Spark企业级大数据项目实战 第3课

DATAGURU专业数据分析社区

法律声明

【声明】本视频和幻灯片为炼数成金网络课程的教学资料，所有资料只能在课程内使用，不得在课程以外范围散播，违者将可能被追究法律和经济责任。

课程详情访问炼数成金培训网站

http://edu.dataguru.cn

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

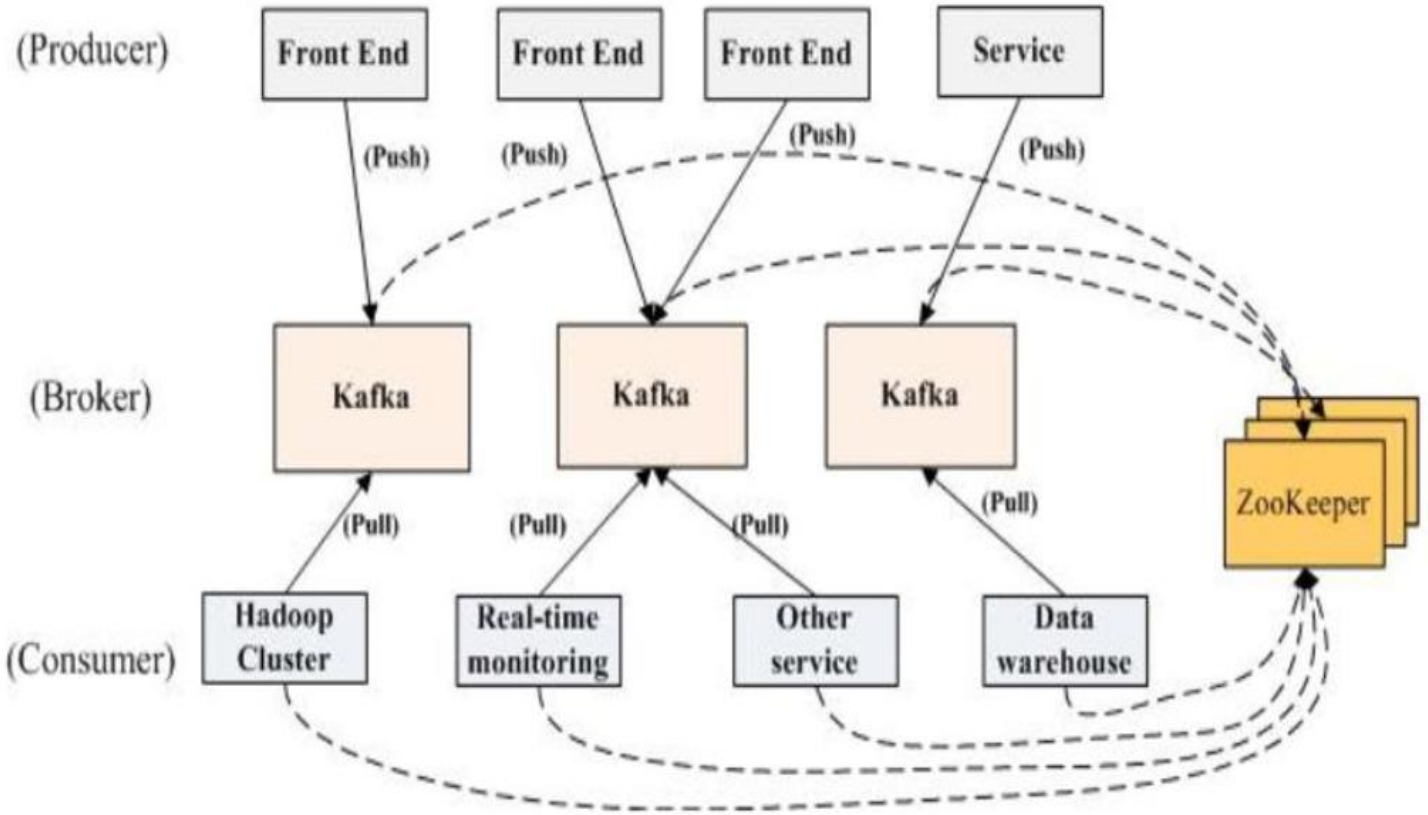
本课内容

- Kafka基础和核心功能
- Kafka消息检索过程
- Spark Streaming + Kafka整合
- Kafka的Offset管理（Checkpoint、Zookeeper、Hbase、Kafka）

Kafka-简介

Kafka是一种分布式的，基于发布/订阅的消息系统

- 高吞吐量： 每秒可处理几十万条记录
- 分布式： 支持热扩展
- 持久化： 消息持久化到磁盘
- 容错： 副本容错
- 高并发：客户端同时高并发读写



使用背景

- 峰值处理能力
- 统一接口服务
- 解耦

…

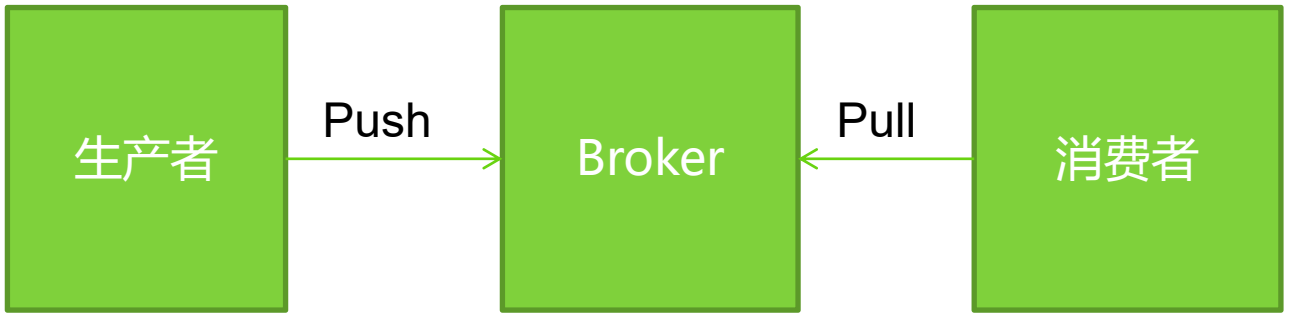
Kafka-测试环境安装

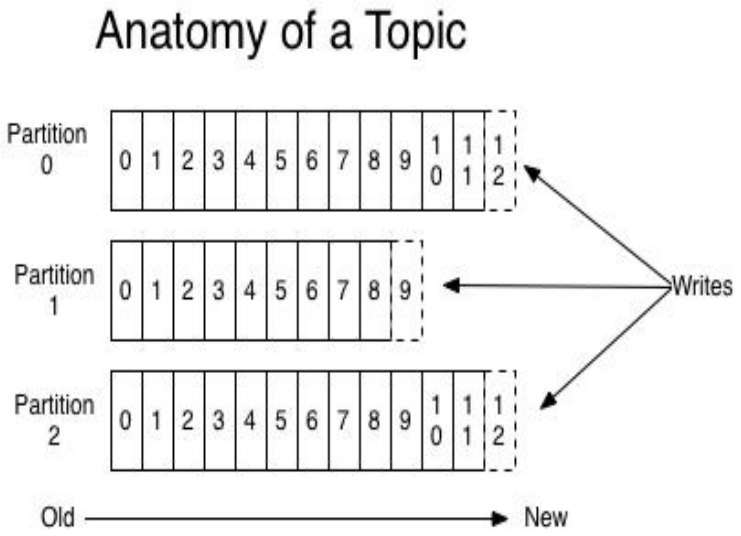
zookeeper安装、kafka安装

Kafka-相关概念

基本概念

- 生产者（Producer）：负责发布消息到Kafka broker
- 消费者（Consumer）：从消息队列中请求消息的客户端应用程序
- 代理（Broker）：Kafka集群包含一个或多个服务器，这种服务器被称为broker





Kafka-相关概念

Topic

- 在实际业务中， 通常一个业务对应一个topic
- Kafka使用topic来组织消息
- 一个topic消息可以包含多个partition， 分布在不同的broker上
- 一个partition可以指定多个副本
- 生产消息、订阅消息， 需要指定topic

Kafka-相关概念

Partition（分区）

- 一个topic按照多个分区组织消息
- 增加partition数量， 可以提升读写并发
- 一个partition对应的物理文件：log文件和index文件，每个log文件又称作segment，索引文件分offset索引文件和时间戳索引文件
- 一个partition可以指定多个副本， 但是只有一个副本是leader
- partiton的读写只能通过leader
- segment文件（log文件)文件名规范： 这个文件里面第一条消息的offset - 1

```
[hadoop@spark123 mytest1-0]$ ll
总用量 32
-rw-rw-r-- 1 hadoop hadoop 10485760 2月   4 17:55 00000000000000000000.index
-rw-rw-r-- 1 hadoop hadoop    24260 2月   4 18:02 00000000000000000000.log
-rw-rw-r-- 1 hadoop hadoop 10485756 2月   4 11:59 00000000000000000000.timeindex
[hadoop@spark123 mytest1-0]$
[hadoop@spark123 mytest1-0]$
```

Kafka-相关概念

Message（消息）

- 一个消息对应kafka的一个offset
- 消息只会追加到segment上面， 不能修改、删除
- segment会定期删除（配置项：配置项log.retention.{ms,minutes,hours}和log.retention.bytes）
- segment默认配置保留时间为7天

```
# from the end of the log.

# The minimum age of a log file to be eligible for deletion
log.retention.hours=168

# A size-based retention policy for logs. Segments are pruned from the log as long as the remaining
# segments don't drop below log.retention.bytes.
#log.retention.bytes=1073741824
```

Kafka-相关概念

Offset（偏移量）

- 一个消息对应kafka的一个offset
- offset是一个有序的序列
- offset最大长度为8字节

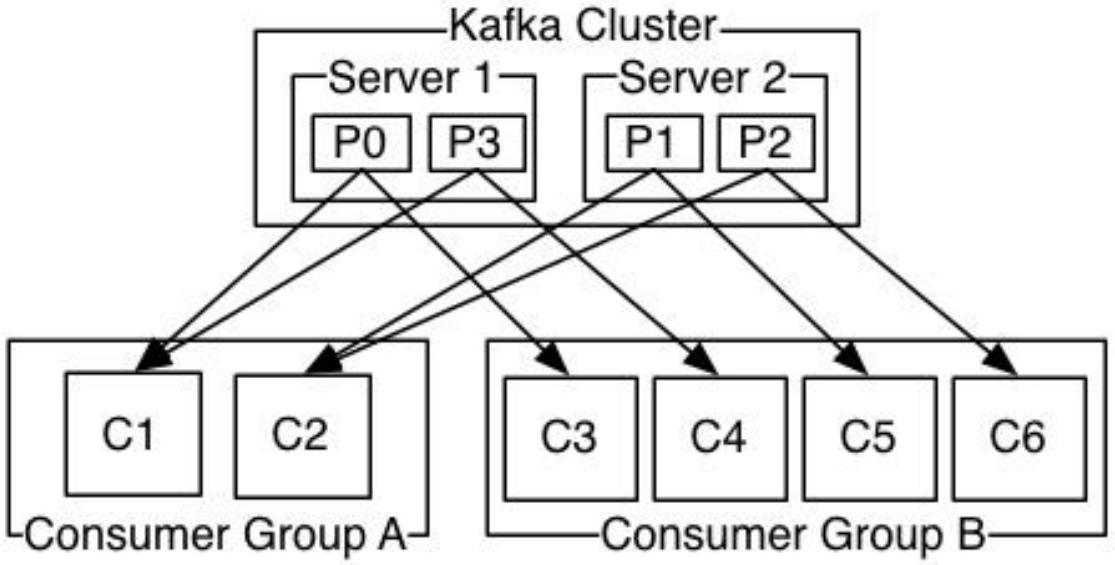
offset	实际消息
35000	abc
35001	deff
35002	aaa
35003	fdfer
…	…
39005	rere

Log File

Kafka-相关概念

Comsumer Group（消费组）

- 一个消费组可以由多个消费实例
- 每个消费组之间独立消费， 互不影响
- 一个消费组中的每个消费实例并行消费数据， 每个消费实例消费的数据，不会重复
- 根据partition数量，为每个消费组分配合理的并发数



Kafka-消费模型

- High Level Consumer API
  - 不需要自己管理offset
  - 默认实现最少一次消息传递语义（At least once）
  - comsumer数量 大于 partiton数量， 浪费。
  - comsumer数量 小于 partiton数量， 一个comsumer对应多个partiton
  - 最好partiton数目是consumer数目的整数倍

- Low Level Consumer API（Simple Consumer API）
  - 需要自己管理offset
  - 可以实现各种消息传递语义

Kafka-消息组织

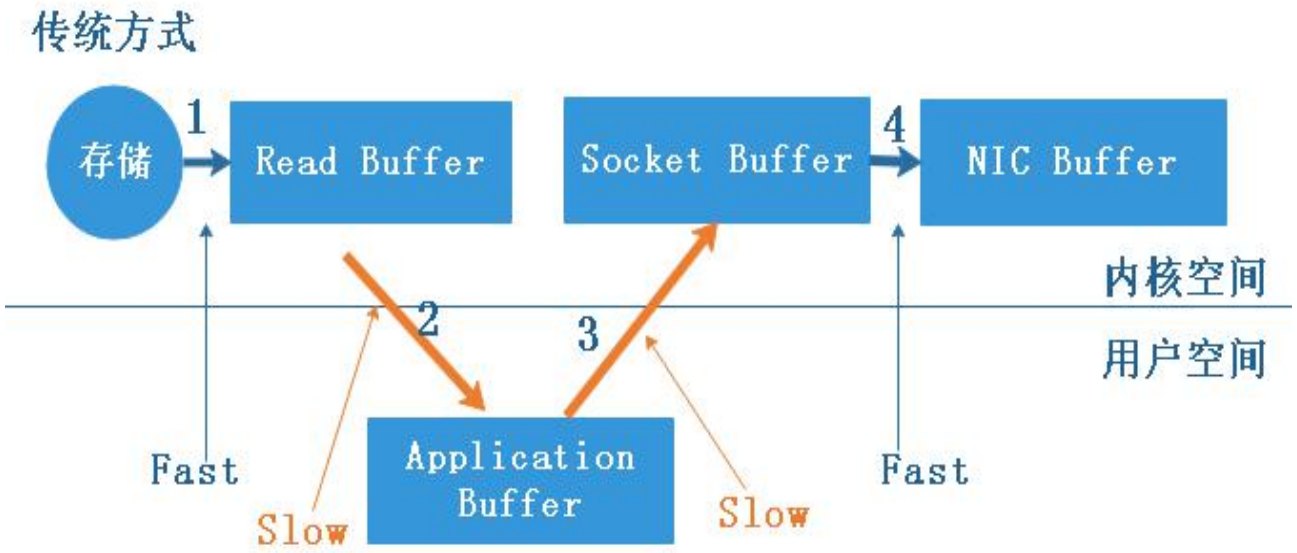
- 磁盘顺序读写（sequential disk access ）
  采用预读和批量大数据量写
  寻道

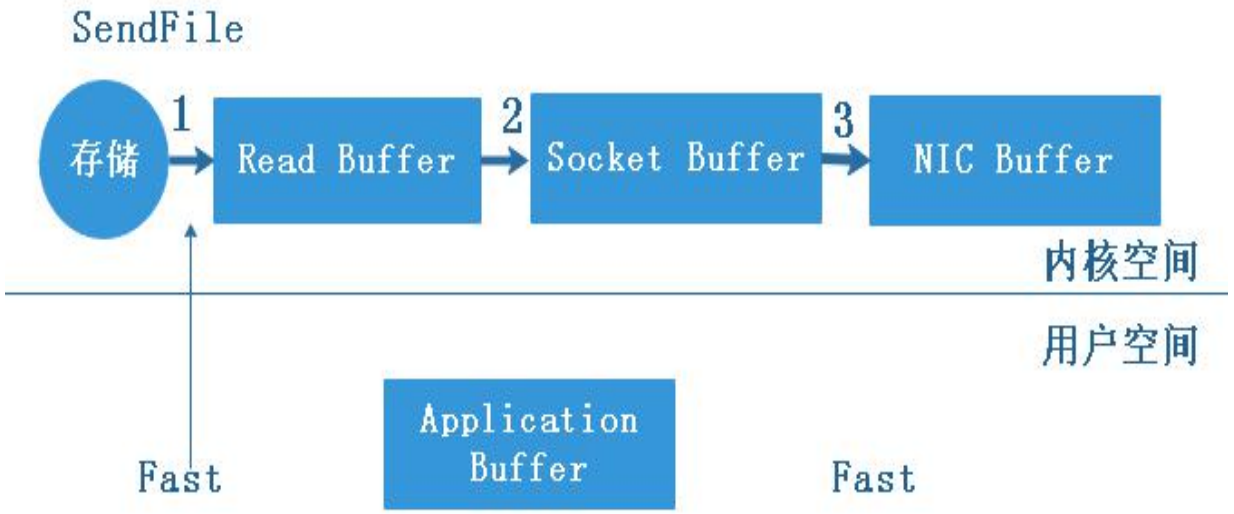
- 零字节拷贝（sendfile system call）

Kafka-零字节拷贝

传统：

1. 数据从磁盘读取到内核空间的pagecache中
2. 应用程序从内核空间读取数据到用户空间缓冲区
3. 应用程序将数据从内核空间写到套接字缓冲区
4. 从套接字缓冲区复制到NIC缓冲区





SendFile：

数据从内核空间复制到套接字缓冲区
从套接字缓冲区复制到NIC缓冲区

数据都是在内核空间传递， 效率高。
减少了两次拷贝

Kafka-消息检索原理

8 bytes offset	表示该消息在partition中的第几条消息
4 bytes message size	消息的大小
1 byte magic size	kafka程序服务协议号
4 bytes crc	crc32校验
其他	压缩、编码、key等信息
payload	实际消息的数据

Kafka-消息检索原理

```
adius-0]$ ll

   196304 Feb  1 22:19 00000000001557670354.index
536870081 Feb  1 22:19 00000000001557670354.log
        0 Feb  1 22:19 00000000001557670354.timeindex
   223688 Feb  2 01:00 00000000001558906210.index
536864928 Feb  2 01:00 00000000001558906210.log
        0 Feb  2 01:00 00000000001558906210.timeindex
   250968 Feb  2 04:02 00000000001560140916.index
536855285 Feb  2 04:02 00000000001560140916.log
        0 Feb  2 04:02 00000000001560140916.timeindex
   248464 Feb  2 07:03 00000000001561377216.index
536864802 Feb  2 07:03 00000000001561377216.log
        0 Feb  2 07:03 00000000001561377216.timeindex
   179640 Feb  2 09:11 00000000001562614860.index
536861548 Feb  2 09:11 00000000001562614860.log
        0 Feb  2 09:11 00000000001562614860.timeindex
```

序号	地址
1	0
3	245
5	456
9	678
...	
N	

00000000001560140916.index

消息offset	文件中偏移字节
msg1560140917	0
msg1560140918	234
msg1560140919	245
msg1560140920	289
msg1560140921	456
msg1560140922	480
msg1560140923	500
msg1560140924	580
msg1560140925	678
...	
msg1560140917+N	positionx

00000000001560140916.log文件

index文件的序号就是message在日志文件中的相对偏移量

OffsetIndex是稀疏索引，也就是说不会存储所有的消息的相对offset和position

Kafka-消息检索过程

以这个partition目录下面，00000000001560140916为例
定位offset 为1560140921的message
1. 定位到具体的segment日志文件
由于log日志文件的文件名是这个文件中第一条消息的offset-1.
因此可以根据offset定位到这个消息所在日志文件：00000000001560140916.log

2. 计算查找的offset在日志文件的相对偏移量
segment文件中第一条消息的offset = 1560140917
计算message相对偏移量：需要定位的offset - segment文件中第一条消息的offset + 1 = 1560140921 - 1560140917 + 1 = 5

查找index索引文件， 可以定位到该消息在日志文件中的偏移字节为456.

综上， 直接读取文件夹00000000001560140916.log中偏移456字节的数据即可。

1560140922 -1560140917 + 1 = 6
如果查找的offset在日志文件的相对偏移量在index索引文件不存在， 可根据其在index索引文件最接近的上限偏移量， 往下顺序查找

Receiver-based Approach

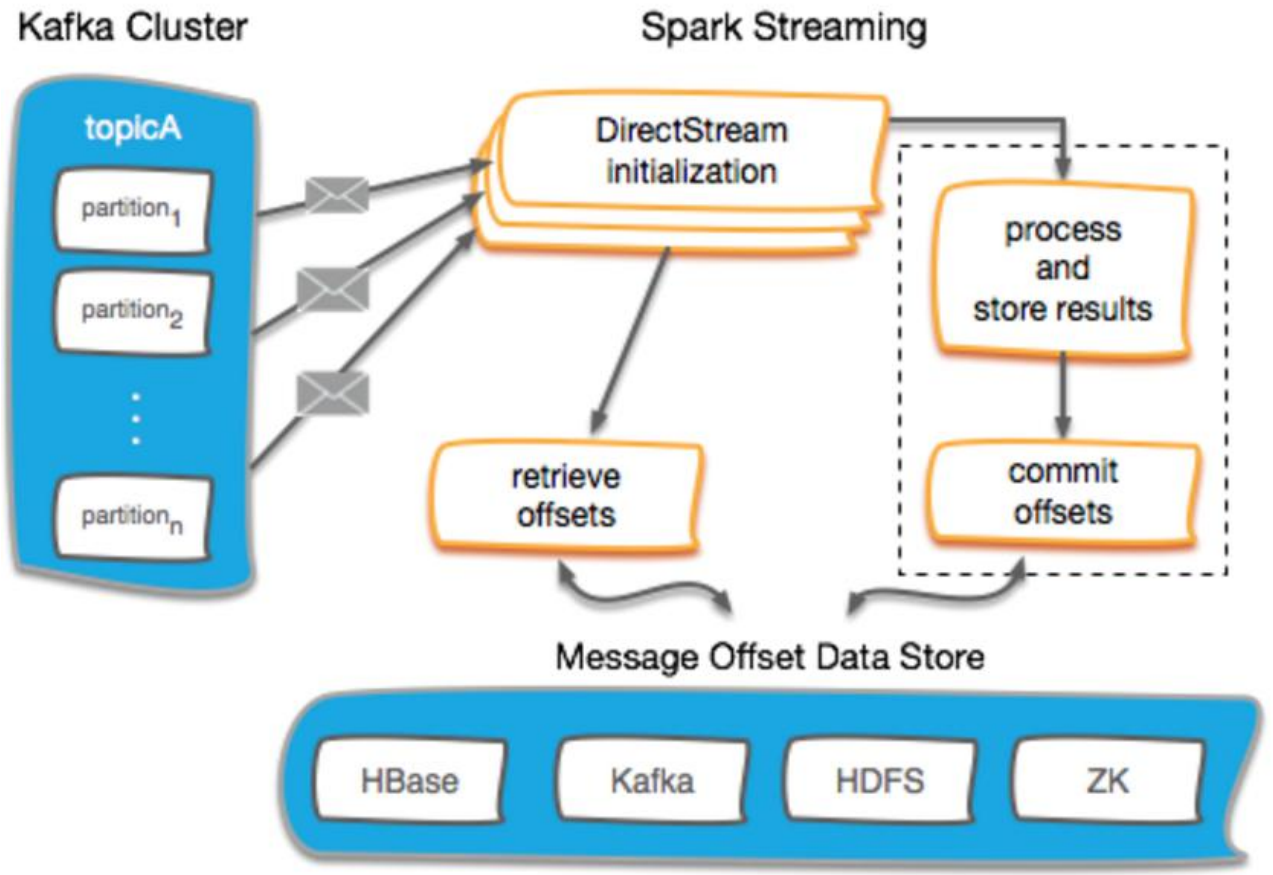
- Kafka的topic分区和Spark Streaming中生成的RDD分区没有关系。 在KafkaUtils.createStream中增加分区数量只会增加单个receiver的线程数， 不会增加Spark的并行度
- 可以创建多个的Kafka的输入DStream， 使用不同的group和topic， 使用多个receiver并行接收数据。
- 如果启用了HDFS等有容错的存储系统， 并且启用了写入日志，则接收到的数据已经被复制到日志中。因此，输入流的存储级别设置StorageLevel.MEMORY_AND_DISK_SER（即使用KafkaUtils.createStream（...，StorageLevel.MEMORY_AND_DISK_SER））的存储级别。

Direct Approach (No Receivers)

- 简化的并行性：不需要创建多个输入Kafka流并将其合并。 使用directStream，Spark Streaming将创建与使用Kafka分区一样多的RDD分区，这些分区将全部从Kafka并行读取数据。 所以在Kafka和RDD分区之间有一对一的映射关系。
- 效率：在第一种方法中实现零数据丢失需要将数据存储在预写日志中，这会进一步复制数据。 这实际上是效率低下的，因为数据被有效地复制了两次 - 一次是Kafka，另一次是由预先写入日志（Write Ahead Log）复制。 这个第二种方法消除了这个问题，因为没有接收器，因此不需要预先写入日志。只要Kafka数据保留时间足够长。
- 正好一次（Exactly-once）的语义：第一种方法使用Kafka的高级API来在Zookeeper中存储消耗的偏移量。传统上这是从Kafka消费数据的方式。虽然这种方法（结合提前写入日志）可以确保零数据丢失（即至少一次语义），但是在某些失败情况下，有一些记录可能会消费两次。发生这种情况是因为Spark Streaming可靠接收到的数据与Zookeeper跟踪的偏移之间的不一致。因此，在第二种方法中，我们使用不使用Zookeeper的简单Kafka API。在其检查点内，Spark Streaming跟踪偏移量。这消除了Spark Streaming和Zookeeper / Kafka之间的不一致，因此Spark Streaming每次记录都会在发生故障的情况下有效地收到一次。为了实现输出结果的一次语义，将数据保存到外部数据存储区的输出操作必须是幂等的，或者是保存结果和偏移量的原子事务。

Kafka Offset 管理

- 使用外部存储保存offset
  - Checkpoints
  - HBase
  - ZooKeeper
  - Kafka
  - ...
- 不保存offset



Kafka Offset 管理--Checkpoint

1. 启用Spark Streaming的checkpoint是存储偏移量最简单的方法。

2. 流式checkpoint专门用户保存应用程序的状态， 比如保存在HDFS上，
   在故障时能恢复

3. Spark Streaming的checkpoint无法跨越应用程序进行恢复

4. Spark 升级也将导致无法恢复

5. 在关键生产应用， 不建议使用spark检查的管理offset

Kafka Offset 管理--Zookeeper

1. 路径：
 val zkPath = s"${kakfaOffsetRootPath}/${groupName}/${o.topic}/${o.partition}"

2. 如果Zookeeper中未保存offset,根据kafkaParam的配置使用最新或者最旧的offset

3.如果 zookeeper中有保存offset,我们会利用这个offset作为kafkaStream 的起始位置

```
[zk: spark123:12181(CONNECTED) 14] ls /kafka0.9/mykafka/consumers/offsets/testp/mytest1
[0, 1, 2]
[zk: spark123:12181(CONNECTED) 15]
[zk: spark123:12181(CONNECTED) 15]
[zk: spark123:12181(CONNECTED) 15] get /kafka0.9/mykafka/consumers/offsets/testp/mytest1/0
11164
cZxid = 0x1810
ctime = Mon Feb 05 04:00:08 CST 2018
mZxid = 0x1854
mtime = Mon Feb 05 04:00:56 CST 2018
pZxid = 0x1810
cversion = 0
dataVersion = 22
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 5
numChildren = 0
[zk: spark123:12181(CONNECTED) 16]
```

Kafka Offset 管理--Hbase

1. 基于Hbase的通用设计， 使用同一表保存可以跨越多个spark streaming程序的topic的offset

2. rowkey = topic名称 + groupid + streaming的batchtime.milliSeconds . 尽管batchtime.milliSeconds不是必须的， 但是它可以看到历史的批处理任务对offset的管理情况。

3. kafka的offset保存在下面的表中， 30天后自动过期
Hbase表结构
create 'spark_kafka_offsets', {NAME=>'offsets', TTL=>2592000}

4.offset的获取场景
场景1：Streaming作业首次启动。 通过zookeeper来查找给定topic中分区的数量，然后返回“0”作为所有topic分区的offset。

场景2：长时间运行的Streaming作业已经停止，新的分区被添加到kafka的topic中。 通过zookeeper来查找给定topic中分区的数量， 对于所有旧的topic分区，将offset设置为HBase中的最新偏移量。 对于所有新的topic分区，它将返回“0”作为offset。

场景3：长时间运行的Streaming作业已停止，topic分区没有任何更改。 在这种情况下，HBase中发现的最新偏移量作为每个topic分区的offset返回。

Kafka Offset 管理--Hbase

```
hbase(main):009:0> scan 'spark_kafka_offsets'
ROW                                   COLUMN+CELL
 mytest1:testp:1517775582000          column=offsets:0, timestamp=1517775582690, value=11164
 mytest1:testp:1517775582000          column=offsets:1, timestamp=1517775582690, value=11147
 mytest1:testp:1517775582000          column=offsets:2, timestamp=1517775582690, value=11149
 mytest1:testp:1517775606000          column=offsets:0, timestamp=1517775606048, value=11194
 mytest1:testp:1517775606000          column=offsets:1, timestamp=1517775606048, value=11176
 mytest1:testp:1517775606000          column=offsets:2, timestamp=1517775606048, value=11178
 mytest1:testp:1517775612000          column=offsets:0, timestamp=1517775612055, value=11224
 mytest1:testp:1517775612000          column=offsets:1, timestamp=1517775612055, value=11205
 mytest1:testp:1517775612000          column=offsets:2, timestamp=1517775612055, value=11207
 mytest1:testp:1517775616000          column=offsets:0, timestamp=1517775616086, value=11254
 mytest1:testp:1517775616000          column=offsets:1, timestamp=1517775616086, value=11234
 mytest1:testp:1517775616000          column=offsets:2, timestamp=1517775616086, value=11236
4 row(s) in 0.0490 seconds
```

Kafka Offset 管理--Kafka

```
stream.foreachRDD { rdd =>
  val offsetRanges =
rdd.asInstanceOf[HasOffsetRanges].offsetRanges
  // some time later, after outputs have completed

stream.asInstanceOf[CanCommitOffsets].commitAsync(off
setRanges)
}
```

http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html#kafka-itself

Kafka Offset 管理--HDFS等

1. 可以将offset保存在HDFS上

2. 与其他系统（Zookeeper、Hbase）相比， HDFS具有更高的延迟。 此外， 如果管理不当， 在HDFS上写入每个批次的offsetRanges可能会导致小文件问题

Kafka Offset 管理--不保存offset

根据业务需要是否管理offset

对于一些streaming应用程序， 如实时活动监控， 只需要当前最新的数据， 这种情况不需要管理offset 。

在这种场景下， 如果使用老的kafka的api， 可以将参数auto.offset.reset设置为largest 或者smallest 。 如果使用新的kafka的api, 可以将参数 auto.offset.reset设置为earliest 或者latest 。

Thanks

FAQ时间