# Kafka

## 简介

Kafka is a distributed,partitioned,replicated commit logservice。它提供了类似于JMS的特性，但是在[设计](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%C9%E8%BC%C6&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=4&seller_id=1&di=128)实现上完全不同，此外它并不是JMS规范的实现。kafka对消息保存时根据Topic进行归类，发送消息者成为Producer,消息接受者成为Consumer,此外kafka集群有多个kafka实例组成，每个实例([server](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=server&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=5&seller_id=1&di=128))成为broker。无论是kafka集群，还是producer和consumer都依赖于zookeeper来保证系统可用性集群保存一些meta信息。



## Topics/logs

一个Topic可以认为是一类消息，每个topic将被分成多个partition(区),每个partition在存储层面是append log文件。任何发布到此partition的消息都会被直接追加到log文件的尾部，每条消息在文件中的位置称为offset（偏移量），offset为一个long型数字，它是唯一标记一条消息。它唯一的标记一条消息。kafka并没有提供其他额外的索引机制来存储offset，因为在kafka中几乎不允许对消息进行“随机读写”。



kafka和JMS（Java Message Service）实现(activeMQ)不同的是:即使消息被消费,消息仍然不会被立即删除.日志文件将会根据broker中的配置要求,保留一定的时间之后删除;比如log文件保留2天,那么两天后,文件会被清除,无论其中的消息是否被消费.kafka通过这种简单的手段,来释放磁盘空间,以及减少消息消费之后对文件内容改动的磁盘IO开支.

对于consumer而言,它需要保存消费消息的offset,对于offset的保存和使用,有consumer来控制;当consumer正常消费消息时,offset将会"线性"的向前驱动,即消息将依次顺序被消费.事实上consumer可以使用任意顺序消费消息,它只需要将offset重置为任意值..(offset将会保存在zookeeper中,参见下文)

 kafka集群几乎不需要维护任何consumer和producer状态信息,这些信息有zookeeper保存;因此producer和consumer的[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%BF%CD%BB%A7%B6%CB&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=3&seller_id=1&di=128)实现非常轻量级,它们可以随意离开,而不会对集群造成额外的影响.

partitions的[设计](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%C9%E8%BC%C6&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=4&seller_id=1&di=128)目的有多个.最根本原因是kafka基于文件存储.通过分区,可以将日志内容分散到多个[server](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=server&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=5&seller_id=1&di=128)上,来避免文件尺寸达到单机磁盘的上限,每个partiton都会被当前server(kafka实例)保存;可以将一个topic切分多任意多个partitions,来消息保存/消费的效率.此外越多的partitions意味着可以容纳更多的consumer,有效提升并发消费的能力.(具体原理参见下文).

## Distribution

 一个Topic的多个partitions,被分布在kafka集群中的多个server上;每个server(kafka实例)负责partitions中消息的读写操作;此外kafka还可以配置partitions需要备份的个数(replicas),每个partition将会被备份到多台机器上,以提高可用性.

基于replicated方案,那么就意味着需要对多个备份进行调度;每个partition都有一个[server](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=server&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=5&seller_id=1&di=128)为"leader";leader负责所有的读写操作,如果leader失效,那么将会有其他follower来接管(成为新的leader);follower只是单调的和leader跟进,同步消息即可..由此可见作为leader的server承载了全部的请求压力,因此从集群的整体考虑,有多少个partitions就意味着有多少个"leader",kafka会将"leader"均衡的分散在每个实例上,来确保整体的性能稳定.

**Producers**

    Producer将消息发布到指定的Topic中,同时Producer也能决定将此消息归属于哪个partition;比如基于"round-robin"方式或者通过其他的一些算法等.

**Consumers**

    本质上kafka只支持Topic.每个consumer属于一个consumer group;反过来说,每个group中可以有多个consumer.发送到Topic的消息,只会被订阅此Topic的每个group中的一个consumer消费.

    如果所有的consumer都具有相同的group,这种情况和queue模式很像;消息将会在consumers之间负载均衡.

    如果所有的consumer都具有不同的group,那这就是"发布-订阅";消息将会广播给所有的消费者.

    在kafka中,一个partition中的消息只会被group中的一个consumer消费;每个group中consumer消息消费互相独立;我们可以认为一个group是一个"订阅"者,一个Topic中的每个partions,只会被一个"订阅者"中的一个consumer消费,不过一个consumer可以消费多个partitions中的消息.kafka只能保证一个partition中的消息被某个consumer消费时,消息是顺序的.事实上,从Topic角度来说,消息仍不是有序的.

    kafka的[设计](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%C9%E8%BC%C6&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=4&seller_id=1&di=128)原理决定,对于一个topic,同一个group中不能有多于partitions个数的consumer同时消费,否则将意味着某些consumer将无法得到消息.

**Guarantees**

    1) 发送到partitions中的消息将会按照它接收的顺序追加到日志中

    2) 对于消费者而言,它们消费消息的顺序和日志中消息顺序一致.

    3) 如果Topic的"replicationfactor"为N,那么允许N-1个kafka实例失效.

## 使用场景

**Messaging**

    对于一些常规的消息系统,kafka是个不错的选择;partitons/replication和容错,可以使kafka具有良好的扩展性和性能优势.不过到目前为止,我们应该很清楚认识到,kafka并没有提供JMS中的"事务性""消息传输担保(消息确认机制)""消息分组"等企业级特性;kafka只能使用作为"常规"的消息系统,在一定程度上,尚未确保消息的发送与接收绝对可靠(比如,消息重发,消息发送丢失等)

**Websit activity tracking**

    kafka可以作为"网站活性跟踪"的最佳工具;可以将网页/用户操作等信息发送到kafka中.并实时监控,或者离线统计分析等

**Log Aggregation**

    kafka的特性决定它非常适合作为"日志收集中心";application可以将操作日志"批量""异步"的发送到kafka集群中,而不是保存在本地或者DB中;kafka可以批量提交消息/压缩消息等,这对producer端而言,几乎感觉不到性能的开支.此时consumer端可以使hadoop等其他系统化的存储和分析系统.

## 设计原理

kafka的[设计](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%C9%E8%BC%C6&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=4&seller_id=1&di=128)初衷是希望作为一个统一的信息收集平台,能够实时的收集反馈信息,并需要能够支撑较大的数据量,且具备良好的容错能力

**持久性**

kafka使用文件存储消息,这就直接决定kafka在性能上严重依赖文件系统的本身特性.且无论任何OS下,对文件系统本身的优化几乎没有可能.文件缓存/直接内存映射等是常用的手段.因为kafka是对日志文件进行append操作,因此磁盘检索的开支是较小的;同时为了减少磁盘写入的次数,broker会将消息暂时buffer起来,当消息的个数(或尺寸)达到一定阀值时,再flush到磁盘,这样减少了磁盘IO调用的次数.

**性能**

需要考虑的影响性能点很多,除磁盘IO之外,我们还需要考虑网络IO,这直接关系到kafka的吞吐量问题.kafka并没有提供太多高超的技巧;对于producer端,可以将消息buffer起来,当消息的条数达到一定阀值时,批量发送给broker;对于consumer端也是一样,批量fetch多条消息.不过消息量的大小可以通过配置文件来指定.对于kafka broker端,似乎有个sendfile系统调用可以潜在的提升网络IO的性能:将文件的数据映射到系统内存中,socket直接读取相应的内存区域即可,而无需进程再次copy和交换. 其实对于producer/consumer/broker三者而言,CPU的开支应该都不大,因此启用消息压缩机制是一个良好的策略;压缩需要消耗少量的CPU资源,不过对于kafka而言,网络IO更应该需要考虑.可以将任何在网络上传输的消息都经过压缩.kafka支持gzip/snappy等多种压缩方式.

**生产者**

负载均衡: producer将会和Topic下所有partition leader保持socket连接;消息由producer直接通过socket发送到broker,中间不会经过任何"路由层".事实上,消息被路由到哪个partition上,有producer[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%BF%CD%BB%A7%B6%CB&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=3&seller_id=1&di=128)决定.比如可以采用"random""key-hash""轮询"等,如果一个topic中有多个partitions,那么在producer端实现"消息均衡分发"是必要的.

其中partition leader的位置(host:port)注册在zookeeper中,producer作为zookeeper client,已经注册了watch用来监听partition leader的变更事件.

异步发送：将多条消息暂且在客户端buffer起来，并将他们批量的发送到broker，小数据IO太多，会拖慢整体的网络延迟，批量延迟发送事实上提升了网络效率。不过这也有一定的隐患，比如说当producer失效时，那些尚未发送的消息将会丢失。

**消费者**

consumer端向broker发送"fetch"请求,并告知其获取消息的offset;此后consumer将会获得一定条数的消息;consumer端也可以重置offset来重新消费消息.

在JMS实现中,Topic模型基于push方式,即broker将消息推送给consumer端.不过在kafka中,采用了pull方式,即consumer在和broker建立连接之后,主动去pull(或者说fetch)消息;这中模式有些优点,首先consumer端可以根据自己的消费能力适时的去fetch消息并处理,且可以控制消息消费的进度(offset);此外,消费者可以良好的控制消息消费的数量,batch fetch.

其他JMS实现,消息消费的位置是有prodiver保留,以便避免重复发送消息或者将没有消费成功的消息重发等,同时还要控制消息的状态.这就要求JMS broker需要太多额外的工作.在kafka中,partition中的消息只有一个consumer在消费,且不存在消息状态的控制,也没有复杂的消息确认机制,可见kafka broker端是相当轻量级的.当消息被consumer接收之后,consumer可以在本地保存最后消息的offset,并间歇性的向zookeeper注册offset.由此可见,consumer[客户端](http://cpro.baidu.com/cpro/ui/uijs.php?rs=1&u=http%3A%2F%2Fwww%2Eaboutyun%2Ecom%2Fthread%2D9341%2D1%2D1%2Ehtml&p=baidu&c=news&n=10&t=tpclicked3_hc&q=92051019_cpr&k=%BF%CD%BB%A7%B6%CB&k0=java&kdi0=8&k1=%B1%E0%B3%CC&kdi1=8&k2=%BF%CD%BB%A7%B6%CB&kdi2=8&k3=%C9%E8%BC%C6&kdi3=8&k4=server&kdi4=1&sid=4ebca4a25f27e407&ch=0&tu=u1692056&jk=fb2f0911808fa875&cf=29&fv=14&stid=9&urlid=0&luki=3&seller_id=1&di=128)也很轻量级.



## 消息传送机制

对于JMS实现,消息传输担保非常直接:有且只有一次(exactly once).在kafka中稍有不同:

1) at most once: 最多一次,这个和JMS中"非持久化"消息类似.发送一次,无论成败,将不会重发.

2) at least once: 消息至少发送一次,如果消息未能接受成功,可能会重发,直到接收成功.

3) exactly once: 消息只会发送一次.

at most once: 消费者fetch消息,然后保存offset,然后处理消息;当client保存offset之后,但是在消息处理过程中出现了异常,导致部分消息未能继续处理.那么此后"未处理"的消息将不能被fetch到,这就是"at most once".

at least once: 消费者fetch消息,然后处理消息,然后保存offset.如果消息处理成功之后,但是在保存offset阶段zookeeper异常导致保存操作未能执行成功,这就导致接下来再次fetch时可能获得上次已经处理过的消息,这就是"at least once"，原因offset没有及时的提交给zookeeper，zookeeper恢复正常还是之前offset状态.

exactly once: kafka中并没有严格的去实现(基于2阶段提交,事务),我们认为这种策略在kafka中是没有必要的.

通常情况下"at-least-once"是我们搜选.(相比at most once而言,重复接收数据总比丢失数据要好).

## 复制备份

## 日志

## 分配