Day23_Kafka

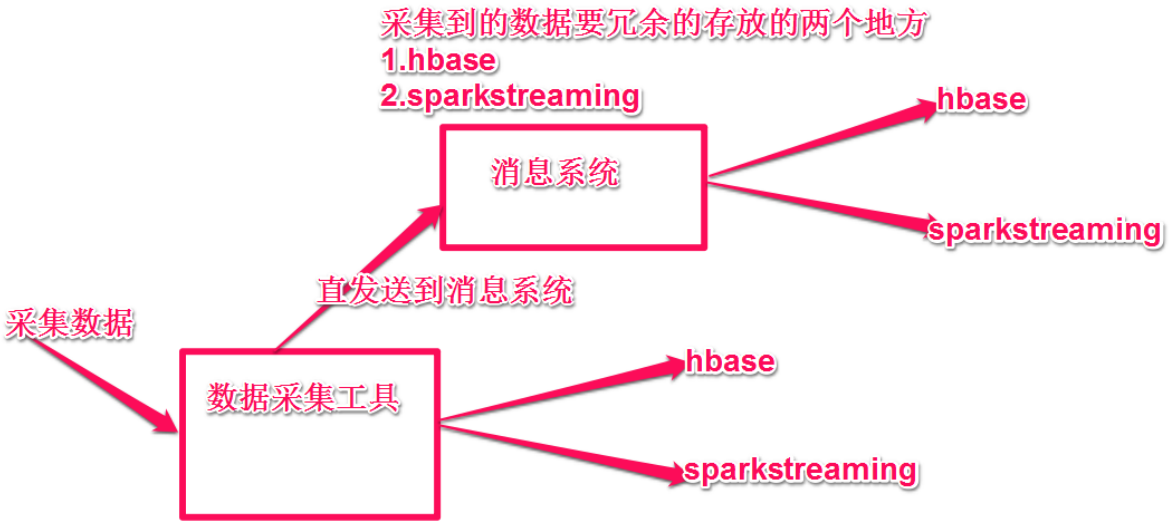
大数据-张军锋 Day23 kafka 安装 操作

Day23_Kafka
- 批量数据和流数据的区别
- Kafka简介
- kafka架构
- 为什么使用消息系统
- kafka安装
- kafka容错机制
- kafka命令介绍
- kafka删除topic
- kafka topic结构
- Kafka delivery guarantee

批量数据和流数据的区别

- 批量数据
  - 定期产生
  - 数量相对比较大
  - 有特定的生成周期
  - 比如：日志文件、业务数据
  - 去向—>分布式系统
  - 来源—>定期产生的文件、定期传输的文件、定期数据库导出
- 流数据
  - 随时都有可能产生
  - 搜集工具7*24小时运行

- 数据量相对比较小
- 实时性强
- 比如：实时数据分析日志数据、实时业务数据
- 去向—>分布式文件系统、流计算应用程序
- 来源—>网络端口、文件、数据库(文件和数据库是通过监听程序监听，然后将数据发送到网络端口，然后再通过流采集工具采集)



Kafka简介

Kafka是一种分布式的，基于发布/订阅的消息系统。主要设计目标如下：

- 以时间复杂度为O(1)的方式提供**消息持久化**能力，即使对TB级以上数据也能保证常数时间复杂度的访问性能。
- **高吞吐率**。即使在非常廉价的商用机器上也能做到单机支持每秒100K条以上消息的传输。
- 支持Kafka Server间的消息分区，及**分布式**消费，同时保证每个Partition内的消息顺序传输。
- **同时支持离线数据处理和实时数据处理。**
- Scale out：**支持在线水平扩展。**

kafka架构

- **broker** 是专门提供消息的读写操作的，并且能够保存消息
- **topic** 是对消息进行分类的
- **partition** 一个partition只能有一个broker负责，但是一个topic可以包含多个partition
- **preducer** 生产者，发送消息
- **consumer** 消费者，获取数据
- **consumer Group** 消费者组，确保消费的数据不重复，提升消费效率

> Kafka 是一个基于分布式的消息发布-订阅系统，它被设计成快速、可扩展的、持久的。与其他消息发布-订阅系统类似，Kafka 在主题当中保存消息的信息。生产者向主题写入数据，消费者从主题读取数据。由于 Kafka 的特性是支持分布式，同时也是基于分布式的，所以主题也是可以在多个节点上被分区和覆盖的。

为什么使用消息系统

• 解耦
在项目启动之初来预测将来项目会碰到什么需求，是极其困难的。消息系统在处理过程中间插入了一个隐含的、基于数据的接口层，两边的处理过程都要实现这一接口。这允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束。

• 冗余
有些情况下，处理数据的过程会失败。除非数据被持久化，否则将造成丢失。消息队列把数据进行持久化直到它们已经被完全处理，通过这一方式规避了数据丢失风险。许多消息队列所采用的”插入-获取-删除”范式中，在把一个消息从队列中删除之前，需要你的处理系统明确的指出该消息已经被处理完毕，从而确保你的数据被安全的保存直到你使用完毕。

• 扩展性
因为消息队列解耦了你的处理过程，所以增大消息入队和处理的频率是很容易的，只要另外增加处理过程即可。不需要改变代码、不需要调节参数。扩展就像调大电力按钮一样简单。

• 灵活性 & 峰值处理能力
在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见；如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。

• 可恢复性
系统的一部分组件失效时，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。

• 顺序保证
在大多使用场景下，数据处理的顺序都很重要。大部分消息队列本来就是排序的，并且能保证数据会按照特定的顺序来处理。Kafka保证一个Partition内的消息的有序性。

• 缓冲
在任何重要的系统中，都会有需要不同的处理时间的元素。例如，加载一张图片比应用过滤器花费更少的时间。消息队列通过一个缓冲层来帮助任务最高效率的执行———写入队列的处理会尽可能的快速。该缓冲有助于控制和优化数据流经过系统的速度。

• 异步通信

很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。想向队列中放入多少消息就放多少，然后在需要的时候再去处理它们。

kafka安装

1. 解压 `tar -zxvf kafka_2.11-0.10.1.1.tgz`
2. 配置环境变量

```
# set kafka enviroment
export KAFKA_HOME=/opt/software/kafka/kafka_2.11-0.10.1.1
export PATH=$PATH:$KAFKA_HOME/bin
```

3. 设置broker.id，整个集群中broker.id不能相同

```
# The id of the broker. This must be set to a unique integer for ea
ch broker.
broker.id=0
```

4. 设置端口号，如果在一台机器上布置多个kafka，需要改端口，否则，默认即可

```
############################# Socket Server Settings
#############################

# The address the socket server listens on. It will get the value r
eturned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = security_protocol://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://:9093
```

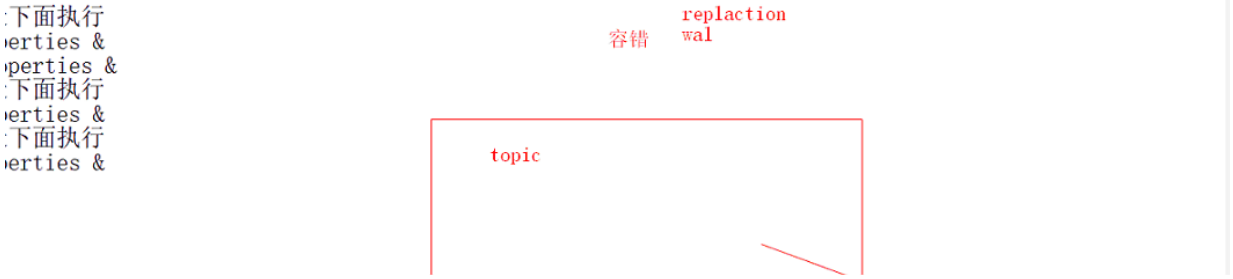
5. 指定日志位置，如果在同一台机器上指定多个kafka，需要设置日志的另一个输出目录，否则，默认即可

```
# A comma seperated list of directories under which to store log fi
les
log.dirs=/opt/software/kafka/kafka_2.11-0.10.1.1/kafka-logs
```

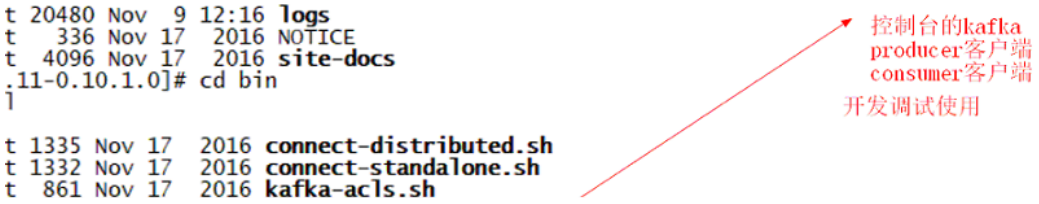
启动kafka `kafka-server-start.sh server.properties &`，&符号是后天启动

kafka容错机制

kafka是采用副本的方式进行容错的。



kafka命令介绍



```
// 创建topic
kafka-topics.sh --zookeeper master:2181 --create --partitions 2 --r
eplication-factor 3 --topic bd14first

// 列出所有的topic
kafka-topics.sh --zookeeper master:2181 --list

// 查看bd14first的详细信息
kafka-topics.sh --zookeeper master:2181 --describe --topic bd14firs
t


// 列举客户端指令信息
kafka-console-producer.sh

// 连接topic
kafka-console-producer.sh --broker-list slaver1:9092,slaver2:9092 -
-topic bd14first


// 列举消费者指令信息
kafka-console-producer.sh

// 从下次输入开始消费
kafka-console-consumer.sh --bootstrap-server slaver1:9092 slaver2:9
092 --topic bd14first

// 从头开始消费
kafka-console-consumer.sh --bootstrap-server slaver1:9092 slaver2:9
092 --topic bd14first --from-beginning

// 从特定的partition上进行消费 --offset earliest表示从开始处消费
kafka-console-consumer.sh --bootstrap-server slaver1:9092 slaver2:9
092 --topic bd14first --offset earliest --partition 1

// 只是标记删除topic,如果想真的删除，在server。propert文件中设置delete.top
ic.enable为true，重启即可
// 还可以在zookeeper中删除相对应的topic
kafka-topics.sh --zookeeper master:2181 --delete --topic bd14first

// 进入zookeeper客户端
zkCli.sh

// 列举出zookeeper中的文件
ls /

// 列举出brokers中的文件
ls /brokers
```

kafka删除topic

> 默认情况下删除操作，只是打个标记，在重新启动kafka的时候，才会删除。如果想要立即删除，有两种方式。

方式一：在配置文件[conf/server.properties]中开启立即删除操作

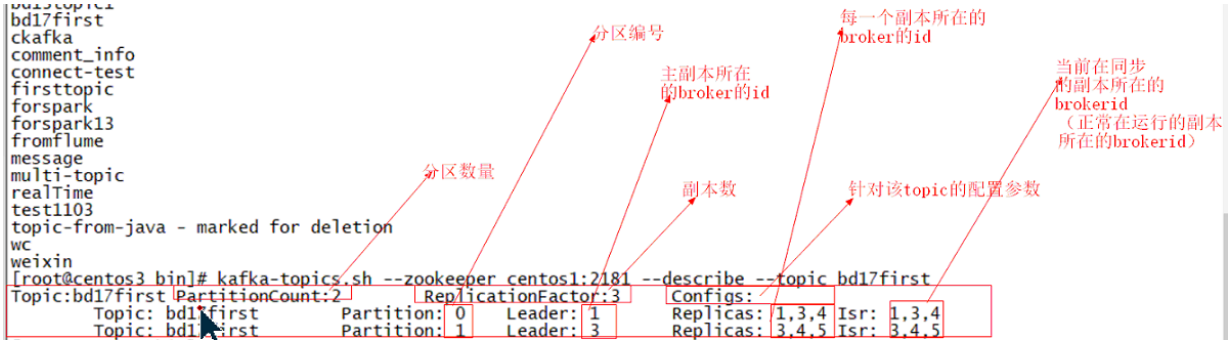
```
# Licensed to the Apache Software Foundation (ASF) under one or more
# contributor license agreements.  See the NOTICE file distributed with
# this work for additional information regarding copyright ownership.
# The ASF licenses this file to You under the Apache License, Version 2.0
# (the "License"); you may not use this file except in compliance with
# the License.  You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# see kafka.server.KafkaConfig for additional details and defaults

############################# Server Basics #############################

# The id of the broker. This must be set to a unique integer for each broker.
broker.id=0

# Switch to enable topic deletion or not, default value is false
#delete.topic.enable=true

############################# Socket Server Settings #############################

# The address the socket server listens on. It will get the value returned from
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = security_protocol://host_name:port
"server.properties" 121L, 5336C                               24,1          Top
```


方式二:

1. zkCli.sh进入zookeeper客户端
2. 使用 `ls 路径` ，查看zookeeper
3. 删除 `deleteall /brokers/topics/kafkatest`

```
[zk: localhost:2181(CONNECTED) 3]
[zk: localhost:2181(CONNECTED) 3] ls /brokers/topics
[__consumer_offsets, bd14second, from-java, kafkatest]
[zk: localhost:2181(CONNECTED) 4] deleteall kafkatest
Command failed: java.lang.IllegalArgumentException: Path must start with / character
[zk: localhost:2181(CONNECTED) 5] deleteall /brokers/topics/kafkatest
[zk: localhost:2181(CONNECTED) 6] ls /brokers/topics
[__consumer_offsets, bd14second, from-java]
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7]
```

kafka topic结构



> Kafka中的Message是以topic为基本单位组织的，不同的topic之间是相互独立的。每个topic又可以分成几个不同的partition。
> partition是以文件的形式存储在文件系统中，比如，创建了一个名为page_visits的topic，其有5个partition，那么在Kafka的数据目录中(由配置文件中的log.dirs指定的)中就有这样5个目录: page_visits-0， page_visits-1，page_visits-2，page_visits-3，page_visits-4，其命名规则为 `topic_name-partition_id` ，里面存储的分别就是这5个partition的数据。
> Partition中的每条Message由offset来表示它在这个partition中的偏移量，这个offset不是该Message在partition数据文件中的实际存储位置，而是逻辑上一个值，它唯一确定了partition中的一条Message。因此，可以认为offset是partition中Message的id。partition中的每条Message包含了以下三个属性：offset、messagesize、data

Kafka delivery guarantee

有这么几种可能的delivery guarantee

- At most once 消息可能会丢，但绝不会重复传输
- At least one 消息绝不会丢，但可能会重复传输
- Exactly once 每条消息肯定会被传输一次且仅传输一次，很多时候这是用户所想要的。

到目前为止Producer向broker发送消息时，还没有提供确实可行的Exactly once方案

消息从broker到Consumer的delivery guarantee语义。
Consumer在从broker读取消息后，可以选择commit，该操作会在Zookeeper中保存该Consumer在该Partition中读取的消息的offset。该Consumer下一次再读该Partition时会从下一条开始读取。如未commit，下一次读取的开始位置会跟上一次commit之后的开始位置相同。当然可以将Consumer设置为autocommit，即Consumer一旦读到数据立即自动commit。如果只讨论这一读取消息的过程，那Kafka是确保了Exactly once。但实际使用中应用程序并非在Consumer读取完数据就结束了，而是要进行进一步处理，而数据处理与commit的顺序在很大程度上决定了消息从broker和consumer的delivery guarantee semantic。

- 读完消息先commit再处理消息。这种模式下，如果Consumer在commit后还没来得及

处理消息就crash了，下次重新开始工作后就无法读到刚刚已提交而未处理的消息，这就对应于At most once

- 读完消息先处理再commit。这种模式下，如果在处理完消息之后commit之前Consumer crash了，下次重新开始工作时还会处理刚刚未commit的消息，实际上该消息已经被处理过了。这就对应于At least once。在很多使用场景下，消息都有一个主键，所以消息的处理往往具有幂等性，即多次处理这一条消息跟只处理一次是等效的，那就可以认为是Exactly once。（笔者认为这种说法比较牵强，毕竟它不是Kafka本身提供的机制，主键本身也并不能完全保证操作的幂等性。而且实际上我们说delivery guarantee 语义是讨论被处理多少次，而非处理结果怎样，因为处理方式多种多样，我们不应该把处理过程的特性——如是否幂等性，当成Kafka本身的Feature）
- 如果一定要做到Exactly once，就需要协调offset和实际操作的输出。精典的做法是引入两阶段提交。如果能让offset和操作输入存在同一个地方，会更简洁和通用。这种方式可能更好，因为许多输出系统可能不支持两阶段提交。比如，Consumer拿到数据后可能把数据放到HDFS，如果把最新的offset和数据本身一起写到HDFS，那就可以保证数据的输出和offset的更新要么都完成，要么都不完成，间接实现Exactly once。（目前就high level API而言，offset是存于Zookeeper中的，无法存于HDFS，而low level API的offset是由自己去维护的，可以将之存于HDFS中）
  总之，Kafka默认保证At least once，并且允许通过设置Producer异步提交来实现At most once。而Exactly once要求与外部存储系统协作，幸运的是Kafka提供的offset可以非常直接非常容易得使用这种方式。

> 发送消息时，如果message散布在不同的partition上，那么customer在消费message时，是不会整体有序的，customer是消费完一个partition后，再去消费另一个partition的。

kafka维护的消费组的offset上有：

1. 所属组
2. 所属topic
3. consumer group名称
4. offset的当前值

flume只能收集文本文件，不能收集视频文件
如果使用kafka进行视频信息采集时，需要设置一个partition，否则无法保证读取的时候有序(kafka 在同一个partition中有序)