How Expensive is Thread.getStackTrace()?

Ask Question

15

8

In a logging system, every log output is done by a helper class with a method such as this one

```
public void debug(String message) {
    Logger logger = Logger.getLogger(getCallingClass());
    logger.debug(message);
}
...
public Class getCallingClass() {
/*
Calls Thread.getStackTrace() and back traces until the class on the stack trace
!= this.getClass().
*/
    return classFound;
}
```

How expensive is this to run and could it have significant performance hits?

java logging

asked Feb 27 '10 at 15:25
Jaime Garcia
3,224 ●5 ●39 ●57

4 Answers

8 ✓

Yes, there is some overhead to this call, but in all likelyhood, you're going to do something like this:

```
public static boolean DEBUG_ON = true; //c
```

then,

```
public void debug(String message){
  if(DEBUG_ON){
     //stack code here
  }

}
```

Which will cause you to not take the hit in your real code.

Even then, for exceptions, you're going to throw a whole stack traced Exception in your production build.

Note that if you are using a decent logging subsystem, they will probably already do something based on the logging level (in our

proves to be a real performance problem.
Premature Optimization is the root of all evil :)

answered Feb 27 '10 at 15:29
Kylar
5,815 ● 4 ● 35 ● 70

1 instead of using a boolean flag this condition could be used logger.isDebugEnabled() – Inv3r53 Feb 27 '10 at 15:37

1 a better way is to use preprocessing directives. – Orentet Feb 27 '10 at 15:38

There's benefits to both. with the static final boolean, the compiler will just remove anything inside the if() statement, so it's kinda like a preprocessor directive ;) – Kylar Feb 27 '10 at 16:04

5 At any given time my (Windows) system is running about 50-80 programs, services and device drivers. If every one of them uses just 20% more resources (CPU and memory) than they would have if the programmer thought just a little about performance, that amounts to a huge hit in what my hardware can do. Thinking about performance **before** you write the code **is not** the root of all evil (the love of money is). – Lawrence Dol Feb 27 '10 at 19:38

7

It looks like getting the current thread (and its associated ID) is not expensive, but getting the current thread and its stack trace is. The new throwable().getStackTrace() pattern seems to be a lot faster than the thread's stack trace pattern.

Also, note: this benchmark has almost no stack depth since its just a main method, so in a server environment this penalty will be a lot heavier.

Benchmark results:

Simple loop took 2 ms

Getting current thread took 10 ms

Getting stack trace took 29564 ms

Getting throwable stack trace took 19910 ms

Code:

```
int trials = 10_000_000;

    long start = System.currentTimeMillis(

    long a = 1;
    for (int i = 0; i < trials; i += 1) {
        a += 1;
    }
    a = 1;
    for (int i = 0; i < trials; i += 1) {
        a += 1;
        Thread.currentThread().getId();
    }

    duration = System.currentTimeMillis()
    System.out.println("Getting current th

    start = System.currentTimeMillis();

    a = 1;
    for (int i = 0; i < trials; i += 1) {
        a += 1;
        Thread.currentThread().getStackTra
    }

    duration = System.currentTimeMillis()
    System.out.println("Getting stack trac

            start = System.currentTimeMill

    a = 1;
    for (int i = 0; i < trials; i += 1) {
        a += 1;
        (new Throwable()).getStackTrace();
    }

    duration = System.currentTimeMillis()
    System.out.println("Getting throwable 
```

edited May 16 '12 at 19:13

answered May 16 '12 at 19:08

Ilya
548 ● 5 ● 8

according to source code, i would say currentThread is responsible for the difference – njzk2 Nov 14 '12 at 16:42

3 This benchmark is broken. javac/JVM will optimize it to the extent theat the 1st and the 2nd loops are completely removed. Also, millis cannot be used here. – Roman Aug 5 '14 at 11:01

5 "Getting stack trace took 29564 ms" is an absurd statement; the test code indicates it took 2,900 *nano*seconds. Additionally @Roman is entirely correct about optimization altering results. – gerardw Jul 27 '15 at 13:35

4

From what I recall, there's some impact in using `Thread.getStackTrace()` - especially with large stacks (such as when using in server-side or J2EE situations). You could try `Throwable.getStackTrace()` for better performance.

At any rate, calling those functions regularly (as opposed to doing so in an exception situation) will impact your app.

answered Feb 27 '10 at 15:36

Traveling Tech Guy
15k ● 15 ● 81 ● 135

`Thread.getStackTrace()` under the hood. – Lawrence Dol Feb 27 '10 at 19:41

6 @SoftwareMonkey actually it does: Throwable.fillInStackTrace() can take advantage of knowing that it's examining the stack for the same thread that's calling the method, whereas Thread.getStackTrace() has to be thread-safe and is much more expensive. See bugs.sun.com/bugdatabase/view_bug.do?bug_id=6375302 – David Moles Dec 8 '11 at 23:32

@David: Good to know, thanks. – Lawrence Dol Dec 9 '11 at 1:15

@Lawrence : if you do Thread.currentThread().getStackTrace() then it should use same fillInStackTrace without worrying about thread safety, isnt it? – chitresh Oct 20 '14 at 17:53

1 @chitresh: I am not sure what you are asking. The `Thread.currentThread()` has already incurred some overhead, and if I understand @David correctly, the `.getStackTrace()` is "much higher overhead" than the same method on Throwable because the Thread object itself must preserve thread safety in getting the stack for the associated thread, whereas Throwable filling in the stacktrace on `new Throwable()` already knows that it's for the calling thread. – Lawrence Dol Oct 20 '14 at 19:13

3

Now with JDK 9 & 10, you can use StalkWalker, which is not an expensive call.

```
private void invoke006() {
        var stack =
StackWalker.getInstance(StackWalker.Option
s.collect(Collectors.toList()));
        stack.forEach(stackFrame -> {
            if (stackFrame.getMethodName()
                System.err.println("master
                System.err.println(StackWa
s.collect(Collectors.toList())).get(0).getI
StackWalker.getInstance().walk((s) ->
s.collect(Collectors.toList())).get(0).getI
            }
        });
    }
```

answered Mar 4 '18 at 3:38

user3892260

490 ● 1 ● 5 ● 11