TOP1 100

主办方 msup

快手Java运行时领域实践

殷芳玺 快手 技术专家

讲师简介

殷芳玺
资深技术专家

殷芳玺，2019年加入快手，担任程序语言运行时/性能优化团队负责人，负责公司内部JVM、基础库等相关产品的研发，致力于提升Java/C++服务的性能和稳定性。曾在阿里巴巴、DynaTrace等公司任职。
在十多年的工作经历中，一直专注于JVM、性能优化、APM、分布式系统领域的研发，主导了一系列性能优化产品的开发，涉及JVM优化、微架构优化、基础库优化、性能监控分析平台产品等诸多方向，在相关领域具有丰富的研发经验。热爱知识分享，旧金山CodeOne演讲嘉宾，ICSE等多篇学术论文一作。

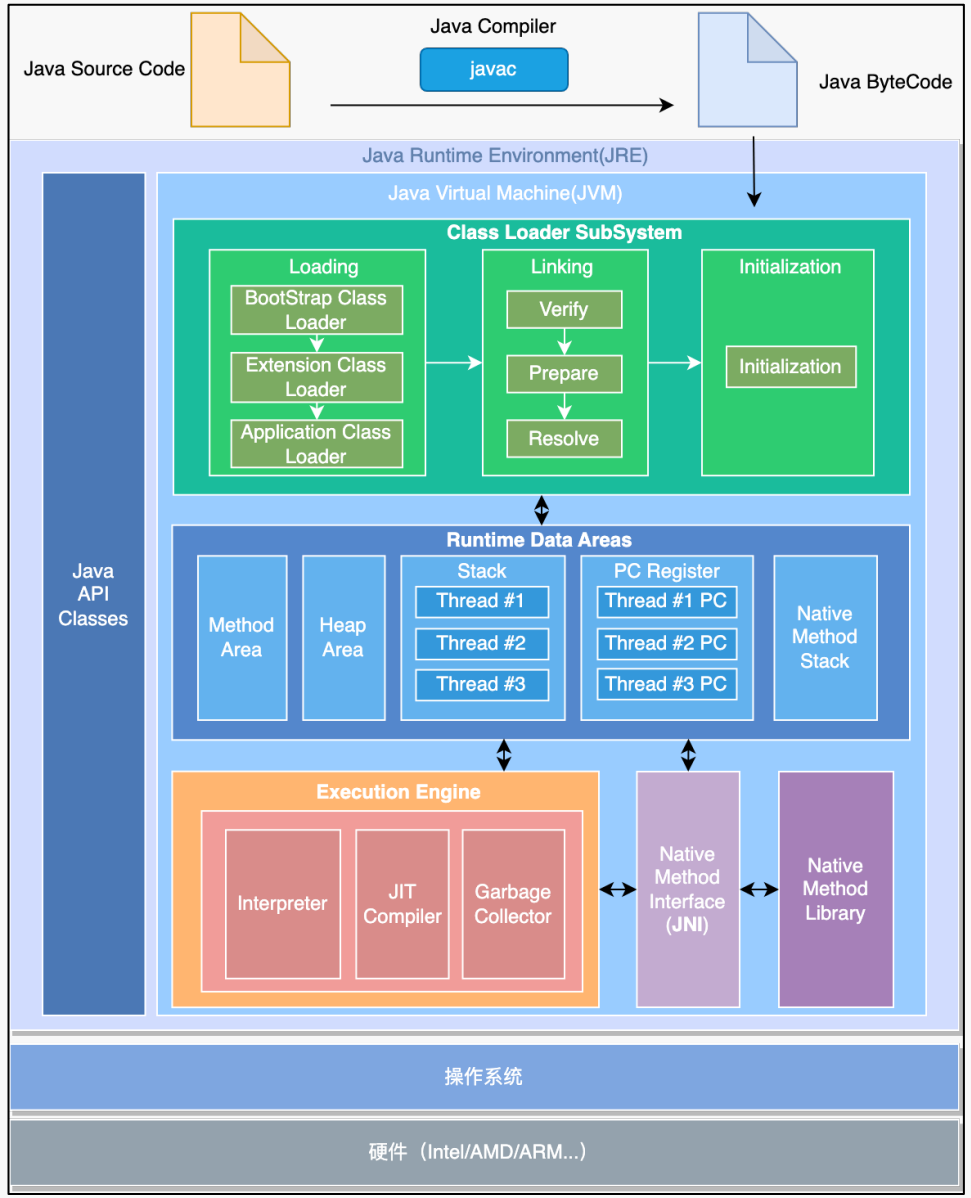
目录

亮点介绍

- 了解系统软件产品对于快手的价值
- 了解快手内部系统软件团队的工作方法
- 了解快手大规模线上生产环境使用Java17的落地案例和经验
- 提供了快手在具体困难问题上的解决方法

案例背景

- 快手服务开发的主要编程语言: Java/C++
- Java服务的运行环境: JVM(Hotspot VM)
- Hotspot VM历经20多年的发展，已经发展成为世界上最先进的高级语言虚拟机之一。



JVM团队 – WHY?

问题与挑战

- Oracle JDK收费策略调整的潜在商业风险。
- Java服务稳定性挑战: 三方JDK无源码支持，疑难杂症无法排查。
- 快手内部特殊场景，缺乏针对性优化。
- 无法享受技术进步的红利: 前沿的技术成果和社区迭代的进步。

业界的成功经验

- 当业务严重依赖某个编程语言的时候，拥有对应的编程语言运行时团队是一件理所当然的事情。
- Meta/HHVM: PHP太慢？我们可以魔改它。

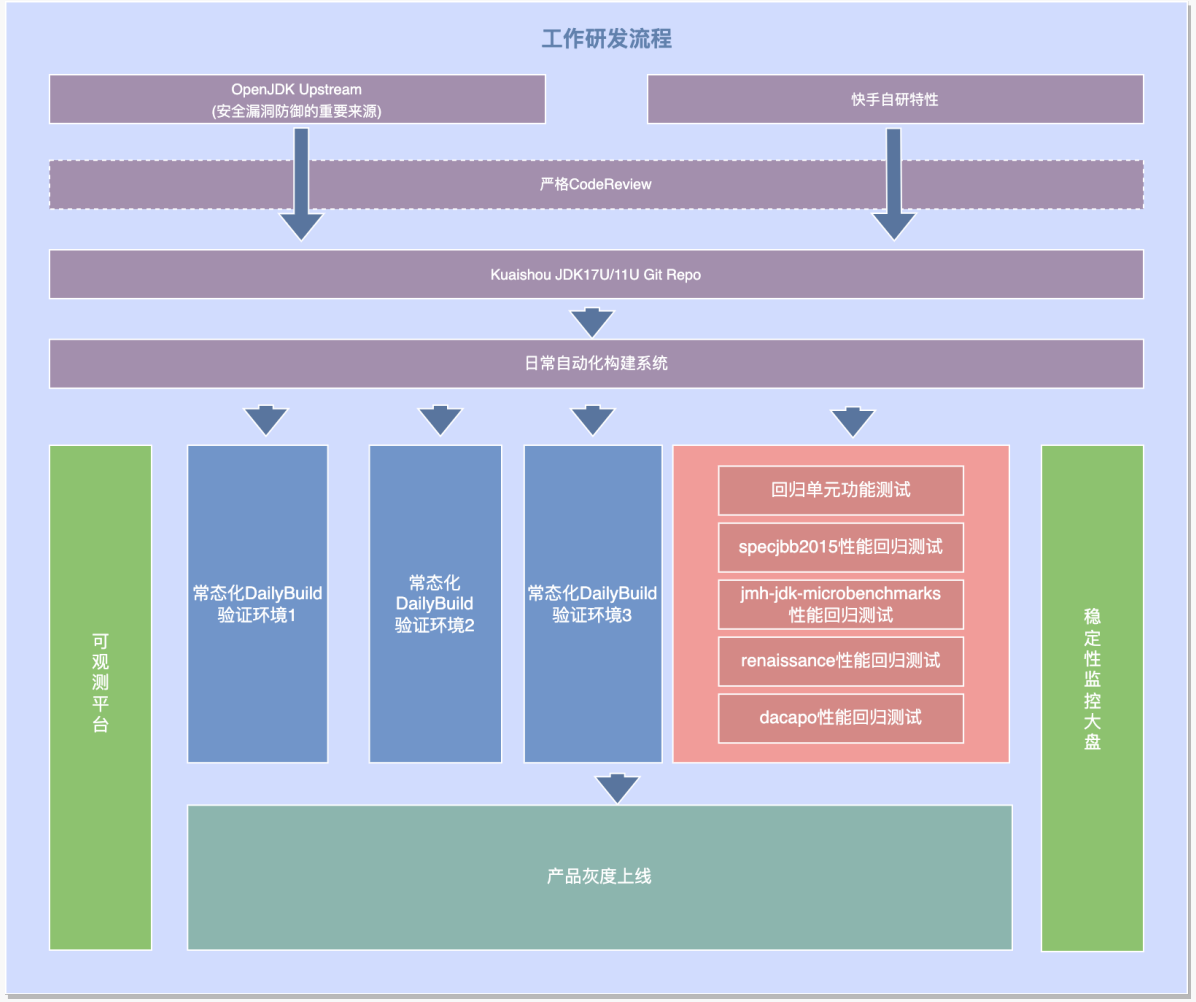
$$\text{绩效} = \frac{\text{有价值的成效}}{\text{行为代价}}$$

编程语言运行时领域的银弹效应

VM团队 – WHAT?

工作内容和职责

- 开发维护快手内部Java语言运行时产品
  - 快手魔改的OpenJDK(Java11、Java17、Java21)
  - Java周边工具套件及平台
- 应用学术界/工业界的最新成果, 改造JVM虚拟机，提升Java服务的稳定性、性能和效率。
- 作为技术方案提供方，服务内部客户。



工作方法

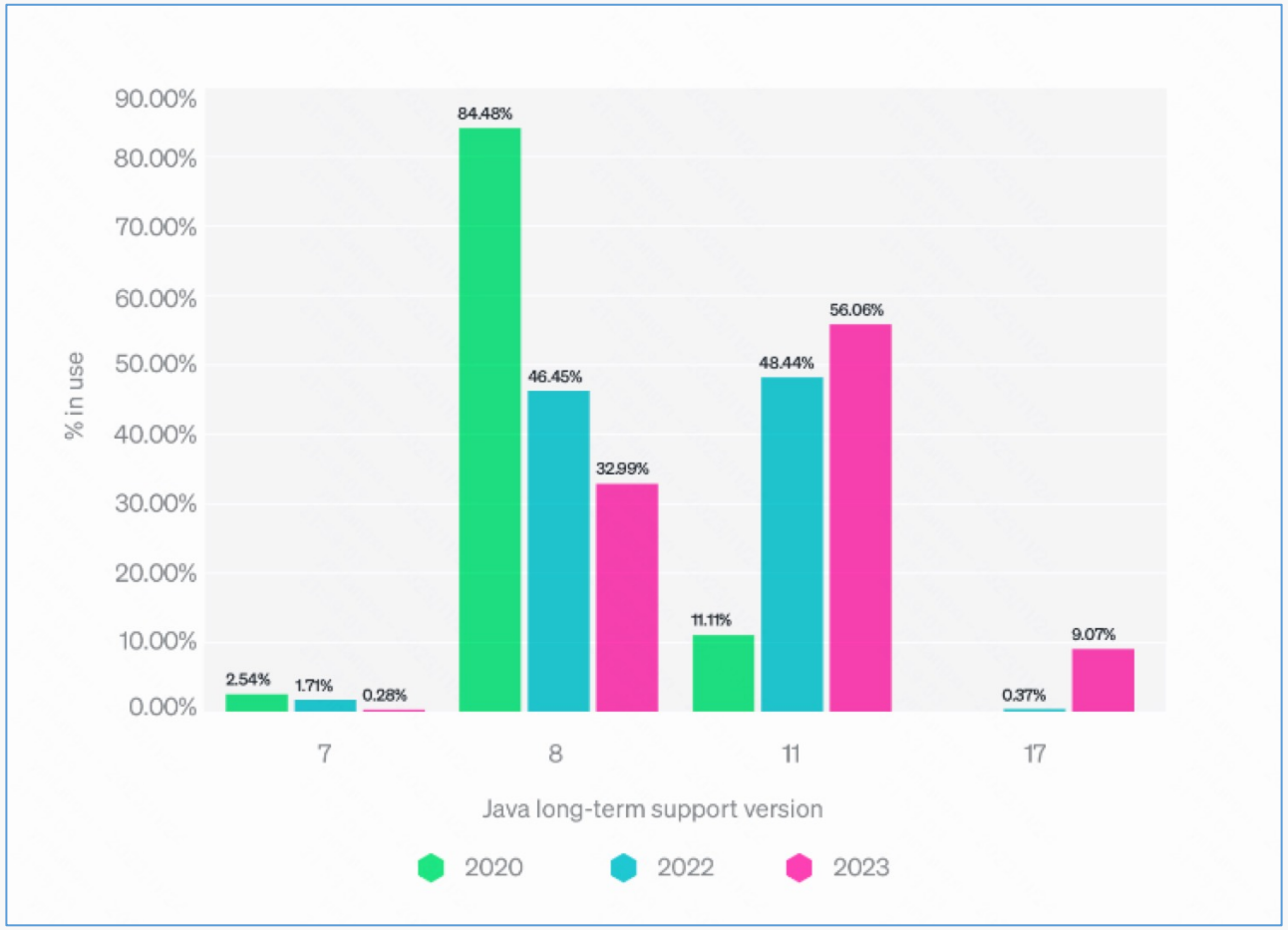
- 产品覆盖率高，影响面大。
  - 客户稳定性问题多，排查难度大。
  - 高位岗位，日常迭代的失误，很容易被放大。
- 技术门槛高，研发难度大。
  - 创新困难，摸着石头过河。
  - 复杂性高，不确定性风险大。
  - 高位岗位，日常迭代的失误，很容易被放大。

小步快跑，脚踏实地得解决重要且实际的问题

核心实践案例

业界第一个大规模Java17生产环境落地实践

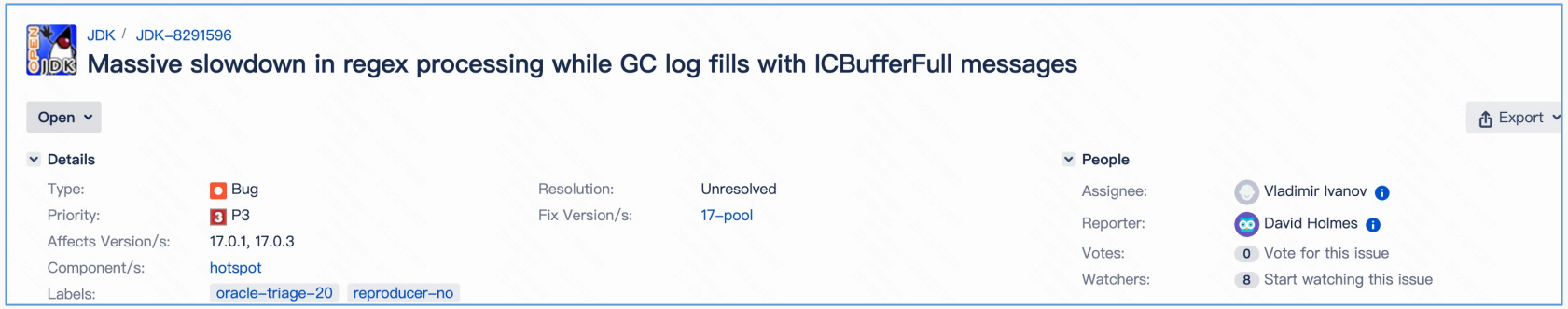
核心实践: Java17叫好不叫座的困惑



业界Java17叫好不叫座的困惑

核心实践: 困难与挑战

- 行业参考缺乏，业务方对Java17稳定性保障信心不足，获取信任难。
- 运维冲击大，透明切换难。
  - JEP403/JEP396隐藏内部API。
  - 底层三方依赖与Java17冲突。
  - 编译/打包/参数调整配合。
  - JDK行为变化(夏令时时间戳转换、Locale语言标准等)
- 快手内部大量自研特性需要平稳迁移到Java17。
- 升级带来的Oncall爆炸: 升级期间的所有稳定性问题都会首先归因到Java17。



相对Java11，Java17稳定性确实相对不足

核心实践: 透明无感的Java17升级

- 参数差异屏蔽: 做快手的Java，而不是全世界的Java。
- 行为差异屏蔽: 尊重历史，向前兼容，不打扰用户。
- 三方库依赖治理: 快手特有的RootPom机制 ， 细致梳理。
- 开发运维流程兼容: 批量后台统一刷新。

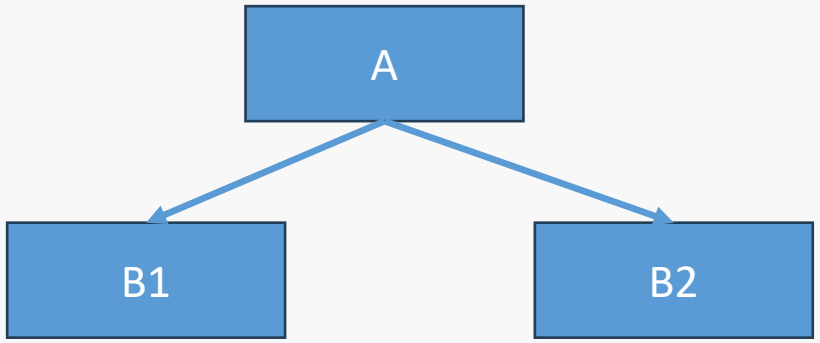
核心实践: ICBufferFull之谜

```
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 7389821 ns, Reaching safepoint: 167546 ns, At safepoint: 6840 ns, Total: 174386 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 27749 ns, Reaching safepoint: 89368 ns, At safepoint: 5710 ns, Total: 95078 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 678872 ns, Reaching safepoint: 145967 ns, At safepoint: 6969 ns, Total: 152936 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 934596 ns, Reaching safepoint: 165826 ns, At safepoint: 5460 ns, Total: 171286 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 16500 ns, Reaching safepoint: 91147 ns, At safepoint: 5770 ns, Total: 96917 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 1124041 ns, Reaching safepoint: 154426 ns, At safepoint: 6280 ns, Total: 160706 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 1222819 ns, Reaching safepoint: 152646 ns, At safepoint: 6920 ns, Total: 159566 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 1070303 ns, Reaching safepoint: 152686 ns, At safepoint: 6029 ns, Total: 158715 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 23650 ns, Reaching safepoint: 83208 ns, At safepoint: 6170 ns, Total: 89378 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 1005014 ns, Reaching safepoint: 148206 ns, At safepoint: 5660 ns, Total: 153866 ns
[info ][safepoint ] Safepoint "ICBufferFull", Time since last: 15110 ns, Reaching safepoint: 84047 ns, At safepoint: 5690 ns, Total: 89737 ns
```

Java17升级后部分服务偶发性的可用性抖动，伴随大量的IBufferFull日志

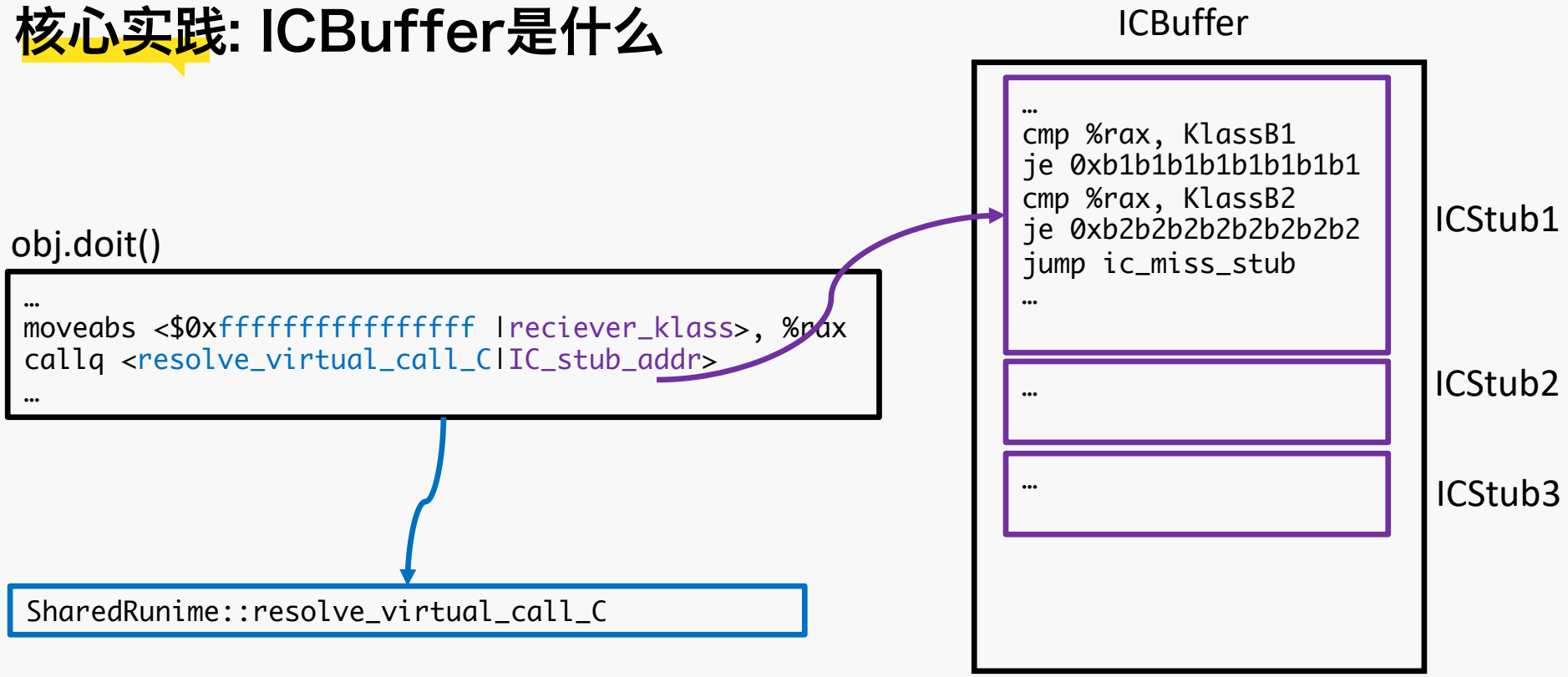
核心实践: ICBuffer是什么

```
abstract class A {
    virtual void doitA();
}

class B1 extends A {
   void doit() { … }
}

class B2 extends A {
   void doit() { … }
}

A obj = makeB();
obj.doitA(); // callsite
```



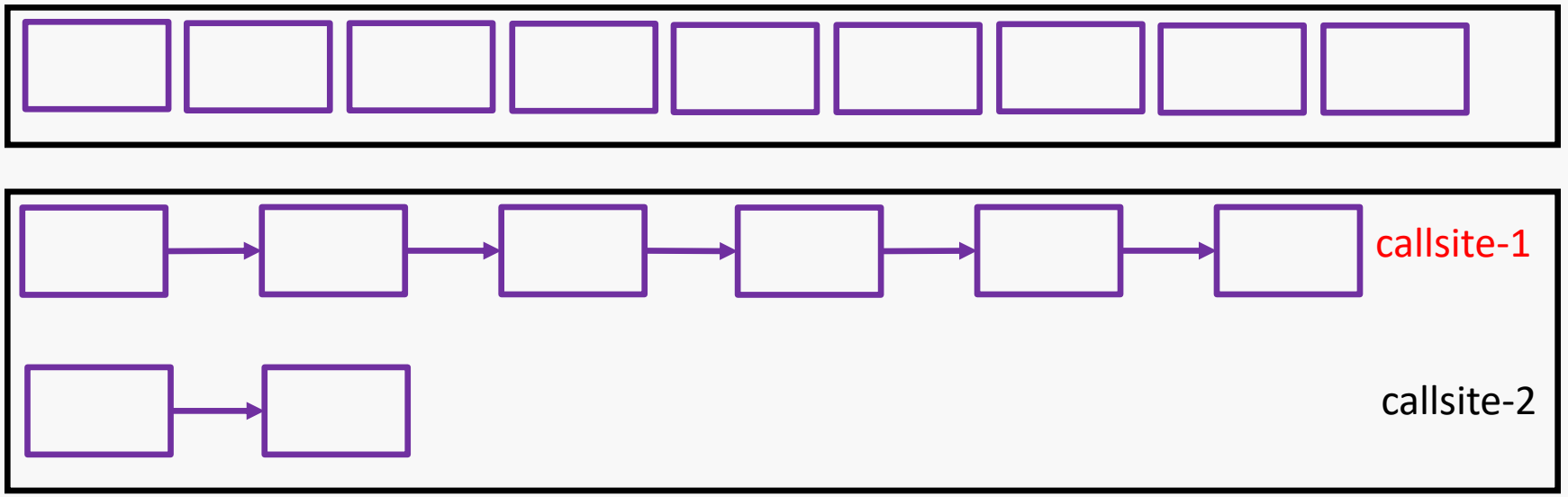
- callsite = caller_method + callee_method + pc
- 昂贵的callsite解析
  - trap到SharedRuntime::resolve_virtual_call_C(VM视角C函数调用开销大)
  - 从obj解码klass
  - 从klass解析vtable
  - 从vtable解析到vtableEntry
  - …
  - 解析到target pc

核心实践: ICBuffer是什么



- ICBuffer是一个virutal call lookup加速缓存。
- ICBuffer的尺寸是HardCode固定的。
- ICStub将ICBuffer填满，需要触发昂贵的safepoint来清理。
- Java某些边缘场景(BUG?)导致投机失败会获得严重的性能惩罚。

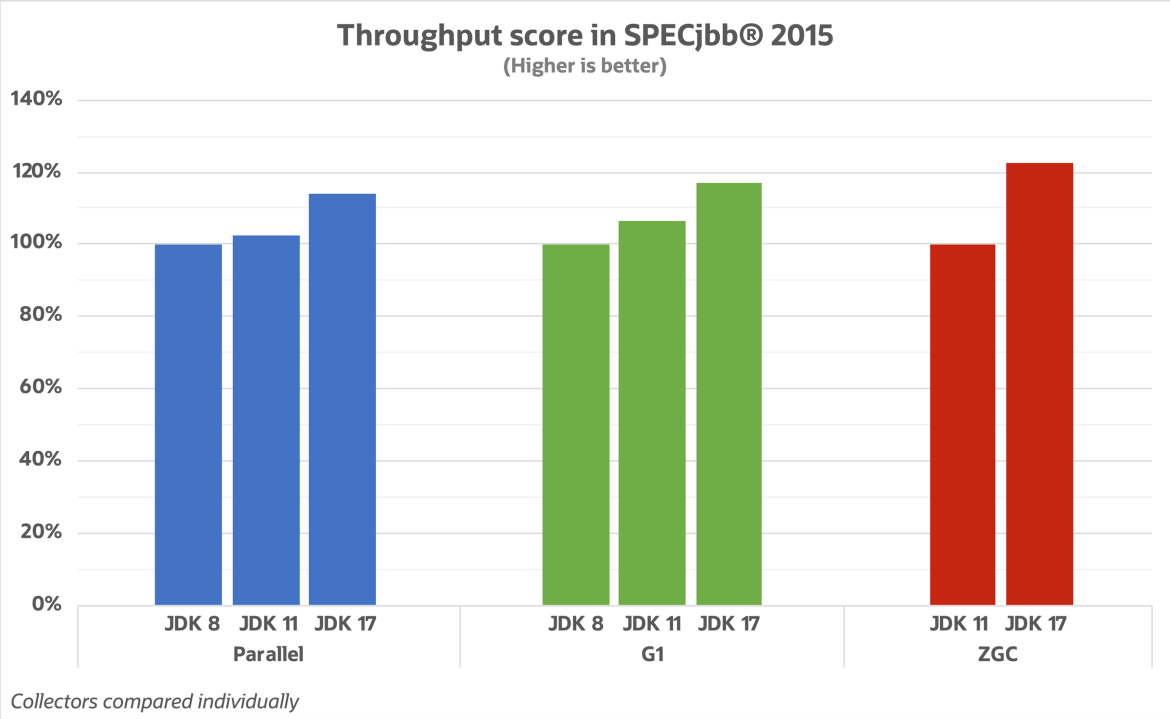
核心实践: ICBuffer的重构优化



- ICBuffer重构为一个多级列表。
- ICBuffer的尺寸可以参数调整。
- 根据链表长度快速检测出有问题的callsite，禁止该callsite的投机优化并进行部分清理。

成果展示

- 快手的Java服务基本上都运行在Java17上，充分享受了技术迭代的性能红利。
- Java17升级过程平稳顺利，无定级故障。
- 用户透明无感：除了名字的差异，快手Java17和Java11在运行时上的差异几乎为0。



数据来源:
https://kstefanj.github.io/2021/11/24/gc-progress-8-17.html

经验和启示

经验与启示

- 系统软件的价值需要重新被考量：对于体量较大的IT公司，系统软件投入性价比很高。
- 重视技术迭代的红利：软件版本升级往往最容易的优化。
- 尊重客户：与其改变客户的想法，不如改变产品。
- 破除技术迷信。

下一步

- 程序语言运行时和基础库团队依然在前进...

yinfangxi@kuaishou.com，**快手真诚期待各位的加入!**

TOP100®

主办方 msup®

微信官方公众号：壹佰案例
关注查看更多年度实践案例