尚硅谷大数据技术之 Zookeeper

(作者：大海哥)

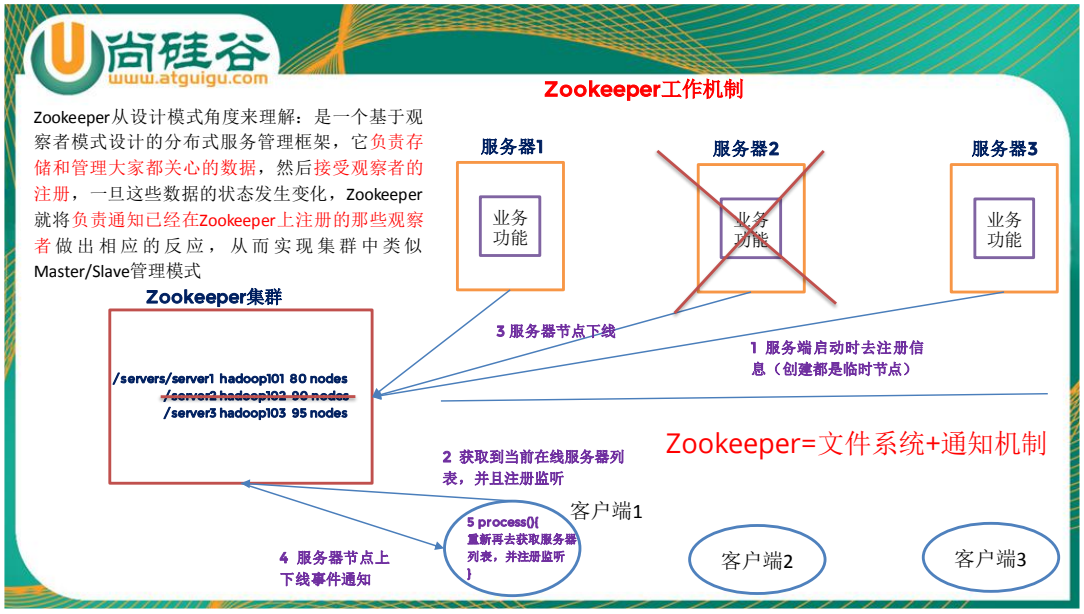
官网：www.atguigu.com

版本：V1.0

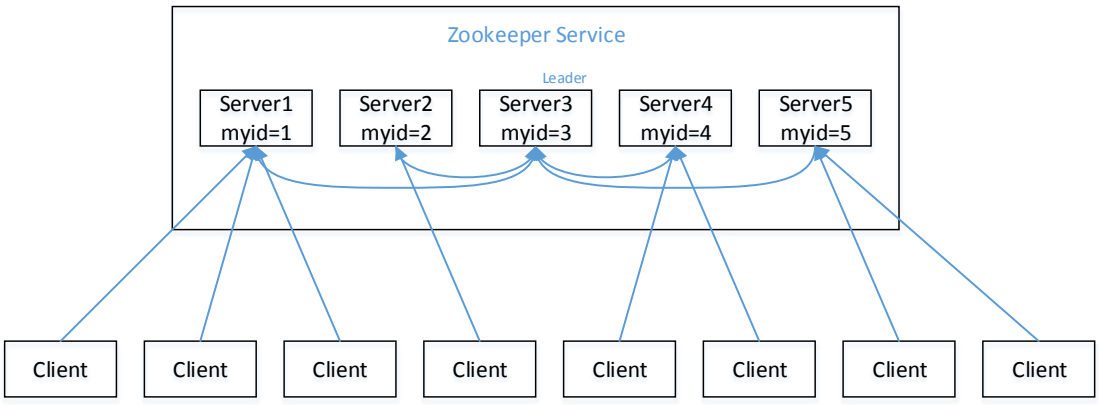
一 Zookeeper 概述

1.1 概述

Zookeeper 是一个开源的分布式的，为分布式应用提供协调服务的 Apache 项目。



1.2 特点



1）Zookeeper：一个领导者（leader），多个跟随者（follower）组成的集群。

2）Leader 负责进行投票的发起和决议，更新系统状态。

3）Follower 用于接收客户请求并向客户端返回结果，在选举 Leader 过程中参与投票。

4）集群中只要有半数以上节点存活，Zookeeper 集群就能正常服务。

5）全局数据一致：每个 server 保存一份相同的数据副本，client 无论连接到哪个 server，数据都是一致的。

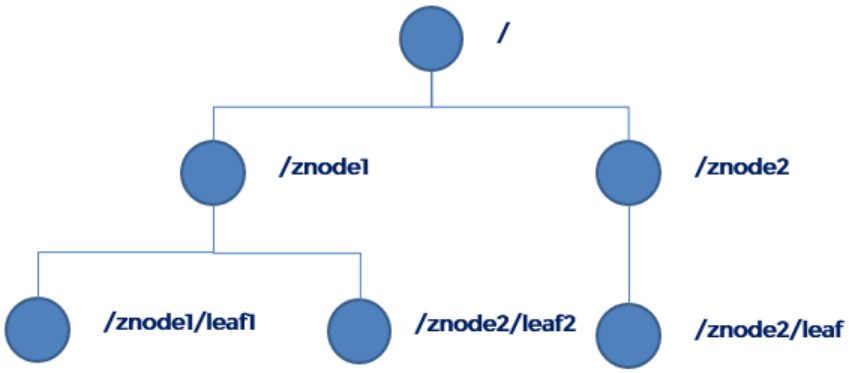
6）更新请求顺序进行，来自同一个 client 的更新请求按其发送顺序依次执行。

7）数据更新原子性，一次数据更新要么成功，要么失败。

8）实时性，在一定时间范围内，client 能读到最新数据。

1.3 数据结构

ZooKeeper 数据模型的结构与 Unix 文件系统很类似，整体上可以看作是一棵树，每个节点称做一个 ZNode。每一个 ZNode 默认能够存储 1MB 的数据，每个 ZNode 都可以通过其路径唯一标识。

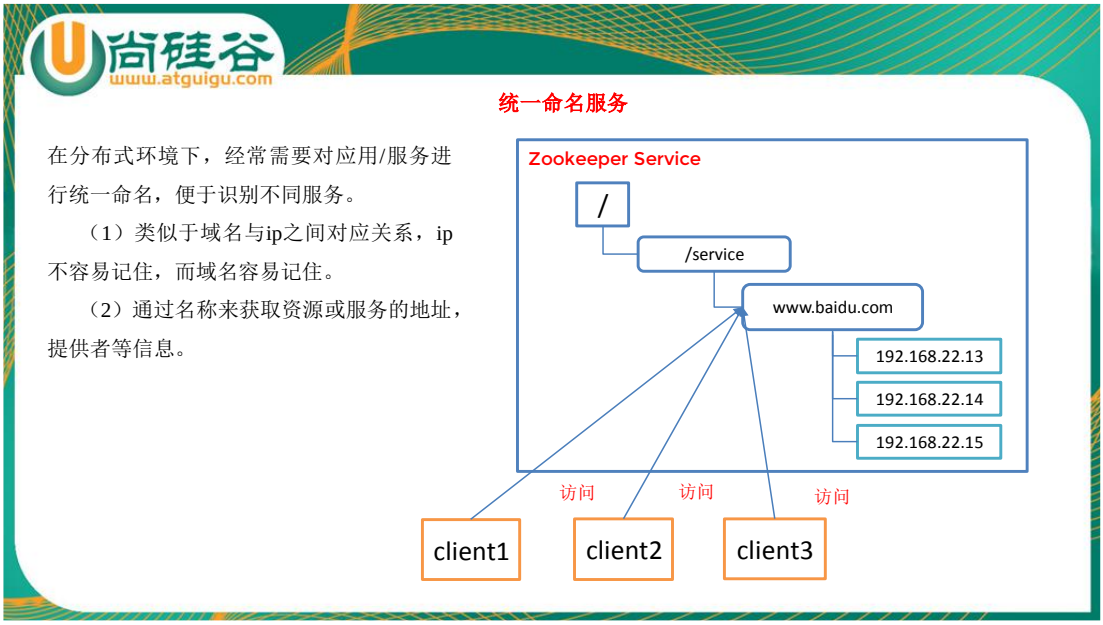


数据结构图

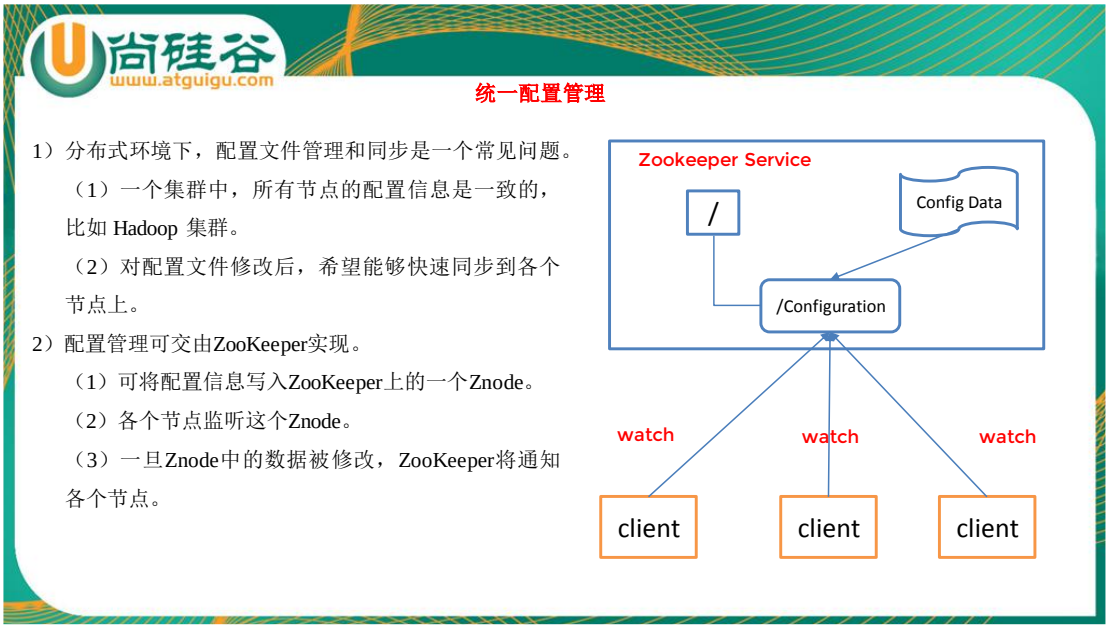
1.4 应用场景

提供的服务包括：统一命名服务、统一配置管理、统一集群管理、服务器节点动态上下线、软负载均衡等。

1.4.1 统一命名服务

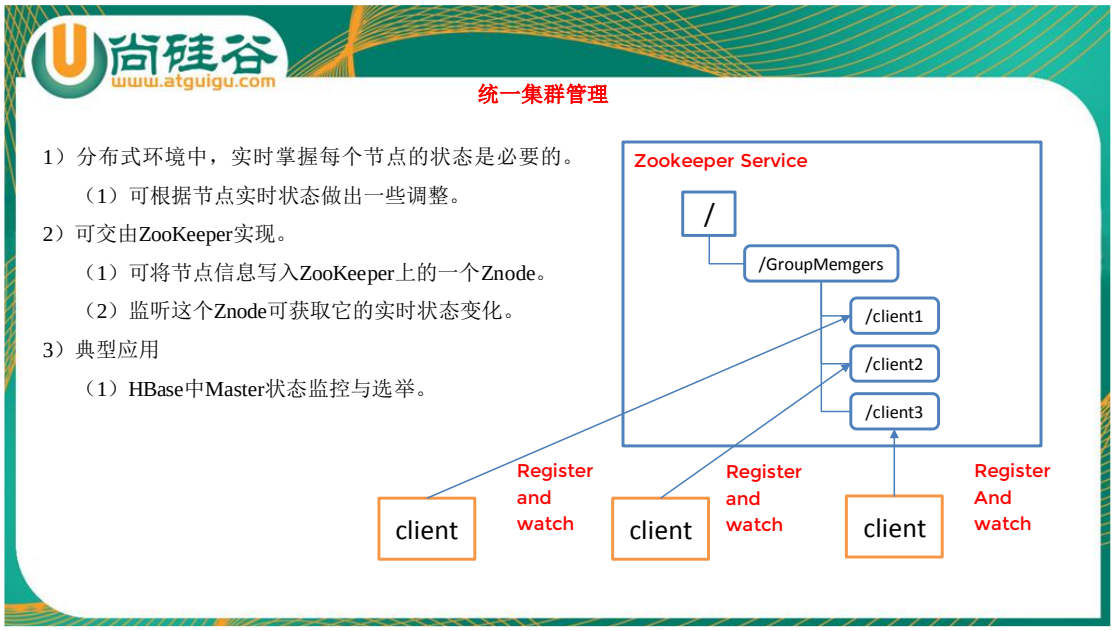


1.4.2 统一配置管理

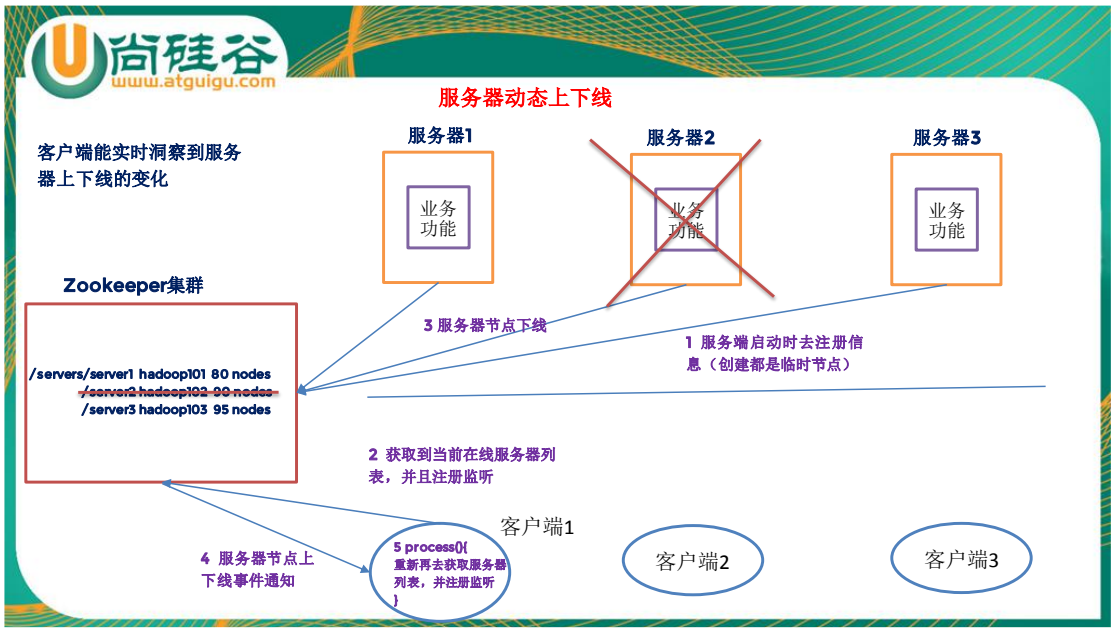


1.4.3 统一集群管理

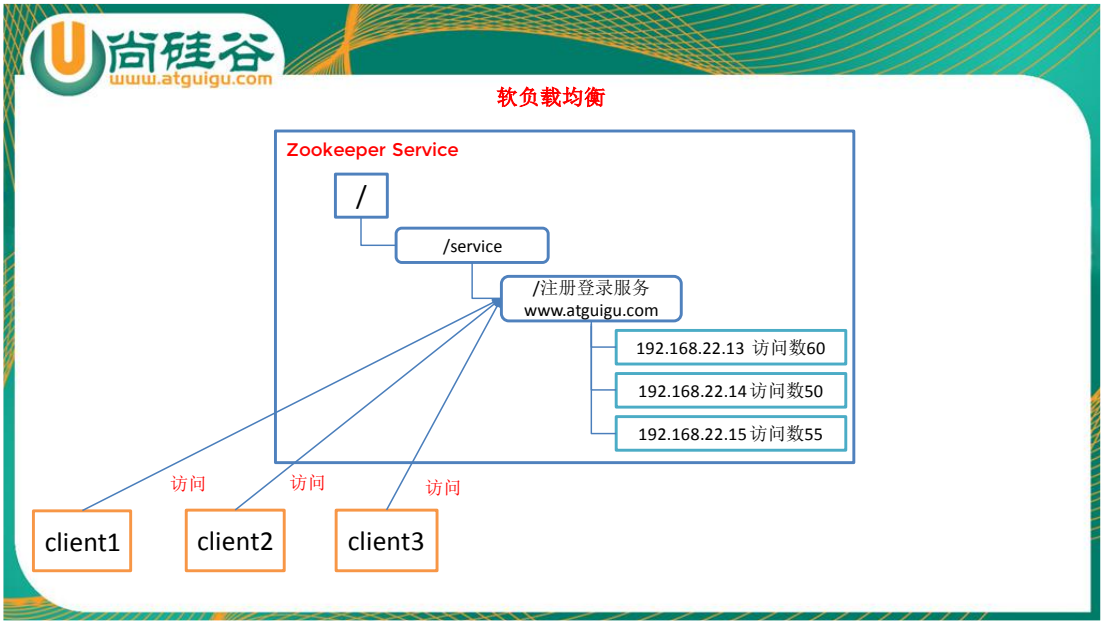
集群管理结构图如下所示。



1.4.4 服务器节点动态上下线



1.4.5 软负载均衡

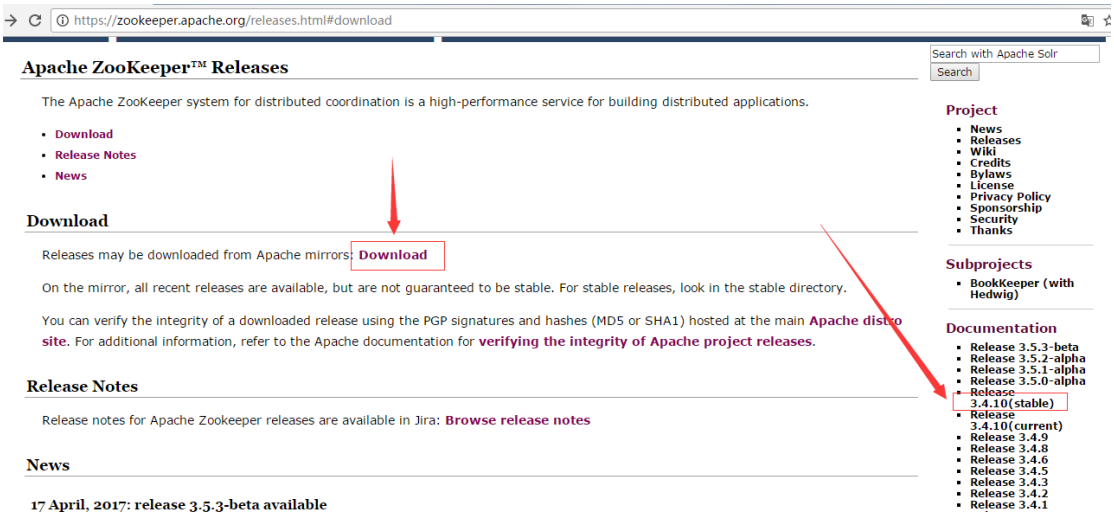


1.5 下载地址

1）官网首页：

https://zookeeper.apache.org/

2）下载截图



二 Zookeeper 安装

2.1 本地模式安装部署

1）安装前准备：

（1）安装 jdk

（2）通过 SecureCRT 工具拷贝 zookeeper 到 linux 系统下

（3）修改 tar 包权限

[atguigu@hadoop102 software]$ chmod u+x zookeeper-3.4.10.tar.gz

（4）解压到指定目录

[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

2）配置修改

将/opt/module/zookeeper-3.4.10/conf 这个路径下的 zoo_sample.cfg 修改为 zoo.cfg；

进入 zoo.cfg 文件：vim zoo.cfg

修改 dataDir 路径为

dataDir=/opt/module/zookeeper-3.4.10/zkData

在/opt/module/zookeeper-3.4.10/这个目录上创建 zkData 文件夹

[atguigu@hadoop102 zookeeper-3.4.10]$ mkdir zkData

3）操作 zookeeper

（1）启动 zookeeper

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh start

（2）查看进程是否启动

[atguigu@hadoop102 zookeeper-3.4.10]$ jps

4020 Jps

4001 QuorumPeerMain

（3）查看状态：

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh status

ZooKeeper JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: standalone

（4）启动客户端：

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkCli.sh

（5）退出客户端：

[zk: localhost:2181(CONNECTED) 0] quit

（6）停止 zookeeper

[atguigu@hadoop102 zookeeper-3.4.10]$ bin/zkServer.sh stop

2.2 配置参数解读

解读zoo.cfg文件中参数含义

1）tickTime=2000：通信心跳数，Zookeeper服务器心跳时间，单位毫秒

Zookeeper使用的基本时间，服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个tickTime时间就会发送一个心跳，时间单位为毫秒。

它用于心跳机制，并且设置最小的session超时时间为两倍心跳时间。(session的最小超

时时间是2*tickTime)

2）initLimit=10：Leader和Follower初始通信时限

集群中的follower跟随者服务器与leader领导者服务器之间初始连接时能容忍的最多心跳数（tickTime的数量），用它来限定集群中的Zookeeper服务器连接到Leader的时限。

投票选举新leader的初始化时间

Follower在启动过程中，会从Leader同步所有最新数据，然后确定自己能够对外服务的起始状态。

Leader允许Follower在initLimit时间内完成这个工作。

3）syncLimit=5：Leader 和 Follower 同步通信时限

集群中Leader与Follower之间的最大响应时间单位，假如响应超过syncLimit * tickTime，Leader认为Follwer死掉，从服务器列表中删除Follwer。

在运行过程中，Leader负责与ZK集群中所有机器进行通信，例如通过一些心跳检测机制，来检测机器的存活状态。

如果L发出心跳包在syncLimit之后，还没有从F那收到响应，那么就认为这个F已经不在线了。

4）dataDir：数据文件目录+数据持久化路径

保存内存数据库快照信息的位置，如果没有其他说明，更新的事务日志也保存到数据库。

5）clientPort=2181：客户端连接端口

监听客户端连接的端口

三 Zookeeper 内部原理

3.1 选举机制

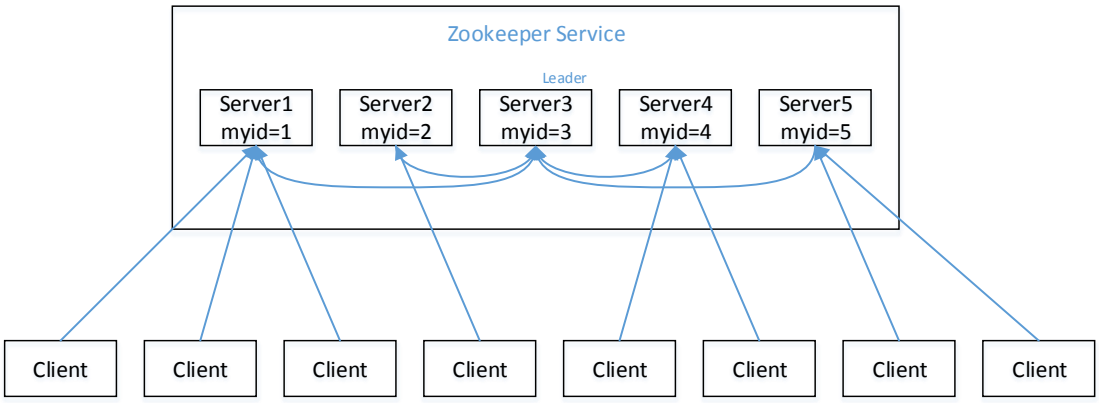
1）半数机制（Paxos 协议）：集群中半数以上机器存活，集群可用。所以 zookeeper 适合装在奇数台机器上。

2）Zookeeper 虽然在配置文件中并没有指定 master 和 slave。但是，zookeeper 工作时，是有一个节点为 leader，其他则为 follower，Leader 是通过内部的选举机制临时产生的。

3）以一个简单的例子来说明整个选举的过程。

假设有五台服务器组成的 zookeeper 集群，它们的 id 从 1-5，同时它们都是最新启动的，

也就是没有历史数据，在存放数据量这一点上，都是一样的。假设这些服务器依序启动，来看看会发生什么。



（1）服务器 1 启动，此时只有它一台服务器启动了，它发出去的报没有任何响应，所以它的选举状态一直是 LOOKING 状态。

（2）服务器 2 启动，它与最开始启动的服务器 1 进行通信，互相交换自己的选举结果，由于两者都没有历史数据，所以 id 值较大的服务器 2 胜出，但是由于没有达到超过半数以上的服务器都同意选举它(这个例子中的半数以上是 3)，所以服务器 1、2 还是继续保持 LOOKING 状态。

（3）服务器 3 启动，根据前面的理论分析，服务器 3 成为服务器 1、2、3 中的老大，而与上面不同的是，此时有三台服务器选举了它，所以它成为了这次选举的 leader。

（4）服务器 4 启动，根据前面的分析，理论上服务器 4 应该是服务器 1、2、3、4 中最大的，但是由于前面已经有半数以上的服务器选举了服务器 3，所以它只能接收当小弟的命了。

（5）服务器 5 启动，同 4 一样当小弟。

3.2 节点类型

1）Znode 有两种类型：

短暂（ephemeral）：客户端和服务器端断开连接后，创建的节点自己删除

持久（persistent）：客户端和服务器端断开连接后，创建的节点不删除

2）Znode 有四种形式的目录节点（默认是 persistent ）

（1）持久化目录节点（PERSISTENT）

客户端与 zookeeper 断开连接后，该节点依旧存在。

（2）持久化顺序编号目录节点（PERSISTENT_SEQUENTIAL）

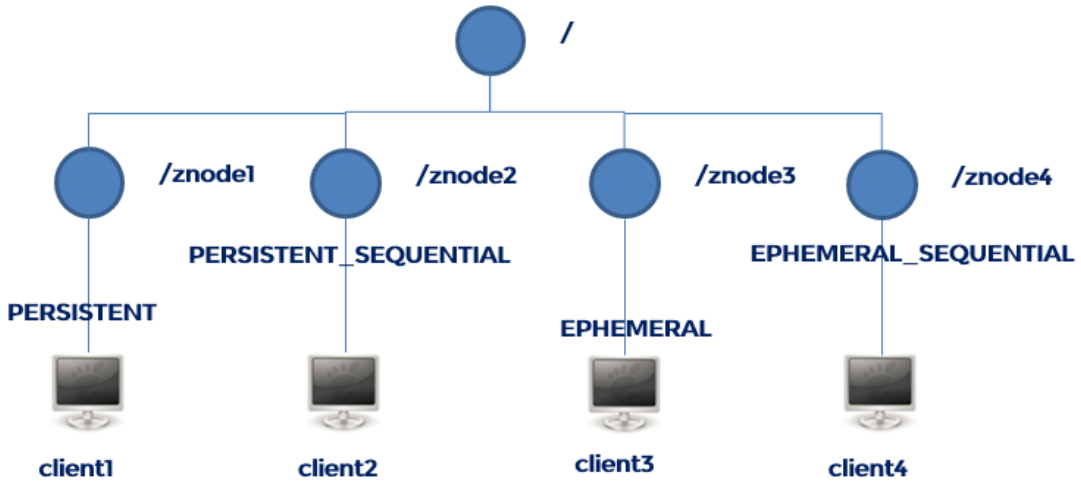
客户端与 zookeeper 断开连接后，该节点依旧存在，只是 Zookeeper 给该节点名称进行顺序编号。

（3）临时目录节点（EPHEMERAL）

客户端与 zookeeper 断开连接后，该节点被删除。

（4）临时顺序编号目录节点（EPHEMERAL_SEQUENTIAL）

客户端与 zookeeper 断开连接后，该节点被删除，只是 Zookeeper 给该节点名称进行顺序编号。



3）创建 znode 时设置顺序标识，znode 名称后会附加一个值，顺序号是一个单调递增的计数器，由父节点维护

4）在分布式系统中，顺序号可以被用于为所有的事件进行全局排序，这样客户端可以通过顺序号推断事件的顺序

3.3 stat 结构体

1）czxid- 引起这个 znode 创建的 zxid，创建节点的事务的 zxid

每次修改 ZooKeeper 状态都会收到一个 zxid 形式的时间戳，也就是 ZooKeeper 事务 ID。

事务 ID 是 ZooKeeper 中所有修改总的次序。每个修改都有唯一的 zxid，如果 zxid1 小于 zxid2，那么 zxid1 在 zxid2 之前发生。

2）ctime - znode 被创建的毫秒数(从 1970 年开始)

3）mzxid - znode 最后更新的 zxid

4）mtime - znode 最后修改的毫秒数(从 1970 年开始)

5）pZxid-znode 最后更新的子节点 zxid

6）cversion - znode 子节点变化号，znode 子节点修改次数

7）dataversion - znode 数据变化号

8）aclVersion - znode 访问控制列表的变化号

9）ephemeralOwner- 如果是临时节点，这个是 znode 拥有者的 session id。如果不是临时节点则是 0。

10）dataLength- znode 的数据长度

11）numChildren - znode 子节点数量

3.4 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（connet），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

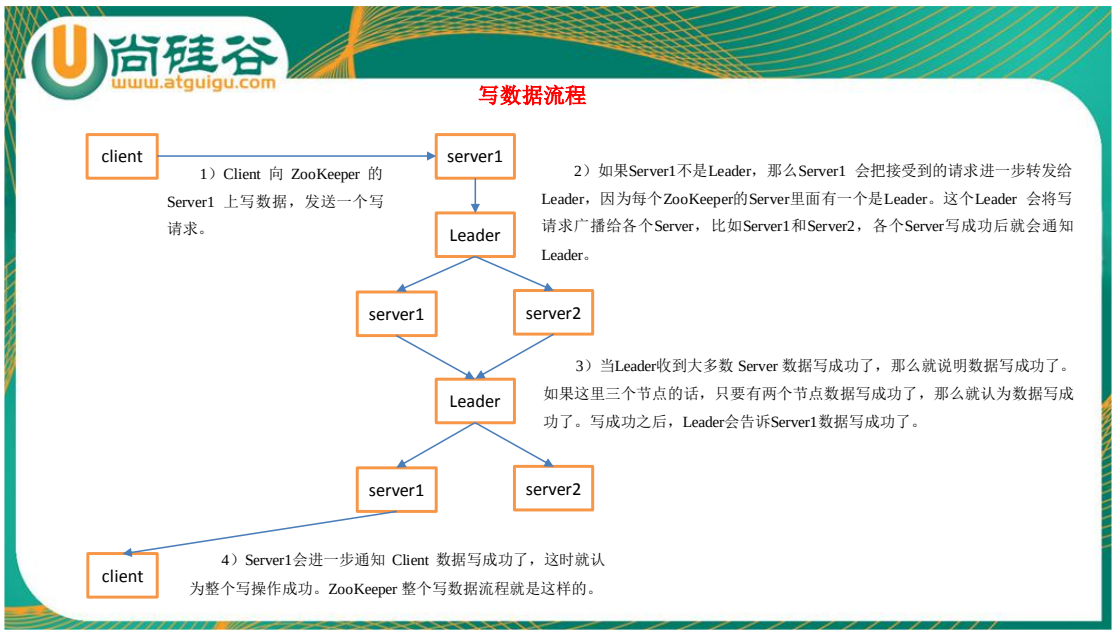
（1）监听节点数据的变化：

get path [watch]

（2）监听子节点增减的变化

ls path [watch]

3.5 写数据流程



四 Zookeeper 实战

4.1 分布式安装部署

0）集群规划

在 hadoop102、hadoop103 和 hadoop104 三个节点上部署 Zookeeper。

1）解压安装

（1）解压 zookeeper 安装包到/opt/module/目录下

[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.4.10.tar.gz -C /opt/module/

（2）在/opt/module/zookeeper-3.4.10/这个目录下创建 zkData

mkdir -p zkData

（3）重命名/opt/module/zookeeper-3.4.10/conf 这个目录下的 zoo_sample.cfg 为 zoo.cfg

mv zoo_sample.cfg zoo.cfg

2）配置 zoo.cfg 文件

（1）具体配置

dataDir=/opt/module/zookeeper-3.4.10/zkData

增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888

server.3=hadoop103:2888:3888

server.4=hadoop104:2888:3888

（2）配置参数解读

Server.A=B:C:D。

A 是一个数字，表示这个是第几号服务器；

B 是这个服务器的 ip 地址；

C 是这个服务器与集群中的 Leader 服务器交换信息的端口；

D 是万一集群中的 Leader 服务器挂了，需要一个端口来重新进行选举，选出一个新的 Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

集群模式下配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面有一个数据就是 A 的值，Zookeeper 启动时读取此文件，拿到里面的数据与 zoo.cfg 里面的配置信息比

较从而判断到底是哪个 server。

3）集群操作

（1）在/opt/module/zookeeper-3.4.10/zkData 目录下创建一个 myid 的文件

touch myid

添加 myid 文件，注意一定要在 linux 里面创建，在 notepad++里面很可能乱码

（2）编辑 myid 文件

vi myid

在文件中添加与 server 对应的编号：如 2

（3）拷贝配置好的 zookeeper 到其他机器上

scp -r zookeeper-3.4.10/ root@hadoop103.atguigu.com:/opt/app/

scp -r zookeeper-3.4.10/ root@hadoop104.atguigu.com:/opt/app/

并分别修改 myid 文件中内容为 3、4

（4）分别启动 zookeeper

[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh start

[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh start

[root@hadoop104 zookeeper-3.4.10]# bin/zkServer.sh start

（5）查看状态

[root@hadoop102 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

[root@hadoop103 zookeeper-3.4.10]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: leader

[root@hadoop104 zookeeper-3.4.5]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

4.2 客户端命令行操作

命令基本语法	功能描述
help	显示所有操作命令
ls path [watch]	使用 ls 命令来查看当前znode中所包含的内容
ls2 path [watch]	查看当前节点数据并能看到更新次数等数据
create	普通创建 -s 含有序列 -e 临时（重启或者超时消失）
get path [watch]	获得节点的值
set	设置节点的具体值
stat	查看节点状态
delete	删除节点
rmr	递归删除节点

1）启动客户端

```
[atguigu@hadoop103 zookeeper-3.4.10]$ bin/zkCli.sh
```

2）显示所有操作命令

```
[zk: localhost:2181(CONNECTED) 1] help
```

3）查看当前 znode 中所包含的内容

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

4）查看当前节点数据并能看到更新次数等数据

```
[zk: localhost:2181(CONNECTED) 1] ls2 /
[zookeeper]
cZxid = 0x0
ctime = Thu Jan 01 08:00:00 CST 1970
mZxid = 0x0
mtime = Thu Jan 01 08:00:00 CST 1970
pZxid = 0x0
```

cversion = -1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 0

numChildren = 1

5）创建普通节点

[zk: localhost:2181(CONNECTED) 2] create /app1 "hello app1"

Created /app1

[zk: localhost:2181(CONNECTED) 4] create /app1/server101 "192.168.1.101"

Created /app1/server101

6）获得节点的值

[zk: localhost:2181(CONNECTED) 6] get /app1

hello app1

cZxid = 0x20000000a

ctime = Mon Jul 17 16:08:35 CST 2017

mZxid = 0x20000000a

mtime = Mon Jul 17 16:08:35 CST 2017

pZxid = 0x20000000b

cversion = 1

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 10

numChildren = 1

[zk: localhost:2181(CONNECTED) 8] get /app1/server101

192.168.1.101

cZxid = 0x20000000b

ctime = Mon Jul 17 16:11:04 CST 2017

mZxid = 0x20000000b

mtime = Mon Jul 17 16:11:04 CST 2017

pZxid = 0x20000000b

cversion = 0

dataVersion = 0

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 13

numChildren = 0

7）创建短暂节点

[zk: localhost:2181(CONNECTED) 9] create -e /app-emphemeral 8888

（1）在当前客户端是能查看到的

[zk: localhost:2181(CONNECTED) 10] ls /

[app1, app-emphemeral, zookeeper]

（2）退出当前客户端然后再重启客户端

[zk: localhost:2181(CONNECTED) 12] quit

[atguigu@hadoop104 zookeeper-3.4.10]$ bin/zkCli.sh

（3）再次查看根目录下短暂节点已经删除

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

[app1, zookeeper]

8）创建带序号的节点

（1）先创建一个普通的根节点 app2

[zk: localhost:2181(CONNECTED) 11] create /app2 "app2"

（2）创建带序号的节点

[zk: localhost:2181(CONNECTED) 13] create -s /app2/aa 888

Created /app2/aa0000000000

[zk: localhost:2181(CONNECTED) 14] create -s /app2/bb 888

Created /app2/bb0000000001

[zk: localhost:2181(CONNECTED) 15] create -s /app2/cc 888

Created /app2/cc0000000002

如果原节点下有 1 个节点，则再排序时从 1 开始，以此类推。

[zk: localhost:2181(CONNECTED) 16] create -s /app1/aa 888

Created /app1/aa0000000001

9）修改节点数据值

[zk: localhost:2181(CONNECTED) 2] set /app1 999

10）节点的值变化监听

（1）在 104 主机上注册监听/app1 节点数据变化

[zk: localhost:2181(CONNECTED) 26] get /app1 watch

（2）在 103 主机上修改/app1 节点的数据

[zk: localhost:2181(CONNECTED) 5] set /app1 777

（3）观察 104 主机收到数据变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/app1

11）节点的子节点变化监听（路径变化）

（1）在 104 主机上注册监听/app1 节点的子节点变化

[zk: localhost:2181(CONNECTED) 1] ls /app1 watch

[aa0000000001, server101]

（2）在 103 主机/app1 节点上创建子节点

[zk: localhost:2181(CONNECTED) 6] create /app1/bb 666

Created /app1/bb

（3）观察 104 主机收到子节点变化的监听

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/app1

12）删除节点

[zk: localhost:2181(CONNECTED) 4] delete /app1/bb

13）递归删除节点

[zk: localhost:2181(CONNECTED) 7] rmr /app2

14）查看节点状态

[zk: localhost:2181(CONNECTED) 12] stat /app1

cZxid = 0x20000000a

ctime = Mon Jul 17 16:08:35 CST 2017

mZxid = 0x200000018

mtime = Mon Jul 17 16:54:38 CST 2017

pZxid = 0x20000001c

cversion = 4

dataVersion = 2

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 3

numChildren = 2

4.3 API 应用

4.3.1 Eclipse 环境搭建

1）创建一个工程

2）解压 zookeeper-3.4.10.tar.gz 文件

3）拷贝 zookeeper-3.4.10.jar、jline-0.9.94.jar、log4j-1.2.16.jar、netty-3.10.5.Final.jar、slf4j-api-1.6.1.jar、slf4j-log4j12-1.6.1.jar 到工程的 lib 目录。并 build 一下，导入工程。

4）拷贝 log4j.properties 文件到项目根目录

log4j.properties

4.3.2 创建 ZooKeeper 客户端

```
private static String connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";
    private static int sessionTimeout = 2000;
    private ZooKeeper zkClient = null;

    @Before
    public void init() throws Exception {

    zkClient = new ZooKeeper(connectString, sessionTimeout, new Watcher() {
            @Override
```

```
                public void process(WatchedEvent event) {
                    // 收到事件通知后的回调函数（用户的业务逻辑）
                    System.out.println(event.getType() + "--" + event.getPath());

                    // 再次启动监听
                    try {
                        zkClient.getChildren("/", true);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            });
    }
```

4.3.3 创建子节点

```
    // 创建子节点
    @Test
    public void create() throws Exception {
        // 数据的增删改查
        // 参数 1：要创建的节点的路径； 参数 2：节点数据 ； 参数 3：节点权限 ；参数 4：节点的类型
        String nodeCreated = zkClient.create("/eclipse", "hello zk".getBytes(), Ids.OPEN_ACL_UNSAFE,CreateMode.PERSISTENT);
    }
```

4.3.4 获取子节点并监听

```
    // 获取子节点
    @Test
    public void getChildren() throws Exception {
        List<String> children = zkClient.getChildren("/", true);

        for (String child : children) {
            System.out.println(child);
        }

        // 延时阻塞
        Thread.sleep(Long.MAX_VALUE);
    }
```

4.3.5 判断 znode 是否存在

```
    // 判断 znode 是否存在
    @Test
    public void exist() throws Exception {
```

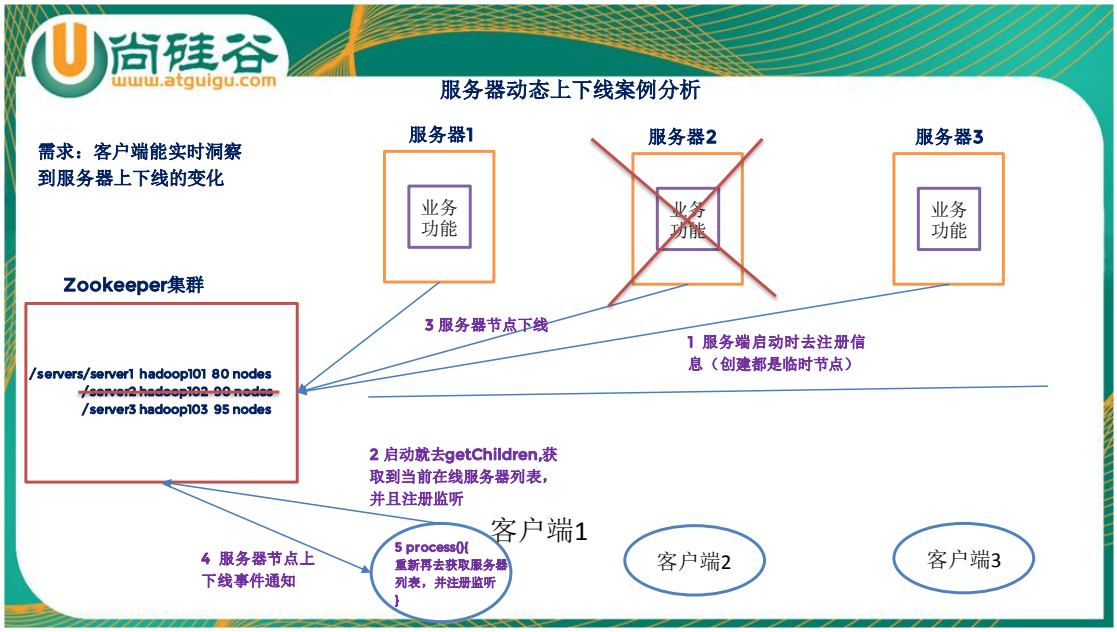
```
        Stat stat = zkClient.exists("/eclipse", false);

        System.out.println(stat == null ? "not exist" : "exist");
    }
```

4.4 案例实战

4.4.1 监听服务器节点动态上下线案例

1）需求：某分布式系统中，主节点可以有多台，可以动态上下线，任意一台客户端都能实时感知到主节点服务器的上下线

2）需求分析



3）具体实现：

（0）现在集群上创建/servers 节点

```
[zk: localhost:2181(CONNECTED) 10] create /servers "servers"
Created /servers
```

（1）服务器端代码

```
package com.atguigu.zkcase;
import java.io.IOException;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.ZooDefs.Ids;
```

```
public class DistributeServer {

    private static String connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";
    private static int sessionTimeout = 2000;
    private ZooKeeper zk = null;
    private String parentNode = "/servers";

    // 创建到 zk 的客户端连接
    public void getConnect() throws IOException{

        zk = new ZooKeeper(connectString, sessionTimeout, new Watcher() {

            @Override
            public void process(WatchedEvent event) {

            }
        });
    }

    // 注册服务器
    public void registServer(String hostname) throws Exception{
        String create = zk.create(parentNode + "/server", hostname.getBytes(), Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);

        System.out.println(hostname +" is noline "+ create);
    }

    // 业务功能
    public void business(String hostname) throws Exception{
        System.out.println(hostname+" is working ...");

        Thread.sleep(Long.MAX_VALUE);
    }

    public static void main(String[] args) throws Exception {
        // 获取 zk 连接
        DistributeServer server = new DistributeServer();
        server.getConnect();

        // 利用 zk 连接注册服务器信息
        server.registServer(args[0]);
```

```
            // 启动业务功能
            server.business(args[0]);
        }
}
```

（2）客户端代码

```
package com.atguigu.zkcase;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooKeeper;

public class DistributeClient {
    private static String connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";
    private static int sessionTimeout = 2000;
    private ZooKeeper zk = null;
    private String parentNode = "/servers";
    private volatile ArrayList<String> serversList = new ArrayList<>();

    // 创建到 zk 的客户端连接
    public void getConnect() throws IOException {
        zk = new ZooKeeper(connectString, sessionTimeout, new Watcher() {

            @Override
            public void process(WatchedEvent event) {

                // 再次启动监听
                try {
                    getServerList();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    //
    public void getServerList() throws Exception {

        // 获取服务器子节点信息，并且对父节点进行监听
        List<String> children = zk.getChildren(parentNode, true);
```

```
            ArrayList<String> servers = new ArrayList<>();

            for (String child : children) {
                byte[] data = zk.getData(parentNode + "/" + child, false, null);

                servers.add(new String(data));
            }

            // 把 servers 赋值给成员 serverList，已提供给各业务线程使用
            serversList = servers;

            System.out.println(serversList);
        }

        // 业务功能
        public void business() throws Exception {
            System.out.println("client is working ...");
Thread.sleep(Long.MAX_VALUE);
        }

        public static void main(String[] args) throws Exception {

            // 获取 zk 连接
            DistributeClient client = new DistributeClient();
            client.getConnect();

            // 获取 servers 的子节点信息，从中获取服务器信息列表
            client.getServerList();

            // 业务进程启动
            client.business();
        }
}
```