愿景："让编程不再难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第一章 玩转分布式流处理平台 Kafka 小白到专家之路课程介绍

第1集 玩转分布式流处理平台 Kafka 小白到专家之路介绍

简介：讲解分布式流处理平台 Kafka 适合人员和学后水平

- 课程介绍

全新录制的视频，从0到1讲解高性能分布式流处理平台 Kafka ,掌握Kakfa核心概念和多种工作模式,点对点/发布订阅模型，生产者、消费者、Broker、Topic、Partition、副本leader/follower等；Linux服务器急速部署Zookeeper、Kafka，多种控制台操作指令，分区控制等，SpringBoot整合Kafka原生多个模块Admin/Producer/Consumer核心API，还有SpringKafka实战。

高级篇-Kafka存储流程和原理讲解LEO+HW+Offset

高级篇-生产者发送消息模型、分区策略和核心配置实战，自定义分区Key策略等

高级篇-消费者消费消息模型、分区策略和重Rebalance实战等

高级篇-Broker数据文件存储模型-ACK和副本可靠性原理分析+ISR模型讲解

高级篇-高可用搭建Zookeeper集群+Kafka集群+SpringBoot项目整合和故障演练

高级篇-Kafka高性能原理分析ZeroCopy+多案例事务消息实战+大数据技术栈路线

超多Kafka架构+设计思想+底层原理+互联网大厂面试题等

- 课程50多集，每集10~20分钟不等，总时长12小时左右
- 一天学习4小时，操作3小时，三天足够掌握 基础+实战+ 高可用+架构原理+面试

- 为什么要学习Kafka分布式流处理平台
  - 多数互联网公司里面用的技术栈，可以承载海量消息处理
  - 三大MQ中间件之一，在多数互联网公司中，Kafka占有率很高，大数据处理领域领头羊
  - 中大型互联网公司-实时计算、离线计算、等基本都离不开Kafka
  - 可以作为公司内部培训技术分享必备知识，可靠性投递、消费、高可用集群等

- 学后水平
  - 从0到1讲解高性能分布式流处理平台 Kafka核心知识+项目实战
  - 掌握Kakfa多种工作模式,点对点/发布订阅模型和应用场景
  - 掌握核心概念 生产者、消费者 Broker/Topic/Partition/leader/follower等；
  - Linux服务器急速部署Zookeeper、Kafka，多种控制台操作

指令，分区控制等
    - SpringBoot整合Kafka原生多个模块Admin/Producer/Consumer核心API+SpringKafka实战。
    - 高级篇-Kafka存储流程和原理讲解LEO+HW+Offset
    - 高级篇-生产者发送消息模型、分区策略和核心配置实战，自定义分区Key策略等
    - 高级篇-消费者消费消息模型、分区策略和重Rebalance实战等
    - 高级篇-Broker数据文件存储模型-ACK和副本可靠性原理分析+ISR模型讲解
    - 高级篇-高可用搭建Zookeeper集群+Kafka集群+SpringBoot项目整合和故障演练
    - 高级篇-Kafka高性能原理分析ZeroCopy+多案例事务消息实战+大数据技术栈路线
    - 超多Kafka架构+设计思想+底层原理+互联网大厂面试题等

- 适合人群
    - 高级后端工程师、高级前端/全栈工程师、运维工程师、CTO更新必备技术栈
    - 大数据工程师、数据分析工程师
    - 从传统软件公司过渡到互联网公司的人员

- 课程技术技术栈和环境说明

- Kafka版本：2.8.0
- Scala版本：2.13
- Zookeeper版本：3.7
- SpringBoot.2.5 + Maven + IDEA旗舰版 + JDK8 或 JDK11

- 学习形式
  - 视频讲解 + 文字笔记 + 原理分析 + 交互流程图
  - 配套源码 + 笔记 + 专属技术群答疑(我答疑，联系客服进群)
  - 进技术群还有加餐内容（多种高可用集群搭建+大厂面试题) , 大厂内推资源、文档大礼包等
  - 课程有配套的源码，在每章-每集的资料里面，如果没用到代码就不保存
  - 只要是我的课程和项目-我会一直维护下去，大家不用担心！！！

第2集 分布式流处理平台Kafka专题课程大纲速览

简介：分布式流处理平台Kafka专题课程大纲速览

- 课程效果演示

- 课程学前基础
  - Linux基础+SpringBoot基础
  - PS:不会上面的基础也没关系,这些基础都有课程，【联系我们客服】即可，且是刚录制的新版课程
- 目录大纲浏览
- 学习寄语
  - 保持谦虚好学、术业有专攻、在校的学生，有些知识掌握也比工作好几年掌握的人厉害 课程有配套的源码，在每章-每集的资料里面，如果没用到代码就不保存。
  - 目录介绍如果自己操作的情况和视频不一样，导入课程代码对

比验证基本就可以发现问题了 官方要求，务必保持版本一致，不然会遇到很多问题。

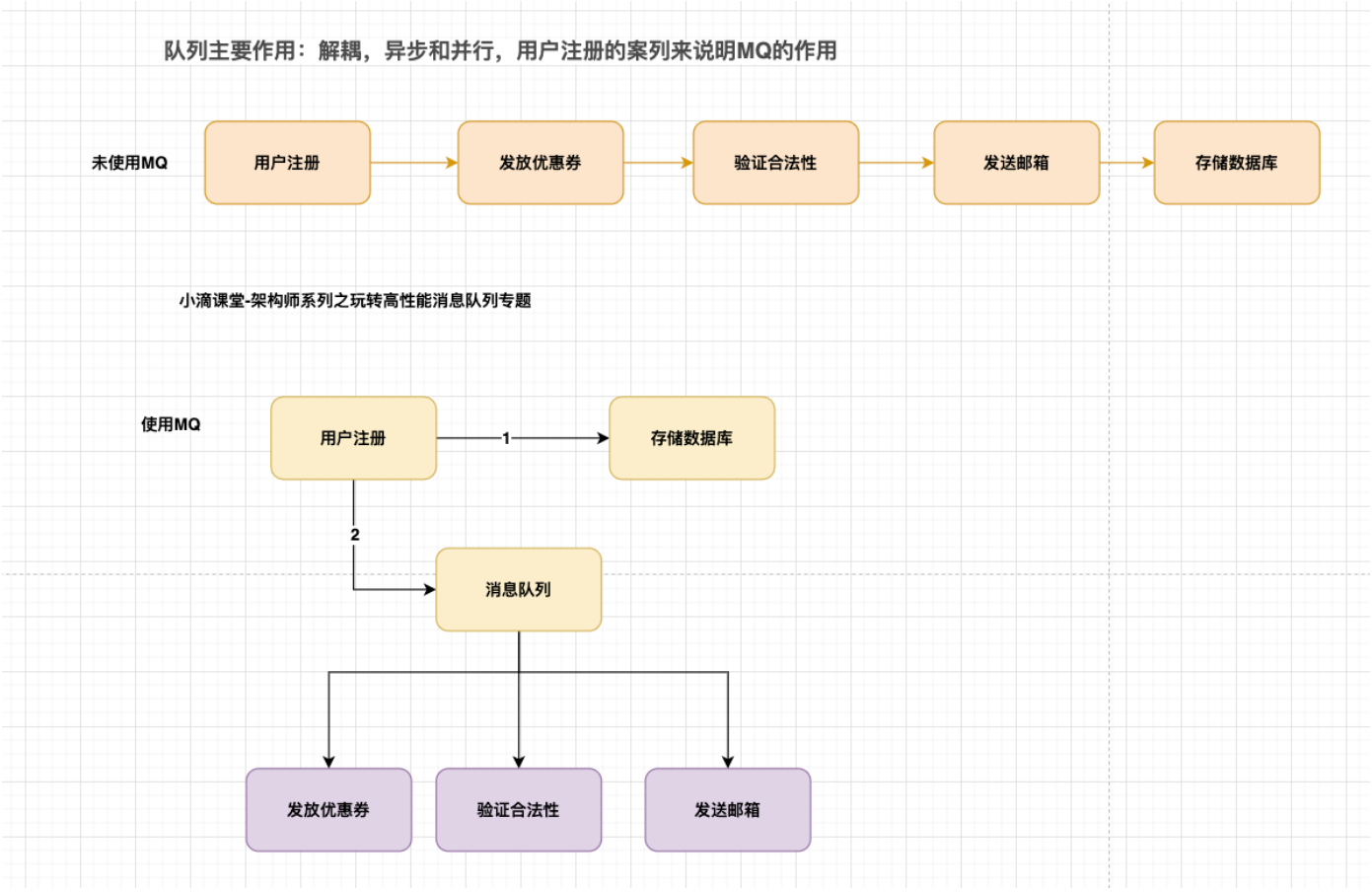
- 对于看不懂的视频，一律归因于写或讲的人太蠢，当然也不一定是事实，但这样的思考方式能让我避免陷入自我怀疑的负面情绪

第二章 大话MQ消息中间件+JMS+AMQP核心知识

第1集 什么是MQ消息中间件和应用场景

简介：介绍什么是MQ消息中间件和应用场景

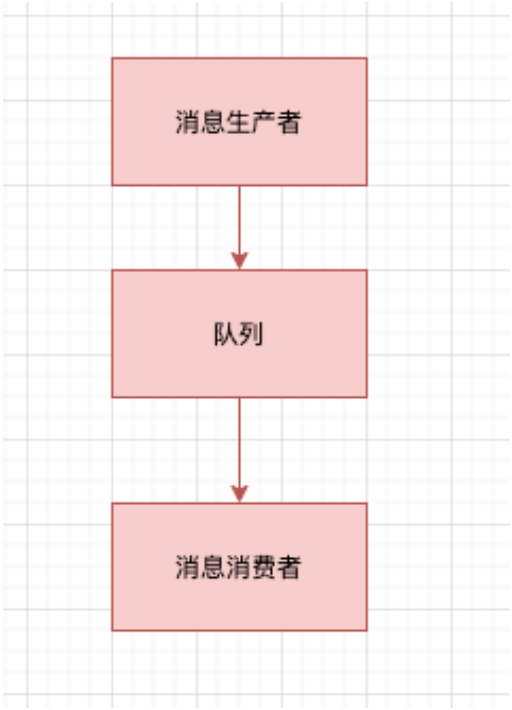
- 什么是MQ消息中间件
  - 全称MessageQueue，主要是用于程序和程序直接通信，异步+解耦
- 使用场景：
  - 核心应用
    - 解耦：订单系统-》物流系统
    - 异步：用户注册-》发送邮件，初始化信息
    - 削峰：秒杀、日志处理
  - 跨平台 、多语言
  - 分布式事务、最终一致性
  - RPC调用上下游对接，数据源变动->通知下属


队列主要作用：解耦，异步和并行，用户注册的案列来说明MQ的作用
未使用MQ
用户注册
发放优惠券
验证合法性
发送邮箱
存储数据库
小滴课堂-架构师系列之玩转高性能消息队列专题
使用MQ
用户注册
1
存储数据库
2
消息队列
发放优惠券
验证合法性
发送邮箱

第2集 JMS消息服务和和常见核心概念介绍

简介：讲解什么是AMQP和JMS消息服务

- 什么是JMS: Java消息服务（Java Message Service),Java平台中关于面向消息中间件的接口
  - JMS是一种与厂商无关的 API，用来访问消息收发系统消息，它类似于JDBC(Java Database Connectivity)。这里，JDBC 是可以用来访问许多不同关系数据库的 API
  - 是由Sun公司早期提出的消息标准，旨在为java应用提供统一的消息操作，包括create、send、receive
  - JMS是针对java的，那微软开发了NMS（.NET消息传递服务）



- 特性

- 面向Java平台的标准消息传递API
- 在Java或JVM语言比如Scala、Groovy中具有互用性
- 无需担心底层协议
- 有queues和topics两种消息传递模型
- 支持事务、能够定义消息格式（消息头、属性和内容）

- 常见概念
  - JMS提供者：连接面向消息中间件的，JMS接口的一个实现，RocketMQ,ActiveMQ,Kafka等等
  - JMS生产者(Message Producer)：生产消息的服务
  - JMS消费者(Message Consumer)：消费消息的服务
  - JMS消息：数据对象
  - JMS队列：存储待消费消息的区域
  - JMS主题：一种支持发送消息给多个订阅者的机制
  - JMS消息通常有两种类型：点对点（Point-to-Point)、发布/订阅（Publish/Subscribe）

- 基础编程模型
  - MQ中需要用的一些类
  - ConnectionFactory： 连接工厂，JMS 用它创建连接
  - Connection： JMS 客户端到JMS Provider 的连接
  - Session： 一个发送或接收消息的线程

- Destination ： 消息的目的地;消息发送给谁.
- MessageConsumer / MessageProducer： 消息消费者，消息生产者

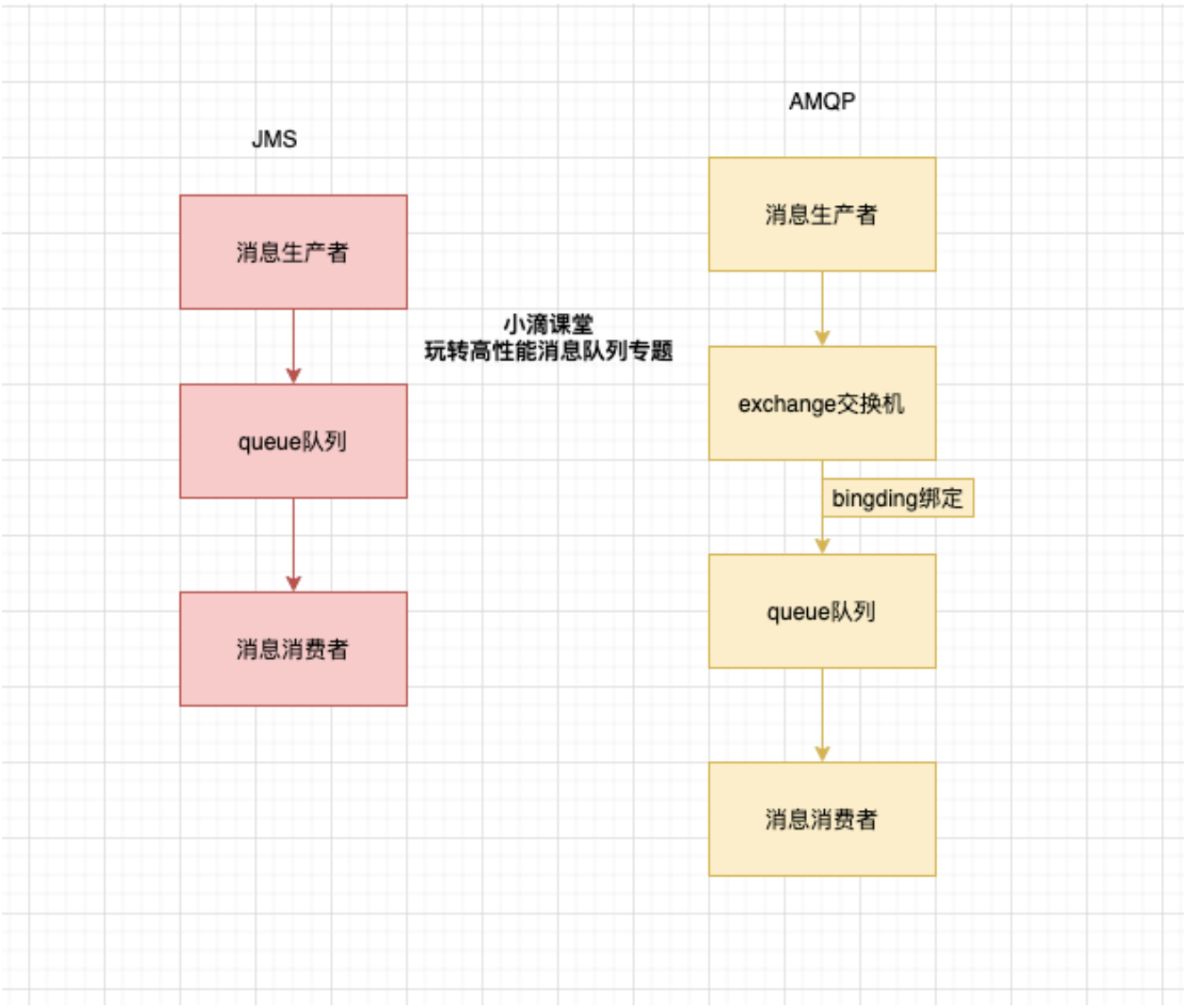
第3集 高级消息队列协议AMQP介绍和MQTT拓展

简介：介绍什么是AMQP高级消息队列协议和MQTT科普

- 背景
  - JMS或者NMS都没有标准的底层协议，API是与编程语言绑定的，每个消息队列厂商就存在多种不同格式规范的产品，对使用者就产生了很多问题, AMQP解决了这个问题，它使用了一套标准的底层协议
- 什么是AMQP
  - AMQP（advanced message queuing protocol）在2003年

时被提出，最早用于解决金融领不同平台之间的消息传递交互问题,就是是一种协议，兼容JMS

- 更准确说的链接协议 binary- wire-level-protocol 直接定义网络交换的数据格式，类似http
- 具体的产品实现比较多，RabbitMQ就是其中一种



- 特性
  - 独立于平台的底层消息传递协议
  - 消费者驱动消息传递
  - 跨语言和平台的互用性、属于底层协议

- 有5种交换类型direct，fanout，topic，headers，system
- 面向缓存的、可实现高性能、支持经典的消息队列，循环，存储和转发
- 支持长周期消息传递、支持事务（跨消息队列）

- AMQP和JMS的主要区别
  - AMQP不从API层进行限定，直接定义网络交换的数据格式,这使得实现了AMQP的provider天然性就是跨平台
  - 比如Java语言产生的消息，可以用其他语言比如python的进行消费
  - AQMP可以用http来进行类比，不关心实现接口的语言，只要都按照相应的数据格式去发送报文请求，不同语言的client可以和不同语言的server进行通讯
  - JMS消息类型：TextMessage/ObjectMessage/StreamMessage等
  - AMQP消息类型：Byte[]

- 科普：大家可能也听过MQTT
  - MQTT: 消息队列遥测传输（Message Queueing Telemetry Transport ）
  - 背景：
    - 我们有面向基于Java的企业应用的JMS和面向所有其他应用需求的AMQP，那这个MQTT是做啥的?

- 原因
  - 计算性能不高的设备不能适应AMQP上的复杂操作,MQTT它是专门为小设备设计的
  - MQTT主要是是物联网（IOT）中大量的使用
- 特性
  - 内存占用低，为小型无声设备之间通过低带宽发送短消息而设计
  - 不支持长周期存储和转发，不允许分段消息（很难发送长消息）
  - 支持主题发布-订阅、不支持事务（仅基本确认）
  - 消息实际上是短暂的（短周期）
  - 简单用户名和密码、不支持安全连接、消息不透明

第4集 架构师的解决方案-业界主流消息队列和技术选型讲解

简介：对比当下主流的消息队列和选择问题

- 业界主流的消息队列：Apache ActiveMQ、Kafka、RabbitMQ、RocketMQ
  - ActiveMQ：http://activemq.apache.org/
    - Apache出品，历史悠久，支持多种语言的客户端和协议，支持多种语言Java, .NET, C++ 等
    - 基于JMS Provider的实现
    - 缺点：吞吐量不高，多队列的时候性能下降，存在消息丢失的情况，比较少大规模使用
  - Kafka：http://kafka.apache.org/
    - 是由Apache软件基金会开发的一个开源流处理平台，由Scala和Java编写。Kafka是一种高吞吐量的分布式发布订阅消息系统(严格意义上是不属于队列产品，是一个流处理平台)，它可以处理大规模的网站中的所有动作流数据(网页浏览，搜索和其他用户的行动)，副本集机制，实现数据冗余，保障数据尽量不丢失；支持多个生产者和消费者
    - 类似MQ，功能较为简单，主要支持常规的MQ功能
  - 它提供了类似于JMS的特性，但是在设计实现上完全不同，它并不是JMS规范的实现
  - 缺点：运维难度大，文档比较少, 需要掌握Scala
- RocketMQ：http://rocketmq.apache.org/
  - 阿里开源的一款的消息中间件, 纯Java开发，具有高吞吐量、

高可用性、适合大规模分布式系统应用的特点, 性能强劲(零拷贝技术)，支持海量堆积, 支持指定次数和时间间隔的失败消息重发,支持consumer端tag过滤、延迟消息等，在阿里内部进行大规模使用，适合在电商，互联网金融等领域

- 基于JMS Provider的实现
- 缺点：社区相对不活跃，更新比较快，纯java支持
- RabbitMQ：http://www.rabbitmq.com/
  - 是一个开源的AMQP实现，服务器端用Erlang语言编写，支持多种客户端，如：Python、Ruby、.NET、Java、C、用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不错
  - 缺点：使用Erlang开发，阅读和修改源码难度大

- 什么我们讲Kafka呢，只要你目标是高级工程师或者架构师，就要多学，才知道解决方案+适合场景
  - 因为这个是Kafka专题课程
  - 下集专门介绍Kafka

我是一个开不起玩笑的人

如果你和我开玩笑
我就当真

愿景："让编程不再难学，让技术与生活更加有趣"

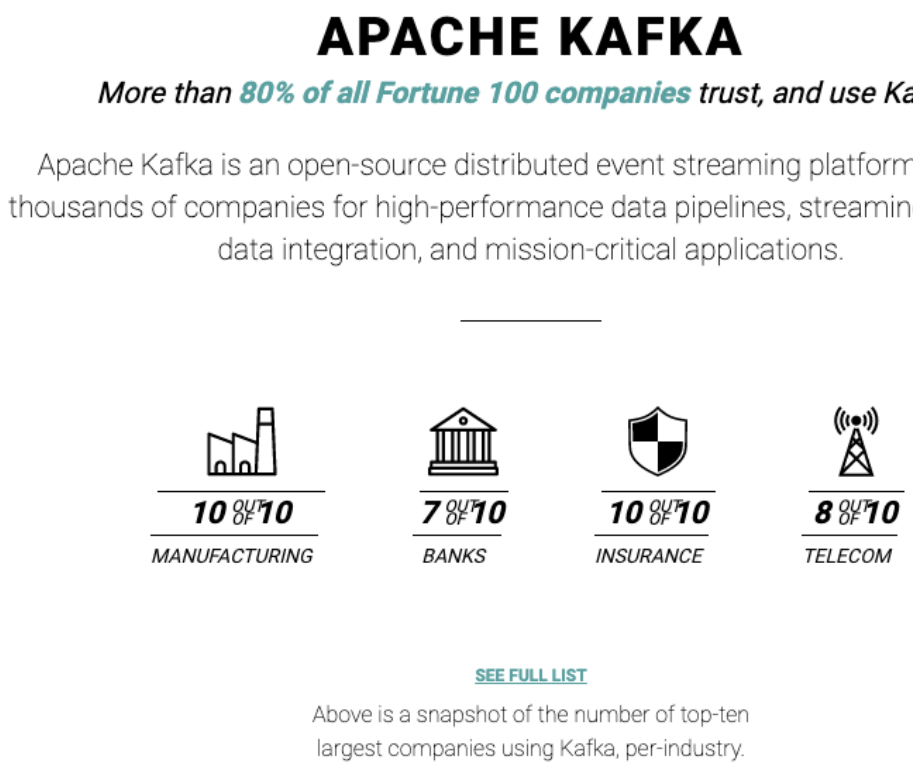
更多架构课程请访问 xdclass.net

第三章 急速入门Apache顶级项目Kafka核心概念+安装部署实战

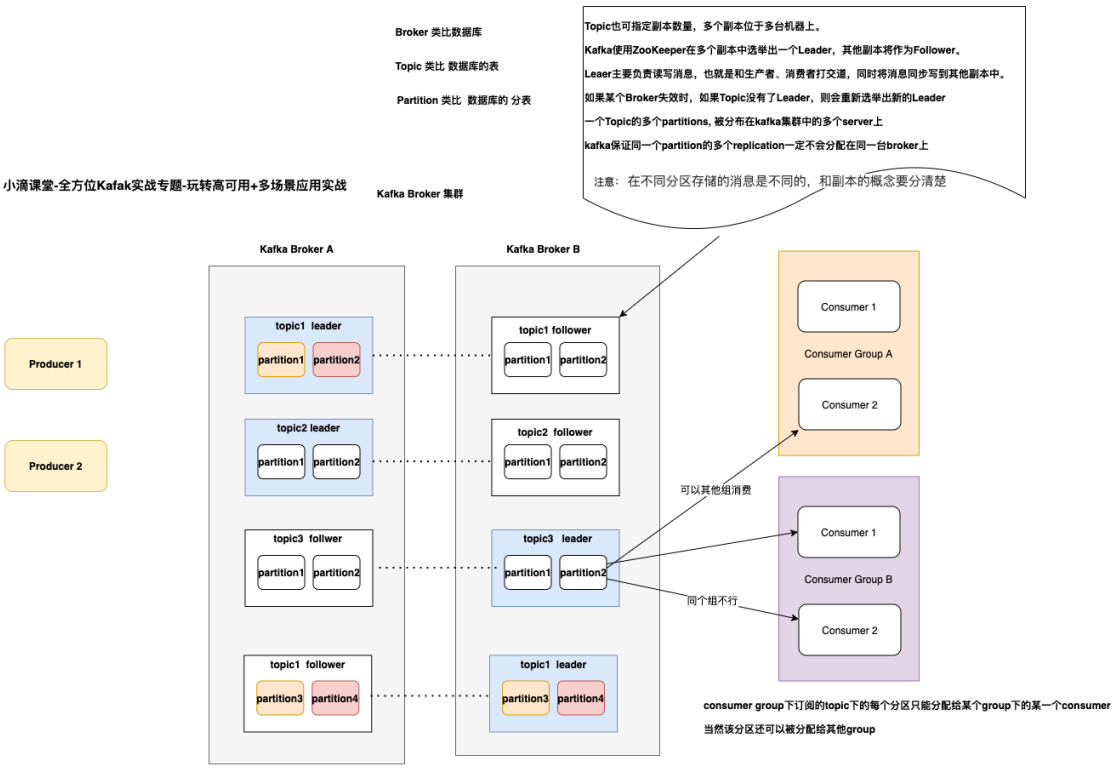
第1集 分布式流处理平台Kafka快速认知

简介：介绍分布式流处理平台kafka快速认知

- Kafka
  - Kafka是最初由Linkedin公司开发，Linkedin于2010年贡献给了Apache基金会并成为顶级开源项目，也是一个开源【分布式流处理平台】，由Scala和Java编写，（也当做MQ系统，但不是纯粹的消息系统）
    - open-source distributed event streaming platform
  - 核心：一种高吞吐量的分布式流处理平台，它可以处理消费者在网站中的所有动作流数据。
    - 比如 网页浏览，搜索和其他用户的行为等，应用于大数据实时处理领域



- 官网：http://kafka.apache.org/
- 快速开始：http://kafka.apache.org/quickstart
- 快速认识概念
  - Broker
    - Kafka的服务端程序，可以认为一个mq节点就是一个broker
    - broker存储topic的数据
  - Producer生产者
    - 创建消息Message，然后发布到MQ中
    - 该角色将消息发布到Kafka的topic中
  - Consumer消费者:
    - 消费队列里面的消息



注意：自 Kafka 2.4 之后，Kafka 提供了有限度的读写分离，也就是说Follower 副本能够对外提供读服务

为啥之前没有

场景不适用：读写分离适用于那种读负载很大，而写操作相对不频繁的场景，可 Kafka 不属于这样的场景。

同步机制：Kafka 采用 PULL 方式实现 Follower 的同步，因此Follower 与 Leader 存 在不一致性窗口。如果允许读 Follower 副本，就势必要处理消息滞后(Lagging)的问题。

第2集 【重要】分布式流处理平台Kafka核心概念介绍

简介：介绍分布式流处理平台kafka核心概念解释

- 核心概念
  - Broker
    - Kafka的服务端程序，可以认为一个mq节点就是一个broker
    - broker存储topic的数据
  - Producer生产者
    - 创建消息Message，然后发布到MQ中
    - 该角色将消息发布到Kafka的topic中
  - Consumer消费者:
    - 消费队列里面的消息
  - ConsumerGroup消费者组
  - 同个topic, 广播发送给不同的group，一个group中只有一个consumer可以消费此消息
  - Topic
  - 每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic，主题的意思

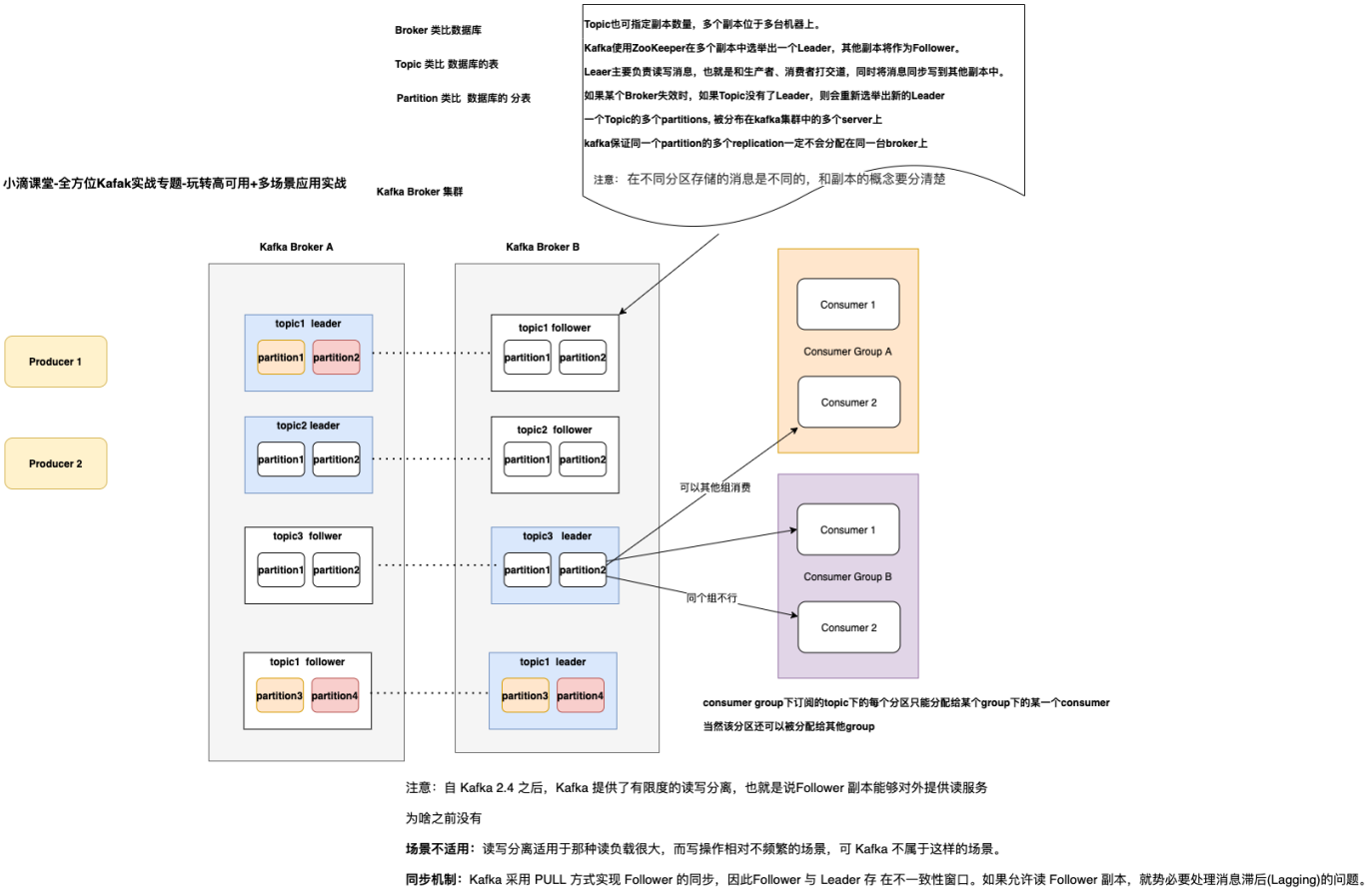
- Partition分区
  - kafka数据存储的基本单元，topic中的数据分割为一个或多个partition，每个topic至少有一个partition，是有序的
  - 一个Topic的多个partitions, 被分布在kafka集群中的多个server上
  - 消费者数量 <=小于或者等于Partition数量

- Replication 副本（备胎）
  - 同个Partition会有多个副本replication ，多个副本的数据是一样的，当其他broker挂掉后，系统可以主动用副本提供服务
  - 默认每个topic的副本都是1（默认是没有副本，节省资源），也可以在创建topic的时候指定
  - 如果当前kafka集群只有3个broker节点，则replication-factor最大就是3了，如果创建副本为4，则会报错

- ReplicationLeader、ReplicationFollower
  - Partition有多个副本，但只有一个replicationLeader负责该Partition和生产者消费者交互
  - ReplicationFollower只是做一个备份，从replicationLeader进行同步

- ReplicationManager

    - 负责Broker所有分区副本信息，Replication 副本状态切换
  - offset
    - 每个consumer实例需要为他消费的partition维护一个记录自己消费到哪里的偏移offset
    - kafka把offset保存在消费端的消费者组里



注意：自 Kafka 2.4 之后，Kafka 提供了有限度的读写分离，也就是说Follower 副本能够对外提供读服务

为啥之前没有

场景不适用：读写分离适用于那种读负载很大，而写操作相对不频繁的场景，可 Kafka 不属于这样的场景。

同步机制：Kafka 采用 PULL 方式实现 Follower 的同步，因此Follower 与 Leader 存 在不一致性窗口。如果允许读 Follower 副本，就势必要处理消息滞后(Lagging)的问题。

- 特点总结
  - 多订阅者
    - 一个topic可以有一个或者多个订阅者
    - 每个订阅者都要有一个partition，所以订阅者数量要少于等于partition数量

- 高吞吐量、低延迟: 每秒可以处理几十万条消息
- 高并发：几千个客户端同时读写
- 容错性：多副本、多分区，允许集群中节点失败，如果副本数据量为n,则可以n-1个节点失败
- 扩展性强：支持热扩展

- 基于消费者组可以实现：
  - 基于队列的模型：所有消费者都在同一消费者组里，每条消息只会被一个消费者处理
  - 基于发布订阅模型：消费者属于不同的消费者组，假如每个消费者都有自己的消费者组，这样kafka消息就能广播到所有消费者实例上

第3集 阿里云Linux服务器选择和常用软件介绍

简介：阿里云Linux服务器购买和常用软件介绍

- 云厂商
  - 阿里云：https://www.aliyun.com/
  - 腾讯云：https://cloud.tencent.com/
  - 亚马逊云：https://aws.amazon.com/
  - 阿里云新用户地址（如果地址失效，联系我或者客服即可，1折）
    - https://www.aliyun.com/minisite/goods?userCode=r5saexap&share_source=copy_link
- 环境问题说明
  - 务必使用CentOS 7 以上版本，64位系统，不要在Windows系统操作！！！！
  - 尽量前面先使用阿里云部署
  - 大家本地用虚拟机记得也要CentOS 7.x系统
    - vmware
- 注意：谁都不能保证每个人-硬件组成-系统版本-虚拟机软件版本都一样
  - 出现问题，大家结合报错日志搜索博文解决
  - 少数同学 -Win7、Win8、Win10、Mac、虚拟机等等，可能存在兼容问题

- 选购实操
- windows工具 putty，xshell, security CRT
  - 参考资料：
    - https://jingyan.baidu.com/article/e75057f210c6dcebc91a89dd.html
    - https://www.jb51.net/softjc/88235.html
- 苹果系统MAC： 通过终端登录
  - ssh root@ip 回车后输入密码
  - ssh root@120.24.216.117

- linux图形操作工具（用于远程连接上传文件）
  - mac: filezilla
    - sftp://120.24.216.117

    - windows: winscp
    - 参考资料：https://jingyan.baidu.com/article/ed2a5d1f346fd409f6be179a.html

- 可以尝试自己通过百度进行找文档， 安装mysql jdk nginx maven git redis等，也可以看我们的课程 xdclass.net

第4集 急速部署-Kafka相关环境准备和安装JDK8

简介： 急速部署-Kafka相关环境准备和安装

- 需要的软件和环境版本说明
    - kafka-xx-yy

        - xx 是scala版本，yy是kafka版本（scala是基于jdk开发，需要安装jdk环境）
        - 下载地址：http://kafka.apache.org/downloads
    - zookeeper
        - Apache 软件基金会的一个软件项目，它为大型分布式计算提供开源的分布式配置服务、同步服务和命名注册
        - 下载地址：https://zookeeper.apache.org/releases.html
    - jdk1.8
- 步骤
    - 上传安装包（zk、jdk、kafka）
    - 安装jdk
        - 配置全局环境变量
            - 解压：tar -zxvf jdk-8u181-linux-x64.tar.gz
            - 重命名
            - vim /etc/profile
            - 配置

```
JAVA_HOME=/usr/local/software/jdk1.8
CLASSPATH=$JAVA_HOME/lib/
PATH=$PATH:$JAVA_HOME/bin
export PATH JAVA_HOME CLASSPATH
```

            - 环境变量立刻生效
                - source /etc/profile

        - 查看安装情况 java -version

第5集 Linux环境下Zookeeper和Kafka安装启动

简介： Linux环境下Zookeeper和Kafka安装启动

- 安装Zookeeper (默认2181端口)
    - 默认配置文件 zoo.cfg
    - 启动zk `tar -zxvf .....`
        - bin/zkServer.sh start

- 安装Kafka (默认 9092端口)
  - config目录下 server.properties

```
#标识broker编号，集群中有多个broker，则每个broker的编号需要设置不同
broker.id=0

#修改下面两个配置 （ listeners 配置的ip和advertised.listeners相同时启动kafka会报错）
listeners(内网Ip)
advertised.listeners(公网ip)



#修改zk地址,默认地址
zookeeper.connection=localhost:2181
```

  - bin目录启动

```
#启动
./kafka-server-start.sh
../config/server.properties &

#停止
kafka-server-stop.sh
```

  - 创建topic

```
./kafka-topics.sh --create --zookeeper
112.74.55.160:2181 --replication-factor 1 --
partitions 1 --topic xdclass-topic
```

- 查看topic

```
./kafka-topics.sh --list --zookeeper
112.74.55.160:2181
```

第6集 Linux环境下daemon守护进程运行Kafka

简介： Linux环境下daemon守护进程运行Kafka

- kafka如果直接启动信息会打印在控制台,如果关闭窗口，kafka随之关闭
- 守护进程方式启动

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

愿景："让编程不再难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第四章 Kafka点对点-发布订阅模型讲解和写入存储流程实战

第1集 Kafka命令行生产者发送消息和消费者消费消息实战

简介： Kafka命令行生产者发送消息和消费者消费消息实战

- 创建topic

```
./kafka-topics.sh --create --zookeeper
192.168.60.11:2181 --replication-factor 1 --
partitions 2 --topic xdclass-topic
```

- 查看topic

```
./kafka-topics.sh --list --zookeeper
192.168.60.11:2181
```

- 生产者发送消息

```
./kafka-console-producer.sh --broker-list
192.168.60.11:9092  --topic version1-topic
```

- 消费者消费消息 (--from-beginning：会把主题中以往所有的数据都读取出来, 重启后会有这个重复消费)

```
./kafka-console-consumer.sh --bootstrap-server
192.168.60.11:9092 --from-beginning --topic t1
```

- 删除topic

```
./kafka-topics.sh --zookeeper 192.168.60.11:2181 -
-delete --topic t1
```

- 查看broker节点topic状态信息

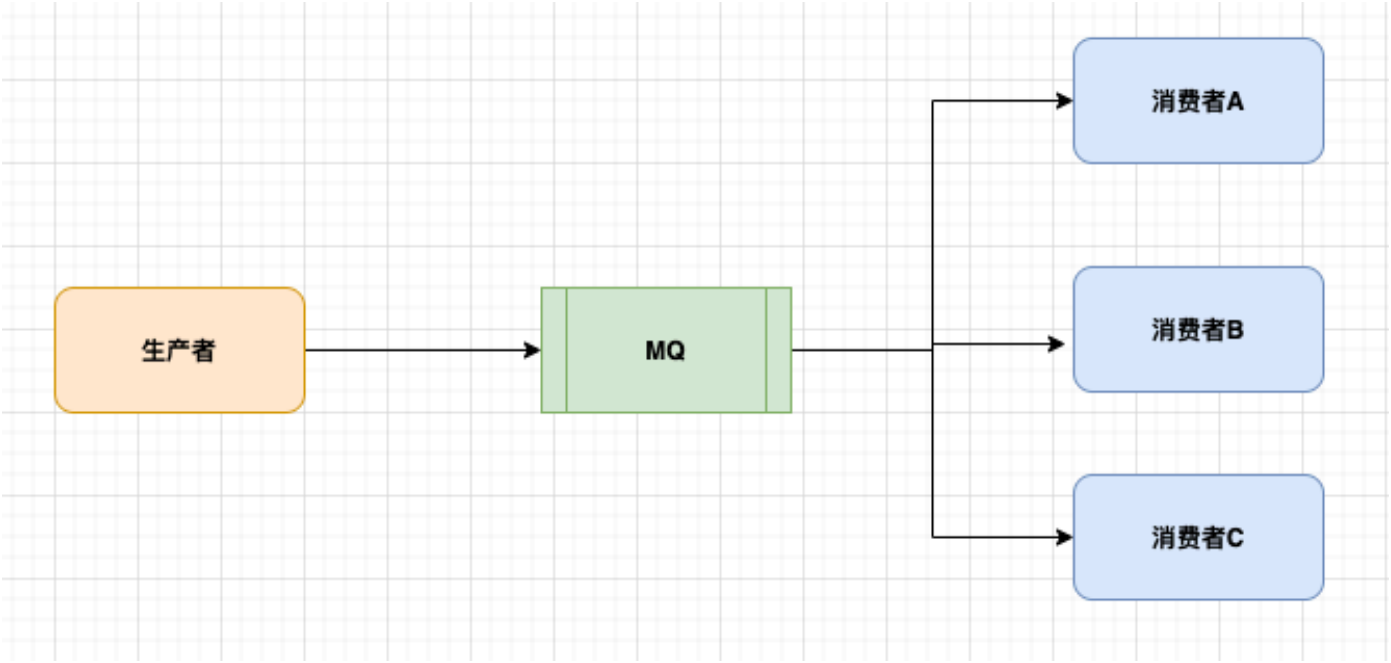
```
./kafka-topics.sh --describe --zookeeper
192.168.60.11:2181  --topic xdclass-topic
```

第2集 Kafka点对点模型和发布订阅模型讲解

简介： Kafka点对点模型和发布订阅模型讲解

- JMS规范目前支持两种消息模型
  - 点对点（point to point)
    - 消息生产者生产消息发送到queue中，然后消息消费者从queue中取出并且消费消息
    - 消息被消费以后，queue中不再有存储，所以消息消费者不可能消费到已经被消费的消息。 Queue支持存在多个消费者，但是对一个消息而言，只会有一个消费者可以消费
  - 发布/订阅（publish/subscribe）

- 消息生产者（发布）将消息发布到topic中，同时有多个消息消费者（订阅）消费该消息。
- 和点对点方式不同，发布到topic的消息会被所有订阅者消费。



第3集 Kafka消费者组配置实现点对点消费模型

简介： Kafka消费者组配置实现点对点消费模型

- 编辑消费者配置（确保同个名称group.id一样）
  - 编辑 config/consumer.properties
- 创建topic, 1个分区

```
./kafka-topics.sh --create --zookeeper
192.168.60.11:2181 --replication-factor 1 --
partitions 2 --topic t1
```

- 指定配置文件启动 两个消费者

```
./kafka-console-consumer.sh --bootstrap-server
112.74.55.160:9092 --from-beginning --topic t1 --
consumer.config ../config/consumer.properties
```

- 现象
  - 只有一个消费者可以消费到数据，一个分区只能被同个消费者组下的某个消费者进行消费

第4集 Kafka消费者组配置实现发布订阅消费模型

简介： Kafka消费者组配置实现发布订阅消费模型

- 编辑消费者配置（确保group.id 不一样）
  - 编辑 config/consumer-1.properties
  - 编辑 config/consumer-2.properties
- 创建topic, 2个分区

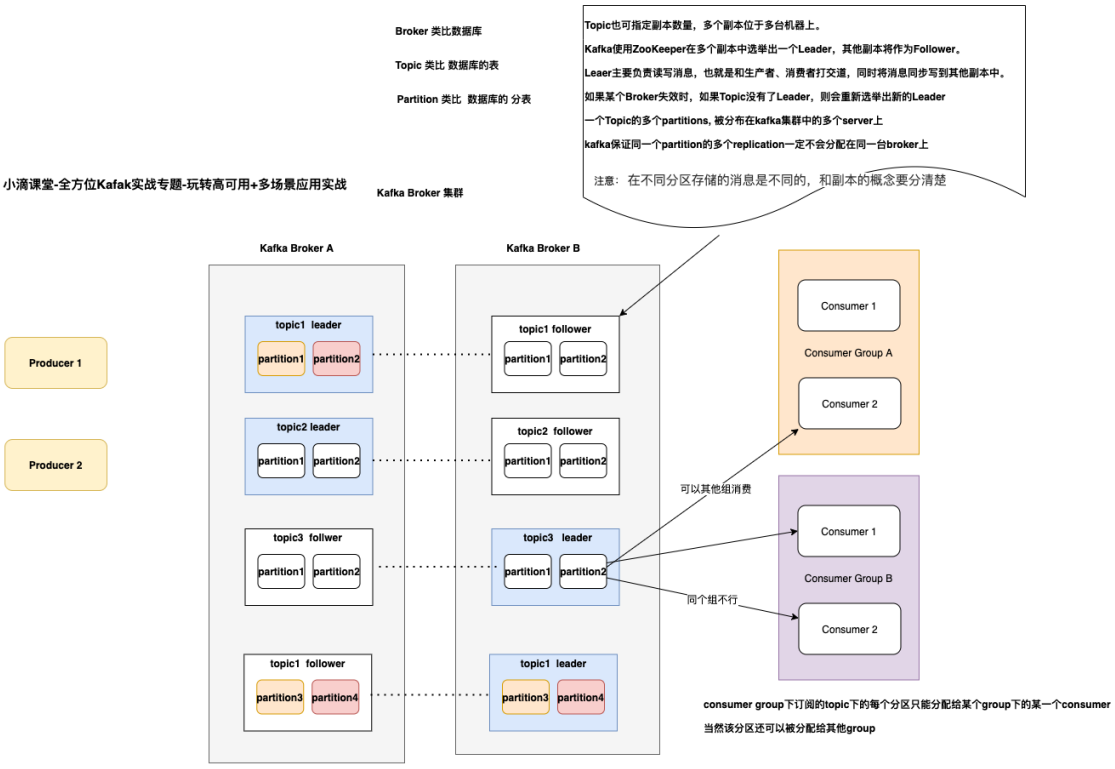
```
./kafka-topics.sh --create --zookeeper
112.74.55.160:2181 --replication-factor 1 --
partitions 2 --topic t2
```

- 指定配置文件启动 两个消费者

```
./kafka-console-consumer.sh --bootstrap-server
112.74.55.160:9092 --from-beginning --topic t1 --
consumer.config ../config/consumer-1.properties

./kafka-console-consumer.sh --bootstrap-server
112.74.55.160:9092 --from-beginning --topic t1 --
consumer.config ../config/consumer-2.properties
```

- 现象
  - 两个不同消费者组的节点，都可以消费到消息，实现发布订阅模型



注意：自 Kafka 2.4 之后，Kafka 提供了有限度的读写分离，也就是说Follower 副本能够对外提供读服务

为啥之前没有

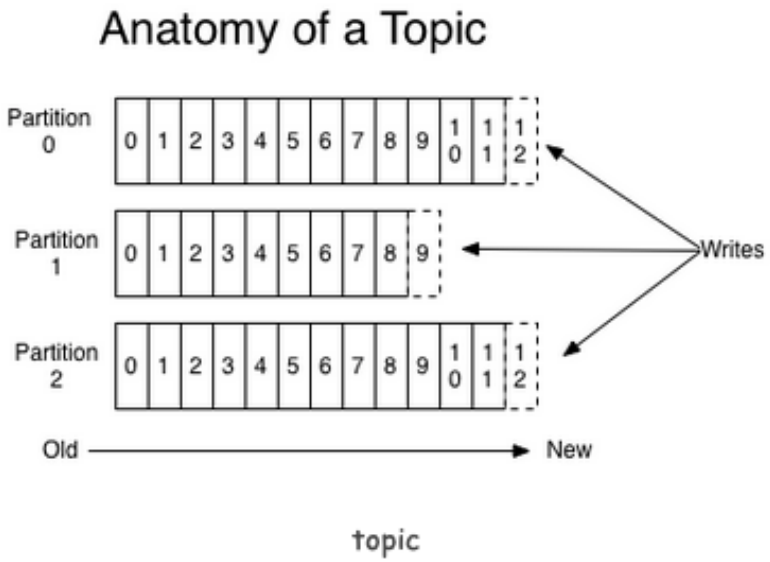
场景不适用：读写分离适用于那种读负载很大，而写操作相对不频繁的场景，可 Kafka 不属于这样的场景。

同步机制：Kafka 采用 PULL 方式实现 Follower 的同步，因此Follower 与 Leader 存 在不一致性窗口。如果允许读 Follower 副本，就势必要处理消息滞后(Lagging)的问题。

第5集 Kafka数据存储流程和原理概述和LEO+HW讲解

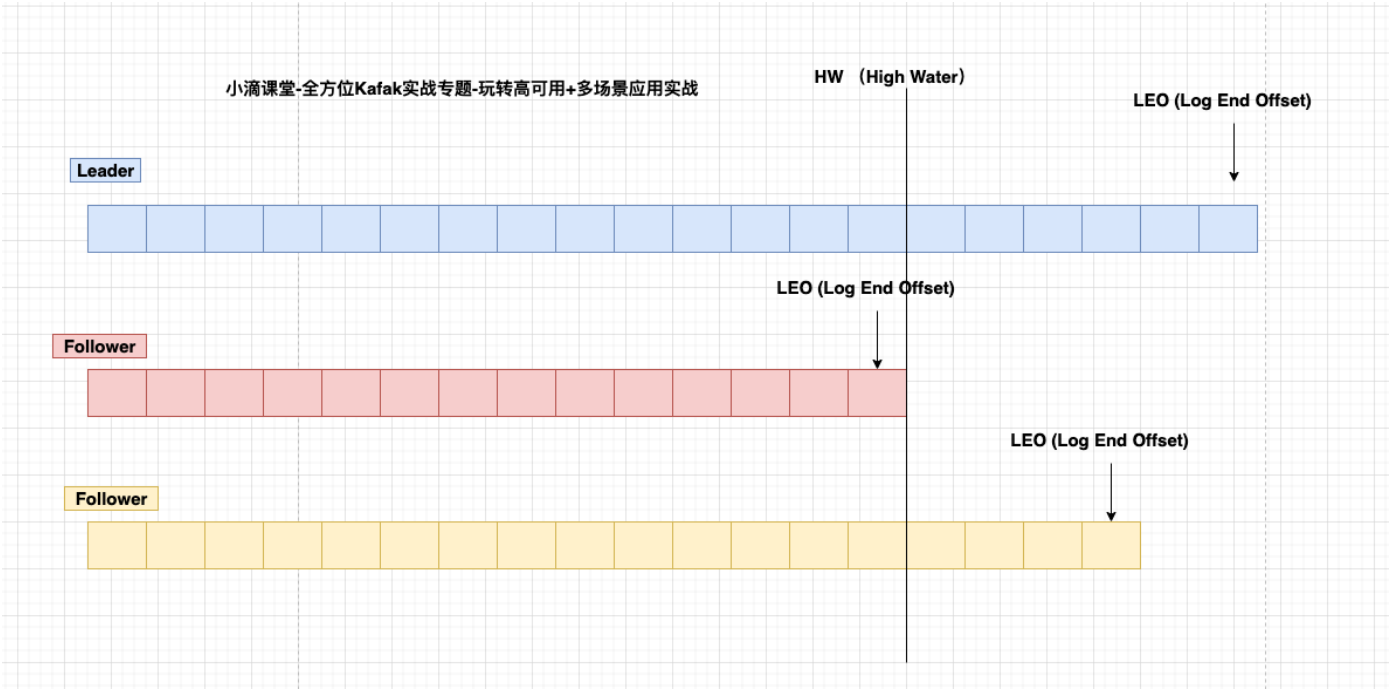
简介： Kafka数据存储流程、原理、LEO+HW讲解

- **Partition**
  - topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列
  - 是以文件夹的形式存储在具体Broker本机上



- **LEO（LogEndOffset）**
  - 表示每个partition的log最后一条Message的位置。

- **HW（HighWatermark）** 表示大家都拿到的数据点位置，
  - 表示partition各个replicas数据间同步且一致的offset位置，即表示allreplicas已经commit的位置
  - HW之前的数据才是Commit后的，对消费者才可见
  - ISR集合里面最小leo



- **offset**：
  - 每个partition都由一系列有序的、不可变的消息组成，这些消息被连续的追加到partition中
  - partition中的每个消息都有一个连续的序列号叫做offset，用于partition唯一标识一条消息
  - 可以认为offset是partition中Message的id

- **Segment**：每个partition又由多个segment file组成；
  - segment file 由2部分组成，分别为index file和data file（log file），
  - 两个文件是一一对应的，后缀”.index”和”.log”分别表示索引文件和数据文件
  - 命名规则：partition的第一个segment从0开始，后续每个segment文件名为上一个segment文件最后一条消息的offset+1

- Kafka高效文件存储设计特点：
  - Kafka把topic中一个parition大文件分成多个小文件段，通过多个小文件段，就容易定期清除或删除已经消费完文件，减少磁盘占用。
  - 通过索引信息可以快速定位message
  - producer生产数据，要写入到log文件中，写的过程中一直追加到文件末尾，为顺序写，官网数据表明。同样的磁盘，顺序写能到600M/S，而随机写只有100K/S

第五章 SpringBoot2.X项目整合-Kafka核心API-Admin实战

第1集 新版SpringBoot2.X项目搭建整合Kafka客户端

简介： SpringBoot2.X项目搭建整合Kafka客户端依赖配置

- 新版SpringBoot2.X介绍
  - 官网：https://spring.io/projects/spring-boot
  - GitHub地址：https://github.com/spring-projects/spring-boot
  - 官方文档：https://spring.io/guides/gs/spring-boot/
  - 视频地址：https://item.taobao.com/item.htm?id=618384570391
- 相关软件环境和作用
  - JDK1.8+以上
  - Maven3.5+
  - 编辑器IDEA(旗舰版)

- 在线创建： https://start.spring.io/
  - 注意：
    - 采用springboot2.5 + jdk11
    - 初次导入项目下载包比较慢 5~20分钟不等
      - 出问题的话: mvn clean install 试试
    - 不建议修改默认maven仓库（可以先还原默认的，防止下载包失败）
    - idea记得配置jdk11
- 在SpringBoot整合kafka很简单
  - 添加依赖 kafka-clients

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>2.4.0</version>
</dependency>
```

第2集 SpringBoot2.x整合Kafka客户端+adminApi单元测试

简介：SpringBoot2.x整合Kafka客户端+adminApi单元测试

- 单元测试配置客户端+创建topic

```
/**
 * 设置admin 客户端
 * @return
 */
public static AdminClient initAdminClient(){
    Properties properties = new Properties();

properties.setProperty(AdminClientConfig.BOOTSTRAP_S
ERVERS_CONFIG,"112.74.55.160:9092");

    AdminClient adminClient =
AdminClient.create(properties);
```

```
    return adminClient;
}

  //创建
    @Test
    public  void createTopic() {
        AdminClient adminClient = initAdminClient();
        // 2个分区，1个副本
        NewTopic newTopic = new NewTopic(TOPIC_NAME,
2 , (short) 1);

        CreateTopicsResult createTopicsResult =
adminClient.createTopics(Arrays.asList(newTopic));
        //future等待创建，成功不会有任何报错，如果创建失败和
超时会报错。
        try {
            createTopicsResult.all().get();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("创建新的topic");

    }
```

- 查看topic

```
./kafka-topics.sh --list --zookeeper
112.74.55.160:2181
```

- 查看broker节点topic状态信息

```
./kafka-topics.sh --describe --zookeeper
112.74.55.160:2181  --topic xdclass-sp-topic-test
```

第3集 Kafka使用JavaAPI-AdminClient删除和列举topic

简介：Kafka使用JavaAPI-AdminClient创建和列举topic

- list列举

```
    //获取
    @Test
    public  void listTopic() throws
ExecutionException, InterruptedException {
        AdminClient adminClient = initAdminClient();

        //是否查看内部的topic,可以不用
```

```
        ListTopicsOptions options = new
ListTopicsOptions();
        options.listInternal(true);

        ListTopicsResult listTopics =
adminClient.listTopics(options);
        Set<String> topics =
listTopics.names().get();
        for (String topic : topics) {
            System.err.println(topic);

        }
    }
```

- 删除

```
    //删除
    @Test
    public  void delTopicTest() {
        AdminClient adminClient = initAdminClient();
        DeleteTopicsResult deleteTopicsResult =
adminClient.deleteTopics(Arrays.asList("xdclass-
sp11-topic"));
        try {
            deleteTopicsResult.all().get();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

第4集 AdminClientApi查看Topic详情和增加分区数量

简介 AdminClientApi查看Topic详情和增加分区数量

- 查看topic详情

```
//获取指定topic的详细信息
    @Test
    public  void getTopicInfo() throws Exception {
        AdminClient adminClient = initAdminClient();
        DescribeTopicsResult describeTopicsResult =
adminClient.describeTopics(Arrays.asList(TOPIC_NAME)
);

        Map<String, TopicDescription>
stringTopicDescriptionMap =
describeTopicsResult.all().get();

        Set<Map.Entry<String, TopicDescription>>
entries = stringTopicDescriptionMap.entrySet();

        entries.stream().forEach((entry)->
System.out.println("name ： "+entry.getKey()+" , desc:
"+ entry.getValue()));
    }
```

- 增加分区数量

```
 /**
     * 增加分区数量
     *
     * 如果当主题中的消息包含有key时(即key不为null)，根据
key来计算分区的行为就会有所影响消息顺序性
```

```
     *
     * 注意：Kafka中的分区数只能增加不能减少，减少的话数据不知怎么处理
     *
     * @throws Exception
     */
    @Test
    public  void incrPartitionsTest() throws Exception{
        Map<String, NewPartitions> infoMap = new HashMap<>();
        NewPartitions newPartitions = NewPartitions.increaseTo(5);

        AdminClient adminClient = initAdminClient();
        infoMap.put(TOPIC_NAME, newPartitions);

        CreatePartitionsResult createPartitionsResult = adminClient.createPartitions(infoMap);
        createPartitionsResult.all().get();
    }
```

愿景："让编程不再难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第六章 玩转Kafka核心API生产者实战详解

第1集 生产者发送到Broker分区策略和常见配置讲解

简介： Kafka的producer生产者发送到Broker分区策略讲解

- 生产者发送到broker里面的流程是怎样的呢，一个 topic 有多个partition分区，每个分区又有多个副本
  - 如果指定Partition ID,则PR被发送至指定Partition (ProducerRecord)
  - 如果未指定Partition ID,但指定了Key, PR会按照hash(key)发送至对应Partition

    - 如果未指定Partition ID也没指定Key，PR会按照默认 round-robin轮训模式发送到每个Partition
        - 消费者消费partition分区默认是range模式
    - 如果同时指定了Partition ID和Key, PR只会发送到指定的Partition (Key不起作用，代码逻辑决定)
    - 注意：Partition有多个副本，但只有一个replicationLeader负责该Partition和生产者消费者交互
- 生产者到broker发送流程
    - Kafka的客户端发送数据到服务器，不是来一条就发一条，会经过内存缓冲区（默认是16KB），通过KafkaProducer发送出去的消息都是先进入到客户端本地的内存缓冲里，然后把很多消息收集到的Batch里面，再一次性发送到Broker上去的，这样性能才可能题高
- 生产者常见配置
    - 官方文档 http://kafka.apache.org/documentation/#producerconfigs

```
#kafka地址,即broker地址
bootstrap.servers

#当producer向leader发送数据时，可以通过request.required.acks参数来设置数据可靠性的级别,分别是0,1, all。
acks
```

#请求失败，生产者会自动重试，指定是0次，如果启用重试，则会有重复消息的可能性
retries

#每个分区未发送消息总字节大小,单位：字节，超过设置的值就会提交数据到服务端，默认值是16KB
batch.size

默认值就是0，消息是立刻发送的，即便batch.size缓冲空间还没有满，如果想减少请求的数量，可以设置 linger.ms 大于#0，即消息在缓冲区保留的时间，超过设置的值就会被提交到服务端
通俗解释是，本该早就发出去的消息被迫至少等待了linger.ms时间，相对于这时间内积累了更多消息，批量发送 减少请求
#如果batch被填满或者linger.ms达到上限，满足其中一个就会被发送
linger.ms

buffer.memory的用来约束Kafka Producer能够使用的内存缓冲的大小的，默认值32MB。
如果buffer.memory设置的太小，可能导致消息快速的写入内存缓冲里，但Sender线程来不及把消息发送到Kafka服务器
会造成内存缓冲很快就被写满，而一旦被写满，就会阻塞用户线程，不让继续往Kafka写消息了
buffer.memory要大于batch.size，否则会报申请内存不足的错误，不要超过物理内存，根据实际情况调整
buffer.memory

```
# key的序列化器，将用户提供的 key和value对象
ProducerRecord 进行序列化处理，key.serializer必须被设置，
即使
#消息中没有指定key，序列化器必须是一个实现
org.apache.kafka.common.serialization.Serializer接口
的类，将#key序列化成字节数组。
key.serializer
value.serializer
```

第2集 Kafka核心API模块-producer API讲解实战

简介： Kafka核心API模块-producer API讲解实战

- 封装配置属性

```
public static Properties getProperties(){
        Properties props = new Properties();

        props.put("bootstrap.servers",
"112.74.55.160:9092");
```

```
//props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
"112.74.55.160:9092");

        // 当producer向leader发送数据时，可以通过
request.required.acks参数来设置数据可靠性的级别,分别是0,
1，all。
        props.put("acks", "all");
        //props.put(ProducerConfig.ACKS_CONFIG,
"all");

        // 请求失败，生产者会自动重试，指定是0次，如果启用重
试，则会有重复消息的可能性
        props.put("retries", 0);
        //props.put(ProducerConfig.RETRIES_CONFIG,
0);


        // 生产者缓存每个分区未发送的消息,缓存的大小是通过
batch.size 配置指定的，默认值是16KB
        props.put("batch.size", 16384);


        /**
         * 默认值就是0，消息是立刻发送的，即便batch.size缓
冲空间还没有满
         * 如果想减少请求的数量，可以设置 linger.ms 大于
0，即消息在缓冲区保留的时间，超过设置的值就会被提交到
服务端
```

```
            * 通俗解释是，本该早就发出去的消息被迫至少等待了
linger.ms时间，相对于这时间内积累了更多消息，批量发送
减少请求
            * 如果batch被填满或者linger.ms达到上限，满足其中
一个就会被发送
            */
           props.put("linger.ms", 1);

           /**
            * buffer.memory的用来约束Kafka Producer能够使
用的内存缓冲的大小的，默认值32MB。
            * 如果buffer.memory设置的太小，可能导致消息快速的
写入内存缓冲里，但Sender线程来不及把消息发送到
Kafka服务器
            * 会造成内存缓冲很快就被写满，而一旦被写满，就会阻塞
用户线程，不让继续往Kafka写消息了
            * buffer.memory要大于batch.size，否则会报申请内
存不#足的错误，不要超过物理内存，根据实际情况调整
            * 需要结合实际业务情况压测进行配置
            */
           props.put("buffer.memory", 33554432);


           /**
            * key的序列化器，将用户提供的 key和value对象
ProducerRecord 进行序列化处理，key.serializer必须被
设置，
            * 即使消息中没有指定key，序列化器必须是一个实
```

```
org.apache.kafka.common.serialization.Serializer接口的类，
          * 将key序列化成字节数组。
          */
         props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

props.put("value.serializer","org.apache.kafka.common.serialization.StringSerializer");

         return props;
     }
```

- 生产者投递消息API实战（同步发送）

```
    /**
     * 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,即batch.size和linger.ms结合
     *
     * 实现同步发送：一条消息发送之后，会阻塞当前线程，直至返回 ack
     * 发送消息后返回的一个 Future 对象，调用get即可
     *
     * 消息发送主要是两个线程：一个是Main用户主线程，一个是Sender线程
```

```
     *  1)main线程发送消息到RecordAccumulator即返回
     *  2)sender线程从RecordAccumulator拉取信息发送到
broker
     *  3) batch.size和linger.ms两个参数可以影响 sender
线程发送次数
     *
     *
     */
    @Test
    public void testSend(){

        Properties props = getProperties();
        Producer<String, String> producer = new
KafkaProducer<>(props);
        for (int i = 1; i < 3; i++){
            Future<RecordMetadata>  future =
producer.send(new ProducerRecord<>("my-topic",
"xdclass-key"+i, "xdclass-value"+i));
            try {
                RecordMetadata recordMetadata =
future.get();//不关心是否发送成功，则不需要这行
                System.out.println("发送状
态："+recordMetadata.toString());

            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
```

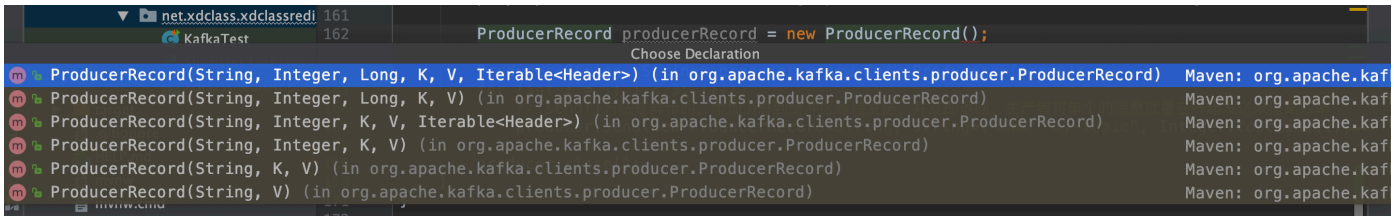
```
                System.out.println(i+"发
送: "+LocalDateTime.now().toString());
            }
            producer.close();
        }
```

第3集 【面试】 ProducerRecord介绍和key的作用

简介： ProducerRecord介绍和key的作用

- ProducerRecord（简称PR）
  - 发送给Kafka Broker的key/value 值对, 封装基础数据信息

```
-- Topic （名字）
-- PartitionID （可选）
-- Key(可选)  确定partioner位置用的
-- Value
```



```java
/**
 * Creates a record to be sent to a specified topic and partition
 *
 * @param topic The topic the record will be appended to
 * @param partition The partition to which the record should be sent
 * @param key The key that will be included in the record
 * @param value The record contents
 */
public ProducerRecord(String topic, Integer partition, K key, V value) {
    this(topic, partition, timestamp: null, key, value, headers: null);
}
```

- key默认是null，大多数应用程序会用到key
  - 如果key为空，kafka使用默认的partitioner，使用RoundRobin算法将消息均衡地分布在各个partition上
  - 如果key不为空，kafka使用自己实现的hash方法对key进行散列，决定消息该被写到Topic的哪个partition，拥有相同key的消息会被写到同一个partition，实现顺序消息

第4集 Kafka核心API模块-producerAPI回调函数实战

简介： Kafka核心API模块-producerAPI回调函数实战

- 生产者发送消息是异步调用，怎么知道是否有异常?
  - 发送消息配置回调函数即可， 该回调方法会在 Producer 收到 ack 时被调用，为异步调用
  - 回调函数有两个参数 RecordMetadata 和 Exception，如果 Exception 是 null，则消息发送成功，否则失败
- 异步发送配置回调函数

```
    @Test
    public void testSendWithCallback(){

        Properties props = getProperties();
        Producer<String, String> producer = new
KafkaProducer<>(props);
        for (int i = 1; i < 3; i++){
            producer.send(new ProducerRecord<>("my-
topic", "xdclass-key" + i, "xdclass-value" + i), new
Callback() {
                @Override
                public void
onCompletion(RecordMetadata metadata, Exception
exception) {
                    if (exception == null) {
                        System.out.println("发送状
态: "+metadata.toString());
```

```
                    } else {
                        exception.printStackTrace();
                    }
                }
            });

            System.out.println(i+"发送："+LocalDateTime.now().toString());
        }
        producer.close();
    }
```

第5集 producer生产者发送指定分区实战

简介：producer生产者发送指定分区实战

- 创建topic，配置5个分区，1个副本
- 发送代码

```
 @Test
    public void testSendWithCallbackAndPartition(){

        Properties props = getProperties();
        Producer<String, String> producer = new
KafkaProducer<>(props);
        for (int i = 0; i < 6; i++){
            producer.send(new ProducerRecord<>("my-
topic",i, "xdclass-key" + i, "xdclass-value" + i),
new Callback() {
                @Override
                public void
onCompletion(RecordMetadata metadata, Exception
exception) {
                    if (exception == null) {
                        System.out.println("发送状
态: "+metadata.toString());
                    } else {
                        exception.printStackTrace();
                    }
                }
            });

            System.out.println(i+"发
送: "+LocalDateTime.now().toString());
        }
        producer.close();

    }
```

第6集 【高级篇】Kafka生产者自定义partition分区规则实战

简介：Kafka 生产者自定义partition分区规则实战

- 源码解读默认分区器

```
org.apache.kafka.clients.producer.internals.DefaultPartitioner
```

- 自定义分区规则
  - 创建类，实现Partitioner接口，重写方法
  - 配置 partitioner.class 指定类即可

```
public class XdclassPartitioner implements Partitioner {
```

```
    @Override
    public int partition(String topic, Object key,
byte[] keyBytes, Object value, byte[] valueBytes,
Cluster cluster) {
        List<PartitionInfo> partitions =
cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();

        if("xdclass".equals(key)) {
            return 0;
        }
        //使用hash值取模，确定分区(默认的也是这个方式)
        return
Utils.toPositive(Utils.murmur2(keyBytes)) %
numPartitions;

    }

    @Override
    public void close() {

    }

    @Override
    public void configure(Map<String, ?> configs)
{

    }
```

```
}
```

```
  @Test
    public void testSend(){

        Properties props = getProperties();
        props.put("partitioner.class",
"net.xdclass.xdclassredis.XdclassPartitioner");
        Producer<String, String> producer = new
KafkaProducer<>(props);
        for (int i = 0; i < 3; i++){
            Future<RecordMetadata>  future =
producer.send(new ProducerRecord<>(TOPIC_NAME,
"xdclass", "xdclass-value"+i));
            try {
                RecordMetadata recordMetadata =
future.get();
                System.out.println("发送状
态："+recordMetadata.toString());

            } catch (InterruptedException e) {
                e.printStackTrace();
            } catch (ExecutionException e) {
                e.printStackTrace();
            }
            System.out.println(i+"发
送："+LocalDateTime.now().toString());
        }
        producer.close();
```

```
    }
```

愿景："让编程不再难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第七章 玩转Kafka核心API消费者模块实战详解

第1集 【面试】Consumer消费者机制和分区策略讲解《上》

简介： Kafka的Consumer消费者机制和分区策略讲解

- 思路：从面试题角度去讲解原理流程，大家更好接收
- 消费者根据什么模式从broker获取数据的?
  - 为什么是pull模式，而不是broker主动push?
    - 消费者采用 pull 拉取方式，从broker的partition获取数据
    - pull 模式则可以根据 consumer 的消费能力进行自己调整，不同的消费者性能不一样
      - 如果broker没有数据，consumer可以配置 timeout 时间，阻塞等待一段时间之后再返回
    - 如果是broker主动push，优点是可以快速处理消息，但是容易造成消费者处理不过来，消息堆积和延迟。
- 消费者从哪个分区进行消费?
  - 一个 topic 有多个 partition，一个消费者组里面有多个消费者，那是怎么分配过
    - 一个主题topic可以有多个消费者，因为里面有多个partition分区 (leader分区)

    - 一个partition leader可以由一个消费者组中的一个消费者进行消费
    - 一个 topic 有多个 partition，所以有多个partition leader，给多个消费者消费，那分配策略如何?

第2集 【面试】Consumer消费者机制和分区策略讲解《下》

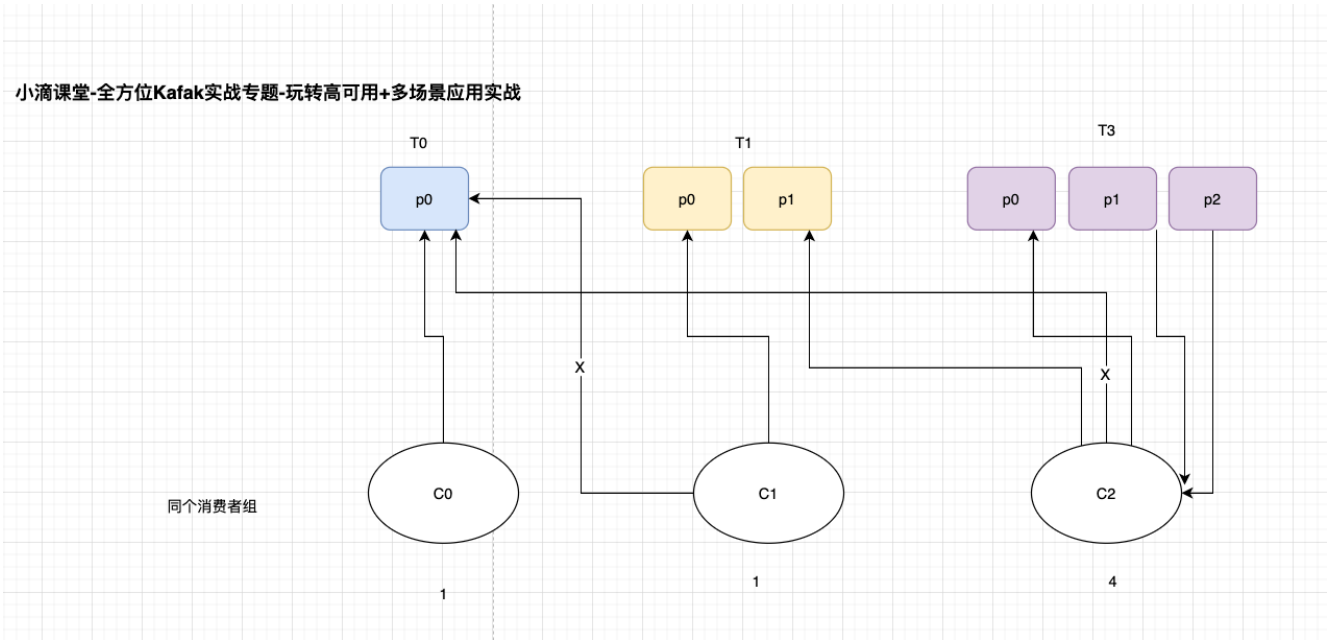
简介： Kafka的Consumer消费者机制和分区策略讲解

- 消费者从哪个分区进行消费？两个策略
    - 顶层接口

```
org.apache.kafka.clients.consumer.internals.AbstractPartitionAssignor
```

    - round-robin （**RoundRobinAssignor**非默认策略）轮训

- 【按照消费者组】进行轮训分配，同个消费者组监听不同主题也一样，是把所有的 partition 和所有的 consumer 都列出来， 所以消费者组里面订阅的主题是一样的才行，主题不一样则会出现分配不均问题，例如7个分区，同组内2个消费者
- topic-p0/topic-p1/topic-p2/topic-p3/topic-p4/topic-p5/topic-p6
- c-1: topic-p0/topic-p2/topic-p4/topic-p6
- c-2:topic-p1/topic-p3/topic-p5
- 弊端
  - 如果同一消费者组内，所订阅的消息是不相同的，在执行分区分配的时候不是轮询分配，可能会导致分区分配的不均匀
  - 有3个消费者C0、C1和C2，他们共订阅了 3 个主题：t0、t1 和 t2
  - t0有1个分区(p0)，t1有2个分区(p0、p1)，t2有3个分区(p0、p1、p2))
  - 消费者C0订阅的是主题t0，消费者C1订阅的是主题t0和t1，消费者C2订阅的是主题t0、t1和t2



- range （**RangeAssignor**默认策略）范围
  - 【按照主题】进行分配，如果不平均分配，则第一个消费者会分配比较多分区， 一个消费者监听不同主题也不影响，例如7个分区，同组内2个消费者
  - topic-p0/topic-p1/topic-p2/topic-p3/topic-p4/topic-p5//topic-p6
  - c-1: topic-p0/topic-p1/topic-p2/topic-p3
  - c-2:topic-p4/topic-p5/topic-p6
  - 弊端
    - 只是针对 1 个 topic 而言，c-1多消费一个分区影响不大
    - 如果有 N 多个 topic，那么针对每个 topic，消费者 C-1 都将多消费 1 个分区，topic越多则消费的分区也越多，则性能有所下降

第3集 【面试】Consumer重新分配策略和offset维护机制

简介： Consumer消费者重新分配策略和offset维护机制

- 什么是Rebalance操作
    - kafka 怎么均匀地分配某个 topic 下的所有 partition 到各个消费者，从而使得消息的消费速度达到最快，这就是平衡（balance），前面讲了 Range 范围分区 和 RoundRobin 轮询分区，也支持自定义分区策略。

- 而 rebalance（重平衡）其实就是重新进行 partition 的分配，从而使得 partition 的分配重新达到平衡状态

- 面试
  - 例如70个分区，10个消费者，但是先启动一个消费者，后续再启动一个消费者，这个会怎么分配?
    - Kafka 会进行一次分区分配操作，即 Kafka 消费者端的 Rebalance 操作 ，下面都会发生rebalance操作
      - 当消费者组内的消费者数量发生变化（增加或者减少），就会产生重新分配patition
      - 分区数量发生变化时(即 topic 的分区数量发生变化时)

- 面试：当消费者在消费过程突然宕机了，重新恢复后是从哪里消费，会有什么问题?
  - 消费者会记录offset，故障恢复后从这里继续消费，这个offset记录在哪里?
  - 记录在zk里面和本地，新版默认将offset保证在kafka的内置topic中，名称是 __consumer_offsets
    - 该Topic默认有50个Partition，每个Partition有3个副本，分区数量由参数offset.topic.num.partition配置
    - 通过groupId的哈希值和该参数取模的方式来确定某个消费者组已消费的offset保存到__consumer_offsets主题的哪

个分区中

- 由 消费者组名+主题+分区，确定唯一的offset的key，从而获取对应的值
- 三元组：**group.id+topic+分区号**，而 value 就是 offset 的值

第4集 Consumer配置讲解和Kafka调试日志配置

简介： Consumer配置讲解和Kafka调试日志配置

- springboot关闭kafka调试日志

```
#yml配置文件修改
logging:
  config: classpath:logback.xml
```

```
#logback.xml内容
<configuration>
    <appender name="STDOUT"
class="ch.qos.logback.core.ConsoleAppender">
        <encoder
class="ch.qos.logback.classic.encoder.PatternLayoutE
ncoder">
            <!-- 格式化输出： %d表示日期， %thread表示线
程名， %-5level: 级别从左显示5个字符宽度 %msg:日志消息， %n
是换行符 -->
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS}
[%thread] %-5level %logger{50} - %msg%n</pattern>
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

- 消费者配置

```
#消费者分组ID，分组内的消费者只能消费该消息一次，不同分组内的
消费者可以重复消费该消息
group.id

#为true则自动提交偏移量
enable.auto.commit
```

```
#自动提交offset周期
auto.commit.interval.ms

#重置消费偏移量策略，消费者在读取一个没有偏移量的分区或者偏移量无效情况下（因消费者长时间失效、包含偏移量的记录已经过时并被删除）该如何处理，
#默认是latest，如果需要从头消费partition消息，需要改为 earliest 且消费者组名变更 才可以
auto.offset.reset

#序列化器
key.deserializer
```

第5集 Kafka消费者Consumer消费消息配置实战

简介： Kafka消费者Consumer消费消息配置实战

- 配置

```
 public static Properties getProperties() {
        Properties props = new Properties();

        //broker地址
        props.put("bootstrap.servers",
"112.74.55.160:9092");

        //消费者分组ID，分组内的消费者只能消费该消息一次，不
同分组内的消费者可以重复消费该消息
        props.put("group.id", "xdclass-g1");

        //开启自动提交offset
        props.put("enable.auto.commit", "true");

        //自动提交offset延迟时间
        props.put("auto.commit.interval.ms",
"1000");

        //反序列化
        props.put("key.deserializer",
"org.apache.kafka.common.serialization.StringDeseria
lizer");
```

```
        props.put("value.deserializer",
"org.apache.kafka.common.serialization.StringDeseria
lizer");

        return props;
    }
```

- 消费订阅

```
@Test
    public void simpleConsumerTest(){
        Properties props = getProperties();
        KafkaConsumer<String, String> consumer = new
KafkaConsumer<>(props);

        //订阅topic主题

consumer.subscribe(Arrays.asList(KafkaProducerTest.T
OPIC_NAME));

       while (true) {
            //拉取时间控制，阻塞超时时间
            ConsumerRecords<String, String> records
= consumer.poll(Duration.ofMillis(500));
            for (ConsumerRecord<String, String>
record : records) {
                System.err.printf("topic = %s,
offset = %d, key = %s, value = %s%n",record.topic(),
record.offset(), record.key(), record.value());
```

```
            }
        }
    }
```

第6集 Consumer从头消费配置和手工提交offset配置

简介： Consumer手工提交offset配置和从头消费配置

- 如果需要从头消费partition消息，怎操作?
  - auto.offset.reset 配置策略即可
  - 默认是latest，需要改为 earliest 且消费者组名变更 ，即可实现从头消费

```
//默认是latest，如果需要从头消费partition消息，需要改为 earliest 且消费者组名变更，才生效
props.put("auto.offset.reset","earliest");
```

- 自动提交offset问题
  - 没法控制消息是否正常被消费
  - 适合非严谨的场景，比如日志收集发送

- 手工提交offset配置和测试
  - 初次启动消费者会请求broker获取当前消费的offset值

- 手工提交offset
  - 同步 commitSync 阻塞当前线程 (自动失败重试)
  - 异步 commitAsync 不会阻塞当前线程 (没有失败重试，回调callback函数获取提交信息，记录日志)

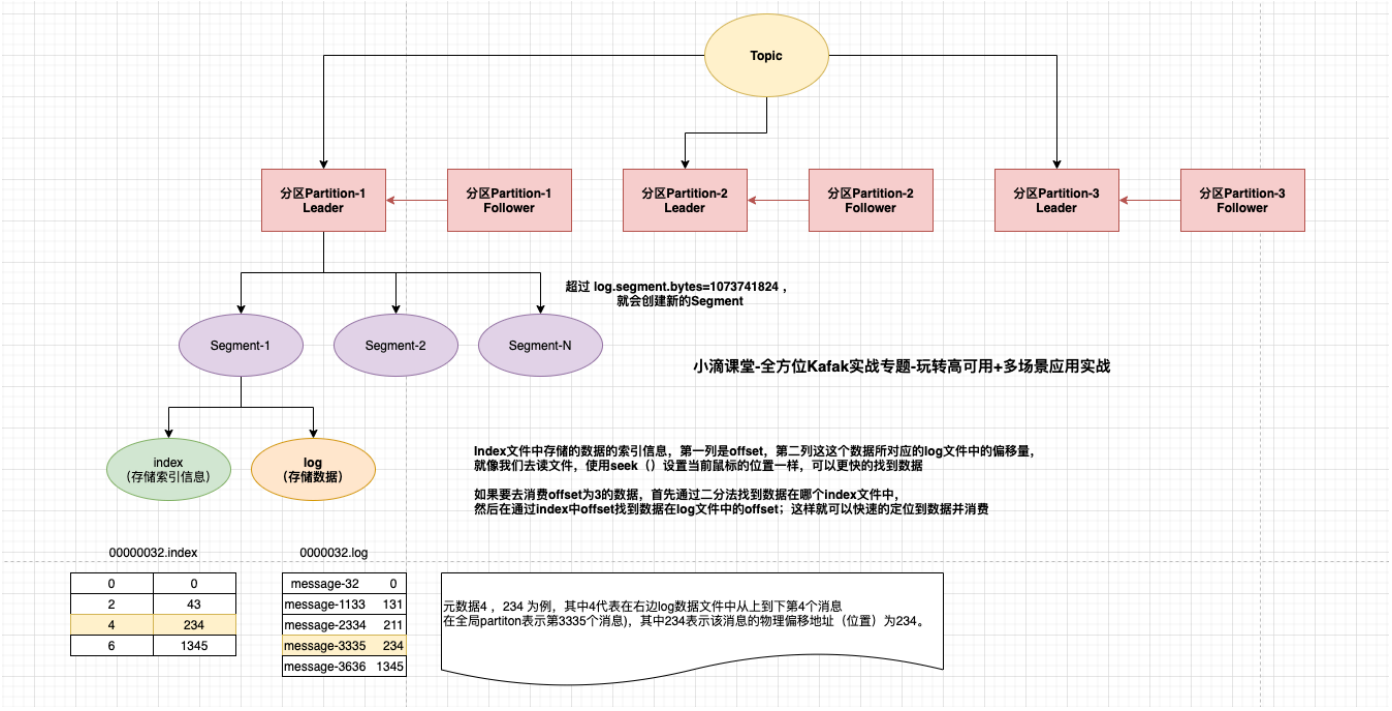
第八章 高级篇-kafka数据文件存储-可靠性保证-ISR核心知识讲解

第1集 Kafka数据存储流程和log日志讲解

简介： Kafka数据存储流程和log日志讲解

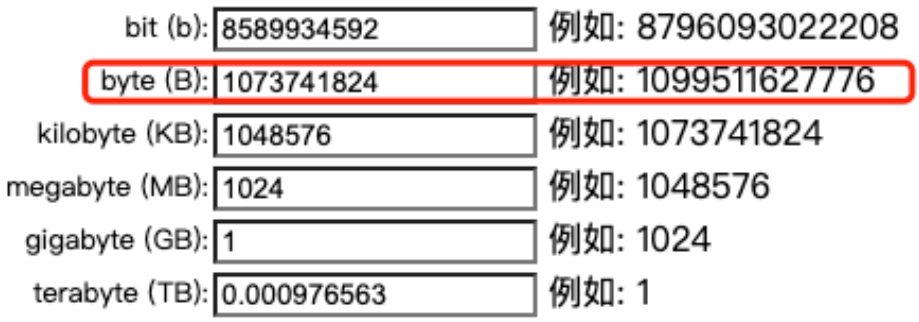
- Kafka 采取了**分片**和**索引**机制，将每个partition分为多个segment，每个segment对应2个文件 log 和 index
- 新增备注

> index文件中并没有为每一条message建立索引，采用了稀疏存储的方式
> 每隔一定字节的数据建立一条索引，避免了索引文件占用过多的空间和资源，从而可以将索引文件保留到内存中
> 缺点是没有建立索引的数据在查询的过程中需要小范围内的顺序扫描操作。



- 配置文件 server.properties

```
# The maximum size of a log segment file. When this size is reached a new log segment will be created. 默认是1G,当log数据文件大于1g后，会创建一个新的log文件（即segment，包括index和log）
log.segment.bytes=1073741824
```



- 例子

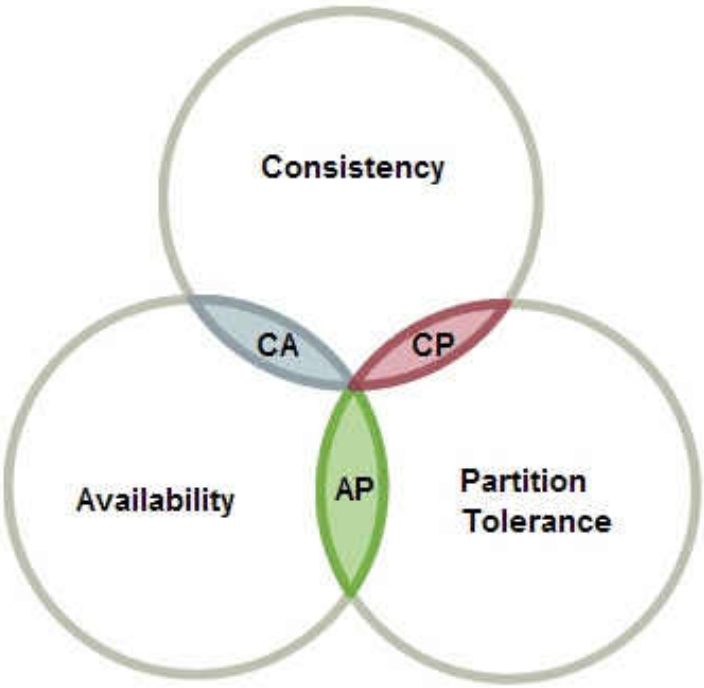
```
#分段一
00000000000000000000.index  00000000000000000000.log
#分段二 数字 1234指的是当前文件的最小偏移量offset，即上个文件的最后一个消息的offset+1
00000000000000001234.index  00000000000000001234.log
#分段三
00000000000000088888.index  00000000000000088888.log
```

第2集 【核心】掌握分布式系统里面你必须知道的CAP理论

简介：讲解分布式应用核心CAP知识

- 可能会有疑惑，可以看多几遍

- CAP定理: 指的是在一个分布式系统中，Consistency（一致性）、 Availability（可用性）、Partition tolerance（分区容错性），三者不可同时获得
    - 一致性（C）： 所有节点都可以访问到最新的数据；锁定其他节点，不一致之前不可读
    - 可用性（A）： 每个请求都是可以得到响应的，不管请求是成功还是失败；被节点锁定后 无法响应
    - 分区容错性（P）： 除了全部整体网络故障，其他故障都不能导致整个系统不可用,；节点间通信可能失败，无法避免
- CAP理论就是说在分布式存储系统中，最多只能实现上面的两点。而由于当前的网络硬件肯定会出现延迟丢包等问题，所以分区容忍性是我们必须需要实现的。所以我们只能在一致性和可用性之间进行权衡



CA： 如果不要求P（不允许分区），则C（强一致性）和A（可用性）是可以保证的。但放弃P的同时也就意味着放弃了系统的扩展性，也就是分布式节点受限，没办法部署子节点，这是违背分布式系统设计的初衷的

CP：如果不要求A（可用），每个请求都需要在服务器之间保持强一致，而P（分区）会导致同步时间无限延长(也就是等待数据同步完才能正常访问服务)，一旦发生网络故障或者消息丢失等情况，就要牺牲用户的体验，等待所有数据全部一致了之后再让用户访问系统

AP：要高可用并允许分区，则需放弃一致性。一旦分区发生，节点之间可能会失去联系，为了高可用，每个节点只能用本地数据提供服务，而这样会导致全局数据的不一致性。

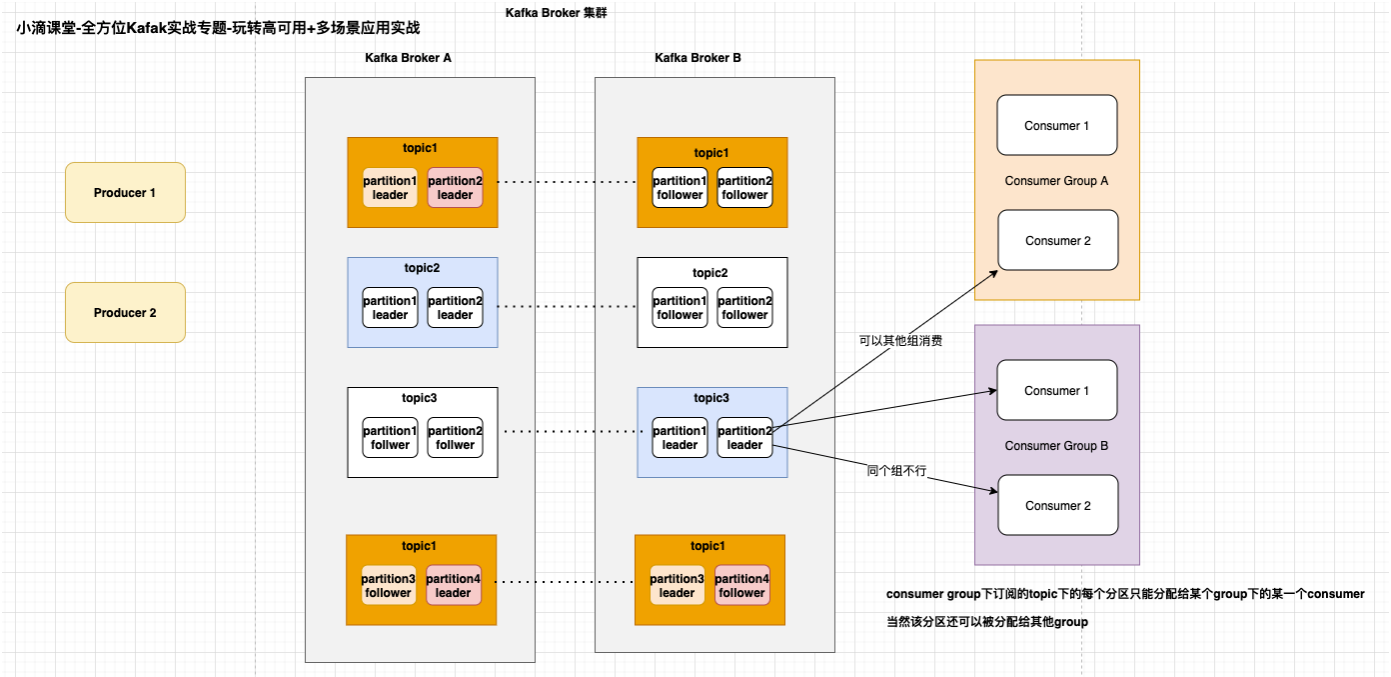
- 结论：
  - 分布式系统中P,肯定要满足，所以只能在CA中二选一
  - 没有最好的选择，最好的选择是根据业务场景来进行架构设计
  - CP： 适合支付、交易类，要求数据强一致性，宁可业务不可用，也不能出现脏数据
  - AP: 互联网业务，比如信息流架构，不要求数据强一致，更想要服务可用

第3集 Kafka数据可靠性保证原理之副本Replica+ACK介绍《上》

简介： Kafka数据可靠性保证原理之副本机制Replica介绍《上》

- 背景
  - Kafka之间副本数据同步是怎样的？一致性怎么保证，数据怎样保证不丢失呢

- kafka的副本（replica）
  - topic可以设置有N个副本, 副本数最好要小于broker的数量
  - 每个分区有1个leader和0到多个follower，我们把多个replica分为Learder replica和follower replica



- 生产者发送数据流程
  - 保证producer 发送到指定的 topic， topic 的每个 partition 收到producer 发送的数据后
  - 需要向 producer 发送 ack 确认收到，如果producer 收到 ack， 就会进行下一轮的发送否则重新发送数据

- 副本数据同步机制
  - 当producer在向partition中写数据时，根据ack机制，默认ack=1，只会向leader中写入数据
  - 然后leader中的数据会复制到其他的replica中，follower会周期性的从leader中pull数据，
  - 对于数据的读写操作都在leader replica中，follower副本只是当leader副本挂了后才重新选取leader，，，，follower并不向外提供服务，假如还没同步完成，leader副本就宕机了，怎么办?

第4集 Kafka数据可靠性保证原理之副本Replica+ACK介绍《下》

简介： Kafka数据可靠性保证原理之副本机制Replica介绍《下》

- 问题点：Partition什么时间发送ack确认机制（要追求高吞吐量，那么就要放弃可靠性）

    - 当producer向leader发送数据时，可以通过request.required.acks参数来设置数据可靠性的级别

    - 副本数据同步策略 , ack有3个可选值，分别是0, 1，all。
        - ack=0
            - producer发送一次就不再发送了，不管是否发送成功
            - 发送出去的消息还在半路，或者还没写入磁盘，Partition Leader所在Broker就直接挂了，客户端认为消息发送成功了，此时就会导致这条消息就丢失

        - ack=1(默认)
            - 只要Partition Leader接收到消息而且写入【本地磁

盘】，就认为成功了，不管他其他的Follower有没有同步过去这条消息了

- 问题：万一Partition Leader刚刚接收到消息，Follower还没来得及同步过去，结果Leader所在的broker宕机了

- ack= all（即-1）
  - producer只有收到分区内所有副本的成功写入全部落盘的通知才认为推送消息成功

  - 备注：leader会维持一个与其保持同步的replica集合，该集合就是ISR，leader副本也在isr里面
  - 问题一：如果在follower同步完成后，broker发送ack之前，leader发生故障，那么会造成数据重复
    - 数据发送到leader后 ，部分ISR的副本同步，leader此时挂掉。比如follower1和follower2都有可能变成新的leader, producer端会得到返回异常，producer端会重新发送数据，数据可能会重复

  - 问题二：acks=all 就可以代表数据一定不会丢失了吗
    - Partition只有一个副本，也就是一个Leader，任何Follower都没有
    - 接收完消息后宕机，也会导致数据丢失，acks=all，

必须跟ISR列表里至少有2个以上的副本配合使用

- 在设置request.required.acks=-1的同时，也要min.insync.replicas这个参数设定 ISR中的最小副本数是多少，默认值为1，改为 >=2，如果ISR中的副本数少于min.insync.replicas配置的数量时，客户端会返回异常

第5集 Kafka的in-sync replica set机制讲解

简介： Kafka数据可靠性保证原理之ISR机制讲解

- 什么是ISR (**in-sync replica set**)
  - leader会维持一个与其保持同步的replica集合，该集合就是ISR，每一个leader partition都有一个ISR，leader动态维护，要保证kafka不丢失message，就要保证ISR这组集合存活（至少有一个存活），并且消息commit成功
  - Partition leader 保持同步的 Partition Follower 集合, 当 ISR 中的Partition Follower 完成数据的同步之后，就会给 leader

发送 ack
    - 如果Partition follower长时间(replica.lag.time.max.ms) 未向leader同步数据，则该Partition Follower将被踢出ISR
    - Partition Leader 发生故障之后，就会从 ISR 中选举新的Partition Leader。
- OSR （out-of-sync-replica set）
- 与leader副本分区 同步滞后过多的副本集合
- AR（Assign Replicas）
- 分区中所有副本统称为AR

第6集 Kafka的HighWatermark的作用你知道多少

简介： Kafka的HighWatermark的作用你知道多少

- 背景 broker故障后

- ACK保障了【生产者】的投递可靠性
- partition的多副本保障了【消息存储】的可靠性
- hw的作用是啥?
- 备注：重复消费问题需要消费者自己处理

- HW作用：保证消费数据的一致性和副本数据的一致性

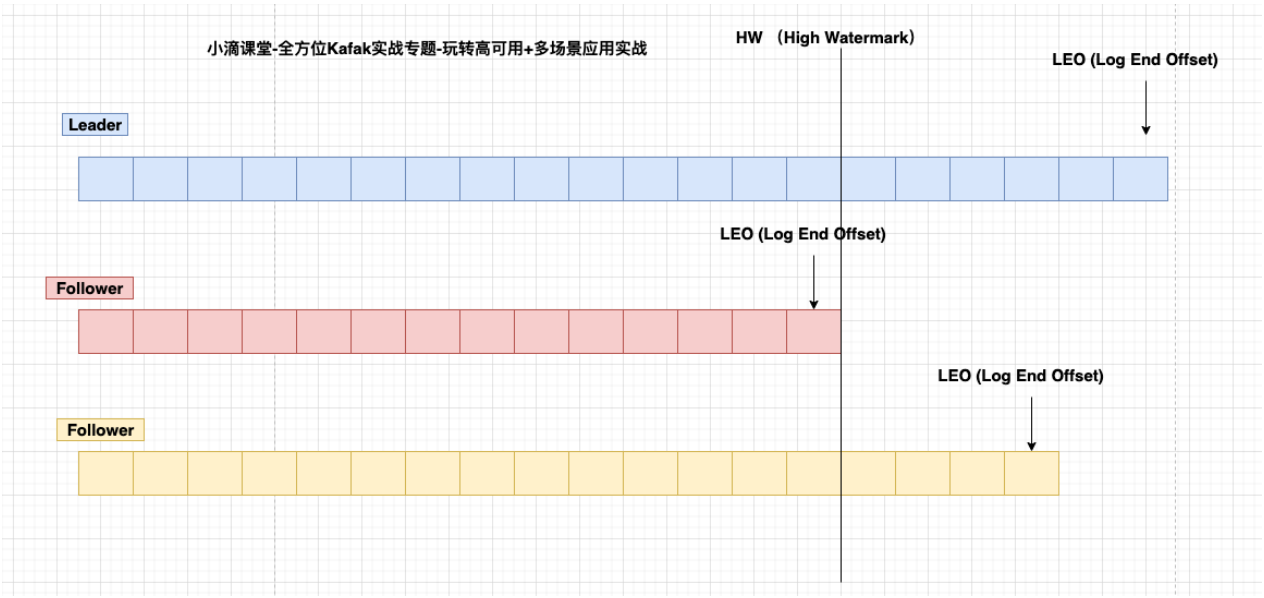
兜底的offect

假设没有HW,消费者消费leader到15，下面消费者应该消费16。

此时leader挂掉,选下面某个follower为leader，此时消费者找新leader消费数据，发现新Leader没有16数据，报错。

HW(High Watermark)是所有副本中最小的LEO。

- Follower故障
  - Follower发生故障后会被临时踢出ISR（动态变化），待该follower恢复后，follower会读取本地的磁盘记录的上次的HW，并将该log文件高于HW的部分截取掉，从HW开始向leader进行同步，等该follower的LEO大于等于该Partition的hw，即follower追上leader后，就可以重新加入ISR
- Leader故障
  - Leader发生故障后，会从ISR中选出一个新的leader，为了保证多个副本之间的数据一致性，其余的follower会先将各自的log文件高于hw的部分截掉（新leader自己不会截掉），然后从新的leader同步数据



小滴课堂 愿景："让编程不再难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第九章 高级篇-kafka高可用集群和高性能讲解

第1集 Kafka高可用集群搭建节点需求规划

简介： Kafka高可用集群搭建节点需求规划

- 注意
  - 没那么多机器，采用伪集群方式搭建（端口号区分）
  - zookeeper部署3个节点
    - 2181
    - 2182
    - 2183
  - kafka部署3个节点
    - 9092
    - 9093
    - 9094
- 网络安全组记得开放端口

第2集 Kafka高可用集群之zookeeper集群环境准备

简介： Kafka高可用集群之zookeeper集群搭建环境准备

- zookeeper节点端口
  - 2181
  - 2182
  - 2183
- cp -r 复制zk节点
- 修改配置zoo.cfg

```
#客户端端口
clientPort=2181

#数据存储路径
dataDir=/tmp/zookeeper/2181

#修改AdminServer的端口：
admin.serverPort=8888
```

- dataDir对应目录下分别创建myid文件，内容对应1、2、3

```
cd /tmp/zookeeper/2183
echo 1 > myid
```

- 配置集群

```
# server.服务器id=服务器IP地址:服务器直接通信端口:服务器之间选举投票端口

server.1=127.0.0.1:2881:3881
server.2=127.0.0.1:2882:3882
server.3=127.0.0.1:2883:3883
```

第3集 Kafka高可用集群之zookeeper集群搭建实战

简介： Kafka高可用集群之zookeeper集群搭建实战

- zk命令

```
#启动zk
./zkServer.sh  start

#查看节点状态
./zkServer.sh status

#停止节点
./zkServer.sh stop
```

第4集 Kafka高可用集群搭建-环境准备

简介： Kafka高可用集群搭建-环境准备

- 伪集群搭建，3个节点同个机器端口区分
  - 9092
  - 9093

    - 9094
- 配置

```
#内网中使用，内网部署 kafka 集群只需要用到 listeners，内外网需要作区分时 才需要用到advertised.listeners
listeners=PLAINTEXT://172.18.123.229:9092

advertised.listeners=PLAINTEXT://112.74.55.160:9092

#每个节点编号1、2、3
broker.id=1

#端口
port=9092

#配置3个
log.dirs=/tmp/kafka-logs-1

#zk地址
zookeeper.connect=localhost:2181,localhost:2182,localhost:2183
```

第5集 Kafka高可用集群搭建实战+SpringBoot项目测试

简介： Kafka高可用集群之多个Kafka节点搭建实战

- 启动Kafka实战

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

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

- 创建topic

```
./kafka-topics.sh --create --zookeeper 192.168.60.6:
2181,192.168.60.6:2182,192.168.60.6: 2183 --
replication-factor 3 --partitions 6 --topic xdclass-
cluster-topic
```

- SpringBoot项目测试
  - 连接zookeeper集群
  - 创建topic
  - 查看topic详情
  - 发送消息

第6集 Kafka高可用集群搭建实战-守护进程方式启动

简介： Kafka高可用集群搭建实战-守护进程方式启动

- 守护进程

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

第7集 【重点】Kafka的中的日志数据清理你知道多少

简介： Kafka的中的日志数据清理你知道多少

- Kafka将数据持久化到了硬盘上，为了控制磁盘容量，需要对过去的消息进行清理
- 问题：如果让你去设计这个日志删除策略，你会怎么设计？【原理思想】很重要的体现，下面是kafka答案
    - 内部有个定时任务检测删除日志，默认是5分钟

log.retention.check.interval.ms
    - 支持配置策略对数据清理
    - 根据segment单位进行定期清理
- 启用cleaner
    - log.cleaner.enable=true
    - log.cleaner.threads = 2 (清理线程数配置)
- 日志删除
    - log.cleanup.policy=delete

```
#清理超过指定时间的消息,默认是168小时，7天,
#还有log.retention.ms, log.retention.minutes,
log.retention.hours，优先级高到低
log.retention.hours=168

#超过指定大小后，删除旧的消息，下面是1G的字节数，-1就是没限制
log.retention.bytes=1073741824

还有基于日志起始位移（log start offset），未来社区还有更多
```

    - 基于【时间删除】 日志说明

配置了7天后删除，那7天如何确定呢?

每个日志段文件都维护一个最大时间戳字段，每次日志段写入新的消息时，都会更新该字段

一个日志段segment写满了被切分之后，就不再接收任何新的消息，最大时间戳字段的值也将保持不变

kafka通过将当前时间与该最大时间戳字段进行比较，从而来判定是否过期

- 基于【大小超过阈值】 删除日志 说明

假设日志段大小是500MB，当前分区共有4个日志段文件，大小分别是500MB，500MB，500MB和10MB

10MB那个文件就是active日志段。

此时该分区总的日志大小是3*500MB+10MB=1500MB+10MB

如果阈值设置为1500MB，那么超出阈值的部分就是10MB，小于日志段大小500MB，故Kafka不会执行任何删除操作，即使总大小已经超过了阈值;

如果阈值设置为1000MB，那么超过阈值的部分就是500MB+10MB > 500MB，此时Kafka会删除最老的那个日志段文件

注意：超过阈值的部分必须要大于一个日志段的大小

- log.retention.bytes和log.retention.minutes任意一个达到要

求，都会执行删除

- 日志压缩
  - log.cleanup.policy=compact 启用压缩策略
  - 按照消息key进行整理，有相同key不同value值，只保留最后一个

第8集 【重点】Kafka的高性能原理分析-ZeroCopy

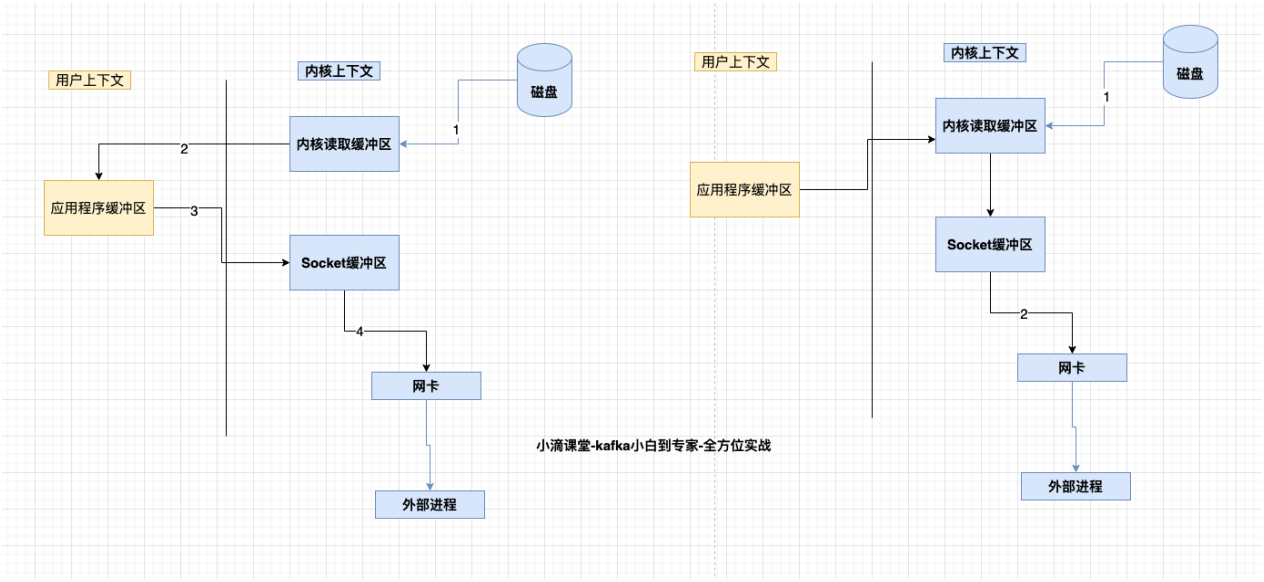
简介： Kafka的高性能原理分析归纳ZeroCopy

- 零拷贝ZeroCopy（SendFile）
  - 例子：将一个File读取并发送出去（Linux有两个上下文，内核态，用户态）
    - File文件的经历了4次copy
      - 调用read,将文件拷贝到了kernel内核态
      - CPU控制 kernel态的数据copy到用户态
      - 调用write时，user态下的内容会copy到内核态的

socket的buffer中

- 最后将内核态socket buffer的数据copy到网卡设备中传送

- 缺点：增加了上下文切换、浪费了2次无效拷贝(即步骤2和3)



- ZeroCopy：
  - 请求kernel直接把disk的data传输给socket，而不是通过应用程序传输。Zero copy大大提高了应用程序的性能，减少不必要的内核缓冲区跟用户缓冲区间的拷贝，从而减少CPU的开销和减少了kernel和user模式的上下文切换，达到性能的提升
  - 对应零拷贝技术有mmap及sendfile
    - mmap:小文件传输快
    - sendfile:大文件传输比mmap快
  - 应用：Kafka、Netty、RocketMQ等都采用了零拷贝技术

第9集 【重点】Kafka的高性能原理分析归纳总结

简介： Kafka的高性能原理分析归纳总结

- kafka高性能
  - 存储模型，topic多分区，每个分区多segment段
  - index索引文件查找，利用分段和稀疏索引
  - 磁盘顺序写入
  - 异步操作少阻塞sender和main线程，批量操作(batch)
  - 页缓存Page cache，没利用JVM内存，因为容易GC影响性能
  - 零拷贝ZeroCopy（SendFile）

第十章 SpringBoot项目整合Spring-kafka和事务消息实战

第1集 Springboot项目整合spring-kafka依赖发送消息

简介： Springboot项目整合spring-kafka依赖包配置

- 添加pom文件

```
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka</artifactId>
    </dependency>
```

- 配置文件修改增加生产者信息

```
spring:
  kafka:

```

```
    bootstrap-servers:
112.74.55.160:9092,112.74.55.160:9093,112.74.55.160:
9094
    producer:
      # 消息重发的次数。
      retries: 0
      #一个批次可以使用的内存大小
      batch-size: 16384
      # 设置生产者内存缓冲区的大小。
      buffer-memory: 33554432
      # 键的序列化方式
      key-serializer:
org.apache.kafka.common.serialization.StringSerializ
er
      # 值的序列化方式
      value-serializer:
org.apache.kafka.common.serialization.StringSerializ
er
      acks: all
```

- 发送消息

```
    private static final String TOPIC_NAME =
"user.register.topic";

    @Autowired
    private KafkaTemplate<String, Object>
kafkaTemplate;
```

```
    /**
     * 发送消息
     * @param phone
     */
    @GetMapping("/api/user/{phone}")
    public void sendMessage1(@PathVariable("phone")
String phone) {
        kafkaTemplate.send(TOPIC_NAME,
phone).addCallback(success -> {
            // 消息发送到的topic
            String topic =
success.getRecordMetadata().topic();
            // 消息发送到的分区
            int partition =
success.getRecordMetadata().partition();
            // 消息在分区内的offset
            long offset =
success.getRecordMetadata().offset();
            System.out.println("发送消息成功:" + topic
+ "-" + partition + "-" + offset);

        }, failure -> {
            System.out.println("发送消息失败:" +
failure.getMessage());
        });

    }
```

第2集 Springboot项目整合spring-kafka监听消费消息

简介： Springboot项目整合spring-kafka监听消费消息

- 配置文件修改增加消费者信息

```
    consumer:
      # 自动提交的时间间隔 在spring boot 2.X 版本是值的类型为Duration 需要符合特定的格式，如1S,1M,2H,5D
      auto-commit-interval: 1S

      # 该属性指定了消费者在读取一个没有偏移量的分区或者偏移量无效的情况下该作何处理：
      auto-offset-reset: earliest
```

```
      # 是否自动提交偏移量，默认值是true,为了避免出现重复数据和数据丢失，可以把它设置为false,然后手动提交偏移量
      enable-auto-commit: false

      # 键的反序列化方式
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      # 值的反序列化方式
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

    listener:
      #手工ack，调用ack后立刻提交offset
      ack-mode: manual_immediate
      #容器运行的线程数
      concurrency: 4
```

- 代码编写

```
@Component
public class MQListener {



    /**
     *  消费监听
     * @param record
```

```
     */
    @KafkaListener(topics =
{"user.register.topic"},groupId = "xdlcass-test-gp")
    public void onMessage1(ConsumerRecord<?, ?>
record, Acknowledgment ack,
@Header(KafkaHeaders.RECEIVED_TOPIC) String topic){
        // 打印出消息内容
        System.out.println("消费：topic()+"-
"+record.partition()+"-"+record.value());

        ack.acknowledge();
    }


}
```

第3集 Kafka事务消息-整合SpringBoot实战

简介： Kafka事务消息-整合SpringBoot实战

- Kafka 从 0.11 版本开始引入了事务支持

    - 事务可以保证对多个分区写入操作的原子性
    - 操作的原子性是指多个操作要么全部成功，要么全部失败，不存在部分成功、部分失败的可能
- 配置

```
spring:
  kafka:
    bootstrap-servers:
112.74.55.160:9092,112.74.55.160:9093,112.74.55.160:
9094
    producer:
      # 消息重发的次数。 配置事务的话：如果用户显式地指定了
retries 参数，那么这个参数的值必须大于0
      #retries: 1
      #一个批次可以使用的内存大小
      batch-size: 16384
      # 设置生产者内存缓冲区的大小。
      buffer-memory: 33554432
      # 键的序列化方式
      key-serializer:
org.apache.kafka.common.serialization.StringSerializ
er
      # 值的序列化方式
      value-serializer:
org.apache.kafka.common.serialization.StringSerializ
er
      #配置事务的话：如果用户显式地指定了 acks 参数，那么这
个参数的值必须-1 all
      #acks: all
```

```
      #事务id
      transaction-id-prefix: xdclass-tran



    consumer:
      # 自动提交的时间间隔 在spring boot 2.X 版本是值的类型为Duration 需要符合特定的格式，如1S,1M,2H,5D
      auto-commit-interval: 1S

      # 该属性指定了消费者在读取一个没有偏移量的分区或者偏移量无效的情况下该作何处理：
      auto-offset-reset: earliest

      # 是否自动提交偏移量，默认值是true,为了避免出现重复数据和数据丢失，可以把它设置为false,然后手动提交偏移量
      enable-auto-commit: false

      # 键的反序列化方式
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      # 值的反序列化方式
      value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

    listener:
```

```
        # 在侦听器容器中运行的线程数。
        concurrency: 4
        #listner负责ack，手动调用
Acknowledgment.acknowledge()后立即提交
        ack-mode: manual_immediate
        #避免出现主题未创建报错
        missing-topics-fatal: false
```

- SpringBoot代码编写

```
    /**
     * 注解方式的事务
     * @param i
     */
    @GetMapping("/kafka/transaction1")
    @Transactional(rollbackFor =
RuntimeException.class)
    public void sendMessage1(int i) {

        kafkaTemplate.send(TOPIC_NAME, "这个是事务里面
的消息：1  i="+i);
            if (i == 0) {
                throw new RuntimeException("fail");
            }
```

```
        kafkaTemplate.send(TOPIC_NAME, "这个是事务里面
的消息：2  i="+i);

    }


    /**
     * 声明式事务支持
     * @param i
     */
    @GetMapping("/kafka/transaction2")
    public void sendMessage2(int i) {

        kafkaTemplate.executeInTransaction(new
KafkaOperations.OperationsCallback() {
            @Override
            public Object
doInOperations(KafkaOperations kafkaOperations) {
                kafkaOperations.send(TOPIC_NAME,"这个
是事务里面的消息：1  i="+i);
                if(i==0)
                {
                    throw new
RuntimeException("input is error");
                }
                kafkaOperations.send(TOPIC_NAME,"这个
是事务里面的消息：2  i="+i);
                return true;
            }
```

```
        });

    }
```

第4集 关于 Kafka的其他特性和技术选型建议

简介： 关于 Kafka的其他特性和技术选型建议

- Kafka很多内容，但是不一定都要学，看自己的需求，有些功能是比较鸡肋的
    - 比如kafka streams 虽然轻量级
        - 但是与Kafka 紧密联系，无法在没有Kafka 的场景下使用
        - 相较于实时计算工具Spark Streaming、Flink等，kafka streams不适用于大型业务场景

    - 有些功能的话虽然kafka有，但还是用更好的工具比较好，且技术更新换代快，掌握设计思想才主要
    - 更主要的是没有万能的框架，技术选型多数都是基于【业务需求】出发，选出最合适的技术
    - kafka/rabbitmq/rocketmq

- 学习技术延伸
    - 项目大课训练营
        - 地址：https://detail.tmall.com/item.htm?id=634842400121
        - 联系店铺客服，看了这个kafka课程后，有特殊折扣哦
    - 海量数据处理大课训练营

愿景："让编程不再难学，让技术与生活更加有趣"

更多架构课程请访问 xdclass.net

第十一章 分布式流处理平台-Kafka小白到专家之路专题课程总结

第1集 分布式流处理平台-Kafka小白到专家之路和大数据技术栈推荐

简介：布式流处理平台-Kafka小白到专家之路专题课程总结

- MQ中间件作用，Kafka介绍安装部署
- SpringBoot整合Kafka多个模块Admin/Producer/ConsumerAPI实战
- Kafka生产者消费者分区策略、日志管理、Offset管理、副本原理和可靠性投递等
- 高可用集群搭建实战+Springkafka整合等

- 同比的MQ产品有很多，业界主要是三大MQ框架
  - RabbitMQ
  - RocketMQ

- Kafka

- 大数据技术栈
  - Javase【上线】
  - Idea+maven+git【上线】
  - Linux【上线】
  - Mysql【上线】
  - Javaweb【上线】
  - Springboot【上线】
  - Redis6.X【上线】
  - Kakfa【上线】
  - ElasticSearch【上线】
  - ELK体系
  - Hadoop生态
  - Zookeeper【上线】
  - Flink
  - Hive
  - Flume
  - Shell【上线】
  - Scala
  - Spark生态
  - Docker【上线】
  - Hbase
  - Zabbix监控
  - Azkaban任务调度

- Canal
- Python【上线】
- ClickHouse
- 阿里云ODPS+DataV
- SpringCloud微服务【上线】
- 多个大数据项目实战
- 企业数据中台+DaaS+PaaS

- 大课训练营 -笔记查看（不对外提供，【原创资料】）
  - 给个学习建议，大课推出的目的，也是很多同学需要这个
  - 项目大课训练营
    - https://detail.tmall.com/item.htm?id=634842400121
  - 海量数据分库分表大课训练营
  - 大数据中台建设大课训练营
  - ...更多

第2集 架构师成长路线- 如何选择IT技术人的职场充电站

简介：小滴课堂架构师学习路线和大课训练营介绍

- 小滴课堂永久会员 & 小滴课堂年度会员
  - 适合IT后端人员，零基础就业、在职充电、架构师学习路线-专题技术方向
    - 包括业界主流技术栈，前端、后端、测试、架构等等课程，接近上百套视频，深度+广度

        - 包括 从零基础到就业班，高级工程师、技术负责人、架构师技术栈 全套学习路线
        - 永久会员可以观看全部专题IT技术，作为一个终生学习平台，每个月更新多套视频
        - 如何获取最新路线图，有你想学的多数主流技术栈，持续更新！！！
    - 学后水平：一线城市 15~25k
    - 适合人员：校招、应届生、培训机构出、工作1~10年的同学都适合

    - 学到技术，涨薪是关键，付出肯定会有收获，未来 一定是持续学习的人
    - 我也在学习每年参加各种技术沙龙和内部分享会投 入超过10万+，但是我认为是值的，且带来的收益更 大
- 大课综合训练营
    - 适合用于专项拔高，适合有一定工作经验的同学，架构师方向发展的训练营，包括多个方向
        - 综合项目大课训练营

    - https://detail.tmall.com/item.htm?id=634842400121
  - 海量数据分库分表大课训练营
  - 架构解决方案大课训练营
  - 全链路性能优化大课训练营
  - 安全攻防大课训练营
  - 数据分析大课训练营
  - 算法刷题大课训练营

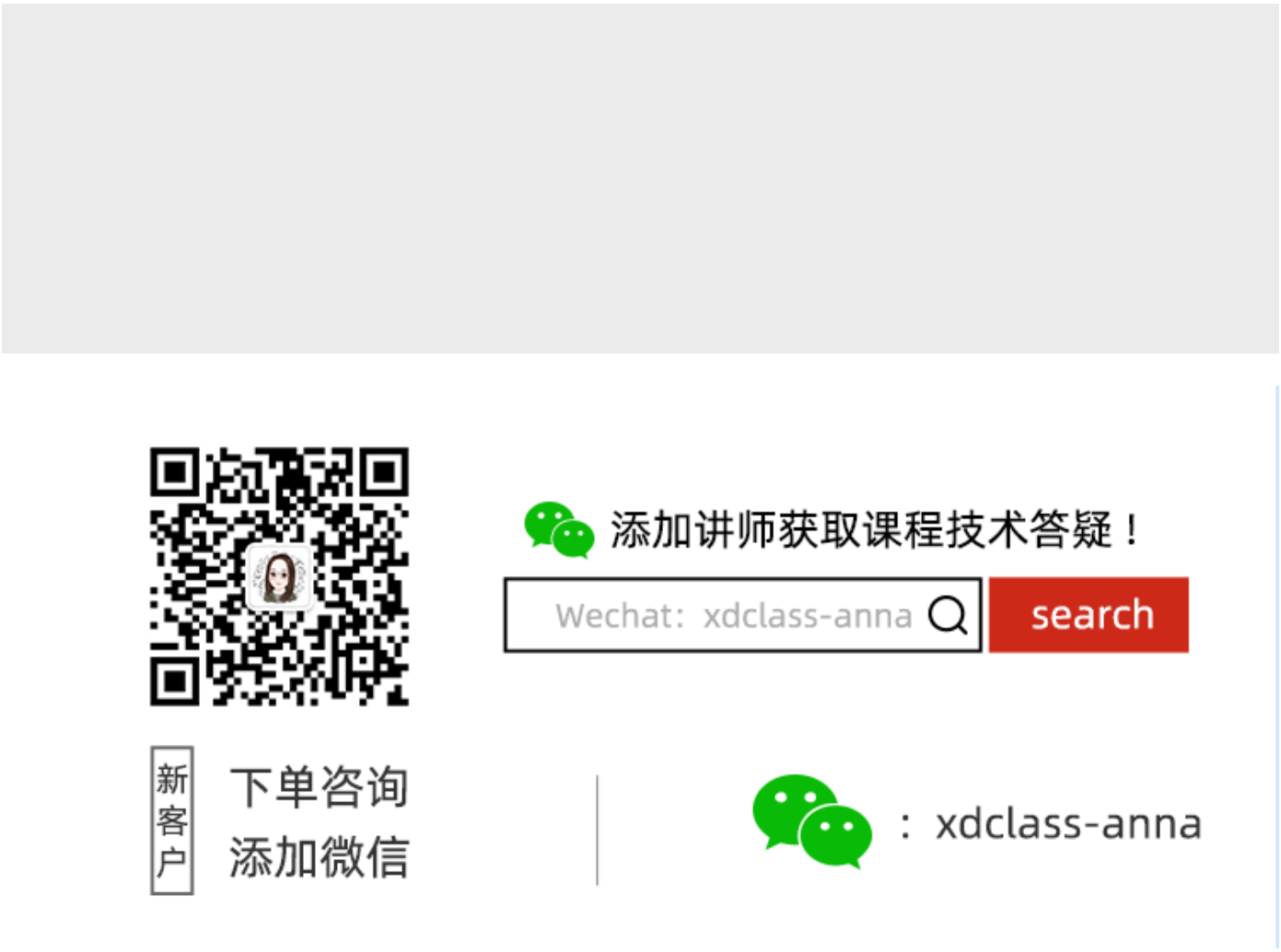
- 直播小班课（留意官网即可或者联系我即可）

- 技术人的导航站（涵盖我们主流的技术网站、社区、工具等等，即将上线）
  - 地址：open1024.com
  - 地址：xdclass.net

二当家小D - (可以加我好友)

广东 广州

扫一扫上面的二维码图案，加我微信



小滴课堂，愿景：让编程不在难学，让技术与生活更加有趣

相信我们，这个是可以让你学习更加轻松的平台，里面的课程绝对会让你技术不断提升

欢迎加小D讲师的微信： xdclass-lw

我们官方网站：https://xdclass.net

千人IT技术交流QQ群： 718617859

重点来啦：加讲师微信 免费赠送你干货文档大集合，包含前端，后端，测试，大数据，运维主流技术文档（持续更新）

https://mp.weixin.qq.com/s/qYnjcDYGFDQorWmSfE7lpQ