ShardingSphere

# 从 ShardingSphere 性能提升场景探析 Java 性能工具及实践

吴伟杰

July 9, 2022

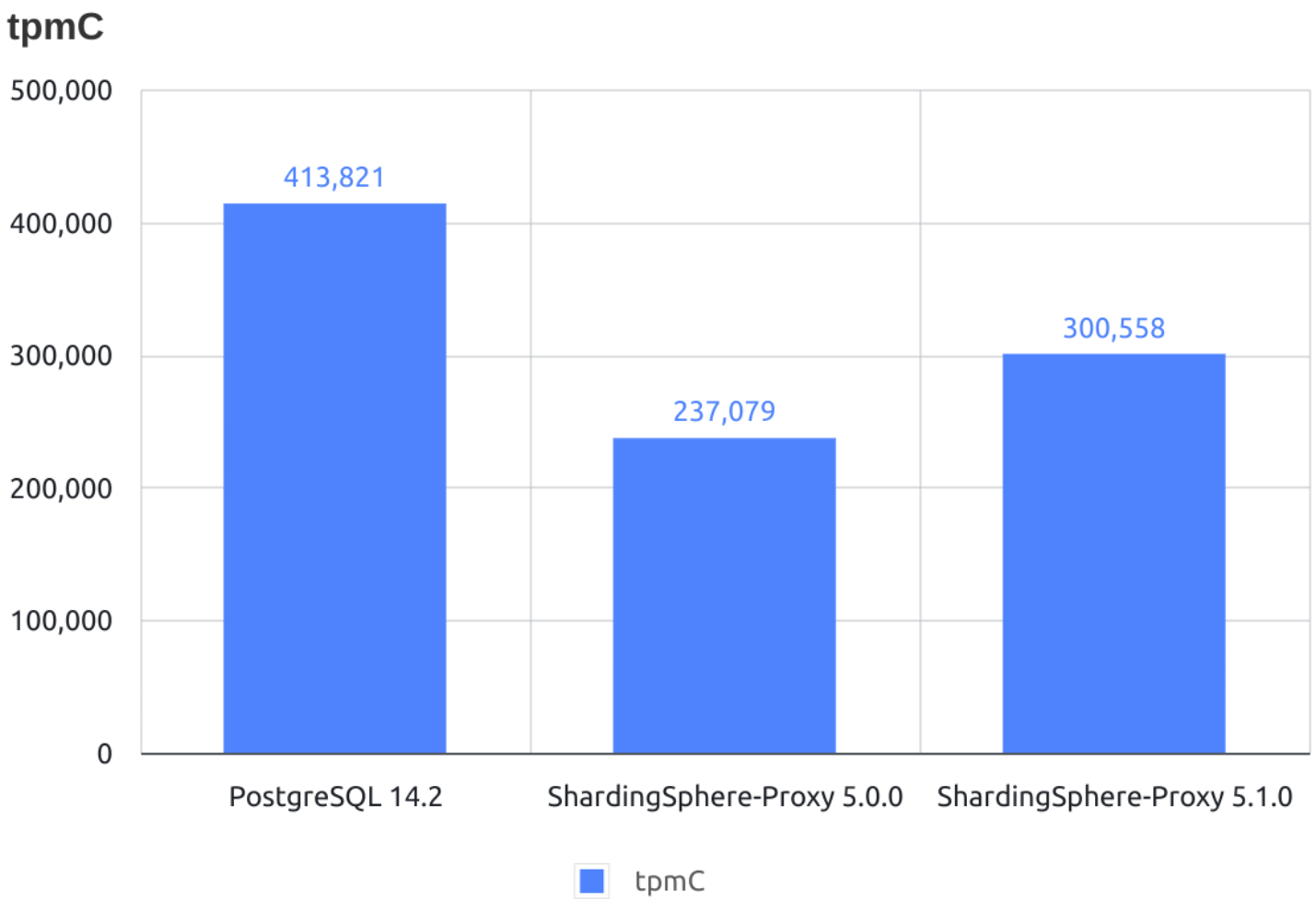

吴伟杰

Apache ShardingSphere PMC
SphereEx 基础设施研发工程师


- 在与 openGauss 社区合作的 16 节点达成 TPC-C 1000 万 tpmC 的性能目标中，参与 ShardingSphere 性能优化
- 目前专注于 ShardingSphere-Proxy 研发

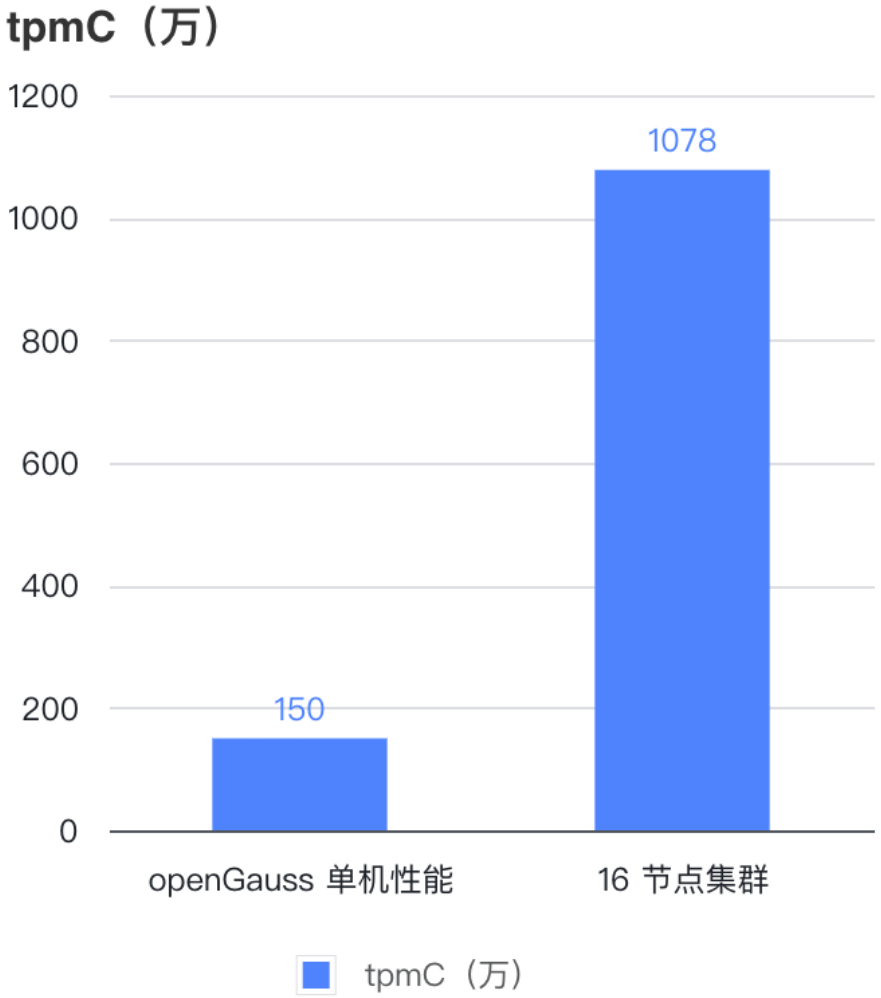
# 目录

# 场景 1：ShardingSphere-Proxy 透传对比直连 PostgreSQL




| | ShardingSphere-Proxy | PostgreSQL | BenchmarkSQL |
|---|---|---|---|
| CPU | 2 * Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz | 2 * Intel(R) Xeon(R) Gold 6146 CPU @ 3.20GHz | 2 * Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz |
| RAM | 96 GB 2400MHz | 512 GB 2666MHz | 96 GB 2400MHz |
| 硬盘 | / | AVAGO SCSI 240GB | / |
| 网卡 | 10 Gb | 10 Gb | 10 Gb |
| 操作系统 | CentOS 7.9 | CentOS 7.9 | CentOS 7.9 |
| 软件环境 | Java 17.0.1 ShardingSphere-Proxy 5.0.0 / 5.1.0 | PostgreSQL 14.2 | BenchmarkSQL 5.0 |
| 其他配置 | 网卡队列绑核 0-1,24-25 ShardingSphere-Proxy 绑核 2-23,26-47 | fsync=off full_page_writes=off shared_buffers=128 GB | |

# 场景 2：共 16 节点 ShardingSphere-JDBC 5.1.1 + openGauss 数据分片

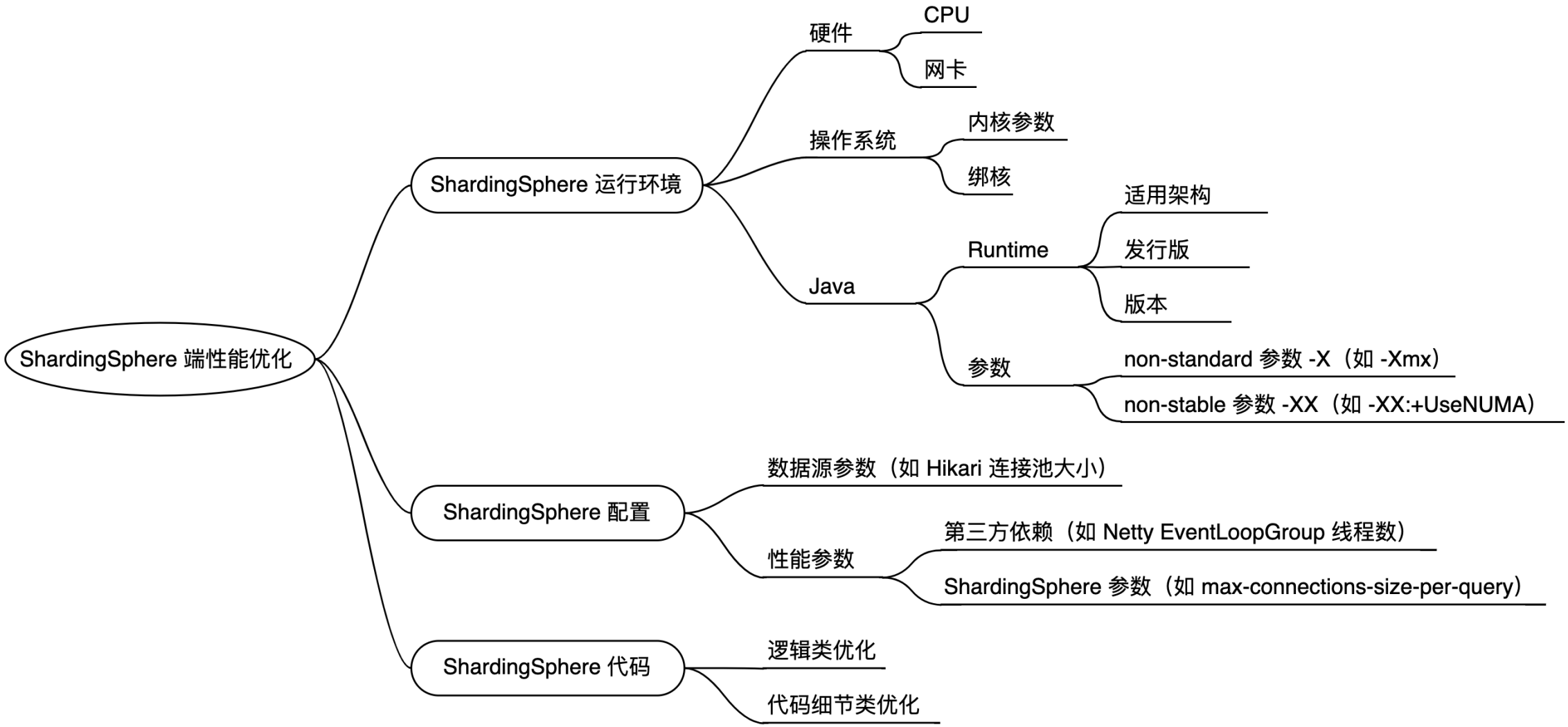



| 配置 | 8 x openGauss | 7 x ShardingSphere-JDBC | 1 x ShardingSphere-Proxy |
|---|---|---|---|
| CPU | 128 核<br>2 路鲲鹏 920 | 128 核<br>2 路鲲鹏 920 | 128 核<br>2 路鲲鹏 920 |
| 内存 | 768GB | 768GB | 768GB |
| 系统盘 | 1TB | 1TB | 1TB |
| 数据盘 | 3 x 4TB NVMe SSD | 无 | 无 |

ShardingSphere

# 目录

# ShardingSphere 性能优化思路



# 案例：云服务器单队列网卡软中断性能瓶颈

```
top - 14:40:59 up 246 days,  3:31,  4 users,  load average: 12.09, 16.67, 29.95
Tasks: 409 total,   1 running, 408 sleeping,   0 stopped,   0 zombie
%Cpu(s): 17.4 us,  5.7 sy,  0.0 ni, 75.2 id,  0.0 wa,  0.0 hi,  1.7 si,  0.0 st
GiB Mem :     94.1 total,     17.4 free,     31.3 used,     45.5 buff/cache
GiB Swap:      0.0 total,      0.0 free,      0.0 used.     45.0 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
47695 [redacted]  20   0   35.8g  16.9g  23168 S  1139 18.0  39:31.25 java -Xmx16g -Xmn16g
```

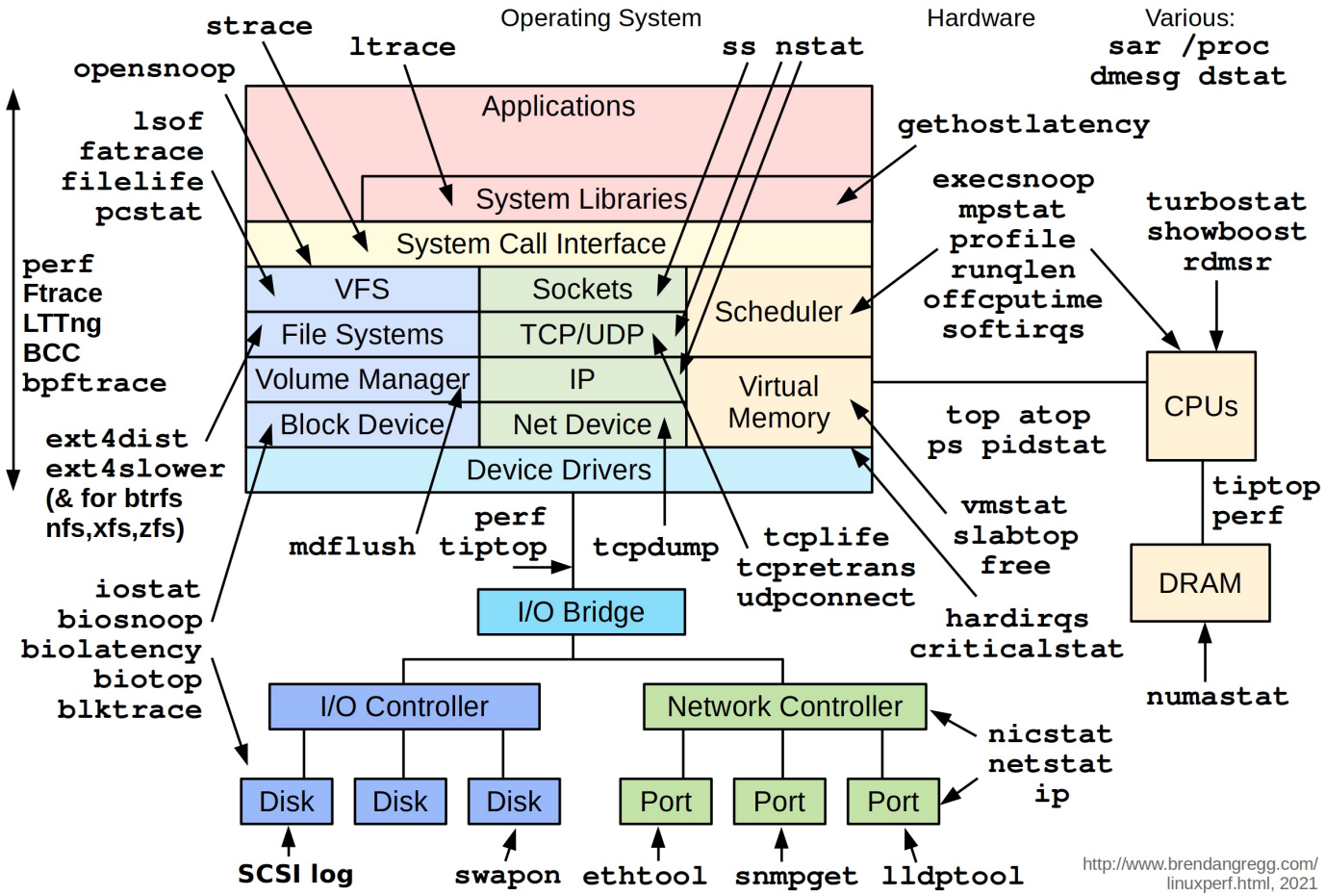
```
top - 14:39:34 up 246 days,  3:30,  4 users,  load average: 15.60, 18.27, 31.68
Tasks: 410 total,   2 running, 408 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.7 us,  0.0 sy,  0.0 ni,  0.3 id,  0.0 wa,  0.0 hi, 99.0 si,  0.0 st
%Cpu1  : 48.5 us, 15.4 sy,  0.0 ni, 36.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 46.6 us, 15.5 sy,  0.0 ni, 37.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 46.4 us, 15.6 sy,  0.0 ni, 38.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 46.5 us, 14.4 sy,  0.0 ni, 39.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 45.5 us, 15.2 sy,  0.0 ni, 39.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu6  : 44.7 us, 16.0 sy,  0.0 ni, 39.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu7  : 44.9 us, 15.2 sy,  0.0 ni, 39.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu8  : 45.2 us, 14.4 sy,  0.0 ni, 40.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu9  : 44.7 us, 14.6 sy,  0.0 ni, 40.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu10 : 44.7 us, 15.3 sy,  0.0 ni, 40.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu11 : 46.2 us, 14.0 sy,  0.0 ni, 39.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu12 : 21.1 us,  7.7 sy,  0.0 ni, 71.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu13 : 20.6 us,  7.3 sy,  0.0 ni, 72.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu14 : 19.8 us,  7.0 sy,  0.0 ni, 73.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu15 : 20.5 us,  7.0 sy,  0.0 ni, 72.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
```

解决思路：

- （硬件支持）网卡多队列绑核
- irqbalance

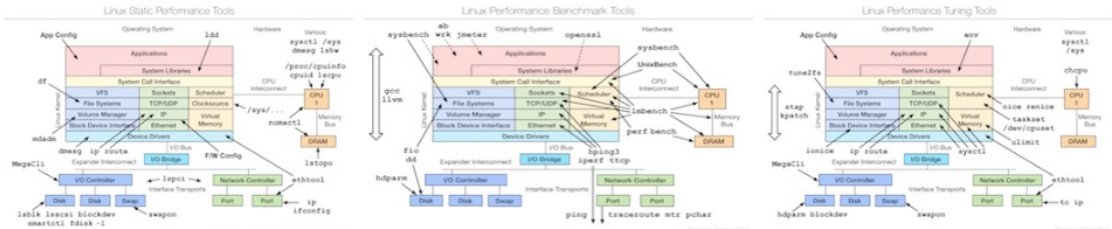
# brendangregg.com

## Linux Performance Observability Tools

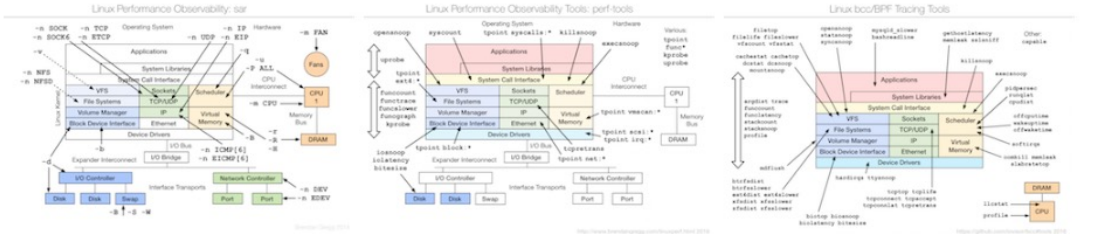



http://www.brendangregg.com/linuxperf.html, 2021

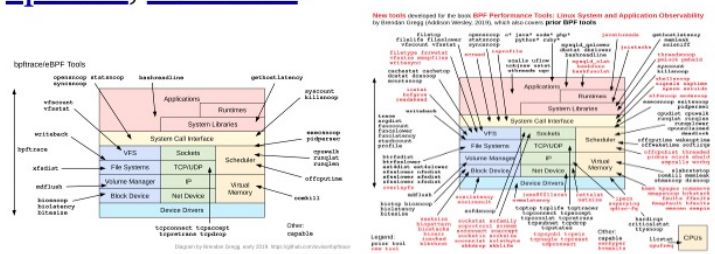
static, benchmarking, tuning:



sar, perf-tools, bcc/BPF:



bpftrace, BPF book:

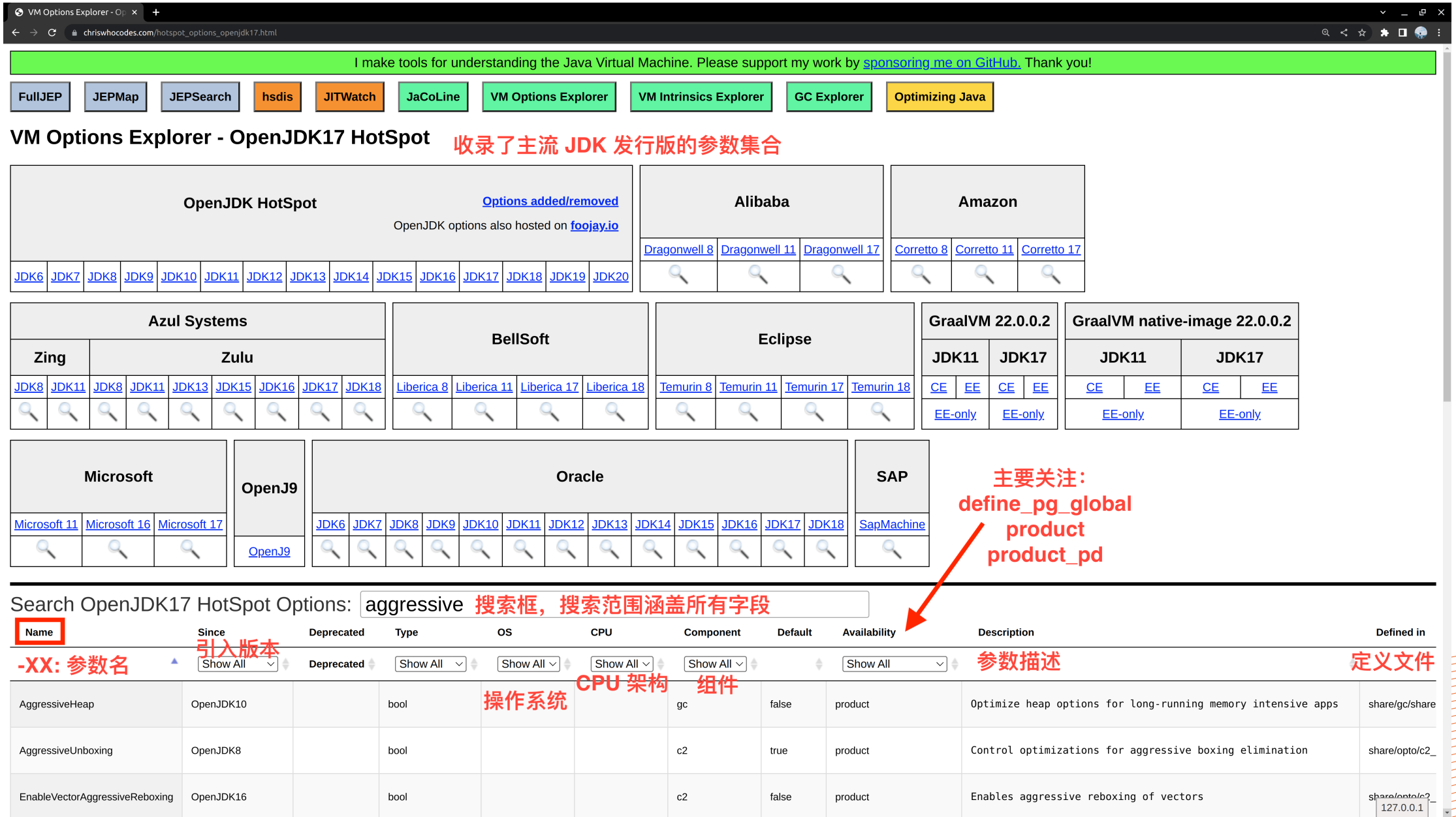


# JVM 升级与参数调优带给 ShardingSphere 的性能收益

```
44  VERSION_OPTS=""
45  if [ $int_version = '8' ] ; then
46      VERSION_OPTS="-XX:+UseConcMarkSweepGC -XX:+UseCMSInitiatingOccupancyOnly -XX:CMSInitiatingOccupancyFraction=70"   Java 8 参数
47  elif [ $int_version = '11' ] ; then
48      VERSION_OPTS="-XX:+SegmentedCodeCache -XX:+AggressiveHeap"   Java 11 参数
49      if $is_openjdk; then
50        VERSION_OPTS="$VERSION_OPTS -XX:+UnlockExperimentalVMOptions -XX:+UseJVMCICompiler"      Java 11 OpenJDK 追加参数
51      fi
52  elif [ $int_version = '17' ] ; then
53      VERSION_OPTS="-XX:+SegmentedCodeCache -XX:+AggressiveHeap"   Java 17 参数
54  else
55      echo "unadapted java version, please notice..."
56  fi
57
58  JAVA_OPTS=" -Djava.awt.headless=true "
59                                                                  公共参数
60  JAVA_MEM_OPTS=" -server -Xmx2g -Xms2g -Xmn1g -Xss1m -XX:AutoBoxCacheMax=4096 -XX:+UseNUMA -XX:+DisableExplicitGC -XX:LargePageSizeInBytes=128m ${VERSION_OPTS}
```

* 在 128 核鲲鹏 920 环境运行 ShardingSphere，使用 Java 17 相比 Java 8 性能提升 5% ~ 10%

# VM Options Explorer -- chriswhocodes.com



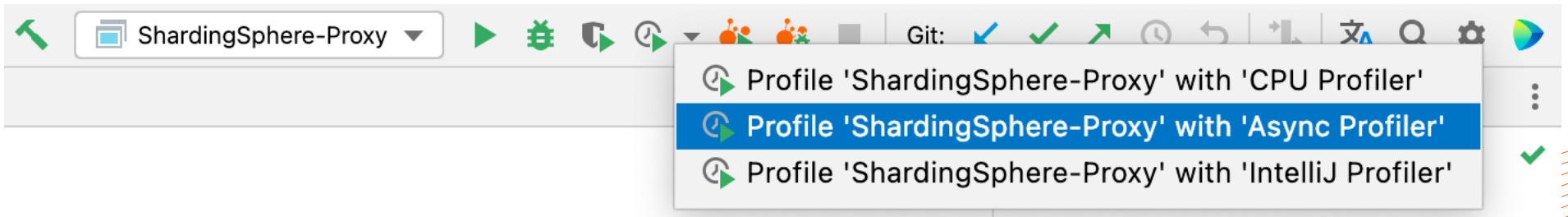
# 目录

# 借助 async-profiler 发现 ShardingSphere 代码优化点

async-profiler 是针对 JVM 的采样分析工具。

特点：
- ✓ 不受 Safepoint Bias 影响
- ✓ 性能开销低（采样频率可调整），生产可用
- ✓ 使用方便，可基于 Java Agent 启动，或指定 PID 连接已有 JVM
- ✓ 支持多种输出格式（HTML、SVG、JFR 等）及格式转换

支持事件：
- ✓ CPU 周期
- ✓ 硬件和软件性能指标，如缓存未命中、分支未命中、缺页、上下文切换等
- ✓ Java 堆中的分配
- ✓ 锁，包括 Java 对象监视器和 ReentrantLock
- ✓ ……



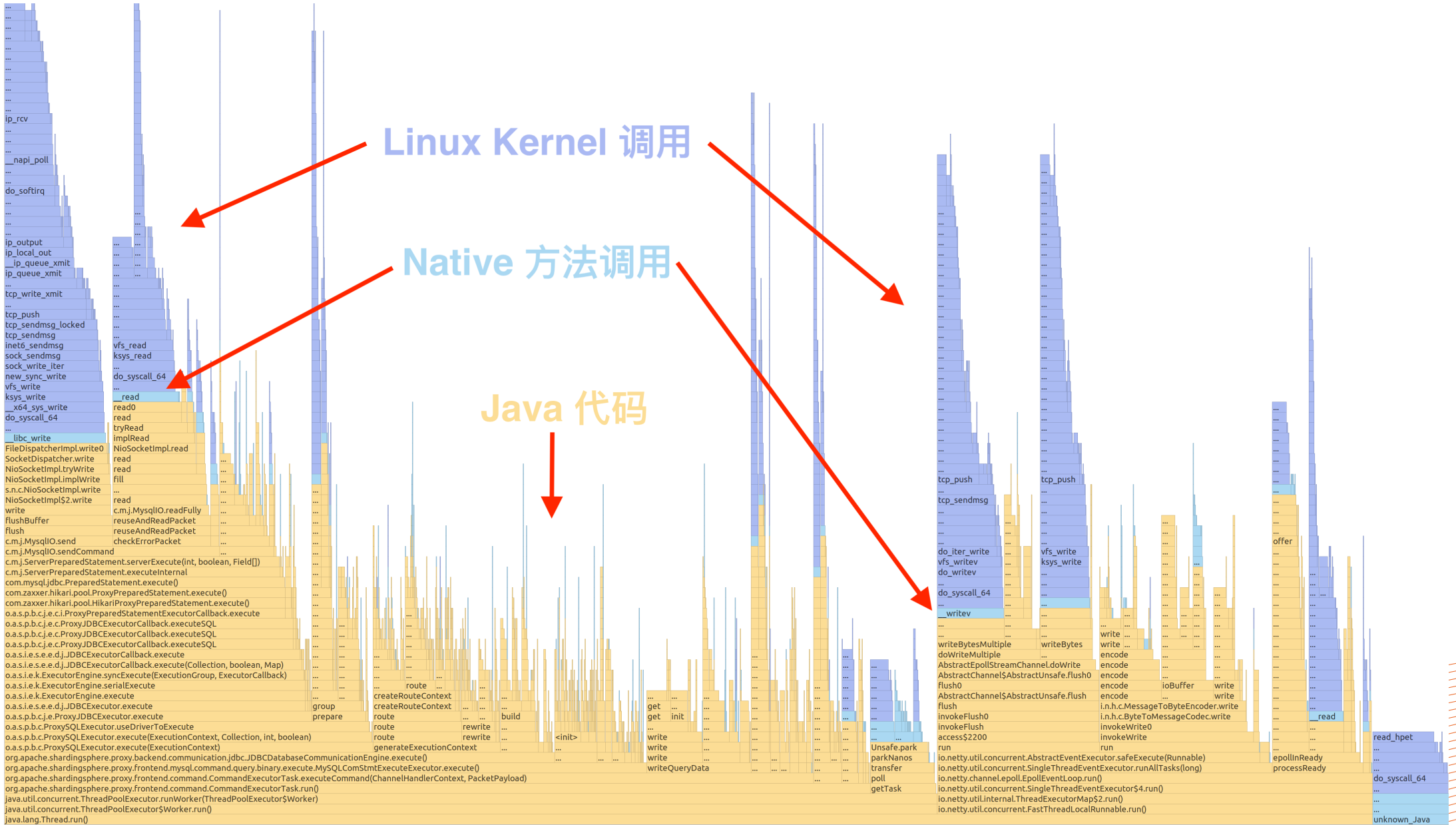

```
└─$ profiler list 163601
Basic events:
  cpu
  alloc
  lock
  wall
  itimer
Java method calls:
  ClassName.methodName
Perf events:
  page-faults
  context-switches
  cycles
  instructions
  cache-references
  cache-misses
  branch-instructions
  branch-misses
  bus-cycles
  L1-dcache-load-misses
  LLC-load-misses
  dTLB-load-misses
  rNNN
  pmu/event-descriptor/
  mem:breakpoint
  trace:tracepoint
  kprobe:func
  uprobe:path
```

# JFR vs perf vs async-profiler

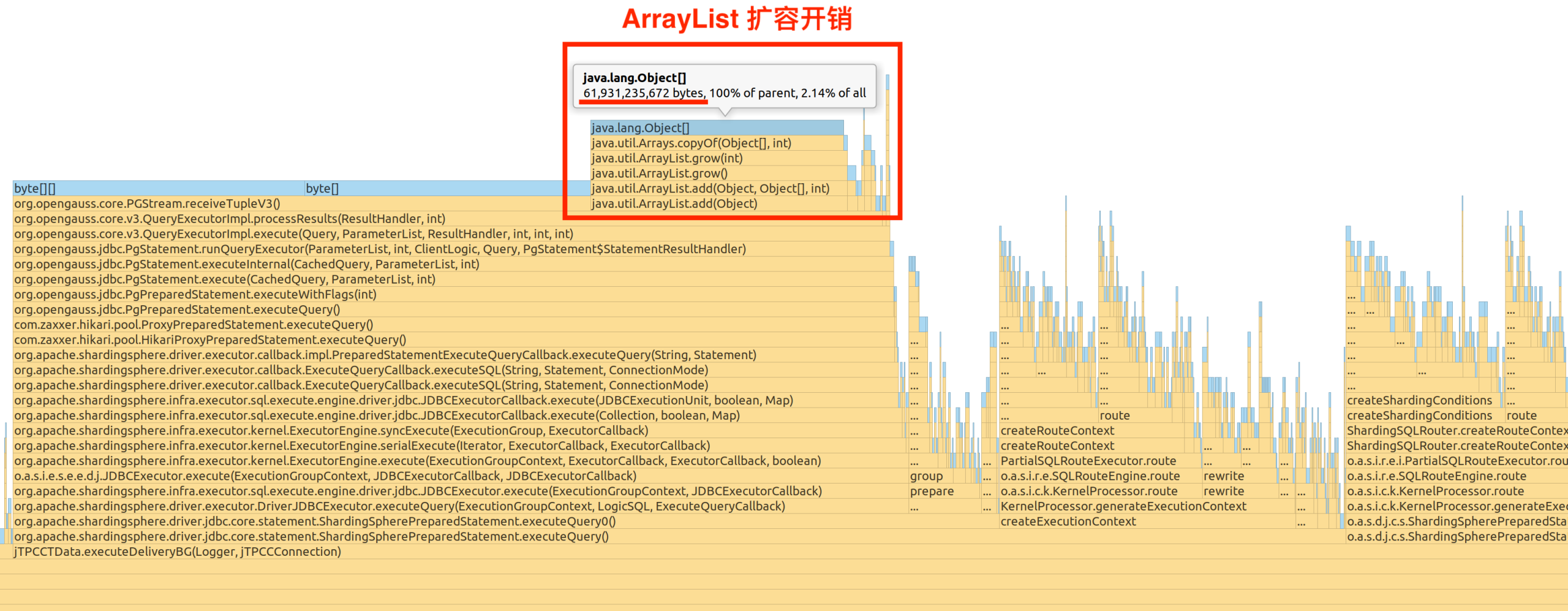
| | JFR | perf | async-profiler |
|---|---|---|---|
| Java stack | Yes | No interpreted | Yes |
| Native stack | No | Yes | Yes |
| Kernel stack | No | Yes | Yes |
| JDK support | 11+, 8u262+ | 8u60+ | 6+ |
| OS support | All | Linux only | Linux, macOS* |
| Permanent overhead | 0 | 1-5% | 0 |
| System-wide profiling | No | Possible | No |

来源：https://github.com/apangin/java-profiling-presentation/blob/master/presentation/java-profiling.pdf

# async-profiler 采集 CPU 火焰图



# async-profiler 采集内存分配火焰图



# async-profiler 采集其他事件

Flame Graph | Call Tree | Method List | Timeline | Events

- Events by type 14.8 ×10⁶
  - Flight Recorder 26
    - Async-profiler Recording 3
    - Async-profiler Setting 23
  - Java Application 11.1 ×10⁶
    - Allocation in new TLAB 11 ×10⁶
    - Allocation outside TLAB 92,809
    - Java Monitor Blocked 0
    - Java Thread Park 18,056
  - Java Virtual Machine 3.66 ×10⁶
    - Profiling 3.66 ×10⁶
      - Method Profiling Sample 3.66 ×10⁶
    - Runtime 25
      - Native Library 25
    - Initial System Property 22
    - JVM Information 1
  - Operating System 122
    - Processor 121
      - CPU Information 1
      - CPU Load 120
    - OS Information 1
  - Profiler 79
    - Log Message 79

| Start Time | JVM User | JVM System | Machine Total |
|---|---|---|---|
| 3/8/22, 11:35:22.799 AM | 68.8 % | 24.8 % | 94.7 % |
| 3/8/22, 11:35:23.799 AM | 66.8 % | 26.2 % | 93.7 % |
| 3/8/22, 11:35:24.799 AM | 64.3 % | 25.6 % | 91.5 % |
| 3/8/22, 11:35:25.799 AM | 23 % | 9.03 % | 32.1 % |
| 3/8/22, 11:35:26.799 AM | 68.3 % | 26.4 % | 96 % |
| 3/8/22, 11:35:27.799 AM | 66.6 % | 25.7 % | 93.2 % |
| 3/8/22, 11:35:28.800 AM | 68.2 % | 26.2 % | 95.6 % |
| 3/8/22, 11:35:29.800 AM | 66.4 % | 26.4 % | 93.7 % |
| 3/8/22, 11:35:30.800 AM | 68.5 % | 26.2 % | 95.8 % |
| 3/8/22, 11:35:31.800 AM | 66.7 % | 26.1 % | 93.5 % |
| 3/8/22, 11:35:32.800 AM | 68.8 % | 26.1 % | 96 % |
| 3/8/22, 11:35:33.800 AM | 66.6 % | 26.6 % | 93.9 % |
| 3/8/22, 11:35:34.800 AM | 68 % | 26.8 % | 95.9 % |
| 3/8/22, 11:35:35.800 AM | 61 % | 24.4 % | 85.3 % |
| 3/8/22, 11:35:36.800 AM | 0.0268 % | 0.0893 % | 0.938 % |
| 3/8/22, 11:35:37.801 AM | 53.4 % | 22.1 % | 75.5 % |
| 3/8/22, 11:35:38.801 AM | 68.8 % | 26.1 % | 96 % |
| 3/8/22, 11:35:39.801 AM | 67.2 % | 25.9 % | 93.9 % |
| 3/8/22, 11:35:40.801 AM | 68 % | 26.7 % | 95.7 % |

# 案例：Java 8 ConcurrentHashMap computeIfAbsent 性能问题

https://github.com/apache/shardingsphere/pull/13275

17 ...apache/shardingsphere/infra/executor/sql/prepare/driver/DriverExecutionPrepareEngine.java

View full Viewe

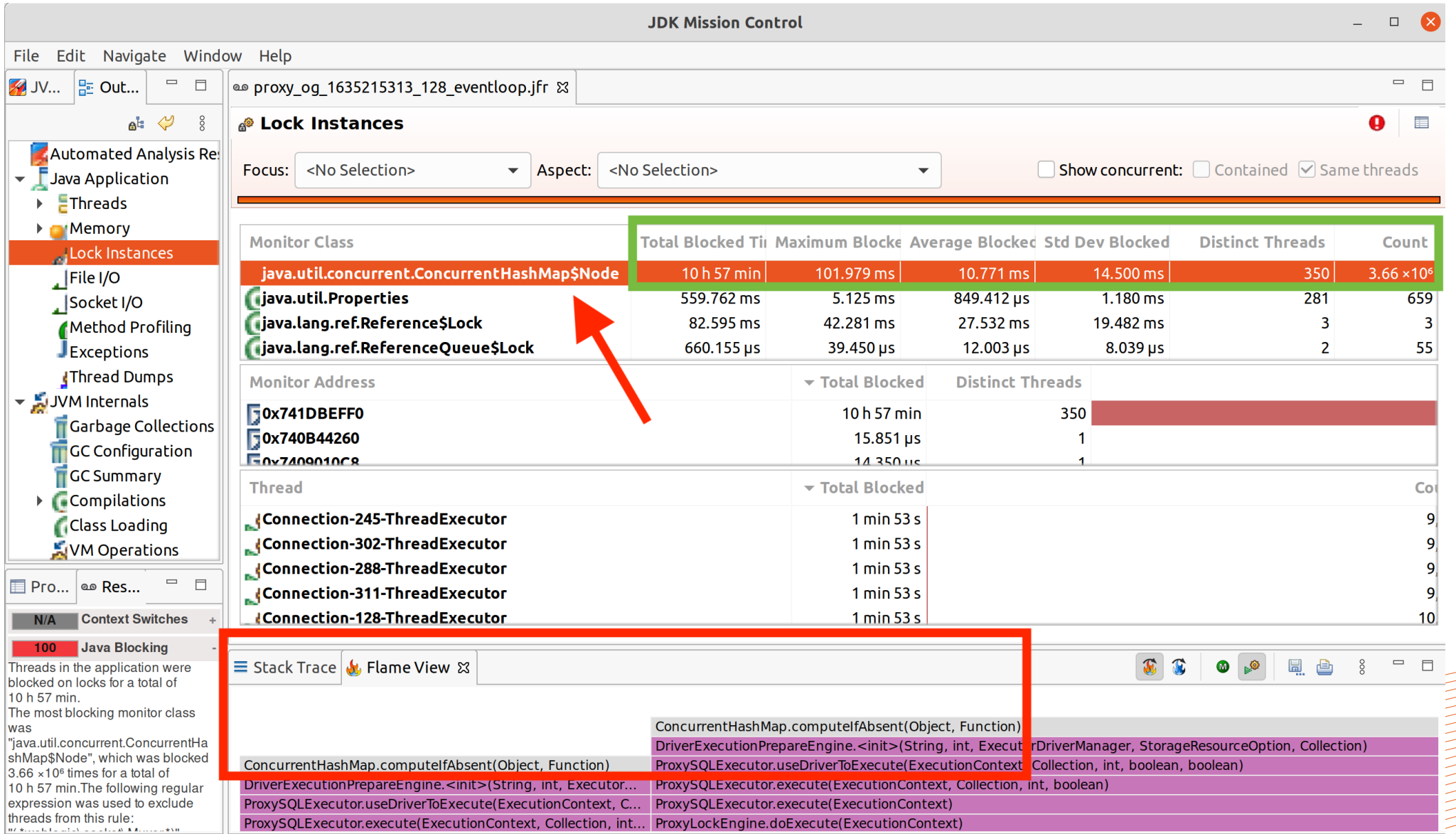
```
@@ -62,7 +62,22 @@ public DriverExecutionPrepareEngine(final String type, final int maxConnectionsS
62 62          super(maxConnectionsSizePerQuery, rules);
63 63          this.executorDriverManager = executorDriverManager;
64 64          this.option = option;
65    -         sqlExecutionUnitBuilder = TYPE_TO_BUILDER_MAP.computeIfAbsent(type, key -> TypedSPIRegistry.getRegisteredService(SQLExecutionUnitBuilder.class, key, new Properties()));
   65 +         sqlExecutionUnitBuilder = getCachedSqlExecutionUnitBuilder(type);
   66 +     }
   67 +
   68 +     /**
   69 +      * Refer to https://bugs.openjdk.java.net/browse/JDK-8161372.
   70 +      *
   71 +      * @param type type
   72 +      * @return sql execution unit builder
   73 +      */
   74 +     @SuppressWarnings("rawtypes")
   75 +     private SQLExecutionUnitBuilder getCachedSqlExecutionUnitBuilder(final String type) {
   76 +         SQLExecutionUnitBuilder result;
   77 +         if (null == (result = TYPE_TO_BUILDER_MAP.get(type))) {
   78 +             result = TYPE_TO_BUILDER_MAP.computeIfAbsent(type, key -> TypedSPIRegistry.getRegisteredService(SQLExecutionUnitBuilder.class, key, new Properties()));
   79 +         }
   80 +         return result;
66 81      }
```

命令示例：使用 async-profiler 采集 JVM 进程中，阻塞超过 100us 的 lock，采样 60 秒并输出到文件 output.jfr

```
./profiler.sh -d 60 -e cpu --lock 1ms -f output.jfr $JVM_PID
```

# 案例：Java 8 ConcurrentHashMap computeIfAbsent 性能问题

## 多线程使用相同 key 调用 computeIfAbsent



ShardingSphere

# 目录

# 代码基准测试反面教材

```java
public class WrongBenchmark {
    2 usages
    private static final int TIMES = 10000;

    public static void main(String[] args) {
        benchThreadLocalRandomUUID();
        benchDefaultRandomUUID();
    }
    1 usage
    private static void benchDefaultRandomUUID() {
        Random random = new Random();
        long from = System.nanoTime();
        for (int i = 0; i < TIMES; i++) new UUID(random.nextLong(), random.nextLong());
        long duration = System.nanoTime() - from;
        System.out.printf("Random UUID() took %4f%n", duration / 1000.0 / 1000);
    }
    1 usage
    private static void benchThreadLocalRandomUUID() {
        Random random = ThreadLocalRandom.current();
        long from = System.nanoTime();
        for (int i = 0; i < TIMES; i++) new UUID(random.nextLong(), random.nextLong());
        long duration = System.nanoTime() - from;
        System.out.printf("ThreadLocalRandom UUID() took %4f%n", duration / 1000.0 / 1000);
    }
}
```

输出结果：

```
/usr/local/java/jdk-17.0.1/bin/java ...
ThreadLocalRandom UUID() took 1.738908
Random UUID() took 1.173566
```

调整 bench 方法顺序后输出结果：

```
/usr/local/java/jdk-17.0.1/bin/java ...
Random UUID() took 1.813381
ThreadLocalRandom UUID() took 1.151652
```

# Java Microbenchmark Harness (JMH)

https://github.com/openjdk/jmh


OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC

Bylaws · Census
Legal

**JEP Process**

**Source code**
Mercurial
GitHub

## Code Tools: jmh

See the JMH Source Repository for additional details.

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

### Links

- Source Repository
- Mailing List (archive)
- Bugs

# 案例：使用 JMH 测试 Java 8 ConcurrentHashMap computeIfAbsent

```java
@State(Scope.Benchmark)
public class ConcurrentHashMapBenchmark {

    private static final String KEY = "key";

    private static final Object VALUE = new Object();

    private final Map<String, Object> map =
                        new ConcurrentHashMap<>();

    @Setup(Level.Iteration)
    public void setup() {
        map.clear();
    }
```

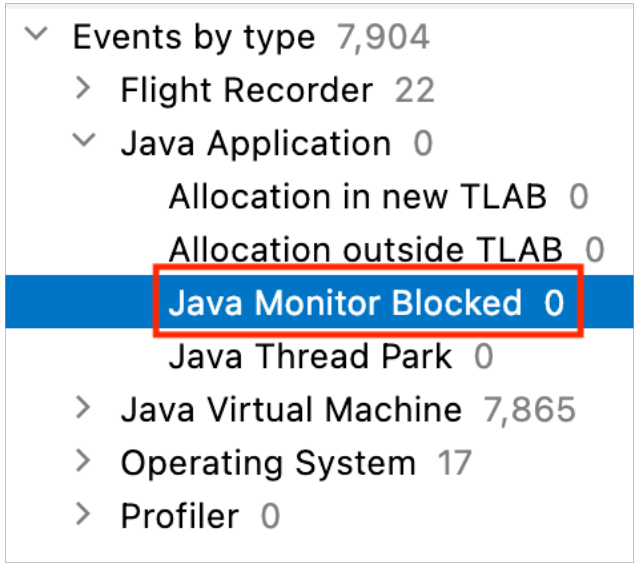
```java
    @Benchmark
    public Object benchGetBeforeComputeIfAbsent() {
        Object result = map.get(KEY);
        return null == result ? map.computeIfAbsent(KEY, __ -> VALUE) : result;
    }

    @Benchmark
    public Object benchComputeIfAbsent() {
        return map.computeIfAbsent(KEY, __ -> VALUE);
    }

    public static void main(String[] args) {
        new Runner(new OptionsBuilder()
                .addProfiler(AsyncProfiler.class,
                        "event=cpu;lock=100us;dir=/tmp/jmh-jfr;output=jfr")
                .threads(32)
                .forks(3)
                .warmupIterations(3).warmupTime(TimeValue.seconds(5))
                .measurementIterations(3).measurementTime(TimeValue.seconds(5))
                .build()).run();
    }
}
```

# 案例：使用 JMH 测试 Java 8 ConcurrentHashMap computeIfAbsent

JMH 测试结果：

```
                                  Mode  Cnt          Score          Error  Units
benchComputeIfAbsent              thrpt    9    7737446.819 ±  2522383.120  ops/s
benchGetBeforeComputeIfAbsent     thrpt    9  783148861.986 ± 56766616.378  ops/s
```

GetBeforeComputeIfAbsent




ComputeIfAbsent

# JMH 使用不当也会测不准代码性能

```java
private double x = Math.PI;

private double compute(double d) {
    for (int c = 0; c < 10; c++) d = d * d / Math.PI;
    return d;
}

@Benchmark
public void baseline() {
    // do nothing, this is a baseline
}

@Benchmark
public void measureWrong() {
    // This is wrong: result is not used and the entire computation is optimized away.
    compute(x);
}

@Benchmark
public double measureRight() {
    // This is correct: the result is being used.
    return compute(x);
}
```

代码来源于 JMHSample_08_DeadCode.java

更多示例请参考 JMH 源码 samples

测试结果：

```
Benchmark                                  Mode  Cnt  Score    Error  Units
JMHSample_08_DeadCode.baseline             avgt    5  0.437 ±  0.001  ns/op
JMHSample_08_DeadCode.measureRight         avgt    5  9.902 ±  0.019  ns/op
JMHSample_08_DeadCode.measureWrong         avgt    5  0.438 ±  0.003  ns/op
```

# 欢迎关注 ShardingSphere

技术干货

加入交流群

**Apache ShardingSphere Website**：https://shardingsphere.apache.org

**Apache ShardingSphere GitHub**： https://github.com/apache/shardingsphere

**Apache ShardingSphere Slack Channel**：https://apacheshardingsphere.slack.com

ShardingSphere

# 谢谢观看