毕昇JDK在大数据场景中的优化实战

语言虚拟机实验室 、彭成寒

HUAWEI

InfoQ 写作平台

InfoQ 写作平台是 InfoQ 开放给**开发者的高端技术社区**，创作者可以在这里**自由创作和发布内容**。

写作平台将为创作者提供**签约、培训、资金扶持**等一系列权益，助力作者成长为高精尖技术人才；同时也为企业提供**品牌、活动打造、内容传播**等服务，与伙伴一同成长。

扫码申请创作者

企业/个人均可申请

扫码进入写作平台

打开技术大门

我是谁

- 来自华为语言虚拟机实验室，从事过应用开发、大数据开发等工作，目前专门从事JVM的研究和开发。
- 目前是毕昇JDK开源的维护者
- 著有《新一代垃圾回收器ZGC设计与实现》、《JVM G1源码分析和调优》

目录

目录

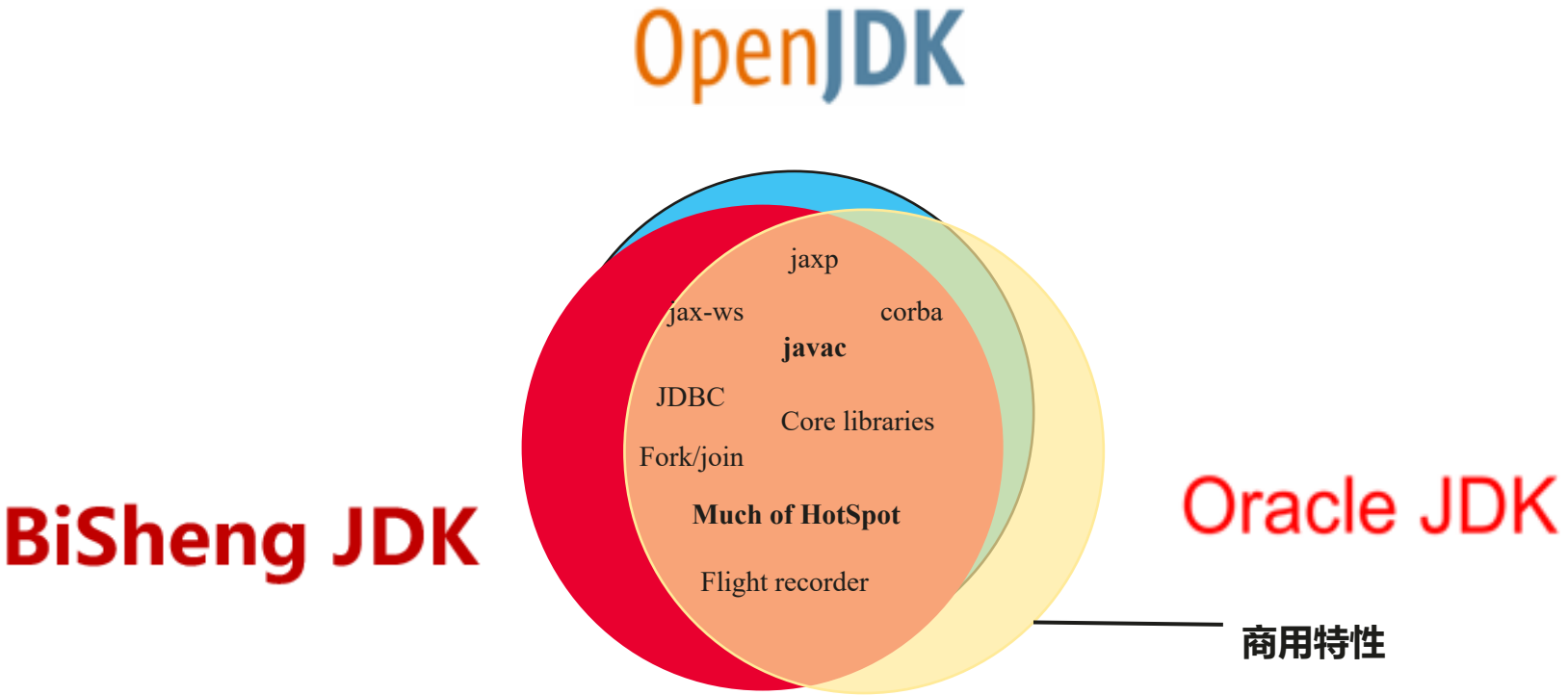
毕昇JDK

- 毕昇JDK是华为基于OpenJDK定制后的开源版本，其主要特点有：
  - >高稳定性：公司内部和部分外部客户广泛使用；
  - >高安全性：及时进行安全漏洞修复；
  - >高性能：GC&JIT优化、高价值特性支持、软硬协同加速等；
  - >专业运维：数十人团队，其中十数人是上游社区的Reviewer、Committer和Author

- 目前暂时仅支持Linux/AArch64架构，同时维护8和11两个LTS版本。同时毕昇JDK遵从GPLv2版权进行开源。

毕昇JDK、OpenJDK和Oracle JDK区别

毕昇JDK和Oracle JDK一样，都是基于开源OpenJDK定制得到。



为什么要做毕昇JDK？

- Oracle JDK授权方式发生变化

Oracle Java

Java is the strategic platform for today's demanding enterprises. For customers facing challenging business and technical requirements—such as lowering costs, simplifying system administration and maintaining high service levels - Java delivers proven results on everything from mission-critical databases to high-performance Web applications.

Go　　Actions

Product	Price	
Oracle Java SE Desktop Subscription 1 Year Term Subscription that combines Java SE Licensing and Support for use on Desktop deployments.	¥102.33 - ¥204.66	View Details
Oracle Java SE Subscription 1 Year Term Subscription that combines Java SE Licensing and Support for use on Desktops, Servers or Cloud deployments.	¥1,023.30 - ¥2,052.00	View Details
Oracle Java Development Tools Support Support for NetBeans, Oracle JDeveloper and Oracle Enterprise Pack for Eclipse.	¥8,184.00	View Details

> 数据来自于Oracle的官网

为什么要做毕昇JDK？

- 高版本JDK上有价值特性的渴望
  - JDK版本众多，不同功能/特性在不同JDK版本。程序员期望在最熟悉的JDK上尽可能多的使用高版本中有价值的特性。例如G1 GC在JDK12中引入了一个特性，把不使用的内存归还给操作系统，该特性在云场景中非常有价值，目前主流使用的还是JDK8，自研JDK中Backport特性能快速满足需求

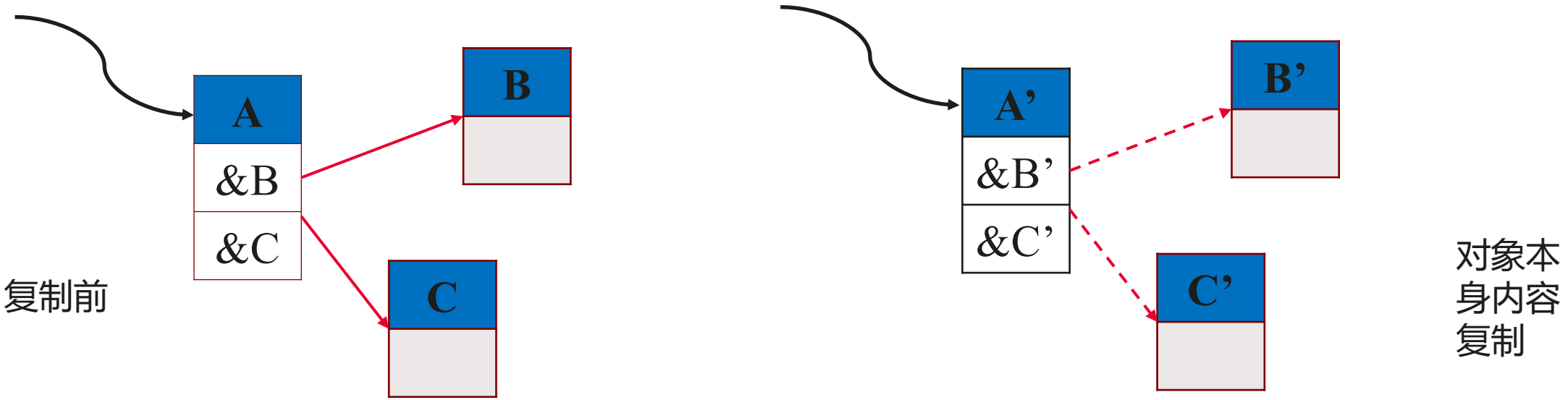
- 应用的定制化优化诉求
  - 应用在运行的硬件、场景有特殊的诉求，但这些诉求短期难以进入到社区。例如大数据应用在数学计算方面有较高诉求，在自研JDK中可以针对数学计算做循环展开、指令优化等编译优化技术，加速计算

目录

大数据场景对JDK的诉求

- 业务多样性。例如：
  - 一些应用在ARM平台上运行时GC占比明显；
  - 一些任务小，运行时间短；
  - 一些任务运行时间非常长，要求极致性的编译优化；
  - 一些应用需要超级大堆，且要求较短的停顿时间，一般的垃圾回收算法均不满足这样的要求；
  - 一些应用堆外内存使用量巨大，而目前堆外内存的使用完全依赖于应用；
  - 内存管理难度大，内存设置不合适会导致OOM或者造成内存使用浪费；
  - 目前的垃圾回收算法对一些应用不够友好，并不完全适配应用的运行机制，回收效率并不高；

Case1：针对弱内存架构的并行复制回收算法优化



对象A和B在并行复制算法中被不同的线程复制，可能由于：

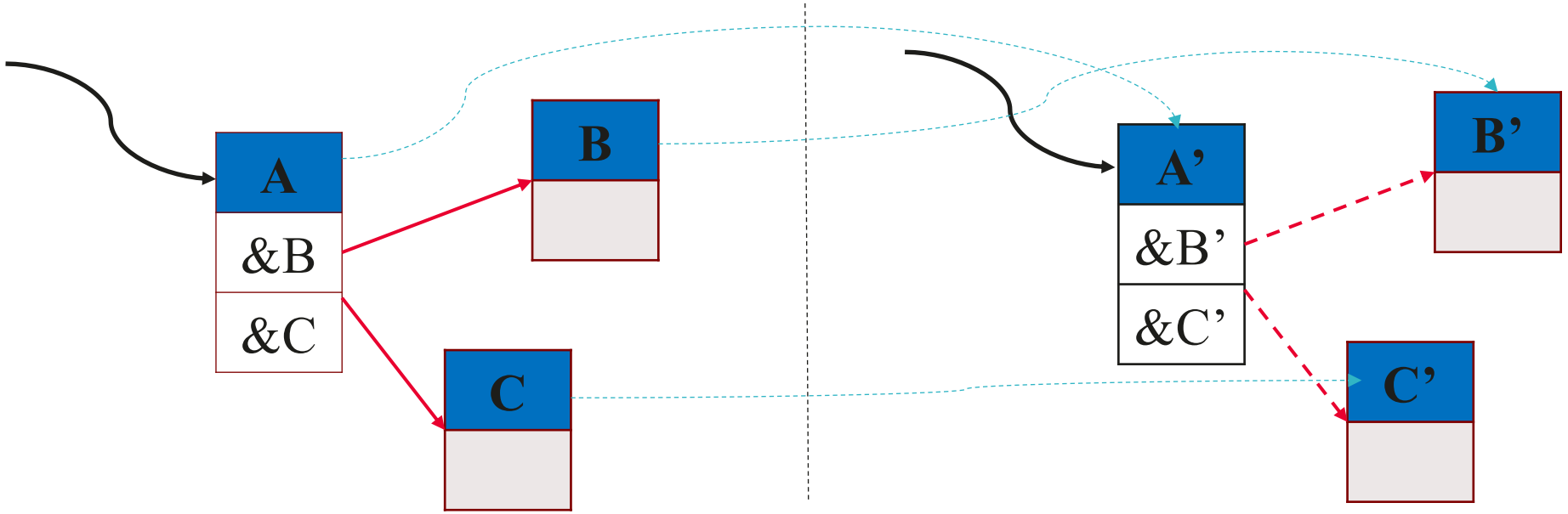
对象A和B有不同到达路径，不同的线程复制

因为任务均衡的问题，线程可以窃取其他线程的复制任务

例如有两个线程 T1和T2分别复制对象A和B，T1： A->A’； T2： B->B’;

在复制时除了复制对象的内容外，还需要使用一个指针（Forwarding Pointer）记录对象转移后地址，防止对象被重复复制

并行复制算法（续）



复制后

架构对并行复制算法的影响

- 多线程的并行工作需要考虑不同架构的内存模型。 X86是一种强内存序架构，ARM则是是一种弱内存序，它们的内存序见右表：

Type	ARM	X86	AMD64
Loads reordered after loads	Y		
Loads reordered after stores	Y		
Stores reordered after stores	Y		
Stores reordered after loads	Y	Y	Y
Atomic reordered with loads	Y		
Atomic reordered with stores	Y		
Dependent loads reordered			
Incoherent instruction cache pipeline	Y	Y	

- 对于并行复制算法来说，在弱内存架构下，**由于内存序的设计**，其他线程可能先观测到**转移指针已经更新**，但是对象尚未复制。为保证一致性，需要在复制和更新对象头之间插入membar，在JVM关于对象头更新统一抽象为CAS函数
- CAS在不同的体系结构实现不同，X86中采用cmpxchgl指令；ARM中采用ldaxr/stlxr指令

并行复制算法的流程

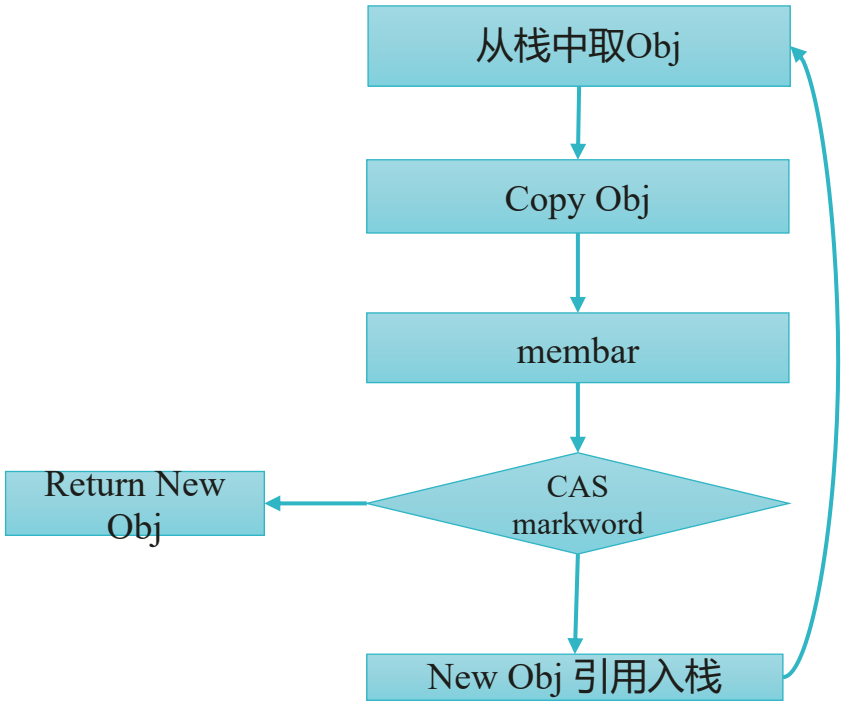
1.拷贝对象obj到新的对象位置new_obj

2.插入Memory Barrier ，对象obj通过CAS设置转移指针，若成功则执行（3），失败执行（4）

3.将new_obj的引用压入栈中，返回new_obj

4.撤销之前分配的对象，将cas成功线程的new_obj返回

在热点分析中，我们发现复制操作的60%Cpu消耗在插入Memory Barrier上。



并行复制算法的优化，减少membar

- 如果不插入Memory barrier，多个线程观察到内存不一致的情况，在什么情况下会引入问题？

- T1：尚未完成对象复制，但是已经将对象入栈
- T2：从T1的线程栈窃取待复制的对象，并对尚未完成复制的对象进行成员变量的复制更新，导致数据不一致

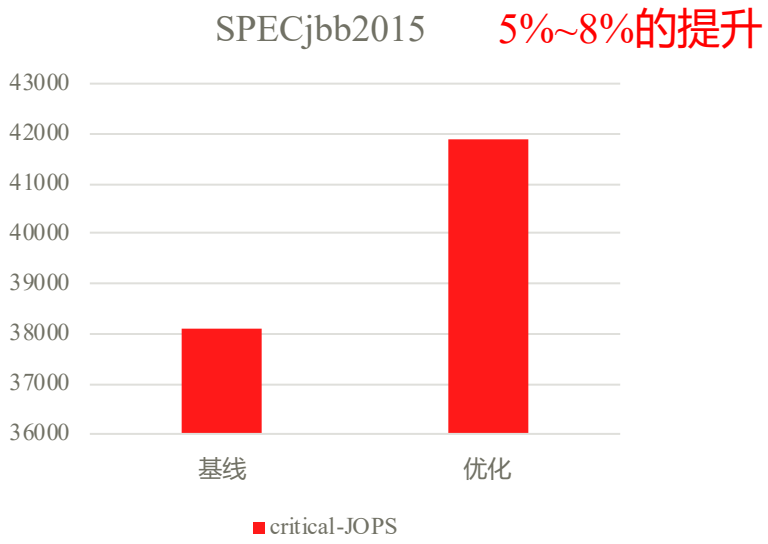
- 对于不需要复制成员变量的对象（例如：对象的成员变量全部是非引用类型；对象的成员变量其引用类型全部为NULL，对象本身是原始类型的数组），还有必要使用Memory Barrier？
- **NO！**

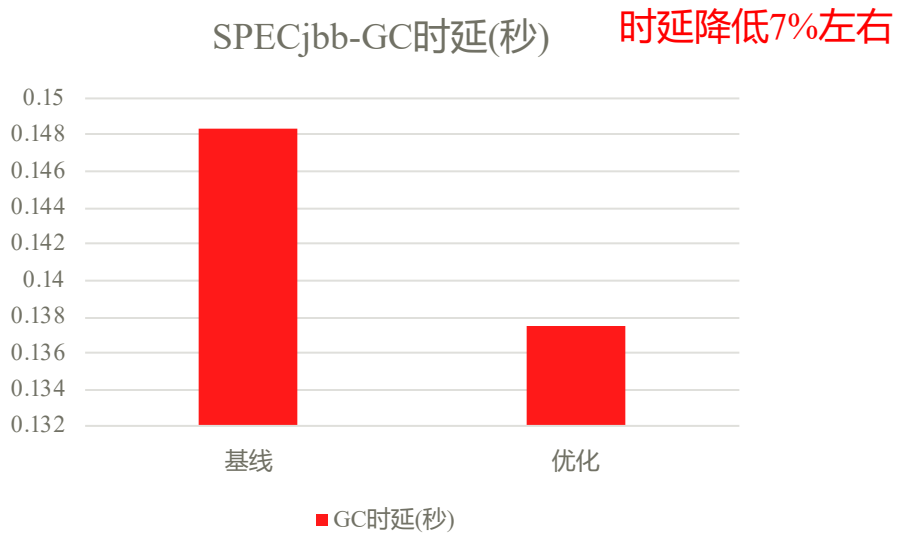
并行复制算法优化识别和优化效果

如何识别这些对象？

静态分析对象：可以发现对象的成员变量全部是非引用类型、原始类型的数组。已经开源

动态分析对象：通过屏障技术识别





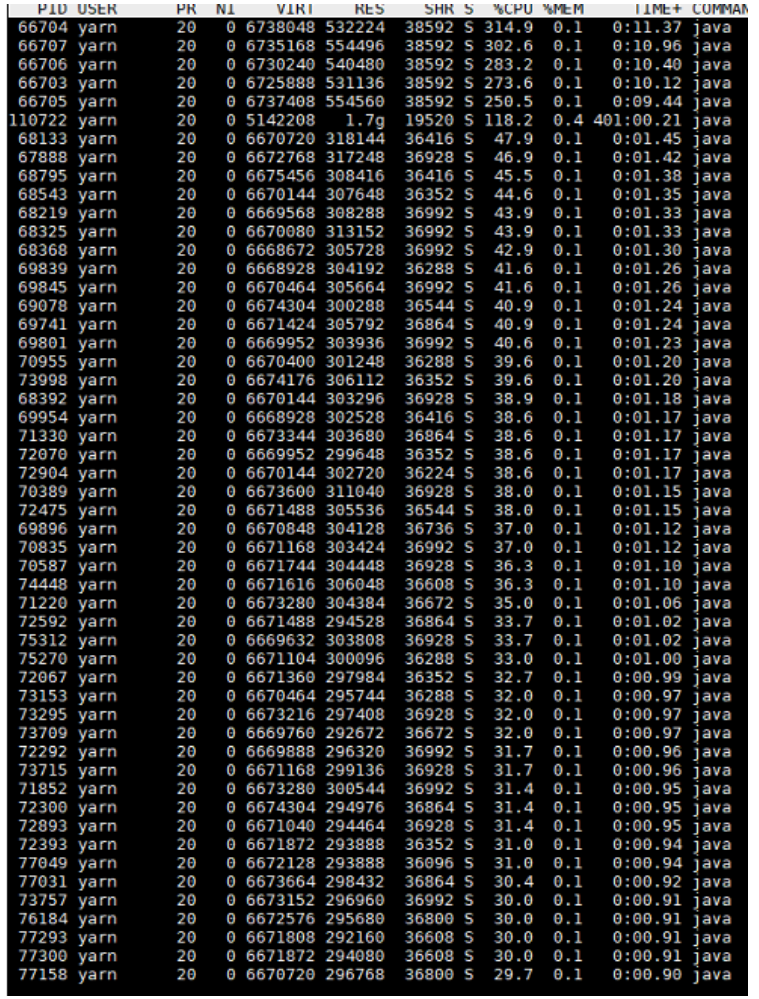
测试环境：
毕昇JDK8u 262
OS： OpenEuler 20.03
Hardwared：Taishang 200

HUAWEI

Case2：大量短时任务

- 在Hive的使用中，通过SQL进行数据查询，一个非常典型的情况：任务多、运行时间短

- JVM作为执行引擎，本身启动需要一定的资源，耗时较长。诉求就是JVM能够：快速完成启动、快速预热

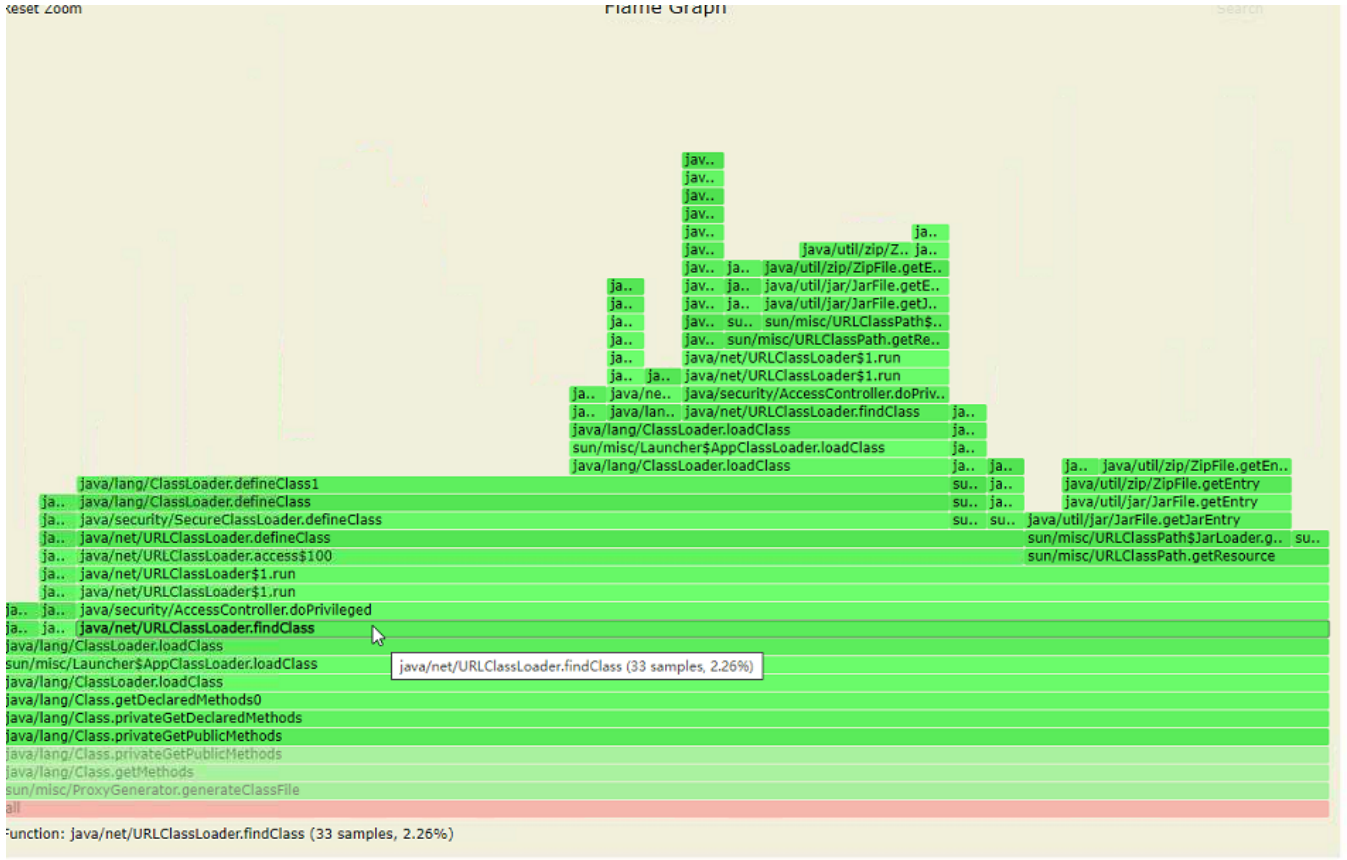


PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAN
66704	yarn	20	0	6738048	532224	38592	S	314.9	0.1	0:11.37	java
66707	yarn	20	0	6735168	554496	38592	S	302.6	0.1	0:10.96	java
66706	yarn	20	0	6730240	540480	38592	S	283.2	0.1	0:10.40	java
66703	yarn	20	0	6725888	531136	38592	S	273.6	0.1	0:10.12	java
66705	yarn	20	0	6737408	554560	38592	S	250.5	0.1	0:09.44	java
110722	yarn	20	0	5142208	1.7g	19520	S	118.2	0.4	401:00.21	java
68133	yarn	20	0	6670720	318144	36416	S	47.9	0.1	0:01.45	java
67888	yarn	20	0	6672768	317248	36928	S	46.9	0.1	0:01.42	java
68795	yarn	20	0	6675456	308416	36416	S	45.5	0.1	0:01.38	java
68543	yarn	20	0	6670144	307648	36352	S	44.6	0.1	0:01.35	java
68219	yarn	20	0	6669568	308288	36992	S	43.9	0.1	0:01.33	java
68325	yarn	20	0	6670080	313152	36992	S	43.9	0.1	0:01.33	java
68368	yarn	20	0	6668672	305728	36992	S	42.9	0.1	0:01.30	java
69839	yarn	20	0	6668928	304192	36288	S	41.6	0.1	0:01.26	java
69845	yarn	20	0	6670464	305664	36992	S	41.6	0.1	0:01.26	java
69078	yarn	20	0	6674304	300288	36544	S	40.9	0.1	0:01.24	java
69741	yarn	20	0	6671424	305792	36864	S	40.9	0.1	0:01.24	java
69801	yarn	20	0	6669952	303936	36992	S	40.6	0.1	0:01.23	java
70955	yarn	20	0	6670400	301248	36288	S	39.6	0.1	0:01.20	java
73998	yarn	20	0	6674176	306112	36352	S	39.6	0.1	0:01.20	java
68392	yarn	20	0	6670144	303296	36928	S	38.9	0.1	0:01.18	java
69954	yarn	20	0	6668928	302528	36416	S	38.6	0.1	0:01.17	java
71330	yarn	20	0	6673344	303680	36864	S	38.6	0.1	0:01.17	java
72070	yarn	20	0	6669952	299648	36352	S	38.6	0.1	0:01.17	java
72904	yarn	20	0	6670144	302720	36224	S	38.6	0.1	0:01.17	java
70389	yarn	20	0	6673600	311040	36928	S	38.0	0.1	0:01.15	java
72475	yarn	20	0	6671488	305536	36544	S	38.0	0.1	0:01.15	java
69896	yarn	20	0	6670848	304128	36736	S	37.0	0.1	0:01.12	java
70835	yarn	20	0	6671168	303424	36992	S	37.0	0.1	0:01.12	java
70587	yarn	20	0	6671744	304448	36928	S	36.3	0.1	0:01.10	java
74448	yarn	20	0	6671616	306048	36608	S	36.3	0.1	0:01.10	java
71220	yarn	20	0	6673280	304384	36672	S	35.0	0.1	0:01.06	java
72592	yarn	20	0	6671488	294528	36864	S	33.7	0.1	0:01.02	java
75312	yarn	20	0	6669632	303808	36928	S	33.7	0.1	0:01.02	java
75270	yarn	20	0	6671104	300096	36288	S	33.0	0.1	0:01.00	java
72067	yarn	20	0	6671360	297984	36352	S	32.7	0.1	0:00.99	java
73153	yarn	20	0	6670464	295744	36288	S	32.0	0.1	0:00.97	java
73295	yarn	20	0	6673216	297408	36928	S	32.0	0.1	0:00.97	java
73709	yarn	20	0	6669760	292672	36672	S	32.0	0.1	0:00.97	java
72292	yarn	20	0	6669888	296320	36992	S	31.7	0.1	0:00.96	java
73715	yarn	20	0	6671168	299136	36928	S	31.7	0.1	0:00.96	java
71852	yarn	20	0	6673280	300544	36992	S	31.4	0.1	0:00.95	java
72300	yarn	20	0	6674304	294976	36864	S	31.4	0.1	0:00.95	java
72893	yarn	20	0	6671040	294464	36928	S	31.4	0.1	0:00.95	java
72393	yarn	20	0	6671872	293888	36352	S	31.0	0.1	0:00.94	java
77049	yarn	20	0	6672128	293888	36096	S	31.0	0.1	0:00.94	java
77031	yarn	20	0	6673664	298432	36864	S	30.4	0.1	0:00.92	java
73757	yarn	20	0	6673152	296960	36992	S	30.0	0.1	0:00.91	java
76184	yarn	20	0	6672576	295680	36800	S	30.0	0.1	0:00.91	java
77293	yarn	20	0	6671808	292160	36608	S	30.0	0.1	0:00.91	java
77300	yarn	20	0	6671872	294080	36608	S	30.0	0.1	0:00.91	java
77158	yarn	20	0	6670720	296768	36800	S	29.7	0.1	0:00.90	java

根因和解决思路

- 通过火焰图得到一些信息，每个Java进程都需要花费时间进行类加载，并需要存储这些类信息。

- 解决思路是：让不同的JVM共享Class-Data信息，从而提升应用程序的启动速度。



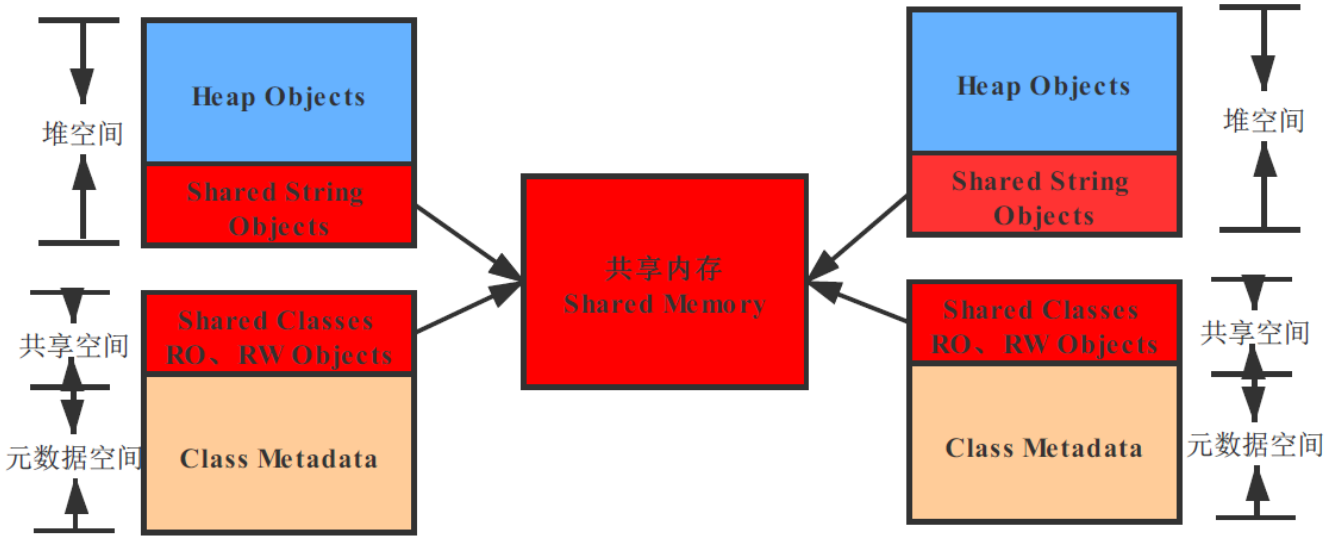
HUAWEI

AppCDS原理和实践

- 在JDK10中AppCDS的特性，其思路是将String对象，类元数据对象存放到一个共享文件中，让多个JVM进程能够通过共享信息，减少类元数据对象的加载、解析

- 毕昇JDK 8通过移植该特性，测试发现取得良好的效果。

- 网友实践：
  https://zhuanlan.zhihu.com/p/335488378



Case3：高效JIT优化

- 在大数据的机器学习框架Spark中，存在较多的非常多的数学计算，例如点积、矩阵运算等。

- 解决方法：

  >通过增强JIT，支持向量化指令加速计算。目前，向量化相关的主要工作在Panama项目中孵化。毕昇JDK在该项目中贡献了近30个patch，近5000行代码

  >通过intrinsic的方式生成编译好的指令替换编译器的工作

Intrinsic的一般步骤

- 具体ddot的实现参考gitee代码 https://gitee.com/src-openeuler/openjdk-1.8.0/blob/master/Ddot-intrinsic-implement.patch

- 通常步骤如下：
- Step 1： 找到函数原型
- Step 2： 定义intrinsic的函数签名信息
- Step 3： 生成intrinsic代码
- Step 4：运行时识别目标函数并使用inline调用intrinsic函数

JVM中浮点数运算的限制

- Java语言对浮点运算结果一致性的要求非常严格，有一个关键字strictfp来保证平台上浮点精度不一致的问题。根据JLS 15.18.2可知浮点加法不支持结合律，JIT在编译float[]和double[]类型的数组计算时需要考虑每个元素的先后顺序，从而导致向量化无法进行，流水线循环内部有依赖关系的浮点累加无法进行，加法转乘法无法进行，折叠乘法无法进行等。

Fortran77

```
      DO I = MP1,N,5
       DTEMP = DTEMP + DX(I)*DY(I) + DX(I+1)*DY(I+1) +
     $             DX(I+2)*DY(I+2) + DX(I+3)*DY(I+3) + DX(I+4)*DY(I+4)
      END DO
```

Java

```
for (i = 0; i < m; i += 4) {
    temp += dx[i] * dy[i];
    temp += dx[i + 1] * dy[i + 1];
    temp += dx[i + 2] * dy[i + 2];
    temp += dx[i + 3] * dy[i + 3];
}
return temp;
```

Java

```
for (int i = 0; i < xd.length; i += 4) {
    sp1 += dx[i] * dy[i];
    sp2 += dx[i + 1] * dy[i + 1];
    sp3 += dx[i + 2] * dy[i + 2];
    sp4 += dx[i + 3] * dy[i + 3];
}
return sp1 + sp2 + sp3 + sp4
```

关键汇编展示及说明

```
void MacroAssembler::f2j_ddot_d4(Register dx, Register dy) {
    ld1(v2, v3, T2D, post(dx, 32));
    ld1(v4, v5, T2D, post(dy, 32));
    fmul(v2, T2D, v2, v4);
    fmul(v3, T2D, v3, v5);
    fadd(v0, T2D, v0, v2);
    fadd(v6, T2D, v6, v3);
}

//等到所有计算结束，  使用fadd和faddp两条指令将累计结果reduce到v0
fadd(v0, T2D, v0, v6);
faddp  d(v0, v0);
```

- ld1指令可以向量化的加载四个double类型元素到两个向量寄存器
- fmul指令计算四组数据的两两乘法，然后将四个乘积结果保存到向量寄存器中
- fadd指令将四个中间结果向四个累加器累加，结果保存在两个向量寄存器中

性能效果，基于HiBench中Kmeans测试

测试环境：
毕昇JDK8u 262
OS： OpenEuler 20.03
Hardwared：Taishang 200

数据集	数据集维度	原生耗时（单位：秒）	JNI加速库耗时	毕昇JDK耗时	JNI加速库性能变化	毕昇JDK性能变化
D20M20	20	94	128	93	-36.6%	0.43%
D20M100	100	135	168	119	-24.17%	12.34%
D20M200	200	191	202	156	-5.7%	18.51%
D20M300	300	240	222	185	7.38%	22.72%
D20M500	500	338	299	254	11.47%	24.89%
D20M1000	1000	661	502	479	24.05%	27.44%

Case4：超大内存诉求

- 在一些以内存为中心的应用中，通常会使用超大内存。例如Presto通常配置的堆空间高达100GB以上
- 在毕昇JDK 11中补齐了ZGC在ARM平台的实现，并将其用在生成环境中
- 同时团队正在探索ZGC分代的实现，优化ZGC的吞吐量

Case5：应用适配的垃圾回收算法

- 目前JDK中存在6种常用的垃圾回收器，分别适用于不同的场景。
  - >串行（Serial GC）：适用于资源差的系统
  - >并行（Parallel Scavenge）：追求高吞吐量的应用
  - >部分并发（CMS、G1）：在停顿时间和吞吐量之间平衡
  - >完全并发（ZGC、Shenandoah GC）：追求低停顿时间

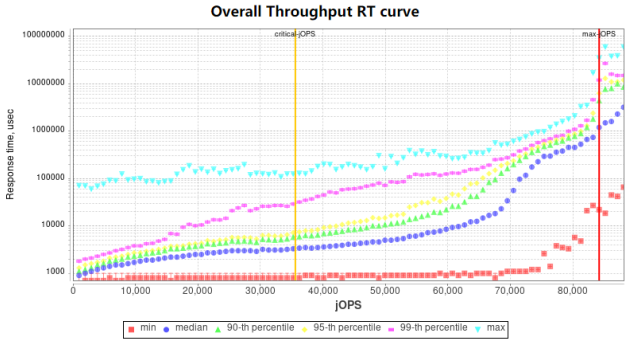
- 但大数据的一些应用有独特的内存使用方式，例如Spark在执行时可以分为控制流和数据流应用，控制流主要负责任务的启动等，而数据流应用主要典型的任务的执行。对于数据流类型的任务典型的行为是执行较快，任务结束后内存可以全部回收
- 内部实现了Epoch GC，通过代码修改的方式主动的分配和回收内存，取得一定的效果。团队正在尝试实现Thread Local GC

目录

毕昇JDK还有哪些有意思的优化

- 经过评估和测试，毕昇JDK目前还以社区的特性为基础Backport了一批有价值的特性

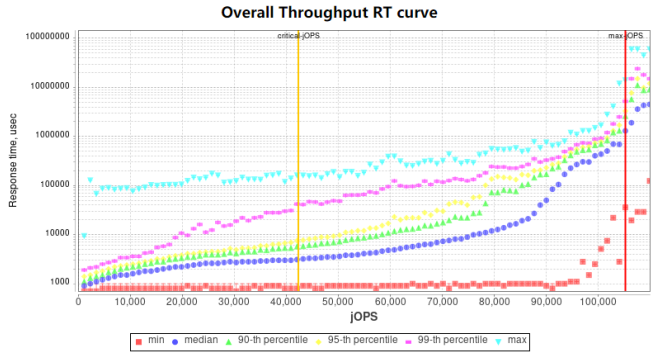
  > G1 NUMA-Aware，该特性能充分发挥NUMA的优势，在多核的硬件平台中效果更佳。毕昇JDK中还在社区的基础上修复了一些问题：例如因为操作系统的线程调度导致线程在多个节点迁移，迁移在NUMA特性上会导致一些内存分区无法得到有效回收；增强了大对象的NUMA-Aware功能



不开启NUMA-Aware，max-jOPS 84218，critical-jOPS 35598

开启NUMA-Aware后max-jOPS提升：24.87%

critical-jOPS提升：18.89%



开启NUMA-Aware：max-jOPS 105164，critical-jOPS 42324

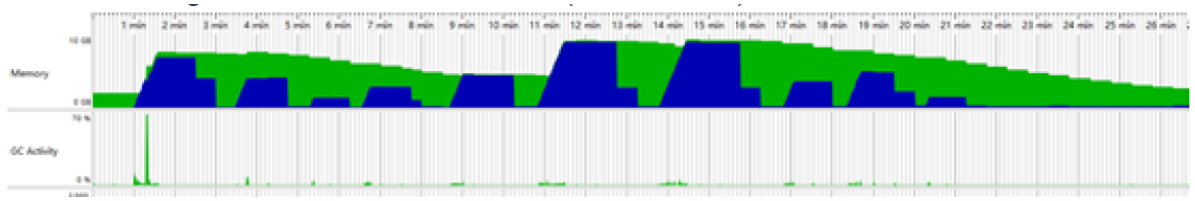
测试环境：
毕昇JDK 11已经发布该功能，毕昇JDK 8将于Q2支持该功能

HUAWEI

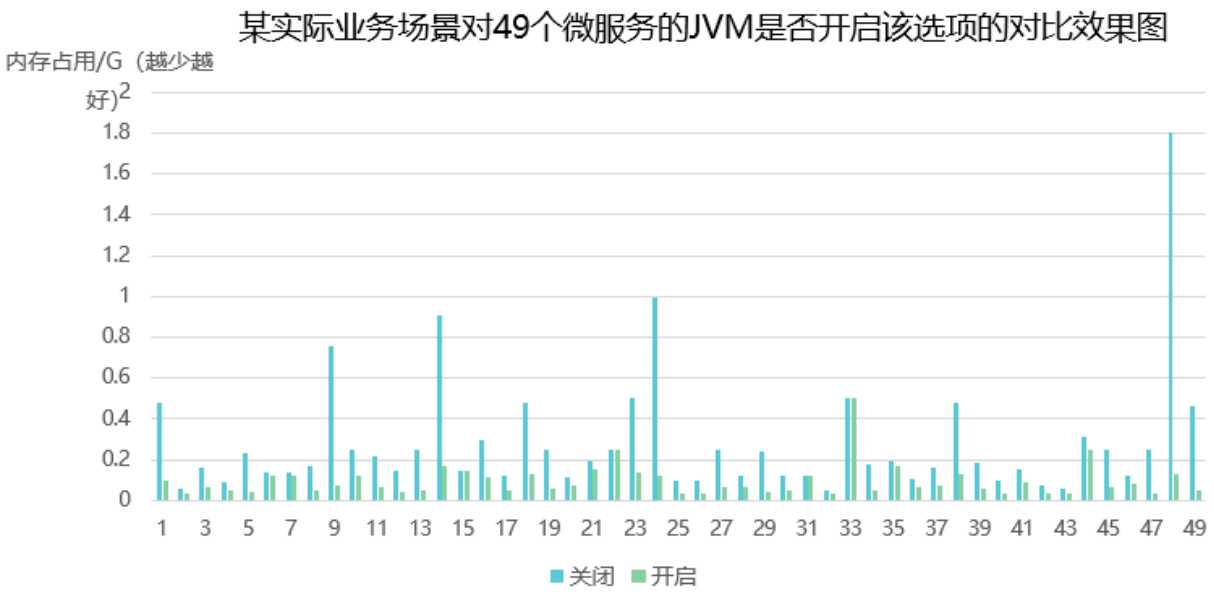
毕昇JDK还有哪些有意思的优化

> G1 Uncommit，在内存使用较低的情况下，会通过周期性的触发GC进行垃圾回收，并将回收后的内存归还给操作系统，该特性对于云场景中，能明显的降低内存的私有量。毕昇JDK在社区版本基础上，将串行的内存释放修改为并发（在最新的JDK 16中也采用了相同的实现）

> 毕昇JDK8 和JDK 11均支持该功能



开启G1UnCommit后，在内存不使用的场景中会稳步下降



HUAWEI

毕昇JDK还有哪些有意思的优化

- 并行任务窃取机制优化，在一些应用发现任务窃取占比很高。对于并行任务窃取社区有一个价值的设计，极大的优化了并行任务窃取。在毕昇JDK中，PS、ParNew、G1、Shenandoah等都因此而受益

- 目前我们正在针对多核的服务器优化任务窃取，待成熟后会继续开源

- 毕昇JDK8和JDK11均支持该功能

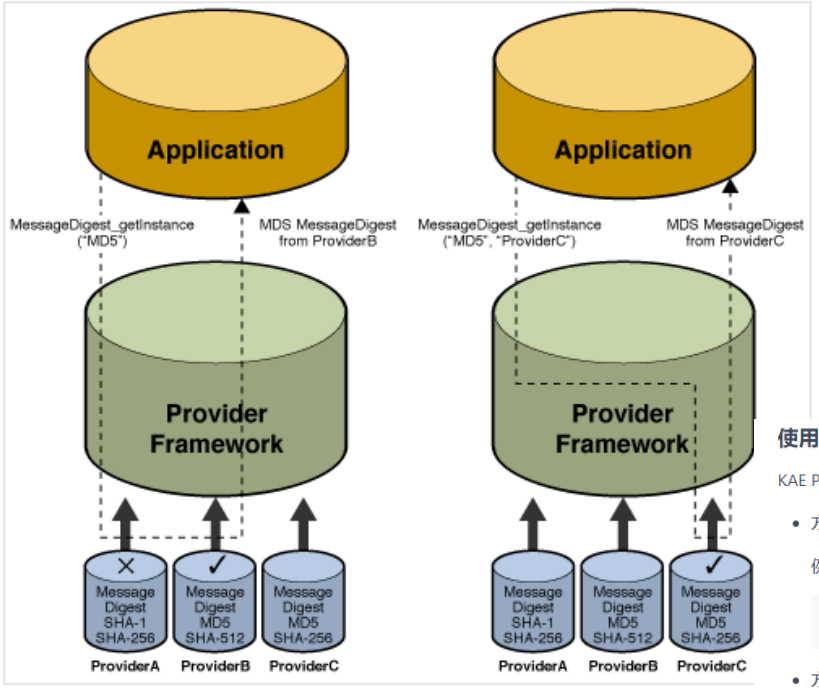
HUAWEI

毕昇JDK还有哪些有意思的优化

> 软硬协同，使用鲲鹏服务器的硬件加速功能。在鲲鹏芯片中有一个独立的I/O DIE用于处理加解密功能。

> 加解密方案是基于JCA。毕昇JDK提供了KAEProvider。在毕昇JDK最新的版本，发布了4款加解密算法（AES、Digest、HMAC、RSA），在针对Benchmark的测试中，部分算法可以加速40%，在安全领域将大大节约运行时间

> 目前仅毕昇JDK8支持该功能



使用KAE Provider

KAE Provider在Bisheng JDK中默认未启用， 可以通过如下方式启用KAE Provider:

- 方式 1： 使用**Security API** 添加KAE Provider ， 并设置其优先级。

  例：设置KAE Provider为最高优先级

```
Security.insertProviderAt(new KAEProvider(), 1);
```

- 方式 2：修改jre/lib/security/java.security文件，添加KAE Provider，并设置其优先级。

  例: 设置KAE Provider为最高优先级

```
security.provider.1=org.openeuler.security.openssl.KAEProvider
security.provider.2=sun.security.provider.Sun
security.provider.3=sun.security.rsa.SunRsaSign
security.provider.4=sun.security.ec.SunEC
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
security.provider.6=com.sun.crypto.provider.SunJCE
security.provider.7=sun.security.jgss.SunProvider
security.provider.8=com.sun.security.sasl.Provider
security.provider.9=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.10=sun.security.smartcardio.SunPCSC
security.provider.11=sun.security.mscapi.SunMSCAPI
```

毕昇JDK的定位和未来发展方向

- **现状**：
  - >毕昇JDK是开源、免费，是ARM上最稳定、最健壮、性能最好的JDK
  - >毕昇JDK社区不仅仅支持ARM平台，还会支持其他平台，任何相关JDK的问题都可以在毕昇JDK社区讨论，都会在第一时间得到回复
- **即将面世的功能：**
  - >完善KAE硬件加速算法，预计Q2发布
  - >G1 GC中并行NUMA-Aware、Full GC将落地于毕昇JDK8，预计Q2发布
  - >Jmap增强，针对CMS做并行dump，预计Q2发布
- **未来：**
  - >积极参与社区中SVE、Vector API特性的开发、演进。目前提交代码超5000行
  - >优化内存管理，正在进行： ZGC分代、Thread Local GC、AOT等项目

如何获得毕昇JDK及帮助

- JDK8的代码仓
- https://gitee.com/openeuler/bishengjdk-8

- JDK11的代码仓
- https://gitee.com/openeuler/bishengjdk-11

- JDK8和11下载链接
- https://www.hikunpeng.com/developer/devkit/compiler?data=JDK

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI

QCon+ 案例研习社

扫码学习大厂案例

学习前沿案例，向行业领先迈进

40 个
热门专题
—
行业专家把关内容筹备，
助你快速掌握最新技术发展趋势

200 个
实战案例
—
了解大厂前沿实战案例，
为 200 个真问题找到最优解

40 场
直播答疑
—
40 位技术大咖，每周分享最新
技术认知，互动答疑

365 天
持续学习
—
视频结合配套 PPT
畅学 365 天