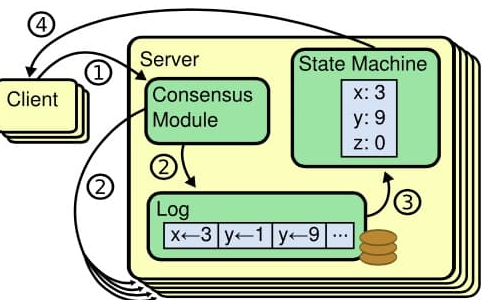
分布式一致性

Raft适用于管理日志一致性协议，包括：领导选取(leader selection)，日志复制(log replication);

解决问题:分布式存储系统通过维护多个副本来提高系统的availablity,代价就是维护多个副本的一致性。Raft协议基于复制状态机(replicated state machine)，即一组server从相同的初始状态起，按相同的顺序执行相同的命令，最终会达到一致的状态。

每台机器保存一份日志，日志来自于客户端的请求，包含一系列的命令，状态机会按顺序执行这些命令。一致性算法管理来自客户端的复制日志，保证状态机处理的日志中的命令的顺序都是一致的，因此会得到相同的执行结果。



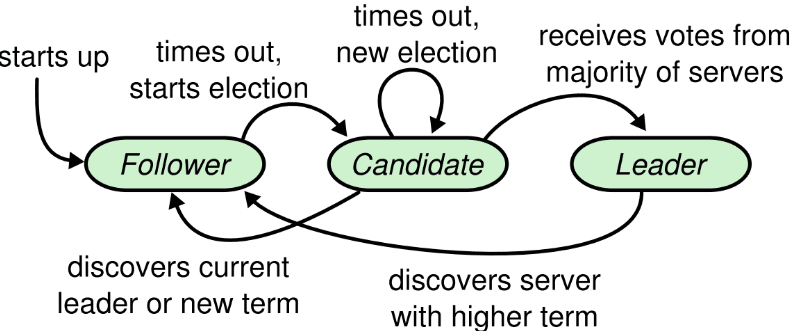
Raft将server划分为3中状态(角色)

Leader:负责和client交互，log复制，同一个cluster中最多存在1个Leader

Follower:被动相应请求RPC，从不主动发起请求RPC

Candidate:一种临时角色，只存在于Leader的选举阶段。某个节点想要变成Leader,那么就发起投票请求，同时自己变成Candidate。如果选举成功，则变成Leader，否则退回Follower。

状态角色转换流程：



算法流程:

Raft算法使用心跳机制来触发Leader选举。当服务器启动时，服务器初始状态是Follower。只要Follower从Leader或者Candidate收到有效的RPC就会保持Follower状态。如果Follower在一段时间内(该段时间被称为election timeout)没有收到消息，则它会假设当前没有可用的Leader，然后开启选举新Leader的流程。

term：Raft算法将时间划分成为任意不同长度的任期(term)。任期用连续的数字表示。每一个任期的开始都是一次选举(election)，一个或多个候选人(Candidate)试图成为Leader。若一个候选人赢得了选举，它就会在该任期的剩余时间担任Leader。在某些情况下，选票被瓜分，有可能没有选出领导人，那么将会开始下一个任期，并且开始下一次选举。Raft算法保证一个任期最多只有一个Leader

RPC: Raft算法中服务器节点之间通信使用RPC。包括三种RPC: RequestVote RPC: 候选人在选举期间发起；AppendEntries RPC:领导人发起的一种心跳机制，复制日志也在该命令中完成。InstallSnapshot RPC:领导者使用该RPC来发送快照给落后太多的Follower

选举流程：

Follower增加当前的tern,转变为Candidate;Candidate投票给自己，并发送RequestVote RPC给cluster中其他服务器；收到RequestVote的服务器，在同一tern会按照先到先得投票给至多一个Candidate。且只会投票给log至少和自身一样新的Candidate;

若节点赢得了选举。即收到大多数节点的投票，则状态转变为Leader状态；

另一个服务器成为了Leader。即收到了Leader的合法心跳包(tern值>=当前自己的tern值)，则其转变为Follower状态。

一段时间后依然没有胜者，会开启新一轮的选举。

日志复制(log replication)：主要用来保证节点的一致性

当Leader选举出来后便开始负责客户端的请求，Leader接收到客户端的日志(事务请求)后先把日志追加到本地log中，然后通过heartbeat把该entry同步给其他Follower，Follower接收到后记录日志(添加到自己的log中)，然后向Leader发送ACK，当Leader收到大多数(n/2+1)Follower的ACK信息后将日志设置为已提交并追加到本地磁盘中，通知客户端并在下个heartbeat中Leader将通知所有的Follower将该日志存储在自己本地磁盘中

脑裂(split brain):

CAP原则：Partition tolerance(分区容错) Consistency(一致性) Availability(可用性)。最多只能同时实现两点，不可能三者兼顾。

分区容忍性：当节点之间无法正常通信时，就产生了分区，而分区产生后，依然能够保证服务可用，就说系统是分区容忍的。(分布式系统中不同网络区域之间不能互相通信，网络硬件出现问题，延迟丢包)。

一致性:在分布式系统中的所有数据备份，在同一时刻是否有同样的值。(分布式系统写入和读出可能不在同一台机器上，数据一致更新，所有数据变动都是同步的)区别于数据库事务的一致性(关注数据的可见性，事务中间状态的数据对外不可见，只能看到初始状态和最终状态)。

可用性(availability):在集群中的一部分节点故障后，集群整体是否还能响应客户端的读写请求。

CAP理论就是说在分布式系统中，最多只能实现两点。由于网络硬件肯定会出现延迟丢包等问题，所以分区容错性是必须区实现的。所以只能在一致性和可用性之间权衡。

一致性和可用性地矛盾：两者不能同时成立，因为可能通信失败(出现分区容错)

如果保证G2地一致性，那么G1必须在写操作时，锁定G2地读操作和写操作。只有数据同步后，才能重新开放读写。锁定期间，G2不能读写，没有可用性。

如果保证G2可用性，那么势必不能锁定G2，所以一致性不成立

系统设计时只能选择一个目标。如果追求一致性，那么无法保证所有节点地可用性；如果追求所有节点地可用性，那就无法做到一致性。

什么场合，可用性高于一致性？

发布一张网页到CDN，多个服务器有这张网页地副本。后来发现一个错误，需要更新网页，这时只能每个服务器都更新一遍。一般来说，网页更新不是特别强调一致性。短时期内，一些用户拿到新版本，另一些拿到旧版本，问题不会特别大。当然，所有人最终都会看到新版本。这个场合就是可用性高于一致性。

CA & AP

CA: 传统关系数据库 AP：key-value数据库

BASE：Basically Available(基本可用) soft state(软状态) eventually consistent(最终一致性)

BASE是对CAP中一致性和可用性权衡的结果，核心思想是即使无法做到强一致性，但每个应用都可以采用适当方式使系统达到最终一致性。

基本可用：基本业务可用，支持分区失败

软状态：状态可以有一段时间不同步，异步。

最终一致性：最终数据一致就可以了。强一致性：即时同步，确保数据都是准确无误的。弱一致性：一个服务器的更新会导致不一致窗口，会持续一段时间，期间读取的数据是错误的。

ZooKeeper:

ZooKeeper是一个分布式协调服务，集群的管理者。监视集群中各个节点的状态。根据节点提交的反馈进行下一步合理操作。

分布式应用程序可以基于ZooKeeper实现诸如数据发布/订阅(发布者将数据发布到ZK节点上，供订阅者动态获取数据，实现数据集中式管理和动态更新。数据量很小，但是数据更新会较快的场景)，负载均衡(kafka就是利用ZK实现消费者和生产者的负载均衡)，命名服务，分布式协调/通知，集群管理，Master选举，分布式锁，分布式队列等功能。

ZooKeeper保证了如下分布式一致性:顺序一致性，原子性，可用性，实时性(最终一致性)

客户端的读请求可以被集群中的任意一台服务器处理。对于写请求，这些请求会同时发给其他Zookeeper服务器并且达成一致后，请求才会返回成功。因此，随着ZooKeeper的集群服务器增多，读请求的吞吐量会提高，但是写请求的吞吐量会下降。

有序性是ZooKeeper中非常重要的一个特性，所有的更新都是全局有序的，每个更新都有一个唯一的时间戳，这个时间戳称为zxid(ZooKeeper Transaction Id)。读请求的返回结果中会带有这个ZooKeeper最新的zxid。

ZooKeeper提供了什么： 文件系统，通知机制

为什么最好使用奇数台服务器构成ZooKeeper集群？

在ZooKeeper中Leader选举算法采用了Zab(ZooKeeper Atomic Broadcast)协议。Zab核心思想是当多数Server写成功时，则任务数据写成功。如果有3个Server则最多允许1个Server挂掉。如果有4个Server，则最多允许1个Server挂掉。既然3个或者4个Server，同样最多允许1个Server挂掉，那么它的可靠性是一样的。所以选择奇数个Zookeeper Server即可，选择3个Server

关于ZooKeeper的一些重要概念：

ZooKeeper本身就是一个分布式程序(只要半数以上节点存活，ZooKeeper就能正常服务)

为了保证高可用，最好是以集群部署ZooKeeper，这样只要集群中大部分机器是可用的(能容忍一定的机器故障)，那么ZooKeeper本身任然是可用的。

ZooKeeper将数据保存在内存中，保证了高吞吐量和低延迟(但内存限制了存储的容量不太大，此限制也是保持了Znode中存储的数据量较小的进一步原因)

ZooKeeper是高性能的在读多于写的应用程序中尤其高性能。因为写会导致所有服务器间同步状态

会话(session): 指的是ZooKeeper服务器与客户端会话。在ZooKeeper中，一个客户端链接是指客户端和服务器之间的一个TCP连接。通过这个连接，客户端能够通过心跳检测与服务器保持有效会话，也能向ZooKeeper服务器发送请求并接收响应，同时还能够通过该连接接收来自服务器的Watch事件通知。Session的sessionTimeout值用来设置一个客户端会话的超时时间。当服务器压力过大，网络故障或客户端主动断开连接，只要sessionTimeout规定的时间内能够重新连接上集群中任意一台服务器，那么之前创建的会话任然有效。服务器为客户端分配sessionID。

ZNode:两类节点(构成集群的机器，称之为机器节点；数据模型中的数据单元，称之为数据节点ZNode)

ZooKeeper将所有数据存储在内存中，数据模型是一棵树(Znode Tree),每个节点都会保存自己的数据内容，同时还会保存一系列属性信息。Node可分为持久性节点和临时性节点两类。持久性节点：一旦这个ZNode被创建了，除非主动进行ZNode的移除操作，否则这个ZNode将一直保存在ZooKeeper上。临时节点的生命周期和客户端会话绑定，一旦客户端会话失效，那么这个客户端创建的所有临时节点都会被移除。

Watcher(事件监听器)：

ZooKeeper允许用户在指定节点上注册一些Watcher,并且在一些特定事件触发的时候，ZooKeeper服务器会将事件通知到感兴趣的客户端上去

ACL：

ZooKeeper采用ACL策略来进行权限控制

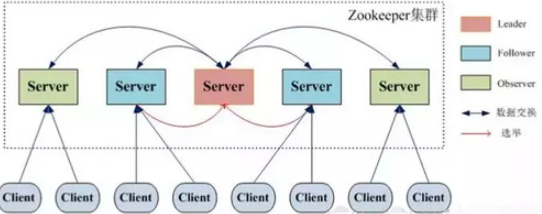
ZooKeeper特点:

顺序一致性：从同一客户端发起的事务请求，最终会严格按照顺序被应用到ZooKeeper中

原子性：所有事务请求的处理结果在整个集群中所有机器上应用情况一直

单一系统映像：无论客户端连接到哪个ZooKeeper服务器上，其看到的服务器数据都是一致的。

ZooKeeper集群角色：Leader, Follower, Observer



Leader既可以为客户端提供写服务又能提供读服务。除了Leader外，Follower和Observer都只能提供读服务。

ZAB(ZooKeeper Atomic Broadcast原子广播)协议：ZooKeeper并没有完全采用Paxos算法，而是使用ZAB协议作为其保证集群中各个副本之间的数据一致性的核心算法。特别为ZooKeeper设计的崩溃可恢复的原子消息广播算法

ZAB协议包括两种基本模式，分别是崩溃恢复和消息广播；当整个服务框架在启动过程中，或是当Leader服务器出现网络中断，崩溃退出与重启等异常情况，ZAB协议就会进入恢复模式并选举产生新的Leader服务器(集群中有过半的server与该leader server完成了状态同步，数据同步)。

消息队列：

分布式应用间交换信息的一种技术。消息队列可驻留在内存或磁盘上，队列存储消息直到它们被应用程序读走。通过消息队列，应用程序可独立运行：它们不需要直到彼此位置，或在继续执行前不需要等待接收程序接收此消息。常用的消息队列技术是Message Queue

Message Queue的通讯模式

点对点通讯：点对点方式是最为传统和常见的通讯方式。它支持一对一，一对多，多对多，多对一等多种配置方式，多种拓扑结构。

发布/订阅(publish/subscribe)模式：发布/订阅功能使消息的分发可以突破目的队列地理指向的限制，使消息按照特定的主题甚至内容进行分发，用户或者应用程序可以根据主题或内容接收到所需的消息。发布/订阅功能使得发送者和接收者之间的耦合关系变得更为松散，发送者不必关心接收者的目的地址而接收者也不必关心消息的发送地址，而是根据消息的主题进行消息的收发。

Kafka流程：

创建一条记录，对该记录要指定对应的topic和value，key和partition可选。先序列化，然后按照topic和partition，放进对应的发送队列中。Kafka生产者都是批量请求，会积攒一批，然后一起发送。不是调用send()就立刻进行网络发包。

填写key：按照key进行哈希，相同key发送到同一个partition

没填写key：通过默认的round-robin算法来选partition

这些要发往同一个partition的请求按照配置，攒一波，然后由一个单独的线程一次性发过去。

API：low level api, high level api

High level api，替我们把offset，路由都做好了；low level api需要我们子集处理offset,路由

Partition分区：

当存在多个副本的情况下，会尽量把多个副本，分配到不同的broker上。Kafka会为partition选出一个leader，之后所有该partition的请求，实际操作的都是leader，然后同步到其它的follower。当一个broker宕机后，在该宕机broker上的partition的leader都会被重新选出。

如何分配partition，如何选leader

Partition的分配(将第i个partition分配到第(i%n)个broker上，将第i个partition的第j个replica分配到第(i+j)%n个broker上)，leader的选举，有一个执行者。在Kafka中，这个执行者叫controller。Kafka使用ZK在broker中选出一个controller，用于partition分配和leader选举

Leader容灾：分区leader选举

Controller会在ZK上注册watch，一旦有broker宕机，它就能知道。当broker宕机后，controller就会给宕机的broker中的partition选出新的leader，controller先从对应partition的ISR列表中，选出一个做leader

Kafka控制器选举：

在kafka集群中会有一个或多个broker，其中有一个broker会被选举为控制器(controller)，它负责整个集群中所有分区和副本的状态。当某个分区的leader副本出现故障时，由控制器负责为该分区选举新的leader副本。当检测到某个分区的ISR集合发生变化时，由控制器负责通知所有broker更新其元数据信息。

Kafka中的控制器选举的工作依赖于ZK，成功竞选为控制器的broker会在ZK中创建/controller这个临时节点(包含自己的brokerid)。在任意时刻，集群中有且仅有一个控制器。每个broker启动时会去尝试读取/controller节点的brokerid的值，如果读取到brokerid的值不为-1，则表示已经有其它broker节点成功竞选为控制器，所以当前broker就会放弃竞选；如果ZK中不存在/controller这个节点，或者这个节点中的数据异常，那么就会尝试去创建/controller这个节点，只有创建成功的那个broker才会成为控制器。

ZK中还有一个与控制器相关的/controller\_epoch节点，节点存放controller\_epoch值。用于记录控制器发生变更的次数，即记录当前的控制器是第几代控制器(控制器纪元)。初始值为1，以后控制器发生变更时，每次选出一个新的控制器就加一。每个和控制器交互的请求都会携带controller\_epoch字段。如果请求的controller\_epoch值小于ZK中的值，则认为这个请求是无效请求。如果大于ZK中的值，那么说明已经有新的控制器当选了。由此可见，kafka通过controller\_epoch来保证控制器的唯一性，进而保证相关操作的一致性。

控制器的职责：

监听partition相关的变化。用来处理分区重分配。用来处理ISR集合变更。用来处理leader副本的选举。

监听topic相关变化。用来处理topic的增减变化

监听broker相关变化。用来处理broker增减变化。

Kafka：三大核心功能：发布和订阅数据流，类似于传统消息队列(RabbitMQ,RocketMQ)的功能；冗余备份的持久性的日志服务。处理流式数据。

为什么用kafka，为什么要使用消息队列

缓冲和削峰：kafka在中间起到一个缓冲的作用，把消息缓冲在kafka，下游服务可以按照自己的节奏慢慢处理。

解耦和扩展性：消息队列可以作为一个接口层，解耦重要的业务流程。

冗余：可以采用一对多的方式，一个生产者发布消息，可以被多个订阅topic的服务消费到供多个毫无关联的业务使用。

异步通信：消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。

Kafka中的broker是干什么的？

broker是消息的代理，生产者往brokers里面指定的topic中写消息，消费者从brokers里面指定的topic拉取消息，然后进行业务处理，broker在中间起到一个保存消息的中转站。

Kafka中的zookeeper起到什么作用，可以不用zookeeper？

ZK是一个分布式协调组件，broker依赖于ZK，ZK在kafka中用来选举kafka的控制器(它负责管理整个集群所有分区和副本的状态。当某个分区的leader出现故障，由控制器负责为该分区选举新的leader副本)和检测broker是否存活。

Kafka中的ISR，AR代表什么？ ISR的伸缩性指什么？

简单来说，分区中(partition)的所有副本统称为AR(Assigned Replicas)。所有与leader副本保持一定程度同步的副本(包括leader副本在内)组成ISR(In Sync Replicas)。ISR集合是AR集合的一个子集。消息会先发送到leader副本，然后follower副本才能从leader中拉取消息进行同步。同步期间，follower副本相对于leader副本而言会有一定程度的滞后。前面所说的“一定程度同步”是指可以忍受的滞后范围，这个范围可以通过参数进行配置。和leader副本同步滞后过多的副本(不包括leader副本)将组成OSR(Out of Sync Replied)。因此，AR= ISR+ OSR。正常情况下，所有的follower副本都应该和leader副本保持一定程度的同步，即AR= ISR，OSR集合为空。

ISR的伸缩性：

Leader副本负责维护和跟踪ISR集合中所有follower副本的滞后状态，当follower副本落后太多或失效时，leader副本会把它从ISR集合中删除。如果OSR集合中所有的follower副本追上了leader副本，那么leader副本会把它从OSR集合移到ISR集合。默认情况下，当leader副本发生故障时，只有在ISR集合中的follower副本才有资格被选举为新的leader，而在OSR集合中的副本没有任何机会(不过可以通过配置改变)。

Kafka follower如何与leader同步数据

Kafka的复制机制既不是完全的同步复制，也不是单纯的异步复制。完全同步复制要求所有的follower复制完成，这条消息才会被认为commit，这种方式极大影响了吞吐率。异步复制下，follower异步从leader复制数据，数据只要被leader写入log就被认为已经commit。这种情况下，如果leader挂掉，会丢失数据。Kafka使用ISR的方式很好均衡了确保数据不丢失以及吞吐率

Kafka为什么快？

Cache文件系统缓存

顺序写。磁盘的顺序写大多数情况下比随机写内存还要快。

零拷贝技术减少拷贝次数。

批量消息处理：合并小的请求，以流的方式进行交互。

Kafka的message格式是什么样的？

一个kafka的message由一个固定长度的header和一个变长的消息体body组成。

Kafka中的consumer group是什么概念？

逻辑上的概念。是kafka实现单播和广播两种消费模型的手段。同一个topic的数据，会广播给不同的group；同一个group中的worker,只有一个worker能拿到这个数据。group内的worker可以用多线程或多进程来实现也可以分散在多台机器上，worker的数量通常不超过partition的数量。同一个group内，一个partition只能被一个worker消费。

Kafka中的消息是否会丢失和重复消费？两方面入手：消息发送和消息消费

消息发送：两种方式同步和异步，默认是同步方式，可以通过producer.type属性配置。Kafka通过配置request.required.acks属性来确认消息的生产：0---表示不进行消息接收是否成功的确认。1---表示当leader接收成功的确认 -1---表示leader和follower都接收成功时确认。

综上所述：

acks=0,不和kafka集群进行消息接收确认，则当网络异常，缓冲区满了等情况，消息可能丢失。

acks=1,同步模式下，只有leader确认接收成功后但挂掉了，副本没有同步，数据可能丢失。

消息消费：

Low-level api:消费者自己维护offset值，可以实现对kafka的完全控制

High-level api:封装了对partition和offset的管理，使用简单

Kafka中是如何体现消息顺序性的？

Kafka每个partition中的消息在写入时都是有序的，消费时，每个partition只能被每一个group中的一个消费者消费，保证了消费时也是有序的。整个topic不保证有序。如果为了保证topic整个有序，那么将partition调整为1

Kafka leader选举机制原理：

Kafka在所有的broker中选出一个controller，所有partition的leader选举都由controller决定。Controller会将leader的改变直接通过rpc的方式通知需要为此做出响应的broker。同时controller也负责增删topic以及replica的重新分配。

当有broker挂掉了后，controller的处理过程如下：

Controller在ZK注册watch一旦有broker宕机，其在ZK上对应的znode会自动被删除，ZK会触发controller注册的watch，controller读取最新的幸存的broker。

Kafka的使用场景：

异步处理；应用解耦；流量削峰；日志处理；消息通讯；

Kafka优点和缺点：

可用性高：Kafka是分布式的，一个数据有多个副本，少数机器(broker)宕机，不会丢失数据，不会导致不可用。

时效性：ms级

由于是批量发送，数据并非真正的实时；仅仅支持同一个分区内消息有序，无法实现全局消息有序。有可能消息重复消费。依赖ZooKeeper进行元数据管理。

解释Kafka的ZooKeeper是什么?可以在没有ZooKeeper的情况下使用Kafka吗？

ZooKeeper是高性能的协调服务，它用于Kafka的分布式应用。

不能没有ZooKeeper，不能越过ZooKeeper，直接联系Kafka broker。一旦ZooKeeper停止工作，它就不能服务客户端请求。

ZooKeeper主要用于在集群中不同节点之间进行通信。分布式同步，Leader检测，复制

Kafka与传统消息系统之间有三个关键区别：

Kafka持久化日志，这些日志可以被重复读取和无限期保存。

Kafka是一个分布式系统，以集群的方式运行，可以灵活伸缩，在内部通过复制数据提升容错能力和高可用性。