干货总结！Kafka 面试大全

（万字长文，37 张图，28 个知识点）

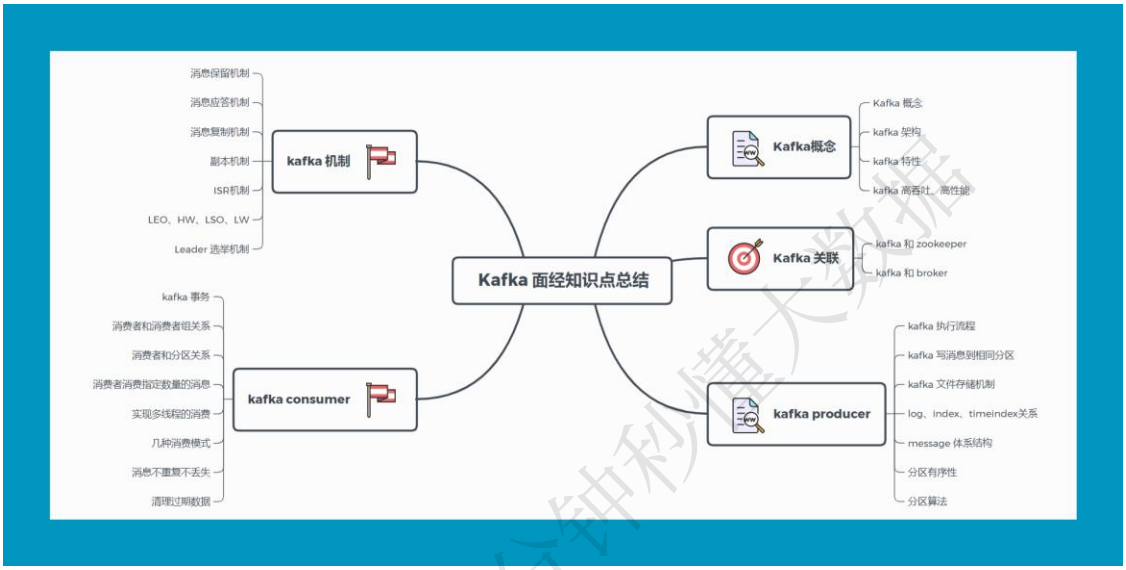
本文作者：在 IT 中穿梭旅行

大家好，我是土哥。

周末用了整整两天时间，整理了一下 Kafka 面试的连环问题，保证你看完后，对 Kafka 有了更深层次的了解。

全文总结的 Kafka 题目之间的 关联性 很强，本文将通过 **问答 + 图解** 的形式 由浅入深 帮助大家进一步学习和理解 Kafka 分布式流式处理平台。

全文总计 2 万字、28 个知识点、40 张原理、流程图。提纲如下：



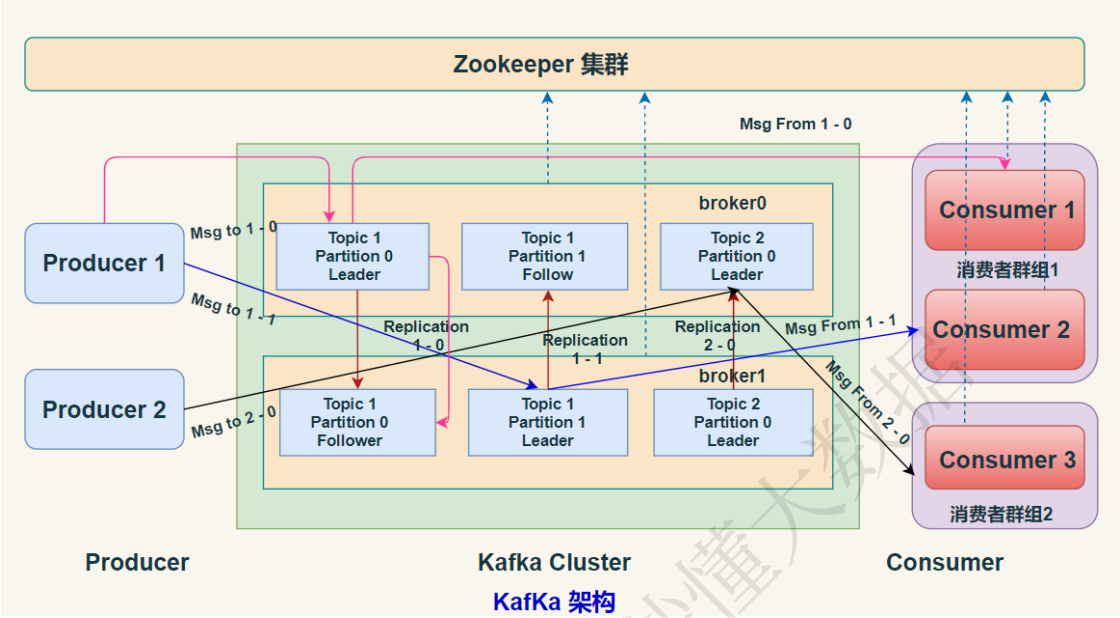
正文

⋆ 1、什么是 kafka？

Kafka 起初是由 Linkedin 公司采用 Scala 语言开发的一个多分区、多副本且基于 ZooKeeper 协调的分布式消息系统，现已被捐献给 Apache 基金会。目前 Kafka 已经定位为一个**分布式流式处理平台**，它以高吞吐、可持久化、可水平扩展、支持流数据处理等多种特性而被广泛使用。

⋆ 2、kafka 的架构描述一下？

如下图所示：



KafKa 架构

Kafak 总体架构图中包含多个概念：

（1）ZooKeeper： Zookeeper 负责保存 broker 集群元数据，并对控制器进行选举等操作。

（2）Producer： 生产者负责创建消息，将消息发送到 Broker。

（3）Broker: 一个独立的 Kafka 服务器被称作 broker，broker 负责接收来自生产者的消息，**为消息设置偏移量**，并将消息存储在磁盘。broker 为消费者提供服务，对读取分区的请求作出响应，返回已经提交到磁盘上的消息。

（4）Consumer： 消费者负责从 Broker 订阅并消费消息。

（5）Consumer Group： Consumer Group 为消费者组，一个消费者组可以包含一个或多个 Consumer 。

使用 **多分区 + 多消费者** 方式可以极大 **提高数据下游的处理速度**，同一消费者组中的消费者不会重复消费消息，同样的，不同消费组中的消费者消费消息时互不影响。Kafka 就是通过消费者组的方式来实现消息 P2P 模式和广播模式。

（6）Topic： Kafka 中的消息 **以 Topic 为单位进行划分**，生产者将消息发送到特定的 Topic，而消费者负责订阅 Topic 的消息并进行消费。

（7）Partition： 一个 Topic 可以细分为多个分区，**每个分区只属于单个主题**。同一个主题下不同分区包含的消息是不同的，分区在存储层面可以看作一个可追加的**日志（Log）文件**，消息在被追加到分区日志文件的时候都会分配一个特定的 **偏移量（offset）**。

（8）Offset： offset 是消息在分区中的唯一标识，**Kafka 通过它来保证消息在分区内的顺序性**，不过 offset 并不跨越分区，也就是说，**Kafka 保证的是分区有序性而不是主题有序性**。

（9）Replication： 副本，是 Kafka 保证数据高可用的方式，Kafka **同一 Partition 的数据可以在多 Broker 上存在多个副本**，通常只有**主副本对外提供读写服务**，当主副本所在 broker 崩溃或发生网络异常，Kafka 会在 Controller 的管理下会重新选择新的 Leader 副本对外提供读写服务。

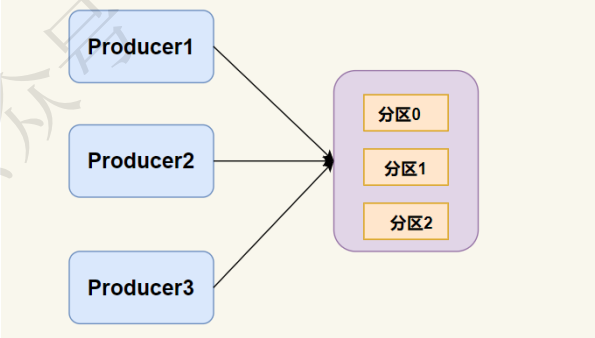
（10）Record： 实际写入 Kafka 中并可以被读取的消息记录。每个 record 包含了 `key`、`value` 和 `timestamp`。

（11）Leader: 每个分区多个副本的 "主" leader,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。

（12）follower: 每个分区多个副本中的"从" follower,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 follow 会成为新的 leader。

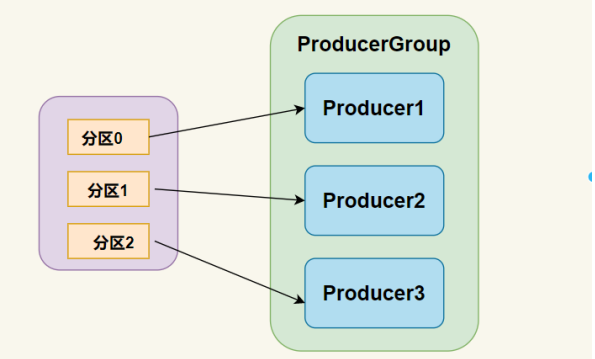
★ 3、发布订阅的消息系统那么多，为啥选择 kafka?

(1) 多个生产者。



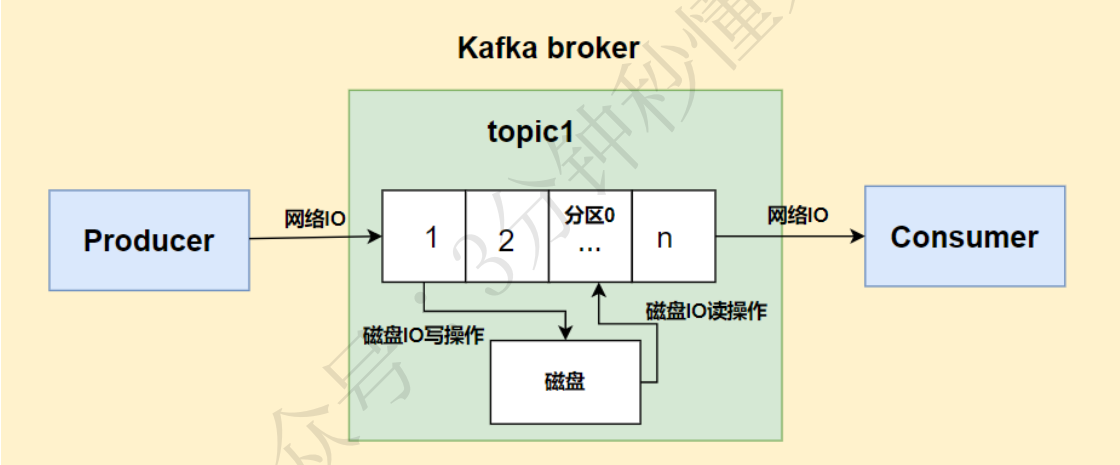
KafKa 可以无缝地支持多个生产者，不管客户端使用一个主题，还是多个主题。**Kafka 适合从多个前端系统收集数据**，并以统一的格式堆外提供数据。

（2）多个消费者



Kafka 支持多个消费者从一个单独的消息流中读取数据，并且消费者之间互不影响。这与其他队列系统不同，其他队列系统一旦被客户端读取，其他客户端就不能 再读取它。并且**多个消费者可以组成一个消费者组，他们共享一个消息流，并保证消费者组对每个给定的消息只消费一次**。

（3）基于磁盘的数据存储



Kafka 允许消费者非实时地读取消息，原因在于 Kafka 将消息提交到磁盘上，设置了保留规则进行保存，无需担心消息丢失等问题。

（4）伸缩性

可扩展多台 broker。用户可以先使用单个 broker，到后面可以扩展到多个 broker

（5）高性能

Kafka 可以轻松处理百万千万级消息流，同时还能保证 **亚秒级** 的消息延迟。

⋆ 4、kafka 如何做到高吞吐量和性能的？

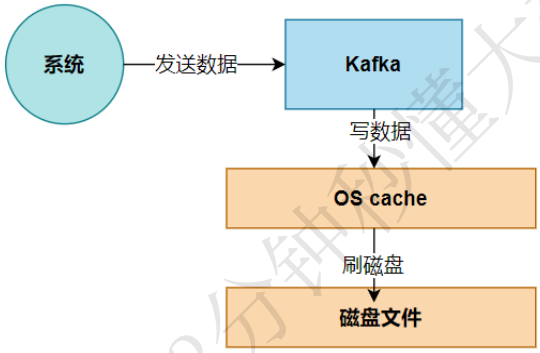
kafka 实现高吞吐量和性能，主要通过以下几点：

1、页缓存技术

Kafka 是基于 **操作系统** 的页缓存来实现文件写入的。

操作系统本身有一层缓存，叫做 **page cache**，是在 **内存里的缓存**，我们也可以称之为 **os cache**，意思就是操作系统自己管理的缓存。

Kafka 在写入磁盘文件的时候，可以直接写入这个 os cache 里，也就是仅仅写入内存中，接下来由操作系统自己决定什么时候把 os cache 里的数据真的刷入磁盘文件中。通过这一个步骤，就可以将磁盘文件**写性能**提升很多了，因为其实这里相当于是在写内存，不是在写磁盘，原理图如下：



2、磁盘顺序写

另一个主要功能是 kafka 写数据的时候，是以磁盘顺序写的方式来写的。也就是说，**仅仅将数据追加到文件的末尾**，不是在文件的随机位置来修改数据。

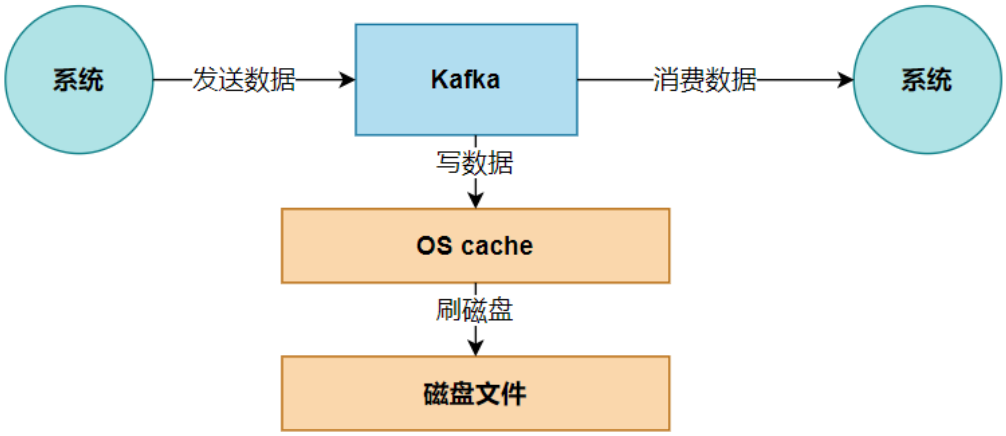
普通的机械磁盘如果你要是随机写的话，确实性能极差，也就是随便找到文件的某个位置来写数据。

但是如果你是 **追加文件末尾** 按照顺序的方式来写数据的话，那么这种磁盘顺序写的性能基本上可以跟写内存的性能相差无几。

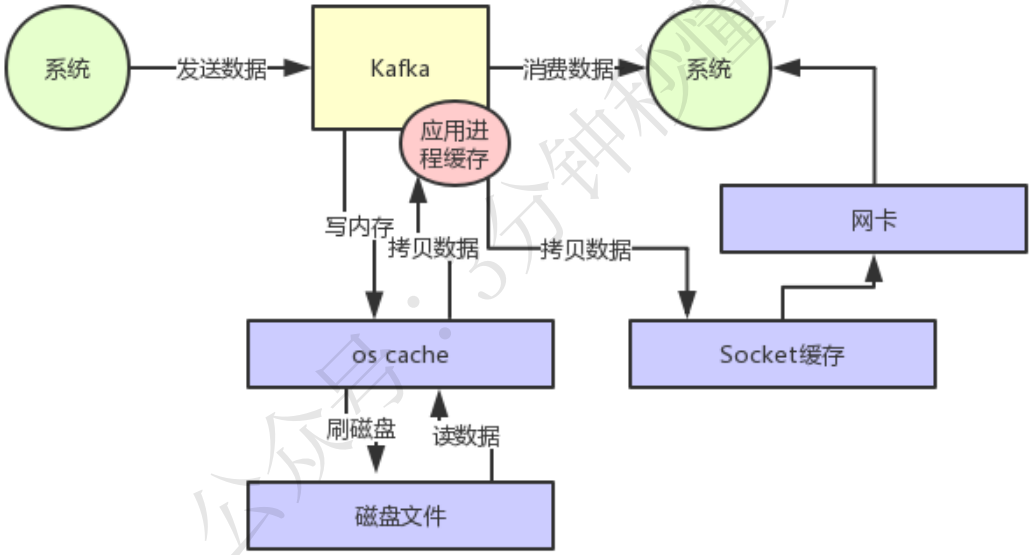
基于上面两点，kafka 就实现了写入数据的超高性能。

3、零拷贝

大家应该都知道，从 Kafka 里经常要消费数据，那么消费的时候实际上就是要从 kafka 的**磁盘文件**里**读取某条数据**然后发送给下游的消费者，如下图所示。



那么这里如果频繁的从磁盘读数据然后发给消费者，会增加两次没必要的拷贝，如下图：

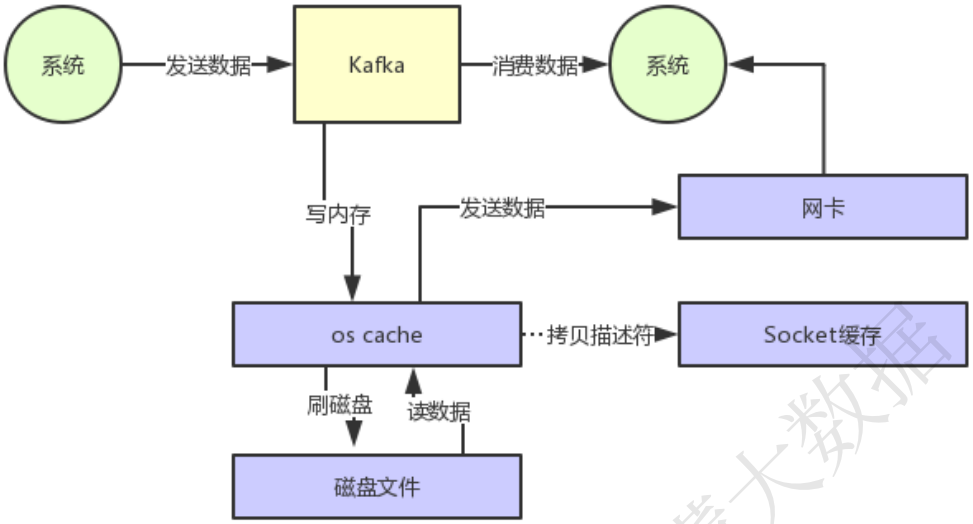


一次是从操作系统的 cache 里拷贝到应用进程的缓存里，接着又从应用程序缓存里拷贝回操作系统的 Socket 缓存里。

而且为了进行这两次拷贝，中间还发生了好几次上下文切换，一会儿是应用程序在执行，一会儿上下文切换到操作系统来执行。所以这种方式来读取数据是比较消耗性能的。

Kafka 为了解决这个问题，在读数据的时候是引入零拷贝技术。

也就是说，直接让操作系统的 cache 中的数据发送到**网卡**后传输给下游的消费者，**中间跳过了两次拷贝数据的步骤**，Socket 缓存中仅仅会拷贝一个描述符过去，不会拷贝数据到 Socket 缓存，如下图所示：



通过 **零拷贝技术**，就不需要把 os cache 里的数据拷贝到应用缓存，再从应用缓存拷贝到 Socket 缓存了，两次拷贝都省略了，所以叫做零拷贝。

对 Socket 缓存仅仅就是拷贝数据的描述符过去，然后数据就直接从 os cache 中发送到网卡上去了，**这个过程大大的提升了数据消费时读取文件数据的性能**。

Kafka 从磁盘读数据的时候，会先看看 os cache 内存中是否有，如果有的话，其实读数据都是直接读内存的。

kafka 集群经过良好的调优，数据直接写入 os cache 中，然后读数据的时候也是从 os cache 中读。相当于 Kafka 完全基于内存提供数据的写和读了，所以这个整体性能会极其的高。

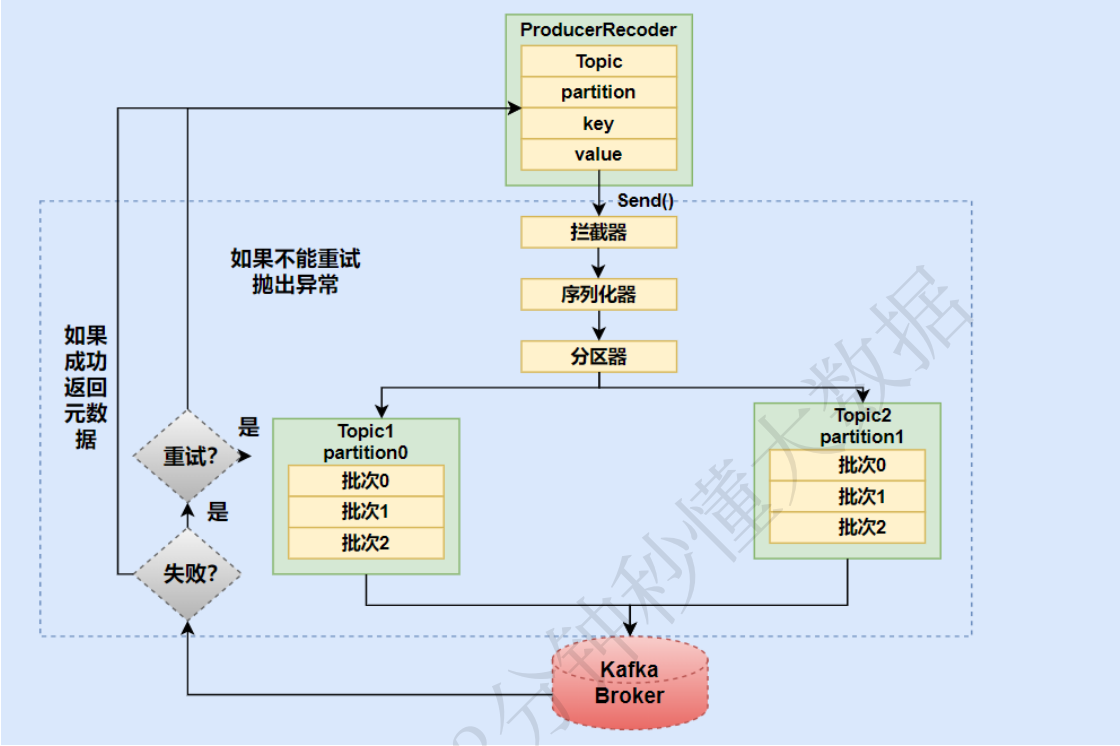
★ 5、 kafka 和 zookeeper 之间的关系

kafka 使用 **zookeeper** 来保存集群的元数据信息和消费者信息(偏移量)，没有 zookeeper，kafka 是工作不起来。 在 `zookeeper` 上会有一个专门用来进行 `Broker` 服务器列表记录的点，节点路径为`/brokers/ids`。

每个 Broker 服务器在启动时，都会到 Zookeeper 上进行注册，即创建 `/brokers/ids/[0-N]` 的节点，然后写入 IP，端口等信息，**Broker 创建的是临时节点**，所以一旦 Broker 上线或者下线，对应 Broker 节点也就被删除了，因此可以通过 zookeeper 上 Broker 节点的变化来动态表征 Broker 服务器的可用性。

⋆ 6、生产者向 Kafka 发送消息的执行流程介绍一下？

如下图所示：



（1）生产者要往 Kafka 发送消息时，需要创建 **ProducerRecoder**,代码如下：

```
ProducerRecord<String,String> record
      = new ProducerRecoder<>("CostomerCountry","Precision Products","F
rance");
      try{
      producer.send(record);
      }catch(Exception e){
        e.printStackTrace();
      }
```

（2）**ProducerRecoder** 对象会包含目标 **topic**，**分区内容**，以及指定的 **key** 和 **value**,在发送 ProducerRecoder 时，生产者会先把键和值对象序列化成字节数组，然后在网络上传输。

（3）生产者在将消息发送到某个 Topic，需要经过**拦截器**、**序列化器**和**分区器**（Partitioner）。

（4）如果消息 **ProducerRecord** 没有指定 partition 字段，那么**就需要依赖分区器**，根据 key 这个字段来计算 partition 的值。**分区器的作用就是为消息分配分区**。

1. 若没有指定分区，且消息的 key 不为空，则使用 murmur 的 Hash 算法（非加密型 Hash 函数，具备高运算性能及低碰撞率）来计算分区分配。

2. 若没有指定分区，且消息的 key 也是空，则用**轮询**的方式选择一个分区。

（5）分区选择好之后，会将消息添加到一个记录批次中，这个批次的所有消息都会被发送到相同的 **Topic** 和 **partition** 上。然后会有一个独立的线程负责把这些记录批次发送到相应的 broker 中。

（6）**broker** 接收到 Msg 后，会作出一个响应。如果成功写入 Kafka 中，就返回一个 **RecordMetaData** 对象，它包含 `Topic` 和 `Partition` 信息，以及记录在分区的 `offset`。

（7）若写入失败，就返回一个错误异常，生产者在收到错误之后尝试重新发送消息，几次之后如果还失败，就返回错误信息。

⋆ 7、kafka 如何保证对应类型的消息被写到相同的分区？

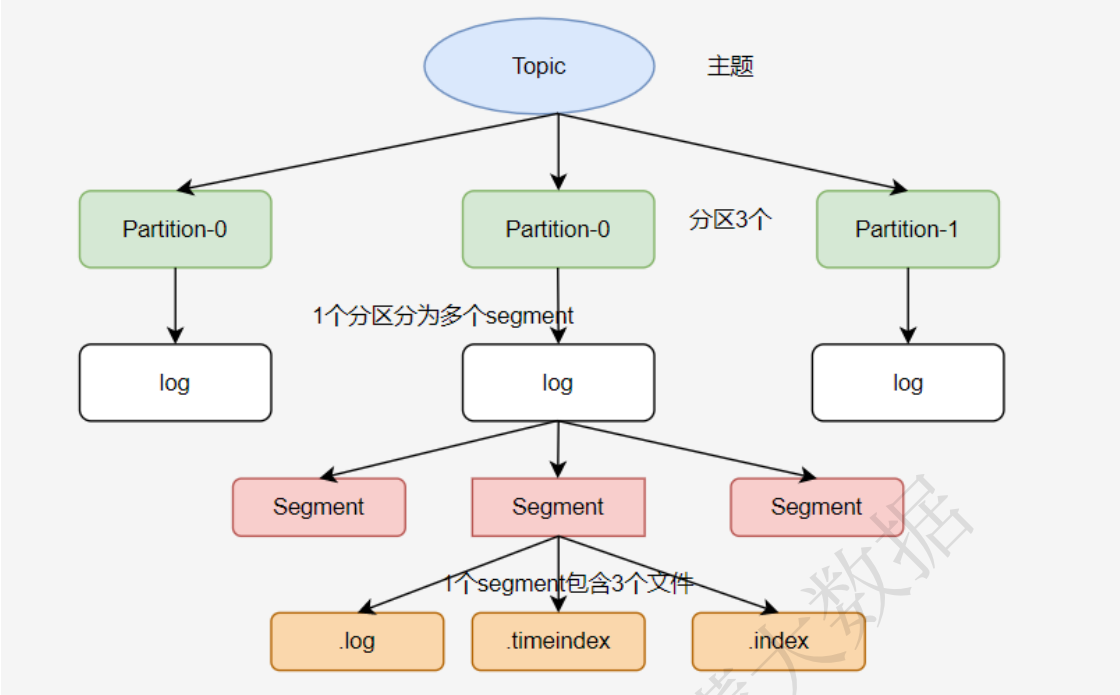
通过 **消息键** 和 **分区器** 来实现，分区器为键生成一个 `offset`，然后使用 offset 对主题分区进行取模，为消息选取分区，这样就可以保证包含同一个键的消息会被写到同一个分区上。

1. 如果 `ProducerRecord` 没有指定分区，且消息的 `key` 不为空，则使用 `Hash` 算法（非加密型 Hash 函数，具备高运算性能及低碰撞率）来计算分区分配。

2. 如果 ProducerRecord 没有指定分区，且消息的 `key` 也是空，则用 **轮询** 的方式选择一个分区。

⋆ 8、kafka 文件存储机制了解吗？

如下图所示：



在 Kafka 中，一个 Topic 会被分割成多个 Partition，而 Partition 由多个更小的 Segment 的元素组成。

一个 Partition 下会包含下图的一些文件，由 `log`、`index`、`timeindex` 三个文件组成一个 `Segment`，**而文件名中的（0）表示的是一个 Segment 的起始 Offset**。

Kafka 会根据 `log.segment.bytes` 的配置来决定单个 Segment 文件（log）的大小，当写入数据达到这个大小时就会创建新的 Segment 。

（1）log 文件解析示意图：

log文件解析示意图

baseOffset	batchLength	...	records
18			record0 record1 record2
...			
27			record0 record1

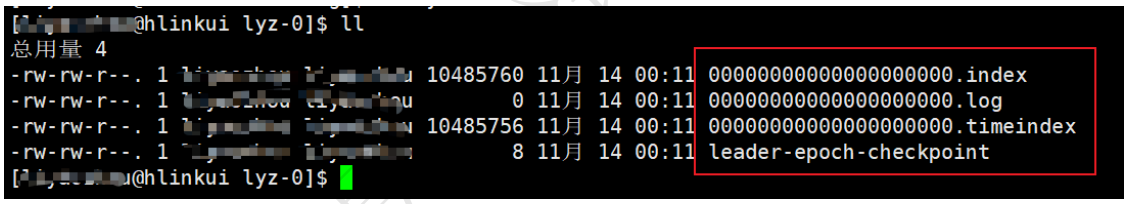
（2）index 文件解析示意图：

index文件解析示意图

offset	position
9	xxx
...	...
21	xxx

（3）timeindex 文件解析示意图：

log、index、timeindex 中存储的都是二进制的数据（ **log 中存储的是 BatchRecords 消息内容，而 index 和 timeindex 分别是一些索引信息**。）



举例：现在创建一个 lyz topic，三个分区，一个副本。

```
# 创建一个 topic
[root@hlinkui bin]$ ./kafka-topics.sh --create --bootstrap-server 192.1
68.244.129:9092 \
--replication-factor 1 --partitions 3 --topic lyz

# 查看创建的 topic 描述信息：
./kafka-topics.sh --describe --bootstrap-server \
192.168.244.129:9092  --topic lyz
```

截图如下：

```
[ @hlinkui bin]$ ./kafka-topics.sh --describe --bootstrap-server 192.168.244.129:9092  --topic lyz
Topic:lyz       PartitionCount:3        ReplicationFactor:1     Configs:segment.bytes=1073741824
        Topic: lyz      Partition: 0    Leader: 0       Replicas: 0     Isr: 0
        Topic: lyz      Partition: 1    Leader: 0       Replicas: 0     Isr: 0
        Topic: lyz      Partition: 2    Leader: 0       Replicas: 0     Isr: 0
[ @hlinkui bin]$
```

那么，其数据文件的目录结构如下所示，3 个分区对应 3 个文件夹，文件夹命名 topic-分区序号：即 lyz-0,lyz-1,lyz-2.

```
drwxrwxr-x. 2  141 11月 14 00:11 lyz-0
drwxrwxr-x. 2  141 11月 14 00:11 lyz-1
drwxrwxr-x. 2  141 11月 14 00:11 lyz-2
```

进入其中一个分区的文件夹中，会有 3 种类型文件，

```
10485760 11月 13 23:59 00000000000000016062.index
       0 10月  9 11:15 00000000000000016062.log
10485756 11月 13 23:59 00000000000000016062.timeindex
      12 11月 13 23:59 leader-epoch-checkpoint
```

具体情况请看这个链接：
https://www.cnblogs.com/hzmark/p/kafka_message_format.html

★ 9、如何根据 offset 找到对应的 Message？

通过 Offset 从存储层中获取 Message 大致分为两步：

1. 第一步，根据 Offset 找到所属的 Segment 文件
2. 第二步，从 Segment 中获取对应 Offset 的消息数据

第一步可以直接根据 Segment 的文件名进行查找（上面已经介绍了 Segment 的文件面就是它包含的数据的起始 Offset ）。

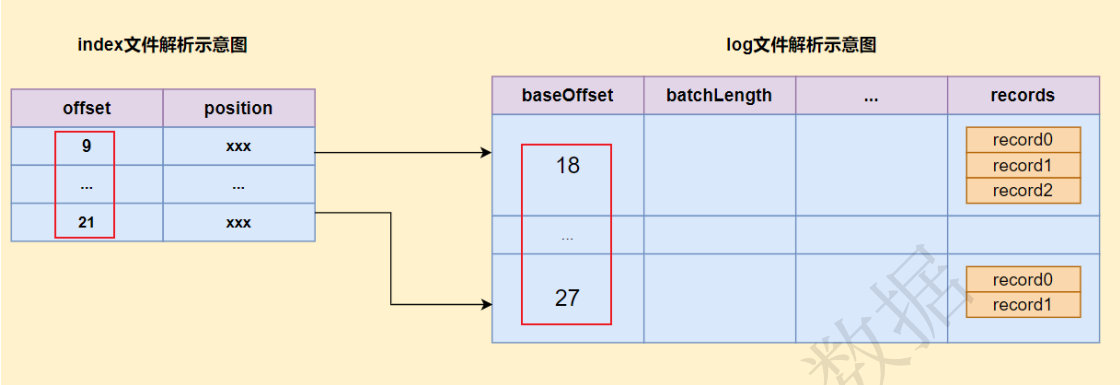
第二步则需要一些索引信息来快速定位目标数据在 Segment 中的位置，否则就要读取整个 Segment 文件了，这里需要的索引信息就是上面的 index 文件存储的内容。

索引文件中包含多个索引条目，每个条目表示数据文件中一条 Message 的索引。索引包含两个部分（均为 4 个字节的数字），分别为相对 offset 和 position, 如下内容所示：

```
[root@hlinkui bin]$ ./kafka-run-class.sh  kafka.tools.DumpLogSegments --files \
../kafkaLog/lyz-0/00000000000000000000.index  --print-data-log
```

```
offset: 9 position: 13713
offset: 15 position: 27799
offset: 21 position: 43149
offset: 27 position: 58432
```

如下图所示：



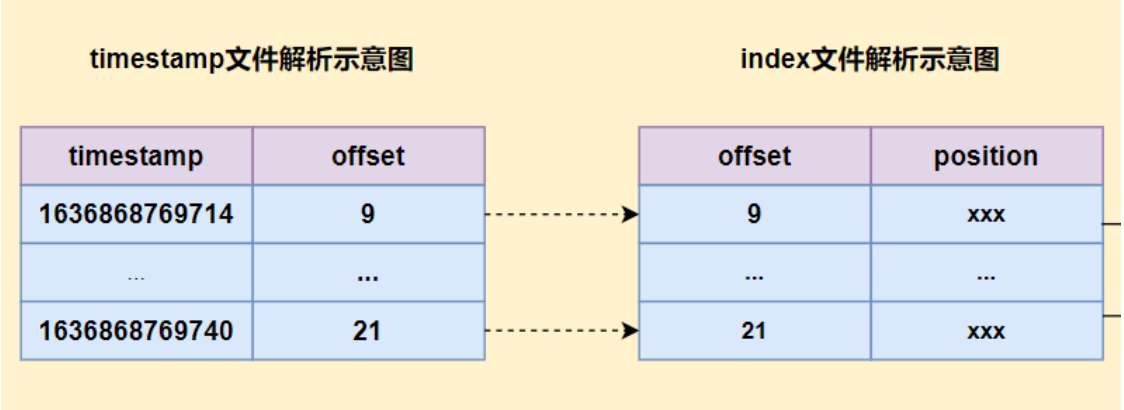
index 文件中存储的是 Offset 和 Position（Offset 对应的消息在 log 文件中的偏移量）的对应关系，这样当有 Offset 时可以快速定位到 Position 读取 BatchRecord，然后再从 BatchRecord 中获取某一条消息。

比如上述 Offset21 会被定位到 27 这个 BatchRecord，然后再从这个 BatchRecord 中取出第二个 Record（27 这个 BatchRecord 包含了 27 、28 两个 Record）。

Kafka 并不会为每个 Record 都保存一个索引，而是根据 log.index.interval.bytes 等配置 **构建稀疏索引信息**。

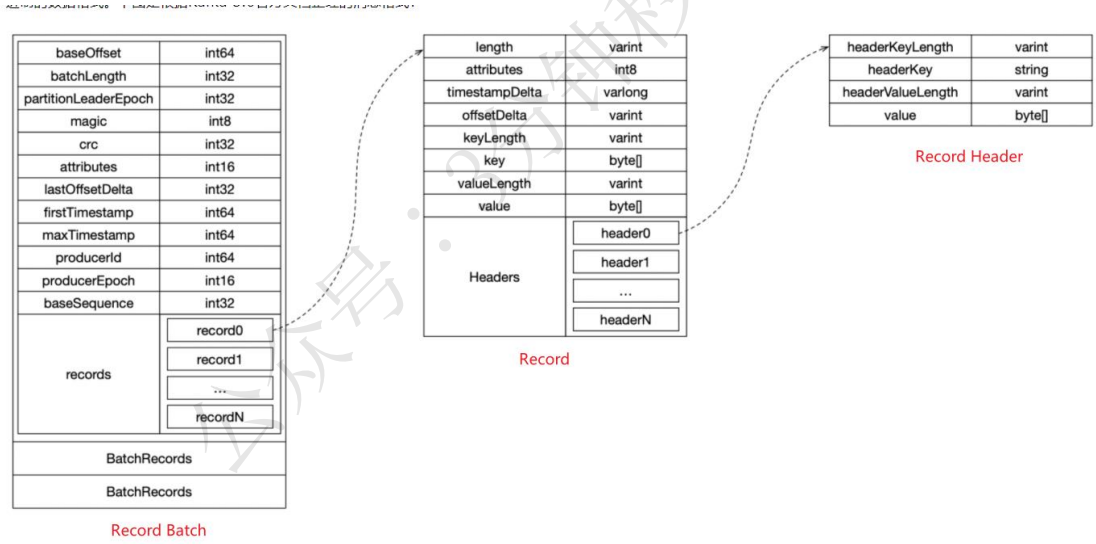
Kafka 中还维护了 timeindex，保存了 Timestamp 和 Offset 的关系，在一些场景需要根据 timestamp 来定位消息。**timeindex 中的一个（ timestampX，offsetY ）元素的含义是所有创建时间大于 timestampX 的消息的 Offset 都大于 offsetY**。

查找方式如下图所示：



⋆ 10、 Producer 发送的一条 message 中包含哪些信息？

消息由 可变长度 的 **报头**、可变长度的 **不透明密钥字节数组**和 可变长度的 **不透明值字节数组** 组成。



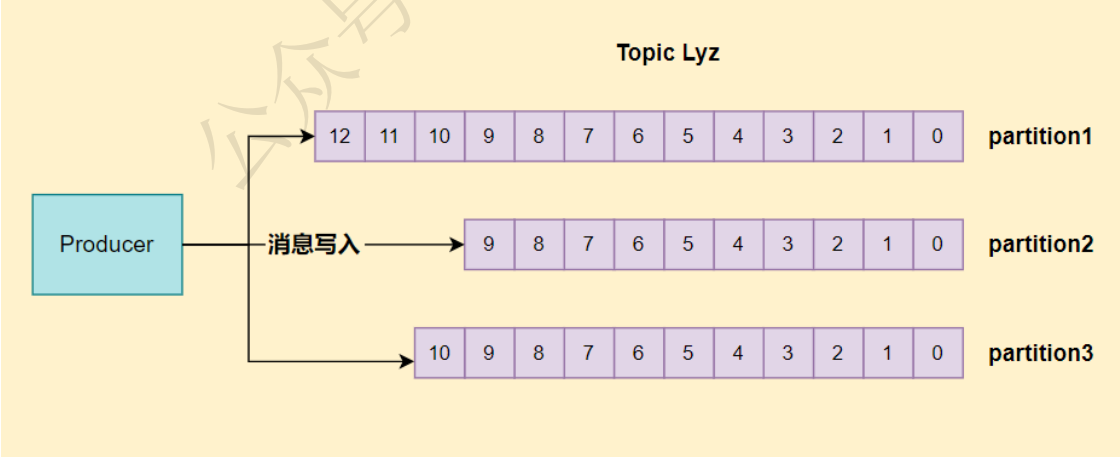
RecordBatch 是 **Kafka** 数据的存储单元，一个 `RecordBatch` 中包含多个 `Record`（即我们通常说的一条消息）。RecordBatch 中各个字段的含义如下：

字段名	含义
baseOffset	这批消息的起始Offset
partitionLeaderEpoch	用于Partition的Recover时保护数据的一致性，具体场景可以见KIP101
batchLength	BatchRecords的长度
magic	魔数字段，可以用于拓展存储一些信息，当前3.0版本的magic是2
crc	crc校验码，包含从attributes开始到BatchRecords结束的数据的校验码
attributes	int16，其中bit0~2中包含了使用的压缩算法，bit3是timestampType，bit4表示是否失误，bit5表示是否是控制指令，bit6~15暂未使用
lastOffsetDelta	BatchRecords中最后一个Offset，是相对baseOffset的值
firstTimestamp	BatchRecords中最小的timestamp
maxTimestamp	BatchRecords中最大的timestamp
producerId	发送端的唯一ID，用于做消息的幂等处理
producerEpoch	发送端的Epoch，用于做消息的幂等处理
baseSequence	BatchRecords的序列号，用于做消息的幂等处理
records	具体的消息内容

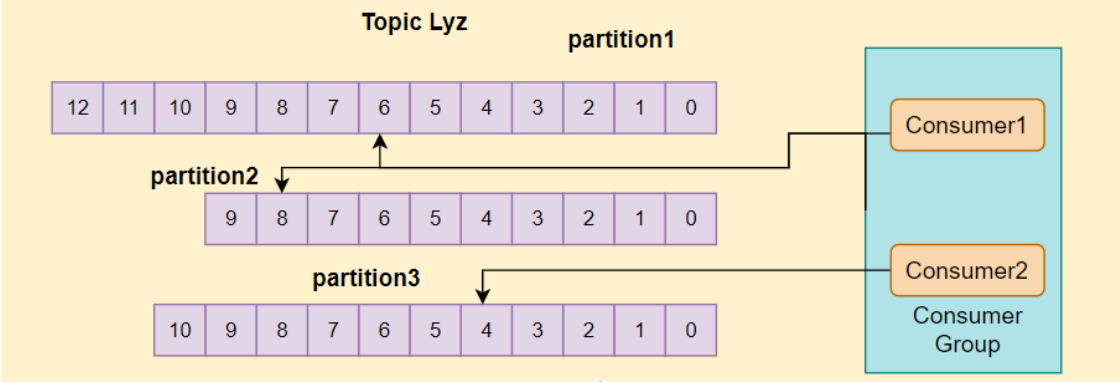
一个 RecordBatch 中可以包含多条消息，即上图中的 Record，而每条消息又可以包含多个 Header 信息，Header 是 Key-Value 形式的。

⋆ 11、kafka 如何实现消息是有序的？

生产者：通过分区的 leader 副本负责数据以先进先出的顺序写入，来保证消息顺序性。



消费者：同一个分区内的消息只能被一个 group 里的一个消费者消费，保证分区内消费有序。



kafka 每个 partition 中的消息在写入时都是有序的，消费时， 每个 partition 只能被每一个消费者组中的一个消费者消费，保证了消费时也是有序的。

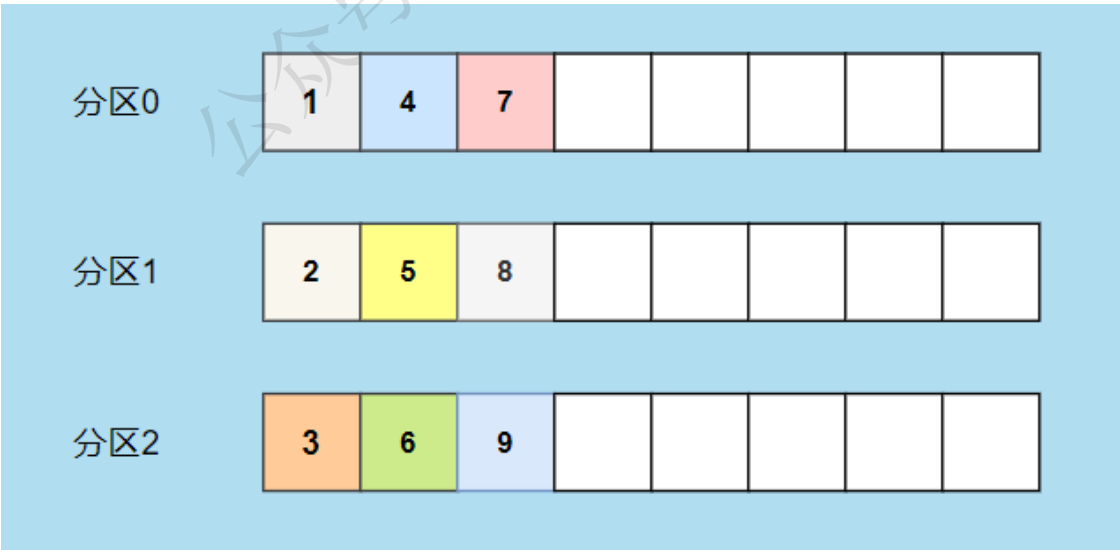
整个 kafka 不保证有序。**如果为了保证 kafka 全局有序，那么设置一个生产者，一个分区，一个消费者。**

⋆ 12、kafka 有哪些分区算法？

kafka 包含三种分区算法：

（1）轮询策略

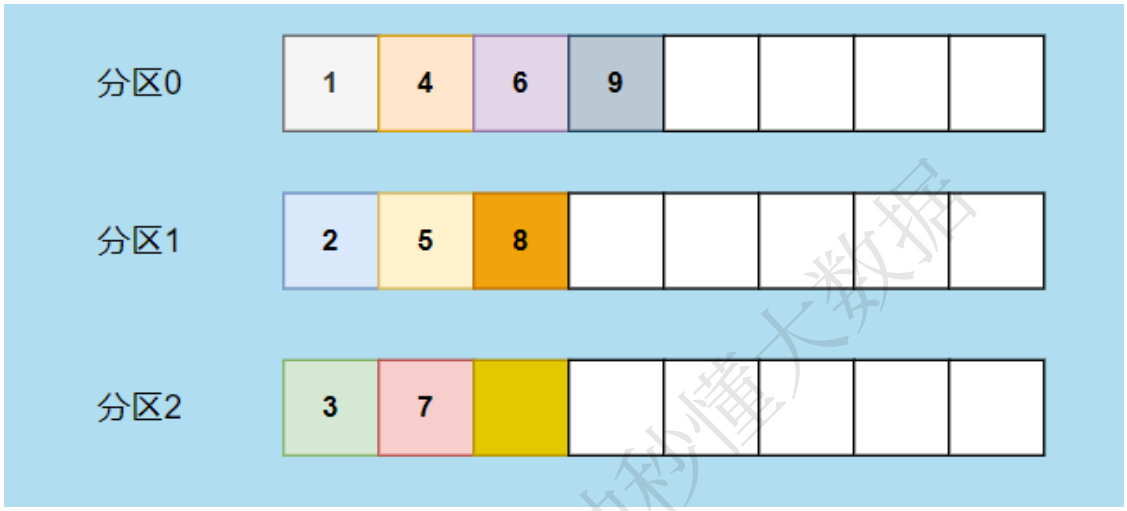
也称 Round-robin 策略，即顺序分配。比如一个 topic 下有 3 个分区，那么第一条消息被发送到分区 0，第二条被发送到分区 1，第三条被发送到分区 2，以此类推。当生产第四条消息时又会重新开始。



轮询策略是 kafka java 生产者 API 默认提供的分区策略。轮询策略有非常优秀的负载均衡表现，它总是能保证消息最大限度地被平均分配到所有分区上，故默认情况下它是最合理的分区策略，也是平时最常用的分区策略之一。

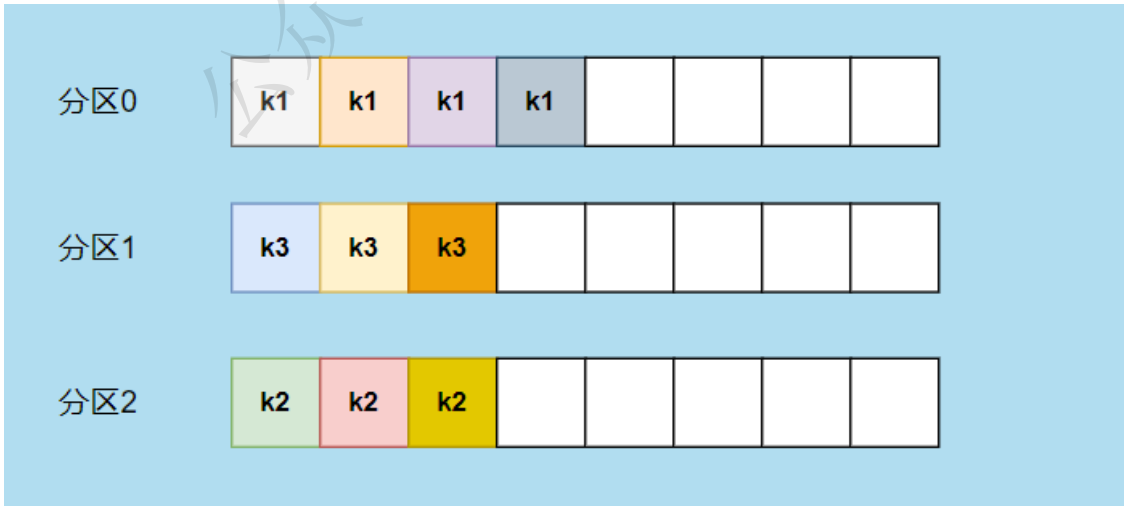
（2）随机策略

也称 Randomness 策略。所谓随机就是我们随意地将消息放置在任意一个分区上，如下图：



（3）按 key 分配策略

kafka 允许为每条消息定义消息键，简称为 key。一旦消息被定义了 key，那么你就可以保证同一个 key 的所有消息都进入到相同的分区里面，由于每个分区下的消息处理都是有顺序的，如下图所示：



⋆ 13、说说 kafka 的默认消息保留策略？

broker 默认的消息保留策略分为两种：

1. 日志片段通过 log.segment.bytes 配置（默认是 1GB）
2. 日志片段通过 log.segment.ms 配置 （默认 7 天）

⋆ 14、kafka 如何实现单个集群间的消息复制？

Kafka 消息负责机制只能在单个集群中进行复制，不能在多个集群之间进行。

kafka 提供了一个叫做 **MirrorMaker** 的核心组件，该组件包含一个生产者和一个消费者，两者之间通过一个队列进行相连，当消费者从一个集群读取消息，生产者把消息发送到另一个集群。

⋆ 15、Kafka 消息确认(ack 应答)机制了解吗？

为保证 producer 发送的数据，能可靠的达到指定的 topic ,Producer 提供了消息确认机制。 生产者往 Broker 的 topic 中发送消息时，**可以通过配置来决定有几个副本收到这条消息才算消息发送成功**。可以在定义 Producer 时通过 **acks** 参数指定，这个参数支持以下三种值：

（1）acks = 0：producer 不会等待任何来自 broker 的响应。

特点：低延迟，高吞吐，数据可能会丢失。

如果当中出现问题，导致 broker 没有收到消息，那么 producer 无从得知，会造成消息丢失。

（2）acks = 1（默认值）：只要集群中 partition 的 Leader 节点收到消息，生产者就会收到一个来自服务器的成功响应。

如果在 follower 同步之前，leader 出现故障，将会丢失数据。

此时的吞吐量主要取决于使用的是 **同步发送** 还是 **异步发送** ，吞吐量还受到发送中消息数量的限制，例如 producer 在收到 broker 响应之前可以发送多少个消息。

（3）acks = -1：只有当所有参与复制的节点全部都收到消息时，生产者才会收到一个来自服务器的成功响应。

这种模式是最安全的，可以保证不止一个服务器收到消息，就算有服务器发生崩溃，整个集群依然可以运行。

根据实际的应用场景，选择设置不同的 acks，以此保证数据的可靠性。

另外，Producer 发送消息还可以选择同步或异步模式,如果设置成异步，虽然会极大的提高消息发送的性能，但是这样会增加丢失数据的风险。如果需要**确保消息的可靠性**，必须将 producer.type 设置为 sync。

```
#同步模式
producer.type=sync
#异步模式
producer.type=async
```

⋆16、说一下什么是副本？

kafka 为了保证数据不丢失，从 `0.8.0` 版本开始引入了分区副本机制。在创建 topic 的时候指定 `replication-factor`,默认副本为 3 。

副本是相对 partition 而言的，一个分区中包含一个或多个副本，其中一个为 `leader` 副本，其余为 `follower` 副本，各个副本位于不同的 `broker` 节点中。

所有的读写操作都是经过 Leader 进行的，同时 follower 会定期地去 leader 上复制数据。当 Leader 挂掉之后，其中一个 follower 会重新成为新的 Leader。**通过分区副本，引入了数据冗余，同时也提供了 Kafka 的数据可靠性**。

Kafka 的分区多副本架构是 Kafka 可靠性保证的核心，把消息写入多个副本可以使 Kafka 在发生崩溃时仍能保证消息的持久性。

⋆17、说一下 kafka 的 ISR 机制？

在分区中，所有副本统称为 AR，Leader 维护了一个动态的 in-sync replica(ISR),**ISR 是指与 leader 副本保持同步状态的副本集合**。当然 `leader` 副本本身也是这个集合中的一员。

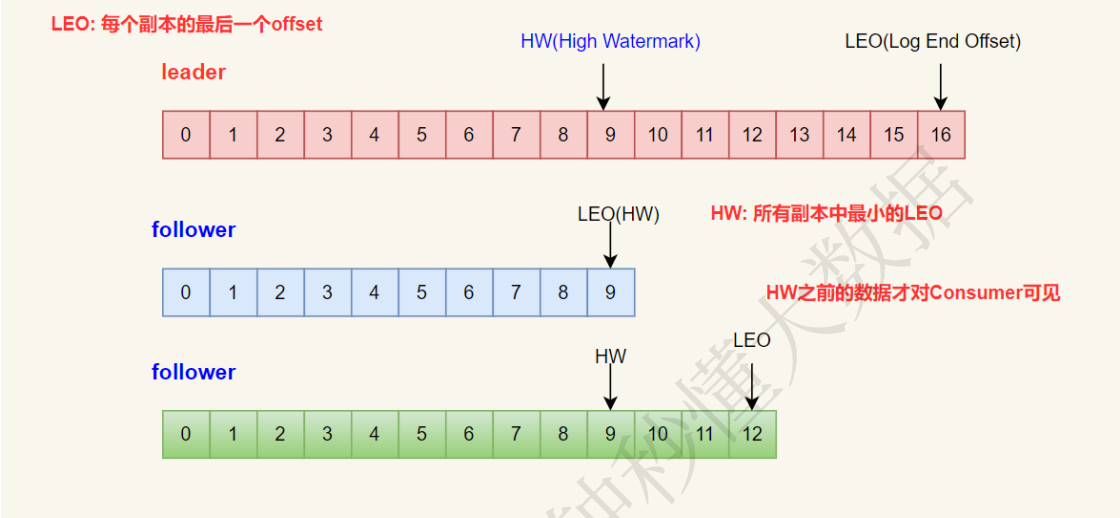
当 ISR 中的 follower 完成数据同步之后， leader 就会给 follower 发送 ack ,如果其中一个 follower 长时间未向 leader 同步数据，该 follower 将会被踢出 ISR 集合，该时间阈值由 replica.log.time.max.ms 参数设定。当 leader 发生故障后，就会从 ISR 集合中重新选举出新的 leader。

⋆18、LEO、HW、LSO、LW 分别代表什么？

- LEO： 是 LogEndOffset 的简称，代表当前日志文件中下一条。
- HW： 水位或水印一词，也可称为高水位（`high watermark`）,通常被用在流式处理领域（`flink、spark`），以表征元素或事件在基于时间层面上的进展。在 kafka 中，水位的概念与时间无关，而是与位置信息相关。严格来说，它表示

的就是位置信息，即位移（offset）。取 partition 对应的 ISR 中最小的 LEO 作为 HW，consumer 最多只能消费到 HW 所在的上一条信息。

- LSO: 是 LastStableOffset 的简称，对未完成的事务而言，LSO 的值等于事务中第一条消息的位置（firstUnstableOffset），对已完成的事务而言，它的值同 HW 相同。
- LW: Low Watermark 低水位，代表 AR 集合中最小的 logStartOffset 值。



⋆19、如何进行 Leader 副本选举？

每个分区的 leader 会维护一个 ISR 集合，ISR 列表里面就是 follower 副本的 Borker 编号，只有“跟得上” Leader 的 follower 副本才能加入到 ISR 里面，这个是通过 replica.lag.time.max.ms 参数配置的。只有 ISR 里的成员才有被选为 leader 的可能。

所以当 Leader 挂掉了，而且 unclean.leader.election.enable=false 的情况下，Kafka 会从 ISR 列表中选择 **第一个** follower 作为新的 Leader，因为这个分区拥有最新的已经 committed 的消息。通过这个可以保证已经 committed 的消息的数据可靠性。

⋆ 20、如何进行 broker Leader 选举？

(1) 在 kafka 集群中，会有多个 broker 节点，集群中第一个启动的 broker 会通过在 zookeeper 中创建临时节点 **/controller** 来让自己成为控制器，其他 broker 启动时也会在 zookeeper 中创建临时节点，但是发现节点已经存在，所以它们会收到一

个异常，意识到控制器已经存在，那么就会在 zookeeper 中创建 **watch** 对象，便于它们收到控制器变更的通知。

(2) 如果集群中有一个 broker 发生异常退出了，那么控制器就会检查这个 broker 是否有分区的副本 leader，如果有那么这个分区就需要一个新的 leader，此时控制器就会去遍历其他副本，决定哪一个成为新的 leader，同时更新分区的 ISR 集合。

(3) 如果有一个 broker 加入集群中，那么控制器就会通过 Broker ID 去判断新加入的 broker 中是否含有现有分区的副本，如果有，就会从分区副本中去同步数据。

(4) 集群中每选举一次控制器，就会通过 zookeeper 创建一个 **controller epoch**，每一个选举都会创建一个更大，包含最新信息的 epoch，如果有 broker 收到比这个 epoch 旧的数据，就会忽略它们，kafka 也通过这个 epoch 来防止集群产生“脑裂”。

⋆ 21、kafka 事务了解吗？

Kafka 在 0.11 版本引入事务支持，事务可以保证 Kafka 在 Exactly Once 语义的基础上，生产和消费可以跨分区和会话，要么全部成功，要么全部失败。

Producer 事务

为了实现跨分区跨会话事务，需要引入一个全局唯一的 Transaction ID,**并将 Producer 获取的 PID 和 Transaction ID 绑定**。这样当 Producer 重启后就可以通过正在进行的 Transaction ID 获取原来的 PID。

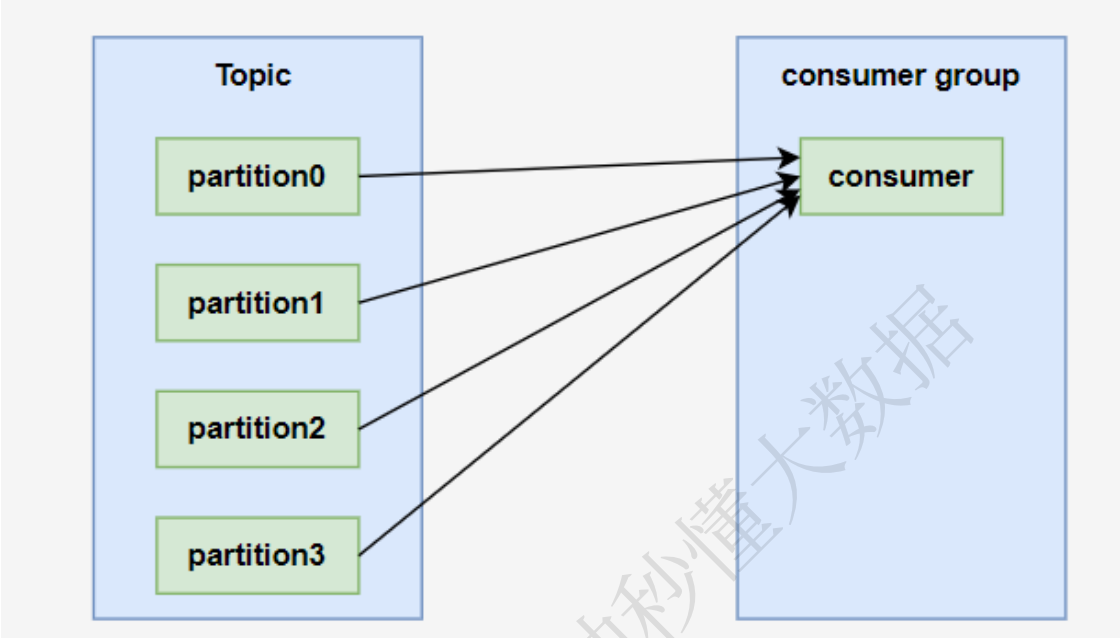
为了管理 Transaction，Kafka 引入了一个新的组件 Transaction Coordinator。**Producer 就是通过和 Transaction Coordinator 交互获得 Transaction ID 对应的任务状态**。Transaction Coordinator 还负责将事务所有写入 Kafka 的一个内部 Topic，这样即使整个服务重启，由于事务状态得到保存，进行中的事务状态可以得到恢复，从而继续进行。

Consumer 事务

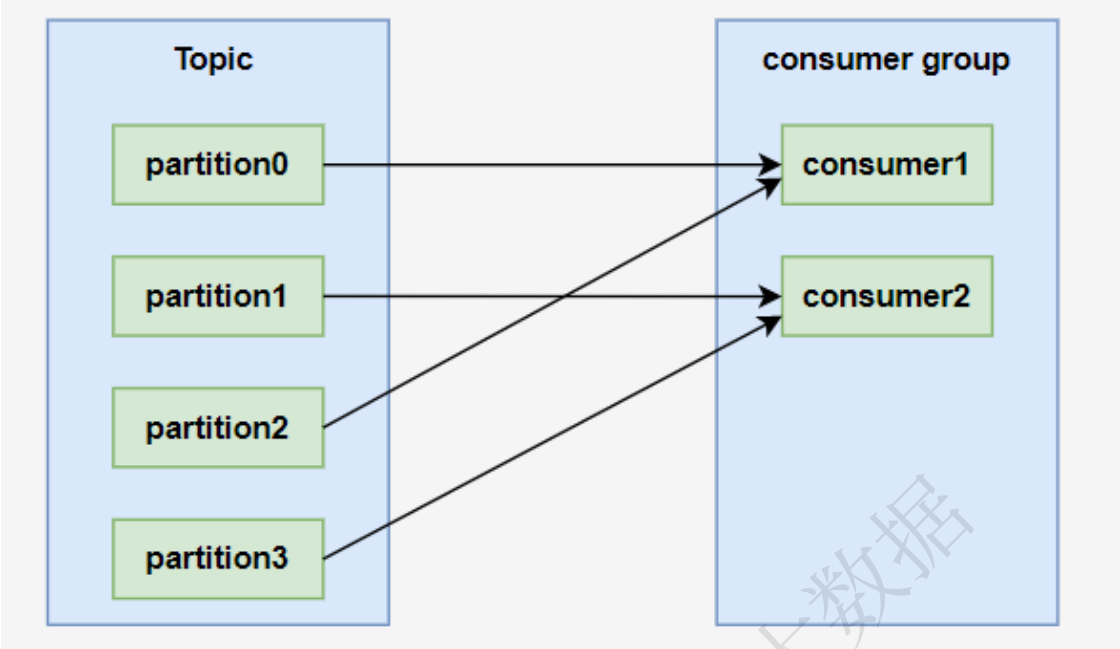
上述事务机制主要是从 Producer 方面考虑，对于 Consumer 而言，事务的保证就会相对较弱，尤其是无法保证 Commit 的信息被精确消费。这是由于 Consumer 可以通过 offset 访问任意信息，而且不同的 Segment File 生命周期不同，同一事务的消息可能会出现重启后被删除的情况。

⋆ 22、kafka 的消费者组跟分区之间有什么关系？

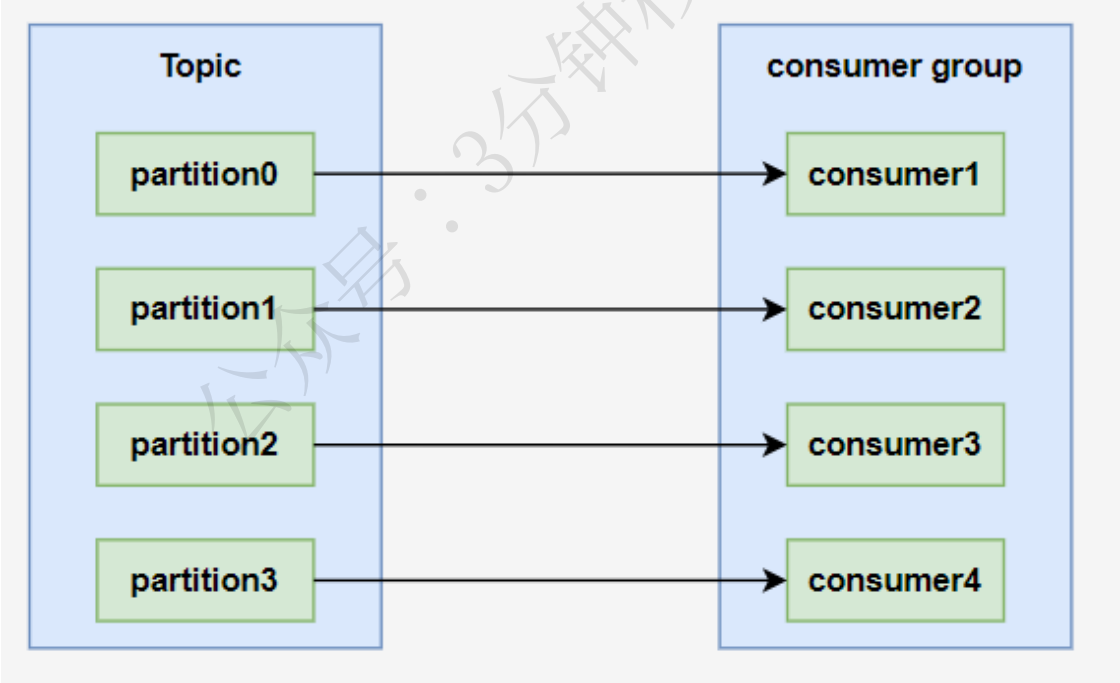
(1) 在 kafka 中，通过消费者组管理消费者，假设一个主题中包含 4 个分区，在一个消费者组中只要一个消费者。那消费者将收到全部 4 个分区的消息。



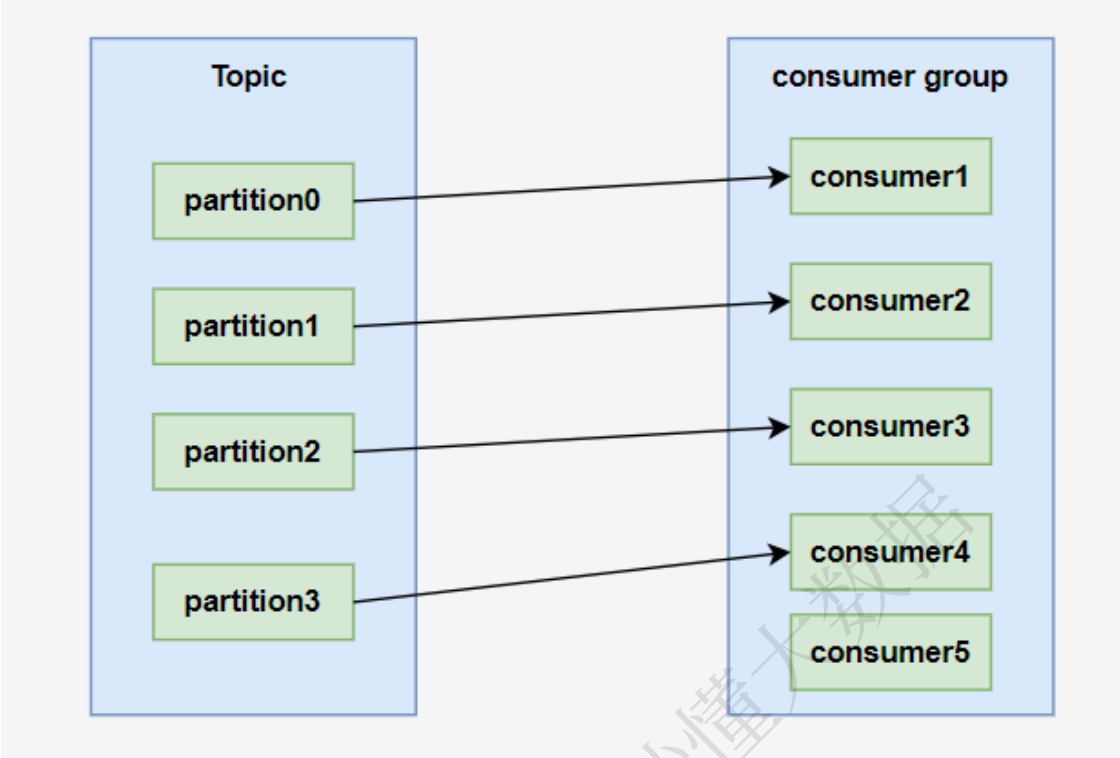
(2) 如果存在两个消费者，那么四个分区将根据分区分配策略分配个两个消费者，如下图所示：



（3）如果存在四个消费者，将平均分配，每个消费者消费一个分区。



（4）如果存在 5 个消费者，就会出现消费者数量多于分区数量，那么多余的消费者将会被闲置，不会接收到任何信息。



⋆ 23、如何保证每个应用程序都可以获取到 Kafka 主题中的所有消息，而不是部分消息？

为每个应用程序创建一个消费者组，然后往组中添加消费者来伸缩读取能力和处理能力，每个群组消费主题中的消息时，互不干扰。

⋆ 24、如何实现 kafka 消费者每次只消费指定数量的消息？

写一个队列，把 consumer 作为队列类的一个属性，然后增加一个消费计数的计数器，当到达指定数量时，关闭 consumer。

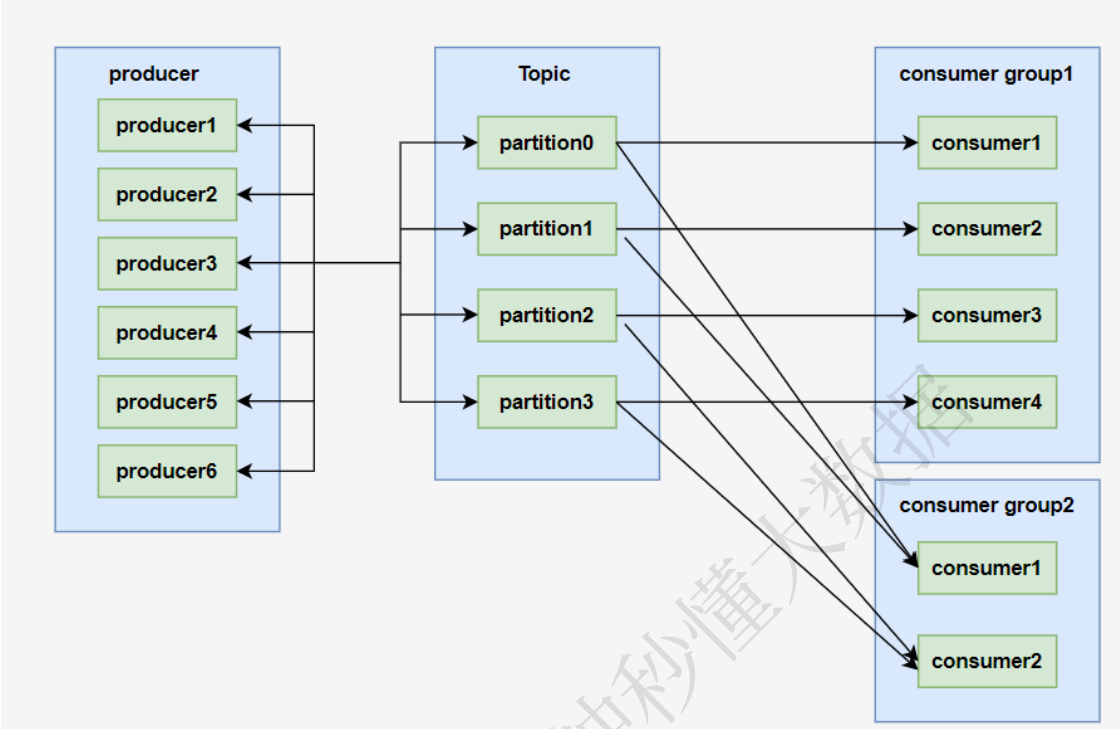
⋆ 25、kafka 如何实现多线程的消费？

kafka 允许同组的多个 partition 被一个 consumer 消费，但不允许一个 partition 被同组的多个 consumer 消费。

实现多线程步骤如下：

1. 生产者随机分区提交数据(自定义随机分区)。

2. 消费者修改单线程模式为多线程，在消费方面得注意，得遍历所有分区，否则还是只消费了一个区。



★ 26、 kafka 消费支持几种消费模式？

kafka 消费消息时支持三种模式：

- at most once 模式 最多一次。保证每一条消息 commit 成功之后，再进行消费处理。消息可能会丢失，但不会重复。
- at least once 模式 至少一次。保证每一条消息处理成功之后，再进行 commit。消息不会丢失，但可能会重复。
- exactly once 模式 精确传递一次。将 offset 作为唯一 id 与消息同时处理，并且保证处理的原子性。消息只会处理一次，不丢失也不会重复。但这种方式很难做到。

kafka 默认的模式是 at least once ，但这种模式可能会产生重复消费的问题，所以在业务逻辑必须做幂等设计。

在业务场景保存数据时使用了 INSERT INTO ...ON DUPLICATE KEY UPDATE 语法，不存在时插入，存在时更新，是天然支持幂等性的。

★ 27、kafka 如何保证数据的不重复和不丢失？

- exactly once 模式 精确传递一次。将 offset 作为唯一 id 与消息同时处理，并且保证处理的原子性。消息只会处理一次，不丢失也不会重复。但这种方式很难做到。

kafka 默认的模式是 at least once ，但这种模式可能会产生重复消费的问题，所以在业务逻辑必须做幂等设计。

使用 exactly Once + 幂等操作，可以保证数据不重复，不丢失。

★ 28、kafka 是如何清理过期数据的？

kafka 将数据持久化到了硬盘上，允许你配置一定的策略对数据清理，清理的策略有两个，**删除和压缩**。

数据清理的方式

1、删除

`log.cleanup.policy=delete` 启用删除策略

直接删除，删除后的消息不可恢复。可配置以下两个策略：

```
#清理超过指定时间清理：
log.retention.hours=16
#超过指定大小后，删除旧的消息：
log.retention.bytes=1073741824
```

为了避免在删除时阻塞读操作，采用了 `copy-on-write` 形式的实现，删除操作进行时，读取操作的二分查找功能实际是在一个静态的快照副本上进行的，这类似于 Java 的 CopyOnWriteArrayList。

2、压缩

将数据压缩，只保留每个 key 最后一个版本的数据。

首先在 broker 的配置中设置 `log.cleaner.enable=true` 启用 cleaner，这个默认是关闭的。

在 topic 的配置中设置 `log.cleanup.policy=compact` 启用压缩策略。