1.需要的jar包说明。

<!-- log4j2的核心jar包，使用log4j2，以及log4j2的加载都是这两个jar包即可。下面说到的其他jar包看需要 -->  
<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-api</artifactId>  
 <version>2.10.0</version>  
</dependency>  
<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-core</artifactId>  
 <version>2.10.0</version>  
</dependency>  
  
<!-- 如果需要slf4j则配置以下jar包,如果直接使用log4j2做日志打印,则不需要下面的jar包 -->  
<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.25</version>  
</dependency>  
<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-slf4j-impl</artifactId>  
 <version>2.10.0</version>  
</dependency>

<!--需要使用到了log4j2的AsyncLogger，即做异步日志输出,则需要包含disruptor的jar包,如果全部是同步日志输出,则不需要此jar包 -->  
<dependency>  
 <groupId>com.lmax</groupId>  
 <artifactId>disruptor</artifactId>  
 <version>3.3.11</version>  
</dependency>

<!-- 桥接：告诉commons-logging使用Log4j2,这个jar包我没加，加上的话应该有利于spring日志打印，因为spring依赖的日志包是common-logging包 -->  
<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-jcl</artifactId>  
 <version>2.10.0</version>  
</dependency>

2.如果项目中已经依赖了spring-boot-starter-web，那么就相当于引入了一下jar包，并且包含了spring boot 默认使用的logback相关的jar包。

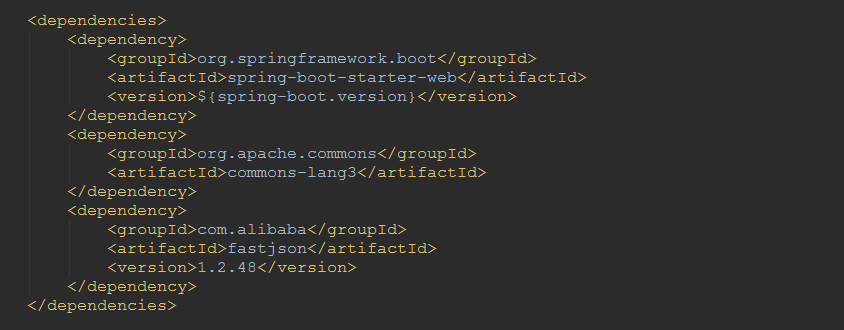
<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-api</artifactId>

<version>2.10.0</version>  
</dependency>

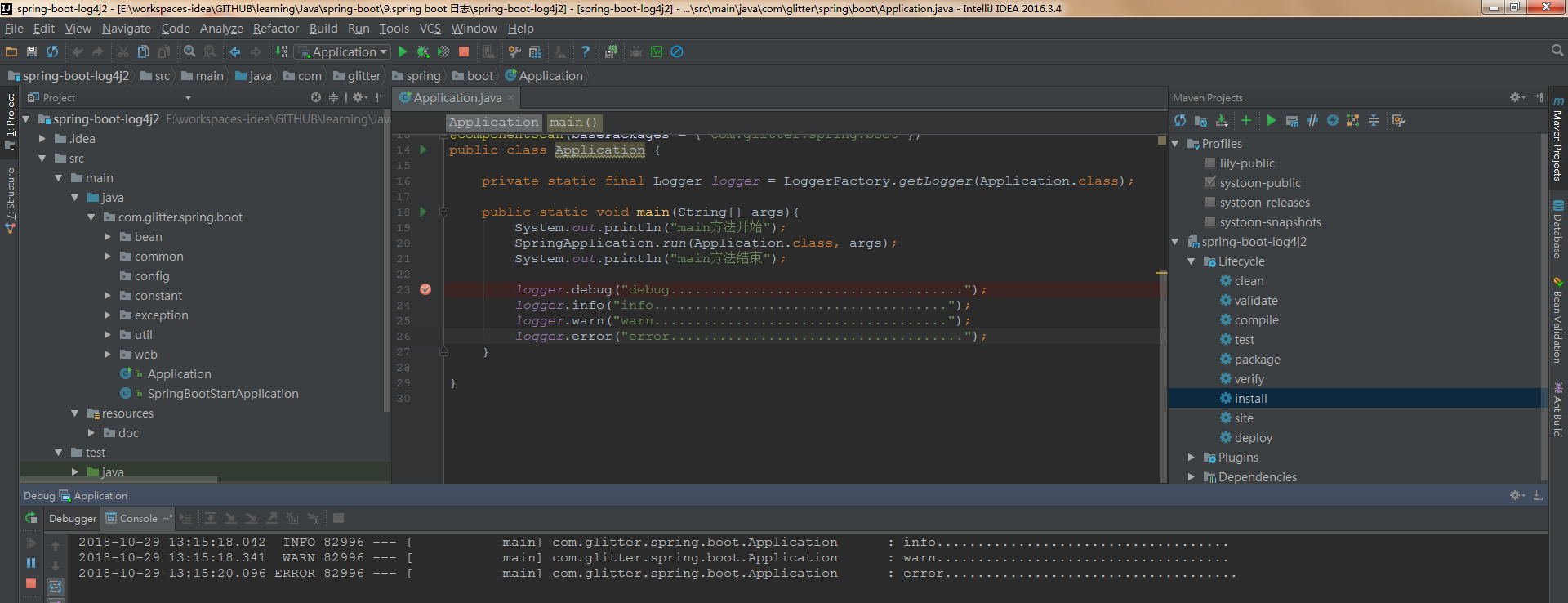
<dependency>  
 <groupId>org.apache.logging.log4j</groupId>  
 <artifactId>log4j-to-slf4j</artifactId>  
 <version>2.10.0</version>  
</dependency>

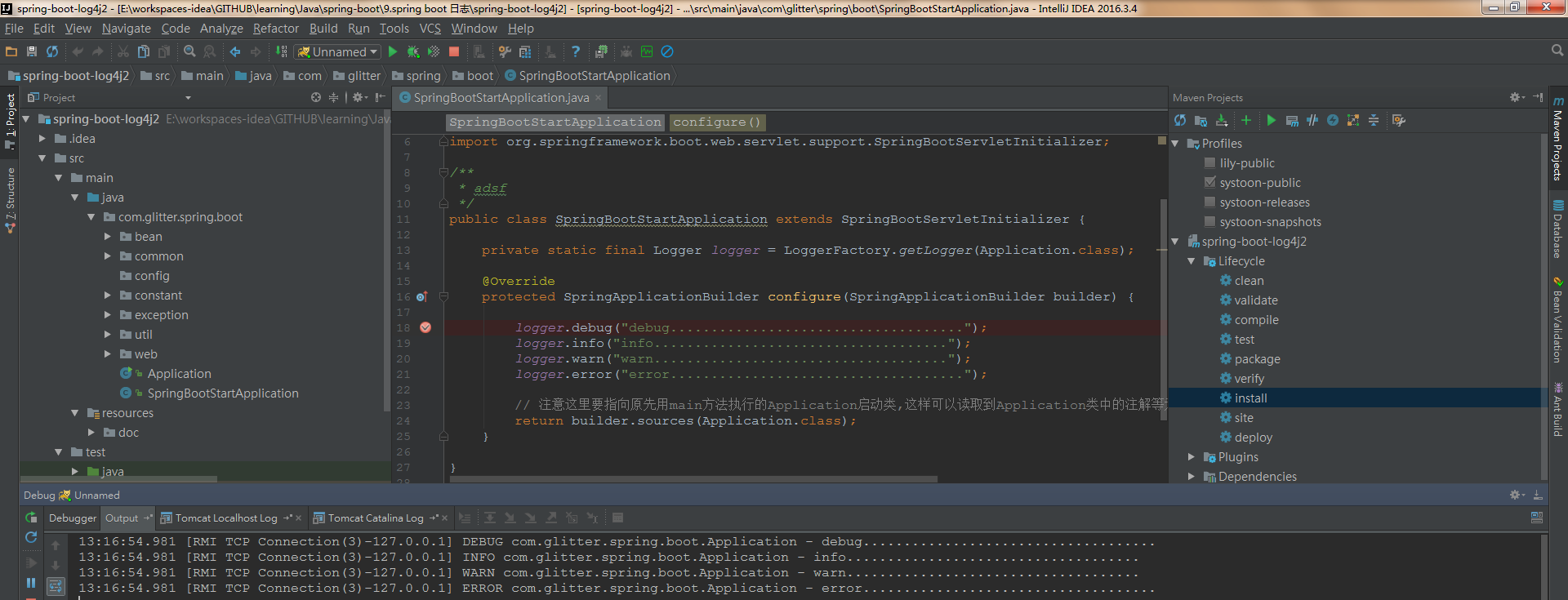
<dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.25</version>  
</dependency>

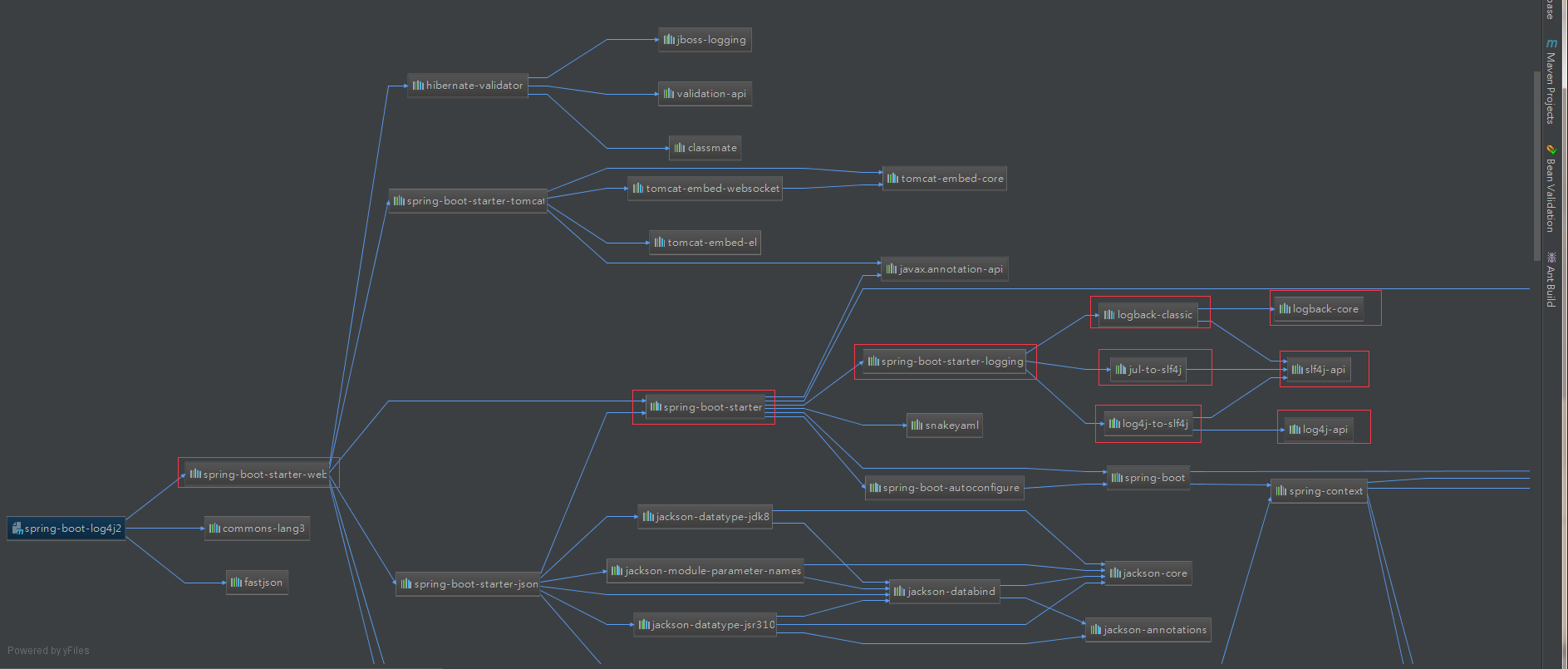
3.如下图，spring boot 的默认日志依赖包是spring-boot-starter-logging，虽然其中也引入是log4j-api等jar包，但是其默认使用的却是logback的。所以我们要排除掉默认的spring-boot-starter-logging，换成spring-boot-starter-log4j2。



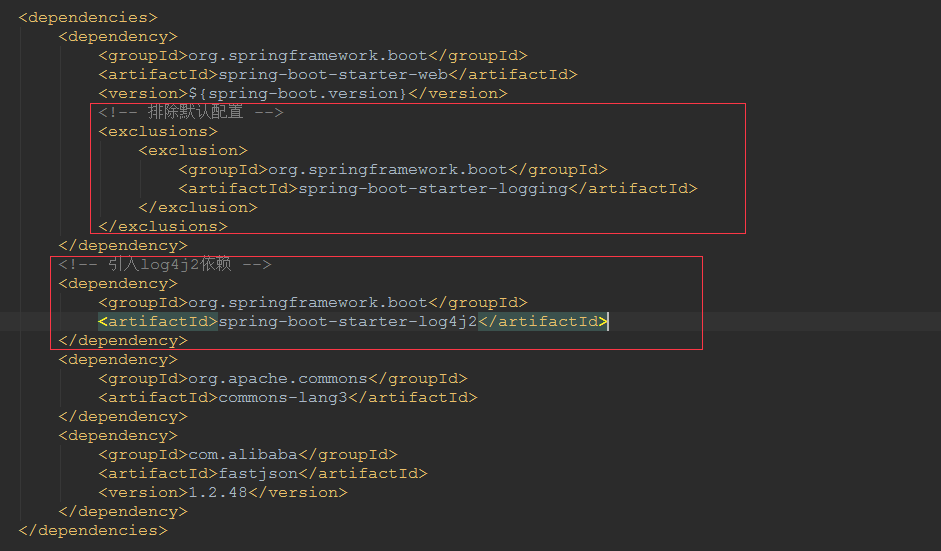
如下图，默认使用logback进行日志打印，包括日志级别和日志格式等都是使用各自容器默认的。

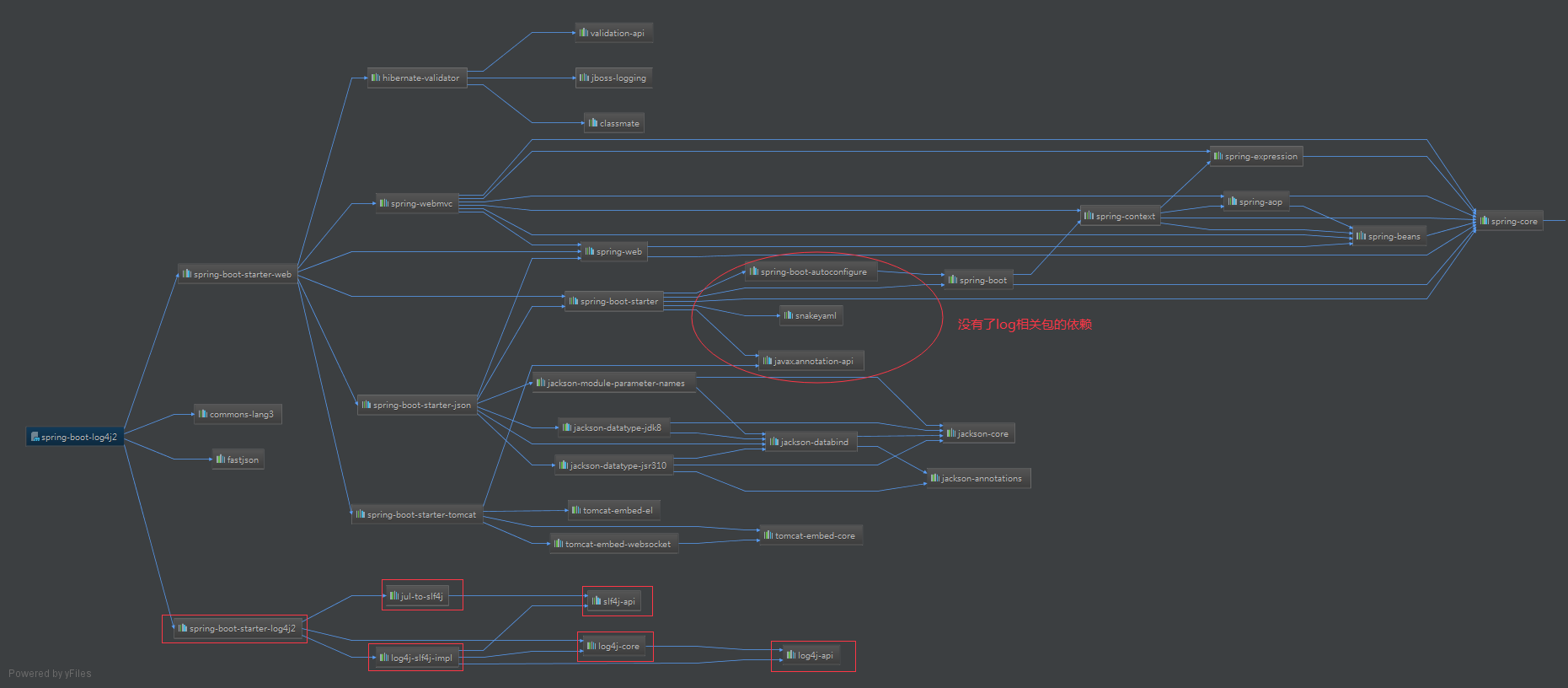




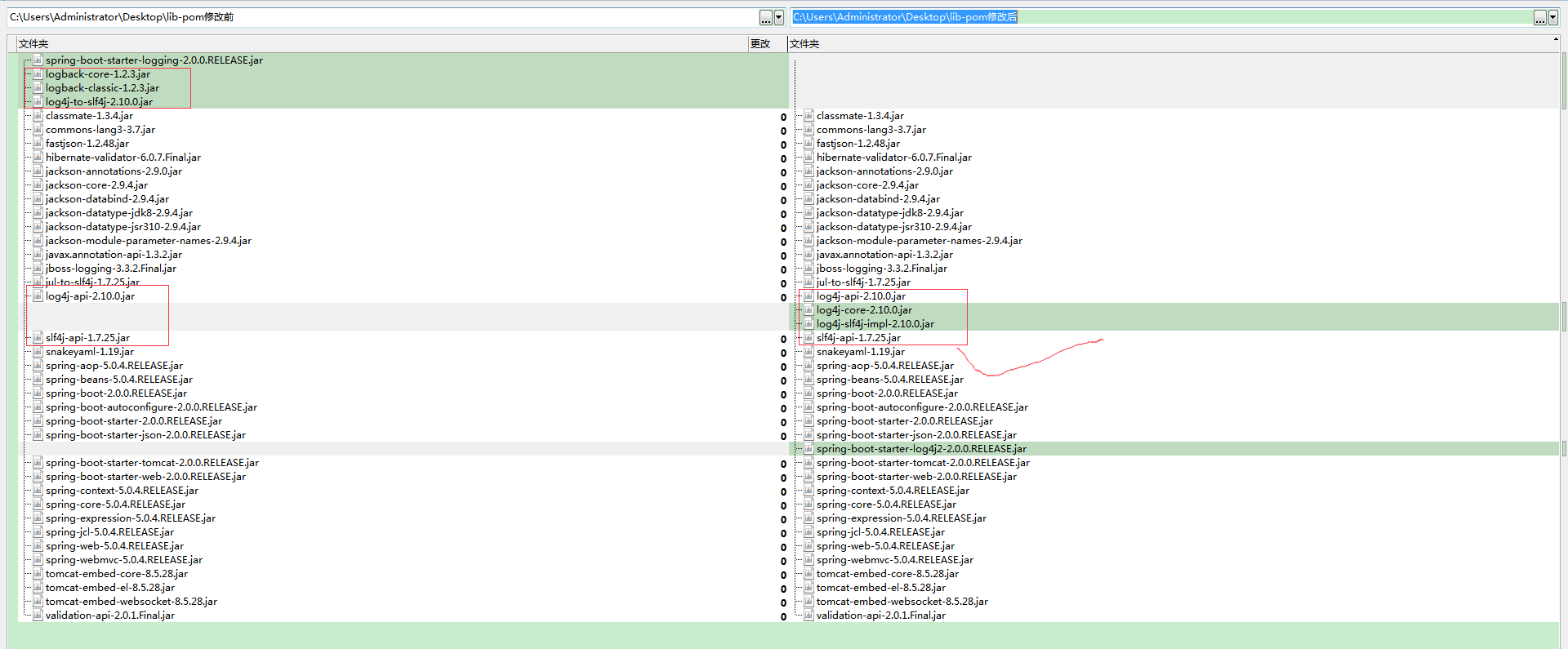


4.要做到第3点所说的，那么就要在pom文件中做相应的修改。



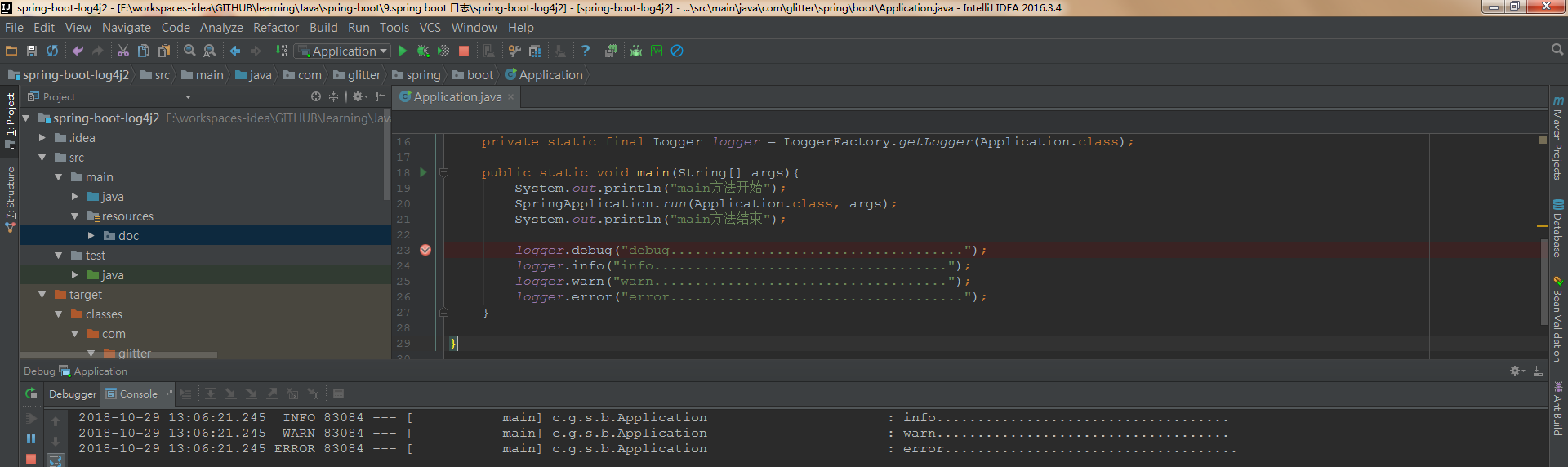


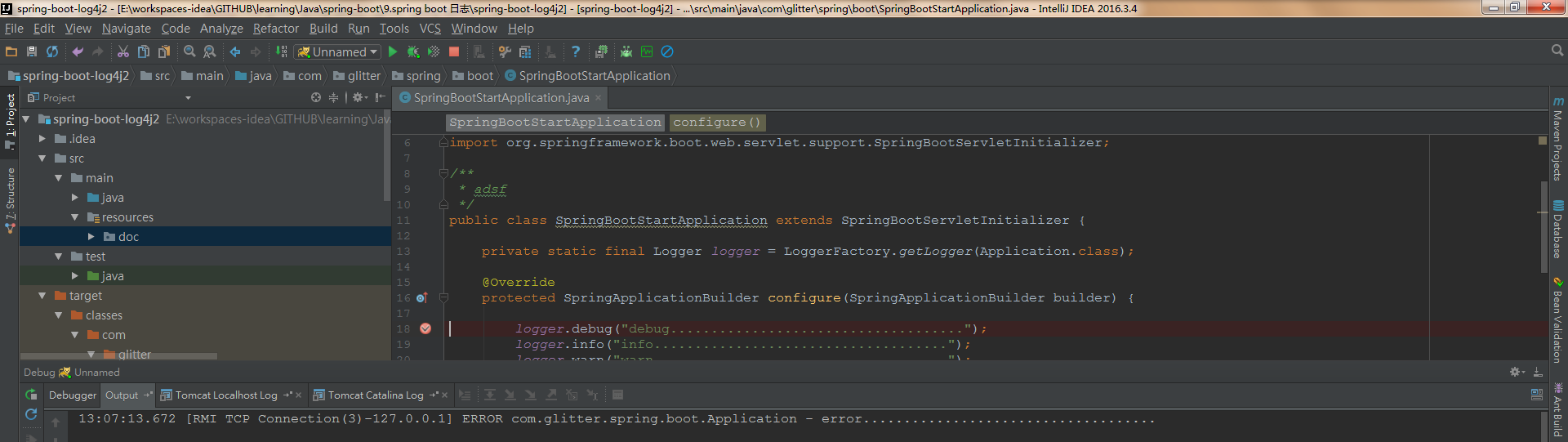
5.附上pom文件调整前后的对比图。



6. 如上，工程中只引入的log4j2的jar包并没有引入任何log4j2的配置文件，在测试的时候可以看到如果是spring boot默认容器启动则输出info及其以上级别的日志信息，如果是tomcat8容器则输入error级别的日志信息。输出的策略完全是所在容器默认控制的，包括格式、级别等。

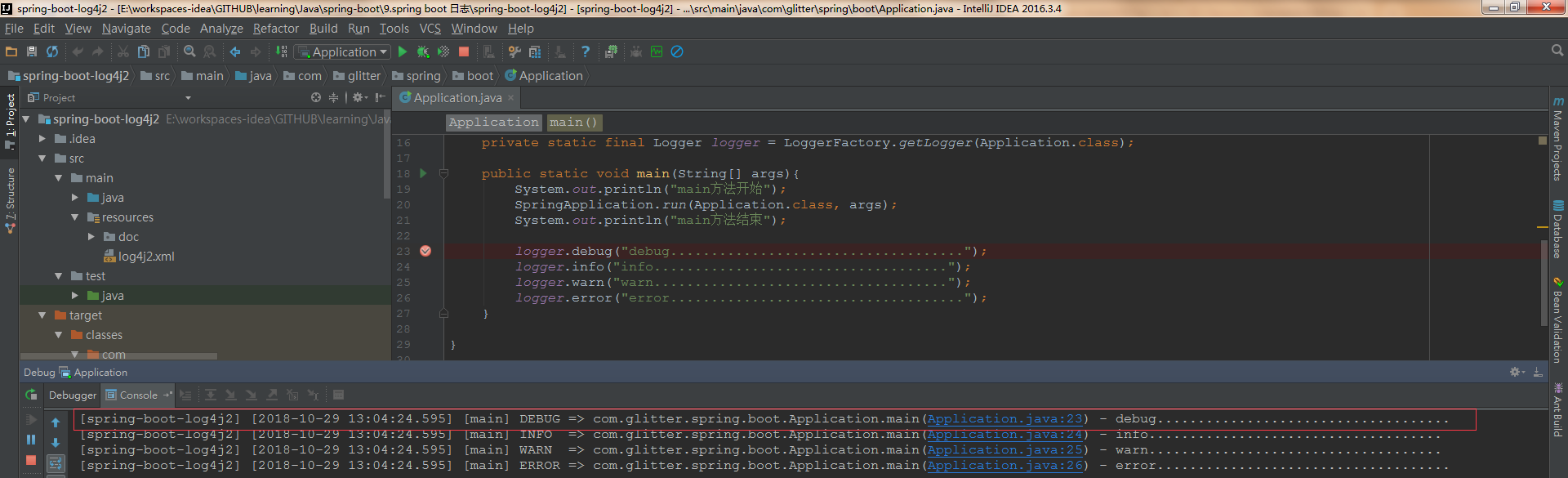
而这些东西都是需要我们自己来控制的，所以，我们需要配置log4j2.xml配置文件。

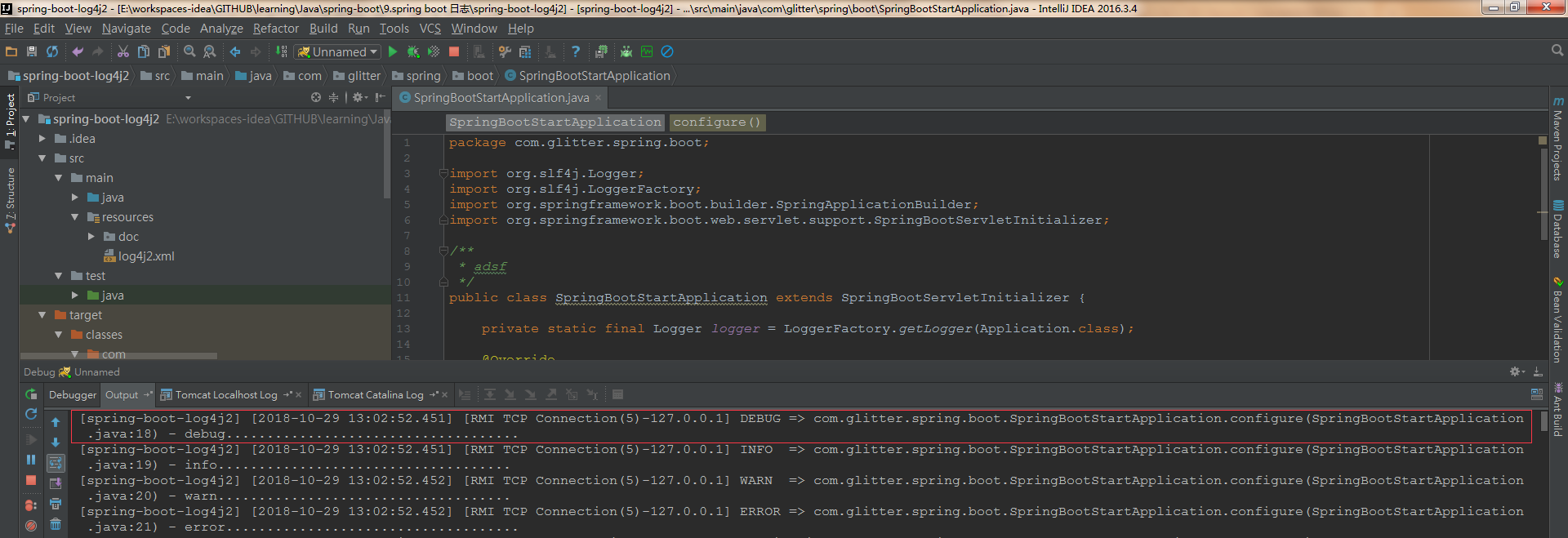




7.在项目中加入log4j2.xml，发现无论是打印的日志级别还是，日志的输出格式都按照我们配置的log4j2.xml在进行打印了。

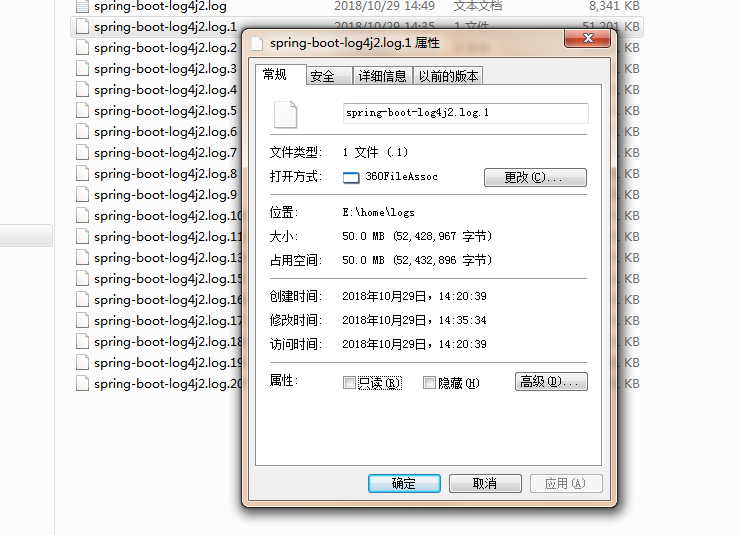
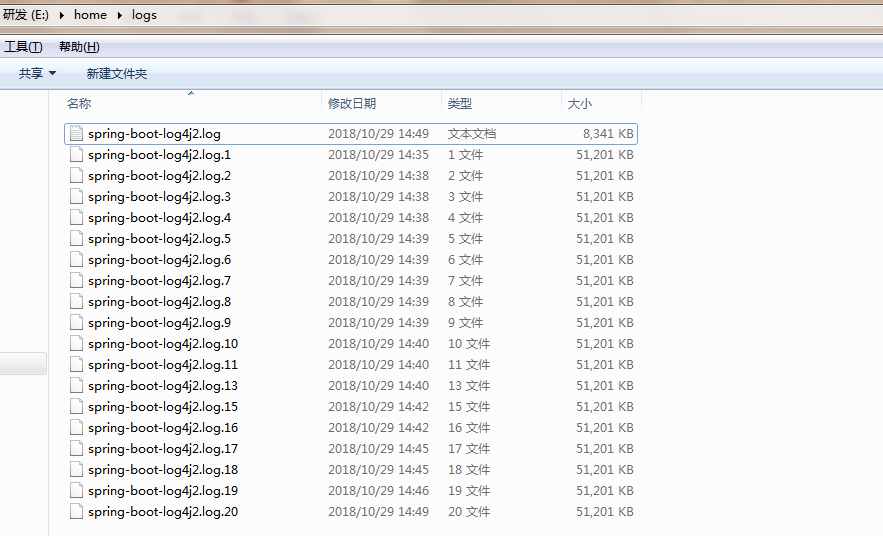
<!-- monitorInterval="1800" 指log4j2每隔1800秒（半小时），自动监控该配置文件是否有变化，如果变化，则自动根据文件内容重新配置 -->  
<!-- status是log4j2本身的日志打印级别，与系统中的日志打印级别无关 -->  
<configuration status="INFO" monitorInterval="1800">  
  
 <!-- appenders下面定义日志输出地 及 在输出地的相关属性比如输出格式,滚动策略,删除策略的配置等等 -->  
 <appenders>  
 <!-- 添加一个控制台追加器 -->  
 <Console name="Console" target="SYSTEM\_OUT">  
 <PatternLayout pattern="[@project.artifactId@] [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %-5p => %l - %m%n" />  
 </Console>  
 </appenders>  
  
 <!-- loggers下面可以配置root和logger子节点.root定义了项目的根日志,logger定义了局部日志. -->  
 <!-- 如果没有配置logger,则按照root的定义来。 -->  
 <!-- 如果配置了logger,则配置的部分按照logger的定义来,即局部覆盖root. -->  
 <loggers>  
 <root level="debug">  
 <appender-ref ref="Console" />  
 </root>  
 </loggers>  
</configuration>

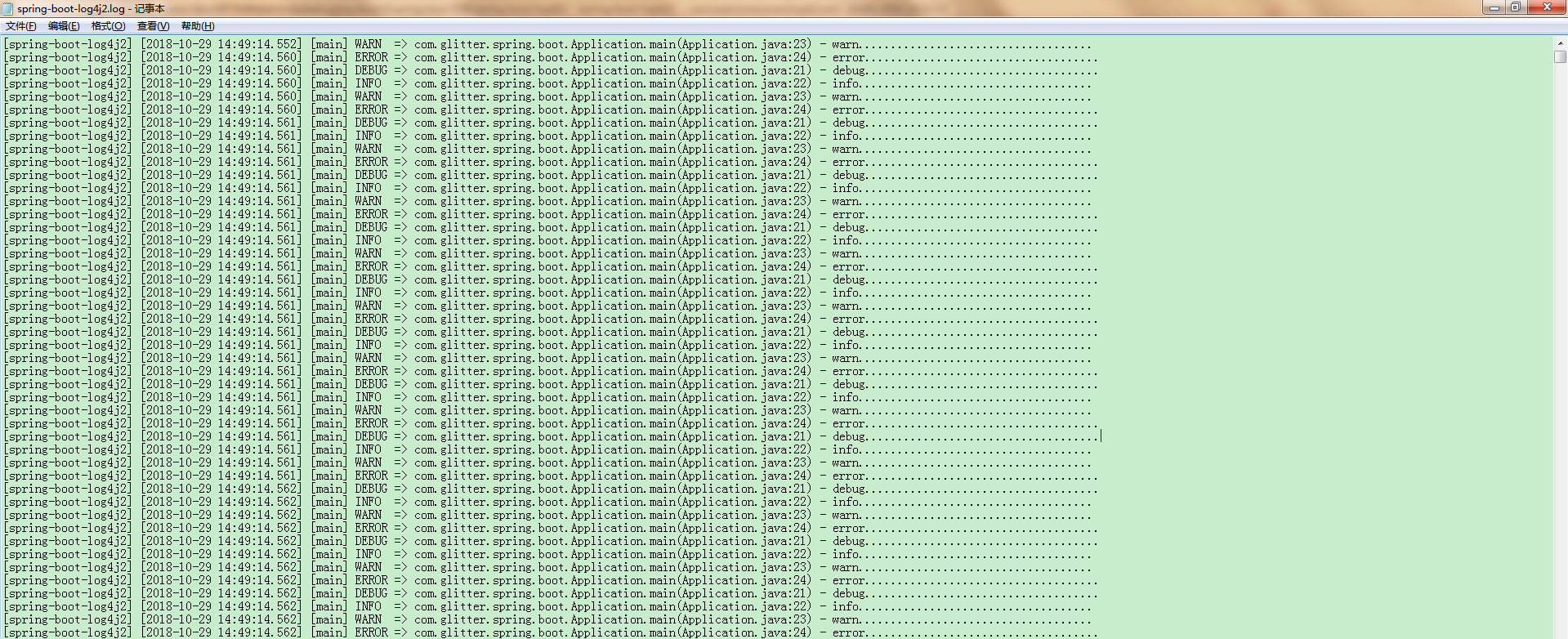




8. 添加一个滚动日志文件追加器,当日志内容达到指定范围后,以某种定义规则打包日志文件。

<!-- monitorInterval="1800" 指log4j2每隔1800秒（半小时），自动监控该配置文件是否有变化，如果变化，则自动根据文件内容重新配置 -->  
<!-- status是log4j2本身的日志打印级别，与系统中的日志打印级别无关 -->  
<configuration status="INFO" monitorInterval="1800">  
  
 <!-- 内部变量定义,做到凡是需要动态调整的参数,只要调整此处即可 -->  
 <Properties>  
 <Property name="LOG\_PATH">/home/logs</Property>  
 <Property name="LOG\_PROJECT\_NAME">@project.artifactId@</Property>  
 <Property name="LOG\_FILE\_FULL\_PATH">${LOG\_PATH}/${LOG\_PROJECT\_NAME}.log</Property>  
 <Property name="LOG\_PATTERN">[${LOG\_PROJECT\_NAME}] [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %-5p => %c.%M(%F:%L) - %m%n</Property>  
 <Property name="ROLLING\_FILE\_MAX\_SIZE">50MB</Property>  
 <Property name="ROLLING\_FILE\_MAX\_NUM">20</Property>  
 <Property name="LOG\_LEVEL">DEBUG</Property>  
 </Properties>  
  
 <!-- appenders下面定义日志输出地 及 在输出地的相关属性比如输出格式,滚动策略,删除策略的配置等等 -->  
 <appenders>  
 <!-- 添加一个控制台日志追加器 -->  
 <Console name="Console" target="SYSTEM\_OUT">  
 <PatternLayout pattern="${LOG\_PATTERN}" />  
 </Console>  
 <!-- 添加一个滚动日志文件追加器,当日志内容达到指定范围后,以某种定义规则打包日志文件 -->  
 <RollingFile name="RollingFile" fileName="${LOG\_FILE\_FULL\_PATH}" filePattern="${LOG\_FILE\_FULL\_PATH}.%i">  
 <PatternLayout pattern="${LOG\_PATTERN}" />  
 <Policies>  
 <!-- 基于大小的触发策略,当日志大小达到50MB时,以上述filePattern定义的格式进行归档 -->  
 <SizeBasedTriggeringPolicy size="${ROLLING\_FILE\_MAX\_SIZE}" />  
 </Policies>  
 <!-- 存档数量不超过20,当日志文件达到20个的时候,会对.log的日志文件进行删除,然后重新开始反复将日志文件写入该文件中,直到${LOG\_PATH}目录下的其他任何一个.log.i的文件有删除 -->  
 <!-- 才会继续创建.log.i进行日志归档 -->  
 <DefaultRolloverStrategy max="${ROLLING\_FILE\_MAX\_NUM}" />  
 </RollingFile>  
 </appenders>  
  
 <!-- loggers下面可以配置root和logger子节点.root定义了项目的根日志,logger定义了局部日志. -->  
 <!-- 如果没有配置logger,则按照root的定义来。 -->  
 <!-- 如果配置了logger,则配置的部分按照logger的定义来,即局部覆盖root. -->  
 <loggers>  
 <root level="${LOG\_LEVEL}">  
 <!--<appender-ref ref="Console" />-->  
 <appender-ref ref="RollingFile" />  
 </root>  
 </loggers>  
</configuration>





9. 常用的日志文件，基本上算是标准格式了，其他的要求可以在这基础上修改。

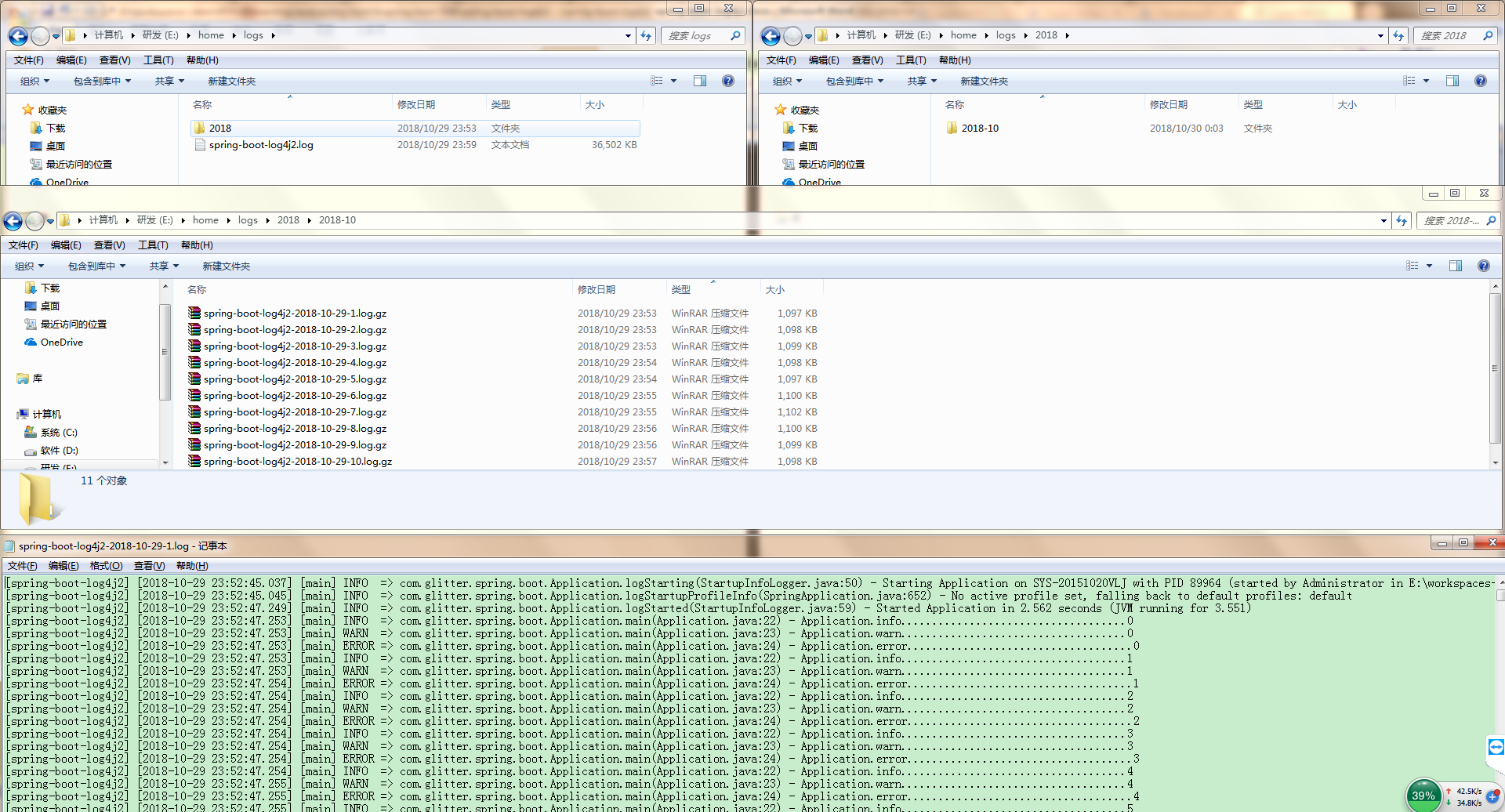
<!-- monitorInterval="1800" 指log4j2每隔1800秒（半小时），自动监控该配置文件是否有变化，如果变化，则自动根据文件内容重新配置 -->  
<!-- status是log4j2本身的日志打印级别,与系统中的日志打印级别无关.当设置成trace时，你会看到log4j2内部各种详细输出 -->  
<configuration status="INFO" monitorInterval="1800">  
  
 <!-- 内部变量定义,做到凡是需要动态调整的参数,只要调整此处即可 -->  
 <Properties>  
 <Property name="LOG\_PATH">/home/logs</Property>  
 <Property name="LOG\_PROJECT\_NAME">@project.artifactId@</Property>  
 <Property name="LOG\_FILE\_FULL\_PATH">${LOG\_PATH}/${LOG\_PROJECT\_NAME}.log</Property>  
 <Property name="LOG\_PATTERN">[${LOG\_PROJECT\_NAME}] [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %-5p => %c.%M(%F:%L) - %m%n</Property>  
 <Property name="ROLLING\_FILE\_MAX\_SIZE">50MB</Property>  
 <Property name="ROLLING\_FILE\_MAX\_NUM">20</Property>  
 <Property name="LOG\_LEVEL\_ROOT">ERROR</Property>  
 <Property name="LOG\_LEVEL\_SPRING">WARN</Property>  
 <Property name="LOG\_LEVEL\_MYBATIS">DEBUG</Property>  
 <Property name="LOG\_LEVEL\_GLITTER">INFO</Property>  
 </Properties>  
  
 <!-- appenders下面定义日志输出地 及 在输出地的相关属性比如输出格式,滚动策略,删除策略的配置等等 -->  
 <appenders>  
 <!-- 添加一个控制台日志追加器 -->  
 <Console name="Console" target="SYSTEM\_OUT">  
 <PatternLayout pattern="${LOG\_PATTERN}" />  
 </Console>  
 <!-- 添加一个滚动日志文件追加器,当日志内容达到指定范围后,以某种定义规则打包日志文件 -->  
 <RollingFile name="RollingFile" fileName="${LOG\_FILE\_FULL\_PATH}" filePattern="${LOG\_FILE\_FULL\_PATH}.%i">  
 <PatternLayout pattern="${LOG\_PATTERN}" />  
 <Policies>  
 <!-- 基于大小的触发策略,当日志大小达到50MB时,以上述filePattern定义的格式进行归档 -->  
 <SizeBasedTriggeringPolicy size="${ROLLING\_FILE\_MAX\_SIZE}" />  
 </Policies>  
 <!-- 存档数量不超过20,当日志文件达到20个的时候,会对.log的日志文件进行删除,然后重新开始反复将日志文件写入该文件中,直到${LOG\_PATH}目录下的其他任何一个.log.i的文件有删除 -->  
 <!-- 才会继续创建.log.i进行日志归档 -->  
 <DefaultRolloverStrategy max="${ROLLING\_FILE\_MAX\_NUM}" />  
 </RollingFile>  
 </appenders>  
  
 <!-- loggers下面可以配置root和logger子节点.root定义了项目的根日志,logger定义了局部日志. -->  
 <!-- 如果没有配置logger,则按照root的定义来。 -->  
 <!-- 如果配置了logger,则配置的部分按照logger的定义来,即局部覆盖root. -->  
 <loggers>  
 <root level="${LOG\_LEVEL\_ROOT}">  
 <appender-ref ref="Console" />  
 <appender-ref ref="RollingFile" />  
 </root>  
 <!--过滤掉spring和mybatis的一些无用的debug信息,默认是继承root的,所以里面的appender-ref都可以省略 -->  
 <logger name="org.springframework" level="${LOG\_LEVEL\_SPRING}"/>  
 <logger name="org.mybatis" level="${LOG\_LEVEL\_MYBATIS}"/>  
 <logger name="com.glitter.spring.boot" level="${LOG\_LEVEL\_GLITTER}"/>  
 </loggers>  
</configuration>

会发现从log4j2开始打印日志开始，就没有打印spring相关的日志，因为spring是WARN级别，而项目运行过程中还没出现WARN及更高级别的情况触发。所以没有打印。

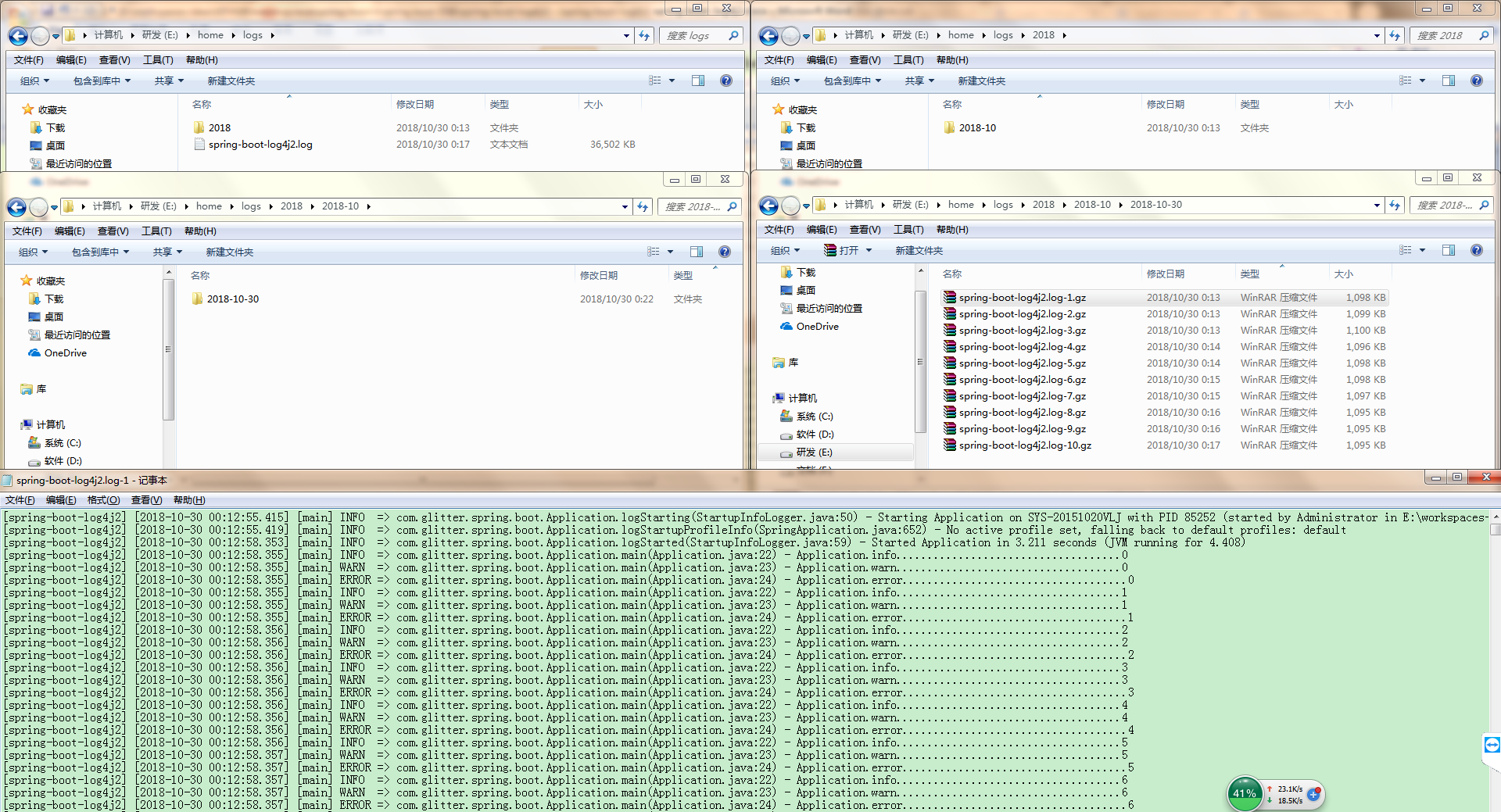
E:\Java\jdk\jdk1.8.0\_151\bin\java -agentlib:jdwp=transport=dt\_socket,address=127.0.0.1:19585,suspend=y,server=n -Dfile.encoding=UTF-8 -classpath "E:\Java\jdk\jdk1.8.0\_151\jre\lib\charsets.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\deploy.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\access-bridge-64.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\cldrdata.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\dnsns.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\jaccess.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\jfxrt.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\localedata.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\nashorn.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\sunec.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\sunjce\_provider.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\sunmscapi.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\sunpkcs11.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\ext\zipfs.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\javaws.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\jce.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\jfr.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\jfxswt.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\jsse.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\management-agent.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\plugin.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\resources.jar;E:\Java\jdk\jdk1.8.0\_151\jre\lib\rt.jar;E:\workspaces-idea\GITHUB\learning\Java\spring-boot\9.spring boot 日志\spring-boot-log4j2\target\classes;E:\Java\MavenRepository\org\springframework\boot\spring-boot-starter-web\2.0.0.RELEASE\spring-boot-starter-web-2.0.0.RELEASE.jar;E:\Java\MavenRepository\org\springframework\boot\spring-boot-starter\2.0.0.RELEASE\spring-boot-starter-2.0.0.RELEASE.jar;E:\Java\MavenRepository\org\springframework\boot\spring-boot\2.0.0.RELEASE\spring-boot-2.0.0.RELEASE.jar;E:\Java\MavenRepository\org\springframework\boot\spring-boot-autoconfigure\2.0.0.RELEASE\spring-boot-autoconfigure-2.0.0.RELEASE.jar;E:\Java\MavenRepository\javax\annotation\javax.annotation-api\1.3.2\javax.annotation-api-1.3.2.jar;E:\Java\MavenRepository\org\springframework\spring-core\5.0.4.RELEASE\spring-core-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\spring-jcl\5.0.4.RELEASE\spring-jcl-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\yaml\snakeyaml\1.19\snakeyaml-1.19.jar;E:\Java\MavenRepository\org\springframework\boot\spring-boot-starter-json\2.0.0.RELEASE\spring-boot-starter-json-2.0.0.RELEASE.jar;E:\Java\MavenRepository\com\fasterxml\jackson\core\jackson-databind\2.9.4\jackson-databind-2.9.4.jar;E:\Java\MavenRepository\com\fasterxml\jackson\core\jackson-annotations\2.9.0\jackson-annotations-2.9.0.jar;E:\Java\MavenRepository\com\fasterxml\jackson\core\jackson-core\2.9.4\jackson-core-2.9.4.jar;E:\Java\MavenRepository\com\fasterxml\jackson\datatype\jackson-datatype-jdk8\2.9.4\jackson-datatype-jdk8-2.9.4.jar;E:\Java\MavenRepository\com\fasterxml\jackson\datatype\jackson-datatype-jsr310\2.9.4\jackson-datatype-jsr310-2.9.4.jar;E:\Java\MavenRepository\com\fasterxml\jackson\module\jackson-module-parameter-names\2.9.4\jackson-module-parameter-names-2.9.4.jar;E:\Java\MavenRepository\org\springframework\boot\spring-boot-starter-tomcat\2.0.0.RELEASE\spring-boot-starter-tomcat-2.0.0.RELEASE.jar;E:\Java\MavenRepository\org\apache\tomcat\embed\tomcat-embed-core\8.5.28\tomcat-embed-core-8.5.28.jar;E:\Java\MavenRepository\org\apache\tomcat\embed\tomcat-embed-el\8.5.28\tomcat-embed-el-8.5.28.jar;E:\Java\MavenRepository\org\apache\tomcat\embed\tomcat-embed-websocket\8.5.28\tomcat-embed-websocket-8.5.28.jar;E:\Java\MavenRepository\org\hibernate\validator\hibernate-validator\6.0.7.Final\hibernate-validator-6.0.7.Final.jar;E:\Java\MavenRepository\javax\validation\validation-api\2.0.1.Final\validation-api-2.0.1.Final.jar;E:\Java\MavenRepository\org\jboss\logging\jboss-logging\3.3.2.Final\jboss-logging-3.3.2.Final.jar;E:\Java\MavenRepository\com\fasterxml\classmate\1.3.4\classmate-1.3.4.jar;E:\Java\MavenRepository\org\springframework\spring-web\5.0.4.RELEASE\spring-web-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\spring-beans\5.0.4.RELEASE\spring-beans-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\spring-webmvc\5.0.4.RELEASE\spring-webmvc-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\spring-aop\5.0.4.RELEASE\spring-aop-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\spring-context\5.0.4.RELEASE\spring-context-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\spring-expression\5.0.4.RELEASE\spring-expression-5.0.4.RELEASE.jar;E:\Java\MavenRepository\org\springframework\boot\spring-boot-starter-log4j2\2.0.0.RELEASE\spring-boot-starter-log4j2-2.0.0.RELEASE.jar;E:\Java\MavenRepository\org\apache\logging\log4j\log4j-slf4j-impl\2.10.0\log4j-slf4j-impl-2.10.0.jar;E:\Java\MavenRepository\org\slf4j\slf4j-api\1.7.25\slf4j-api-1.7.25.jar;E:\Java\MavenRepository\org\apache\logging\log4j\log4j-api\2.10.0\log4j-api-2.10.0.jar;E:\Java\MavenRepository\org\apache\logging\log4j\log4j-core\2.10.0\log4j-core-2.10.0.jar;E:\Java\MavenRepository\org\slf4j\jul-to-slf4j\1.7.25\jul-to-slf4j-1.7.25.jar;E:\Java\MavenRepository\org\apache\commons\commons-lang3\3.7\commons-lang3-3.7.jar;E:\Java\MavenRepository\com\alibaba\fastjson\1.2.48\fastjson-1.2.48.jar;D:\Program Files (x86)\JetBrains\IntelliJ IDEA 2016.3.4\lib\idea\_rt.jar" com.glitter.spring.boot.Application  
Connected to the target VM, address: '127.0.0.1:19585', transport: 'socket'  
main方法开始  
2018-10-29 16:20:38,296 main INFO Log4j appears to be running in a Servlet environment, but there's no log4j-web module available. If you want better web container support, please add the log4j-web JAR to your web archive or server lib directory.  
  
. \_\_\_\_ \_ \_\_ \_ \_  
/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \  
( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  
\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  
' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /  
=========|\_|==============|\_\_\_/=/\_/\_/\_/  
:: Spring Boot :: (v2.0.0.RELEASE)  
  
[spring-boot-log4j2] [2018-10-29 16:20:38.476] [main] INFO => com.glitter.spring.boot.Application.logStarting(StartupInfoLogger.java:50) - Starting Application on SYS-20151020VLJ with PID 81156 (started by Administrator in E:\workspaces-idea\GITHUB\learning\Java\spring-boot\9.spring boot 日志\spring-boot-log4j2)  
[spring-boot-log4j2] [2018-10-29 16:20:38.483] [main] INFO => com.glitter.spring.boot.Application.logStartupProfileInfo(SpringApplication.java:652) - No active profile set, falling back to default profiles: default  
[spring-boot-log4j2] [2018-10-29 16:20:40.867] [main] INFO => com.glitter.spring.boot.Application.logStarted(StartupInfoLogger.java:59) - Started Application in 2.775 seconds (JVM running for 3.53)  
main方法结束  
[spring-boot-log4j2] [2018-10-29 16:20:40.870] [main] INFO => com.glitter.spring.boot.Application.main(Application.java:21) - info....................................  
[spring-boot-log4j2] [2018-10-29 16:20:40.870] [main] WARN => com.glitter.spring.boot.Application.main(Application.java:22) - warn....................................  
[spring-boot-log4j2] [2018-10-29 16:20:40.870] [main] ERROR => com.glitter.spring.boot.Application.main(Application.java:23) - error....................................

11.按时间策略滚动日志文件 - - 年月。

<configuration status="INFO" monitorInterval="1800">  
  
 <Properties>  
 <Property name="LOG\_PATH">/home/logs</Property>  
 <Property name="LOG\_PROJECT\_NAME">@project.artifactId@</Property>  
 <Property name="LOG\_PATTERN">[${LOG\_PROJECT\_NAME}] [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %-5p => %c.%M(%F:%L) - %m%n</Property>  
 <Property name="ROLLING\_FILE\_NAME">${LOG\_PATH}/${LOG\_PROJECT\_NAME}.log</Property>  
 <!-- 按月,日来进行归档并压缩,每个文件大小限制50MB,也缩后大约节省空间几十倍 -->  
 <!-- <Property name="ROLLING\_FILE\_PATTERN">${LOG\_PATH}/$${date:yyyy-MM}/${LOG\_PROJECT\_NAME}-%d{yyyy-MM-dd}-%i.log.gz</Property> -->  
 <!-- 按年,月,日来进行归档并压缩,每个文件大小限制50MB,也缩后大约节省空间几十倍 -->  
 <!-- <Property name="ROLLING\_FILE\_PATTERN">${LOG\_PATH}/${date:yyyy}/${date:yyyy-MM}/${LOG\_PROJECT\_NAME}-%d{yyyy-MM-dd}-%i.log.gz</Property> -->  
 <!-- 按年,月,日来进行归档并压缩,每个文件大小限制50MB,也缩后大约节省空间几十倍 -->  
 <Property name="ROLLING\_FILE\_PATTERN">${LOG\_PATH}/${date:yyyy}/${date:yyyy-MM}/${date:yyyy-MM-dd}/${LOG\_PROJECT\_NAME}.log-%i.gz</Property>  
 <Property name="ROLLING\_FILE\_MAX\_SIZE">50MB</Property>  
 <Property name="ROLLING\_FILE\_MAX\_NUM">20000</Property>  
 <Property name="LOG\_LEVEL\_ROOT">ERROR</Property>  
 <Property name="LOG\_LEVEL\_SPRING">WARN</Property>  
 <Property name="LOG\_LEVEL\_MYBATIS">DEBUG</Property>  
 <Property name="LOG\_LEVEL\_GLITTER">INFO</Property>  
 </Properties>  
  
 <appenders>  
 <Console name="Console" target="SYSTEM\_OUT">  
 <PatternLayout pattern="${LOG\_PATTERN}" />  
 </Console>  
 <RollingFile name="RollingFile" fileName="${ROLLING\_FILE\_NAME}" filePattern="${ROLLING\_FILE\_PATTERN}">  
 <PatternLayout pattern="${LOG\_PATTERN}" />  
 <Policies>  
 <!-- SizeBasedTriggeringPolicy是基于大小的rollover,这个很容易理解，如果大小大于某个阈值，上面是50MB的时候就会滚动。-->  
 <!-- TimeBasedTriggeringPolicy是基于时间的rollover,基于时间的滚动触发策略,很少用。其中一个参数是interval,表示多久滚动一次。默认是1 hour。 -->  
 <!-- 另一个modulate=true用来调整时间：比如现在是早上3am，interval是4，那么第一次滚动是在4am，接着是8am，12am...而不是7am-->  
 <!-- <TimeBasedTriggeringPolicy/> -->  
 <SizeBasedTriggeringPolicy size="${ROLLING\_FILE\_MAX\_SIZE}" />  
 </Policies>  
 <!-- DefaultRolloverStrategy属性如不设置,则默认为最多同一文件夹下7个文件,这里设置了20000,基本相当于不限制个数了。-->  
 <!-- 这是一个大坑,如果默认最大是7个,那么一旦一个文件夹下的日志文件数超过7个,就会覆盖掉文件夹下的第一个日志文件,整个文件夹下的日志文件都可能会错乱,造成日志丢失和一些异常情况 -->  
 <DefaultRolloverStrategy max="${ROLLING\_FILE\_MAX\_NUM}" />  
 </RollingFile>  
 </appenders>  
  
 <loggers>  
 <root level="${LOG\_LEVEL\_ROOT}">  
 <appender-ref ref="Console" />  
 <appender-ref ref="RollingFile" />  
 </root>  
 <logger name="org.springframework" level="${LOG\_LEVEL\_SPRING}"/>  
 <logger name="com.ibatis" level="${LOG\_LEVEL\_MYBATIS}"/>  
 <logger name="com.glitter.spring.boot" level="${LOG\_LEVEL\_GLITTER}"/>  
 </loggers>  
  
</configuration>



按年月日生成日志：



10.TODO 异步日志。

11.TODO filter 日志级别分类。