How to use log levels in java

Asked 10 years, 2 months ago Active 1 month ago Viewed 167k times

55

9

I am developing an application where i need to use the logger functionality. I have read about different levels of logger which are:

- SEVERE (highest)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST

I am not able to understand the usage of each logging level.

Can someone give me a good example showing all the logging levels and their usage?

java logging

Share Edit Follow

edited Aug 20 '13 at 11:26
Jeroen
53.7k 32 172 281

asked Apr 28 '11 at 11:17
vinod
1,068 2 14 41

1 I think this link would be helpful.vogella.de/articles/Logging/article.html – Ammu Apr 28 '11 at 11:19

9 Answers

Active Oldest Votes

64

✓

Generally, you don't need all those levels, SEVERE, WARNING, INFO, FINE might be enough. We're using Log4J (not java.util.logging directly) and the following levels (which might differ in name from other logging frameworks):

- ERROR: Any error/exception that is or might be critical. Our Logger automatically sends an email for each such message on our servers (usage: `logger.error("message");`)
- WARN: Any message that might warn us of potential problems, e.g. when a user tried to log in with wrong credentials - which might indicate an attack if that happens often or in short periods of time (usage: `logger.warn("message");`)

- INFO: Anything that we want to know when looking at the log files, e.g. when a scheduled job started/ended (usage: `logger.info("message");`)
- DEBUG: As the name says, debug messages that we only rarely turn on. (usage: `logger.debug("message");`)

The beauty of this is that if you set the log level to WARN, info and debug messages have next to no performance impact. If you need to get additional information from a production system you just can lower the level to INFO or DEBUG for a short period of time (since you'd get much more log entries which make your log files bigger and harder to read). Adjusting log levels etc. can normally be done at runtime (our JBoss instance checks for changes in that config every minute or so).

Share Edit Follow edited Jun 1 '18 at 13:14 answered Apr 28 '11 at 11:29
Thomas
81.2k 12 112 143

13 The question is about java's built-in logging - not log4j. BTW, why Sun didn't adopt log4j convention? – AlikElzin-kilaka May 9 '14 at 22:23

4 @kilaka actually the OP is not specific on whether he means `java.util.logging` or more generally `logging in Java`. Besides that, I spoke from our experience and since in most cases the different levels would have a similar meaning, I just told him what we use in production and what is sufficient for us in the real world. So instead of having difficulties in deciding between 7 log levels, reducing that complexity to 4 log levels would help the OP, IMHO. – Thomas May 13 '14 at 16:44

8 the question was not about log4j. – John Donn Aug 12 '16 at 8:19

3 @JohnDonn yes, it was about log levels in general and I was using log4j just to provide some experience. Since log4j uses slightly different names for the log levels I felt like providing that information so the reader can adapt. However, the general information stays the same and is independent of the actual logging framework being used. – Thomas Aug 12 '16 at 9:12

1 Do not use java logging. It has problens with some production servers and sometimes its just don't write down. When in doubt use sl4j slf4j.org. It is a facade that suports almost all logging frameworks. If you decide mid terms change logging engine your code will not change. You can even use several logging engines at the same time. Works with java default logging too – Higarian Feb 20 '19 at 12:38

24

This excerpt is from the following awesome post.

> **ERROR** – something terribly wrong had happened, that must be investigated immediately. No system can tolerate items logged on this level. Example: NPE, database unavailable, mission critical use case cannot be continued.
>
> **WARN** – the process might be continued, but take extra caution. Actually I always wanted to have two levels here: one for obvious problems where work-around exists (for example: “Current data unavailable, using cached values”) and second (name it:

ATTENTION) for potential problems and suggestions. Example: “Application running in development mode” or “Administration console is not secured with a password”. The application can tolerate warning messages, but they should always be justified and examined.

INFO – Important business process has finished. In ideal world, administrator or advanced user should be able to understand INFO messages and quickly find out what the application is doing. For example if an application is all about booking airplane tickets, there should be only one INFO statement per each ticket saying “[Who] booked ticket from [Where] to [Where]“. Other definition of INFO message: each action that changes the state of the application significantly (database update, external system request).

DEBUG – Developers stuff. I will discuss later what sort of information deserves to be logged.

TRACE – Very detailed information, intended only for development. You might keep trace messages for a short period of time after deployment on production environment, but treat these log statements as temporary, that should or might be turned-off eventually. The distinction between DEBUG and TRACE is the most difficult, but if you put logging statement and remove it after the feature has been developed and tested, it should probably be on TRACE level.

PS: Read **TRACE** as **VERBOSE**

Share Edit Follow edited May 25 at 3:52 answered Apr 30 '13 at 16:02

Ragunath Jawahar
18.7k 20 102 154

15

The java.util.logging.Level documentation does a good job of defining when to use a log level and the target audience of that log level.

Most of the confusion with `java.util.logging` is in the tracing methods. It should be in the class level documentation but instead the `Level.FINE` field provides a good overview:

> FINE is a message level providing tracing information. All of FINE, FINER, and FINEST are intended for relatively detailed tracing. The exact meaning of the three levels will vary between subsystems, but in general, FINEST should be used for the most voluminous detailed output, FINER for somewhat less detailed output, and FINE for the lowest volume (and most important) messages. In general the FINE level should be used for information that will be broadly interesting to developers who do not have a specialized interest in the specific subsystem. FINE messages might include things like minor (recoverable) failures. Issues indicating potential performance problems are also worth logging as FINE.

One important thing to understand which is not mentioned in the level documentation is that call-site tracing information is logged at `FINER`.

- Logger#entering A LogRecord with message "ENTRY", log level FINER, ...
- Logger#throwing The logging is done using the FINER level...The LogRecord's message is set to "THROW"
- Logger#exiting A LogRecord with message "RETURN", log level FINER...

If you log a message as `FINE` you will be able to configure logging system to see the log output with or without tracing log records surrounding the log message. So use `FINE` only when tracing log records are not required as context to understand the log message.

> FINER indicates a fairly detailed tracing message. By default logging calls for entering, returning, or throwing an exception are traced at this level.

In general, most use of `FINER` should be left to call of entering, exiting, and throwing. That will for the most part reserve `FINER` for call-site tracing when verbose logging is turned on. When swallowing an expected exception it makes sense to use `FINER` in some cases as the alternative to calling trace `throwing` method since the exception is not actually thrown. This makes it look like a trace when it isn't a throw or an actual error that would be logged at a higher level.

> FINEST indicates a highly detailed tracing message.

Use `FINEST` when the tracing log message you are about to write requires context information about program control flow. You should also use FINEST for tracing messages that produce large amounts of output data.

> CONFIG messages are intended to provide a variety of static configuration information, to assist in debugging problems that may be associated with particular configurations. For example, CONFIG message might include the CPU type, the graphics depth, the GUI look-and-feel, etc.

The `CONFIG` works well for assisting system admins with the items listed above.

> Typically INFO messages will be written to the console or its equivalent. So the INFO level should only be used for reasonably significant messages that will make sense to end users and system administrators.

Examples of this are tracing program startup and shutdown.

> In general WARNING messages should describe events that will be of interest to end users or system managers, or which indicate potential problems.

An example use case could be exceptions thrown from AutoCloseable.close implementations.

> In general SEVERE messages should describe events that are of considerable importance and which will prevent normal program execution. They should be reasonably intelligible to end users and to system administrators.

For example, if you have transaction in your program where if any one of the steps fail then all of the steps voided then SEVERE would be appropriate to use as the log level.

Share Edit Follow edited Dec 16 '20 at 21:01 answered Dec 29 '16 at 2:25
jmehrens
9,039 1 31 43

1 Thank you for actually answering the original question. – John Meyer Mar 26 '18 at 19:38

4

Here is a good introduction to logging in Java:
http://www.javapractices.com/topic/TopicAction.do?Id=143

Java comes with a logging API since it's 1.4.2 version:
http://download.oracle.com/javase/1.4.2/docs/guide/util/logging/overview.html

You can also use other logging frameworks like Apache Log4j which is the most popular one:
http://logging.apache.org/log4j

I suggest you to use a logging abstraction framework which allows you to change your logging framework without re-factoring you code. So you can starts by using Jul (Java Util Logging) then swith to Log4j without changing you code. The most popular logging facade is slf4j:
http://www.slf4j.org/

Regards,

Share Edit Follow answered Apr 28 '11 at 11:37
Mehrez Marouani
97 1

3

Those are the levels. You'd consider the severity of the message you're logging, and use the appropriate levels.

It's basically a watermark; the higher the level, the more likely you want to retain the information in the log entry. FINEST would be for messages that are of very little importance, so you'd use it for things you usually don't care about but might want to see in some rare circumstance.

Share Edit Follow

answered Apr 28 '11 at 11:19
Joseph Ottinger
4,721 1 20 23

You have them. Logger.severe("foo");, etc. – Joseph Ottinger Apr 28 '11 at 11:47

1

The use of levels is really up tp you. You need to decide what is severe in your application, what is a warning and what is just information. You need to split your logging so that your users can easily set up a level of logging that doesn't kill the system with excessing IO but which will report serious errors so you can fix them.

Share Edit Follow

answered Apr 28 '11 at 11:21
alpian
4,448 16 18

0

the different log levels are helpful for tools, whose can anaylse you log files. Normally a logfile contains lots of information. To avoid an information overload (or here an stackoverflow^^) you can use the log levels for grouping the information.

Share Edit Follow

answered Apr 28 '11 at 11:21
Reporter
3,557 5 28 45

Actually the messages are not "grouped" by the logging levels ("Grouping" meaning some ordering). – U. Windl Apr 19 '18 at 21:08

0

Logging has different levels such as :
Trace – A fine-grained debug message, typically capturing the flow through the application.
Debug- A general debugging event should be logged under this.
ALL – All events could be logged.
INFO- An informational purpose, information written in plain english.
Warn- An event that might possible lead to an error.
Error- An error in the application, possibly recoverable.

Logging captured with debug level is information helpful to developers as well as other personnel, so it captures in broad range. If your code doesn't have exception or errors then you should be alright to use DEBUG level of logging, otherwise you should carefully choose options.

Share Edit Follow

answered Feb 28 '19 at 2:43
surendrapanday
360 1 12

-4

This tip shows how to use Logger in any java application. Logger needs to configure Formatter and Handler. There are many types of handlers and formatters present. In this example FileHandler is used to store all the log messages in a log file. And Simple formatter is used to format the log messages in human readable form.

```
package MyProject;

import java.io.IOException;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;

public class MyLogger {

  public static void main(String[] args) {

    Logger logger = Logger.getLogger("MyLog");
    FileHandler fh;

    try {

      // This block configure the logger with handler and formatter
      fh = new FileHandler("c:\\MyLogFile.log", true);
      logger.addHandler(fh);
      logger.setLevel(Level.ALL);
      SimpleFormatter formatter = new SimpleFormatter();
      fh.setFormatter(formatter);

      // the following statement is used to log any messages
      logger.log(Level.WARNING,"My first log");

    } catch (SecurityException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }

  }

}
```

some more examples you can find here https://docs.oracle.com/javase/7/docs/api/java/util/logging/Logger.html

Share Edit Follow

edited Aug 9 '16 at 5:25
Runcorn
4,894 4 31 50

answered Apr 28 '11 at 11:23
Noor Khan
540 1 5 12

11 This doesn't even answer the question, and the link is broken. – Alexis Leclerc Dec 4 '14 at 22:24

2

Never printStackTrace in prod. Warning in a success point makes no sense (use info instead). Add log in exception catching and give exception as param. – lpratlong Oct 17 '16 at 9:27

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.