PERFORMANCE TUNING WITH SBT-JMH

Scala Hamburg Meetup 20.06.2018

ABOUT ME

co-founder of inoio

currently doing distributed event sourcing with eventuate

on twitter: @martin_grotzke

MICROBENCHMARKING

...done right.

OPTIMIZATION

...of the right things.

WHY?

we identified a bottleneck

we want to choose an implementation variant based on its performance

just for fun - faster is better!

MICROBENCHMARKING - THE INTUITIVE APPROACH

```
def bench(name: String, repeat: Int)(fun: => Unit): Unit = {
  val start = currentTimeMillis()
  (0 until repeat) foreach (_ => fun)
  val duration = currentTimeMillis() - start
  print(s"$name: ${repeat/duration} ops/ms")
}
```

```
bench("sqrt", 100) {
  math.sqrt(10000)
}
```

demo time!

OH, JIT!

microbenchmarking pitfalls ahead

INCLUDE JIT ANALYSIS / COMPILATION

LOOP OPTIMIZATIONS

DEAD-CODE ELIMINATION

CONSTANT FOLDING

INLINING, DEOPTIMIZATION

MULTI-THREADING - OH MY!

MORE PITFALLS

noisy neighbors

gc pauses

classloaders

machine architecture

measuring

JMH FOR THE RESCUE

Java **M**icrobenchmarking **H**arness

> *JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targetting the JVM.*

http://openjdk.java.net/projects/code-tools/jmh/

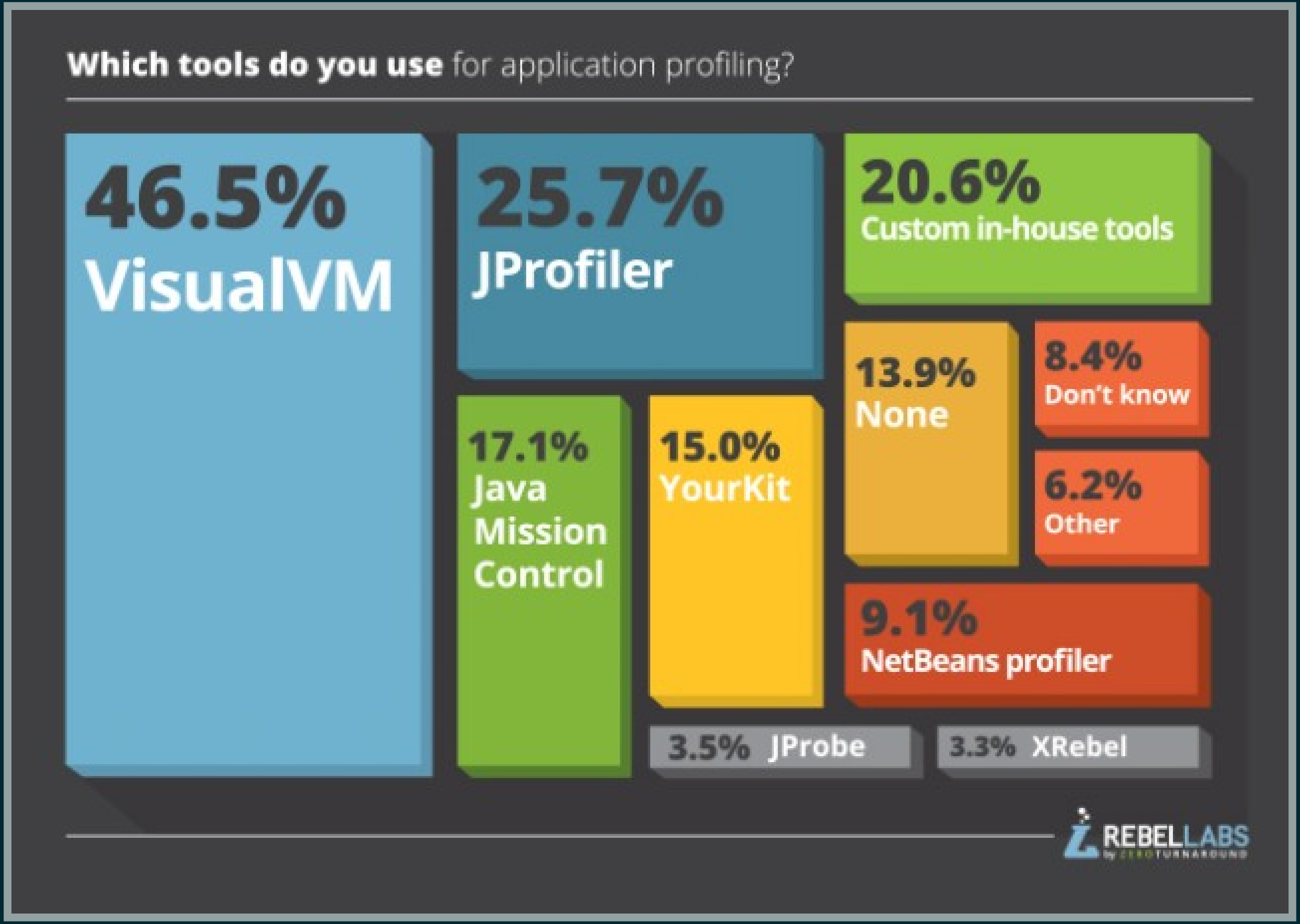
SBT JMH PLUGIN

https://github.com/ktoso/sbt-jmh

demo…!

AWESOME, NUMBERS! NOW WHAT?!

PROFILING?!



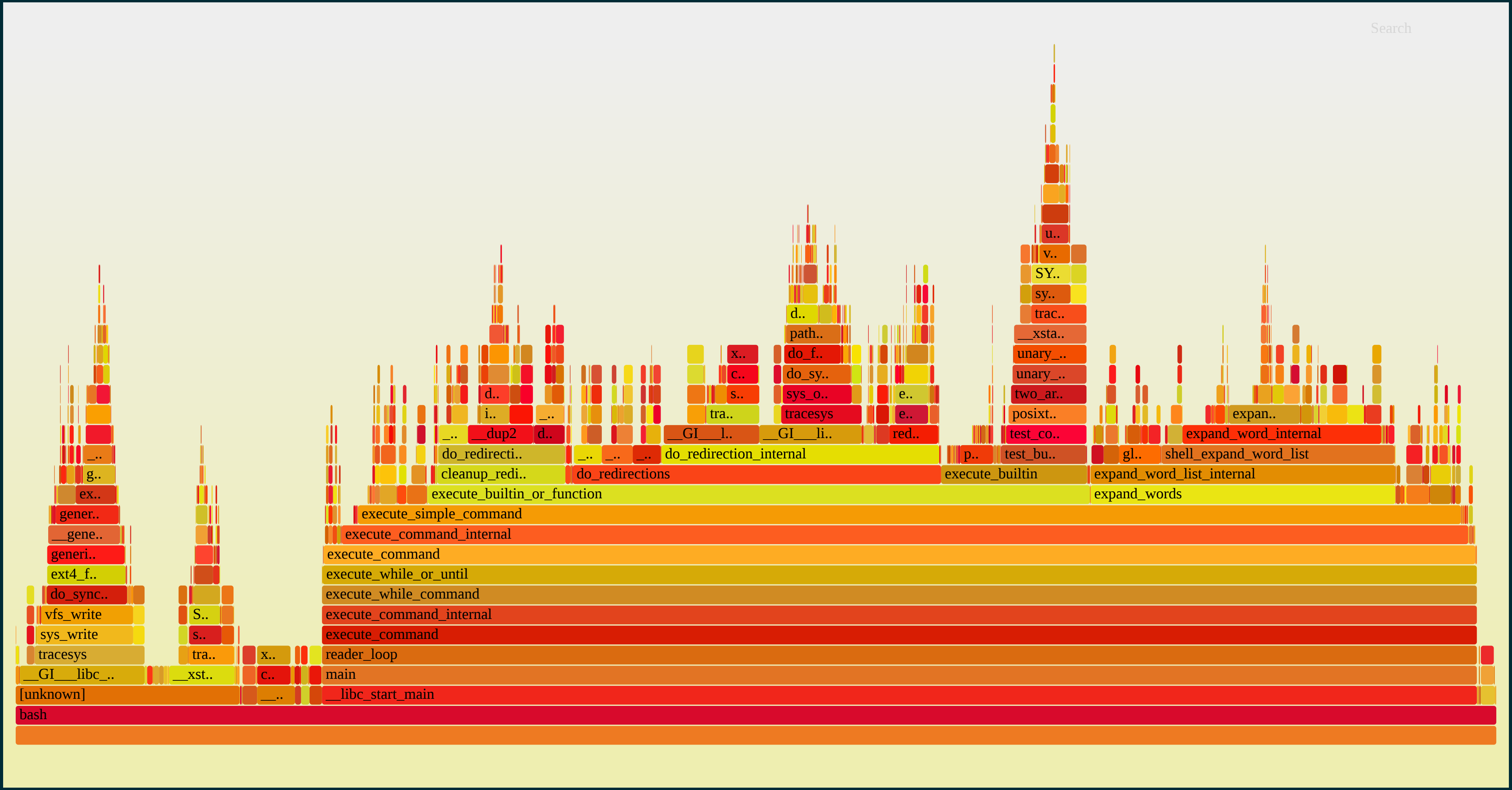
source: survey by zeroturnaround (2015)

ASYNC-PROFILER

> *low overhead sampling profiler for Java that does not suffer from Safepoint bias problem. It features HotSpot-specific APIs to collect stack traces and to track memory allocations.*

https://github.com/jvm-profiling-tools/async-profiler

SUPER AWESOME: FLAME GRAPHS!



USAGE: ASYNC-PROFILER GENERATING FLAME GRAPHS

```
echo 1 > /proc/sys/kernel/perf_event_paranoid
echo 0 > /proc/sys/kernel/kptr_restrict
```

```
PID=$(jps | grep ForkMain | cut -d" " -f 1)
profiler.sh -d 30 -f /tmp/flamegraph.svg $PID
```

demo? demo.

KEY TAKEAWAYS

1. don't build your own `def microbenchmark()`, but use an existing tool!
2. use a low overhead profiler not suffering from safepoint bias.
3. use flame graphs to understand understand your application/code profile.

build - measure - learn

REFERENCES / FURTHER READING

- Brian Goetz: Anatomy of a flawed microbenchmark
- Aleksey Shipilëv: JVM Anatomy Park
- JMH: openjdk.java.net/projects/code-tools/jmh/
- mechanical-sympathy: JMH vs Caliper: reference thread
- sbt-jmh: github.com/ktoso/sbt-jmh
- Nitsan Wakart: Why (Most) Sampling Java Profilers Are Fucking Terrible
- Nitsan Wakart: The Pros and Cons of AsyncGetCallTrace Profilers
- async-profiler: github.com/jvm-profiling-tools/async-profiler
- Brendan Gregg: Flame Graphs