JDJDK

可控与高效的JDK

臧琳
京东零售-数据中台-数据基础平台部-平台架构部-高性能集群研发部
zanglin5@jd.com
2019-03-20

www.jd.com

JD.京东.COM

Contents

JDK8 vs JDJDK

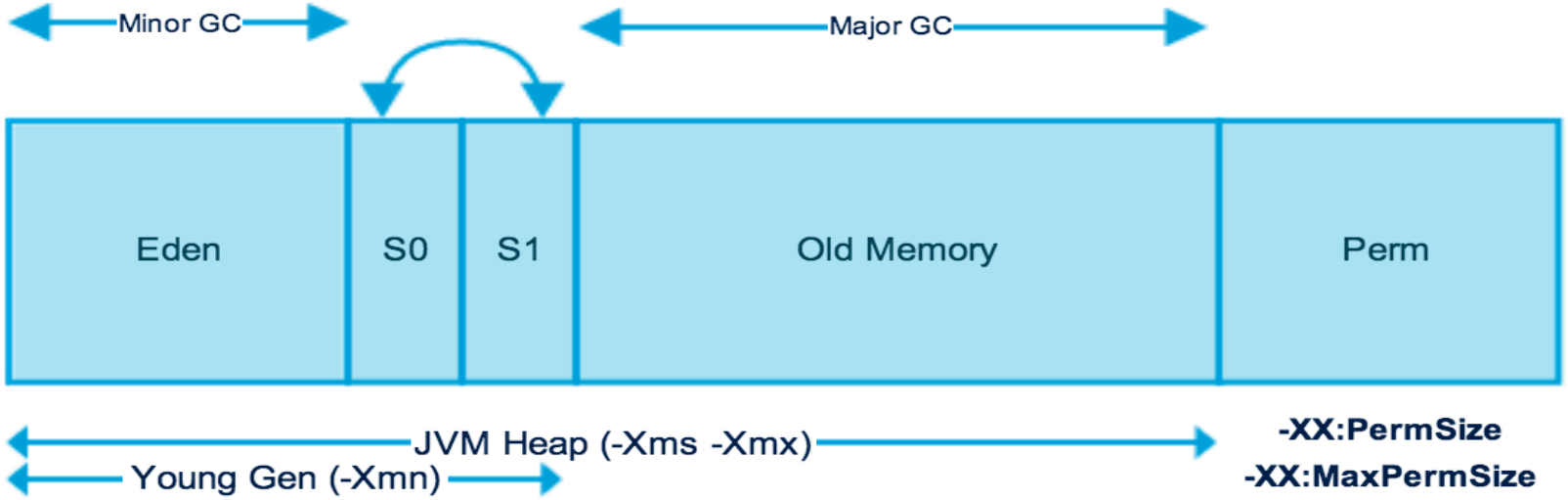
JDK 8 vs JDJDK

- 30+ JDK重要特性
  - Runtime Optimizations
    - Improve Contended Locking
    - App CDS
    - …
  - Compiler optimizations
    - Segmented Code Cache
    - JVMCI
    - AOT
    - …
  - GC Optimizations
    - Unified GC logging
    - Make G1 default
    - G1 parallel Full GC
    - …
  - Tooling/Debugging
    - More tools
    - Stack walking API
    - Unified VM logging
    - JFR
  - 自研特性
    - 3 JDK12重要特性
      - 2 G1 optimizations
      - 1 Default CDS
    - 性能优化
      - 更适合大堆的G1
    - 工具增强
      - Jmap
    - 功能拓展
      - 定时GC
      - 内存管理

回顾 G1 vs CMS

回顾：G1 VS CMS

- CMS
  - 年轻代
    - 连续
    - 固定
  - 老年代
    - 并行非拷贝
  - 碎片化



- G1
  - 年轻代
    - 连续不连续
    - 不固定
  - 老年代
    - 连续不连续
    - 碎片低

回顾：G1 VS CMS

	CMS	G1	备注
分配算法	Bump Pointer + TLAB	Bump Pointer + TLAB	
YoungGC	Parallel Copying + promotion	Regional Parallel Copying + promotion	G1优势: 基于region的批处理
MajorGC (old GC/ MixGC)	Whole old Gen Concurrent marking No copy	Partial old gen Concurrent marking copying	G1优势: 回收目标可配置， 暂停时间可尽量保证， 最小化碎片问题
Full GC	STW Mark Compact	STW parallel Mark Compact	G1优势: 并行压缩算法， 效率高， 暂停短
综合	高效分配， 回收目标不可配置， 暂停时间不可控， 老年代碎片化， Heap利用率低 FullGC易发生	高效分配， 可动态配置的回收目标 暂停时间可尽量保障 无碎片化问题 Heap利用率高 FullGC几乎不发生	G1已经取代CMS成为JDK的默认垃圾收集器 开源社区对于CMS不会在做进一步升级和调优

3

痛点list & JDJDK解决方案

痛点 list

- Full GC 和 Old GC
- 令人头疼的OOM
- JVM 内存占用
- GC停顿时间
- JVM中的玄学 – 参数调优
- 大堆下的分配性能提升
- Jmap 堆信息工具

Full GC And Old GC

- Full GC
  - 相见不如怀念, 怀念不如不见
    - OOM 前的最后一根稻草
      - G1/CMS Full GC
        - » 回收量最大，最全
        - » 回收所有垃圾
      - 停顿时间最长
        - 轻则造成系统抖动，重则系统崩溃
        - HDFS NN 上 180GB Heap 一次Full GC直接造成进程长时间假死
      - 禁止application做FullGC -XX:+DisableExplicitGC
- Old GC
  - OldGC 并发 + 部分收集 / 停顿 + 部分收集
  - 过犹不及， 难以控制
    - 过多 -> 系统抖动, 影响YGC
    - 过少 -> Full GC

Full GC And Old GC

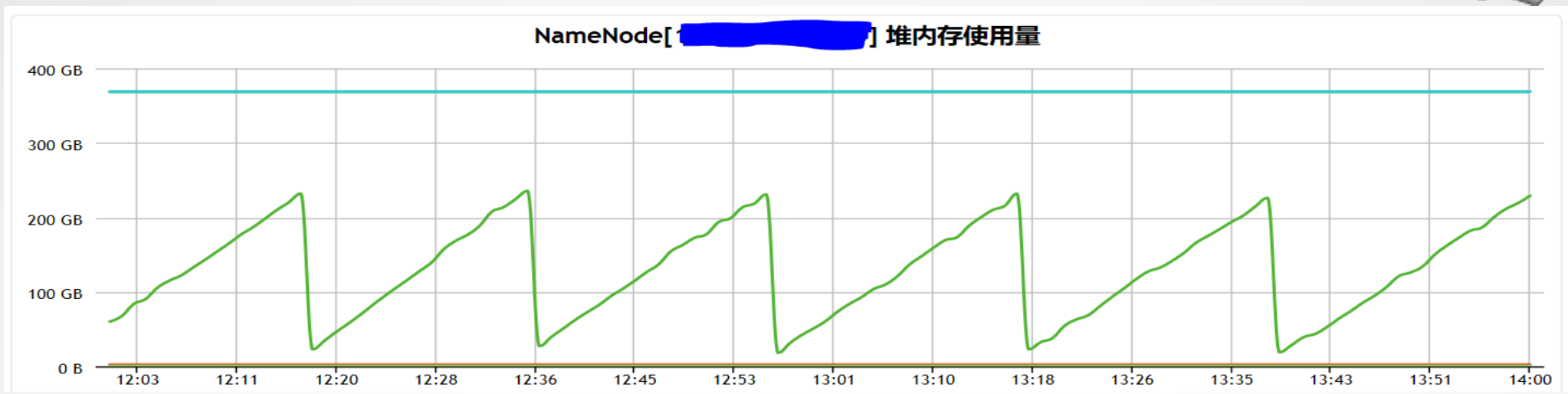
- OpenJDK
  - 参数调优 （见后）
  - 缺少有效的机制
- JDJDK - 在对的时间让对的事情发生
  - G1 - 定时 Old GC <可选GC工作量/可选FullGC>
  - 减少不可控的GC发生的几率和力度
  - 动态参数
    - jcmd vm.set_flag G1PeriodicGCForceSpanBeginAt 18:00
    - jcmd vm.set_flag G1PeriodicGCForceInterval 20000
    - jcmd vm.set_flag G1PeriodicGCForceSpanLengthInMin 3
    - jcmd vm.set_flag G1PeriodicGCForce true

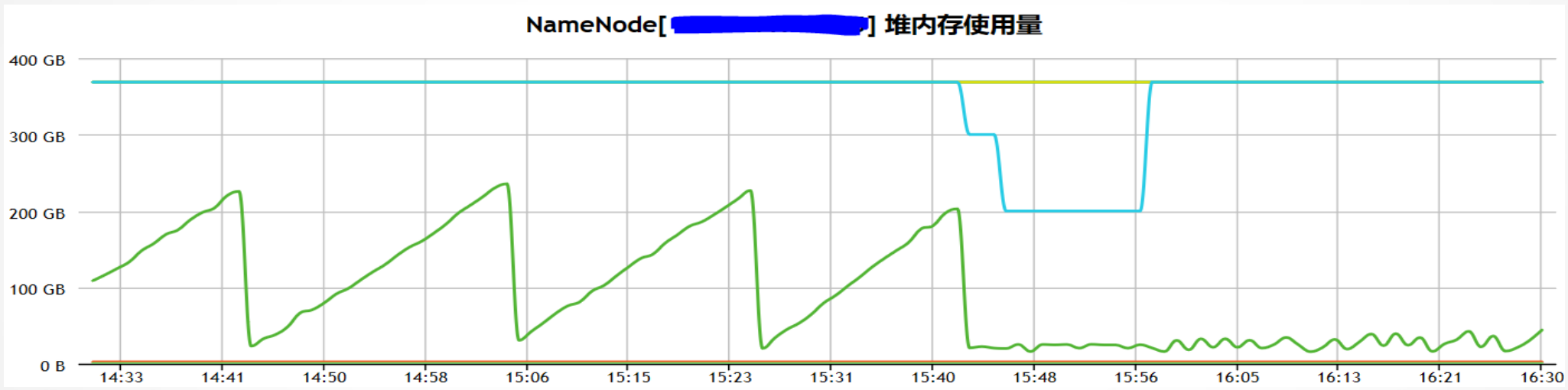
令人头疼的OOM & FullGC

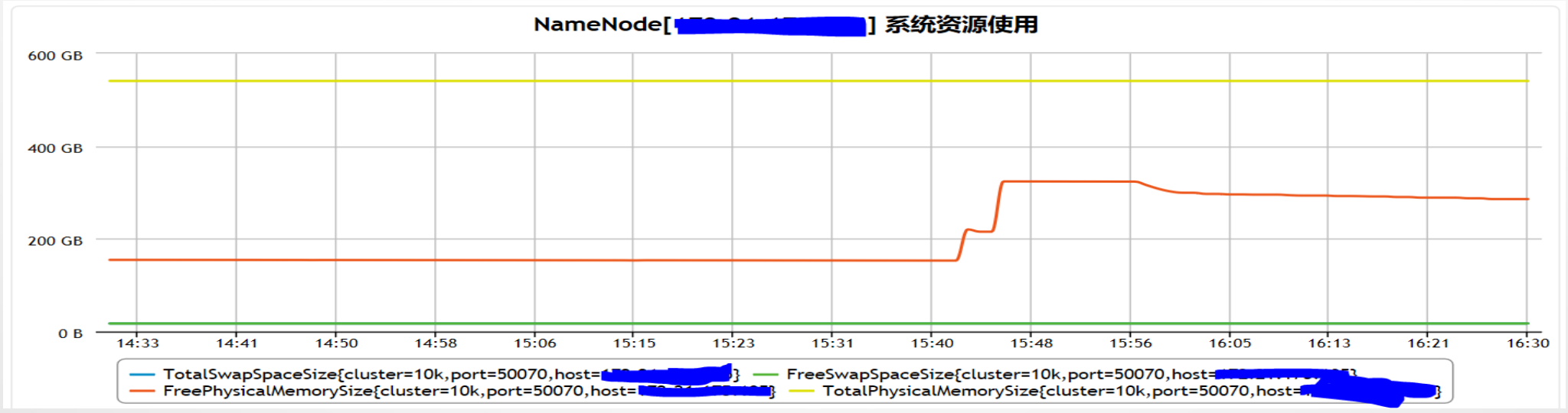
- 行为
  - Allocation -> YGC …->Old GC…->YGC..->…Full GC->OOM
- OOM
  - Heap需求突然增加
- Full GC
  - 尽管有定时GC， 仍存在几率
  - Object 突然井喷
- OpenJDK
  - 加大Xmx重启
- JDJDK
  - 90% Heap利用率预警
  - 动态增长内存：
    - -XX:ExtensibleHeapRatio = 20
    - Jcmd <pid> VM.set_flag UseExtensibleHeap true
    - 不用不发力

JVM 内存占用

- Xmx Xms
  - 性能还是存储
- Used， committed，Capacity
- 是否承诺了太多，还是原本给的就不够?
  - Xmx 大 -> 避免OOM / 系统资源浪费， Native需求不够
  - Xmx 小 -> FullGC/OOM
  - 实际heap使用率可能保持在一定的范围之内
  - 特定情况下，我们需要JVM“让”出一部分内存
    - OpenJDK
      - 又要性能又要存储，我也没有办法啊
    - JDJDK
      -XX:JDSwitchesHeapControlCeil=2000（M）

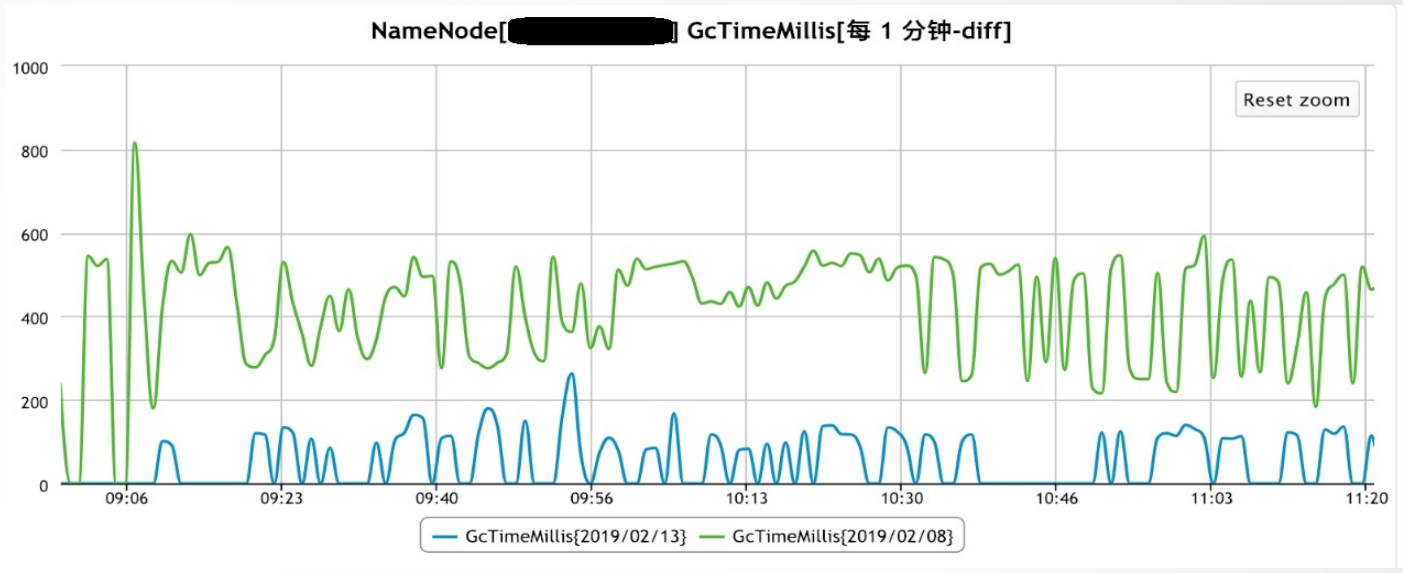

NameNode[ ] 堆内存使用量
400 GB
300 GB
200 GB
100 GB
0 B
12:03
12:11
12:20
12:28
12:36
12:45
12:53
13:01
13:10
13:18
13:26
13:35
13:43
13:51
14:00


NameNode[ ] 堆内存使用量
400 GB
300 GB
200 GB
100 GB
0 B
14:33
14:41
14:50
14:58
15:06
15:15
15:23
15:31
15:40
15:48
15:56
16:05
16:13
16:21
16:30

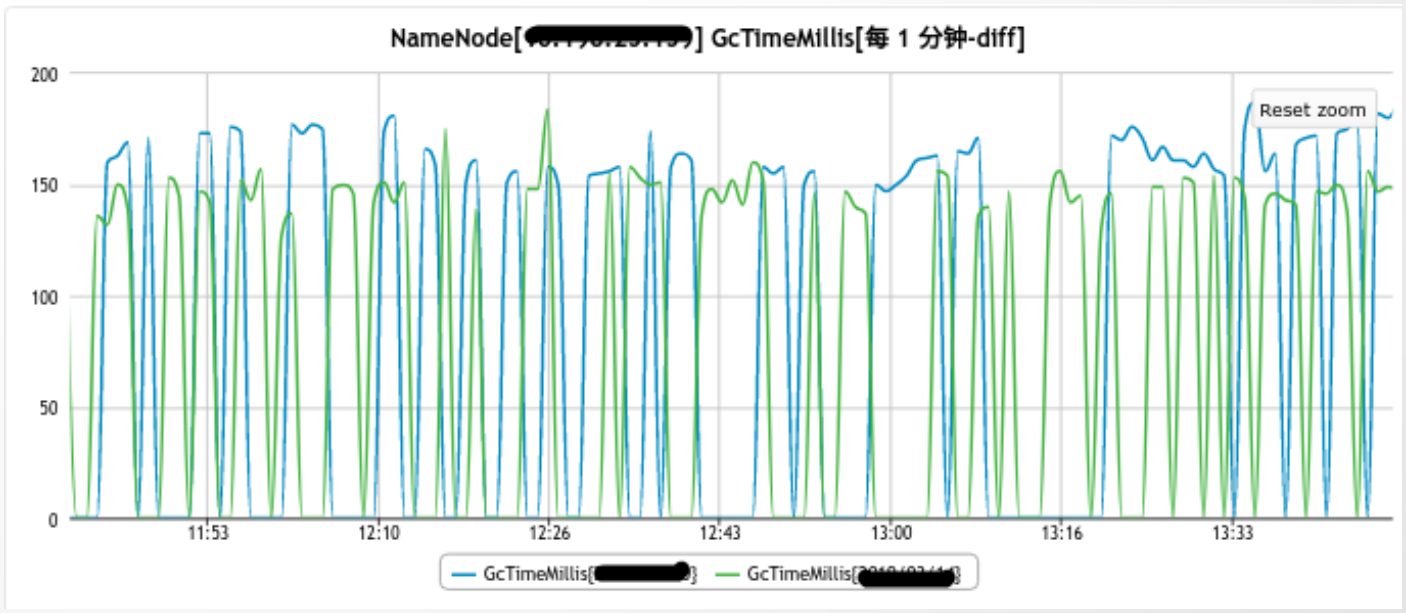

NameNode[ ] 系统资源使用
600 GB
400 GB
200 GB
0 B
14:33
14:41
14:50
14:58
15:06
15:15
15:23
15:31
15:40
15:48
15:56
16:05
16:13
16:21
16:30
TotalSwapSpaceSize{cluster=10k,port=50070,host=
FreeSwapSpaceSize{cluster=10k,port=50070,host=
FreePhysicalMemorySize{cluster=10k,port=50070,host=
TotalPhysicalMemorySize{cluster=10k,port=50070,host=

GC 停顿时间

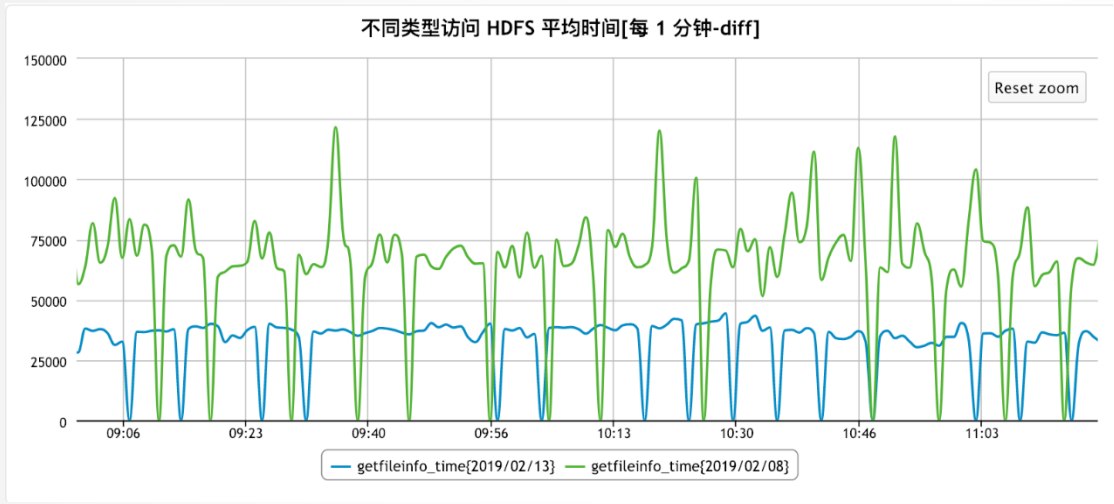
- 别让我等太久
- 停顿时间
  - G1 VS CMS
  - Openjdk VS JDJDK

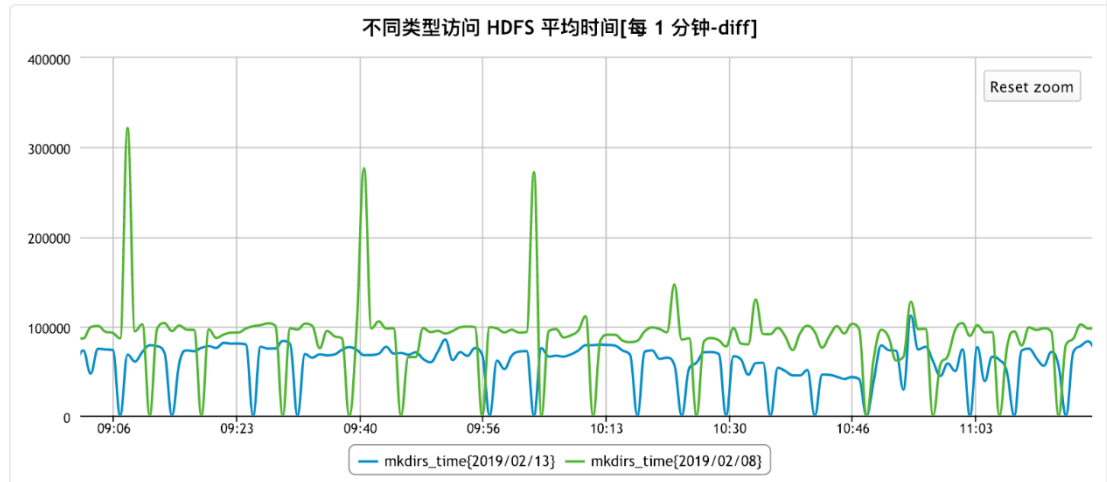


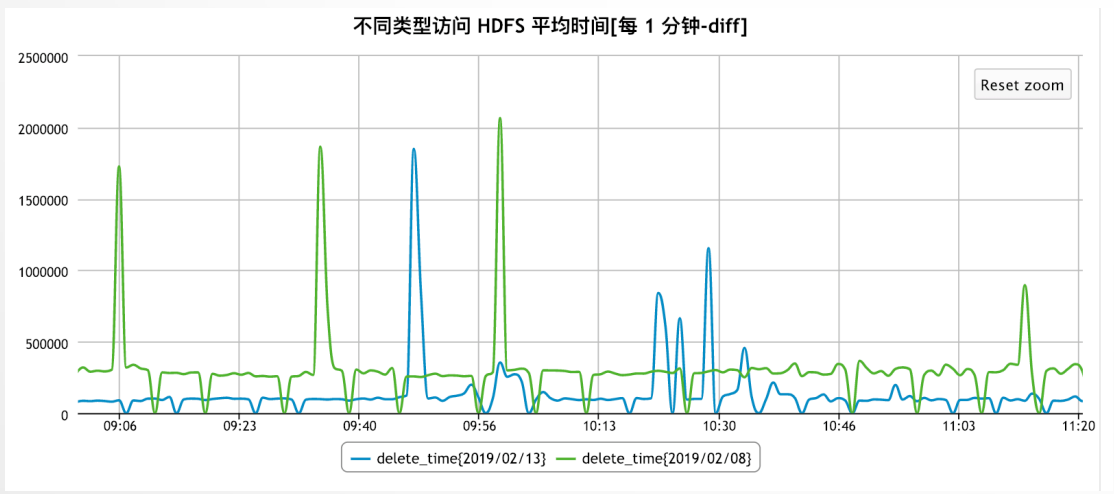
- JDJDK
  - 在JDK12新feature
  - 结合定时GC
  - 自研优化

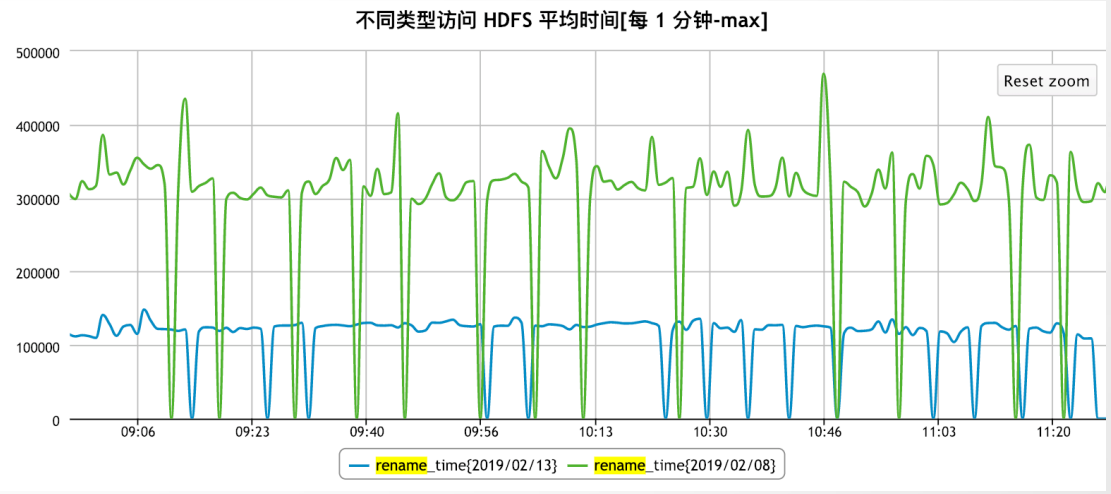


大堆下分配性能


不同类型访问 HDFS 平均时间[每 1 分钟-diff]
Reset zoom
150000
125000
100000
75000
50000
25000
0
09:06
09:23
09:40
09:56
10:13
10:30
10:46
11:03
getfileinfo_time{2019/02/13}
getfileinfo_time{2019/02/08}

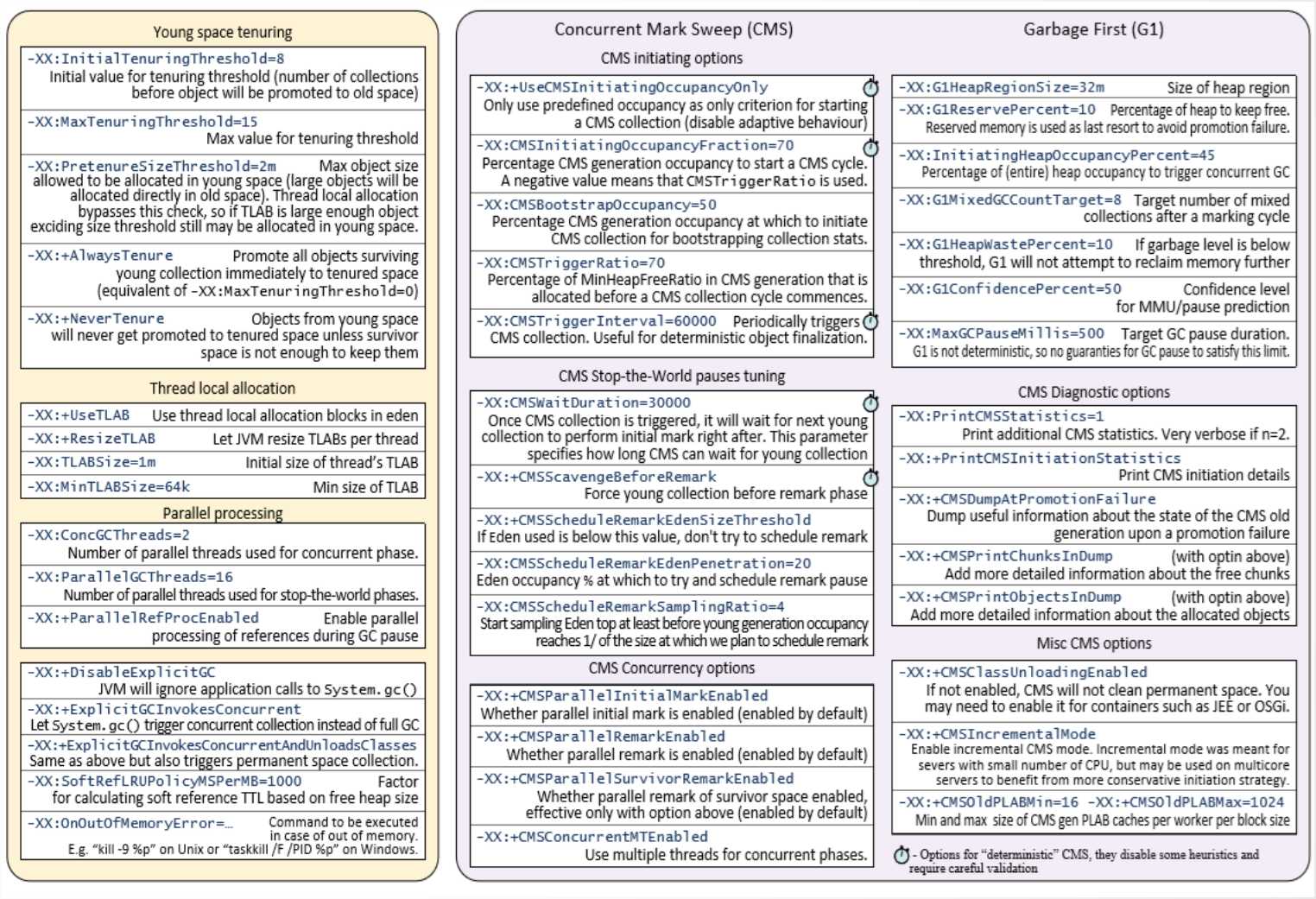

不同类型访问 HDFS 平均时间[每 1 分钟-diff]
Reset zoom
400000
300000
200000
100000
0
09:06
09:23
09:40
09:56
10:13
10:30
10:46
11:03
mkdirs_time{2019/02/13}
mkdirs_time{2019/02/08}


不同类型访问 HDFS 平均时间[每 1 分钟-diff]
Reset zoom
2500000
2000000
1500000
1000000
500000
0
09:06
09:23
09:40
09:56
10:13
10:30
10:46
11:03
11:20
delete_time{2019/02/13}
delete_time{2019/02/08}


不同类型访问 HDFS 平均时间[每 1 分钟-max]
Reset zoom
500000
400000
300000
200000
100000
0
09:06
09:23
09:40
09:56
10:13
10:30
10:46
11:03
11:20
rename_time{2019/02/13}
rename_time{2019/02/08}

一顿操作猛如虎 - 参数调优

- 怎么调参减少GC?
- YGC频繁怎么办
- 该不该触发Old GC?
- 啥是tenringThreshold
- ParallGCThread = ?
- concGCThread = ?
- Xmx Xms Xmn
- AlwaysPretouch
- CMSxxxx



验证一看原地杵

举个栗子（案例）

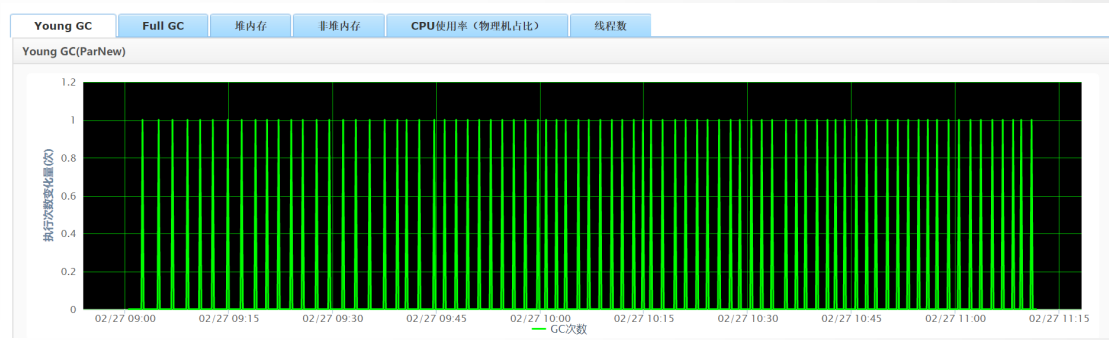
- 问题

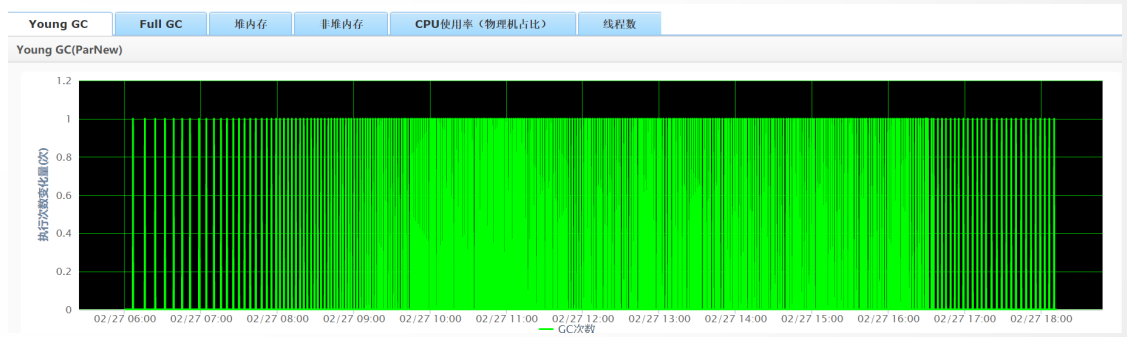
“ParNew+cms上线后发现不再有每小时定时full gc的情况，但是运行几天后会出现应用的tp99逐步升高的情况”

  - ParNew+CMS
  - No FullGC
  - Tp99 slow down
- 猜测:
  - YGC 越来越频繁
  - 或者 OLD GC越来越多
- 怎么确定呢?
  - gclog + 监控

案例（续）

- Gclog显示
  - 一直没有出现 Old GC
  - YGC 频繁
    - 高峰时段100ms发生一次
  - Old space 增长缓慢
    - 每15-18小时 增长100MB
  - 推断任务特点
    - 短/多/快
  - “响应时间都在5ms以内的实时服务”
- 监控显示YGC频率逐渐增高
  - why



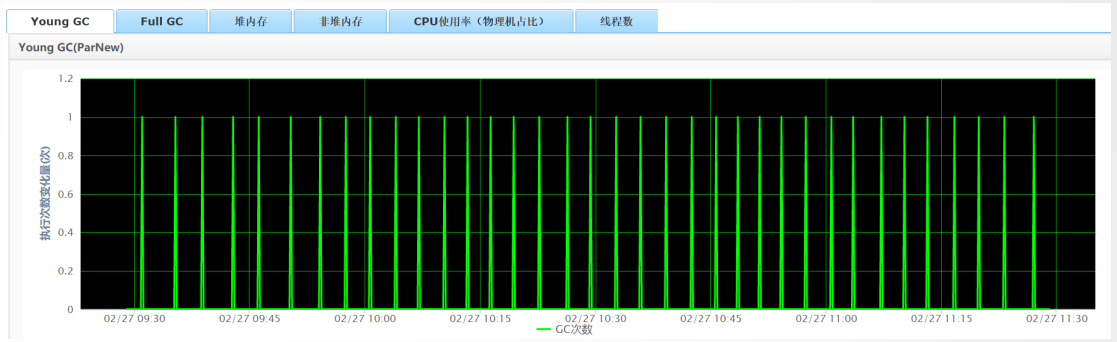


案例（续）

- 启动参数
  - -Xms4096m -Xmx4096m -XX:MaxPermSize=512m -Xmn1024M -XX:PermSize=128M
  - -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70 -XX:+UseCMSInitiatingOccupancyOnly -XX:+CMSClassUnloadingEnabled
  - -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:/export/Logs/sopocs.ofc.jd.com/gc.log
  - -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/export/Instances/sopocs/server1/logs XX:ErrorFile=/export/Instances/sopocs/server1/logs/java_error_%p.log
- 结论
  - 没发生oldGC 所以和old 没关系 吗?
  - YGC频率高，考虑YGC
- 建议
  - 增大Xmn 加大YGC间隔

—顿操作猛如虎！

案例（续）



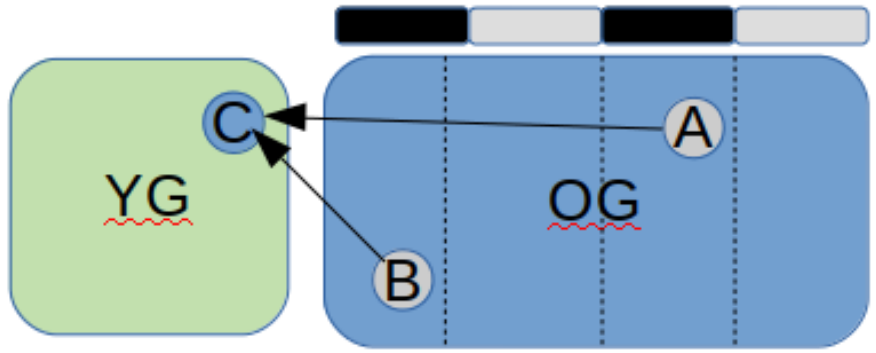
- 结果
  - Xmn=2000m
  - YGC频率确实低了
  - Tp99趋势没变
  - 和xmn=1000m看起来，表现差不多

- 详细现象
  - 系统刚启动的时候，tp99在4ms左右，运行多天后，达到20ms
  - YGC 越来越频繁
  - 始终没有old GC
    - -XX:CMSInitiatingOccupancyFraction=70
    - -XX:+UseCMSInitiatingOccupancyOnly
    - old space增长缓慢
    - 难道是old gc的原因?

测试结果原地杵

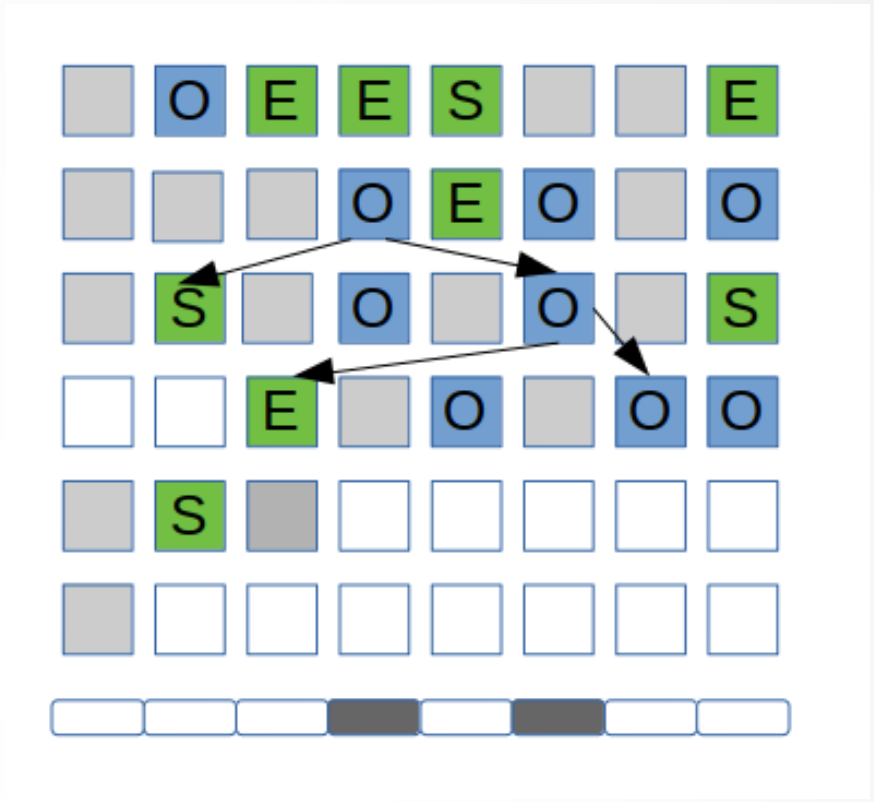
案例（续）

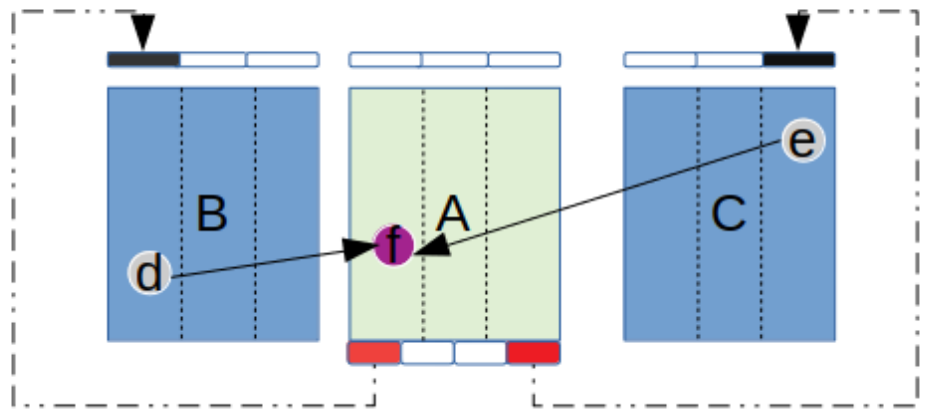
- 分析
  - Old object 不被回收
    - 跨代引用积累
    - Card table 扫描耗时
    - YGC越来越长 越来越频繁
  - 触发老年代gc
    - 性能影响 待查
  - 验证
    - GC 日志分析
- 结果
  - -XX:CMSInitiatingOccupancyFraction=10 (old gen达到100M左右触发OLDGC)
  - -XX:MaxTenuringThreshold=15 （尽量晚（少）把对象加入老年代）
  - 观察结果大概1天半触发一次old gc
  - Tp99稳定



案例（续）

G1 怎么样?





其他重要特性

- Heap
  - 180g → 360g
    - fullGC
  - 文件数承载能力从4亿上升到10亿
    - 系统更加稳定

JDJDK 展望

- 你将看到……
- 高度优化的JVM
  - 半自治堆
  - 智能自适应调优系统
  - 丰富的线上处理工具集
    - Profiling
    - Debugging
    - Gceasy/fastthread… internal version
    - Heap dump analyzer for extrem large heap
    - Java flame graph
  - ……

关于JDJDK

JDJDK 试用版：

http://blog.jd.local/bbs/article/1552622568648

JDJDK issues & 需求：（项目名： JDJDK）

http://jira.jd.com/secure/Dashboard.jspa

JDJDK项目源码：

http://source.jd.com/app/JDJDK

联系我们：

京东JDK交流群：81523801 （timline）

微信二维码：

咚咚: 81523801

THANKS

请美美的使用JDK, 烦心事都交给我们