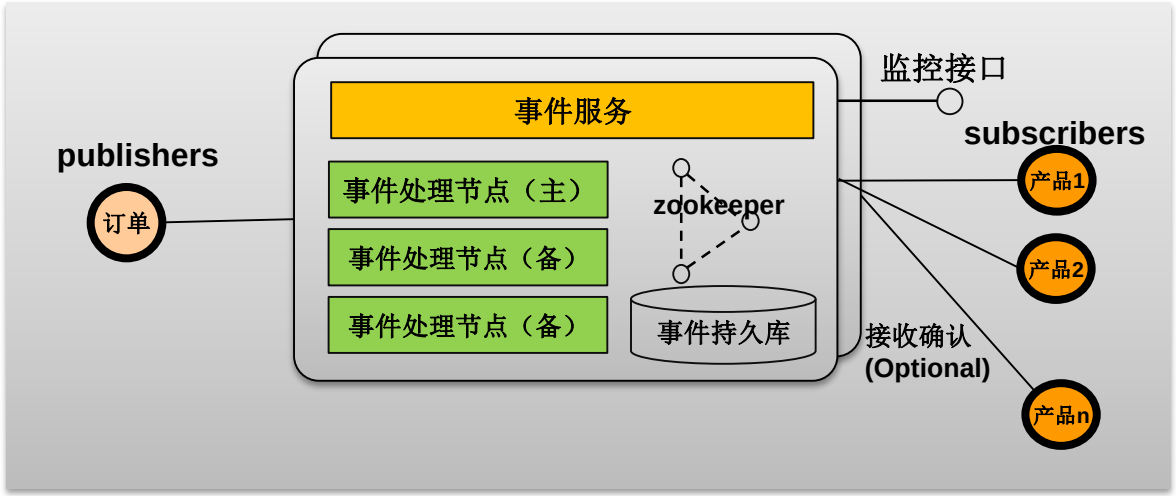
Kafka 课程培训：入门简介

目录

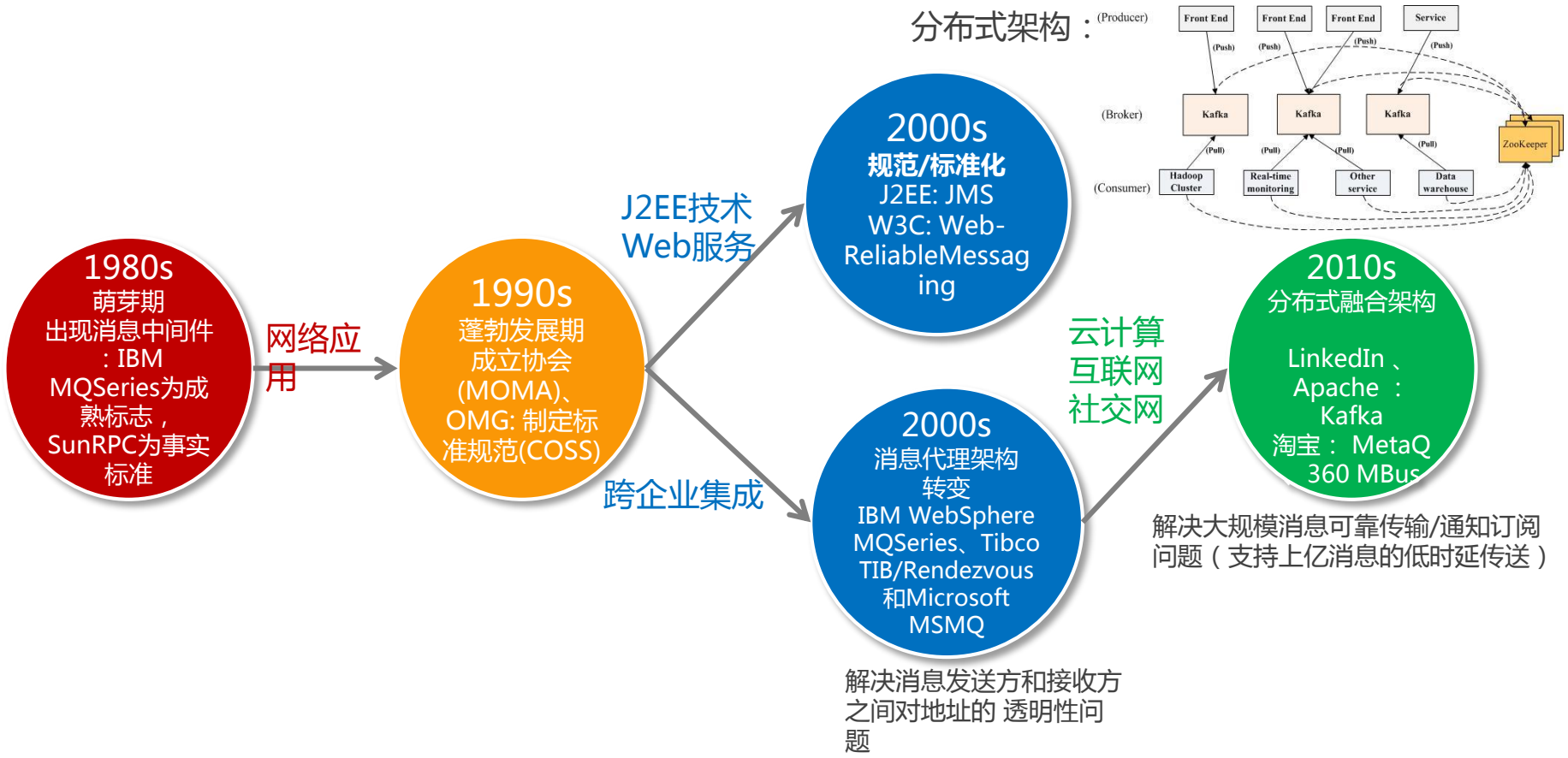
1 为何需要消息服务？为云服务提供统一通信

基于消息中间件构建消息服务，解决云分布式场景的消息通信问题，提供高可靠、高性能（*高并发*、*高吞吐*）、高可扩展的消息管道。

系统解耦	基于发布订阅模型，分布式应用异步解耦，可以增加应用的水平扩展性，增加前端应用快速客户反应能力。
削峰填谷	大促等流量洪流突然来袭时，消息服务可以缓冲突发流量，避免整个系统崩溃。
数据交换	解决跨DC传输，通道安全可靠、服务可用的问题。
异步通知	海量终端接入、高吞吐、低时延实时并发消费。
日志通道	做为重要日志的监控通信管道，将应用日志监控对系统性能影响降到最低。



2 消息中间件的发展历史



消息中间件（message oriented middleware）是指支持与保障分布式应用程序之间同步/异步收发消息的中间件。通过消息中间件，应用程序或组件之间可以进行可靠的异步通讯来降低系统之间的耦合度，从而提高整个系统的可扩展性和可用性。

3 业界分布式MQ横向对比分析

	Kafka	RabbitMQ	RocketMq	Disque	ZeroMQ
社区支持	Apache	Lshift、Vmware、SpringSource	阿里	Pivotal	iMatix
开发语言	Scala	Erlang	JAVA	C	C++
集群管理	支持集群部署：采取zk作为分布式协调部件；内部采取Controller broker中心控制；分区采取leader follower模式，选举leader。	支持集群部署：采取OTP内建集群模式；内嵌分布式no sql数据库Mnesia；镜像模式支持多副本复制；消息的发布与消费都是通过master节点完成。消息的复制采取循环链表，而不是主从复制	支持集群部署：自研的Name Server集群，节点级别的主从设计	支持集群：使用Gossip内建集群，无主设计	不支持集群，用户可以基于zeroMQ接口做二次开发来支撑。
通信协议	不支持AMQP和JMS，自定义消息格式；底层使用NIO-TCP通信。	支持AMQP/MQTT；底层使用OTP-TCP通信	不支持AMQP和JMS，自定义消息格式；底层使用Netty-TCP通信	不支持AMQP和JMS，自定义消息格式，底层使用TCP	不支持AMQP和JMS，底层支持多种传输协议TCP/UDP/IPC/广播等
缓存模型	使用Pagecache缓存消息，从Pagecache落盘到硬盘消息分区：顺序文件，每个分区对应独立的文件。	使用内存队列缓存消息，支撑持久化到硬盘。	与kafka类似采取了文件来存储消息，但是采取了单文件（顺序写，随机读）	使用内存队列缓存消息，支持持久化配置，异步持久化到文件	消息缓存在发送端内存buffer中，不支持持久化，无分区无topic概念
高吞吐	高吞吐：单节点支持10w级别的吞吐量。干净纯粹，抛弃复杂厚重的功能，Pagecache，Sendfile	吞吐较低：相比Kafka低一个数量级	高吞吐：Topic少时吞吐量比kafka弱，大概7~8w左右，topic多的逐步与kafka持平	未发布，数据未知。	高吞吐：比kafka要大，单节点支撑几十万（40w左右）。
高可靠	高可靠：分区多副本复制，消息持久化，消息发送响应确认，消费offset同步提交。	高可靠：镜像模式支持队列多副本复制	高可靠：采取了节点主从设计，同步双写，消息会被同步备份到从	较高：队列有副本设计，通过多写来达到可靠性。无主设计，每个副本都可以单独工作	低，不存在多副本复制，没有高可靠的设计考虑
低时延	采取pull长轮询，pagecache交换数据，时延ms级（5ms左右）	NA	Ms级别，相对kakfa没有特殊的改进	未发布，数据未知。	微妙级时延（30us左右）
高并发	由于采取了文件来存储消息，所以支持的队列数目有限（几千）；	队列数量不受限制，与内存大小相关	采取了单文件设计，相对kafka来说有较大的提升，单机支持5w的队列	内存队列，数目不收限制。使用了linux的epoll机制，server端是单线程	没有broker和分区的概念，客户端基于linux的epoll设计。
轻量化	支持进程级部署，对外依赖zk集群，对cpu和内存消耗不大，但是对硬盘和网络资源消耗巨大，没有限流措施。	支持AMQP，需要Erlang环境，较厚重	与kafka差不多，支持进程部署，对外依赖nameserver集群。	较轻：进程级部署，内建集群，对外无依赖。对内存和网络资源依赖较大	轻量化，没有broker，只是一个通信库。
跨DC	采取多集群镜像方式支持跨DC	采取联邦模式，使用插件Shovel 支持	没看到有特殊支持（阿里有DRC）	暂不支持	不支持
特点	注重性能，是高吞吐的代表作，主要用于大规模日志收集，数据流，事件中心等场景 1、P/S模型：Broker+重Topic设计+分区机制 2、文件缓存机制：PageCache+文件作为消息缓存。 3、外置协调集群：ZK+Controller方式	基于AMQP协议的老牌MQ：功能齐全，使用灵活。主要用于企业级高可靠场景（openstack） 1、P/S模型：Broker+轻Topic（exchange+队列机制） 2、内存缓存机制：使用内存作为消息缓存 3、内建集群：OTP（actor）+Mnesia	参考Kafka的Java仿制版本，主要用于阿里的数据流传递，binlog同步等场景 1、P/S模型：Broker+重Topic设计 2、文件缓存机制 3、外置协调集群：Nameserver	易用性高。轻量级，还未商用。 1、P/S模型：Broker+轻Topic：无分区和exchange概念 2、内存缓存机制：采取内存作为消息缓存。 3、内建集群：Gossip协议	轻量级通信库，用在证券交易，交易平台中。 1、P2P模型：无Broker设计 2、内存缓存机制：采取内存作为消息缓存。 3、无集群设计

Kafka：一种分布式、基于发布/订阅的消息中间件

- **高性能、高吞吐**

  顺序写盘、消息压缩

- **在线扩容**

  分区机制，在线扩容

  副本在线扩容

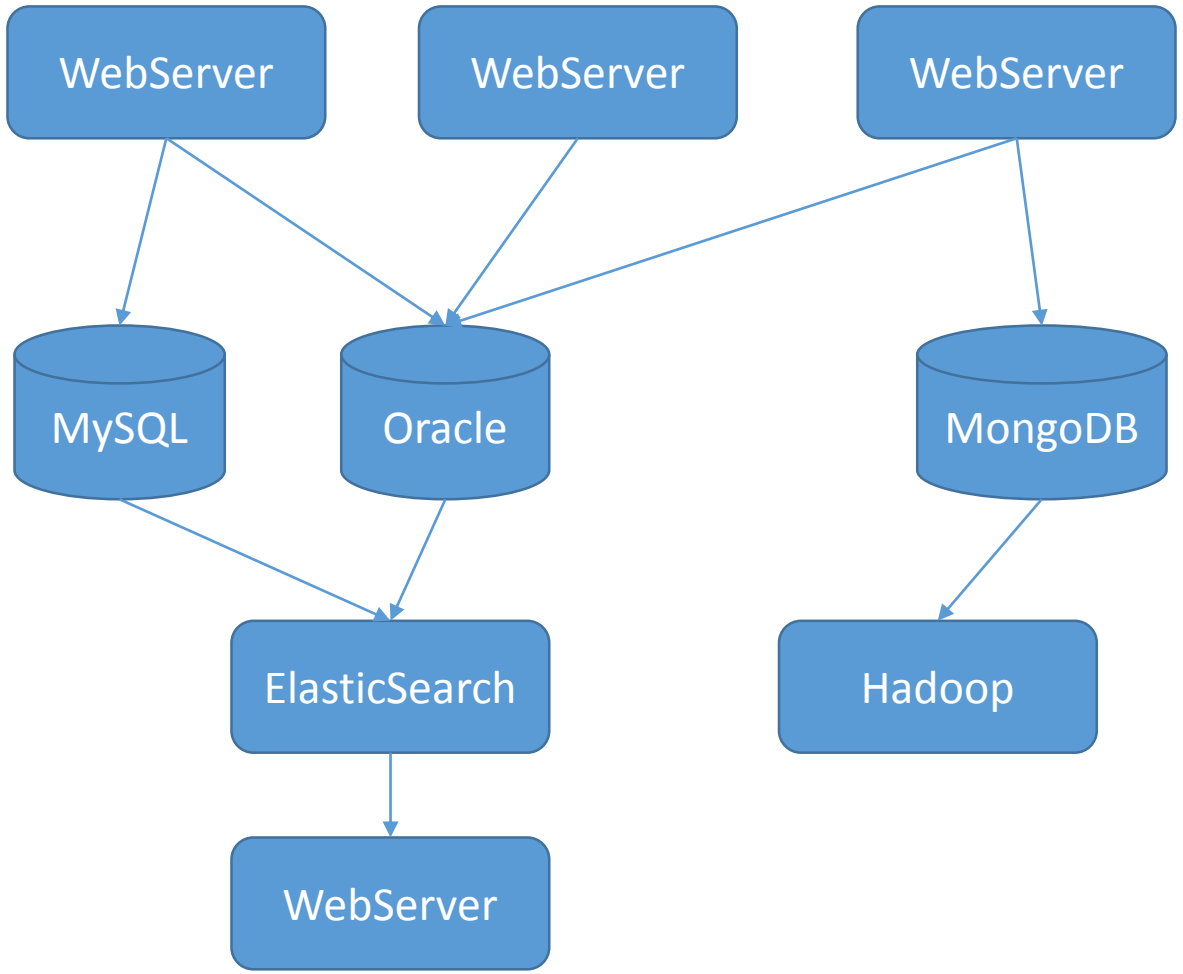
  节点在线扩容

- **高可靠：集群部署、多副本机制**

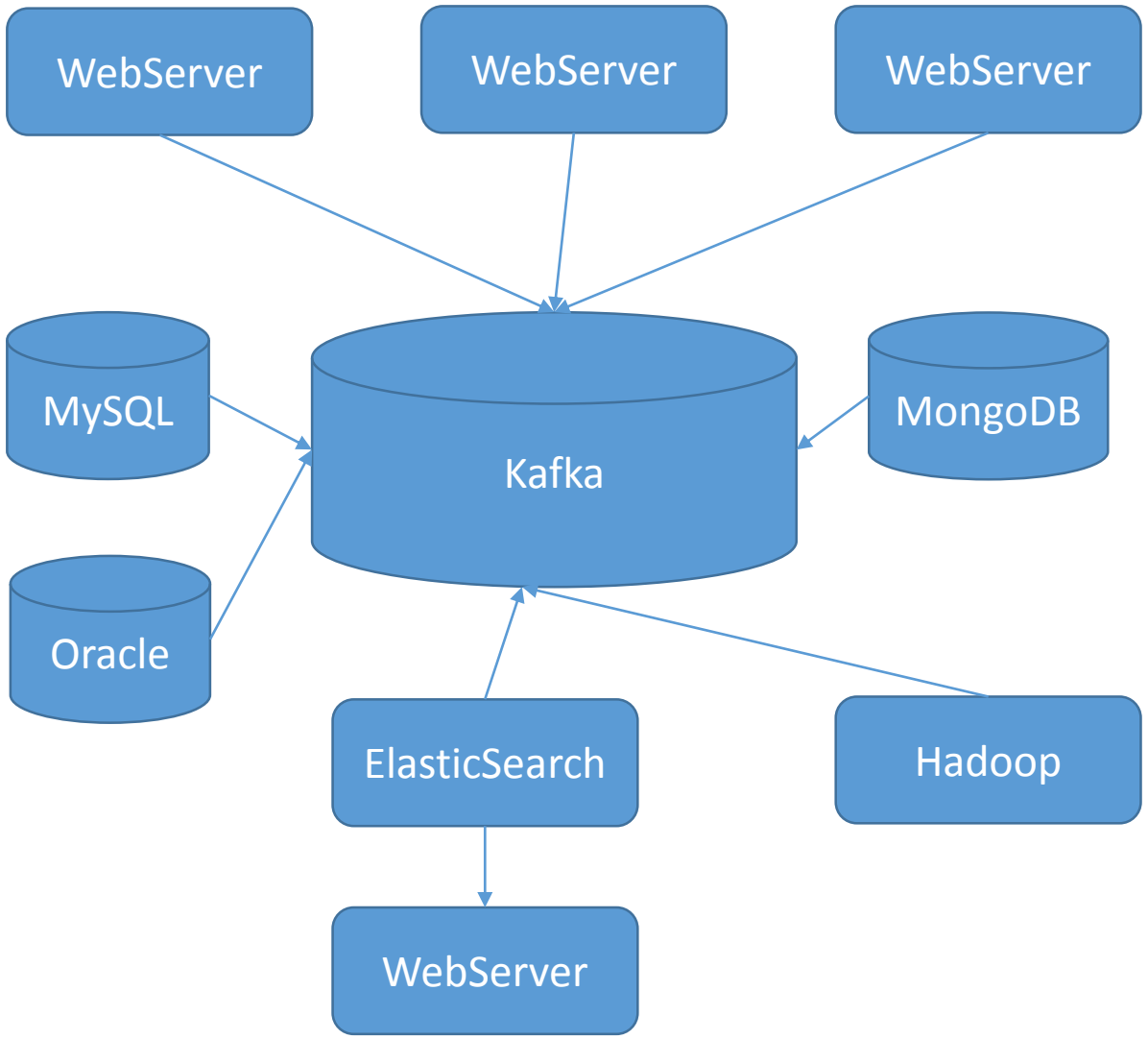
  多副本机制，支持在线扩容

  支持分区在线迁移

以Kafka为中心的解决方案

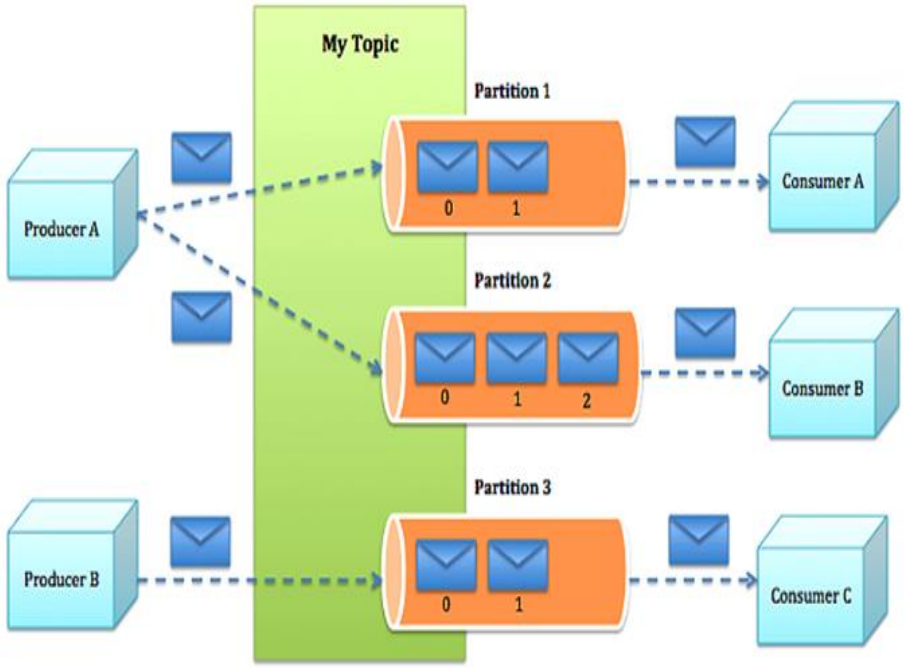


以Kafka为中心的解决方案

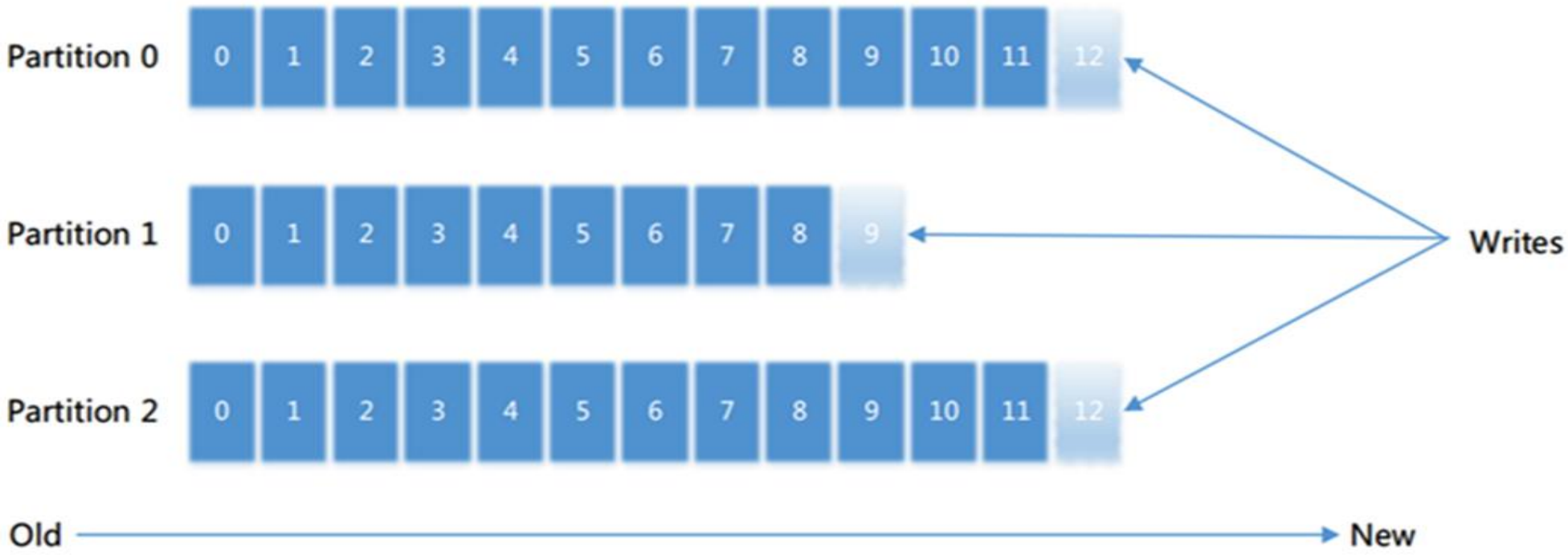


4. Kafka 基本概念

- **Broker**： Kafka集群包含一个或多个服务实例，这种服务实例被称为broker。
- **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。
- **Partition**： Partition是物理上的概念，每个Topic包含一个或多个Partition。
- **Producer**：负责发布消息到Kafka Broker。
- **Consumer**：消息消费者，向Kafka Broker读取消息的客户端。
- **Consumer Group**：每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，否则属于默认的group）。



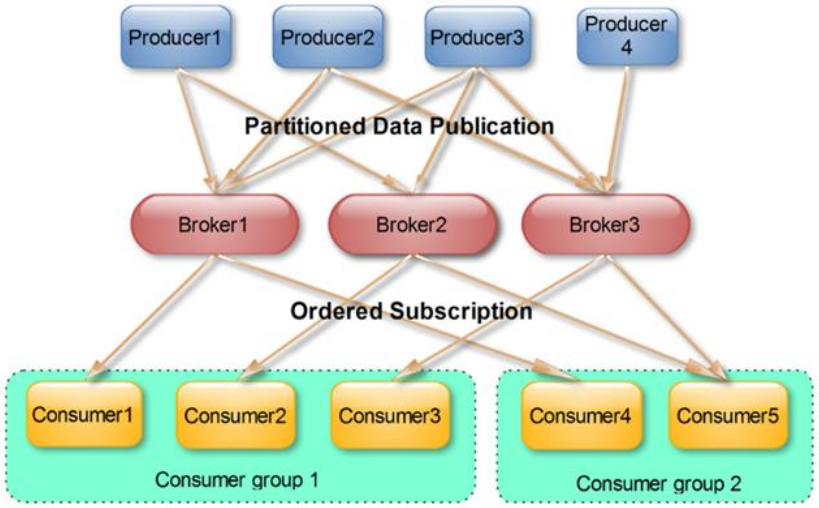
4.1 Kafka- Partition & Offset



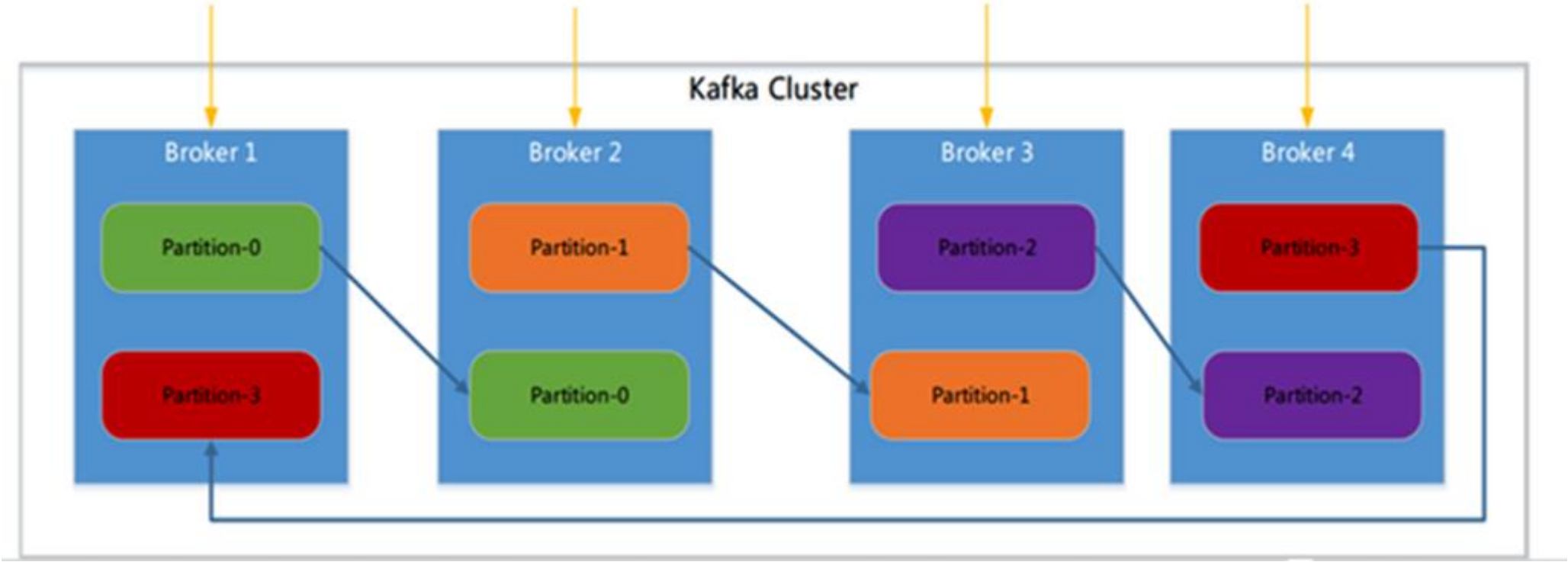
1. 每个分区消息只能通过追加消息方式增加消息，消息都有一个偏移量(offset)，顺序写入确保性能高效。
2. 消费者通过分区中offset定位消息和记录消费的位置

4.2 Kafka 消费者组与分区

1. 每个消费者都属于一个消费组内，通过消费组概念可以实现Topic消息的广播（发给所有消费组）或者单播（组内消息均衡分担）。
2. 消费者采用Pull模式进行消费，方便消费进度记录在客户端，服务端无状态。
3. 组内的消费者以Topic分区个数进行均衡分配，所以组内消费者最多只能有分区个数的消费者。



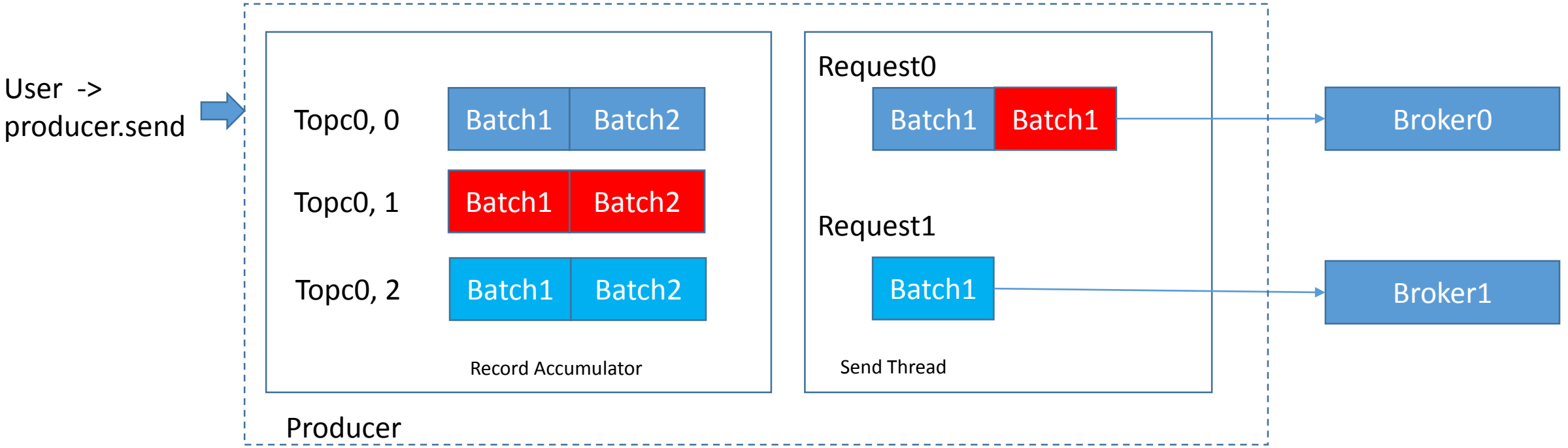
4.3 Kafka 分区副本



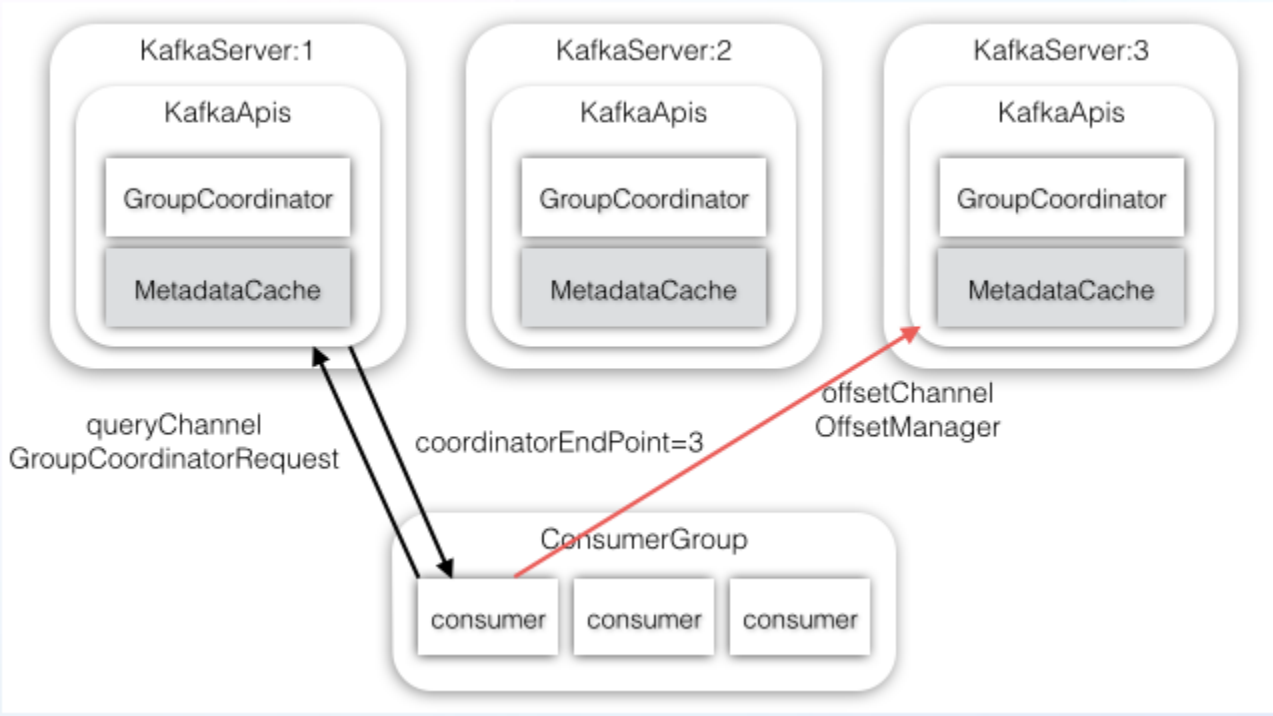
1. Kafka通过副本方式达到高可用的目标，每个分区可以有1个或者多个副本，分别分配在不同的节点上。
2. 多个副本之间只有一个Leader，其他副本通过Pull模式同步Leader消息，处于同步状态的副本集合称为ISR
3. 生产者和消费者都只能从Leader写入或者读取数据，Leader故障后，会优先从ISR集合中选择副本作为Leader
4. ISR同步副本的集合

4.4 Kafka 批量生产机制

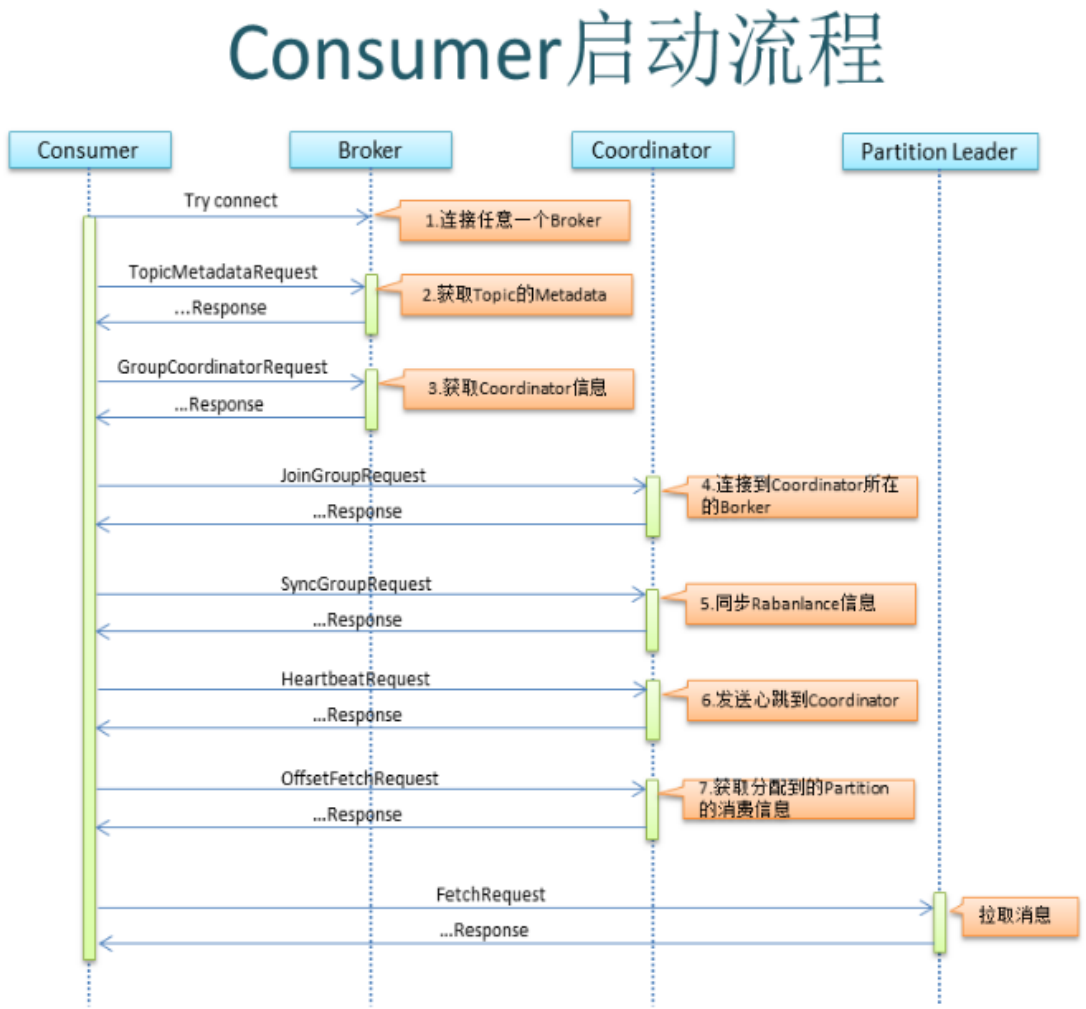
- 1. 在客户端通过异步接口发送消息，消息首先在客户端根据分区打包
- 2. 发送线程根据分区Leader所在broker，把多个batch组成一个请求，打包发送。
- 3. 2个参数控制打包速度：当 batch.size 达到或者linger.ms时间达到。
  批量发送包的大小越大吞吐量越高，但是时延也响应增大



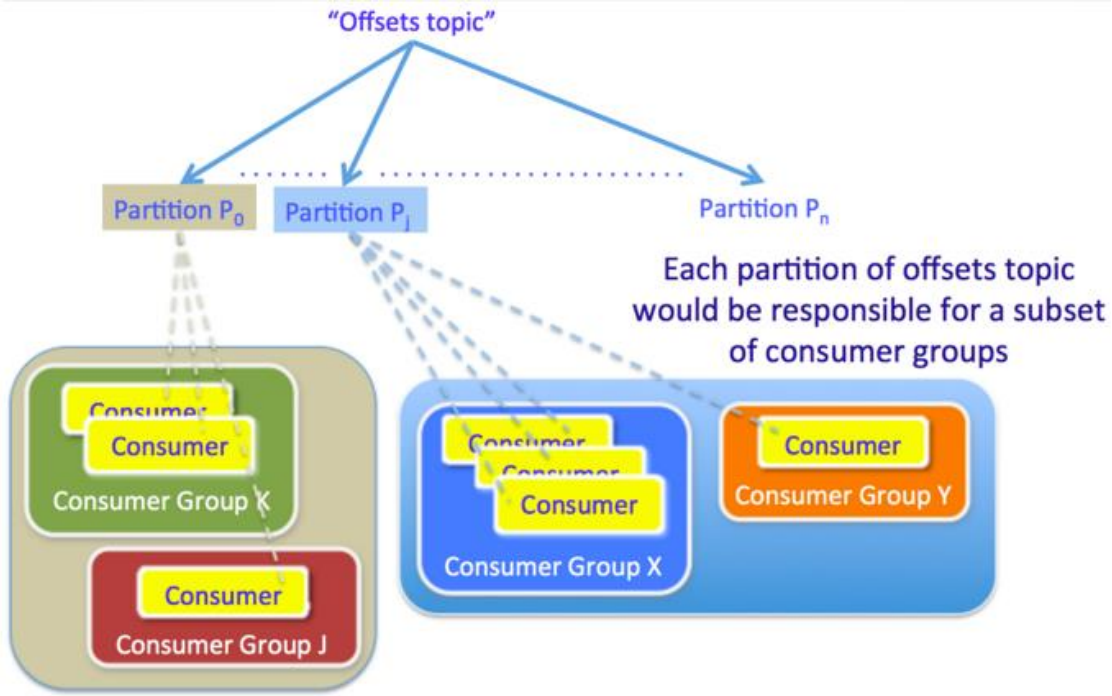
4.5 Kafka 消费机制

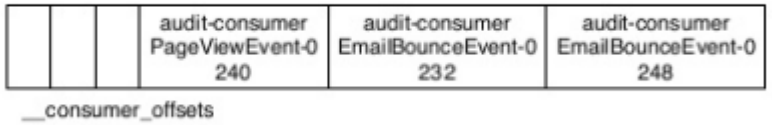


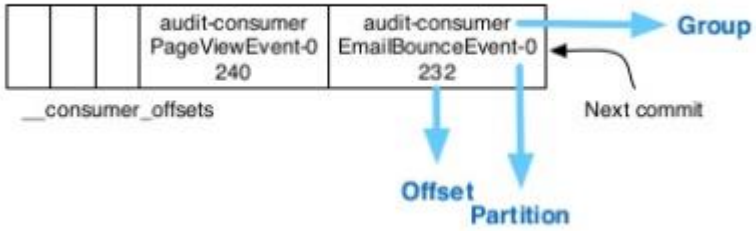
1. 每个消费组里面的消费者都需要先查找一个协调者
2. 消费者加入到这个组内，主要目的是为了对分区进行分区
3. 分配分区完毕后，再查找各自分区的Leader，进行消费



4.6 Kafka 消费进度管理机制







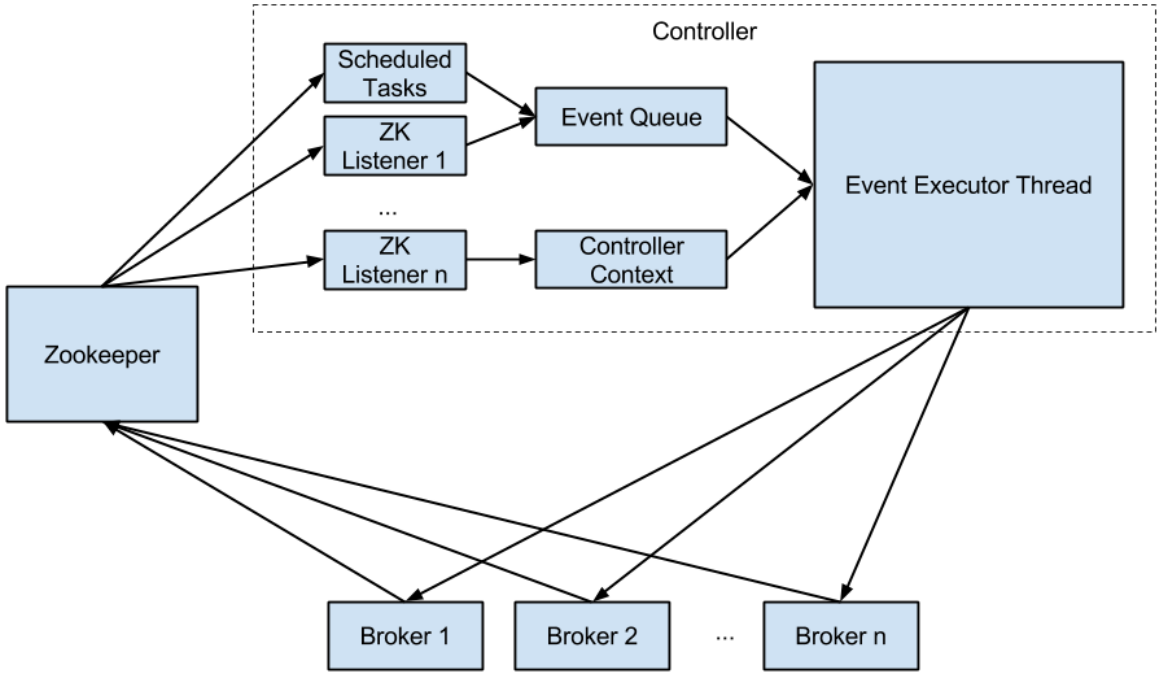
4.7 Kafka 高可靠机制

分布式系统下，单点故障不可避免，kafka如何管理节点故障?

1. 从Kafka的broker中选择一个节点作为分区管理与副本状态变更的控制，称为controller。
2. 统一侦听zk元数据变化，通知各节点状态信息。
3. 管理broker节点的故障恢复， 对故障节点所在分区进行重新Leader选举，帮助业务故障切换到新的Leader

如果Controller节点本身故障？

各个Broker节点通过watch zk的/controller节点，如果controller故障，会触发节点进行争夺创建/controller节点，创建上的节点成为新controller



实战演练

- 创建topic：topic-test01

- 使用kafka-console-producer.sh生产消息

bin/kafka-console-producer.sh --broker-list 127.0.0.1:9092 --topic topicName

- 使用kafka-console-consumer.sh消费消息

bin/kafka-console-consumer.sh --bootstrap-server 127.0.0.1:9092 --topic topicName --group testgroup

--consumer-property enable.auto.commit=true --from-beginning

Thank You