详细讲Spark Streaming和Kafka之间重复消费漏消费的问题

参考答案：

1、数据丢失问题

1）receiver模式

丢失原因：

首先，receiver task 接收 kafka 中的数据，并备份到其他 executor 中的blockmanager里，然后将偏移量提交给 zookeeper ,接着 存在备份的 executor 将数据的地址封装并发送到 driver 中的 receiver tracker，然后由 driver 发送 task ，以及监测任务执行和回收结果。

在这个过程中，如果数据已经提交到了 zookeeper ，此时，driver 挂了，executor 也会被 kill 掉，当 driver 重启时，内存中就没有数据的地址信息了，而且kafka 会从新的偏移量处发送数据，即发生数据丢失。

解决方案：

开启 WAL 机制，在数据备份的时候，同时将数据拷贝一份到 hdfs ，等数据备份完成之后，再提交偏移量。同时，driver启动时，如果 hdfs 上存在未消费的数据，则先消费该数据。

这样，即使zookeeper 提交偏移量之后 driver 挂了，当driver重启之后，依旧能从hdfs 上消费数据。

存在问题：

开启WAL机制可能导致数据重复消费等问题。

2）direct模式

Spark Streaming 2.2 direct 模式采用的是Kafka的 simple consumer api，该情况下，偏移量可以手动管理，只要保证数据都消费之后再提交偏移量，就不存在数据丢失问题。

2、数据重复消费问题

1）receiver模式

原因：

开启 WAL 机制后，如果数据成功备份到 HDFS 之后，driver 挂了，此时偏移量还未提交给 Zookeeper，重启时，driver会先消费 HDFS 中的数据，由于偏移量未提交，该数据会再次接收并消费。

解决方案：

以receiver基于ZooKeeper的方式，当读取数据时去访问Kafka的元数据信息，在处理代码中例如 foreachRDD或
transform时，将信息写入到内存数据库中（memorySet），在计算时读取内存数据库信息，判断是否已处理过，如果以处理过则跳过计算。这些元数据信息可以保存到内存数据结构或者memsql，sqllite中。

2）direct模式

原因：

偏移量手动提交到 redis ，这种情况下，当数据处理完成之后，还未提交偏移量，此时如果发生故障，则会导致数据重复消费。

解决方案：

通过事务控制，如写一个 jdbc 事务。将业务逻辑，偏移量的提交放在一个事务中。

3、其它优化方式

1）采用 direct 模式代替 receiver 模式，在数据堆积、处理延迟、偏移量管理等的问题上direct 模式更具优势。

2）基于Spark Core优化。

- 增加并行度；
- 采用 Kryo 序列化机制，流失计算生成RDD可能会持久化到内存，默认持久化级别是memory_only_ser，默认就会减少GC开销；
- 采用 CMS垃圾回收器降低 GC 开销；
- 选择高性能的算子（mapPartitions,foreachPartitions,aggregateByKey等）；

3）repartition的使用

由于Spark Streaming程序中一批数据量一般较小，repartition 的时间短，可以解决一些由于topicpartition中数据分配不均匀导致的数据倾斜问题。

4）任务启动优化

如果每一秒钟启动的task数量太多，假设50个，即1s的batch时间间隔和50ms的block时间间隔。如果发送这些task去worker上的executor，那么性能开销会比较大，这是很难到到毫秒级的延迟了。

（1）task序列化

使用Kryo序列化类库来序列化task，可以减小task的大小从而减少driver发送这些task到各个executor的发送时间，即节省网络资源。

（2）执行模式

在standalone模式下运行Spark，可以达到更少的task启动时间。