 生成一个 `spring.log` 文件。

85

<

21

Q

Q

<

>

赏

脉

《Python从小白到大牛》  
图书视频源代码讲师专属答疑群

关闭

Q

举报



```
logging:
  pattern:
    console: "%d - %msg%n"
  path: /var/log/tomcat/
  file: /var/log/tomcat/sell
```

指定文件名

```
tomcat cd /var/log/tomcat/
tomcat
tomcat
tomcat ll
tal 0
wxrwxrwx    2 root    wheel      68    6    2  18:21  .
wxr-xr-x   61 root    wheel    2074    6    2  10:46  ..
tomcat ll
tal 16
wxrwxrwx    3 root    wheel     102    6    2  18:21  .
wxr-xr-x   61 root    wheel    2074    6    2  10:46  ..
w-r--r--    1 admin   wheel    4489    6    2  18:21  spring.l
tomcat vim spring.log

tomcat
tomcat
tomcat
tomcat ll
tal 32
wxrwxrwx    4 root    wheel     136    6    2  18:23  .
wxr-xr-x   61 root    wheel    2074    6    2  10:46  ..
w-r--r--    1 admin   wheel    4489    6    2  18:23  sell
w-r--r--    1 admin   wheel    4489    6    2  18:21  spring.l
```

如果要编写除控制台输出之外的日志文件，则需在 `application.properties` 中设置 `logging.file` 或 `logging.path` 属性。

- `logging.file`，设置文件，可以是绝对路径，也可以是相对路径。如：`logging.file=my.log`
- `logging.path`，设置目录，会在该目录下创建 `spring.log` 文件，并写入日志内容，如：`logging.path=/var/log`  
如果只配置 `logging.file`，会在项目的当前路径下生成一个 `xxx.log` 日志文件。  
如果只配置 `logging.path`，在 `/var/log` 文件夹生成一个日志文件为 `spring.log`

注：二者不能同时使用，如若同时使用，则只有 `logging.file` 生效

默认情况下，日志文件的大小达到 `10MB` 时会切分一次，产生新的日志文件，默认级别为：`ERROR`、`WARN`、`INFO`

##级别控制

85

<

21

Q

Q

<

>

赏

脉

《Python从小白到大牛》  
图书视频源代码讲师专属答疑群

关闭

Q

举报

所有支持的日志记录系统都可以在 Spring 环境中设置记录级别（例如在 `application.properties` 中）

格式为: `'logging.level.* = LEVEL'`

`logging.level`: 日志级别控制前缀, \*为包名或Logger名

`LEVEL`: 选项 `TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF`

举例:

```
1 | logging.level.com.dudu=DEBUG: com.dudu包下所有class以DEBUG级别输出
2 | logging.level.root=WARN: root日志以WARN级别输出
```



##自定义日志配置

根据不同的日志系统，你可以按如下规则组织配置文件名，就能被正确加载：

- Logback: `logback-spring.xml`, `logback-spring.groovy`, `logback.xml`, `logback.groovy`
- Log4j: `log4j-spring.properties`, `log4j-spring.xml`, `log4j.properties`, `log4j.xml`
- Log4j2: `log4j2-spring.xml`, `log4j2.xml`
- JDK (Java Util Logging): `logging.properties`

Spring Boot 官方推荐优先使用带有 `-spring` 的文件名作为你的日志配置（如使用 `logback-spring.xml`，而不是 `logback.xml`），命名为 `logback-` 的日志配置文件，`spring boot` 可以为它添加一些 `spring boot` 特有的配置项（下面会提到）。

默认的命名规则，并且放在 `src/main/resources` 下面即可

如果你即想完全掌控日志配置，但又不想用 `logback.xml` 作为 Logback 配置的名字，`application.yml` 可以通过 `logging.config` 属性指定自己

```
1 | logging.config=classpath:logging-config.xml
```

虽然一般并不需要改变配置文件的名字，但是如果你想针对不同运行时 `Profile` 使用不同的日志配置，这个功能会很有用。

一般不需要这个属性，而是直接在 `logback-spring.xml` 中使用 `springProfile` 配置，不需要 `logging.config` 指定不同环境使用不同配置文件。配置在下面介绍。

###根节点包含的属性

- 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 30 秒。
- 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`。

根节点 `<configuration>` 有5个子节点，下面——会详细介绍。

85

85

21

85

85

85

85

85

85

《Python从小白到大牛》  
图书视频源代码讲师专属答疑群  
关闭

默认的时间间隔为 30 秒。  
举报

### ####子节点一：<root>

root节点是必选节点，用来指定最基础的日志输出级别，只有一个level属性。

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。

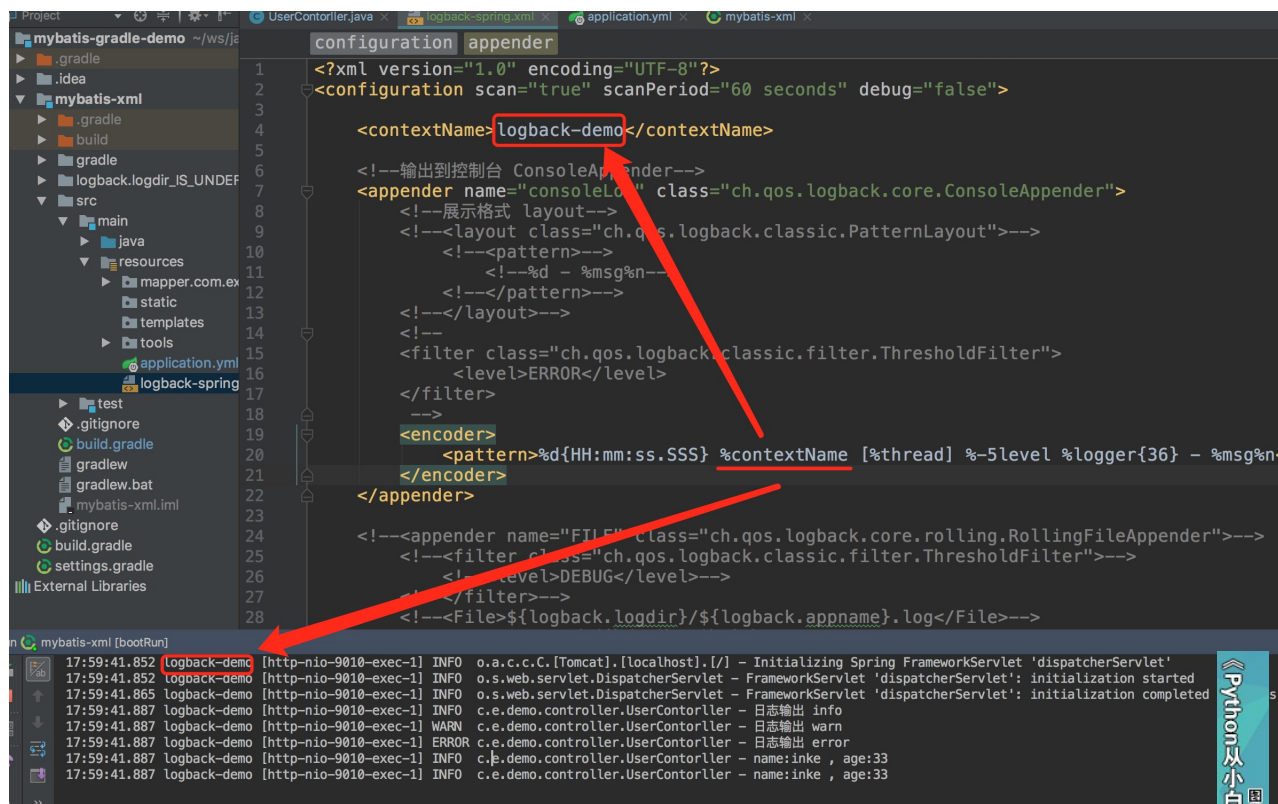
可以包含零个或多个元素，标识这个appender将会添加到这个logger。

```
1 <root level="debug">
2   <appender-ref ref="console" />
3   <appender-ref ref="file" />
4 </root>
```

### ####子节点二：<contextName> 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”。但可以使用设置成其他名字，用于区分不同应用程序的记录。一旦设置就不能通过%contextName来打印日志上下文名称，一般来说我们不用这个属性，可有可无。

```
1 <contextName>logback</contextName>
```



### ####子节点三：<property> 设置变量

用来定义变量值的标签，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。通过定义的值会被插入到log变量后，可以使“\${}”来使用变量。

```
1 <property name="logback.logdir" value="/Users/inke/dev/log/tomcat"/>
2 <property name="logback.appname" value="app"/>
```

这里可以看后通过 application.yml 传递参数过来。



####子节点四: <appender>

appender用来格式化日志输出节点, 有两个属性name和class, class用来指定哪种输出策略, 常用就是控制台输出策略和文件输出策略。

####控制台输出 ConsoleAppender :

• 示例一:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration scan="true" scanPeriod="60 seconds" debug="false">
3
4     <contextName>logback-demo</contextName>
5
6     <!--输出到控制台 ConsoleAppender-->
7     <appender name="consoleLog1" class="ch.qos.logback.core.ConsoleAppender">
8         <!--展示格式 layout-->
9         <layout class="ch.qos.logback.classic.PatternLayout">
10             <pattern>%d -1 %msg%n</pattern>
11         </layout>
12     </appender>
13
14     <!--输出到控制台 ConsoleAppender-->
15     <appender name="consoleLog2" class="ch.qos.logback.core.ConsoleAppender">
16         <encoder>
17             <pattern>%d -2 %msg%n</pattern>
18         </encoder>
19     </appender>
20
21     <!--指定最基础的日志输出级别-->
22     <root level="INFO">
23         <!--appender将会添加到这个logger-->
24         <appender-ref ref="consoleLog1"/>
25         <appender-ref ref="consoleLog2"/>
26     </root>
27
28 </configuration>
```

```
2017-07-11 18:10:14,168 -1 日志输出 info
2017-07-11 18:10:14,168 -2 日志输出 info
2017-07-11 18:10:14,168 -1 日志输出 warn
2017-07-11 18:10:14,168 -2 日志输出 warn
2017-07-11 18:10:14,168 -1 日志输出 error
2017-07-11 18:10:14,168 -2 日志输出 error
2017-07-11 18:10:14,168 -1 name:inke , age:33
2017-07-11 18:10:14,168 -2 name:inke , age:33
2017-07-11 18:10:14,168 -1 name:inke , age:33
2017-07-11 18:10:14,168 -2 name:inke , age:33
```

可以看到 layout 和 encoder, 都可以将事件转换为格式化后的日志记录, 但是控制台输出使用 layout, 文件输出使用 encoder, 具体原理见[http://blog.csdn.net/cw\\_hello1/article/details/51969554](http://blog.csdn.net/cw_hello1/article/details/51969554)

• 示例二:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration scan="true" scanPeriod="60 seconds" debug="false">

    <contextName>logback-demo</contextName>

    <!--输出到控制台 ConsoleAppender-->
    <appender name="consoleLog1" class="ch.qos.logback.core.ConsoleAppender">
        <!--展示格式 layout-->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>
```

85

85

21

85

85

85

85

85

85

Python从小白到大牛  
图书视频源代码讲师专属答疑群  
关闭

举报

```
        <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pattern>
    </pattern>
</layout>
<!--
<filter class="ch.qos.logback.classic.filter.ThresholdFilter">
    <level>ERROR</level>
</filter>
-->
</appender>

<!--指定最基础的日志输出级别-->
<root level="INFO">
    <!--appender将会添加到这个logger-->
    <appender-ref ref="consoleLog1"/>
    <appender-ref ref="consoleLog2"/>
</root>
</configuration>
```

输出日志:

```
1 | 18:15:22.148 logback-demo [http-nio-9010-exec-1] INFO c.e.demo.controller.UserContorller - 日志输出 info
2 | 18:15:22.148 logback-demo [http-nio-9010-exec-1] WARN c.e.demo.controller.UserContorller - 日志输出 warn
3 | 18:15:22.148 logback-demo [http-nio-9010-exec-1] ERROR c.e.demo.controller.UserContorller - 日志输出 error
4 | 18:15:22.148 logback-demo [http-nio-9010-exec-1] INFO c.e.demo.controller.UserContorller - name:inke , age:33
5 | 18:15:22.149 logback-demo [http-nio-9010-exec-1] INFO c.e.demo.controller.UserContorller - name:inke , age:33
```

<encoder> 表示对日志进行编码:

- %d{HH: mm:ss.SSS}——日志输出时间
- %thread——输出日志的进程名字，这在Web应用以及异步任务处理中很有用
- %-5level——日志级别，并且使用5个字符靠左对齐
- %logger{36}——日志输出者的名字
- %msg——日志消息
- %n——平台的换行符

ThresholdFilter为系统定义的拦截器，例如我们用ThresholdFilter来过滤掉ERROR级别以下的日志不输出到文件中。如果不用记得注释掉，不会发现没日志~

####输出到文件 RollingFileAppender

另一种常见的日志输出到文件，随着应用的运行时间越来越长，日志也会增长的越来越多，将他们输出到同一个文件并非一个好办法。RollingFileAppender切分文件日志:

```
<appender name="fileInfoLog" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <!--如果只是想要 Info 级别的日志，只是过滤 info 还是会输出 Error 日志，因为 Error 的级别高，
    所以我们使用下面的策略，可以避免输出 Error 的日志-->
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
        <!--过滤 Error-->
        <level>ERROR</level>
        <!--匹配到就禁止-->
        <onMatch>DENY</onMatch>
        <!--没有匹配到就允许-->
        <onMismatch>ACCEPT</onMismatch>
    </filter>
    <!--日志名称，如果没有File 属性，那么只会使用FileNamePattern的文件路径规则
    如果同时有<File>和<FileNamePattern>，那么当天日志是<File>，明天会自动把今天的
    的日志改名为今天的日期。即，<File> 的日志都是当天的。
    -->
    <File>${logback.logdir}/info.${logback.appname}.log</File>
    <!--滚动策略，按照时间滚动 TimeBasedRollingPolicy-->
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!--文件路径，定义了日志的切分方式—把每一天的日志归档到一个文件中,以防止日志填满整个磁盘空间-->
        <FileNamePattern>${logback.logdir}/info.${logback.appname}.%d{yyyy-MM-dd}.log</FileNamePattern>
        <!--只保留最近90天的日志-->
        <maxHistory>90</maxHistory>
        <!--用来指定日志文件的上限大小，那么到了这个值，就会删除旧的日志-->
        <!--<totalSizeCap>1GB</totalSizeCap>-->
    </rollingPolicy>
```

👍  
85

🔗

💬  
21

🔖

📱

<

>

👑

👤

《Python从小白到大牛》  
图书视频源代码讲师专属答疑群  
关闭

🔄

举报




```

<!-- 日志输出编码格式化-->
<encoder>
    <charset>UTF-8</charset>
    <pattern>%d [%thread] %-5level %logger{36} %line - %msg%n</pattern>
</encoder>
</appender>

<appender name="fileErrorLog" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <!-- 如果只是想要 Error 级别的日志，那么需要过滤一下，默认是 info 级别的，ThresholdFilter-->
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>Error</level>
    </filter>
    <!-- 日志名称，如果没有File 属性，那么只会使用FileNamePattern的文件路径规则
        如果同时有<File>和<FileNamePattern>，那么当天日志是<File>，明天会自动把今天
        的日志改名为今天的日期。即，<File> 的日志都是当天的。
    -->
    <File>${logback.logdir}/error.${logback.appname}.log</File>
    <!-- 滚动策略，按照时间滚动 TimeBasedRollingPolicy-->
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <!-- 文件路径，定义了日志的切分方式——把每一天的日志归档到一个文件中，以防止日志填满整个磁盘空间-->
        <FileNamePattern>${logback.logdir}/error.${logback.appname}.%d{yyyy-MM-dd}.log</FileNamePattern>
        <!-- 只保留最近90天的日志-->
        <maxHistory>90</maxHistory>
        <!-- 用来指定日志文件的上限大小，那么到了这个值，就会删除旧的日志-->
        <!--<totalSizeCap>1GB</totalSizeCap>-->
    </rollingPolicy>
    <!-- 日志输出编码格式化-->
    <encoder>
        <charset>UTF-8</charset>
        <pattern>%d [%thread] %-5level %logger{36} %line - %msg%n</pattern>
    </encoder>
</appender>

```

如果同时有 `<File>` 和 `<FileNamePattern>`，根据日期分割日志，代码注释写的很清楚了。

	error.app.log	今天 上午10:30
	info.app.2017-07-11.log	昨天 下午7:27
	info.app.log	今天 上午10:30

如果要区分 `Info` 和 `Error` 级别的日志，那么需要使用过滤规则的策略，代码注释写的很清楚了。

### ####子节点五 <logger>

`<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 `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，还有一个特俗值INHERITED或者同义词N/A表示继承上级的级别。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在实际使用的时候有两种情况

先来看一看代码中如何使用

```

package com.dudu.controller;
@Controller
public class LearnController {
    private Logger logger = LoggerFactory.getLogger(this.getClass());

    @RequestMapping(value = "/login",method = RequestMethod.POST)
    @ResponseBody
    public Map<String,Object> login(HttpServletRequest request, HttpServletResponse response){
        //日志级别从低到高分TRACE < DEBUG < INFO < WARN < ERROR < FATAL，如果设置为WARN，则低于WARN的信息都不会输出。

```

85

85

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

21

```

        logger.trace("日志输出 trace");
        logger.debug("日志输出 debug");
        logger.info("日志输出 info");
        logger.warn("日志输出 warn");
        logger.error("日志输出 error");
        Map<String,Object> map =new HashMap<String,Object>();
        String userName=request.getParameter("userName");
        String password=request.getParameter("password");
        if(!userName.equals("") && password!=""){
            User user =new User(userName,password);
            request.getSession().setAttribute("user",user);
            map.put("result","1");
        }else{
            map.put("result","0");
        }
        return map;
    }
}

```

85



21



这是一个登录的判断的方法，我们引入日志，并且打印不同级别的日志，然后根据logback-spring.xml中的配置来看看打印了哪几种级别日志。

####第一种：带有 logger 的配置，不指定级别，不指定 appender

logback-spring.xml 增加 logger 配置如下：

```

1 | <logger name="com.dudu.controller"/>

```

<logger name="com.dudu.controller" /> 将控制 controller 包下的所有类的日志的打印，但是并没有设置打印级别，所以继承他的上级的日志级别；  
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="info"> 将 root 的打印级别设置为 “info”，指定了名字为 “console” 的 appender。

当执行 com.dudu.controller.LearnController 类的 login 方法时，LearnController 在包com.dudu.controller中，所以首先执行 <logger name="com.dudu.controller"/>，将级别为 “info” 及大于 “info” 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 “console” 的 appender 处理，“console” appender 将信息打印到控制台；

打印结果如下：

```

1 | 16:00:17.407 logback [http-nio-8080-exec-8] INFO com.dudu.controller.LearnController - 日志输出 info
2 | 16:00:17.408 logback [http-nio-8080-exec-8] WARN com.dudu.controller.LearnController - 日志输出 warn
3 | 16:00:17.408 logback [http-nio-8080-exec-8] ERROR com.dudu.controller.LearnController -

```

####第二种：带有多个 logger 的配置，指定级别，指定 appender

logback-spring.xml 增加 logger 配置如下：

```

1 | <configuration>
2 |     ...
3 |
4 |     <!--logback.LogbackDemo: 类的全路径 -->
5 |     <logger name="com.dudu.controller.LearnController" level="WARN" additivity="false">
6 |         <appender-ref ref="console"/>
7 |     </logger>
8 | </configuration>

```

控制 com.dudu.controller.LearnController 类的日志打印，打印级别为 “WARN” ；

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 “console” 的 appender；

这时候执行 com.dudu.controller.LearnController 类的login方法时，先执行 <logger name="com.dudu.controller.LearnController" level="WARN" additivity="false">，将级别为 “WARN” 及大于 “WARN” 的日志信息交给此 logger 指定的名为 “console” 的 appender 处理，在控制台中打出日志上级 root 传递打印信息。

打印结果如下：

```

16:00:17.408 logback [http-nio-8080-exec-8] WARN com.dudu.controller.LearnController - 日志输出 warn
16:00:17.408 logback [http-nio-8080-exec-8] ERROR com.dudu.controller.LearnController - 日志输出 error

```



当然如果你把 `additivity="false"` 改成 `additivity="true"` 的话，就会打印两次，因为打印信息向上级传递，logger本身打印一次，root接到后又打

注意：

```
1 <configuration>
2   ...
3
4   <logger name="com.example.demo.controller" level="WARN" additivity="false">
5     <appender-ref ref="consoleLog"/>
6   </logger>
7
8   <logger name="com.example.demo.controller"/>
9
10  <logger name="com.example.demo"/>
11 </configuration>
```

范围有重叠的话，范围小的，有效。

## 多环境日志输出

```
1 <configuration>
2   ...
3
4   <!-- 测试环境+开发环境，多个使用逗号隔开，-->
5   <springProfile name="test,dev">
6     <logger name="com.example.demo.controller" level="DEBUG" additivity="false">
7       <appender-ref ref="consoleLog"/>
8     </logger>
9   </springProfile>
10
11  <!-- 生产环境，-->
12  <springProfile name="prod">
13    <logger name="com.example.demo.controller" level="INFO" additivity="false">
14      <appender-ref ref="consoleLog"/>
15    </logger>
16  </springProfile>
17 </configuration>
```

`application.yml` 增加环境选择的配置 `active: dev`

```
1 server:
2   port: 9010
3
4 spring:
5   profiles:
6     active: dev
7   datasource:
8     url: jdbc:mysql://localhost:3306/test?characterEncoding=utf8
9     username: root
10    password: root
11
12 mybatis:
13   type-aliases-package: org.larry.springboot.entity
14   mapper-locations: classpath:mapper/**/*.xml
15   check-config-location: true
```

`active: 【test、dev、prod】`，根据 `active` 的环境，自动采用上面配置的 `springProfile` 的 `logger` 日志

## ##自定义日志路径 (application.yml)

`application.yml` 增加日志相关自定义配置

```
1 logback:
2   logdir: /Users/inke/dev/log/tomcat/sell
3   appname: sell
```

85

<

21

Q

Q

<

>

赏

脉

《Python从小白到大牛》  
图书视频源代码讲师专属答疑群

关闭

🔊

举报



在 logback-spring.xml

12121

85

21

赏

脉

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration scan="true" scanPeriod="60 seconds" debug="false">
3
4     <!--application.yml 传递参数，不能使用logback 自带的<property>标签 -->
5     <springProperty scope="context" name="appname" source="logback.appname"/>
6     <springProperty scope="context" name="logdir" source="logback.logdir"/>
7
8     <contextName>${appname}</contextName>
9
10    <!--输出到控制台 ConsoleAppender-->
11    <appender name="consoleLog" class="ch.qos.logback.core.ConsoleAppender">
12        <!--展示格式 layout-->
13        <layout class="ch.qos.logback.classic.PatternLayout">
14            <pattern>
15                <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pattern>
16            </pattern>
17        </layout>
18    </appender>
19    . . .
20    . . .
```

参考：  
<https://juejin.im/post/58f86981b123db0062363203>  
<http://blog.csdn.net/vitech/article/details/53812137>

文章最后发布于: 2017

有 0 个人打赏

SpringBoot整合+logback日志配置

阅读数 4万+

本次演示的代码结构如下，基于maven，整合SpringBoot、Spring、Mybaits的SSM框架。同时测试logback日志... 博文 来自: qq296398300的博...

想对作者说点什么

Black.. 3个月前 #19楼  
优秀

酱油-程序员 5个月前 #18楼  
图片出不来啊

无量虚空神主 7个月前 #17楼  
图片不能看啊兄dei

查看 21 条热评

【系统学习SpringBoot】SpringBoot配置logging日志及输出日志

阅读数 7万+

SpringBoot默认配置了【org.slf4j】，，，所以配置日志输出到文件只需要在，application配置文件中稍作修改即可，... 博文 来自: 小鼠标的博客

Springboot yml方式 日志配置


阅读数 1万+

最近在输出日志上遇到点不便：想要在程序中打印DEBUG信息，但是又不想单独使用logback.xml文件那么繁重的... 博文 来自: woyaoyonghanziz...

Spring Boot logback 日志配置

阅读数 4544

spring-boot 项目中默认的日志已经是 logback 了，大型项目中，日志的管理也是很重要的，需要合理规划配置，本... 博文 来自: zombres的博客



申请个发明专利 大概要多少费用?有哪些手续

申请发明专利

7403阅读

举报

Python从小白到大牛

图书视频源代码讲师专属答疑群

关闭