Logging in Java

Overview and HowTo

Erik Fäßler

Jena Language & Information Engineering (JULIE) Lab
Friedrich-Schiller-Universität Jena, Germany

http://www.julielab.de/

Logging - what for?

Logging via `System.out` has a range of disadvantages

- Only two channels (STDOUT and, via `System.err`, STDERR)
- Logging to file via output redirection
- Always all log messages are printed
- Hardcoded, not configurable

Logging Frameworks for Java (incomplete)

- Java Logging
- log4j
- log4j2
- Logback

Core Principles

- Logging levels
  - Java Logging: ALL, FINEST, FINER, FINE, CONFIG, INFO, WARNING, SEVERE, OFF
  - Log4j, Logback: ALL, TRACE, DEBUG, INFO, WARN, ERROR, FATAL, OFF
- Loggers are named
  - de.julielab.jcore.ae.jsbd.SentenceAnnotator
  - Kaesekuchen
  - org.apache
- Logger names are *hierarchical*
  de.julielab.jcore.ae.jsbd.SentenceAnnotator < de.julielab.jcore.ae.jsbd < de.julielab.jcore.ae < . . . < de < ROOT
- Log level of a Logger is *inherited* by their descendants

  https://logback.qos.ch/manual/architecture.html#effectiveLevel

In the Code I

- Java Logging:

```
Logger log =
  Logger.getLogger(MyClass.class.getName());
log.log(Levels.FINE, "Writing␣data␣to␣file");
```

- Log4j:

```
Logger log =
  Logger.getLogger(MyClass.class);
log.info("Writing␣data␣to␣file");
```

- Logback:

```
Logger log =
  LoggerFactory.getLogger(MyClass.class);
log.debug("Writing␣data␣to␣file");
```

In the Code II

Typical declaration point of a logger: private static final field of the class, e.g. Logback:

```
public class MyClass {

  private final static Logger log =
    LoggerFactory.getLogger(MyClass.class);

  public void writeFile(File file) {
    log.info("Writing data to file");
  }

}
```

Code Example I

- Example project: https://www.coling.uni-jena.de/svn/coling/trunk/praktikum/praktikum-logger
- Class: de.julielab.praktikum.logger.LoggerExample

Implementation Independence

- Everyone has his or her favourite logger framework
- They all come with different ways of configuration
- Yet they do very similar things
- Solution: Intermediary API layer
  - Apache Commons Logging
  - *SLF4J* – this is what we use!

Simple Logging Facade for Java (SLF4J)

- To use SLF4J, your project depends on the *slf4j-api* and an implementation, or *binding*
- Examples of bindings
    - slf4j-log4j12
    - slf4j-jdk14
    - slf4j-simple *Simple implementation, which outputs all events to System.err. Only messages of level INFO and higher are printed. This binding may be useful in the context of small applications.*
    - slf4j-jcl
    - logback-classic *Erik's favourite*

How to use and best practices

- The code looks exactly the same as in the Logback examples since Logback is a native SLF4J logger
- SLF4J and Logback are both managed by our julielab-parent-pom. Please look at the examples on how to fetch the Maven dependencies.
- To switch to another Logger implementation, just define the dependency to another SLF4J binding
- Best Practices
  - Always use SLF4J as a dependency
  - Never (!!) put a compile dependency for a logger binding on a library
    - This would import the logger binding to dependent project via transitive dependency resolution
    - Use either the Maven test scope (libraries) or provided scope (applications that still could be used as a library)

Configuration Basics

- From now on, it is assumed we are working with Logback
- Configuration can be done in two ways
  - Programatically (see first code example on how to set the logger level, for example)
  - Much better: configuration file
- Logback checks, in this order, for files in a *classpath directory*
  - logback-test.xml
  - logback.groovy
  - logback.xml
  - service-provider loading facility

Example Logback Configuration File

Reference: https://logback.qos.ch/manual/configuration.html

```
<configuration>
  <appender name="STDOUT"
    class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>
        %d{HH:mm:ss} %-5level %logger{36} - %msg%n
      </pattern>
    </encoder>
  </appender>

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

Code Example II

- Example project: https://www.coling.uni-jena.de/svn/coling/trunk/praktikum/praktikum-logger
- Class: de.julielab.praktikum.logger.BasicConfigurationExample

Appenders

Reference: https://logback.qos.ch/manual/appenders.html

- Appenders are channels where logging messages can be directed to
  - STDOUT, STDERR
  - Files
  - Databases
  - Network connections
- Each appender might be added to multiple loggers
- Appender additivity: Appenders are accumulated down the logger hierarchy – each logger writes to the appenders of its ancestors

Code Example III

- Example project: https://www.coling.uni-jena.de/svn/coling/trunk/praktikum/praktikum-logger
- Class: de.julielab.praktikum.logger.ComplexConfigurationExample

Code Example IV: Packaging Data

By specifying `packagingData="true"` to the `<configuration>` element of the configuration file, exception stack traces show class file source (e.g JAR packages)

- Example project: https://www.coling.uni-jena.de/svn/coling/trunk/praktikum/praktikum-logger
- Class: de.julielab.praktikum.logger.PackagingData