Kafka

- 讲师：pansir

一个分布式MQ，Scala语音编写，运行在JVM上

kafka简介

基本组成

1. topic：消息种子分类，每一类消息称为topic(主题)
    - Partition：topic在物理上的分组，多个partition被分散存储到不同到kafka节点，单个partition的消息是有序的，但整个topic消息不一定有序
    - segment：包含消息内容但指定大小但文件，由index文件和log文件组成；一个partition由多个segment文件组成
        1. offset：segment文件中消息的索引值，从0开始计数
    - replica(N)：消息冗余备份，每个partition都会由N个完全相同的冗余备份，这些备份会被尽量分散存储到不同机器
2. Producer：topic生产者
3. consumer：topic消费者，可以订阅一个或多个主题
4. broker：消息代理；

关键能力

1. 消息队列：点对点，发布订阅模式
2. 容错存储
3. 流处理

技术实现

- 通过Partitions实现了高并发
- 通过Partitions复制 + zookeeper 实现了高可用

目标

1. 实现大数据

原理

topic

一个消息队列对应一个topic，kafka将一个topic分成了多个partition，写入topic的消息会被平均分配到各个partition。partition会为消息设置一个唯一id，叫做偏移量(offset)；可以理解为一个索引值，通过这个索引值可以唯一确定这个消息

问题：这样做的目的是什么?

kafka的目标就是大数据，使用中心队列势必缺少可伸缩性，无论生产者/消费者/消息数量的增加，都可能导致机器的性能或存储耗尽。引入partition后，当性能/存储不足时，kafka就可以通过增加partition实现横向扩展

消费模型

- 点对点模式(队列模式)：每条消息只发送给一个消费者，就算某个队列有多个消费者，但是消息只能被消费一次
  - 优点：多个消费者消费同一个队列，效率更高
- 发布/订阅模式：每条消息发送给多个主题(队列)，多个消费者来订阅这个主题
  - 优点：一个消息可以被多次消费，可以完成冗余消费(相同的消费者)，也可以实现广播操作(不同的消费者)

Partition与消费模型

- 每个消费组跟partition是发布订阅的模式，即：每个消费组，会拿到topic中所有的消息
- 一个group内部的每个消费者，跟partition是点对点工作模式

消费组

一个消费组可以包含1个或多个消费者，整个消费组跟topic(partition)是发布订阅的关系

思考

> 1. topic打散消息顺序后，放入partition，消费组拿到所有的消息后，如何保证消息顺序?

无法重建topic中的消息顺序，只能保证partition内的消息是有序的，无法顾及全局消息顺序，除非仅一个partition

> 2. 不同的消费组是如何实现多次消费partition中的数据呢?

我们给partition设置保留时间，比如2天，则进入一条消息后，2天内，这条消息被消费任意次，partition都不会删除它；2天后，消息会自动删除

> 3. partition如何知道每个消费组消费到哪里了?

partition为每个消费组保存了一个偏移量，记录group消费到的位置

> 4. 怎么理解一个消费组内，一个partition只能被一个消费者消费

如果消费者比partition多，则会有消费者空闲；如果消费者比partition少，则1个消费者可能消费多个partition

高可用

每个partition都需要落地到物理机(broker)上，可用是单节点，也可以是一个集群。需要对数据做冗余(replication)，来保证高可用，即：存储多份partition到不同到broker，并且它们自动进行数据同步

分布式系统设计：将topic分成3个partition，每个partition保存两个副本，副本平均分配到3个broker节点，即：其中1个挂掉，剩余两个仍能维持整个系统正常工作

kafka每个broker都需要和zookeeper通信，每个partition的多个副本之间通过zookeeper的leader选举机制，选出主副本，所有该partition上的读写都通过这个主副本进行；其他的冗余副本会从主副本同步新消息

总结

1. Topic 是顶级概念，对应于一个消息队列。
2. Kafka 是以 Partition 为单位存储消息的，Consumer 在消费时也是按 Partition 进行的。即 Kafka 会保证一个 Consumer 收到的消息中，来自同一个 Partition 的所有消息是有序的。而来自不同 Partition 的消息则不保证有序。
3. Partition 会为其中的消息分配 Partition 内唯一的 ID，一般称作偏移量(offset) 。Kafka 会保留所有的消息，直到消息的保留时间（例如设置保留 2 天）结束。这样 Consumer 可以自由决定如何读取消息，例如读取更早的消息，重新消费等。
4. Kafka 有 Consumer Group 的概念。每个 Group 独立消费某个 Topic 的消息，互相不干扰。事实上，Kafka 会为每个 Group 保存一个偏移量，记录消费的位置。每个 Group 可以包含多个 Consumer，它们共同消费这个 Topic。
5. 对于一个 Consumer Group，一个 Partition 只能由 Group 中的一个 Consumer 消费。具体哪个 Consumer 监听哪个 Partition 是由 Kafka 分配的。算法可以指定为 Range 或 RoundRobin。
6. 物理上，消息是存在 Broker 上的，一般对应为一台物理机或集群。存储时，每个 Partition 都可以有多个副本。它们会被“均匀”地存储在各个 Broker 中。
7. 对于一个 Partition，它的多个复本存储一般存储在不同 Broker 中，在同一时刻会由 Zookeeper 选出一个主副本来负责所有的读写操作。

python操作kafka

python有三个客户端：pykafka，kafka-python，confluent-kafka-python；pykafka支持zookeeper，但kafka-python不支持

生产者

```
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaProducer
# 连接服务端
producer = KafkaProducer(bootstrap_servers=['10.211.55.5:9092'])
# 1. topic参数，必须指定
# 2. key：必须bytes类型，可以不指定，但和value必须指定一个，默认None
# 3. value：必须bytes类型，可以不指定，但和value必须指定一个，默认None
future = producer.send('topic_dcs', key=b'', value=b'hellIo', partition=0)
# 等待消息发送完成，超时时间为10ms
result = future.get(timeout=10)
# 打印详情
```

```
logging.info(result)
```

消息内容说明：

```
ConsumerRecord(topic='topic_dcs', partition=0, offset=9,
timestamp=1577539258566, timestamp_type=0, key=b'', value=b'hell1o', headers=
[], checksum=None, serialized_key_size=0, serialized_value_size=6,
serialized_header_size=-1)
```

1. topic：topic名称
2. partition：分区编号
3. offset：本条消息的偏移量
4. timestamp：时间戳
5. timestamp_type：时间戳类型
6. key：key值，bytes类型
7. value：value值，bytes类型
8. headers：
9. checksum：消息的校验和
10. serialized_key_size：序列化key的大小
11. serialized_value_size：序列号value的大小，当value为None时，大小为-1
12. serialized_header_size：头序列化大小

消费者

详情信息

```
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaConsumer

consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
    # consumer_timeout_ms=1000
)

logging.info("[分区信息]:
{}".format(consumer.partitions_for_topic("topic_dcs")))
logging.info("[当前订阅topic]: {}".format(consumer.subscription()))
logging.info("[当前topic和分区信息]: {}".format(consumer.assignment()))
logging.info("[可消费的偏移量]:
{}".format(consumer.beginning_offsets(consumer.assignment())))
```

默认消费

默认读取当前可读取的最早的消息

```
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaConsumer
# 1. topic名称必须指定
# 2. group_id：消费组，可以不指定
# 3. bootstrap_server：kafka服务，如果集群则写多个即可
consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092']
)
for msg in consumer:
    time.sleep(1)
    logging.info("[当前接收到到数据是]: {}".format(msg))
```

指定分区

消费时可以主动指定分区

```
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaConsumer
consumer = KafkaConsumer(
    group_id='group2',
    bootstrap_servers=['10.211.55.5:9092']
)
consumer.assign([TopicPartition(topic= 'topic_dcs', partition= 0)])
for msg in consumer:
    time.sleep(1)
    logging.info("[当前接收到到数据是]: {}".format(msg))
    logging.info(msg.partition)
    logging.info(msg.offset)
    logging.info(msg.key)
    logging.info(msg.value)
```

重复消费(自定义offset消费)

不像RabbitMQ，消费完成后直接删除，而是会默认保存7天，7天后自动清除，所以7天内可以从头消费，只需要指定partition和offset

```
import json
import logging
import time

from kafka import KafkaConsumer, TopicPartition

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s %(message)s')

consumer = KafkaConsumer(
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
)

# 指定分区
tp = TopicPartition("topic_dcs", 0)
# 初始化
consumer.assign([tp])
# 寻找分区的最早可用的偏移量
consumer.seek_to_beginning()
# 指定偏移量
consumer.seek(tp, 5)
# 开始消费
for msg in consumer:
    logging.info("[当前接收到到数据是]: {}".format(msg))
    logging.info(msg.value)
```

超时处理

默认为空，不指定consumer_timeout_ms，默认一直循环等待接收，若指定，则超时返回，不再等待。单位：ms

```
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s %(message)s')
from kafka import KafkaConsumer
consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
    consumer_timeout_ms=1000  # 1000ms后，若没有消息进来，则终止进程
)
for msg in consumer:
```

```
    logging.info("[当前接收到到数据是]: {}".format(msg))
```

订阅多个topic

- 指定不同的topic

```
import logging
from kafka import KafkaConsumer

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
)
# 订阅多个topic，用列表传参
consumer.subscribe(topics=['msgTopic', 'msgTopic1'])
for msg in consumer:
    logging.info("[当前接收到到数据是]: {}".format(msg))
```

- 订阅一类topic

```
import logging
from kafka import KafkaConsumer

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
)
# 订阅一类
consumer.subscribe(pattern='.*pic.*')
for msg in consumer:
    logging.info("[当前接收到到数据是]: {}".format(msg))
```

主动轮询

```
import json
import logging
import time
```

```
from kafka import KafkaConsumer

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

consumer = KafkaConsumer(
    "topic_dcs",
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
)

while True:
    msg = consumer.poll(timeout_ms=5)  # 从kafka获取消息
    logging.info("当前拉取的数据：{}".format(msg))
    time.sleep(5)
```

编码与解码

key 与 value，都可以进行编码与解码

字符串编码/解码

- 生产者编码

```
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaProducer

# 声明编码格式，默认是utf-8
producer = KafkaProducer(
    bootstrap_servers=['10.211.55.5:9092'],
    key_serializer=str.encode,
    value_serializer=str.encode
)
future = producer.send('topic_dcs', key='key', value='你好', partition=0)

result = future.get(timeout=10)
logging.info(result)
```

- 消费者解码

```
import logging
from kafka import KafkaConsumer
```

```
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
    key_deserializer=bytes.decode,
    value_deserializer=bytes.decode
)
# 订阅多个topic，用列表传参
consumer.subscribe(pattern='.*pic.*')
for msg in consumer:
    logging.info("[当前接收到到数据是]: {}".format(msg))
```

- 一般的英文字母，str和bytes格式看着一样，使用中午就可以明显看到编码后的不同

json数据编码/解码

- 生产者编码json数据，先序列化，再编码

```
import json
import logging
import timeout

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaProducer

# 声明编码格式，默认是utf-8
# 注意，value_serializer接收的是一个匿名函数，接收一个字典对象，返回一个bytes对象
producer = KafkaProducer(
    bootstrap_servers=['10.211.55.5:9092'],
    key_serializer=str.encode,
    value_serializer=lambda m: json.dumps(m).encode('utf-8')
)
future = producer.send('topic_dcs', key='key', value={"name": "开心麻花", "age":
18}, partition=0)

result = future.get(timeout=10)
logging.info(result)
```

- 消费者解码：先解码，在反序列化

```
import json
import logging
```

```
from kafka import KafkaConsumer

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
    key_deserializer=bytes.decode,
    value_deserializer=lambda m: json.loads(m.decode('utf-8'))  # 先解码，再反序
列化
)
# 订阅多个topic，用列表传参
consumer.subscribe(topics=["topic_dcs"])
for msg in consumer:
    logging.info("[当前接收到到数据是]: {}".format(msg))
```

msgpack序列化

msgpack是一种高效的二进制序列化库，效率比json高

- 生产者序列化发送

```
import json
import logging
import time

import msgpack

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaProducer

producer = KafkaProducer(
    bootstrap_servers=['10.211.55.5:9092'],
    key_serializer=str.encode,
    value_serializer=msgpack.dumps
)
future = producer.send('topic_dcs', key='key', value={"name": "开心麻花", "age":
18}, partition=0)

result = future.get(timeout=10)
logging.info(result)
```

- 消费者反序列化

```
import json
import logging

import msgpack
from kafka import KafkaConsumer

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

consumer = KafkaConsumer(
    'topic_dcs',
    group_id='group1',
    bootstrap_servers=['10.211.55.5:9092'],
    key_deserializer=bytes.decode,
    value_deserializer=msgpack.loads,
)
# 订阅多个topic，用列表传参
consumer.subscribe(topics=["topic_dcs"])
for msg in consumer:
    logging.info("[当前接收到到数据是]: {}".format(msg))
    logging.info(msg.value)
```

注意：msgpack反序列化后的结果会出现乱码，仍需要手动再进行处理

消息压缩

发送时，可以进行消息压缩，支持：gzip；snappy；lz4

```
import json
import logging
import time

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from kafka import KafkaProducer

producer = KafkaProducer(
    bootstrap_servers=['10.211.55.5:9092'],
    key_serializer=str.encode,
    value_serializer=lambda m: json.dumps(m).encode('utf-8'),
    compression_type='gzip',  # 指定压缩格式
)
future = producer.send('topic_dcs', key='key', value={"name": "开心麻花", "age":
18}, partition=0)

result = future.get(timeout=10)
logging.info(result)
```