4.5

Logging

Logging

Logging is the process of tracking all the events that happen after a piece of code is run. It is a very important aspect of software development as it helps to track where exactly the code crashes and thus eases debugging.

A logging framework can be used to perform all the tasks like setting log file destinations, customizing log messages , etc.

SLF4J

To make logging easier for programmers, Java provides a variety of logging frameworks like : log4J, java.util.logging (JUL), tiny log, logback, etc.

Spring Boot comes with SLF4J inbuilt, which is an abstraction of all these logging frameworks. SLF4J stands for **Simple Logging Façade for Java**. It allows users to work with any of the logging frameworks with a single dependency.

Elements of Logging Framework

Every logging framework comes with three elements.

1. Logger — capture the messages
2. Formatter — formats the messages captured by the logger
3. Handler — Dispatches the messages by printing them on the console , or storing them in a file , sending an email, etc.

Log Levels

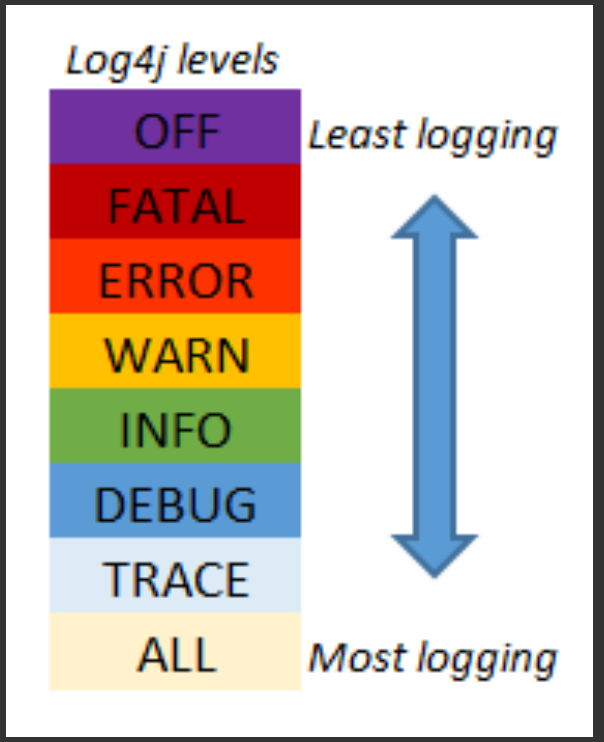
The messages logged can be of various security levels . Spring Boot supports five log levels which are

1. FATAL - fatal error crashing the system
2. ERROR – runtime errors
3. WARN – warning
4. INFO – events occurring at the run time
5. DEBUG – Information about the flow of the system
6. TRACE – more detailed information about the flow of the system

Setting Log Levels

When you enable a level, Log4j logs these events at that level and all levels above it. For example, enabling WARN events will show WARN through FATAL events, but not INFO through TRACE.

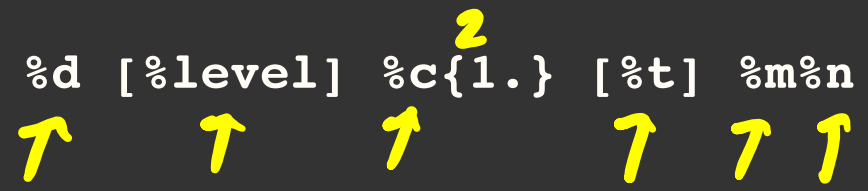
```
logging.level.root=INFO
logging.level.com.myPackageName =DEBUG
```



Log Formatters

The log messages can be formatted and customized according to our requirements by setting colors , message format , etc.

```
logging.pattern.console= %d [%level] %c{1.} [%t] %m%n
```



%d – date
% level – log level
%c – class path
%t – thread executing
%m – message
%n – new line

Log Handlers

```
To set the file name:
logging.file.name = error.log

To set the pattern for the log file
logging.pattern.file    =    %clr(%d{yyyy-MM-dd    HH:mm:ss.SSS}){green}
[%level] %c{1.} [%t] %m%n
```

Logback Configuration XML

```xml
<configuration>
    <property name="LOG_FILE" value="application.log" />

    <appender name="ROLLING" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_FILE}</file>
        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_FILE}.%d{yyyy-MM-dd}.%i.zip</fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>10MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
            <maxHistory>30</maxHistory>
        </rollingPolicy>
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="INFO">
        <appender-ref ref="ROLLING" />
    </root>
</configuration>
```