读书笔记 4.2 应用场景分析 2
吞吐量优先场景

Broker 端消息过滤： 有效减少无效消息到 Consumer

对同一个应用， 尽可能只用一个 Topic，通过消息子类型的 tag（每条消息一个 tag）
其次是 key，broker 有专门的索引文件，存储 key =》 消息的映射， 尽可能时 key 唯一

Tag 在 consumer 代码中，服务端过滤，
key 主要用于命令行查询

3 种方式：

1. **消息的 Tag 和 Key**
   tag 在创建消息的时候添加， 一个消息一个 tag
   不用担心 hash 冲突， 被消费前，会 对比 Message Tag 字符串，消除 hash 冲突的误读

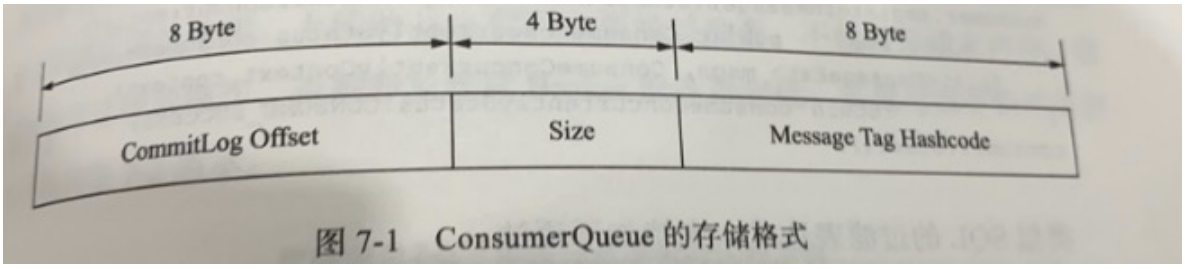


图 7-1 ConsumerQueue 的存储格式

2、SQL 表达式的方式过滤
3、Filter Server 过滤

提高 Consumer 处理能力

1. 提高消费并发度： 同一个 ConsumerGroup 下（clustering 方式），增加 consumer 实例， 注意数量不要超过 Topic 下 Read queue
   超过的实例收不到消息

   或者提高单个实例并发数 consumerThreadMin ThreadMax

2. 批量方式消费
   某些场景，例如批量 update 一次 10 条远远小于 十次 update
   通过 consumerMessageBatchMaxSize 默认 1， 设置 N 则每次收到 的是 长度

为N的消息链表

3. 检测延迟情况，跳过非重要消息
   严重的消息积压时，丢弃不重要的消息
   如下图实例 ，积压超过9000 丢弃消息

```
public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {
    long offset = msgs.get(0).getQueueOffset();
    String maxOffSet = msgs.get(0).getProperty(MessageConst.PROPERTY_MAX_OFFSET);
    long diff = Long.parseLong(maxOffSet) - offset;
    if (diff > 90000) {
        return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
    }
    //正常业务
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
}
```

Consumer的负载均衡：

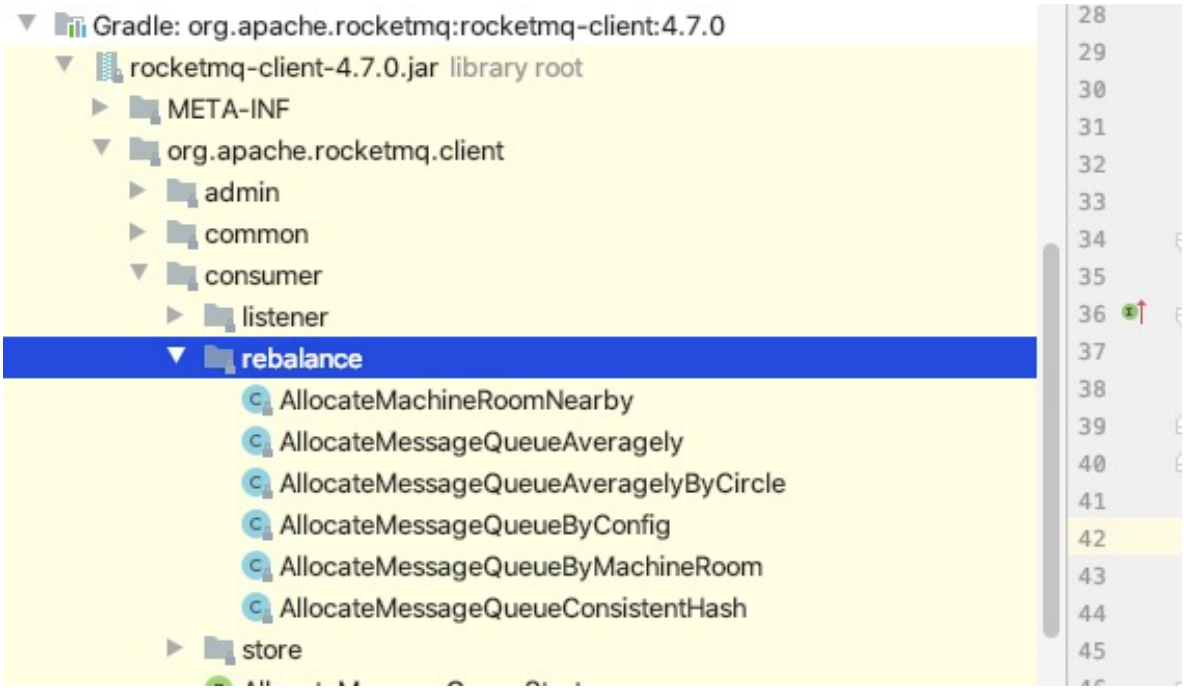
上面提到可以开启多个consumer，多个consumer间的负载均衡如何实现?

实现负载均衡前 必须的全局信息：
一个consumerGroup 有多少consumer， 分配算法

RocketMq中，负载均衡 / 消息分配时在Consumer端完成的， consumer从broker获取全局消息，自己负载均衡，只处理自己的部分

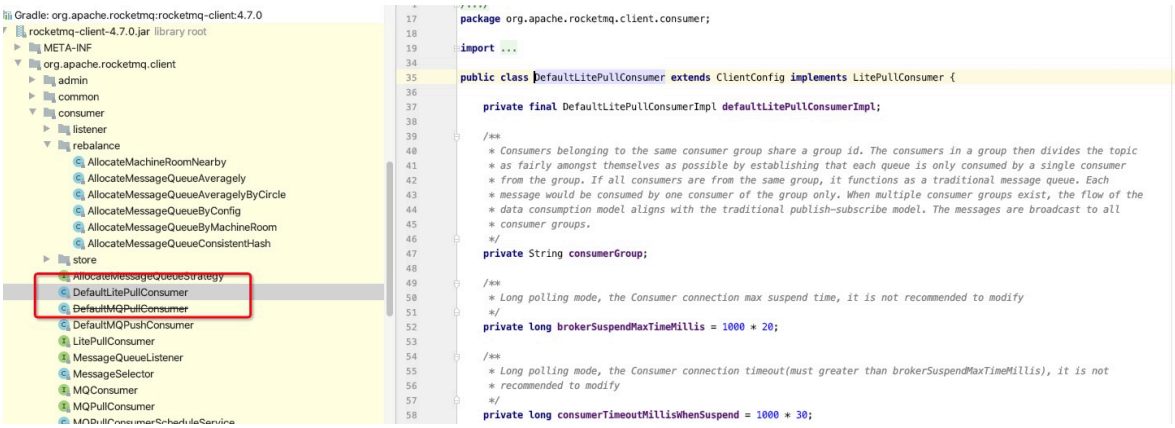
DefaultMqPushConsumer 的负载均衡策略：

不需要使用者，客户端自动处理，
每个DefaultMqPushConsumer启动后， 会触发一个 doRebalance
同一个ConsumerGroup 里加入新DefaultMqPushConsumer， 所有consumer都会 doRebalance



4.7 中 DefaultMQPullConsumer 不建议使用
建议 DefaultLitePullConsumer

DefaultMQPullConsumer
pull consumer 可以 看到所有 Message queue，从哪个读取 & offset 都由使用者决定，可以使用任意方式



提高 Producer 的发送速度
发送消息步骤： 客户端发送请求、服务器处理、服务器向客户端返回应答

可靠性不高的场景，例如日志， 可以采用 oneway 方式，只发请求，不用等待响应（数据写入 socket 缓冲区就返回）

多个 producer 一起发