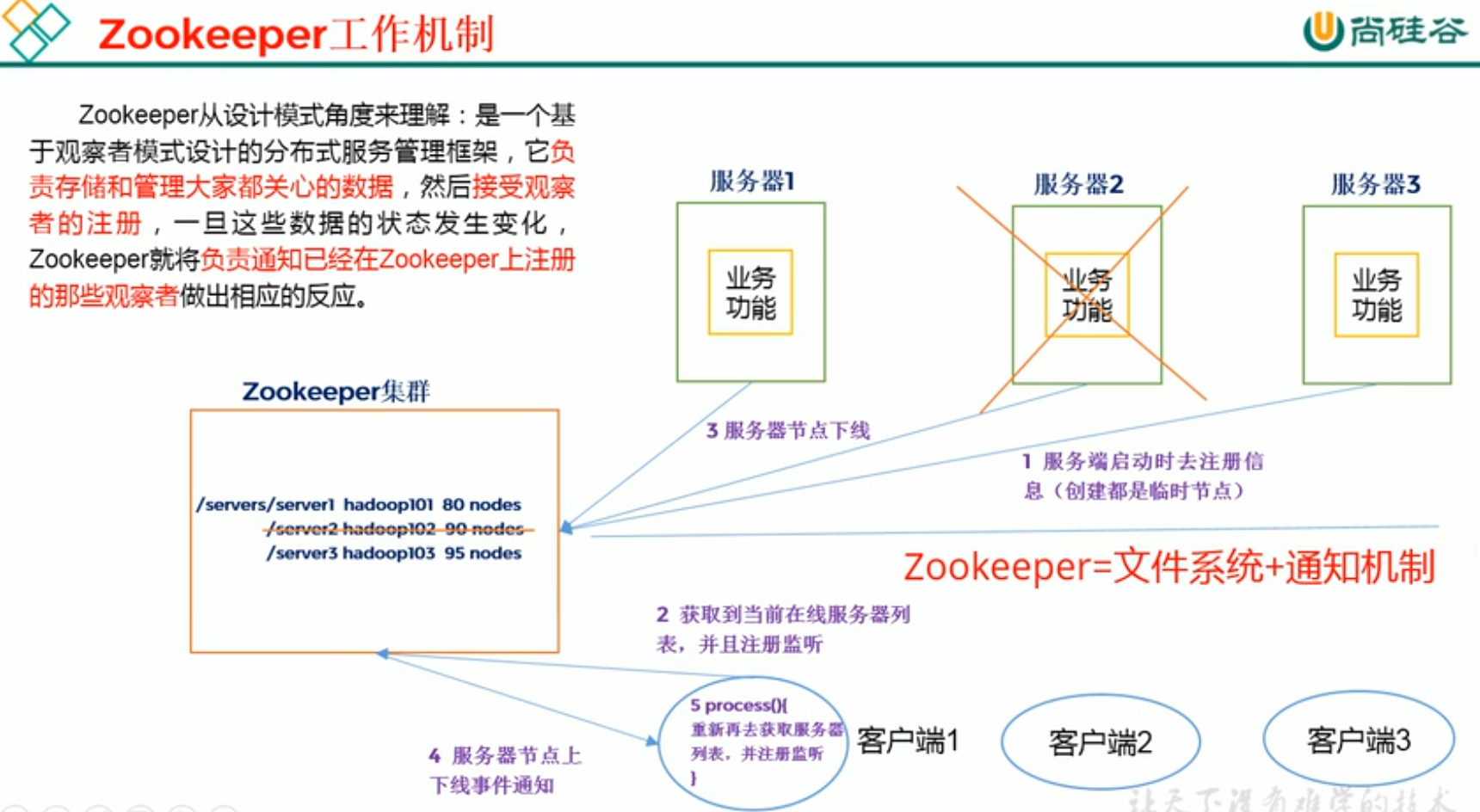
# Zookeeper

# 1 Zookeeper入门

## 1.1 概述

zookeeper是一个开源的分布式的协调服务管理者，本质上：一个小型的文件存储系统。目录树的数据存储，对树中的节点进行有效管理，从而来维护和监控存储的数据的状态变化，以此达到对数据的集群管理。如统一命名、分布式配置管理、分布式消息队列、分布式锁、分布式协调、服务器节点动态上下线、软负载均衡等。

**zookeeper=文件系统+通知机制**



观察者就是各个客户端

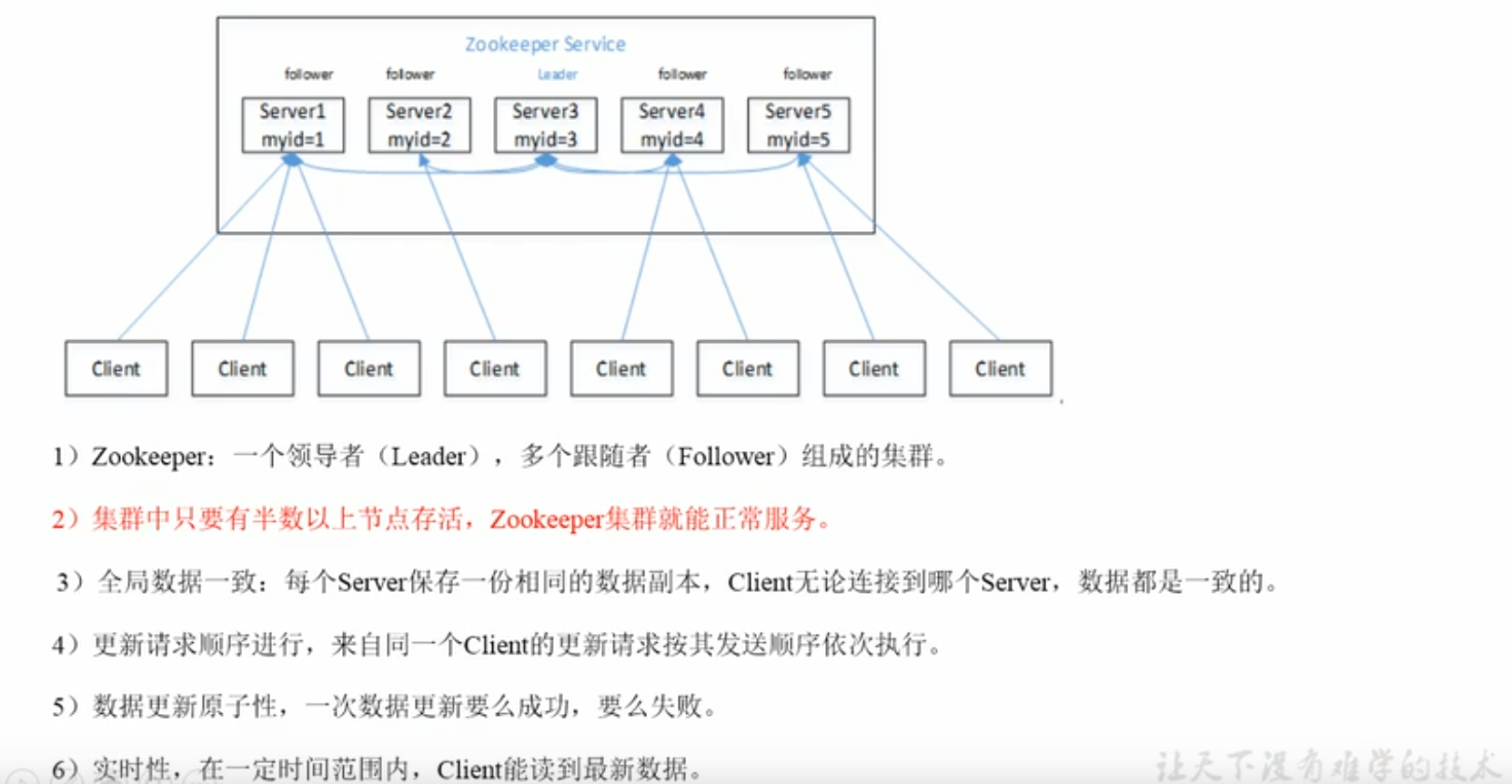
## 1.2 特点

1）全局数据一致，事务操作都给leader，统一编号、操作

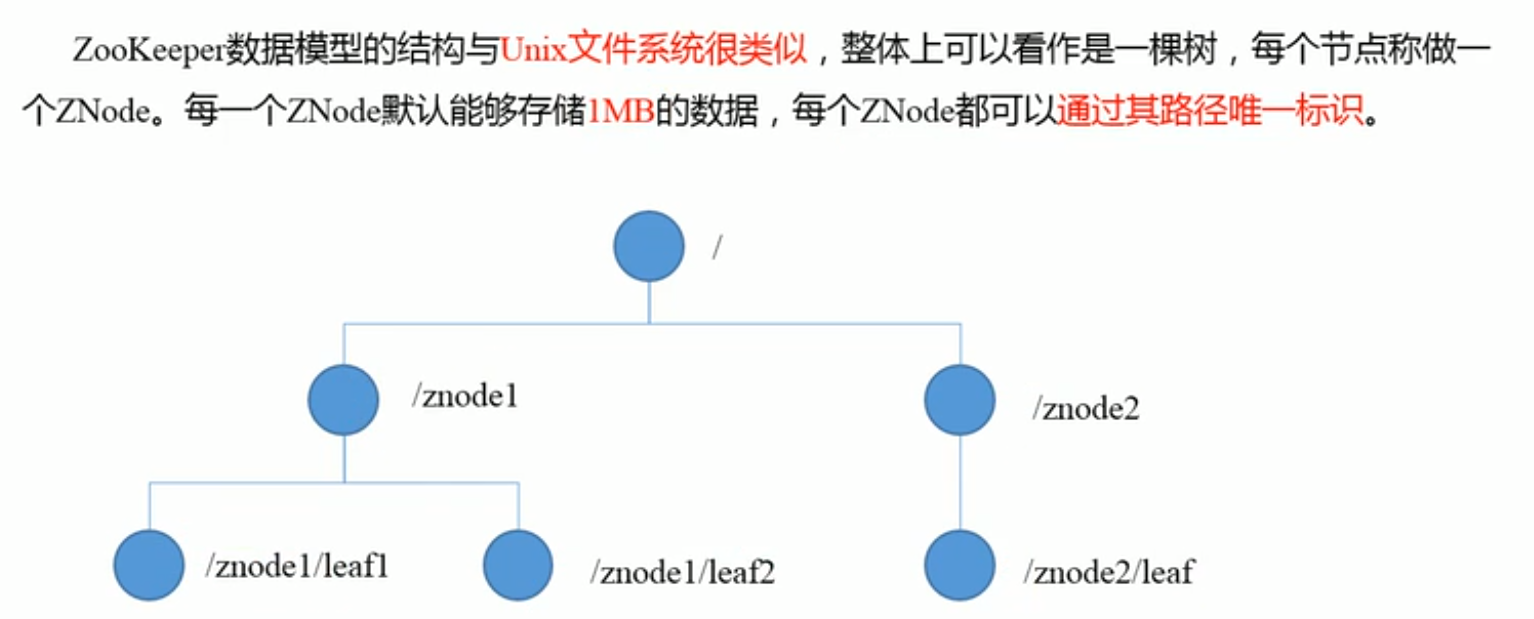
2）可靠性

3）顺序性

4）数据更新原子性



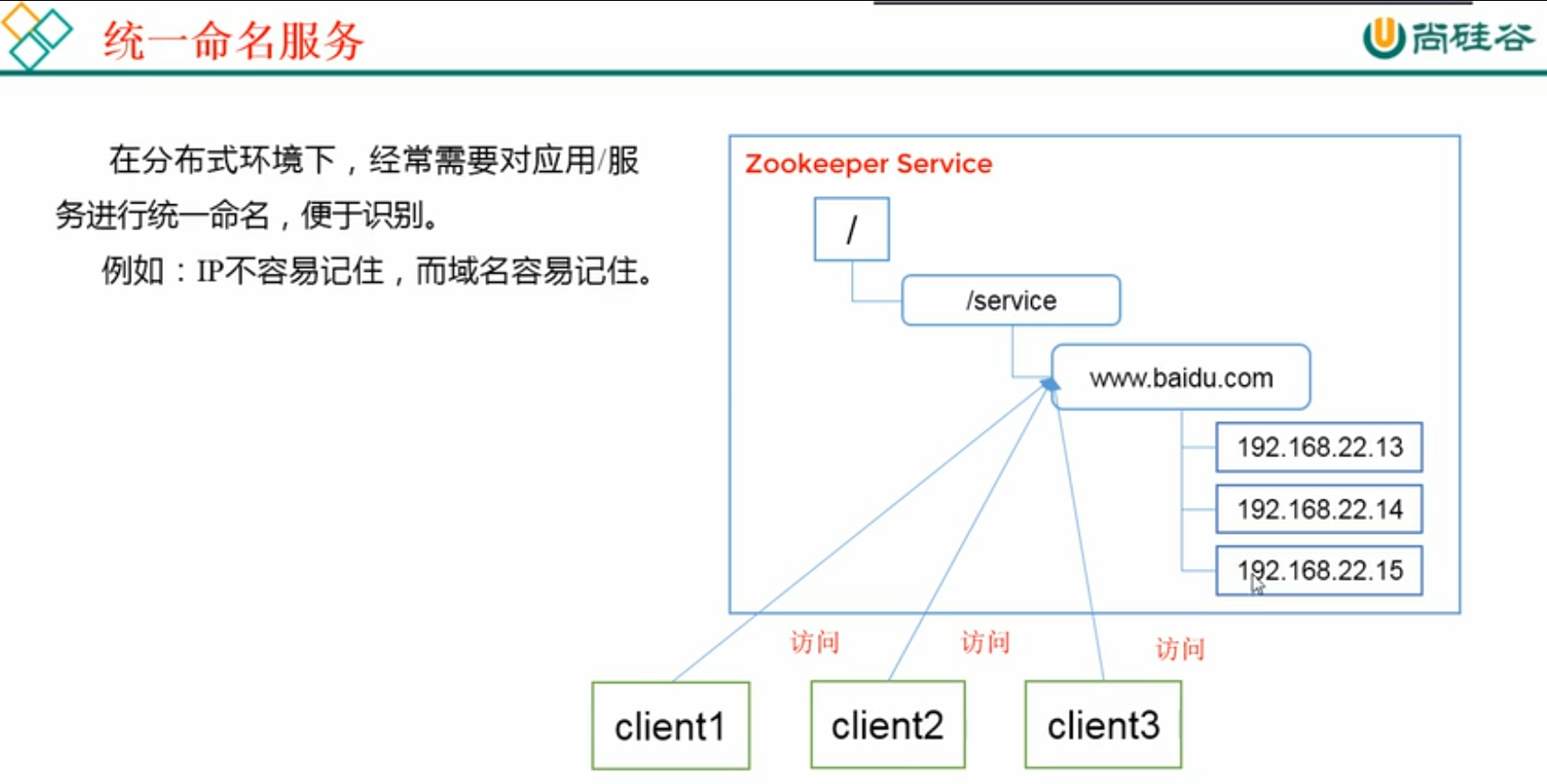
## 1.3 数据结构



## 1.4 应用场景

统一命名、分布式配置管理、分布式消息队列、分布式锁、分布式协调、服务器节点动态上下线、软负载均衡等。

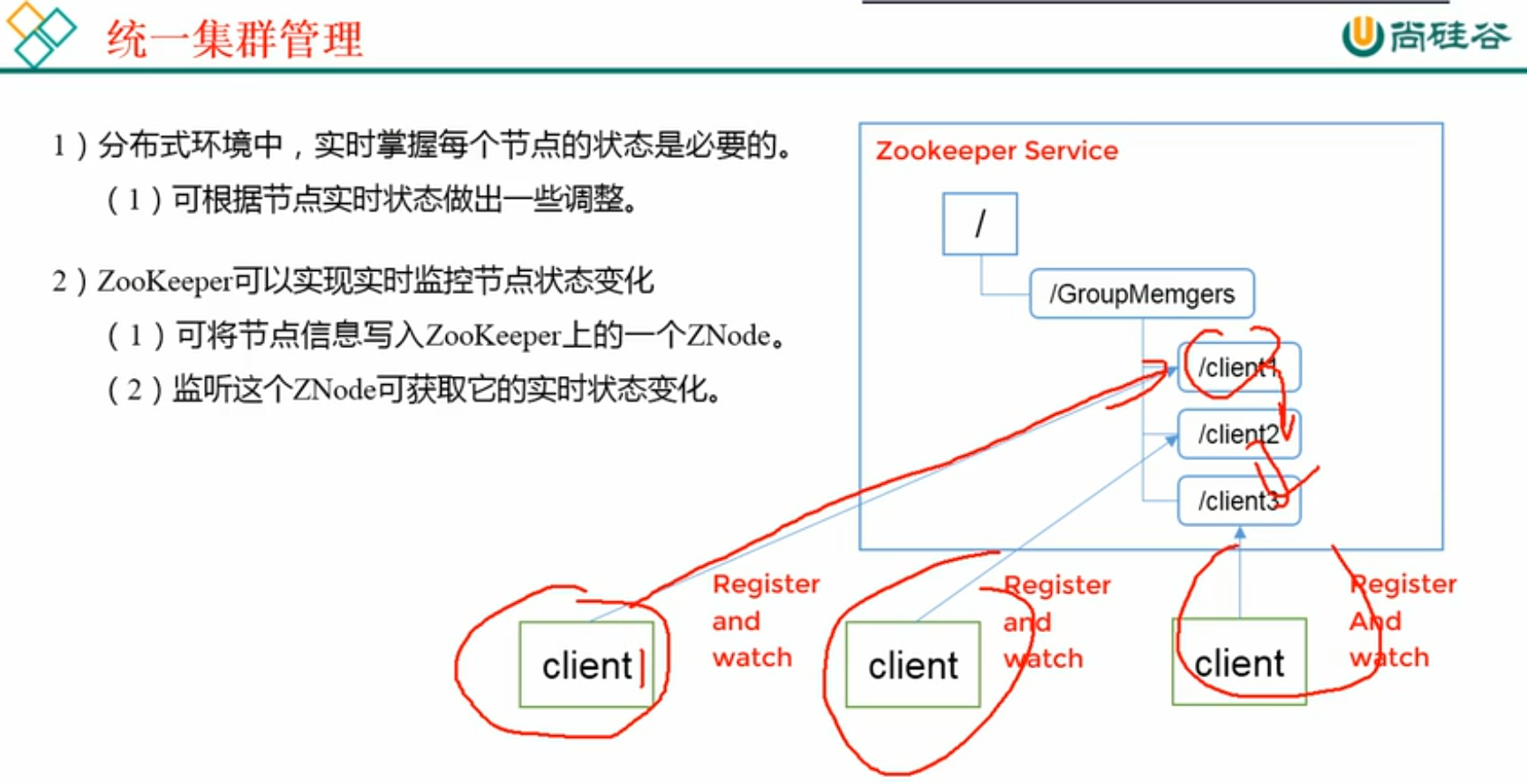
### 1.4.1统一命名



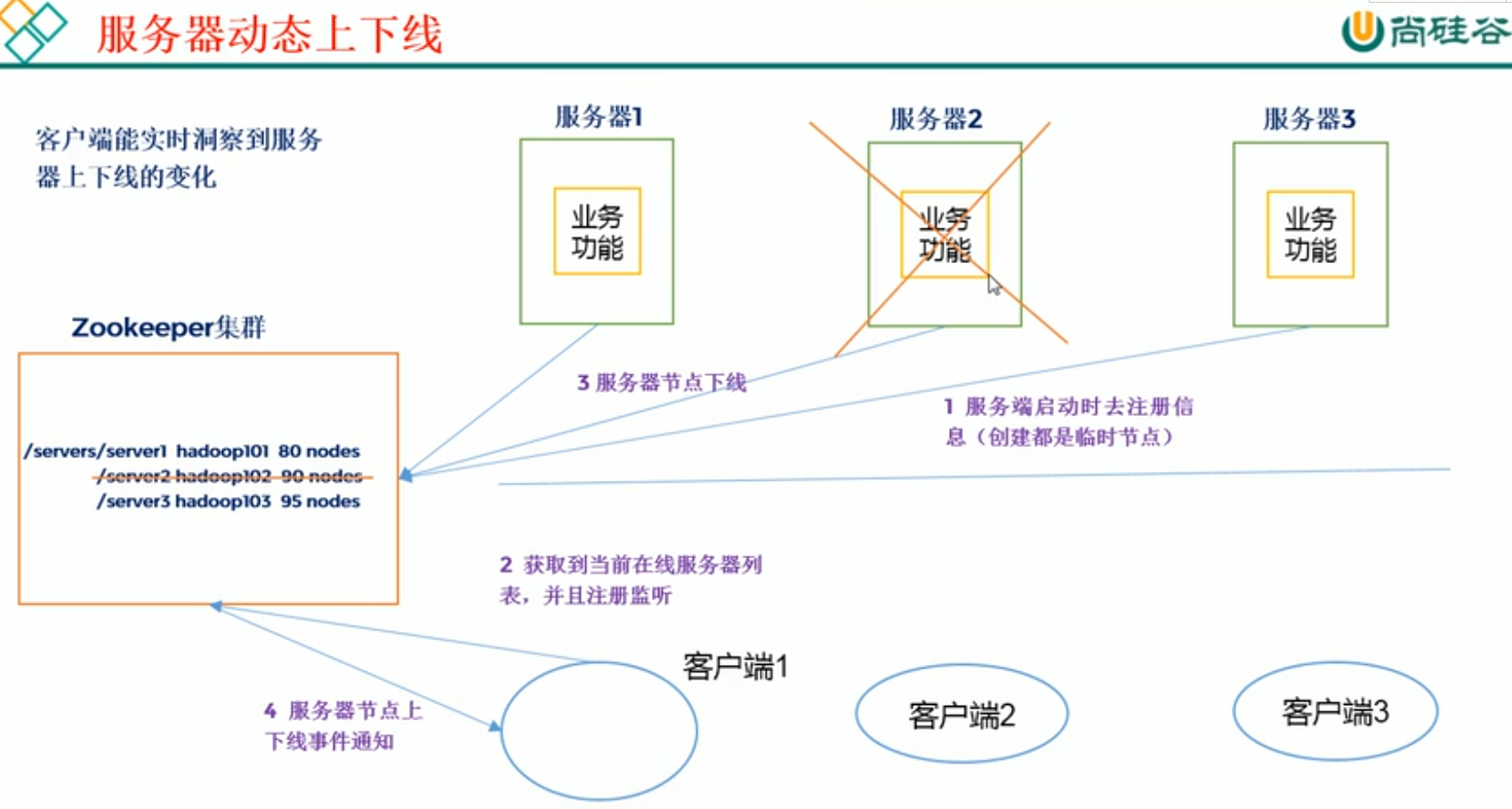
### 1.4.2统一配置管理



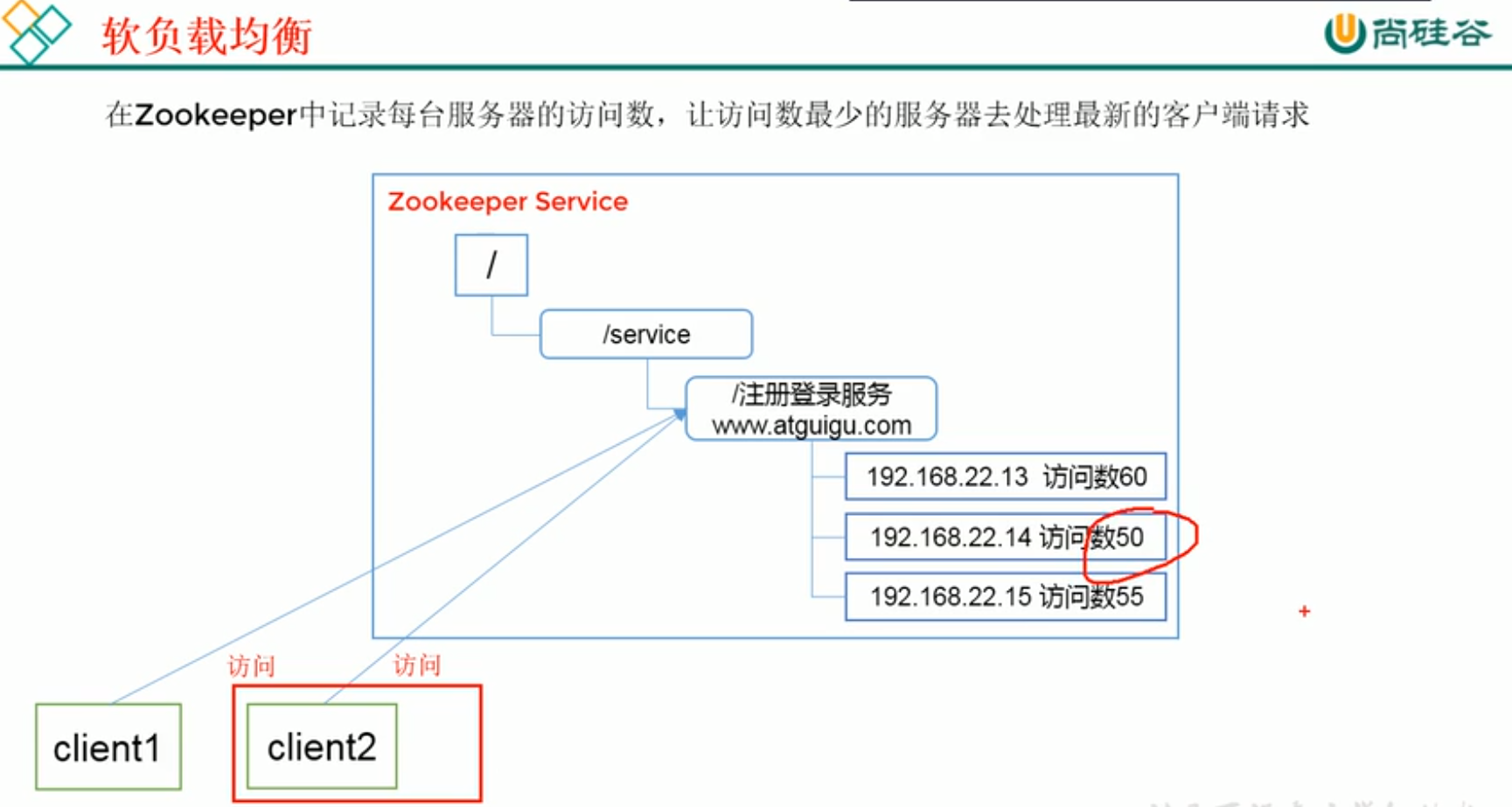
### 1.4.3统一集群管理



### 1.4.4服务器动态上下线



### 1.4.5软负载均衡



# 2 Zookeeper安装（集群）

所谓集群就是分布式的安装，通常由2n+1台servers。为了保证leader选举（基于Paxos算法的实现）能够得到多说的支持。所以zookeeper集群的数量都是奇数。

Zookeeper运行需要Java环境。安装集群（leader+follower）的大致模式如下

● 配置主机名称到IP地址映射配置

● 修改Zookeeper配置文件

● 远程复制分发安装文件

● 设置myid

● 启动Zookeeper集群

如果想要使用Observer模式，课在对应节点的配置文件添加如下配置：

peerType=observber

其次，必须在配置文件制定哪些节点为Observer，如：

server.1:localhost:2181:3181:observer

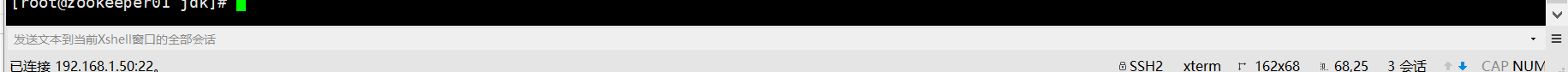
## 2.1 提前安装好JDK

参照《001.工具之VMware12搭建Centos7.7-20200102》中的JDK安装

## 2.2 基础环境检查

### 2.2.1 集群时间是否同步

发送命令给全部会话



命令：date

自动同步互联网时间， 用 ntpdate从时间服务器更新时间

如果系统没有 ntpdate 命令，可在线安装：yum -y install ntp

同步： ntpdate time.nist.gov

验证： date

有时候同步不了，就手动同步date -s

### 2.2.2防火墙是否关闭，我们不关闭，只开要用到端口

开通80、2181、3306端口

### 2.2.3主机、ip映射有没有配对

vim /etc/hosts

添加

192.168.3.50 zookeeper01

192.168.3.51 zookeeper02

192.168.3.52 zookeeper03

## 2.3 安装zookeeper

版本：zookeeper-3.4.8.tar.gz

位置：/pl/zookeeper

解压zookeeper

tar -xvf zookeeper-3.4.8.tar.gz

## 2.4 修改环境变量（全部）

vim /etc/profile

export ZOOKEEPER\_HOME=/pl/zookeeper

export PATH=$PATH:$ZOOKEEPER\_HOME/bin

重新加载配置文件

source /etc/profile

## 2.5 修改zookeeper配置文件

cd /pl/zookeeper/zookeeper-3.4.8

cp zoo\_sample.cfg zoo.cfg

vim zoo.cfg

**添加内容：**

**dateDir= /pl/data/zkdata** #运行时数据存放位置

#(心跳端口、选举端口)

server.1=zookeeper01:2888:3888

server.2=zookeeper02:2888:3888

server.3=zookeeper03:2888:3888

**不变内容**：

clientPort=2181 #端口号，默认都是2181，一般不要改

**tickTime** #zookeeper服务器和客户端之间，服务器之间维持心跳的时间间隔

**initLimit** #配集群中的follower服务器(F)与leader服务器(L)之间 初始连接 时能容忍的最多心跳数；

此配置表示，允许 follower （相对于 leader 而言的“客户端”）连接 并同步到  leader 的初始化连接时间，它以 tickTime 的倍数来表示。当超过设置倍数的 tickTime 时间，则连接失败。

      如果在设定的时间段内，半数以上的跟随者未能完成同步，领导者便会宣布放弃领导地位，进行另一次的领导选举。如果zk集群环境数量确实很大，同步数据的时间会变长，因此这种情况下可以适当调大该参数。默认为10。

**syncLimit** #配置标识leader和follower之间发送消息、请求、应答的时间长度，最长不超过多少个tickTime

server.A=B:C:D #A代表第几号服务器，B是这个服务器的IP地址，C心跳端口，D选举端口；。如果 follower 在设置的时间内不能与leader 进行通信，那么此 follower 将被丢弃。所有关联到这个跟随者的客户端将连接到另外一个跟随着。

**创建myid文件：**

cd /pl/data/zkdata

在data文件夹下新建myid文件，内容为：

cd zkdata

echo 1 > myid

## 2.6 分发安装包到其他机器

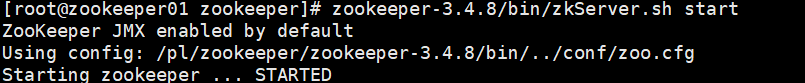
远程分发zookeeper

scp –r /pl/zookeeper root@zookeeper\_02: /pl

scp –r /pl/zookeeper root@zookeeper\_03: /pl

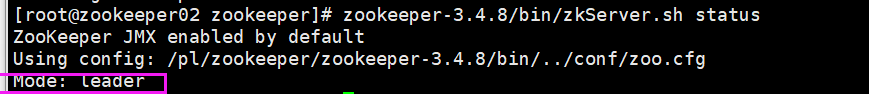
## 2.7 启动（每台机器）

zkServer.sh start



查看状态

zkServer.sh status



关闭

zkServer.sh stop

或者编写一个脚本来批量启动所有机器

for host in “zookeeper01 zookeeper02 zookeeper03”

do

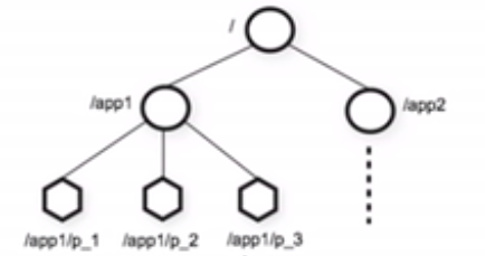
sh $host “source/etc/profile;/pl/zookeeper/bin/zkServer.sh start”

## 2.8 jps

Java Virtual Machine Process Status Tool (Java虚拟机进程状态工具)

# 3 Zookeeper数据模型

## 3.1数据结构图



图中的每个节点称为一个Znode。每个Znode由3部分组成：

① stat：状态信息，描述Znode的版本、权限等信息；

② data：与该Znode关联的数据；

③ children：该Znode下的子节点；

## 3.2 节点类型（2种）

**临时节点**、**永久节点**

节点类型在创建时即被确定，并且不能改变。

**临时节点：**生命周期依赖于创建它们的会话。一旦会话结束，临时节点将被自动删除，当然也可以手动删除。临时节点不允许拥有子节点。

**永久节点：**其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，它们才会被删除。

Znode还有一个序列化的特性，如果创建的时候指定的话，该Znode的名字后面会自动追加一个不断增加的序列号。序列号对于此节点的父节点来说是唯一的，这样便会记录每个子节点创建的先后顺序。它的格式为“%10d”（10位数字，没有数值的数位用0补充，例如“00000001”）。正阳便会存在四种节点：

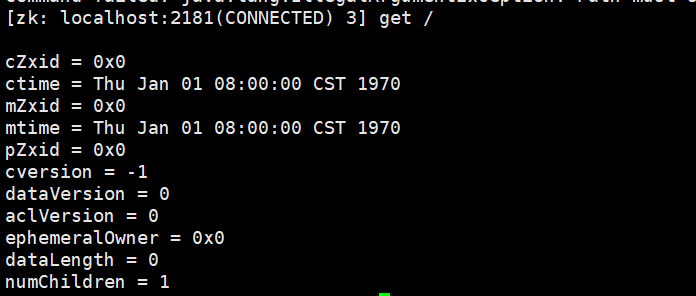
persistent：永久节点

ephemeral：临时节点

persistent\_sequential：永久节点、序列化

ephemeral\_ sequential：临时节点、序列化

## 3.3 节点属性



每个znode都包含了一系列的属性，通过命令get,获取节点属性。

**cZxid：**Znode创建的事务id

**ctime：**Znode创建时的时间戳。

**mZxid：**Znode被修改的事务id，即每次对znode的修改都会更新mZxid。对于zk来说，每次的变化都会产生一个唯一的事务id，zxid（Zookeeper Transaction ID）。通过zxid确定更新操作的先后顺序。

**mtime：**节点最新一次更新发生时的时间戳。

**cversion：**子节点的版本号，当znode的子节点变化时，cversion的值会加1

**dataversion：**数据版本号，每次对节点进行set操作，dataversion的值都会加1（即使设置的是相同的数据），有效避免数据更新时出现的先后顺序问题。

**aclVersion：**ACL的版本号

**ephemeralOwner：**如该节点为临时节点，值为与该节点绑定的session id。如不是值为零。0x0是永久节点。

**numChildren：**子节点个数。

client和sever通信之前，首先要建立连接，该连接称为session。连接建立后，如果发生超时、授权失败，或者显示关闭连接，连接处于CLOSED状态，此时session结束。

# 4 Zookeeper shell

## 4.1 客户端连接

运行 **zkCli.sh –server ip** 进入命令行工具

–server可选，不写得的话，当前主机；写的话，远程主机；因为是分布式，连接一台，就是连上了集群。

## 4.2 shell基本操作

### 4.2.1 创建节点

create [-s] [-e] path data acl

其中，-s或-e分别指定节点特性，顺序或临时节点，若不指定，则表示持久节点；acl用来进行权限控制。

**创建顺序节点**： create –s /test 123

**创建临时节点：** create –e /test-temp 123temp

### 4.2.2 读取节点

命令有ls和get；

ls path [watch] #显示path下有多少节点

get path [watch] #显示path下节点的属性

[watch]：

### 4.2.3 更新节点

set path data [version]

### 4.2.4 删除节点

delete path [version]

如果删除的节点存在子节点，那么无法删除该节点，必须先删除子节点。

递归删除： Rmr path

### 4.2.5 quota（权限操作）

setquota –n|-b val path 对接点增加权限

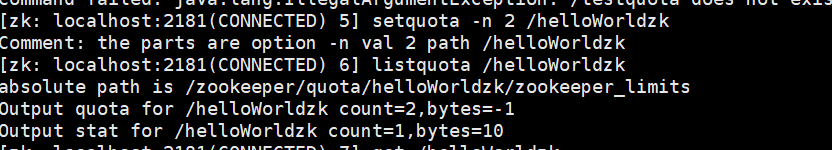
n：表示子节点的最大个数

b：表示数据值的最大长度

val：子节点最大个数或数据值的最大长度

如：setquota –n 2 /quota-test

slistquota path 列出指定节点的quota

-1表示没有限制

delquota [-n|-b] path 删除指定节点的quota（限制，权限）

如果超出了限制，只会警告，但不会报错拒绝，还是会创建出来。

### 4.2.6 其他命令

history 列出历史命令，默认10条

redo 重新执行命令变好的历史命令，命令编号通过history查看；如： redo 10

# 5 Zookeeper Watcher

Zookeeper提供了分布式数据发布/订阅功能。一个典型的发布/订阅模型系统定义了一种一对多的订阅关系，能让多个订阅者同时监听某一个主题对象，当这个主题对象自身状态变化时，会通知所有订阅者，使他们能够做出相应处理。

Zookeeper中，引入了Watcher机制来来实现这种分布式的通知功能。Zookeeper允许客户端向服务端注册一个Watcher监听（就是一个订阅），当服务端的一些事件触发了这个Watcher，那么就会向指定客户端发送一个事件通知来实现分布式的通知功能。

触发事件种类很多，如：节点创建、节点删除、节点改变、子节点改变等。

总的来说可以概括Watcher为以下三个过程：

1. 客户端向服务器注册Watcher（订阅，比如订阅一个杂志），
2. 服务器端事件触发Watcher（产生变化，新杂志出来了），
3. 客户端回调Watcher得到触发事件情况（通知，比如有新杂志，你可以选择要不要）

## 5.1 Watch机制特点

### 5.1.1 一次性触发

事件发生触发监听，一个watcher event就会发送到设置监听的客户端，这种效果是一次性的，后续再次发生同样的事件，不会再次触发。

### 5.1.2事件封装

Zookeeper使用WatcherEvent对象来封装服务端事件并传递。

WatcherEvent包含了每一个事件的三个基本属性

1. 通知状态（keepState）
2. 事件类型（EventType）
3. 节点路径（path）

### 5.1.3 event异步发送

watcher的通知事件从服务端发送到客户端是异步的

### 5.1.4 先注册再触发

Zookeeper中的watch机制，必须客户端先去服务端注册监听，这样事件发生才会触发监听，通知客户端

### 5.1.5 监听器原理

1、监听原理详解：

1）首先要有一个main()线程；

2）在main线程中创建Zookeeper客户端，这时就会创建两个线程，

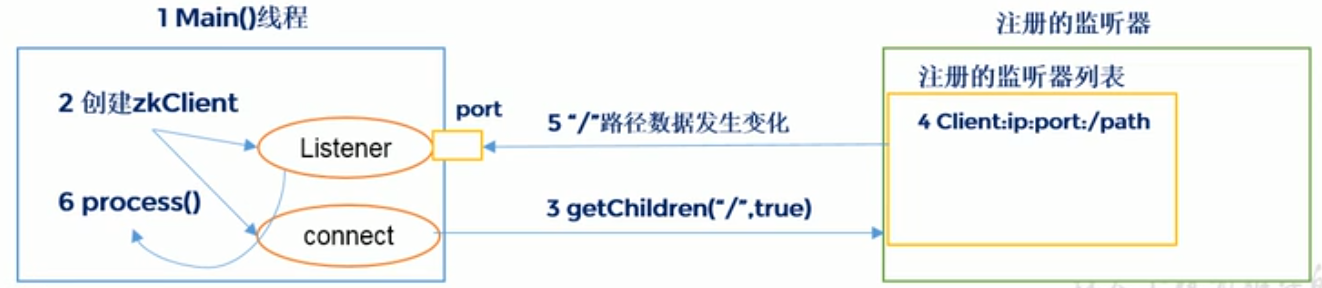
一个负责网络连接通信（connect），一个负责监听（listener）；

3）通过connect线程将注册的监听事件发送给zookeeper；

4）在zookeeper的注册监听器列表中将注册的监听事件添加到列表中；

5）zookeeper监听到有数据或路径变化，就会将这个消息发送给listener线程；

6）listener线程内部调用了process()方法；

 2、常见的监听

1）监听节点数据的变化：get path [watch]

2）监听子节点增减的变化：ls path [watch]

## 5.2 通知状态和事件类型

同一个事件类型在不同的通知状态中代表的含义有所不同，下表列举了常见的通知状态和事件类型

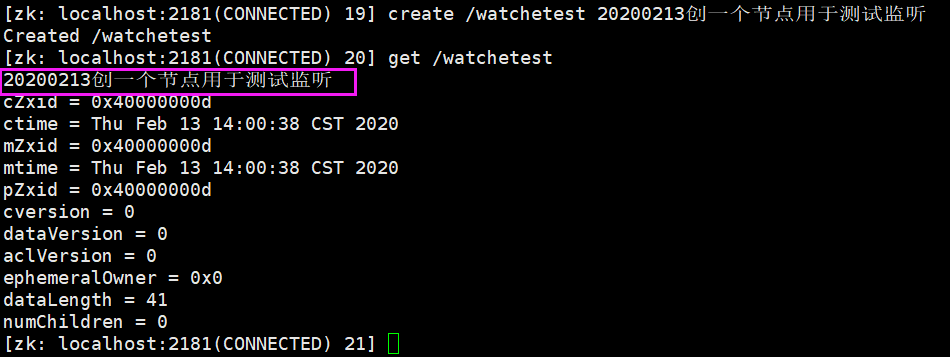
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **keepState** | **EventType** | **触发事件** | **说明** |
| 1 | SyncConnected  (0) | None  (-1) | 客户端与服务端成功建立连接 |  |
| 2 | NodeCreated  (1) | Watcher监听的对应数据节点被创建 |  |
| 3 | NodeDeleted  (2) | Watcher监听的对应数据节点被删除 | 此时客户端和服务器处于连接状态 |
| 4 | NodeDataChanged  (3) | Watcher监听的对应数据节点数据内容发生变更 |  |
| 5 | NodeChildChanged  (4) | Watcher监听的对应数据节点的子节点列表发生变更 |  |
| 6 | Disconnected  （0） | None  (-1) | 客户端与Zookeeper服务器断开连接 | 此时客户端和服务器处于断开连接状态 |
| 7 | Expired  （-112） | Node  (-1) | 会话超时 | 此时客户端会话失效，通常同时也会收到SessionExpiredException异常 |
| 8 | AuthFailed  (4) | None  (-1) | 通常由两种情况，1、使用错误的schema进行权限检查；2、SASL权限检查失败 | 通常同时也会收到AuthFailedException |

其中连接状态事件（type=None，path=null）不需要客户端注册，客户端只要有需要直接处理就行了。

## 5.3 Shell客户端设置watcher

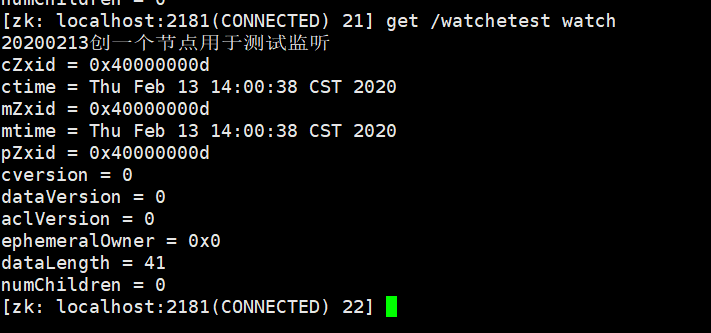
### 5.3.1 创建节点，一会加监听

create [-s] [-e] path data acl

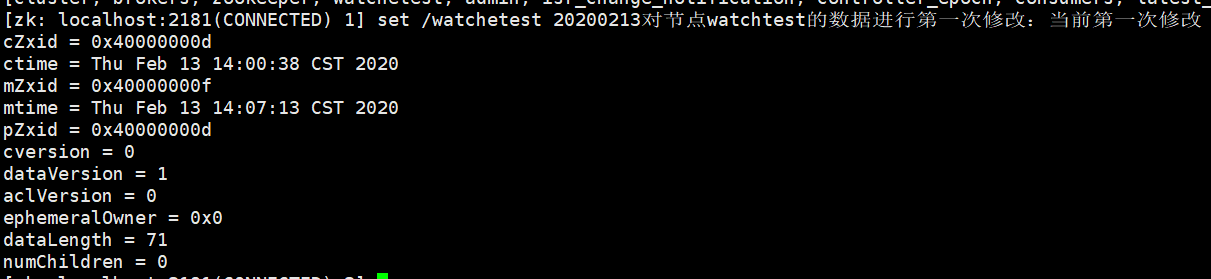


### 5.3.2 给节点添加数据变动监听：

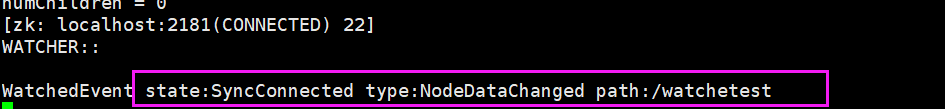
get path watch



### 5.3.3 通过另一个客户端更改节点数据：



### 5.3.4 此时设置监听的节点收到通知：



### 5.3.4 再次重复几次操作，不再有通知

发现没有了通知了，说明是一次性；想收到的话，需要再次注册。

# 6 Zookeeper Java API

org.apache.zookeeper.Zookeeper

Zookeeper是Java客户端主类，负责建立与Zookeeper集群的会话，并提供方法进行操作。

org.apache.zookeeper.Watcher

Watcher接口表示一个标准的事件处理器，其定义了事件通知相关的逻辑，包含KeepState和EventType两个枚举类，分别代表了通知状态和事件类型，同时定义了事件的回调方法：process(WatchedEvent event)。

**process()** 是Watcher接口中的一个回调方法，当Zookeeper向客户端放松一个Watcher事件通知时，客户端就会对响应的process方法进行回调，从而实现对事件的处理。

### 6.1 基本使用

建立java maven项目，引入maven pom坐标。

引入依赖

<dependency>  
 <groupId>org.apache.zookeeper</groupId>  
 <artifactId>zookeeper</artifactId>  
 <version>3.4.9</version>  
</dependency>

java代码创建节点

import org.apache.zookeeper.\*;  
  
*/\*\*  
 \** ***@author*** *: lipu  
 \** ***@since*** *: 2020-01-05 21:56  
 \*/*public class TestZKClient {  
  
  
 public static void main(String[] args) throws Exception{  
 */\*\* connectString ；集群地址；  
 \* sessionTimeout：回话超时时间，默认30秒  
 \* watcher：监听器  
 \*/* ZooKeeper zk = new ZooKeeper("zookeeper01:2181,zookeeper02:2181,zookeeper03:2181", 30000, new Watcher() {  
 //事件通知的回调函数  
 public void process(WatchedEvent event) {  
 System.*out*.println(event.getState());  
 System.*out*.println(event.getType());  
 System.*out*.println(event.getPath());  
 System.*out*.println(event.toString());  
 }  
 });  
 zk.create("/myGirls","中国风".getBytes(), ZooDefs.Ids.*OPEN\_ACL\_UNSAFE*, CreateMode.*PERSISTENT\_SEQUENTIAL*);  
 zk.close();  
 }  
}

### 6.2 更多操作示例

节点的增删改查，数据的改变

import org.apache.zookeeper.\*;  
  
*/\*\*  
 \** ***@author*** *: lipu  
 \** ***@since*** *: 2020-01-05 21:56  
 \*/*public class TestZKClient {  
  
  
 public static void main(String[] args) throws Exception{  
 */\*\* connectString ；集群地址；  
 \* sessionTimeout：回话超时时间，默认30秒  
 \* watcher：监听器  
 \*/* ZooKeeper zk = new ZooKeeper("zookeeper01:2181,zookeeper02:2181,zookeeper03:2181", 30000, new Watcher() {  
 //事件通知的回调函数  
 public void process(WatchedEvent event) {  
 System.*out*.println(event.getState());  
 System.*out*.println(event.getType());  
 System.*out*.println(event.getPath());  
 System.*out*.println(event.toString());  
 }  
 });  
 //存咋的节点不可以再创建  
// zk.create("/myGirls","中国风".getBytes(), ZooDefs.Ids.OPEN\_ACL\_UNSAFE, CreateMode.PERSISTENT);  
 //相当于对节点/myGirls设置了数据变化的监听，一旦节点数据改变，监听就会触发  
 zk.getData("/myGirls",true,null);  
  
 //对节点数据进行修改。触发监听。-1表示版本交给系统自己维护  
 zk.setData("/myGirls","创新高地".getBytes(),-1);  
  
 zk.close();  
 }  
}

## 7 Zookeeper 选举机制

Zookeeper默认的算法是FastLeaderElection，采用投票数大于半数则胜出的逻辑

### 7.1 概念

**服务器ID**

比如有三台服务器，编号分别为1,2,3；编号越大在选择算法中的权重越大；

**选举状态**

LOOKINg，竞选状态

FOLLOWING，随从状态，同步leader状态，参与投票

OBSERVING，观察者状态，同步leader状态，不参与投票

LEADER，领导者状态

**数据ID**

服务器中存放的最新数据version；

值越大说明数据越新，在选举算法中数据越新权重越大。

**逻辑时钟**

也叫投票的次数，同一轮投票过程中的逻辑时钟值是相同的。每次投完一次票，这个数据就会增加，然后与接收到的其他服务器返回的投票信息中数值相比，根据不同的值做出不同的判断。比如挂过的机器，逻辑时钟就比没挂的少，那它的权重就小。

### 7.2 全新集群选举

就是刚搭建的集群，每台服务器都没有数据，他们的编号分别是1,2,3,4,5；按编号以此启动，选举过程如下：

1. 服务器1启动，给自己投票，然后发投票信息，由于别的没动，收不到反馈信息，则1处于LOOKING状态
2. 服务器2启动，给自己投票，同时与之前启动的服务器1交换结果，由于服务器2的编号大，所以服务器2胜出，但此时投票数没有大于半数，所以两个都是LOOKING状态。
3. 服务器3启动，给自己投票，同时与1、2交换信息，由于3的标号最大，所以3胜出，此时投票数正好大于半数，所以3成为leader，1、2成为follower。
4. 服务器4启动，给自己投票，同时与1、2、3交换信息，尽管4的编号最大，但之前3已经胜出，所以4只能成为follower
5. 服务器5启动，后面的逻辑通服务器4成为follower

### 7.3 非全新集群选举

对于运行正常的Zookeeper集群，中途有机器down掉，需要重新选举时，选举过程就需要加入数据ID、服务器ID、逻辑时钟。

数据ID：数据新的version就大，数据每次更新都会更新version；

服务器ID：就是我们配置的myid的值，每个机器一个

逻辑时钟：这个值从0开始，每次选举对应一个值，如在同一此次选举中，这个值是一样的。

选举标准变成：

1. 逻辑时钟小的选举结果被忽略，重新投票；
2. 统一逻辑时钟后，数据ID大的胜出；
3. 数据id相同时，服务器id大的胜出；

根据这个规则选出leader。

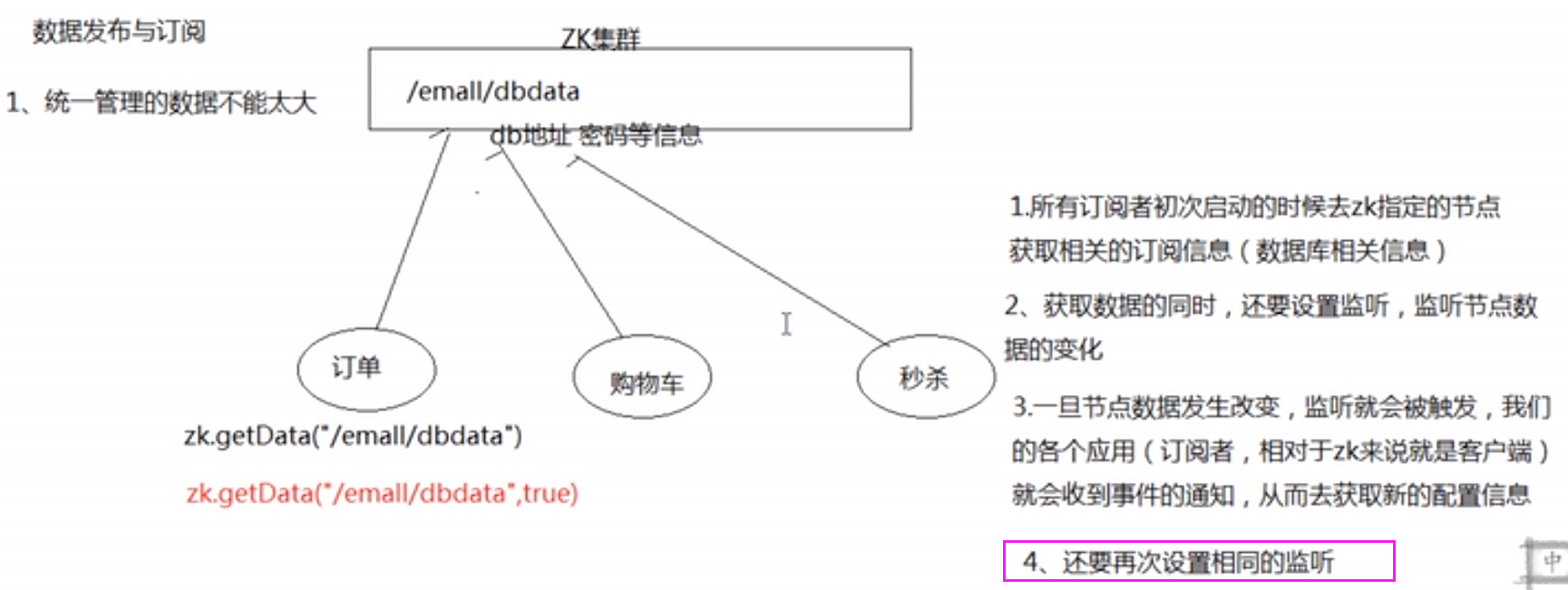
## 8 Zookeeper 典型应用

### 8.1 数据发布与订阅（配置中心）

发布与订阅，即所谓的配置中心；顾名思义就是发布者将数据发布到ZK节点上，供订阅者动态获取数据，实现配置信息的集中式管理和动态更新。、

应用在启动时会主动来获取一次配置，同时，在节点上注册一个Watcher，这样一来，以后每次配置有更新的时候，都会实时通知订阅的客户端，从而达到获取最新配置信息的目的。

适合数据量很小的场景，数据更新快。



### 8.2 命名服务

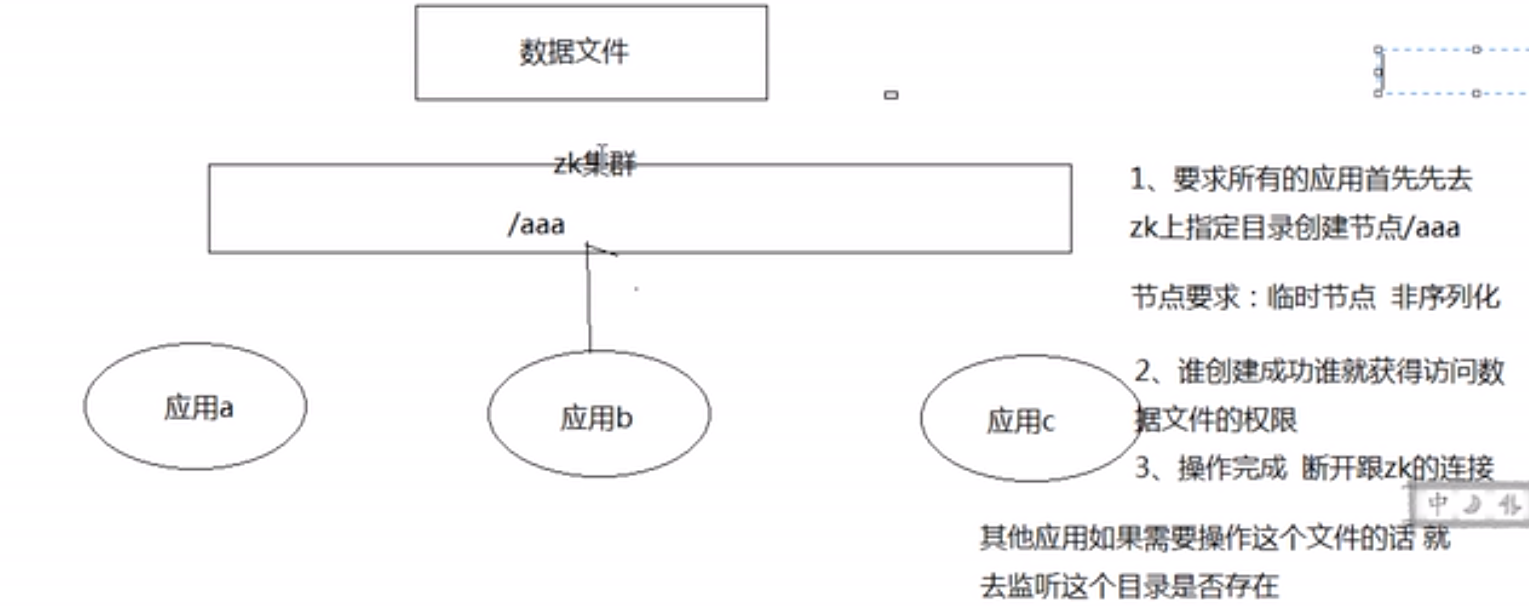
在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址、提供者等信息。被命名的实体通常可以是集群中的机器，提供的服务地址，远程对象等等—这些我们都可以统称它们为名字（name）。其中较为常见的就是一些分布式服务框架中的服务地址列表。通过调用zk提供的创建节点的API，能够很容易创建一个全局唯一的path，这个path就可以作为一个名称。

阿里巴巴集团开源的分布式服务框架Dubbo中使用Zookeeper来作为其命名服务，维护全局的服务地址列表。

### 8.3 分布式锁

分布式锁，这个主要得益于Zookeeper保证了数据的强一致性。锁服务可以分为两类，一个是保持独占，另一个是控制时序。

所谓保持独占，就是所有试图来获取这个锁的客户端，最终只有一个可以获得这把锁。通常的做法是把zk上的一个znode（临时节点）看做是一把锁，通过create znode的方式来实现。所有客户端都去创建/distribute\_lock节点，最终创建成功的那个客户端也拥有了这把锁。



控制时序，就是所有试图来获取这个锁的客户端，最终是都会被安排执行，只是有个全局时序了。做法和上面基本类似，只是这里/distribute\_lock已经存在，客户端下面创建临时有序节点（这个可以通过节点的属性控制：CreateMode.EPHEMERAL\_SENQUENTAL来指定）。ZK的父节点（/distribute\_lock）维持一份sequence，保证子节点创建的时序性，从而也形成了每个客户端的全局时序。

## 9 企业面试真题

### 9.1 简述Zookeeper的选举机制？

### 9.2 Zookeeper的监听原理？

### 9.3 Zookeeper的部署方式有哪几种？集群中的校色有哪些？最少几台

单机、集群；

### 9.4 Zookeeper的常用命令？

ls get create set delete setquota

## 10 网络通信

## XX 001 名词解释

### 2.1 Xerox

1906年，施乐公司于成立于美国[康涅狄格州](https://baike.baidu.com/item/%E5%BA%B7%E6%B6%85%E7%8B%84%E6%A0%BC%E5%B7%9E)的费尔菲尔德县。主要在复印机、打印机行业

1962年，富士施乐成立，富士胶片和美国施乐合资成立了富士施乐株式会社，注册资金各50%，一共2亿日元。同年，富士施乐914款复印机在日本上市。

### 2.3

### 2.4查

、

### 2.5关