CMS 垃圾回收器

1. Young GC（Minor GC）阶段使用ParNew垃圾回收器。
2. Old GC（Major GC）阶段使用 Background CMS GC。
3. Background CMS GC，当Old区内存占比超过阈值时触发，会经历CMS GC的所有阶段，STW时间短。
4. Foreground CMS GC，只走CMS GC其中一些阶段（省去并行阶段），最后会在STW下做内存整理。
5. 正常的Full GC，Young GC + Foreground CMS GC
6. 并行的Full GC，Young GC + Background CMS GC

lendengine的JVM调优

1. Minor GC频率高，STW长；Major GC的STW长

问题

1. 业务高峰期，Minor GC 每秒 2 次，每次 60 - 80ms。
2. 业务高峰期，Full GC 每分钟 2 次，耗时 1 秒。

背景

1. 每次get操作confplus配置时，均会对配置字符串反序列化。资金配置中心配置500KB以上，反序列化耗时20ms，会产生大量对象，导致内存分配压力大，增加GC压力，导致YGC频率上升。

优化

1. Minor GC频繁且耗时长80ms左右：获取业务配置使用ThreadLocal优化；优化StringTable初始容量，避免频繁ReHash及优化Minor GC。
2. Full GC停顿200ms左右，业务难以忍受。改为G1，停顿80ms以内
3. 中间件confplus完善配置缓存机制，读取远端缓存时直接反序列化，而后业务流程使用时直接获取反序列化的结果。大大降低内存分配压力，减少YGC压力，增加业务流程的时效性，路由耗时降低200ms+。

效果

1. 业务高峰期，Minor GC 每秒 1 次，每次 40 - 60ms。
2. 业务高峰期，Full GC 每小时 2 次，耗时 1 秒。

2. 应用偶尔重启的问题，POD的OOMKilled，因为sidecar应用占用内存

一、复盘全过程

问题

某个时间，lendengine应用开始频繁重启，平均每 5 分钟所有Pod重启一遍。

定位及解决问题

1. 异常时间和系统上线时间不重合，基本排除上线导致的问题。
2. 排查技术监控「JVM、接口TPS波动等」及业务监控，以期发现问题。都很正常。
3. 业务运维同时开始问题排查，拉取异常时间点的配置变更及应用变更。发现消息中间件的配置有变更。
4. 小会讨论配置变更及解决方案。
5. 业务运维回滚配置，应用恢复正常。

复盘单及复盘讨论会

1. 中间件维护同事为复盘责任人，负责撰写复盘单。复盘单包括（1）问题发现、排查、解决的全过程及相应的时间节点。（2）问题造成的影响。（3）后续的改进措施。
2. 开复盘讨论会：（1）如何避免之后发生此问题。（2）各个系统如何改进，减少问题的影响，高可用性。

改进项的开发、上线

1. 基于复盘会的决议生成改进项，交由各个业务系统开发、上线。

二、技术分析

技术背景

1. 架构方案：中间件对日志打印做了优化「降低日志推送的频率，更换压缩算法等」，导致对直接内存的使用较多。应用日志通过中间件组件输出日志到DirectByteBuffer，基于Kafka的性能优化「Kafka消息批量发送性能更高」日志可缓存一定时间，而后适时推送至Broker。消息发送时间间隔可以自由配置。导致容器OOMkilled。
2. 因为lendengine借款路由日志量偏大，占用大量的堆外内存，
3. 资金方撮合匹配，单次业务操作产生的日志大小：500KB，相当于对资金撮合匹配操作打了快照，方便后续的问题追踪。
4. 消息每10秒批量发送一次。
5. 业务高峰期TPS：100
6. 最大日志量 500MB：100 X 10 X 500KB，日志并不会全都落日志系统， 大部分日志通过Flink实时计算统计值，用于业务分析、监控、告警等。
7. 每次撮合匹配实际落日志10KB，一天日志量10GB = 10MB * 1000 = 10KB * 100W

优化方法

1. 优化Pod的内存分配：lendengine调低堆内存，由70%降低至60%「2800MB -> 2400MB」，中间件优化直接内存的占用。
2. lendengine优化线程：调整 -Xss，降低一半，到512KB。线程池治理，减少线程池数量。
3. 「JVM虚拟机 - 参考2.4.4节」直接内存（Direct Memory）的容量大小可通过-XX：MaxDirectMemorySize参数来指定，如果不去指定，则默认与Java堆最大值（由-Xmx指定）一致。通过限制一个较小的值「如256MB，当前未配置，等于堆内存」，可以看到「直接内存溢出异常」。**可以快速知道应用重启原因**

```
Exception in thread "main" java.lang.OutOfMemoryError
    at sun.misc.Unsafe.allocateMemory(Native Method)
```

4. 基于服务网格，日志采集中间件采用sidecar方案采集日志，go语言，中间件逻辑与业务应用容器解耦，避免因为中间件的问题导致影响应用容器，避免问题难以排查「不知道谁导致了这个问题」。降低业务应用的复杂度。
5. 监控：基础架构部门优化对直接内存、sidecar的监控，监控其异常的波动，监控是否达到pod临界值。
6. 中间件进行调整前发布通知。

影响

1. 业务运维角色及时拉取了故障发生时的配置变更记录，定位到了异常配置项，及时进行了回滚处理。因为业务运维管控了所有的上线变更、配置变更，可以迅速对单个应用回滚proplus，回滚到上一个制品，对生产不会有明显的影响。
2. 接口请求的失败率上升，用户有感知。