

Log4J & Log4Cocoa

*For those rare times it's
not a hardware problem*

Bob Frank
<http://cawug.org>

Jonathan 'Wolf' Rentzsch
<http://rentzsch.com>

What is Logging?

- The act of inserting statements into your code to trace execution
- Low-tech
- **Earliest** technique for debugging
- **Easiest** technique for debugging

Why Logging?

- Debugging
 - debugger not always available
 - debugging sessions are transient
- Self Documenting Code
- Provides context from a specific run
- Log message generation persist

Why Logging?

- Logging Philosophy:

Adding meaningful logging at critical places causes more thinking than just stepping through code looking for wrong values and will produce better code.

- paraphrased from Kernighan & Pike

- (see Log4J site for full quote)

What is Log4J?

- Part of the Apache Software Foundation's Jakarta project
- Distributed under a BSD-style license
- Best logging package I've seen
- Fast, flexible, yet easy to use

What's Wrong with System.out.println()?

- **What isn't?**
- Wordy
- Either on or off at compile time
 - must comment or delete to remove
- WYPIWYG: What you print is what you get
 - Must explicitly add contextual info

Why Log4J?

- Log4J is better than any println()
- Control, *at runtime*, where output goes:
 - Console, File, syslog, NT Event Logger...
 - Some or all of the above
 - or write your own
 - (voice mail, electroshock the user, etc.)

Log4J is Better

- Less wordy:
 - `log.debug("my debug log message")`
 - Means you're more likely to use it
- Categorizes all output by priority:
 - debug, info, warn, error, fatal

Log4J is Better

- Categorizes all output by class:
 - typically one Logger per class
 - Loggers used to be called Categories
- You can, at runtime, collect all logging classes and turn them on or off
- You can also filter out by class, by priority

Log4J is Better

- Logging can get quite sophisticated
 - loggers, different levels, etc ...
 - can all be configured to go to various locations
- Caution: Scrolling Blindness
 - can be caused by turning on all loggers

Log4J Rocks Because:

- It's Easy

- `log.debug("my debugging statement");`

- It's Popular

- Ported to: C, C++ (3), Eiffel, Perl, LotusScript, .Net, PHP, Python (2), Ruby

- ...and now, Objective-C
(Bob's Log4Cocoa segment)

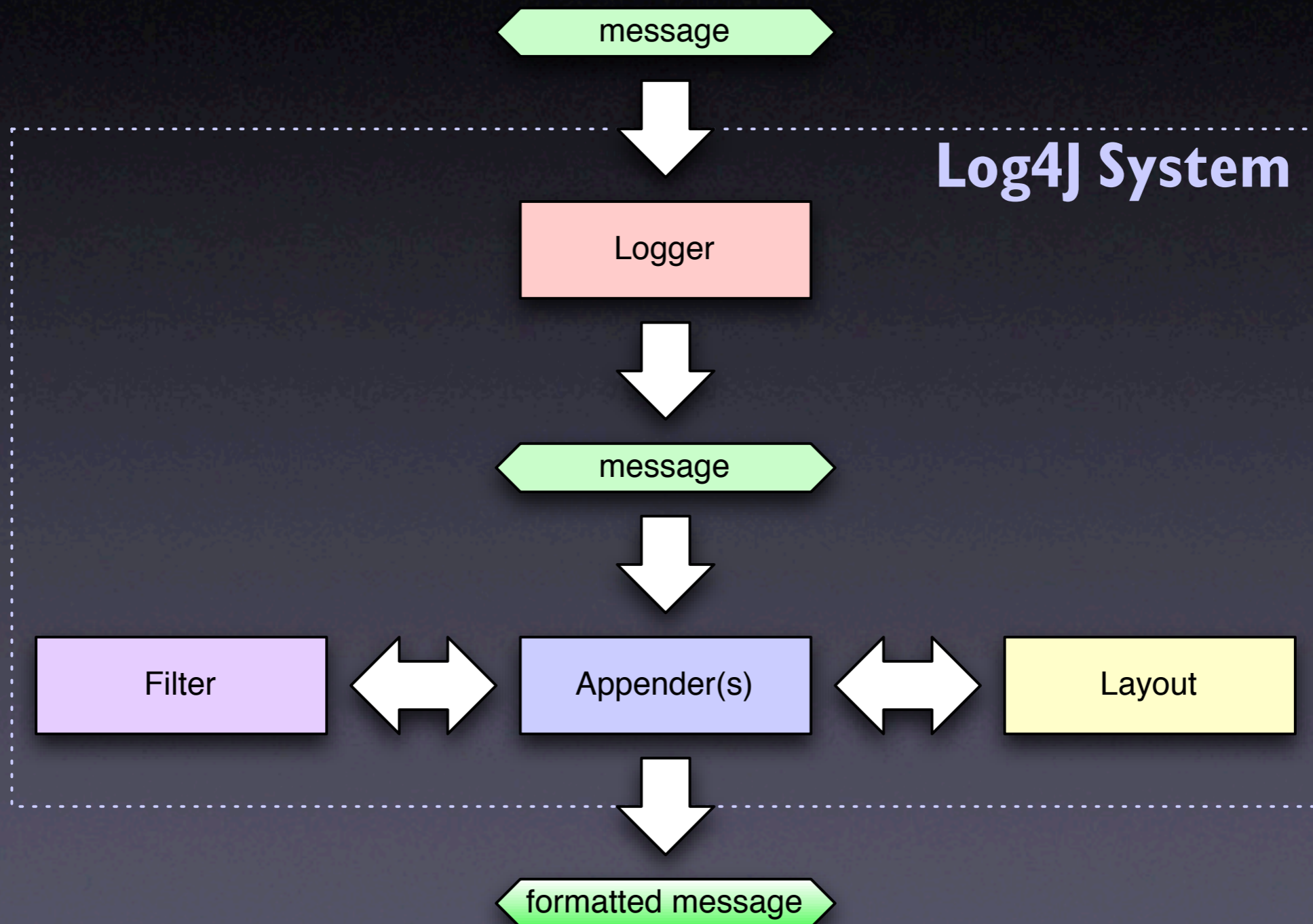
Log4J Rocks Because:

- It's Flexible. Five primary abstractions:
 - Levels
 - Loggers
 - Appenders
 - Filters
 - Layouts

Log4j Rocks Because:

- It's Fast
 - 233 MHz Thinkpad
 - Eval & skip: 46 nanoseconds
 - Eval & print: 79 to 164 nanoseconds

Log4j Overview



Log4j Details: Levels

- Log4j defines five different logging "levels".
In increasing severity:
 - `Level.DEBUG` i.e. "foo() was called"
 - `Level.INFO` i.e. "processing XML file"
 - `Level.WARN` i.e. "config file missing, using defaults."
 - `Level.ERROR` i.e. "an exception was thrown"
 - `Level.FATAL` i.e. "data corruption discovered, exiting"

Log4J Details: Loggers

- Loggers are objects which log messages
- Usually one Logger per class
 - most intuitive usage pattern
 - can create any usage pattern for your app
 - ex. cross class functional loggers
(sqlLogger - to log all database access)

Log4j Details: Loggers

```
class MyClass {  
    private static Logger log =  
Logger.getLogger(MyClass.class);  
  
    public void foo() {  
        log.debug("foo() was called");  
    }  
}
```

Log4j Details: Loggers

- Tree-like hierarchy
 - root logger always exists
 - tree structure allows for high performance.
 - maps naturally to object class hierarchies

Log4J Details: Loggers

- Each logger has an "assigned level"
 - when a message is logged, its level is compared against the logger's assigned level
 - if the message's severity level is equal to or greater than the message is logged

Log4j Details: Loggers

- Level examples:
 - `log.debug()` when `log.level == Level.DEBUG`
-> LOGGED
 - `log.debug()` when `log.level == Level.WARN`
-> NOT LOGGED
 - `log.warn()` when `log.level == Level.DEBUG`
-> LOGGED

Log4j Details: Loggers

- Loggers "inherit" an assigned level if not specifically assigned
- 'root' Logger must have a level

Log4j Details: Appenders

- Responsible for outputting log messages
- Loggers accept your messages and hand them over to Appenders for actual logging
- Lots of prewritten appenders
 - Console, File, rolling Files, JDBC, SMTP, Syslog, etc.

Log4j Details: Appenders

- Easy to write your own
- Can have multiple appenders active simultaneously.

Log4j Details: Filters

- You can have additional filters on log messages
- Example: different destinations of same kind
 - error file vs. warn file

Log4j Details: Layouts

- What the Appender calls to format your debugging message.
- one layout per appender
 - some appenders have built-in layouts (i.e. `socketAppender`)

Log4j Details: Layouts

- "PatternLayout" is most commonly used
 - printf-style formatting
 - outputs a wealth of information
- HTML and XML layouts also offered

Log4j Details: Layouts

- Custom Object Renderers
 - Instead of:
 - `log.info(myCustomer.someInfo());`
 - you could write:
 - `log.info(myCustomer);`

JDK Logger

Compatibility changes

- Some Log4j changes for 1.4 compatibility:
- “Category” class renamed “Logger”
- Two “meta” logging levels were added
 - “all” & “off”
 - **all** < debug < info < warn < error < fatal < **off**

Log4J vs. Java 1.4 Logging API

- Sadness
- Sun didn't just rubber-stamp the community-created code
- JDK Logging not as good as Log4J, but superficially almost the same
- Log4J works back to Java 1.1
- JDK requires 1.4

Log4J vs. Java 1.4 Logging API

- Jakarta "Commons" project offers logging-package agnostic wrapper classes
 - Log4J
 - JDK 1.4 logging
 - Avalon

Log4J Sore Spots

- While logging isn't expensive, log message-building can be
- ```
log.debug(
 "bigComplicatedDatabaseObject:
 "+bigComplicatedDatabaseObject);
```

# Log4J Sore Spots

- May need to wrap logging statements for efficiency:
- ```
if (log.isDebugEnabled())  
    log.debug(  
        "bigComplicatedDatabaseObject: "  
        +bigComplicatedDatabaseObject );
```

Log4J Sore Spots

- Default property-file based configuration syntax is necessarily weird
- XML is a better match, but requires explicit call to DOMConfigurator at startup

Demos

Log4Cocoa

- Hosted on SourceForge, why?
 - Easy
 - Public
 - Well known
 - <http://log4cocoa.sourceforge.net/>

License

- BSD
- Because its most open

Current Log4Cocoa Status

- Beta
- Actively Porting:
 - Appenders
 - Configuration code
- Unit Testing
 - using ObjcUnit

Porting Experience

- Architecture Port
 - initially: strict conformance to Log4J
 - now: adapting to Cocoa environment
 - more "Cocoa-esque"

Porting Experience

- GCC pre-processor / macros
 - makes things easy that are hard in Java
 - can completely disable at compile time
 - capture location info (line #, method name, etc.)
- var-args

Categories (yea!)

- A favorite tool of Cocoa programmers
- Class extension without subclassing
- You can add methods to other classes
 - powerful programming tool
 - security issues (rare)

Default Configuration

- Log4J
 - fails if not configured
- Log4Cocoa
 - has some default behaviors

Performance

- Eval & Log: 2.5x slower than NSLog()
- Eval & Skip: > 900x faster
- Compare w/ Log4J
 - skip vs. print 2-3x slower

Todo list

- Current Beta Unoptimized
 - few orders of magnitude slower than $\text{Log}_4 J$
 - Substantial opportunity for improvements
- more appenders
- multi-threading support

Todo list cont.

- more layouts & test cases
- configuration options
 - plist, defaults, xml files, database, etc.
- GUI log managers
 - (i.e. chainsaw / LogFactor5)
 - .nib in a package

Future Ideas

- Scripts
 - auto add log debug statements to methods
- Cocoa like config options

Demos

Q & A