Spring Professional Exam Tutorial v5.0

Question 07

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot allows you to configure following aspects of logging:

- Logging Levels
- Logging Pattern
- Logging Colors
- Logging Output – console, file
- Logging Rotation
- Logging Groups
- Logging System used
  - Logback – default
  - log4j2
  - JDK (Java Util Logging)
- Logging System specific configuration:
  - Logback - `logback-spring.xml`
  - log4j2 - `log4j2-spring.xml`
  - JDK (Java Util Logging) - `logging.properties`

Question 07 - Can you control logging with Spring Boot? How?

Logging Levels can be set via `application.properties`:

```
logging.level.root=WARN
app.service.a.level=ALL
app.service.b.level=FINEST
app.service.c.level=FINER
```

or by using logging system specific configuration, `logback-spring.xml` example:

```
<logger name="app.service.a" level="INFO"/>
<logger name="app.service.b" level="DEBUG"/>
<logger name="app.service.c" level="WARN"/>
```

You can also use `--debug` or `--trace` argument when launching spring boot application:

```
$ java -jar myapp.jar --debug
```

It is also possible to specify `debug=true` or `trace=true` in `application.properties`.

Question 07 - Can you control logging with Spring Boot? How?

Logging patterns can be set via `application.properties`:

```
logging.pattern.console=%clr(%d{yy-MM-dd E HH:mm:ss.SSS}) \
         {blue} %clr(%-5p) %clr(${PID}){faint} \
         %clr(---){faint} %clr([%8.15t]){cyan} \
         %clr(%-40.40logger{0}){blue} \
         %clr(:){red} %clr(%m){faint}%n
```

or by using logging system specific configuration, `logback-spring.xml` example:

```
<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
     <encoder>
        <pattern>%d{yyyy-MM-dd} | %d{HH:mm:ss.SSS} | %thread | %5p | %logger{25} | %12(ID: %8mdc{id}) | %m%n</pattern>
        <charset>utf8</charset>
     </encoder>
</appender>
```

Question 07 - Can you control logging with Spring Boot? How?

When ANSI support for logging output is enabled, you can use colors to format your logs. Colors are used with `%clr` word.

Example:

```
%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){yellow}
```

Example usage in `logback-spring.xml`:

```
<appender name="2CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%clr(%thread){red} | %5p | %logger{25} | %m%n</pattern>
            <charset>utf8</charset>
        </encoder>
</appender>
```

Following colors are supported:

- blue
- cyan
- faint
- green
- magenta
- red
- yellow

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot by default logs only to console. You can change this behavior via `application.properties` or by using logging system specific configuration.

If you want to change this behavior via `application.properties`, you need to set one of following property:

- `logging.file`
- `logging.path`

You can also do this via logging system specific configuration, for example `logback-spring.xml`:

```
...
<root level="INFO">
    <appender-ref ref="CONSOLE"/>
    <appender-ref ref="FILE"/>
    <appender-ref ref="ROLLING-APPENDER"/>
</root>
...
```

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot allows you to control logs rotation by specifying maximum file size and maximum number of logs file to keep in history.

To achieve this behavior through `application.properties`, you need to set following properties:

- `logging.file.max-size`
- `logging.file.max-history`

You can also configure logging system specific settings, for example in logback-spring.xml you can configure rolling appender:

```
<rollingPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedRollingPolicy">
    <fileNamePattern>
        ${LOG_PATH}/archived/log_%d{dd-MM-yyyy}_%i.log
    </fileNamePattern>
    <maxFileSize>10MB</maxFileSize>
    <maxHistory>10</maxHistory>
    <totalSizeCap>100MB</totalSizeCap>
</rollingPolicy>
```

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot can group loggers into group, which simplifies log management.

You can do this on `application.properties` level in following way:

```
logging.group.service-d-and-e=app.service.d, app.service.e
logging.level.service-d-and-e=DEBUG
```

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot allows you to chose between logging subsystem.

To use default `Logback`, you just need to use `spring-boot-starter` dependency, autoconfiguration will setup all required beans:

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

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot allows you to chose between logging subsystem.

To use `log4j2`, you just need to exclude `spring-boot-starter-logging` and add dependency to `log4j2`:

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

Question 07 - Can you control logging with Spring Boot? How?

Spring Boot allows you to chose between logging subsystem.

To use `JDK (Java Util Logging)`, you need to exclude `spring-boot-starter-logging`:

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

Then initialize JDK logging in the code:

```
LogManager.getLogManager().readConfiguration(
        SpringBootConsoleApplication.class.getResourceAsStream("/logging.properties")
);
```