Kafka怎么保证数据不丢失，不重复?

可回答：Kafka如何保证生产者不丢失数据，消费者不丢失数据?

参考答案：

存在数据丢失的几种情况

- 使用同步模式的时候，有3种状态保证消息被安全生产，在配置为1（只保证写入leader成功）的话，如果刚好leader partition挂了，数据就会丢失。
- 还有一种情况可能会丢失消息，就是使用异步模式的时候，当缓冲区满了，如果配置为0（还没有收到确认的情况下，缓冲池一满，就清空缓冲池里的消息），数据就会被立即丢弃掉。

避免方法的一些概述

1、在数据生产时避免数据丢失的方法

只要能避免上述两种情况，那么就可以保证消息不会被丢失。

1）在同步模式的时候，确认机制设置为-1，也就是让消息写入leader和所有的副本。

2）在异步模式下，如果消息发出去了，但还没有收到确认的时候，缓冲池满了，在配置文件中设置成不限制阻塞超时的时间，也就说让生产端一直阻塞，这样也能保证数据不会丢失。

在数据消费时，避免数据丢失的方法：确认数据被完成处理之后，再更新offset值。低级API中需要手动控制offset值。

消息队列的问题都要从源头找问题，就是生产者是否有问题。

讨论一种情况，如果数据发送成功，但是接受response的时候丢失了，机器重启之后就会重发。

重发很好解决，消费端增加去重表就能解决，但是如果生产者丢失了数据，问题就很麻烦了。

2、数据重复消费的情况，处理情况如下

1）去重：将消息的唯一标识保存到外部介质中，每次消费处理时判断是否处理过；

2）不管：大数据场景中，报表系统或者日志信息丢失几条都无所谓，不会影响最终的统计分析结。

Kafka到底会不会丢数据(data loss)? 通常不会，但有些情况下的确有可能会发生。下面的参数配置及Best practice列表可以较好地保证数据的持久性(当然是trade-off，牺牲了吞吐量)。

如果想要高吞吐量就要能容忍偶尔的失败（重发漏发无顺序保证）。

```
block.on.buffer.full = true
acks = all
retries = MAX_VALUE
max.in.flight.requests.per.connection = 1
使用KafkaProducer.send(record, callback)
callback逻辑中显式关闭producer：close(0)
unclean.leader.election.enable=false
replication.factor = 3
min.insync.replicas = 2
replication.factor > min.insync.replicas
enable.auto.commit=false
```

消息处理完成之后再提交位移

给出列表之后，我们从两个方面来探讨一下数据为什么会丢失：

`Producer端`

新版本的Kafka替换了Scala版本的old producer，使用了由Java重写的producer。新版本的producer采用异步发送机制。KafkaProducer.send(ProducerRecord)方法仅仅是把这条消息放入一个缓存中(即RecordAccumulator，本质上使用了队列来缓存记录)，同时后台的IO线程会不断扫描该缓存区，将满足条件的消息封装到某个batch中然后发送出去。显然，**这个过程中就有一个数据丢失的窗口**：若IO线程发送之前client端挂掉了，累积在accumulator中的数据的确有可能会丢失。

Producer的另一个问题是**消息的乱序问题**。假设客户端代码依次执行下面的语句将两条消息发到相同的分区

```
1   producer.send(record1);
2   producer.send(record2);
```

如果此时由于某些原因(比如瞬时的网络抖动)导致record1没有成功发送，同时Kafka又配置了重试机制和max.in.flight.requests.per.connection大于1(默认值是5，本来就是大于1的)，那么重试record1成功后，record1在分区中就在record2之后，从而造成消息的乱序。很多某些要求强顺序保证的场景是不允许出现这种情况的。发送之后重发就会丢失顺序。

鉴于producer的这两个问题，我们应该如何规避呢？？对于消息丢失的问题，很容易想到的一个方案就是：既然异步发送有可能丢失数据， 我改成同步发送总可以吧？比如这样：

```
1   producer.send(record).get();
```

这样当然是可以的，但是性能会很差，不建议这样使用。以下的配置清单应该能够比较好地规避producer端数据丢失情况的发生：(特此说明一下，软件配置的很多决策都是trade-off，下面的配置也不例外：应用了这些配置，你可能会发现你的producer/consumer 吞吐量会下降，这是正常的，因为你换取了更高的数据安全性)。

block.on.buffer.full = true 尽管该参数在0.9.0.0已经被标记为“deprecated”，但鉴于它的含义非常直观，所以这里还是显式设置它为true，使得producer将一直等待缓冲区直至其变为可用。否则如果producer生产速度过快耗尽了缓冲区，producer将抛出异常。缓冲区满了就阻塞在那，不要抛异常，也不要丢失数据。

acks=all 很好理解，所有follower都响应了才认为消息提交成功，即"committed"。

retries = MAX 无限重试，直到你意识到出现了问题。

max.in.flight.requests.per.connection = 1 限制客户端在单个连接上能够发送的未响应请求的个数。设置此值是1表示kafka broker在响应请求之前client不能再向同一个broker发送请求。注意：设置此参数是为了避免消息乱序。

使用KafkaProducer.send(record, callback)而不是send(record)方法 自定义回调逻辑处理消息发送失败，比如记录在日志中，用定时脚本扫描重处理。

callback逻辑中最好显式关闭producer：close(0) 注意：设置此参数是为了避免消息乱序（仅仅因为一条消息发送没收到反馈就关闭生产者，感觉代价很大）。

unclean.leader.election.enable=false 关闭unclean leader选举，即不允许非ISR中的副本被选举为leader，以避免数据丢失。

replication.factor >= 3 参考了Hadoop及业界通用的三备份原则。

min.insync.replicas > 1 消息至少要被写入到这么多副本才算成功，也是提升数据持久性的一个参数。与acks配合使用保证replication.factor > min.insync.replicas 如果两者相等，当一个副本挂掉了分区也就没法正常工作了。通常设置replication.factor = min.insync.replicas + 1即可。

Consumer端

consumer端丢失消息的情形比较简单：如果在消息处理完成前就提交了offset，那么就有可能造成数据的丢失。由于Kafka consumer默认是自动提交位移的，所以在后台提交位移前一定要保证消息被正常处理了，因此不建议采用很重的处理逻辑，如果处理耗时很长，则建议把逻辑放到另一个线程中去做。为了避免数据丢失，现给出两点建议：

- enable.auto.commit=false，关闭自动提交位移
- 在消息被完整处理之后再手动提交位移