1. Zk

可以理解成 分布式 文件数据库 ，使用时就是在指定目录下创建不同类型的文件 来满足不同分布式场景下的需要 拥有

临时节点、临时有序节点

永久节点 、永久有序节点

带过期时间的永久节点 (3.5.3 版本后新增)

带过期时间的顺序永久节点(3.5.3 版本后新增)

注意：

1. 带过期时间的功能需要手动打开 。 编辑： zkServer.sh 脚本
2. 当没有被删除时 如果从新操作此节点， 时间会被重置
3. 带过期时间删除是定时任务删的 。 因此不是到了过期时间立即 删除而是过期时间到了，在下次定时任务中被删除

容器节点 (3.5.3 版本后新增)

用保存数据或者子节点 ，zk 会定时检查(60s一次)容器节点下是否有子节点

如果没有子节点则会进行删除

1. ZK 能保证强一致性(cp) 非事务操作有 follower完成、事务性操作有 master完成
2. ZK 临时节点 ： 客户端与server 异常断开后 并不会立即删除而是通过设置的过期 时间开始倒计时 ， 默认是10s。当过期时间 == 0 时 ，服务端会删除临时节 点 。

除非 quit 主动断开才会立即删除临时节点

2、Zk 特性

* 1. 全局数据一致性：每台server上的数据都是一致的（文件夹下的数据都是一样的）
  2. 可靠性：如果消息（对目录中的数据进行增删改查）被其中一台服务器接受 ，那么将会被所有服务器接受然后一起操作
  3. 顺序性：包括全局有序 和 偏序。

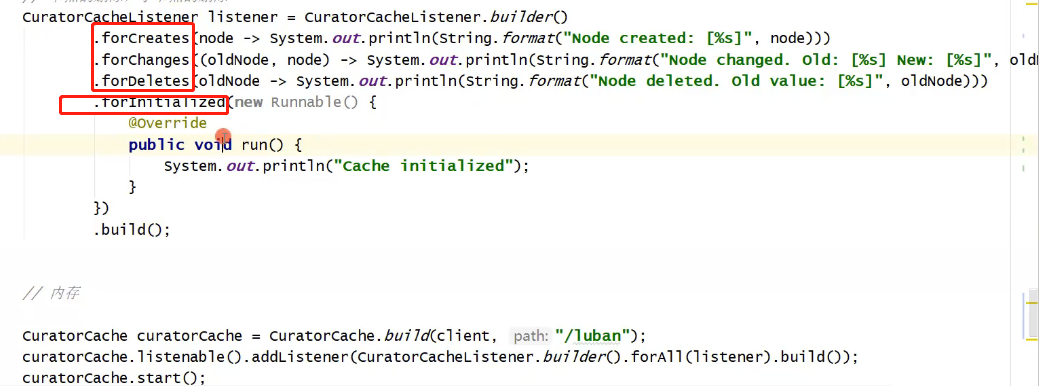
假如再server1上先创建1.txt,再创建2.txt 。那么每台server创建文件夹的顺序都是一样的

* 1. 原子性：要么成功（成功的标准半数server节点成功，只要半数节点成功后面的节点会自动同步），要么失败
  2. 实时性：数据的实时

3、Zk 的crud

* 1. 查询

1. ls 路径 查看节点名称
2. get 路径 查看节点保存的值
3. stat 路径 查看节点的属性
4. Ephemeralowner 0 ： 永久节点 1：临时节点
5. .dataVersion 版本号 ，每操作一次进行 + 1
6. .cVersion 子节点版本号，每操作一次(包括删除子节点)子节点进行 + 1
7. mTime 节点修改时间
8. .cTime 子节点修改时的时间
9. **mZxid 当前节点的事务id ，所有数据的节点mZxid是一致的**
10. **cZxid 当前节点的子节点事务id 所有数据的节点mZxid是一致的**
    1. 创建
11. 永久 ：create 路径 data
12. 有序 ：create -s 路径 data
13. 临时 ：create -e 路径 data
14. 容器节点： create -c 路径 data
15. 带国过期时间 ： create -t 时间 路径 data
    1. 删除 ： (3.5后 rmr删除节点命令已被删除)
16. delete 路径 (只能删除没有 子节点的 节点)
17. deleteall 路径 (删除当前节点及子节点)
    1. 修改 ： set 路径 data
18. Curator 客户端的使用
    1. 监听分类
19. . 一次性监听 (使用少)
20. . 不是一次性的 一直监听
21. nodeCache 监听某个节点数据的变化(已过期)
22. pathChildrenCache 监听某个节点子节点数据的变化(已过期)
23. treeCache 监听节点 + 子节点数据的变化(已过期)
24. CuratorCache(新版本) 并且支持缓存监听节点的数据
25. .CuratorCache 会查询节点内容,并自动缓存 然后回调forInitialized 方法
26. . 增删改 会触发各自的监听方法 forCreate、fordelete、forChange



* 1. 分布式锁实现步骤

1. **.创建锁对象 创建 /locks 节点 ，在此节点下创建临时有序节点**

*InterProcessMutex lock = new InterProcessMutex(client, "/locks");*

1. **上锁**

*lock.acquire();*

*lock.acquire(3,秒); //带阻塞时间*

1. **释放锁**

*lock.release();*

1. Zk 集群中的角色
   1. Leader
   2. Follower
   3. Obsever

可以在集群中 指定每个节点是什么角色 。如果不指定 则自动分配

其中 leader 是需要选举的 ， leader 是如何进行选举的 ？

1. **启动选举流程：**

根据集群配置的 myid ， myid越大 启动的节点都会投给启动后的大的myid

但是一旦 leader确定后 (启动2台就会确定leader，myid大的那个) 再有较大的 myid节点启动， 此时不会进行leader变更

1. **运行时leader下线，重新选举流程**

假如集群中的zxid是100 ， 当事务操作请求leader时 zxid + 1 = 101 并记录一次 操作的日志，在数据还未同步到其他follower的情况下 ， leader挂了。 此 时 会返回客户端为eroor

当leader下线后 ， 剩下follower节点就会重新进行投票选举， 会投给 myid 最 大的节点作为leader。 当挂掉的leader重启后 会作为follower加入集群

加入集群时 会跟leader比较zxid 如果zxid不一致则会 进行数据同步 新选举的 leader 的zxid还是100，新的follower zxid=101让自己的节点的数据和 leader保 持一致(同步日志信息) 从而zxid 也保持一致

1. .