How can i benchmark method execution time in java?

Asked 14 years, 7 months ago Modified 3 years, 8 months ago Viewed 53k times

57

I have a program which i have myself written in java, but I want to test method execution times and get timings for specific methods. I was wondering if this is possible, by maybe somehow an eclipse plug-in? or maybe inserting some code?

I see, it is quite a small program, nothing more than 1500 lines, which would be better a dedicated tool or `System.currentTimeMillis()`?

java optimization

Share Edit Follow

edited Jun 2, 2019 at 11:47
Matthew
1,943 ●3 ●21 ●27

asked Mar 31, 2010 at 13:37
KP65
13.6k ●13 ●46 ●46

4 You should use `System.nanoTime()` to measure *elapsed time* (see stackoverflow.com/questions/238920/...), not `System.currentTimeMillis()`. Other points (warmup, JIT) mentioned by @Stephen in a comment are still valid. – Pascal Thivent Mar 31, 2010 at 14:15

2 Use stopwatch class for this task – jarvo69 Aug 25, 2016 at 8:44

10 Answers

Sorted by: Highest score (default)

61

Other than using a profiler, a simple way of getting what you want is the following:

```
public class SomeClass{
   public void somePublicMethod()
   {
       long startTime = System.currentTimeMillis();
       someMethodWhichYouWantToProfile();
       long endTime = System.currentTimeMillis();
       System.out.println("Total execution time: " + (endTime-startTime) + "ms");
   }
 }
```

Share Edit Follow

answered Mar 31, 2010 at 13:43
Chris Knight
25k ●24 ●91 ●137

1 @Stephen C - fair points. The above is the method I primarily use when trying to get a quick idea to method efficiency. My requirements in most cases aren't needed to single ms precision. – Chris Knight Mar

31, 2010 at 14:13

1 I might want to add to this answer that you could take warmup into consideration. By using the `-XX:CompileThreshold=100` flag you force the JVM to compile your method to native code when it has been executed 100 times. So you could execute it 100 times without timing. After that its has been compiled to native code and you could measure it properly. – Christophe De Troyer May 29, 2014 at 13:56

Is there a way to compensate for the time it takes just to call the method? For example I don't want to count the time it takes for the parameters to be copied over and the return address to be given etc. – Celeritas May 29, 2014 at 17:40

4 This is one of the worst ways of measuring execution times of small pieces of code in Java because (i) does not warm-up the code, making measuerements fluctuate so much you can't effectively measure nothing (ii) can be affected by DCE, CP and other optmizations not present in the real code (iii) the granularity of millis is extremely coarse. Check my answer for a better method. – El Marce Aug 16, 2016 at 8:43

1 @ElMarce's answer is clearly superior in the modern context. – Visionary Software Solutions Feb 6, 2019 at 20:46

35

If the bottleneck is big enough to be observed with a profiler, use a profiler.

If you need more accuracy, the best way of measuring an small piece of code is the use of a Java microbenchmark framework like OpenJDK's JMH or Google's Caliper. I believe they are as simple to use as JUnit and not only you will get more accurate results, but you will gain the expertise from the community to do it right.

Follows a JMH microbenchmark to measure the execution time of `Math.log()`:

```
private double x = Math.PI;

@Benchmark
public void myBenchmark() {
    return Math.log(x)
}
```

Using the `currentMillis()` and `nanoTime()` for measuring has many limitations:

1. They have latency (they also take time to execute) which bias your measurements.
2. They have ***VERY*** limited precision, this means you can mesure things from 26ns to 26ns in linux and 300 or so in Windows has described here
3. The warmup phase is not taken into consideration, making your measurements fluctuate A LOT.

The `currentMillis()` and `nanoTime()` methods in Java can be useful but must be used with **EXTREME CAUTION** or you can get wrong measurements errors like this where the order of the measured snippets influence the measurements or like this where the author wrongly conclude that several millions of operations where performed in less than a ms, when in fact the JMV realised no operations where made and hoisted the code, running no code at all.

Here is a wonderful video explaining how to microbenchmark the right way: https://shipilev.net/#benchmarking

Share Edit Follow edited Aug 25, 2016 at 8:49 answered Dec 3, 2015 at 15:15
El Marce
3,344 ● 1 ● 29 ● 40

6

For quick and dirty time measurement tests, don't use wall-clock time (`System.currentTimeMillis()`). Prefer `System.nanoTime()` instead:

```
public static void main(String... ignored) throws InterruptedException {
    final long then = System.nanoTime();
    TimeUnit.SECONDS.sleep(1);
    final long millis = TimeUnit.NANOSECONDS.toMillis(System.nanoTime() - then);
    System.out.println("Slept for (ms): " + millis); // = something around 1000.
}
```

Share Edit Follow edited Jan 11, 2016 at 18:12 answered Dec 3, 2015 at 15:32
Martin Andersson
19.4k ● 10 ● 91 ● 119

3

You should use a profiler like

- jprofiler
- yourkit

They will easily integrate with any IDE and show whatever detail you need.

Of course these tools are complex and meant to be used to profile complex programs, if you need just some simple benchmarks I suggest you to use `System.currentTimeMillis()` or `System.nanoTime()` and calculate delta of millisecs between calls by yourself.

Share Edit Follow answered Mar 31, 2010 at 13:40
Jack
133k ● 33 ● 247 ● 347

I see, it is quite a small program, nothing more than 1500 lines, which would be better a dedicated tool or System.currentTimeMillis()? – KP65 Mar 31, 2010 at 13:41

If you just need to check how fast some methods perform go with **System.currentTimeMillis()**, but mind that it's not precise at all! You can have quantization of tenths of millisecs and similar issues. Profiling is far more the best way to do what you need to do but it needs a little bit of learning. – Jack Mar 31, 2010 at 14:17

3

Using a profiler is better because you can find out average execution times and bottlenecks in your app.

I use VisualVM. slick and simple.

Share Edit Follow

answered Mar 31, 2010 at 13:44
Midhat
17.8k ● 22 ● 90 ● 116

3

Google Guava has a stopwatch, makes things simple and easy.

```
Stopwatch stopwatch = Stopwatch.createStarted();
myFunctionCall();
LOGGER.debug("Time taken by myFunctionCall: " + stopwatch.stop());
```

Share Edit Follow

answered Jul 15, 2015 at 22:29
zengr
38.8k ● 38 ● 132 ● 195

1 Okay but how accurate this is? – Arefe Jun 12, 2020 at 9:43

1

Jprofiler and yourkit are good, but cost money.

There is a free plugin for eclispe called TPTP (Test & Performance Tools Platform) That can give you code execution times. Here is a tutorial that a quick google search brought up. http://www.eclipse.org/articles/Article-TPTP-Profiling-Tool/tptpProfilingArticle.html

Share Edit Follow

answered Mar 31, 2010 at 13:44
thattmatt
41 ● 1 ● 3

0

Another Custom made solution could be based on the the following post : http://www.mkyong.com/spring/spring-aop-examples-advice/

You then have also the possibility to use the utilities around application monitoring & snmp. If you need to "time" your methods on a regular basis in a production environment, you proabably should consider using one of the those SNMP tools

Share Edit Follow

answered Nov 14, 2013 at 17:59

user2991363
1

0

Usually I store the time in a .txt file for analise the outcome

```
StopWatch sWatch = new StopWatch();
sWatch.start();

//do stuff that you want to measure
downloadContent();

sWatch.stop();

//make the time pretty
long timeInMilliseconds = sWatch.getTime();
long hours = TimeUnit.MILLISECONDS.toHours(timeInMilliseconds);
long minutes = TimeUnit.MILLISECONDS.toMinutes(timeInMilliseconds - 
TimeUnit.HOURS.toMillis(hours));
long seconds = TimeUnit.MILLISECONDS.toSeconds(timeInMilliseconds - 
TimeUnit.HOURS.toMillis(hours) - TimeUnit.MINUTES.toMillis(minutes));
long milliseconds = timeInMilliseconds - TimeUnit.HOURS.toMillis(hours) - 
TimeUnit.MINUTES.toMillis(minutes) - TimeUnit.SECONDS.toMillis(seconds);

String t = String.format("%02d:%02d:%02d:%d", hours, minutes, seconds, 
milliseconds);

//each line to store in a txt file, new line
String content = "Ref: " + ref + "  -  " + t + "\r\n";

//you may want wrap this section with a try catch
File file = new File("C:\\time_log.txt");
FileWriter fw = new FileWriter(file.getAbsoluteFile(), true); //append content 
set to true, so it does not overwrite existing data
BufferedWriter bw = new BufferedWriter(fw);
bw.write(content);
bw.close();
```

Share Edit Follow

answered Feb 26, 2021 at 3:03

Philippe
535 ● 1 ● 7 ● 18

-1

You can add this code and it will tell you how long the method took to execute.

```
long millisecondsStart = System.currentTimeMillis();
executeMethod();
long timeSpentInMilliseconds = System.currentTimeMillis() - millisecondsStart;
```

Share Edit Follow

answered Mar 31, 2010 at 13:41

Shervin Asgari
24.5k ● 32 ● 107 ● 144

13 Three problems. 1) the millisecond time may be quantized; e.g. to 20ms granularity. 2) you are actually measuring the time of `executeMethod()` + the time of `System.currentTimeMillis()`. 3) this does not account for JVM warmup effects; e.g. JIT compilation. – Stephen C Mar 31, 2010 at 13:52