Sue

springboot默认日志logback配置解析

前言

在开发过程中，配置并正确的记录日志是很重要的，它便于我们迅速的排查和定位问题。那么在使用springboot的时候我们要怎样正确的配置日志呢？这篇文章将介绍如何使用logbak进行日志记录。

Spring Boot已经默认集成了一些日志系统，如：Java Util Logging，Log4J, Log4J2和Logback。每种Logger都可以通过配置使用控制台或者文件输出日志内容。

SLF4J——Simple Logging Facade For Java，它是一个针对于各类Java日志框架的统一Facade抽象。日志框架众多——常用的有java.util.logging, log4j, logback，commons-logging, Spring框架使用的是Jakarta Commons Logging API (JCL)。而SLF4J定义了统一的日志抽象接口，slf4j入口是众多接口的集合。而真正的日志实现则是在运行时决定的——它提供了各类日志框架的binding。

Logback是log4j框架的作者开发的新一代日志框架，它效率更高、能够适应诸多的运行环境，同时天然支持SLF4J。

开始使用

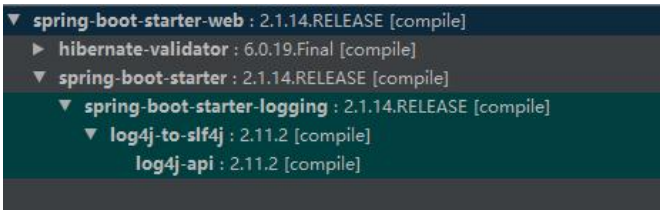
一、添加依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-logging</artifactId>
</dependency>
```

假如maven依赖中添加了spring-boot-starter-logging，那么，我们的Spring Boot应用将自动使用logback作为应用日志框架，Spring Boot启动的时候，由org.springframework.boot.logging.Logging-Application-Listener根据情况初始化并使用。

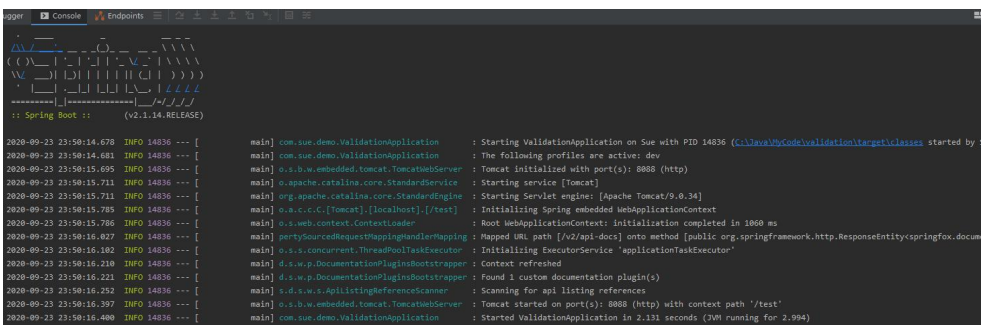
但是通常情况下我们不需要直接添加此依赖，因为spring-boot-starter-web中已经包含了 spring-boot-starter-logging，该依赖内容就是 Spring Boot 默认的日志框架 logback。

通过Maven Helper插件可以看出确实已经包含该依赖，所以我们不需要再额外引入。



二、默认配置

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。在运行应用程序时，可以看到在未手动配置的情况下已经输出了很多INFO级别的日志。



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

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

当该属性置为true的时候，核心Logger（包含嵌入式容器、hibernate、spring）会输出更多内容，但是你自己应用的日志并不会输出为DEBUG级别。

文件输出

默认情况下，SpringBoot将日志输出到控制台，不会写到日志文件。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，与src目录同级。
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

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：

- logging.level.com.sue.demo.controller=DEBUG：com.sue.demo.controller包下所有class以DEBUG级别输出
- logging.level.root=WARN：root日志以WARN级别输出

三、自定义日志配置

由于日志服务一般都在ApplicationContext创建前就初始化了，它并不是必须通过Spring的配置文件控制。因此通过系统属性和传统的Spring Boot外部配置文件依然可以很好的支持日志控制和管理。
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-spring.xml的日志配置文件，spring boot可以为它添加一些spring

2020年9月(6)
2020年8月(3)
2020年7月(3)
2020年6月(9)
2020年5月(2)
2020年4月(2)
2020年3月(6)
2020年1月(4)
2019年12月(7)
2019年11月(2)
2019年10月(2)
2019年9月(7)
更多

目录导航

文章分类

https(1)
mysql(3)
接口设计(2)
设计模式(4)

相册

sundry(5)

阅读排行榜

1. linux 软连接的使用(62438)
2. linux下后台启动springboot项目(43460)
3. linux环境使用clash实现网络代理访问外网(25830)
4. Java8 LocalDateTime和Date相互转换(17392)
5. SpringBoot整合ssm(15413)

评论排行榜

1. linux环境使用clash实现网络代理访问外网(4)
2. MAVEN打包同时引入本地jar包(4)
3. centos7.5误删python2.7之后，导致yum和Python命令无法使用(4)
4. Quartz基础+实例(4)
5. springboot引入其他项目jar包并实现对数据库的操作(3)

boot特有的配置项。

解释：

```
logback和logback-spring.xml都可以用来配置logback，但是两者的加载顺序是不一样的。
logback.xml--->application.properties--->logback-spring.xml.
logback.xml加载早于application.properties，所以如果你在logback.xml使用了变量时，而恰好这个变量是写在
```

上面是默认的命名规则，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，可以在springboot配置文件中通过logging.config属性指定自定义的名字：

```
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。

下面我们来看看一个基础的logback-spring.xml例子

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration  scan="true" scanPeriod="60 seconds" debug="false">
    <contextName>logback</contextName>
    <property name="log.path" value="/Users/tengjun/Documents/log" />
    <!--输出到控制台-->
    <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
       <!-- <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>ERROR</level>
        </filter>-->
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pa
        </encoder>
    </appender>

    <!--输出到文件-->
    <appender name="file" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${log.path}/logback.%d{yyyy-MM-dd}.log</fileNamePattern>
        </rollingPolicy>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pa
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="console" />
        <appender-ref ref="file" />
    </root>

    <!-- logback为java中的包 -->
    <logger name="com.dudu.controller"/>
    <!--logback.LogbackDemo：类的全路径 -->
    <logger name="com.dudu.controller.LearnController" level="WARN" additivity="false">
        <appender-ref ref="console"/>
    </logger>
</configuration>
```

属性详解

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。

根节点<configuration>分别有如下一些常用的属性和节点：

目录
导航

属性一：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,可以通过%contextName来打印日志上下文名称。

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

属性二：设置变量<property>

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logger上下文中。定义变量后，可以使“${}”来使用变量。

```
<property name="log.path" value="D:/Documents/logs/edu"
```

子节点一<appender>

appender用来格式化日志输出节点，有俩个属性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：

```
<!--此日志appender是为开发使用，只配置最底级别，控制台输出的日志级别是大于或等于此级别的日志信息-->
<appender name="console" class="ch.qos.logback.core.ConsoleAppender">
    <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>ERROR</level>
    </filter>
    <encoder>
        <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pattern
    </encoder>
</appender>
```

- %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

关于这里各字符的详细解释，见附录

ThresholdFilter为系统定义的拦截器，如果我们配置为error的话那么只会输出ERROR级别及以上的日志。通常开发的时候我们配置为info或debug级别。

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用于切分文件日志：

```
<!--输出到文件-->
<appender name="file" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>${log.path}/logback.%d{yyyy-MM-dd}.log</fileNamePattern>
        <maxHistory>30</maxHistory>
        <totalSizeCap>100M</totalSizeCap>
    </rollingPolicy>
    <encoder>
        <pattern>%d{HH:mm:ss.SSS} %contextName [%thread] %-5level %logger{36} - %msg%n</pattern
    </encoder>
</appender>
```

其中重要的是rollingPolicy的定义，上例中<fileNamePattern>${log.path}/logback.%d{yyyy-MM-dd}.log</fileNamePattern>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，<maxHistory>30</maxHistory>表示只保留最近30天的日志，以防止日志填满整个磁盘空间。同理，可以使用%d{yyyy-MM-dd_HH-mm}来定义精确到分的日志切分方式。<totalSizeCap>100M</totalSizeCap>用来指定日志文件的上限大小，例如设置为100M的话，那么到了这个值，就会删除旧的日志。

补:如果你想把日志直接放到当前项目下，把${log.path}/去掉即可。

logback 每天生成和大小生成冲突的问题可以看这个解答：传送门

目录
导航

子节点二<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，不区分大小写：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和 OFF，不能设置为INHERITED或者同义词NULL。默认是DEBUG。

如果这里的level设置为info的话，那么<appender-ref ref="console" /> <appender-ref ref="file" />最高也只能输出info级别的日志。

```
<root level="debug">
    <appender-ref ref="console" />
    <appender-ref ref="file" />
</root>
```

子节点三<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和一个可选的addtivity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。
- addtivity: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，先来看一看代码中的使用

```
@GetMapping("/testLog")
public void testLog() {
    //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
    LOG.trace("日志输出 trace");
    LOG.debug("日志输出 debug");
    LOG.info("日志输出 info");
    LOG.warn("日志输出 warn");
    LOG.error("日志输出 error");
}
```

我们来分析一下对于如上的语句，我们使用不同的配置，会有怎样的输出

第一种：带有logger的配置，不指定级别，不指定appender

```
<logger name="com.sue.controller"/>
```

将控制controller包下的所有类的日志的打印，**没有设置打印级别，所以继承他的上级的日志级别**“info”；

没有设置addtivity，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.sue.demo.controller.LoggerController类的testLog方法时，LoggerController在包com.sue.demo.controller中，所以首先执行<logger name="com.sue.demo.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；

```
22:36:33.348 logback [http-nio-8088-exec-1] INFO  c.s.demo.controller.LoggerController - 日志输出 info
22:36:33.348 logback [http-nio-8088-exec-1] WARN  c.s.demo.controller.LoggerController - 日志输出 warn
22:36:33.348 logback [http-nio-8088-exec-1] ERROR c.s.demo.controller.LoggerController - 日志输出 error
```

第二种：带有多个logger的配置，指定级别，指定appender

```
<logger name="com.sue.demo.controller.TestController" level="WARN" additivity="false">
    <appender-ref ref="console"/>
</logger>
```

控制com.sue.demo.controller.LoggerController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.sue.demo.controller.LoggerController类的testLog方法时，先执行<logger name="com.sue.demo.controller.LoggerController" 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，不再向上级root传递打印信息。

```
23:05:16.586 logback [http-nio-8088-exec-1] WARN  c.s.demo.controller.LoggerController - 日志输出 warn
23:05:16.586 logback [http-nio-8088-exec-1] ERROR c.s.demo.controller.LoggerController - 日志输出 error
```

目录
导航

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接到后又打印一次。
注：使用mybatis的时候，sql语句是**debug**下才会打印，而这里我们只配置了info，所以想要查看sql语句的话，有以下两种操作：

- 第一种把<root level="info">改成<root level="DEBUG">这样就会打印sql，不过这样日志那边会出现很多其他消息。
- 第二种就是单独给dao下目录配置debug模式，代码如下，这样配置sql语句会打印，其他还是正常info级别。

通常我们可以在yml配置文件中进行如下配置，因为springboot的配置文件加载早于logback-spring.xml

```
logging:
  config: classpath:logback-spring.xml
  level:
    # root日志以WARN级别输出
    root: info
    # 此包下所有class以DEBUG级别输出
    com.cmbchina.ccd.itpm.leanmanage.dao: debug
    org.mybatis: debug
```

多环境日志输出

据不同环境（prod:生产环境，test:测试环境，dev:开发环境）来定义不同的日志输出，在 logback-spring.xml 中使用 springProfile 节点来定义，方法如下:

```
<!-- 测试环境+开发环境. 多个使用逗号隔开. -->
<springProfile name="test,dev">
    <logger name="com.dudu.controller" level="info" />
</springProfile>
<!-- 生产环境. -->
<springProfile name="prod">
    <logger name="com.dudu.controller" level="ERROR" />
</springProfile>
```

总结

到此为止终于介绍完日志框架了，平时使用的时候推荐用自定义logback-spring.xml来配置，代码中使用日志也很简单，类里面添加private Logger logger = LoggerFactory.getLogger(this.getClass());即可。

附录

完整logback配置及注释：logback.xml

View Code

pattern节点参数解释

```
%c{参数} 或 %logger{参数}  ##输出日志名称
%C{参数} 或 %class{参数    ##输出类型
%d{参数}{时区te{参数}{时区} ##输出时间
%F|%file                  ##输出文件名
highlight{pattern}{style} ##高亮显示
%l  ##输出错误的完整位置
%L  ##输出错误行号
%m 或 %msg 或 %message ##输出错误信息
%M 或 %method ##输出方法名
%n            ##输出换行符
%level{参数1}{参数2}{参数3} ##输出日志的级别
%t 或 %thread              ##创建logging事件的线程名
```

关于上文中%logger{36}的解释，可以查看如下链接：
http://logback.qos.ch/manual/layouts.html#conversionWord

参数	说明	例子

目录导航

<table>
<tr><td rowspan="11">%c</td><td rowspan="11">列出logger名字空间的全称，如果加上{<层数>}表示列出从最内层算起的指定层数的名字空间</td><td>log4j配置文件参数举例</td><td>输出显示媒介</td></tr>
<tr><td colspan="2">假设当前logger名字空间是“a.b.c”</td></tr>
<tr><td>%c</td><td>a.b.c</td></tr>
<tr><td>%c{2}</td><td>b.c</td></tr>
<tr><td>%20c</td><td>（若名字空间长度小于20，则左边用空格填充）</td></tr>
<tr><td>%-20c</td><td>（若名字空间长度小于20，则右边用空格填充）</td></tr>
<tr><td>%.30c</td><td>（若名字空间长度超过30，截去多余字符）</td></tr>
<tr><td>%20.30c</td><td>（若名字空间长度小于20，则左边用空格填充；若名字空间长度超过30，截去多余字符）</td></tr>
<tr><td>%-20.30c</td><td>（若名字空间长度小于20，则右边用空格填充；若名字空间长度超过30，截去多余字符）</td></tr>
<tr><td rowspan="3">%C</td><td rowspan="3">列出调用logger的类的全名（包含包路径）</td><td colspan="2">假设当前类是“org.apache.xyz.SomeClass”</td></tr>
<tr><td>%C</td><td>org.apache.xyz.SomeClass</td></tr>
<tr><td>%C{2}</td><td>xyz.SomeClass</td></tr>
<tr><td rowspan="4">%d</td><td rowspan="4">显示日志记录时间，{<日期格式>}使用ISO8601定义的日期格式</td><td>%d{yyyy/MM/dd HH:mm:ss,SSS}</td><td>2005/10/12 22:23:30,117</td></tr>
<tr><td>%d{ABSOLUTE}</td><td>22:23:30,117</td></tr>
<tr><td>%d{DATE}</td><td>12 Oct 2005 22:23:30,117</td></tr>
<tr><td>%d{ISO8601}</td><td>2005-10-12 22:23:30,117</td></tr>
<tr><td>%F</td><td>显示调用logger的源文件名</td><td>%F</td><td>MyClass.java</td></tr>
<tr><td>%l</td><td>输出日志事件的发生位置，包括类目名、发生的线程，以及在代码中的行数</td><td>%l</td><td>MyClass.main(MyClass.java:129)</td></tr>
<tr><td>%L</td><td>显示调用logger的代码行</td><td>%L</td><td>129</td></tr>
</table>

%m	显示输出消息	%m	This is a message for debug.
%M	显示调用logger的方法名	%M	main
%n	当前平台下的换行符	%n	Windows平台下表示rn UNIX平台下表示n
%p	显示该条日志的优先级	%p	INFO
%r	显示从程序启动时到记录该条日志时已经经过的毫秒数	%r	1215
%t	输出产生该日志事件的线程名	%t	MyClass
%x	按NDC（Nested Diagnostic Context，线程堆栈）顺序输出日志	假设某程序调用顺序是MyApp调用com.foo.Bar	
		%c %x - %m%n	MyApp - Call com.foo.Bar. com.foo.Bar - Log in Bar MyApp - Return to MyApp.
%X	按MDC（Mapped Diagnostic Context，线程映射表）输出日志。通常用于多个客户端连接同一台服务器，方便服务器区分是那个客户端访问留下来的日志。	%X{5}	（记录代号为5的客户端的日志）
%%	显示一个百分号	%%	%

目录导航

分类: log

好文要顶 关注我 收藏该文

少说点话
关注 - 13
粉丝 - 42
+加关注

0 推荐 0 反对

« 上一篇： centos7.6源码离线安装msyql 5.7.30
» 下一篇： springboot使用log4j2代替内置log4j

posted @ 2020-09-24 23:40 少说点话 阅读(434) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

登录后才能发表评论，立即 登录 或 注册， 访问 网站首页

历史上的今天：
2019-09-24 springboot上传文件过大，全局异常捕获，客户端没有返回值

目录
导航

网站运行：2年361天14时31分47秒