

分布式消息队列 Kafka

范颖捷 | 2018年7月

目录 >

CONTENTS

- 1 Kafka简介
- 2 Kafka原理
- 3 Kafka使用



1

chapter

Kafka简介

- ✓ 什么是Kafka
- ✓ 应用场景

➤ 概念

- 基于发布/订阅的分布式消息系统
- 由Linkedin开发，用Scala语言编写

➤ 特性

- 消息持久化：采用时间复杂度 $O(1)$ 的磁盘存储结构，即使TB级以上数据也能保证常数时间的访问速度
- 高吞吐：即使在廉价的商用机器上，也能达到单机每秒10万条消息的传输
- 高容错：多分区多副本
- 易扩展：新增机器，集群无需停机，自动感知
- 同时支持离线、实时数据处理

➤ 异步通信

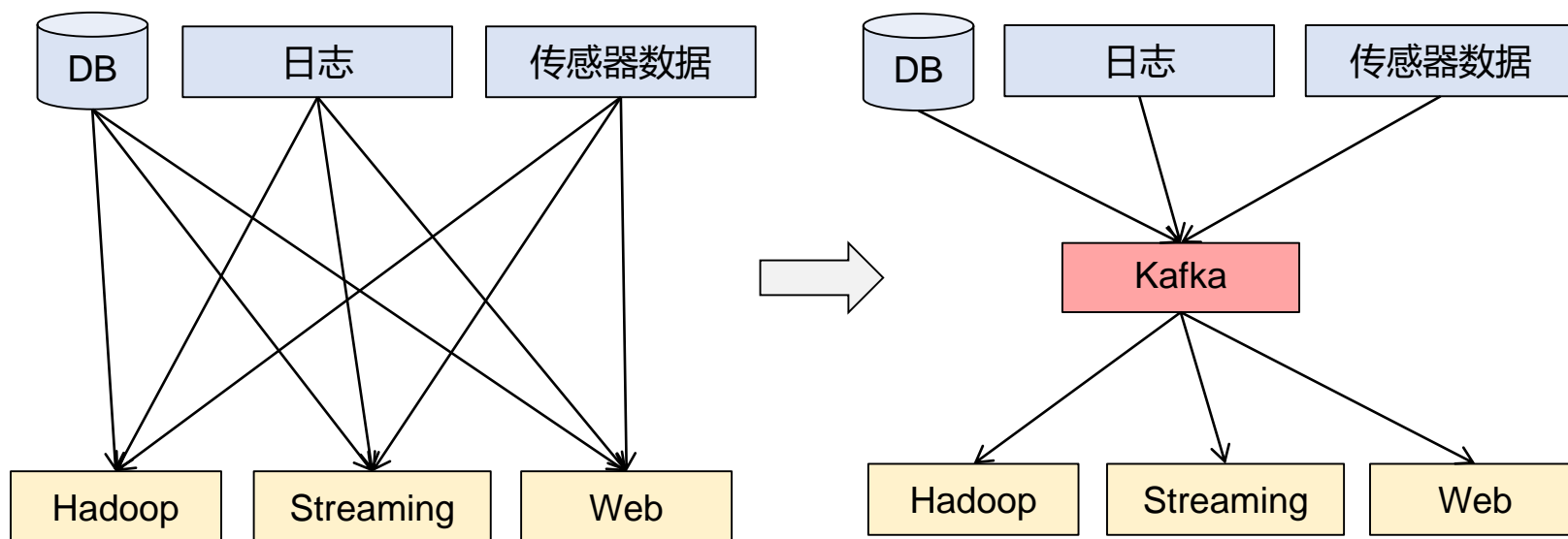
- 将消息放入Kafka，可以不立即处理


➤ 应用解耦

- 在应用处理过程中插入一个隐含的、基于数据的接口层

➤ 峰值处理

- 使关键应用能够顶住访问峰值，不会因超出负荷而崩溃





2

chapter

Kafka原理

- ✓ 基本概念
- ✓ 工作机制
- ✓ 数据存储
- ✓ 高可用

➤ Broker (代理)

- Kafka的一个实例或节点，一个或多个Broker组成一个Kafka集群

➤ Topic (主题)

- Topic是Kafka中同一类数据的集合，相当于数据库中的表
- Producer将同一类数据写入同一个Topic，Consumer从同一个Topic中读取同类数据
- Topic是逻辑概念，用户只需指定Topic就可以生产或消费数据，不必关心数据存于何处

➤ Partition (分区)

- 分区是一个有序的、不可修改的消息队列，分区内消息有序存储
- 一个Topic可分为多个分区，相当于把一个数据集分成多份，分别存储不同的分区中
- Partition是物理概念，每个分区对应一个文件夹，其中存储分区的数据和索引文件

➤ Replication (副本)

- 一个分区可以设置多个副本，副本存储在不同的Broker中

➤ Producer (消息生产者)

- 向Broker发布消息的客户端

➤ Consumer (消息消费者)

- 从Broker消费消息的客户端

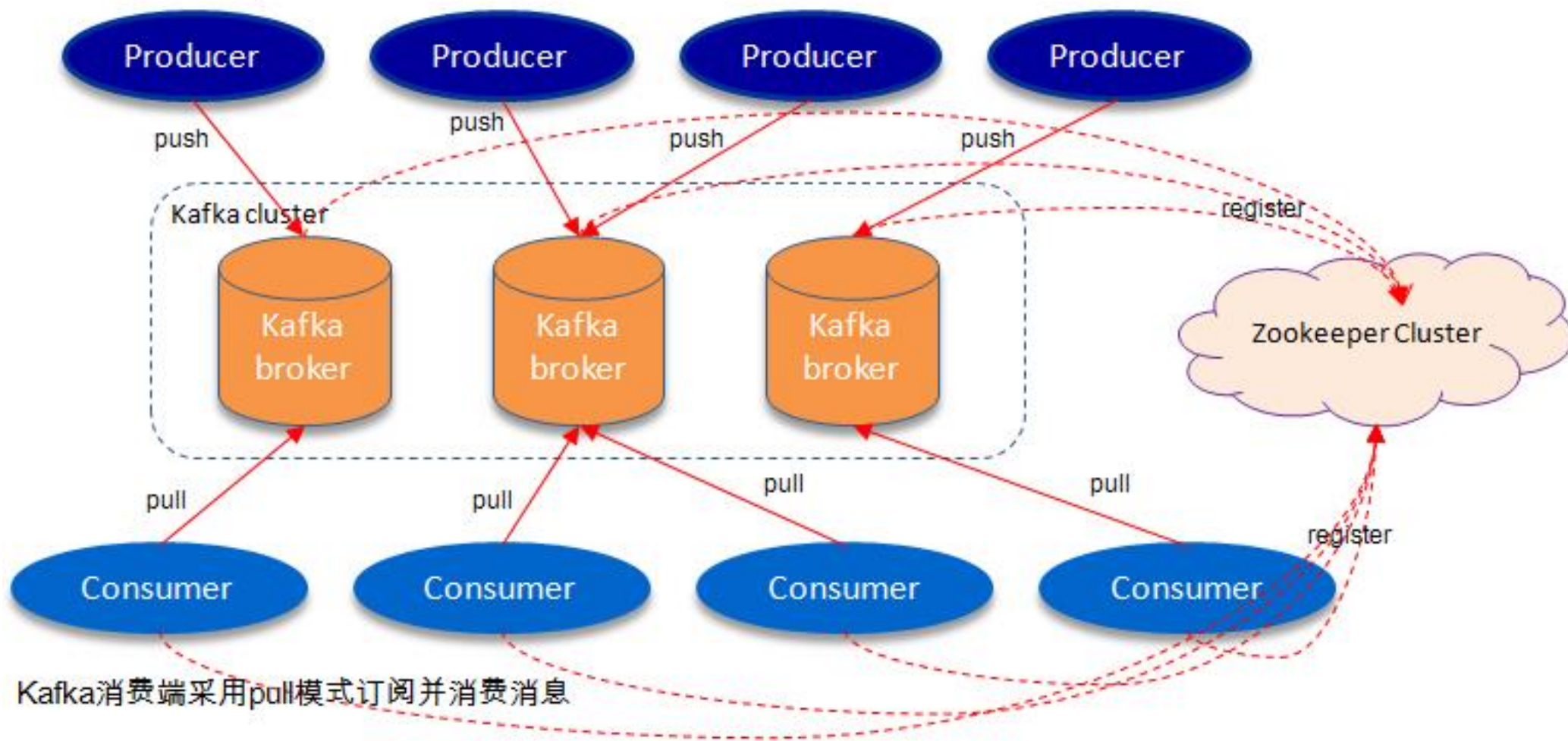
➤ Consumer Group (CG , 消费者组)

- 每个Consumer都隶属于一个特定的CG
- 一条消息可以发送给多个不同的CG，但一个CG中只能有一个Consumer读取该消息

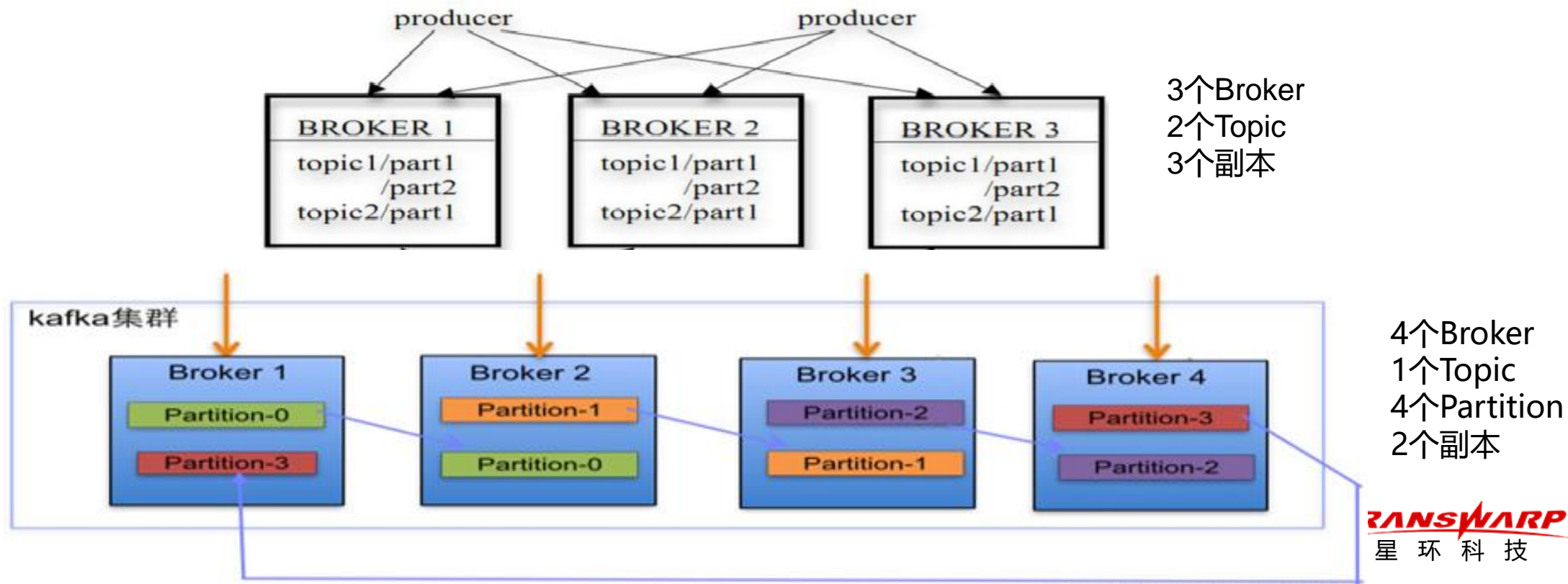
➤ Zookeeper

- Kafka将元数据保存在Zookeeper中
- 负责Kafka集群管理，包括配置管理、动态扩展、Broker负载均衡、Leader选举，以及Consumer Group变化时的Rebalance等

Kafka发送端采用push模式将消息发送到broker

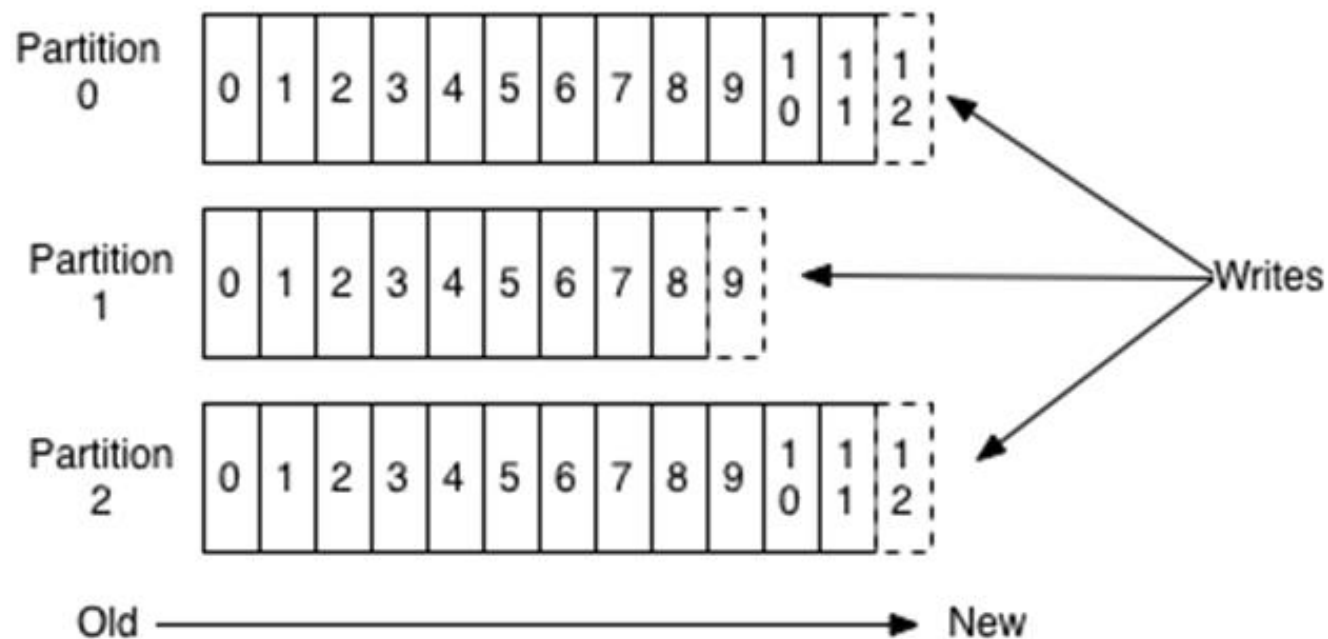


- 消息在Broker中按Topic（主题）进行分类，相当于为每个消息打个标签
- 一个Topic可划分为多个Partition（分区）
- 每个Partition可以有多个Replication（副本）
- 消息存储在Broker的某一Topic的某一Partition中，同时存在多个副本

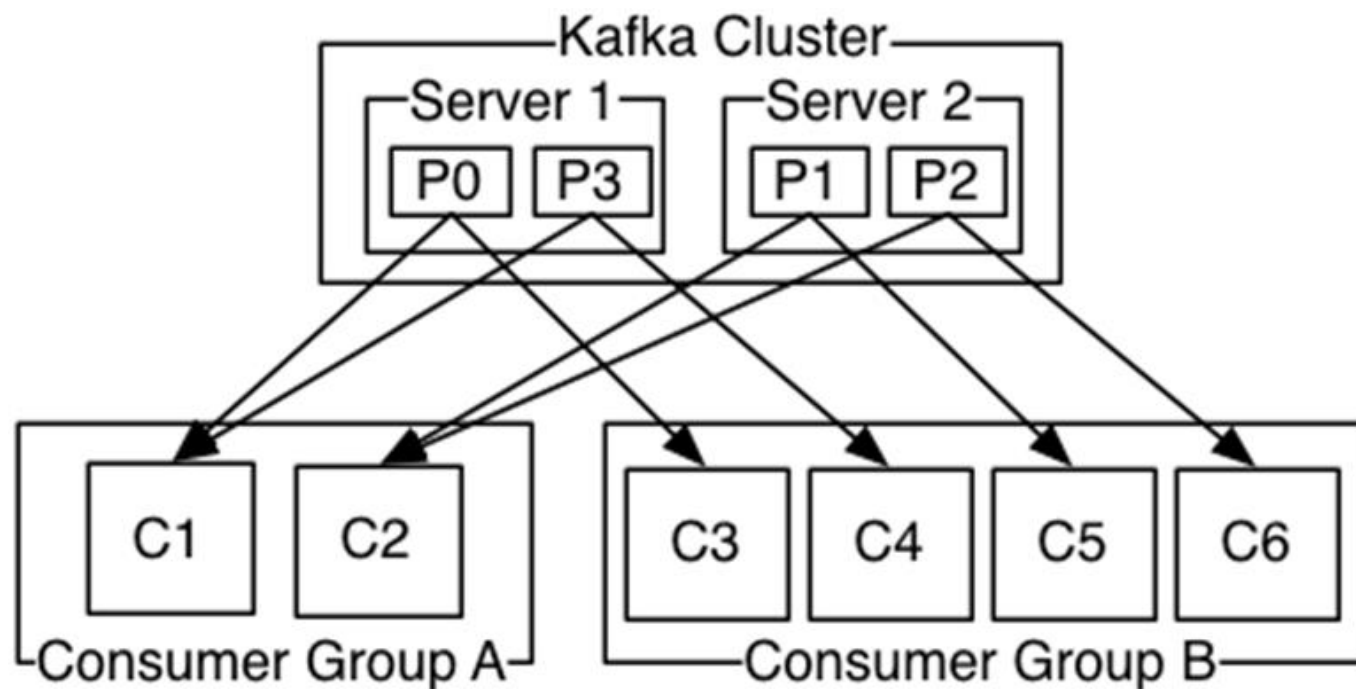


- Partition是一个FIFO队列，写入消息采用在队列尾部追加的方式，消费消息采用在队列头部顺序读取的方式
- 一个Topic可分为多个Partition，仅保证同一分区内消息有序存储，不保证Topic整体（多个分区之间）有序

Anatomy of a Topic



- 为了加快读取速度，多个Consumer可划分为一个组（Consumer Group, CG），并行消费同一个Topic
- 一个Topic可以被多个CG订阅，CG之间是平等的，即一个消息可同时被多个CG消费
- 一个CG中可以有多Consumer，CG中的Consumer之间是竞争关系，即一个消息在一个CG中只能被一个Consumer消费



- 每个Partition副本都是一个目录，目录中包含若干Segment文件

```
drwxrwxr-x. 2 hadoop hadoop 4096 1月 4 09:57 topictest1-0
drwxrwxr-x. 2 hadoop hadoop 4096 1月 4 09:57 topictest1-1
drwxrwxr-x. 2 hadoop hadoop 4096 1月 4 09:57 topictest1-2
drwxrwxr-x. 2 hadoop hadoop 4096 1月 4 09:58 topictest1-3
drwxrwxr-x. 2 hadoop hadoop 4096 1月 4 09:58 topictest1-4
```

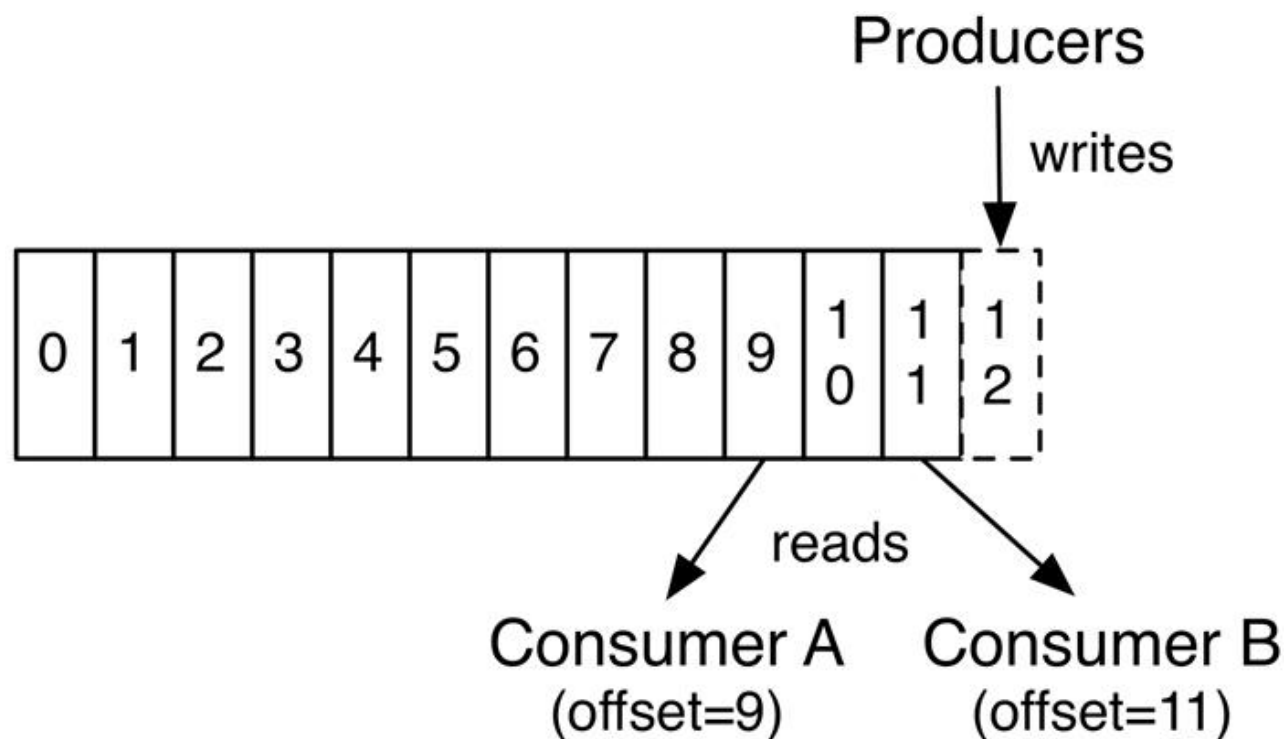
- Segment (段文件)

- Segment文件是Kafka的最小数据存储单元，一个Partition包含多个Segment文件
- Segment文件由以Message在Partition中的起始偏移量命名的数据文件 (*.log) 和索引文件 (*.index、*.timeindex) 组成

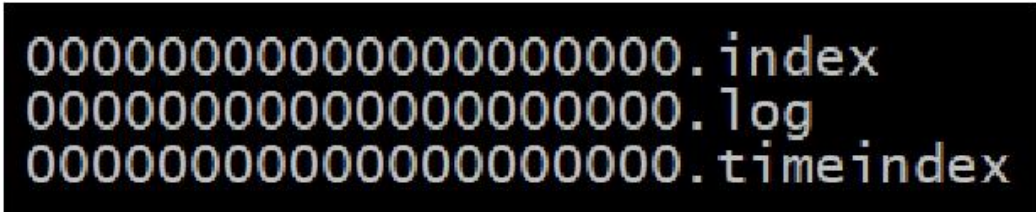
```
000000000000000000000000.log
000000000000000000000000.index
000000000000000000000000.timeindex
00000000000000000000170410.log
00000000000000000000170410.index
00000000000000000000170410.timeindex
```


➤ Offset (偏移量)

- Offset是用于定位分区中消息的顺序编号
- Offset用于在分区中唯一标识消息
- 使用Zookeeper维护Offset

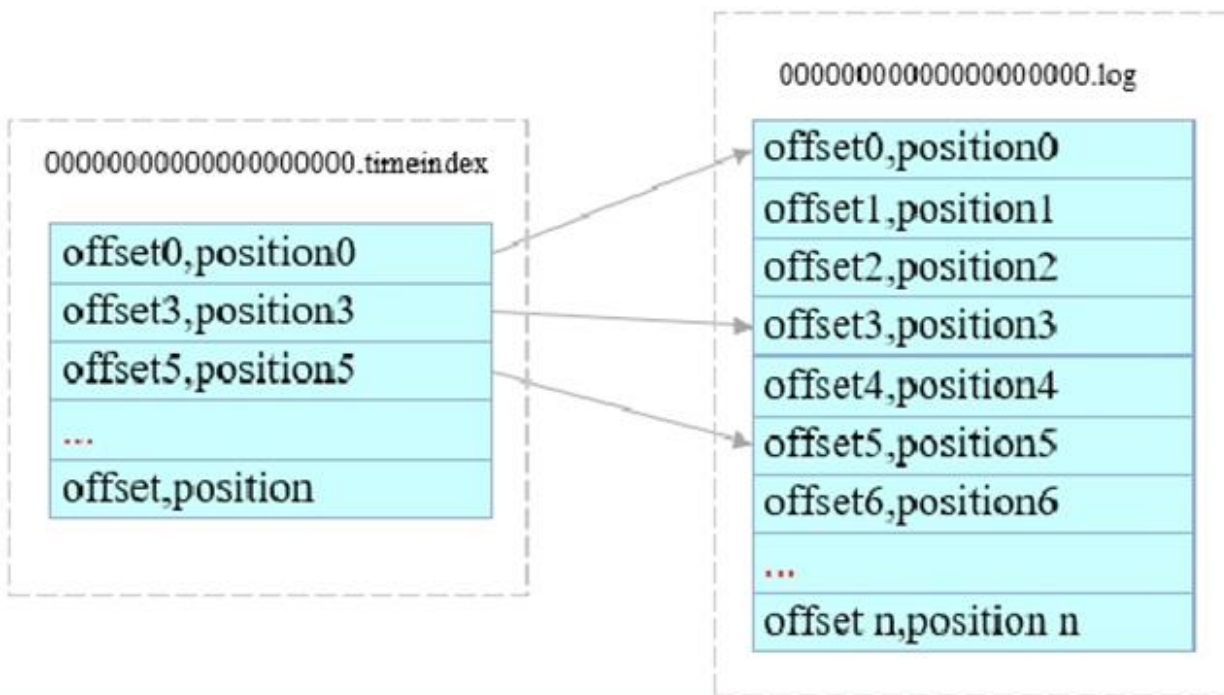


► Kafka索引

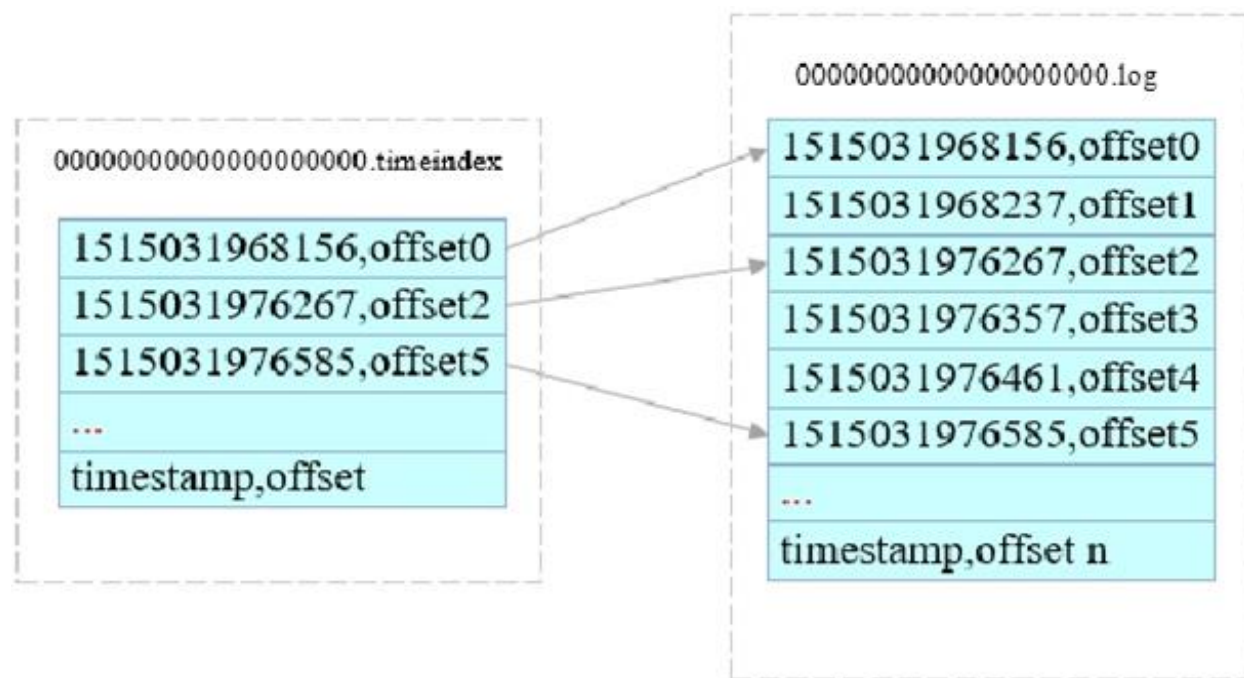
- 为了提高消息写入和查询速度，为每个Partition创建索引，索引文件存储在Partition文件夹下
 - 偏移量索引
 - 文件以offset偏移量为名称，以index为后缀
 - 索引内容格式：offset,position
 - 采用稀疏存储方式
 - 时间戳索引
 - 文件以timeindex为后缀
 - 索引内容格式：timestamp,offset
 - 采用稀疏存储方式
- 

```
00000000000000000000000000.index  
00000000000000000000000000.log  
00000000000000000000000000.timeindex
```

➤ Kafka索引



偏移量索引



时间戳索引

➤ 多分区多副本

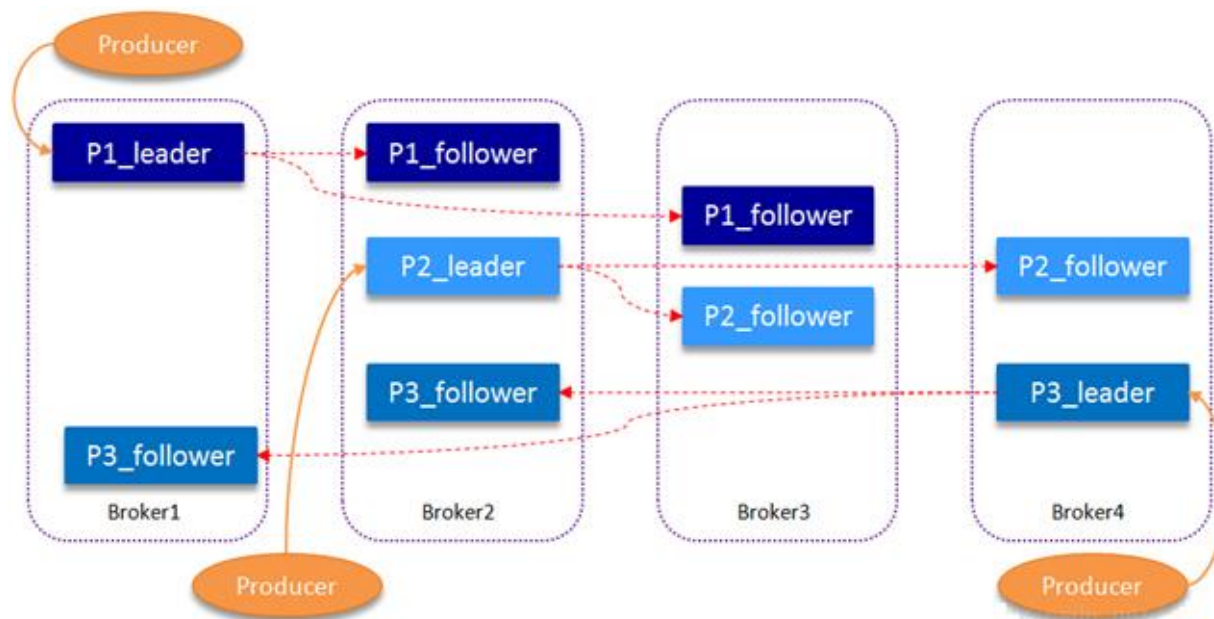
- Kafka早期版本没有Replication概念，一旦某个Broker宕机，其上的分区数据就可能丢失
- 一个Topic可以有多个分区，每个分区可以有多个副本，副本存储在不同的Broker中
- 从一个分区的多个副本中选举一个Partition Leader，由Leader负责读写，其他副本作为Follower从Leader同步消息


➤ Kafka Controller Leader选举

- 每个Broker启动时都会创建一个Kafka Controller进程
- 通过Zookeeper，从Kafka集群中选举出一个Broker作为Kafka Controller Leader
- Kafka Controller Leader负责管理Kafka集群的分区和副本状态，避免分区副本直接在Zookeeper上注册Watcher和竞争创建临时Znode，导致Zookeeper集群负载过重

➤ Kafka Partition Leader选举

- Kafka Controller Leader负责Partition Leader的选举
- ISR列表（In Sync Replica）
 - ISR是Zookeeper中的候选副本同步列表，负责保存候选副本（Partition Follower）的状态信息
 - Partition Leader负责跟踪和维护ISR
 - Partition Follower定期从Leader同步数据，若Follower心跳超时或消息落后太多，将被移除出ISR
- Partition Leader挂掉后，Kafka Controller Leader从ISR中选择一个Follower作为新的Leader





3 chapter

Kafka使用

- ✓ Kafka安装
- ✓ Kafka命令

➤ 安装Kafka服务

- 第一步：登录Transwarp Manager → 首页 → +服务
- 第二步：选择安装“Kafka”，并选择版本
- 第三步：点击“下一步”，“分配角色 → 配置服务 → 配置安全”各步骤按实际情况设置即可（测试环境建议按照默认配置）
- 第四步：点击“确认”，完成安装

➤ 安装Kafka客户端

- 安装TDH Client（集成Kafka客户端）
 - TDH Client下载：Transwarp Manager → 管理 → 下载客户端（tdh-client.tar），并解压
 - TDH Client初始化：执行TDH Client目录下的init.sh脚本

➤ 创建Topic

- 可指定Partition、Replication数量
- 建议--replication-factor > 1，否则Broker宕掉后将无法写入消息

```
# ./kafka-topics.sh
--create --topic demo
--zookeeper tdh-203:2181, tdh-204:2181, tdh-205:2181
--partitions 3 --replication-factor 3
```

➤ 查看Topic详情

```
# ./kafka-topics.sh
--describe --topic demo
--zookeeper tdh-203:2181
```

➤ 查看Topic列表

```
# ./kafka-topics.sh  
--list  
--zookeeper tdh-203:2181, tdh-204:2181, tdh-205:2181
```

➤ 创建Producer

```
# ./kafka-console-producer.sh  
--broker-list tdh-203:9092, tdh-204:9092, tdh-205:9092  
--topic demo
```

➤ 创建Consumer

```
# ./kafka-console-consumer.sh  
--bootstrap-server tdh-203:9092,tdh-204:9092,tdh-205:9092  
--topic demo
```

➤ 重置Offset

- 含义：当数据消费出现异常时，将offset设置为某个值或最小值，从分区的Offset开始重新读取消息

- 重置前

– logSize为分区消息总数，Lag为未消费消息数，Offset为已消费消息数

```
./kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zookeeper tdh-203:2181 -g group1 --topic demo1
```

Group	Topic	Pid	Offset	logSize	Lag	Owner
group1	topic1	0	3	3	0	none

- 重置

```
./kafka-run-class.sh kafka.tools.UpdateOffsetsInZK earliest ../config/consumer.properties demo
```

- 重置后

```
./kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zookeeper tdh-203:2181 -g group1 --topic demo1
```

Group	Topic	Pid	Offset	logSize	Lag	Owner
group1	topic1	0	0	3	3	none



Q&A

TRANSWARP
星环科技