

Logging in Java

Dominik Grether
Bergell 07

Logging Motivation

- Executing code calling method

```
QueueLink.recalcCapacity(...)
```

- Output:

```
link 25700 too small: enlarge spaceCap  
link 25701 too small: enlarge spaceCap  
link 25734 too small: enlarge spaceCap  
link 25735 too small: enlarge spaceCap  
link 25756 too small: enlarge spaceCap  
link 25757 too small: enlarge spaceCap  
link 25758 too small: enlarge spaceCap  
link 25759 too small: enlarge spaceCap  
link 25760 too small: enlarge spaceCap  
link 25761 too small: enlarge spaceCap  
...
```

Logging Motivation (2)

- **Problems:**
 - Hard to read debug messages above
 - No filtering of messages
 - No information like Class and time
- **Solutions:**
 - Writing an own logging framework
 - Using existing logging framework
- **Principle of software development:**

Writing own code is mostly worse than using existing framework

Java Logging API Introduction

- Java provides API for logging
- Cheap costs, i.e. outsourcing expensive operations:
 - Localization
 - Formatting
- Several log levels and loggers
- Pure OO-Design (Logger objects)

Java Logging API

- Hierarchical organized log levels, e.g.:
 - Info
 - Warning
 - Config
- Log method for each level:

```
void warning(String msg);  
void info(String msg);  
...
```

Java Logging API (2)

- Several handlers:
 - StreamHandler
 - ConsoleHandler
 - FileHandler
 - SocketHandler
 - MemoryHandler
- 2 standard formatters:
 - SimpleFormatter
 - XMLFormatter

Java Logging API Configuration

- 2 ways to configure Java logging

- Directly in code:

```
loggerObject.setLevel(Logger.INFO)
```


- By standard java property configuration File

- Configuration by file makes logging more transparent

Log4j Introduction

- Mostly used Java logging framework
- Similar to java logging API
- Simple configuration and usage
- Low costs
- Very flexible

Log4j vs. Java Logging API

- Advantages of Log4j:
 - Simpler log level hierarchy and log methods
 - More handlers
 - Mighty formatters
 - Optimized logging costs
 - Simpler log level hierarchy:
 - Trace
 - Debug
 - Info
 - Warn
 - Error
 - Fatal
- 

Log4j - Java Logging

```
public class Test {  
    private static final Logger log =  
        Logger.getLogger(Test.class);  
  
    public Test () {  
        log.debug(„a debug message“);  
        log.info(„an info message“);  
        log.warn(„a warning message“);  
        log.error(„an error message“);  
        log.fatal(„a fatal message“);  
    }  
}
```

Log4j - Java Logging Output

```
2007-10-10 22:48:08,221 DEBUG Test:10 a debug message
2007-10-10 22:48:08,221 INFO Test:11 an info message
2007-10-10 22:48:08,222 WARN Test:12 a warning message
2007-10-10 22:48:08,254 ERROR Test:13 an error message
2007-10-12 17:45:40,161 FATAL Test:14 a fatal message
```

Log4j - Advanced logging

- Problem with logging output: String concatenation very expensive
- Solution in log4j:

```
if (log.isDebugEnabled()) {  
    log.debug(„this“ + „is“ + „a “ + 9  
        + „times“ + „more“ + „expensive“  
        + „operation“ + „than“ + „it“  
        + „should“ + „be“);  
}
```

Log4j - Configuration files

- Log4j configuration in configuration file
- log4j.properties simple Java properties file

```
# Set root logger level to DEBUG
log4j.rootLogger=DEBUG, A1
# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
#log4j.appender.A1.layout.ConversionPattern
#   =%-4r [%t] %-5p %c %x - %m%n
log4j.appender.A1.layout.ConversionPattern
#   =%-6r [%15.15t] %-5p %30.30c %x - %m%n
log4j.appender.A1.layout.ConversionPattern
#   =%d{ISO8601} %5p %C{1}:%L %m%n
```

Conclusion

- Proposal: Use Log4j instead of System.out
 - log4j.properties in CVS
 - Add lib/log4j.jar
- References:
 - <http://java.sun.com/j2se/1.5.0/docs/guide/logging/overview.html>
 - <http://logging.apache.org/log4j/1.2/index.html>