Logging

Logging is the process of writing log messages during the execution of a program to a central place. This logging allows you to report and persist error and warning messages as well as info messages (e.g., runtime statistics) so that the messages can later be retrieved and analyzed.

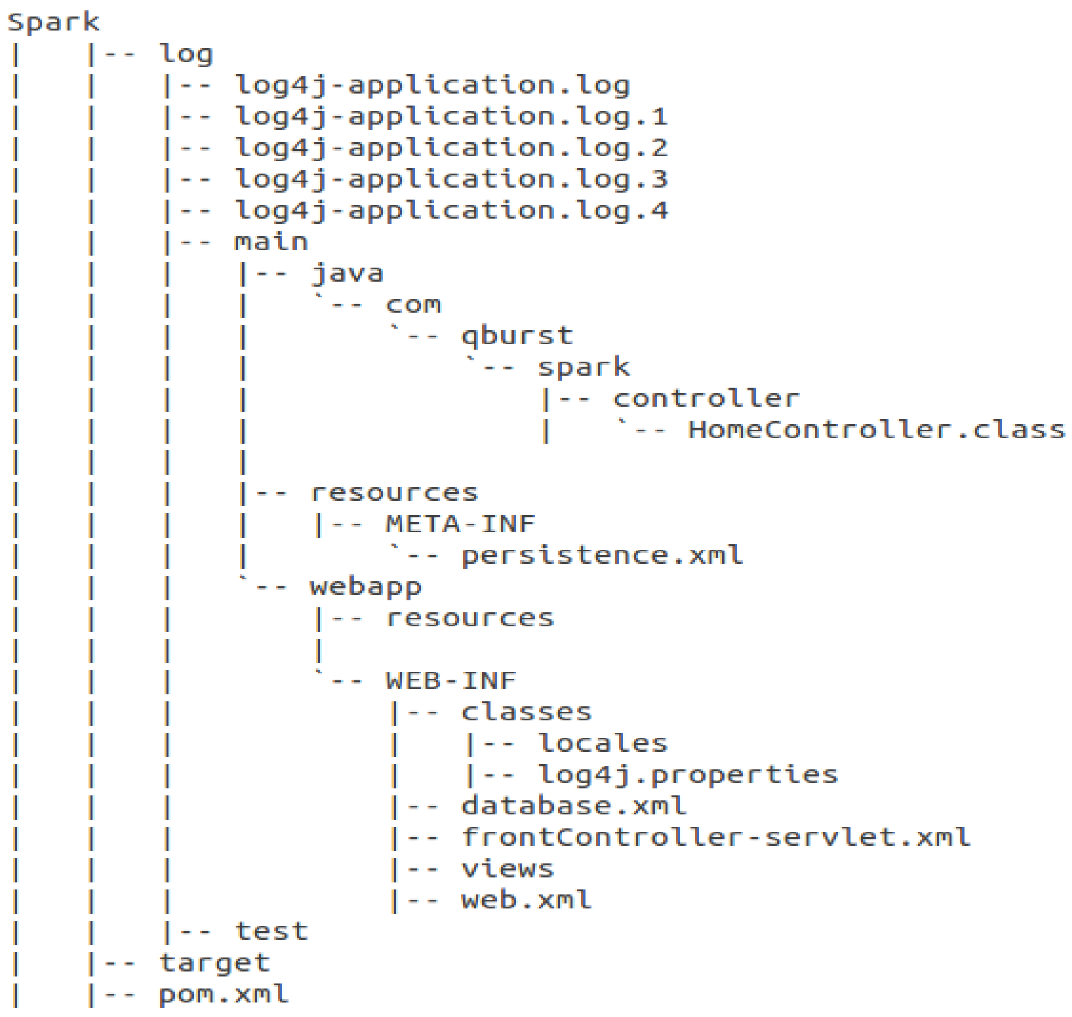
In our application, Logging is done using log4j and slf4j. Slf4j is preferred over commons-logging api.

Logger class object is used to output all messages to .log file located in a folder within the specified directory as specified within log4j.properties file.

Configuration of output directory and log4j output file is specified below.

Configuration :

1. Project Directory



```
Spark
|   |-- log
|   |   |-- log4j-application.log
|   |   |-- log4j-application.log.1
|   |   |-- log4j-application.log.2
|   |   |-- log4j-application.log.3
|   |   |-- log4j-application.log.4
|   |   |-- main
|   |   |   |-- java
|   |   |   |   `-- com
|   |   |   |       `-- qburst
|   |   |   |           `-- spark
|   |   |   |               |-- controller
|   |   |   |               |   `-- HomeController.class
|   |   |   |
|   |   |   |-- resources
|   |   |   |   |-- META-INF
|   |   |   |       `-- persistence.xml
|   |   |   `-- webapp
|   |   |       |-- resources
|   |   |       |
|   |   |       `-- WEB-INF
|   |   |           |-- classes
|   |   |           |   |-- locales
|   |   |           |   |-- log4j.properties
|   |   |           |-- database.xml
|   |   |           |-- frontController-servlet.xml
|   |   |           |-- views
|   |   |           |-- web.xml
|   |   |-- test
|   |-- target
|   |-- pom.xml
```

2. Add the following Depndencies to pom.xml

Pom.xml

```
            <dependency>
                  <groupId>org.slf4j</groupId>
                  <artifactId>slf4j-api</artifactId>
                  <version>${org.slf4j-version}</version>
            </dependency>

            <dependency>
                  <groupId>org.slf4j</groupId>
                  <artifactId>jcl-over-slf4j</artifactId>
                  <version>${org.slf4j-version}</version>
                  <scope>runtime</scope>
            </dependency>

            <dependency>
                  <groupId>org.slf4j</groupId>
                  <artifactId>slf4j-log4j12</artifactId>
                  <version>${org.slf4j-version}</version>
                  <scope>runtime</scope>
            </dependency>
```

3. Create a log4j.properties file, and put it in WEB-INF/classes.

log4j.properties

```
# Root logger option
log4j.rootLogger=INFO, stdout, file

# Redirect log messages to console
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=${user.home}/training_workspace/Spark/log/log4j-
application.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n
```

Attributes of properties file

1. The level of the root logger is defined as **INFO** and configures the appenders: one for console logging and one for file logging (path of the log file is specified as "*${user.home}/training_workspace/Spark/log/log4j-application.log*", `specify the path according to your project directory)`
2. The appender for **file** is defined as **`org.apache.log4j.RollingFileAppender`**. It writes to a file named **`log4j-application.log`** located in the directory specified in properties file. The console output is printed using Console Appender.
3. The layout pattern defined is **`%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n`**, which means the printed logging message will be preceeded by a time stamp and followed by a newline character. The console is written using a simple pattern in which only the messages are printed, but not the more verbose information (logging level, timestamp and so on).
4. The maximum file size is 5MB and maximum backup index is 10.

4. Log the messages using logger

Here a controller class is given to show the use of logger for logging.

HomeController.java

```java
package com.qburst.spark.controller;

import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;
import org.springframework.stereotype.Controller;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Controller
public class HomeController {

private static final Logger logger
=LoggerFactory.getLogger(HomeController.class);

@RequestMapping(value = {"/","/home"}, method = RequestMethod.GET)
      public String home(Locale locale, Model model) {
            logger.info("Welcome home! The client locale is {}.", locale);

            Date date = new Date();
            DateFormat dateFormat =
DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

            String formattedDate = dateFormat.format(date);
            model.addAttribute("serverTime", formattedDate );
            model.addAttribute("message", "welcome");
            return "home";
      }

}
```

Note:

In addition, you can change the component logging levels to any of these:

- •DEBUG : Designates fine-grained informational events that are most useful to debug a crawl configuration.
- •TRACE : Designates fine-grained informational events than DEBUG.
- •ERROR : Designates error events that might still allow the crawler to continue running.
- •FATAL : Designates very severe error events that will presumably lead the crawler to abort.
- •INFO : Designates informational messages that highlight the progress of the crawl at a coarse-grained level.
- •OFF : Has the highest possible rank and is intended to turn off logging.
- •WARN : Designates potentially harmful situations.

These levels allow you to monitor events of interest at the appropriate granularity without being overwhelmed by messages that are not relevant. When you are initially setting up your crawl configuration, you might want to use the DEBUG level to get all messages, and change to a less verbose level in production.