Apache Kafka 原理与架构

本文主要讲解 Kafka 是什么、Kafka 的架构包括工作流程和存储机制，以及生产者和消费者，最终大家会掌握 Kafka 中最重要的概念，分别是 broker、producer、consumer、consumer group、topic、partition、replica、leader、follower，这是学会和理解 Kafka 的基础和必备内容。

1. 定义

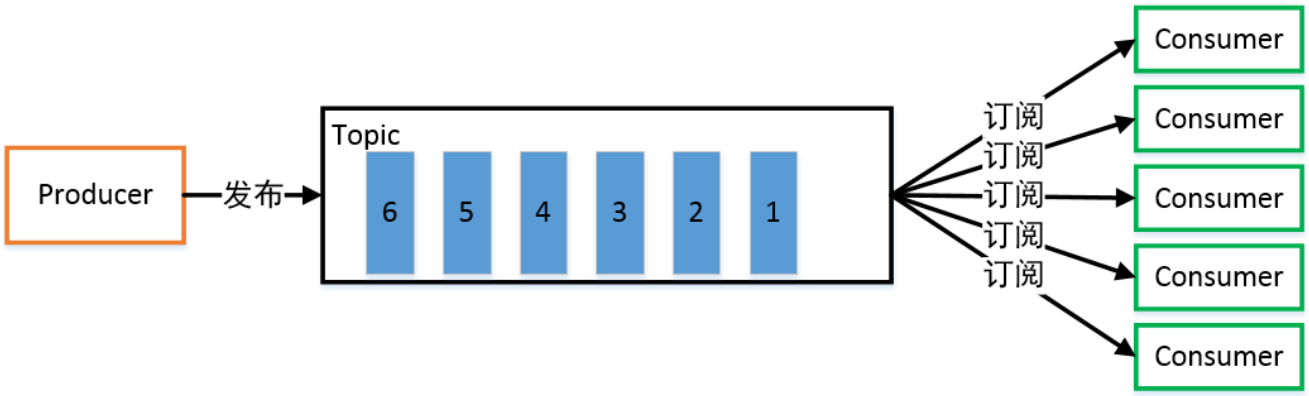
Kafka 是一个分布式的基于发布/订阅模式的消息队列（Message Queue），主要应用与大数据实时处理领域。

1.1 消息队列

Kafka 本质上是一个 MQ（Message Queue），使用消息队列的好处？（面试会问）

1. 解耦：允许我们独立的扩展或修改队列两边的处理过程。
2. 可恢复性：即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。
3. 缓冲：有助于解决生产消息和消费消息的处理速度不一致的情况。
4. 灵活性&峰值处理能力：不会因为突发的超负荷的请求而完全崩溃，消息队列能够使关键组件顶住突发的访问压力。
5. 异步通信：消息队列允许用户把消息放入队列但不立即处理它。

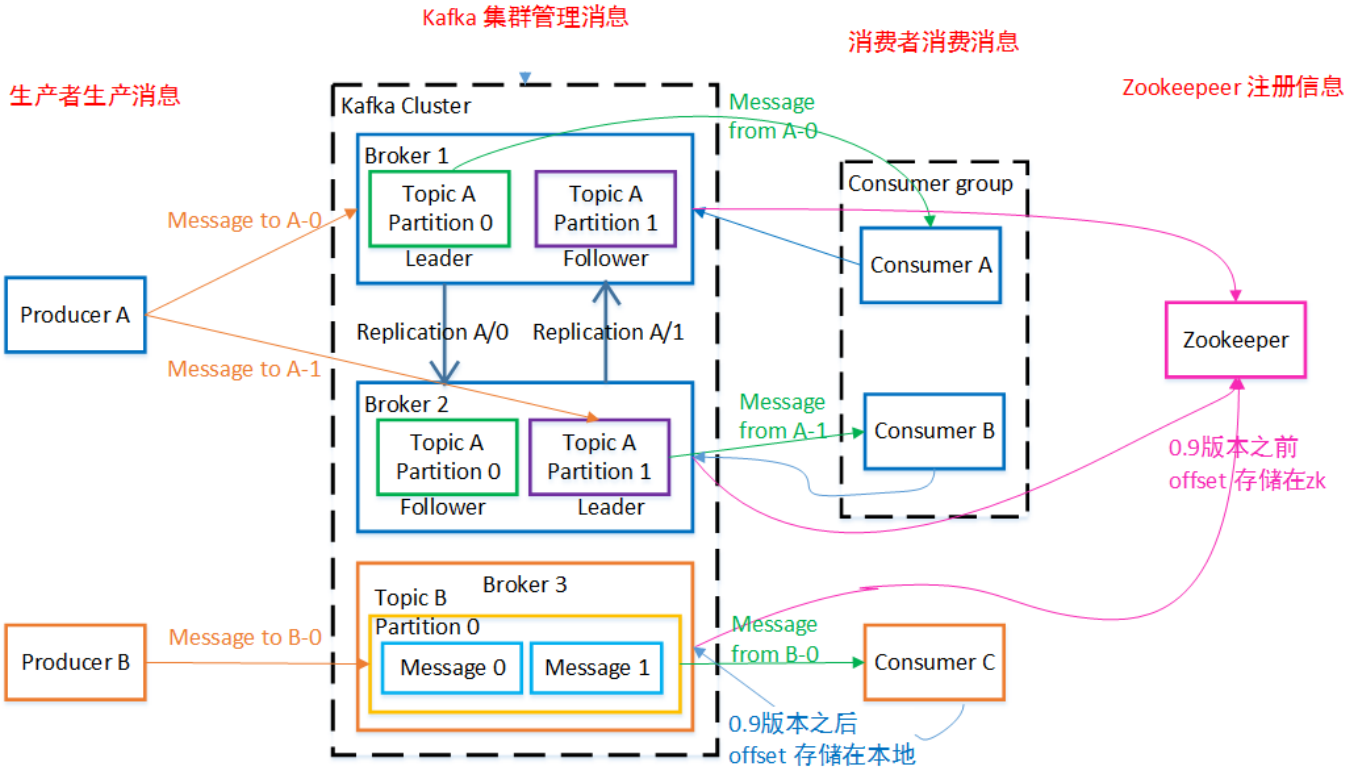
1.2 发布/订阅模式



一对多，生产者将消息发布到 topic 中，有多个消费者订阅该主题，发布到 topic

的消息会被所有订阅者消费，被消费的数据不会立即从 topic 清除。

2. 架构



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

Kafka 存储的消息来自任意多被称为 Producer 生产者的进程。数据从而可以被发布到不同的 Topic 主题下的不同 Partition
分区。在一个分区内，这些消息被索引并连同时间戳存储在一起。其它被称为 Consumer
消费者的进程可以从分区订阅消息。Kafka
运行在一个由一台或多台服务器组成的集群上，并且分区可以跨集群结点分布。
下面给出 Kafka 一些重要概念，让大家对 Kafka
有个整体的认识和感知，后面还会详细的解析每一个概念的作用以及更深入的原理。

- Producer： 消息生产者，向 Kafka Broker 发消息的客户端。
- Consumer： 消息消费者，从 Kafka Broker 取消息的客户端。
- Consumer Group： 消费者组（CG），消费者组内每个消费者负责消费不同分区的数据，提高消费能力。一个分区只能由组内一个消费者消费，消费者组之间互不影响。所有的消费者都属于某个消费者组，即消费者组是逻辑上的一个订阅者。
- Broker： 一台 Kafka 机器就是一个 broker。一个集群由多个 broker 组成。一个 broker 可以容纳多个 topic。
- Topic： 可以理解为一个队列，topic 将消息分类，生产者和消费者面向的是同一个 topic。
- Partition： 为了实现扩展性，提高并发能力，一个非常大的 topic 可以分布到多个 broker（即服务器）上，一个 topic 可以分为多个 partition，每个 partition 是一个

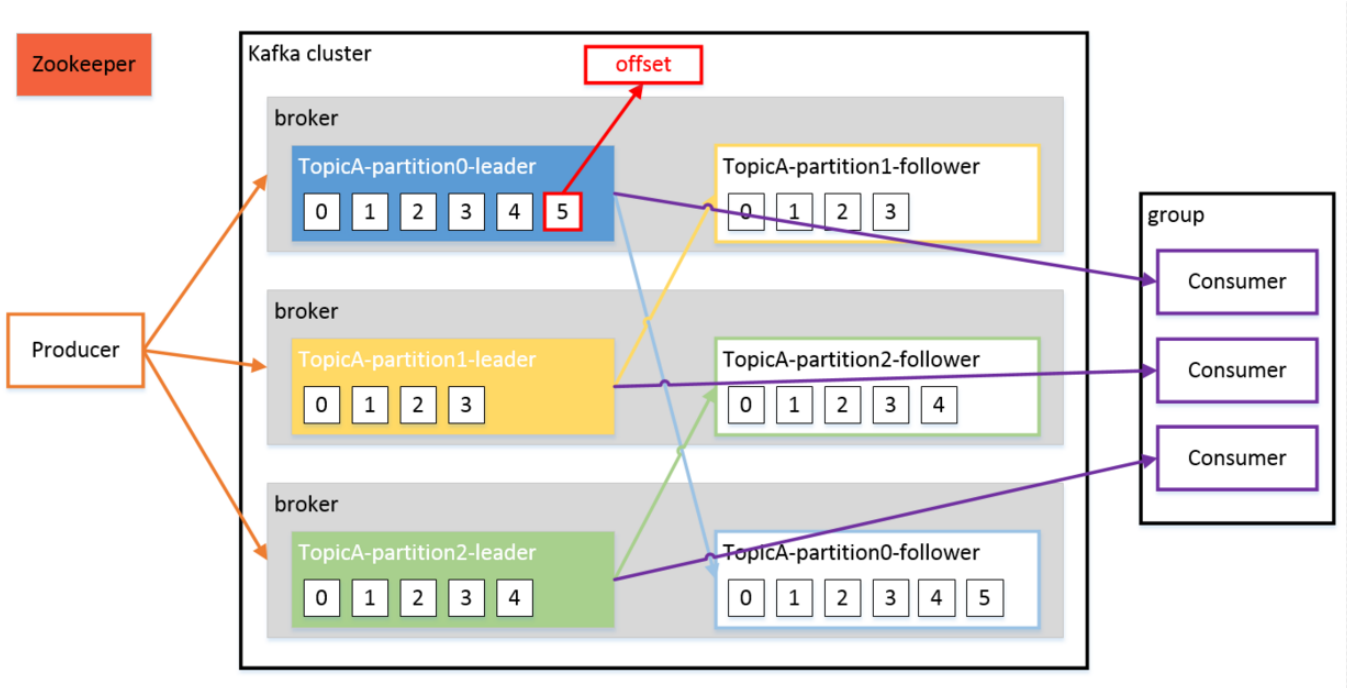
有序的队列。

- Replica： 副本，为实现备份的功能，保证集群中的某个节点发生故障时，该节点上的 partition 数据不丢失，且 Kafka 仍然能够继续工作，Kafka 提供了副本机制，一个 topic 的每个分区都有若干个副本，一个 leader 和若干个 follower。
- Leader： 每个分区多个副本的“主”副本，生产者发送数据的对象，以及消费者消费数据的对象，都是 leader。
- Follower： 每个分区多个副本的“从”副本，实时从 leader 中同步数据，保持和 leader 数据的同步。leader 发生故障时，某个 follower 还会成为新的 leader。
- offset： 消费者消费的位置信息，监控数据消费到什么位置，当消费者挂掉再重新恢复的时候，可以从消费位置继续消费。
- Zookeeper： Kafka 集群能够正常工作，需要依赖于 zookeeper，zookeeper 帮助 Kafka 存储和管理集群信息。

3 工作流程

Kafka集群将 Record 流存储在称为 topic 的类别中，每个记录由一个键、一个值和一个时间戳组成。Kafka 是一个分布式流平台，这到底是什么意思？

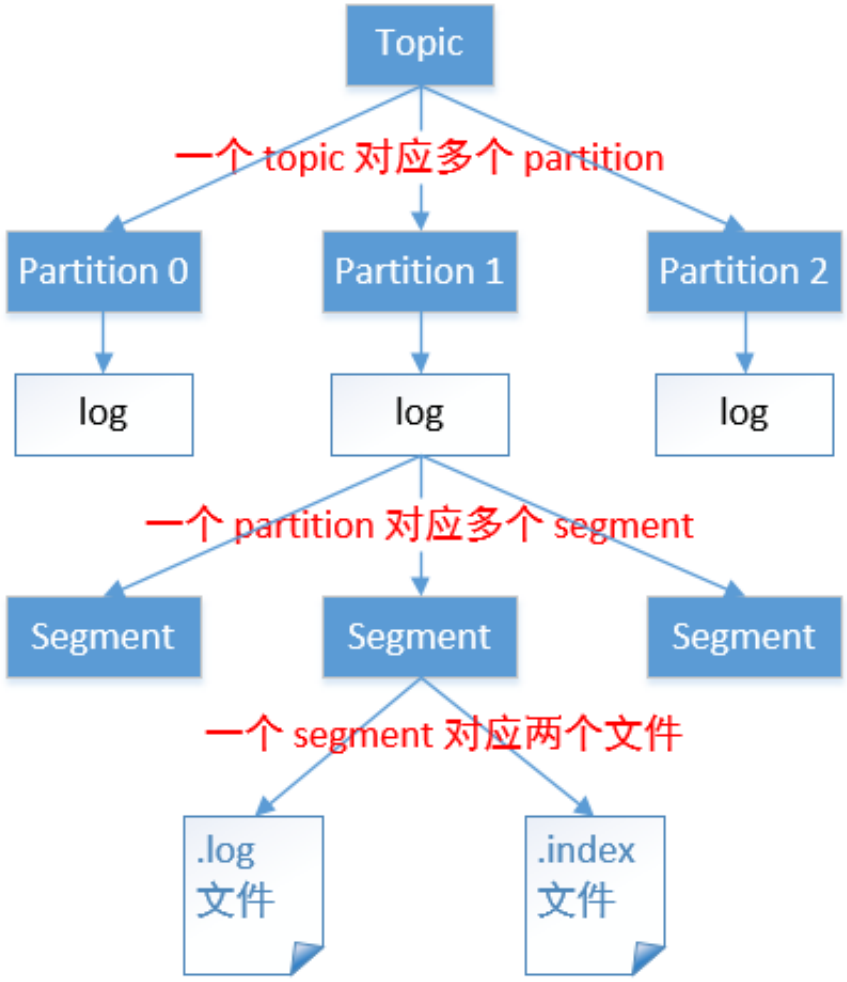
- 发布和订阅记录流，类似于消息队列或企业消息传递系统。
- 以容错的持久方式存储记录流。
- 处理记录流。



Kafka 中消息是以 topic 进行分类的，生产者生产消息，消费者消费消息，面向的都是同一个 topic。

topic 是逻辑上的概念，而 partition 是物理上的概念，每个 partition 对应于一个 log 文件，该 log 文件中存储的就是 Producer 生产的数据。Producer 生产的数据会不断追加到该 log 文件末端，且每条数据都有自己的
offset。消费者组中的每个消费者，都会实时记录自己消费到了哪个
offset，以便出错恢复时，从上次的位置继续消费。

4 存储机制



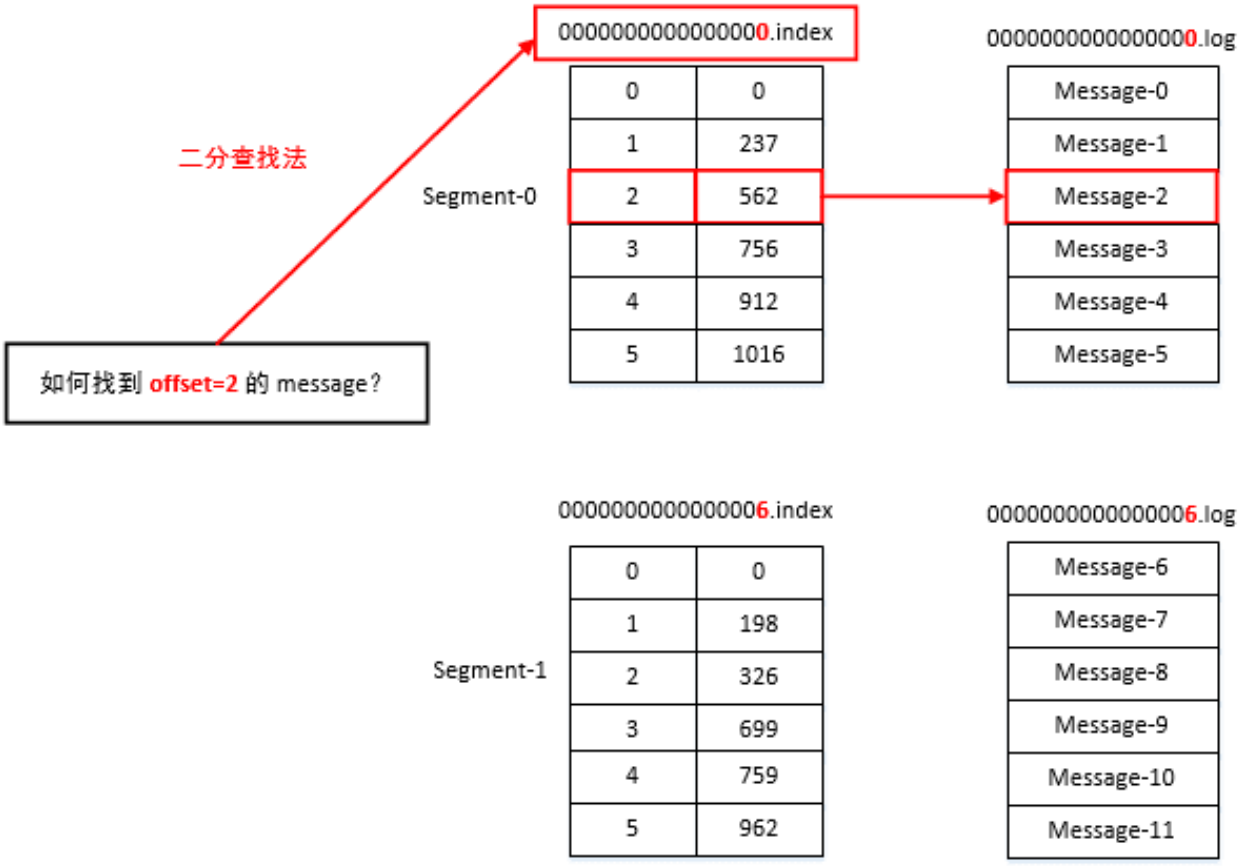
如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

由于生产者生产的消息会不断追加到 log 文件末尾，为防止 log 文件过大导致数据定位效率低下，Kafka 采取了分片和索引机制，将每个 partition 分为多个 segment，每个 segment 对应两个文件：“.index” 索引文件和 “.log” 数据文件。这些文件位于同一文件下，该文件夹的命名规则为：topic 名-分区号。例如，first 这个 topic 有三分分区，则其对应的文件夹为 first-0，first-1，first-2。

```
# ls /root/data/kafka/first-0
00000000000000009014.index
```

00000000000000009014.log
00000000000000009014.timeindex
00000000000000009014.snapshot
leader-epoch-checkpoint

index 和 log 文件以当前 segment 的第一条消息的 offset 命名。下图为 index 文件 和 log 文件的结构示意图。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

“.index” 文件存储大量的索引信息，“.log”
文件存储大量的数据，索引文件中的元数据指向对应数据文件中 message 的物理偏移量。

5. 生产者

5.1 分区策略

5.1.1 分区原因

- 方便在集群中扩展，每个 partition 可以通过调整以适应它所在的机器，而一个 topic 又可以有多个 partition 组成，因此可以以 partition 为单位读写了。
- 可以提高并发，因此可以以 partition 为单位读写了。

5.1.2 分区原则

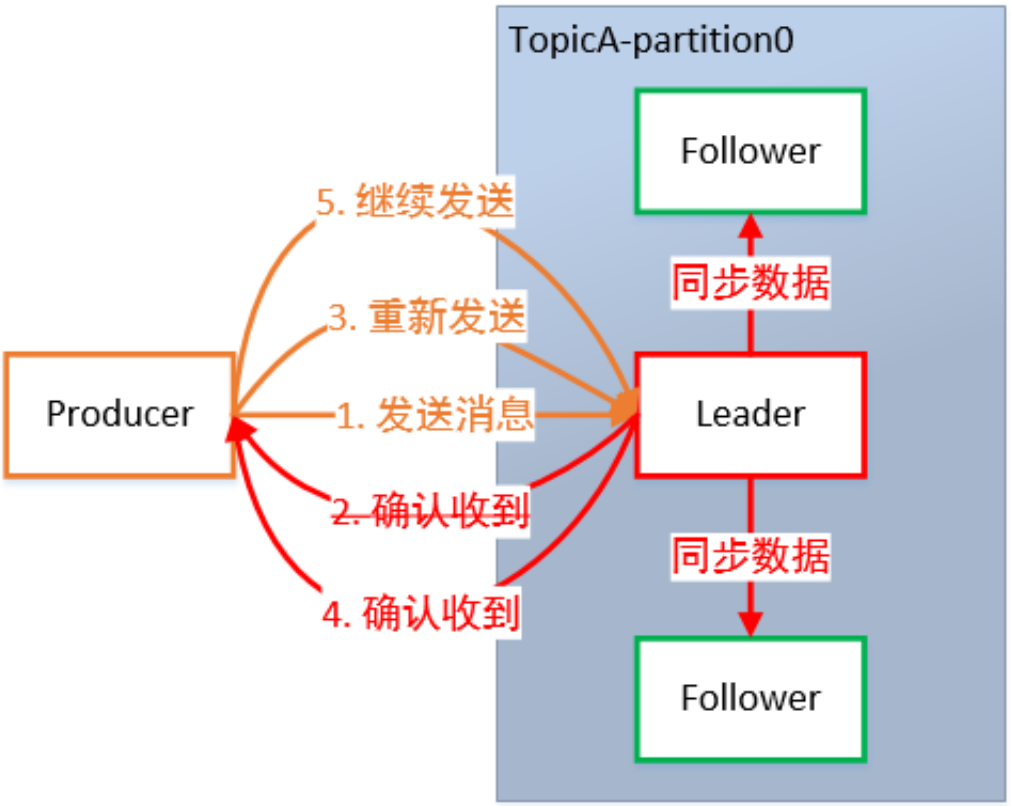
我们需要将 Producer 发送的数据封装成一个 ProducerRecord 对象。该对象需要指定一些参数：

- topic：string 类型，NotNull
- partition：int 类型，可选
- timestamp：long 类型，可选
- key：string类型，可选
- value：string 类型，可选
- headers：array 类型，Nullable

（1）指明 partition 的情况下，直接将给定的 value 作为 partition 的值。
（2）没有指明 partition 但有 key 的情况下，将 key 的 hash 值与分区数取余得到 partition 值。
（3）既没有 partition 有没有 key 的情况下，第一次调用时随机生成一个整数（后面每次调用都在这个整数上自增），将这个值与可用的分区数取余，得到 partition 值，也就是常说的 round-robin 轮询算法。

5.2 数据可靠性保证

为保证 producer 发送的数据，能可靠地发送到指定的 topic，topic 的每个 partition 收到 producer 发送的数据后，都需要向 producer 发送 ack（acknowledge 确认收到），如果 producer 收到 ack，就会进行下一轮的发送，否则重新发送数据。



5.2.1 副本数据同步策略

（1）何时发送 ack？
确保有 follower 与 leader 同步完成，leader 再发送 ack，这样才能保证 leader 挂掉之后，能在 follower 中选举出新的 leader 而不丢数据。
（2）多少个 follower 同步完成后发送 ack？
全部 follower 同步完成，再发送 ack。

方案	优点	缺点
半数以上完成同步，就发送 ack。	延迟低。	选举新的 leader 时，容忍 n 台节点故障，需要 2n+1 个副本。
全部完成同步，才发送 ack。	选举新的 leader 时，容忍 n 台节点故障，需要 n+1 个副本。	延迟高。

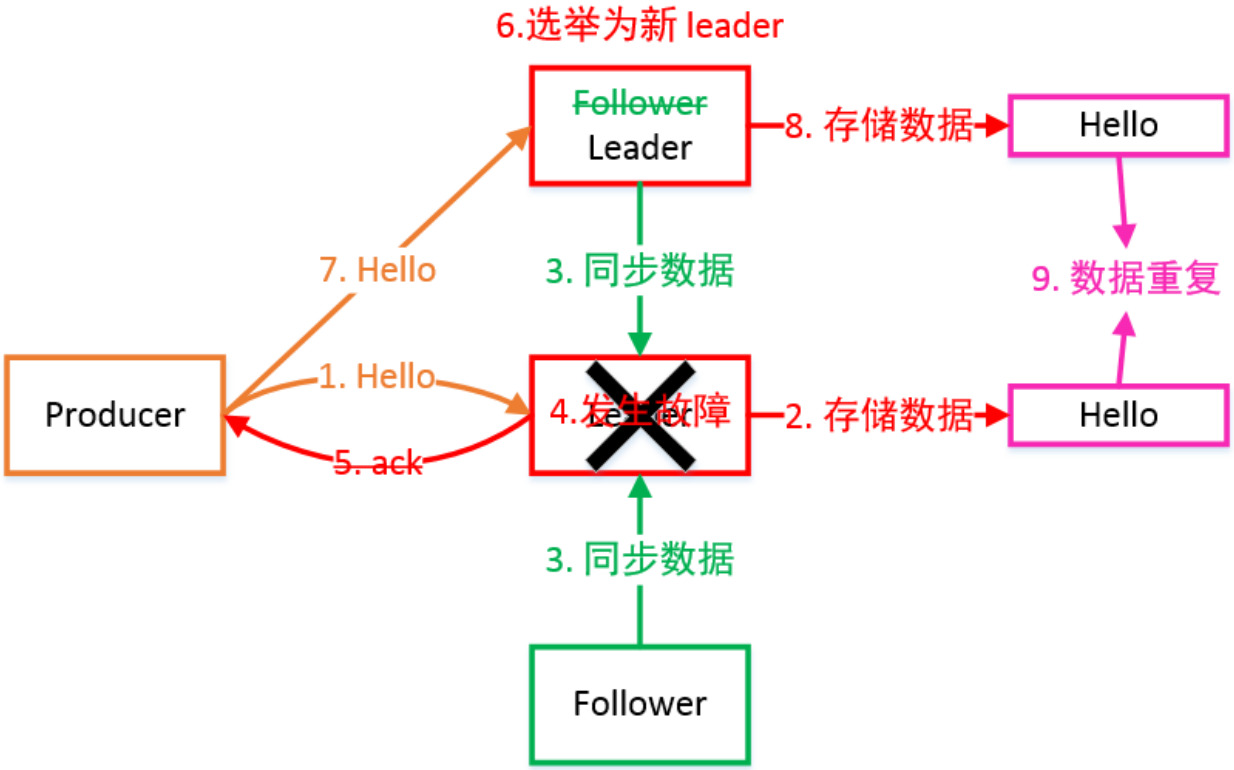
5.2.2 ISR

采用第二种方案，所有 follower 完成同步，producer 才能继续发送数据，设想有一个 follower 因为某种原因出现故障，那 leader 就要一直等到它完成同步。这个问题怎么解决？
leader维护了一个动态的 in-sync replica set（ISR）：和 leader 保持同步的 follower 集合。当 ISR 集合中的 follower 完成数据的同步之后，leader 就会给 follower 发送 ack。如果 follower 长时间未向 leader 同步数据，则该 follower 将被踢出 ISR 集合，该时间阈值由 replica.lag.time.max.ms 参数设定。leader 发生故障后，就会从 ISR 中选举出新的 leader。

5.2.3 ack 应答机制

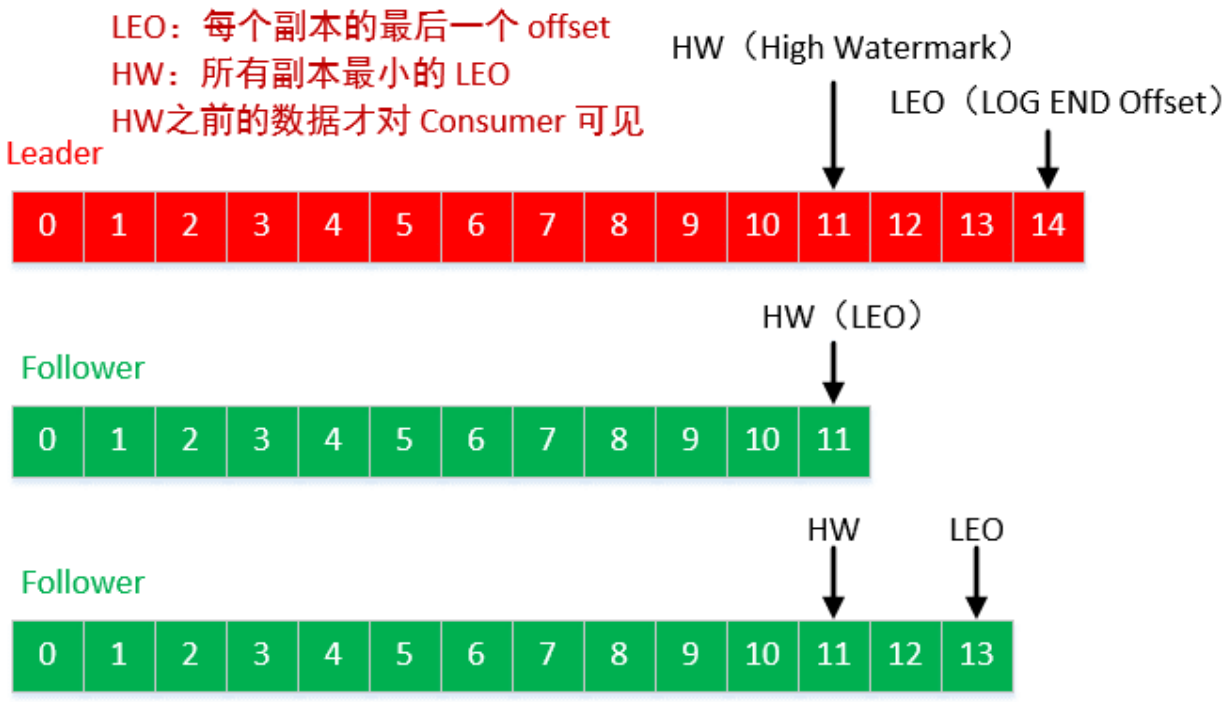
对于某些不太重要的数据，对数据的可靠性要求不是很高，能够容忍数据的少量丢失，所以没必要等 ISR 中的 follower 全部接受成功。
所以 Kafka
为用户提供了三种可靠性级别，用户根据可靠性和延迟的要求进行权衡，选择以下的配置。
（1）ack 参数配置：

- 0：producer 不等待 broker 的 ack，这提供了最低延迟，broker 一收到数据还没有写入磁盘就已经返回，当 broker 故障时有可能丢失数据。
- 1：producer 等待 broker 的 ack，partition 的 leader 落盘成功后返回 ack，如果在 follower 同步成功之前 leader 故障，那么将会丢失数据。
- -1（all）：producer 等待 broker 的 ack，partition 的 leader 和 follower 全部落盘成功后才返回 ack。但是在 broker 发送 ack 时，leader 发生故障，则会造成数据重复。



如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

5.2.4 故障处理细节



LEO：每个副本最大的 offset。
HW：消费者能见到的最大的 offset，ISR 队列中最小的 LEO。
（1）Follower 故障
follower 发生故障后会被临时踢出 ISR 集合，待该 follower 恢复后，follower 会读取本地磁盘记录的上次的 HW，并将 log 文件高于 HW 的部分截取掉，从 HW 开始向 leader 进行同步数据操作。等该 follower 的 LEO 大于等于该 partition 的 HW，即 follower 追上 leader 后，就可以重新加入 ISR 了。
（2）Leader 故障
leader 发生故障后，会从 ISR 中选出一个新的 leader，之后，为保证多个副本之间的数据一致性，其余的 follower 会先将各自的 log 文件高于 HW 的部分截掉，然后从新的 leader 同步数据。
注意：这只能保证副本之间的数据一致性，并不能保证数据不丢失或者不重复。

5.3 Exactly Once 语义

将服务器的 ACK 级别设置为-1，可以保证 producer 到 server 之间不会丢失数据，即 At Least Once 语义。相对的，将服务器 ACK 级别设置为0，可以保证生产者每条消息只会被发送一次，即At Most Once 语义。
At Least Once 可以保证数据不丢失，但是不能保证数据不重复；相对的，At Most Once 可以保证数据不重复，但是不能保证数据不丢失。但是，对于一些非常重要的信息，比如交易数据，下游数据消费者要求数据既不重复也不丢失，即 Exactly Once 语义。
0.11版本的 Kafka，引入了幂等性：producer 不论向 server 发送多少重复数据，server

端都只会持久化一条。即：

At Least Once + 幂等性 = Exactly Once
复制代码

要启用幂等性，只需要将 producer 的参数中 enable.idompotence 设置为 true
即可。开启幂等性的 producer 在初始化时会被分配一个 PID，发往同一 partition 的消息会附带
Sequence Number。而 borker 端会对 <PID,Partition,SeqNumber>
做缓存，当具有相同主键的消息提交时，broker 只会持久化一条。
但是 PID 重启后就会变化，同时不同的 partition
也具有不同主键，所以幂等性无法保证跨分区会话的 Exactly Once。

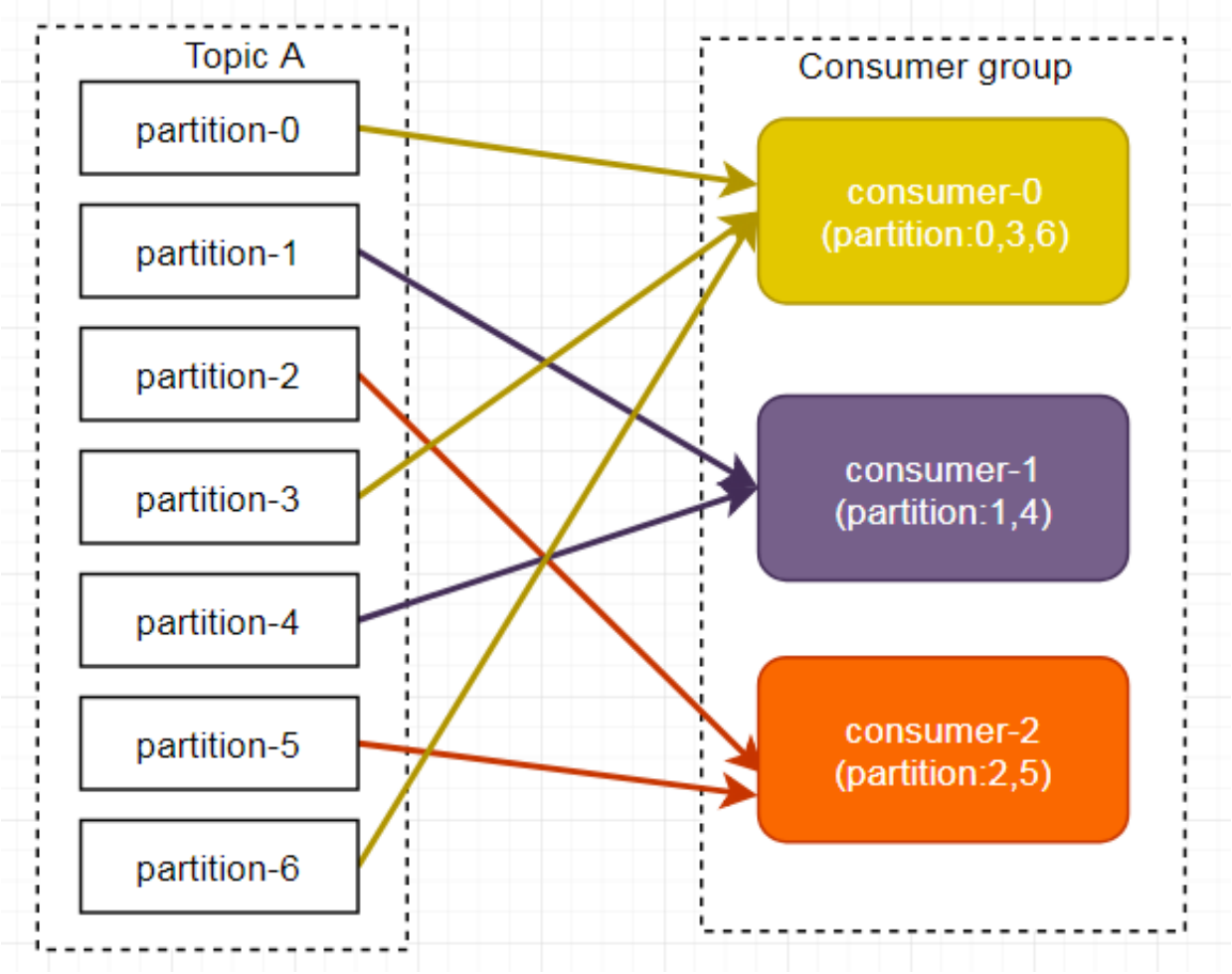
6. 消费者

6.1 消费方式

consumer 采用 pull（拉取）模式从 broker 中读取数据。
consumer 采用 push（推送）模式，broker 给 consumer 推送消息的速率是由 broker 决定的，很难适应消费速率不同的消费者。它的目标是尽可能以最快速度传递消息，但是这样很容易造成 consumer 来不及处理消息，典型的表现就是拒绝服务以及网络拥塞。而 pull 模式则可以根据 consumer 的消费能力以适当的速率消费消息。
pull 模式不足之处是，如果 Kafka
没有数据，消费者可能会陷入循环中，一直返回空数据。因为消费者从 broker
主动拉取数据，需要维护一个长轮询，针对这一点， Kafka
的消费者在消费数据时会传入一个时长参数 timeout，如果当前没有数据可供消费，consumer 会等待一段时间之后再返回，这段时长即为 timeout。

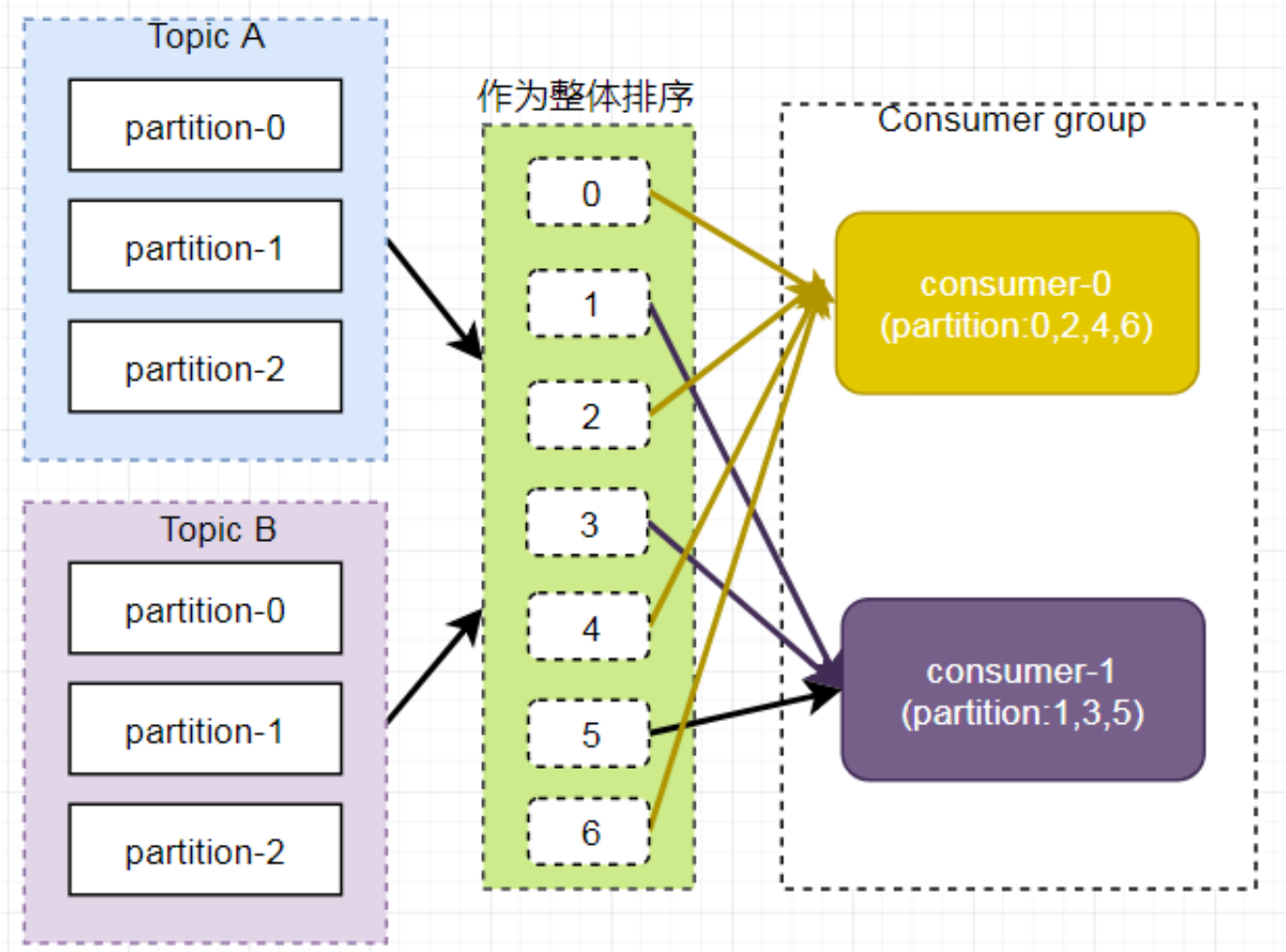
6.2 分区分配策略

一个 consumer group 中有多个 consumer，一个 topic 有多个 partition，所以必然会涉及到 partition 的分配问题，即确定哪个 partition 由哪个 consumer 来消费。
Kafka 有两种分配策略，一个是 RoundRobin，一个是 Range，默认为range，当消费者组内消费者发生变化时，会触发分区分配策略（方法重新分配）。
（1）RoundRobin

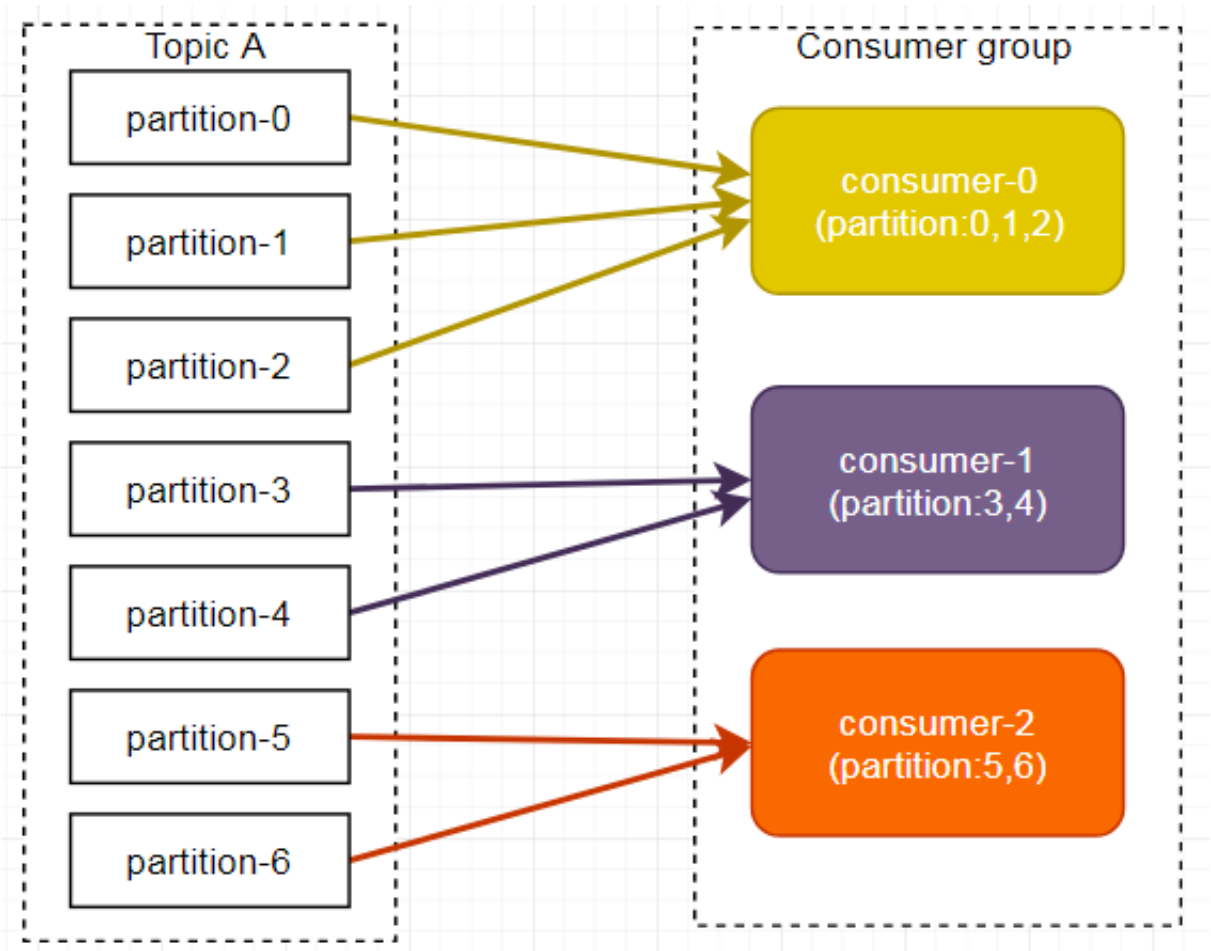


如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

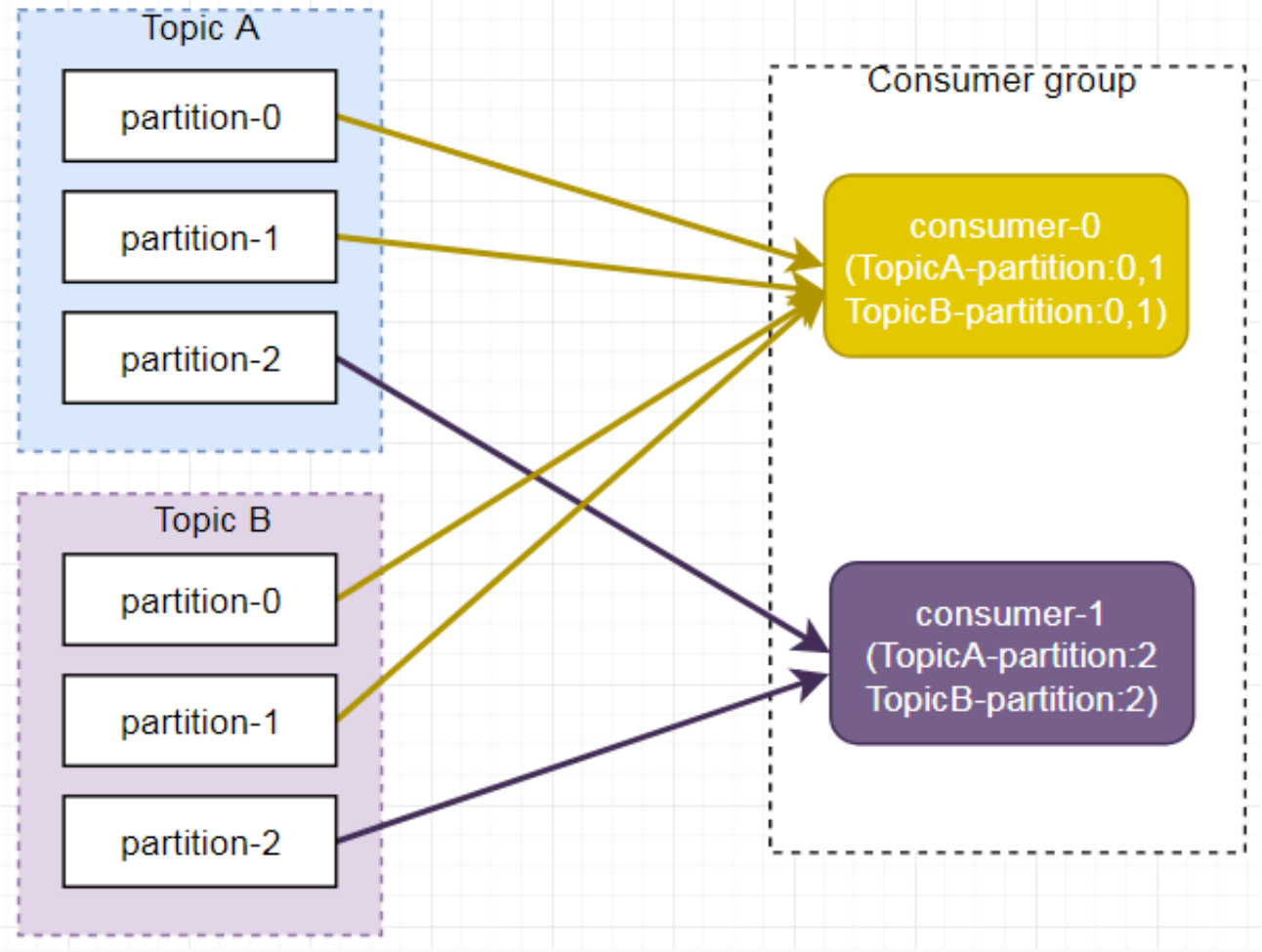
RoundRobin 轮询方式将分区所有作为一个整体进行 hash 排序，消费者组内分配分区个数最大差别为1，是按照组来分的，可以解决多个消费者消费数据不均衡的问题。
但是，当消费者组内订阅不同主题时，可能造成消费混乱，如下图所示，consumer0 订阅主题A，consumer1 订阅主题B，将 A、B主题的分区排序后分配给消费者组，TopicB 分区中的数据可能分配到 consumer0 中。



（2）Range



range 方式是按照主题来分的，不会产生轮询方式的消费混乱问题。
但是，如下图所示，consumer0、consumer1 同时订阅了主题A和B，可能造成消息分配不对等问题，当消费者组内订阅的主题越多，分区分配可能越不均衡。



6.3 offset 的维护

由于 consumer 在消费过程中可能会出现断电宕机等故障，consumer 恢复后，需要从故障前的位置继续消费，所以 consumer 需要实时记录自己消费到了哪个 offset，以便故障恢复后继续消费。
Kafka 0.9 版本之前，consumer 默认将 offset 保存在 Zookeeper 中，从 0.9 版本开始，consumer 默认将 offset 保存在 Kafka 一个内置的 topic 中，该 topic 为 __consumer_offsets。

本文原文 Kafka 概述：深入理解架构