一、基础知识

1.介绍

Zookeeper 是 Google 的 Chubby一个开源的实现，最初是 Hadoop 的分布式协调服务。它包含一个简单的原语集，分布式应用程序可以基于它实现同步服务，配置维护和命名服务等

2.节点类型

ZooKeeper中的节点有两种，分别为临时节点（**Ephemeral Nodes**）和永久节点（**Persistent Nodes**）和时序节点（**SEQUENTIAL**），一般是结合在一起使用，可以生成一下4中节点类型。节点的类型在创建时即被确定，并且不能改变。

- 持久节点（PERSISTENT）：所谓持久节点，是指在节点创建后，就一直存在，直到有删除操作来主动清除这个节点——不会因为创建该节点的客户端会话失效而消失。
- 持久顺序节点（PERSISTENT_SEQUENTIAL）：这类节点的基本特性和持久节点一致。额外的特性是，在ZK中，每个父节点会为它的第一级节点维护一份时序，会记录每个子节点创建的先后顺序。基于这个特性，在创建子节点的时候，可以设置这个属性，那么在创建节点过程中，ZK会自动为给定节点加上一个数字后缀，作为新的节点名。这个数字后缀的范围是整数的最大值。
- 临时节点：该节点的生命周期依赖于创建他们的回话。一旦会话（Session）结束，临时节点将被自动删除，当然也可以手动删除。虽然每个临时的Znode都会绑定到一个客户端会话，但它们对所有的客户端还是可见的。另外，ZooKeeper的临时节点不允许拥有子节点。
- 临时顺序节点（EPHEMERAL_SEQUENTIAL）：可以用来实现分布式锁。客户端调用create()方法创建名为“_locknode_/guid-lock-”的节点，需要注意的是，这里节点的创建类型需要设置为EPHEMERAL_SEQUENTIAL。客户端调用getChildren(“_locknode_”)方法来获取所有已经创建的子节点，注意，这里不注册任何Watcher。客户端获取到所有子节点path之后，如果发现自己在步骤1中创建的节点序号最小，那么就认为这个客户端获得了锁。如果在步骤3中发现自己并非所有子节点中最小的，说明自己还没有获取到锁。此时客户端需要找到比自己小的那个节点，然后对其调用exist()方法，同时注册事件监听。之后当这个被关注的节点被移除了，客户端会收到相应的通知。这个时候客户端需要再次调用getChildren(“_locknode_”)方法来获取所有已经创建的子节点，确保自己确实是最小的节点了，然后进入步骤3。
- 永久节点：该节点的生命周期不依赖于会话，并且只有在客户端显示执行删除操作的时候，它们才能被删除。

为了保证子节点名字唯一性和顺序性，ZooKeeper还引入了顺序节点（Sequence Nodes）这个概念，前面两种节点只要在路径后缀加上10位数字就可以有顺序了。这个数字是由内部计数器（counter）来实现的，格式为%010d，即10位整数，不满10位左边补0。计数范围不能超过整数类型（4字节）表示的最大值21 4748 3647，否则会溢出。

3.一致性协议

为了解决分布式一致性问题，最著名的就是二阶段提交协议、三阶段提交协议和Paxos算法。

（1）2PC

在分布式系统中，每一个机器节点虽然都能够明确地知道自己在进行事务操作过程中的结果是成功或是事变，但却无法直接获取到其他分布式节点的操作结果。因此，当一个事务操作需要跨越多个分布式节点的时候，为了保持事务处理的ACID特性，就需要引入一个称为“协调者（Coordinator）”的组件来同一调度所有分布式节点的执行逻辑，这些被调度的分布式节点则被称为“参与者（Participant）”。协调者负责调度参与者的行为，并最终决定这些参与者是否要把事务真正进行提交。基于这一思想，衍生出了二阶段提交和三阶段提交两种协议。

2PC，是Two-Phase Commit的缩写，即二阶段提交，是计算机网络尤其是在数据库领域内，为了使基于分布式系统架构下的所有节点在进行事务处理过程中能够保持原子性和一致性而设计的一种算法。

（2）3PC

（3）Paxos算法

一、分布式协调技术

分布式协调技术主要用来解决分布式环境中多个进程之间的同步控制，让他们有序地去访问某种临界资源，防止造成“脏数据”的后果。单机环境下可以利用简单的读写锁机制来实现，在分布式环境下就需要使用分布式锁，这个分布式锁也就是分布式协调技术实现的核心内容。

分布式锁面临的问题就是网络的不可靠性（拜占庭将军问题）。如在单机环境中，进程对一个资源的获取要么成功，要么失败。但在分布式环境中，对一个资源的访问或者一个服务的调用，如果返回失败消息，但也可能实际访问成功或者调用成功了，或者时间上不同的两个节点对另外一个节点顺序调用，那么请求调用一定是按照顺序到达的吗？这些都涉及网络问题，所以分布式协调远比在同一台机器上对多个线程的调度要难得多，于是一种通用性好、伸缩性好、高可靠、高可用的协调机制应运而生——ZooKeeper。

ZooKeeper包含一个可以基于它实现同步的简单原语集，一般用于分布式应用程序服务、配置维护和命名服务等，它具有如下特征：

- ZooKeeper是简单的。
- ZooKeeper是富有表现力的。
- ZooKeeper具有高可用性。
- ZooKeeper采用松耦合交互方式。
- ZooKeeper是一个资源库。

拜占庭将军问题：是由莱斯利兰伯特提出的点对点通信中的基本问题。在分布式计算中，不同的计算机透过信息交换，尝试达成共识；但有时候，系统上协调计算机或成员计算机可能因系统错误并交换错误的信息，导致影响最终的系统一致性。

1.角色

在ZooKeeper集群中有Leader何Follower两种角色。Leader可以接收Client请求，也接受其他Server转发的写请求，负责更新系统状态。Follower也可以接收Client请求，如果是写请求将转发给Leader来更新系统状态，读请求则由Follower的内存数据库直接响应。ZooKeeper集群组成图如下：

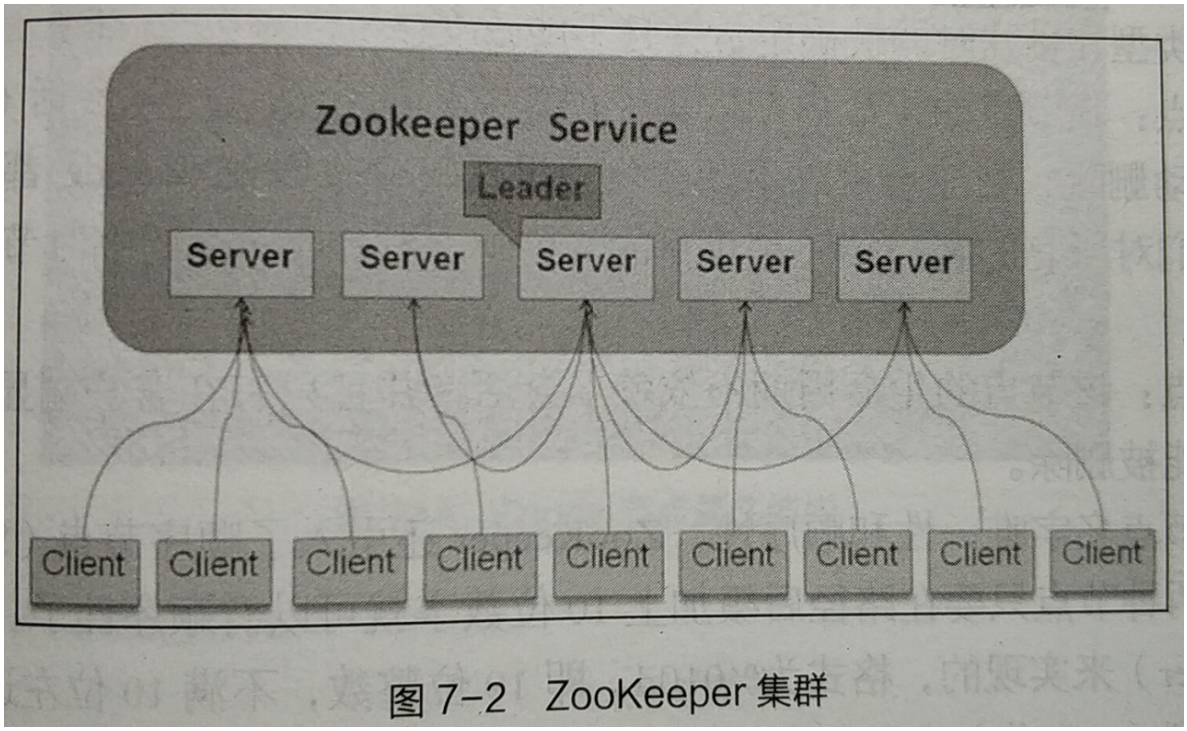

图 7-2 ZooKeeper 集群

改变Server状态的写请求，需要通过一致性协议来处理，这个协议就是Zab（ZooKeeper Atomic Broadcast）协议，用来作为其一致性复制的核心。简单来说，Zab协议规定：来自Client的所有写请求，都要转发个体ZK服务中唯一的Server——Leader，由Leader根据该请求发起一个Proposal。然后其他的Server对该Proposal进行投票。之后，Leader对投票进行收集，当投票数量过半时，Leader会向所有的Server发送一个通知消息。最后，当Client所连接的Server收到该消息时，会把该操作更新到内存中并对Client的写请求做出回应。

2.ZooKeeper中的数据模型

（1）Znode

ZooKeeper的数据存储结构和标准系统文件非常类似，都是采用的树形层次结构，而树当中的每个节点被称作Znode。他通过绝对路径的引用，和UNIX类似，必须由斜杠开头，路径的表示也是唯一的，使用如下命令可以列出根目录下的所有Znode。

```
1 ls /
```

```
[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper]
```

其中/zookeeper文件用来保存ZooKeeper的配额管理信息，不能轻易删除。就像Java中的file既可以是目录也可以是文件，Znode具有文件和目录两种特点，既向文件一样维护着数据、元信息、ACL、时间戳等数据结构，又像目录一样可以作为路径标识的一部分。每个Znode都由三部分组成。

- stat：此为状态信息，描述该Znode的版本、权限等信息
- data：与该Znode关联的数据
- children：该Znode下的子节点

ZooKeeper虽然可以关联一些数据，但并没有被设计为常规的数据库或者大数据存储，相反的是，它用来管理调度数据，如分布式应用中的配置文件信息、状态信息、汇集位置等。这些数据的共同特性就是他们都是很小的数据，通常以KB为单位。ZooKeeper的服务器和客户端都被设计为严格检查并限制每个Znode的数据至多1MB，但常规使用中应该远小于此

值。

客户端在节点上设置watch，称之为监视器。当节点状态发生改变时（Znode的增、删、改）将会出发watch所对应的操作。当watch被触发时，ZooKeeper将会向客户端发送且仅发送一条通知，因为watch只能被触发一次，这样可以减少网络流量。

（2）ZooKeeper中的时间

ZooKeeper有多重记录时间的形式，其中包括以下两个主要属性。

Zxid:

致使ZooKeeper节点状态改变的每一个操作都将使节点接收到一个Zxid格式的时间戳，并且这个时间戳全局有序。也就是说，每个对节点的改变都将产生一个唯一的Zxid。如果Zxid1的值小于Zxid2的值，那么Zxid1所对应的事件发生在Zxid2所对应的事件之前。实际上，ZooKeeper的每个节点维护着三个Zxid值，分别为cZxid、mZxid、pZxid。

- cZxid：是每个节点的创建时间所对应的Zxid格式时间戳
- mZxid：是节点的修改时间所对应的Zxid格式时间戳
- pZxid：是最近一次子节点修改时间所对应的Zxid格式时间戳

实现中Zxid是一个64位的数字，它高32位是epoch用来表示leader关系是否改变，每次一个leader被选出来，它都会有一个新的epoch。低32位是递增计数。

版本号:

对节点的每一个操作都将致使这个节点的版本号增加。每个节点维护着三个版本号，它们分别如下:

- version：节点数据版本号
- cversion：子节点版本号
- aversion：节点所拥有的的ACL版本号

（3）Znode节点属性

通过“get ”命令可以获取该路径的Znode属性，如下图:

```
[zk: localhost:2181(CONNECTED) 10] get /zk
mydata
cZxid = 0x3
ctime = Mon Apr 15 22:56:07 CST 2019
mZxid = 0x3
mtime = Mon Apr 15 22:56:07 CST 2019
pZxid = 0x3
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
[zk: localhost:2181(CONNECTED) 11]
```

属性描述如下:

属性	描述

属性	描述
cZxid	节点被创建的zxid
mZxid	节点被修改的zxid
ctime	节点被创建时间
	节点被修改的时间
pZxid	最近一次子节点被修改修改的zxid
cversion	节点所拥有的的子节点被修改的版本号
aclversion	节点所拥有的的ACL版本号
	如果此节点为临时节点，那么它的值为这个节点拥有者的会话ID；否则，它的值为0
dataLength	节点数据长度
	节点拥有的子节点的数量（不包括子节点的子节点）

（4）watch触发器

ZooKeeper可以为所有的读操作设置watch，这些读操作包括exists()、getChildren()及getData()。watch事件是一次性的触发器，当watch的对象状态发生改变时，将会出发此对象上watch所对应的事件。watch事件将衣服地发送给客户端，并且ZooKeeper为watch机制提供了有序的一致性保证。理论上，客户端接收watch事件的时间要快于其看到watch对象状态变化的时间。

watch类型：

ZooKeeper所管理的watch可以分为两类：

- 数据watch（data watches）：getData和exists负责设置数据watch
- 孩子watch（child watches）：getChildren负责设置孩子watch

可通过操作返回的数据来设置不同类型的watch。

- getData和exists：返回关于节点的数据信息
- getChildren：返回孩子列表

因此，一个成功的setData操作将处罚Znode的数据watch，一个成功的create操作将触发Znode的数据watch以及孩子watch，一个成功的delete操作将触发Znode的数据watch以及孩子watch。

表 7-2　watch 设置操作及相应的触发器

设置 watch	watch 触发器				
	create		delete		setData
	Znode	child	Znode	child	Znode
exists	NodeCreated		NodeDeleted		NodeDataChanged
getdata			NodeDeleted		NodeDataChanged
getChildren		NodeChildrenChanged	NodeDeleted	NodeDeletedChanged	

watch由客户端所连接的ZooKeeper服务器在本地维护，因此watch可以非常容易地设置、管理和分派。当客户端连接到一个新的服务器时，任何的会话事件都将可能触发watch。另外，当从服务器断开连接的时候，watch将不会被接收。但是，当一个客户端重新建立连接的时候，任何先前注册过的watch都会被重新注册。

3.常用shell操作

```
查看指定路径下的Znode列表：
ls /
创建一个新的Znode节点“zk”及和它相关的字符
create /zk mydata
获取节点信息
get /zk
修改和“zk”相关的字符
set /zk newData
删除Znode
delete /zk
添加watch触发器
get /zk watch

ZooKeeper -server host:port cmd args
        stat path [watch]
        set path data [version]
        ls path [watch]
        delquota [-n|-b] path
        ls2 path [watch]
        setAcl path acl
        setquota -n|-b val path
        history
        redo cmdno
        printwatches on|off
        delete path [version]
        sync path
        listquota path
        rmr path
        get path [watch]
        create [-s] [-e] path data acl
        addauth scheme auth
        quit
        getAcl path
        close
        connect host:port
```

二、应用场景

1.分布式锁

基于ZooKeeper分布式锁的流程：

- 在zookeeper指定节点（locks）下创建临时顺序节点node_n
- 获取locks下所有子节点children
- 对子节点按节点自增序号从小到大排序
- 判断本节点是不是第一个子节点，若是，则获取锁；若不是，则监听比该节点小的那个节点的删除事件
- 若监听事件生效，则回到第二步重新进行判断，直到获取到锁

关键在于监听前一个Znode的状态变化。

具体实现如下：

```
package distributedLock;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;

import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooDefs;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.data.Stat;

public class ZooKeeperDistributedLock implements Lock,Watcher{
    private ZooKeeper zk = null;
    // 根节点
    private String ROOT_LOCK = "/locks";
    // 竞争的资源
    private String lockName;
    // 等待的前一个锁
    private String WAIT_LOCK;
    // 当前锁
    private String CURRENT_LOCK;
    // 计数器
    private CountDownLatch countDownLatch;
    private int sessionTimeout = 30000;
    private List<Exception> exceptionList = new
ArrayList<Exception>();

    public ZooKeeperDistributedLock(String config, String
lockName) {
```

```
        this.lockName = lockName;
        try {
            // 连接zookeeper，添加一个watcher
            zk = new ZooKeeper(config, sessionTimeout, this);
            Stat stat = zk.exists(ROOT_LOCK, false);
            if (stat == null) {
                // 如果根节点不存在，则创建根节点
                zk.create(ROOT_LOCK, new byte[0],
ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (KeeperException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void process(WatchedEvent event) {
        if(this.countDownLatch != null){
            this.countDownLatch.countDown();
        }
    }

    @Override
    public void lock() {
        try {
            if (exceptionList.size() > 0) {
                throw new LockException(exceptionList.get(0));
            }
            if(this.tryLock()){

System.out.println(Thread.currentThread().getName()+"  "+ lockName
+ " 成功获取锁");
                return;
            }else{
                // 等待锁
                waitForLock(WAIT_LOCK, sessionTimeout);
            }
        } catch (KeeperException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void lockInterruptibly() throws InterruptedException {
        this.lock();
    }

    @Override
    public boolean tryLock() {
        try {
```

```
            String splitStr = "_lock_";
            if (lockName.contains(splitStr)) {
                throw new LockException("锁名有误");
            }
            // 创建临时有序节点
            CURRENT_LOCK = zk.create(ROOT_LOCK + "/" + lockName +
splitStr, new byte[0], ZooDefs.Ids.OPEN_ACL_UNSAFE,
CreateMode.EPHEMERAL_SEQUENTIAL);
            System.out.println(Thread.currentThread().getName()+"
"+CURRENT_LOCK+" 已经创建");
            // 取所有子节点
            List<String> subNodes = zk.getChildren(ROOT_LOCK,
false);
            // 取出所有lockName的锁
            List<String> lockObjects = new ArrayList<String>();
            for (String node : subNodes) {
                String _node = node.split(splitStr)[0];
                if (_node.equals(lockName)) {
                    lockObjects.add(node);
                }
            }
            // 按照升序的形式对获取的锁进行排序
            Collections.sort(lockObjects);
            // 若当前节点为最小节点，则获取锁成功
            if (CURRENT_LOCK.equals(ROOT_LOCK + "/" +
lockObjects.get(0))) {
                return true;
            }
            // 若不是最小节点，则找到自己的前一个节点
            String prevNode =
CURRENT_LOCK.substring(CURRENT_LOCK.lastIndexOf("/") + 1);
            // 获取当前需要等待哪个节点
            WAIT_LOCK =
lockObjects.get(Collections.binarySearch(lockObjects, prevNode) -
1);
        } catch (KeeperException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return false;
    }

    @Override
    public boolean tryLock(long time, TimeUnit unit)
            throws InterruptedException {
        try{
            if (this.tryLock()) {
                return true;
            }
            return waitForLock(WAIT_LOCK, time);
        }catch(Exception e){
        }
        return false;
    }
```

```
135
136     // 等待锁
137     private boolean waitForLock(String prev, long waitTime) throws
    KeeperException, InterruptedException{
138         // 向服务器注册watch
139         Stat stat = zk.exists(ROOT_LOCK + "/" + prev, true);
140         if(stat != null){
141             System.out.println(Thread.currentThread().getName() +
    "等待锁 " + ROOT_LOCK + "/" + prev);
142             this.countDownLatch = new CountDownLatch(1);
143             // 计数等待，若等到前一个节点消失，则precess中进行
    countDown，停止等待，获取锁
144             this.countDownLatch.await(waitTime,
    TimeUnit.MILLISECONDS);
145             this.countDownLatch = null;
146             System.out.println(Thread.currentThread().getName()+"
    "+CURRENT_LOCK+" 已经创建");
147         }
148         return true;
149     }
150
151     @Override
152     public void unlock() {
153         try {
154             System.out.println("释放锁 " + CURRENT_LOCK);
155             zk.delete(CURRENT_LOCK, -1);
156             CURRENT_LOCK = null;
157             zk.close();
158         } catch (InterruptedException e) {
159             e.printStackTrace();
160         } catch (KeeperException e) {
161             e.printStackTrace();
162         }
163     }
164
165     @Override
166     public Condition newCondition() {
167         return null;
168     }
169
170     public class LockException extends RuntimeException {
171         private static final long serialVersionUID = 1L;
172         public LockException(String e){
173             super(e);
174         }
175         public LockException(Exception e){
176             super(e);
177         }
178     }
179 }
```

```
1 package distributedLock;
2
3 public class ZooKeeperWatchTest {
```

```
    static int n = 500;

    public static void secskill() {
        System.out.println(Thread.currentThread().getName() +"中n="+ (--n));
    }

    public static void main(String[] args) {

        Runnable runnable = new Runnable() {
            public void run() {
                ZooKeeperDistributedLock lock = null;
                try {
                    lock = new ZooKeeperDistributedLock("localhost:2181", "test1");
                    lock.lock();

 System.out.println("========================="+Thread.currentThread().getName() + "正在运行");
                    secskill();
                } finally {
                    if (lock != null) {
                        lock.unlock();
                    }
                }
            }
        };

        for (int i = 0; i < 10; i++) {
            Thread t = new Thread(runnable);
            t.start();
        }
    }
}
```

三、集群搭建

一下是在单机上构建zookeeper集群

1.配置zoo.cfg文件

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
# 数据目录，根据需要自行修改
```

```
dataDir=D:/Program Files/zookeeper/zookeeper1/data
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60

# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
#
http://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance
#
# The number of snapshots to retain in dataDir
#autopurge.snapRetainCount=3
# Purge task interval in hours
# Set to "0" to disable auto purge feature
#autopurge.purgeInterval=1
# 配置集群，server.A=B:C:D，A代表的是server的id（myid文件），B代表的是域名或者ip地址
# C代表的是和leader服务器进行数据交换的端口，D表示的是执行选举时服务器相互通信的端口
server.1=localhost:2880:2888
server.2=localhost:3880:3888
server.3=localhost:4880:4888
```

2.新建data文件夹

新建data文件夹，并向data文件夹中新建myid的文件，文件内容为”当前这个server的id（id在集群中唯一不重复）“

3.启动

在windows中运行zkServer.cmd就可以成功启动了，在linux中使用zkServer.sh start启动

四、Leader节点选举

从ZooKeeper 3.4.0版本开始，选举算法只保留了TCP版本的FastLeaderElection选举算法。

FastLeader选主算法：

首先给出几个名词定义： （1）Serverid：在配置server时，给定的服务器的标示id。（2）Zxid：服务器在运行时产生的数据id，zxid越大，表示数据越新。 （3）Epoch：选举的轮数，即逻辑时钟。随着选举的轮数++ （4）Server状态：LOOKING,FOLLOWING,OBSERVING,LEADING

步骤：

1. Server刚启动（宕机恢复或者刚启动）准备加入集群，此时读取自身的zxid等信息。

2. 所有Server加入集群时都会推荐自己为leader，然后将（leader id 、 zixd 、 epoch）作为广播信息，广播到集群中所有的服务器(Server)。然后等待集群中的服务器返回信息。
3. 收到集群中其他服务器返回的信息，此时要分为两类：该服务器处于looking状态，或者其他状态。

   （1）服务器处于Looking状态

   首先判断逻辑时钟Epoch；

   a）如果接收到Epoch大于自己目前的逻辑时钟（说明自己所保存的逻辑时钟落伍了）。更新本机逻辑时钟Epoch，同时 Clear其他服务发送来的选举数据（这些数据已经OUT了）。然后判断是否需要更新当前自己的选举情况（一开始选择的leader id是自己）。判断规则rules judging：保存的zxid最大值和leader Serverid来进行判断的。先看数据zxid，数据zxid大者胜出；其次再判断leader Serverid, leader Serverid大者胜出；然后再将自身最新的选举结果(也就是上面提到的三种数据（leader Serverid，Zxid，Epoch）广播给其他server）

   b）如果接收到的Epoch小于目前的逻辑时钟。说明对方处于一个比较OUT的选举轮数，这时只需要将自己的 （leader Serverid，Zxid，Epoch）发送给他即可。 c）如果接收到的Epoch等于目前的逻辑时钟。再根据a）中的判断规则，将自身的最新选举结果广播给其他 server。

   同时Server还要处理2种情况：

   a）如果Server接收到了其他所有服务器的选举信息，那么则根据这些选举信息确定自己的状态（Following,Leading），结束Looking，退出选举。

   b）即使没有收到所有服务器的选举信息，也可以判断一下根据以上过程之后最新的选举leader是不是得到了超过半数以上服务器的支持，如果是则尝试接受最新数据，倘若没有最新的数据到来，说明大家都已经默认了这个结果,同样也设置角色退出选举过程。

   （2）服务器处于其他状态（Following, Leading）

   a）如果逻辑时钟Epoch相同，将该数据保存到 recvset，如果所接收服务器宣称自己是leader,那么将判断是不是有半数以上的服务器选举它，如果是则设置选举状态退出选举过程

   b）否则这是一条与当前逻辑时钟不符合的消息，那么说明在另一个选举过程中已经有了选举结果，于是将该选举结果加入到outofelection集合中，再根据outofelection来判断是否可以结束选举，如果可以也是保存逻辑时钟，设置选举状态，退出选举过程。

以上就是FAST选举过程。

zookeeper开机顺序为：先开server1和server2，过了会儿，再开server3。有了如下日志输出：

server3：

```
# 表示有一个新的选举
2019-05-24 11:20:33,044 [myid:3] - INFO
[QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@813] -
New election. My id =  3, proposed zxid=0x0
```

```
2019-05-24 11:20:33,044 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
2019-05-24 11:20:33,259 [myid:3] - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@847] - Notification time out: 400
2019-05-24 11:20:33,259 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
2019-05-24 11:20:33,665 [myid:3] - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@847] - Notification time out: 800
2019-05-24 11:20:33,665 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
2019-05-24 11:20:34,475 [myid:3] - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@847] - Notification time out: 1600
2019-05-24 11:20:34,475 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
2019-05-24 11:20:36,086 [myid:3] - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@847] - Notification time out: 3200
2019-05-24 11:20:36,086 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
2019-05-24 11:20:39,291 [myid:3] - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@847] - Notification time out: 6400
2019-05-24 11:20:39,291 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
2019-05-24 11:20:45,694 [myid:3] - INFO [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:FastLeaderElection@847] - Notification time out: 12800
2019-05-24 11:20:45,694 [myid:3] - INFO [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LOOKING (my state)
```

```
16 2019-05-24 11:22:14,777 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LOOKING (n.state), 1 (n.sid), 0x0 (n.peerEpoch) LOOKING
   (my state)
17 2019-05-24 11:22:14,777 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
18 2019-05-24 11:22:14,777 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
19 2019-05-24 11:22:14,777 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
20 2019-05-24 11:22:14,777 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
21 2019-05-24 11:22:14,787 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
22 2019-05-24 11:22:14,787 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
23 2019-05-24 11:22:14,787 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), FOLLOWING (n.state), 1 (n.sid), 0x1 (n.peerEpoch)
   LOOKING (my state)
24 2019-05-24 11:22:17,217 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LOOKING (n.state), 2 (n.sid), 0x0 (n.peerEpoch) LOOKING
   (my state)
25 2019-05-24 11:22:17,227 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch) LOOKING
   (my state)
26 2019-05-24 11:22:17,227 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch) LOOKING
   (my state)
```

```
27 2019-05-24 11:22:17,227 [myid:3] - INFO
   [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:QuorumPeer@980] -
   FOLLOWING
28 2019-05-24 11:22:17,227 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch)
   FOLLOWING (my state)
29 2019-05-24 11:22:17,237 [myid:3] - INFO
   [QuorumPeer[myid=3]/0:0:0:0:0:0:0:0:4181:Learner@86] - TCP NoDelay
   set to: true
30 2019-05-24 11:22:17,237 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch)
   FOLLOWING (my state)
31 2019-05-24 11:22:17,237 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch)
   FOLLOWING (my state)
32 2019-05-24 11:22:17,237 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch)
   FOLLOWING (my state)
33 2019-05-24 11:22:17,247 [myid:3] - INFO
   [WorkerReceiver[myid=3]:FastLeaderElection@595] - Notification: 1
   (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1
   (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch)
   FOLLOWING (my state)
```

server2:

```
2019-05-24 11:18:11,108 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 2 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LEADING (n.state), 2 (n.sid), 0x1 (n.peerEpoch) LEADING (my state)
2019-05-24 11:20:33,044 [myid:2] - INFO [localhost/127.0.0.1:3888:QuorumCnxManager$Listener@743] - Received connection request /127.0.0.1:51870
# 收到其他节点的选举请求
2019-05-24 11:22:17,217 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
2019-05-24 11:22:17,217 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
2019-05-24 11:22:17,227 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
2019-05-24 11:22:17,227 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
2019-05-24 11:22:17,227 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
2019-05-24 11:22:17,227 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
2019-05-24 11:22:17,227 [myid:2] - INFO [WorkerReceiver[myid=2]:FastLeaderElection@595] - Notification: 1 (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round), LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) LEADING (my state)
```

server1:

```
1 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
2 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
3 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
4 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
5 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
6 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
7 2019-05-24 11:22:14,777 [myid:1] - INFO
  [WorkerReceiver[myid=1]:FastLeaderElection@595] - Notification: 1
  (message format version), 3 (n.leader), 0x0 (n.zxid), 0x1 (n.round),
  LOOKING (n.state), 3 (n.sid), 0x0 (n.peerEpoch) FOLLOWING (my state)
```