Logging in Java

Using SLF4J and Logback

Olivier Cailloux

LAMSADE, Université Paris-Dauphine

Version of 12th February, 2017

LAMSADE UMR CNRS 7243
laboratoire d'analyse et modélisation de systèmes pour l'aide à la décision

DAUPHINE UNIVERSITÉ PARIS

PSL RESEARCH UNIVERSITY

What's logging?

- Write out statements about what happens in the program
  - Which parts of the code were executed
  - In what order
  - With what parameters
  - Etc.
- Similar but ≠: temporary debug statements in the code
- Logging (generally) designates:
  - statements that stay in the code
  - and that follow a systematic strategy
- Write to temporary output: ?

What's logging?

- Write out statements about what happens in the program
  - Which parts of the code were executed
  - In what order
  - With what parameters
  - Etc.
- Similar but ≠: temporary debug statements in the code
- Logging (generally) designates:
  - statements that stay in the code
  - and that follow a systematic strategy
- Write to temporary output: ?standard output stream (or standard error stream)
- Or write to persistent output: ?

What's logging?

- Write out statements about what happens in the program
  - Which parts of the code were executed
  - In what order
  - With what parameters
  - Etc.
- Similar but ≠: temporary debug statements in the code
- Logging (generally) designates:
  - statements that stay in the code
  - and that follow a systematic strategy
- Write to temporary output: ?standard output stream (or standard error stream)
- Or write to persistent output: ?file

Why log?

- Debug information for the programmer
- Document your code
- Keep the statements in your code but disable output on demand
- Usually: save time comparing to step-by-step debugging
- Debug hard to reproduce problems
- See what the libraries you use do
- Permit fine selection of what to show
- Example: only see statements related to some computation

Frameworks

- (Too) Many popular logging frameworks in Java
- "Most" standard (JSR 47): Java util logging (JUL)
- But not the best technically
- Here: SLF4J + Logback
- SLF4J?

Frameworks

- (Too) Many popular logging frameworks in Java
- "Most" standard (JSR 47): Java util logging (JUL)
- But not the best technically
- Here: SLF4J + Logback
- SLF4J? Simple Logging Facade for Java
- $\neq$ solutions generally interface well, e.g. Hibernate use JBoss Logging, but can interface with SLF4J

Overview

- Interfaces (and some basic code) are in SLF4J
- You depend on `slf4j-api-xxx.jar`
- Permit to declare and invoke loggers in your code
- Declare a logger LOGGER in your class
- Invoke with `LOGGER.debug(String)`, `LOGGER.info(String)`...
- At run time: depend on a logger provider (an SLF4J binding)
- E.g. *logback*
- Provide a configuration file
- Specifies what and where to log

Declare logger

- Loggers have names
- Permit to classify statements
- Typical idiom: logger name = class name
- Logger for a given class is in field `private static final Logger LOGGER`
- Because: no need to distinguish loggers per instance
- Obtain the logger using `LoggerFactory.getLogger(MyClass.class);`
- Advantage wrt `LoggerFactory.getLogger("app.pack1.MyClass")`?

Declare logger

- Loggers have names
- Permit to classify statements
- Typical idiom: logger name = class name
- Logger for a given class is in field `private static final Logger LOGGER`
- Because: no need to distinguish loggers per instance
- Obtain the logger using `LoggerFactory.getLogger(MyClass.class);`
- Advantage wrt `LoggerFactory.getLogger("app.pack1.MyClass")`?
  Refactoring: explicit link to class

Declare logger

- Loggers have names
- Permit to classify statements
- Typical idiom: logger name = class name
- Logger for a given class is in field `private static final Logger LOGGER`
- Because: no need to distinguish loggers per instance
- Obtain the logger using `LoggerFactory.getLogger(MyClass.class);`
- Advantage wrt `LoggerFactory.getLogger("app.pack1.MyClass")`? Refactoring: explicit link to class

```
@SuppressWarnings("unused")
private static final Logger LOGGER =
  LoggerFactory.getLogger(MyClass.class);
```

Log statements

Normal use of a LOGGER:

```
logger.debug("Temperature set to {}.
    Old temperature was {}.", t, oldT);
```

Log an exception:

```
String s = "Hello world";
try {
  Integer i = Integer.valueOf(s);
} catch (NumberFormatException e) {
  logger.error("Failed to format {}", s, e);
}
```

SLF4J bindings

- Multiple bindings exist
- At runtime, have only one binding
- A set of classes that bind to SLF4J
- Put the right jar in the classpath
- Example, *Simple* binding: Sends all log statements to `System.err`
- Or JDK14: sends log statements to JUL
- We will rather use *Logback*

Logback

- The main implementation for SLF4J
- No configuration file: outputs logs to `System.out`
- Much flexibility with configuration file(s)
- Put into the classpath
- Name: `logback.xml` or `logback.groovy` or `logback-test.xml`
- Configure: Logger, Appender and Layouts
- To enable and direct specific statements
- can also configure programatically

Logger level

- Loggers in a hierarchy: dot-separated
- Every logger may have an *assigned* log level (or `null`) root logger always has an assigned log level, default is DEBUG
- Every logger has an *effective* log level
- Used to filter received logs
- Example: a log request of level DEBUG is not sent to a logger with effective level INFO
- Log level not configured explicitly ⇒ inherited from parent
- If root has level DEBUG and no other is configured: all DEBUG
- If root has level DEBUG and X.Y has level INFO, X has effective level?

Logger level

- Loggers in a hierarchy: dot-separated
- Every logger may have an *assigned* log level (or `null`) root logger always has an assigned log level, default is DEBUG
- Every logger has an *effective* log level
- Used to filter received logs
- Example: a log request of level DEBUG is not sent to a logger with effective level INFO
- Log level not configured explicitly ⇒ inherited from parent
- If root has level DEBUG and no other is configured: all DEBUG
- If root has level DEBUG and X.Y has level INFO, X has effective level? DEBUG; X.Z?

Logger level

- Loggers in a hierarchy: dot-separated
- Every logger may have an *assigned* log level (or `null`) root logger always has an assigned log level, default is DEBUG
- Every logger has an *effective* log level
- Used to filter received logs
- Example: a log request of level DEBUG is not sent to a logger with effective level INFO
- Log level not configured explicitly ⇒ inherited from parent
- If root has level DEBUG and no other is configured: all DEBUG
- If root has level DEBUG and X.Y has level INFO, X has effective level? DEBUG; X.Z? DEBUG; X.Y.Z? INFO

Appenders

- Appenders attached to a logger
- Indicate where to log
- Its children inherit the appenders
- Can log to console, a file, a DB, a remote server...

License