HowToDoInJava

Logging in Spring Boot

Last Updated: January 7, 2022 By: Lokesh Gupta Spring Boot Logging Java Logging

Logging in spring boot is very flexible and easy to configure. Spring boot supports various logging providers through simple configurations. In this tutorial, we will look at various logging options and configurations supported by Spring boot.

Table Of Contents

1. Default Zero Configuration Logging

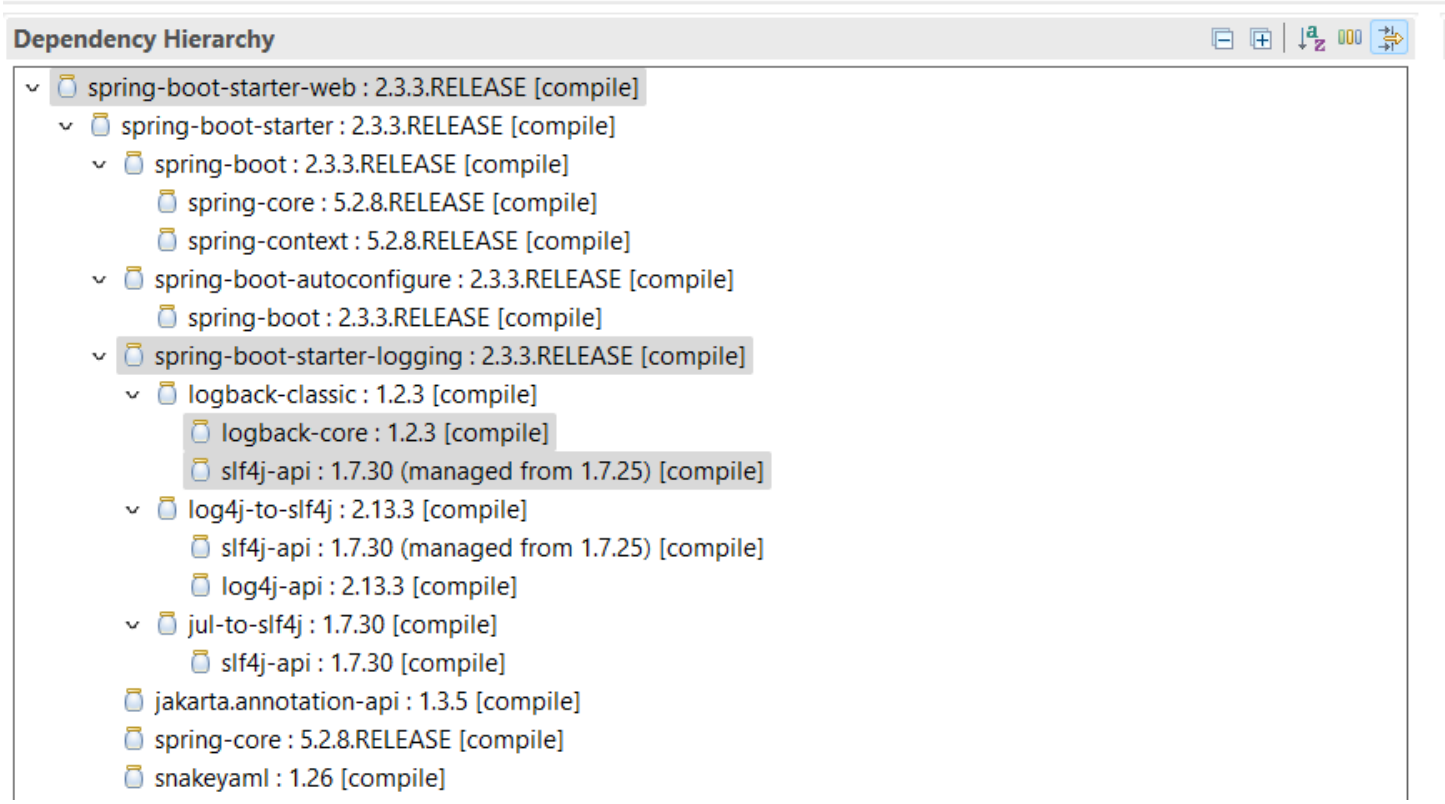
Spring boot's active enabled logging is determined by spring-boot-starter-logging artifact and its auto-configuration that enables any one of the supported logging providers (*Java Util Logging*, Log4J2, and Logback) based on the configuration provided.

1.1. Default Logging Provider is Logback

If we do not provide any logging-specific configuration, we will still see logs printed in "*console*" because **default logging uses Logback to log DEBUG messages into the Console**.

Spring boot's internal logging is written with *Apache Commons Logging* so it is one and only mandatory dependency. Till Spring boot version 1.x – we had to import commons-logging manually. Since boot 2.x, it is downloaded transitively.

To be more precise, most boot starters, such as `spring-boot-starter-web`, depends on `spring-boot-starter-logging`, which pulls in `logback` for us.



1.2. Default Logging Configuration

By default, when no default configuration file is found, logback will add a `ConsoleAppender` to the root logger and this will log all the messages in the Console.

The output is formatted using a PatternLayoutEncoder set to the pattern '*%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} – %msg%n*'. Also, **by default, the root logger is assigned the *DEBUG* level**.

This is the equivalent configuration used by default.

```
<configuration debug="true">
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender"
```

```
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg
    </encoder>
  </appender>

  <root level="debug">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

1.3. Log Statements using SLF4J

To add log statements in application code, use `org.slf4j.Logger` and `org.slf4j.LoggerFactory` from SLF4J. It provides lots of useful methods for logging and also decouples the logging implementation from the application.

Application.java

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application
{
    private static final Logger LOGGER=LoggerFactory.getLogger(Applic

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);

        LOGGER.info("Simple log statement with inputs {}, {} and {}",
    }
}
```

Output

```
2019-07-28 12:16:57.129  INFO 3416 --- [main]
com.howtodoinjava.demo.Application: Simple log statement with inputs
```

2. Custom Logging with Logback

The default logging is good enough for getting started and POC purposes. But in real-life enterprise applications, we need more fine control over logging with other complex requirements. In that case, having a dedicated logging configuration is suitable.

Spring boot by default uses logback, so to customize its behavior, all we need to **add logback.xml in classpath** and define customization over the file.

The given configuration file uses the console appender and the rolling file appended.

logback.xml

```
<configuration>
    <property name="LOG_ROOT" value="c:/temp/logs" />
    <property name="LOG_FILE_NAME" value="application" />

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppende
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}
        </encoder>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingF
        <file>${LOG_ROOT}/${LOG_FILE_NAME}.log</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimel
            <fileNamePattern>${LOG_ROOT}/${LOG_FILE_NAME}-%d{yyyy-MM-
            <!-- each archived file's size will be max 10MB -->
            <maxFileSize>10MB</maxFileSize>
            <!-- 30 days to keep -->
            <maxHistory>30</maxHistory>
```

```
            <!-- total size of all archive files, if total size > 100
            <totalSizeCap>100GB</totalSizeCap>
        </rollingPolicy>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36}
        </encoder>
    </appender>

    <logger name="com.howtodoinjava.app" level="INFO" additivity="fals
                <appender-ref ref="FILE"/>
        </logger>

    <root level="ERROR">
        <appender-ref ref="STDOUT" />
        <appender-ref ref="FILE" />
    </root>
</configuration>
```

3. Logging with Log4j2

3.1. Excluding Logback and Including Log4j2

To exclude default logging, exclude `spring-boot-starter-logging` dependency and explicitly import add spring-boot-starter-log4j2 to the classpath.

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

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
</dependency>
```

3.2. Add Log4j2 Configuration File

Now, add log4j2 specific configuration file in the classpath (typically in **resources** folder). It can be named as any of the following:

- log4j2-spring.xml
- log4j2.xml

If we have the logging configuration in any other file (e.g. log4j2.properties, applogs.xml etc), we can use `logging.file` property name to specify its path in that `application.properties` file.

log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN" monitorInterval="30">
    <Properties>
        <Property name="LOG_PATTERN">%d{yyyy-MM-dd'T'HH:mm:ss.SSSZ} %
        <Property name="APP_LOG_ROOT">c:/temp</Property>
    </Properties>
    <Appenders>
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout pattern="${LOG_PATTERN}" />
        </Console>

        <RollingFile name="file"
            fileName="${APP_LOG_ROOT}/SpringBoot2App/application.log"
            filePattern="${APP_LOG_ROOT}/SpringBoot2App/application-%
            <PatternLayout pattern="${LOG_PATTERN}" />
            <Policies>
                <SizeBasedTriggeringPolicy size="19500KB" />
            </Policies>
```

```
            <DefaultRolloverStrategy max="1" />
        </RollingFile>

    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="console" />
            <AppenderRef ref="file" />
        </Root>
    </Loggers>
</Configuration>
```

Drop me your questions related to **logging configurations in spring boot**.

Happy Learning !!

Recommended Reading:

1. Spring boot logging with application.properties
2. Spring Boot Logging with application.yml
3. Spring boot console logging configuration example
4. Spring boot profile specific logging example
5. Spring Boot – Performance Logging

6. Spring Boot Logging with Lombok
7. RESTEasy + Tomcat 7 + Log4j Logging Example
8. RESTEasy + Tomcat 7 + SLF4J Logging Example
9. Jersey Logging Request and Response Entities using Filter
0. Java Fluent Logging with Flogger

1 thought on “Logging in Spring Boot”

Saurabh

January 7, 2022 at 7:07 pm

Hello Admin,
While implementing the above code example of Logback Logging, I faced logback configuration error.
In the example, we used
, but in configuration, there is rolling file policy, i think that is cause of error.

So the appender should be of class — ch.qos.logback.core.rolling.RollingFileAppender, so rolling file policy can be applied.
Thanks.

Reply

Leave a Comment

Name *

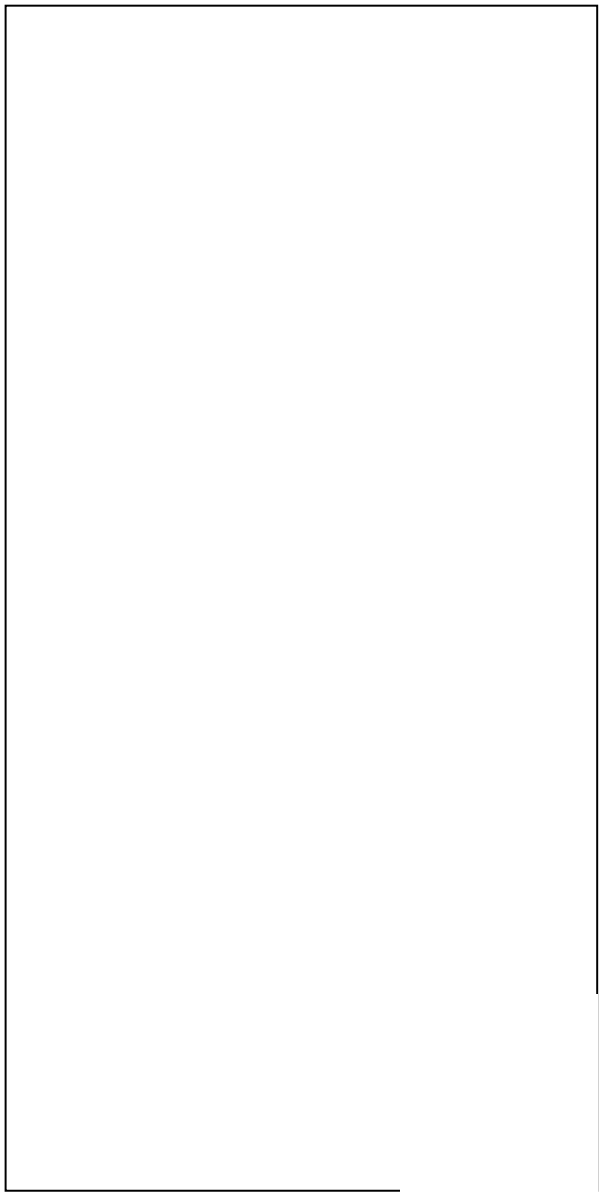
Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …



HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial