Direct方式连接kafka数据不同于receiver方式，direct方式与kafka的partition一一对应，有多少个partition就有多少个rdd分区，offset以长整形变量的形式存在，不同应用各自维护自己的offset，默认是从最新数据开始读取。

1. receiver方式读取

这种方式使用Receiver来获取数据。Receiver是使用Kafka的高层次Consumer API来实现的。receiver从Kafka中获取的数据都是存储在Spark Executor的内存中的，然后Spark Streaming启动的job会去处理那些数据。

然而，在默认的配置下，这种方式可能会因为底层的失败而丢失数据。如果要启用高可靠机制，让数据零丢失，就必须启用Spark Streaming的预写日志机制（Write Ahead Log，WAL）。该机制会同步地将接收到的Kafka数据写入分布式文件系统（比如HDFS）上的预写日志中。所以，即使底层节点出现了失败，也可以使用预写日志中的数据进行恢复。

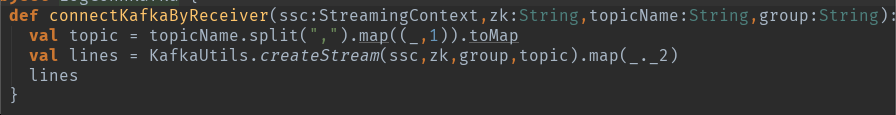
**需要注意的要点**

**1**、Kafka中的topic的partition，与Spark中的RDD的partition是没有关系的。所以，在KafkaUtils.createStream()中，提高partition的数量，只会增加一个Receiver中，读取partition的线程的数量。不会增加Spark处理数据的并行度。

**2**、可以创建多个Kafka输入DStream，使用不同的consumer group和topic，来通过多个receiver并行接收数据。

**3**、如果基于容错的文件系统，比如HDFS，启用了预写日志机制，接收到的数据都会被复制一份到预写日志中。因此，在KafkaUtils.createStream()中，设置的持久化级别是StorageLevel.MEMORY\_AND\_DISK\_SER

实现：



1. direct方式读取

这种新的不基于Receiver的直接方式，是在Spark 1.3中引入的，从而能够确保更加健壮的机制。替代掉使用Receiver来接收数据后，这种方式会周期性地查询Kafka，来获得每个topic+partition的最新的offset，从而定义每个batch的offset的范围。当处理数据的job启动时，就会使用Kafka的简单consumer api来获取Kafka指定offset范围的数据。

**这种方式有如下优点：**

1、简化并行读取：如果要读取多个partition，不需要创建多个输入DStream然后对它们进行union操作。Spark会创建跟Kafka partition一样多的RDD partition，并且会并行从Kafka中读取数据。所以在Kafka partition和RDD partition之间，有一个一对一的映射关系。

2、高性能：如果要保证零数据丢失，在基于receiver的方式中，需要开启WAL机制。这种方式其实效率低下，因为数据实际上被复制了两份，Kafka自己本身就有高可靠的机制，会对数据复制一份，而这里又会复制一份到WAL中。而基于direct的方式，不依赖Receiver，不需要开启WAL机制，只要Kafka中作了数据的复制，那么就可以通过Kafka的副本进行恢复。

3、一次且仅一次的事务机制：

基于receiver的方式，是使用Kafka的高阶API来在ZooKeeper中保存消费过的offset的。这是消费Kafka数据的传统方式。这种方式配合着WAL机制可以保证数据零丢失的高可靠性，但是却无法保证数据被处理一次且仅一次，可能会处理两次。因为receiver会周期性提交offset到zk，Spark和ZooKeeper之间可能是不同步的。

基于direct的方式，使用kafka的简单api，Spark Streaming自己就负责追踪消费的offset，并保存在checkpoint中。Spark自己一定是同步的，因此可以保证数据是消费一次且仅消费一次。