Top 10 Tips on Logging in Java - Tutorial

Java logging or logging in java is as much an art as science. knowing write tools and API for java logging is definitely science part but choosing format of java logs , format of messages, what to log in java , which logging level to use for which kind of messages are purely an experienced based things and sort of art which you learn by applying logging in java. Since its proven fact that java logging severely affects performance and I have seen latency of online stock trading application goes multiple times if they run in DEBUG mode than in WARN or higher level mode. Since latency and speed is major concern for any electronic trading system or high volume low latency stock trading system, it becomes absolutely necessary to understand and learn java logging in great details and best practices and tips available on logging in java. This is not just for finance and investment banking domain but also to any java server or client application which requires speed and java logging at same time.

Top 10 tips on logging in Java

In this java logging tutorial I have shared my experience with logging in Java, I have tried to answer fundamental questions on java logging like "Why we need logging in Java", "What are different logging level in Java and how to choose correct logging level in java", "How incorrect java logging affect performance" and discussed the tools, libraries and API available for logging in Java e.g. log4j.jar and java.util.logging package.

Why we need logging in Java

This is pretty basic java logging question and everybody argue that if we Java System.out.println() for printing messages then whey we use logging. Everybody who starts java starts with System.out.println() for printing message in java console. But this is not at all powerful as compared to advanced Java logging API like log4j and java.util.logging. If you are writing a java application server then only way to know what your server is doing is by seeing log file of your server. suppose you don't write anything in your java log file then no body knows what your sever is doing, it becomes increasingly important if your application is connected to upstream and downstream like in many stock trading systems or electronic trading system and get input from upstream , transforms and normalize it and send down to downstream. In case of any issue without java logs you won't be able to figure out what went wrong. That's why logging in java is most important while writing any Java server application. Logging in Java is not by choice it's must to understand .

What are different logging level in Java

Anybody who is using logging in java must be familiar with basic java logging level e.g. DEBUG, INFO, WARN and ERROR.

DEBUG is the lowest restricted java logging level and we should write everything we need to debug an application, this java logging mode should only be used on Development and Testing environment and must not be used in production environment.

INFO is more restricted than DEBUG java logging level and we should log messages which are informative purpose like Server has been started, Incoming messages, outgoing messages etc in INFO level logging in Java.

WARN is more restricted than INFO java logging level and used to log warning sort of messages e.g. Connection lost between client and server. Database connection lost, Socket reaching to its limit. These messages and java logging level are almost important because you can setup alert on these logging messages in Java and let your support team monitor health of your java application and react on this warning messages. In Summary WARN level is used to log warning message for logging in Java.

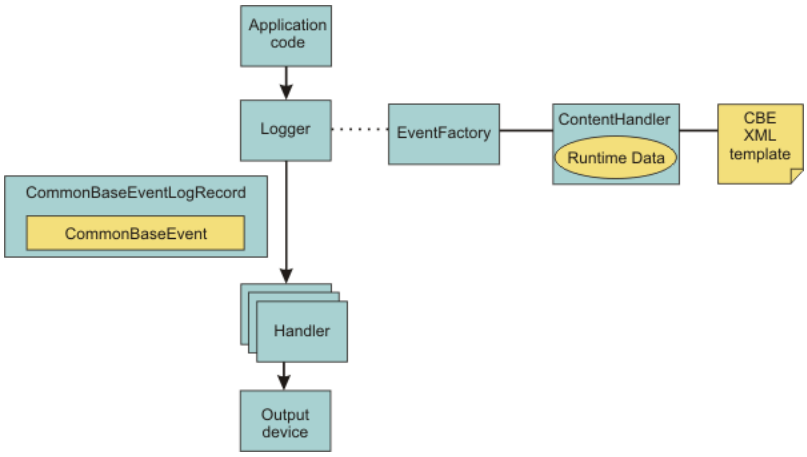
ERROR is the more restricted java logging level than WARN and used to log Errors and Exception, you can also setup alert on this java logging level and alert monitoring team to react on this messages. ERROR is serious for logging in Java and you should always print it.

FATAL java logging level designates very severe error events that will presumably lead the application to abort. After this mostly your application crashes and stopped.

OFF java logging level has the highest possible rank and is intended to turn off logging in Java.

These java logging levels are based on log4j logging level and little bit different than java.util.logging API which provides some more logging level like SEVERE, FINER, FINEST, FATAL etc as name suggest based upon criticality of your logging message you can choose any of this level for logging in Java.

Using log4j or java.util.logging API for logging in Java



I would recommend using log4j for java logging, you may argue that why not java.util.logging API. I agree java.util.logging API is also very powerful but I find log4j more intuitive and easy to use than java.util.logging API. You have seen logging levels in log4j they have optimized number of java logging level and each of them fully describes what it does. Another flexibility of log4j is that you can change logging level of your java application without restarting your java application, by the way this you can do in java.util.logging API by using JMX if you have implemented that.

Log4j also provides flexibility to set logging level based on per class in its configuration file log4j.xml. You can either use log4j.properties file or log4j.xml for configuring java logging in your application while using log4j for logging in java. Also log4j is thread-safe. Log4j components are designed to be used in heavily multithreaded systems. On the other hand I found Formatter and Appender facility of java.util.logging API quite useful especially Formatter allows you to format java logging output as you want for logging in Java.

How logging in Java affects performance

Java logging severely affects performance of your application. Its quite common sense that more you log, more you perform file IO which slows down your application. That's why choosing correct java logging level for every single message is quite important. Since having no java logging is not a choice you have to have logging in java application, what you can control is logging level and logging messages on that level. So always log DEBUG messages inside isDebugEnabled() block as shown in below example of debug mode in java.

```
if(logger.isDebugEnabled()){

  logger.debug("java logging level is DEBUG Enabled");
```

}

Uses either WARN ERROR or FINER, FINEST java logging level in production environment. Never use DEBUG level logging in java in production, I have seen sometime production running very slow and found that some one has put the java log on DEBUG mode.

10 tips on logging in Java

1) Use isDebugEnabled() for putting debug log in Java , it will save lot of string concatenation activity if your code run in production environment with production logging level instead of DEBUG logging level.

2) Carefully choose which kind of message should go to which level for logging in Java, It become extremely important if you are writing server application in core java and only way to see what happening is Java logs. If you log too much information your performance will be affected and same time if you don't log important information like incoming messages and outgoing messages in java logs then it would become extremely difficult to identify what happened in case of any issue or error because nothing would be in java logs.

3) Use either log4j or java.util.logging for logging in Java, I would recommend log4j because I have used it a lot and found it very flexible. It allows changing logging level in java without restarting your application which is very important in production or controlled environment. To do this you can have log4j watchdog which continuously look for log4j.xml in a particular directory and if founds loads it and reset logging in java.

4) By using log4j.xml you can have different logger configuration for different Java classes as well. You can have some classes in INFO mode, some in WARN mode or ERROR mode. It’s quite flexible to do this to customize java logging.

5) Another important point to remember is format of java logging, this you specify in logger. Properties file in case of java.util.logging API for logging to use which java logging Formatter. Don’t forget to include Thread Name and fully qualified java class Name while printing logs because it would be impossible to find sequence of events if your code is executed by multiple threads without having thread name on it. In my opinion this is the most important tips you consider for logging in Java.

6) By carefully choosing format of java logging at logger level and format of writing log you can have generate reports from your java log files. Be consistent while logging messages, be informative while logging message, print data with message wherever required.

7) While writing message for logging in Java try to use some kind of prefix to indicate which part of your code is printing log e.g. client side , Database site or session side, later you can use this prefix to do a "grep" or "find" in Unix and have related logs at once place. Believe me I have used this technique and it helped a lot while debugging or investigating any issues and your log file is quite large. For example you can put all Database level log with a prefix "DB_LOG:" and put all session level log with prefix "SESSION_LOG:"

8) If a given logger is not assigned a level, then it inherits one from its closest ancestor. That's why we always assign log level to root logger in configuration file log4j.rootLogger=DEBUG.

9) Both no logging and excessive logging is bad so carefully planned what to log and on which level you log that messages so that you can run fast in production environment and at same time able to identify any issue in QA and TEST environment.

10) I found that for improving logging its important you look through your log and monitor your log by yourself and tune it wherever necessary. It's also important to log in simple English and it should make sense and human readable since support team may wan to put alert on some logging message and they may want to monitory your application in production.

11) if you are using SLFJ for logging in java use parametrized version of various log methods they are faster as compared to normal method.

logger.debug("No of Orders " + noOfOrder + " for client : " + client); // slower

logger.debug("No of Executions {} for clients:{}", noOfOrder , client); // faster

These tips and examples on logging in java is based on my experience and how I use logging in Java and by no means complete, I would love to hear some more tips from you guys and how you guys are using and customizing java logging. I would recommend reading detailed and official documentation for both java.util.logging and log4j to get complete and detailed information on java logging as well.

Update:

Logging is more art than science and its one of those skill which separates good developers with great developers. but logging has many aspect and its a general concept which is equally applicable to Java and other programming language:

1) Which information should you log?

2) Which information goes to which level of logging?

Apart from this fundamental question which appears in everybody's mind while doing logging I will share some of the best practices I follow while writing logs for my java programmers:

1) Never log sensitive information like Password, Social Security number, credit card numbers or account number as plain text in log file. Its better simply don't store these information in application and remove it as soon as you are done with it.

2) Always log decision making statements. for example you have a Java application which loads some settings from preference file or environment and if it doesn't found than loads default settings. If you are using default setting than log this information like below :

logger.info("Not able to load personal settings, default Setting selected for user : {user});

key point this log statement is missing is why its not able to load personals setting so exception should also be logged if necessary. but if you look at log it conveys pretty useful information like {user} which tells for which particular user this happens, which is very important if this is a server side application and you have many users.

3) Consistency : Consistency is key knot just in logging but also in coding. doesn't matter which format you follow if you are consistent with that format than only it adds a value but spend some time to decide your logging format so that it can capture all useful information.

4) log all important information which is necessary to debug or troubleshoot a problem if it happens.

One example is this we often convert String to Date in our application and if String is not a valid date it throws ParseException but I have seen code which is catching this Exception, assigning null to Date and printing log :

logger.info("failed to convert String to date");

if you see this line in log file you will not be able to determine for which date you got invalid value if you have multiple

date fields like startDate, endDate. This is also not printing the value of String which it tries to convert into Date. You need

those value to troubleshoot or find the actual cause, a better logging statement could be:

logger.info("invalid startDate : {startDate});

Thanks for reading so far. If you like this article please share with your friends.