第8章 RocketMQ消息中间件

广东财经大学信息学院
罗 东 俊 博士
ZSUJONE@126.COM

（内部资料，请勿外传）

目的和要求

- 了解为什么要使用消息中间件。
- 熟悉RocketMQ消息中间件的基本概念和工作原理。
- 掌握Spring Boot与RocketMQ的整合实现与应用。

主要内容

- 8.1 消息服务概述
- 8.2 RocketMQ基础
- 8.3 RocketMQ整合案例

8.2 RocketMQ基础

- 8.2.1 RocketMQ概述
- 8.2.2 RocketMQ单节点安装
- 8.2.3 RocketMQ工作原理
- 8.2.4 RocketMQ应用模式

8.2.3 RocketMQ工作原理

- 1.工作流程
- 2.相关原理

1.工作流程

- 1）启动NameServer，NameServer启动后开始监听端口，等待Broker、Producer、Consumer连接。
- 2）启动Broker时，Broker会与所有NameServer建立并保持长连接，然后每30秒向NameServer定时发送心跳包。
- 3）发送消息前，先通过RocketMQ控制台创建Topic，创建Topic时需要指定该Topic要存储在哪些Broker上，也可以在发送消息时自动创建Topic。

工作流程

- 4）Producer发送消息，启动时先跟NameServer集群中的一台建立长连接，并从NameServer中获取到Topic路由信息（该Topic路由表及Broker列表），然后以某种Queue选择算法从队列列表中选择一个Queue，并与Queue所在的Broker（Master）建立长连接，且定时向Master发送心跳，从而向Broker发消息。
  - ✧Producer在获取到路由信息后，会首先将路由信息缓存到本地，再每30秒从NameServer更新一次路由信息。

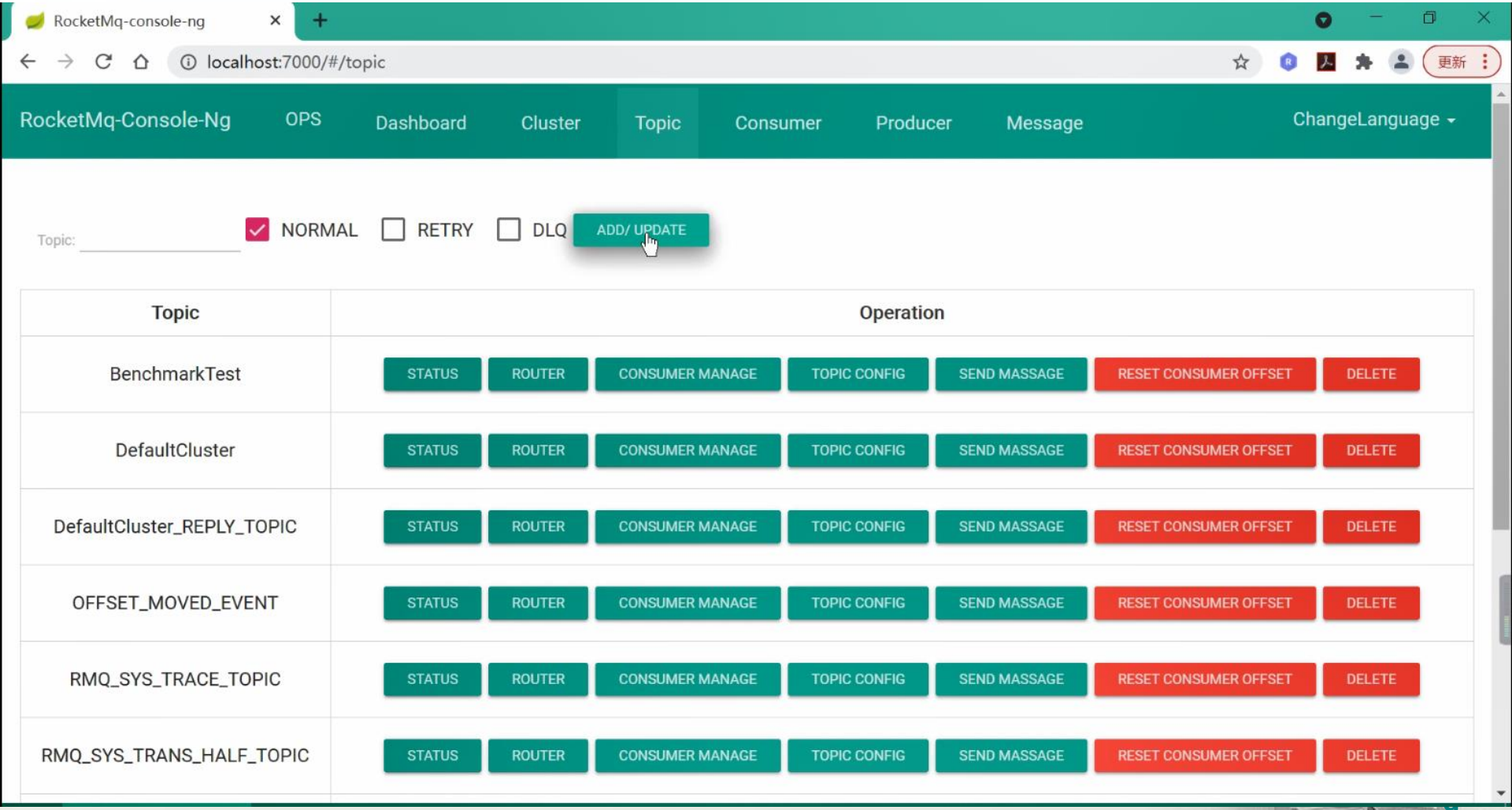
工作流程

- 5）Consumer跟Producer类似，与其中一台NameServer建立长连接后，从NameServer中获取到Topic路由信息，然后按某种Queue分配策略从路由信息中获取到其所要消费的Queue，并与Queue所在的Broker（Master和Slave）建立长连接，且定时向Master和Slave发送心跳，从而开始消费消息。
  - ✧Consumer在获取到路由信息后，同样也会将路由信息缓存到本地，再每30秒从NameServer更新一次路由信息。

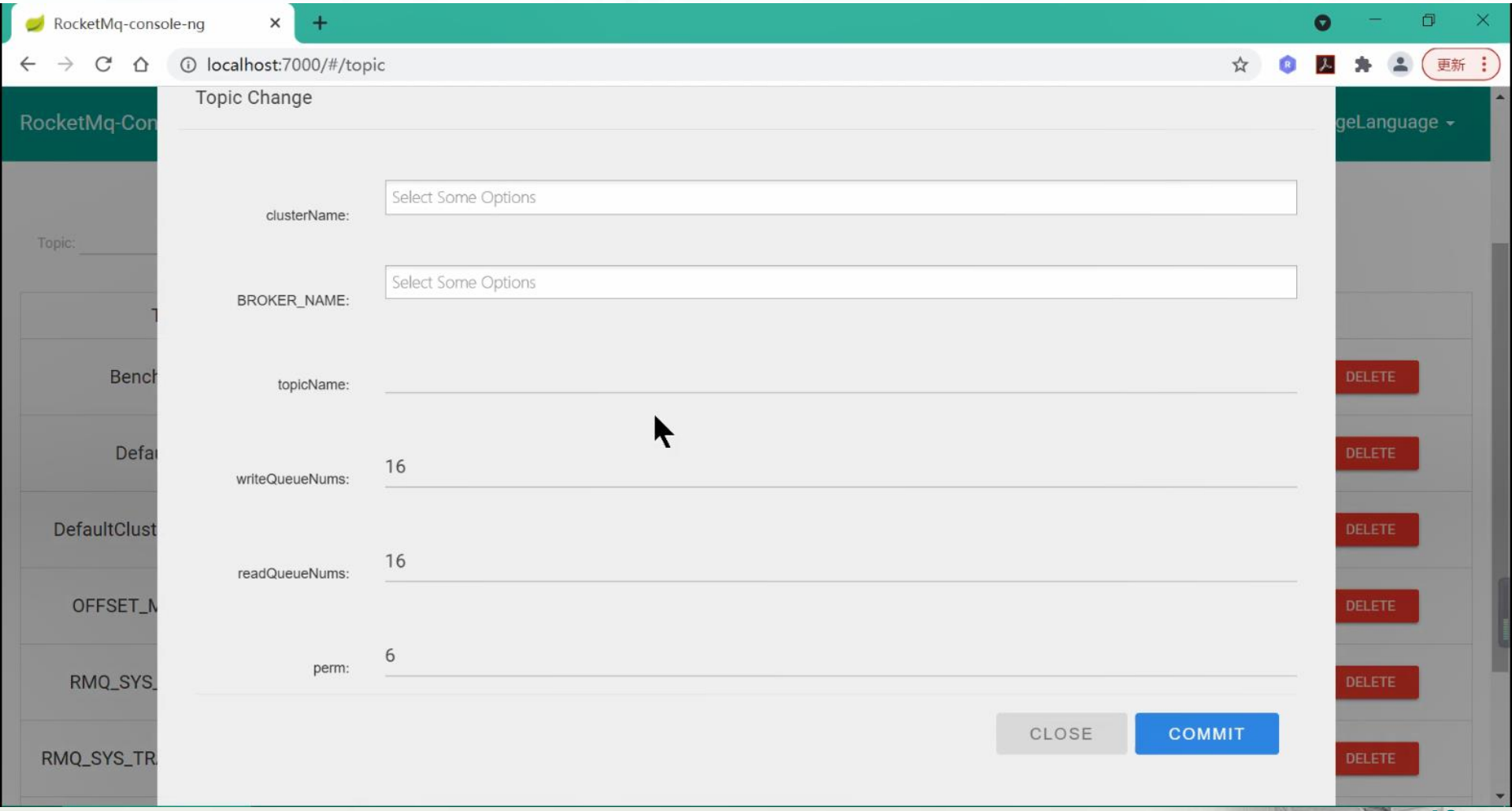
2.相关原理

- 1）Topic的创建
- 2）Queue的选择
- 3）Queue的分配
- 4）Queue的再均衡(Rebalance)
- 5）消息的存储
- 6）消息的消费

1）Topic的创建


RocketMq-console-ng
localhost:7000/#/topic
更新
RocketMq-Console-Ng
OPS
Dashboard
Cluster
Topic
Consumer
Producer
Message
ChangeLanguage
Topic:
NORMAL
RETRY
DLQ
ADD/ UPDATE
Topic
Operation
BenchmarkTest
DefaultCluster
DefaultCluster_REPLY_TOPIC
OFFSET_MOVED_EVENT
RMQ_SYS_TRACE_TOPIC
RMQ_SYS_TRANS_HALF_TOPIC
STATUS
ROUTER
CONSUMER MANAGE
TOPIC CONFIG
SEND MASSAGE
RESET CONSUMER OFFSET
DELETE

Topic的创建


RocketMq-console-ng
localhost:7000/#/topic
更新
Topic Change
clusterName:
Select Some Options
BROKER_NAME:
Select Some Options
topicName:
writeQueueNums:
16
readQueueNums:
16
perm:
6
CLOSE
COMMIT
RocketMq-Con
geLanguage
Topic:
Bench
Defa
DefaultClust
OFFSET_M
RMQ_SYS_
RMQ_SYS_TR
DELETE

Topic的创建

- 自动创建Topic时，系统会为每个Broker默认创建4个Queue（writeQueueNums=4，readQueueNums=4，perm=6）。
  - ✧物理上只会创建4个队列。
  - ✧读写队列数量设置机制，其设计目的是为了方便Topic的Queue的缩容，以致不会造成任何消息的丢失。
  - ✧其中perm用于设置对当前创建Topic的操作权限：2表示只写，4表示只读，6表示读写

Queue缩容例子

- 例如，原来创建的Topic中包含16个Queue，如何能够使其Queue缩容为8个，还不会丢失消息？
- 解决方案：可以动态修改写队列数量为8，读队列数量不变。
  - ✧此时，新的消息只能写入到前8个队列，而消费的却是16个队列中的数据。
  - ✧当发现后8个Queue中的消息消费完毕后，就可以再将读队列数量动态设置为8。整个缩容过程，没有丢失任何消息。

2）Queue的选择

- Queue选择算法，也称为消息投递算法，常见的有两种：
  - ①轮询（round-robin）算法
  - ②最小投递延迟算法

①轮询算法

- 这是默认选择算法。
  - ✧该算法可保证每个Queue中均匀获取到消息。
- 该算法存在一个问题：由于某些原因，在某些Broker上的Queue可能投递延迟较严重，从而导致Producer的缓存队列中出现较大的消息积压，影响消息的投递性能。

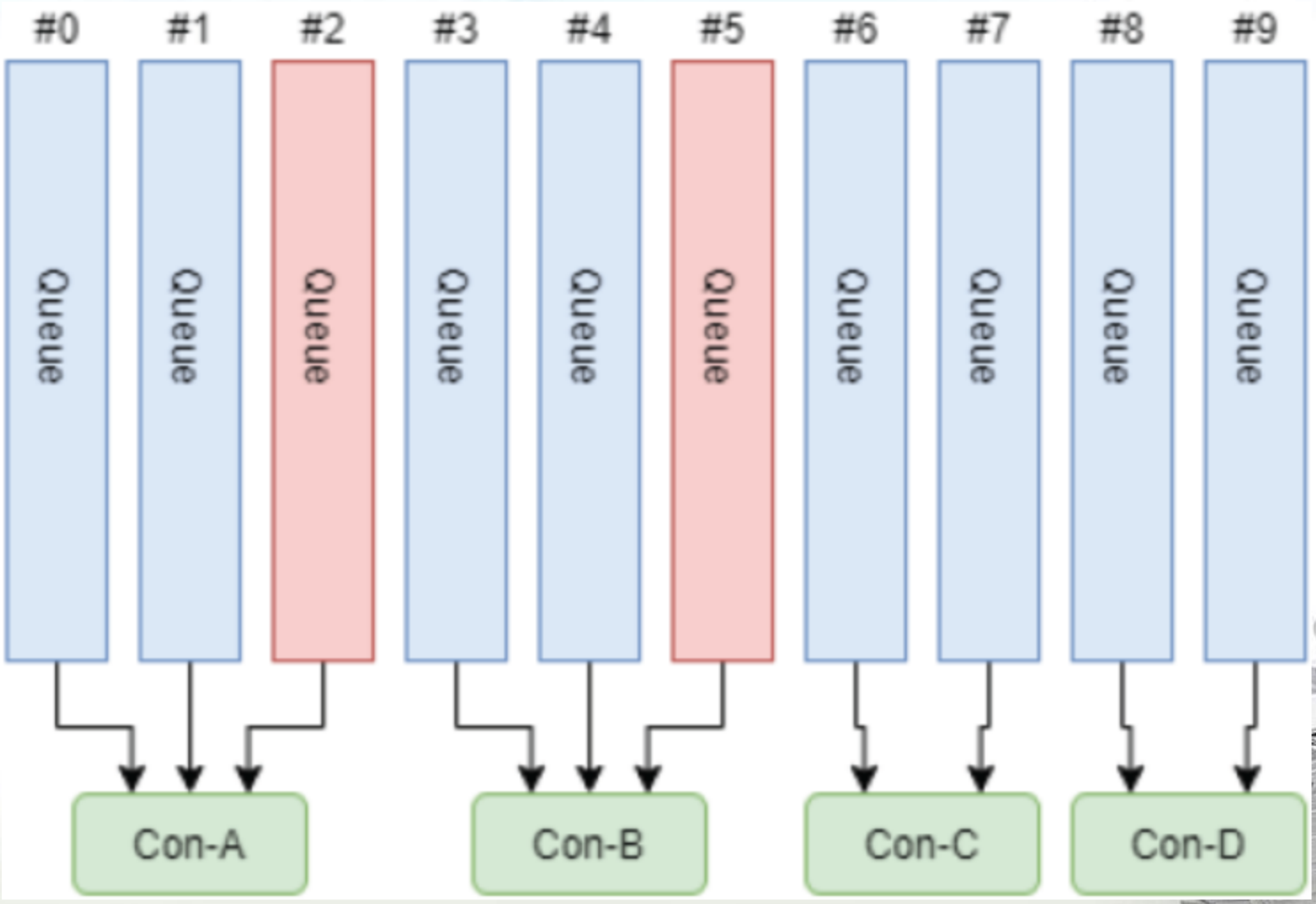
②最小投递延迟算法

- 该算法会统计每次消息投递的时间延迟，然后根据统计出的结果将消息投递到时间延迟最小的Queue。
  - ✧如果延迟相同，则采用轮询算法投递。
  - ✧该算法可以有效提升消息的投递性能。
- 该算法也存在一个问题：消息在Queue上的分配不均匀。
  - ✧投递延迟小的Queue其可能会存在大量的消息，而对该Queue的消费者压力会增大，降低消息的消费能力，可能会导致MQ中消息的堆积。

3）Queue的分配

- 一个Topic中的一个Queue只能由Consumer Group中的一个Consumer进行消费，而一个Consumer可以同时消费多个Queue中的消息。
- Queue的分配常见有四种策略，这些策略会通过创建Consumer时的构造方法传进去。
  - ✧①平均分配策略
  - ✧②环形平均策略
  - ✧③一致性hash策略
  - ✧④同机房策略

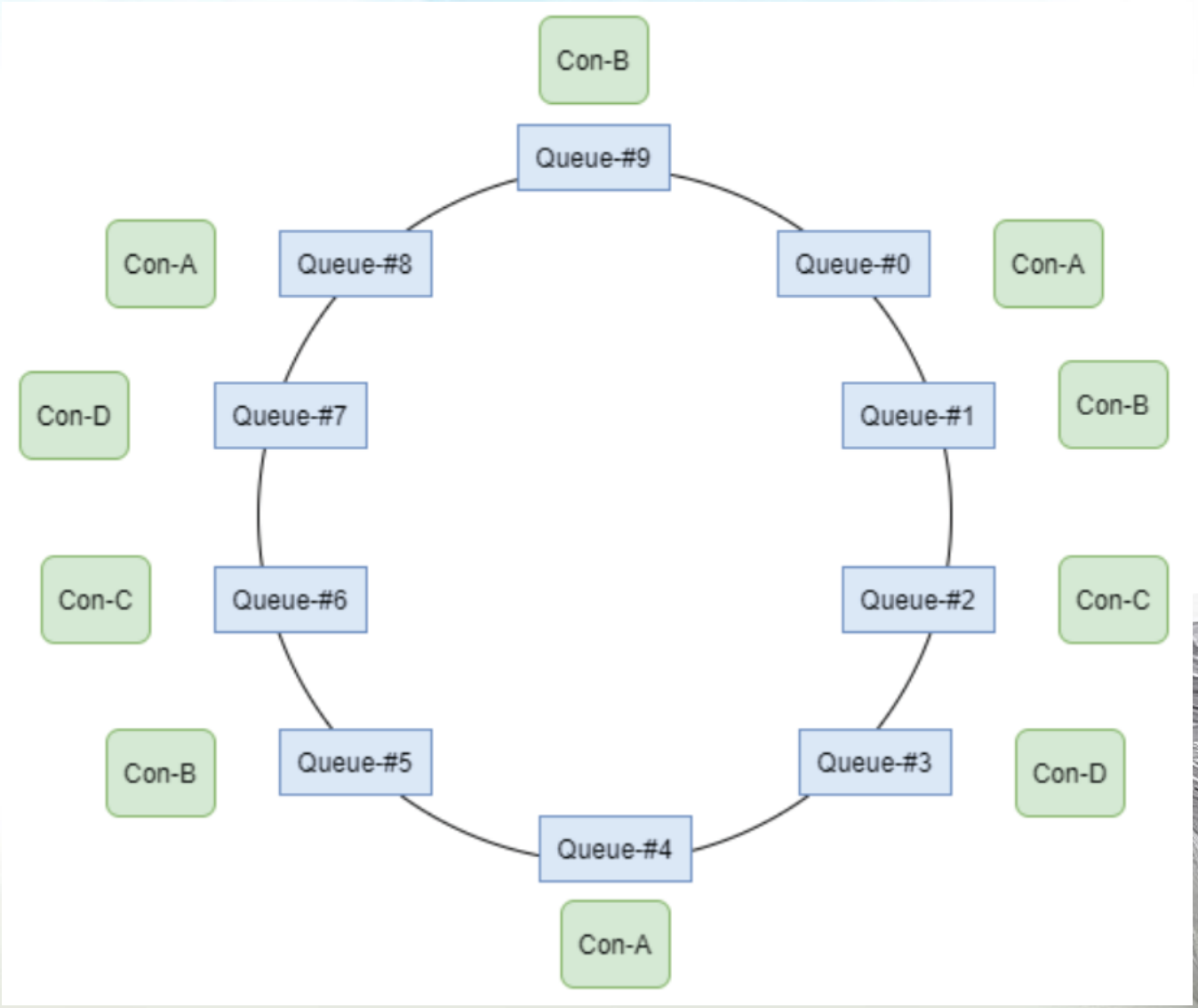
①平均分配策略



平均分配策略

- 该算法根据avg = QueueCount / ConsumerCount的计算结果进行分配。
  - ✧如果能够整除，则按顺序将avg个Queue逐个分配Consumer；
  - ✧如果不能整除，则将多余出的Queue按照Consumer顺序逐个分配。
- 该算法即先计算好每个Consumer应该分得几个Queue，然后再依次将这些数量的Queue逐个分配各Consumer。
- 该算法存在的问题：如果消费者组扩容或缩容会带来大量Consumer的Rebalance（再均衡）

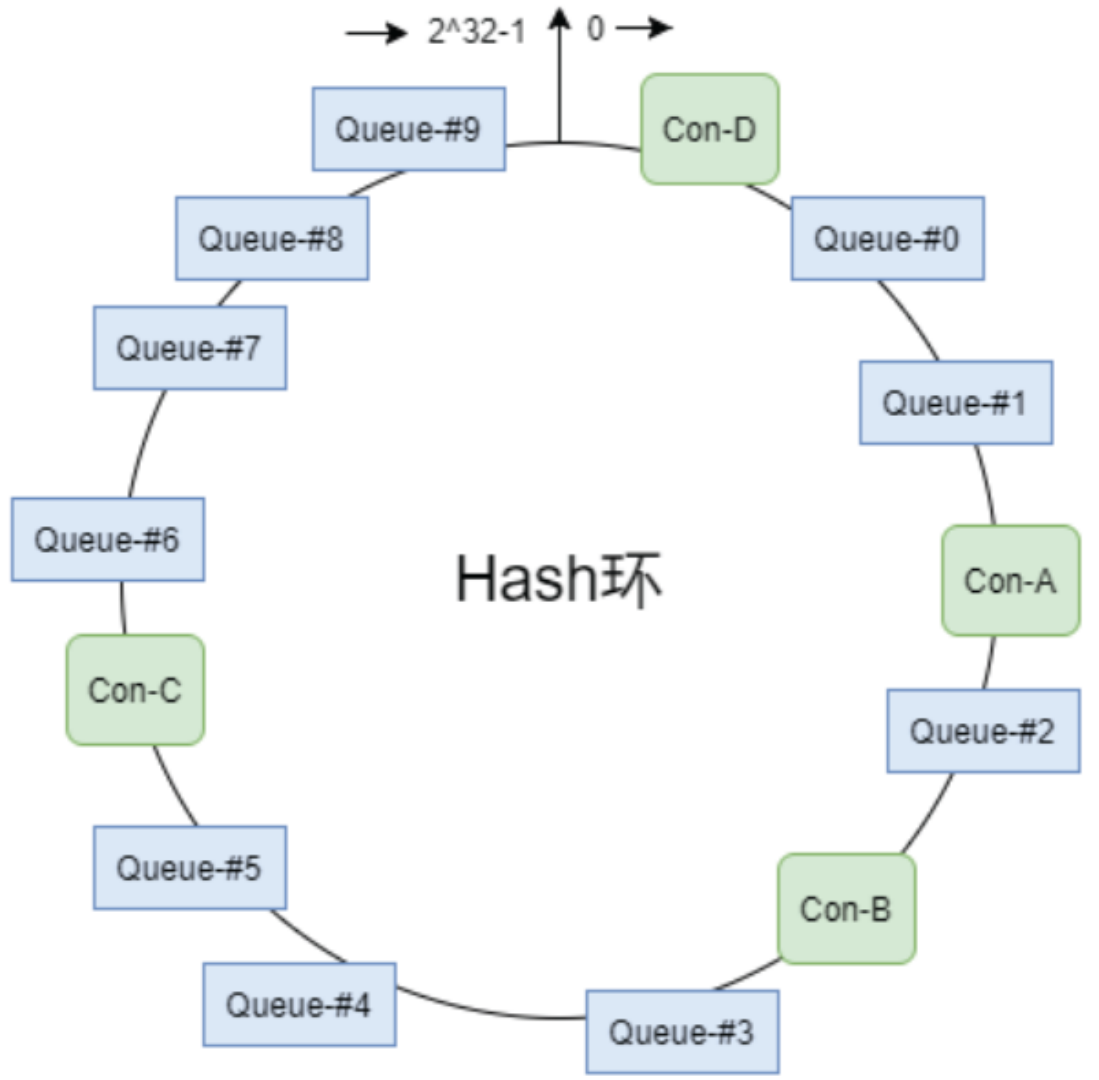
②环形平均策略


Con-B
Queue-#9
Con-A
Queue-#8
Queue-#0
Con-A
Con-D
Queue-#7
Queue-#1
Con-B
Con-C
Queue-#6
Queue-#2
Con-C
Con-B
Queue-#5
Queue-#3
Con-D
Queue-#4
Con-A

环形平均策略

- 环形平均算法是根据消费者的顺序，依次在由Queue队列组成的环形图中逐个分配。
- 该算法不用事先计算每个Consumer需要分配几个Queue，直接一个一个分即可。
- 该算法存在的问题：如果消费者组扩容或缩容也会带来大量的Rebalance（重分配）。

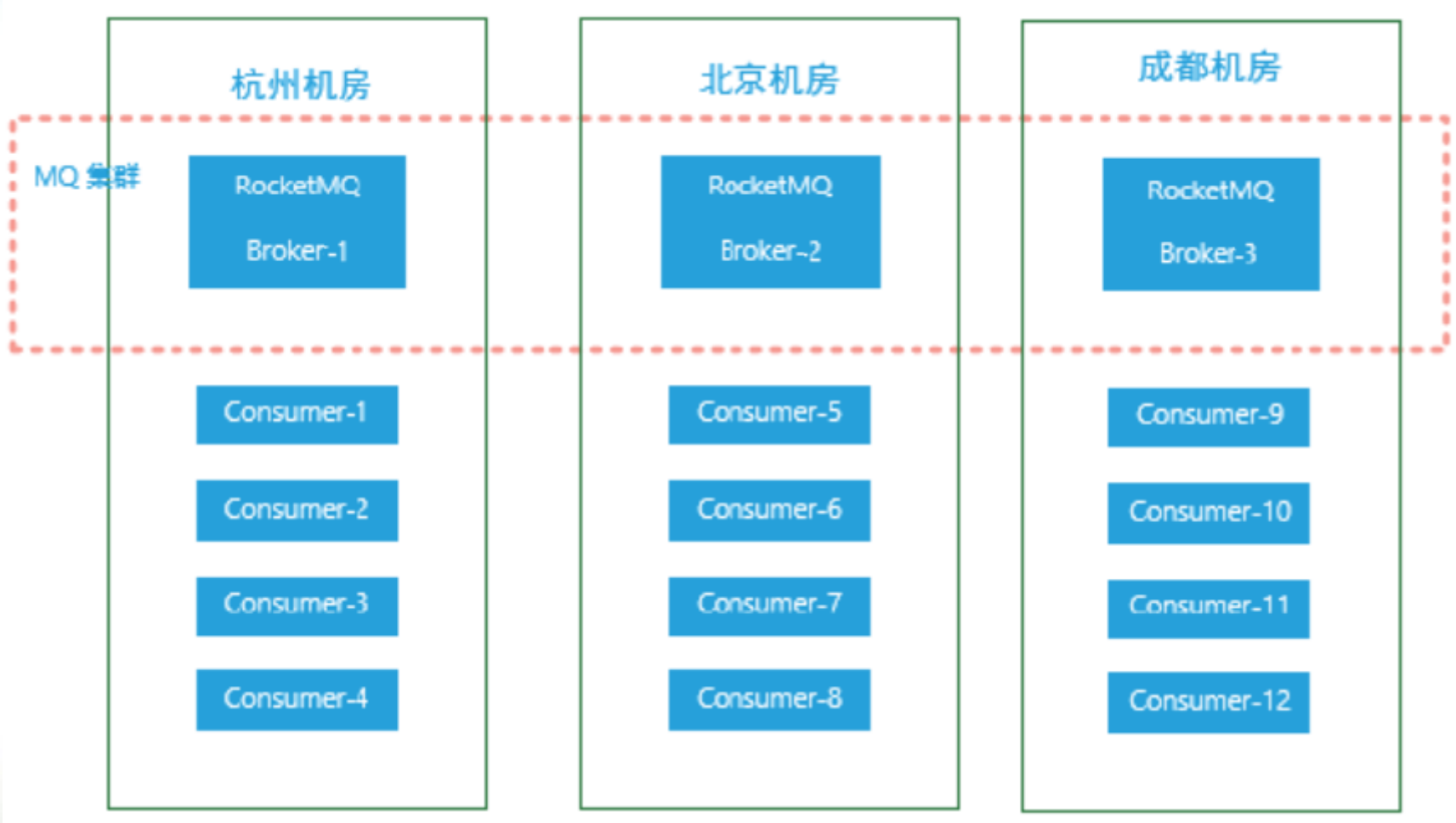
③一致性hash策略


2^32-1
0
Queue-#9
Con-D
Queue-#8
Queue-#0
Queue-#7
Queue-#1
Queue-#6
Hash环
Con-A
Con-C
Queue-#2
Queue-#5
Con-B
Queue-#4
Queue-#3

一致性hash策略

- 该算法会将consumer的hash值作为Node节点存放到hash环上，然后将queue的hash值也放到hash环上，通过顺时针方向，距离queue最近的那个consumer就是该queue要分配的consumer。
- 该算法存在的问题：分配不均。

④同机房策略


杭州机房
北京机房
成都机房
MQ 集群
RocketMQ
Broker-1
RocketMQ
Broker-2
RocketMQ
Broker-3
Consumer-1
Consumer-2
Consumer-3
Consumer-4
Consumer-5
Consumer-6
Consumer-7
Consumer-8
Consumer-9
Consumer-10
Consumer-11
Consumer-12

同机房策略

- 该算法会根据queue的部署机房位置和consumer的位置，过滤出当前consumer相同机房的queue。
- 然后按照平均分配策略或环形平均策略对同机房queue进行分配。
- 如果没有同机房queue，则按照平均分配策略或环形平均策略对所有queue进行分配。

4）Queue的再均衡(Rebalance)

- Rebalance即再均衡，指将一个Topic下的多个Queue在同一个Consumer Group中的多个Consumer间进行重新分配的过程。
- Rebalance产生的原因：
  - ✧消费者所订阅Topic的Queue数量发生变化；
  - ✧或消费者组中消费者的数量发生变化。

Rebalance过程

- Broker中有3种Manager的Map，据此，Broker一旦发现有引起Rebalance因素，就会立即向Consumer Group中的每个实例发出Rebalance通知。
- Consumer实例在接收到通知后会采用Queue分配算法自主进行Rebalance，从而获取到相应的Queue。

3种Manager中的Map

- ①TopicConfigManager
  - ✧key是topic名称，value是TopicConfig。
  - ✧TopicConfig中维护着该Topic中所有Queue的数据。
- ②ConsumerManager
  - ✧key是Consumser GroupId，value是ConsumerGroupInfo。
  - ✧ConsumerGroupInfo中维护着该Group中所有Consumer实例数据。
- ③ConsumerOffsetManager
  - ✧key为Topic与订阅该Topic的Group的组合,即topic@group，value是一个内层Map。
  - ✧内层Map的key为QueueId，内层Map的value为该Queue的消费进度offset。

Rebalance的危害

- Rebalance在提升消费能力的同时，也带来一些问题：
  - ✧①消费暂停
  - ✧②消费重复
  - ✧③消费突刺

①消费暂停

- 当发生Rebalance时，原Consumer需要暂停部分队列的消费，等到这些队列分配给新的Consumer后，这些暂停消费的队列才能继续被消费。

②消费重复

- Consumer在消费新分配给自己的队列时，必须接着之前Consumer提交的消费进度的offset继续消费。
- 然而默认情况下，消费进度的offset是异步提交的，这个异步性导致提交到Broker的offset与Consumer实际消费的消息的offset并不一致，这个不一致就可能导致重复消费消息。

③消费突刺

- 由于**Rebalance**可能导致重复消费，如果需要重复消费的消息过多，或者因**Rebalance**暂停时间过长从而导致积压了部分消息，那么有可能会导致在**Rebalance**结束之后瞬间需要消费很多消息。

5）消息的存储

- RocketMQ中的消息存储在Broker本地文件系统中，这些相关文件默认在BrokerOS当前用户主目录下的store目录里。

```
[root@rocketmqOS store]# ll
总用量 8
-rw-r--r-- 1 root root    0 7月   1 17:43 abort
-rw-r--r-- 1 root root 4096 7月   1 17:44 checkpoint
drwxr-xr-x 2 root root   34 6月  26 10:35 commitlog
drwxr-xr-x 2 root root  246 7月   1 17:44 config
drwxr-xr-x 3 root root   23 6月  26 10:28 consumequeue
drwxr-xr-x 2 root root   31 6月  26 10:28 index
-rw-r--r-- 1 root root    4 7月   1 17:43 lock
[root@rocketmqOS store]#
```

store目录下的目录和文件

- abort：该文件在Broker启动后会自动创建，正常关闭Broker，该文件会自动消失。
  - ✧ 若在没有启动Broker的情况下，发现这个文件是存在的，则说明之前Broker的关闭是非正常关闭。
- checkpoint：其中存储着commitlog、consumequeue、index文件的最后刷盘时间戳
- commitlog：其中存放着很多mappedFile文件，而消息就写在mappedFile文件中
- config：存放着Broker运行期间的一些配置数据
- consumequeue：其中存放着consumequeue文件，队列就存放在这个目录中
- index：其中存放着消息索引文件indexFile
- lock：运行期间使用到的全局资源锁

①commitlog文件

- commitlog文件，即commitlog目录中的mappedFile文件，文件名由20位十进制数构成，表示当前文件中的第一条消息的起始位移偏移量。
- mappedFile文件大小为小于等于1G，当第一个文件放满1G时，则会自动生成第二个文件继续存放消息。
- mappedFile文件内容由一个个的消息单元构成，第m+1个消息单元的commitlog offset偏移量为：
  - ✧L(m+1) = L(m) + MsgLen(m) (m >= 0)
- commitlog文件的过期时间为3天，3 天后会被自动删除。

消息单元的结构

- **mappedFile**文件内容中的消息单元包含近**20**余项消息相关属性：
  - ✧消息总长度**MsgLen**、消息的物理位置**physicalOffset**、消息体内容**Body**、消息体长度**BodyLength**、消息主题**Topic**、**Topic**长度**TopicLength**、消息生产者**BornHost**、消息发送时间戳**BornTimestamp**、消息所在的队列**QueueId**、消息在**Queue**中存储的偏移量**QueueOffset**。

mappedFile文件n中的消息单元

Commitlog Offset	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
n	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L1	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L2	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L3	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L4	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L5	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L6	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....
L7	MsgLen	PhysicalOffset	Body	BodyLength	BornHost	BornTimestamp	Topic	QueueId	QueueOffset	.....

②consumequeue文件

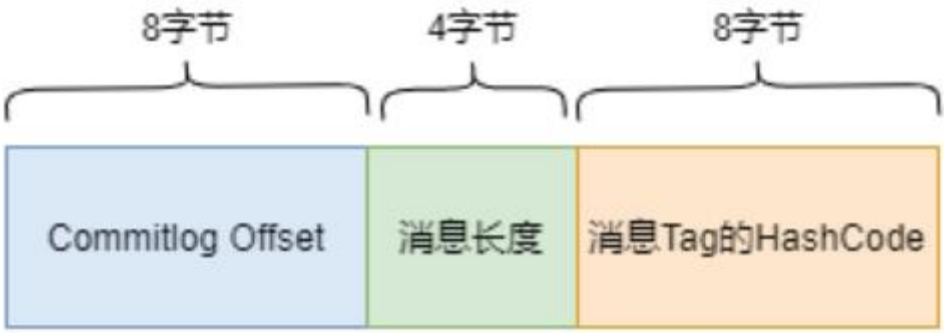
- 在consumequeue目录中MQ会为每个Topic创建一个目录，目录名为Topic名称。在该Topic目录下，会再为每个该Topic的Queue建立一个目录，目录名为queueId。该目录中存放着若干consumequeue文件。
  - ✧ 一个consumequeue文件中所有消息的Topic一定是相同的。但每条消息的Tag可能是不同的。
- consumequeue文件是commitlog的索引文件，文件名也由20位十进制数构成，表示当前文件的第一个索引条目的起始位移偏移量。
  - ✧ consumequeue文件内容由一个个的索引条目构成，每个索引条目20字节，一个consumequeue文件可以包含30w个索引条目，当第一个consumequeue文件放满时，则会自动生成第二个consumequeue文件继续存放索引条目。
  - ✧ 第m+1个索引条目的queue offset偏移量为：
    - q(m+1) = q(m) + 20 (m >= 0)

consumequeue文件

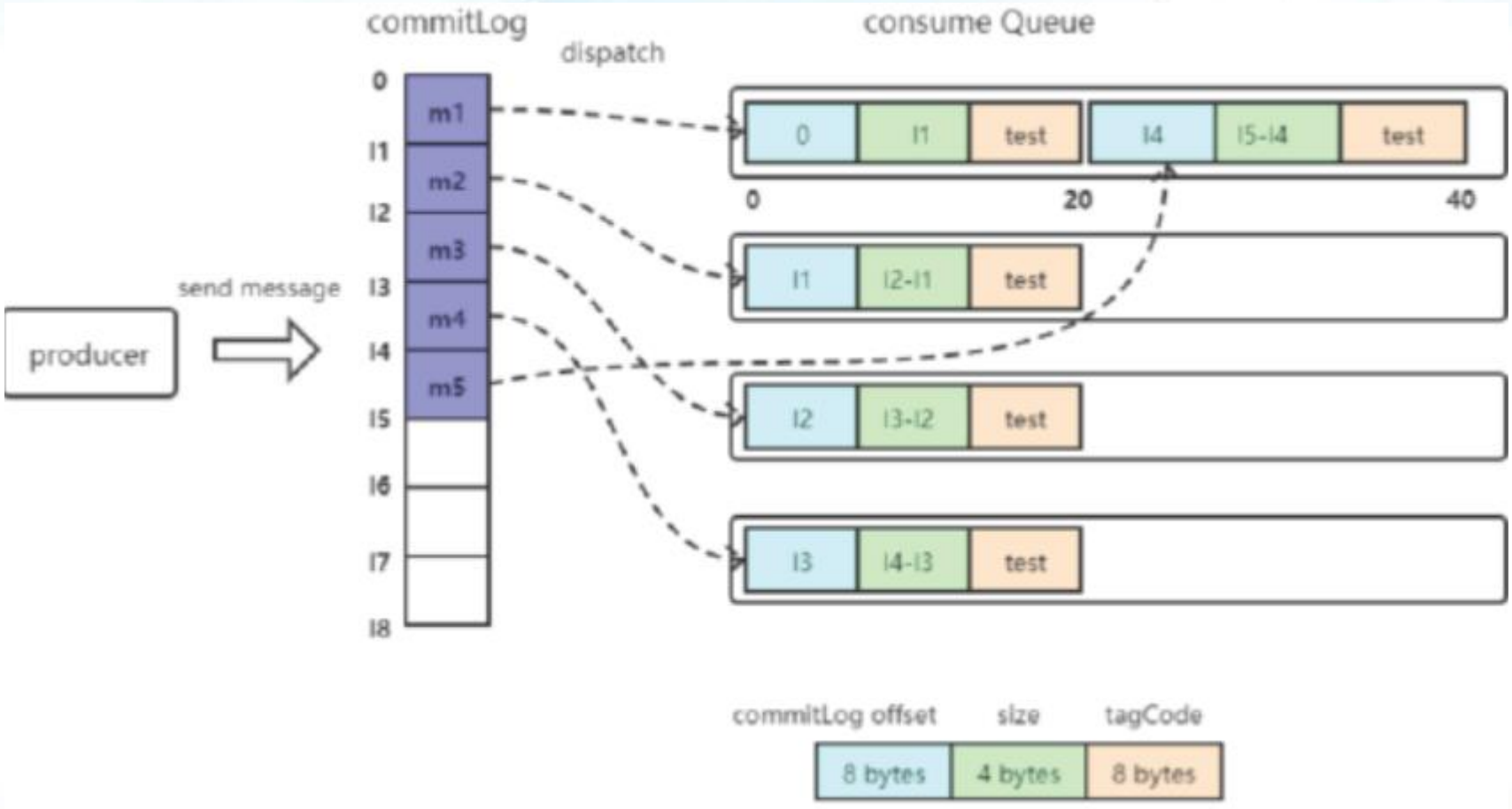
```
[root@mqOS ~]# ll store/consumequeue/
总用量 0
drwxr-xr-x 6 root root 42 1月   13 13:09 aaaxxxaaa
drwxr-xr-x 6 root root 42 1月   13 12:13 abcOOO
drwxr-xr-x 6 root root 42 1月   13 12:11 abcXxx
drwxr-xr-x 6 root root 42 1月   17 13:58 myTopic
drwxr-xr-x 6 root root 42 1月   13 12:03 persons
drwxr-xr-x 6 root root 42 1月   13 12:09 RMQ_SYS_TRACE_TOPIC
drwxr-xr-x 3 root root 15 1月   28 17:27 SCHEDULE_TOPIC_XXXX
drwxr-xr-x 6 root root 42 1月   17 13:57 someTopic
drwxr-xr-x 6 root root 42 1月   17 13:56 someTopicTest
drwxr-xr-x 6 root root 42 1月   29 10:23 Topic1
drwxr-xr-x 6 root root 42 1月   13 10:44 TopicTest
drwxr-xr-x 6 root root 42 1月   17 13:55 TopicTest1
[root@mqOS ~]#
[root@mqOS ~]# ll store/consumequeue/persons
总用量 0
drwxr-xr-x 2 root root 34 1月   13 12:03 0
drwxr-xr-x 2 root root 34 1月   13 12:03 1
drwxr-xr-x 2 root root 34 1月   13 12:03 2
drwxr-xr-x 2 root root 34 1月   13 12:03 3
[root@mqOS ~]#
[root@mqOS ~]# ll store/consumequeue/persons/0
总用量 4
-rw-r--r-- 1 root root 6000000 1月   13 12:05 00000000000000000000
[root@mqOS ~]#
```

consumequeue文件

索引条目的结构



消息写入过程


commitLog
dispatch
consume Queue
0
I1
I2
I3
I4
I5
I6
I7
I8
m1
m2
m3
m4
m5
send message
producer
0
I1
test
I4
I5-I4
test
0
20
40
I1
I2-I1
test
I2
I3-I2
test
I3
I4-I3
test
commitLog offset
size
tagCode
8 bytes
4 bytes
8 bytes

消息写入过程

- 一条消息进入到Broker后经历了以下几个过程才最终被持久化。
  - Broker根据queueId，获取到该消息对应索引条目要在consumequeue目录中的写入偏移量，即QueueOffset
  - 将queueId、queueOffset等数据，与消息一起封装为消息单元
  - 将消息单元写入到commitlog
  - 同时，形成消息索引条目
  - 将消息索引条目分发到相应的consumequeue

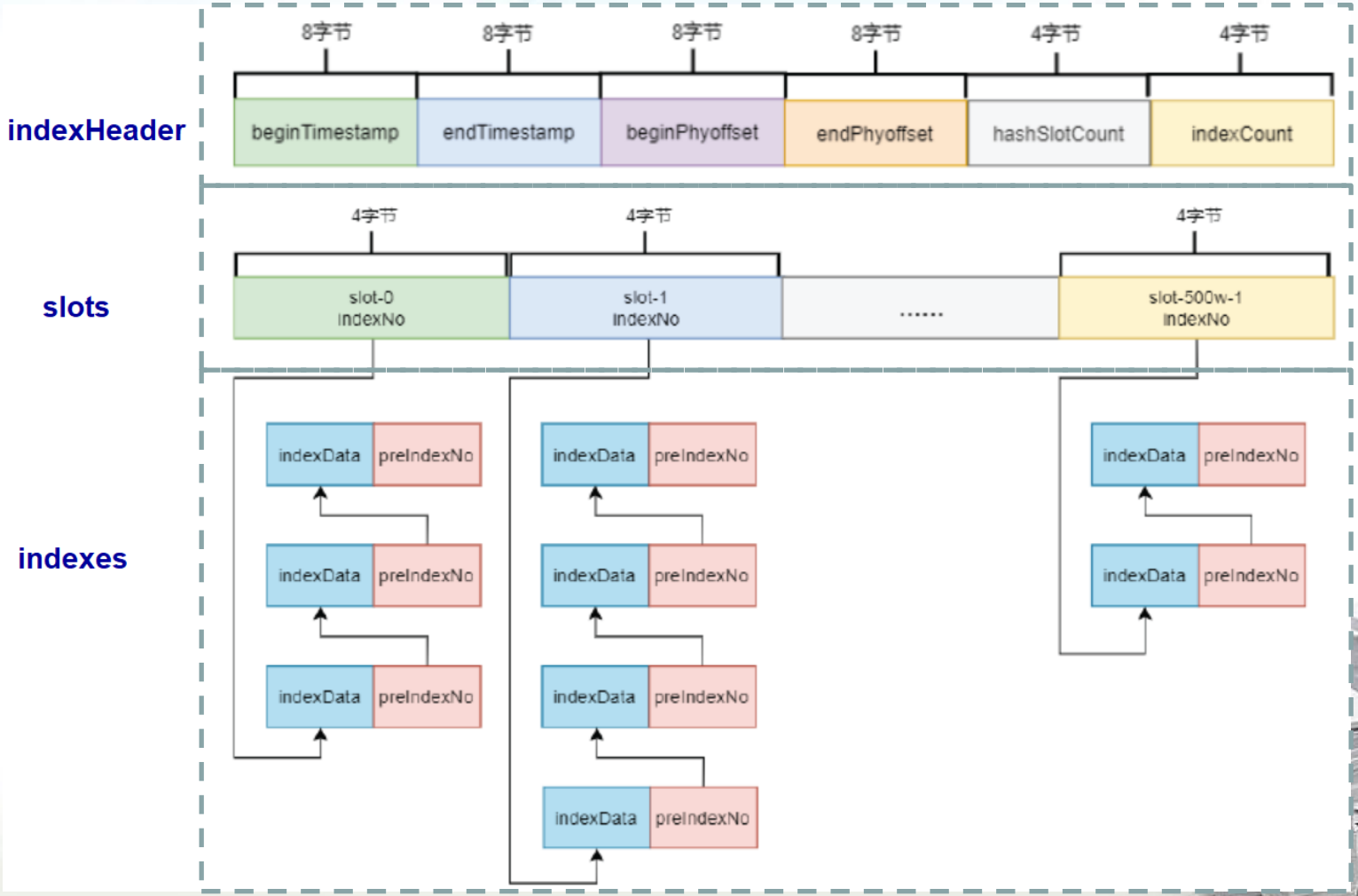
消息拉取过程

- 当Consumer来拉取消息时会经历以下几个步骤：
  - Consumer获取到其要消费消息所在Queue的消费进度offset，nextBeginOffset=消费进度offset+1即为其要消费的消息的offset
    - 消费进度offset即消费到了该Queue的第几条消息
  - Consumer向Broker发送拉取请求，其中包含其要拉取消息的Queue、nextBeginOffset及消息Tag。
  - Broker计算在该consumequeue中的queueOffset=nextBeginOffset * 20字节
  - 从该queueOffset处开始向后查找第一个指定Tag的索引条目
  - 解析该索引条目的前8个字节，即可定位到该消息在commitlog中的commitlog offset
  - 从对应commitlog offset中读取消息单元，并发送给Consumer

③indexFile文件

- indexFile文件是指index目录中的索引文件，它们以这个indexFile被创建时的时间戳命名，便于查询指定时间戳之前存储的最新消息，并且只有包含了业务key的消息被发送到Broker时才写入该索引文件。
- 每个indexFile文件由三部分构成：indexHeader，slots槽位，indexes索引条目。
  - ✧indexHeader固定40个字节。
  - ✧每个indexFile文件中包含500w个slot槽。
  - ✧每个索引条目默认20个字节，按消息到达Broker顺序存放，具有唯一的存放号indexNo，分别挂载在这些slot槽上。

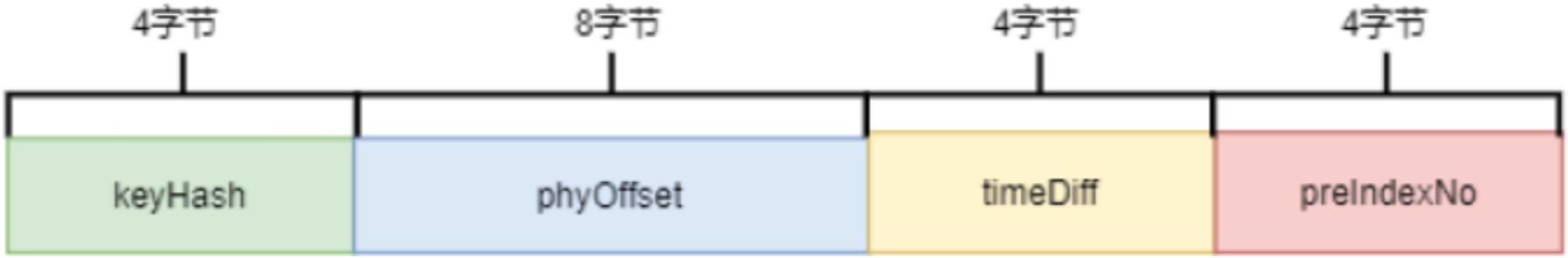
indexFile文件结构



indexHeader结构

- beginTimestamp：该indexFile中第一条消息的存储时间
- endTimestamp：该indexFile中最后一条消息存储时间
- beginPhyoffset：该indexFile中第一条消息在commitlog中的偏移量commitlog offset
- endPhyoffset：该indexFile中最后一条消息在commitlog中的偏移量commitlog offset
- hashSlotCount：挂载有索引条目的slot槽的数量
- indexCount：该indexFile中包含的所有索引条目个数

索引条目结构

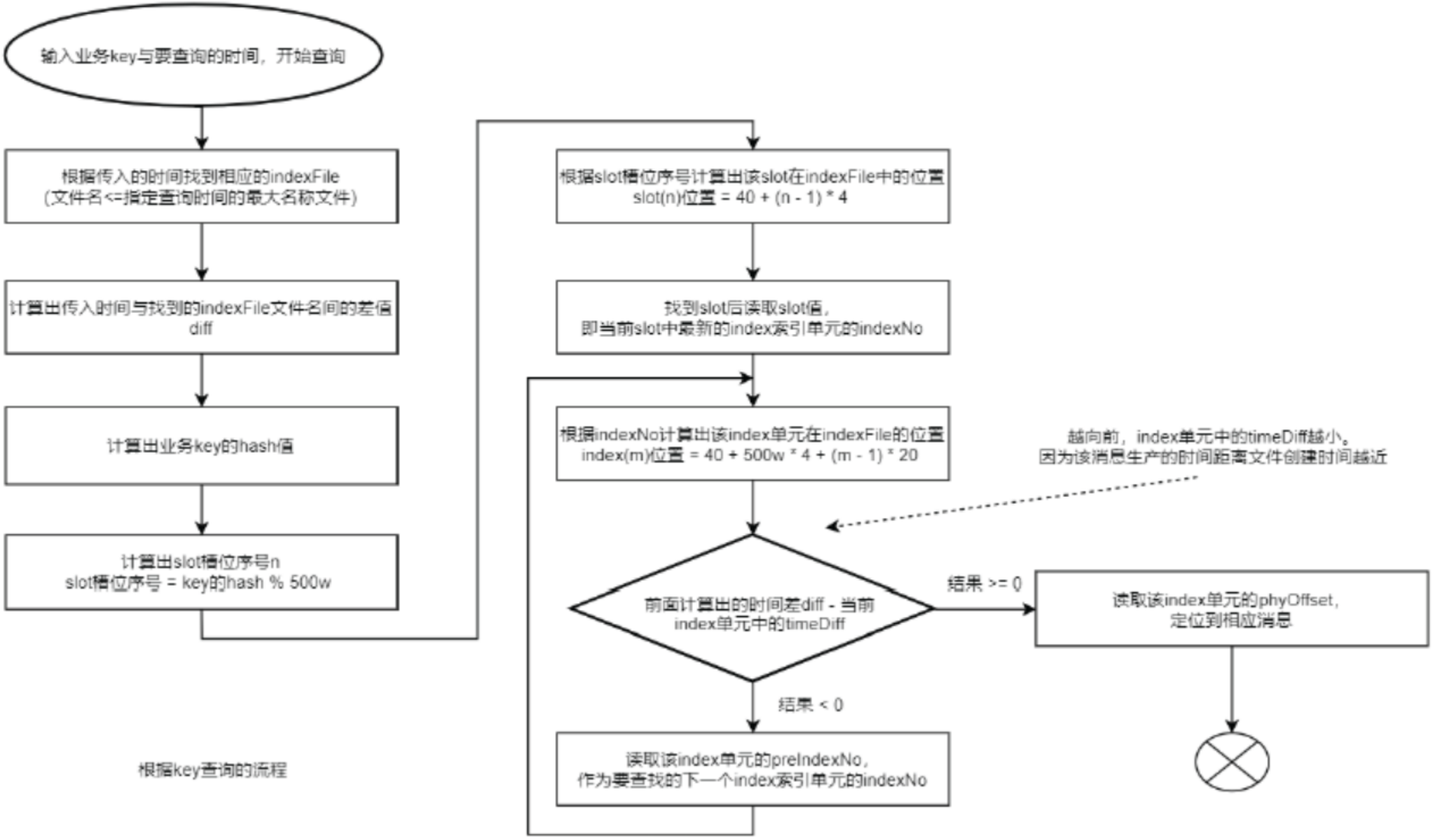


- keyHash：消息中指定的业务key的hash值， keyHash % 500w即为slot槽位
- phyOffset：当前key对应的消息在commitlog中的偏移量 commitlog offset
- timeDiff：当前key对应消息的存储时间与当前indexFile创建时间的时间差
- preIndexNo：当前slot下当前索引条目的前一个索引条目的indexNo

存储位置

- 槽位序号为n的slot在indexFile中的起始位置：
  - slot(n)位置 = 40 + (n - 1) * 4
- indexNo为m的索引条目在indexFile中的位置：
  - index(m)位置 = 40 + 500w * 4 + (m - 1) * 20

查询流程



根据key查询的流程

6）消息的消费

- 消息的消费包括以下几个方面：
  - ①消息获取模式
  - ②消息模式
  - ③订阅和消费原则
  - ④消费进度管理
  - ⑤消费幂等
  - ⑥消费延迟

①消息获取模式

- 拉取式（Pull）获取
  - ✧Consumer主动从Broker中拉取消息到本地缓冲队列中，主动权由Consumer控制。
  - ✧需要应用去实现对关联Queue的遍历，实时性差，即Broker中有了新的消息时消费者并不能及时发现并消费。
- 推送式（Push）获取
  - ✧该模式下Broker收到数据后会主动推送给Consumer。
  - ✧是典型的发布-订阅模式，即Consumer向其关联的Queue注册了监听器，一旦发现有新的消息到来就会触发回调的执行，回调方法是Consumer去Queue中拉取消息。
  - ✧封装了对关联Queue的遍历，实时性强。

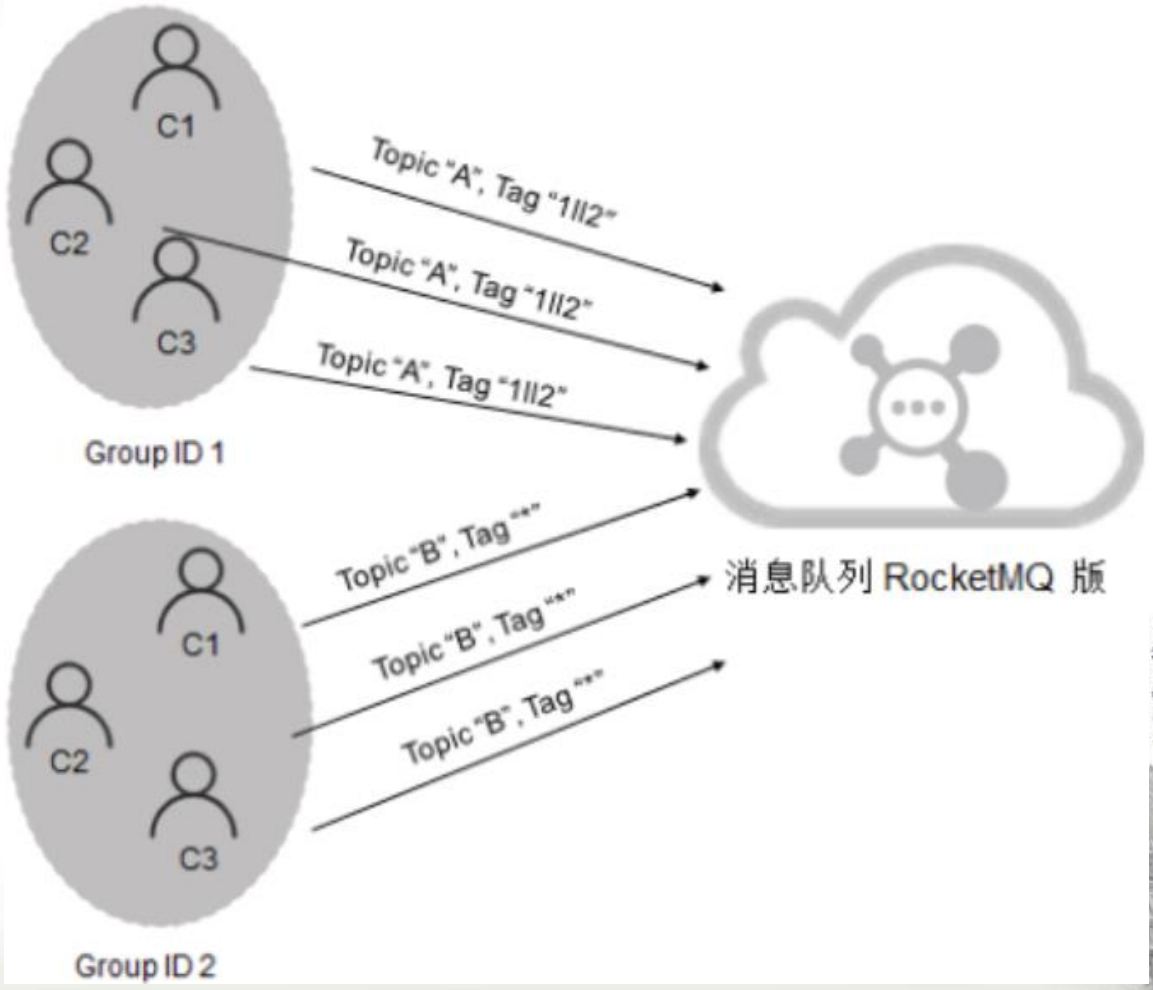
②消息模式

- 广播式消息
  - ✧该模式下，相同Consumer Group的每个Consumer实例都接收同一个Topic的全量消息。
    - 即每条消息都会被发送到Consumer Group中的每个Consumer。
- 集群式消息
  - ✧该模式下，相同Consumer Group的所有Consumer共同消费同一个Topic中的消息，且同一条消息只会被消费一次。
    - 即每条消息只会被发送到Consumer Group中的某个Consumer。

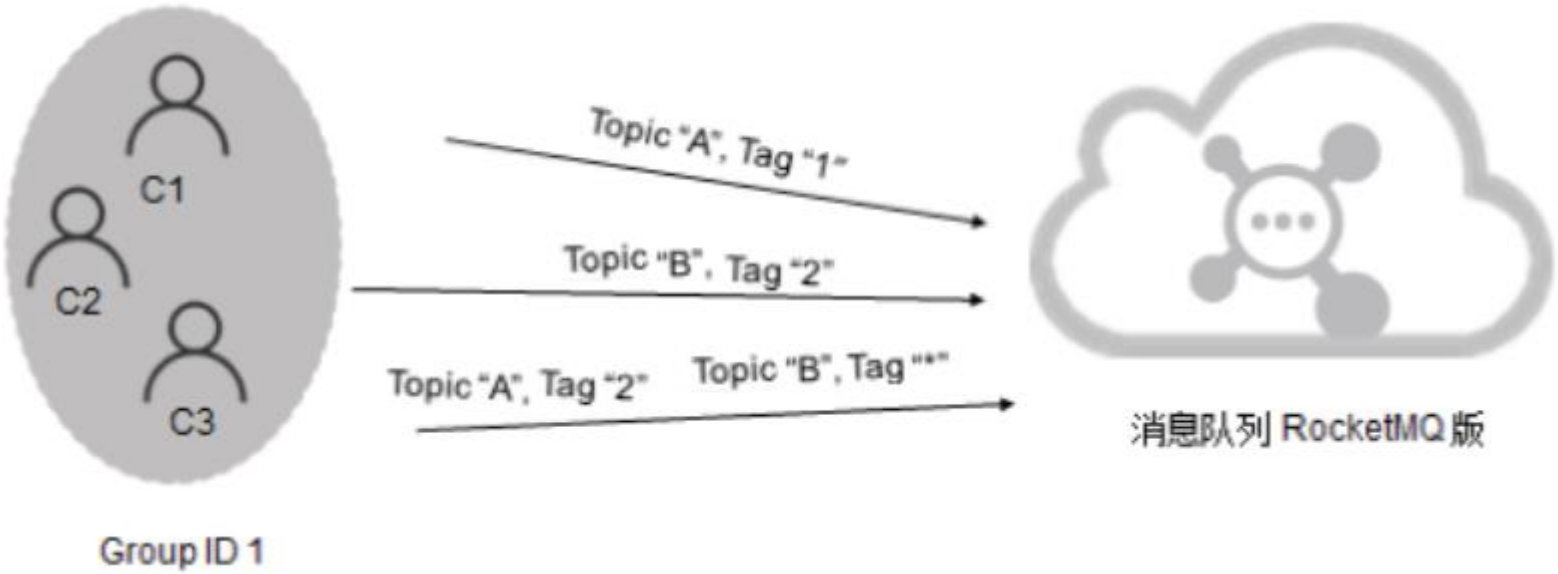
③订阅和消费原则

- 订阅关系一致性原则
  - ✧订阅关系的一致性指的是，同一个消费者组（Group ID相同）下所有Consumer实例所订阅的Topic与Tag的类型和数量及对消息的处理逻辑必须完全一致。
- 至少一次原则
  - ✧即每条消息必须要被成功消费一次。
  - ✧所谓成功消费是指Consumer在消费完消息后会向其消费进度记录器提交其消费消息的offset，offset被成功记录到记录器中。
    - 对于广播式消息消费，Consumer本身就是消费进度记录器
    - 对于集群式消息消费，Broker是消费进度记录器。

正确的订阅关系


C1
C2
C3
Group ID 1
Topic "A", Tag "1||2"
Topic "A", Tag "1||2"
Topic "A", Tag "1||2"
消息队列 RocketMQ 版
C1
C2
C3
Group ID 2
Topic "B", Tag "*"
Topic "B", Tag "*"
Topic "B", Tag "*"

错误的订阅关系


C1
C2
C3
Topic "A", Tag "1"
Topic "B", Tag "2"
Topic "A", Tag "2"
Topic "B", Tag "*"
消息队列 RocketMQ 版
Group ID 1

④消费进度管理

- 消费进度管理通过消费进度offset来记录每个Queue的不同消费组的消费进度。
- 而消费者要消费的第一条消息的起始位置是用户自己通过consumer.setConsumeFromWhere()方法指定，可制定的起始位置有：
  - ✧CONSUME_FROM_LAST_OFFSET：从queue的当前最后一条消息开始消费
  - ✧CONSUME_FROM_FIRST_OFFSET：从queue的第一条消息开始消费
  - ✧CONSUME_FROM_TIMESTAMP：从指定的具体时间戳位置的消息开始消费。
    - 这个具体时间戳通过另外一个语句指定：consumer.setConsumeTimestamp(“20210701080000”) yyyyMMddHHmmss

消费进度管理模式

- 根据消费进度记录器的不同，可以分为两种管理模式：
  - ✧本地管理模式
  - ✧远程管理模式

本地管理模式

- 当消费广播式消息时，每个消费者各自管理各自的消费进度，消费进度**offset**相关数据以**json**的形式持久化到**Consumer**本地磁盘文件中。
  - ✧默认文件路径为当前用户主目录下的**.rocketmq_offsets/${clientId}/${group}/Offsets.json**，其中
    - **${clientId}**为当前消费者**id**，默认为**ip@DEFAULT**；
    - **${group}**为消费者组名称。

远程管理模式

- 当消费集群式消息时，所有Consumer共享Queue的消费进度，消费进度offset相关数据以json的形式持久化到Broker磁盘文件中。
  - ✧文件路径为当前用户主目录下的 store/config/consumerOffset.json 。
    - Broker启动时会加载这个文件，并写入到ConsumerOffsetManager的双层Map里。

消费进度offset的提交

- 集群消息模式下，Consumer消费完一批消息后会向Broker提交消费进度offset，其提交方式分为两种：
  - ✧同步提交：等待返回成功ACK后才读取下一批消息进行消费。
  - ✧异步提交：不需等待返回成功ACK就读取下一批消息进行消费。
    - Consumer可从Broker中直接获取nextBeginOffset
- Broker在收到消费进度后会将其更新到ConsumerOffsetManager的双层Map及consumerOffset.json文件中，然后向该Consumer进行ACK，而ACK内容中包含三项数据：
  - ✧当前消费队列的最小offset（minOffset）、最大offset（maxOffset）、及下次消费的起始offset（nextBeginOffset）。

⑤消费幂等

- 消费幂等是指当出现消费者对某条消息重复消费的情况时，重复消费的结果与消费一次的结果是相同的，并且多次消费并未对业务系统产生任何负面影响。
- 引起重复消费最常见的三种情况：
  - ✧发送时消息重复
  - ✧消费时消息重复
  - ✧**Rebalance**时重复消费

发送时消息重复

- 当一条消息已被成功发送到Broker并完成持久化，此时出现网络闪断，Producer得不到应答就会尝试再次发送消息，Broker中就可能出现两条内容相同并且MessageID也相同的消息。

消费时消息重复

- 当消息已投递到Consumer并完成业务处理，此时出现网络闪断， Broker得不到应答就会尝试再次发送消息（至少一次性原则），Consumer就会收到与之前处理过的内容相同、MessageID也相同的消息。

Rebalance时重复消费

- Rebalance时Consumer在消费新分配给自己的队列时，必须接着之前Consumer提交的消费进度的offset继续消费，但由于默认情况下消费进度的offset是异步提交，这个异步性导致提交到Broker的offset与Consumer实际消费的消息的offset并不一致，这个不一致就可能导致重复消费消息。
  - ✧发生Rebalance，不会导致消息重复，但可能出现重复消费

实现幂等的两要素

- 实现幂等的两要素：
  - ✧幂等令牌：通常指具备唯一业务标识的字符串。
    - 例如，订单号、流水号。
    - 一般由Producer随着消息一同发送。
  - ✧唯一性处理：服务端通过采用一定的算法策略，保证同一个业务逻辑不会被重复执行成功多次。
    - 例如，对同一笔订单的多次支付操作只会成功一次。

实现幂等通用方案

- 实现幂等的通用方案：
  - 首先在缓存中查询幂等令牌是否存在。若存在，则说明本次操作是重复性操作；若不存在，则进入下一步；
  - 唯一性处理前，在DB中查询幂等令牌作为索引的数据是否存在。若存在，则说明本次操作为重复性操作；若不存在，则进入下一步。
  - 唯一性处理后，将幂等令牌写入到缓存，并将幂等令牌作为唯一索引的数据写入到DB中。

⑥消费延迟

- 消息处理流程中，如果Consumer的消费速度跟不上Producer的发送速度，Consumer本地缓冲队列达到上限，就会停止从服务端拉取消息，MQ中未处理的消息就会越来越多（进多出少），从而出现消息堆积，进而会造成消息的消费延迟。

如何避免消费延迟

- 引起消息堆积的主要原因在于客户端的消费能力，而消费能力由消费耗时和消费并发度决定。
- 所以，要避免消费延迟主要从以下两方面进行完善：
  - ✧梳理消息的消费耗时
  - ✧设置消费并发度

梳理消息的消费耗时

- 梳理消息的消费耗时需要关注以下信息：
  - 消息消费逻辑的计算复杂度是否过高，代码是否存在无限循环和递归等缺陷。
  - 消息消费逻辑中的I/O操作是否是必须的，能否用本地缓存等方案规避。
  - 消费逻辑中的复杂耗时的操作是否可以做异步化处理。如果可以，是否会造成逻辑错乱。

设置消费并发度

- 对于消息消费并发度的计算，可以通过以下两步实施：
  - ✧逐步调大单个Consumer节点的线程数，并观测节点的系统指标，得到单个节点最优的消费线程数和消息吞吐量。
    - 理想环境下单节点的最优线程数计算模型为：C *（T1 + T2）/ T1，其中C：CPU内核数；T1：CPU内部逻辑计算耗时；T2：外部IO操作耗时
  - ✧根据上下游链路的流量峰值计算出需要设置的节点数
    - 节点数 = 流量峰值 / 单个节点消息吞吐量

8.2.4 RocketMQ应用模式

- 1.消息范型
- 2.消息过滤
- 3.消息重试

1.消息范型

- 1）普通消息
- 2）顺序消息
- 3）延时消息
- 4）事务消息
- 5）批量消息

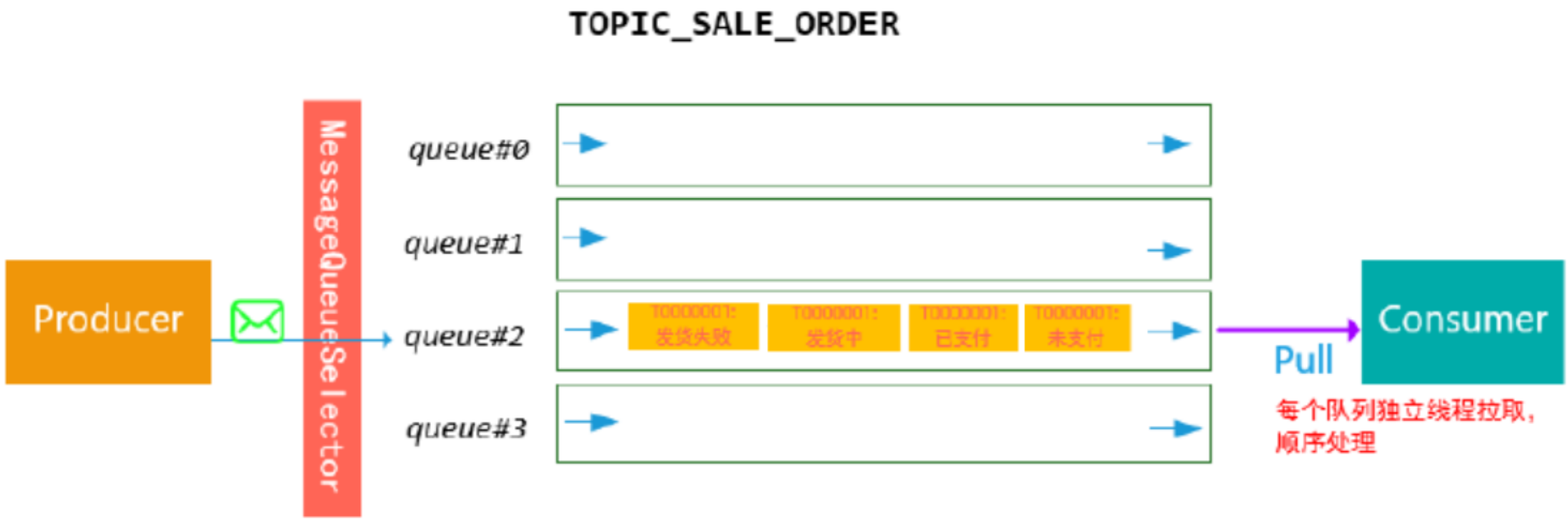
1）普通消息

- Producer对于消息的发送方式有多种选择，不同的方式会产生不同的系统效果。
  - ✧同步发送消息：指Producer发出一条消息后，会在收到MQ返回的ACK之后才发下一条消息。
    - 该方式的消息可靠性最高，但消息发送效率太低。
  - ✧异步发送消息：指Producer发出消息后无需等待MQ返回ACK，直接发送下一条消息。
    - 该方式的消息可靠性可以得到保障，消息发送效率也可以。
  - ✧单向发送消息：指Producer仅负责发送消息，不等待、不处理MQ的ACK。该发送方式时MQ也不返回ACK。
    - 该方式的消息发送效率最高，但消息可靠性较差。

2）顺序消息

- 顺序消息是指严格按照消息的发送顺序进行消费的消息(FIFO)
- 根据有序范围的不同，有序性可分：
  - ✧分区有序：有多个Queue参与，仅可保证在该Queue分区队列上的消息有序
  - ✧全局有序：只有一个Queue参与，保证整个Topic中的消息有序

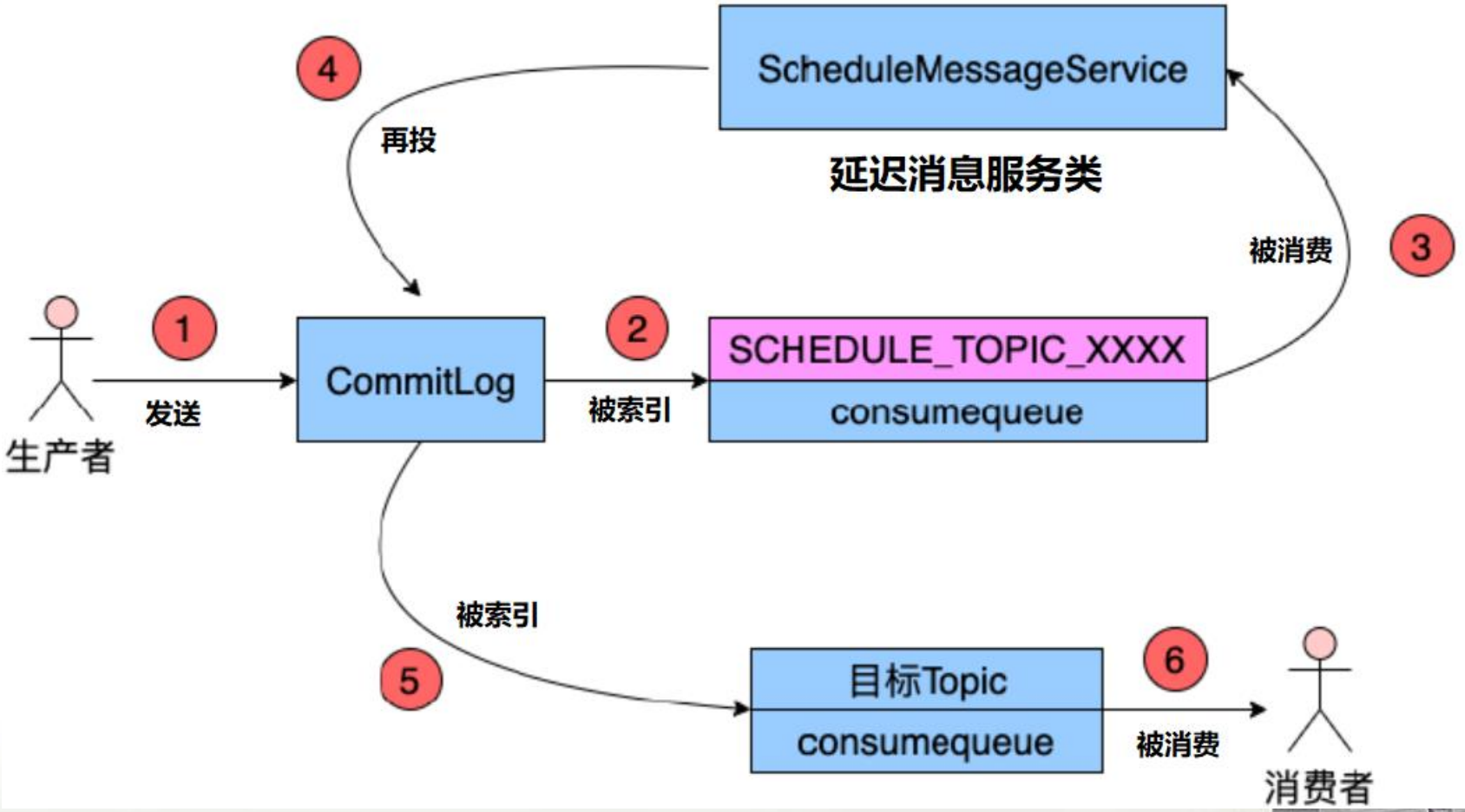
示例


TOPIC_SALE_ORDER
Producer
MessageQueueSelector
queue#0
queue#1
queue#2
queue#3
T0000001:
发货失败
T0000001:
发货中
T0000001:
已支付
T0000001:
未支付
Consumer
Pull
每个队列独立线程拉取，
顺序处理

3）延时消息

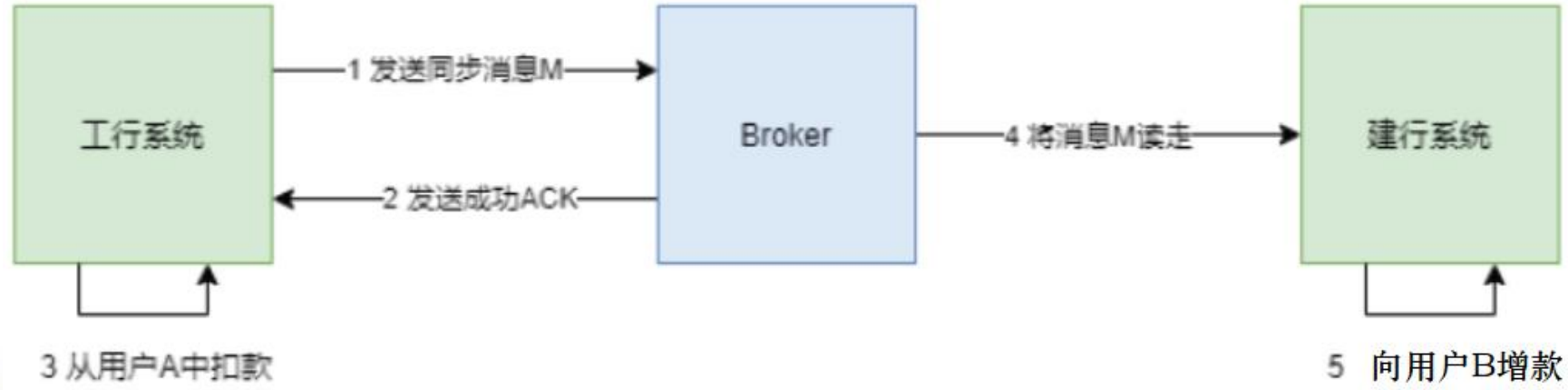
- 当消息写入到Broker后，在指定的时长后才可被消费处理的消息，称为延时消息。
  - ✧再投时间 = 消息存储时间 + 延时等级时间
  - ✧延时等级，可以通过在broker加载的配置文件broker.conf（在RocketMQ安装目录下的conf目录下）中新增如下配置（例如下面增加了1天这个等级1d）：
    - messageDelayLevel = 1s 5s 10s 30s 1m 2m 3m 4m 5m 6m 7m 8m 9m 10m 20m 30m 1h 2h 1d
- 典型应用：
  - ✧超时未支付取消订单。

延时消息实现原理

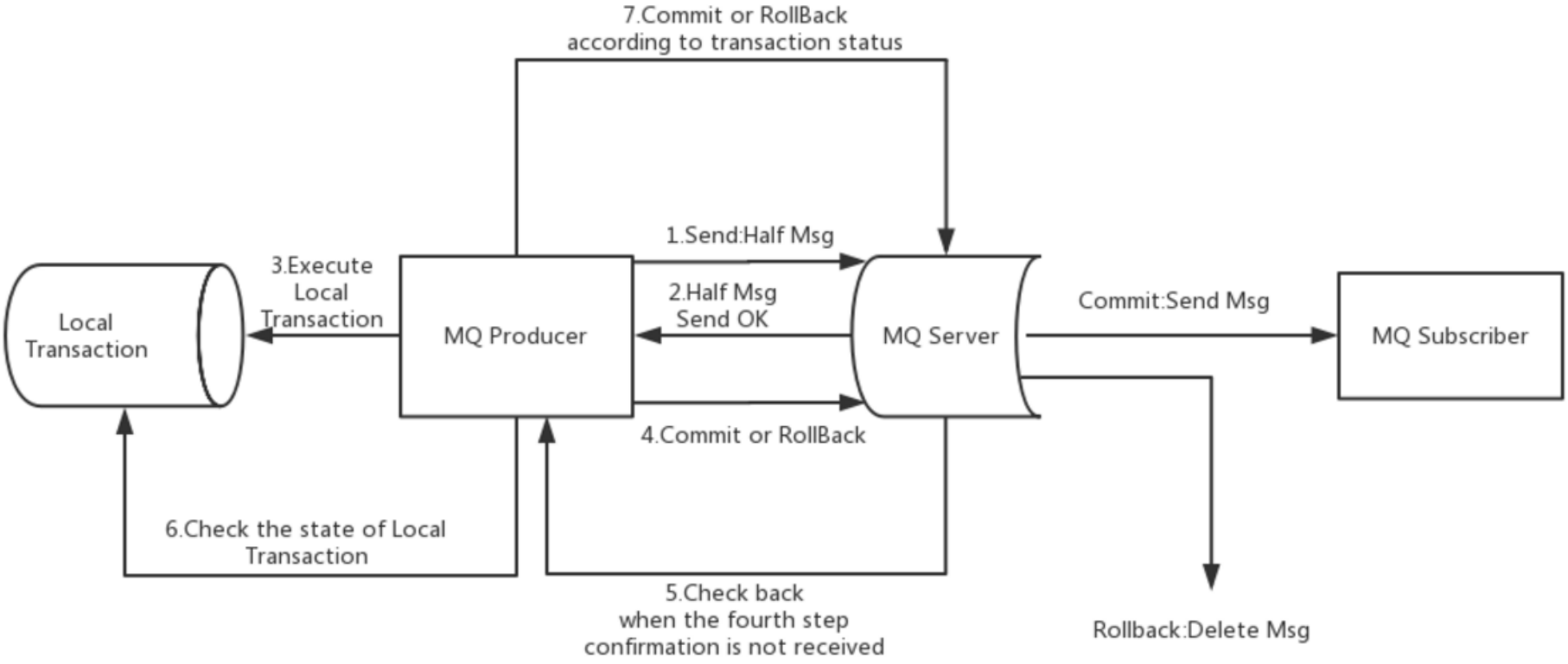

ScheduleMessageService
延迟消息服务类
4
再投
被消费
3
1
2
CommitLog
SCHEDULE_TOPIC_XXXX
consumequeue
发送
被索引
生产者
被索引
5
目标Topic
consumequeue
6
被消费
消费者

4）事务消息

- 问题引入：如何保证分布式系统中的数据一致性？
- 解决思路：让第1、2、3步具有原子性。



解决方案


7.Commit or RollBack
according to transaction status
1.Send:Half Msg
3.Execute
Local
Transaction
2.Half Msg
Send OK
Commit:Send Msg
Local
Transaction
MQ Producer
MQ Server
MQ Subscriber
4.Commit or RollBack
6.Check the state of Local
Transaction
5.Check back
when the fourth step
confirmation is not received
Rollback:Delete Msg

5）批量消息

- 生产者进行消息发送时可以一次发送多条消息，这可以大大提升Producer的发送效率。
- 不过需要注意以下几点：
  - ✧批量发送的消息必须具有相同的Topic
  - ✧批量发送的消息必须具有相同的刷盘策略
  - ✧批量发送的消息不能是延时消息与事务消息

2.消息过滤

- 消息者在进行消息订阅时，除了可以指定要订阅消息的Topic外，还可以对指定Topic中的消息根据指定条件进行过滤，即可以订阅比Topic更加细粒度的消息类型。
- 对于指定Topic消息的过滤有两种过滤方式：
  - ✧1）Tag过滤
    - 如果订阅多个Tag的消息，Tag间使用或运算符(双竖线||)连接
  - ✧2）SQL过滤
    - 是一种通过特定表达式对事先埋入到消息中的用户属性进行筛选过滤的方式
    - 只有使用PUSH模式的消费者才能使用SQL过滤

3.消息重试

- 1）消息发送重试
- 2）消息消费重试
- 3）死信消息处理

1）消息发送重试

- 消息发送重试可以保证消息尽可能发送成功、不丢失，但可能会造成消息重复。
  - ✧消息重复无法避免，但要避免消息的重复消费。
  - ✧只有普通消息（同步或异步）具有发送重试机制，单向消息、顺序消息是没有的。
- 消息发送重试有三种策略：
  - ✧①同步发送失败策略
  - ✧②异步发送失败策略
  - ✧③消息刷盘失败策略

①同步发送失败策略

- 如果发送失败，默认重试2次。
- 但在重试时会尽量发送到未发生过发送失败的Broker或其它Queue。
- 如果超过重试次数，则抛出异常，由Producer去保证消息不丢。
  - ✧但当生产者出现RemotingException、MQClientException和MQBrokerException时，Producer会自动重投消息

②异步发送失败策略

- 异步发送失败重试时，异步重试不会选择其他broker，仅在同一个broker上做重试，所以该策略无法保证消息不丢。

③消息刷盘失败策略

- 消息刷盘超时（Master或Slave）或slave不可用（slave在做数据同步时向master返回状态不是SEND_OK）时，默认不会将消息尝试发送到其他Broker。
- 但对于重要消息可以通过在Broker的配置文件进行如下设置来开启：
  - ✧retryAnotherBrokerWhenNotStoreOK=true

2）消息消费重试

- ①顺序消息的消费重试
- ②无序消息的消费重试
- ③消费重试实现原理

①顺序消息的消费重试

- 对于顺序消息，当Consumer消费消息失败后，为了保证消息的顺序性，其会自动不断地进行消息重试，直到消费成功。
  - ✧消费重试默认间隔时间为1000毫秒。
  - ✧重试期间应用会出现消息消费被阻塞的情况。

②无序消息的消费重试

- 对于无序消息（普通消息、延时消息、事务消息），在集群消费方式下，当Consumer消费消息失败时，每条消息默认最多重试16次，但每次重试的时间间隔不同，会逐渐变长。
  - ✧若第16次重试仍然失败，则将消息索引到死信队列。
  - ✧广播消费方式不提供失败重试特性。

每次重试的时间间隔

重试次数	与上次重试的间隔时间	重试次数	与上次重试的间隔时间
1	10秒	9	7分钟
2	30秒	10	8分钟
3	1分钟	11	9分钟
4	2分钟	12	10分钟
5	3分钟	13	20分钟
6	4分钟	14	30分钟
7	5分钟	15	1小时
8	6分钟	16	2小时

消费重试的时间间隔与延时消费的延时等级十分相似，除了没有延时等级的前两个时间外，其它的时间都是相同的。

③消费重试实现原理

- Broker对于重试消息的处理是通过延时消息实现的。
  - ✧当且只有当一个消费者组出现需要进行重试消费的消息时，Broker会为该消费者组设置一个Topic名称为%RETRY%consumerGroup@consumerGroup的重试队列。
  - ✧然后，Broker先将消息索引到SCHEDULE_TOPIC_XXXX延迟队列，延迟时间到后，会将消息索引到消费者组的重试队列中，而后由消费者进行再次消费。

3）死信消息处理

- 当一条消息被消费时达到最大重试次数后，若依然失败，则该消息被称为死信消息（**Dead-Letter Message，DLM**）
- 当且只有当一个消费者组出现死信消息时，**Broker**会为该消费者组设置一个**Topic**名称为**%DLQ%consumerGroup@consumerGroup**的死信队列。
- 死信队列中的消息不会再被消费者正常消费，**3**天后会被自动删除或需由人工处理。


聪明等于勤奋
努力就是幸福