Using Logback

In this lesson, you will learn about using Logback.

WE'LL COVER THE FOLLOWING

Logback

Logback is intended as a successor to the popular log4j project that is used for logging to console and file. The main advantage of using *logback* over *log4j* is that its internals has been re-written to perform about ten times faster and has a smaller memory footprint as well.

Here, we can find a reason to use logback over log4j.

Dependency to add

Maven

```
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
```

Gradle

```
compile group: 'ch.qos.logback', name: 'logback-classic', version: '1.2.3'
```

```
compile group: 'ch.qos.logback', name: 'logback-classic', version: '1.2.3'
```

A sample ***logback.xml*** for logging to console and writing to file:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <appender name="CONSOLE"
        class="ch.qos.logback.core.ConsoleAppender">
        <append>false</append>
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>info</level>
        </filter>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg - \(%fil
e:%line\) %n</pattern>
        </encoder>
        <immediateFlush>false</immediateFlush>
    </appender>
    <appender name="WARN" class="ch.qos.logback.core.FileAppender">
        <file>logs/warn.log</file>
        <append>false</append>
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>warn</level>
        </filter>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg - \(%fil
e:%line\) %n</pattern>
        </encoder>
    </appender>
    <appender name="DEBUG"
        class="ch.qos.logback.core.FileAppender">
        <file>logs/debug.log</file>
        <append>false</append>
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>debug</level>
        </filter>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg - \(%fil
e:%line\) %n</pattern>
        </encoder>
    </appender>
    <appender name="INFO" class="ch.qos.logback.core.FileAppender">
        <file>logs/info.log</file>
        <append>false</append>
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
```

```
            <level>info</level>
        </filter>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg - \(%fil
e:%line\) %n</pattern>
        </encoder>
    </appender>
    <appender name="ERROR"
        class="ch.qos.logback.core.FileAppender">
        <file>logs/error.log</file>
        <append>false</append>
        <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
            <level>error</level>
        </filter>
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg - \(%fil
e:%line\) %n</pattern>
        </encoder>
    </appender>

    <root level="DEBUG">
        <appender-ref ref="CONSOLE" />
        <appender-ref ref="INFO" />
        <appender-ref ref="DEBUG" />
        <appender-ref ref="WARN" />
        <appender-ref ref="ERROR" />
    </root>

</configuration>
```

Logging levels

As discussed in the Logback Architecture, the following is the hierarchy of logging levels, where `ALL` logs all the levels, `TRACE` logs levels greater than or equal to `TRACE`, and so on.

```
ALL < TRACE < DEBUG < INFO < WARN < ERROR < FATAL
```

Using Logback in the project

Once the ***logback.xml*** is created under any of these locations in the project `<project.dir>/src/main/resources` or `<project.dir>/src/test/resources`, we can use the below Java code to log.

Main.java

logback.xml

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Main {

    private static final Logger LOG = LoggerFactory.getLogger(Main.class);

    public static void main( String args[] ) {

        LOG.trace("log trace");
        LOG.trace("log trace with arguments '{}'", "arg");

        LOG.warn("log warn");
        LOG.warn("log warn with arguments '{}'", "arg");

        LOG.info("log info");
        LOG.info("log info with arguments '{}'", "arg");

        LOG.debug("log debug");
        LOG.debug("log debug with arguments '{}", "arg");

        LOG.error("log error");
        LOG.error("log error with arguments '{}'", "arg");
    }
}
```

When we run the above Java code, which is logging statements in different log levels, the content will be printed on the console and written to different files according to the levels (`INFO` level in `info.log`, `DEBUG` level in `debug.log`, and so on).

- From the starting lines until ***Registering current configuration as safe fallback point***, Logback is trying to look for the configuration files and creating appenders for the found configurations. These lines can be ignored.
- Log level for the console is set to `INFO`. So all the levels till `INFO` (i.e., `INFO`, `WARN`, `ERROR`, `FATAL`) will be logged. That's the reason why we see log statements at levels - `WARN`, `INFO`, `ERROR` on the console.
- Subsequently, for the other log levels, filters are applied and logged only until their respective levels in their respective log files (i.e., `INFO` in

with their respective levels in their respective log files (i.e., `INFO` in `info.log` and so on). Log files will be created if not present already. If they are already present, it will be overwritten.

Now that we are familiar with Logback, in the next lesson, you'll learn how to automatically log using `AspectJ`.