**ZK设计原理简述**

**Zookeeper从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper就将负责通知已经在Zookeeper上注册的那些观察者做出相应的反应，从而实现集群中类似Master/Slave管理模式。  
ZooKeeper中有下面的应用：  
统一命名服务（NameService）  
配置管理（ConfigurationManagement）  
集群管理（GroupMembership）  
共享锁（Locks）  
队列管理**

**1. Zk数据模型和层级namespace**：  
    ZK的名称空间非常类似文件系统，名称为一系列的path，例如”/root/app/task”,ZK中每个node都是通过path来标识的。和文件系统不同的是，每个node可以有关联的data，字节点也是如此，这一点可以类比为文件系统允许path为file或者目录。ZK的设计是为了存储“协调”数据：状态信息，配置，位置信息等；所以每个节点的data都非常的小，b-kb范围。  
    Znode维护着一个“状态结构”（stat structure），其中包括data/ACL变更的version和时间戳，用来校验协调数据的变更，每次znode数据变更，将会导致version号增加，每次获取数据时，version信息也会一同获取。  
    znode中的data读写是原子性的，read获取znode中关联的所有数据，write采用replace的方式“写入”数据，每个node都有ACL（access control list）来限制操作。  
    zk还有一种node为“临时节点“（ephemeral node），这种节点的生命周期和相应的session一致，session失效后，节点会被删除；临时节点，在很多设计中具有重要的意义。  
**2. Watches**：  
    Zk提供了一种”watch“的机制，client可以在znode上设置（注册）watch。watch将会在znode节点变更时触发，当watch触发，client将会收到znode变更的数据消息。并且当client与server的链接失效后，client也会受到一个本地的通知。Watch机制为zookeeper客户端提供了一种”异步”以及callback方式数据获取,客户端可以针对自己感兴趣(手动注册的watch)或者系统事件作出相应操作.  
1) 节点变更事件类型(Watcher.Event.EventType):

* None:空事件类型,表示当前事件不是节点变更事件(极有可能是KeeperState事件)
* NodeCreated :当注册的watcher所关注的节点被创建时
* NodeDeleted :当注册的watcher所关注的节点被删除时
* NodeDataChanged :当注册的watcher所关注的节点数据被update时
* NodeChildrenChanged:当注册的watcher所关注的节点的子节点列表有变更时.

2) Server状态事件 (Watcher.Event.KeeperState):此类型事件为了告知client端,此时server端状态发生了何种变更

* Disconnected:当client端失去链接时,将会发出一个本地消息(此event非server端返回),接下来也极有可能会抛出异常(例如: ConnectionLossException, SessionExpiredException, SessionMovedException等..)
* SyncConnected:client已经成功和ZK server建立连接,通常和Follower或者Leader建立连接.
* AuthFailed:授权验证失败,比如建立连接时,链接重建或者session校验时.
* ConnectedReadOnly:client和一个read-only server建立了链接.可能因为环境问题,导致当前client无法找到ZK 环境中的”Followers”或者客户端被允许连接到Observer类型的server上.
* Expired:当前session已经过期,其数据已经被cluster移除;这种信息是非常致命的,可能是因为client端长时间离群,或者ZK 集群已经失效太久;一旦出现这个事件,唯一的办法就是new Zookeeper(…)

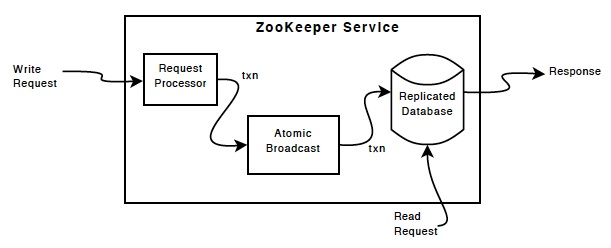
    Zk非常快速而且简单，这是它的目标，不过zk也提供了构建复杂服务的基础，例如同步，它提供了一些保证：

1. 顺序一致性：数据的更新将按照client发送请求的顺序执行。
2. 原子性：数据更新必须有确切的结果,成功或者失败。
3. 一致性视图：无论client链接哪个server，它将获得一致的服务的视图（view）。
4. 可靠性：一旦更新被执行，它将被持久存储。
5. 实时性：在一定时间内，系统保证client的视图是实时的。

**3. 简单API**：  
    ZK的一个目标就是提供简单的API接口，如下就是其支持的操作:(请参考:org.apache.zookeeper.Zookeeper)

1. create：新增节点: public String create(String path,byte[] data,List<ACL> acl,CreateMode mode),同步创建;public void create(String path,byte[] data,List<ACL> acl,CreateMode mode,StringCallback cb,Object ctx),使用callback方式.
2. delete：删除节点
3. exites：检测节点是否存在 –--支持Watcher
4. getData：获取node上关联的数据    ----支持Watcher
5. setData：重置node上关联的数据
6. sync：同步操作，等待数据的传播

**4. 实现**(implementation)：



     这种可”复制“的database(解释:我们成为replicas database,每个ZK Server都持有一个本地的全数据镜像数据库副本)以data tree的方式保存在内存中，更新操作首先被持久在log中以便恢复；write在应用在内存之前首先被存储在disk中。每个ZK server都能服务clients，client向其链接的server提交请求，对于read请求将会直接从本地”复制“的内存数据库中获取；对于更改服务状态的write请求，将会采用agreement protocal(ZK中采用二步提交方式)。agreement protocal将会把client提交的write请求转发给单一的server,即leader。对于其他的folower，接受leader的消息提议，同意其消息传输。消息层会关注leader失效时的交替以及folower和leader的数据同步。  
    Zk使用自定义的原子性消息传播机制（ZAB），因为消息层是原子性的，ZK能够确保本地备份不会”偏离“；当leader接受到write请求后，在应用write操作时它会”计算“系统的状态，并以事务的方式更新状态。  
    Zk本身并不是数据库，也不是为数据存储而生，所以znode上挂载的数据尽量的小，默认配置为1M，一般我们建议此数据应该尽量的小，因为存储太大的数据将会造成server的操作耗时（数据库在集群中传输时间较长，网络环境复杂也会增加数据传输出错的机会），带来请求响应延迟增大；如果你的应用需要ZK维护一些较大的数据，建议数据存储采用其他的文件系统(或数据库)，ZK的node上只存储文件的引用位置。

**5. Sequence Nodes**：  
    对于普通的节点创建,我们指定path,这适合一般场景;不过,在创建节点时，你可以要求ZK为你在path的结尾追加一个自增的计数。此计数器对于其父节点而言是唯一的。计数器的格式为%010d—10个数字前缀用0来填充：000000001，其最大值为Integer.MAX\_VALUE.  
  
**6. Zk的时间跟踪**：

1. zid：事务ID，每个数据变更操作的请求，ZK Leader都会为其生成一个全局唯一的zid，zid也标识这请求变更的顺序，如果zid1 < zid2，则认为zid1在zid2之前发生。
2. version number：对node的任何更新都将会导致其version增加，每个节点都有3种version: version(znode节点数据变更)，cversion(子节点变更)，aversion(此节点ACL变更)。
3. Ticks：当使用多个zk server时，server需要使用ticks来声明事件的耗时，比如状态装载，session过期，peer间的链接超时等。
4. real time：ZK不使用绝对时间，对于数据的变更使用了时间戳。

**7. ZK stat 结构**：

* czxid:node创建时的zxid。
* mzxid：znode的数据最后更新的zxid
* ctime：此节点创建时来自epoch的time毫秒数
* mtime：znode最后更新的毫秒数，来自epoch
* version：znode数据更新的版本号
* cversion:znode的字节点变更的版本号
* aversion：znode的ACL变更的版本号
* ephemeralOwner:临时节点的session id，如果不是临时节点，将为0
* dataLength：znode中挂载的数据的长度
* numChildren：znode中子节点的个数

    ZK 数据结构,请参考:

    1) org.apache.zookeeper.server.DataNode; 数据的最小表示单元,每个path最终将会以DataNode来表示

    2) org.apache.zookeeper.server.DataTree; DataNode的维护关系,根据path层级关系将DataNode逻辑上存储为Tree

    3) org.apache.zookeeper.server.ZKDatabase: ZK本地数据库,存储了DataTree,ACL,Sessions列表,Wathers列表等.

    4) org.apache.zookeeper.server.SessionTracker.Session:客户端和server的链接信息

    5) org.apache.zookeeper.server.persistence.FileTxnLog: ZK server所接收到的变更操作,以日志方式存储