# Kafka总结

## 基础知识

### Broker

在kafka集群中代表每一节点，就是kafka节点

### **Topic**

topic是Kafka数据写入操作的基本单元，可以指定副本

1，一个Topic包含一个或多个Partition，建Topic的时候可以手动指 定 Partition个数，个数与服务器个数相当

2，每条消息属于且仅属于一个Topic

3，Producer发布数据时，必须指定将该消息发布到哪个Topic

4，Consumer订阅消息时，也必须指定订阅哪个Topic的信息

### **Partition**

每个Partition只会在一个Broker上，物理上每个Partition对应的是一个文件夹

Kafka默认使用的是hash进行分区，所以会出现不同的分区数据不一样的情况，但是partitioner是可以override的

Partition包含多个Segment，每个Segment对应一个文件，Segment可以手动指定大小，当Segment达到阈值时，将不再写数据，每个Segment都是大小相同的

Segment由多个不可变的记录组成，记录只会被append到Segment中，不会被单独删除或者修改，每个Segment中的Message数量不一定相等

### **Segment**

segment存放的是消息message,在parttion中有多个segment，segment由两个文件组成，一个是log，一个是index

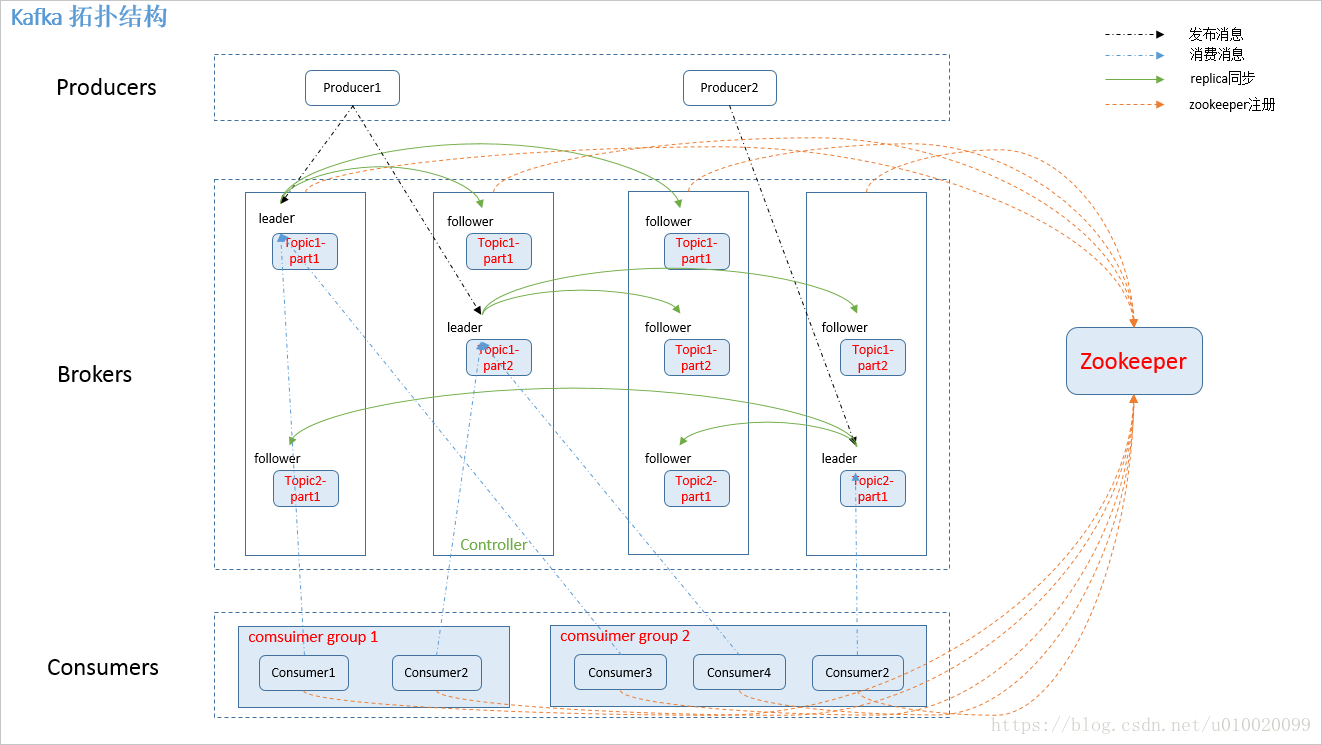
log文件就是数据文件，里面存放的就是Message，index文件是索引文件。Index文件记录了元数据信息，指向对应的数据文件中Message的物理偏移量

LogSegment文件命名的规则是，partition全局的第一个Segment从0（20个0）开始，后续的每一个文件的文件名是上一个文件的最后一条消息的offset值，这样命名的好处是什么呢？假如我们有一个Consumer已经消费到了offset=x，那么如果要继续消费的话，就可以使用二分查找法来进行查找，对LogSegment文件进行查找，就可以定位到某个文件，然后拿x值去对应的index文件中去找第x条数据所在的位置。Consumer读数据的时候，实际是读Index的offset，并且会记录上次读到哪里

****

Index文件，上图的左半部分是Index文件，里面存储的是n对key-value，其中key是Message在log文件中的编号，比如1，3，6，8.....，表示第1条、第3条、第6条、第8条消息等，但是因为Index文件中并没有为数据文件中的每条消息都建立索引，而是采用了稀疏存储的方式，每隔一定字节的数据建立一条索引。这样避免了索引文件占用过多的空间，从而可以将索引文件保留在内存中，但缺点是没有建立索引的Message不能一次定位到其在log文件中的位置，这种情况下就需要做一次顺序扫描，不过这次顺序扫描的范围就会很小了。value值表示该消息的物理偏移地址。

**group消息分组**



Consumer Group 是Kafka提供的可扩展且具有容错性的消费者机制。**在组内多个消费者实例(Consumer Instance ),它们共享一个公共的ID即 Group ID** 。组内的所有消费者协调在一起消费订阅主题（Subscribed Topics）的所有分区(Partition)。**当然一个分区只能有同一个消费者组的一个Consumer 实例消费**。  
Consumer Group 有三个特性：

1. Consumer Group 下可以有一个或多个Consumer 实例。 这里的实例可以是一个单独的进程，也可以是同一进程下的线程；
2. Group ID 是一个字符串， 在Kafka集群中唯一标识，Consumer Group；
3. Consumer Group  
   下所有实例订阅主体的单个分区，只能分配给组内某个Consumer实例消费。同一个分区消息可能被多个Group 消费。

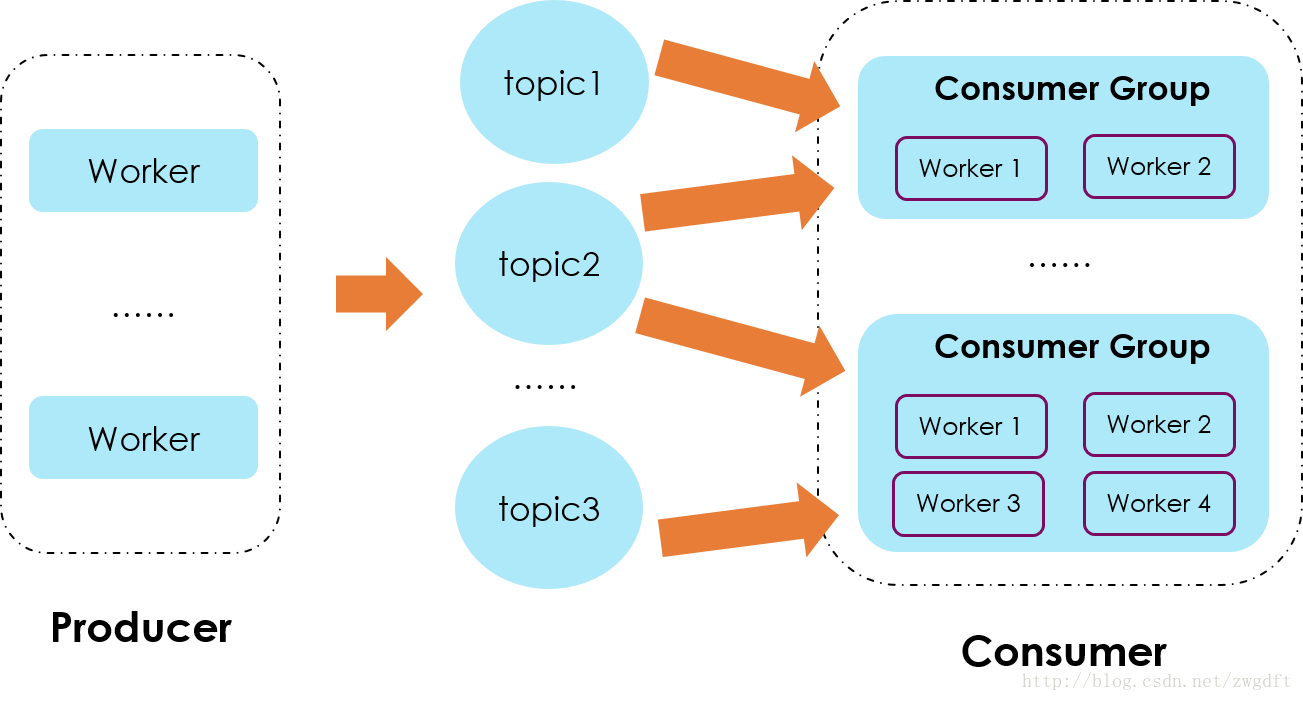
**Ps：消费组内的消费者实例数量最好等于分区数量，刚好每个消费组消费一个分区，或者为1/n倍刚好每个消费组均匀消费分区**

## **应用部分**

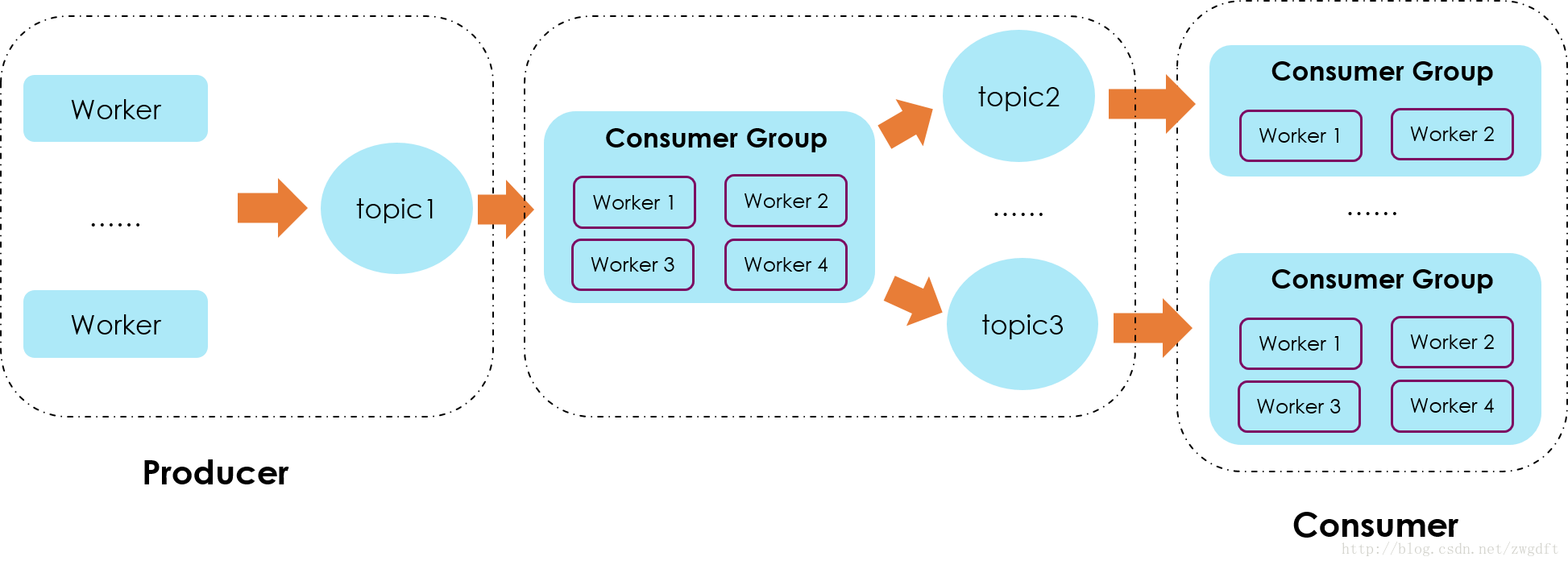
### Kafka实现订阅模式

**方案一：**继续使用上述生产消费者的模式，在不同的group中过滤出自己感兴趣的事件数据，然后进行处理。这种方式简单有效，**缺点就是每个group都会收到很多自己不感兴趣的垃圾数据**。

**方案二：** 把每个事件的数据推送到不同的topic中，即以事件名称来作为topic分类，在Consumer端，建立自己的group来消费自己感兴趣的一组topic。这种方式适用于事件个数可以明确评估并且数量较少，如果事件种类很多，会导致topic的数量过多，创建过多的topic和partition则会影响到Kafka的性能，因为Kafka的每个Topic、每个分区都会对应一个物理文件，当Topic数量增加时，消息分散的落盘策略会导致磁盘IO竞争激烈成为瓶颈。



**方案三：**采用流处理方式对数据进行分类，即增加一个中间数据流处理，将数据按照订阅规则进行归类，然后写入不同的topic中，在Consumer端，每个group可以拿到仅仅是自己感兴趣的数据。这种方式适用于数据量较大、但是Consumer端的消费group有限的情况，否则也会出现上述的topic碎片化的问题。



**方案四：**自己做partition的分配，但是不容易控制，应尽量避免。

### Kafka三种模式（具体实现要做深入探讨）

**默认配置**

采用默认配置情况下，既不能完全保证At-least-once 也不能完全保证at-most-once；比如：在自动提交之后，数据消费流程失败，这样就会有丢失，不能保证at-least-once；数据消费成功，但是自动提交失败，可能会导致重复消费，这样也不能保证 at-most-once；但是将自动提交时长设置得足够小，则可以最大限度地保证at-most-once；

**at most onece模式**

基本思想是保证每一条消息commit成功之后，再进行消费处理；

设置自动提交为false，接收到消息之后，首先commit，然后再进行消费

**at least onece模式**

基本思想是保证每一条消息处理成功之后，再进行commit；

设置自动提交为false；消息处理成功之后，手动进行commit；

采用这种模式时，最好保证消费操作的“幂等性”，防止重复消费；

**exactly onece模式**

核心思想是将offset作为唯一id与消息同时处理，并且保证处理的原子性；

设置自动提交为false；消息处理成功之后再提交；

比如对于关系型数据库来说，可以将id设置为消息处理结果的唯一索引，再次处理时，如果发现该索引已经存在，那么就不处理

## **Kafka常见面试问题**

1、为什么要使用 kafka，为什么要使用消息队列

 缓冲和削峰  解耦和扩展性  冗余  健壮性  异步通信

2、Kafka的acks参数对消息持久化的影响

往kafka写数据的时候，就可以来设置这个acks参数。然后这个参数实际上有三种常见的值可以设置，分别是：0、1 和 all（等价于-1）。

API对应ProducerConfig.ACKS\_CONFIG

第一种选择是把**acks参数设置为0，意思就是我的KafkaProducer在客户端，只要把消息发送出去，不管那条数据有没有在哪怕Partition Leader上落到磁盘，我就不管他了**，直接就认为这个消息发送成功了。

如果你采用这种设置的话，那么你必须注意的一点是，可能你发送出去的消息还在半路。结果呢，Partition Leader所在Broker就直接挂了，然后结果你的客户端还认为消息发送成功了，此时就会导致这条消息就丢失了。

第二种选择是设置 **acks = 1，意思就是说只要Partition Leader接收到消息而且写入本地磁盘了，就认为成功了**，不管他其他的Follower有没有同步过去这条消息了。

**这种设置其实是kafka默认的设置，大家请注意，划重点！这是默认的设置**

也就是说，默认情况下，你要是不管acks这个参数，只要Partition Leader写成功就算成功。

但是这里有一个问题，万一Partition Leader刚刚接收到消息，Follower还没来得及同步过去，结果Leader所在的broker宕机了，此时也会导致这条消息丢失，因为人家客户端已经认为发送成功了。

最后一种情况，**就是设置acks=all，这个意思就是说，Partition Leader接收到消息之后**，还**必须要求ISR列表里跟Leader保持同步的那些Follower都要把消息同步过去，才能认为这条消息是写入成功了**。

如果说Partition Leader刚接收到了消息，但是结果Follower没有收到消息，此时Leader宕机了，那么客户端会感知到这个消息没发送成功，他会重试再次发送消息过去。

此时可能Partition 2的Follower变成Leader了，此时ISR列表里只有最新的这个Follower转变成的Leader了，那么只要这个新的Leader接收消息就算成功了。

3、kafka幂等性

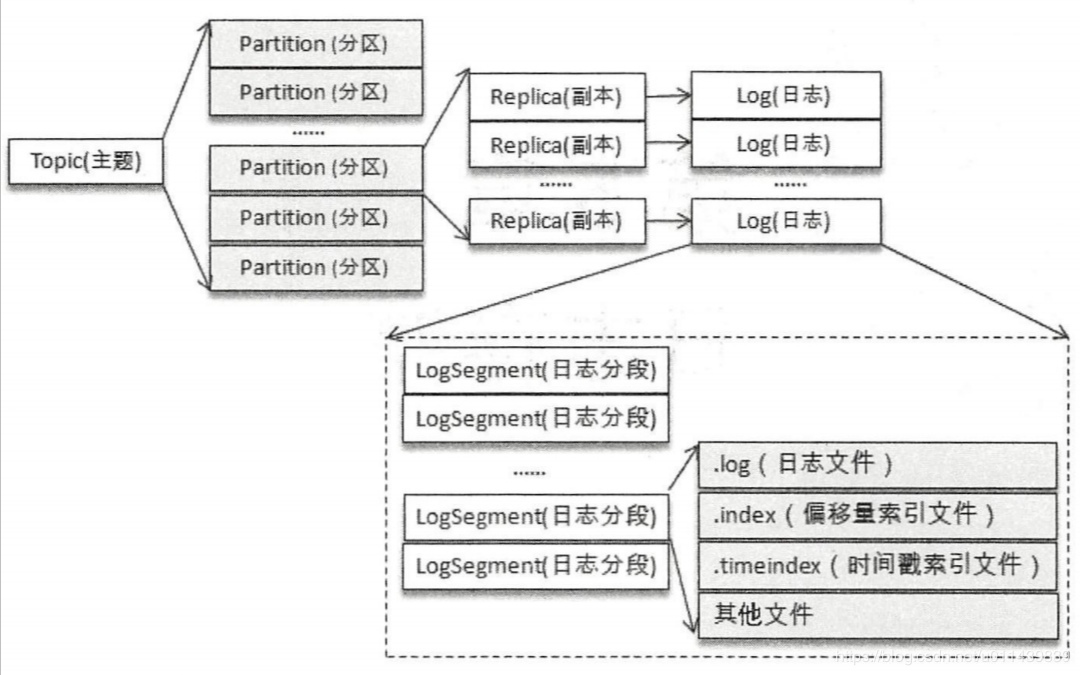
解决发送消息重复的问题

4、kafka的分区与消费者关系

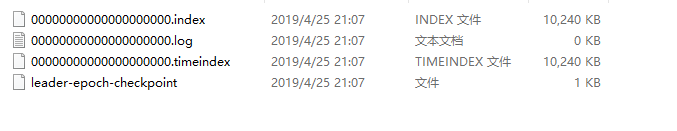
分区（partition）  
kafka中的topic可以细分为不同的partition，一个topic可以将消息存放在不同的partion中。

leader和follower  
每个partition可以设置一个leader和多个follower。kafka的消息没有设置读写分离，每个消息发送时，都是发送至对应的partition的leader-partition，follower-partition主要是为了备份数据而存在，当leader-partition出现故障时，数据已经完全同步的follower-partition也会切换成leader-partition。

AR和ISR  
**AR：分区中所有的副本统称为AR。**  
**ISR：所有与leader节点保持同步的副本（包括leader节点）组成的节点**，生产者首先将消息发送给leader副本，然后follower从leader中同步消息。  
ISR是AR的子集。

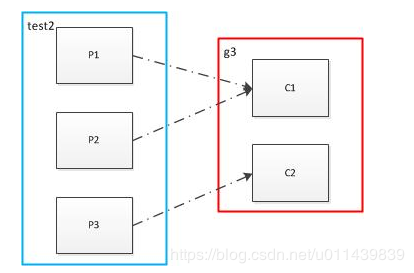
数据的存储  
在partion中，一个topic中的数据存放在不同的partion中，一个分区的内容会存储成一个log文件，为了防止log过大，引入了日志分段，根据一定规则将log切分为多个logSegment，相当于一个巨型文件被切分成了很多不同的文件。log和logSegment关系如下：  


 Log在物理上只以文件夹的形式存储,日志文件在磁盘的存储如下：

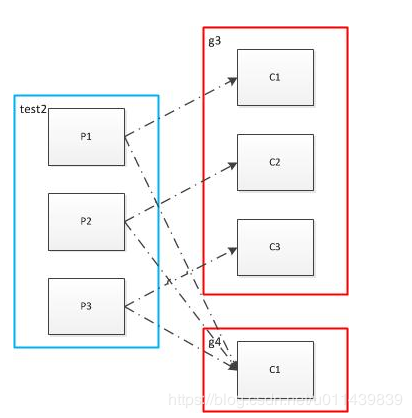


消费者（consumer）  
分组（group）  
消费者从partion中消费数据，consumer有group的概念，每个group可以消费完整的一份topic中的数据。

消费者分区分配  
RangeAssignor 分配策略（kafka默认的分区策略）  
kafka中，consumer消费数据时，consumer消费partition中的数据和consumer与partition的相对数量有关。

1、partition大于consumer  
当partition大于consumer时，一个consumer可能同时消费多个partition中的数据  


 2、partition小于consumer  
当partition小于consumer时，每个consumer会去消费一个partition中的数据。多出来的consumer则是处于空闲状态



3、partition等于consumer  
当partition的数量等于consumer时，每个consumer消费一个partition中的数据。

除了这种分区分配策略之外，还有RoundRobinAssignor以及StickyAssignor分区分配策略，当然，也可以自定义分区分配策略。