Networks and Database Systems

Professor Tom Ellman
Lecture 13
Logging Web Application Events

- System.out.println writes to a console on the machine where the JVM is running.
- It is not useful for debugging a web application running on a remote server.
- Need facility that writes debugging information to files.
Log4j

- Logging facility from Apache.
- Writes logging information to files.
- Programmer can control:
  - Overall level of detail of logging records.
  - Which parts of server program are logged in greater or lesser detail.
Logger Initialization Servlet

• Only purpose is to initialize loggers.

• Does not process HTTP requests.
private static final String rootLogPath = "/WEB-INF/logs/Root.log";
private static final String entryLogPath = "/WEB-INF/logs/Entry.log";
private static final String exitLogPath = "/WEB-INF/logs/Exit.log";

public void init() {
    FileAppender rootAppender = getAppender(rootLogPath);
    FileAppender entryAppender = getAppender(entryLogPath);
    FileAppender exitAppender = getAppender(exitLogPath);
    initLogger(null, rootAppender, Level.ERROR);
    initLogger("EntryLogger", entryAppender, Level.DEBUG);
    initLogger("ExitLogger", exitAppender, Level.ERROR);
}

LoggerDemo: InitLog4j.java
Creating a FileAppender

```java
private FileAppender getAppender(String fileName) {
    RollingFileAppender appender = null;
    try {
        appender = new RollingFileAppender(
            new PatternLayout("%-5p %c %t%n29d - %m%n"),
            getServletContext().getRealPath(fileName),
            true);
        appender.setMaxBackupIndex(5);
        appender.setMaxFileSize("1MB");
    } catch (IOException ex) {
        System.out.println("Could not create appender for "
            + fileName + ":" + ex.getMessage());
    }
    return appender;
}
```

The file appender of a logger encodes the file to which the messages will be written; the context information to be written with the message, and the format of the logged data.
private void initLogger(String name,
            FileAppender appender,
            Level level) {
    Logger logger;
    if (name == null) {
        logger = Logger.getRootLogger();
    } else {
        logger = Logger.getLogger(name);
    }
    logger.setLevel(level);
    logger.addAppender(appender);
    logger.info("Starting " + logger.getName());
}
Accessing Loggers

public void init() {
    entryLogger = Logger.getLogger("EntryLogger");
    exitLogger = Logger.getLogger("ExitLogger");
}

LoggerDemo: Controller.java
Invoking Loggers

protected void processRequest(HttpServletRequest request,
                               HttpServletResponse response)
    throws ServletException, IOException {

    entryLogger.debug("Entry logger debug message.");
    entryLogger.error("Entry logger error message.");

    // ... Omitted ...

    exitLogger.debug("Exit logger debug message.");
    exitLogger.error("Exit logger error message.");
}

LoggerDemo: Controller.java
Logger Levels

- DEBUG  (Highest level of detail.)
- INFO
- WARN
- ERROR
- FATAL  (Lowest level of detail.)
Use of Logger Levels

• The init() method of the InitLog4j servlet
  – We set the level of the logger.
  – Logger Level.

• In the processRequest() method of the Controller servlet:
  – We invoke loggers using different methods for different levels.
  – Message Level.
Logging Condition

• A logger writes to a message to a file, if and only if the *logger level* is greater than or equal to the *message level*.

• Example1: If logger level is ERROR then fatal and error messages will be written, but debug messages will not be written.

• Example2: If logger level is DEBUG then fatal, error and debug messages will all be written.
Logging Exercise

• Open projects: LoggerDemo, HitCountWebApp.

• Modify HitCountWebApp using LoggerDemo as an Example:
  – Add a servlet (“InitLog4j”) to initialize four loggers in four files.
    • One root logger with level set to ERROR.
    • One logger (“RequestTraceLogger”) with level set to DEBUG.
    • Two loggers (“FastHitLogger”, “SlowHitLogger”) with level set to INFO.
  – Modify Controller servlet:
    • RequestTraceLogger should log .debug messages on entering exiting processRequest method.
    • FastHitLogger should log .info messages with the date and time of every Nth web site hit.
    • SlowHitLogger should log .info messages with the date and time every Mth web site hit.
    • N and M should be initialization parameters to the Controller servlet.

• Test your modified HitCountWebApp.
  – Modify the levels of FastHitLogger & SlowHitLogger and observe results.
  – Modify the level of the root logger and observe what happens.