



Java 日志框架培训

Java 编程技巧培训之一

郝金隆 (haojinlong@189.cn)

2014 年 6 月



1

Java 日志框架概述

2

Java 日志接口框架

3

常用Java 日志实现实现

4

其他日志使用技巧

从 System.out.println 说起



- 在 jar 编程的学习过程中，大家学到的第一句 java 语言绝大多数应该就是 System.out.println()。但是在实际的项目过程中，不可能通过 System.out.println() 来记录系统的日志，主要原因如下：
 - 无法控制是否要输入
 - 输出到控制台的日志难以查询
 - 只有输出内容，没有日志输出位置、时间等信息，大型项目分析困难
 - 无法定义日志级别
 - 导致大量的系统开销
 -
- 为此 Apache 基金会发起了 log4j 项目，提供了灵活而强大的日志记录机制，而后 jdk1.4 吸收了 log4j 的思想，发布了第一个日志接口，并提供了一个简单的实现；
- 后来为了使得 Java 项目能够随意进行切换日志实现，common-logging 应运而生
- 在往后 logback、log4j2、slf4j 等应运而生

java 日志的基本概念（引自 log4j）



- 日志的级别（LEVEL），从低到高依次为：
 - DEBUG（常用）：指出细粒度信息事件对调试应用程序是非常有帮助的
 - INFO：突出强调应用程序的运行过程
 - WARN（常用）：表示会出现潜在的错误
 - ERROR（常用）：指出虽然发生错误事件，但仍然不影响系统的继续运行
 - FATAL（slf4j中取消）：表示严重的错误事件将会导致应用程序的退出
- 日志的输出方式（APPENDER）
 - ConsoleAppender：控制台，常用于开发过程中
 - FileAppender：文件
 - RollingFileAppender：大小受限的日志文件，常用于运行过程中
 - DailyRollingAppender：按日期记录的日志文件，常用于运行过程中
 -
- 日志的布局（Layout）：SimpleLayout、PatternLayout（常用）等



1

Java 日志框架概述

2

Java 日志接口框架

3

常用Java 日志实现实现

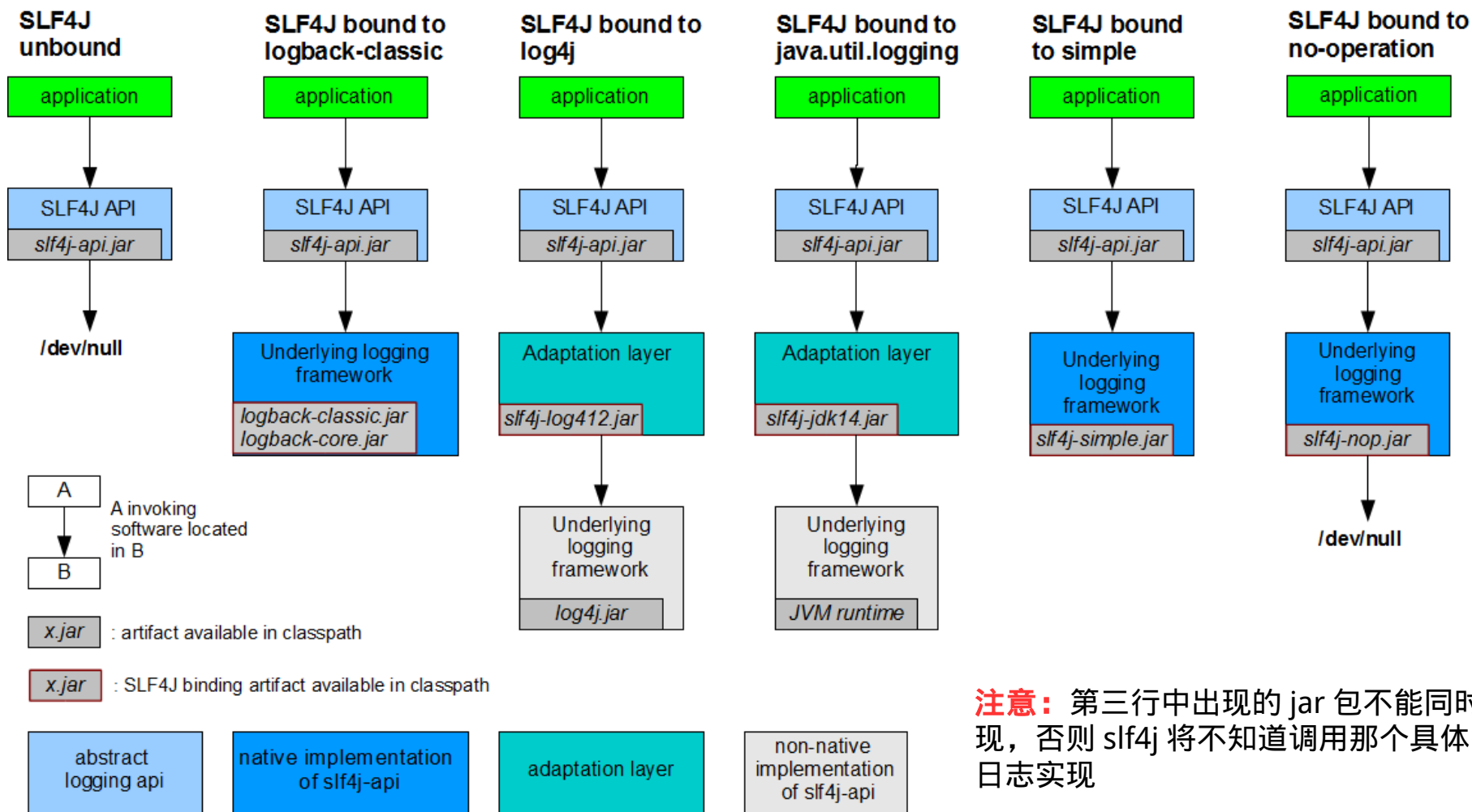
4

其他日志使用技巧



- 常用的 java 日志接口框架
 - Common-logging
 - apache 的一个子项目，最早的日志接口框架，通过动态查找的机制，使用 ClassLoader 寻找和载入底层的日志库，在运行过程中自动选择日志框架实现，如有有 log4j 就默认使用 log4j，否则使用 jdk 内部的日志实现：
 - <http://commons.apache.org/proper/commons-logging/index.html>
 - slf4j（建议框架）
 - log4j 作者写的一个日志接口框架，相对于 common-logging，取消了 fatal 日志级别（实际使用过程中没有意义），并且采用了静态绑定的方式来指定日志框架的实现，支持参数化的 log 字符串，可选多种其他的日志框架，如 logback，log4j、log4j2 等等，并且可以通过引入一个 jar 包、去除 common-logging 的 jar 包来覆盖 common-logging 接口
 - <http://www.slf4j.org/>

slf4j 与日志实现的对应关系



注意：第三行中出现的 jar 包不能同时出现，否则 slf4j 将不知道调用那个具体的日志实现



1

Java 日志框架概述

2

Java 日志接口框架

3

常用 Java 日志实现实现

4

其他日志使用技巧

常见的 java 日志实现： log4j



- log4j：最早的 java 日志实现框架
 - 最新版本： 1.2.17
 - 官方下载地址：
<http://www.apache.org/dyn/closer.cgi/logging/log4j/1.2.17/log4j-1.2.17.zip>
 - 所需要的 jar 包
 - log4j-1.2.17.jar
 - 与 common-logging 集成所需的 jar 包
 - 无
 - 与 slf4j 集成所需的 jar 包
 - slf4j-log4j12.jar
 - 配置文件
 - \${CLASSPATH}/log4j.properties
-

常见的 java 日志实现： log4j 配置示例



```
# - log4j.properties --  
# 全局日志级别为 ERROR，输出到 A2  
log4j.rootLogger=ERROR, A2  
  
# pkg1 包下日志级别为 DEBUG，输出到 A1  
log4j.logger.com.github.haojinlong.training.jcl.log4j.pkg1=DEBUG, A1  
  
# A1 设置为输出到控制台，并设置 A1 的输出格式  
log4j.appender.A1=org.apache.log4j.ConsoleAppender  
log4j.appender.A1.layout=org.apache.log4j.PatternLayout  
log4j.appender.A1.layout.ConversionPattern=%d %-5p [%t] %-17c{2} (%13F:%L) %3x - %m%n  
  
# 设置 A2 输出到滚动文件，并设置文件地址和大小  
log4j.appender.A2=org.apache.log4j.RollingFileAppender  
log4j.appender.A2.File=/home/hjl/test.log  
log4j.appender.A2.MaxFileSize=2MB  
  
# 设置 A2 的输出格式  
log4j.appender.A2.layout=org.apache.log4j.PatternLayout  
log4j.appender.A2.layout.ConversionPattern=%d %-5p [%t] %-17c{2} (%13F:%L) %3x - %m%n
```

log4j 输出格式说明



1. -X 号: X 信息输出时左对齐。
2. %p: 输出日志信息优先级, 即 DEBUG, INFO, WARN, ERROR, FATAL。
3. %d: 输出日志时间点的日期或时间, 默认格式为 ISO8601, 也可以在其后指定格式, 比如: %d{yyy MMM dd HH:mm:ss,SSS}, 输出类似: 2002 年 10 月 18 日 22:10:28, 921。
4. %r: 输出自应用启动到输出该 log 信息耗费的毫秒数。
5. %c: 输出日志信息所属的类目, 通常就是所在类的全名。
6. %t: 输出产生该日志事件的线程名。
7. %l: 输出日志事件的发生位置, 相当于 %C.%M(%F:%L) 的组合, 包括类目名、发生的线程, 以及在代码中的行数。举例: Testlog4.main(TestLog4.java:10)。
8. %x: 输出和当前线程相关联的 NDC(嵌套诊断环境), 尤其用到像 java servlets 这样的多客户多线程的应用中。
9. %%: 输出一个 "%" 字符。
- 10.1%F: 输出日志消息产生时所在的文件名称。
- 11.%L: 输出代码中的行号。
- 12.%m: 输出代码中指定的消息, 产生的日志具体信息。
- 13.1%n: 输出一个回车换行符, Windows 平台为 "\r\n", Unix 平台为 "\n" 输出日志信息换行。
- 14.可以在 % 与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如:
 - %20c: 指定输出 category 的名称, 最小的宽度是 20, 如果 category 的名称小于 20 的话, 默认的情况下右对齐。
 - %-20c: 指定输出 category 的名称, 最小的宽度是 20, 如果 category 的名称小于 20 的话, "-" 号指定左对齐。
 - %.30c: 指定输出 category 的名称, 最大的宽度是 30, 如果 category 的名称大于 30 的话, 就会将左边多出的字符截掉, 但小于 30 的话也不会有空格。
 - %20.30c: 如果 category 的名称小于 20 就补空格, 并且右对齐, 如果其名称长于 30 字符, 就从左边交远销出的字符截掉。

常见的 java 日志实现： log4j2



- log4j2 : log4j 的第二代
 - 最新版本： 2.0-rc2
 - 官方地址： <http://logging.apache.org/log4j/2.x/>
 - 所需要的 jar 包
 - log4j-api-2.0-rc2.jar
 - log4j-core-2.0-rc2.jar
 - 与 common-logging 集成所需的 jar 包
 - log4j-jcl-2.0-rc2.jar
 - 与 slf4j 集成所需的 jar 包
 - log4j-slf4j-impl-2.0-rc2.jar
 - 配置文件
 - \${CLASSPATH}/log4j2.xml

常见的 java 日志实现： log4j2 配置示例



```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="OFF">
  <appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout
        pattern="%d{yyyy.MM.dd 'at' HH:mm:ss.SSS} %-5level %class{3} %L %M - %msg%xEx%n" />
      </Console>
    <RollingFile name="RollingFile" fileName="/home/hjl/app.log"
      filePattern="/home/hjl/${date:yyyy-MM}/app-%d{yyyy-MM-dd}-%i.log.zip">
      <PatternLayout
        pattern="%d{yyyy.MM.dd 'at' HH:mm:ss z} %-5level %class{3} %L %M - %msg%xEx%n" />
      <SizeBasedTriggeringPolicy size="50 K" />
    </RollingFile>
  </appenders>
  <loggers>
    <logger name="com.github.haojinlong.training.jcl.log4j2.pkg1.LogTest1"
      level="debug" additivity="false">
      <appender-ref ref="Console" />
    </logger>
    <root level="error">
      <appender-ref ref="RollingFile" />
    </root>
  </loggers>
</configuration>
```

常见的 java 日志实现： logback



- logback : log4j 作者写的新一代日志框架
 - 最新版本： 1.1.2
 - 官方地址： <http://logback.qos.ch/>
 - 所需要的 jar 包
 - logback-core-1.1.2.jar
 - logback-classic-1.1.2.jar
 - 与 common-logging 集成所需的 jar 包
 - 貌似没有找到合适的集成方式
 - 与 slf4j 集成所需的 jar 包
 - 无，直接集成
 - 配置文件
 - \${CLASSPATH}/logback.xml

常见的 java 日志实现： logback 配置示例



```
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d %-5level %logger{35} %L - %msg %n</pattern>
    </encoder>
  </appender>
  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>/home/hjl/app.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>/home/hjl/app-%d{yyyy-MM-dd}-%i.log.zip</fileNamePattern>
      <timeBasedFileNamingAndTriggeringPolicy
        class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
          <maxFileSize>5MB</maxFileSize>
        </timeBasedFileNamingAndTriggeringPolicy>
      </rollingPolicy>
      <encoder>
        <pattern>%d %-5level %logger{35} %L - %msg %n</pattern>
      </encoder>
    </appender>
  <root level="ERROR">
    <appender-ref ref="FILE" />
  </root>
  <logger name="com.github.haojinlong.training.slf4j.logback.pkg1.LogTest1" level="debug" additivity="false">
    <appender-ref ref="STDOUT" />
  </logger>
</configuration>
```



1

Java 日志框架概述

2

Java 日志接口框架

3

常用Java 日志实现实现

4

其他日志使用技巧



- 不要直接使用语言拼接，就 `logger.debug(a+b)` 这种方式
 - 会造成不必要的性能开销
 - 系统会首先将 `a` 和 `b` 进行拼接，生成字符串，然后在执行 `debug` 方法，也就是说字符串的拼接会在判断日志级别以前
- 建议的使用方式
 - 进行日志输出以前，首先调用 `logger.isDebugEnabled()` 方法进行判断
 - `if(logger.isDebugEnabled())`
 - `{`
 - `logger.debug(a+b)`
 - `}`
 - 使用参数化的字符串（`commons-logging` 不支持，且参数不能超过 3 个，否则需要转化成数组）
 - `logger.debug("{} {}" , a, b)`

使用技巧一：性能测试



测试一（不判断级别）

```
String sayHello = "hello";
String name = "Haojinlong";
long beginTM = System.currentTimeMillis();
for (int i = 0; i <= 1000000; i++) {
    logger.debug(sayHello + " " + name
        + "!");
}
long endTM = System.currentTimeMillis();
System.out.println(endTM - beginTM);
```

输出结果：

145

测试二（变参）

```
String sayHello = "hello";
String name = "Haojinlong";
long beginTM = System.currentTimeMillis();
for (int i = 0; i <= 1000000; i++) {
    logger.debug("{} {}", sayHello,
        name);
}
long endTM = System.currentTimeMillis();
System.out.println(endTM - beginTM);
```

输出结果：

16

测试三（先判断级别）

```
String sayHello = "hello";
String name = "Haojinlong";
long beginTM = System.currentTimeMillis();
for (int i = 0; i <= 1000000; i++) {
    if (logger.isDebugEnabled()) {
        logger.debug(sayHello + " " +
            name + "!");
    }
}
long endTM = System.currentTimeMillis();
System.out.println(endTM - beginTM);
```

输出结果：

15

注：完整版本代码参见附录中代码示例：6-slf4j-logback-efficient

使用技巧二：协同 commons-lang3 使用



- commons-lang3
 - 介绍：apache 下的子项目，提供针对核心 java 类的一些额外的操作
 - 地址：<http://commons.apache.org/proper/commons-lang/>
 - 最新版本：3.3.2
 - 需要引入的 jar 包：commons-lang3-3.3.2.jar
- 使用方式
 - 重写所有 JavaBean 的 toString 方法为：
 - *return ToStringBuilder.reflectionToString(this);*
 - debug 时，直接传入相应的 JavaBean 即可
 - `logger.debug(“the value of A: {}” , a);`

注：完整版本代码参见附录中代码示例：7-slf4j-logback-lang3

使用技巧三： eclipse 代码辅助



- 创建类时自动 import slf4j 的类
 - Window-> Preferences->Java->Code Style->Code Templates->Code->New Java Files 修改为：
- 创建类时自动创建静态 logger 属性
 - Window-> Preferences->Java->Code Style->Code Templates->Code->Class Body 修改为：

```
${filecomment}  
${package_declaration}  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
${typecomment}  
${type_declaration}
```

```
static Logger logger = LoggerFactory.  
getLogger(${type_name}.class);
```

使用技巧三： eclipse 代码辅助



- 代码辅助： logdebug

- Window-> Preferences->Java->Editors->Templates->New :

- name:logdebug

```
logger.debug("${cursor}", ${var});
```

使用方法：

输入 `logdebug` ， 按 `Alt+/`

- 代码辅助： toString

- Window-> Preferences->Java->Editors->Templates->New :

- name:toString

```
@Override  
public String toString() {  
    return ReflectionToStringBuilder.  
reflectionToString(this);  
}
```

使用方法：

输入 `toString` ， 按 `Alt+/`



- <https://github.com/haojinlong/training/tree/master/src/1>
 - 使用 commons-logging 和 log4j : 1-jcl-log4j
 - 使用 commons-logging 和 log4j2 : 2-jcl-log4j2
 - 使用 slf4j 和 log4j : 3-slf4j-log4j
 - 使用 slf4j 和 log4j2 : 4-slf4j-log4j2
 - 使用 slf4j 和 logback : 5-slf4j-logback
 - 性能测试代码: 6-slf4j-logback-efficient
 - 使用 commons-lang3 示例: 7-slf4j-logback-lang3
 - 使用 slf4j 替代 commons-logging 示例: 8-slf4j-jcl-logback
- 更多郝金隆 Java 培训尽在
 - <https://github.com/haojinlong/training/tree/master/doc/pdf>