Zookeeper

OutLine

- Zookeeper概述
- Zookeeper应用场景
- 【基础实践】Zookeeper编程

分布式系统的问题

- 与单机系统不同
  - 内存地址一致
  - 单机出问题概率低
- 分布式系统
  - 一致性问题
  - 容灾容错
  - 执行顺序问题
  - 事务性问题

Zookeeper 分布式协调服务

不同机器,不同进程,不同的内存数据

ZK的一致性保证:
顺序一致性:Zk按照客户端发送请求的顺序更新数据.比较zxid.
原子性:数据要么更新成功,要么失败.
单一性:访问集群中任意节点得到的数据是同样的.
可靠性:一旦数据更新成功,持久化存储.
实时性：Zookeeper保证客户端将在一个时间间隔范围内获得服务器的更新信息或者服务器失效的信息。

核心问题

- 没有一个全局的主控，协调或控制中心
  - 单机不可靠
  - 环中有环
- 特殊场景下，特殊实现，但不通用
- 我们需要什么？
  - 一个松散耦合的分布式系统中粗粒度锁以及可靠性存储（低容量）的系统

运行在集群之上，通过paxos算法实现选主逻辑，写数据的时候和leader 交互.

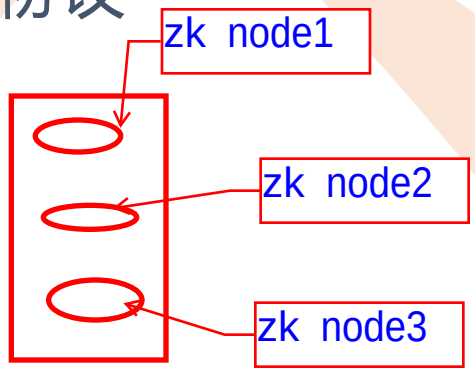
zk的交互支持参与者不了解对方.

Zookeeper

- Hadoop系统
- 开源的，高效的，可靠的协同工作系统
- 名字服务器，分布式同步，组服务
- Google内部实现叫Chubby
- 基于Paxos协议

数据保存在内存中,同时记录操作日志+内存快照(二进制)

帕索斯协议



ZK写数据机制:
所有写请求转发个Leader,Leader采取两阶段提交的方式:
1:先本地生成自增的zxid,(如果Leader挂了,后续选主的时候要用到zxid.)生成Proposal日志(持久化)
2:广播所有的Flower节点,并有单独的线程统计Ack Proposl的数量
3:Proposal ack 数量过半以后,认为写入成功,广播commit,并把这个request丢到各自的CommitProcessor里面处理
4:MasterCommit日志,更新lastCommitZxid,并把数据apply到内存树中,返回Client操作成功.
备注:如果Leader挂了 选主 先比较zxid 如果有一个node的zxid是最新的 则这个node直接成为leader,如果有多个节点都有最新的zxid,则比较node 的Id 就是 zk 的myid文件里面配的那个

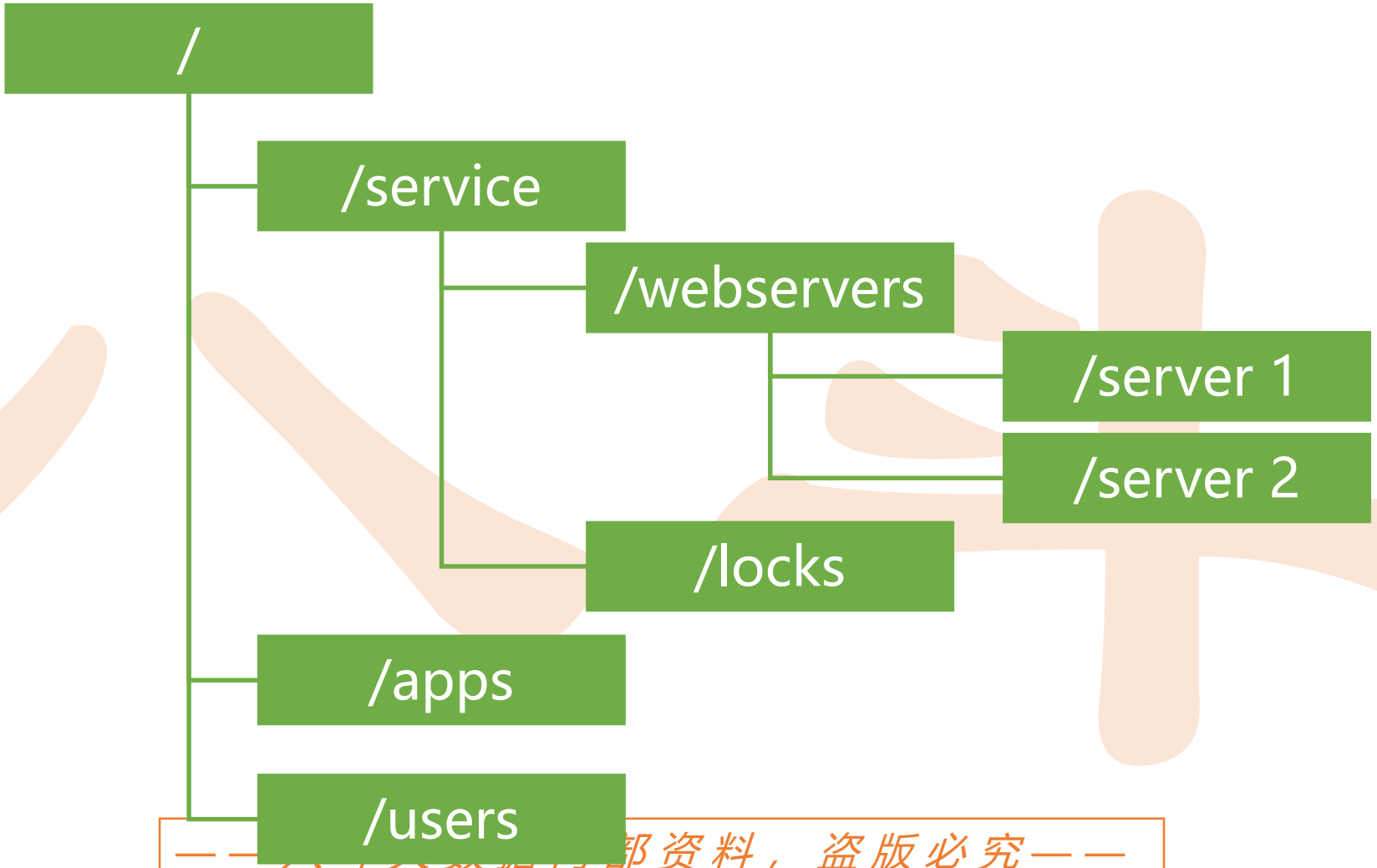
尝试解释一下zk保证数据一致性模型:
数据一致性是靠Paxos算法保证的，Paxos可以说是分布式一致性算法的鼻祖，是ZooKeeper的基础
1:写请求进来,Leader广播给所有Flower节点, node1,node2写入成功后会有心跳建立 我俩写完了 并且超过半数了,Leader节点可以返回给client了,node3写入失败 发现没有节点和自己抱团了,会主动把自己下了 不对外提供服务.这样client在读数据的时候 只有1,2 提供服务 数据是对的.

Zookeeper概述

Zk直接两种模式：默认模式（CP模式 选举时停止读写请求）、Readonlymode模式（AP模式 选举时停止写请求，但是可以读）

- 数据模型
  - 命名空间
  - 与标准文件系统很相似
  - 以 / 为间隔的路径名序列组成
  - 只有绝对路径，没有相对路径
- 每个节点自身的信息
  - 数据
  - 数据长度
  - 创建时间
  - 修改时间
- 具有文件，路径的双重特点

Zookeeper结构

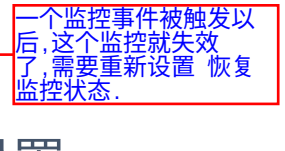


Zookeeper概述

- Persistent Nodes
  - 永久有效地节点,除非client显式的删除,否则一直存在
- Ephemeral Nodes
  - 临时节点,仅在创建该节点client保持连接期间有效,一旦连接丢失,zookeeper会自动删除该节点
- Sequence Nodes
  - 顺序节点,client申请创建该节点时,zk会自动在节点路径末尾添加递增序号,这种类型是实现分布式锁,分布式queue等特殊功能的关键

Zookeeper概述

- 监控机制(watch)
  - 数据节点上设置
  - 客户端被动收到通知
  - 各种读请求，如getData()，getChildren()，和exists()
- 三个关键点
  - 一次性监控，触发后，需要重新设置
  - 保证先收到事件，再收到数据修改的信息
  - 传递性
    - 如create会触发节点数据监控点，同时也会触发父节点的监控点
    - 如delete会触发节点数据监控点，同时也会触发父节点的监控点



Zookeeper概述

- 风险
  - 客户端有可能看不到所有数据的变化
  - 多个事件的监控，有可能只会触发一次
    - 一个客户端设置了关于某个数据点exists和getData的监控，则当该数据被删除的时候，只会触发“文件被删除”的通知。
  - 客户端网络中断的过程的无法收到监控的窗口时间，要由模块进行容错设计

Zookeeper概述

- 数据访问
  - 每个节点上的“访问控制链”(ACL, Access Control List)保存了各客户端对于该节点的访问权限。
  - 用一个三元组来定义客户端的访问权限：(scheme:expression, perms)
    - ip:19.22.0.0/16, READ）表示IP地址以19.22开头的主机有该数据节点的读权限。

Zookeeper概述

- 数据访问

权限	描述	备注
CREATE	有创建子节点的权限	
READ	有读取节点数据和子节点列表的权限	
WRITE	有修改节点数据的权限	无创建和删除子节点的权限
DELETE	有删除子节点的权限	
ADMIN	有设置节点权限的权限	

zookeeper本身提供了ACL机制，表示为 scheme:id:permissions，第一个字段表示采用哪一种机制，第二个id表示用户，permissions表示相关权限（如只读，读写，管理等）

Zookeeper概述

模式	描述
World	它下面只有一个id, 叫anyone, world:anyone代表任何人，zookeeper中对所有人有权限的结点就是属于world:anyone的
Auth	已经被认证的用户（可以用过用户名：密码的方式,kerberos）
Digest	通过username：password字符串的MD5编码认证用户
Host	匹配主机名后缀，如，host:corp.com匹配host:host1.corp.com, host:host2.corp.com，但不能匹配host:host1.store.com
IP	通过IP识别用户，表达式格式为 addr/bits

Zookeeper概述

```
public class NewDigest {

    public static void main(String[] args) throws Exception {
        List<ACL> acls = new ArrayList<ACL>();
        //添加第一个id，采用用户名密码形式
        Id id1 = new Id("digest",
                DigestAuthenticationProvider.generateDigest("admin:admin"));
        ACL acl1 = new ACL(ZooDefs.Perms.ALL, id1);
        acls.add(acl1);
        //添加第二个id，所有用户可读权限
        Id id2 = new Id("world", "anyone");
        ACL acl2 = new ACL(ZooDefs.Perms.READ, id2);
        acls.add(acl2);

        // zk用admin认证，创建/test ZNode。
        ZooKeeper zk = new ZooKeeper("host1:2181,host2:2181,host3:2181", 2000, null);
        zk.addAuthInfo("digest", "admin:admin".getBytes());
        zk.create("/test", "data".getBytes(), acls, CreateMode.PERSISTENT);
    }
}
```

Zookeeper概述

- 一致性保证
  - 序列一致性：客户端发送的更新将按序在Zookeeper进行更新
  - 原子一致性：更新只能成功或者失败，没有中间状态
  - 单系统镜像：无论连接哪台Zookeeper服务器，客户端看到的服务器数据一致
  - 可靠性：任何一个更新成功后都会持续生效，直到另一个更新将它覆盖。可靠性有两个关键点: 第一，当客户端的更新得到成功的返回值时，可以保证更新已经生效，但在某些异常情况下（超时，连接失败），客户端无法知道更新是否成功；第二，当更新成功后，不会回滚到以前的状态，即使是在服务器失效重启之后
  - 实时性：Zookeeper保证客户端将在一个时间间隔范围内获得服务器的更新信息，或者服务器失效的信息。但由于网络延时等原因，Zookeeper不能保证两个客户端能同时得到刚更新的数据，如果需要最新数据，应该在读数据之前调用sync()接口

Zookeeper概述

- String create(String path, byte[] data, List<ACL> acl, CreateMode createMode)
  - 创建一个给定的目录节点 path, 并给它设置数据，CreateMode 标 识有四种形式的目录节点，分别是 PERSISTENT：持久化目录节点，这个目录节点存储的数据不会丢失；PERSISTENT_SEQUENTIAL：顺序自动编号的目录节点，这种目 录节点会根据当前已近存在的节点数自动加 1，然后返回给客户端已经成功创建的目录节点名；EPHEMERAL：临时目录节点，一旦创建这个节点的客户端与服务器端口也就是 session 超时，这种节点会被自动删除；EPHEMERAL_SEQUENTIAL：临时自动编号节点

Zookeeper概述

- Stat exists(String path, boolean watch)
  - 判断某个 path 是否存在，并设置是否监控这个目录节点，这里的 watcher 是在创建 ZooKeeper 实例时指定的 watcher
- Stat exists(String path, Watcher watcher)
  - 重载方法，这里给某个目录节点设置特定的 watcher，Watcher 在 ZooKeeper 是一个核心功能，Watcher 可以监控目录节点的数据变化以及子目录的变化，一旦这些状态发生变化，服务器就会通知所有设置在这个目录节点上的 Watcher，从而每个客户端都很快知道它所关注的目录节点的状态发生变化，而做出相应的反应

Zookeeper概述

- void delete(String path, int version)
  - 删除 path 对应的目录节点，version 为 -1 可以匹配任何版本，也就删除了这个目录节点所有数据
- List<String> getChildren(String path, boolean watch)
  - 获取指定 path 下的所有子目录节点，同样 getChildren方法也有一个重载方法可以设置特定的 watcher 监控子节点的状态
- Stat setData(String path, byte[] data, int version)
  - 给 path 设置数据，可以指定这个数据的版本号，如果 version 为 -1 怎可以匹配任何版本

不为-1的时候，需要和节点当前版本做一个比对，做乐观锁控制.

Zookeeper概述

- byte[] getData(String path, boolean watch, Stat stat)
  - 获取这个 path 对应的目录节点存储的数据，数据的版本等信息可以通过 stat 来指定，同时还可以设置是否监控这个目录节点数据的状态
- void addAuthInfo(String scheme, byte[] auth)
  - 客户端将自己的授权信息提交给服务器，服务器将根据这个授权信息验证客户端的访问权限。
- List<ACL> getACL(String path, Stat stat)
  - 获取某个目录节点的访问权限列表

Zookeeper概述

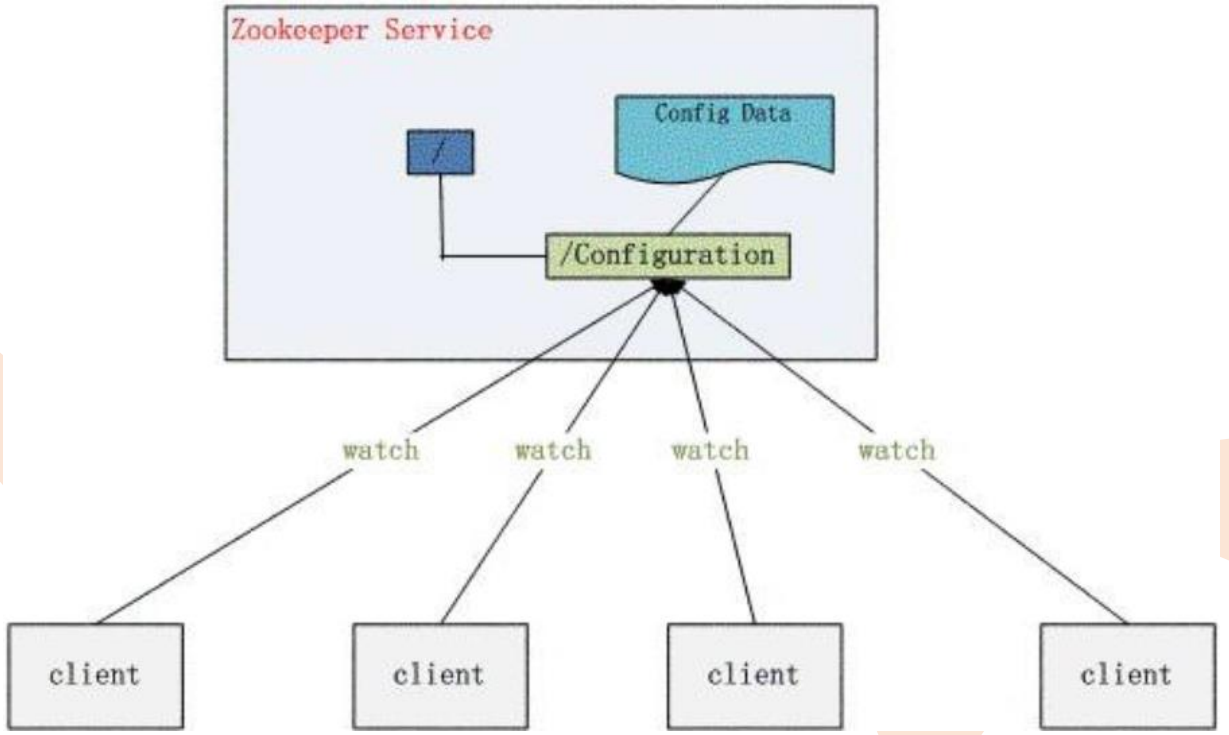
- Stat setACL(String path, List<ACL> acl, int version)
  - 给某个目录节点重新设置访问权限，需要注意的是 Zookeeper 中的目录节点权限不具有传递性，父目录节点的权限不能传递给子目录节点。目录节点 ACL 由两部分组成：perms 和 id。 Perms 有 ALL、READ、WRITE、CREATE、DELETE、ADMIN 几种 而 id 标识了访问目录节点的身份列表，默认情况下有以下两种： ANYONE_ID_UNSAFE = new Id("world", "anyone") 和 AUTH_IDS = new Id("auth", "") 分别表示任何人都可以访问和创建者拥有访问权限。

OutLine

- Zookeeper概述
- Zookeeper应用场景
- 【基础实践】Zookeeper编程

Zookeeper应用场景

- 配置管理（Configuration Management）
  - 全局的系统配置
  - 容错并且统一



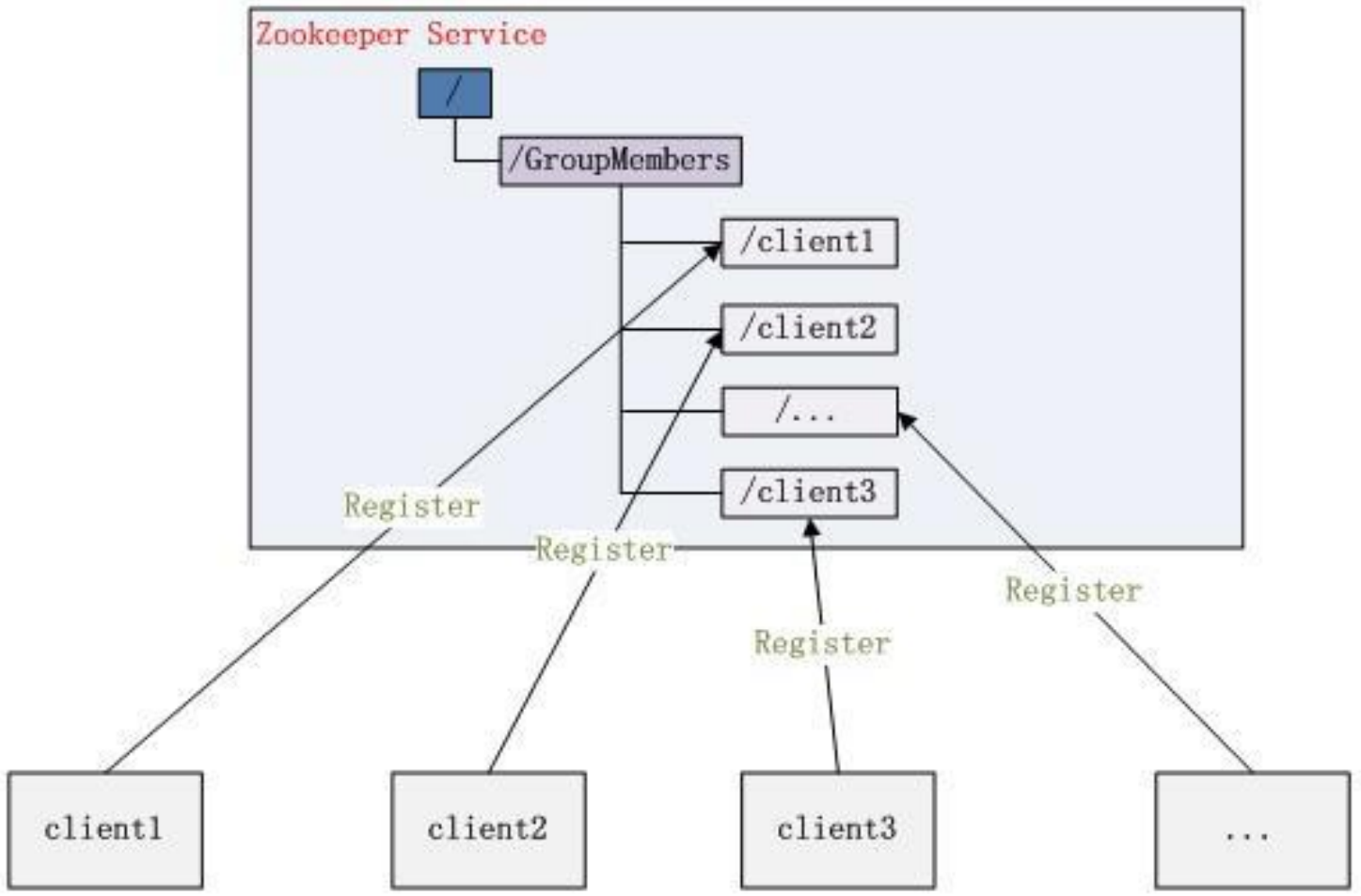
Zookeeper应用场景

- 集群管理（Group Membership）
  - 了解每台服务器的状态
  - 新增删除服务，需要周知
  - 选择一个主服务器

Zookeeper应用场景

- 集群管理（Group Membership）-集群状态
  - EPHEMERAL节点
  - 所有的server getChildren(String path, boolean watch) 方法
  - 某台服务器下线，对应的节点自动删除

Zookeeper应用场景

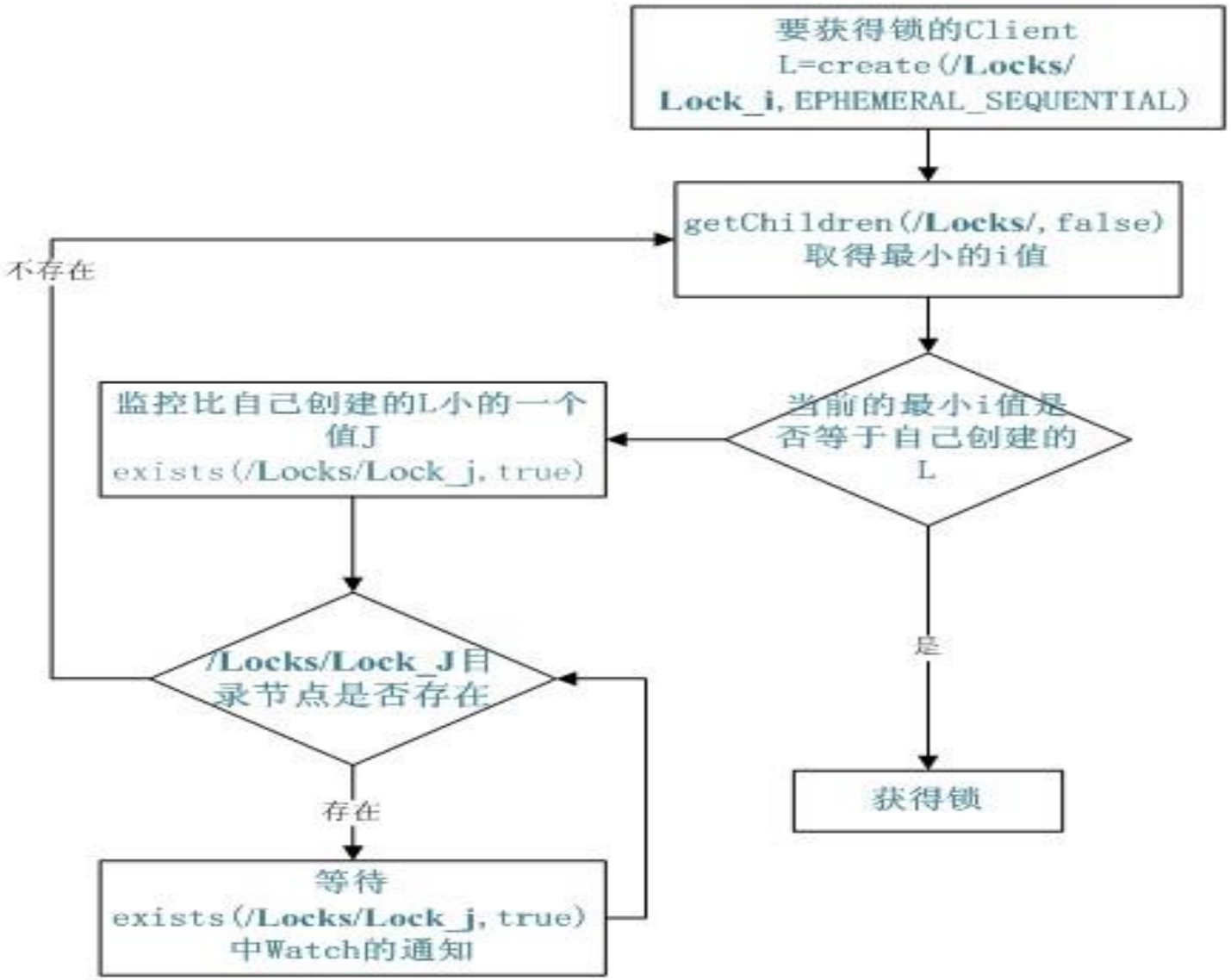


Zookeeper应用场景

- 集群管理（Group Membership）-选主节点
  - EPHEMERAL_SEQUENTIAL
  - 选择当前是最小编号的 Server 为 Master
  - 最小编号的 Server 死去，由于是 EPHEMERAL节点，死去的Server 对应的节点也被删除，所以当前的节点列表中又出现一个最小编号的节点

Zookeeper应用场景

- 共享锁（Locks）

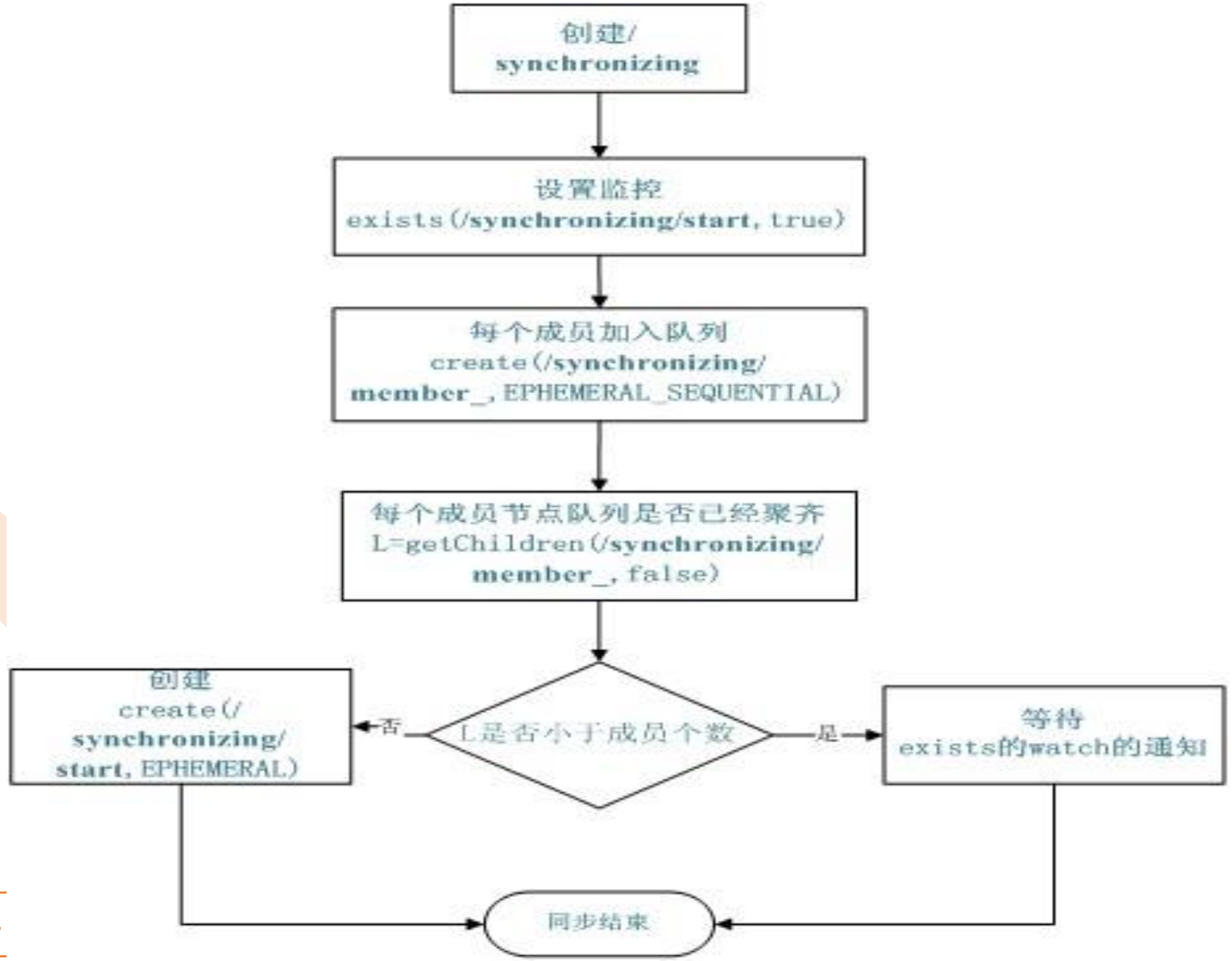


Zookeeper应用场景

- 队列管理
  - 同步队列
    - 所有成员都聚齐才可使用
  - FIFO队列
    - 生产消费者

Zookeeper应用场景

- 同步队列



Zookeeper应用场景

- FIFO
  - 创建 SEQUENTIAL 类型的子目录 /queue_i，这样就能保证所有成员加入队列时都是有编号的，出队列时通过getChildren( ) 方法可以返回当前所有的队列中的元素，然后消费其中最小的一个，这样就能保证 FIFO。

OutLine

- Zookeeper概述
- Zookeeper应用场景
- 【基础实践】Zookeeper编程

Zookeeper使用

- zkServer.sh start
- zkServer.sh status
- zkServer.sh stop
- zkCli.sh -server zookeeper:2181

- 执行客户端 zkCli.sh
  - ls / 查看当前目录
  - create /text "test" 创建节点
  - create -e /text "test" 创建临时节点
  - create -s /text "test" 创建序列节点
  - get /test 查看节点

Q & A

@八斗学院