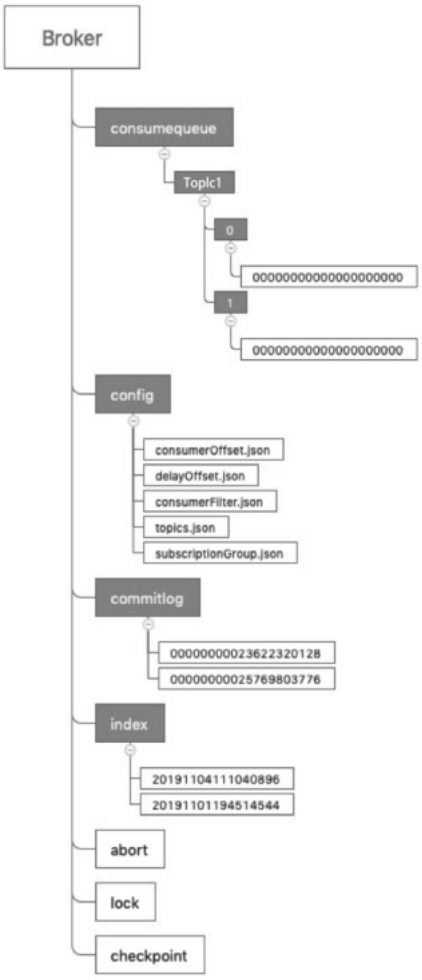
RocketMQ消息存储与检索原理

Broker存储目录结构

- Commitlog：这是一个目录，其中包含具体的commitlog文件。文件名长度为20个字符，文件名由该文件保存消息的最大物理offset值在高位补0组成。每个文件大小一般是1GB，可以通过mapedFileSizeCommitLog进行配置。
- consumequeue：这是一个目录，包含该 Broker 上所有的 Topic 对应的消费队列文件信息。消费队列文件的格式为“./consumequeue/Topic名字/queue id/具体消费队列文件”。每个消费队列其实是commitlog的一个索引，提供给消费者做拉取消息、更新位点使用。
- Index：这是一个目录，全部的文件都是按照消息key创建的Hash索引。文件名是用创建时的时间戳命名的。
- Config：这是一个目录，保存了当前Broker中全部的Topic、订阅关系和消费进度。这些数据Broker会定时从内存持久化到磁盘，以便宕机后恢复。
- abort：Broker 是否异常关闭的标志。正常关闭时该文件会被删除，异常关闭时则不会。当Broker重新启动时，根据是否异常宕机决定是否需要重新构建Index索引等操作。
- checkpoint：Broker最近一次正常运行时的状态，比如最后一次正常刷盘的时间、最后一次正确索引的时间等。



Tips:如果没有在配置文件中显式指定存储位置，这些文件默认存放在Linux的"/root/store"目录下

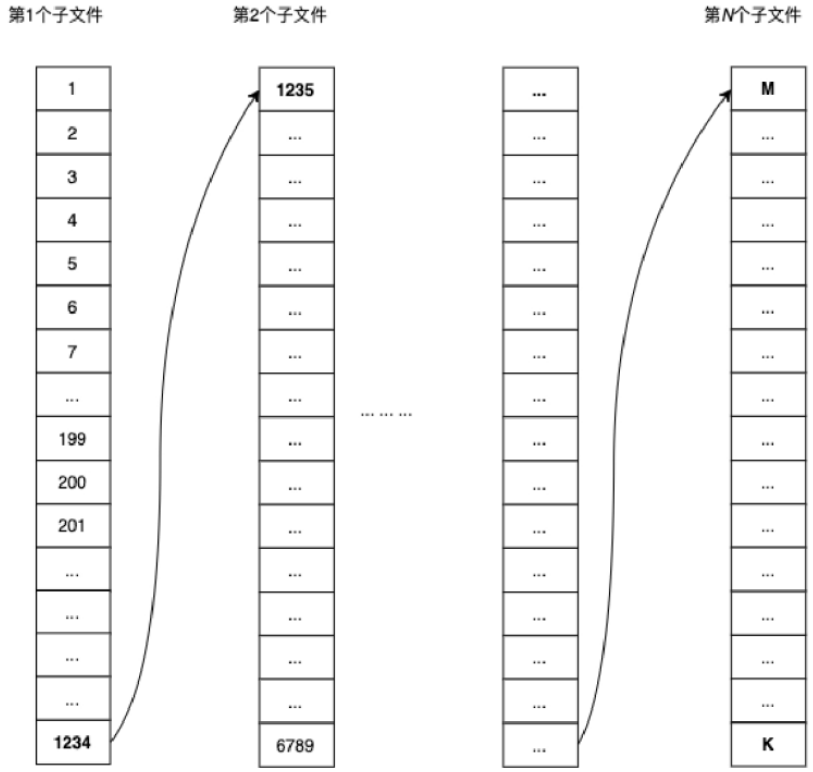
```
[root@localhost store]# pwd
/root/store
[root@localhost store]# ll
总用量 12
-rw-r--r--.  1 root root    0 2月  22 08:51 abort
-rw-r--r--.  1 root root 4096 2月  25 14:21 checkpoint
drwxr-xr-x.  2 root root   34 1月   1 19:34 commitlog
drwxr-xr-x.  2 root root  280 2月  25 14:22 config
drwxr-xr-x. 16 root root 4096 2月  22 11:50 consumequeue
drwxr-xr-x.  2 root root   31 2月  25 11:13 index
-rw-r--r--.  1 root root    4 2月  22 08:51 lock
[root@localhost store]#
```

RocketMQ 消息的构成

写入顺序	字　　段	说　　明	数据类型	字 节 数
1	MsgLen	消息总长度	Int	4
2	MagicCode	魔法数	Int	4
3	BodyCRC	消息内容 CRC	Int	4
4	QueueId	消息所在分区 id	Int	4
5	QueueOffset	消息所在分区的位置	Long	8
6	PhysicalOffset	消息所在 Commitlog 文件的位置	Long	8
7	SysFlag	系统标志	Int	4
8	BornTimestamp	发送消息时间戳	Long	8
9	BornHost	发送消息主机	Long	8
10	StoreTimestamp	存储消息时间戳	Long	8
11	StoreHost	存储消息主机	Long	8
12	ReconsumeTimes	重试消息重试第几次	Int	4
13	PreparedTransationOffset	事务消息位点	Long	8
14	BodyLength	消息体内容长度	Int	4
15	Body	消息体	byte[]	数组长度
16	TopicLength	Topic 长度	byte	1
17	Topic	Topic 名字	byte[]	数组长度
18	PropertiesLength	扩展信息长度	short	2
19	Properties	扩展信息	byte[]	数组长度

CommitLog的作用

CommitLog目录下有多个CommitLog文件。其实CommitLog只有一个文件，为了方便保存和读写被切分为多个子文件，所有的子文件通过其保存的第一个和最后一个消息的物理位点进行连接。



Broker按照时间和物理的offset顺序写CommitLog文件，每次写的时候需要加锁，每个 CommitLog 子文件的大小默认是1GB（1024×1024×1024B），可以通过mapedFileSizeCommitLog进行配置。当一个CommitLog写满后，创建一个新的CommitLog，继续上一个ComiitLog的Offset写操作，直到写满换成下一个文件。所有CommitLog子文件之间的Offset是连续的，所以最后一个CommitLog总是被写入的。

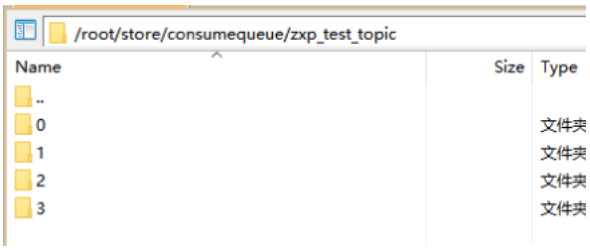
RocketMQ消息的索引机制

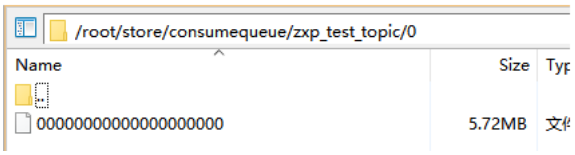
利用MessageID提取消息

因为MessageID就是用broker+offset生成的（这里MsgId指的是服务端的），所以很容易就找到对应的commitLog文件来读取消息。

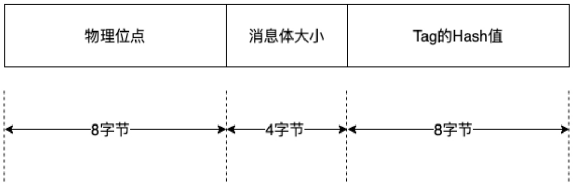
基于ConsumeQueue实现Tag查询

- ConsumeQueue存储结构
  - zxp_test_topic：主题名
    - 0|1|2|3：队列编号
      - ConsumeQueue文件，每个文件默认600W字节=5.72MB





- ComsumeQueue数据格式



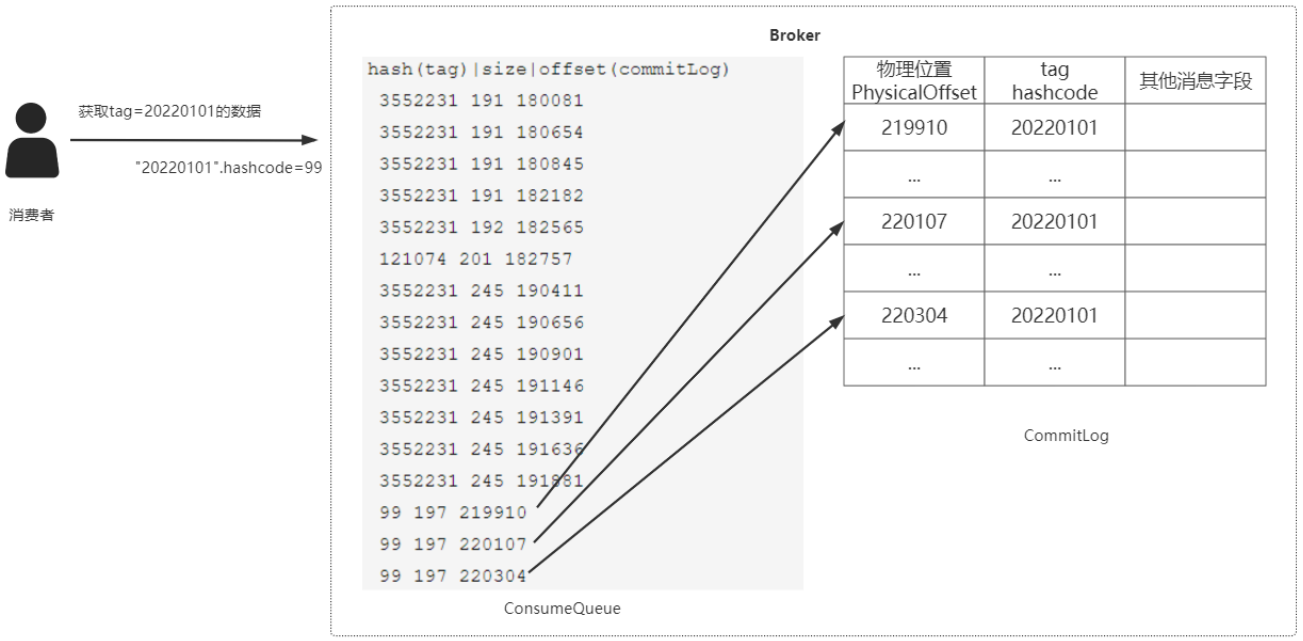
- 处理过程

第一步，消费者要获取tag=20220101的消息，首先通过执行"20220101".hashcode=99得到Hash值；

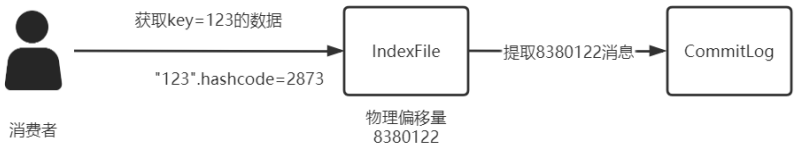
第二步，在ConsumeQueue文件中查找hash(tag)=99的offset数据；

第三步，根据物理位置Offset到对应的CommitLog文件中提取消息，因为可能会出现Hash碰撞，所以再次对这些命中数据以字符串匹配方式筛选"20220101"的消息；

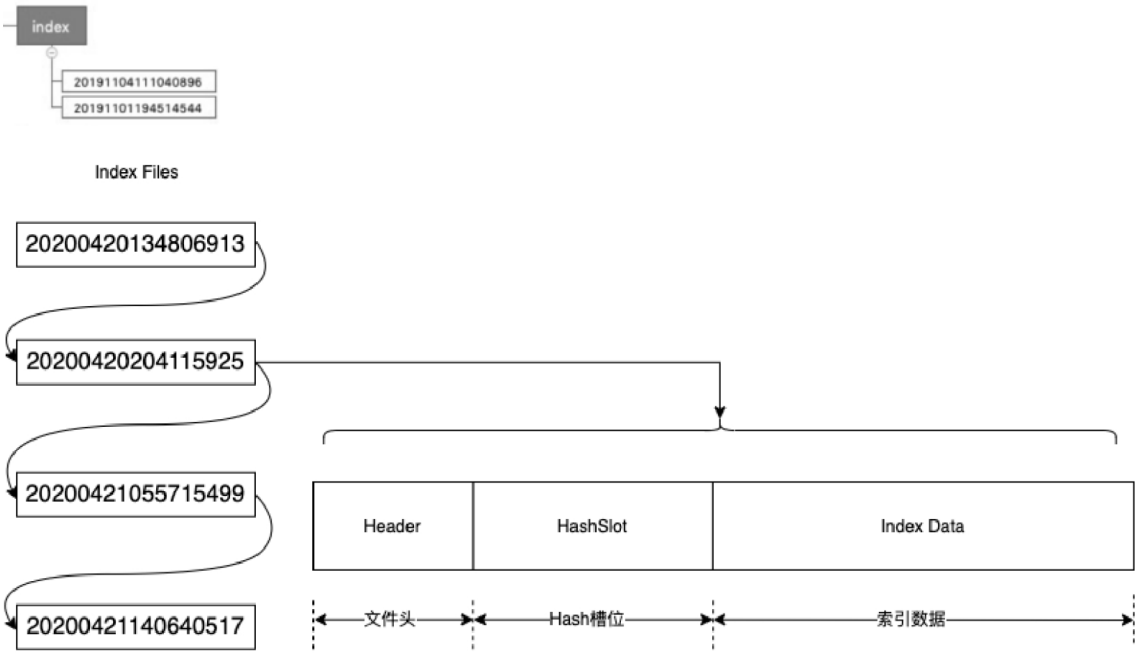
第四步，将“20220101”消息提取，封装为Message对象返回。



基于IndexFile实现Key查询

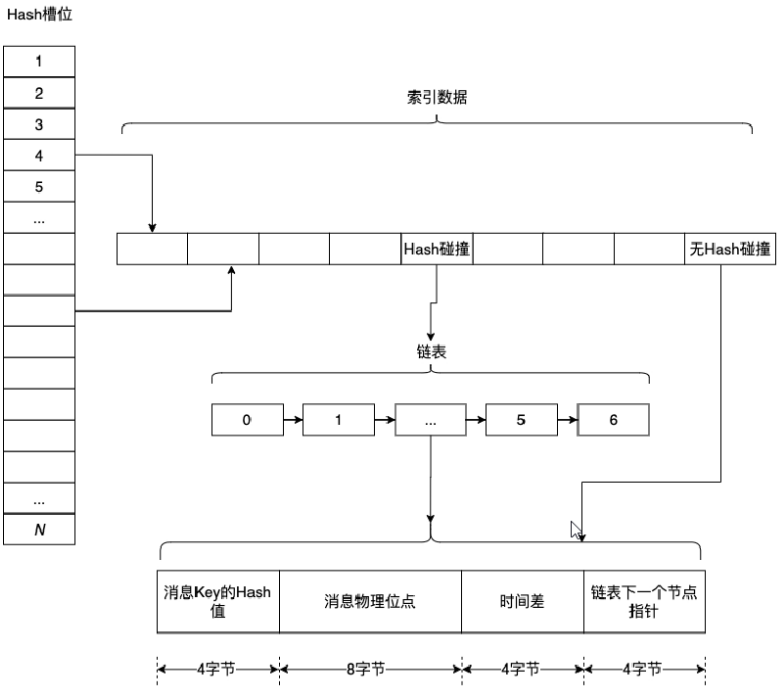


每个Index File文件包含文件头、Hash槽位、索引数据。每个文件的Hash槽位个数、索引数据个数都是固定的。Hash 槽位可以通过Broker 启动参数 maxHashSlotNum进行配置，默认值为500万；索引数据可以通过Broker启动参数maxIndexNum进行配置，默认值为500万×4=2000万，一个Index File约为400MB。



每个Index File文件包含文件头、Hash槽位、索引数据。每个文件的Hash槽位个数、索引数据个数都是固定的。

基于Slot+链表的查询过程



Index File的索引设计在一定程度上参考了Java中的HashMap设计，只是当Index File遇到Hash碰撞时只会用链表，而Java 8中在一定情况下链表会转化为红黑树。

这里具体实现相对ComsumeQueue比较复杂，有兴趣的童鞋可以阅读下面详细学习：

https://www.cnblogs.com/xjwhaha/p/15772846.html

	异步实时刷盘	异步定时刷盘	同步刷盘
数据一致性	中	低	高
数据可靠性	低	低	高
数据可用性	中	低	高
系统吞吐量	高	高	低