# 1 Kafka基础概念

## 消息

消息是Kafka的数据单元，由字节数组组成（类似Java中的byte[]）。

### 键

键是消息的一个可选的元数据，也是由字节数组组成。

### 偏移量

偏移量是消息的一个元数据，在消息创建时由Kafka添加到消息中，它是一个不断增加的整数值。

（注意：同个分区中每个消息的偏移量都是唯一的）

## 分区

一个主题被分为多个分区。当生产者向该topic发送消息msg时，如果该消息msg被指定了key，Kafka会将这个key求哈希值，然后对该主题的分区数求余数，这个余数就是该msg会被放到哪个分区中。

（注意：同一个分区中的消息以先入先出的顺序读取。）

（注意：每一个分区在物理上对应一个文件夹，该文件夹的命名方式为：”topicName\_partitionIndex”，partitionIndex从0~该topic的分区数-1）

图1

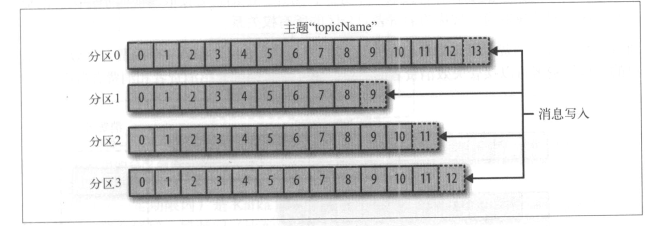
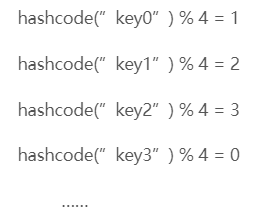


图2



### 一个主题的多个分区横跨多个服务器

一个主题的不同分区可以部署在不同的服务器上，所以说一个主题可以横跨多个服务器。

### 一个分区同时只能被一个消费者读取

一个分区同时只能被一个消费者读取，但一个消费者可以读取多个分区。

### 分区的所有权关系

分区的所有权关系是指：消费者和分区之间的映射关系。

### 分区复制

如果把一个分区分配给多个broker则会发生分区复制。分区复制冗余了分区的消息，但当一个broker失效时，其他broker的分区可以使用。

## 主题

主题就是一个分类。消息类似于表中的一条记录，主题类似于表。

## 生产者

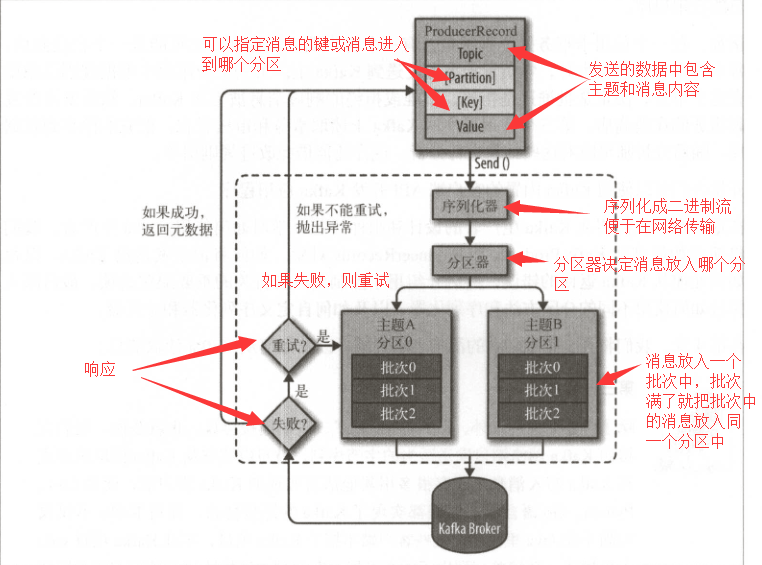
生产者默认情况下，将消息均匀分布在一个主题的所有分区上。

（注意：也可以指定写入某一个分区上）

### 生产者生产消息的流程

1. 创建一个ProducerRecord对象，包含目标主题Topic和消息内容Value。还可以指定分区Partition，和消息的键Key。
2. ProducerRecord对象会被序列化器系列化成二进制流，便于在网络上传输。
3. 分区器会根据指定的分区将消息放入该分区的批次（生产者的内存缓冲区）中，如果没有指定分区，则根据键决定将消息放入哪个分区的批次中。
4. 这个批次的消息由一个单独的线程放到相同的主题和分区中。
5. 如果消息成功写入Kafka，会返回一个RecordMetaData对象，包含了主题和分区信息，以及消息在 分区中的偏移量；如果失败会返回一个错误，生产者会尝试重试。

图1



### 生产者属性

1. bootstrap.servers：指定broker的地址，地址格式是host:port，至少包含2个broker的地址，其中一个宕机后，生产者仍能通过另一个连到集群中。

（注意：不需要配置集群中所有broker的地址，因为生产者会从给定的broker中找到其他broker的信息）

1. key.serializer和value.serializer：指定消息的键和值的序列化器。因为broker接收到的消息的键和值都必须是字节数据，但友好的提供输入可以是Java对象，所以需要提供序列化器将消息的键和值转换成字节数组。

图1



1. arks：指定了必须多少个分区副本收到消息，生产者才认为消息发送成功。
2. buffer.memory：指定了生产者内存缓冲区的大小。
3. comression.type：指定了消息压缩的方式。
4. retries：对服务器返回的临时性错误重发消息。

（一般情况下，生产者会自动重试，所以我们只需要处理不可重试的错误或重试次数超出上限的情况）

1. batch.size：指定了一个批次可使用的内存大小。

（注意：不一定是批次满了才发送，可能半满或只有1个消息也会发送）

1. linger.ms：指定了生产者在发送批次之前等待更多消息进入该批次的时间。当批次满了或这个时间到了，就会发送批次，哪怕批次中只有1个消息。
2. client.id：服务器会用它来识别消息的来源。
3. max.in.flight.requests.per.connection：指定了生产者在收到服务器响应之前可以发送多少个消息。

（注意：如果设成1，则每次发送1个，保证了顺序发送；如果设置成2，如果第一个消息失败，第二个成功，然后第一个重试成功，则顺序就反过来了）

1. timeout.ms，request.timeout.ms，metadata.fetch.timeout.ms：timeout.ms指定了broker等待同步副本返回消息确认的时间；request.timeout.ms指定了生产者发送数据时等待服务器返回响应的时间，metadata.fetch.timeout.ms指定了生产者在获取元数据时等待服务器返回响应的时间。
2. max.block.ms：指定了调用send()或使用partitionsFor()方法获取元数据时生产者的阻塞时间。
3. max.request.size：可以指定单个消息的最大值，也可以指定单个请求里全部消息的总大小。

（注意：一次请求可以发送多个消息）

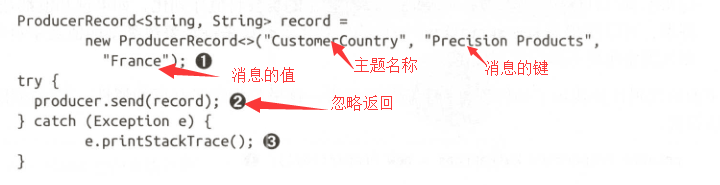
1. receive.buffer.bytes和send.buffer.bytes：指定TCP socket发送和接收缓冲区的大小。

### 发送消息的方式

发送消息有3种方式：

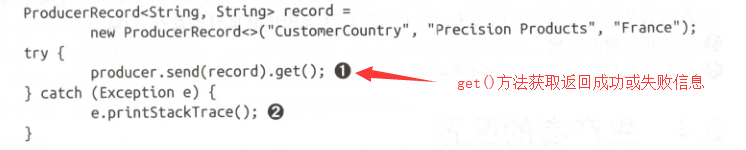
1. 发送并忘记：发送后，不管消息是否能正常到达。

图1



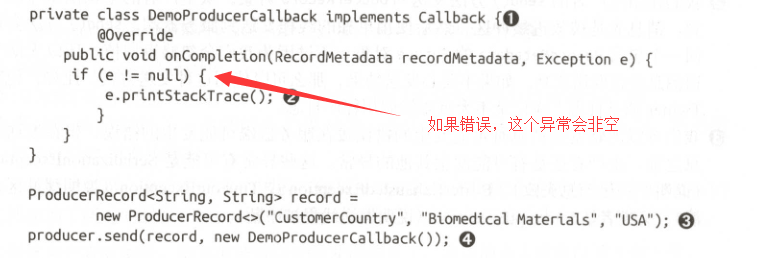
1. 同步发送：发送后，返回一个Future对象，然后可以用Future对象的get()方法等待，可以知道消息是否发送成功。

图2



1. 异步发送：指定一个回调函数，服务器响应时会调用这个回调函数。

图3



## 消费者

消费者订阅一个或多个主题，并按照消息生成的顺序读取（因为同个分区中的消息是先入先出的）。

### 消费者读取消息偏移量

消费者会把每个分区的最后读取的消息偏移量保存到Zookeeper或Kafka上，如果消费者关闭，读取状态不会丢失。

## broker

一个独立的Kafka服务器称为broker。

### broker和集群的关系

每个集群中会有一个broker充当集群控制器的角色（从集群的活跃成员中选举出来）。

### broker和分区的关系

一个分区属于一个broker（该broker称为该分区的头领），如果把一个分区分配给多个broker则会发生分区复制。

（注意：一个分区属于一个broker，因为一个分区在物理上就是一个文件夹，命名方式为”topicName\_partitionIndex”，partitionIndex从0~该topic的分区数-1）

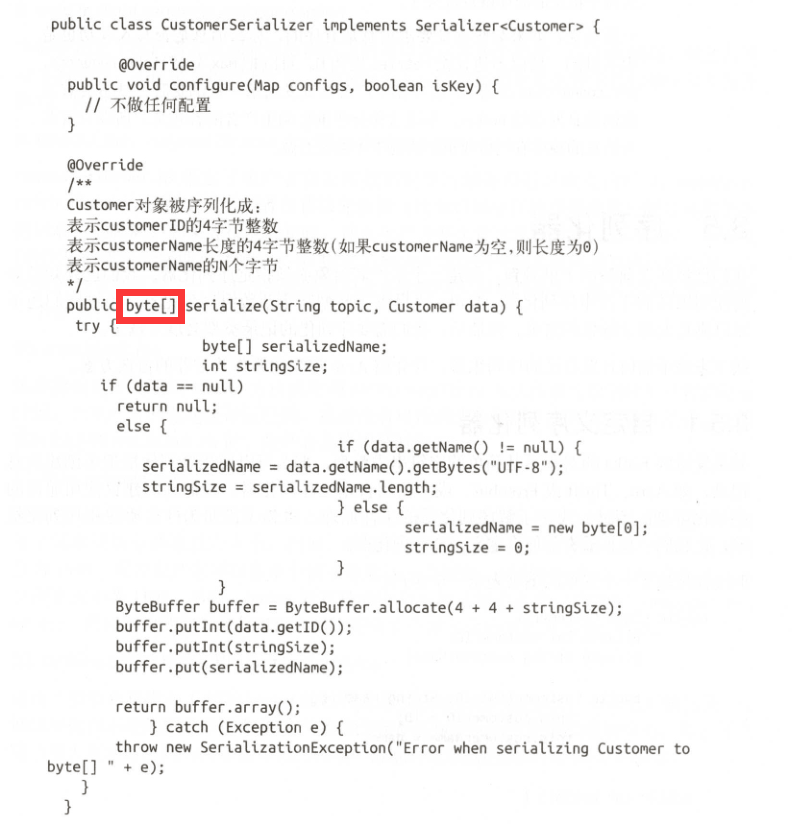
## 批次

批次就是一组消息，Kafka不会一条一条消息的放入主题的分区中，而是将等待消息到达批次的上限，然后将同一批次的消息全部放入某个分区中。

# 2 序列化器

## 2.1 自定义序列化器

图1



缺点：

1. 新增一个属性就要改一下序列化器（即对类的改动，需要改变序列化器）。

## 2.2 Avro序列化

Avroaaa