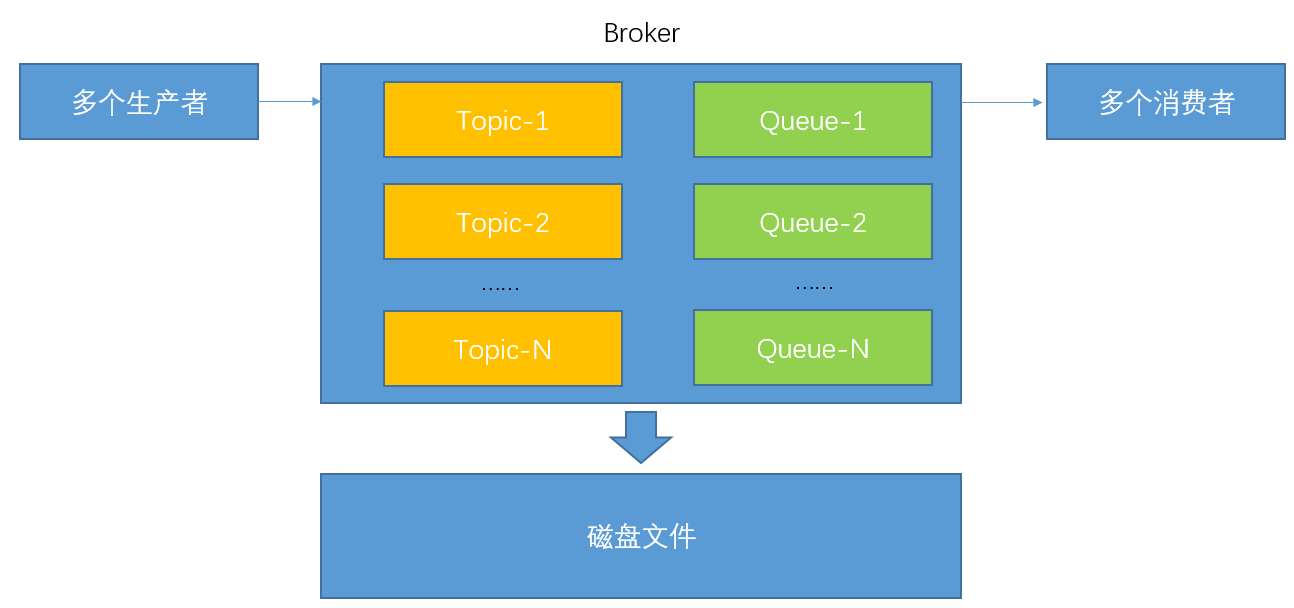
# 一总体架构



1多个生产者并发发送消息到broker

2broker通过消息头判断属于什么topic或queue，分别存入对应的topic或queue

3有单独的线程把topic或queue中的数据，定时刷入磁盘

4消费者直接从broker中读取topic或queue，无需通过磁盘文件。topic或queue中的消息任何变动（新增消息、ACK等等），都会异步同步到磁盘文件，磁盘文件类似redis的rdb镜像。

5当进程被kill掉，重新启动时，会从磁盘中恢复未被消费的消息到broker的内存中，磁盘文件只用作异常恢复消息，平时消息的交互都在内存中进行。

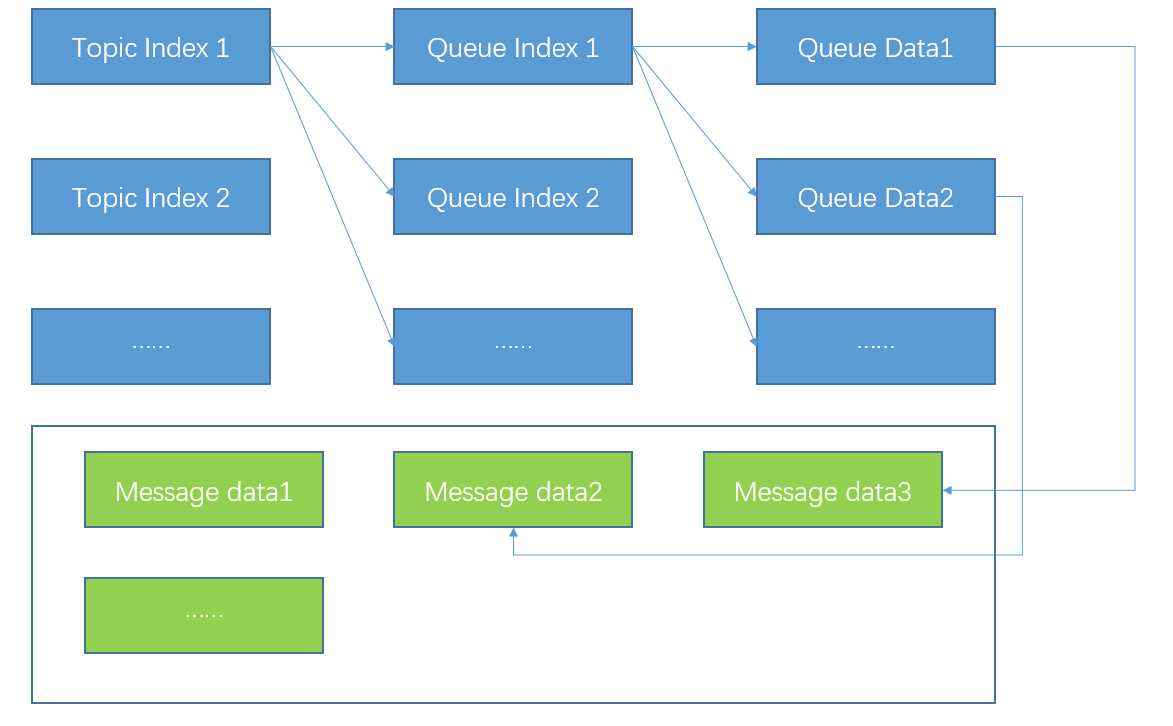
# 二broker存储的数据结构

## 背景

1）linux系统+启动进程约占用0.5G内存，剩余3.5G内存可用。

2）分别有约100个topic和100个queue

## 存储结构间的关系



**Topic Index**：Topic-Index文件存储。保存了Topic的名称、ID、包含的多个Queue的ID。1个Topic包含多个Queue ID。

**Queue Index**：Queue-Index文件存储。保存了Queue的名称、ID、队列在堆外内存的起点、整个队列的长度（终点，暂定每个队列最大长度10MB）、当前队列指针（消费到哪个消息了，每消费一条消息+1）。1个索引可以找到1个队列在堆外内存中的起点和终点。

**Queue Data**：Queue-Data文件保存，是保存在具体队列中的消息，可以通过它，在堆外内存中找到具体的消息。一个Queue中包含多个Queue Data。Queue Data的顺序就是消息的顺序。

**Message data**：接收到的具体的消息，序列化后保存在堆外内存中。可以通过Queue Data找到对应的Message data。Topic消息、Queue消息都保存在此。

## 消息存储结构



Queue Index：

* Queue名，队列名称；
* Queue ID，对了的ID，整个中间件中会有约100个Queue和Topic，Queue Index、Topic Index与Queue和Topic的数量相同。因此不会超过256
* 起点：整个队列在堆外内存中的起点地址。
* 长度：整个队列的长度。起点+长度=队列结束地址，长度/单个消息长度 = 消息数量。每新增一条消息，其长度都会增加。
* Position：记录现在队列中消费到第几条消息。每消耗一条+1。

Topic Index

* Topic名，Topic的名字
* Topic ID，Topic的ID
* Queue IDs，本Topic包含的所有队列的ID，是序列化后的字符串。可以通过这些ID找到对应的Queue。

Queue Data

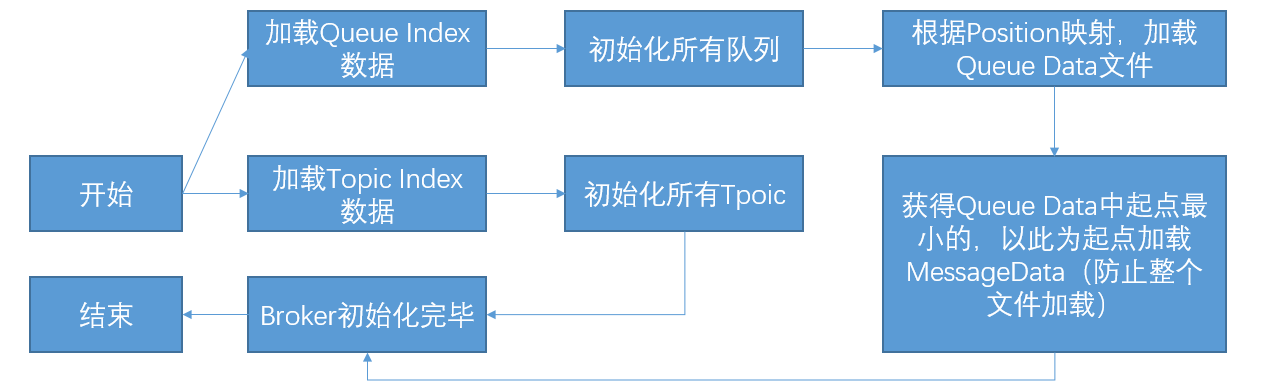
* Queue ID，所属队列的ID
* Page ID，页ID。消息序列化后保存在文件中，每个文件称为一页。由于消费掉的消息还会存在文件中，而文件是通过内存文件映射读取的，随着文件的增大，内存的占用也增大，并且是消费掉的消息占用了空间。当文件大小达到阀值后，先把旧文件中的消息消费完，再新建一个文件（新页），重新进行内存文件映射，防止过大的文件占用过多的内存。这个Page ID就是用来确定，具体消息保存在哪一个文件中。
* 起点：某一条消息在堆外内存中的起点
* 长度：某一条消息的长度。起点+长度=消息结束地址

Message Data

* 消息头：序列化后的消息头数据，最大128字节，即16K
* 消息属性：序列化后的消息属性，最大128字节，即16K
* 消息体：序列化后的具体消息，长度不限，由于有的消息大，有的消息小。当长度限定很大，但保存消息很小的时候，会导致空间的浪费。因此做成不限。通过Queue Data的起点+长度，可以知道消息的终点地址。

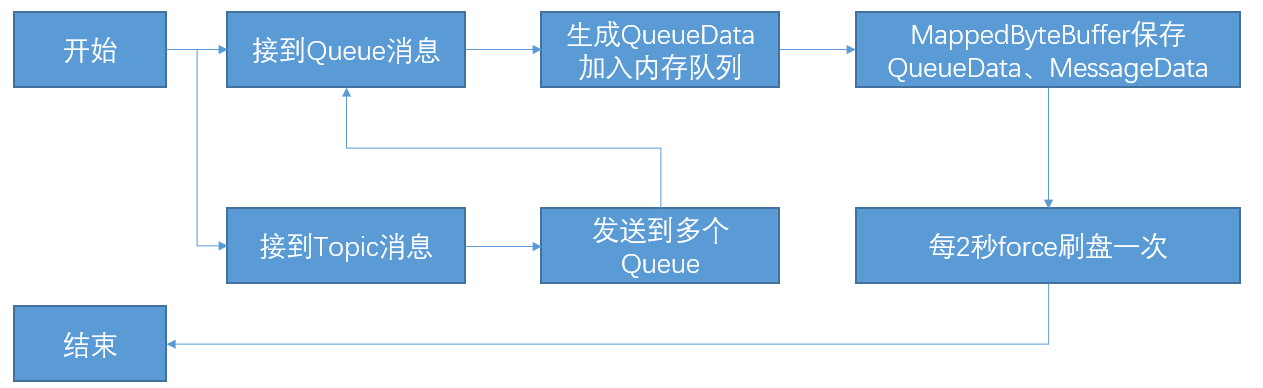
# 工作流程

## 启动/宕机重启时流程



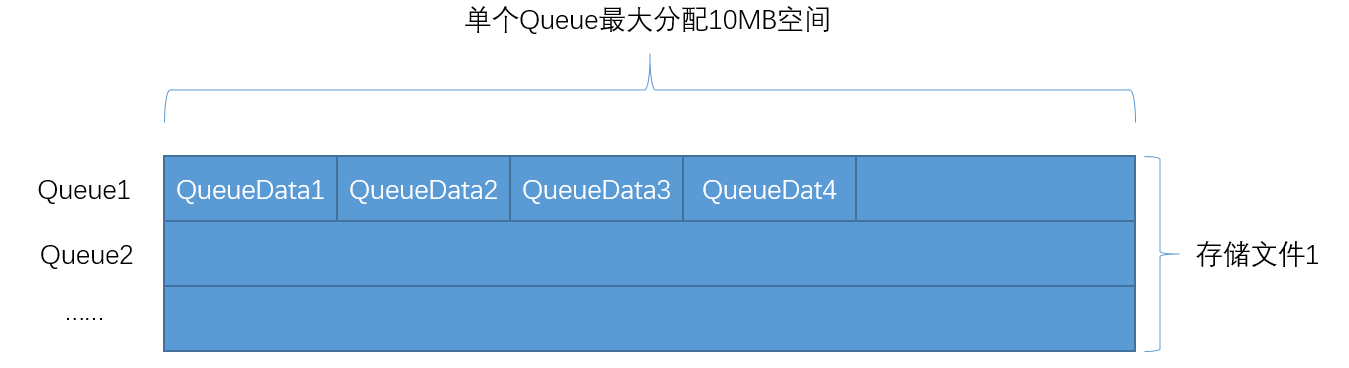
* 进程启动后，Queue和Topic的Index可以并行加载，并行初始化。
* Queue初始化完成后，初始化每个队列中的数据，根据Position得知消费到哪一条消息，以此为起点进行文件映射，防止Queue Data全文件加载。
* Queue Data加载完成后，得到Queue Data中起始地址最小的一个，并以这个地址为起点加载Message Data，防止Message Data全文件加载导致的内存溢出。因为Message Data的文件是以时间为顺序追加消息进去的，因此以最小的地址开始，肯定能包含所有未消费的Queue Data。

## 运行时流程



QueueData和MessageData都是内存文件映射的，即使消费掉的消息也保存在文件和内存中。

因此，定时判断channel的size，暂定QueueData中某个Queue已经写满10MB，则在QueueIndex中为本Queue新增一个同名索引，但保存的文件（Page ID）不同，新消息保存到新的页文件，客户端消费则先把老文件中的消息消费完。待某个页中所有的队列的Position都等于10MB时，说明该页已经全被消费完，释放本页的内存映射。QueueData的文件存储如下图所示：



MessageData暂定超过2G重新生成新的页文件，重新映射，消费时老文件继续保持映射，等到老文件中最后一个消息被消费，则销毁旧映射。（所有队列中，当前Position指向的Queue Data的PageID都不是本页的时候，说明已经没有引用了，可以GC掉）

由于QueueIndex和TopicIndex中的数据量不大，暂时不做内存映射释放的处理。

为什么不删除或覆盖就数据？

删除会出现文件空洞，肯定不行。

循环覆盖，在生产者>消费者时，队列被挤爆，只有动态生成新文件的方式可以无限扩大队列。另外，MQ有时候需要消息重放，并不是ACK掉的消息就可以扔掉的。