Logging in Spring Boot

Spring Boot supports various logging frameworks, such as Logback, Log4j2 and Java Util Logging (JUL).

Logback: A popular logging framework that serves as default in many Spring Boot applications. It offers flexible configuration and good performance.

Log4j2: Another widely used logging framework with features such as asynchronous logging and support for various output formats.

Java Util Logging: The default logging framework included in the Java Standard Edition.

Spring Boot comes with a default logging configuration that uses Logback as the default logging implementation.

If you want to customize the logging configuration, you can create your own **`logback.xml`** file in the **`src/main/resources`** directory. When spring boot detects this file in the project, it will use it instead of the default configuration.

Logging Levels

Logging Levels help in categorizing log statements based on their severity. The common logging levels are:

* OFF (output no logs)
* TRACE
* DEBUG
* INFO
* WARN
* ERROR

Spring boot allows us to configure logging using properties or YAML files.

INFO, WARN, ERROR are by default enabled.

If we want to enable TRACE and DEBUG also, we must enable it using **`application.properties`** or **`application.yaml`.**

A close-up of a logo

Description automatically generated

The log format and ANSI colors are configured By Spring Boot beforehand to make the logging more readable. The following components are present in the log output:

1. Date of logging
2. Time with millisecond precision
3. Log level shows INFO, WARN and ERROR by default
4. Process ID
5. ----- acts as a separator
6. Name of the thread enclosed in square brackets
7. Name of the logger that shows source class name
8. The log message



It creates an appender of class **`ConsoleAppender`** which will output log messages to the console like **`System.out.print`** normally would. A pattern is set that the log messages will adhere to which come provided with some notations that are replaced with generated values depending on message that has been sent to the logger. Some notations have been included in the example and below are explanations of what each does.

* %d - outputs the time which the log message occurred in formats that **`SimpleDateFormat`** allows.
* %thread - outputs the name of the thread that the log message occurred in.
* $-5level - outputs the logging level of the log message.
* %logger{36} - outputs the package + class name the log message occurred in. The number inside the brackets represents the maximum length of the package + class name. If the output is longer than the specified length it will take a substring of the first character of each individual package starting from the root package until the output is below the maximum length. The class name will never be reduced. A nice diagram of this can be found in the Conversion word docs.
* %M - outputs the name of the method that the log message occurred in (apparently this is quite slow to use and not recommended unless you’re not worried about performance, or the method name is particularly important to you).
* %msg - outputs the actual log message.
* %n - line break
* %magenta() - sets the color of the output contained in the brackets to magenta (other colors are available).
* highlight() - sets the color of the output contained in the brackets depending on the logging level (for example ERROR = red).

Filtering Logs

Deciding what log information gets processed based on the log level is a good way to get started, but at some point, that’s simply not enough.

Logback has solid support for additional filtering, beyond just the log level, this is done with the help of filters – which determine whether a log message should be displayed or not.

Simply put, a filter needs to implement the Filter class, with a single decide() method. This method returns enumeration values of type **`FilterReply`**: DENY, NEUTRAL or ACCEPT.

* The DENY value indicates the log event will not be processed, while ACCEPT means the log event is processed, skipping the evaluation of the remaining filters.
* Finally, NEUTRAL allows the next filters in the chain to be evaluated. If there are no more filters, the message is logged.

There are three types of filters:

* LevelFilter
* ThresholdFilter
* EvaluatorFilter.

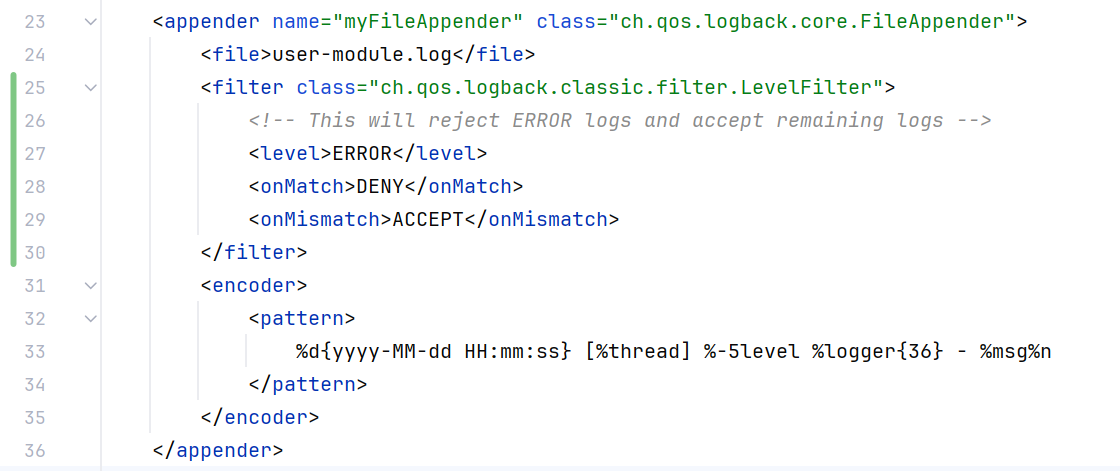
Threshold Filter

A screen shot of a computer program

Description automatically generated

This will save only ERROR logs because we have applied filter. The **`ThresholdFilter`** filters events below the specified threshold.

Level Filter



The LevelFilter and ThresholdFilter are related to the log level, with the difference that LevelFilter verifies if a log message is equal to a given level, while the ThresholdFilter checks if log events are below a specified level.

Evaluator Filter

The EvaluatorFilter implements the same decide() method as the two level-based filters above and uses an EventEvaluator object to determine whether a log message is accepted or denied.