ISSN: 2754-6705

Journal of Mathematical & Computer Applications

SCIENTIFIC Research and Community

Review Article

Open Access

Fluent Logging In Java Using Flogger

Nilesh Jagnik

Mountain View, USA

ABSTRACT

Systems logs are very important for detecting problems in applications. Not only this, they are useful for auditing, monitoring and analysis too. Therefore, it is important to have adequate logging in code. Often adding too many log statements can lead to problems like performance issues and log spam. This means tweaking log behavior is necessary, which can add a lot of management overhead for developers. To solve this problem, logging APIs exist which make this task easier. Logging APIs can improve the readability, performance and ease of logging. Several logging APIs are available for logging in Java. The Flogger API has several advantages over other options. It is simple and fluent, thus, increasing readability. It has support for several quality-of-life features which not only make it easy to add logs in code, but also ensure the performance overhead of logging is minimal. Log behavior is configurable and can be tweaked at runtime. In this paper, we review the Flogger API, its usage, features and advantages over other options.

***Corresponding author**

Nilesh Jagnik, Mountain View, USA.

Received: October 06, 2022; **Accepted:** October 14, 2022, **Published:** October 26, 2022

Keywords: Software Logging, Fluent APIs, Software Debugging, Auditing Software

Introduction

Visibility into the behavior of real-world applications is quite important, not only for debugging issues but also for understanding the system behavior and asserting that certain assumptions related to it are true. Visibility is especially important when new features are launched.

Although there are several tools that can be used to gain different types of insights about system behavior, one of the most basic ways is to use logging. Logging refers to the use of print statements to capture summaries of the state, input and functioning of the system.

Logging provides developers a way to record and analyze the behavior of systems. There are also other niche uses of logging, which we will discuss in this paper. Although print statements are a simple way to do logging, they can be quite restrictive and even detrimental for production systems. Production systems require so much logging that the use of a logging API along with a logging framework is highly recommended. In addition, print statements can often introduce slowness and bottlenecks in the system.

Google released Flogger, which is a fluent logging API for Java applications. Flogger handles the complexities of logging and provides many features which come in handy for various debugging, troubleshooting and auditing scenarios.

Importance of Logging

There are many reasons to utilize logging in your applications.

Troubleshooting and Debugging

Logs can be used to identify bugs and incorrect behavior in systems. Logs provide a detailed history of events leading up to the failure, which can be used to identify root causes.

Performance Monitoring

Logging can be used to track performance. This includes latency, errors and resource utilization. Note that logs don't provide aggregation for these metrics, only per event monitoring.

Auditing

Logs can be used to audit certain events occurring in the system. Logs related to security-related events like login attempts and access control violations can give insights about security breach attempts. Logs can also help assert that compliance requirements are not violated.

User Behavior

Logs can help understand user behavior and usage which can be used for improving usability for applications.

Why Use APIs for Logging

Although logging using print statement might work fine for smaller applications, the use of logging APIs is highly recommended for applications serving real-world traffic for the following reasons:

Flexibility

The use of logging APIs provides features like control over the level of detail, format of log messages, and also the destination to which logs must be stored to.

Structure

With the use of logging APIs, log messages can be made to conform to a structure. This makes it easier to query, filter and analyze log data.

Logging Backends

Logging APIs are capable of interacting with logging services that collect, process and store logs for access. These logging services are able to provides centralized logs collected from different parts of a system. Logging services may also provide interfaces for easy reading, filtering and analysis of logs.

Performance
Logging APIs can often be more performant as compared to simple print statements. Without these APIs, your application may spend too much time formatting, preparing and persisting logs.

Flogger Usage
Flogger is an open-source logging API for Java developed by Google. This is a fluent API and offers many useful features for logging. Let us take a look at Flogger usage and features. The use of Flogger involves a few simple steps.

Dependencies
First, add necessary build dependencies. We need to add two dependencies here. The first one is for Flogger itself. We also need an additional dependency for specifying which logging backend service to use. Fig. 1 shows the maven dependencies which should be added. In this example we use the default flogger-system-backend backend. If your service is already logging to another service, use that backend instead.

```
<dependency>
    <groupId>com.google.flogger</groupId>
    <artifactId>flogger</artifactId>
    <version>0.3.1</version>
</dependency>

<dependency>
    <groupId>com.google.flogger</groupId>
    <artifactId>flogger-system-backend</artifactId>
    <version>0.3.1</version>
</dependency>
```

Figure 1: Maven Dependencies for Flogger.

Code
Writing code using Flogger is simple and fluent. First, add an import statement. Then instantiate a logger. The logger instance should be a private, static and final member of the class it is logging in.

After this setup is done, we can start using the logger instance for logging. Log messages can be constructed using printf format specifiers.

```
// Add Flogger import
import com.google.common.flogger.FluentLogger;

public class MyLoggingClass {
  // Declare a private static final logger instance.
  private static final FluentLogger logger =
                  FluentLogger.forEnclosingClass();

  public void doWork(String taskId) {
    logger
      .atInfo()
      .log("Work started for taskId=%s.", taskId);
    try {
      doActualWork();
    } catch (SomeException ex) {
      logger
        .atWarning()
        .withCause(ex)
        .log(
          "Found error while processing taskId=%s.",
          taskId);
    }
    logger
      .atInfo()
      .log("Work finished for taskId=%s.", taskId);
  }
}
```

Figure 2: Using Flogger for Fluent Logging in Java.

Log Levels
Flogger supports various logging levels. These can be specified by adding atFine(), atInfo(), atWarning(), atSevere(), etc., in the fluent chain. This allows the program to decide logging behavior at runtime, i.e., which levels to actually log and which to treat as no-op. atFine() and below can be used for logging a high number of events since these are disabled by default have to be specifically enabled when finer logging is needed.

Exceptions
Exceptions can be logged with the withCause() method in the fluent chain. It is also possible to specify the size of the stacktrace using withStackTrace() method. This means that users don't need to manually convert exceptions into strings for logging. Figure 2 shows an example of logging exceptions.

Logging Frequency
To avoid log spam and logging backend overload, users can specify the frequency of a log statement using the every() method in the fluent chain.

```
logger.atInfo().every(10).log("Prints 1 in 10 times");
```

Figure 3: Specifying frequency of Flogger log statements.

Lazy Arguments
According to best practice for logging, expensive work should be avoided at log sites. This is because this work will be done regardless of whether the log statement will be printed or skipped (based on configuration and frequency). To help with this, Flogger provides a LazyArgs class. Fig. 4 shows the usage of this class. Note that this is only available in Java 8 and above.

```
logger.atInfo().log(
  "Summary: %s",
  lazy(() -> summarize(s)));
```

Figure 4: Lazy evalutation of a log argument

Benefits of Flogger
As seen in the previous section, Flogger offers a lot of features that enable fluency and simplicity in logging.

Readability
The fluent style of Flogger improves readability of code since the log statements are more expressive and easier to read. In addition, the features offered by Flogger such as frequency control, log levels, exception logging, etc., make it much simpler to log by removing conditional logic and argument processing from code.

Performance
Due to the support for lazy arguments and configurable log levels, no work is done for log levels which are disabled. This allows adding finer logging at lower levels without worrying about log pollution.

In addition, Flogger permits use of a variety of different logging backends like Log4j2, SLF4j, etc., which further improves application performance.

Extensibility
If you have special needs for extending the functionality of Flogger by adding more methods to the fluent chain, it is easy to extend the FluentLogger class to support this type of functionality [1-6].

Conclusion

Logging events in applications is quite important for debugging, monitoring and auditing application behavior. The use of logging APIs can make logging easy and hassle-free. For Java applications, the Flogger library offers a fluent API for logging which not only is expressive but also offers several features that improve the readability and performance of logging code.

References

1. (2019) Flogger: A Fluent Logging API for Java https://google.github.io/flogger.
2. Anil Kurmi (2019) Flogger Java Logger-A Fluent Logging API for Java developer (May 2019) https://medium.com/@anilkkurmi/flogger-java-logger-a-fluent-logging-api-for-java-developer-6a03b131cd51.
3. Dustin Schultz, "Google Releases New Java Logging Framework (Apr 2019)," https://www.infoq.com/news/2019/04/java-logging-framework-flogger.
4. Lokesh Gupta (2022) Java Fluent Logging with Flogger https://howtodoinjava.com/java/library/fluent-logging-with-flogger.
5. (2021) Not another logger! (Mar 2021) https://dev.to/dansiviter/not-another-logger-2lc4.
6. (2020) Class FluentLogger https://www.javadoc.io/doc/com.google.flogger/flogger/0.5/com/google/common/flogger/FluentLogger.html.