### 1、说说你对kafka的理解

Kafka是一个分布式的消息发布订阅系统，使数据解耦，可用于应用之间通信。主要概念有：生产者，消费者(消费组)，Broker,Topic ，分区,offset（消息位移）等。消息的每一个分类可以定义成一个topic，topic里面可以分多个分区，每个分区最多对应同一个消费组中的一个消费组，所以 分区数>=消费组的消费者数目，每个消息都保存在实现分配好的分区中且由一个唯一的偏移量，所以消息不会过期，我们可以通过指定偏移量去消费获取指定的消息。每个消费组消费的消息位移由一个叫\_consumers\_offset 的topic维护，所以每个消费组消费同一个topic里分区的消息，不受干扰。

目前，我们项目中主要应用于：

1. 交易状态的异步终态通知
2. 应用中缓存数据的修改
3. 平台用户的行为跟踪、记录。

### 2、传统的消息传递机制

1、队列：消息存储在线程安全的任务队列里面，由线程池里面的消费者一条一条去消费，一条消息一般由一个消费者消费

2、发布-订阅:消息是通过广播的方式发送给所有消费者

### 3、为什么使用kafka，优势在哪里

1、快速: Kafka代理(broker)可以处理成千上万的客户端，broker和消费者的状态由zk管理。

2、可伸缩:在一组机器上对数据进行分区和简化，以支持更大的数据

3、持久:消息是持久性的，并在集群中进行复制，以防止数据丢失。

4、设计:它提供了容错保证和持久性

1. 处理优势:数据解耦，异步执行，减少系统响应时间。

### 4、kafka中的broker是什么

Broker是kafka的代理服务器，是存储Topic的服务器。

Broker在集群模式中由几个比较重要的属性

broker.id

每一个Kafka的broker都有一个整数的标识。我们设置broker.id来标识它。默认这个整数是0。这个整数必须是在一个集群中是唯一的，且必须>=0。推荐该值能对应上broker所在的主机名。

port

该值是监听TCP的端口。默认是配置9092。可以设置可用的任意端口，不过需要注意的是如果该值小于1024，Kafka必须以root权限启动。使用root权限启动并不推荐。

zookeeper.connect

连接zk的配置，默认配置localhost:2181。存放broker的元数据。配置值格式是：hostname:port/path，如果多个以分号隔开。

Hostname:zk的hostname或者ip地址

port:客户端的与zk通讯的端口

path:一个可选的zk路径参数。kafka集群如果使用了chroot环境，需要设置该参数。如果没有配置，使用root路径。如果path配置了，但是kafka集群没有使用chroot环境，那么该path路径会在broker启动的时候创建。（推荐使用该配置，可以与其他kafka集群共用一个zk，且没有冲突）

log.dirs

Kafka会保留消息到磁盘，并且这些日志片段会按目录区分存放在log.dirs配置的路径，多个配置使用英文逗号隔开

num.partitions

topic创建多少分区，默认是1，分区一旦设置，只能增加不能减少

### 5、kafka集群中的zookeeper存在的意义

Zookeepr是一个分布式的开源的分布式应用协调服务，是一个为分布式应用提供一致性的软件。

在kafka中，broker consumer 需要连接zookeeper来协调服务，provider直接连接到zookeeper.在kafka中，zookeeper的作用主要作用如下:

1. broker的注册

如果是在集群模式下，每个broker之间是相互独立，zookeeper将各个broker统一管理起来。建立服务节点.每个broker启动的时候都会到zk上注册自己。

1. topic的注册

在kafka中，topic可能由于性能考虑，会分成多个分区分布在不同的broker上，各分区和broker的联系是由zookeeper维护的。

1. 生产者 和 消费者的负载均衡

Zookeeper负责控制生产者如何将详细均衡的发送到不同的broker上。

Zookeeper负责消费者消费存储在broker各分区的消息的分配方案

1. 消费者的注册

注册到消费者分组。每个消费者服务器启动时，都会到Zookeeper的指定节点下创建一个属于自己的消费者节点，例如/consumers/[group\_id]/ids/[consumer\_id]，完成节点创建后，消费者就会将自己订阅的Topic信息写入该临时节点。

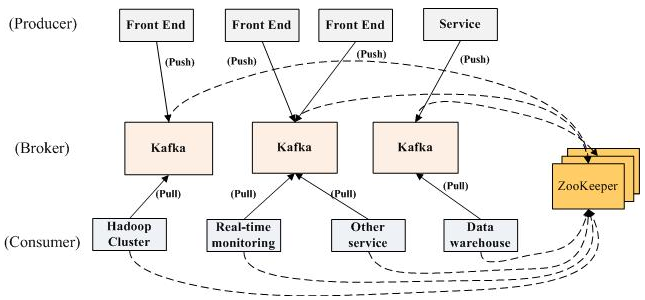
5、kafka的reblance

对 消费者分组 中的 消费者 的变化注册监听。每个 消费者 都需要关注所属 消费者分组 中其他消费者服务器的变化情况，即对/consumers/[group\_id]/ids节点注册子节点变化的Watcher监听，一旦发现消费者新增或减少，就触发消费者的rebalance

对Broker服务器变化注册监听。消费者需要对/broker/ids/[0-N]中的节点进行监听，如果发现Broker服务器列表发生变化，那么就根据具体情况来决定是否需要进行消费者rebalance

### kafka的消费者如何消费消息的

kafka生产者发送的消息是存储在broker(服务器)的Topic上的，在集群模式下，一个Topic的消息会存储在多个分区之中，分区会分散在多个不同的broker上，由zookeeper进行分布式协调管理。

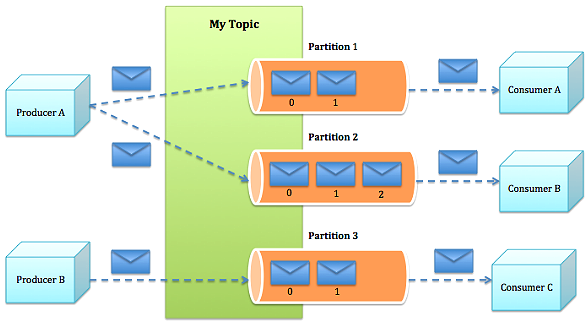


Kafka为了使系统有更高的性能，可以将消息分类来保存在不同类别下，这个类别就是topic。每个topic可以由多个分区组成，提升消息的生产和消费效率。

消费组是kafka 提供的可扩展 具有容错性的消费机制，组内会有多个消费者实例，共用一个group.id,组内所有消费者一起订阅和消费某个topic的所有分区。

需要注意的是一个消费者可以订阅多个topic，但消费组的消费者个数不应该多于订阅topic的最大分区数，会造成消费者实例的浪费。消费组的各消费者实例与订阅的topic的分区的订阅消费关系确认是通过rebalance实现的。

生产者生产的消息发送到topic的分区后，消费者从zookeeper订阅到自己需要的分区后，开始消费消息，如果不存在消息就会发生阻塞，直到有消息发送过来立即唤醒自己开始消费消息，kafka 的消息是存储起来的，消费消息的进度是通过位移(offset)实现的，所以我们可以指定位移量去消费指定的消息。新版本的消息位移是存储再一个名叫 \_consumers\_offset的topic里面。



1、Producer根据指定的partition方法（round-robin、hash等），将消息发布到指定topic的partition里面

2、kafka集群接收到Producer发过来的消息后，将其持久化到硬盘，并保留消息指定时长（可配置），而不关注消息是否被消费。

3、Consumer从kafka集群pull数据，并控制获取消息的offset

提交与位移

Kafka消息的消费进度是通过位移(offset)实现的，位移是会被记录的， 所以不同的消息有不同的位移，消费最新的消息是通过读取kafka消息当前位移(offset)

实现的。所以消费者消费完消息是需要提交位移量的。我们在kafka初始化连接属性中设置为 自动提交 手动指定提交位移 异步提交。 但这样都有可能会导致消费信息完成后 服务端宕机，无法更新位移重复消费的问题。

### 如何构建高性能、高吞吐量的kafka集群

kafka是一个高吞吐量分布式消息系统，并且提供了持久化。

其高性能的有两个重要特点：

1、利用了磁盘连续读写性能远远高于随机读写的特点；

2、并发，将一个topic拆分多个partition。

要充分发挥kafka的性能，就需要满足这两个条件

kafka读写的单位是partition，因此，将一个topic拆分为多个partition可以提高吞吐量。但是，这里有个前提，就是不同partition需 要位于不同的磁盘（可以在同一个机器）。如果多个partition位于同一个磁盘，那么意味着有多个进程同时对一个磁盘的多个文 件进行读写，使得操作系统会对磁盘读写进行频繁调度，也就是破坏了磁盘读写的连续性。

在linkedlin的测试中，每台机器就加载了6个磁盘，并且不做raid，就是为了充分利用多磁盘并发读写，又保证每个磁盘连续读写 的特性。

具体配置上，是将不同磁盘的多个目录配置到broker的log.dirs，例如

log.dirs=/disk1/kafka-logs,/disk2/kafka-logs,/disk3/kafka-logs

kafka会在新建partition的时候，将新partition分布在partition最少的目录上，因此，一般不能将同一个磁盘的多个目录设置到log.dirs

同一个ConsumerGroup内的Consumer和Partition在同一时间内必须保证是一对一的消费关系

任意Partition在某一个时刻只能被一个Consumer Group内的一个Consumer消费(反过来一个Consumer则可以同时消费多个Partition)

Kafka实现高吞吐量的设计:

1、数据磁盘持久化：消息不在内存中cache，直接写入到磁盘，充分利用磁盘的顺序读写性能

2、zero-copy：减少IO操作步骤

3、数据批量发送

4、数据压缩

5、Topic划分为多个partition，提高parallelism

### 8、如何保证kafka消息的可靠性

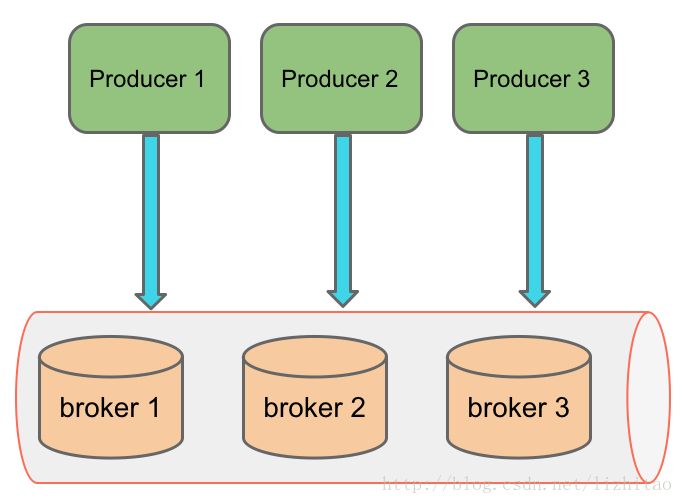
Kafka没有像数据库那样的事务机制，无法保证消息传递百分之百的成功率，由于消费消息是通过offset实现的，所以极端情况下，消息会存在丢失或者重复消费的可能性。

保证kafka消息的可靠性，一般要做到这2点:

1. 消息不会丢失
2. 消息不会重复消费

从这两点看kafka存在不可靠的原因。

**1、Producer发送消息到Broker**



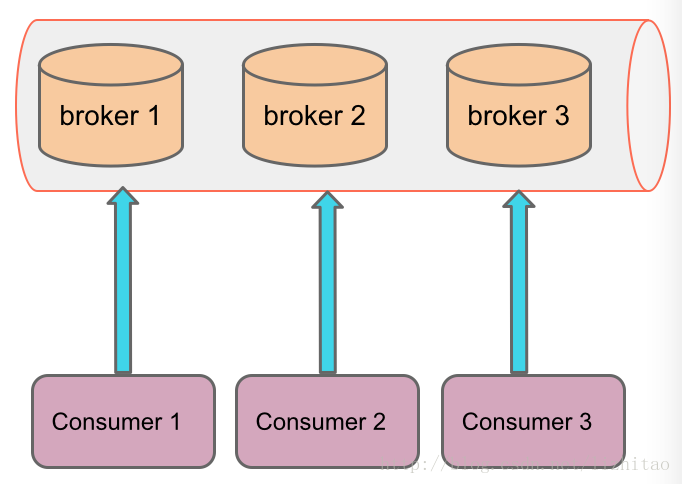
目前生产者发送消息(request.required.acks)有三种方式。

acks = 0: producer不会等待broker发送ack ，因为发送消息网络超时或broker crash (1.Partition的Leader还没有commit消息 2.Leader与Follower数据不同步)，既有可能丢失也可能会重发。

acks = 1: 当leader接收到消息之后发送ack，丢会重发，丢的概率很小。

acks = -1: 当所有的follower都同步消息成功后发送ack. 丢失消息可能性比较低。

**2.Consumer从Broker拉取消息**



Kafka中有两种consumer接口，分别为Low-level API和High-levelAPI

(1). Low-level API SimpleConsumer

这套接口比较复杂的，使用者必须要考虑很多事情，优点就是对Kafka可以有完全的控制。

(2). High-level API ZookeeperConsumerConnector

High-level API使用比较简单，已经封装了对partition和offset的管理，默认是会定期自动commit offset，这样可能会丢数据的，因为consumer可能拿到数据没有处理完crash。 High-level API接口的特点，自动管理，使用简单，但是对Kafka的控制不够灵活。

**3. Broker存储消息**

(1). 对于broker，落盘的数据，除非磁盘坏了，一般不会丢的。

(2). 对于内存脏(没有flush磁盘)数据，broker重启会丢。

可以通过log.flush.interval.messages和log.flush.interval.ms来配置flush间隔，interval大丢的数据多些，小会影响性能。

但在0.8.x版本以后，可以通过replication机制保证数据不丢，代价就是需要更多资源，尤其是磁盘资源，kafka当前支持GZIP和Snappy压缩，来缓解这个问题。

是否使用replication取决于在可靠性和资源代价之间的平衡。

**解决方案分析**

消费端重复消费：很容易解决，建立去重表

消费端丢失数据：也容易解决，关闭自动提交offset，处理完之后受到移位

生产端重复发送：这个不重要，消费端消费之前从去重表中判重就可以

生产端丢失数据：这个是最麻烦的情况

**解决策略：**

1.异步方式缓冲区满了，就阻塞在那，等着缓冲区可用，不能清空缓冲区

2.发送消息之后回调函数，发送成功就发送下一条，发送失败就记在日志中，等着定时脚本来扫描

（发送失败可能并不真的发送失败，只是没收到反馈，定时脚本可能会重发）

**如何保证有序：**

如果有一个发送失败了，后面的就不能继续发了，不然重发的那个肯定乱序了

生产者在收到发送成功的反馈之前，不能发下一条数据，但我感觉生产者是一个流，阻塞生产者感觉业务上不可行，怎么会因为一条消息发出去没收到反馈，就阻塞生产者

同步发送模式：发出消息后，必须阻塞等待收到通知后，才发送下一条消息

异步发送模式：一直往缓冲区写，然后一把写到队列中去

两种都是各有利弊：

同步发送模式虽然吞吐量小，但是发一条收到确认后再发下一条，既能保证不丢失消息，又能保证顺序