前言

> 声明：参考来源互联网，有任何争议可以留言。站在前人的肩上，我们才能看的更远。

> 本教程纯手打，致力于最实用教程，不需要什么奖励，只希望多多转发支持。
> 欢迎来我公众号，希望可以结识你，你有什么想看的可以催更，微信搜索：[JavaPub]

> 有任何问题都可以来谈谈，等你哦！

> 如果你对zookeeper有一定了解，那么直接跳到你需要的知识点。

1. 什么是Zookeeper

1.2.Zookeeper简介

> **ZooKeeper: A Distributed Coordination Service for Distributed Applications**
> ZooKeeper is a distributed, open-source coordination service for distributed applications. It exposes a simple set of primitives that distributed applications can build upon to implement higher level services for synchronization, configuration maintenance, and groups and naming. It is designed to be easy to program to, and uses a data model styled after the familiar directory tree structure of file systems. It runs in Java and has bindings for both Java and C.
> Coordination services are notoriously hard to get right. They are especially prone to errors such as race conditions and deadlock. The motivation behind ZooKeeper is to relieve distributed applications the responsibility of implementing coordination services from scratch.
> https://zookeeper.apache.org/doc/current/zookeeperOver.html
>
> 官网地址：https://zookeeper.apache.org/doc/r3.4.12/index.html

上边是Zookeeper官网的描述，as everyone knows，ZooKeeper是分布式应用程序的分布式协调服务。

1.3.为什么要用Zookeeper

学习一个东西，anyhow，知道为什么学它至关重要。

看到一个比较靠谱的例子：

> 一个团队里面，需要一个leader，leader是干嘛用的？管理什么的咱不说，就说如果外面的人，想问关于这个团队的一切事情，首先就会去找这个leader，因为他知道的最多，而且他的回答最靠谱。
>
> 比如产品经理小饼过来要人，作为leader，老吕发现小耀最近没有项目安排，于是把小耀安排给了小饼的项目；
>
> 过了一会，另一个产品小西也过来要人，老吕发现刚刚把小耀安排走了，已经没人，于是就跟小西说，人都被你们产品要走了，你们产品自己去协调去。
>
> 如果老吕这时候忘了小耀已经被安排走了，把小耀也分配给小西，那到时两个产品就要打架了。
>
> 这就是leader在团队里的【协调作用】。
>
> 同样的，在分布式系统中，也需要这样的协调者，来回答系统下各个节点的提问。

Zookeeper能完美解决分布式协调服务这个问题

> 但是这个例子属于单机模式，当我们扩展为三台服务器集群，小西过来问leader02要人，这时leader们信息还没有同步。

> 这时就会涉及到Zookeeper的其他几点特性：
> 1、配置信息同步
> 2、分布式锁控制
> 3、消息的发布与订阅（典型的生产者消费者模型）
> 4、集群内节点状态的快速感知
>
> 当信息还没有同步完成时，不对外提供服务，阻塞住查询请求，等待信息同步完成，再给查询请求返回信息。

这样的系统，就叫分布式协调系统。谁能把这个数据同步的时间压缩的更短，谁的请求响应就更快，谁就更出色，Zookeeper就是其中的佼佼者。

它用起来像单机一样，能够提供数据强一致性，但是其实背后是多台机器构成的集群，不会有SPOF。
单点故障

2. Zookeeper介绍

2.1 百度百科

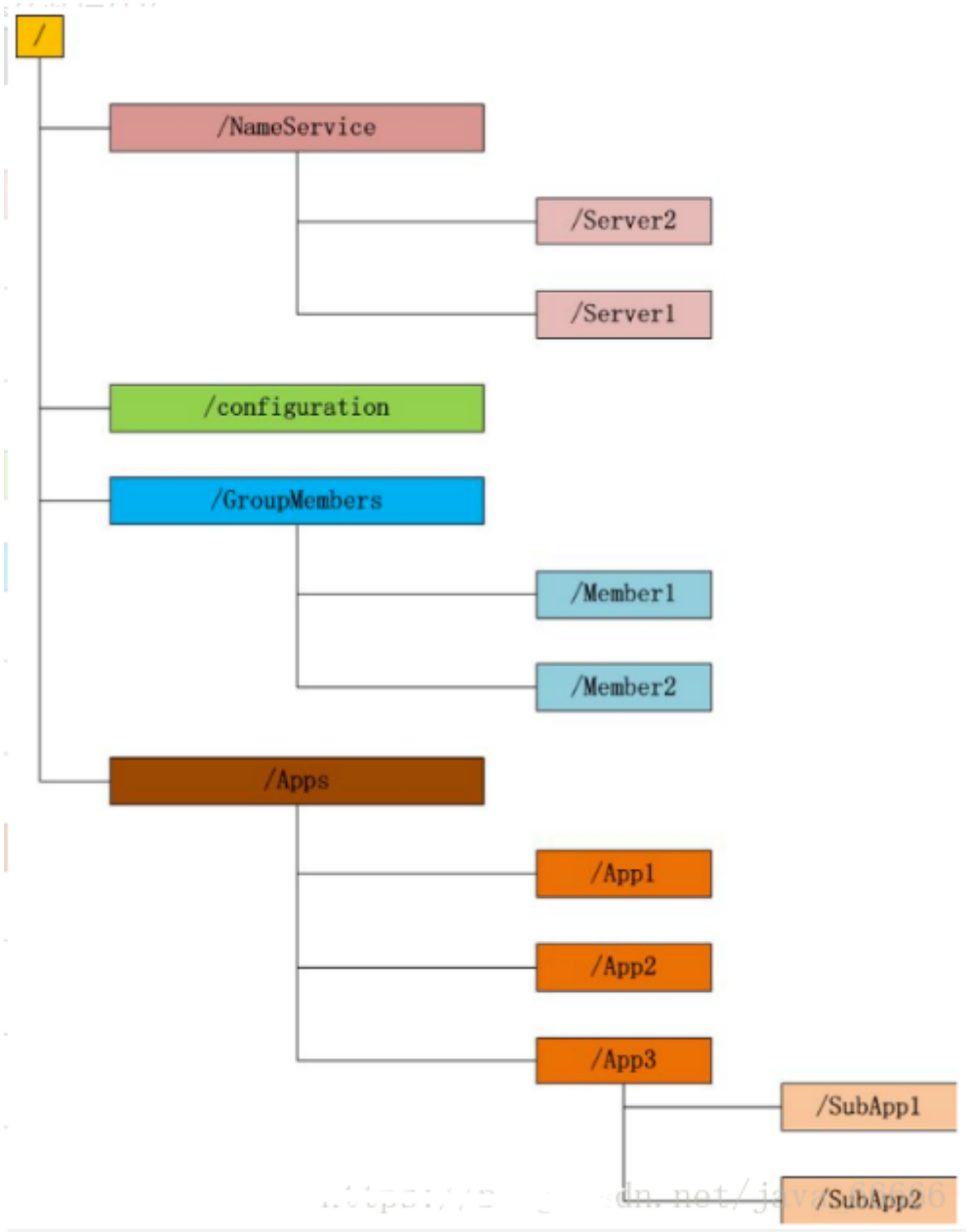
> ZooKeeper是一个分布式的，开放源码的分布式应用程序协调服务，是Google的Chubby一个开源的实现，是Hadoop和Hbase的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。
> ZooKeeper的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。
> ZooKeeper包含一个简单的原语集，提供Java和C的接口。
> ZooKeeper代码版本中，提供了分布式独享锁、选举、队列的接口，代码在$zookeeper_home\src\recipes。其中分布锁和队列有Java和C两个版本，选举只有Java版本。

看了上面的介绍，你还不知道zookeeper是什么，那么简单来说：**zookeeper=文件系统+监听通知机制。**

包含如下四种节点：临时节点（EPHEMERAL）、永久节点（persistent）、有编号节点（Persistent_sequential）、临时有编号（Ephemral_ sequential）

2.2. 文件系统

Zookeeper维护一个类似文件系统的数据结构：



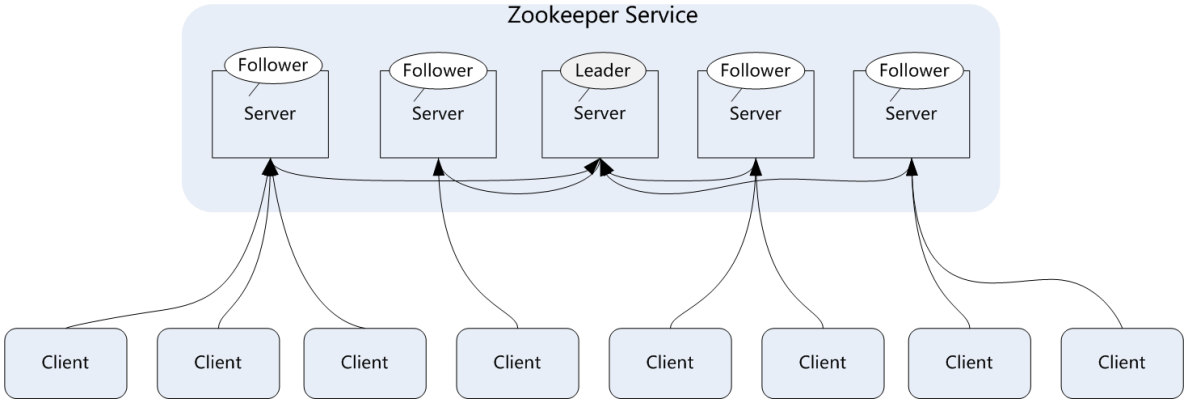
2.3. 监听通知机制

客户端注册监听它关心的目录节点，当目录节点发生变化（数据改变、被删除、子目录节点增加删除）时，zookeeper会通知客户端。

- 3、 Zookeeper能做什么
  zookeeper功能非常强大，可以实现诸如分布式应用配置管理、统一命名服务、状态同步服务、集群管理等功能，我们这里拿比较简单的分布式应用配置管理为例来说明。

假设我们的程序是分布式部署在多台机器上，如果我们要改变程序的配置文件，需要逐台机器去修改，非常麻烦，现在把这些配置全部放到zookeeper上去，保存在 zookeeper 的某个目录节点中，然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 zookeeper 的通知，然后从 zookeeper 获取新的配置信息应用到系统中。

3. Zookeeper整体架构



4. 快速入门(quick start)

4.1.安装

- 单机安装

Zookeeper是解压包，只需要安装、解压就可以启动使用

```
wget https://mirrors.tuna.tsinghua.edu.cn/apache/zookeeper/zookeeper-3.4.14/zookeeper-3.4.14.tar.gz
```

```
tar -zxf ./zookeeper-3.4.14.tar.gz
```

在conf目录下，有默认启动配置文件【zoo_sample.cfg】，复制一份到同级目录下【zoo.cfg】。

配置文件解读：

```
# tickTime这个时间是作为zookeeper服务器之间或客户端与服务器之间维持心跳的时间间隔,也就是说每个tickTime时间就会发送一个心跳。（以毫秒为单位）
tickTime = 2000
# dataDir ZooKeeper的状态存储位置，看名字就知是数据目录。在你的系统中检查这个目录是否存在，如果不存在手动创建，并且给予可写权限。
dataDir = /path/to/zookeeper/data
# 这个端口就是客户端连接Zookeeper服务器的端口,Zookeeper会监听这个端口接受客户端的访问请求;
clientPort = 2181
# initLimit这个配置项是用来配置zookeeper接受客户端（这里所说的客户端不是用户连接zookeeper服务器的客户端,而是zookeeper服务器集群中连接到leader的follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。
# 当已经超过10个心跳的时间（也就是tickTime）长度后 zookeeper 服务器还没有收到客户端的返回信息,那么表明这个客户端连接失败。总的时间长度就是 5*2000=10秒。
initLimit = 5
# syncLimit这个配置项标识leader与follower之间发送消息,请求和应答时间长度,最长不能超过多少个tickTime的时间长度,总的时间长度就是2*2000=4秒
syncLimit = 2
# 日志存放的位置
dataLogDir=/path/to/zookeeper/log

# 2888,3888 are election port
# 2888端口是zookeeper服务之间的通讯的端口，3888是zookeeper与其他应用程序通讯的端口。
```

```
# server.A=B:C:D中的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第一个端口用来集群成员的信息交换,表示这个服务器与集群中的leader服务器交换信息的端口，D是在leader挂掉时专门用来进行选举leader所用的端口。
server.1=localhost:2888:3888
```

- 集群安装

  本例搭建的是伪集群模式，即一台机器上启动三个zookeeper实例组成集群，真正的集群模式无非就是实例IP地址不同，搭建方法没有区别

  - 配置说明

    1. tickTime：这个时间是作为 Zookeeper 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。
    2. initLimit：这个配置项是用来配置 Zookeeper 接受客户端（这里所说的客户端不是用户连接 Zookeeper 服务器的客户端，而是 Zookeeper 服务器集群中连接到 Leader 的 Follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。当已经超过 10个心跳的时间（也就是 tickTime）长度后 Zookeeper 服务器还没有收到客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是 10*2000=20 秒
    3. syncLimit：这个配置项标识 Leader 与 Follower 之间发送消息，请求和应答时间长度，最长不能超过多少个 tickTime 的时间长度，总的时间长度就是 5*2000=10秒
    4. dataDir：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。
    5. clientPort：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。
    6. server.A=B：C：D：其中 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 表示的是这个服务器与集群中的 Leader 服务器交换信息的端口；D 表示的是万一集群中的 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。如果是伪集群的配置方式，由于 B 都是一样，所以不同的 Zookeeper 实例通信端口号不能一样，所以要给它们分配不同的端口号。

1. 伪分布式，三个节点。

   ```
   # cp conf/zoo_sample.cfg conf/zoo-1.cfg

   # cp conf/zoo_sample.cfg conf/zoo-2.cfg

   # cp conf/zoo_sample.cfg conf/zoo-3.cfg
   ```

2. 修改dataDir和clientPort不同即可

   ```
   # vim conf/zoo-2.cfg

   dataDir=/tmp/zookeeper-2

   clientPort=2182

   # vim conf/zoo-2.cfg

   dataDir=/tmp/zookeeper-3

   clientPort=2183
   ```

3. 标识Server ID，设置每个节点的server id

   ```
   # cd /tmp/zookeeper-1

   # vim myid

   # cd /tmp/zookeeper-2

   # vim myid
   ```

> # cd /tmp/zookeeper-3
>
> # vim myid

4. 启动

> # bin/zkServer.sh start conf/zoo-1.cfg
> # bin/zkServer.sh start conf/zoo-2.cfg
> # bin/zkServer.sh start conf/zoo-3.cfg

5. 完成

> 至此，集群搭建完成，可以连接试试了

> bin/zkServer.sh status conf/zoo-1.cfg

4.2.启动

```
[root@iz2zehz5b1m03ahtrhebcaz bin]# ./zkServer.sh
ZooKeeper JMX enabled by default
Using config: /home/soft/zookeeper-3.4.8/bin/../conf/zoo.cfg
Usage: ./zkServer.sh {start|start-foreground|stop|restart|status|upgrade|print-cmd}
```

进入到bin目录下，

启动

```
./zkServer.sh start
```

状态

```
./zkServer.sh status
```

4.3.查询

> Zookeeper客户端连接指令

进入zookeeper下bin目录

```
[root@iz2zehz5b1m03ahtrhebcaz bin]# pwd
/home/soft/zookeeper-3.4.8/bin
[root@iz2zehz5b1m03ahtrhebcaz bin]# ll
total 44
-rwxr-xr-x 1 elasticsearch elasticsearch  232 Feb  6  2016 README.txt
-rwxr-xr-x 1 elasticsearch elasticsearch 1937 Feb  6  2016 zkCleanup.sh
-rwxr-xr-x 1 elasticsearch elasticsearch 1056 Feb  6  2016 zkCli.cmd
-rwxr-xr-x 1 elasticsearch elasticsearch 1534 Feb  6  2016 zkCli.sh
-rwxr-xr-x 1 elasticsearch elasticsearch 1628 Feb  6  2016 zkEnv.cmd
-rwxr-xr-x 1 elasticsearch elasticsearch 2696 Feb  6  2016 zkEnv.sh
-rwxr-xr-x 1 elasticsearch elasticsearch 1089 Feb  6  2016 zkServer.cmd
-rwxr-xr-x 1 elasticsearch elasticsearch 6773 Feb  6  2016 zkServer.sh
-rw-r--r-- 1 root          root          7850 May  4 13:26 zookeeper.out
```

可以看到很多脚本文件，通过*zkCli.sh*连接客户端

> ./zkCli.sh -server 127.0.0.1:2181

通过*ls /*，查看已注册服务，
例如查询dubbo

> ls /dubbo

可以看到dubbo服务地外提供的接口

消费者、生产者

> ls /dubbo/com.ivan.service.provider.UserService/consumers

> ls /dubbo/com.ivan.service.provider.UserService/providers

5. 常用指令

> 客户端连接后整体使用和linux很相似，上一章做了一些介绍

- zookeeper通过 **./bin/zkServer.sh start** 命令启动后,通过客户端连接

> ./bin/zkCli.sh -server ip:port

```
[root@iz2zehz5b1m03ahtrhebcaz zookeeper-3.4.8]# ./bin/zkCli.sh -server 127.0.0.1:2181
Connecting to 127.0.0.1:2181
2020-05-14 13:52:21,530 [myid:] - INFO  [main:Environment@100] - Client
environment:zookeeper.version=3.4.8--1, built on 02/06/2016 03:18 GMT
2020-05-14 13:52:21,533 [myid:] - INFO  [main:Environment@100] - Client
environment:host.name=iz2zehz5b1m03ahtrhebcaz
2020-05-14 13:52:21,533 [myid:] - INFO  [main:Environment@100] - Client
environment:java.version=1.8.0_144
2020-05-14 13:52:21,539 [myid:] - INFO  [main:Environment@100] - Client
environment:java.vendor=Oracle Corporation
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:java.home=/home/soft/java/jdk1.8.0_144/jre
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:java.class.path=/home/soft/zookeeper-
3.4.8/bin/../build/classes:/home/soft/zookeeper-
3.4.8/bin/../build/lib/*.jar:/home/soft/zookeeper-3.4.8/bin/../lib/slf4j-log4j12-
1.6.1.jar:/home/soft/zookeeper-3.4.8/bin/../lib/slf4j-api-
1.6.1.jar:/home/soft/zookeeper-3.4.8/bin/../lib/netty-
3.7.0.Final.jar:/home/soft/zookeeper-3.4.8/bin/../lib/log4j-
1.2.16.jar:/home/soft/zookeeper-3.4.8/bin/../lib/jline-
0.9.94.jar:/home/soft/zookeeper-3.4.8/bin/../zookeeper-3.4.8.jar:/home/soft/zookeeper-
3.4.8/bin/../src/java/lib/*.jar:/home/soft/zookeeper-
3.4.8/bin/../conf:.:/home/soft/java/jdk1.8.0_144/lib/dt.jar:/home/soft/java/jdk1.8.0_1
44/lib/tools.jar
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:java.library.path=/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr
/lib
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:java.io.tmpdir=/tmp
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:java.compiler=<NA>
```

```
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:os.name=Linux
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:os.arch=amd64
2020-05-14 13:52:21,540 [myid:] - INFO  [main:Environment@100] - Client
environment:os.version=3.10.0-514.26.2.el7.x86_64
2020-05-14 13:52:21,541 [myid:] - INFO  [main:Environment@100] - Client
environment:user.name=root
2020-05-14 13:52:21,541 [myid:] - INFO  [main:Environment@100] - Client
environment:user.home=/root
2020-05-14 13:52:21,541 [myid:] - INFO  [main:Environment@100] - Client
environment:user.dir=/home/soft/zookeeper-3.4.8
2020-05-14 13:52:21,542 [myid:] - INFO  [main:ZooKeeper@438] - Initiating client
connection, connectString=127.0.0.1:2181 sessionTimeout=30000
watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@799f7e29
Welcome to ZooKeeper!
JLine support is enabled
2020-05-14 13:52:21,579 [myid:] - INFO  [main-
SendThread(127.0.0.1:2181):ClientCnxn$SendThread@1032] - Opening socket connection to
server 127.0.0.1/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown
error)
2020-05-14 13:52:21,688 [myid:] - INFO  [main-
SendThread(127.0.0.1:2181):ClientCnxn$SendThread@876] - Socket connection established
to 127.0.0.1/127.0.0.1:2181, initiating session
2020-05-14 13:52:21,709 [myid:] - INFO  [main-
SendThread(127.0.0.1:2181):ClientCnxn$SendThread@1299] - Session establishment
complete on server 127.0.0.1/127.0.0.1:2181, sessionid = 0x171de1d5d640005, negotiated
timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null
```

- 查看当前包含的内容

```
[zk: 127.0.0.1:2181(CONNECTED) 2] ls /
[cluster, controller_epoch, brokers, zookeeper, admin, isr_change_notification,
consumers, log_dir_event_notification, latest_producer_id_block, config]
```

- 创建一个 znode ,使用create /zkPro myData

```
[zk: 127.0.0.1:2181(CONNECTED) 3] create /zkPro MyData
Created /zkPro
```

- 查一下创建的内容

```
[zk: 127.0.0.1:2181(CONNECTED) 4] ls /
[cluster, controller_epoch, brokers, zookeeper, zkPro, admin, isr_change_notification,
consumers, log_dir_event_notification, latest_producer_id_block, config]
```

- 我们运行 get 命令来确认第二步中所创建的 znode 是否包含我们所创建的字符串：

```
[zk: 127.0.0.1:2181(CONNECTED) 5] get /zkPro MyData
MyData
cZxid = 0x338
ctime = Thu May 14 13:57:29 CST 2020
mZxid = 0x338
mtime = Thu May 14 13:57:29 CST 2020
pZxid = 0x338
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

- 通过 set 命令来对 zk 所关联的字符串进行设置：

```
WatchedEvent state:SyncConnected type:NodeDataChanged path:/zkPro
cZxid = 0x338
ctime = Thu May 14 13:57:29 CST 2020
mZxid = 0x339
mtime = Thu May 14 14:06:01 CST 2020
pZxid = 0x338
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 9
numChildren = 0
```

- 删除节点

```
[zk: 127.0.0.1:2181(CONNECTED) 7] delete /zkPro
[zk: 127.0.0.1:2181(CONNECTED) 8]
```

6. 应用场景

场景一 配置文件

我们在开发的时候，有时候需要获取一些公共的配置，比如数据库连接信息等，并且偶然可能需要更新配置。如果我们的服务器有N多台的话，那修改起来会特别的麻烦，并且还需要重新启动。这里Zookeeper就可以很方便的实现类似的功能。

场景二 分布式锁

在我们日常的开发中，如果是单个进程中对共享资源的访问，我们只需要用synchronized或者lock就能实现互斥操作。但是对于跨进程、跨主机、跨网络的共享资源似乎就无能为力了。

场景三 分布式队列

在日常使用中，特别是像生产者消费者模式中，经常会使用BlockingQueue来充当缓冲区的角色。但是在分布式系统中这种方式就不能使用BlockingQueue来实现了，但是Zookeeper可以实现。

场景四 负载均衡

首先我们需要简单的理解分布式和集群，通俗点说：分布式就是将一个系统拆分到多个独立运行的应用中（有可能在同一台主机也有可能在不同的主机上），集群就是将单个独立的应用复制多分放在不同的主机上来减轻服务器的压力。而Zookeeper不仅仅可以作为分布式集群的服务注册调度中心（例如dubbo），也可以实现集群的负载均衡。

- Zookeeper是一个功能非常强大的应用，除了上面几种应用外，还有命名服务、分布式协调通知等也是常用的场景。

7. 选举机制

> 选举机制，顾名思义就是投票选举。

分布式集群开发的目的就是为了保证系统的稳定运行，如果有一个服务挂掉，不会对整个系统造成大的影响。

Leader选举是保证分布式数据一致性的关键所在。当Zookeeper集群中的一台服务器出现以下两种情况之一时，需要进入Leader选举。

1. 服务器初始化启动。
2. 服务器运行期间无法和Leader保持连接。

- 情况一

1．服务器启动时期的Leader选举

若进行Leader选举，则至少需要两台机器，这里选取3台机器组成的服务器集群为例。在集群初始化阶段，当有一台服务器Server1启动时，其单独无法进行和完成Leader选举，当第二台服务器Server2启动时，此时两台机器可以相互通信，每台机器都试图找到Leader，于是进入Leader选举过程。选举过程如下

(1) 每个Server发出一个投票。由于是初始情况，Server1和Server2都会将自己作为Leader服务器来进行投票，每次投票会包含所推举的服务器的myid和ZXID，使用(myid, ZXID)来表示，此时Server1的投票为(1, 0)，Server2的投票为(2, 0)，然后各自将这个投票发给集群中其他机器。

(2) 接受来自各个服务器的投票。集群的每个服务器收到投票后，首先判断该投票的有效性，如检查是否是本轮投票、是否来自LOOKING状态的服务器。

(3) 处理投票。针对每一个投票，服务器都需要将别人的投票和自己的投票进行PK，PK规则如下

- 优先检查ZXID。ZXID比较大的服务器优先作为Leader。
- 如果ZXID相同，那么就比较myid。myid较大的服务器作为Leader服务器。

对于Server1而言，它的投票是(1, 0)，接收Server2的投票为(2, 0)，首先会比较两者的ZXID，均为0，再比较myid，此时Server2的myid最大，于是更新自己的投票为(2, 0)，然后重新投票，对于Server2而言，其无须更新自己的投票，只是再次向集群中所有机器发出上一次投票信息即可。

(4) 统计投票。每次投票后，服务器都会统计投票信息，判断是否已经有过半机器接受到相同的投票信息，对于Server1、Server2而言，都统计出集群中已经有两台机器接受了(2, 0)的投票信息，此时便认为已经选出了Leader。

(5) 改变服务器状态。一旦确定了Leader，每个服务器就会更新自己的状态，如果是Follower，那么就变更为FOLLOWING，如果是Leader，就变更为LEADING。

- 情况二

2．服务器运行时期的Leader选举

在Zookeeper运行期间，Leader与非Leader服务器各司其职，即便当有非Leader服务器宕机或新加入，此时也不会影响Leader，但是一旦Leader服务器挂了，那么整个集群将暂停对外服务，进入新一轮Leader选举，其过程和启动时期的Leader选举过程基本一致。假设正在运行的有Server1、Server2、Server3三台服务器，当前Leader是Server2，若某一时刻Leader挂了，此时便开始Leader选举。选举过程如下

(1) 变更状态。Leader挂后，余下的非Observer服务器都会讲自己的服务器状态变更为LOOKING，然后开始进入Leader选举过程。

(2) 每个Server会发出一个投票。在运行期间，每个服务器上的ZXID可能不同，此时假定Server1的ZXID为123，Server3的ZXID为122；在第一轮投票中，Server1和Server3都会投自己，产生投票(1, 123)，(3, 122)，然后各自将投票发送给集群中所有机器。

(3) 接收来自各个服务器的投票。与启动时过程相同。

(4) 处理投票。与启动时过程相同，此时，Server1将会成为Leader。

(5) 统计投票。与启动时过程相同。

(6) 改变服务器的状态。与启动时过程相同。

总结：

1. 在选举时每个节点都有一个(myid,ZXID)表示；
2. 对于初始化或leader宕机时，每个server发出一个投票给集群其他机器，所有请求挂起，开始选举（如实例一）；
3. 超过半数的投票，就会成为Leader；
4. 比较自己的选票和接收到的投票，优先比较ZXID，再比较myid。如果大于自己，更换自己的选票并告诉其他server；

> - ZXID 是指当前服务器数据越新，其成为Leader可能性越大。
> - myid 是指当前server编号

8. 三大功能

1. 为用户提供数据的注册和查询服务
2. 为用户提供数据节点的监听注册服务
3. 跟用户之间保持心跳通信以感知用户的状态

9. Java Api 操作zookeeper

- **分布式配置中心**

1. jar引入

```
        <dependency>
            <groupId>org.apache.zookeeper</groupId>
            <artifactId>zookeeper</artifactId>
            <version>3.6.1</version>
        </dependency>
```

2. 创建节点

```
[zk: localhost:2181(CONNECTED) 6] create /username javapub
Created /username
```

3. 启动客户端代码

```
package javapub;

/**
 * @author wangshiyu rodert
 * @date 2020/5/18 13:15
 * @description
 */
import java.util.concurrent.CountDownLatch;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.Watcher.Event.EventType;
import org.apache.zookeeper.Watcher.Event.KeeperState;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.data.Stat;

/**
 * 分布式配置中心demo
 * @author
 *
 */
public class ZooKeeperProSync implements Watcher {

    private static CountDownLatch connectedSemaphore = new CountDownLatch(1);
    private static ZooKeeper zk = null;
    private static Stat stat = new Stat();

    public static void main(String[] args) throws Exception {
        //zookeeper配置数据存放路径
        String path = "/username";
        //连接zookeeper并且注册一个默认的监听器
        zk = new ZooKeeper("127.0.0.1:2181", 5000, //
                new ZooKeeperProSync());
        //等待zk连接成功的通知
        connectedSemaphore.await();
        //获取path目录节点的配置数据，并注册默认的监听器
        System.out.println(new String(zk.getData(path, true, stat)));

        Thread.sleep(Integer.MAX_VALUE);
    }

    public void process(WatchedEvent event) {
        if (KeeperState.SyncConnected == event.getState()) {  //zk连接成功通知事件
            if (EventType.None == event.getType() && null == event.getPath()) {
                connectedSemaphore.countDown();
            } else if (event.getType() == EventType.NodeDataChanged) {  //zk目录节点数
据变化通知事件
```

```
                try {
                    System.out.println("配置已修改，新值为：" + new
String(zk.getData(event.getPath(), true, stat)));
                } catch (Exception e) {
                }
            }
        }
    }
}
```

> 启动java客户端代码，正确读取到 /username 目录下数据，javapub

4. 修改 /username 下数据

```
[zk: localhost:2181(CONNECTED) 7] set /username javapub-rodert
cZxid = 0x4
ctime = Mon May 18 13:20:59 CST 2020
mZxid = 0x6
mtime = Mon May 18 13:27:54 CST 2020
pZxid = 0x4
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 14
numChildren = 0
```

> 客户端同步如下:
> 配置已修改，新值为：javapub-rodert