尚硅谷大数据技术之 Zookeeper

（作者：尚硅谷大数据研发部）

版本：V3.2

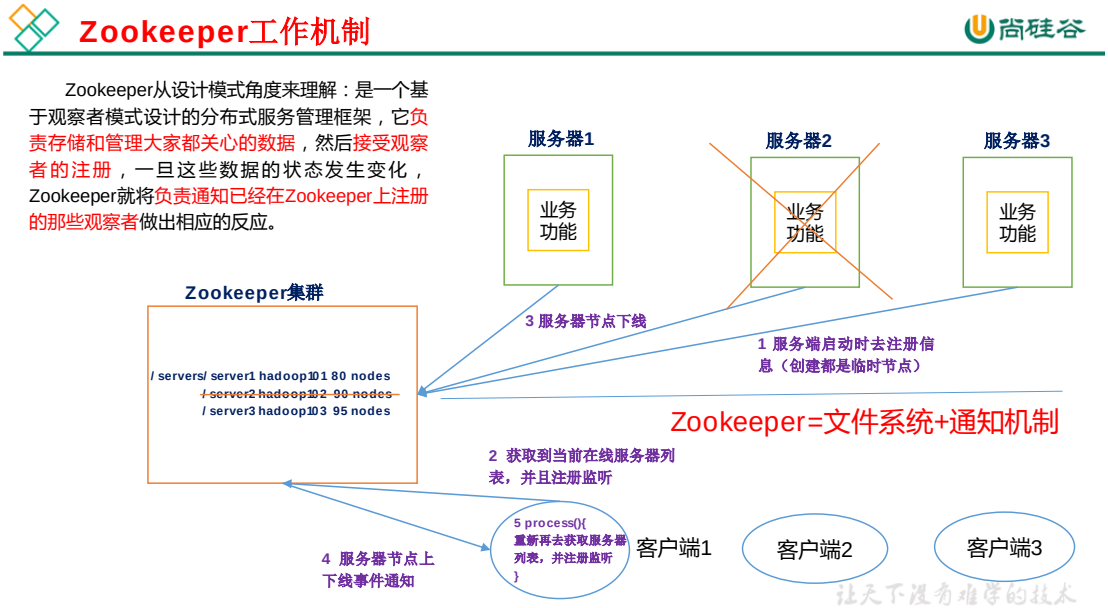
第 1 章 Zookeeper 入门

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。

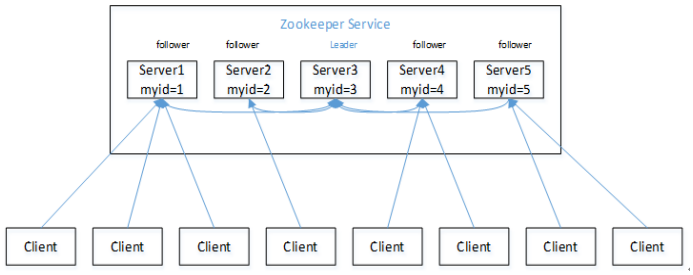
Zookeeper 从设计模式角度来理解，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生了变化，Zookeeper 就负责通知已经在 Zookeeper 上注册的那些观察者做出相应的反应.

Zookeeper = 文件系统 + 通知机制



1.2 特点

Zookeeper特点



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

2）集群中只要有半数以上节点存活，Zookeeper集群就能正常服务。

3）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。

4）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。

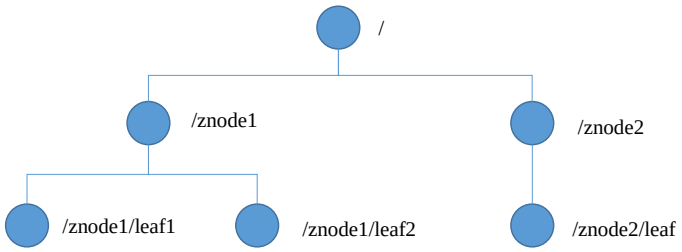
5）数据更新原子性，一次数据更新要么成功，要么失败。

6）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

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。



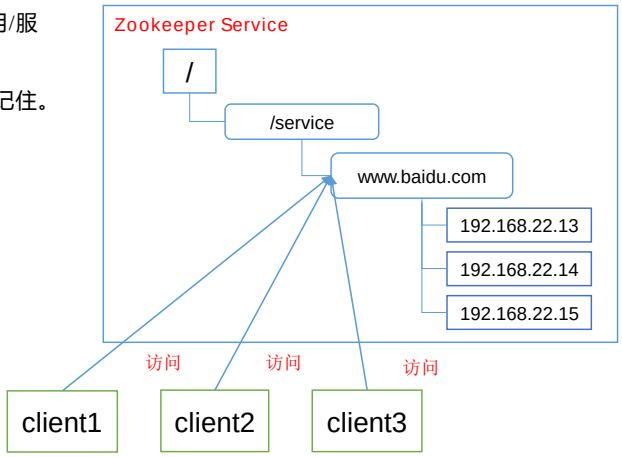
1.4 应用场景

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。

统一命名服务

在分布式环境下，经常需要对应用/服务进行统一命名，便于识别。

例如：IP不容易记住，而域名容易记住。



统一配置管理

1）分布式环境下，配置文件同步非常常见。

（1）一般要求一个集群中，所有节点的配置信息是一致的，比如 Kafka 集群。

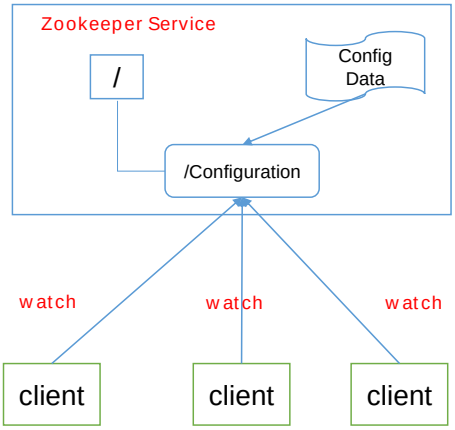
（2）对配置文件修改后，希望能够快速同步到各个节点上。

2）配置管理可交由ZooKeeper实现。

（1）可将配置信息写入ZooKeeper上的一个Znode。

（2）各个客户端服务器监听这个Znode。

（3）一旦Znode中的数据被修改，ZooKeeper将通知各个客户端服务器。



统一集群管理

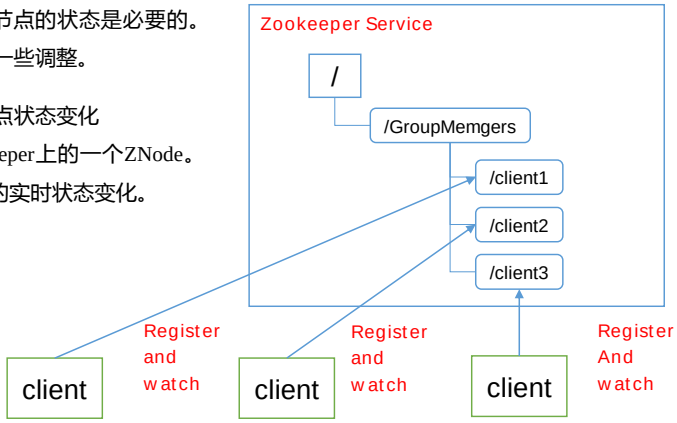
1）分布式环境中，实时掌握每个节点的状态是必要的。

（1）可根据节点实时状态做出一些调整。

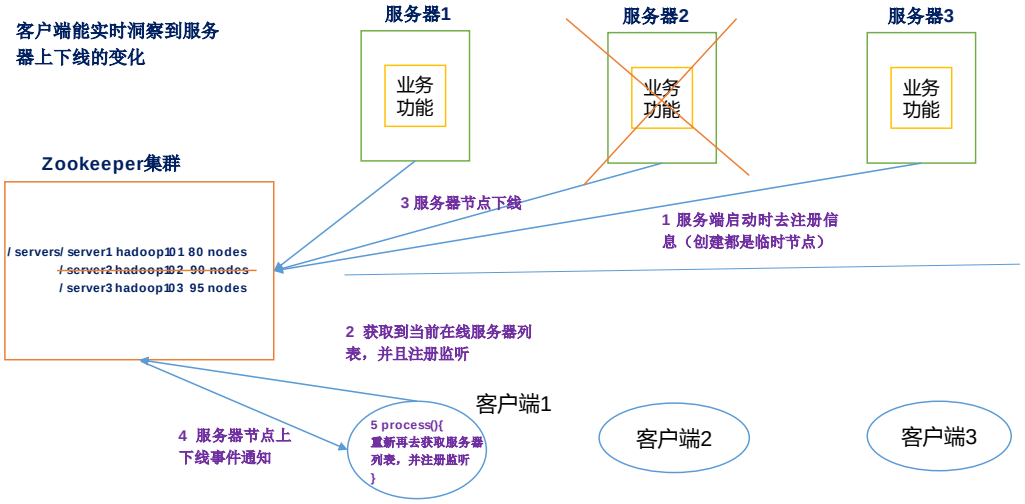
2）ZooKeeper可以实现实时监控节点状态变化

（1）可将节点信息写入ZooKeeper上的一个ZNode。

（2）监听这个ZNode可获取它的实时状态变化。

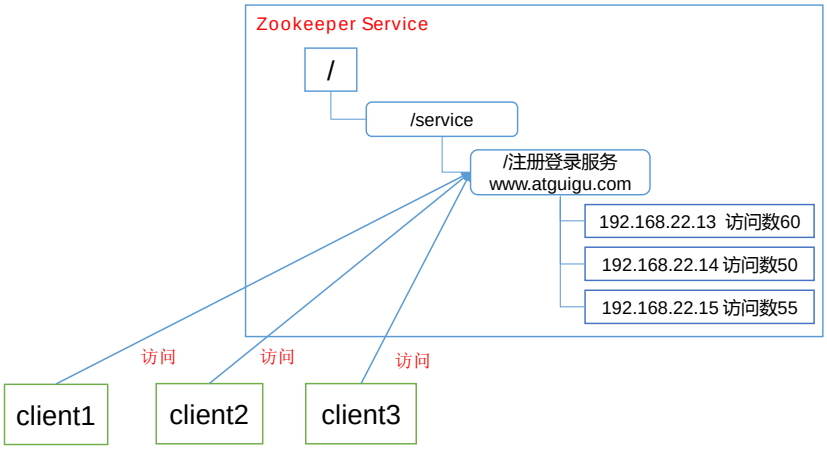


服务器动态上下线



软负载均衡

在Zookeeper中记录每台服务器的访问数，让访问数最少的服务器去处理最新的客户端请求

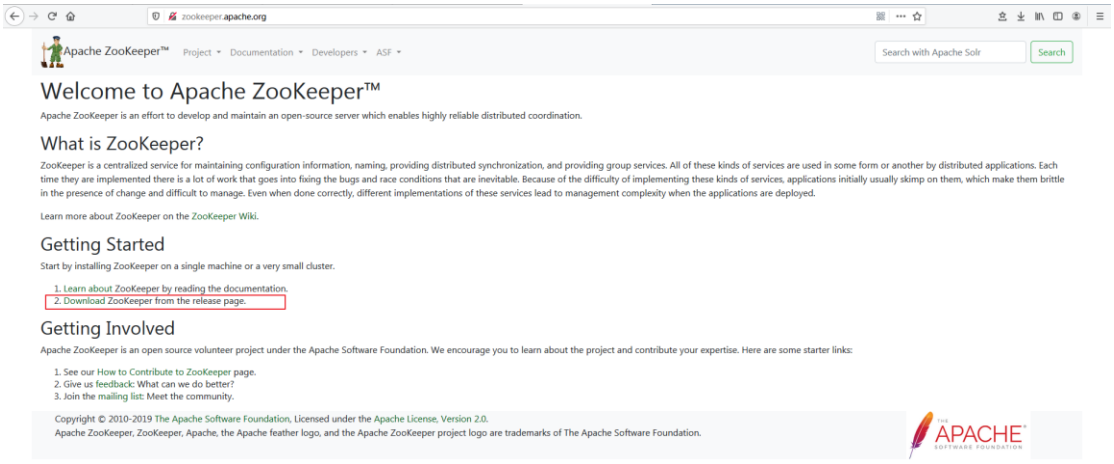


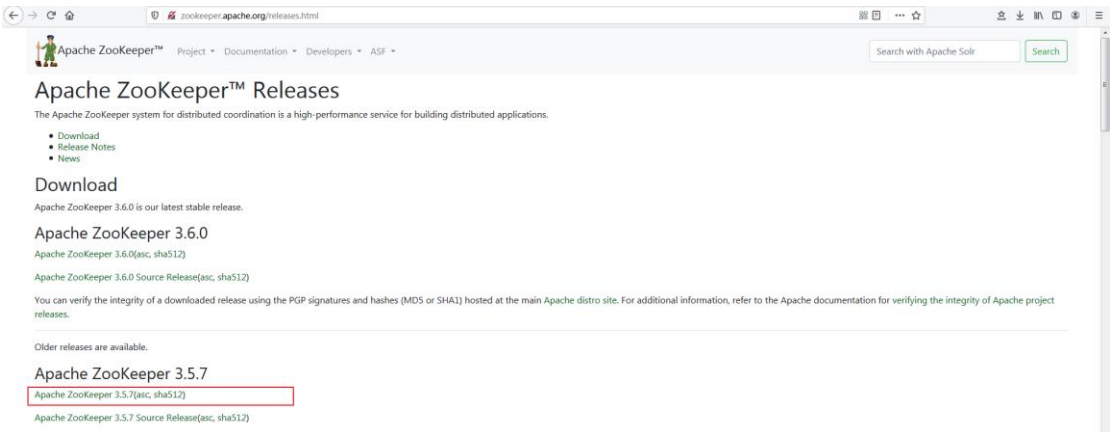
1.5 下载地址

1）官网首页：

https://zookeeper.apache.org/

2）下载截图





第 2 章 Zookeeper 安装

2.1 本地模式安装部署

1）安装前准备

（1）安装 Jdk

（2）拷贝 Zookeeper 安装包到 Linux 系统下

（3）解压到指定目录

```
[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.5.7.tar.gz
-C /opt/module/
```

2）配置修改

（1）将/opt/module/zookeeper-3.5.7/conf 这个路径下的 zoo_sample.cfg 修改为 zoo.cfg；

```
[atguigu@hadoop102 conf]$ mv zoo_sample.cfg zoo.cfg
```

（2）打开 zoo.cfg 文件，修改 dataDir 路径：

```
[atguigu@hadoop102 zookeeper-3.5.7]$ vim zoo.cfg
```

修改如下内容：

dataDir=/opt/module/zookeeper-3.5.7/zkData

（3）在/opt/module/zookeeper-3.5.7/这个目录上创建 zkData 文件夹

```
[atguigu@hadoop102 zookeeper-3.5.7]$ mkdir zkData
```

3）操作 Zookeeper

（1）启动 Zookeeper

```
[atguigu@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start
```

（2）查看进程是否启动

```
[atguigu@hadoop102 zookeeper-3.5.7]$ jps
4020 Jps
4001 QuorumPeerMain
```

（3）查看状态：

```
[atguigu@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh status
ZooKeeper JMX enabled by default
Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg
Mode: standalone
```

（4）启动客户端：

```
[atguigu@hadoop102 zookeeper-3.5.7]$ bin/zkCli.sh
```

（5）退出客户端：

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

（6）停止 Zookeeper

```
[atguigu@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh stop
```

2.2 配置参数解读

Zookeeper中的配置文件zoo.cfg中参数含义解读如下：

1）tickTime =2000：通信心跳数，Zookeeper服务器与客户端心跳时间，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：LF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。

3）syncLimit =5：LF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。

4）dataDir：数据文件目录+数据持久化路径

主要用于保存 Zookeeper 中的数据。

5）clientPort =2181：客户端连接端口

监听客户端连接的端口。

第 3 章 Zookeeper 实战（开发重点）

3.1 分布式安装部署

1）集群规划

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

2）解压安装

（1）解压 Zookeeper 安装包到/opt/module/目录下

```
[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.5.7.tar.gz -C /opt/module/
```

（2）同步/opt/module/zookeeper-3.5.7 目录内容到 hadoop103、hadoop104

```
[atguigu@hadoop102 module]$ xsync zookeeper-3.5.7/
```

3）配置服务器编号

（1）在/opt/module/zookeeper-3.5.7/这个目录下创建 zkData

```
[atguigu@hadoop102 zookeeper-3.5.7]$ mkdir -p zkData
```

（2）在/opt/module/zookeeper-3.5.7/zkData 目录下创建一个 myid 的文件

```
[atguigu@hadoop102 zkData]$ 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

（3）编辑 myid 文件

```
[atguigu@hadoop102 zkData]$ vi myid
```

在文件中添加与 server 对应的编号：

```
2
```

（4）拷贝配置好的 zookeeper 到其他机器上

```
[atguigu@hadoop102 zkData]$ xsync myid
```

并分别在 hadoop103、hadoop104 上修改 myid 文件中内容为 3、4

4）配置zoo.cfg文件

（1）重命名/opt/module/zookeeper-3.5.7/conf 这个目录下的 zoo_sample.cfg 为 zoo.cfg

```
[atguigu@hadoop102 conf]$ mv zoo_sample.cfg zoo.cfg
```

（2）打开 zoo.cfg 文件

```
[atguigu@hadoop102 conf]$ vim zoo.cfg
```

修改数据存储路径配置

```
dataDir=/opt/module/zookeeper-3.5.7/zkData
```

增加如下配置

```
#######################cluster##########################
server.2=hadoop102:2888:3888
server.3=hadoop103:2888:3888
server.4=hadoop104:2888:3888
```

（3）同步 zoo.cfg 配置文件

```
[atguigu@hadoop102 conf]$ xsync zoo.cfg
```

（4）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；

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。

B 是这个服务器的地址；

C 是这个服务器 Follower 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。

5）集群操作

（1）分别启动 Zookeeper

```
[atguigu@hadoop102 zookeeper-3.5.7]$ bin/zkServer.sh start
[atguigu@hadoop103 zookeeper-3.5.7]$ bin/zkServer.sh start
[atguigu@hadoop104 zookeeper-3.5.7]$ bin/zkServer.sh start
```

（2）查看状态

```
[atguigu@hadoop102 zookeeper-3.5.7]# bin/zkServer.sh status
JMX enabled by default
Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg
Mode: follower
[atguigu@hadoop103 zookeeper-3.5.7]# bin/zkServer.sh status
JMX enabled by default
Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg
Mode: leader
[atguigu@hadoop104 zookeeper-3.5.7]# bin/zkServer.sh status
JMX enabled by default
Using config: /opt/module/zookeeper-3.5.7/bin/../conf/zoo.cfg
Mode: follower
```

3.2 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	使用 ls 命令来查看当前 znode 的子节点 -w 监听子节点变化 -s 附加次级信息
create	普通创建 -s 含有序列 -e 临时（重启或者超时消失）
get path	获得节点的值 -w 监听节点内容变化 -s 附加次级信息
set	设置节点的具体值
stat	查看节点状态
delete	删除节点
deleteall	递归删除节点

1）启动客户端

```
[atguigu@hadoop103 zookeeper-3.5.7]$ 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）查看当前节点详细数据

```
[zk: localhost:2181(CONNECTED) 1] ls -s /
[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）分别创建2个普通节点

```
[zk: localhost:2181(CONNECTED) 3] create /sanguo "diaochan"
Created /sanguo
[zk: localhost:2181(CONNECTED) 4] create /sanguo/shuguo
"liubei"
Created /sanguo/shuguo
```

6）获得节点的值

```
[zk: localhost:2181(CONNECTED) 5] get /sanguo
diaochan
[zk: localhost:2181(CONNECTED) 6] get -s /sanguo
diaochan
cZxid = 0x100000003
ctime = Wed Aug 29 00:03:23 CST 2018
mZxid = 0x100000003
mtime = Wed Aug 29 00:03:23 CST 2018
pZxid = 0x100000004
cversion = 1
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 7
numChildren = 1
[zk: localhost:2181(CONNECTED) 7]
[zk: localhost:2181(CONNECTED) 7] get -s /sanguo/shuguo
liubei
cZxid = 0x100000004
ctime = Wed Aug 29 00:04:35 CST 2018
mZxid = 0x100000004
mtime = Wed Aug 29 00:04:35 CST 2018
pZxid = 0x100000004
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

7）创建临时节点

```
[zk: localhost:2181(CONNECTED) 7] create -e /sanguo/wuguo
"zhouyu"
Created /sanguo/wuguo
```

（1）在当前客户端是能查看到的

```
[zk: localhost:2181(CONNECTED) 3] ls /sanguo
[wuguo, shuguo]
```

（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-3.5.7]$ 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 /sanguo
[shuguo]
```

8）创建带序号的节点

（1）先创建一个普通的根节点/sanguo/weiguo

```
[zk: localhost:2181(CONNECTED) 1] create /sanguo/weiguo
"caocao"
Created /sanguo/weiguo
```

（2）创建带序号的节点

```
[zk: localhost:2181(CONNECTED) 2] create /sanguo/weiguo
"caocao"
Node already exists: /sanguo/weiguo
[zk: localhost:2181(CONNECTED) 3] create -s /sanguo/weiguo
"caocao"
Created /sanguo/weiguo0000000000
[zk: localhost:2181(CONNECTED) 4] create -s /sanguo/weiguo
"caocao"
Created /sanguo/weiguo0000000001
[zk: localhost:2181(CONNECTED) 5] create -s /sanguo/weiguo
"caocao"
Created /sanguo/weiguo0000000002
[zk: localhost:2181(CONNECTED) 6] ls /sanguo
[shuguo, weiguo, weiguo0000000000, weiguo0000000001,
weiguo0000000002, wuguo]
[zk: localhost:2181(CONNECTED) 6]
```

如果节点下原来没有子节点，序号从 0 开始依次递增。如果原节点下已有 2 个节点，则再排序时从 2 开始，以此类推。

9）修改节点数据值

```
[zk: localhost:2181(CONNECTED) 6] set /sanguo/weiguo "caopi"
```

10）节点的值变化监听

（1）在 hadoop104 主机上注册监听/sanguo 节点数据变化

```
[zk: localhost:2181(CONNECTED) 26] [zk:
localhost:2181(CONNECTED) 8] get -w /sanguo
```

（2）在 hadoop103 主机上修改/sanguo 节点的数据

```
[zk: localhost:2181(CONNECTED) 1] set /sanguo "xishi"
```

（3）观察 hadoop104 主机收到数据变化的监听

```
WATCHER::
WatchedEvent state:SyncConnected type:NodeDataChanged
path:/sanguo
```

11）节点的子节点变化监听（路径变化）

（1）在 hadoop104 主机上注册监听/sanguo 节点的子节点变化

```
[zk: localhost:2181(CONNECTED) 1] ls -w /sanguo
[aa0000000001, server101]
```

（2）在 hadoop103 主机/sanguo 节点上创建子节点

```
[zk: localhost:2181(CONNECTED) 2] create /sanguo/jin "simayi"
Created /sanguo/jin
```

（3）观察 hadoop104 主机收到子节点变化的监听

```
WATCHER::
WatchedEvent    state:SyncConnected    type:NodeChildrenChanged
path:/sanguo
```

12）删除节点

```
[zk: localhost:2181(CONNECTED) 4] delete /sanguo/jin
```

13）递归删除节点

```
[zk: localhost:2181(CONNECTED) 15] deleteall /sanguo/shuguo
```

14）查看节点状态

```
[zk: localhost:2181(CONNECTED) 17] stat /sanguo
cZxid = 0x100000003
ctime = Wed Aug 29 00:03:23 CST 2018
mZxid = 0x100000011
mtime = Wed Aug 29 00:21:23 CST 2018
pZxid = 0x100000014
cversion = 9
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 4
numChildren = 1
```

3.3 API 应用

3.3.1 IDEA 环境搭建

1）创建一个Maven Module

2）添加pom文件

```
<dependencies>
       <dependency>
           <groupId>junit</groupId>
           <artifactId>junit</artifactId>
           <version>RELEASE</version>
       </dependency>
       <dependency>
           <groupId>org.apache.logging.log4j</groupId>
           <artifactId>log4j-core</artifactId>
           <version>2.8.2</version>
       </dependency>
       <dependency>
           <groupId>org.apache.zookeeper</groupId>
           <artifactId>zookeeper</artifactId>
           <version>3.5.7</version>
       </dependency>
</dependencies>
```

3）拷贝log4j.properties文件到项目根目录

需要在项目的 src/main/resources 目录下，新建一个文件，命名为“log4j.properties”，在

文件中填入。

```
log4j.rootLogger=INFO, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d    %p    [%c]
- %m%n
log4j.appender.logfile=org.apache.log4j.FileAppender
log4j.appender.logfile.File=target/spring.log
log4j.appender.logfile.layout=org.apache.log4j.PatternLayout
log4j.appender.logfile.layout.ConversionPattern=%d    %p    [%c]
- %m%n
```

3.3.2 初始化 ZooKeeper 客户端

```
public class Zookeeper {

    private String connectString;
    private int sessionTimeout;
    private ZooKeeper zkClient;

    @Before   //获取客户端对象
    public void init() throws IOException {

        connectString = "hadoop102:2181,hadoop103:2181,hadoop104:2181";
        int sessionTimeout = 10000;

       //参数解读 1 集群连接字符串  2 连接超时时间 单位:毫秒  3 当前客户端默认的监控器
        zkClient   =   new   ZooKeeper(connectString,   sessionTimeout,   new
Watcher() {
            @Override
            public void process(WatchedEvent event) {
            }
        });
    }

    @After //关闭客户端对象
    public void close() throws InterruptedException {
        zkClient.close();
    }
}
```

3.3.3 获取子节点列表,不监听

```
@Test
public void ls() throws IOException, KeeperException, InterruptedException
{
    //用客户端对象做各种操作
    List<String> children = zkClient.getChildren("/", false);
    System.out.println(children);
}
```

3.3.4 获取子节点列表,并监听

```
@Test
public void lsAndWatch() throws KeeperException, InterruptedException {
    List<String> children = zkClient.getChildren("/atguigu", new Watcher()
{
        @Override
        public void process(WatchedEvent event) {
            System.out.println(event);
        }
```

```
    });
    System.out.println(children);

    //因为设置了监听,所以当前线程不能结束
    Thread.sleep(Long.MAX_VALUE);
}
```

3.3.5 创建子节点

```
@Test
public void create() throws KeeperException, InterruptedException {
    //参数解读 1 节点路径  2 节点存储的数据
    //3 节点的权限(使用 Ids 选个 OPEN 即可) 4 节点类型 短暂 持久 短暂带序号 持久带序号
     String path = zkClient.create("/atguigu", "shanguigu".getBytes(),
ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);

    //创建临时节点
    //String path = zkClient.create("/atguigu2", "shanguigu".getBytes(),
ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL);

    System.out.println(path);

    //创建临时节点的话,需要线程阻塞
    //Thread.sleep(10000);
}
```

3.3.6 判断 Znode 是否存在

```
@Test
public void exist() throws Exception {

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

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

3.3.7 获取子节点存储的数据,不监听

```
@Test
public void get() throws KeeperException, InterruptedException {
    //判断节点是否存在
    Stat stat = zkClient.exists("/atguigu", false);
    if (stat == null) {
        System.out.println("节点不存在...");
        return;
    }

    byte[] data = zkClient.getData("/atguigu", false, stat);
    System.out.println(new String(data));
}
```

3.3.8 获取子节点存储的数据,并监听

```
@Test
public void getAndWatch() throws KeeperException, InterruptedException {
    //判断节点是否存在
    Stat stat = zkClient.exists("/atguigu", false);
    if (stat == null) {
        System.out.println("节点不存在...");
        return;
    }
```

```
    byte[] data = zkClient.getData("/atguigu", new Watcher() {
        @Override
        public void process(WatchedEvent event) {
            System.out.println(event);
        }
    }, stat);
    System.out.println(new String(data));
    //线程阻塞
    Thread.sleep(Long.MAX_VALUE);
}
```

3.3.9 设置节点的值

```
@Test
public void set() throws KeeperException, InterruptedException {
    //判断节点是否存在
    Stat stat = zkClient.exists("/atguigu", false);
    if (stat == null) {
        System.out.println("节点不存在...");
        return;
    }
    //参数解读 1 节点路径 2 节点的值 3 版本号
    zkClient.setData("/atguigu", "sgg".getBytes(), stat.getVersion());
}
```

3.3.10 删除空节点

```
@Test
public void delete() throws KeeperException, InterruptedException {
    //判断节点是否存在
    Stat stat = zkClient.exists("/aaa", false);
    if (stat == null) {
        System.out.println("节点不存在...");
        return;
    }
    zkClient.delete("/aaa", stat.getVersion());
}
```

3.3.11 删除非空节点,递归实现

```
//封装一个方法,方便递归调用
public void deleteAll(String path, ZooKeeper zk) throws KeeperException,
InterruptedException {
    //判断节点是否存在
    Stat stat = zkClient.exists(path, false);
    if (stat == null) {
        System.out.println("节点不存在...");
        return;
    }
    //先获取当前传入节点下的所有子节点
    List<String> children = zk.getChildren(path, false);
    if (children.isEmpty()) {
        //说明传入的节点没有子节点,可以直接删除
        zk.delete(path, stat.getVersion());
    } else {
        //如果传入的节点有子节点,循环所有子节点
        for (String child : children) {
            //删除子节点,但是不知道子节点下面还有没有子节点,所以递归调用
            deleteAll(path + "/" + child, zk);
        }
        //删除完所有子节点以后,记得删除传入的节点
        zk.delete(path, stat.getVersion());
```

```
    }
}
//测试 deleteAll
@Test
public void testDeleteAll() throws KeeperException, InterruptedException
{
    deleteAll("/atguigu",zkClient);
}
```

第 4 章 Zookeeper 内部原理

4.1 节点类型

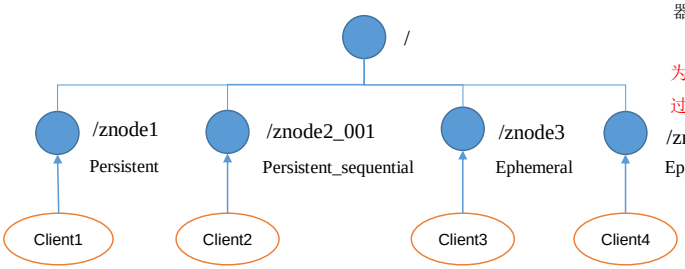
节点类型

持久（Persistent）：客户端和服务器端断开连接后，创建的节点不删除

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

说明：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

注意：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



（1）持久化目录节点

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

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

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

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。

4.2 Stat 结构体

（1）czxid-创建节点的事务 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 最后更新的事务 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

4.3 监听器原理（面试重点）

监听器原理

尚硅谷

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。

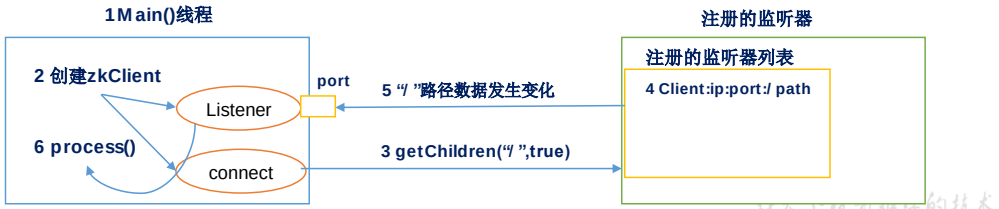
2、常见的监听

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]



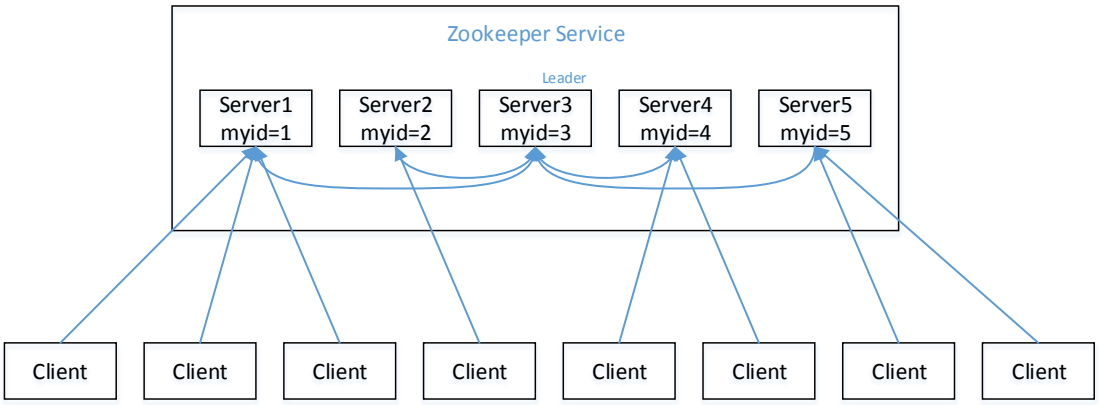
4.4 选举机制（面试重点）

（1）半数机制：集群中半数以上机器存活，集群可用。所以 Zookeeper 适合安装奇数台服务器。

（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。



Zookeeper 的选举机制

（1）服务器 1 启动，发起一次选举。服务器 1 投自己一票。此时服务器 1 票数一票，不够半数以上（3 票），选举无法完成，服务器 1 状态保持为 LOOKING；

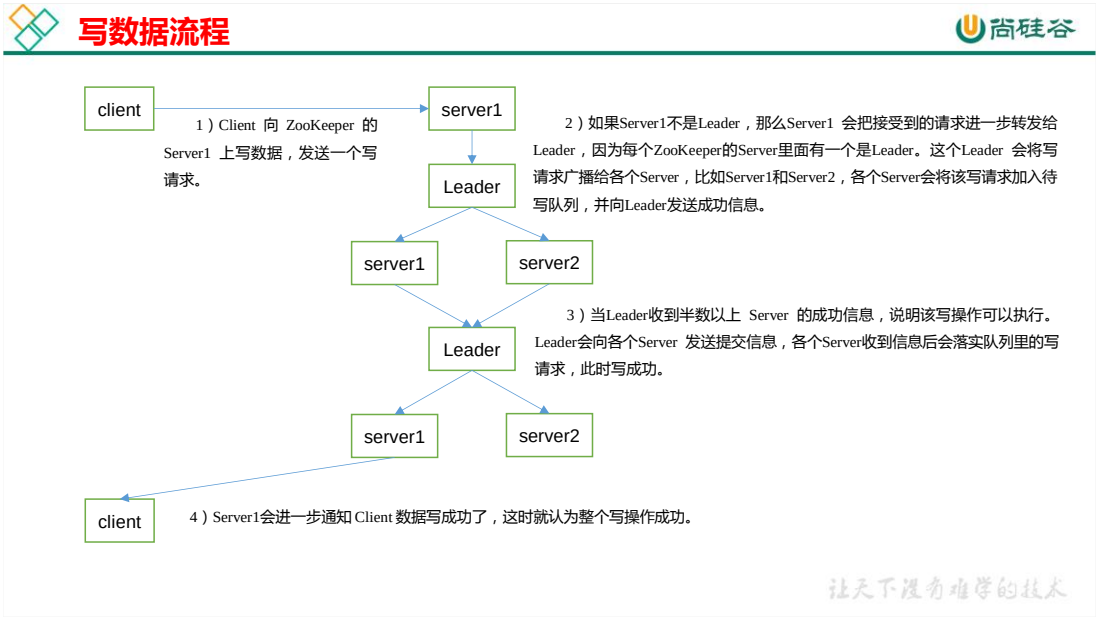
（2）服务器 2 启动，再发起一次选举。服务器 1 和 2 分别投自己一票并交换选票信息：此时服务器 1 发现服务器 2 的 ID 比自己目前投票推举的（服务器 1）大，更改选票为推举服务器 2。此时服务器 1 票数 0 票，服务器 2 票数 2 票，没有半数以上结果，选举无法完成，服务器 1，2 状态保持 LOOKING

（3）服务器 3 启动，发起一次选举。此时服务器 1 和 2 都会更改选票为服务器 3。此次投票结果：服务器 1 为 0 票，服务器 2 为 0 票，服务器 3 为 3 票。此时服务器 3 的票数已经超过半数，服务器 3 当选 Leader。服务器 1，2 更改状态为 FOLLOWING，服务器 3 更改状态为 LEADING；

（4）服务器 4 启动，发起一次选举。此时服务器 1，2，3 已经不是 LOOKING 状态，不会更改选票信息。交换选票信息结果：服务器 3 为 3 票，服务器 4 为 1 票。此时服务器 4 服从多数，更改选票信息为服务器 3，并更改状态为 FOLLOWING；

（5）服务器 5 启动，同 4 一样当小弟。

4.5 写数据流程



第 5 章 企业面试真题

5.1 请简述 ZooKeeper 的选举机制

详见 4.4。

5.2 ZooKeeper 的监听原理是什么？

详见 4.3。

5.3 ZooKeeper 的部署方式有哪几种？集群中的角色有哪些？集群最少需要几台机器？

（1）部署方式单机模式、集群模式
（2）角色：Leader 和 Follower
（3）集群最少需要机器数：3

5.4 ZooKeeper 的常用命令

ls create get delete set…