

- 一、快速了解Kafka
 - 1、MQ的作用
 - 2、Kafka产品介绍
 - 3、Kafka的特点
- 二、快速上手Kafka
 - 1、快速搭建单机服务
 - 2、简单收发消息
 - 3、理解消费者组
 - 4、理解Kafka的消息传递机制
- 三、理解Kafka的集群工作机制
 - 1、搭建Kafka集群
 - 2、理解服务端的Topic、Partition和Broker
- 四、章节总结：Kafka集群的消息流转模型

一、Kafka快速上手

-- 楼兰

这一章节主要是快速了解Kafka产品，搭建Kafka服务，并开始了解Kafka的基础功能。

一、快速了解Kafka

1、MQ的作用

MQ：MessageQueue，消息队列。队列，是一种FIFO 先进先出的数据结构。消息则是跨进程传递的数据。一个典型的MQ系统，会将消息消息由生产者发送到MQ进行排队，然后根据一定的顺序交由消息的消费者进行处理。

QQ和微信就是典型的MQ。只不过他对接的使用对象是人，而Kafka需要对接的使用对象是应用程序。

MQ的作用主要有以下三个方面：

- 异步

例子：快递员发快递，直接到客户家效率会很低。引入菜鸟驿站后，快递员只需要把快递放到菜鸟驿站，就可以继续发其他快递去了。客户再按自己的时间安排去菜鸟驿站取快递。

作用：异步能提高系统的响应速度、吞吐量。

- 解耦

例子：《Thinking in JAVA》很经典，但是都是英文，我们看不懂，所以需要编辑社，将文章翻译成其他语言，这样就可以完成英语与其他语言的交流。

作用：

1、服务之间进行解耦，才可以减少服务之间的影响。提高系统整体的稳定性以及可扩展性。

2、另外，解耦后可以实现数据分发。生产者发送一个消息后，可以由一个或者多个消费者进行消费，并且消费者的增加或者减少对生产者没有影响。

- 削峰

例子：长江每年都会涨水，但是下游出水口的速度是基本稳定的，所以会涨水。引入三峡大坝后，可以把水储存起来，下游慢慢排水。

作用：以稳定的系统资源应对突发的流量冲击。

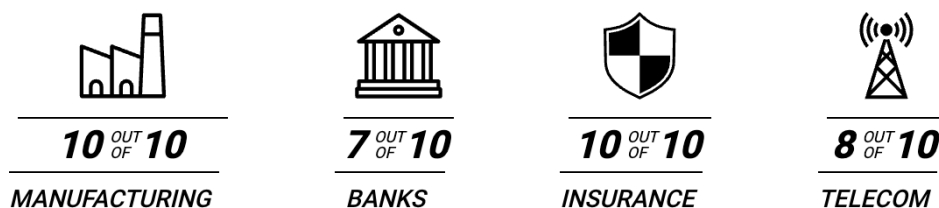
2、Kafka产品介绍

Kafka是目前最具影响力的开源MQ产品，官网地址：<https://kafka.apache.org/>

APACHE KAFKA

*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.



[SEE FULL LIST](#)

Above is a snapshot of the number of top-ten largest companies using Kafka, per-industry.

Apache Kafka最初由LinkedIn开发并于2011年开源。他主要解决大规模数据的实时流式处理和数据管道问题。

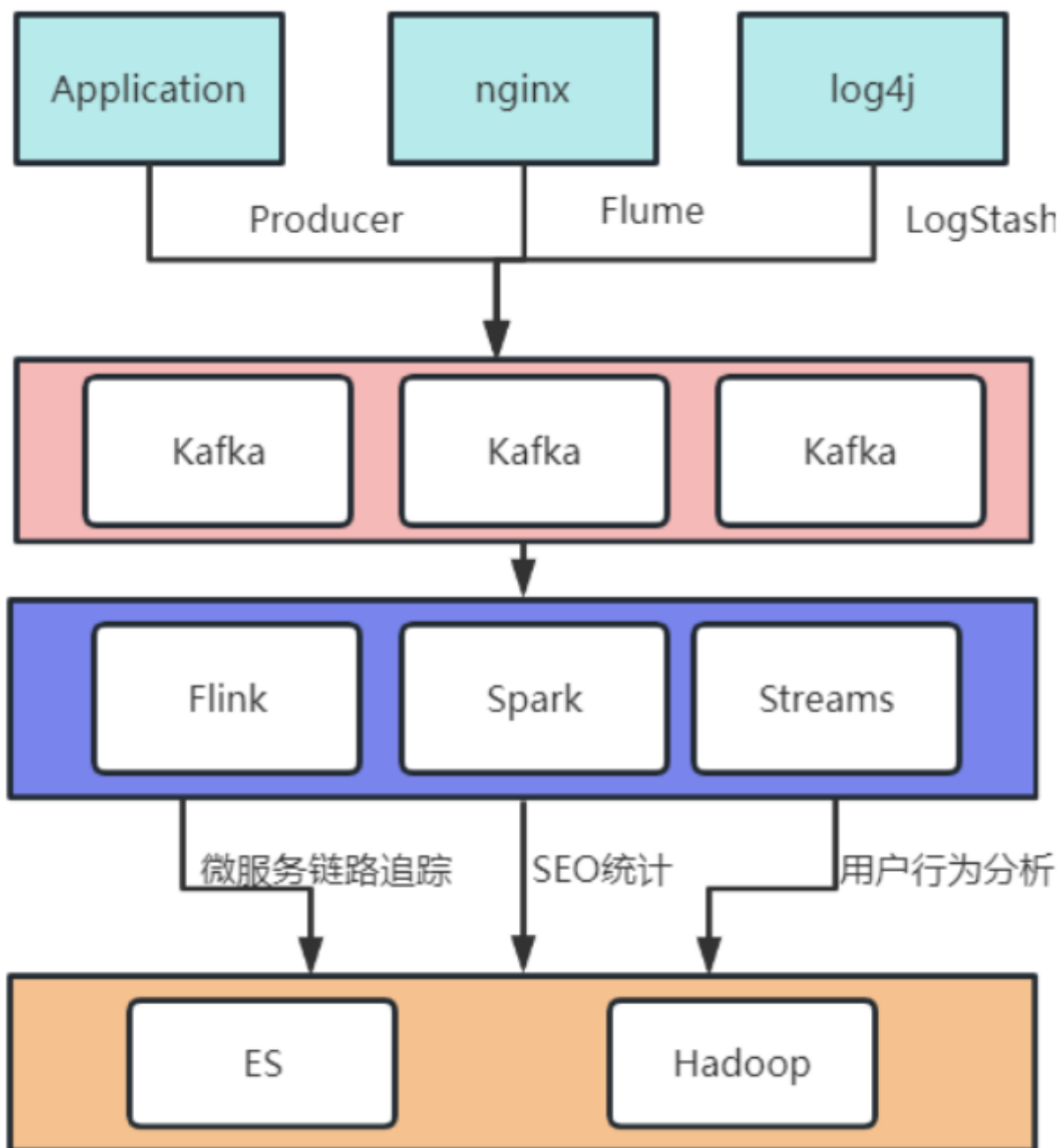
Kafka是一个分布式的发布-订阅消息系统，可以快速处理高吞吐量的数据流，并将数据实时地分发到多个消费者中。Kafka消息系统由多个broker（服务器）组成，这些broker可以在多个数据中心之间分布式部署，以提供高可用性和容错性。

Kafka使用高效的数据存储和管理技术，能够轻松地处理TB级别的数据量。其优点包括高吞吐量、低延迟、可扩展性、持久性和容错性等。

Kafka在企业级应用中被广泛应用，包括实时流处理、日志聚合、监控和数据分析等方面。同时，Kafka还可以与其他大数据工具集成，如Hadoop、Spark和Storm等，构建一个完整的数据处理生态系统。

3、Kafka的特点

Kafka最初诞生于LinkedIn公司，其核心作用就是用来收集并处理庞大复杂的应用日志。一个典型的日志聚合应用场景如下：



业务场景决定了产品的特点。所以Kafka最典型的产品特点有以下几点：

- 1、**数据吞吐量很大**：需要能够快速收集各个渠道的海量日志
- 2、**集群容错性高**：允许集群中少量节点崩溃
- 3、**功能不需要太复杂**：Kafka的设计目标是高吞吐、低延迟和可扩展，主要关注消息传递而不是消息处理。所以，Kafka并没有支持死信队列、顺序消息等高级功能。
- 4、**允许少量数据丢失**：在海量的应用日志中，少量的日志丢失是不会影响结果的。所以Kafka的设计初衷是允许少量数据丢失的。当然Kafka本身也在不断优化数据安全问题。

二、快速上手Kafka

1、快速搭建单机服务

Kafka的运行环境非常简单，只要有JVM虚拟机就可以进行。这里，我们使用一台安装了JDK1.8的CentOS9机器作为演示。

JDK的安装过程略

下载Kafka。官网下载地址：<https://kafka.apache.org/downloads> 这里我们选择下载kafka_2.13-3.8.0.tgz

关于kafka的版本，前面的2.13是开发kafka的scala语言的版本，后面的3.8.0是kafka应用的版本。

Scala是一种运行于JVM虚拟机之上的语言。在运行时，只需要安装JDK就可以了，选哪个Scala版本没有区别。但是如果要调试源码，就必须选择对应的Scala版本。因为Scala语言的版本并不是向后兼容的。

下载Zookeeper，下载地址 <https://zookeeper.apache.org/releases.html>，Zookeeper的版本并没有强制要求，这里我们选择3.8.4版本。

kafka的安装程序中自带了Zookeeper，可以在kafka的安装包的libs目录下查看到zookeeper的客户端jar包。但是，通常情况下，为了让应用更好维护，我们会使用单独部署的Zookeeper，而不使用kafka自带的Zookeeper。

下载完成后，将这两个工具包上传到三台服务器上，解压后，分别放到/app/kafka和/app/zookeeper目录下。然后配置KAFKA_HOME环境变量指向kafka安装目录。接下来将这两个组件部署目录下的bin目录路径配置到path环境变量中。

下载下来的Kafka安装包不需要做任何的配置，就可以直接单击运行。这通常是快速了解Kafka的第一步。

启动Kafka之前需要先启动Zookeeper

这里就用Kafka自带的Zookeeper。启动脚本在bin目录下。

```
cd $KAFKA_HOME
nohup bin/zookeeper-server-start.sh config/zookeeper.properties &
```

注意下脚本是不是有执行权限。

从nohup.out中可以看到zookeeper默认会在2181端口启动。通过jps指令看到一个QuorumPeerMain进程，确定服务启动成功。

启动Kafka

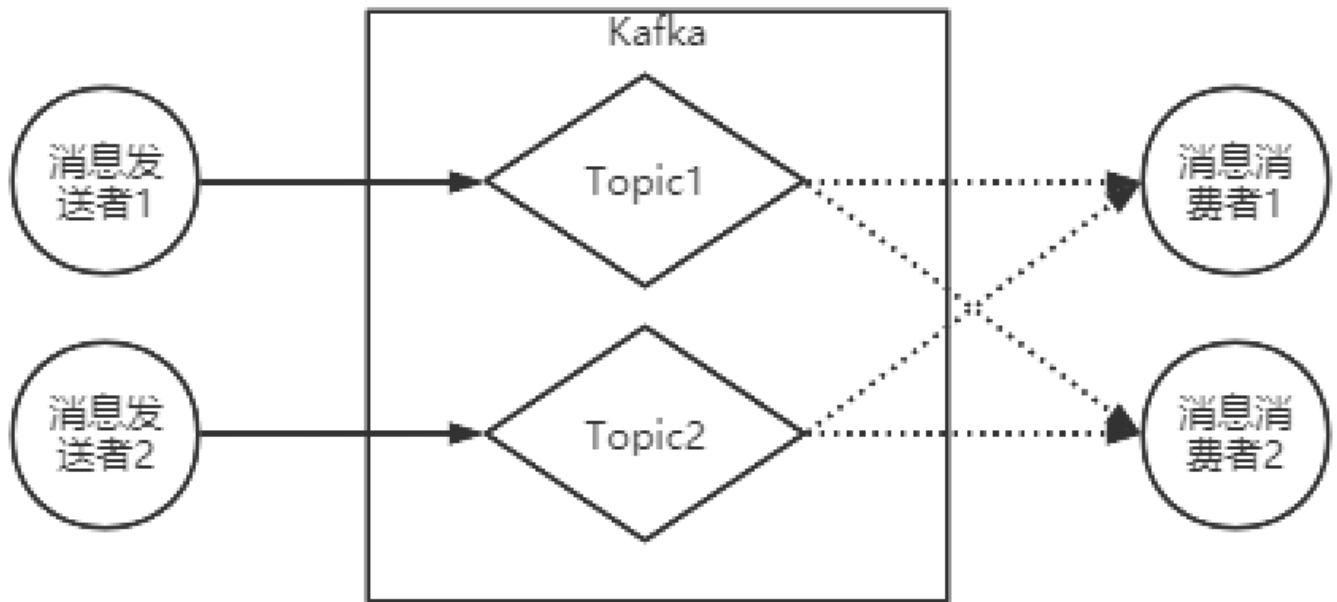
```
nohup bin/kafka-server-start.sh config/server.properties &
```

启动完成后，使用jps指令，看到一个kafka进程，确定服务启动成功。服务会默认在9092端口启动。

2、简单收发消息

Kafka的基础工作机制是消息发送者可以将消息发送到kafka上指定的topic，而消息消费者，可以从指定的topic上消费消息。

![] (file:///Users/roykingw/Desktop/a-work/kafka/%E7%AC%AC%E5%85%AD%E6%9C%9FVIP/img/1-1.png?lastModify=1723207746)



首先，可以使用Kafka提供的客户端脚本创建Topic

```
#创建Topic
bin/kafka-topics.sh --create --topic test --bootstrap-server localhost:9092
#查看Topic
bin/kafka-topics.sh --describe --topic test --bootstrap-server localhost:9092
```

然后，启动一个消息发送者端。往一个名为test的Topic发送消息。

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```

当命令行出现 > 符号后，随意输入一些字符。Ctrl+C 退出命令行。这样就完成了往kafka发消息的操作。

如果不提前创建Topic，那么在第一次往一个之前不存在的Topic发送消息时，消息也能正常发送，只是会抛出LEADER_NOT_AVAILABLE警告。

```
[oper@worker1 kafka_2.13-3.2.0]$ bin/kafka-console-producer.sh --broker-list
localhost:9092 --topic test
>123
12[2021-03-05 14:00:23,347] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 1 : {test=LEADER_NOT_AVAILABLE}
(org.apache.kafka.clients.NetworkClient)
3[2021-03-05 14:00:23,479] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 3 : {test=LEADER_NOT_AVAILABLE}
(org.apache.kafka.clients.NetworkClient)

[2021-03-05 14:00:23,589] WARN [Producer clientId=console-producer] Error while
fetching metadata with correlation id 4 : {test=LEADER_NOT_AVAILABLE}
(org.apache.kafka.clients.NetworkClient)
>>123
```

这是因为Broker端在创建完主题后，会显示通知Clients端LEADER_NOT_AVAILABLE异常。Clients端接收到异常后，就会主动去更新元数据，获取新创建的主题信息。

然后启动一个消息消费端，从名为test的Topic上接收消息。

```
[oper@worker1 kafka_2.13-3.2.0]$ bin/kafka-console-consumer.sh --bootstrap-server
localhost:9092 --topic test
qwe
qwe
123
123
123
^CProcessed a total of 5 messages
```

这样就完成了一个基础的交互。这其中，生产者和消费者并不需要同时启动。他们之间可以进行数据交互，但是又并不依赖于对方。没有生产者，消费者依然可以正常工作，反过来，没有消费者，生产者也依然可以正常工作。这也体现出了生产者和消费者之间的解耦。

如果想要查看这个脚本的详细参数，可以直接访问这个脚本，不配置任何参数即可。

4、其他消费模式

之前我们通过kafka提供的生产者和消费者脚本，启动了一个简单的消息生产者以及消息消费者，实际上，kafka还提供了丰富的消息消费方式。

指定消费进度

通过kafka-console.consumer.sh启动的控制台消费者，会将获取到的内容在命令行中输出。如果想要消费之前发送的消息，可以通过添加--from-beginning参数指定。

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic
test
```

如果需要更精确的消费消息，甚至可以指定从哪一条消息开始消费。

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --partition 0 --offset 4 -
-topic test
```

这表示从第0号Partition上的第四个消息开始读起。Partition和Offset是什么呢，后面会介绍到。

3、理解消费者组

对于每个消费者，可以指定一个消费者组。kafka中的同一条消息，只能被同一个消费者组下的某一个消费者消费。而不属于同一个消费者组的其他消费者，也可以消费到这一条消息。在kafka-console-consumer.sh脚本中，可以通过--consumer-property group.id=testGroup来指定所属的消费者组。例如，可以启动三个消费者组，来验证一下分组消费机制：

```
#两个消费者实例属于同一个消费者组
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --consumer-property
group.id=testGroup --topic test
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --consumer-property
group.id=testGroup --topic test
#这个消费者实例属于不同的消费者组
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --consumer-property
group.id=testGroup2 --topic test
```

查看消费者组的偏移量

接下来，还可以使用kafka-consumer-groups.sh观测消费者组的情况。包括他们的消费进度。

```
bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group
testGroup
```

```
[oper@worker1 kafka_2.13-3.4.0]$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group testGroup
Consumer group 'testGroup' has no active members.
```

GROUP	TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
testGroup	test	0	10	10	0	-	-	-

当前消费者组无消费者

当前消费组没有消费过的消息

分区

当前消费进度

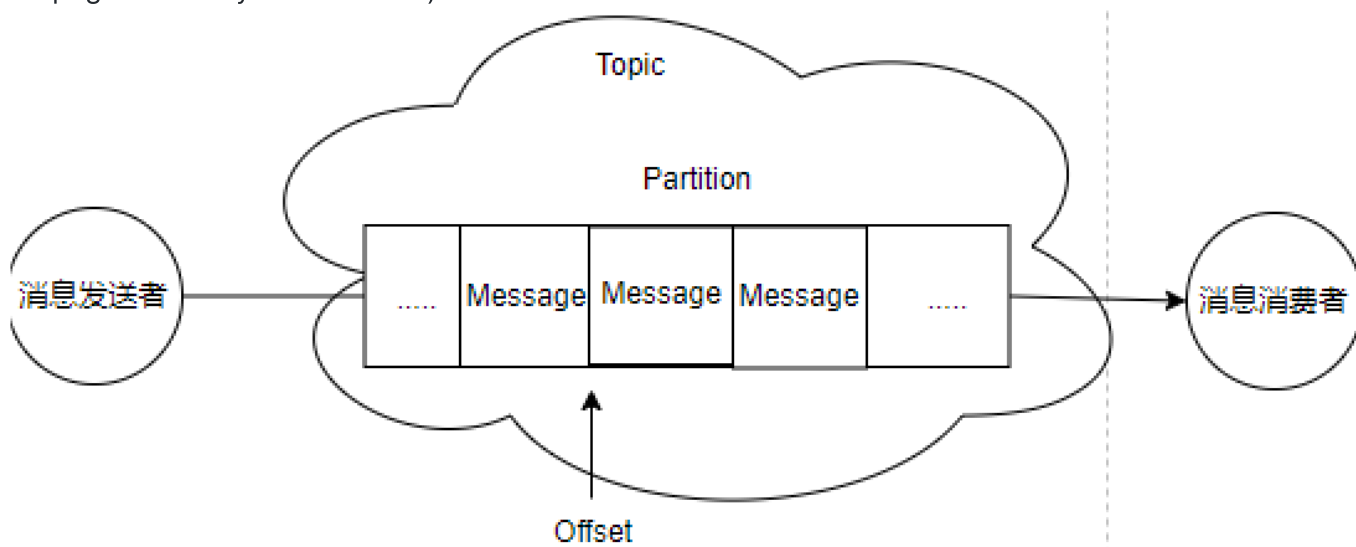
日志中的最大消息进度

从这里可以看到，Kafka是以消费者组为单位来分别记录每个Partition上的消息偏移量的。而增加新的消费者组，并不会影响Kafka的消息数据，只是需要新增一条偏移量记录就可以了。所以，Kafka的消息复读效率是很高的。

4、理解Kafka的消息传递机制

从之前的实验可以看到，Kafka的消息发送者和消息消费者通过Topic这样一个逻辑概念来进行业务沟通。但是实际上，所有的消息是存在服务端的Partition这样一个数据结构当中的。

![] (file:///Users/roykingw/Desktop/a-work/kafka/%E7%AC%AC%E5%85%AD%E6%9C%9FVIP/img/1-05.png?lastModify=1723208714)



在Kafka的技术体系中，有以下一些概念需要先熟悉起来：

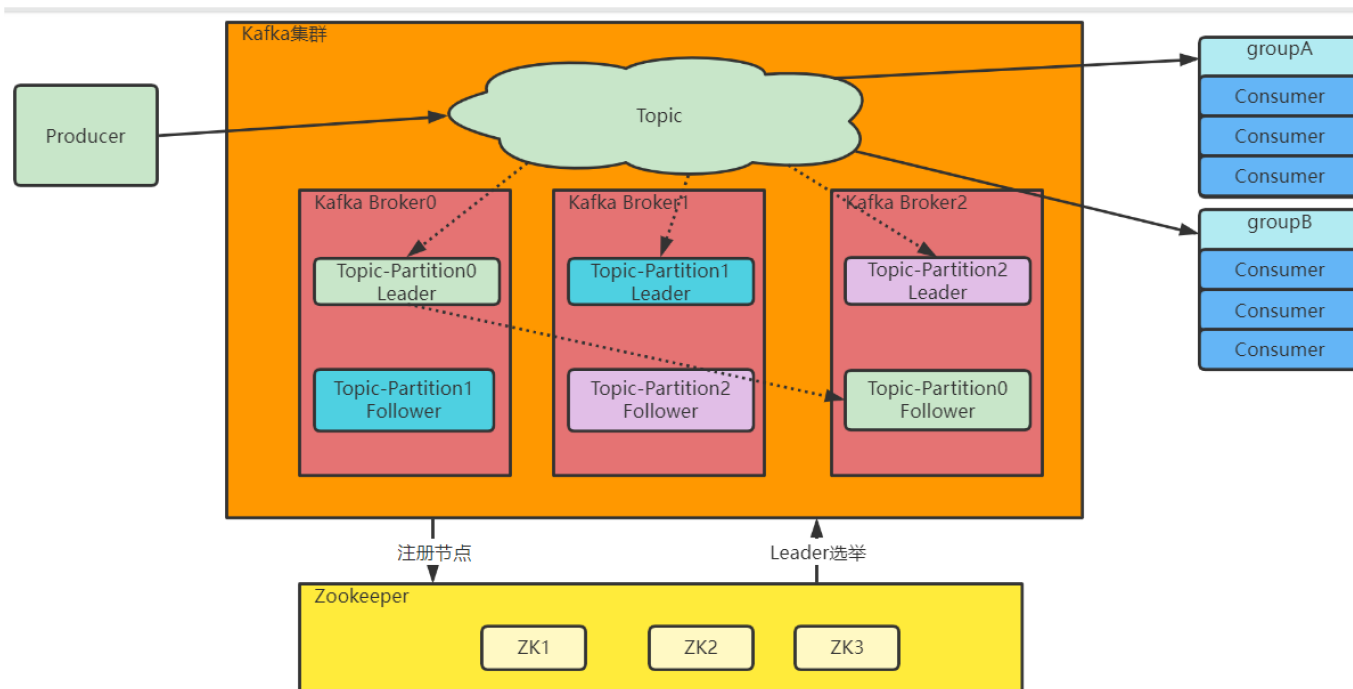
- 客户端Client：包括消息生产者 和 消息消费者。之前简单接触过。
- 消费者组：每个消费者可以指定一个所属的消费者组，相同消费者组的消费者共同构成一个逻辑消费者组。每一个消息会被多个感兴趣的消费者组消费，但是在每一个消费者组内部，一个消息只会被消费一次。
- 服务端Broker：一个Kafka服务器就是一个Broker。
- 话题Topic：这是一个逻辑概念，一个Topic被认为是业务含义相同的一组消息。客户端都通过绑定Topic来生产或者消费自己感兴趣的话题。
- 分区Partition：Topic只是一个逻辑概念，而Partition就是实际存储消息的组件。每个Partiton就是一个queue队列结构。所有消息以FIFO先进先出的顺序保存在这些Partition分区中。

三、理解Kafka的集群工作机制

对于Kafka这样一个追求消息吞吐量的产品来说，集群基本上是必备的。接下来，我们就动手搭建一个Kafka集群，并来理解一下Kafka集群的工作机制。

1、搭建Kafka集群

Kafka的集群架构大体是这样的：



为什么要用集群？

单机服务下，Kafka已经具备了非常高的性能。TPS能够达到百万级别。但是，在实际工作中使用时，单机搭建的Kafka会有很大的局限性。

一方面：消息太多，需要分开保存。Kafka是面向海量消息设计的，一个Topic下的消息会非常多，单机服务很难存得下来。这些消息就需要分成不同的Partition，分布到多个不同的Broker上。这样每个Broker就只需要保存一部分数据。这些分区的个数就称为分区数。

另一方面：服务不稳定，数据容易丢失。单机服务下，如果服务崩溃，数据就丢失了。为了保证数据安全，就需要给每个Partition配置一个或多个备份，保证数据不丢失。Kafka的集群模式下，每个Partition都有一个或多个备份。Kafka会通过一个统一的Zookeeper集群作为选举中心，给每个Partition选举出一个主节点Leader，其他节点就是从节点Follower。主节点负责响应客户端的具体业务请求，并保存消息。而从节点则负责同步主节点的数据。当主节点发生故障时，Kafka会选举出一个从节点成为新的主节点。

最后：Kafka集群中的这些Broker信息，包括Partition的选举信息，都会保存在额外部署的Zookeeper集群当中，这样，kafka集群就不会因为某一些Broker服务崩溃而中断。

Kafka也提供了另外一种不需要Zookeeper的集群机制，Kraft集群。这种方式会在后面进行介绍。

准备实验环境

准备三台同样的CentOS服务器，预先安装好了JDK，并关闭防火墙

```
service firewalld stop
systemctl disable firewalld
```

分别配置机器名worker1,worker2,worker3

```
vi /etc/hosts

192.168.232.128 worker1
192.168.232.129 worker2
192.168.232.130 worker3
```

IP地址换成自己的

接下来我们就动手部署一个Kafka集群，来体验一下Kafka是如何面向海量数据进行横向扩展的。

我们先来部署一个基于Zookeeper的Kafka集群。其中，选举中心部分，Zookeeper是一种多数同意的选举机制，允许集群中少数节点出现故障。因此，在搭建集群时，通常都是采用3，5，7这样的奇数节点，这样可以最大化集群的高可用特性。在后续的实验过程中，我们会在三台服务器上都部署Zookeeper和Kafka。

1、部署Zookeeper集群

这里采用之前单独下载的Zookeeper来部署集群。Zookeeper是一种多数同意的选举机制，允许集群中少数节点出现故障。因此，在搭建集群时，通常采用奇数节点，这样可以最大化集群的高可用特性。在后续的实现过程中，我们会在三台服务器上都部署Zookeeper。

先将下载下来的Zookeeper解压到/app/zookeeper目录。

然后进入conf目录，修改配置文件。在conf目录中，提供了一个zoo_sample.cfg文件，这是一个示例文件。我们只需要将这个文件复制一份zoo.cfg(cp zoo_sample.cfg zoo.cfg)，修改下其中的关键配置就可以了。其中比较关键的修改参数如下：

```
#Zookeeper的本地数据目录，默认是/tmp/zookeeper。这是Linux的临时目录，随时会被删掉。
dataDir=/app/zookeeper/data
#Zookeeper的服务端口
clientPort=2181
#集群节点配置
server.1=192.168.232.128:2888:3888
server.2=192.168.232.129:2888:3888
server.3=192.168.232.130:2888:3888
```

其中，clientPort 2181是对客户端开放的服务端口。

集群配置部分，server.x这个x就是节点在集群中的myid。后面的2888端口是集群内部数据传输使用的端口。3888是集群内部进行选举使用的端口。

接下来将整个Zookeeper的应用目录分发到另外两台机器上。

然后需要构建对应的myid文件

```
#进入配置的data目录
cd /app/zookeeper/data
# 生成myid文件
echo 1 > myid
```

这个myid文件的内容就是在zoo.cfg中配置的对应的server.id

接下来可以在三台机器上都启动Zookeeper服务了。

```
bin/zkServer.sh --config conf start
```

启动完成后，使用jps指令可以看到一个QuorumPeerMain进程就表示服务启动成功。

三台机器都启动完成后，可以查看下集群状态。

```
[root@hadoop02 zookeeper-3.5.8]# bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /app/zookeeper/zookeeper-3.5.8/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost.
Mode: leader
```

这其中Mode 为leader就是主节点，follower就是从节点。

2、部署Kafka集群

kafka服务并不需要进行选举，因此也没有奇数台服务的建议。

部署Kafka的方式跟部署Zookeeper差不多，就是解压、配置、启服务三板斧。

首先将Kafka解压到/app/kafka目录下。

然后进入config目录，修改server.properties。这个配置文件里面的配置项非常多，下面列出几个要重点关注的配置。

```
#broker 的全局唯一编号，不能重复，只能是数字。
broker.id=0
#服务监听地址
listeners=PLAINTEXT://worker1:9092
#数据文件地址。同样默认是给的/tmp目录。
log.dirs=/app/kafka/logs
#默认的每个Topic的分区数
num.partitions=1
#zookeeper的服务地址
zookeeper.connect=worker1:2181,worker2:2181,worker3:2181
#可以选择指定zookeeper上的基础节点。
#zookeeper.connect=worker1:2181,worker2:2181,worker3:2181/kafka
```

broker.id需要每个服务器上不一样，分发到其他服务器上时，要注意修改一下。

多个Kafka服务注册到同一个zookeeper集群上的节点，会自动组成集群。

配置文件中的注释非常细致，可以关注一下。下面是server.properties文件中比较重要的核心配置

Property	Default	Description
broker.id	0	broker的“名字”，你可以选择任意你喜欢的数字作为id，只要id是唯每个broker都可以用一个唯一的非负整数id进行标识；这个id可以作为一的即可。
log.dirs	/tmp/kafka-logs	kafka存放数据的路径。这个路径并不是唯一的，可以是多个，路径之间只需要使用逗号分隔即可；每当创建新partition时，都会选择在包含最少partitions的路径下进行。
listeners	PLAINTEXT://127.0.0.1:9092	server接受客户端连接的端口，ip配置kafka本机ip即可
zookeeper.connect	localhost:2181	zookeeper连接地址。hostname:port。如果是Zookeeper集群，用逗号连接。
log.retention.hours	168	每个日志文件删除之前保存的时间。
num.partitions	1	创建topic的默认分区数
default.replication.factor	1	自动创建topic的默认副本数量
min.insync.replicas	1	当producer设置acks为-1时，min.insync.replicas指定replicas的最小数目（必须确认每一个replica的写数据都是成功的），如果这个数目没有达到，producer发送消息会产生异常
delete.topic.enable	false	是否允许删除主题

接下来就可以启动kafka服务了。启动服务时需要指定配置文件。

```
bin/kafka-server-start.sh -daemon config/server.properties
```

-daemon表示后台启动kafka服务，这样就不会占用当前命令窗口。

通过jps指令可以查看Kafka的进程

2、理解服务端的Topic、Partition和Broker

接下来可以对比一下之前的单机服务，快速理解Kafka的集群当中核心的Topic、Partition、Broker。

```
# 创建一个分布式的Topic
[oper@worker1 bin]$ ./kafka-topics.sh --bootstrap-server worker1:9092 --create --
replication-factor 2 --partitions 4 --topic disTopic
Created topic disTopic.
# 列出所有的Topic
[oper@worker1 bin]$ ./kafka-topics.sh --bootstrap-server worker1:9092 --list
__consumer_offsets
disTopic
# 查看列表情况
[oper@worker1 bin]$ ./kafka-topics.sh --bootstrap-server worker1:9092 --describe --topic
disTopic
Topic: disTopic TopicId: vX4ohhIER6aDpDZgTy10tQ PartitionCount: 4      ReplicationFactor:
2   Configs: segment.bytes=1073741824
    Topic: disTopic Partition: 0      Leader: 2      Replicas: 2,1   Isr: 2,1
    Topic: disTopic Partition: 1      Leader: 1      Replicas: 1,0   Isr: 1,0
    Topic: disTopic Partition: 2      Leader: 0      Replicas: 0,2   Isr: 0,2
    Topic: disTopic Partition: 3      Leader: 2      Replicas: 2,0   Isr: 2,0
```

从这里可以看到，

1、--create创建集群，可以指定一些补充的参数。大部分的参数都可以在配置文件中指定默认值。

- partitons参数表示分区数，这个Topic下的消息会分别存入这些不同的分区中。示例中创建的disTopic，指定了四个分区，也就是说这个Topic下的消息会划分为四个部分。
- replication-factor表示每个分区有几个备份。示例中创建的disTopic，指定了每个partition有两个备份。

2、--describe查看Topic信息。

- partiton参数列出了四个partition，后面带有分区编号，用来标识这些分区。
- Leader表示这一组partiton中的Leader节点是哪一个。这个Leader节点就是负责响应客户端请求的主节点。从这里可以看到，Kafka中的每一个Partition都会分配Leader，也就是说每个Partition都有不同的节点来负责响应客户端的请求。这样就可以将客户端的请求做到尽可能的分散。
- Replicas参数表示这个partition的多个备份是分配在哪些Broker上的。也称为AR。这里的0,1,2就对应配置集群时指定的broker.id。但是，Replicas列出的只是一个逻辑上的分配情况，并不关心数据实际是不是按照这个分配。甚至有些节点服务挂了之后，Replicas中也依然会列出节点的ID。
- ISR参数表示partition的实际分配情况。他是AR的一个子集，只列出那些当前还存活，能够正常同步数据的那些Broker节点。

接下来，我们还可以查看Topic下的Partition分布情况。在Broker上，与消息，联系最为紧密的，其实就是Partition了。之前在配置Kafka集群时，指定了一个log.dirs属性，指向了一个服务器上的日志目录。进入这个目录，就能看到每个Broker的实际数据承载情况。


```
[oper@worker1 kafka-logs]$ ./app/kafka/kafka_2.13-3.2.0/bin/kafka-topics.sh --bootstrap-server worker1:9092 --describe --topic disTopic
Topic: disTopic TopicId: vX4ohhIER6aDpDZgTy10tQ PartitionCount: 4      ReplicationFactor: 2   Configs: segment.bytes=1073741824
    Topic: disTopic Partition: 0      Leader: 2      Replicas: 2,1   Isr: 2,1
    Topic: disTopic Partition: 1      Leader: 1      Replicas: 1,0   Isr: 1,0
    Topic: disTopic Partition: 2      Leader: 0      Replicas: 0,2   Isr: 0,2
    Topic: disTopic Partition: 3      Leader: 2      Replicas: 2,0   Isr: 2,0

[oper@worker1 kafka-logs]$ pwd
/app/kafka/kafka-logs
[oper@worker1 kafka-logs]$ ls
cleaner-offset-checkpoint  __consumer_offsets-18  __consumer_offsets-21  __consumer_offsets-24  __consumer_offsets-27  __consumer_offsets-30  __consumer_offsets-33  __consumer_offsets-36  __consumer_offsets-39  __consumer_offsets-42  __consumer_offsets-45  __consumer_offsets-48  __consumer_offsets-6  __consumer_offsets-9  disTopic-1  disTopic-2  disTopic-3  log-start-offset-checkpoint  meta.properties  recovery-point-offset-checkpoint  replication-offset-checkpoint
```

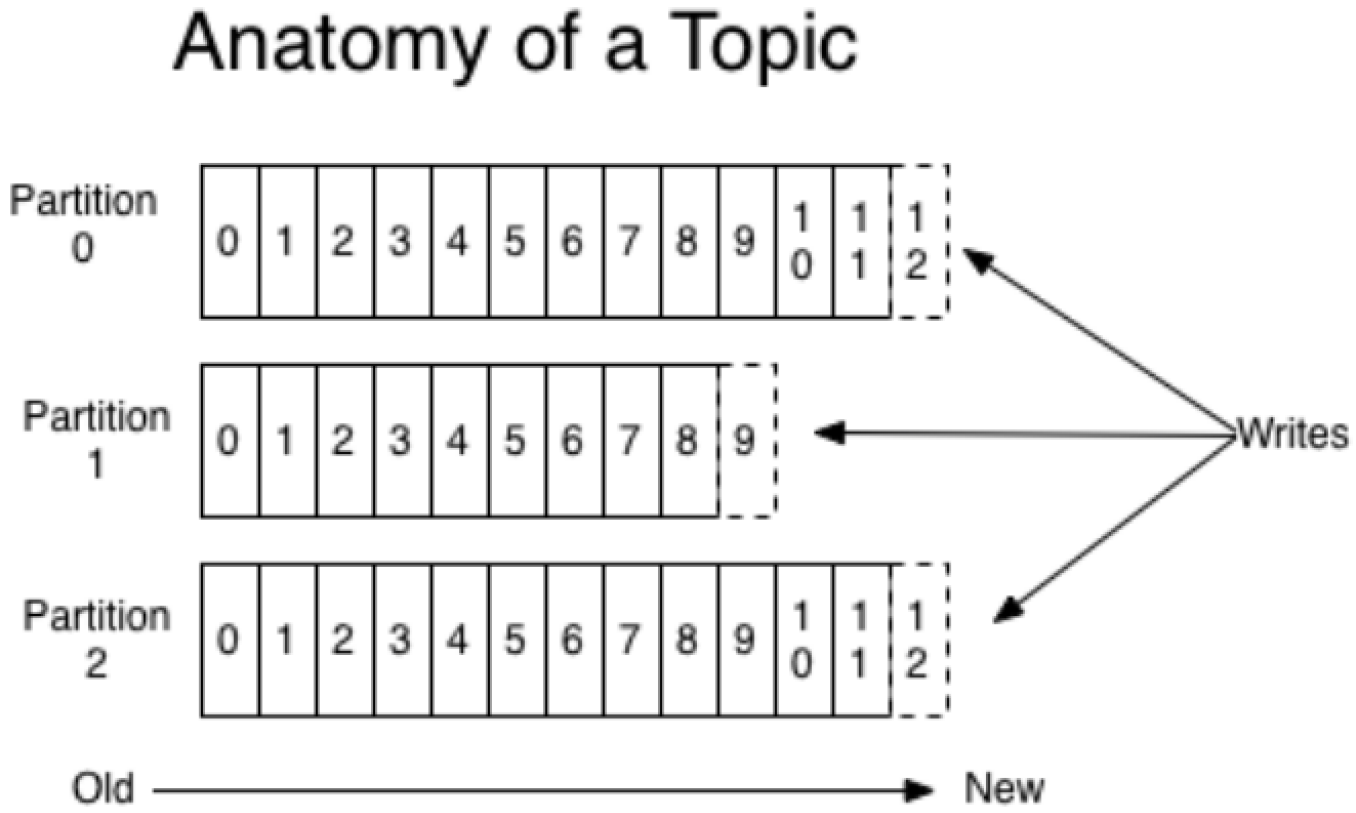
一个目录就对应当前Broker上的一个partition。分别保存partition的历史日志，也就是消息

当前服务器的broker.id=0，承载了1,2,3号partition

日志目录

从这里可以看到，Broker上的一个Partition对应了日志目录中的一个目录。而这个Partition上的所有消息，就保存在这个对应的目录当中。

从整个过程可以看到，Kafka当中，Topic是一个数据集的逻辑单元。同一个Topic下的数据，实际上是存储在Partition分区中的，Partition就是数据存储的物理单元。而Broker是Partition的物理载体，这些Partition分区会尽量均匀的分配到不同的Broker机器上。而之前接触到的offset，就是每个消息在partition上的偏移量。

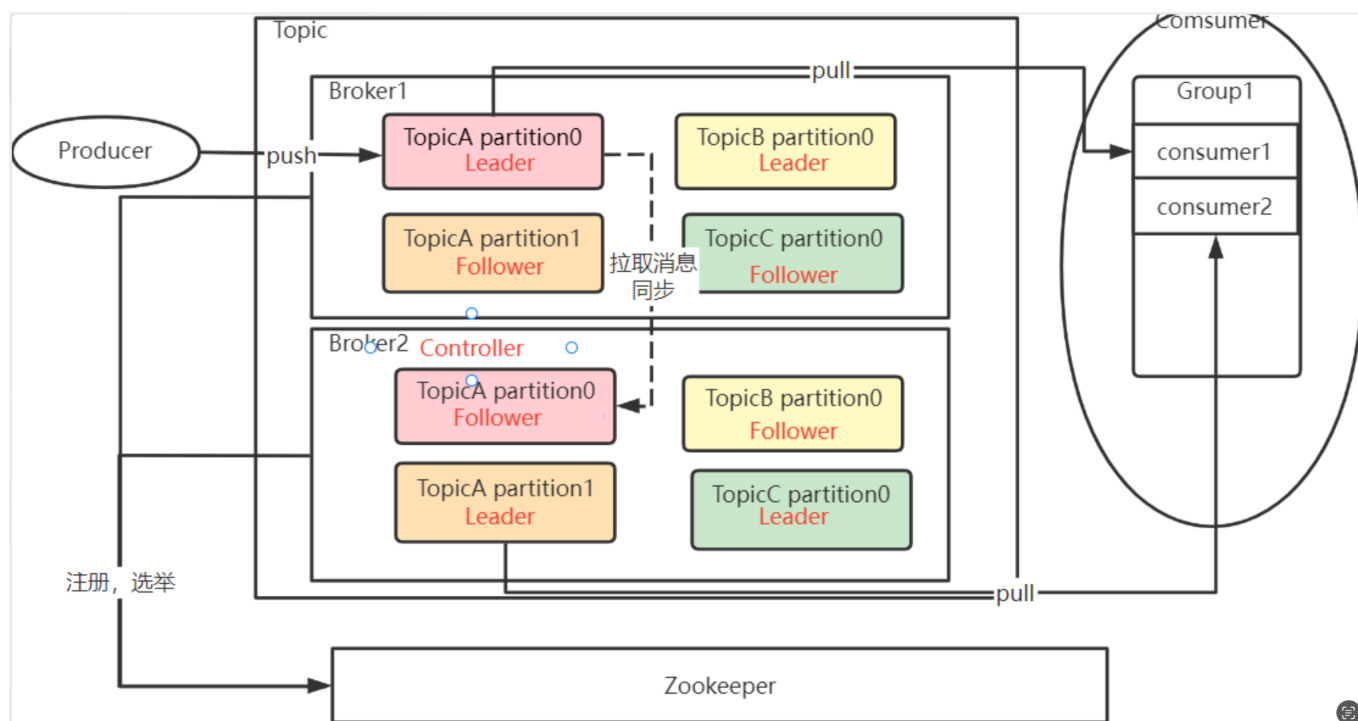


这样设计解决了什么问题？

- 1、Kafka设计需要支持海量的数据，而这样庞大的数据量，一个Broker是存不下的。那就拆分成多个Partition，每个Broker只存一部分数据。这样极大的扩展了集群的吞吐量。
- 2、每个Partition保留了一部分的消息副本，如果放到一个Broker上，就容易出现单点故障。所以就给每个Partition设计Follower节点，进行数据备份，从而保证数据安全。另外，多备份的Partition设计也提高了读取消息时的并发度。
- 3、在同一个Topic的多个Partition中，会产生一个Partition作为Leader。这个Leader Partition会负责响应客户端的请求，并将数据往其他Partition分发。

四、章节总结：Kafka集群的消息流转模型

经过上面的实验，我们接触到了很多Kafka中的概念。将这些基础概念整合起来，就形成了Kafka集群的整体结构。这次我们先把这个整体结构梳理清楚，后续再一点点去了解其中的细节。



- 1、Topic是一个逻辑概念，Producer和Consumer通过Topic进行业务沟通。
- 2、Topic并不存储数据，Topic下的数据分为多组Partition，尽量平均的分散到各个Broker上。每组Partition包含Topic下一部分的消息。每组Partition包含一个Leader Partition以及若干个Follower Partition进行备份，每组Partition的个数称为备份因子 replica factor。
- 3、Producer将消息发送到对应的Partition上，然后Consumer通过Partition上的Offset偏移量，记录自己所属消费者组Group在当前Partition上消费消息的进度。
- 4、Producer发送给一个Topic的消息，会由Kafka推送给所有订阅了这个Topic的消费者组进行处理。但是在每个消费者组内部，只会有一个消费者实例处理这一条消息。
- 5、最后，Kafka的Broker通过Zookeeper组成集群。然后在这些Broker中，需要选举产生一个担任Controller角色的Broker。这个Controller的主要任务就是负责Topic的分配以及后续管理工作。在我们实验的集群中，这个Controller实际上是通过ZooKeeper产生的。