Log

dasturlash.uz

Log 1

- If you are working or worked on any java project, you might have logged some debug ,error or info statements for debugging or tracking purpose. Those statements are written by logging libraries to a file on disk or on some other medium.
- Logging is a critical aspect of any project, so various framework has been written
- Some of java logging framework are **Log4j** ,**logback** , **Apache common logging**, SLF4J , **java.util.Logger** **etc.**
- All these logger do the work of logging your logs to file or other specified media.

What is SLF4J

- Spring Boot uses SLF4J (Simple Logging Façade for Java)
- Writing log messages with SLF4J is very easy.
- You first need to call the *getLogger* method on the *LoggerFactory* to instantiate a new *Logger* object.
- . You can then call one of the *debug*, *info*, *warning*, *error* or *fatal* methods on the *Logger* to write a log message with the corresponding log level.
- public class MyClass {

```
        Logger log = LoggerFactory.getLogger(MyClass .class);

          public void myMethod() {
               log.info("This is an info message"); //
          }

}
```

dasturlash.uz

Apache Log4j

- Apache Log4j is a very old logging framework and was the most popular one for several years.
- It introduced basic concepts, like hierarchical log levels and loggers, that are still used by modern logging frameworks.
- The development team announced Log4j’s end of life in 2015.
- private static final Logger log = Logger.getLogger(App.class);

Logback

- Logback was written by the same developer who implemented Log4j with the goal to become its successor.
- It follows the same concepts as Log4j but was rewritten to improve the performance, to support SLF4J natively, and to implement several other improvements like advanced filtering options and automatic reloading of logging configurations.
- The framework consists of 3 parts:
  - logback-core
  - logback-classic
  - logback-access

Apache Log4j2

- Apache Log4j2 is the youngest of these three frameworks, and its goal is to improve on both of them by providing its own improvements on Log4j, including some of the improvements included in Logback and avoiding problems of Log4j and Logback.
- So like Logback, Log4j2 provides support for SLF4J, automatically reloads your logging configuration, and supports advanced filtering options

Back to SLF4J - Write Log Messages

- import org.slf4j.Logger;
  import org.slf4j.LoggerFactory;
- private static final Logger LOGGER = LoggerFactory.getLogger(AppController.class);
- And then you can write log messages for different log levels as follows:
  - LOGGER.trace("for tracing purpose");
  - LOGGER.debug("for debugging purpose");
  - LOGGER.info("for informational purpose");
  - LOGGER.warn("for warning purpose");
  - LOGGER.error("for logging errors");
  - LOGGER.fatal(“fatal “)

Enable DEBUG and TRACE mode in Spring Boot

- The default log level used by Spring Boot is INFO, meaning that only logging messages written in the levels INFO, WARN and ERROR are outputted to the console.
- So to enable lower log levels i.e. DEBUG or TRACE, you need to specify the following entry in the application.properties file:
- logging.level.com.company=DEBUG
- And the following line specifies log level DEBUG for only loggers in the package com.company:
- logging.level.com.company=DEBUG

Configure Log Level in Spring Boot

- Spring Boot also allows you to specify a specific log level for a specific logger name in the application.properties file.
- For example, the following line sets the log level to WARN for all loggers:
- logging.level.root=WARN

Customize Log Pattern in Spring Boot

- If you want to customize/change the default log pattern used by Spring Boot in the console, you can use the following properties:
- logging.pattern.console: specifies the log pattern for output to the console.
- logging.pattern.dateformat: specifies the log date format
- logging.pattern.level: specifies the pattern for log level.

Customize Log Pattern in Spring Boot

- Here are some examples. To customize format for the entire logging line:
- logging.pattern.console
  - %d{yyyy-MM-dd HH:mm:ss} - %-5level - %msg%n
  - %d{yyyy-MM-dd HH:mm:ss} [%thread] - %-5level %logger{36} - %msg%n
  - %date{dd MMM yyyy;HH:mm:ss.SSS} [%thread] %highlight(%-5level) %cyan(%logger{36}) - %green(%msg%n)
- To format only the date time part in the logging line:
  - logging.pattern.dateformat=dd-MM-yyyy - HH:mm:ss
- To format only the log level part in the logging line:
  - logging.pattern.level=%highlight(%-5level)
- Note that the pattern for console will override the patterns for date and log level. And these properties work with Logback only.

How to use Log File in Spring Boot

- By default, Spring Boot outputs log messages to the console.
- If you want to write logs to files, you need to specify the following properties in the application.properties file:
- logging.file.name: specifies name of the log file.
  - Spring Boot will create the file under the application’s directory.
- logging.file.path: specifies absolute path to the log file (not including file name)
  - Spring Boot will create the log file name is spring.log under this path.
- logging.pattern.file: specifies log pattern used for the log file.
  - If not, the console pattern is used.
- Note that you can’t use both the properties logging.file.name and logging.file.path together, though they seem to be related. You must use one or another, not both.

How to use Log File

- Here are some examples. Specify log file name:
  - logging.file.name=logs/myLogFile.log
- Specify log file path (file name will be spring.log):
  - logging.file.path=D:/Temp/logs
- Specify log pattern for log file:
  - logging.pattern.file=%date{dd MMM yyyy - HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
- Maximum log file size
  - logging.file.max-size=10MB
- **Logback Rolling File Logging**
  - logging.pattern.rolling-file-name=MyApp-%d{yyyy-MM-dd}.%i.log
  - myLogFile.log
  - myLogFile-2020-07-29.0.log
  - myLogFile-2020-07-28.0.log

Spring Boot Logging with Lombok

- Project Lombok is very handy tool for removing the boilerplate code from application.
- Lombok can also be used to configure logging in spring boot applications and thus removing the boilerplate code for getting the logger instance.
- Before using it, we must import the lombok in spring boot application. Don’t forget to install lombok into eclipse before using it.

Lombok Log annotations

- @CommonsLog - org.apache.commons.logging.Log
  - private static final Log log = LogFactory.getLog(LogExample.class);
- @Log - java.util.logging.Logger
  - private static final Logger log = Logger.getLogger(LogExample.class.getName());
- @Log4j2 - org.apache.logging.log4j.Logger
  - private static final Logger log = LogManager.getLogger(LogExample.class);
- @Slf4j - org.slf4j.Logger
  - private static final Logger log = LoggerFactory.getLogger(LogExample.class);