

kafka企业级消息系统

课程目标

- 生产者原理掌握
- 消费者原理掌握
- 自定义分区实现
- 生产者写数据流程
- 生产者数据分发策略
- 消费高阶API及低级API
- 消费者消费数据流程
- 消费者自动或手动提交offset值
- kafka的消息不丢失机制
- kafka的消息存储机制
- kafka的监控及运维了解

1、kafka原理部分

1.1、生产-消费流程

数据从生产-消费-提交offset过程，分一下几个阶段来进行

1.1.1、生产者

生产者是一个向kafka Cluster发布记录的客户端；**生产者是线程安全的**，跨线程共享单个生成器实例通常比具有多个实例更快。

1.1.1.1、必要条件

生产者要进行生产数据到kafka Cluster中，必要条件有以下三个：

#1、地址

`bootstrap.servers=hadoop-01:9092`

#2、序列化

`key.serializer=org.apache.kafka.common.serialization.StringSerializer`

`value.serializer=org.apache.kafka.common.serialization.StringSerializer`

#3、主题 (topic)

需要制定具体的某个topic (order) 即可。

1.1.1.2、生产者自定义分区

- 实现生产者自定义分区有以下几个步骤实现即可：

1、创建自定义类，实现org.apache.kafka.clients.producer.Partitioner接口

2、重写以下的方法

```
/**
 * Compute the partition for the given record.
 *
 * @param topic The topic name
 * @param key The key to partition on (or null if no key)
 * @param keyBytes The serialized key to partition on( or null if no key)
 * @param value The value to partition on or null
 * @param valueBytes The serialized value to partition on or null
 * @param cluster The current cluster metadata
 */
public int partition(String topic, Object key, byte[] keyBytes, Object value,
byte[] valueBytes, Cluster cluster);
```

3、配置项中加入partitioner.class属性

- 代码如下所示：

```
import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;

import java.util.Map;

/**
 * 实现自定义的生产者分区
 */
public class MyPartitioner implements Partitioner {

    /**
     * 手机号码分区--区分移动，联通，电信
     * <p>
     * 135开头---->移动运营商---->0号分区
     * 130开头---->联通运营商---->1号分区
     * 180开头---->电信运营商---->2号分区
     *
     * @param topic      主题
     * @param key
     * @param keyBytes    key的字节数组
     * @param value
     * @param valueBytes
     * @param cluster     集群信息
     * @return
     */
    public int partition(String topic, Object key, byte[] keyBytes, Object value,
byte[] valueBytes, Cluster cluster) {
```

```

//拿到主题下有几个分区
//如果分区数不为3, 那么可以进入0
Integer count = cluster.partitionCountForTopic(topic);
//分区值为;[0-count-1]

String keyString = key.toString();

if (count == 3 && keyString != null) {
    if (keyString.startsWith("135")) {
        return 0;
    }else if(keyString.startsWith("130")){
        return 1;
    }else if(keyString.startsWith("180")){
        return 2;
    }
}

return 0;
}

/**
 * 关闭-如果不需要, 就不要实现它
 */
public void close() {

}

/**
 * 配置项--如果不需要, 就不要实现它
 * @param configs
 */
public void configure(Map<String, ?> configs) {

}}

```

- 配置信息

```

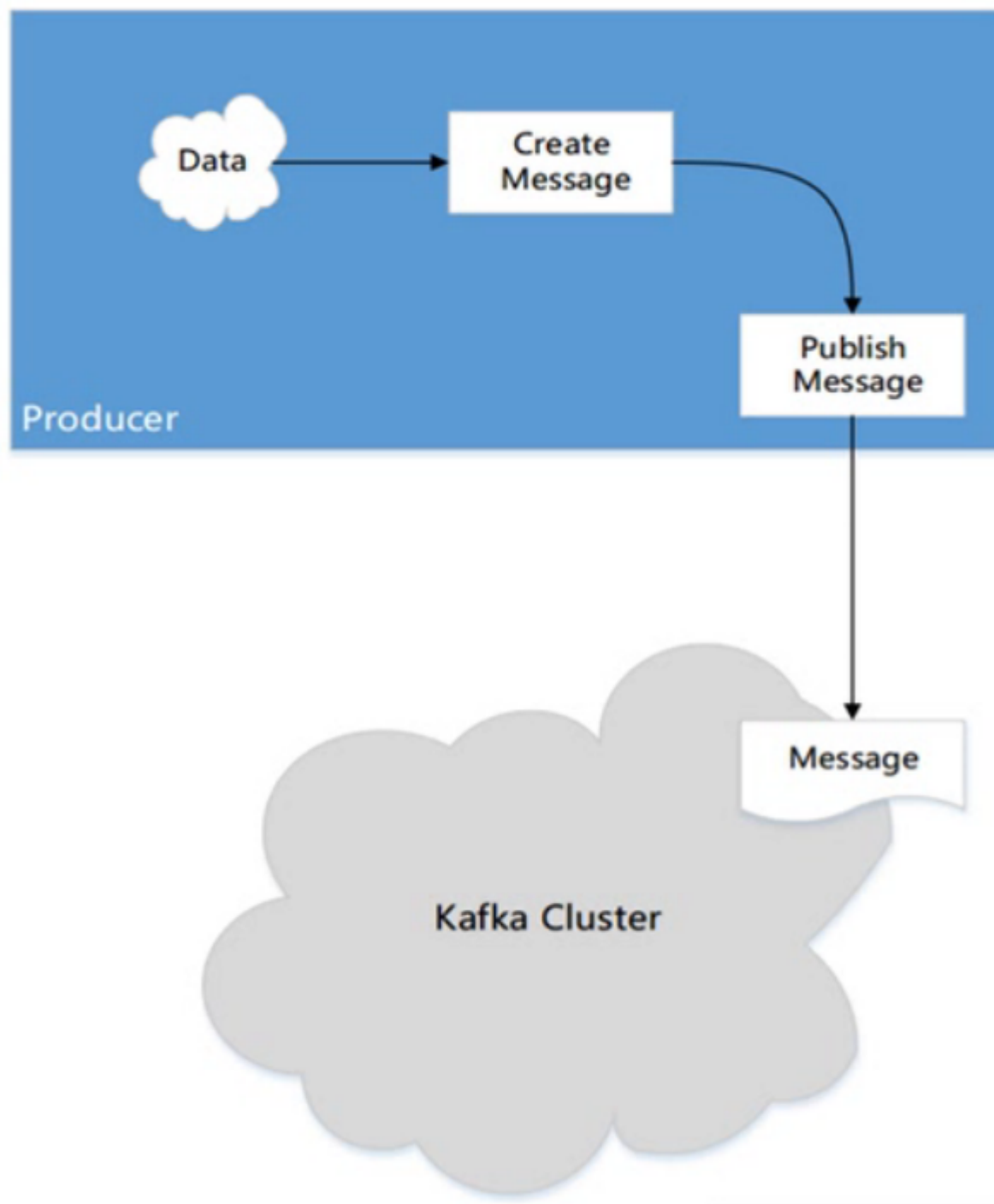
bootstrap.servers=localhost:9092
acks=all
retries=0
batch.size=16384
linger.ms=1
buffer.memory=33554432
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer

#配置项中加入partitioner.class属性
partitioner.class=cn.itcast.java.wordcount.utils.MyPartitioner

```

1.1.1.3、生产者 (Producer) 写数据

1.1.1.3.1、生产者 (Producer) 写数据流程图



1.1.1.3.2、流程描述

1、总体流程

Producer连接任意活着的Broker，请求指定Topic，Partion的Leader元数据信息，然后直接与对应的Broker直接连接，发布数据

2、开放分区接口(生产者数据分发策略)

2.1、用户可以指定分区函数，使得消息可以根据key，发送到指定的Partition中。

2.2、kafka在数据生产的时候，有一个数据分发策略。默认的情况使用DefaultPartitioner.class类。这个类中就定义数据分发的策略。

2.3、如果是用户制定了partition，生产就不会调用DefaultPartitioner.partition()方法

2.4、当用户指定key，使用hash算法。如果key一直不变，同一个key算出来的hash值是个固定值。如果是固定值，这种hash取模就没有意义。

```
Utils.toPositive(Utils.murmur2(keyBytes)) % numPartitions
```

2.5、当用既没有指定partition也没有key。

```
/**
 * The default partitioning strategy:
 * <ul>
 * <li>If a partition is specified in the record, use it
 * <li>If no partition is specified but a key is present choose a partition based on a
hash of the key
 * <li>If no partition or key is present choose a partition in a round-robin fashion
 */
```

2.6、数据分发策略的时候，可以指定数据发往哪个partition。当ProducerRecord的构造参数中有partition的时候，就可以发送到对应partition上。

1.1.1.3.3、生产者数据分发策略

生产者数据分发策略有如下四种：(总的来说就是调用了一个方法，参数不同而已)

```
//可根据主题和内容发送
public ProducerRecord(String topic, V value)
//根据主题, key、内容发送
public ProducerRecord(String topic, K key, V value)
//根据主题、分区、key、内容发送
public ProducerRecord(String topic, Integer partition, K key, V value)
//根据主题、分区、时间戳、key, 内容发送
public ProducerRecord(String topic, Integer partition, Long timestamp, K key, V value)
```

a、可根据主题和内容发送

```
Producer<String, String> producer = new KafkaProducer<String, String>(props);
//可根据主题和内容发送
producer.send(new ProducerRecord<String, String>("my-topic", "具体的数据"));
```

b、根据主题, key、内容发送

```
Producer<String, String> producer = new KafkaProducer<String, String>(props);
//可根据主题、key、内容发送
producer.send(new ProducerRecord<String, String>("my-topic", "key", "具体的数据"));
```

c、根据主题、分区、key、内容发送

```
Producer<String, String> producer = new KafkaProducer<String, String>(props);
//可根据主题、分区、key、内容发送
producer.send(new ProducerRecord<String, String>("my-topic", 1, "key", "具体的数据"));
```

d、根据主题、分区、时间戳、key, 内容发送

```
Producer<String, String> producer = new KafkaProducer<String, String>(props);
//可根据主题、分区、时间戳、key、内容发送
producer.send(new ProducerRecord<String, String>("my-topic", 1, 12L, "key", "具体的数据"));
```

1.1.2、消费者

消费者是一个从kafka Cluster中消费数据的一个客户端；该客户端可以处理kafka brokers中的故障问题，并且可以适应在集群内的迁移的topic分区；该客户端还允许消费者组使用消费者组来进行负载均衡。

消费者维持一个TCP的长连接来获取数据，使用后未能正常关闭这些消费者问题会出现，因此**消费者不是线程安全的**。

1.1.2.1、必要条件

消费者要从kafka Cluster进行消费数据，必要条件有以下四个

#1、地址

bootstrap.servers=hadoop-01:9092

#2、序列化

key.serializer=org.apache.kafka.common.serialization.StringSerializer

value.serializer=org.apache.kafka.common.serialization.StringSerializer

#3、主题 (topic)

需要制定具体的某个topic (order) 即可。

#4、消费者组

group.id=test

1.1.2.2、消费者代码-自动提交offset值

```
/**
 * 消费订单数据--- javaben.tojson
 */
public class OrderConsumer {
    public static void main(String[] args) {
        // 1\连接集群
        Properties props = new Properties();
        props.put("bootstrap.servers", "hadoop-01:9092");
        props.put("group.id", "test");

        //以下两行代码 ---消费者自动提交offset值
        props.put("enable.auto.commit", "true");
        props.put("auto.commit.interval.ms", "1000");

        props.put("key.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        KafkaConsumer<String, String> kafkaConsumer = new KafkaConsumer<String, String>
(props);
        // 2、发送数据 发送数据需要, 订阅下要消费的topic。 order
        kafkaConsumer.subscribe(Arrays.asList("order"));
        while (true) {
            ConsumerRecords<String, String> consumerRecords =
kafkaConsumer.poll(100); // jdk queue offer插入、poll获取元素。 blockingqueue put插入原生,
take获取元素
            for (ConsumerRecord<String, String> record : consumerRecords) {
                System.out.println("消费的数据为: " + record.value());
            }
        }
    }
}
```

1.1.2.3、消费者代码-手动提交offset

如果Consumer在获取数据后，需要加入处理，数据完毕后才确认offset，需要程序来控制offset的确认？

关闭自动提交确认选项

```
props.put("enable.auto.commit", "false");
```

手动提交offset值

```
kafkaConsumer.commitSync();
```

完整代码如下所示：

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
//关闭自动提交确认选项
props.put("enable.auto.commit", "false");
props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("foo", "bar"));
final int minBatchSize = 200;
List<ConsumerRecord<String, String>> buffer = new ArrayList<>();
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records) {
        buffer.add(record);
    }
    if (buffer.size() >= minBatchSize) {
        insertIntoDb(buffer);
        // 手动提交offset值
        consumer.commitSync();
        buffer.clear();
    }
}
```

1.1.2.4、消费者代码-完成处理每个分区中的记录后提交偏移量

上面的示例使用commitSync将所有已接收的记录标记为已提交。在某些情况下，您可能希望通过明确指定偏移量来更好地控制已提交的记录。在下面的示例中，我们在完成处理每个分区中的记录后提交偏移量。

```
try {
    while(running) {
        ConsumerRecords<String, String> records = consumer.poll(Long.MAX_VALUE);
```



```

        for (TopicPartition partition : records.partitions()) {
            List<ConsumerRecord<String, String>> partitionRecords =
records.records(partition);
            for (ConsumerRecord<String, String> record : partitionRecords) {
                System.out.println(record.offset() + ": " + record.value());
            }
            long lastOffset = partitionRecords.get(partitionRecords.size() -
1).offset();
            consumer.commitSync(Collections.singletonMap(partition, new
OffsetAndMetadata(lastOffset + 1)));
        }
    }
} finally {
    consumer.close();
}

```

注意事项:

提交的偏移量应始终是应用程序将读取的下一条消息的偏移量。因此，在调用commitSync（偏移量）时，应该在最后处理的消息的偏移量中添加一个

1.1.2.5、消费者代码-手动指定消费指定分区的数据

- 1、如果进程正在维护与该分区关联的某种本地状态（如本地磁盘上的键值存储），那么它应该只获取它在磁盘上维护的分区的记录。
- 2、如果进程本身具有高可用性，并且如果失败则将重新启动（可能使用YARN，Mesos或AWS工具等集群管理框架，或作为流处理框架的一部分）。在这种情况下，Kafka不需要检测故障并重新分配分区，因为消耗过程将在另一台机器上重新启动。

```

Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test");
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
//consumer.subscribe(Arrays.asList("foo", "bar"));

//手动指定消费指定分区的数据---start
String topic = "foo";
TopicPartition partition0 = new TopicPartition(topic, 0);
TopicPartition partition1 = new TopicPartition(topic, 1);

```

```

consumer.assign(Arrays.asList(partition0, partition1));
//手动指定消费指定分区的数据---end

while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s%n",
record.offset(), record.key(), record.value());
}

```

注意事项:

- 1、要使用此模式，您只需使用要使用的分区的完整列表调用assign (Collection)，而不是使用subscribe订阅主题。
- 2、主题与分区订阅只能二选一

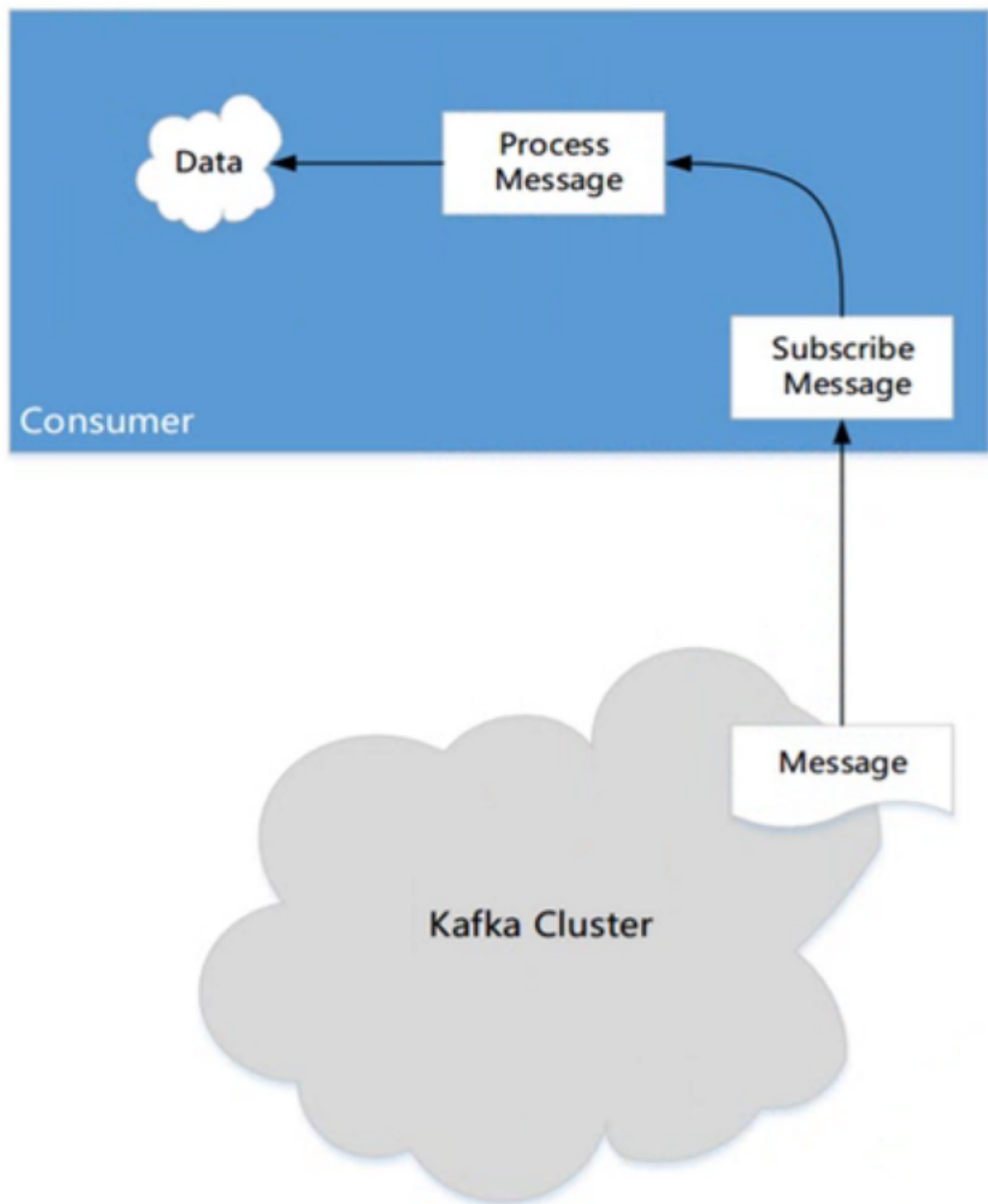
1.1.2.6、消费者数据丢失-数据重复

说明:

- 1、已经消费的数据对于kafka来说，会将消费组里面的offset值进行修改，那什么时候进行修改了？是在数据消费完成之后，比如在控制台打印完后自动提交；
- 2、提交过程：是通过kafka将offset进行移动到下个message所处的offset的位置。
- 3、拿到数据后，存储到hbase中或者mysql中，如果hbase或者mysql在这个时候连接不上，就会抛出异常，如果在处理数据的时候已经进行了提交，那么kafka伤的offset值已经进行了修改了，但是hbase或者mysql中没有数据，这个时候就会出现**数据丢失**。
- 4、什么时候提交offset值？在Consumer将数据处理完成之后，再进行offset的修改提交。默认情况下offset是自动提交，需要修改为手动提交offset值。
- 5、如果在处理代码中正常处理了，但是在提交offset请求的时候，没有连接到kafka或者出现了故障，那么该次修改offset的请求是失败的，那么下次在进行读取同一个分区中的数据时，会从已经处理掉的offset值再进行处理一次，那么在hbase中或者mysql中就会产生两条一样的数据，也就是**数据重复**

1.1.2.7、消费者 (Consumer) 读数据

1.1.2.7.1、消费者 (Consumer) 读数据流程图



1.1.2.7.2、流程描述

Consumer连接指定的Topic partition所在leader broker，采用pull方式从kafka logs中获取消息。

1.1.2.7.3、高阶API (High Level API)

kafka消费者高阶API简单；隐藏Consumer与Broker细节；相关信息保存在zookeeper中。

```
/* create a connection to the cluster */
ConsumerConnector connector = Consumer.create(consumerConfig);

interface ConsumerConnector {

    /**
     * This method is used to get a list of KafkaStreams, which are iterators over
     * MessageAndMetadata objects from which you can obtain messages and their
     * associated metadata (currently only topic).
     * Input: a map of <topic, #streams>
     * Output: a map of <topic, list of message streams>
     */
    public Map<String,List<KafkaStream>> createMessageStreams(Map<String,Int>
topicCountMap);

    /**
     * You can also obtain a list of KafkaStreams, that iterate over messages
     * from topics that match a TopicFilter. (A TopicFilter encapsulates a
     * whitelist or a blacklist which is a standard Java regex.)
     */
    public List<KafkaStream> createMessageStreamsByFilter(
        TopicFilter topicFilter, int numStreams);

    /* Commit the offsets of all messages consumed so far. */
    public commitOffsets()

    /* Shut down the connector */
    public shutdown()
}
```

说明：大部分的操作都已经封装好了，比如：当前消费到哪个位置下了，但是不够灵活（工作过程推荐使用）

1.1.2.7.4、低级API(Low Level API)

kafka消费者低级API非常灵活；需要自己负责维护连接Controller Broker。保存offset，Consumer Partition对应关系。

```
class SimpleConsumer {

    /* Send fetch request to a broker and get back a set of messages. */
    public ByteBufferMessageSet fetch(FetchRequest request);
```

```

/* Send a list of fetch requests to a broker and get back a response set. */
public MultiFetchResponse multifetch(List<FetchRequest> fetches);

/**
 * Get a list of valid offsets (up to maxSize) before the given time.
 * The result is a list of offsets, in descending order.
 * @param time: time in millisecs,
 *             if set to OffsetRequest$.MODULE$.LATEST_TIME(), get from the latest
offset available.
 *             if set to OffsetRequest$.MODULE$.EARLIEST_TIME(), get from the
earliest offset available.
 */
public long[] getOffsetsBefore(String topic, int partition, long time, int
maxNumOffsets);
}

```

说明：没有进行包装，所有的操作有用户决定，如自己的保存某一个分区下的记录，你当前消费到哪个位置。

1.1.3、kafka的log-存储机制

1.1.3.1、kafka中log日志目录及组成

kafka在我们指定的log.dir目录下，会创建一些文件夹；名字是【主题名字-分区名】所组成的文件夹。

- 在【主题名字-分区名】的目录下，会有两个文件存在，如下所示：

```

#索引文件
00000000000000000000.index
#日志内容
00000000000000000000.log

```

- 在目录下的文件，会根据log日志的大小进行切分，**.log文件的大小为1G**的时候，就会进行切分文件；

```

-rw-r--r--. 1 root root 389K 4月 10 16:43 00000000000000000000.index
-rw-r--r--. 1 root root 1.0G 4月 10 16:43 00000000000000000000.log
-rw-r--r--. 1 root root 10M 4月 10 16:43 000000000000000077894.index
-rw-r--r--. 1 root root 127M 4月 10 16:43 000000000000000077894.log

```

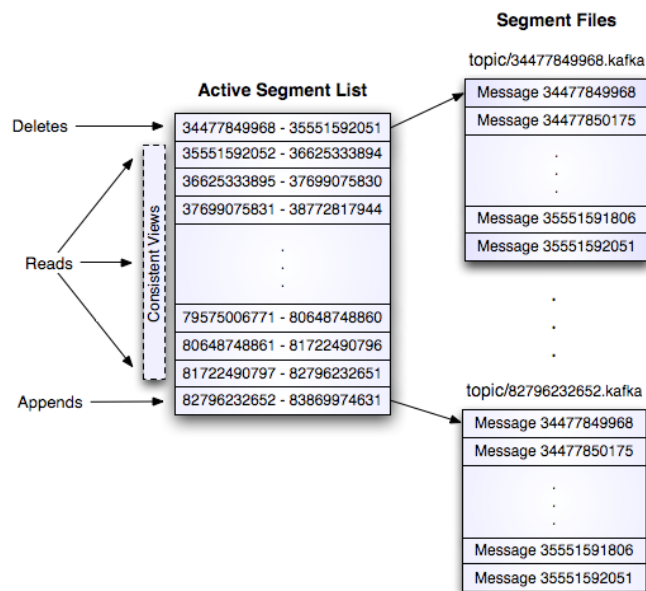
- 在kafka的设计中，将offset值作为文件名的一部分

比如：topic的名字为：test，有三个分区，生成的目录如下如下所示：

```
test-0
test-1
test-2
```

- kafka日志的组成
 - segment file组成：由两个部分组成，分别为index file和数据file，此两个文件一一对应且成对出现；后缀.index和.log分别表示为segment的索引文件、数据文件。
- segment文件命名规则：partition全局的第一个segment从0开始，后续每个segment文件名为上一个全局partition的最大offset（偏移message数）。数值最大为64位long大小，19位数字字符长度，没有数字就用0填充。

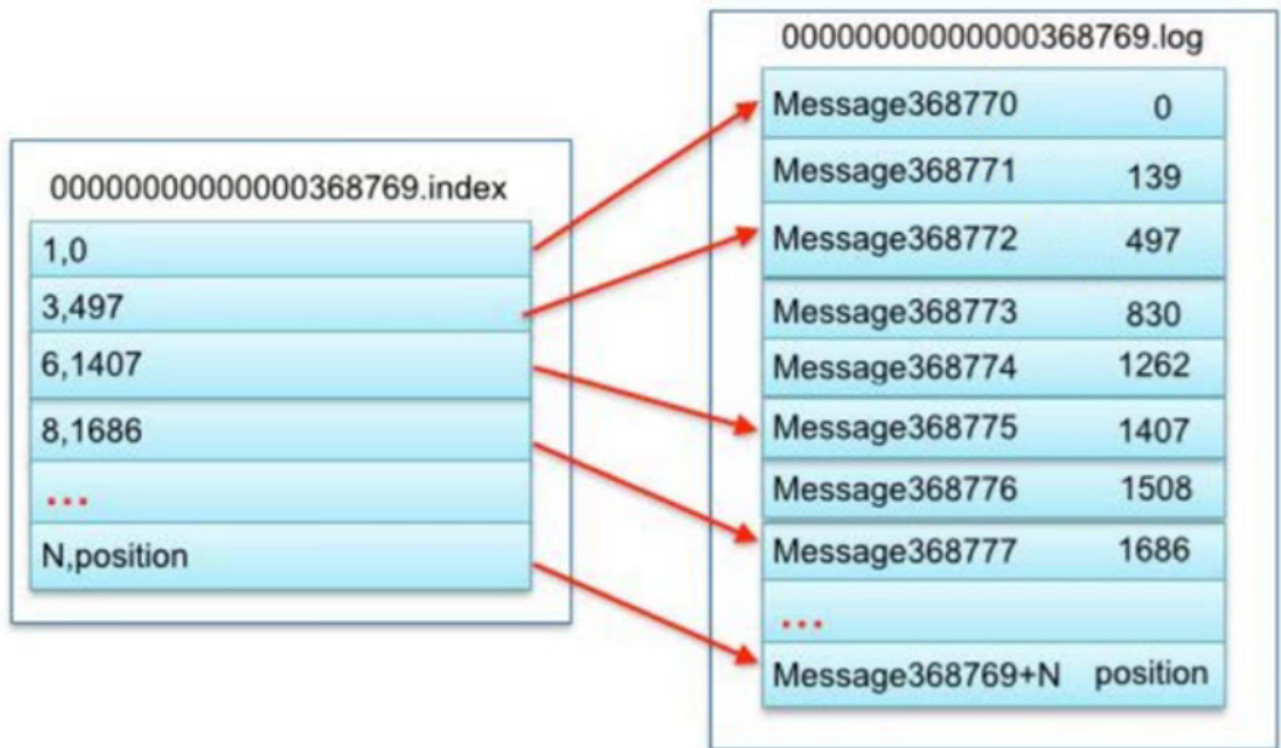
Kafka Log Implementation



- 通过索引信息可以快速定位到message。通过index元数据全部映射到memory，可以避免segment file的IO磁盘操作；
- 通过索引文件稀疏存储，可以大幅降低index文件元数据占用空间大小。
- 稀疏索引：为了数据创建索引，但范围并不是为每一条文件，而是为某一个区间创建；

好处：就是可以减少索引值的数量。

不好的地方：找到索引区间之后，要得进行第二次处理。



1.1.3.2、kafka Message的物理结构及介绍

- kafka Message的物理结构，如下图所示：

message物理结构

8 byte offset

4 byte message size

4 byte CRC32

1 byte "magic"

1 byte "attributes"

4 byte key length

K byte key

4 byte payload length

value bytes
payload

- 关键字及说明:

! [1547891737446] (C:\Users\Deborah\AppData\Roaming\Typora\typora-user-images\1547891737446.png)

1.1.3.3、kafka中log CleanUp

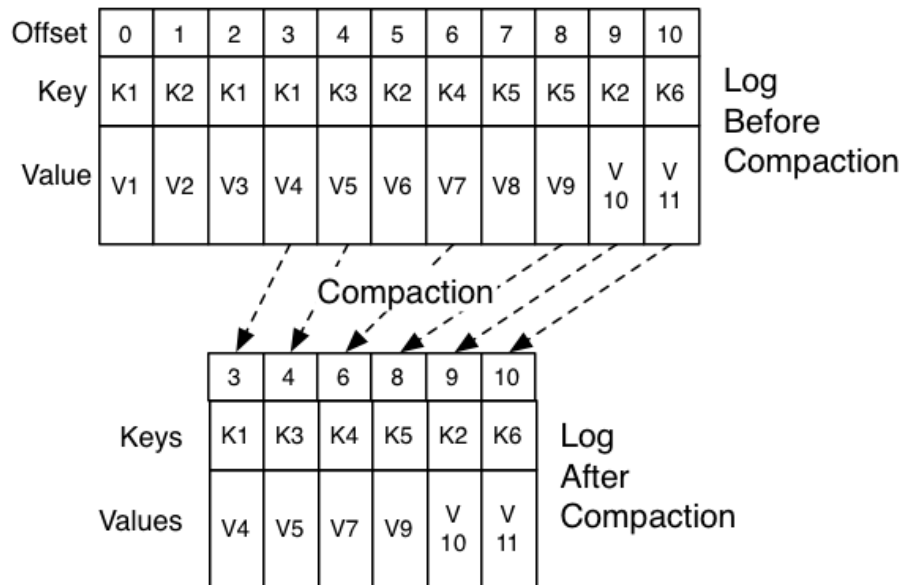
- kafka中清理日志的方式有两种：delete和compact。
- 删除的阈值有两种：过期的时间和分区内总日志大小。
- 在kafka中，因为数据是存储在本地磁盘中，并没有像hdfs的那样的分布式存储，就会产生磁盘空间不足的情况，可以采用删除或者合并的方式来进行处理
- 可以通过时间类删除、合并：默认7天
- 还可以通过字节大小、合并
- 如果是在server.properties里面设置，将在没有特殊指定参数的topic中，所有topic生效。
- 如果是在topic的配置中加入了，那只对该topic有效 --config;

log.cleanup.policy	The default cleanup policy for segments beyond the retention window. A comma separated list of valid policies. Valid policies are: "delete" and "compact"	list	delete	[compact, delete]	medium	cluster-wide

log.retention.hours	The number of hours to keep a log file before deleting it (in hours), tertiary to log.retention.ms property	int	168		high	read-only
log.retention.minutes	The number of minutes to keep a log file before deleting it (in minutes), secondary to log.retention.ms property. If not set, the value in log.retention.hours is used	int	null		high	read-only
log.retention.ms	The number of milliseconds to keep a log file before deleting it (in milliseconds), If not set, the value in log.retention.minutes is used	long	null		high	cluster-wide

og.retention.bytes	The maximum size of the log before deleting it	long	-1		high	cluster-wide

- 合并过程
- 相同的key, 保存offset值大的 (最新的消息记录)



2、kafka的CAP理论

2.1、什么是kafka的CAP

2.1.1、CAP包含以下三点：

- C:(Consistency: 一致性)
- A:(Availability: 可用性)
- P:(Partition Tolerance 分区容错性)

2.1.2、CAP特性详解

- **C:(Consistency: 一致性)**

A read is guaranteed to return the most recent write for a given client.

对于一致性的特性可以从以下几个方面进行考虑：

- 1、在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）
- 2、注：在一个节点上修改数据，在另一个节点上能读取到修改后的数据
- 3、某个节点的数据更新结果对后面通过其它节点的读操作可见
- 4、立即可见，称为强一致性
- 5、部分或者全部感知不到该更新，称为弱一致性
- 6、在一段时间（通常该时间不固定）后，一定可以感知该更新，称为最终一致性

- **A:(Availability: 可用性)**

A non-failing node will return a reasonable response within a reasonable amount of time(no error or timeout).

对于可用性的特性可以从以下几个方面进行考虑：

- 1、在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可用性）
- 2、注：在用户可容忍的范围内返回数据
- 3、一个没有发生故障的节点必须在有限的时间内返回合理的结果
- 4、Vs 数据库的HA：一个节点宕机，其它节点仍然可用

- **P:(Partition Tolerance 分区容错性)**

The system will continue to function when network partitions occur.

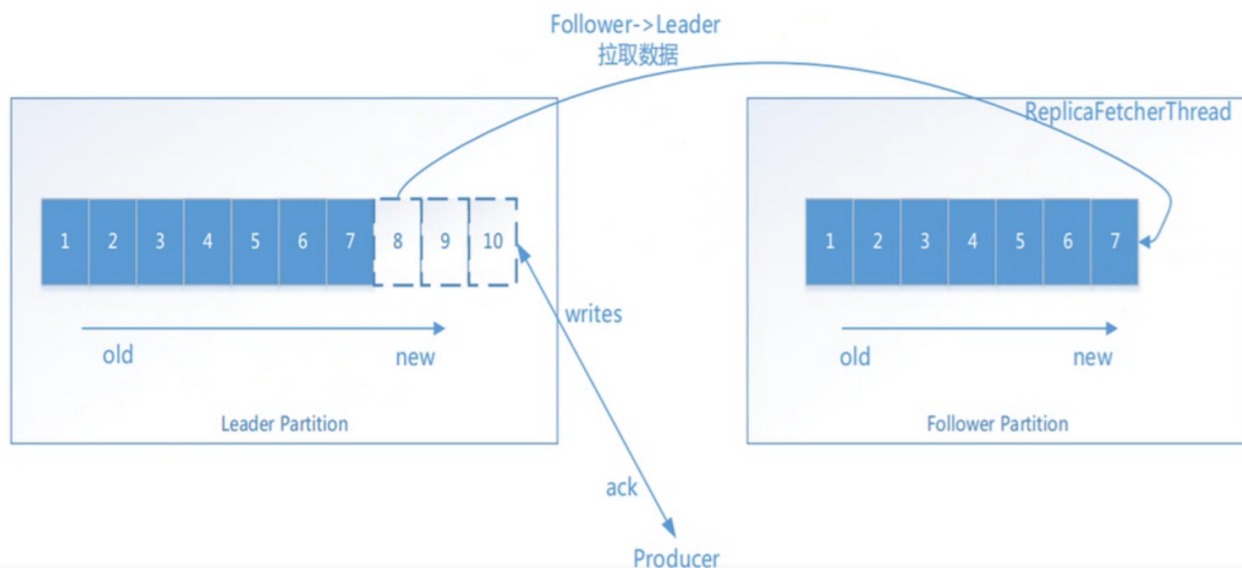
对于分区容错的特性可以从以下几个方面进行考虑：

- 1、以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在C和A之间做出选择。
- 2、注：一个分布式集群，分成多个小的集群，小的集群内部可相互通信，对于用户透明
- 3、部分节点宕机或者无法与其它节点通信时，各分区还可保持分布式系统的功能

3、kafka消息不丢失机制

3.1、生产者生产数据不丢失

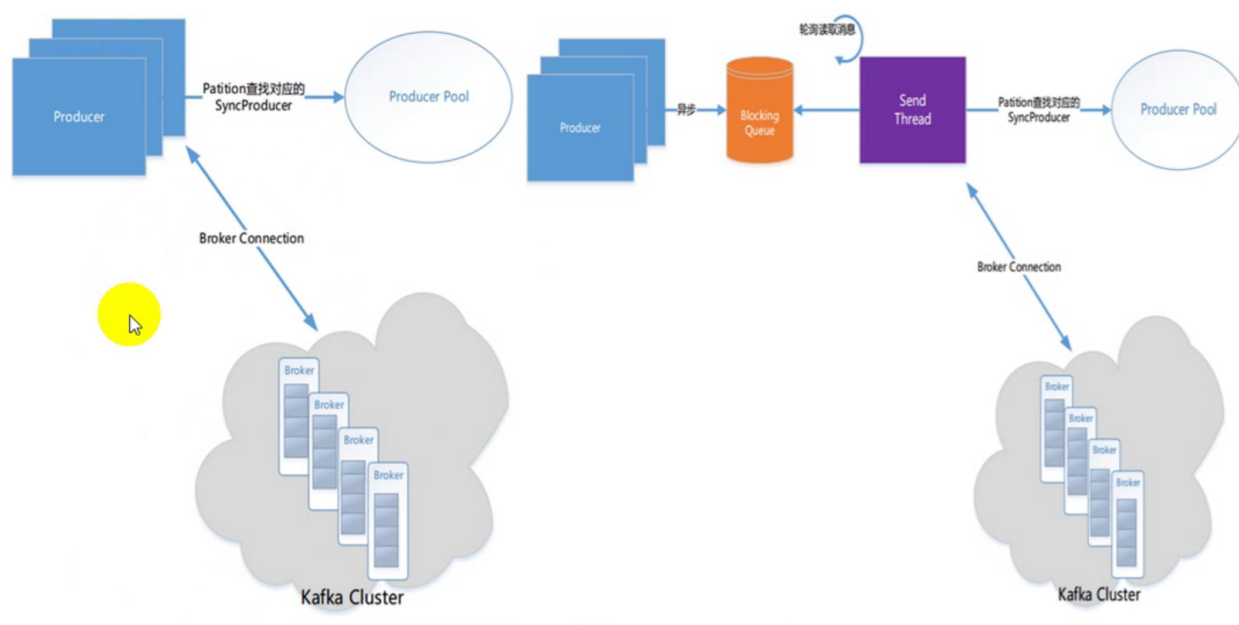
3.1.1、生产者数据不丢失过程图



说明：有多少个分区，就启动多少个线程来进行同步数据

3.1.2、发送数据方式

- 可以采用同步或者异步的方式-过程图



可以采用同步或者异步的方式

- 同步：发送一批数据给kafka后，等待kafka返回结果

- 1、生产者等待10s，如果broker没有给出ack相应，就认为失败。
- 2、生产者重试3次，如果还没有相应，就报错

- 异步：发送一批数据给kafka，只是提供一个回调函数。

- 1、先将数据保存在生产者端的buffer中。buffer大小是2万条
- 2、满足数据阈值或者数量阈值其中的一个条件就可以发送数据。
- 3、发送一批数据的大小是500条

说明：如果broker迟迟不给ack，而buffer又满了，开发者可以设置是否直接清空buffer中的数据。

3.1.3、ack机制（确认机制）

生产者数据不抵事，需要服务端返回一个确认码，即ack响应码；ack的响应有三个状态值

- 0：生产者只负责发送数据，不关心数据是否丢失，响应的状态码为0（丢失的数据，需要再次发送）
- 1：partition的leader收到数据，响应的状态码为1
- 1：所有的从节点都收到数据，响应的状态码为-1

说明：如果broker端一直不给ack状态，producer永远不知道是否成功；producer可以设置一个超时时间10s，超过时间认为失败。

3.2、kafka的broker中数据不丢失

在broker中，保证数据不丢失主要是通过副本因子（冗余），防止数据丢失

3.3、消费者消费数据不丢失

在消费者消费数据的时候，只要每个消费者记录好offset值即可，就能保证数据不丢失。

4、解析配置文件工具类

- 将所有的配置信息放入到一个配置文件中：以producer.properties为例说明，该文件中的内容如下所示：

```
bootstrap.servers=localhost:9092
acks=all
retries=0
batch.size=16384
linger.ms=1
buffer.memory=33554432
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer
partitioner.class=cn.itcast.java.wordcount.utils.MyPartitioner
```

- 解析生产者或者消费者配置信息工具类如下所示：

```
import java.io.InputStream;
import java.util.Properties;

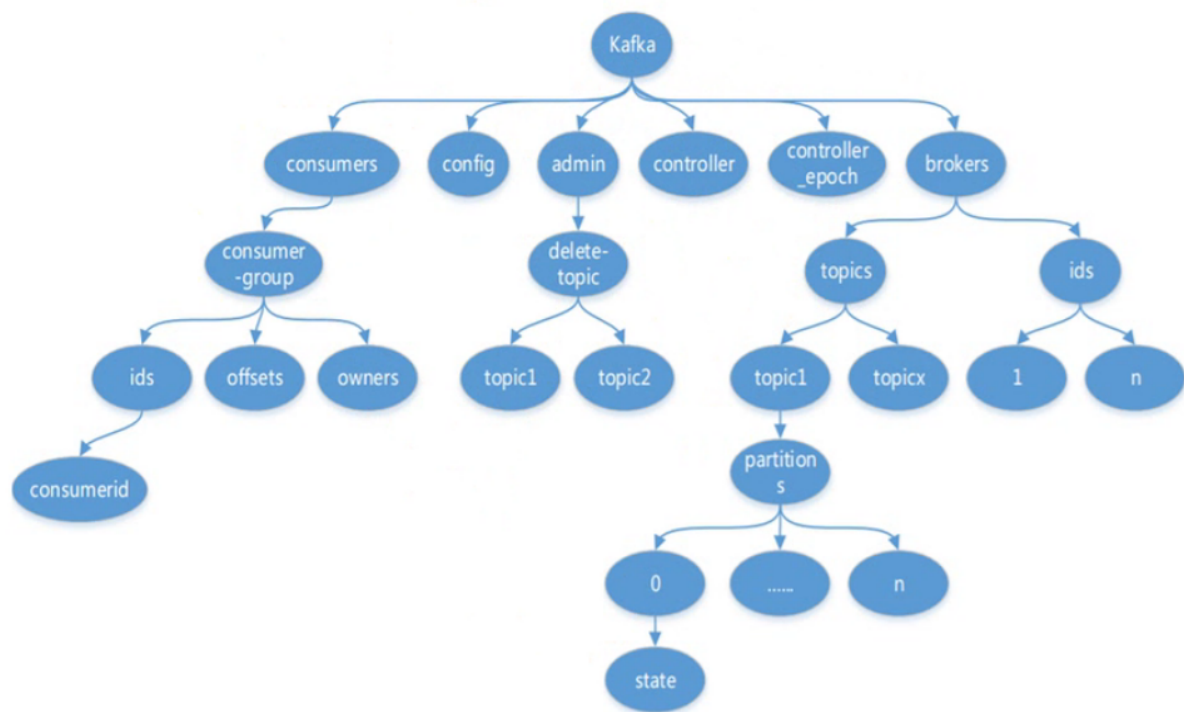
/**
 * 读取kafka配置文件，将所有配置信息写在配置文件中，然后生成一个properties即可
 * 读取配置文件信息
 */
public class KafkaUtils2ReadProperties {

    /**
     * 文件后缀名
     */
    public static String PROPERTIES=".properties";

    /**
     * 解析配置文件，生成Properties对象工具类
     * @param fileName 文件名，不需要带后缀
     * @return Properties
     * @throws Exception
     */
    public static Properties getProperties(String fileName)throws Exception{
        InputStream inputStream =
        KafkaUtils2ReadProperties.class.getClassLoader().getResourceAsStream(fileName+PROPERTIES);
        Properties properties = new Properties();
        properties.load(inputStream);
        return properties;
    }
}
```

5、kafka in zookeeper

- kafka在zookeeper中注册的图如下所示：



kafka集群中：包含了很多的broker，但是在这么多的broker中也会有一个老大存在；是在kafka节点中的一个临时节点，去创建相应的数据，这个老大就是 **Controller Broker**。**Controller Broker职责**：管理所有的broker。

6、kafka监控及运维

6.1、一键启动zookeeper

- 创建一个文件，名字为【slave】

```
hadoop01
hadoop02
hadoop03
```

- startzk.sh

```
cat /export/servers/zk/bin/slave | while read line
do
{
echo $line
ssh $line "source /etc/profile;nohup zkServer.sh start >/dev/null 2>&1 &"
}&
wait
done
```

- stopzk.sh

```
cat /export/servers/zk/bin/slave | while read line
do
{
echo $line
ssh $line "source /etc/profile;jps |grep QuorumPeerMain |cut -c 1-4 |xargs kill -s 9"
}&
wait
done
```

说明：详细代码见附录文件夹

6.2、一键启动kafka

- 创建一个文件，名字为【slave】

```
hadoop01
hadoop02
hadoop03
```

- startkafka.sh


```
cat /export/servers/kafka/bin/slave | while read line
do
{
echo $line
ssh $line "source /etc/profile;nohup kafka-server-start.sh
/export/servers/kafka/config/server.properties >/dev/null 2>&1 &"
}&
wait
done
```

- stopkafka.sh

```
cat /export/servers/kafka/bin/slave | while read line
do
{
echo $line
ssh $line "source /etc/profile;jps |grep Kafka |cut -c 1-4 |xargs kill -s 9 "
}&
wait
done
```

6.3、kafka监控工具

6.3.1、kafka-eagle概述

在开发工作中，消费在Kafka集群中消息，数据变化是我们关注的问题，当业务前提不复杂时，我们可以使用Kafka命令提供带有Zookeeper客户端工具的工具，可以轻松完成我们的工作。随着业务的复杂性，增加Group和Topic，那么我们使用Kafka提供命令工具，已经感到无能为力，那么Kafka监控系统目前尤为重要，我们需要观察消费者应用的细节。

监控系统行业有很多优秀的开源监控系统。我们在早期，使用KafkaMonitor和Kafka Manager，但随着业务的快速发展，以及Internet Co的一些具体要求，现有的开源效率监控系统，以及在性能上扩展DEVS的使用，并且一直无法满足。

因此，我们在过去的的时间里，从互联网公司的需求出发，从DEVS开始，使用经验和反馈，结合一些业界的大型开源Kafka消息监控，从一些关于监控，设计和开发的想法开始监控系统现在Kafka集群消息：Kafka Eagle。

6.3.2、环境和安装

6.3.2.1、环境

6.3.2.1.1、安装JDK

如果Linux服务器上有JDK环境，则可以忽略此步骤，并安装链接的第二部分。如果没有JDK，请先到Oracle官方网站下载JDK

6.3.2.1.2、JAVA_HOME配置

JDK提取可以根据自己的实际情况来提取路径，这里我们解压缩到解压缩 `/usr/java/jdk1.8`，如下图所示：

```
cd /usr/java
tar -zxvf jdk-xxxx.tar.gz
mv jdk-xxxx jdk1.8
...
vi /etc/profile

export JAVA_HOME=/usr/java/jdk1.8
export PATH=$PATH:$JAVA_HOME/bin
```

然后，我们使用它 `./etc/profile` 来使配置立即生效。

5.3.2.1.3、检验环境配置

最后，我们 `java -version` 根据以下信息输入：

```
java version "1.8.0_60"
Java(TM) SE Runtime Environment (build 1.8.0_60-b27)
Java HotSpot(TM) 64-Bit Server VM (build 25.60-b23, mixed mode)
```

6.3.2.2、安装

6.3.2.2.1、下载安装包

1、下载地址： (<http://download.smartloli.org/>)

```
2https://github.com/smartloli/kafka-eagle-bin/archive/v1.2.6.tar.gz
```

2、下载地址： (<https://github.com/smartloli/kafka-eagle>)

6.3.2.2.2、解压

这里我们提取到/data/soft/new目录并解压缩，如下图所示：

```
tar -zxvf kafka-eagle-${version}-bin.tar.gz
```

如果您之前安装了该版本，请删除修改后的版本，并重命名当前版本，如下所示：

```
rm -rf kafka-eagle
mv kafka-eagle-${version} kafka-eagle
```

6.3.2.2.3、配置Kafka Eagle环境变量

vi /etc/profile

```
export KE_HOME=/data/soft/new/kafka-eagle
export PATH=$PATH:$KE_HOME/bin
```

5.3.2.2.4、配置Kafka Eagle系统配置文件

进入kafka-eagle的安装目录

```
cd ${KE_HOME}/conf
```

vi system-config.properties

```
# Multi zookeeper&kafka cluster list -- The client connection address of the zookeeper
cluster is set here
kafka.eagle.zk.cluster.alias=cluster1,cluster2
cluster1.zk.list=tdn1:2181,tdn2:2181,tdn3:2181
cluster2.zk.list=xdn1:2181,xdn2:2181,xdn3:2181

# Zkcli limit -- Zookeeper cluster allows the number of clients to connect to
kafka.zk.limit.size=25

# Kafka Eagle webui port -- webConsole port access address
kafka.eagle.webui.port=8048

# Kafka offset storage -- Offset stored in a Kafka cluster, if stored in the zookeeper,
you can not use this option
cluster1.kafka.eagle.offset.storage=kafka
cluster2.kafka.eagle.offset.storage=kafka

# Whether the Kafka performance monitoring diagram is enabled
kafka.eagle.metrics.charts=false

# If offset is out of range occurs, enable this property -- Only suitable for kafka sql
kafka.eagle.sql.fix.error=false
```

```
# Delete kafka topic token -- Set to delete the topic token, so that administrators can
have the right to delete
kafka.eagle.topic.token=keadmin

# kafka sasl authenticate, current support SASL_PLAINTEXT
kafka.eagle.sasl.enable=false
kafka.eagle.sasl.protocol=SASL_PLAINTEXT
kafka.eagle.sasl.mechanism=PLAIN
kafka.eagle.sasl.client=<kafka_client_jaas.conf file path>

# Default use sqlite to store data
kafka.eagle.driver=org.sqlite.JDBC
# It is important to note that the '/hadoop/kafka-eagle/db' path must exist.
kafka.eagle.url=jdbc:sqlite:/hadoop/kafka-eagle/db/ke.db
kafka.eagle.username=root
kafka.eagle.password=smartlo1i

# <Optional> set mysql address
#kafka.eagle.driver=com.mysql.jdbc.Driver
#kafka.eagle.url=jdbc:mysql://127.0.0.1:3306/ke?useUnicode=true&characterEncoding=UTF-
8&zeroDateTimeBehavior=convertToNull
#kafka.eagle.username=root
#kafka.eagle.password=smartlo1i
```

6.3.2.2.5、启动


进入该目录中

```
cd ${KE_HOME}/bin
```

给启动脚本赋值权限并启动，如下所示


```
chmod +x ke.sh
./ke.sh starth
```

6.3.2.2.6、主界面




4
Brokers

View Details




48
Topics

View Details



3
Zookeepers

View Details



5
Consumers

View Details

