GIAC | 全球互联网架构大会
GLOBAL INTERNET ARCHITECTURE CONFERENCE

Java性能优化指南，及唯品会的实战

肖桦（江南白衣）
唯品会资深架构师

2017.thegiac.com

ABOUT ME

70后，喜欢编码的架构师

唯品会平台架构部

服务化框架，容器云平台

SpringSide 春天的旁边

CONCENTS

WRONG THINGS

RIGHT THINGS

JIT & MICRO BENCHMARK

TOOLS

PRACTICES

WRONG THINGS

从过时的经验开始说

互联网时代的Java优化经验

For many years, one of the top 3 hits on Google for “Java Performance Tuning” was an article from 1997.8, the page housed advice that was

completely out of date

-- 两位大大<Optimizing Java> 开篇的吐槽

Java演进凡二十年，不同时代的信息全堆到网上

-- 缺乏 失效机制

谁都可以分享自己的经验，然后被搬运网站转载，再被搜索引擎展示

-- 缺乏stackoverflow式 纠错机制

过时的经典语录 - 设为NULL

将用完的变量设为NULL，能加快回收？

-- No， JDK 比你想象的更好

有效的优化：

- 对象定义贴近使用它的逻辑分支
  (scanned by Sonar)

过时的经典语录 – Getter/Setter

（初级版）直接访问对象属性，能提升性能？

减少函数调用的消耗？ -- No， JIT 方法内联

（高级版） -XX:+UseFastAccessors ？

-- No , Only for JDK6

还有拿已是默认值的来骗感情，如指针压缩

[生产环境同版java] [应用完整参数] **-XX:+PrintFlagsFinal** –version | grep [待查参数]

过时的经典语录 - final

（初级版）函数参数设为 final，能提升性能？

-- No，已经没人能说出为什么了

（中级版） 不变对象，能加快GC跨代扫描速度？

如果老生代对象的属性老变，就会指向新生代中的对象？

-- YES，但只在乎属性实际变没变，不在于final定义

（高级版）类设为final，能有助内联？

JIT想为调用者内联方法时，不知道该内联哪个子类的方法，设成final能提示没有子类了？

-- No，JDK自己有类层次分析(CHA)

不盲信，多看代码 - slf4j

```
logger.info( “Hello {}” ,name);
```

有魔术吗？ -- MessageFormatter类

```
for (int i = 0; i < args.length; i++) {
      j = messagePattern.indexOf( “{}” , i);
      ……
}
```

没有预编译，输出每条日志, 都要循环查找“{}”，
然后subString取出“Hello”，最后拼接参数name

如果日志确定输出，自己直接拼接字符串更快

不盲信，多做试验 – 魔幻参数

```
-XX:ParGCCardsPerStrideChunk=32K
```

第一次出现在LinkedIn工程师的博客，
大家不明觉厉，先抄为敬

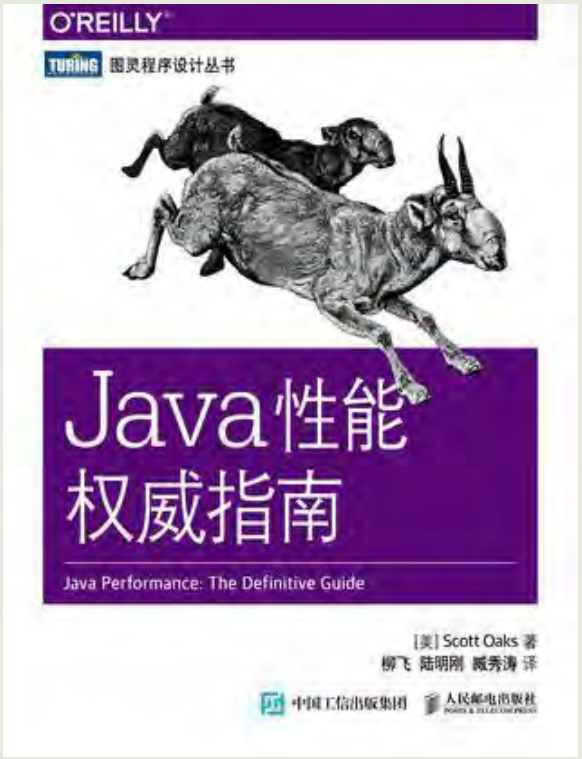
Twitter试验说， 8K 才是合适的
俄国哥们（这个参数的发明者）的测试，4K 最优

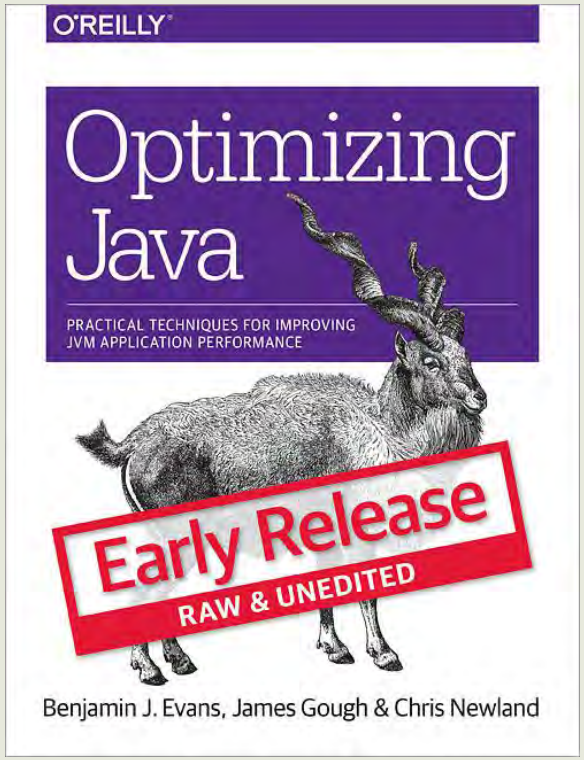
难道他们说的都是真的？，视乎应用的新老生代特征

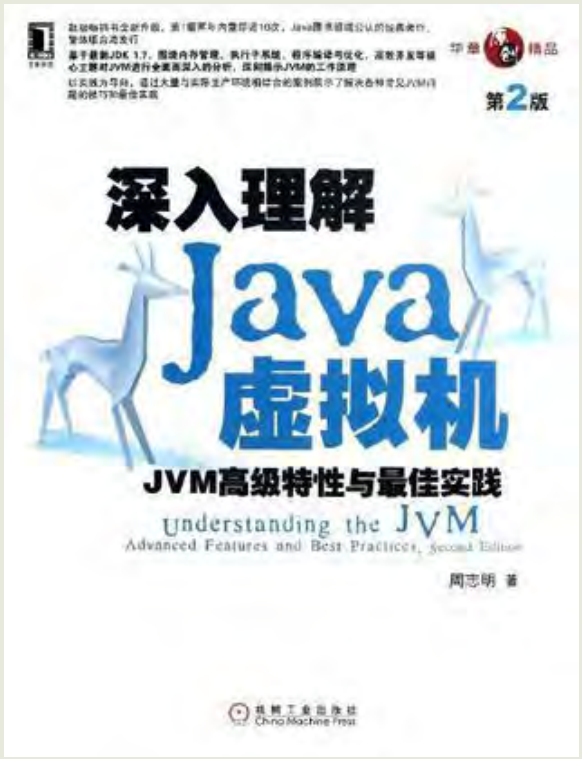
靠谱的

人和书
看代码，看代码

可靠的书







其他如某Java调优圣经，10年前了

可靠的人

R大

RednaxelaFX，莫枢

原Azul JVM ，现Databricks

知乎上的1368个回答

JavaEye上的N个帖子

知乎：R大是谁

笨神

原阿里JVM

公众号：你假笨

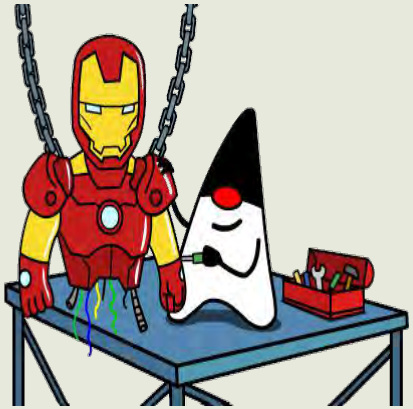
小程序：JVMPocket

(JVM参数锦囊)

最可靠的，是代码

遇到问题，会否看 com.sun.*， 与C写的native方法，

是一个 分水岭



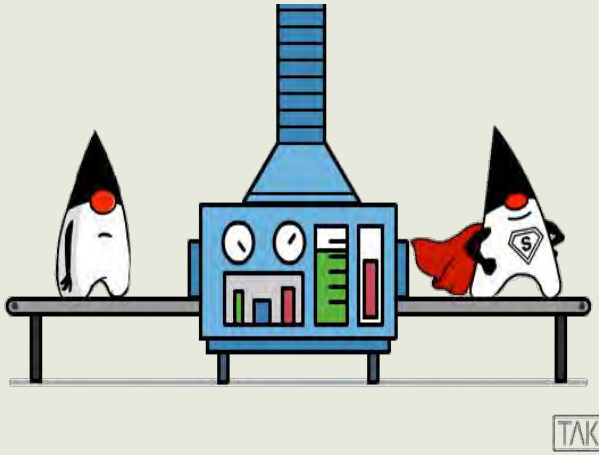
根据tags下载生产环境同款的小版本

http://hg.openjdk.java.net/jdk7u/jdk7u/hotspot/
http://hg.openjdk.java.net/jdk7u/jdk7u/jdk/

微基准测试你的优化

先懂一点JIT
然后知道微基准测试的必要

JIT浅说



JIT = 动态编译 + 优化

JIT后，Java并不比C慢

JNI 调 C方法，不用于加速

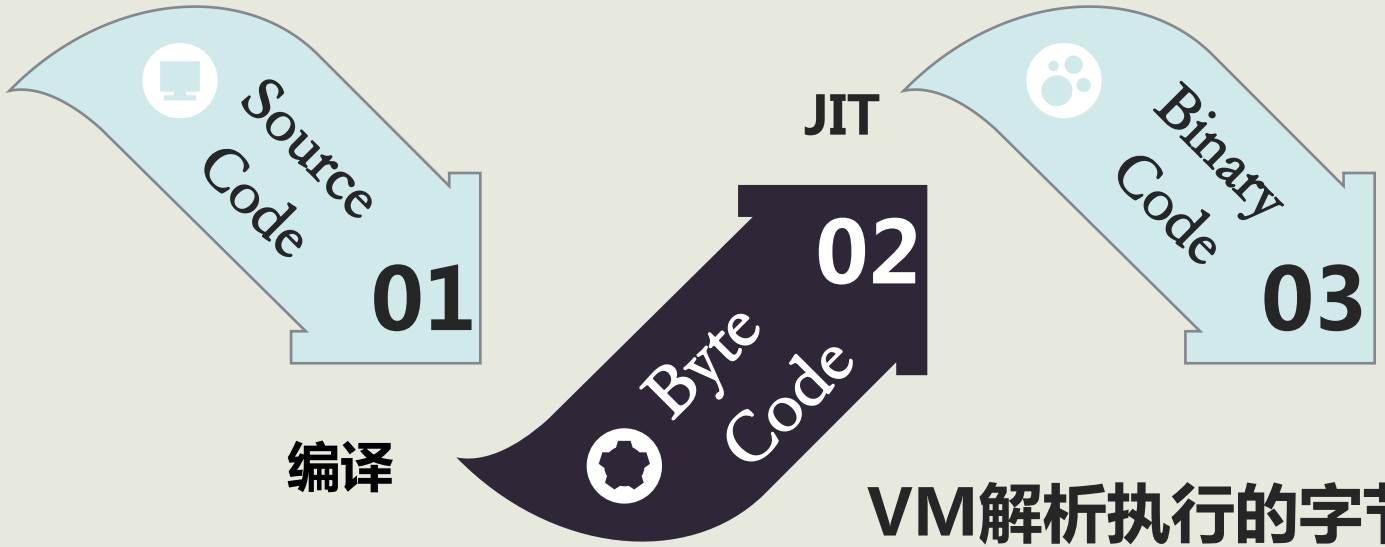
JIT – Code

文本源码

*.java
*.py
*.php

机器直接执行的机器码

Java　　原生支持，内存中
Python　某些非官方版本
PHP　　鸟哥在捣鼓



VM解析执行的字节码

*.class
*.pyc
opCache (php5.5+)

JIT – Compiler

C1 编译器

立即编译
静态轻量优化

未深度优化

C2 编译器

采集1万次
方法调用样本后
动态深度优化

1万次前
及温热方法，
执行慢

JDK8 多层编译

启动时，C1编译
样本足够，C2编译

刚启动时，
狂吃CPU
更多停顿

JDK9 AOT

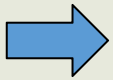
启动前，静态
编译*.so文件
样本足够，JIT

试验阶段

JIT – 方法内联

```
public int a(int count)
{
   count = count*2;
   return b(count);
}

private int b(int count
{
    return count+1;
}
```



```
public int a(int count) {
   count = count*2;
   return count+1;
}
```

每个方法只有几行的 **优雅代码** 的基础

Getter/Setter 无需优化的原因

内联条件

第一次访问：-XX:MaxInlineSize=35 Byte
频繁的访问：-XX:FreqInlineSize=325 Byte

最多18层内联，及其他条件....

可视化工具： **JITWatch**

JIT – 逃逸分析

分析对象有否逃逸出当前线程，当前方法

1. 同步消除

如果实际只有一条线程访问对象
消除对象方法的Synchronized

例外：StringBuffer，因内联父类失败

2. 标量替换

局部对象只在方法内被使用

R大说： 不是传说中的栈上分配对象
而是只创建对象的属性

堆内存	**线程栈内存**
Java对象	局部原子变量 / 引用 方法结束即弹栈，没有GC

```
A a = new A();          int b = 1;

a.b = 1;
```

JIT – 代码优化

JVM工程师十多年的积累与骄傲所在， 一些人肉优化变得多余

无用代码消除

循环优化

空值检测消除

数组边界检查消除

公共子表达式消除

JIT优化项一览(2009)

……

```
for (int i=0; i <10000; i++)
{
    int x =str.indexOf(c);
}
```

变量x上下文没有作用
整个循环被消除

测试一切 - 微基准测试的要点

没有测试数据证明的论断，都是 **可疑** 的。

一个简单的main()测试，也是 **不可靠** 的。

01

预热

触发JIT的调用次数

后台编译所需的时间

02

防止无用代码消除

03

避免干扰

GC

生成测试数据的消耗

简化基准测试编写 - JMH

像JUnit那样，编写测试代码，标注annotation，JMH完成剩下一切

运行预热循环

与被测函数交互防止代码消除

数据准备接口

迭代前主动GC

并发线程控制

结果统计

```
@BenchmarkMode(Mode.SampleTime)
@Warmup(iterations = 5)
@Measurement(iterations = 10, time = 5, timeUnit = TimeUnit.SECONDS)
@Threads(24)
public class SecureRandomTest {
    private SecureRandom random;

    @Setup(Level.Trial)
    public void setup() {
        random = new SecureRandom();
    }

    @Benchmark
    public long randomWithNative() {
        return random.nextLong();
    }

    @Benchmark
    @Fork(jvmArgsAppend = "-Djava.security.egd=file:/dev/./urandom")
    public long randomWithSHA1() {
        return random.nextLong();
    }
}
```

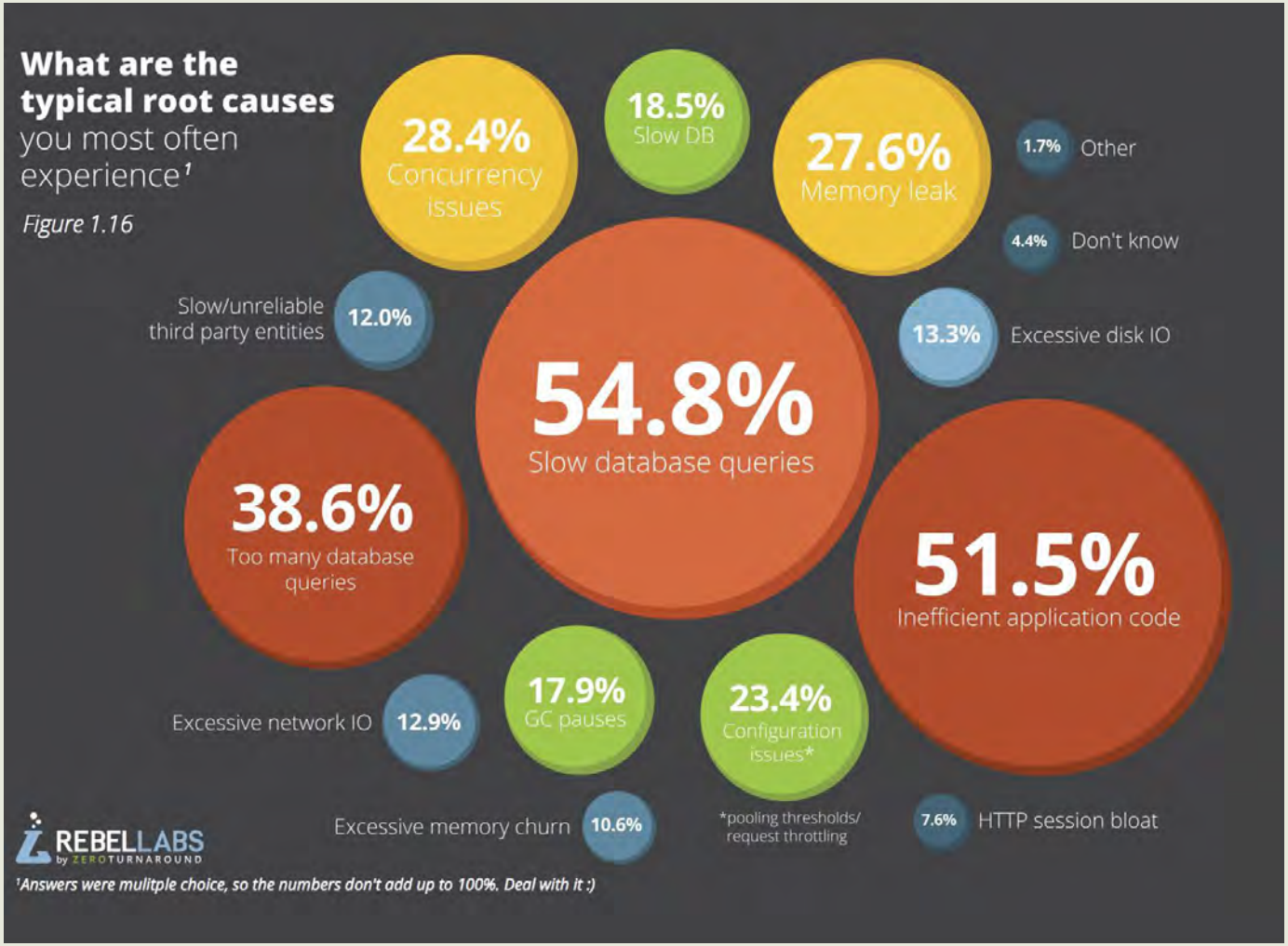
上例对比两种SecureRandom生成算法的性能

工具

先定位问题，再解决问题

应用缓慢的众多原因

SQL慢占了大头，所以先定位问题，别急着优化Java代码



Profiler的两大分类

Instrumentation - based

AOP式在方法前后植入代码

JProfiler的Instrument模式

能准确统计每个方法的
调用次数，耗时

性能成量级的衰退，
结果无意义

植入代码导致方法膨胀，
无法内联

Sampling - based

Thread Dump式采样

统计Stack Trace上的方法

只能统计方法的热度，
相对百分比

性能损耗 < 10%

可在生产环境运行

Instrumentation模式只适合监控少量方法

Sampling的两大分类

ThreadDump

JProfiler的Sampling模式
JVMTop .....

JvmtiEnv::GetAllStackTraces()

ThreadMXBean.dumpAllThreads()

有停顿

不可用于延迟敏感应用

AsyncGetCallTrace

Java Mission Control
Honest Profiler .....

Hostpot隐藏API

无停顿

采样间隔可低至10ms

JMC：JDK7 u40 以上自带，注意License
对热身后的JVM采样，定制化settings文件，-XX:+DebugNonSafepoints

在线排查 - BTrace

BTrace是神器，每一个需要每天解决线上问题，
但完全不用BTrace的Java工程师，都是可疑的。
- by 凯尔文。肖

通过自己编写的脚本，attach进应用获得一切信息

不再需要修改代码，加上System.out.println()

然后重启，然后重启，然后 重启！！！

只要定义脚本时不作大死，可在 生产环境 打开

BTrace – 典型的场景

1. 服务偶发慢，找出慢在哪一步？

2. 下列情况发生时，上下文是什么？

什么情况下进入了这个处理分支？
谁调用了System.gc()？
谁构造了一个超大的ArrayList？

```
@OnMethod(clazz = "+com.vip.demo.OspFilter", method = "doFilter", location = @Location(Kind.RETURN))
public static void onDoFilter(@ProbeClassName String pcn, @Duration long duration) {
    if (duration / 1000000 > 1)
        println(pcn + ".doFilter:" + (duration / 1000000));
}
```

打印实现了OspFilter接口的Filter链中，执行时间超过了1毫秒的Filter

执行命令：./btrace <pid> HelloWorld.java

TBJMap and DIY

排查对象缓慢泄漏时，临时对象的干扰太大

我可以只看老生代的对象统计吗？

-- **TBJMap**

我可以只看Survivor区 age>3的对象统计吗？

-- Attach JVM 后，DIY，少年

```
hotspotAgent.attach(pid);     //暂停应用
CollectedHeap heap = VM.getVM().getUniverse().heap();
hotspotAgent.detach();        //恢复应用
```

实践

应用停顿排查
性能调优案例

STOP THE WORLD

Java程序员最头痛的事情

学名：安全点

GC

- YGC
  - 停顿时间与GC后剩下对象的多少，成 正比 关系
  - 让持久对象尽快晋升：降低晋升阈值， Cache对象预热阶段主动GC
- CMS GC
  - 对两个STW阶段，讲究个见招拆招，如Remark前先YGC等
  - JDK9因找不到人维护，可能放弃
- G1
  - JDK9默认，8G 以上Heap建议
  - 重新踩坑，重新学习
- ZGC
  - 最终效果类似Azul Zing的C4
  - 大如2TB 的Heap，停顿只在 10毫秒 以下

停顿时间

- GC的停顿， 不止是垃圾收集

  -- 等待所有线程停止的时间，等待GC日志写入的时间

```
[DefNew:... [Times: user=0.29， sys=0.00，real=0.015 secs]
Total time for which application threads were stopped: 1.021 seconds
```

- JVM的停顿，不止是GC

  -- JIT，Class Redefinition(AOP)，取消偏向锁，Thread Dump...

停顿时间的查看

GC日志的应用停顿时间：

必需品，获得大部分JVM记得打印的停顿

-XX:+PrintGCApplicationStoppedTime

jHiccup：

获得实际的停顿时间

包含底层操作系统对JVM的影响

安全点日志：

获得非GC的停顿原因等细节

http://gceasy.io：

在线分析GC日志

一些意外的GC停顿

无数种的可能，八卦网上每一篇实战分析

使用了Swap

vm.swappiness = 0

写GC日志被锁

日志写到 /dev/shm目录
（内存文件系统）

抢不到CPU

限制非业务进程的CPU
减少GC线程数

写/tmp/hsperfdata被锁

关闭pefdata，用 JMX 代替 jstat

调优案例

各种思路的典型，随便聊聊

面向GC编程 - int *vs* Integer

更少内存，更少GC

4 bytes *vs* 16 bytes

Java对象最小 16 bytes

12 bytes的固定header

Why Integer?

能表达Null语义

集合 / 泛型需要

避免无谓转换

int<->Integer的自动转换是有代价的，尽量一致

Integer缓存数组，默认缓存**-128～127**

-XX:AutoBoxCacheMax=20000后，应用整体提升 4% QPS

面向GC编程 – Netty的优化

AtomicIntegerFieldUpdater

写法当有海量对象，海量数字属性时：

n * int + 静态的updater *VS* *n * AtomicInteger*

IntObjectHashMap

1. int[] keys, V[] values的数据结构

2 开发地址法 vs 链表法

不能更省的数据结构，微基准测试性能提升 50%

并发与锁 – 无锁

锁，是应用慢的Top3原因

CAS 是lock-free，不是wait-free

竞争写时有等待, 有消耗

Atomic* 系列
ConcurrentLinkedQueue

不变对象

不修改对象属性
直接替换为新对象

CopyOnWriteArrayList

ThreadLocal

ThreadLocalRandom

很多非线程安全类的处理

TLAB，Heap内存分配的避免冲突

TAKIPI 脑洞大开

普通线程池，有锁的任务队列

Netty EventLoop：
N个容量为1的线程池
任务随机分配给一个池

应用局部异常时的性能 - Exception StackTrace

异常的StackTrace很有用，但构造的消耗很大

- 出处明确时，取消StackStrace

绕过调fillStakTrace()的构造函数

1. Netty的静态异常
2. Cloneable异常

- 别取消我的StackStrace

大量重复的JDK Exception会被优化

但日志滚动后，就看不到NPE的出处

-XX:-OmitStackTraceInFastThrow

平衡 - 性能与问题排查便捷性

Fast Enough，不求极致 - 反射

反射好用，很多框架类库在用， 但会慢么？

Java 是一门以Fast Enough为设计理念的语言

- Method对象的获取，慢

  缓存起来反复使用， Dozer *vs* Apache BeanUtils

- Method的调用，快

  开始，基于NativeAccessor

  15次后，生成GeneratedMethodAccessorXXX类，bytecode直调实际方法

好工具库

不要过早优化，不要过细的优化，但....

一直按好的习惯来编程， 很多时候没有额外的成本
把实践都封装起来， 让业务开发 默认 就获得 最优 的性能

公司级别的工具库

直接使用： Apache Commons Lang 与 Google Guava

参考吸收：

- 大厂工具库：Facebook JCommon，twitter commons，linkedin-utils等

- 开源项目中的工具库：ElasticSearch，Cassandra，Netty等

广告时间

公众号：春天的旁边

唯品会即将开源的VJTools项目

GIAC | 全球互联网架构大会
GLOBAL INTERNET ARCHITECTURE CONFERENCE

GIAC

全球互联网架构大会

GLOBAL INTERNET ARCHITECTURE CONFERENCE

扫码关注GIAC公众号

2017.thegiac.com