# 官网

https://kafka.apache.org/

# 下载地址

https://kafka.apache.org/downloads

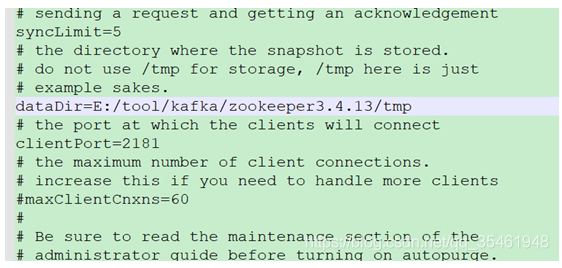
# windows安装Kafka

## 1、安装jdk环境

## 2、安装zookeeper

解压下载的文件zookeeper-3.4.13.tar.gz，并重命名解压后的文件为zookeeper3.4.13，复制到指定位置（此处以E:\tool\kafka\zookeeper3.4.13为例）。

## 3、配置Zookeeper



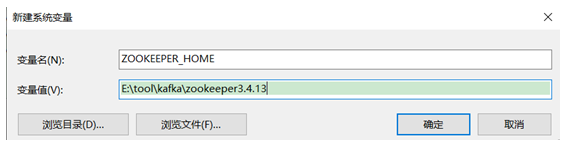
将 E:\tool\kafka\zookeeper3.4.13\conf下的zoo\_sample.cfg文件更改为zoo.cfg，打开“zoo.cfg”找到并编辑dataDir=E:/tool/kafka/zookeeper3.4.13/tmp

注意此处配置的路径一定是“/”而不是“\”，不然启动后会报错并自动关闭服务。

## 4、配置Zookeeper环境变量

我的电脑→右键选择属性→高级系统设置→环境变量

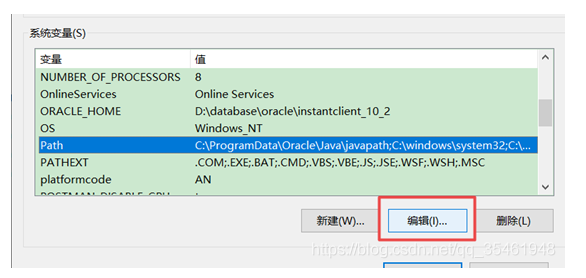
配置ZOOKEEPER\_HOME

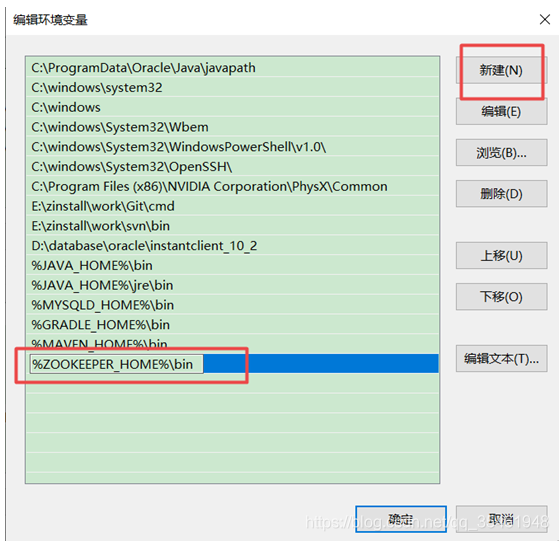


输入解压的目录：

E:\tool\kafka\zookeeper3.4.13

在系统环境变量Path中，增加“%ZOOKEEPER\_HOME%\bin”

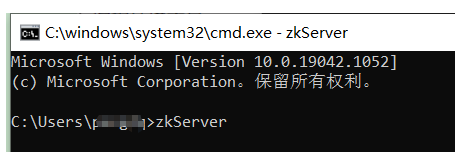


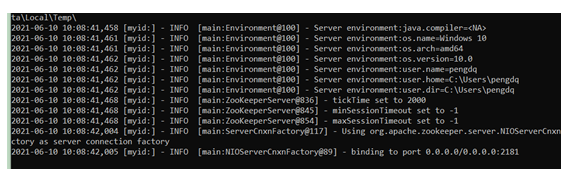


## 5、启动Zookeeper

按： windows+r， 输入cmd, 打开dos窗口，输入命令：

zkServer

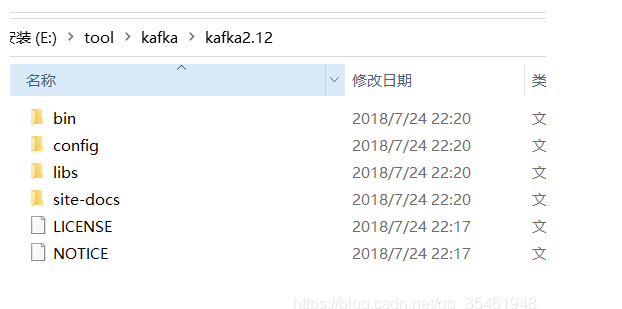




启动成功：  
注意在使用kafka 的时候，一定保持窗口不能关闭。

## 6、下载并解压Kafka

解压下载的文件kafka\_2.12-2.0.0.tgz，并重命名为kafka2.12，并复制到指定位置（此处以E:\tool\kafka\kafka2.12为例）



## 7、配置Kafka

打开E:\tool\kafka\kafka2.12\config\server.properties，里面的路径用 /

1、编辑log.dirs= E:/tool/kafka/kafka2.12/logs 或是：log.dirs= E:\\tool\\kafka\\kafka2.12\\logs

2、编辑broker.id=1。 配置broker.id必须是一个整数，且不可以和其他的kafak代理节点的值重复，即每个kafka节点为一个broker，拥有唯一的id值。

3、单机多BROKER集群配置：

新生成几个配置文件config/server1.properties、config/server2.properties。

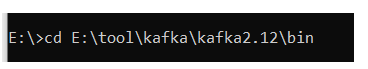
然后修改里面的配置文件broker.id、 log.dir、  listeners，分别指定不同的broker、日志文件位置、监听端口。



然后分别使用这些配置文件启动即可。

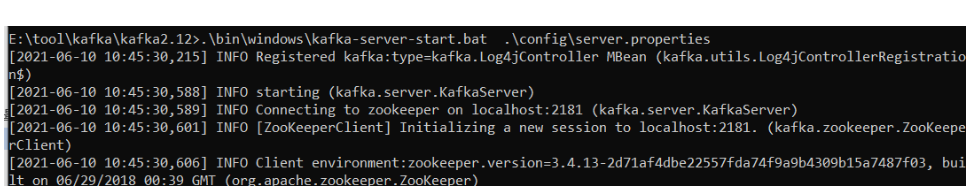
## 8、启动Kafka

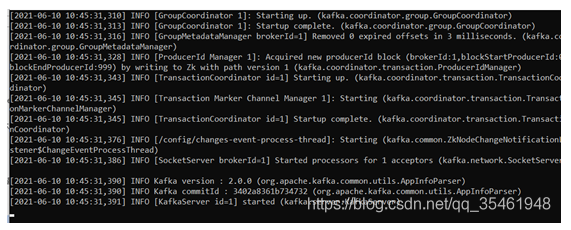
按：windows + r，输入cmd，新打开dos，进入kafka安装目录



输入：

.\bin\windows\kafka-server-start.bat .\config\server.properties





启动成功：

如果出现启动时错误： Found directory /home/admin/kafka\_2.12-1.1.0/bin, 'bin' is not in the form of topic-partition or topic-partition.uniqueId-delete (if marked for deletion).

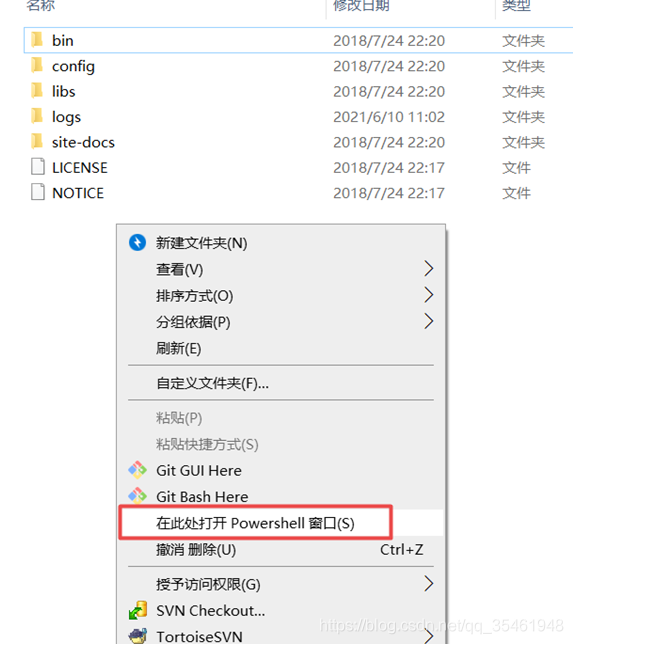
错误原因：server.properties里的配置log.dir路径不对

## 9、测试和使用

### （1）创建主题

使用shell创建

shift+鼠标右键，选中“在此处打开命令行窗口（W）”，打开命令行

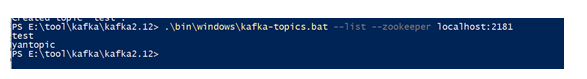


输入以下命令，创建一个topic主题： test

.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test

再查看创建的主题列表：

.\bin\windows\kafka-topics.bat --list --zookeeper localhost:2181



可以看到上面那个创建的topic和现在创建的topic

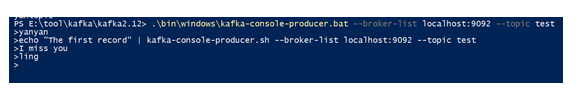
### （2）测试生产者和消费者

#### 启动生产者,可以发送消息

打开dos窗口，输入命令：

.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test

往里面输入消息

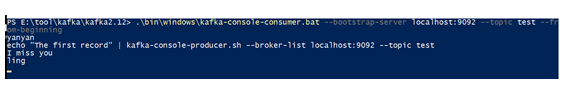


#### 接收消息

窗口不要关闭，同上新打开一个shell窗口，输入命令：

.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --from-beginning

接收



## 停止zookeeper

1、先进入zookeeper的bin目录

2、window环境下执行下面的命令：

zkServer.cmd stop

3、linux环境下执行下面的命令：

zkServer.sh stop

## 停止Kafka

1、打开cmd窗口，先进入kafka目录

2、window环境下执行下面的命令：

bin\windws\kafka-server-stop.bat

3、linux环境下执行下面的命令：

bin\kafka-server-stop.sh

## 客户端工具

可以使用Kafka Tool工具连接Kafka。

# CentOS搭建Kafka集群

https://www.cnblogs.com/frankdeng/p/9403883.html

# Kafka Tool使用教程

## 1、下载

下载地址：http://www.kafkatool.com/download.html



## 2、安装

根据不同的系统下载对应的版本

图形用户界面

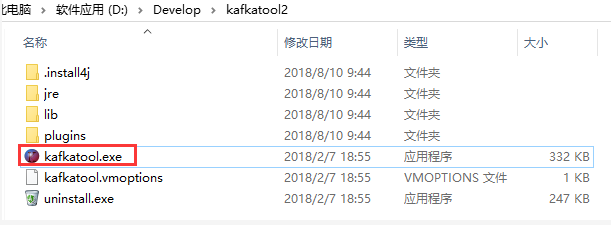
描述已自动生成

双击下载完成的exe图标，傻瓜式完成安装。

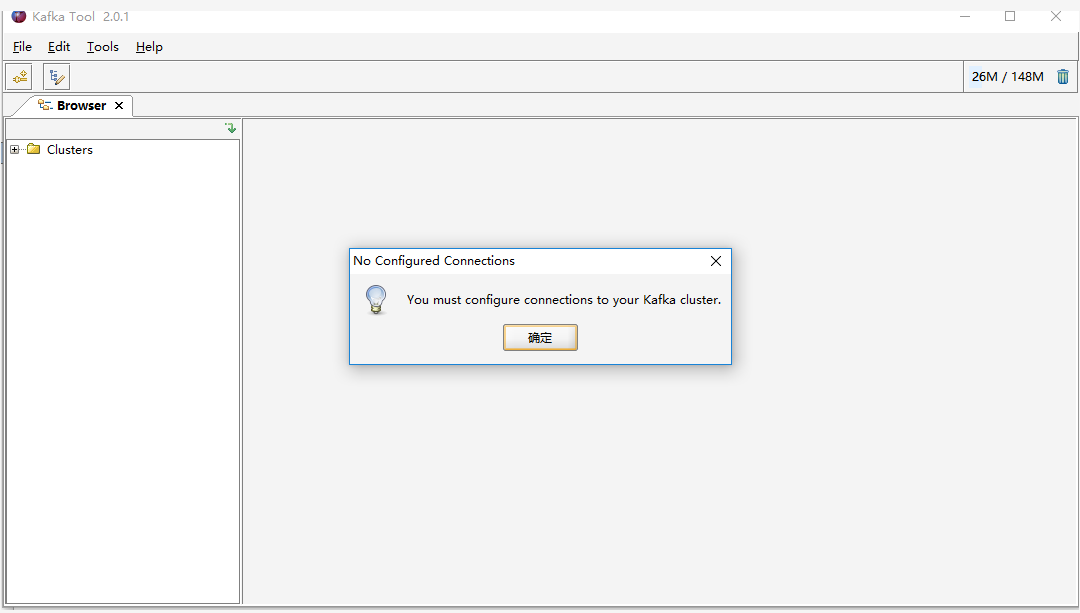
## 3、使用

### （1）连接kafka

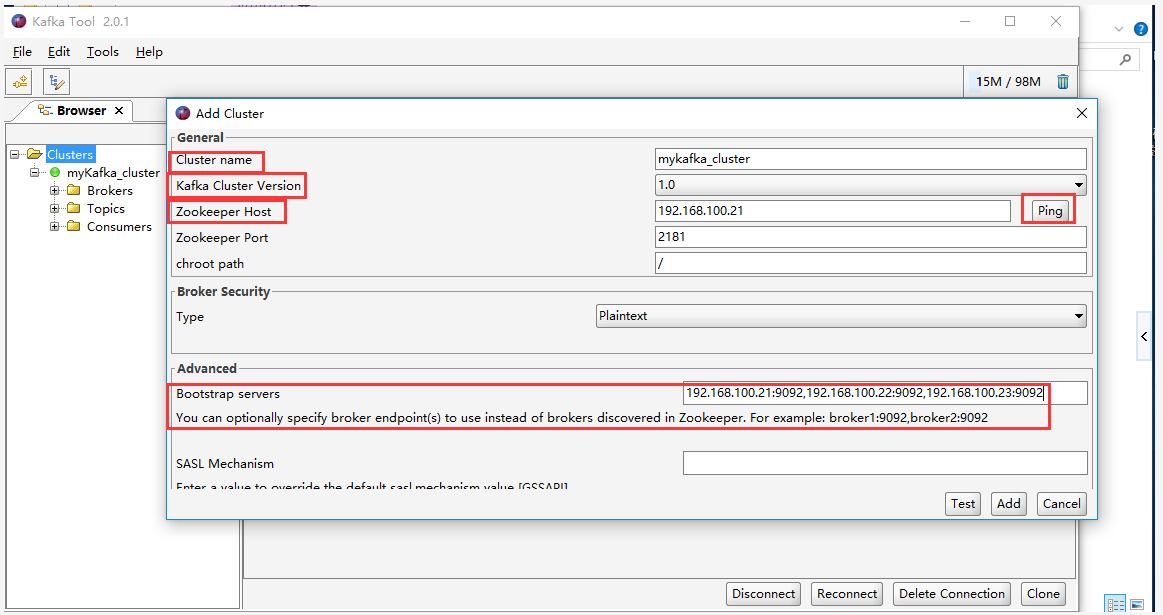
打开kafka tool安装目录，点击exe文件



提示设置kafka集群连接



点击确定，设置



设置完了，点击Test测试是否能连接，连接通了，然后点击Add，添加完成设置。出现如下界面

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

### （2）简单使用

配置以字符串的形式显示kafka消息体

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

图形用户界面, 文本

描述已自动生成

或者通过如下界面配置

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

注释：更改完Content Types，要点击Update和Refresh按钮

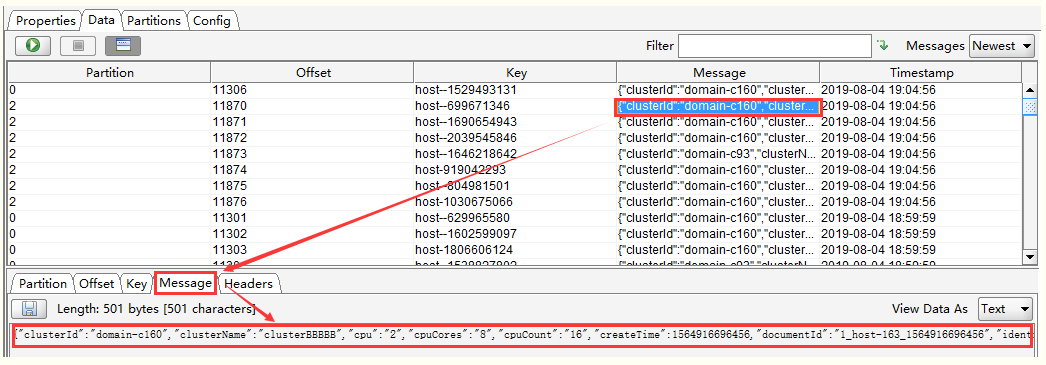
再次查看kafka的数据：

图形用户界面, 应用程序, 表格

描述已自动生成

### （3）查看具体某一条消息

选中某一个message，点击“Message”查看详情，默认是“Text”格式（如果想看到JSON格式，只需要设置下“View Data As”即可）



图形用户界面, 文本, 应用程序

描述已自动生成

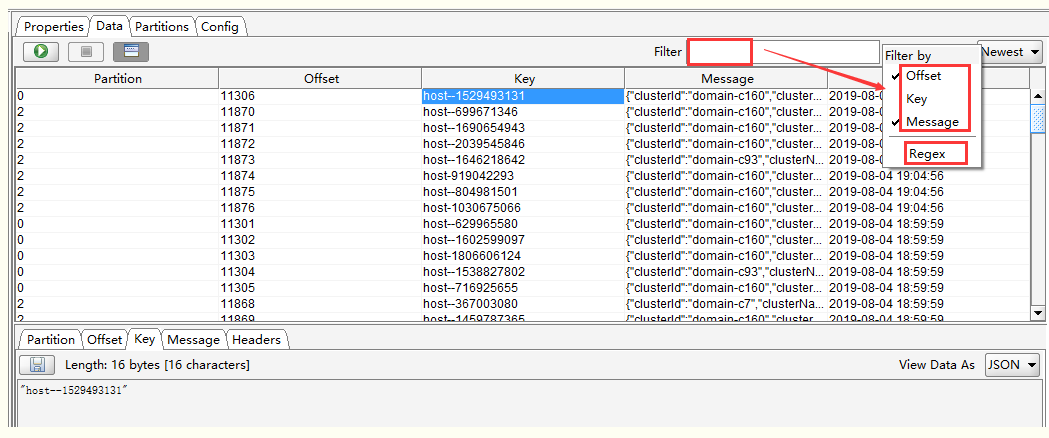
“View Data As”设置为 JSON 后，消息信息格式如下

图形用户界面, 应用程序, 表格

描述已自动生成

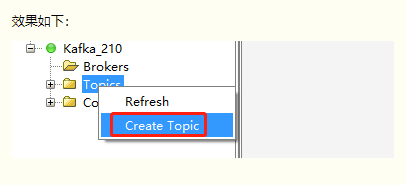
### （4）过滤查询 Kafka 消息

可以按照Offset（偏移量），Key（消息key），Message（消息主体），Regex（正则表达式）进行单个或者组合查询



### （5）创建topic

步骤如下：  
1、鼠标右键点击 “Topics” 文件夹图标 --> 选择 “Create Topic”，或者点击“Topics”，单击右侧页面中的“+”，会弹出“Add Topic”页面；  
2、填写需要创建的Topic名称，分区数和备份数，点击“Add”添加，即可创建 topic。



图形用户界面, 文本, 应用程序, 电子邮件

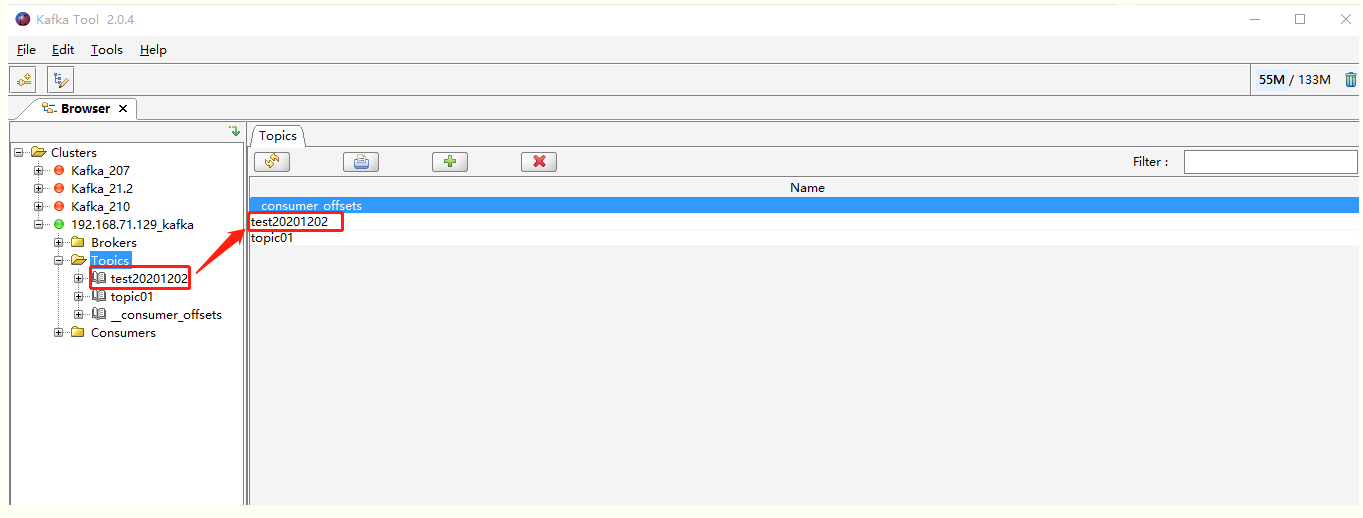
描述已自动生成

图形用户界面, 文本, 应用程序, Word

描述已自动生成

### （6）查看创建的topic

创建的出来的 topic， 可以在 Topic 列表或者通过 Kafka topic 命令查看。  
自测的截图如下：



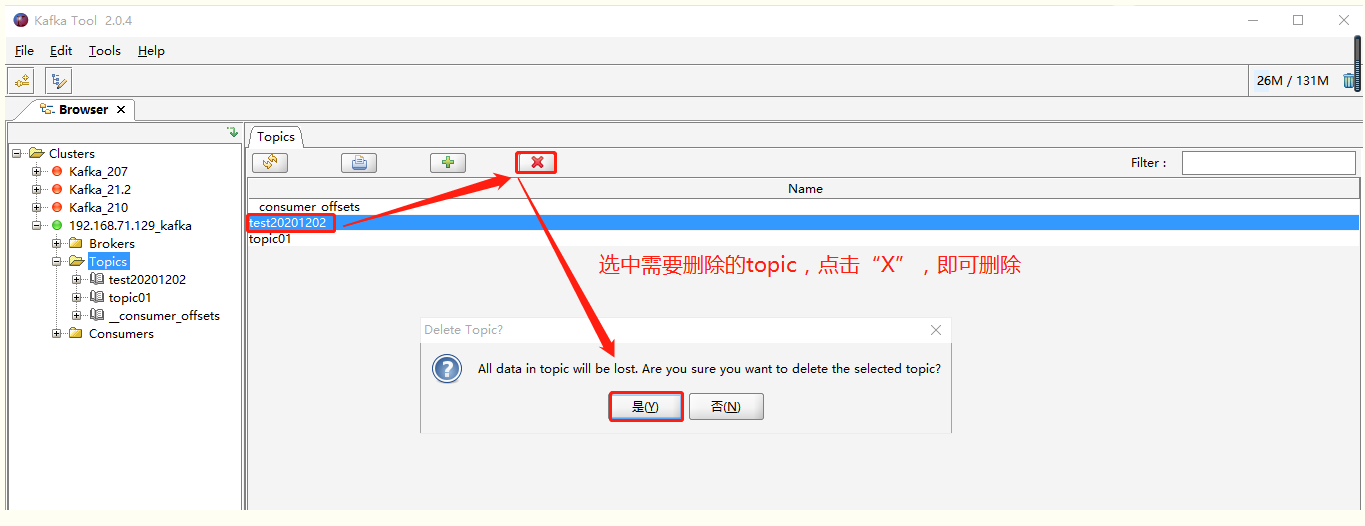
文本

描述已自动生成

### （7）删除topic

如果因为失误导致创建的 topic 不是自己想要的，想进行删除，也可以通过 KafkaTool 进行界面化删除。

效果如下：



图形用户界面, 文本, 应用程序, 电子邮件

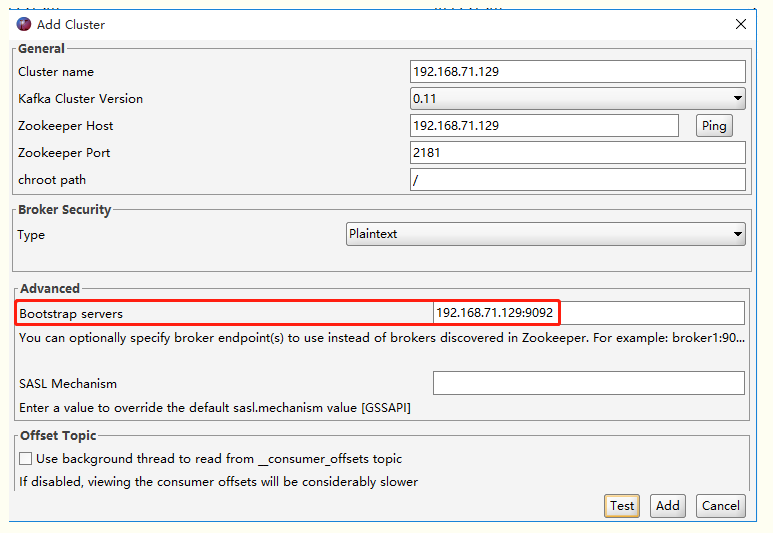
描述已自动生成

补充说明：

如果通过 Kafka Tool 创建 topic 失败，请确认如下两点：

**是否按照如下截图配置的Cluster Connection**

（本次举例是单台机器，如果实际是集群，需要把 bootstrap servers进行集群配置，如3台集群配置：192.168.71.129:9092,192.168.71.130:9092,192.168.71.131:9092）



**是否按照博客里的说明在hosts文件中配置了域名**

图形用户界面, 应用程序, Word

描述已自动生成

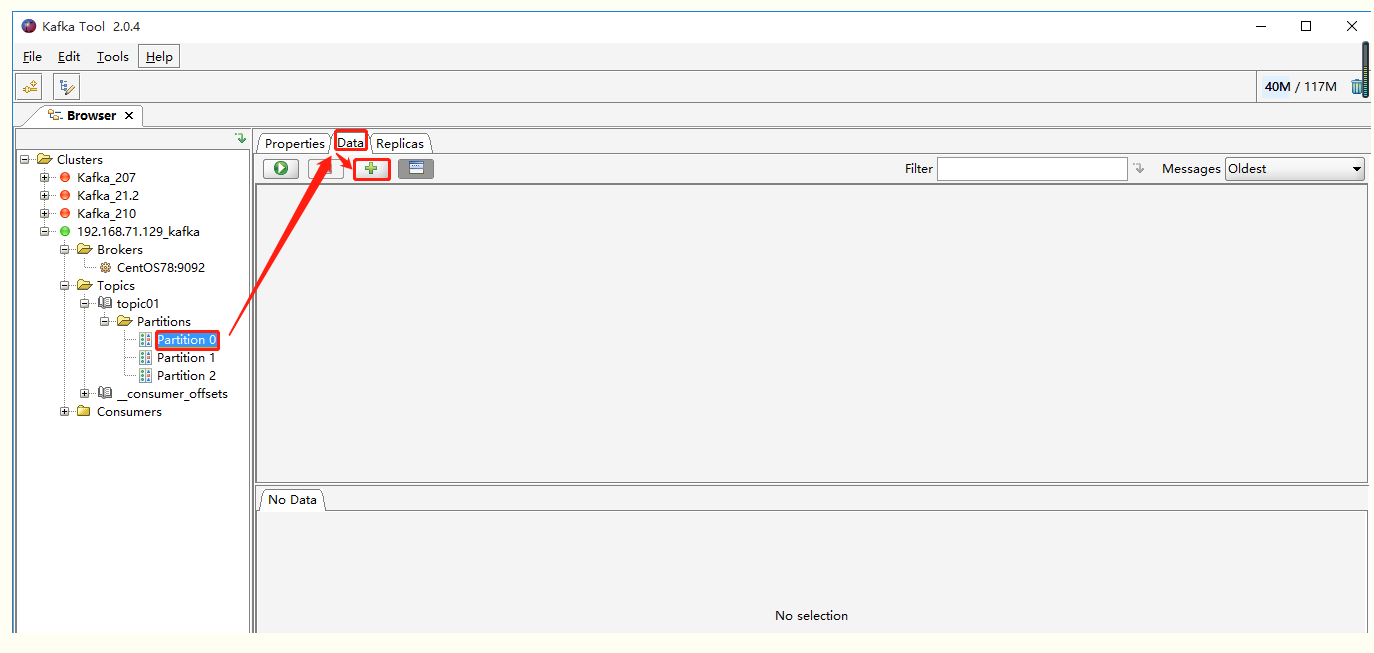
如果上述两点均已确认无误，依旧不能通过 Kafka Tool 正确创建出 topic。请确认 Kafka 服务本身是否正常，是否可以通过 kafka topic 命令创建出来 topic。

### （8）使用 Kafka Tool 模拟发送 Messages

操作步骤如下：

点击“Topics” --> 选择一个 topic（如：topic01） --> 选择分区（如：Partition 0），点击右侧“Data”里的“+”，会弹出“Add Message”页面。  
其中，Key 和 Message 均支持两种方式：From file 和 Enter Manually[Text]  
**From file**：就是数据来源于文件，需要上传文件（个人建议上传txt格式文件）  
**Enter Manually[Text]**：手动输入  
二者可以混合搭配使用，并且 Key 和 Message **二者必须有值**，否则会报错

效果如下：



#### 场景1：Key 和 Message 均选择手动输入

图形用户界面, 应用程序

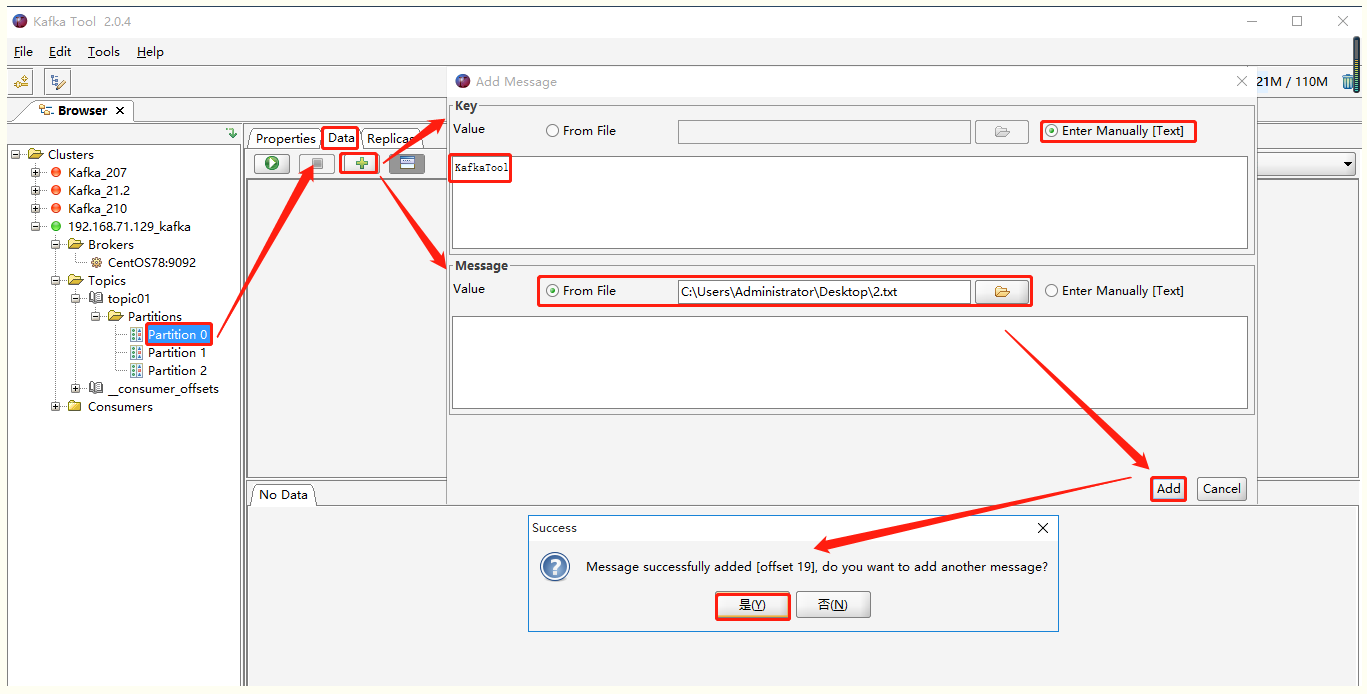
描述已自动生成

#### 场景2：Key 和 Message 均选择来自文件

图形用户界面, 应用程序

描述已自动生成

#### 场景3：Key 和 Message 其中一个来自文件，一个来自手动输入



图形用户界面, 应用程序

描述已自动生成

2.txt 文件中内容如下：

文本

描述已自动生成

#### 消费者消费到的模拟数据

截图：

图形用户界面, 文本

描述已自动生成

消费命令如下：



# kafka默认端口号

9092

# 查看Kafka的版本号

## 在linux系统下面执行的命令

bin/kafka-broker-api-versions.sh --bootstrap-server localhost:9092 –version

# Kafka哪个版本不需要zk

2021年4月19日，Kafka 2.8.0正式发布！这次升级包括了很多重要的改动，其中最引人瞩目的就是kafka通过自我管理的仲裁来替代ZooKeeper，通俗的说，Kafka将不再需要ZooKeeper，正式分手。

# 基本概念

Kafka是分布式发布-订阅消息系统，它最初是由LinkedIn公司开发的，之后成为Apache项目的一部分，Kafka是一个分布式，可划分的，冗余备份的持久性的日志服务，它主要用于处理流式数据。

# 优点

## 1、高吞吐量

单机每秒处理几十上百万的消息量。即使存储了许多TB的消息，它也保持稳定的性能。

## 2、高性能

单节点支持上千个客户端，并保证零停机和零数据丢失。单机测试能达到 100w tps。

## 3、低延迟

Kafka可以轻松处理这些消息，具有毫秒级的极低延迟。

## 4、容错

Kafka能够抵抗集群中的节点/机器故障。

## 5、耐久性

由于Kafka支持消息复制，因此消息永远不会丢失，这是耐久性背后的原因之一。

## 6、可扩展性

Kafka可以扩展，而不需要通过添加额外的节点而在运行中造成任何停机。

# 缺点

1、由于是批量发送，所以数据达不到真正的实时。

2、对于mqtt协议不支持。

3、不支持物联网传感数据直接接入。

4、只能支持统一分区内消息有序，无法实现全局消息有序。

5、监控不完善，需要安装插件。

6、需要配合zookeeper进行元数据管理。

# 为什么要使用 kafka

## （1）缓冲和削峰

上游数据时有突发流量，下游可能扛不住，或者下游没有足够多的机器来保证冗余，kafka在中间可以起到一个缓冲的作用，把消息暂存在kafka中，下游服务就可以按照自己的节奏进行慢慢处理。

## （2）解耦和扩展性

项目开始的时候，并不能确定具体的需求。消息队列可以作为一个接口层，解耦重要的业务流程。只需要遵守约定，针对数据编程即可获取扩展能力。

## （3）冗余

可以采用一对多的方式，一个生产者发布消息，可以被多个订阅topic的服务消费到，供多个毫无关联的业务使用。

## （4）健壮性

消息队列可以堆积请求，所以消费端业务即使短时间死掉，也不会影响主要业务的正常进行。

## （5）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。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。

# 主要特征

1、快速持久化，可以在O(1)的系统开销下进行消息持久化；

2、高吞吐，在一台普通的服务器上既可以达到10W/s的吞吐速率

3、完全的分布式系统，Broker、Producer、Consumer都原生自动支持分布式，自动实现负载均衡

4、支持同步和异步复制两种HA

5、支持数据批量发送和拉取

6、zero-copy：减少IO操作步骤

7、数据迁移、扩容对用户透明

8、无需停机即可扩展机器

# Zookeeper在Kafka中的作用

1、帮助kafka集群存储一些信息。

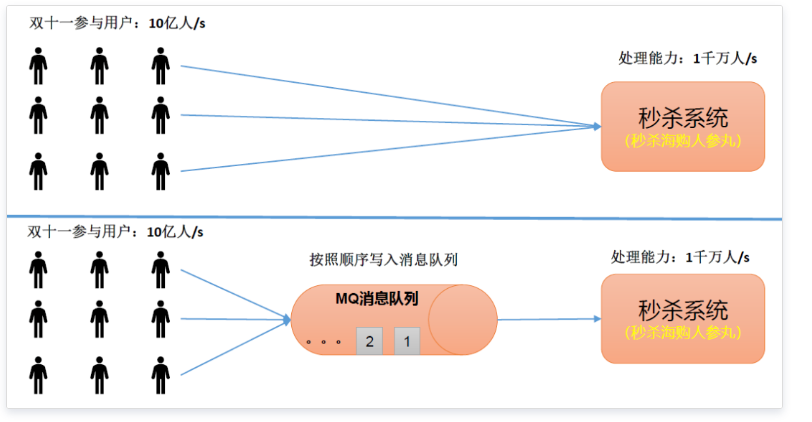
2、帮助消费者存储消费到的位置信息。

说明：在kafka0.9版本之前offset消费到的消息的偏移量是存储在zk中，0.9版本及以后offset存在kafka本地的系统topic中，无论是存储在哪都是为了记录消费的位置，以便集群宕掉后重启能继续原来的位置接着消费，zk虽然可以用来存储数据，但是它并不是存储数据的首选方案，如果存储在zk中，消费者会频繁地从zk中拉取消息，背离了zk的设计初衷。

# 应用场景

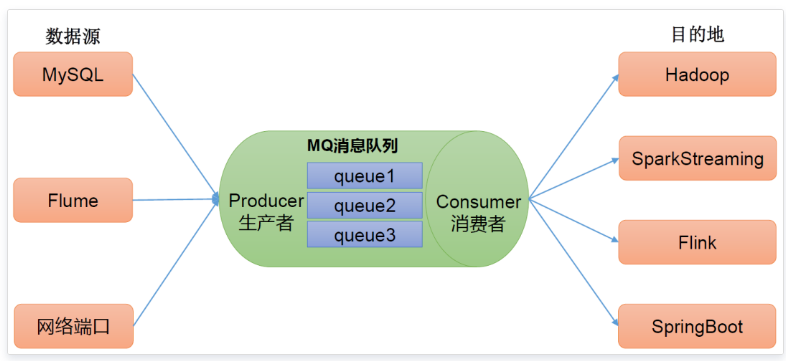
## 1、****削峰填谷****

所谓的“削峰填谷”就是指缓冲上下游瞬时突发的流量，使其更平滑。



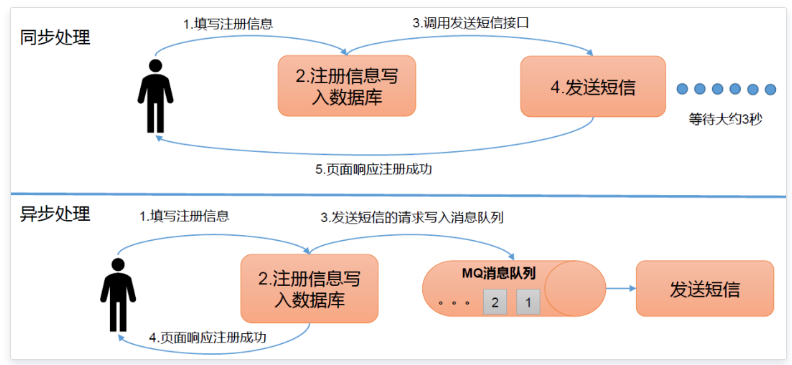
## ****2、解耦****

即允许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。

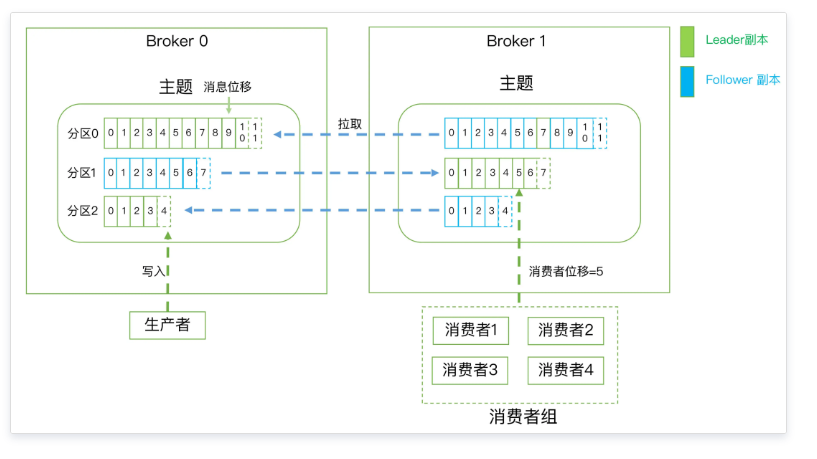


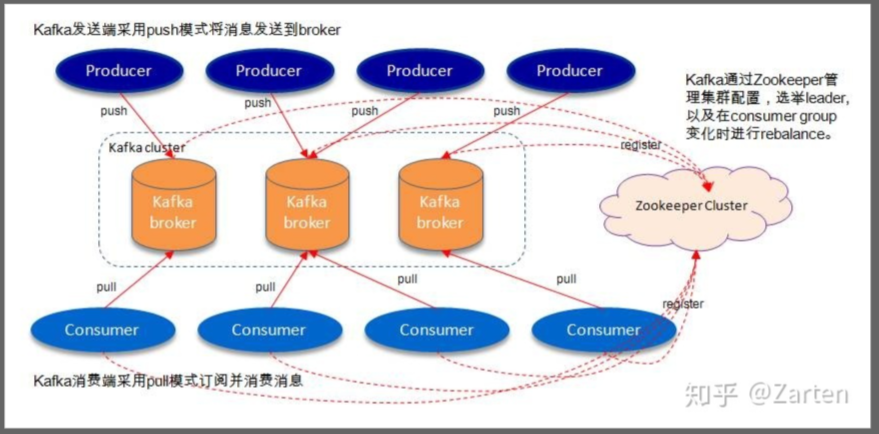
## 3、异步通信

允许把一个消息放入队列，但并不立即处理它们，然后在需要的时候才去处理它们。



# 架构图





# 组件

## 主题（Topic）

主题是一堆或一组消息。kafka将消息以 topic 为单位进行归类。topic 特指 kafka 处理的消息源（feeds of messages）的不同分类。

## 分区（partition）

partition是物理上的概念，每个topic 包含一个或多个partition。kafka 分配的单位是partition。Partition是Kafka中比较特色的部分，一个Topic 可以分为多个 Partition，每个Partition是一个有序的队列，Partition中的每条消息都存在一个有序的偏移量（Offest），同一个Consumer Group中，只有一个Consumer实例可消费某个 Partition 的消息。partition可以看作一个有序的队列，里面的数据是储存在硬盘中的，追加式的。

partition的作用就是提供分布式的扩展，一个topic可以有许多partions，多个partition可以并行处理数据，所以可以处理相当多的数据。只有partition的leader才会进行读写操作，follower仅进行复制，客户端是感知不到的。

## 消息位移（Offset）

表示分区中每条消息的位置信息，是一个单调递增且不变的值。

## 副本（Replica）

Kafka中同一条消息能够被拷贝到多个地方以提供数据冗余，这些地方就是所谓的副本。副本还分为领导者副本和追随者副本，各自有不同的角色划分。副本是在分区层级下的，即每个分区可配置多个副本实现高可用。

为了保证集群中的某个节点发生故障时，该节点上的partition数据不丢失，并且kafka 仍然能够继续工作，kafka 提供了副本机制，一个topic的每个分区都有若干个副本,一个leader 和若干个 follower。

## Leader

每个分区多个副本的“主”，生产者发送数据的对象，以及消费者消费数据的对象都是 leader。

## follow

每个分区多个副本中的“从”，实时从leader中同步数据，保持和 leader 数据的同步。leader发生故障时，某个follower 会成为新的leader。

## 生产者（Producer）

主要是用于生产消息，是 kafka 当中的消息生产者，生产的消息通过 topic 进行归类，保存到kafka的broker里面去。

## 消费者（Consumer）

consumer是kafka当中的消费者，主要用于消费 kafka 当中的数据，消费者一定是归属于某个消费组中的。

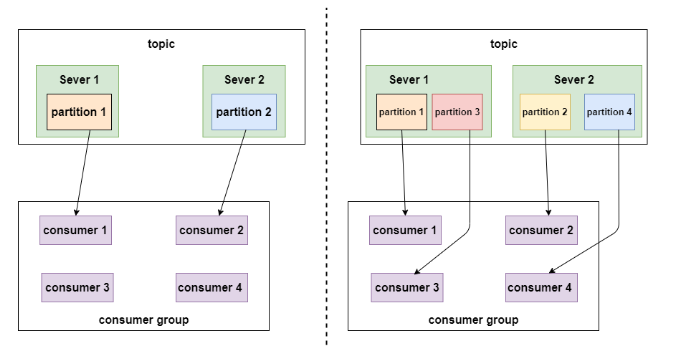
## 消费者位移

表示消费者消费的进度，每个消费者都有自己的消费者位移。

## consumer group（消费者组）

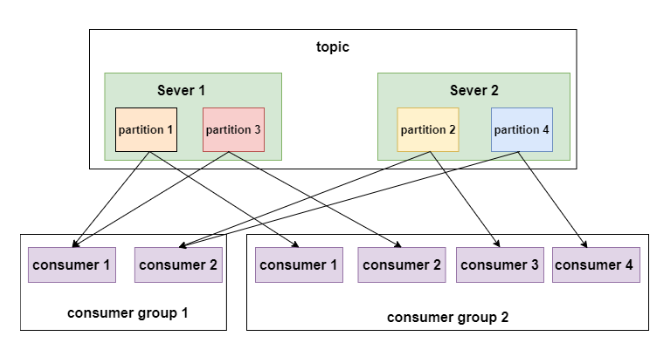
消费者组由一个或者多个消费者组成，同一个组中的消费者对于同一条消息只消费一次。每个消费者都属于某个消费者组，如果不指定，那么所有的消费者都属于默认的组。每个消费者组都有一个ID，即group ID。组内的所有消费者协调在一起来消费一个订阅主题( topic)的所有分区(partition)。当然每个分区只能由同一个消费组内的一个消费者(consumer)来消费，可以由不同的消费组来消费。一个partition分区中的消息只能被某一个消费者组中的一个消费者消费，这样设计的目的是为了提高消费者组的并发度。

partition数量决定了每个consumer group中并发消费者的最大数量。如下图：



如上面左图所示，如果只有两个分区，即使一个组内的消费者有4个，也会有两个空闲的。如上面右图所示，有 4 个分区，每个消费者消费一个分区，并发量达到最大4。

在来看如下一幅图：



如上图所示，不同的消费者组消费同一个topic，这个 topic 有 4 个分区，分布在两个节点上。左边的消费组 1 有两个消费者，每个消费者就要消费两个分区才能把消息完整的消费完，右边的消费组2 有四个消费者，每个消费者消费一个分区即可。

### ****总结下kafka 中分区与消费组的关系****

消费组：由一个或者多个消费者组成，同一个组中的消费者对于同一条消息只消费一次。某一个主题下的分区数，对于消费该主题的同一个消费组下的消费者数量，应该小于等于该主题下的分区数。

如：某一个主题有 4 个分区，那么消费组中的消费者应该小于等于 4，而且最好与分区数成整数倍1 2 4 这样。**同一个分区下的数据，在同一时刻不能同一个消费组的不同消费者消费**。

总结：分区数越多，同一时间可以有越多的消费者来进行消费，消费数据的速度就会越快，提高消费的性能。当一个消费者组中的消费者个数与主题的分区数一致时才最合理，如果消费者个数过多就造成了性能浪费。

## Broker

Kafka系统中服务节点，通常情况下，Kafka系统中一台服务器主机被称为Kafka系统的一个代理节点。代理节点的主要作用是负责消息数据的存储、为客户端提供服务、保证Kafka系统的正常运行等。代理节点是Kafka系统组建集群的最小单位，一个Kafka集群由一个或者多个代理节点组成。

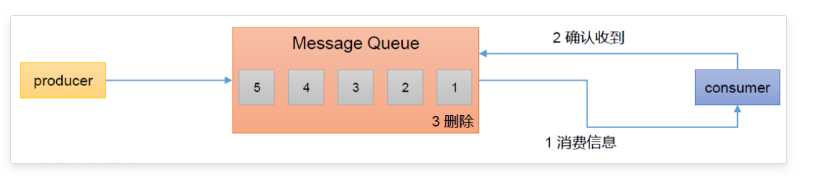
## 重平衡

消费者组内某个消费者实例挂掉后，其他消费者实例自动重新分配订阅主题分区的过程。Rebalance是Kafka 消费者端实现高可用的重要手段。

# 支持的消息传输模式

## 1、****点对点模型****

消费者主动拉取数据，消息收到后清除消息。

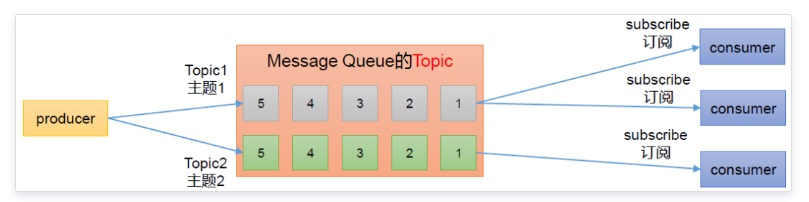


## ****2、发布/订阅模型****

1、可以有多个topic主题（例如：浏览、点赞、收藏、评论等）

2、消费者消费数据之后，不删除数据

3、每个消费者相互独立，都可以消费到数据



**Kafka同时支持这两种消息引擎模型**。

# 消息的有效期

kafka的消息数据存储在磁盘中，默认有效期是7天，如果遇到磁盘空间小，存放数据量大，可以设置缩短这个时间。如果存储在内存中不宜保存这么久。

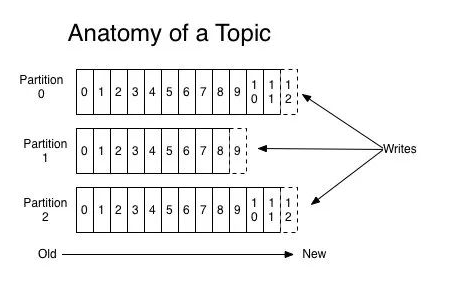
# 偏移的作用

给分区中的消息提供了一个顺序ID号，我们称之为偏移量。因此为了唯一地识别分区中的每条消息，我们使用这些偏移量。

# Kafka为什么那么快

## 1、利用 Partition实现并行处理

## 2、顺序写磁盘



**Kafka 中每个分区是一个有序的，不可变的消息序列**，新的消息不断追加到 partition 的末尾，这个就是顺序写。

由于磁盘有限，不可能保存所有数据，实际上作为消息系统 Kafka 也没必要保存所有数据，需要删除旧的数据。又由于顺序写入的原因，所以 Kafka 采用各种删除策略删除数据的时候，并非通过使用“读 - 写”模式去修改文件，而是将 Partition 分为多个 Segment，每个 Segment 对应一个物理文件，通过删除整个文件的方式去删除 Partition 内的数据。这种方式清除旧数据的方式，也避免了对文件的随机写操作。

## 3、充分利用 Page Cache

## 4、零拷贝技术

## 5、批处理

## 6、数据压缩

# Kafka的message 格式

一个Kafka的Message 由一个固定长度的 header和一个变长的消息体 body 组成。

1、header部分，由一个字节的 magic(文件格式)和四个字节的 CRC32(用于判断body 消息体是否正常)构成。

2、当 magic 的值为 1的时候，会在 magic 和 crc32 之间多一个字节的数据attributes(保存一些相关属性，比如是否压缩、压缩格式等等)

3、如果 magic的值为 0，那么不存在attributes 属性。

4、body 部分 由N个字节构成的一个消息体，包含了具体的 key/value 消息。

# ISR副本同步机制

## 基本概念

### AR：（Assigned Replicas）

一个partition的所有副本（就是replica，不区分leader或follower）

### ISR（In-Sync Replicas）

能够和 leader 保持同步的 follower + leader本身组成的集合。

### OSR（Out-Sync Relipcas）

不能和 leader 保持同步的follower集合。

ISR（in-sync replica）就是 Kafka 为某个分区维护的一组同步集合，即每个分区都有自己的一个 ISR 集合，处于 ISR 集合中的副本，意味着 follower 副本与 leader 副本保持同步状态，只有处于 ISR 集合中的副本才有资格被选举为 leader。一条 Kafka消息，只有被ISR中的副本接收到才被视为“已同步”状态。

# Kafka API

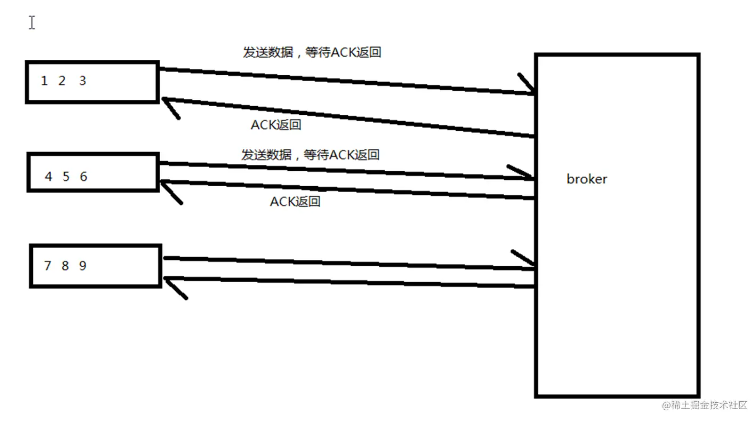
## 基本概念

Kafka的API有Producer API，Consumer API还有自定义Interceptor (自定义拦截器)，以及处理的流使用的Streams API和构建连接器的Kafka Connect API。

## Producer API​

Kafka的Producer发送消息采用的是异步发送的方式。在消息发送过程中，涉及两个线程：main线程和Sender线程，以及一个线程共享变量RecordAccumulator。main线程将消息发送给RecordAccmulator，Sender线程不断地从RecordAccumulator中拉取消息发送给Kafka broker。

这里的ACk机制，不是生产者得到ACK返回信息才开始发送，ACK保证的是生产者不丢失数据。例如：



而是只要有消息数据，就向broker发送。

### 消息发送流程

生产者使用send方法，经过拦截器之后在经过序列化器，然后在走分区器。然后通过分批次把数据发送到RecordAccumulator，main线程到此过程就结束了，然后在回去执行send。Sender线程不断的获取RecordAccumulator的数据发送到topic。消息发送流程是异步发送的，并且顺序是一定的拦截器-》序列化器-》分区器。

### 异步发送API

需要用到的类：

KafkaProducer: 需要创建一个生产者对象，用来发送数据

ProducerConfig：获取所需要的一系列配置参数

ProducerRecord：每条数据都要封装成一个ProducerRecord对象

示例：



配置参数说明：

send():方法是异步的，添加消息到缓冲区等待发送，并立即返回。生产者将单个的消息批量在一起发送来提高效率。

ack:是判断请求是否完整的条件（就会判断是不是成功发送了，也就是上次说的ACK机制），指定all将会阻塞消息，性能低但是最可靠。

retries:如果请求失败，生产者会自动重试，我们指定是1次，但是启动重试就有可能出现重复数据。

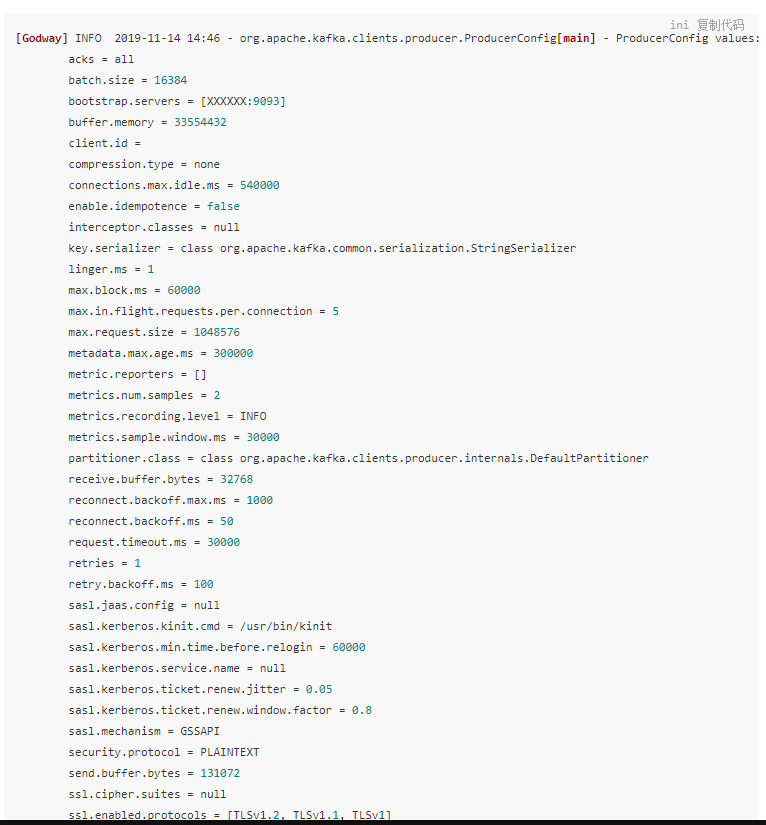
batch.size:指定缓存的大小，生产者缓存每个分区未发送的消息。值越大的话将会产生更大的批量，并需要更大的内存（因为每个活跃的分区都有一个缓存区）。

linger.ms:指示生产者发送请求之前等待一段时间，设置等待时间是希望更多地消息填补到未满的批中。默认缓冲可以立即发送，即便缓冲空间还没有满，但是如果想减少请求的数量可以设置linger.ms大于0。需要注意的是在高负载下，相近的时间一般也会组成批，即使等于0。

buffer.memory:控制生产者可用的缓存总量，如果消息发送速度比其传输到服务器的快，将会耗尽这个缓存空间。当缓存空间耗尽，其他发送调用将被阻塞，阻塞时间的阈值通过max.block.ms设定，之后将会抛出一个TimeoutException

key.serializer和value.serializer将用户提供的key和value对象ProducerRecord转换成字节，你可以使用附带的ByteArraySerializaer或StringSerializer处理简单的string或byte类型。

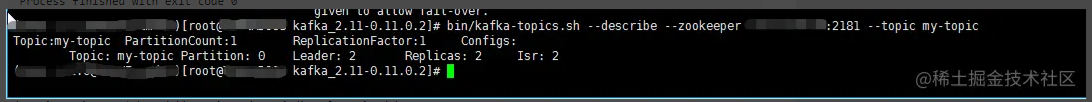
运行日志：



文本

描述已自动生成

有一条警告{my-topic=LEADER\_NOT\_AVAILABLE} 提示该topic不存在，但是没有关系kafka会自动给你创建一个topic，不过创建的topic是有一个分区和一个副本：



查看一下该topic的消息：

图形用户界面, 文本

中度可信度描述已自动生成

消息已经在topic里了

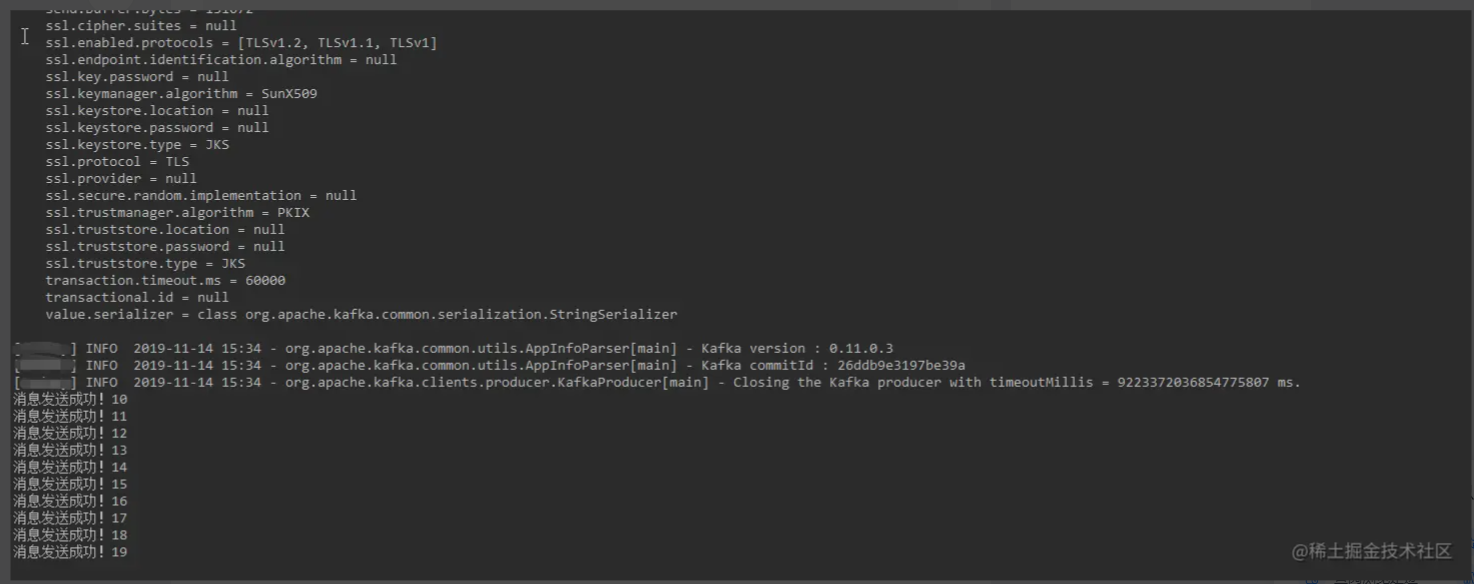
上面的实例是没有回调函数的，send方法是有回调函数的：

文本

描述已自动生成

回调函数会在producer收到ack时调用，为异步调用，该方法有两个参数，分别是RecordMetadata和Exception，如果Exception为null,说明消息发送成功，如果Exception不为null说明消息发送失败。

注意： 消息发送失败会自动重试，不需要我们在回调函数中手动重试，使用回调也是无阻塞的。而且callback一般在生产者的IO线程中执行，所以是非常快的，否则将延迟其他的线程消息发送。如果需要执行阻塞或者计算的回调（耗时比较长），建议在callbanck主体中使用自己的Executor来并行处理。



### 同步发送API

同步发送的意思就是，一条消息发送之后，会阻塞当前的线程，直到返回ack（此ack和异步的ack机制不是一个ack）。此ack是Future阻塞main线程，当发送完成就返回一个ack去通知main线程已经发送完毕，继续往下走了

public Future send(ProducerRecord<K,V> record,Callback callback)

send是异步的，并且一旦消息被保存在等待发送的消息缓存中，此方法就立即返回。这样并行发送多条消息而不阻塞去等待每一条消息的响应。发送的结果是一个RecordMetadata，它指定了消息发送的分区，分配的offset和消息的时间戳。如果topic使用的是CreateTime，则使用用户提供的时间戳或发送的时间（如果用户没有指定指定消息的时间戳）如果topic使用的是LogAppendTime，则追加消息时，时间戳是broker的本地时间。

由于send调用是异步的，它将为分配消息的此消息的RecordMetadata返回一个Future。如果future调用get()，则将阻塞，直到相关请求完成并返回该消息的metadata，或抛出发送异常。

Throws:

InterruptException - 如果线程在阻塞中断。

SerializationException - 如果key或value不是给定有效配置的serializers。

TimeoutException - 如果获取元数据或消息分配内存话费的时间超过max.block.ms。

KafkaException - Kafka有关的错误（不属于公共API的异常）。



## Consumer API​

### 基本概念

kafka客户端通过TCP长连接从集群中消费消息，并透明地处理kafka集群中出现故障服务器，透明地调节适应集群中变化的数据分区。也和服务器交互，平衡均衡消费者。

### 偏移量和消费者的位置

kafka为分区中的每条消息保存一个偏移量（offset），这个偏移量是该分区中一条消息的唯一标示符。也表示消费者在分区的位置。例如，一个位置是5的消费者(说明已经消费了0到4的消息)，下一个接收消息的偏移量为5的消息。实际上有两个与消费者相关的“位置”概念：

消费者的位置给出了下一条记录的偏移量。它比消费者在该分区中看到的最大偏移量要大一个。 它在每次消费者在调用poll(long)中接收消息时自动增长。

“已提交”的位置是已安全保存的最后偏移量，如果进程失败或重新启动时，消费者将恢复到这个偏移量。消费者可以选择定期自动提交偏移量，也可以选择通过调用commit API来手动的控制(如：commitSync 和 commitAsync)。

这个区别是消费者来控制一条消息什么时候才被认为是已被消费的，控制权在消费者。

### 消费者组和主题订阅

# Kafka怎么保证消息不重复

Kafka可以通过消费者组来保证消息不重复消费。在Kafka中，每个消费者组中的消费者只能消费同一个主题的不同分区，同一个分区只能由同一个消费者组中的一个消费者进行消费。因此，如果要保证消息不重复消费，可以将同一个生产者发送的消息都发送到同一个分区中，然后由同一个消费者组中的一个消费者进行消费，这样就可以保证同一个分区中的消息只会被消费一次。

另外Kafka还提供了一个offset的概念，可以用来记录消费者消费的位置。消费者可以通过指定offset来消费指定位置之后的消息，从而避免消息的重复消费。如果消费者在消费过程中出现了故障，可以通过记录消费者的offset来恢复消费进度，从而避免消息的重复消费或者丢失。

# 怎么保证消息幂等性

# Kafka怎么保证顺序消费

1、1个Topic（主题）只创建1个Partition(分区)，这样生产者的所有数据都发送到了一个Partition(分区)，保证了消息的消费顺序。

2、生产者在发送消息的时候指定要发送到哪个Partition(分区)。

怎么指定呢？我们需要将 producer 发送的数据封装成一个 ProducerRecord 对象。

（1）指明partition的情况下，直接将指明的值直接作为 partiton 值。

（2）没有指明 partition 值但有 key 的情况下，将 key 的 hash 值与 topic 的 partition数进行[取余](https://so.csdn.net/so/search?q=%E5%8F%96%E4%BD%99&spm=1001.2101.3001.7020)得到 partition 值。

在Producer往Kafka插入数据时，控制同一Key分发到同一Partition，并且设置参数max.in.flight.requests.per.connection=1，也即同一个链接只能发送一条消息，如此便可严格保证Kafka消息的顺序。

既没有 partition 值又没有 key 值的情况下，第一次调用时随机生成一个整数（后面每次调用在这个整数上自增），将这个值与 topic可用的 partition 总数取余得到 partition值，也就是常说的 round-robin 算法。

那么问题来了：在1个topic中，有3个partition，那么如何保证数据的消费？

1、如顺序消费中的 "**第①点"**和 "Kafka 要保证消息的消费顺序第二个方法" 说明，生产者在写的时候，可以指定一个 key，比如说我们指定了某个订单 id 作为 key，那么这个订单相关的数据，一定会被分发到同一个 partition 中去，而且这个 partition 中的数据一定是有顺序的。

2、消费者从 partition 中取出来数据的时候，也一定是有顺序的。到这里，顺序还是 ok 的，没有错乱。

3、但是消费者里可能会有多个线程来并发来处理消息。因为如果消费者是单线程消费数据，那么这个吞吐量太低了。而多个线程并发的话，顺序可能就乱掉了。

解决方案：

写N个queue，将具有相同key的数据都存储在同一个queue，然后对于N个线程，每个线程分别消费一个queue即可。

注：在单线程中，一个 topic，一个 partition，一个 consumer，内部单线程消费，这样的状态数据消费是有序的。但由于单线程吞吐量太低，在数据庞大的实际场景很少采用。

但是以上消费线程模型，存在一个问题：在消费过程中，如果 Kafka 消费组发生重平衡，此时的分区被分配给其它消费组了，如果拉取回来的消息没有被消费，虽然 Kakfa 可以实现 ConsumerRebalanceListener 接口，在新一轮重平衡前主动提交消费偏移量，但这貌似解决不了未消费的消息被打乱顺序的可能性？

因此在消费前，还需要主动进行判断此分区是否被分配给其它消费者处理，并且还需要锁定该分区在消费当中不能被分配到其它消费者中（但 kafka 目前做不到这一点）

# 如何保证消息不丢失

# 延迟队列功能

# 为什么Kafka不支持读写分离

# 遇到的问题

## 1、win10启动kafka报错: ERROR Failed to clean up log for \_\_consumer\_offsets-39 in dir d:\work\data\kafka-logs due to IOException (kafka.server.LogDirFailureChannel) java.nio.file.FileSystemException: d:\work\data\kafka-logs\_\_consumer\_offsets-39\00000000000000000000.timeindex.cleaned: 另一个程序正在使用此文件，进程无法访问。

### 解决方法

可能是由于非正常关闭, 导致kafka无法完成对日志文件的解锁错误信息中指出了日志的存放路径, 我这里是d:\work\data\kafka-logs, 将这个目录下的所有文件删除后重新启动kafka即可。