### Kafka 消息队列培训

Michael He

#### 什么是消息队列

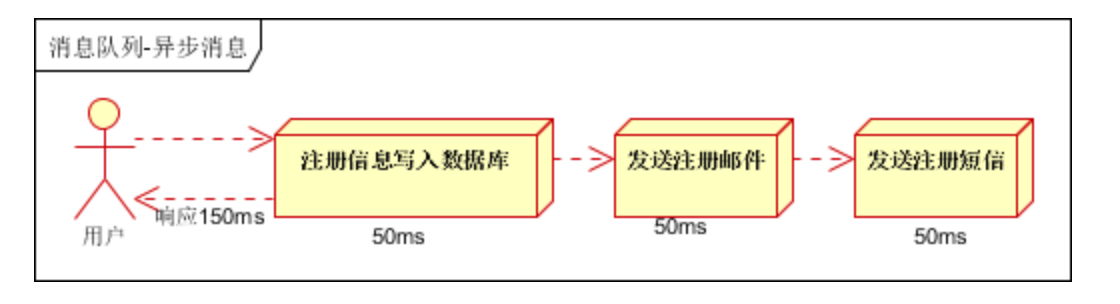
消息队列是在消息的传输过程中保存消息的容器。消息队列管理器在将消息从它的源中继到它的目标时充当中间人。队列的主要目的是提供路由并保证消息的传递；如果发送消息时接收者不可用或获取信息不成功，消息队列会保留消息，直到可以成功地传递它。

#### 消息队列的使用场景

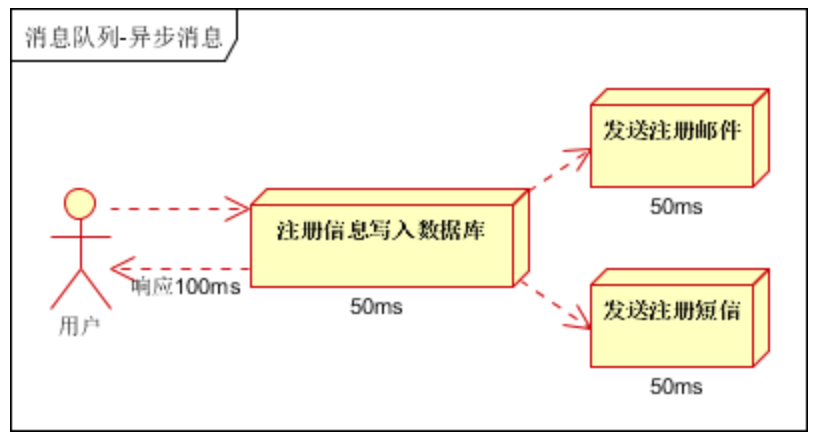
##### 2.1 异步处理

场景: 用户注册完成以后，需要发注册短信和注册邮件，传统的有串行和并行两种做法

1. **串行方式:** 把注册信息成功写入数据库后，再发送邮件和短信，最后再返回给客户。假设每个步骤的执行时间为50ms, 响应时间为150ms

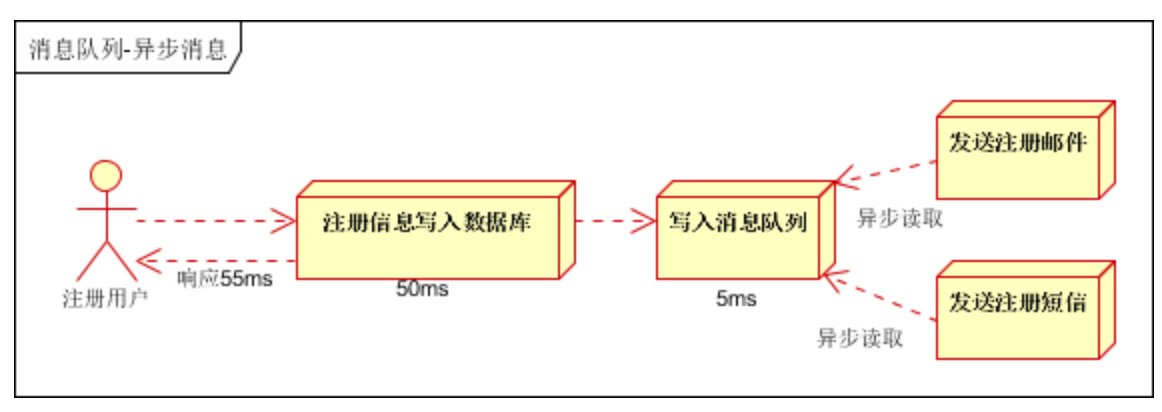


1. **并行方式:** 把注册信息成功写入数据库后，同时发送邮件和短信，最后再返回给客户。在不考虑网络开销的前提下，响应时间为100ms。



**小结:** 因为CPU在单位时间内处理的请求数是一定的，假设CPU 1秒内吞吐量是100次。则串行方式1秒内CPU可处理的请求量是7次（1000/150）。并行方式处理的请求量是10次（1000/100）

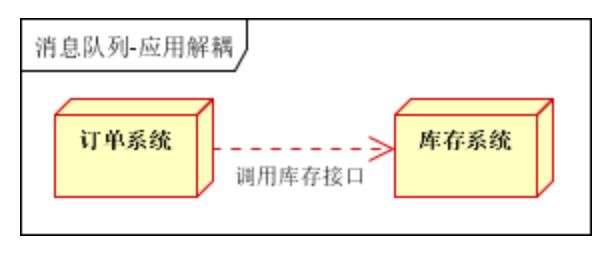
1. **引入消息队列**，将不是必须的业务逻辑，异步处理。改造后的架构如下:



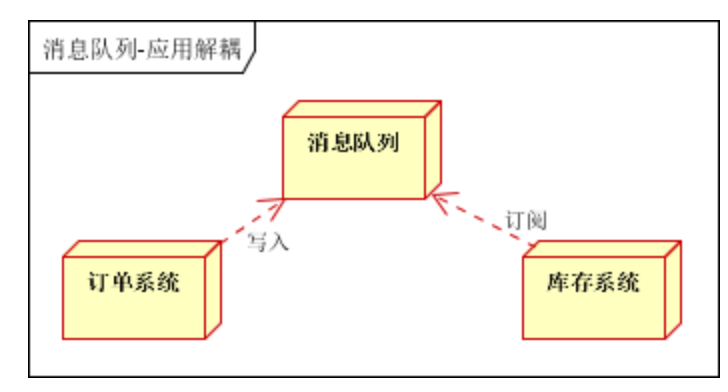
用户的响应时间相当于是注册信息写入数据库的时间，也就是50ms。注册邮件，发送短信写入消息队列后，直接返回，因此写入消息队列的速度很快，基本可以忽略，因此用户的响应时间可能是50毫秒。因此架构改变后，系统的吞吐量提高到每秒20 QPS。比串行提高了3倍，比并行提高了两倍

##### 2.2 应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如下图



这种方式的问题在于，如果库存系统发生问题, 无法减少库存数量，会导致整个订单生成的过程失败。如何解决这个问题呢，引入消息队列后，解决方案如下:



订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作

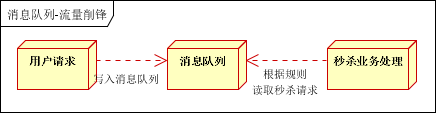
假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

##### 2.3 流量削锋

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

* 可以控制活动的人数
* 可以缓解短时间内高流量压垮应用



* 用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面
* 秒杀业务根据消息队列中的请求信息，再做后续处理

##### 2.4 数据传输

消息队列还可以用于进行数据的传输，以日志处理为例

日志处理是指将消息队列用在日志处理中，比如Kafka的应用，解决大量日志传输的问题。架构简化如下

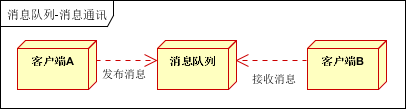


* 日志采集客户端，负责日志数据采集，定时写入Kafka队列
* Kafka消息队列，负责日志数据的接收，存储和转发
* 日志处理应用：订阅并消费Kafka 队列中的日志数据

##### 2.5 消息通讯

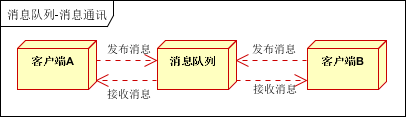
消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等

点对点通讯：



客户端A和客户端B使用同一队列，进行消息通讯。

聊天室通讯：



客户端A，客户端B，客户端N订阅同一主题，进行消息发布和接收。实现类似聊天室效果。

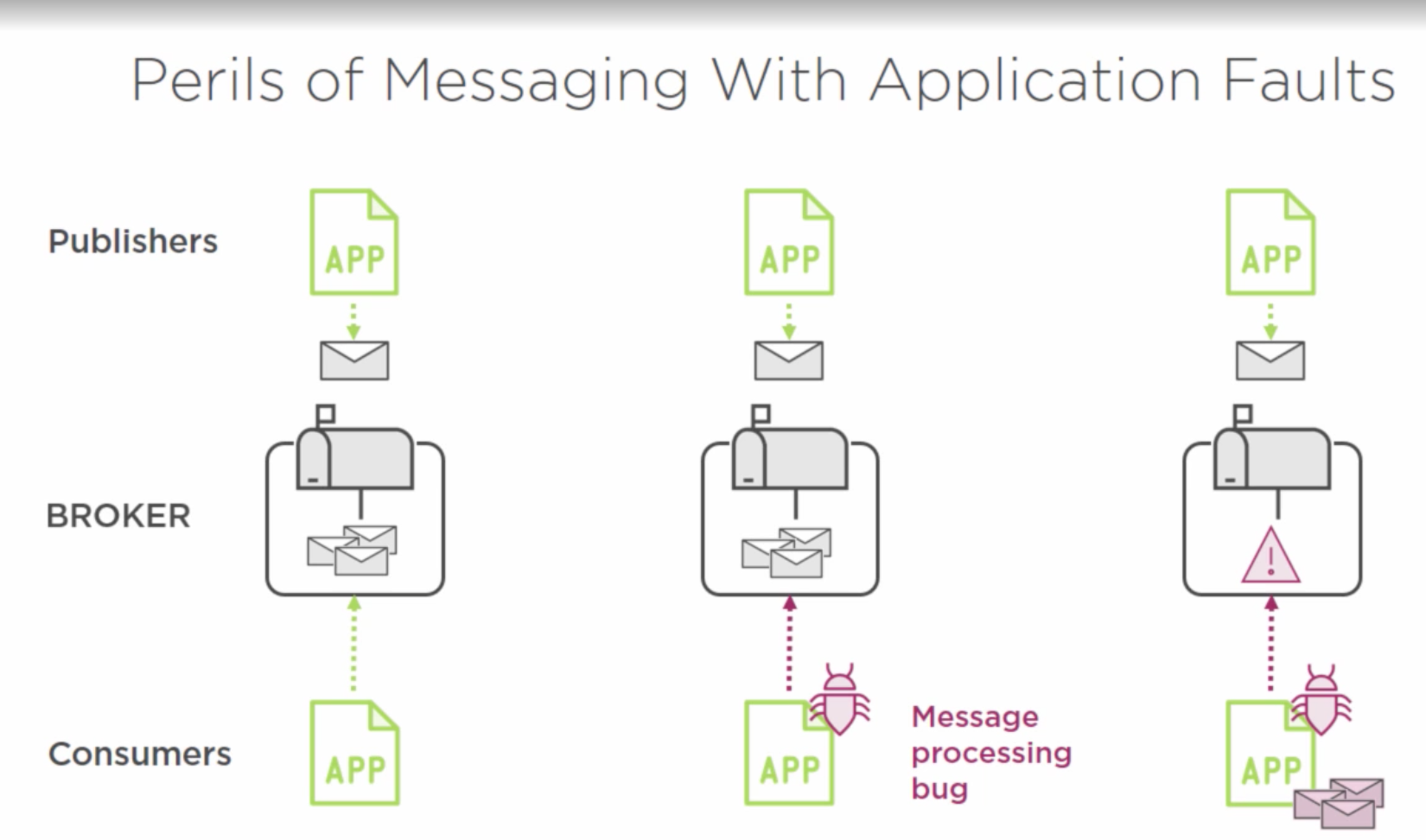
#### 传统的消息队列的几个问题

##### 3.1 数据量过大/消费太慢



当要publish 的消息过多或过大时, 因为消息队列服务器容量或网络的等物理条件的限制，整个队列有可能会出现堵塞。如果consumer 消费message 的速度不够快，那会更加雪上加霜。等待我们的就是**“404”**

##### 3.2 数据不可重复获取



如果消费程序有bug, 获取完队列数据后进行业务处理时报错了, 因为传统的队列会在确认消息被消费后将其从队列中删除，此时消费程序并没有正常处理获取的数据，但又无法重复获取数据, 实际上相当于丢失了数据。

Kafka 消息队列系统比较完美的解决了以上2个问题

#### Kafka 消息队列介绍

Kafka是由LinkedIn开发的一个**分布式**的消息系统，使用Scala编写，它以可水平扩展和高吞吐率而被广泛使用。目前越来越多的开源分布式处理系统如Cloudera、Apache Storm、Spark都支持与Kafka集成。

##### 4.1 整体架构

* **Broker**

Kafka集群包含一个或多个服务器，这种服务器被称为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。（物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处）

* **Partition**

Partition是物理上的概念，每个Topic包含一个或多个Partition, 同一个topic的partition 可以存在同一个broker 上也可以存在不同的broker上.

* Producer

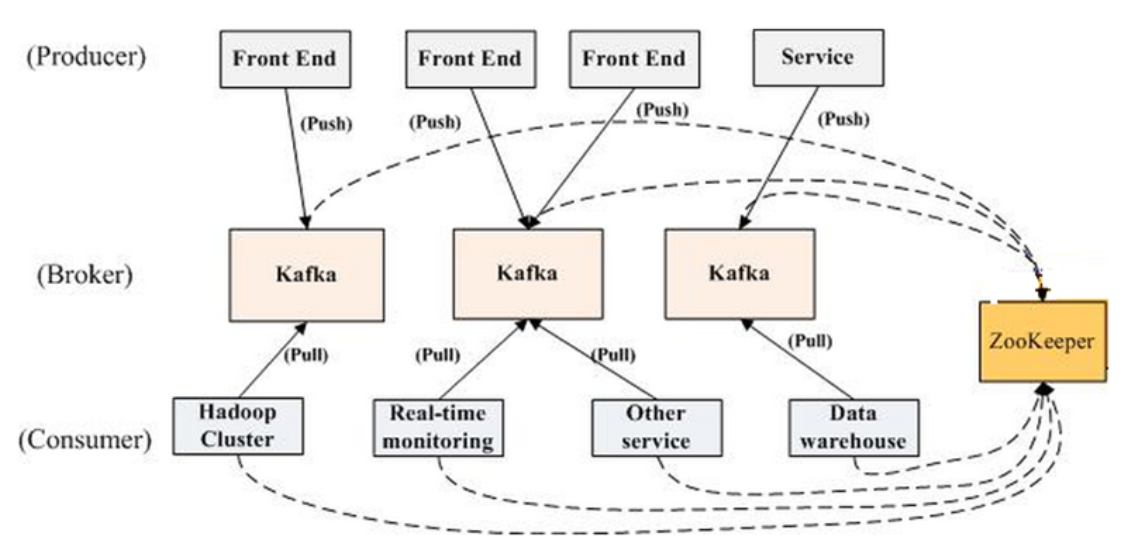
负责发布消息到Kafka broker

Consumer

消息消费者，向Kafka broker读取消息的客户端。

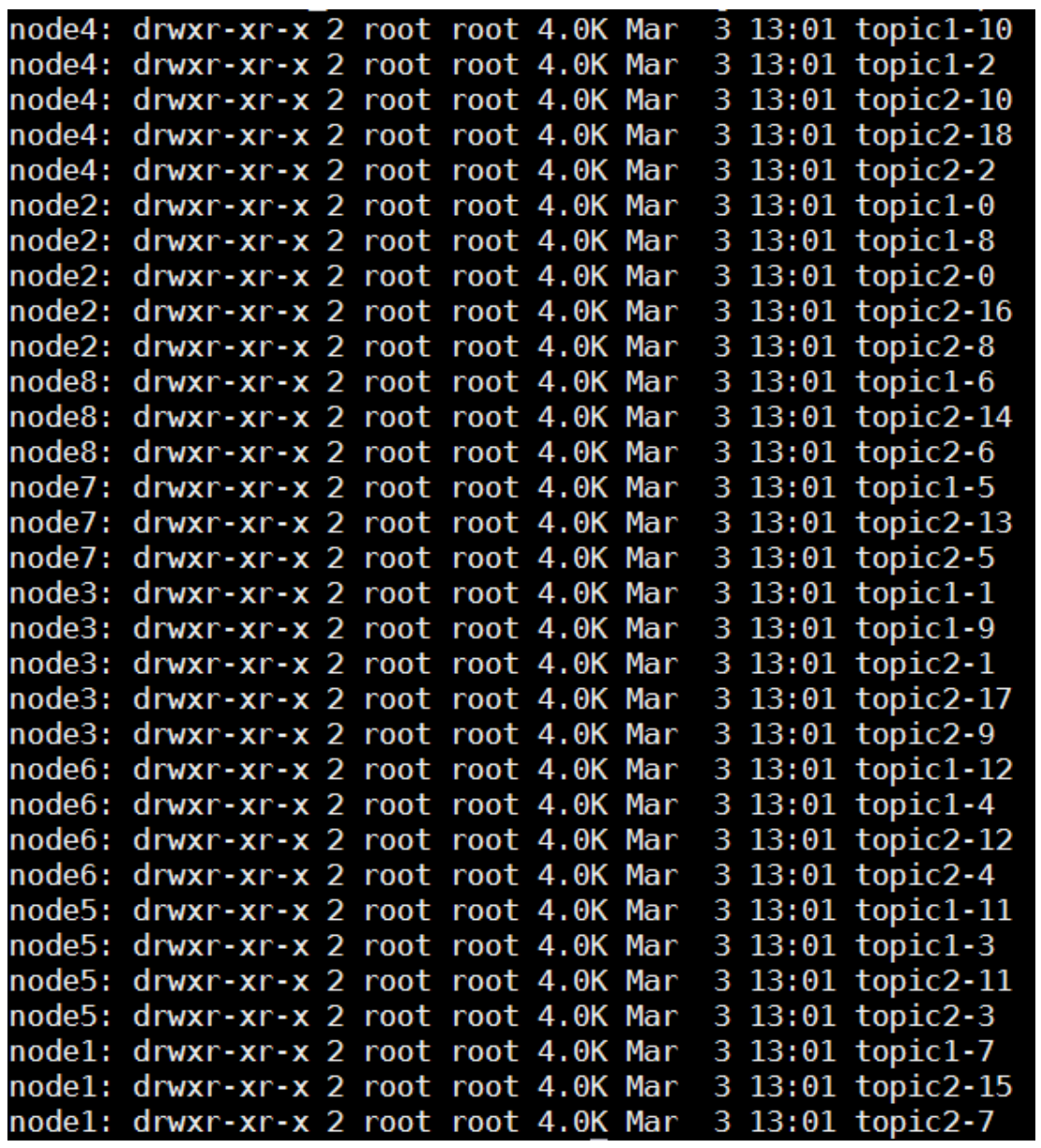
* Consumer Group

每个Consumer属于一个特定的Consumer Group（可为每个Consumer指定group name，若不指定group name则属于默认的group）

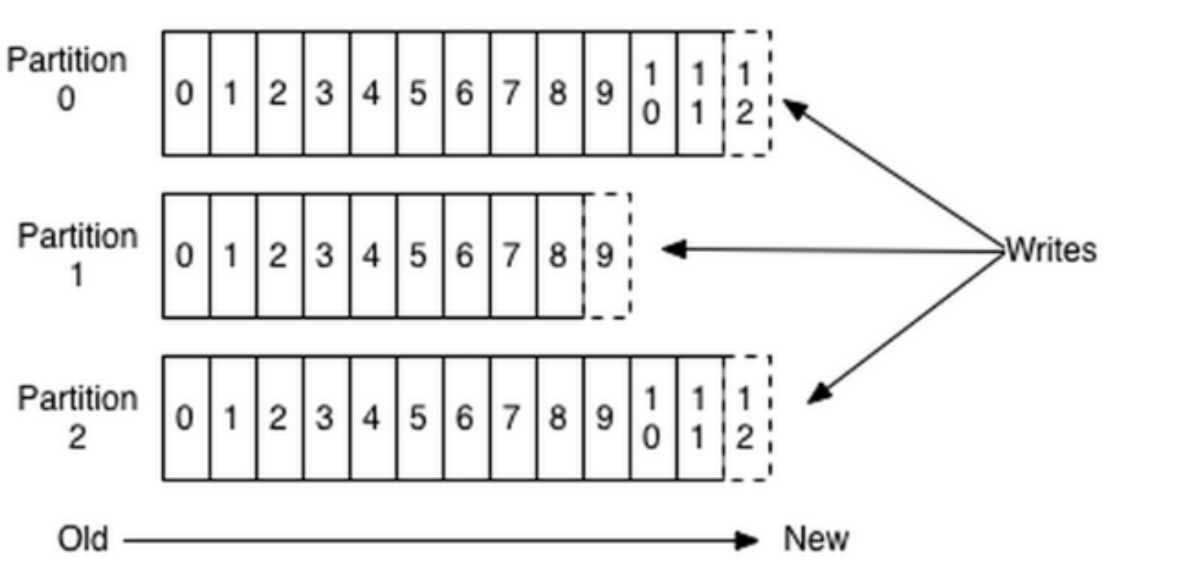


##### 4.2 Topics & Partition

Topic在逻辑上可以被认为是一个queue，每条消费都必须指定它的Topic，可以简单理解为必须指明把这条消息放进哪个queue里。为了使得Kafka的吞吐率可以线性提高，物理上把Topic分成一个或多个Partition，每个Partition在物理上对应一个文件夹，该文件夹下存储这个Partition的所有消息和索引文件。若创建topic1和topic2两个topic，且分别有13个和19个分区，则整个集群上会相应会生成共32个文件夹。假设集群中有8个broker这32个文件夹会被相对均匀的存放在8台服务器上：



Producer发送消息到broker时，会根据Partition机制选择将其存储到哪一个Partition。如果Partition机制设置合理，所有消息可以均匀分布到不同的Partition里，这样就实现了负载均衡。如果一个Topic对应一个文件，那这个文件所在的机器I/O将会成为这个Topic的性能瓶颈，而有了Partition后，不同的消息可以并行写入不同broker的不同Partition里，极大的提高了吞吐率。



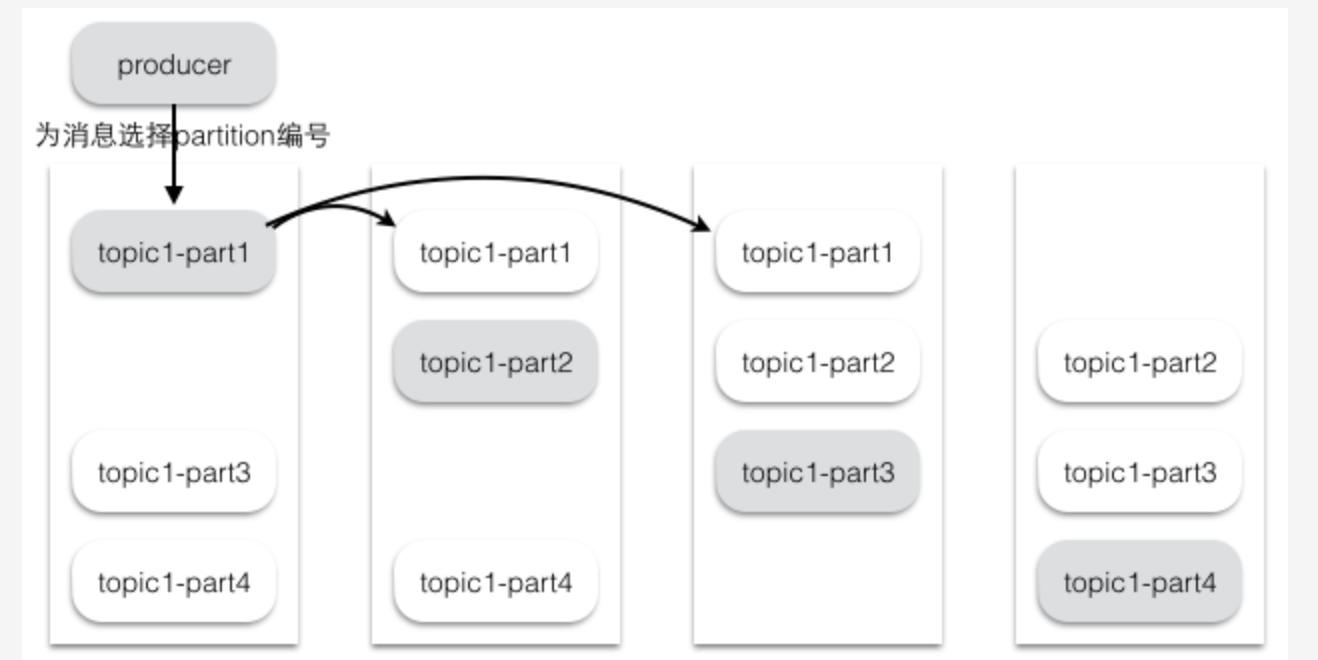
对于传统的message queue而言，一般会删除已经被消费的消息，而Kafka集群会保留所有的消息，无论其被消费与否。当然，因为磁盘限制，不可能永久保留所有数据（实际上也没必要），因此Kafka提供两种策略删除旧数据。一是基于时间，二是基于Partition文件大小。例如可以通过配置$KAFKA\_HOME/config/server.properties，让Kafka删除一周前的数据，也可在Partition文件超过1GB时删除旧数据

##### 4.3 Replication

Kafka 0.8后提供了Replication机制来保证Broker的failover。

引入Replication之后，同一个Partition可能会有多个Replica，而这时需要在这些Replication之间选出一个Leader，Producer和Consumer只与这个Leader交互，其它Replica作为Follower从Leader中复制数据。

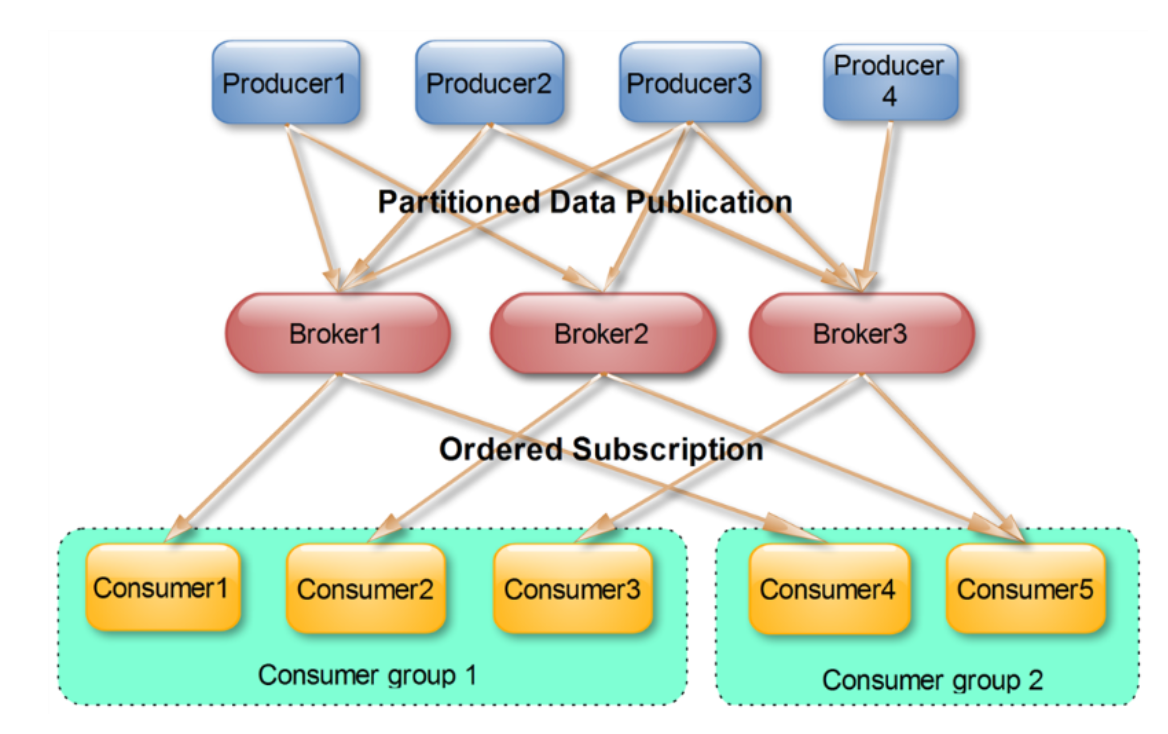
假设集群一共有4个brokers，一个topic有4个partition，每个Partition有3个副本。下图是每个Broker上的副本分配情况。



当leader 处于Inactive状态时, Kafka 会根据一定的规则选择一个可用Follower 作为新的Leader, 从而保证了整个系统的高可用性.

##### 4.4 Consumer Group

使用Consumer high level API时，同一Topic的一条消息只能被同一个Consumer Group内的一个Consumer消费，但多个Consumer Group可同时消费这一消息。



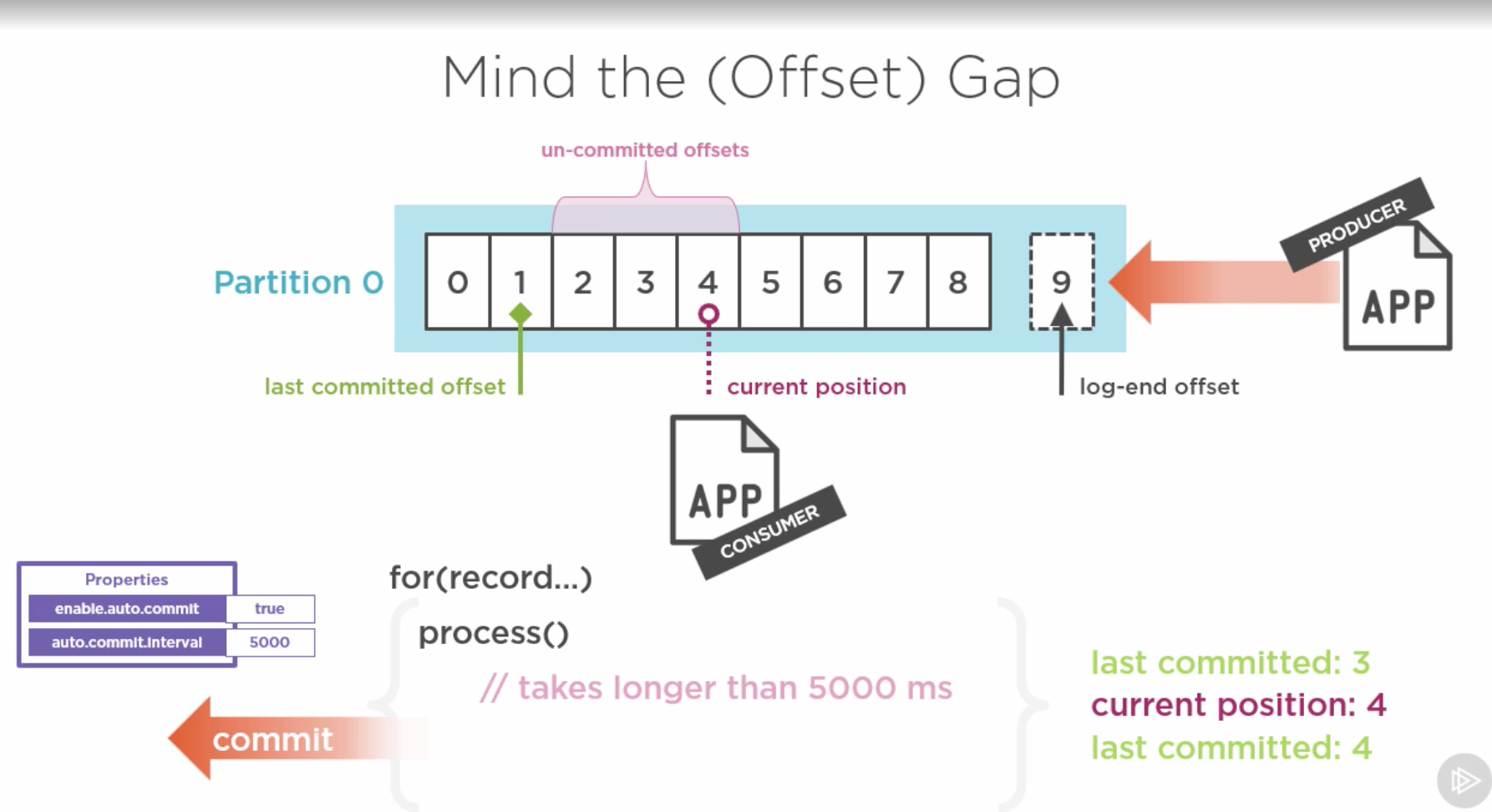
这是Kafka用来实现一个Topic消息的广播（发给所有的Consumer）和单播（发给某一个Consumer）的手段。一个Topic可以对应多个Consumer Group。如果需要实现广播，只要每个Consumer有一个独立的Group就可以了。要实现单播只要所有的Consumer在同一个Group里。用Consumer Group还可以将Consumer进行自由的分组而不需要多次发送消息到不同的Topic。

##### 4.5 Consumer Offset

Consumer 使用Offset 来控制数据的读取，Consumer 每读取一条数据Current Position 就向前移动一个位置。每隔一定的Interval(可配置)， 系统会自动进行一次Offset commit, 把所有Consumer 的Current Position 更新为Last Committed Offset, Kafka 会记录每个Consumer 的Last Committed Offset，如果Consumer 宕机，下次启动时会从上一次的Last Committed Offset 开始重新读取数据

Kafka 的Consumer API 提供了接口用于手动触发Offset 的 Commit, 开发者可以自行决定每消费一条数据就commit 还是处理完一批数据后再commit.

另外, Consumer 可以自由设置Current position 的位置，即使已经读取完所有的数据也可以通过position 的Reset 来实现数据的重复读取.

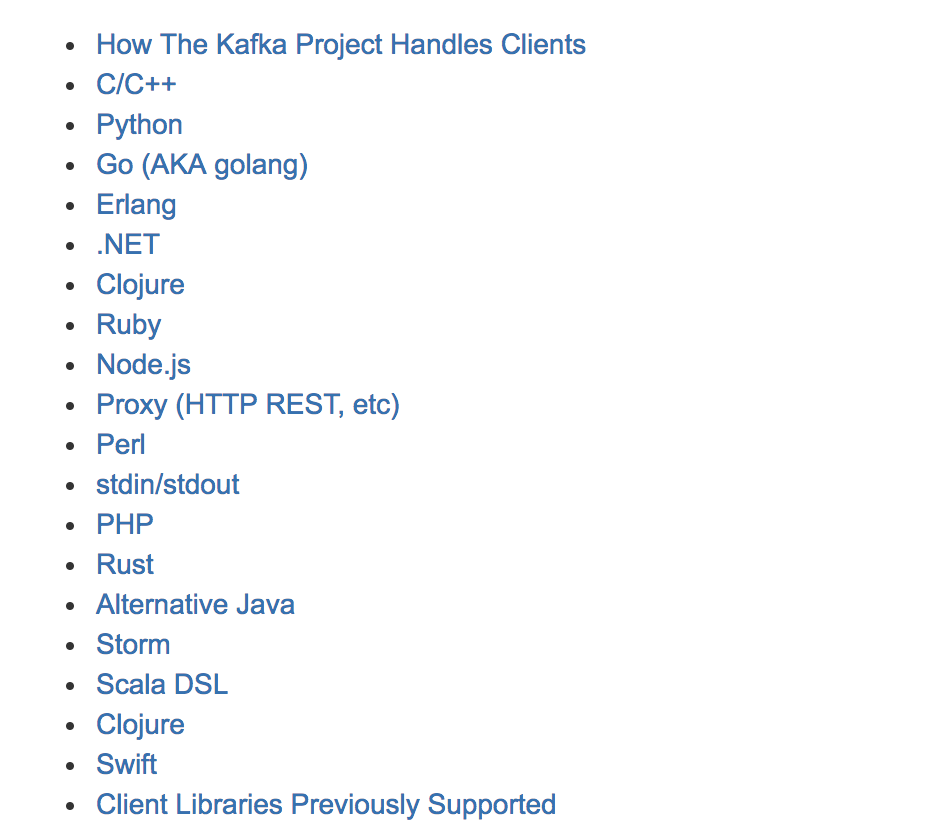


#### Kafka 开发

应用系统的开发者主要是调用Kafka 的Producer API 和Consumer API 来发送和

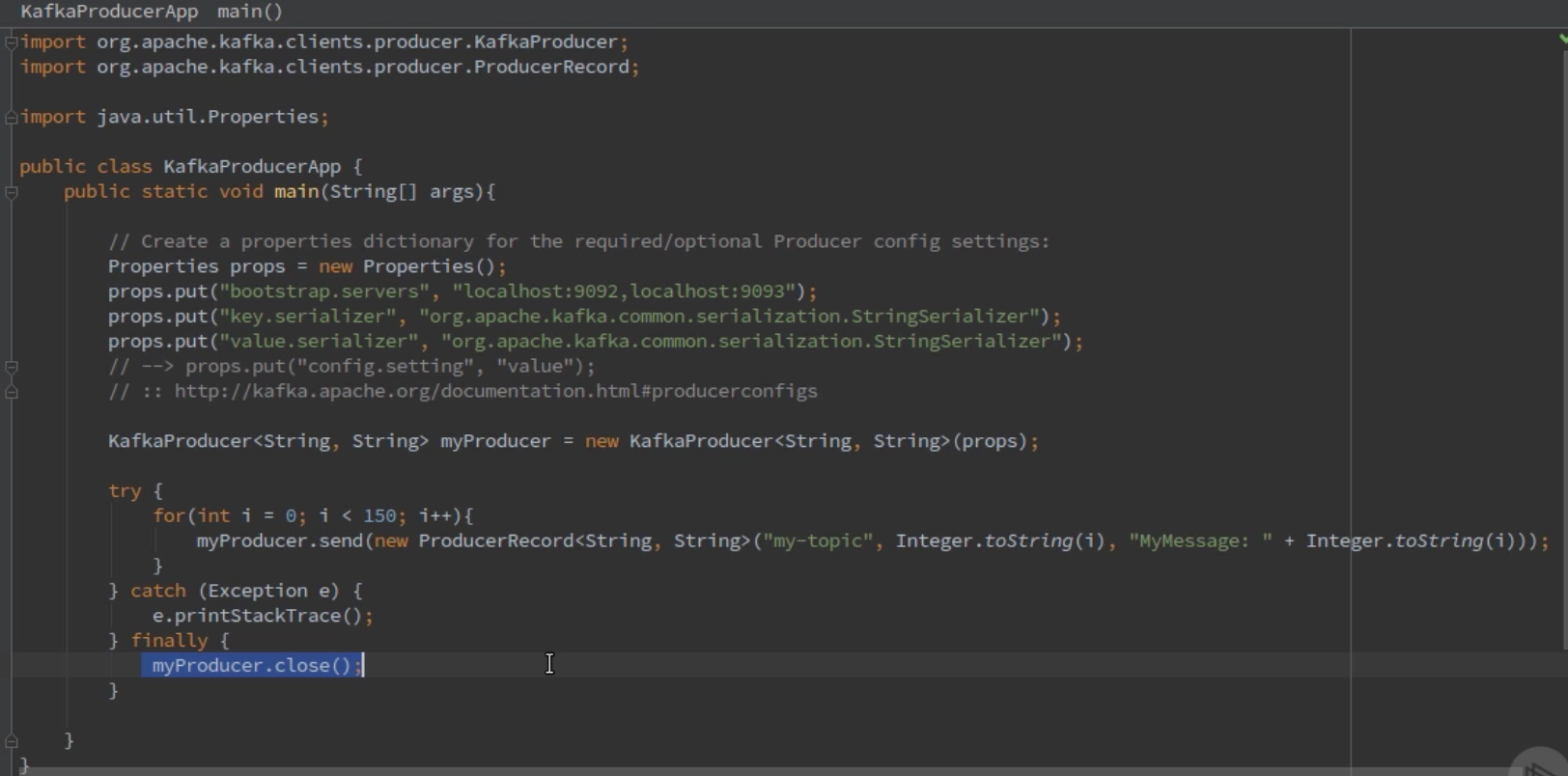
接收数据

Kafka API 支持大多数主流开发语言



##### 5.1 Producer

Java 版本



##### 5.2 Consumer

