How do I time a method's execution in Java?

Asked 16 years, 2 months ago Modified 6 months ago

Viewed 980k times

1037

1. How do I get a method's execution time?
2. Is there a `Timer` utility class for things like timing how long a task takes, etc?

Most of the searches on Google return results for timers that schedule threads and tasks, which is not what I want.

java timing execution-time

Share

Improve this question

Follow

edited May 25, 2020 at 8:13

nabster

1,665 ●2 ●21 ●33

asked Oct 7, 2008 at 20:10

Ogre Psalm33

21.9k ●17 ●77 ●92

JAMon API is a free, simple, high performance, thread safe, Java API that allows developers to easily monitor the performance and scalability of production applications. JAMon tracks hits, execution times (total, avg, min, max, std dev), and more. http://jamonapi.sourceforge.net/ download :

http://sourceforge.net/project/showfiles.php?group_id=96550 – Mike Pone Oct 7, 2008 at 21:47

2 You might also want to look at the Apache Commons Lang StopWatch class. A simple but useful utility class. – user101561 May 5, 2009 at 13:10

Later similar Question: How do I write a correct micro-benchmark in Java? – Basil Bourque Jul 1, 2016 at 22:48

Java 8 using `Instant` class: stackoverflow.com/a/30975902/1216775 – akhil_mittal Aug 1, 2019 at 11:36

If you come here to write benchmarks, know this: benchmarking on the JVM is hard to get right because of dynamic optimizations. Since Java12, the so-called "Microbenchmarking Harness" is provided to alleviate these problems. – julaine May 31 at 9:18

43 Answers

Sorted by: Highest score (default)

1 2 Next

1469

There is always the old-fashioned way:

```
long startTime = System.nanoTime();
methodToTime();
long endTime = System.nanoTime();

long duration = (endTime - startTime);  //divide by 10
milliseconds.
```

Share Improve this answer Follow

edited Jul 8, 2016 at 21:49

michaelsnowden
6,182 ● 2 ● 42 ● 88

answered Oct 7, 2008 at 20:16

Diastrophism
15.4k ● 1 ● 18 ● 7

277 actually, its "new-fashioned" because you used nanoTime, which wasn't added until java5 – John Gardner Oct 7, 2008 at 22:26

11 This (or using System.currentTimeMillis()) seems to be the way it's usually done in Java...that I've seen anyway. It still mildly suprises me that there's no spiffy built-in class, like Timer t = new Timer(); String s = t.getElapsed(format); etc... – Ogre Psalm33 Oct 8, 2008 at 12:48

18 nanoTime does not guarantee accuracy better than currentTimeMillis(), though it usually does. forums.sun.com/thread.jspa?messageID=9460663 and simongbrown.com/blog/2007/08/20/… – James Schek Oct 8, 2008 at 17:20

11 Of course, it's always important to remember the pitfalls of micro-benchmarking, such as compiler/JVM optimizations that may distort the result =8-) – Yuval Jan 7, 2009 at 15:26

18 There is no need for a finally block as endTime won't be used if an exception is thrown. – Peter Lawrey May 5, 2009 at 19:42

258

I go with the simple answer. Works for me.

```
long startTime = System.currentTimeMillis();

doReallyLongThing();

long endTime = System.currentTimeMillis();
```

```
System.out.println("That took " + (endTime - startTime
```

It works quite well. The resolution is obviously only to the millisecond, you can do better with System.nanoTime(). There are some limitations to both (operating system schedule slices, etc.) but this works pretty well.

Average across a couple of runs (the more the better) and you'll get a decent idea.

Share Improve this answer Follow

answered Oct 7, 2008 at 20:14

MBCook
14.5k ● 7 ● 39 ● 41

64 Actually, System.currentTimeMillis() is only accurate above 15ms. For really low values it can't be trusted. The solution for this (as mentioned) is System.nanoTime(); – Steve g Oct 7, 2008 at 21:38

Ok, I was about to accept this as the official answer until I read Steve g's comment. Great tidbit, Steve! – Ogre Psalm33 Oct 8, 2008 at 12:15

6 nanoTime() does not guarantee accuracy better than currentTimeMillis, but many JVM implementations do have better accuracy with nanoTime. – James Schek Oct 8, 2008 at 17:22

8 @JamesSchek You really need to watch your wording, as I already mentioned to this identical comment elsewhere; `nanoTime` is guaranteed to be *at least as resolute* as `currentTimeMillis`. docs.oracle.com/javase/7/docs/api/java/lang/… – arkon May 18, 2013 at 15:37

1 The one slight advantage of `currentTimeMillis` is that it's an actual timestamp, and could be used to log start/end times as well, while `nanoTime` "can only be used to measure elapsed time and is not related to any other notion of system or wall-clock time." – Brad Parks Nov 28, 2016 at 14:17

204

Come on guys! Nobody mentioned the Guava way to do that (which is arguably awesome):

```
import com.google.common.base.Stopwatch;

Stopwatch timer = Stopwatch.createStarted();
//method invocation
LOG.info("Method took: " + timer.stop());
```

The nice thing is that Stopwatch.toString() does a good job of selecting time units for the measurement. I.e. if the value is small, it'll output 38 ns, if it's long, it'll show 5m 3s

Even nicer:

```
Stopwatch timer = Stopwatch.createUnstarted();
for (...) {
   timer.start();
   methodToTrackTimeFor();
   timer.stop();
   methodNotToTrackTimeFor();
}
LOG.info("Method took: " + timer);
```

Note: Google Guava requires Java 1.6+

Share Improve this answer
Follow

edited Jun 25, 2014 at 2:37
bramp
9,721 ● 5 ● 42 ● 47

answered Mar 13, 2013 at 20:11
Dmitry Kalashnikov
2,041 ● 1 ● 12 ● 3

36 Unfortunately, Guava's Stopwatch isn't thread-safe. i learned this the hard way. – Dexter Legaspi Dec 15, 2014 at 14:50

6 @DexterLegaspi Would be very interested in your experience! Care to share? – Siddhartha Jan 19, 2015 at 19:30

1 Using stopwatch in parallel would lead to you calling `start()` multiple times in a row (same for `stop()`). – Mingwei Samuel Aug 16, 2019 at 17:36

Stopwatch/Ticker is wrapper around `System.nanoTime()`, so +1 – Charlie Reitzel Jul 26, 2023 at 18:30

174

Using Instant and Duration from Java 8's new API,

```
Instant start = Instant.now();
Thread.sleep(5000);
Instant end = Instant.now();
System.out.println(Duration.between(start, end));
```

outputs,

```
PT5S
```

Share Improve this answer

Follow

edited Jun 22, 2015 at 9:03

beldaz

4,461 ● 3 ● 47 ● 65

answered Feb 2, 2015 at 9:19

Sufiyan Ghori

18.7k ● 17 ● 83 ● 111

2 Thanks, How can I output the result without having the PT in front? – java123999 Mar 16, 2016 at 15:31

1 The problem with method is that Instant does not problem milli and nano second precision. Ref: stackoverflow.com/questions/20689055/… – prashantsunkari Apr 12, 2017 at 22:56

10 @java123999: You can call `Duration.between(start, end).getSeconds()`. `Duration` also has methods to convert to other time units, e.g. `toMillis()` which converts to milliseconds. – Emil Lunde May 29, 2018 at 13:29

165

Gathered all possible ways together into one place.

Date

```
Date startDate = Calendar.getInstance().getTime();
long d_StartTime = new Date().getTime();
Thread.sleep(1000 * 4);
Date endDate = Calendar.getInstance().getTime();
long d_endTime = new Date().getTime();
System.out.format("StartDate : %s, EndDate : %s \n", s
System.out.format("Milli = %s, ( D_Start : %s, D_End :
d_StartTime),d_StartTime, d_endTime);
```

System.currentTimeMillis()

```
long startTime = System.currentTimeMillis();
Thread.sleep(1000 * 4);
long endTime = System.currentTimeMillis();
long duration = (endTime - startTime);
System.out.format("Milli = %s, ( S_Start : %s, S_End :
startTime, endTime );
System.out.println("Human-Readable format : "+millisTo
```

Human Readable Format

```
public static String millisToShortDHMS(long duration)
    String res = "";    // java.util.concurrent.TimeUn
    long days       = TimeUnit.MILLISECONDS.toDays(dur
    long hours      = TimeUnit.MILLISECONDS.toHours(du

TimeUnit.DAYS.toHours(TimeUnit.MILLISECONDS.toDays(dur
    long minutes    = TimeUnit.MILLISECONDS.toMinutes(

TimeUnit.HOURS.toMinutes(TimeUnit.MILLISECONDS.toHours
    long seconds    = TimeUnit.MILLISECONDS.toSeconds(

TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMin
    long millis     = TimeUnit.MILLISECONDS.toMillis(d

TimeUnit.SECONDS.toMillis(TimeUnit.MILLISECONDS.toSeco

    if (days == 0)      res = String.format("%02d:%02d
minutes, seconds, millis);
    else                res = String.format("%dd %02d:
hours, minutes, seconds, millis);
    return res;
}
```

Guava: Google StopwatchJAR « An object of Stopwatch is to measures elapsed time in nanoseconds.

```
com.google.common.base.Stopwatch g_SW = Stopwatch.crea
g_SW.start();
Thread.sleep(1000 * 4);
g_SW.stop();
System.out.println("Google StopWatch  : "+g_SW);
```

Apache Commons LangJAR **«** **StopWatch** provides a convenient API for timings.

```
org.apache.commons.lang3.time.StopWatch sw = new StopW
sw.start();
Thread.sleep(1000 * 4);
sw.stop();
System.out.println("Apache StopWatch  : "+ millisToSho
```

JODA-TIME

```
public static void jodaTime() throws InterruptedExcept
    java.text.SimpleDateFormat ms_SDF = new SimpleDate
HH:mm:ss.SSS");
    String start = ms_SDF.format( new Date() ); // jav

    Thread.sleep(10000);

    String end = ms_SDF.format( new Date() );
    System.out.println("Start:"+start+"\t Stop:"+end);

    Date date_1 = ms_SDF.parse(start);
    Date date_2 = ms_SDF.parse(end);
    Interval interval = new org.joda.time.Interval( da
date_2.getTime() );
    Period period = interval.toPeriod(); //org.joda.ti

    System.out.format("%dY/%dM/%dD, %02d:%02d:%02d.%04
        period.getYears(), period.getMonths(), period.
        period.getHours(), period.getMinutes(), period
period.getMillis());
}
```

Java date time API from Java 8 « A Duration object represents a period of time between two Instant objects.

```
Instant start = java.time.Instant.now();
    Thread.sleep(1000);
Instant end = java.time.Instant.now();
Duration between = java.time.Duration.between(start, e
System.out.println( between ); // PT1.001S
System.out.format("%dD, %02d:%02d:%02d.%04d \n", betwe
        between.toHours(), between.toMinutes(), betwee
between.toMillis()); // 0D, 00:00:01.1001
```

Spring Framework provides StopWatch utility class to measure elapsed time in Java.

```
StopWatch sw = new org.springframework.util.StopWatch(
sw.start("Method-1"); // Start a named task
    Thread.sleep(500);
sw.stop();

sw.start("Method-2");
    Thread.sleep(300);
sw.stop();

sw.start("Method-3");
    Thread.sleep(200);
sw.stop();

System.out.println("Total time in milliseconds for all
:\n"+sw.getTotalTimeMillis());
System.out.println("Table describing all tasks perform
:\n"+sw.prettyPrint());

System.out.format("Time taken by the last task : [%s]:
        sw.getLastTaskName(),sw.getLastTaskTimeMillis(

System.out.println("\n Array of the data for tasks per
Taken");
TaskInfo[] listofTasks = sw.getTaskInfo();
```

```
for (TaskInfo task : listofTasks) {
    System.out.format("[%s]:[%d]\n",
            task.getTaskName(), task.getTimeMillis());
}
```

OutPut:

```
Total time in milliseconds for all tasks :
999
Table describing all tasks performed :
StopWatch '': running time (millis) = 999
-----------------------------------------
ms     %     Task name
-----------------------------------------
00500  050%  Method-1
00299  030%  Method-2
00200  020%  Method-3

Time taken by the last task : [Method-3]:[200]
 Array of the data for tasks performed « Task Name: Ti
[Method-1]:[500]
[Method-2]:[299]
[Method-3]:[200]
```

Share Improve this answer Follow

edited Apr 26, 2018 at 9:20

answered Dec 4, 2015 at 10:43

Yash
9,548 ● 2 ● 73 ● 79

Stopwatch of Guava, Apache Commons and Spring Framework are not thread safe. Not safe for production usage. – Deepak Puthraya Jun 12, 2018 at 19:25

@DeepakPuthraya then which library to use which is safe for production usage? – Gaurav Sep 24, 2019 at 7:24

1 @DeepakPuthraya you can use java 8 provided Java date time API. Which is simple. – Yash Sep 24, 2019 at 7:48

2 IMO this post would benefit if every solution would also show the output of the system outs. – BAERUS Apr 28, 2020 at 15:23

1 `new Date().getTime()` is just a `System.currentTimeMillis()` in disguise. `new Date()` does the same as `new Date(System.currentTimeMillis())` and `getTime()` will just return that `long` value (and that's the only feature of this class that has not been marked as deprecated). Likewise, `Calendar.getInstance().getTime()` produces exactly the same as `new Date()`, but with significantly more overhead, as it will prepare things that this code snippet doesn't use. – Holger Nov 3, 2022 at 16:46

92

Use a profiler (JProfiler, Netbeans Profiler, Visual VM, Eclipse Profiler, etc). You'll get the most accurate results and is the least intrusive. They use the built-in JVM mechanism for profiling which can also give you extra information like stack traces, execution paths, and more comprehensive results if necessary.

When using a fully integrated profiler, it's faily trivial to profile a method. Right click, Profiler -> Add to Root Methods. Then run the profiler just like you were doing a test run or debugger.

Share Improve this answer edited Oct 7, 2008 at 23:24

Follow

answered Oct 7, 2008 at 21:35

James Schek
18k ● 7 ● 53 ● 64

This was also a great suggestion, and one of those "duh" light-bulb moments for me when I read this answer. Our project uses JDeveloper, but I checked, and sure enough, it's got a built-in profiler! – Ogre Psalm33 Oct 8, 2008 at 12:14

2 From java 7 build 40 (i think) they included the former JRockits Flight Recorder to java (search for Java Mission Control) – Niels Bech Nielsen Mar 17, 2014 at 13:30

Sure enough @NielsBechNielsen! oracle.com/technetwork/java/javaseproducts/mission-control/… – Ogre Psalm33 Jul 7, 2014 at 14:43

52

`System.currentTimeMillis();` IS NOT a good approach for measuring the performance of your algorithms. It measures the total time you experience as a user watching the computer screen. It includes also time consumed by everything else running on your computer in the background. This could make a huge difference in case you have a lot of programs running on your workstation.

Proper approach is using `java.lang.management` package.

From
http://nadeausoftware.com/articles/2008/03/java_tip_how_get_cpu_and_user_time_benchmarking website
(archive link):

- "User time" is the time spent running your application's own code.
- "System time" is the time spent running OS code on behalf of your application (such as for I/O).

`getCpuTime()` method gives you sum of those:

```
import java.lang.management.ManagementFactory;
import java.lang.management.ThreadMXBean;

public class CPUUtils {

    /** Get CPU time in nanoseconds. */
    public static long getCpuTime( ) {
        ThreadMXBean bean = ManagementFactory.getThrea
        return bean.isCurrentThreadCpuTimeSupported( )
            bean.getCurrentThreadCpuTime( ) : 0L;
    }

    /** Get user time in nanoseconds. */
    public static long getUserTime( ) {
        ThreadMXBean bean = ManagementFactory.getThrea
        return bean.isCurrentThreadCpuTimeSupported( )
            bean.getCurrentThreadUserTime( ) : 0L;
    }

    /** Get system time in nanoseconds. */
    public static long getSystemTime( ) {
        ThreadMXBean bean = ManagementFactory.getThrea
        return bean.isCurrentThreadCpuTimeSupported( )
            (bean.getCurrentThreadCpuTime( ) - bean.ge
)) : 0L;
    }
```

```
}
```

Share Improve this answer

Follow

edited Jun 16, 2020 at 16:15

Joshua Goldberg
5,303 ● 3 ● 36 ● 41

answered Apr 6, 2014 at 13:46

TondaCZE
2,680 ● 1 ● 20 ● 20

5 This is definitely a good point, that "user time" (wall-clock time) is not always a great measure of performance, especially in a multi-threaded program. – Ogre Psalm33 Apr 7, 2014 at 12:52

This is the answer I am looking for. – ZhaoGang Jan 6, 2020 at 2:14

Agree and disagree on "user time" – sometimes it's correct to measure how long the code itself takes, and exclude wall-clock time; but other times the total elapsed time is what should be measured. – Kaan Jun 18, 2022 at 17:20

Elapsed time using `System.nanoTime()` is the best approach, imo, for measuring single thread performance. Almost always, you can tune the code used by each thread individually and measure the impact on overall throughput using coarser methods, e.g. you can index 50 GB/hour using 10 threads vs. 10 GB/hour with 1 thread or whatever. – Charlie Reitzel Jul 26, 2023 at 18:43

This probably isn't what you wanted me to say, but this is a good use of AOP. Whip an proxy interceptor around your method, and do the timing in there.

43

The what, why and how of AOP is rather beyond the scope of this answer, sadly, but that's how I'd likely do it.

Edit: Here's a link to Spring AOP to get you started, if you're keen. This is the most accessible implementation of AOP that Iive come across for java.

Also, given everyone else's very simple suggestions, I should add that AOP is for when you don't want stuff like timing to invade your code. But in many cases, that sort of simple and easy approach is fine.

Share Improve this answer Follow

edited Jan 26, 2012 at 8:37
Buhake Sindi
89.1k ● 30 ● 174 ● 232

answered Oct 7, 2008 at 20:13
skaffman
403k ● 96 ● 824 ● 774

4 Here is a tutorial on how to do this with Spring: veerasundar.com/blog/2010/01/… – David Tinker Jan 4, 2011 at 11:05

35

With Java 8 you can do also something like this with every normal **methods**:

```
Object returnValue = TimeIt.printTime(() -> methodeWit
//do stuff with your returnValue
```

with TimeIt like:

```
public class TimeIt {

public static <T> T printTime(Callable<T> task) {
    T call = null;
    try {
        long startTime = System.currentTimeMillis();
        call = task.call();
        System.out.print((System.currentTimeMillis() -
"s");
    } catch (Exception e) {
        //...
    }
    return call;
}
}
```

With this methode you can make easy time measurement anywhere in your code without breaking it. In this simple example i just print the time. May you add a Switch for TimeIt, e.g. to only print the time in DebugMode or something.

If you are working with **Function** you can do somthing like this:

```
Function<Integer, Integer> yourFunction= (n) -> {
        return IntStream.range(0, n).reduce(0, (a, b)
    };

Integer returnValue = TimeIt.printTime2(yourFunction).
//do stuff with your returnValue

public static <T, R> Function<T, R> printTime2(Functio
    return (t) -> {
        long startTime = System.currentTimeMillis();
        R apply = task.apply(t);
        System.out.print((System.currentTimeMillis() -
                + "s");
        return apply;
    };
}
```

Share Improve this answer Follow

edited Nov 28, 2015 at 19:36

answered Nov 25, 2015 at 22:47

Stefan
448 ● 4 ● 9

This looks much better than other solutions. Its closer to Spring AOP yet lighter than that. True java 8 way! +1 Thanks! – Amit Kumar Dec 5, 2017 at 14:09

Maybe this looks good to you, because Stefan is using fancy new java functions. But I think this is needlessly difficult to read and understand. – Stimpson Cat Jul 5, 2019 at 9:25

21

Also We can use StopWatch class of Apache commons for measuring the time.

Sample code

```
org.apache.commons.lang.time.StopWatch sw = new
org.apache.commons.lang.time.StopWatch();

System.out.println("getEventFilterTreeData :: Start Ti
sw.start();

// Method execution code

sw.stop();
System.out.println("getEventFilterTreeData :: End Time
```

Share Improve this answer

Follow

edited Dec 3, 2011 at 4:40

animuson ♦
54.7k ● 28 ● 141 ● 150

answered Dec 3, 2011 at 4:37

Narayan
211 ● 2 ● 2

21

JEP 230: Microbenchmark Suite

FYI, JEP 230: Microbenchmark Suite is an OpenJDK project to:

> Add a basic suite of microbenchmarks to the JDK source code, and make it easy for developers to run existing microbenchmarks and create new ones.

This feature arrived in Java 12.

Java Microbenchmark Harness (JMH)

For earlier versions of Java, take a look at the Java Microbenchmark Harness (JMH) project on which JEP 230 is based.

Share Improve this answer Follow

edited May 13, 2019 at 14:25

answered Jul 1, 2016 at 22:59

Basil Bourque
336k ● 119 ● 917 ● 1.2k

2 This should be way higher, everyone here answers the literal question without thinking about why someone would time a single method execution and what problems they may face. People who rely on the highly-voted answers will write some terrible benchmarks. – julaine May 31 at 9:15

15

We are using AspectJ and Java annotations for this purpose. If we need to know to execution time for a method, we simple annotate it. A more advanced version could use an own log level that can enabled and disabled at runtime.

```
public @interface Trace {
  boolean showParameters();
}

@Aspect
public class TraceAspect {
  [...]
  @Around("tracePointcut() && @annotation(trace) && !w
  public Object traceAdvice ( ProceedingJintPoint jP,

    Object result;
    // initilize timer

    try {
      result = jp.procced();
      } finally {
        // calculate execution time
      }

      return result;
    }
    [...]
  }
```

Share Improve this answer

Follow

answered Oct 8, 2008 at 6:40

murdochjohn

15

Just a small twist, if you don't use tooling and want to time methods with low execution time: execute it many times, each time doubling the number of times it is executed until you reach a second, or so. Thus, the time of the Call to System.nanoTime and so forth, nor the accuracy of System.nanoTime does affect the result much.

```
    int runs = 0, runsPerRound = 10;
    long begin = System.nanoTime(), end;
    do {
        for (int i=0; i<runsPerRound; ++i) timedMethod
        end = System.nanoTime();
        runs += runsPerRound;
        runsPerRound *= 2;
    } while (runs < Integer.MAX_VALUE / 2 && 100000000
    System.out.println("Time for timedMethod() is " +
        0.000000001 * (end-begin) / runs + " seconds")
```

Of course, the caveats about using the wall clock apply: influences of JIT-compilation, multiple threads / processes etc. Thus, you need to first execute the method *a lot* of times first, such that the JIT compiler does its work, and then repeat this test multiple times and take the lowest execution time.

Share Improve this answer Follow

edited Jul 15, 2015 at 9:47

answered Nov 5, 2008 at 7:20

Dr. Hans-Peter Störr
25.9k ● 30 ● 107 ● 144

12

Really good code.

http://www.rgagnon.com/javadetails/java-0585.html

```
import java.util.concurrent.TimeUnit;

long startTime = System.currentTimeMillis();
........
........
........
long finishTime = System.currentTimeMillis();

String diff = millisToShortDHMS(finishTime - startTime


  /**
   * converts time (in milliseconds) to human-readable
   *  "<dd:>hh:mm:ss"
   */
  public static String millisToShortDHMS(long duration
    String res = "";
    long days  = TimeUnit.MILLISECONDS.toDays(duration
    long hours = TimeUnit.MILLISECONDS.toHours(duratio
                   -
TimeUnit.DAYS.toHours(TimeUnit.MILLISECONDS.toDays(dur
    long minutes = TimeUnit.MILLISECONDS.toMinutes(dur
                        -
TimeUnit.HOURS.toMinutes(TimeUnit.MILLISECONDS.toHours
    long seconds = TimeUnit.MILLISECONDS.toSeconds(dur
                      -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMin
    if (days == 0) {
      res = String.format("%02d:%02d:%02d", hours, min
    }
    else {
      res = String.format("%dd%02d:%02d:%02d", days, h
    }
    return res;
  }
```

Share Improve this answer

Follow

answered Jun 15, 2012 at 8:50

iceberg
1,961 ● 1 ● 22 ● 26

3 Actually the question was how to calculate the amount of time a method takes, not how to format it. However this question is quite old (almost four years!). Try to avoid resurrecting old threads unless the response will add something new and significant over existing responses.
– Leigh Jun 15, 2012 at 8:58

1 And to add remaining millis to the end, make the following changes: `long millis = TimeUnit.MILLISECONDS.toMillis(duration) - TimeUnit.SECONDS.toMillis(TimeUnit.MILLISECONDS.toSeconds(duration)); if (days == 0) {   res = String.format("%02d:%02d:%02d.%02d", hours, minutes, seconds, millis); } else {   res = String.format("%dd%02d:%02d:%02d.%02d", days, hours, minutes, seconds, millis); }`
– Rick Barkhouse Jan 23, 2013 at 22:19

12

Spring provides a utility class org.springframework.util.StopWatch, as per JavaDoc:

> Simple stop watch, allowing for timing of a number of tasks, exposing total running time and running time for each named task.

Usage:

```
StopWatch stopWatch = new StopWatch("Performance Test

stopWatch.start("Method 1");
doSomething1();//method to test
stopWatch.stop();

stopWatch.start("Method 2");
doSomething2();//method to test
stopWatch.stop();

System.out.println(stopWatch.prettyPrint());
```

Output:

```
StopWatch 'Performance Test Result': running time (mil
-----------------------------------------
ms     %     Task name
-----------------------------------------
11907  036%  Method 1
00922  064%  Method 2
```

With Aspects:

```
@Around("execution(* my.package..*.*(..))")
public Object logTime(ProceedingJoinPoint joinPoint) t
```

```
    StopWatch stopWatch = new StopWatch();
    stopWatch.start();
    Object retVal = joinPoint.proceed();
    stopWatch.stop();
    log.info(" execution time: " + stopWatch.getTotalT
    return retVal;
}
```

Share Improve this answer Follow

edited Oct 21, 2015 at 7:35

answered Aug 6, 2015 at 10:26

Sunil Manheri
2,343 ● 2 ● 19 ● 20

10

You can use Perf4j. Very cool utility. Usage is simple

```
String watchTag = "target.SomeMethod";
StopWatch stopWatch = new LoggingStopWatch(watchTag);
Result result = null; // Result is a type of a return
try {
    result = target.SomeMethod();
    stopWatch.stop(watchTag + ".success");
} catch (Exception e) {
    stopWatch.stop(watchTag + ".fail", "Exception was
    throw e;
}
```

~~More information can be found in Developer Guide~~

Edit: Project seems dead

Share Improve this answer Follow

edited May 25, 2016 at 0:39

Graham Russell
1,047 ● 14 ● 25

answered Feb 26, 2012 at 15:07
mergenchik
1,139 ● 17 ● 28

1 Perf4j can also generate nice statistics. – Paaske Mar 28, 2012 at 12:00

10

I have written a method to print the method execution time in a much readable form. For example, to calculate the factorial of 1 Million, it takes approximately 9 minutes. So the execution time get printed as:

```
Execution Time: 9 Minutes, 36 Seconds, 237 MicroSecond
```

The code is here:

```
public class series
{
    public static void main(String[] args)
    {
        long startTime = System.nanoTime();

        long n = 10_00_000;
        printFactorial(n);

        long endTime = System.nanoTime();
        printExecutionTime(startTime, endTime);

    }

    public static void printExecutionTime(long startTi
    {
        long time_ns = endTime - startTime;
        long time_ms = TimeUnit.NANOSECONDS.toMillis(t
        long time_sec = TimeUnit.NANOSECONDS.toSeconds
        long time_min = TimeUnit.NANOSECONDS.toMinutes
        long time_hour = TimeUnit.NANOSECONDS.toHours(

        System.out.print("\nExecution Time: ");
        if(time_hour > 0)
            System.out.print(time_hour + " Hours, ");
        if(time_min > 0)
            System.out.print(time_min % 60 + " Minutes
        if(time_sec > 0)
            System.out.print(time_sec % 60 + " Seconds
        if(time_ms > 0)
            System.out.print(time_ms % 1E+3 + " MicroS
        if(time_ns > 0)
            System.out.print(time_ns % 1E+6 + " NanoSe
    }
}
```

Share Improve this answer Follow

edited Nov 16, 2018 at 11:13

answered Oct 29, 2018 at 19:18

Pratik Patil
3,753 ● 3 ● 32 ● 33

I think you just missed one time unit. The next unit from seconds is milliseconds and not microseconds.
– Wilson Barbosa Aug 1, 2021 at 4:40

▲
10

In Spring framework we have a call called StopWatch (org.springframework.util.StopWatch)

```
//measuring elapsed time using Spring StopWatch
        StopWatch watch = new StopWatch();
        watch.start();
        for(int i=0; i< 1000; i++){
            Object obj = new Object();
        }
        watch.stop();
        System.out.println("Total execution time to cr
using StopWatch in millis: "
                + watch.getTotalTimeMillis());
```

Share Improve this answer Follow

edited Jul 4, 2022 at 16:31

answered May 16, 2020 at 5:04

Bhaskara Arani
1,637 ● 1 ● 27 ● 44

From the docs: `This class is normally used to verify performance during proof-of-concept work and in development, rather than as part of production applications.` – q99 Jun 17, 2020 at 15:59

@q99 True, this kind of logic we dont put in production envrionments, before moving it to production we need to test – Bhaskara Arani Jun 19, 2020 at 9:05

it uses `System.nanoTime()` under the hood which is not good(applies to `System.currentTimeMillis()` too), see @TondaCZE answer – Eboubaker Nov 21, 2021 at 8:52

```
new Timer(""){{
    // code to time
}}.timeMe();
```

8

```
public class Timer {

    private final String timerName;
    private long started;

    public Timer(String timerName) {
        this.timerName = timerName;
        this.started = System.currentTimeMillis();
    }

    public void timeMe() {
        System.out.println(
        String.format("Execution of '%s' takes %dms.",
                timerName,
                started-System.currentTimeMillis()));
    }

}
```

Share Improve this answer Follow

answered Apr 3, 2013 at 14:13

Maciek Kreft
882 ● 9 ● 14

1 Roll your own simple class is a good choice when you already have the build system and dependent OTS set up, and don't want to bother pulling in another OTS package that includes a utility timer class. – Ogre Psalm33 Apr 3, 2013 at 21:06

8

Using AOP/AspectJ and `@Loggable` annotation from jcabi-aspects you can do it easy and compact:

```
@Loggable(Loggable.DEBUG)
public String getSomeResult() {
    // return some value
}
```

Every call to this method will be sent to SLF4J logging facility with `DEBUG` logging level. And every log message will include execution time.

Share Improve this answer Follow

edited Jan 19, 2015 at 8:24

answered Jan 6, 2013 at 20:12

yegor256
105k ● 130 ● 460 ● 620

8

You can use Metrics library which provides various measuring instruments. Add dependency:

```
<dependencies>
    <dependency>
        <groupId>io.dropwizard.metrics</groupId>
        <artifactId>metrics-core</artifactId>
        <version>${metrics.version}</version>
    </dependency>
</dependencies>
```

And configure it for your environment.

Methods can be annotated with @Timed:

```
@Timed
public void exampleMethod(){
    // some code
}
```

or piece of code wrapped with Timer:

```
final Timer timer = metricsRegistry.timer("some_name")
final Timer.Context context = timer.time();
// timed code
context.stop();
```

Aggregated metrics can exported to console, JMX, CSV or other.

`@Timed` metrics output example:

```
com.example.ExampleService.exampleMethod
             count = 2
         mean rate = 3.11 calls/minute
     1-minute rate = 0.96 calls/minute
     5-minute rate = 0.20 calls/minute
    15-minute rate = 0.07 calls/minute
               min = 17.01 milliseconds
               max = 1006.68 milliseconds
              mean = 511.84 milliseconds
            stddev = 699.80 milliseconds
            median = 511.84 milliseconds
              75% <= 1006.68 milliseconds
              95% <= 1006.68 milliseconds
              98% <= 1006.68 milliseconds
              99% <= 1006.68 milliseconds
            99.9% <= 1006.68 milliseconds
```

Share Improve this answer Follow

answered Jul 31, 2017 at 10:53

Justinas Jakavonis
8,758 ● 12 ● 79 ● 117

7

I basically do variations of this, but considering how hotspot compilation works, if you want to get accurate results you need to throw out the first few measurements and make sure you are using the method in a real world (read application specific) application.

If the JIT decides to compile it your numbers will vary heavily. so just be aware

Share Improve this answer Follow

answered Oct 7, 2008 at 20:17

luke
14.8k ● 5 ● 49 ● 57

7

There are a couple of ways to do that. I normally fall back to just using something like this:

```
long start = System.currentTimeMillis();
// ... do something ...
long end = System.currentTimeMillis();
```

or the same thing with System.nanoTime();

For something more on the benchmarking side of things there seems also to be this one: http://jetm.void.fm/ Never tried it though.

Share Improve this answer Follow

answered Oct 7, 2008 at 20:18

Horst Gutmann
11.3k ● 2 ● 30 ● 31

6

If you want wall-clock time

```
long start_time = System.currentTimeMillis();
object.method();
long end_time = System.currentTimeMillis();
long execution_time = end_time - start_time;
```

Share Improve this answer

Follow

answered Oct 7, 2008 at 20:14

David Nehme

21.6k ● 8 ● 81 ● 121

5

As "skaffman" said, use AOP OR you can use run time bytecode weaving, just like unit test method coverage tools use to transparently add timing info to methods invoked.

You can look at code used by open source tools tools like Emma (http://downloads.sourceforge.net/emma/emma-2.0.5312-src.zip?modtime=1118607545&big_mirror=0). The other opensource coverage tool is http://prdownloads.sourceforge.net/cobertura/cobertura-1.9-src.zip?download.

If you eventually manage to do what you set out for, pls. share it back with the community here with your ant task/jars.

Share Improve this answer Follow

answered Oct 7, 2008 at 20:21

anjanb
13.8k ● 19 ● 80 ● 106

5

```
long startTime = System.currentTimeMillis();
// code goes here
long finishTime = System.currentTimeMillis();
long elapsedTime = finishTime - startTime; // elapsed
```

Share Improve this answer Follow

edited Oct 7, 2008 at 20:22

answered Oct 7, 2008 at 20:16

Ryan Rodemoyer
5,672 ● 12 ● 46 ● 57

4

I modified the code from correct answer to get result in seconds:

```
long startTime = System.nanoTime();

methodCode ...

long endTime = System.nanoTime();
double duration = (double)(endTime - startTime) / (Mat
Log.v(TAG, "MethodName time (s) = " + duration);
```

Share Improve this answer Follow

answered Dec 27, 2013 at 15:18

Denis Kutlubaev
16.1k ● 6 ● 87 ● 71

4

Ok, this is a simple class to be used for simple simple timing of your functions. There is an example below it.

```
public class Stopwatch {
    static long startTime;
    static long splitTime;
    static long endTime;

    public Stopwatch() {
        start();
    }

    public void start() {
        startTime = System.currentTimeMillis();
        splitTime = System.currentTimeMillis();
        endTime = System.currentTimeMillis();
    }
    public void split() {
        split("");
    }

    public void split(String tag) {
        endTime = System.currentTimeMillis();
        System.out.println("Split time for [" + tag +
splitTime) + " ms");
        splitTime = endTime;
    }

    public void end() {
        end("");
    }
    public void end(String tag) {
        endTime = System.currentTimeMillis();
        System.out.println("Final time for [" + tag +
startTime) + " ms");
    }
}
```

Sample of use:

```
public static Schedule getSchedule(Activity activity_c
        String scheduleJson = null;
        Schedule schedule = null;
/*->*/  Stopwatch stopwatch = new Stopwatch();

        InputStream scheduleJsonInputStream =
activity_context.getResources().openRawResource(R.raw.
/*->*/  stopwatch.split("open raw resource");

        scheduleJson =
FileToString.convertStreamToString(scheduleJsonInputSt
/*->*/  stopwatch.split("file to string");

        schedule = new Gson().fromJson(scheduleJson, S
/*->*/  stopwatch.split("parse Json");
/*->*/  stopwatch.end("Method getSchedule");
```

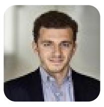
```
    return schedule;
}
```

Sample of console output:

```
Split time for [file to string]: 672 ms
Split time for [parse Json]: 893 ms
Final time for [get Schedule]: 1565 ms
```

Share Improve this answer Follow

answered Jan 27, 2015 at 18:44



msysmilu
2,024 ● 23 ● 24

4

In Java 8 a new class named `Instant` is introduced. As per doc:

> Instant represents the start of a nanosecond on the time line. This class is useful for generating a time stamp to represent machine time. The range of an instant requires the storage of a number larger than a long. To achieve this, the class stores a long representing epoch-seconds and an int representing nanosecond-of-second, which will always be between 0 and 999,999,999. The epoch-seconds are measured from the standard Java epoch of 1970-01-01T00:00:00Z where instants after the epoch have positive values, and earlier instants have negative values. For both the epoch-second and

> nanosecond parts, a larger value is always later on the time-line than a smaller value.

This can be used as:

```
Instant start = Instant.now();
try {
    Thread.sleep(7000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
Instant end = Instant.now();
System.out.println(Duration.between(start, end));
```

It prints `PT7.001S`.

Share Improve this answer

Follow

answered Jun 22, 2015 at 9:02

akhil_mittal

24.1k ● 8 ● 97 ● 96

4

You can use stopwatch class from spring core project:

Code:

```
StopWatch stopWatch = new StopWatch()
stopWatch.start();  //start stopwatch
// write your function or line of code.
stopWatch.stop();  //stop stopwatch
stopWatch.getTotalTimeMillis() ; ///get total time
```

Documentation for Stopwatch: **Simple stop watch, allowing for timing of a number of tasks, exposing total running time and running time for each named**

task. Conceals use of System.currentTimeMillis(), improving the readability of application code and reducing the likelihood of calculation errors. Note that this object is not designed to be thread-safe and does not use synchronization. This class is normally used to verify performance during proof-of-concepts and in development, rather than as part of production applications.

Share Improve this answer

Follow

answered Mar 8, 2018 at 8:18

praveen jain

788 ● 2 ● 8 ● 23

1 2 Next