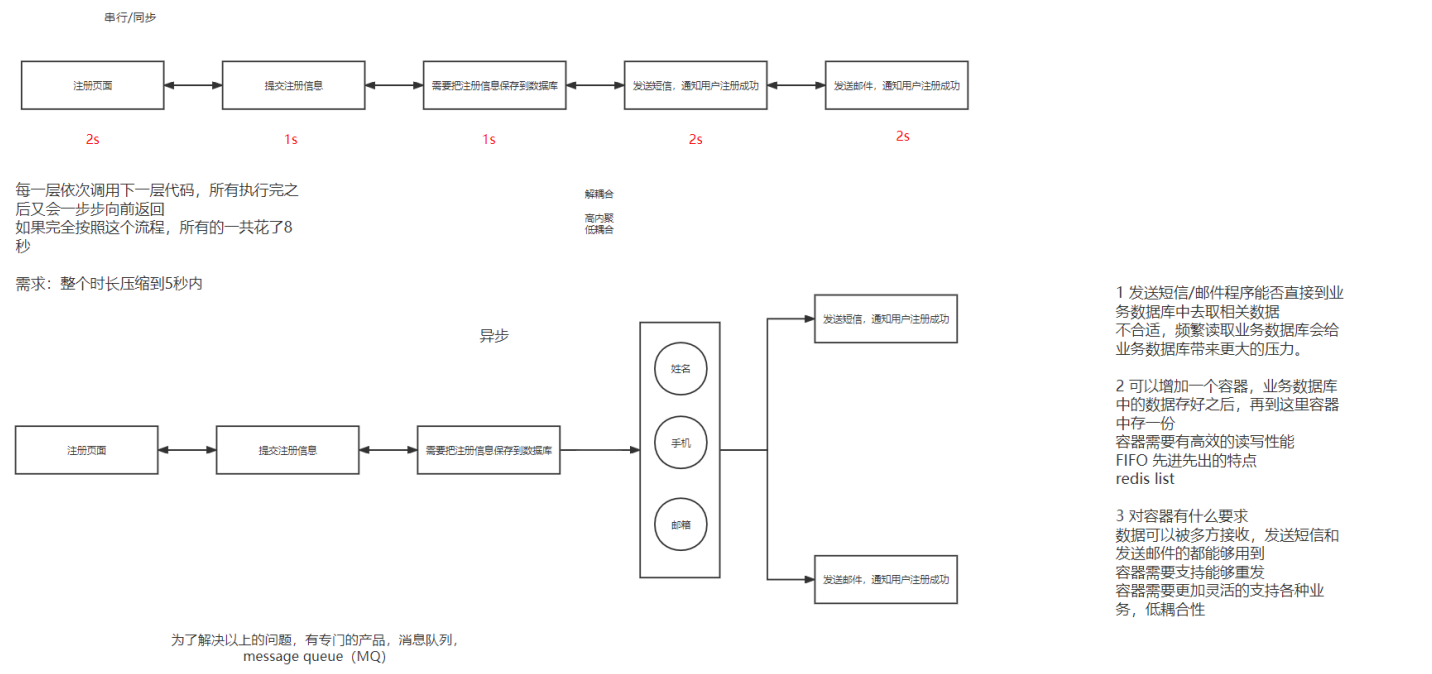


# 1.消息队列的基本介绍

## 1.产生的背景



## 2.常见的产品

- activeMQ（message queue）apache开源的一款消息队列，出现时间早，活跃度低
- RabbitMQ，目前是在java领域使用较多，在业务系统中使用较多
- RocketMQ，阿里出品，在阿里系使用较多，目前支持java，对于其他语言支持不多
- kafka，是大数据消息队列产品，在大数据领域内一统天下，kafka在极端下会出现重复消费，领英

## 3.作用

- 应用解耦和
- 进行异步处理，不用相互等待

- 1 快递员给你电话说，十二点某地把快递给你--同步
- 2 快递员给你电话说，快递放在快递柜，你自己去取 -- 异步

- 限流削峰
- 消息驱动系统

## 4.两种消费模型

- 点对点模式
  - 一条消息最终只会被一个消费者消费
  - 私聊 打电话
- 发布订阅模式
  - 一条消息最终可以被多个消费者所消费

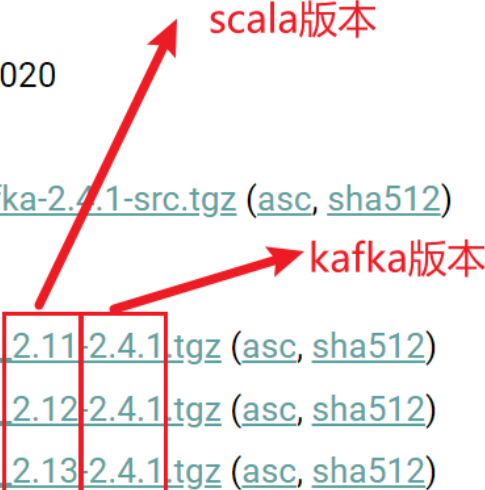
- 群发, 广播, 直播, 线上
- JMS java message server 类似jdbc

## 2.kafka介绍

### • 1.基本介绍

- kafka领英旗下消息队列产品, scala语言编写, (对java的兼容性很强) kafka实现了一部分jms, kafka构建集群, 分布式 分区 多副本, 可以提供高并发的数据处理能力。kafka依赖zookeeper, 本质上就是消息传递的中间件, broker (中介), 消息通过broker从生产者发送到消费者。
- kafka只依赖zookeeper, kafka的数据只存在系统磁盘, 不存在hdfs中
- 特点
  - 可靠性, 基于分布式 分区 多副本
  - 可扩展, 非常方便添加节点
  - 耐用性, 数据都存在磁盘上
  - 高性能, 单台节点可以达到10W并发量, 还可以通过扩展提高性能
- 版本

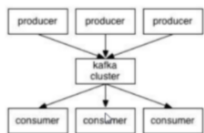
### 2.4.1

- Released March 12, 2020
  - [Release Notes](#)
  - Source download: [kafka-2.4.1-src.tgz](#) ([asc](#), [sha512](#))
  - Binary downloads:
    - Scala 2.11 - [kafka\\_2.11-2.4.1.tgz](#) ([asc](#), [sha512](#))
    - Scala 2.12 - [kafka\\_2.12-2.4.1.tgz](#) ([asc](#), [sha512](#))
    - Scala 2.13 - [kafka\\_2.13-2.4.1.tgz](#) ([asc](#), [sha512](#))
- 

We build for multiple versions of Scala. This only matters if you are using  
Otherwise any version should work (2.12 is recommended).

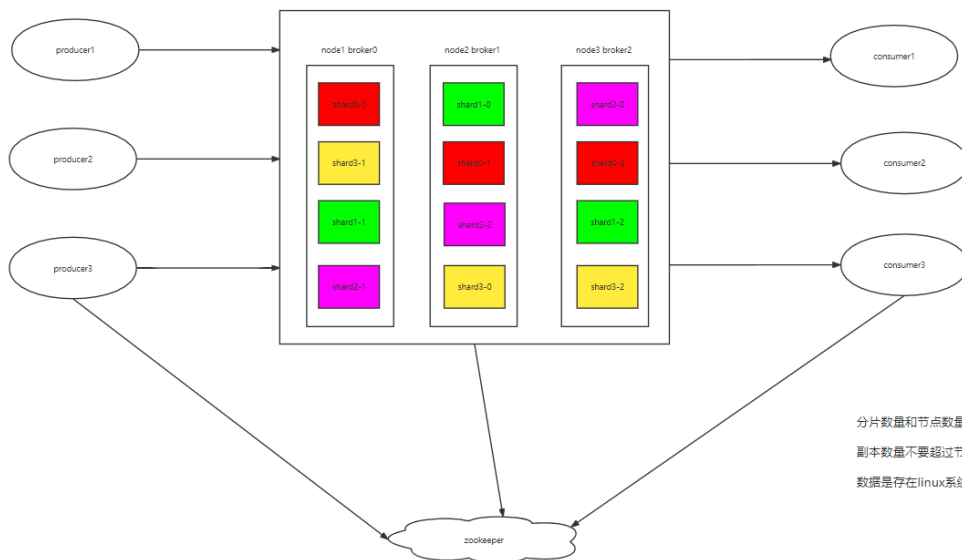
For more information, please read the detailed [Release Notes](#).

## 2.架构



是一个逻辑概念

kafka 集群



主题: test  
4分片  
会创建4个目录, 分别存在不同节点上  
test-0  
test-1  
test-2  
test-3  
每个目录根据副本数量的设定, 又会在其他节点存

分片数量和节点数量无关, 建议是节点数量的两倍  
副本数量不要超过节点数量, 超过之后没有意义  
数据是存在linux系统磁盘上, 没有存在hdfs中

- 1 kafka cluster kafka 集群
- 2 broker kafka集群的节点, 运行kafka服务
- 3 producer 生产者 产生数据的一方
- 4 consumer 消费者 使用数据的一方
- 5
- 6 topic 主题 是一个容器(逻辑概念, 落在磁盘上以目录的形式存在), 负责数据存储
- 7 shared 分片 一个topic可以有无数个分片(可以有无数个前半部分名字一样, 后半部分编号不一样的目录 test-0 test-1 test-2...)
- 8
- 9 replicas 副本 每个分片的消息都会有若干个副本, 就是把分片的目录复制存在其他节点上, 一般不会超过节点数量
- 10
- 11 zookeeper 旧版中管理元数据, 新版中作用是生产者 消费者 broker进行通信, 新版中的元数据是由kafka自己的主题管理
- 12
- 13 topics每个分片都有多个副本, 存在主备之分, 主副本负责数据的读写操作, 备份副本只负责同步主副本数据

## 3.shell命令的操作

### • 1.创建主题

```
1 kafka-topics.sh --create --bootstrap-server node1:9092,node2:9092,node3:9092 --
  partitions 3 --replication-factor 2 --topic "test001"
2
3 --create 表示创建主题
4 --bootstrap-server 指定kafka服务器
5 --partitions 指定消息分区数量
6 --replication-factor 指定每个分区几个副本
7 --topic 指定主题名
8
9 kafka-topics.sh --create --zookeeper node1:2181,node2:2181,node3:2181 --partitions 2 --
  replication-factor 2 --topic "test02"
10
11 --zookeeper 指定zookeeper地址端口
```

```
1 在三台节点指定
2 ll /export/server/kafka/data
3
4 分别可以在三个节点看到对应的副本数
```

## 2.shell脚本生产者

```
1 kafka-console-producer.sh --broker-list node1:9092,node2:9092,node3:9092 --topic test01
2
3 --broker-list 指定broker地址
4
```

## 3.shell脚本消费者

```
1 kafka-console-consumer.sh --bootstrap-server node1:9092,node2:9092,node3:9092 --topic
  test001
2
3 --bootstrap-server 指定broker地址
```

## 4.查看主题

```
1 kafka-topics.sh --bootstrap-server node1:9092,node2:9092,node3:9092 --list
```

## 5.查看主题详情

```
1 kafka-topics.sh --bootstrap-server node1:9092 --describe --topic test01
```

```
[root@node1 test01-0]# kafka-topics.sh --describe --bootstrap-server node1:9092 --topic test01
Topic: test01 PartitionCount: 2 ReplicationFactor: 2 Configs: segment.bytes=1073741824
Topic: test01 Partition: 0 Leader: 0 Replicas: 0,2 Isr: 0,2
Topic: test01 Partition: 1 Leader: 2 Replicas: 2,1 Isr: 2,1
```

可用副本在哪些节点

分区编号      当前分区主副本在个节点上      每个分区所有的副本在哪些broker上

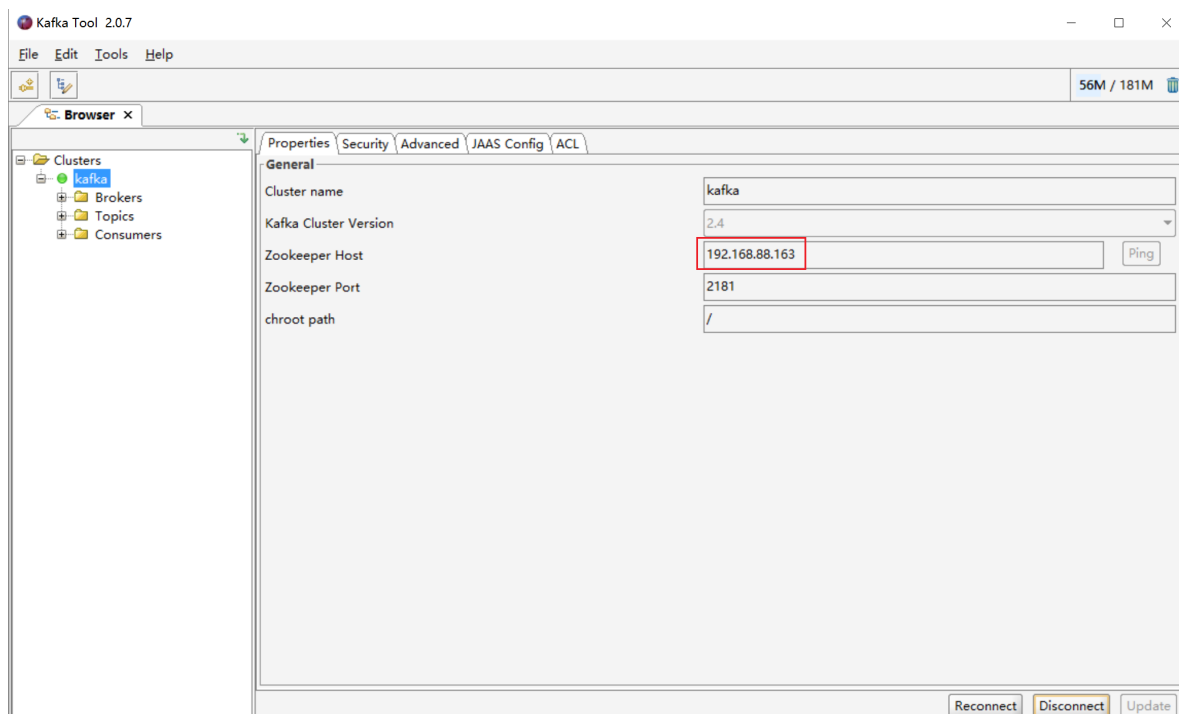
## 6.修改分区数量

```
1 kafka-topics.sh --bootstrap-server node1:9092 --alter --topic test01 --partitions 4
2 --alter --partitions 4 修改分区数量为4个
3 注意：副本数量无法修改
```

## 7.删除主题

```
1 kafka-topics.sh --delete --topic test01 --bootstrap-server node1:9092
2 删除主题时，先标记为删除状态，并没有立即删除
3 如果执行立即删除需要配置
4
5 delete.topic.enble=true
```

## 4.kafka工具



## 5.kafka的基准测试

### • 1.创建一个topic , benchmark

- 1 `kafka-topcis.sh --create --topic benchmark --bootstrap-server node1:9092,node2:9092,node3:9092 --partitions 3 --replication-factor 1`
- 2
- 3 partitions数量在broker无上限，越多越好
- 4 replication-factor 越大效率越低

### • 2.测试写入效率

- 1 `kafka-producer-perf-test.sh --topic benchmark --num-records 5000000 --throughput -1 --record-size 1000 --producer-props bootstrap.servers=node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 acks=1`
- 2
- 3 `--topics` 主题名称
- 4 `--num-records` 生产的数据量
- 5 `--throughput` 指定吞吐量 -1表示不限流
- 6 `--record-size` 每条消息大小
- 7 `--producer-props bootstrap.servers kafka`集群地址
- 8 acks ack模式

```
152400 records sent, 30480.0 records/sec (29.07 MB/sec), 1067.2 ms avg latency, 1160.0 ms max latency.
162880 records sent, 32576.0 records/sec (31.07 MB/sec), 1017.8 ms avg latency, 1142.0 ms max latency.
162112 records sent, 32409.4 records/sec (30.91 MB/sec), 1004.6 ms avg latency, 1076.0 ms max latency.
160304 records sent, 32060.8 records/sec (30.58 MB/sec), 1026.8 ms avg latency, 1097.0 ms max latency.
161568 records sent, 32242.7 records/sec (30.75 MB/sec), 1012.4 ms avg latency, 1086.0 ms max latency.
162384 records sent, 32476.8 records/sec (30.97 MB/sec), 1021.1 ms avg latency, 1086.0 ms max latency.
162016 records sent, 32403.2 records/sec (30.90 MB/sec), 1010.2 ms avg latency, 1110.0 ms max latency.
165552 records sent, 33110.4 records/sec (31.58 MB/sec), 981.2 ms avg latency, 1066.0 ms max latency.
5000000 records sent, 29346.167391 records/sec (27.99 MB/sec), 1106.85 ms avg latency, 3453.00 ms max latency, 1016 ms 50th, 153
2 ms 95th, 2768 ms 99th, 3218 ms 99.9th.
[root@node1 config]#
```

- 1 平均每秒3w条左右，顶峰30MB/s
- 2 平均延迟 1s 最大延迟1s多
- 3
- 4 虚拟机
- 5 cpu i7 两个核
- 6 每台4G内存，固态硬盘

## ● 3.测试消费者效率

- 1 kafka-consumer-perf-test.sh --broker-list node1:9092,node2:9092,node3:9092 --topic benchmark --fetch-size 1048576 --messages 5000000
- 2
- 3 --broker-list kafka集群地址
- 4 --topic 主题名
- 5 --fetch-size 每次拉取数据大小
- 6 --messages 总消费消息数量

# 6.kafka的JavaAPI的操作

## 1.数据产生到kafka

- 导入依赖

```
1 <dependencies>
2     <dependency>
3         <groupId>org.apache.kafka</groupId>
4         <artifactId>kafka-clients</artifactId>
5         <version>2.4.1</version>
6     </dependency>
7 </dependencies>
```

- 生产者实现

```

1 package cn.itcast.kafka;
2
3 import org.apache.kafka.clients.producer.KafkaProducer;
4 import org.apache.kafka.clients.producer.Producer;
5 import org.apache.kafka.clients.producer.ProducerRecord;
6 import java.util.Properties;
7
8 public class KafkaProducerTest {
9     public static void main(String[] args) {
10         Properties props = new Properties();
11         props.put("bootstrap.servers",
12             "node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092");
13
14         props.put("acks", "all"); // acks确认机制，保证生产者发送数据不丢失
15
16         // 序列化
17         props.put("key.serializer",
18             "org.apache.kafka.common.serialization.StringSerializer");
19         props.put("value.serializer",
20             "org.apache.kafka.common.serialization.StringSerializer");
21
22         Producer<String, String> producer = new KafkaProducer<>(props);
23         for (int i = 0; i < 10; i++) { // for循环一定要加{}
24
25             ProducerRecord<String, String> producerRecord = new ProducerRecord<>
26                 ("test02", Integer.toString(i));
27             producer.send(producerRecord);
28         }
29
30         producer.close();
31     }
32 }

```

- 验证



```
1 1 在命令行启动消费者
2 kafka-console-consumer.sh --bootstrap-server
   node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02
3
4 2 启动生产者代码
5
6 3 在终端中查看消费者会把生产者数据打印出来
7
```

## 2.消费者代码

```
1 package cn.itcast.kafka;
2
3 import org.apache.kafka.clients.consumer.ConsumerRecord;
4 import org.apache.kafka.clients.consumer.ConsumerRecords;
5 import org.apache.kafka.clients.consumer.KafkaConsumer;
6
7 import java.time.Duration;
8 import java.util.Arrays;
9 import java.util.Properties;
10
11 public class KafkaConsumerTest {
12     public static void main(String[] args) {
13         Properties props = new Properties();
14         props.setProperty("bootstrap.servers",
15             "node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092");
16         props.setProperty("group.id", "test");
17         props.setProperty("enable.auto.commit", "true");
18         props.setProperty("auto.commit.interval.ms", "1000");
19         props.setProperty("key.deserializer",
20             "org.apache.kafka.common.serialization.StringDeserializer");
21         props.setProperty("value.deserializer",
22             "org.apache.kafka.common.serialization.StringDeserializer");
23     }
24 }
```

```

20         KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
21         consumer.subscribe(Arrays.asList("test02"));
22         while (true) {
23             ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(100));
24             for (ConsumerRecord<String, String> record : records) {
25                 System.out.printf("偏移量 = %d, 键 = %s, 值 = %s%n", record.offset(),
record.key(), record.value());
26             }
27         }
28     }
29 }
30

```

- 验证

- 1 先启动消费者代码
- 2 再启动生产者代码
- 3 消费者输出生产者数据

## 7.kafka的分区与副本机制

分区：一个topic是一个逻辑容器（一个主题可能是分成多个目录存，test01主题对应的目录 test01-0 test01-1），分区时把容器拆分成若干个小容器，每个小容器就是一个分区，分区数量一般不超过节点数量的两倍

- 分区解决了什么问题：
  - 可以解决负载均衡，有活大家一起干
  - 可以提高并行度
  - 解决单台节点存储空间有限的问题

副本：每个分区都可以由副本，防止数据丢失

- 副本数量一般不会超过集群节点数量，超过了没意义
- 副本保证数据的高可用

## 8.如何保证数据不丢失

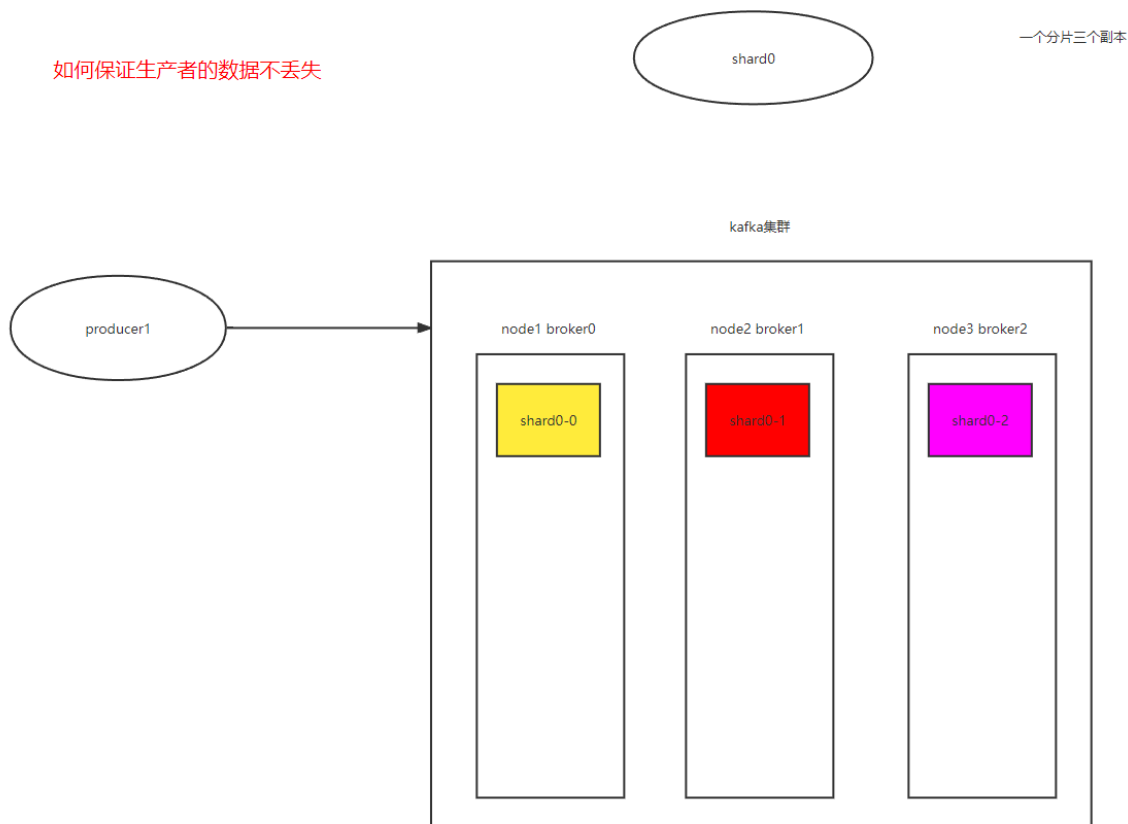
### 1.生产者发送数据不丢失

- 1 ack校验机制
- 2 all (-1)，当ack设置为all或者-1时，表示生产者需要所有副本都返回确认信息，此时生产者认为发送成功
- 3
- 4 1，ack设置为1，表示生产者只用等待主副本返回确认信息即可，一旦主副本返回去确认信息，则生产者认为发送成功
- 5
- 6 0，ack设置为0，表示生产者只管发送数据，完全不关心数据是否被broker接受完成，生产者不接受校验码
- 7
- 8 实际生产中怎么设置
- 9 从安全角度出发 -1 > 1 > 0
- 10 从效率角度0 > 1 > -1
- 11 例如，系统需要采集其他组件的日志，通过生产者采集发送到broker，有其他工具进行分析
- 12 假设日志级别info warning error
- 13 info 数据不是特别重要，比较多设置ack 为0
- 14 warning 数据有一点重要，可以设置为1
- 15 leerror 很重要 需要设置为-1

## 在代码中如何设置

```
1 Properties props = new Properties();
2 props.put("acks", "all"); // acks确认机制，保证生产者发送数据不丢失
3 Producer<String, String> producer = new KafkaProducer<>(props);
```

如何保证生产者的数据不丢失



- 1 1.如果ack设置的是-1 或者 1 ， 每发一条信息需要broker返回校验信息，broker没有返回校验信息怎么办
- 2 解决方案：
- 3 设置等待时间，如果超过等待时间仍然没有收到，此时进行重试，就是重新发送数据
- 4 如果重发之后仍然没有接收到校验信息，怎么办？
- 5 一般重发3~5次，最后仍然没收到校验信息，则基于告警系统通知相关人员处理
- 6
- 7 2.发送一条信息，broker返回一个校验信息，是否会对网络带宽产生影响？
- 8 会有影响，该怎么处理
- 9 引入缓冲池，先把数据放在缓冲池中，当信息数据达到阈值时一次性发给broker，broker只用返回这一个批次校验信息。
- 10 采用的异步发送的方法，一个批次的数据必须属于一个同一个分区，（某个分区对应的是一个broker），如果数据是多个分区的，底层也会拆分成几个小批次分别发给对应的broker
- 11
- 12 3.采用缓冲池进行批量发送数据，broker没有响应，缓冲池满了，怎么办？
- 13 可以把数据存入到临时容器中（文件，数据库。。。），基于告警系统通知相关人员

## 具体配置

- 1 1 数据发送方式：同步 异步
- 2 2 默认等待时长，设置两分钟
- 3 `delivery.timeout.ms`
- 4 3 重试次数 `retries`
- 5 设置成3-5

```
6 5 默认缓冲池大小 32M
7 buffer.memory
8 5 默认批次大小
9 batch.size 16KB
10
11 这些属性都是在生产者properties中设置
```

## 2.broker端如何保证

- broker采用副本机制保证接受的数据不会丢失，副本越多数据越安全
- 生产端需要ack校验码设置为-1

## 3.消费端如何保证数据不丢失

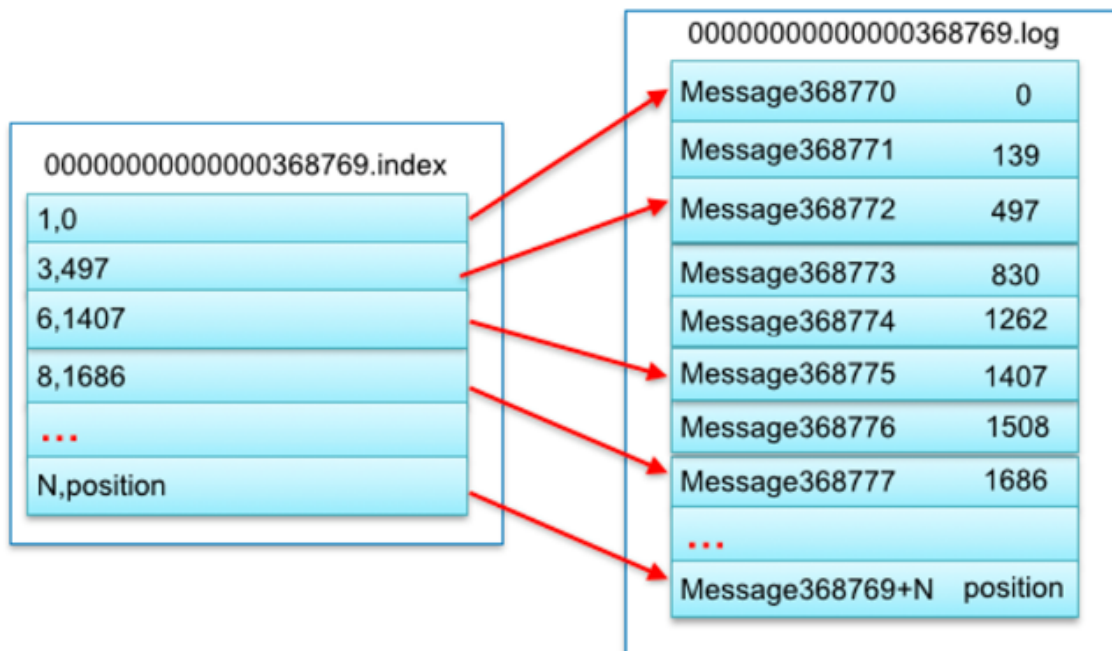
```
1 kafka消息队列中维护了消费者在这个队列中的偏移量
2
3 消费者消费的流程
4 1.消费端启动后，首先询问broker，当前这个topic，我上一次消费到哪里了
5 2.broker就会根据消费者的组编号，以及消费者的编号，去查询这个消费者租上次消费到哪个偏移量
6 3.如果没找到对应消费者编号，肯定找不到对应的偏移量，broker认为消费者第一次来，设置偏移量为当前队列的最后一个位置，让消费者从这里消费。如果由偏移量，就从偏移量后面开始消费
7
8 4.消费者开始消费后，可以手动也可以自动提交偏移量，broker负责记录即可，偏移量存在
   __consumer_offsets主题中，在0.8.x 之前是zookeeper进行维护的
9
10 通过这一套机制，数据绝对不会丢失，但是可能存在重复消费，因为消费端偏移量提交不及时，下次再连上时
    就有可能会重复消费。很多公司是自己维护这个偏移量，需要提交偏移量，否则每次逗号从第一个的地方获取
    数据
11
12 消费者a会不会重复消费“消费者b已经消费过的数据”
13     如果是同一个消费者组内，a b 不会重复消费对方已经消费的数据
14     如果不是同一个消费者组，各不相干
15
16 从偏移量开始消费，到哪里结束呢？
17 到队列的结尾
```

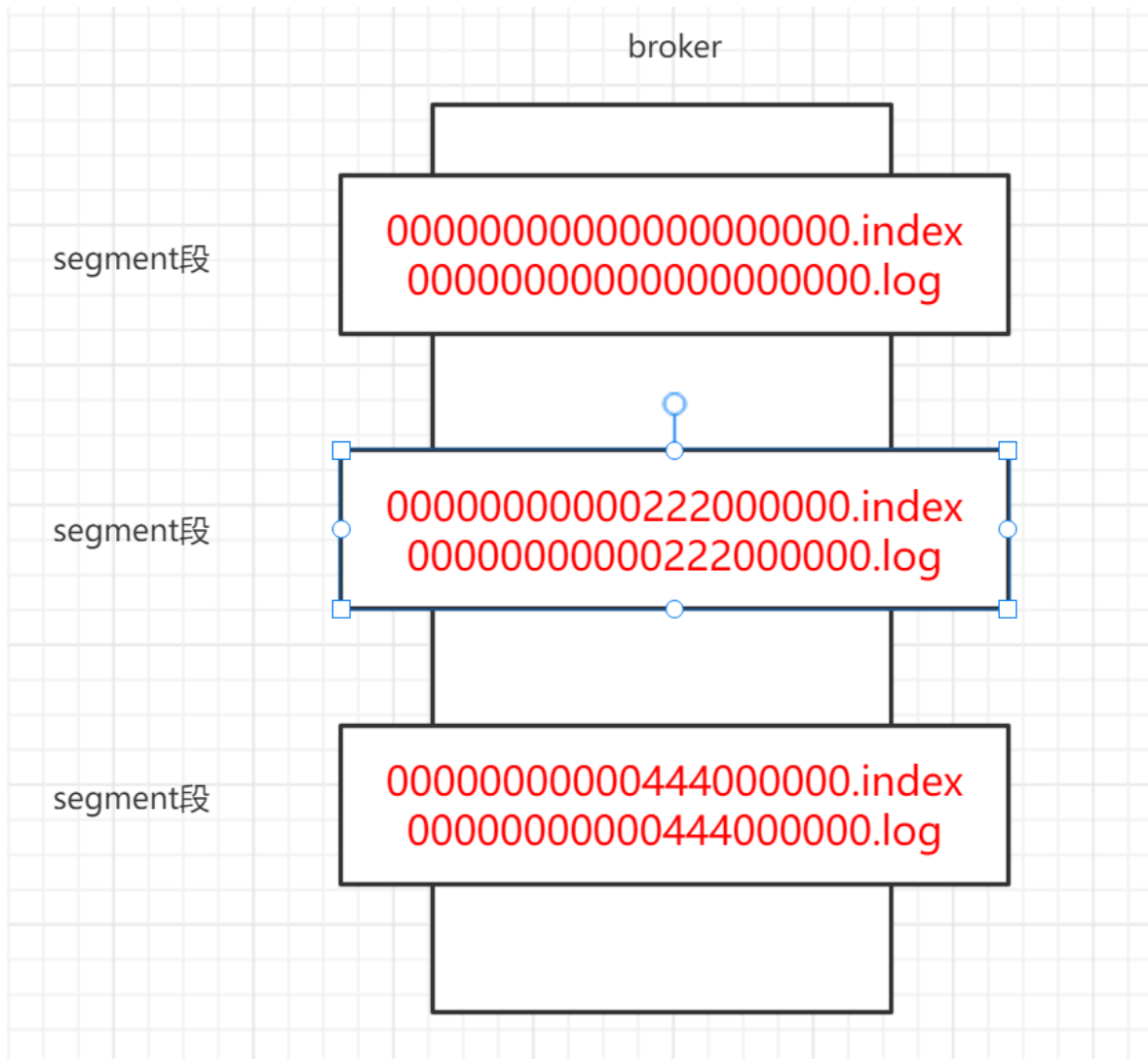
## 9.消息存储和查询机制

- 1.消息存储

1 每个分片中有一个log文件index文件，这两个文件一起被称为segment文件，消息存在log文件中，index文件是对log文件进行索引

- kafka中的数据会过期，过期时间是168小时（7天）
- 1.记录在某个地方，过期后到这里查找数据，然后删除
- 2.存到文件中，文件中所有消息都过期后把文件删除
- 当前log文件达到1GB时，会生成第二个log文件
- index文件中对log文件中的消息进行了索引，消息在log文件中的相对位置和开始的字节数，index文件只对部分消息进行索引，节省空间
- 每个主题都可以分区存储（分目录），每个目录内又是份文件存储
- 消息队列中的数据存储磁盘
- 每个log文件名是上一个log文件的最后一条消息消息编号
- kafka本质就是一个消息孳地的中间件，只是临时存储数据，一旦数据被消费或者过期，就没什么用了





## • 2.数据查询机制

需求: 读取 offset=368776 的message消息数据, 数据集如下

```
00000000000000000000.index
00000000000000000000.log
0000000000000000368769.index
0000000000000000368769.log
0000000000000000737337.index
0000000000000000737337.log
00000000000000001105814.index
00000000000000001105814.log
```

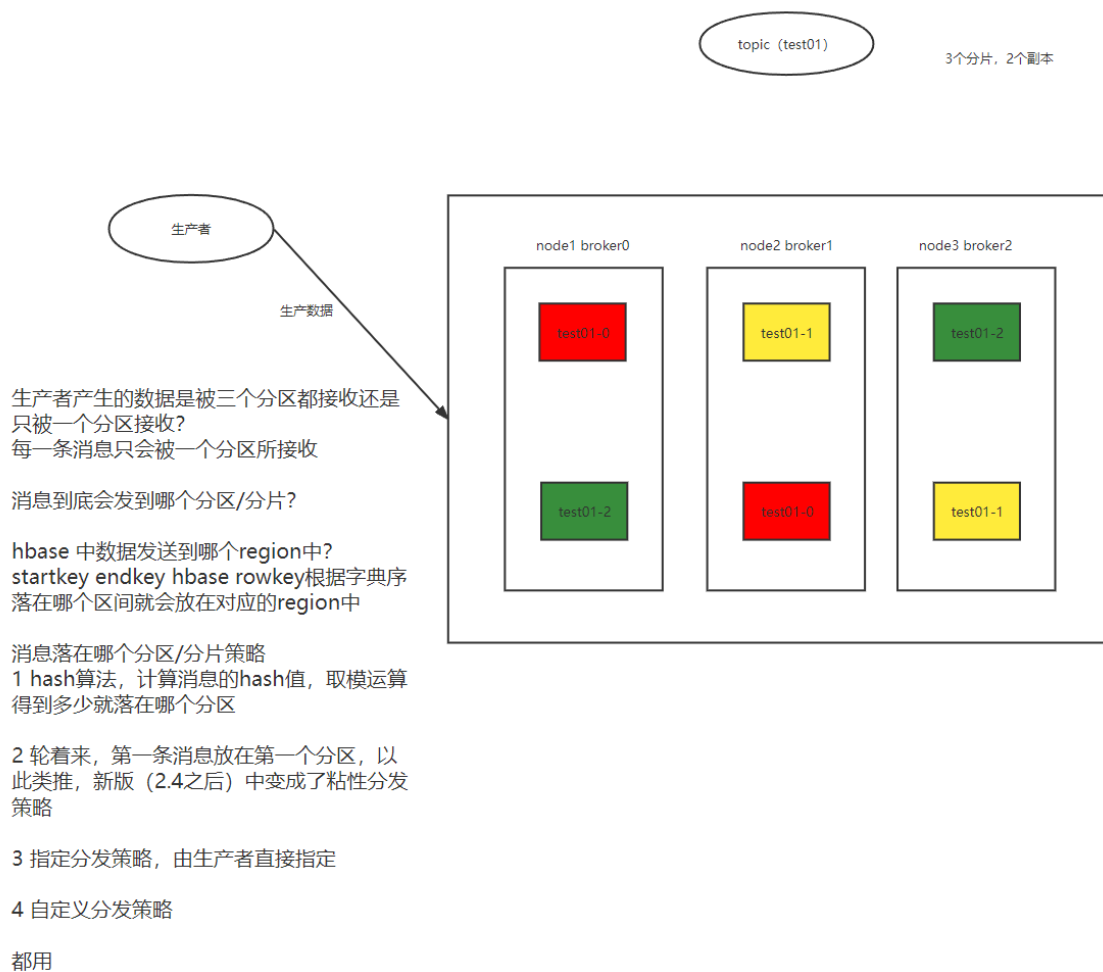
- 1 1 3 5 11 12 13 17 19 22
- 2 查找13是否在数组中
- 3 顺序查找, 查找6次才能找到13 平均时间复杂度 $O(n)$
- 4 二分查找, 对有序数据, 通过3次就找到13  $O(\log(n))$

1. 确定数据在哪个log文件中，使用二分查找法，368776这个消息在368769.log中
2. 根据log文件名和要查找的消息计算出消息在log文件中的相对偏移量7
3. 根据相对偏移量到index中，又根据二分查找法可以找到消息的起始字节数

- 1 消息主题是确定
- 2 对一个的目录确定 /export/server/kafka/data/test02
- 2 可以把目录中文件全部拿出来
- 3 把所有的log文件名获取到
- 4 把文件名转成数字
- 5 消息编号也可以转成数字
- 6 用二分查找法找到消息所在的文件
- 8 0-368769 368769-737337 737337-1105814 1105814~
- 9 368776

## 10.生产者的数据分发策略

### • 分发策略介绍



### • 如何使用不同的分发策略



```

1 The default partitioning strategy:
2 If a partition is specified in the record, use it # 指定分区分发策略
3 If no partition is specified but a key is present choose a partition based on a hash of
  the key # 根据keyhash值选择分区
4 If no partition or key is present choose the sticky partition that changes when the
  batch is full. See KIP-480 for details about sticky partitioning. # 粘性分发策略
5
6 1 DefaultPartitioner 默认分区类，底层提供不同的分发测率支持的一个类
7 指定分区分发策略，如果指定分区，则发送到该分区
8 根据keyhash值选择分区 如果设定key，则根据key计算hash，取模
9 粘性分发策略 如果前面都没设定，会随机选择一个分区采用粘性分发
10
11 2 ProducerRecord 不同分发策略接口
12
13 public ProducerRecord(String topic, Integer partition, Long timestamp, K key, V
  value, Iterable<Header> headers) {
14     if (topic == null)
15         throw new IllegalArgumentException("Topic cannot be null.");
16     if (timestamp != null && timestamp < 0)
17         throw new IllegalArgumentException(
18             String.format("Invalid timestamp: %d. Timestamp should always be
  non-negative or null.", timestamp));
19     if (partition != null && partition < 0)
20         throw new IllegalArgumentException(
21             String.format("Invalid partition: %d. Partition number should
  always be non-negative or null.", partition));
22     this.topic = topic;
23     this.partition = partition;
24     this.key = key;
25     this.value = value;
26     this.timestamp = timestamp;
27     this.headers = new RecordHeaders(headers);
28 }
29
30
31 /**
32  * Creates a record to be sent to a specified topic and partition
33  *
34  * @param topic The topic the record will be appended to
35  * @param partition The partition to which the record should be sent
36  * @param key The key that will be included in the record

```

```

37     * @param value The record contents
38     * @param headers The headers that will be included in the record
39     */
40     public ProducerRecord(String topic, Integer partition, K key, V value,
41         Iterable<Header> headers) {
42         this(topic, partition, null, key, value, headers);
43     }
44     /**
45     * Creates a record to be sent to a specified topic and partition
46     *
47     * @param topic The topic the record will be appended to
48     * @param partition The partition to which the record should be sent
49     * @param key The key that will be included in the record
50     * @param value The record contents
51     */
52     // 根据设定的partition分发
53     public ProducerRecord(String topic, Integer partition, K key, V value) {
54         this(topic, partition, null, key, value, null);
55     }
56
57     /**
58     * Create a record to be sent to Kafka
59     *
60     * @param topic The topic the record will be appended to
61     * @param key The key that will be included in the record
62     * @param value The record contents
63     */
64     // 计算key的hash, 取模得到得到分区号
65     public ProducerRecord(String topic, K key, V value) {
66         this(topic, null, null, key, value, null);
67     }
68
69     /**
70     * Create a record with no key
71     *
72     * @param topic The topic this record should be sent to
73     * @param value The record contents
74     */
75     // 如果调用这个函数, 采用粘性分发策略

```

```

76     public ProducerRecord(String topic, V value) {
77         this(topic, null, null, null, value, null);
78     }
79
80
81 3 自定义分发策略
82 直接抄DefaultPartitioner中的实现方法

```

- 验证不同分发策略
  - 创建一个主题，由三个分区

```

1 kafka-topics.sh --create --bootstrap-server
  node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --partitions 3 --
  replication-factor 1 --topic test001

```

- 开发三个终端，分别启动三个消费者，每个消费者绑定到一个分区

```

1 kafka-console-consumer.sh --bootstrap-server
  node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test001 --
  partition 0
2
3 kafka-console-consumer.sh --bootstrap-server
  node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test001 --
  partition 1
4
5 kafka-console-consumer.sh --bootstrap-server
  node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test001 --
  partition 2
6

```

- 代码 生产者端

```

1 package sz.base.kafka;
2
3 import org.apache.hadoop.hbase.util.Bytes;
4 import org.apache.kafka.clients.producer.KafkaProducer;
5 import org.apache.kafka.clients.producer.Producer;
6 import org.apache.kafka.clients.producer.ProducerRecord;
7
8 import java.util.Properties;
9
10 public class KafkaProducerTest {
11     public static void main(String[] args) {
12         Properties props = new Properties();

```

```

13     props.put("bootstrap.servers", "node1:9092,node2:9092,node3:9092");
14     props.put("delivery.timeout.ms", 120000);
15     props.put("acks", "all");//acks确认机制，保证生产者发送数据不丢失
16     //序列化
17     props.put("key.serializer",
18 "org.apache.kafka.common.serialization.StringSerializer");
19     props.put("value.serializer",
20 "org.apache.kafka.common.serialization.StringSerializer");
21     //构造kafka
22     Producer<String, String> producer = new KafkaProducer<>(props);
23     for (int i = 0; i < 10; i++) {
24         //往哪个主题发，发什么内容，然后发送。ProducerRecord: 定义分发策略
25         //粘性分发
26         // ProducerRecord<String,String> producerRecord=new ProducerRecord<>
27         ("test001",Integer.toString(i));
28         //指定分区策略
29         // ProducerRecord<String,String> producerRecord=new ProducerRecord<>
30         ("test001",0,"123abc",Integer.toString(i));
31         //hash取模分发策略
32         ProducerRecord<String,String> producerRecord=new ProducerRecord<>("test001",
33 i+"",Integer.toString(i));
34         producer.send(producerRecord);
35     }
36     producer.close();
37 }
38 }
39 }

```

- 消费者端

```

1  import org.apache.kafka.clients.consumer.ConsumerRecord;
2  import org.apache.kafka.clients.consumer.ConsumerRecords;
3  import org.apache.kafka.clients.consumer.KafkaConsumer;
4
5  import java.time.Duration;
6  import java.util.Arrays;
7  import java.util.Properties;
8
9  public class KafkaConsumerTestManual {
10     public static void main(String[] args) {
11         Properties props = new Properties();
12         props.setProperty("bootstrap.servers", "node1:9092,node2:9092,node3:9092");

```

```

13         props.setProperty("group.id", "test");
14         //设置为自动提交偏移量
15         //         props.setProperty("enable.auto.commit", "true");
16         //         props.setProperty("auto.commit.interval.ms", "1000");
17         props.setProperty("key.deserializer",
18             "org.apache.kafka.common.serialization.StringDeserializer");
19         props.setProperty("value.deserializer",
20             "org.apache.kafka.common.serialization.StringDeserializer");
21         KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>
22             (props);
23         consumer.subscribe(Arrays.asList("test02"));
24         while (true) {
25             ConsumerRecords<String, String> records =
26             consumer.poll(Duration.ofMillis(100));
27             for (ConsumerRecord<String, String> record : records) {
28                 System.out.printf("偏移量 = %d , 键 = %s , 值= %s\n ", record.offset(),
29                     record.key(), record.value());
30                 //此时消费已经完成, 提交偏移量
31                 consumer.commitAsync();//异步提交
32                 //         consumer.commitSync();//同步提交
33             }
34         }
35     }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```

## 11.java API设置为同步异步

### 生产者端

```

1 package sz.base.kafka;
2
3 import org.apache.kafka.clients.producer.*;
4
5 import java.util.Properties;
6
7 public class KafkaProducerTestAsync {
8     public static void main(String[] args) {
9         Properties props = new Properties();
10        props.put("bootstrap.servers", "node1:9092,node2:9092,node3:9092");
11    }
12 }

```

```

11     props.put("delivery.timeout.ms",120000);
12     props.put("acks", "all");//acks确认机制，保证生产者发送数据不丢失
13     //序列化
14     props.put("key.serializer",
15 "org.apache.kafka.common.serialization.StringSerializer");
16     props.put("value.serializer",
17 "org.apache.kafka.common.serialization.StringSerializer");
18     //构造kafka
19     Producer<String, String> producer = new KafkaProducer<>(props);
20     for (int i = 0; i < 10; i++) {
21         //往哪个主题发，发什么内容，然后发送
22         ProducerRecord<String,String> producerRecord=new ProducerRecord<>
23 ("test02",Integer.toString(i));
24         producer.send(producerRecord, new Callback() {
25             @Override
26             public void onComplete(RecordMetadata recordMetadata, Exception e) {
27                 if (e != null ){//异步
28                     //如果exception不等于空，则表明发生异常，说明消息发送失败
29                     //通过告警系统通知相关人员
30                 }
31             }
32         });
33     }
34     producer.close();
35 }

```

```

1 package sz.base.kafka;
2
3 import org.apache.kafka.clients.producer.KafkaProducer;
4 import org.apache.kafka.clients.producer.Producer;
5 import org.apache.kafka.clients.producer.ProducerRecord;
6
7 import java.util.Properties;
8 import java.util.concurrent.ExecutionException;
9
10 public class KafkaProducerTestSync {
11     public static void main(String[] args) {
12         Properties props = new Properties();

```

```

13      //设置kafka集群的地址
14      props.put("bootstrap.servers", "node1:9092,node2:9092,node3:9092");
15      props.put("delivery.timeout.ms", 120000);
16      //ack模式，all是最慢但最安全的
17      props.put("acks", "all");//acks确认机制，保证生产者发送数据不丢失
18      //失败重试次数
19      //props.put("retries", 0);
20      //每个分区未发送消息总字节大小（单位：字节），超过设置的值就会提交数据到服务端
21      //props.put("batch.size", 10);
22      //props.put("max.request.size", 10);
23      //消息在缓冲区保留的时间，超过设置的值就会被提交到服务端
24      //props.put("linger.ms", 10000);
25      //整个Producer用到总内存的大小，如果缓冲区满了会提交数据到服务端
26      //buffer.memory要大于batch.size，否则会报申请内存不足的错误
27      //props.put("buffer.memory", 10240);
28      //序列化
29      props.put("key.serializer",
30      "org.apache.kafka.common.serialization.StringSerializer");
31      props.put("value.serializer",
32      "org.apache.kafka.common.serialization.StringSerializer");
33      //构造kafka
34      Producer<String, String> producer = new KafkaProducer<>(props);
35      for (int i = 0; i < 10; i++) {
36          //往哪个主题发，发什么内容，然后发送
37          ProducerRecord<String,String> producerRecord=new ProducerRecord<>
38          ("test02",Integer.toString(i));
39          try {
40              producer.send(producerRecord).get();//同步发送数据
41          } catch (Exception e) {
42              //如果出现异常说明前面已经是多次重试，等待，并且失败了，所有才会走到这里，通知
43              告警系统
44              e.printStackTrace();
45          }
46      }
47      producer.close();
48  }
49  }

```

## 消费者端

```
1 package sz.base.kafka;
2
3 import org.apache.kafka.clients.consumer.ConsumerRecord;
4 import org.apache.kafka.clients.consumer.ConsumerRecords;
5 import org.apache.kafka.clients.consumer.KafkaConsumer;
6
7 import java.time.Duration;
8 import java.util.Arrays;
9 import java.util.Properties;
10
11 public class KafkaConsumerTestManual {
12     public static void main(String[] args) {
13         Properties props = new Properties();
14         //设置kafka集群的地址
15         props.setProperty("bootstrap.servers", "node1:9092,node2:9092,node3:9092");
16         //设置消费者组，组名字自定义，组名字相同的消费者在一个组
17         props.setProperty("group.id", "test");
18         //设置为自动提交偏移量，//开启offset自动提交
19         // props.setProperty("enable.auto.commit", "true");
20         //自动提交时间间隔
21         // props.setProperty("auto.commit.interval.ms", "1000");
22         props.setProperty("key.deserializer",
23             "org.apache.kafka.common.serialization.StringDeserializer");
24         props.setProperty("value.deserializer",
25             "org.apache.kafka.common.serialization.StringDeserializer");
26         //序列化器
27         KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>
28             (props);
29         //实例化一个消费者
30         consumer.subscribe(Arrays.asList("test02"));
31         //死循环不停的从broker中拿数据
32         while (true) {
33             ConsumerRecords<String, String> records =
34                 consumer.poll(Duration.ofMillis(100));
35             for (ConsumerRecord<String, String> record : records) {
36                 System.out.printf("偏移量 = %d , 键 = %s , 值= %s%n ", record.offset(),
37                     record.key(), record.value());
38                 //此时消费已经完成，提交偏移量
39             }
40         }
41     }
42 }
```

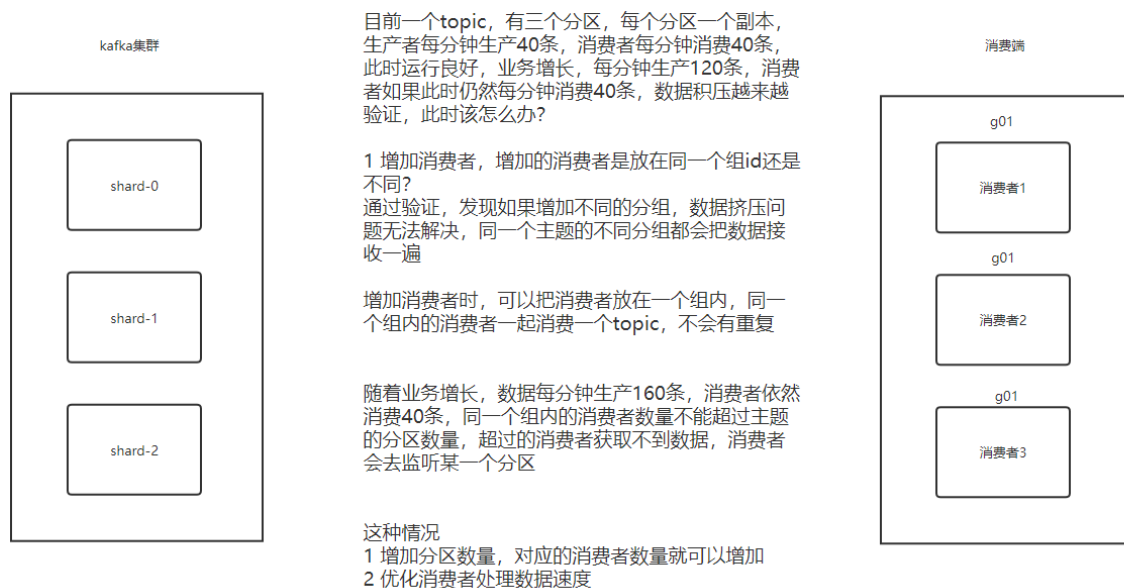


```

34         consumer.commitAsync();//异步提交
35     //         consumer.commitSync();//同步提交
36     }
37 }
38 }
39 }
40

```

## 12.kafka消费者负载均衡的机制



- 1 1 启动不同分组的消费者
- 2 test02分区数量是3个
- 3
- 4 kafka-console-consumer.sh --bootstrap-server node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group g01
- 5 kafka-console-consumer.sh --bootstrap-server node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group g02
- 6 kafka-console-consumer.sh --bootstrap-server node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group g03
- 7
- 8
- 9 2 启动同一个分组的消费者
- 10 kafka-console-consumer.sh --bootstrap-server node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group g04

```
11 kafka-console-consumer.sh --bootstrap-server
   node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group
   g04
12 kafka-console-consumer.sh --bootstrap-server
   node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group
   g04
13 接着又增加一个
14 kafka-console-consumer.sh --bootstrap-server
   node1.itcast.cn:9092,node2.itcast.cn:9092,node3.itcast.cn:9092 --topic test02 --group
   g04 新增的这个获取不到数据
```

- 1 消息队列的消费模式
- 2
- 3 点对点
- 4 方案1 只有一个消费者消费某个topic
- 5 方案2 消费这个topic的消费者放在一个分组内，每条消息都只会被一个消费者所接收
- 6
- 7
- 8 发布订阅
- 9 把多个消费者放在不同的组中，每个消费者都会把所有的额消息都接收到