

ZooKeeper

尚硅谷 JAVA 研究院

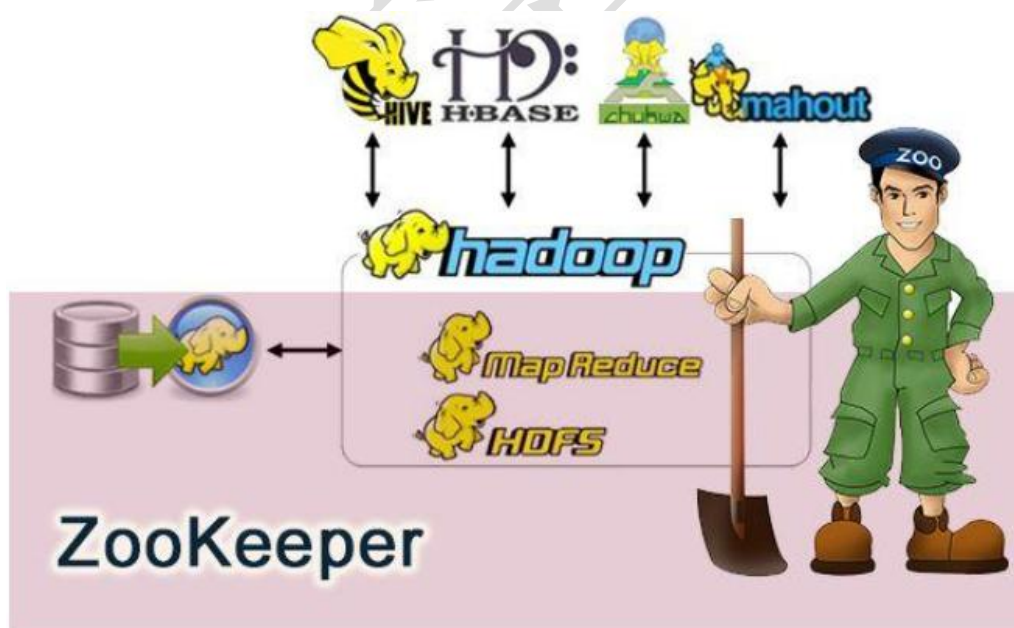
版本：V1.1

第 1 章 ZooKeeper 简介

1. 是什么

ZooKeeper 顾名思义：动物园管理员。

1.1 应用场景



它是拿来管大象(Hadoop)、蜜蜂(Hive)、小猪(Pig)的管理员， Apache Hbase 和 Apache Solr 以及阿里的 Dubbo 等项目中都采用到了 Zookeeper。

一句话：**ZooKeeper** 是一个分布式协调技术、高性能的，开源的分布式系统的协调(Coordination)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程序一致性和分布式协调技术

服务的软件。

1.2 用设计模式来理解

ZooKeeper 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变化，ZooKeeper 就将负责通知已经在 ZooKeeper 上注册的那些观察者做出相应的反应，从而实现集群中类似 Master/Slave 管理模式。

1.3 一句话总结

ZooKeeper=类似 Unix 文件系统+通知机制+Znode 节点

作用：服务注册+分布式系统的一致性通知协调

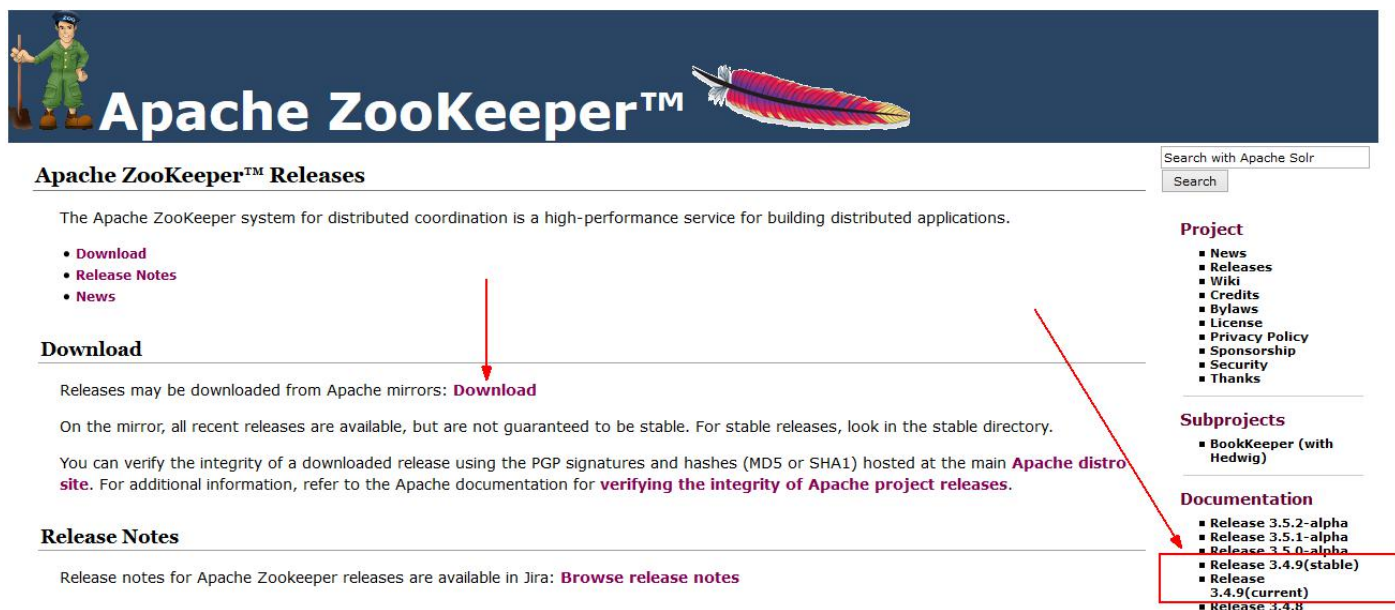
例子：

科技园综合楼总索引		
1F 101 北京森德科技股份有限公司 104 北京山水风尚科技发展有限公司 105 学富时代（北京）教育投资有限公司 易第优（北京）教育咨询股份有限公司	3F 尚硅谷咨询热线010-56253825 301 北京程程华科教育科技有限公司	518 北京捷扬科贸有限公司 521 北京东方创奇风湿骨病中医研究院 522 北京纪鑫鸿业服装贸易有限公司 5-2 科技园管委会 5-7 科技园财务室 5-19 科技园综合办公室
2F 2-1 北京博维恒信科技发展有限公司 201 北京捷力科达机电设备有限公司 204 北京汉正瑞科机电设备有限公司 205 北京达科为生物技术有限公司 206 北京爱迪博生物科技有限公司 2-3 北京同洲维普科技有限公司 2-3-A 北京华澳维普科技有限公司	4F 北京镭宝光电技术有限公司	6F 602 北京中翼天泽科技有限公司 604 北京博力扬环保科技有限公司 609 北京盛奇扬科技有限公司 610 尚硅谷办公室
	5F 501 北京点银津达科技有限公司 501A 北京欣远诚业科技有限公司 502 唐航高科（北京）国际控制技术有限公司 503 北京虎印天下科技有限公司	
北京宏福科技孵化器股份有限公司 联系电话：010-81785567		

2. 去哪下

官网地址：<https://zookeeper.apache.org/>

下载地址：<https://zookeeper.apache.org/releases.html#download>



Apache ZooKeeper™ Releases

The Apache ZooKeeper system for distributed coordination is a high-performance service for building distributed applications.

- [Download](#)
- [Release Notes](#)
- [News](#)

Download

Releases may be downloaded from Apache mirrors: [Download](#)

On the mirror, all recent releases are available, but are not guaranteed to be stable. For stable releases, look in the stable directory.

You can verify the integrity of a downloaded release using the PGP signatures and hashes (MD5 or SHA1) hosted at the main [Apache distro site](#). For additional information, refer to the Apache documentation for [verifying the integrity of Apache project releases](#).

Release Notes

Release notes for Apache Zookeeper releases are available in Jira: [Browse release notes](#)

Project

- [News](#)
- [Releases](#)
- [Wiki](#)
- [Credits](#)
- [Bylaws](#)
- [License](#)
- [Privacy Policy](#)
- [Sponsorship](#)
- [Security](#)
- [Thanks](#)

Subprojects

- [BookKeeper \(with Hedwig\)](#)

Documentation

- [Release 3.5.2-alpha](#)
- [Release 3.5.1-alpha](#)
- [Release 3.5.0-alpha](#)
- [Release 3.4.9\(stable\)](#)
- [Release 3.4.9\(current\)](#)
- [Release 3.4.8](#)

3. 怎么用

3.1 统一命名服务

统一命名服务（Name Service 如 Dubbo 服务注册中心）

Dubbo 是一个分布式服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，是阿里巴巴 SOA 服务化治理方案的核心框架，每天为 2,000+ 个服务提供 3,000,000,000+ 次访问量支持，并被广泛应用于阿里巴巴集团的各成员站点。

在 Dubbo 实现中：

服务提供者在启动的时候，向 ZK 上的指定节点 `/dubbo/${serviceName}/providers` 目录下写入自己的 URL 地址，这个操作就完成了服务的发布。

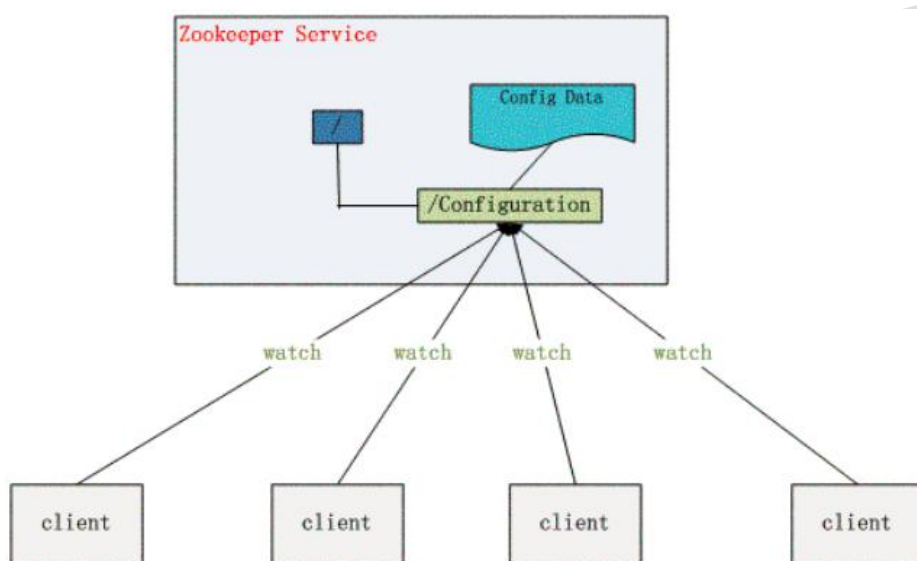
服务消费者启动的时候，订阅 `/dubbo/${serviceName}/providers` 目录下的提供者 URL 地址，并向 `/dubbo/${serviceName}/consumers` 目录下写入自己的 URL 地址。

注意，所有向 ZooKeeper 上注册的地址都是临时节点，这样就能够保证服务提供者和消费者能够自动感应资源的变化。另外，Dubbo 还有针对服务粒度的监控，方法是订阅 `/dubbo/${serviceName}` 目录下所有提供者和消费者的信息。

3.2 配置管理

配置管理（Configuration Management 如淘宝开源配置管理框架 Diamond）。

在大型的分布式系统中，为了服务海量的请求，同一个应用常常需要多个实例。如果存在配置更新的需求，常常需要逐台更新，给运维增加了很大的负担同时带来一定的风险(配置会存在不一致的窗口期，或者个别节点忘记更新)。ZooKeeper 可以用来做集中的配置管理，存储在 ZooKeeper 集群中的配置，如果发生变更会主动推送到连接配置中心的应用节点，实现一处更新处处更新的效果。



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就好。

3.3 Java 操作 API

可以类比于 Redis，ZooKeeper 也有自己的操作命令，Java 也可以通过第三方 jar 包操作 ZooKeeper。

第 2 章 安装配置

4. Linux 下安装

4.1 官网下载安装包

版本 zookeeper-3.4.9.tar.gz

4.2 拷贝解压

拷贝进入到/opt 目录下并解压

```
[root@atguigu opt]# pwd
/opt
[root@atguigu opt]# ls -l zookeeper-3.4.9.tar.gz
-rwxrw-rw- 1 root root 22724574 01-22 23:30 zookeeper-3.4.9.tar.gz
[root@atguigu opt]#
```

4.3 创建专属目录

新建专属 zookeeper 目录，mkdir /myzookeeper，随后将上一步解压的 zookeeper 内容拷贝进/myzookeeper 目录内。

```
[root@atguigu myzookeeper]# pwd
/myzookeeper
[root@atguigu myzookeeper]# ls -l
总计 4
drwxr-xr-x 10 1001 1001 4096 2016-08-23 zookeeper-3.4.9
[root@atguigu myzookeeper]#
```

4.4 拷贝配置文件

进入 conf 文件夹，拷贝 zoo_sample.cfg 改为 zoo.cfg


```
[root@atguigu zookeeper-3.4.9]# pwd
/myzookeeper/zookeeper-3.4.9
[root@atguigu zookeeper-3.4.9]# ls
bin          contrib      ivy.xml      README_packaging.txt  zookeeper-3.4.9.jar
build.xml    dist-maven   lib          README.txt           zookeeper-3.4.9.jar.asc
CHANGES.txt docs         LICENSE.txt  recipes              zookeeper-3.4.9.jar.md5
conf         ivysettings.xml NOTICE.txt  src                  zookeeper-3.4.9.jar.sha1
[root@atguigu zookeeper-3.4.9]#

[root@atguigu conf]# pwd
/myzookeeper/zookeeper-3.4.9/conf
[root@atguigu conf]# ls -l
总计 16
-rw-rw-r-- 1 1001 1001 535 2016-08-23 configuration.xml
-rw-rw-r-- 1 1001 1001 2161 2016-08-23 log4j.properties
-rw-r--r-- 1 root root 922 03-18 10:35 zoo.cfg
-rw-rw-r-- 1 1001 1001 922 2016-08-23 zoo_sample.cfg
[root@atguigu conf]#
```

拷贝zoo_sample.cfg
改名新建

4.5 配置文件解读

1、tickTime:

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:

这个配置项是用来配置 ZooKeeper 接收 Follower 客户端（这里所说的客户端不是用户链接 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,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）初始化连接是最长能忍受多少个心跳的时间间隔数。

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返回的信息, 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:

LF 同步通信时限。集群中 Leader 与 Follower 之间的最大响应时间单位。

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，

假如响应超过 $\text{syncLimit} * \text{tickTime}$ (假设 $\text{syncLimit}=5$ ，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。)，Leader 认为 Follower 死掉，从服务器列表中删除 Follower。

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:

客户端连接端口，监听客户端连接的端口。

4.6 确认启动环境

启动 Zookeeper 服务之前需要先安装好 Java 环境

4.7 开启服务和客户端连接

启动和关闭服务

/myzookeeper/zookeeper-3.4.9/bin 路径下

```
[root@atguigu bin]# ls -l *.sh
-rwxr-xr-x 1 1001 1001 1937 2016-08-23 zkCleanup.sh
-rwxr-xr-x 1 1001 1001 1534 2016-08-23 zkCli.sh
-rwxr-xr-x 1 1001 1001 2696 2016-08-23 zkEnv.sh
-rwxr-xr-x 1 1001 1001 6773 2016-08-23 zkServer.sh
[root@atguigu bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /myzookeeper/zookeeper-3.4.9/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@atguigu bin]# ps -ef|grep zookeeper
root      5250      1   8 15:16 pts/2    00:00:00 /opt/jdk1.7.0_67/bin/java -Dzookeeper.log.dir=. -Dzookeeper.root.logger=INFO,CONSOLE -cp /myzookeeper/zookeeper-3.4.9/bin/./build/classes:/myzookeeper/zookeeper-3.4.9/bin/./build/lib/*.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/slf4j-log4j12-1.6.1.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/slf4j-api-1.6.1.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/netty-3.10.5.Final.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/log4j-1.2.16.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/jline-0.9.94.jar:/myzookeeper/zookeeper-3.4.9/bin/./zookeeper-3.4.9.jar:/myzookeeper/zookeeper-3.4.9/bin/./src/java/lib/*.jar:/myzookeeper/zookeeper-3.4.9/bin/./conf/./opt/jdk1.7.0_67/lib/dt.jar:/opt/jdk1.7.0_67/lib/tools.jar -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.local.only=false org.apache.zookeeper.server.quorum.QuorumPeerMain /myzookeeper/zookeeper-3.4.9/bin/./conf/zoo.cfg
root      5273  5089   0 15:16 pts/2    00:00:00 grep zookeeper
[root@atguigu bin]# ./zkServer.sh stop
ZooKeeper JMX enabled by default
Using config: /myzookeeper/zookeeper-3.4.9/bin/./conf/zoo.cfg
Stopping zookeeper ... STOPPED
[root@atguigu bin]#
```

客户端连接: ./zkCli.sh

退出: quit

```
[root@atguigu bin]# ./zkCli.sh
Connecting to localhost:2181
2017-03-18 15:23:34.917 [myid:] - INFO [main:Environment@100] - Client environment:zookeeper.version=3.4.9-1757313, built on 08/23/2016 06:50 GMT
2017-03-18 15:23:34.923 [myid:] - INFO [main:Environment@100] - Client environment:host.name=atguigu.cloud
2017-03-18 15:23:34.923 [myid:] - INFO [main:Environment@100] - Client environment:java.version=1.7.0_67
2017-03-18 15:23:34.925 [myid:] - INFO [main:Environment@100] - Client environment:java.vendor=Oracle Corporation
2017-03-18 15:23:34.926 [myid:] - INFO [main:Environment@100] - Client environment:java.home=/opt/jdk1.7.0_67/jre
2017-03-18 15:23:34.926 [myid:] - INFO [main:Environment@100] - Client environment:java.class.path=/myzookeeper/zookeeper-3.4.9/bin/./build/classes:/myzookeeper/zookeeper-3.4.9/bin/./build/lib/*:/myzookeeper/zookeeper-3.4.9/bin/./lib/slf4j-log4j12-1.6.1.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/slf4j-api-1.6.1.jar:/myzookeeper/zookeeper-3.4.9/bin/./lib/netty-3.10.5.Final.jar:/myzookeeper/zookeeper-3.4.9/bin/./src/java/lib/*:/myzookeeper/zookeeper-3.4.9/bin/./conf:./opt/jdk1.7.0_67/lib/rt.jar:/opt/jdk1.7.0_67/lib/tools.jar
2017-03-18 15:23:34.926 [myid:] - INFO [main:Environment@100] - Client environment:java.library.path=/usr/java/packages/lib/i386:/lib:/usr/lib
2017-03-18 15:23:34.926 [myid:] - INFO [main:Environment@100] - Client environment:java.io.tmpdir=/tmp
2017-03-18 15:23:34.926 [myid:] - INFO [main:Environment@100] - Client environment:java.compiler=<NA>
2017-03-18 15:23:34.927 [myid:] - INFO [main:Environment@100] - Client environment:os.name=Linux
2017-03-18 15:23:34.927 [myid:] - INFO [main:Environment@100] - Client environment:os.arch=i386
2017-03-18 15:23:34.927 [myid:] - INFO [main:Environment@100] - Client environment:os.version=2.6.18-194.el5
2017-03-18 15:23:34.927 [myid:] - INFO [main:Environment@100] - Client environment:user.name=root
2017-03-18 15:23:34.927 [myid:] - INFO [main:Environment@100] - Client environment:user.home=/root
2017-03-18 15:23:34.927 [myid:] - INFO [main:Environment@100] - Client environment:user.dir=/myzookeeper/zookeeper-3.4.9/bin
2017-03-18 15:23:34.929 [myid:] - INFO [main:ZooKeeper@438] - Initiating client connection, connectString=localhost:2181 sessionTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@130f672
Welcome to ZooKeeper!
2017-03-18 15:23:34.970 [myid:] - INFO [main-SendThread(atguigu.cloud:2181):ClientCnxnSndThread@1032] - Opening socket connection to server atguigu.cloud/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)
JLine support is enabled
2017-03-18 15:23:35.013 [myid:] - INFO [main-SendThread(atguigu.cloud:2181):ClientCnxnSndThread@876] - Socket connection established to atguigu.cloud/127.0.0.1:2181, initiating session
[zookeeper:2181(CONNECTING) 0] 2017-03-18 15:23:35.027 [myid:] - INFO [main-SendThread(atguigu.cloud:2181):ClientCnxnSndThread@1299] - Session establishment complete on server atguigu.cloud/127.0.0.1:2181, sessionId = 0x15ae0494a3e0001, negotiated timeout = 30000
```

4.8 验证服务是否启动

(1) 四字命令

在 Zookeeper 服务器成功启动的前提下, 在 Linux 侧的 shell 命令端口执行下面的 ruok 四字命令, 如果能够显示 imok, 表示 zk 服务器端成功启动。

使用四字命令:

```
echo ruok | nc 127.0.0.1 2181
```

```
[root@atguigu Packages]# echo ruok | nc 127.0.0.1 2181
imok[root@atguigu Packages]#
```

(2) CentOS6.8 下面 nc 命令的安装

1 路径:

/media/CentOS_6.8_Final/Packages

2 命令:


```
rpm -ivh nc-1.84-24.el6.x86_64.rpm
```

```
[root@atguigu Packages]# pwd
/media/CentOS_6.8_Final/Packages
[root@atguigu Packages]#
[root@atguigu Packages]#
[root@atguigu Packages]#
[root@atguigu Packages]# rpm -ivh nc-1.84-24.el6.x86_64.rpm
warning: nc-1.84-24.el6.x86_64.rpm: Header V3 RSA/SHA1 Signature, key ID c105b9de: NOKEY
Preparing...
##### [100%]
package nc-1.84-24.el6.x86_64 is already installed
[root@atguigu Packages]#
```

(3) CentOS7 下面 nc 命令的安装

- 1 路径:
`/run/media/root/CentOS 7 x86_64/Packages`
- 2 命令:
`rpm -ivh nmap-ncat-6.40-7.el7.x86_64`

一般默认安装

```
[root@cocoon Packages]# rpm -qa|grep nmap-ncat
nmap-ncat-6.40-7.el7.x86_64
[root@cocoon Packages]#
```

5. 永远的 HelloWorld

5.1 查看获取数据信息

查看和获得 zookeeper 服务器上的数据存储信息

```
[zk: localhost:2181(CONNECTED) 0] ls /  
[zookeeper]  
[zk: localhost:2181(CONNECTED) 1] ls /zookeeper  
[quota]  
[zk: localhost:2181(CONNECTED) 2] ls /zookeeper/quota  
[]  
[zk: localhost:2181(CONNECTED) 3] get /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  
[zk: localhost:2181(CONNECTED) 4]
```

5.2 文件系统

ZooKeeper 维护一个类似文件系统的数据结构。

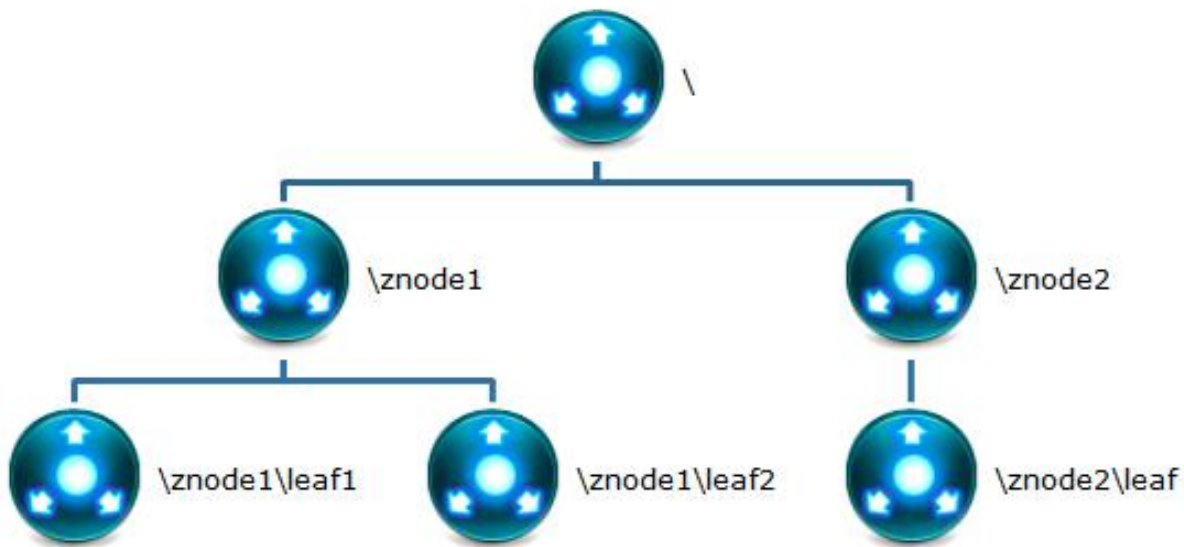
所使用的数据模型风格很像文件系统的目录树结构，简单来说，有点类似 windows 中注册表的结构。

有名称，有树节点，有 Key(键)/Value(值)对的关系，可以看做一个树形结构的数据库，分布在不同的机器上做名称管理。

5.3 初识 znode 节点

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

很显然 ZooKeeper 集群自身维护了一套数据结构。这个存储结构是一个树形结构，其上的每一个节点，我们称之为“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



第 3 章 数据模型/znode 节点深入

1. Znode 的数据模型

1.1 是什么

Znode 维护了一个 stat 结构，这个 stat 包含数据变化的版本号、访问控制列表变化、还有时间戳。版本号和时戳一起，可让 ZooKeeper 验证缓存和协调更新。每次 znode 的数据发生了变化，版本号就增加。

例如，无论何时客户端检索数据，它也一起检索数据的版本号。并且当客户端执行更新或删除时，客户端必须提供他正在改变的 znode 的版本号。如果它提供的版本号和真实的数据版本号不一致，更新将会失败。

1.2 Stat 结构体

ZooKeeper Stat Structure

The Stat structure for each znode in ZooKeeper is made up of the following fields:

- **czxid**
The zxid of the change that caused this znode to be created.
- **mzxid**
The zxid of the change that last modified(修改) this znode.
- **pzxid**
The zxid of the change that last modified children of this znode.
- **ctime**
The time in milliseconds(毫秒) from epoch(世) when this znode was created.
- **mtime**
The time in milliseconds from epoch when this znode was last modified(改进的).
- **version**
The number of changes to the data of this znode.
- **cversion**
The number of changes to the children of this znode.
- **aversion**
The number of changes to the ACL of this znode.
- **ephemeralOwner**
The session(会议) id of the owner of this znode if the znode is an ephemeral(短暂的) node. If it is not an ephemeral node, it will be zero.
- **dataLength**
The length of the data field of this znode.
- **numChildren**
The number of children of this znode.

czxid- 引起这个 znode 创建的 zxid，创建节点的事物的 zxid（ZooKeeper Transaction Id）

ctime - znode 被创建的毫秒数(从 1970 年开始)

mzxid - znode 最后更新的 zxid

mtime - znode 最后修改的毫秒数(从 1970 年开始)

pZxid-znode 最后更新的子节点 zxid

cversion - znode 子节点变化号，znode 子节点修改次数

dataversion - znode 数据变化号

aclVersion - znode 访问控制列表的变化号

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。

dataLength- znode 的数据长度

numChildren - znode 子节点数量

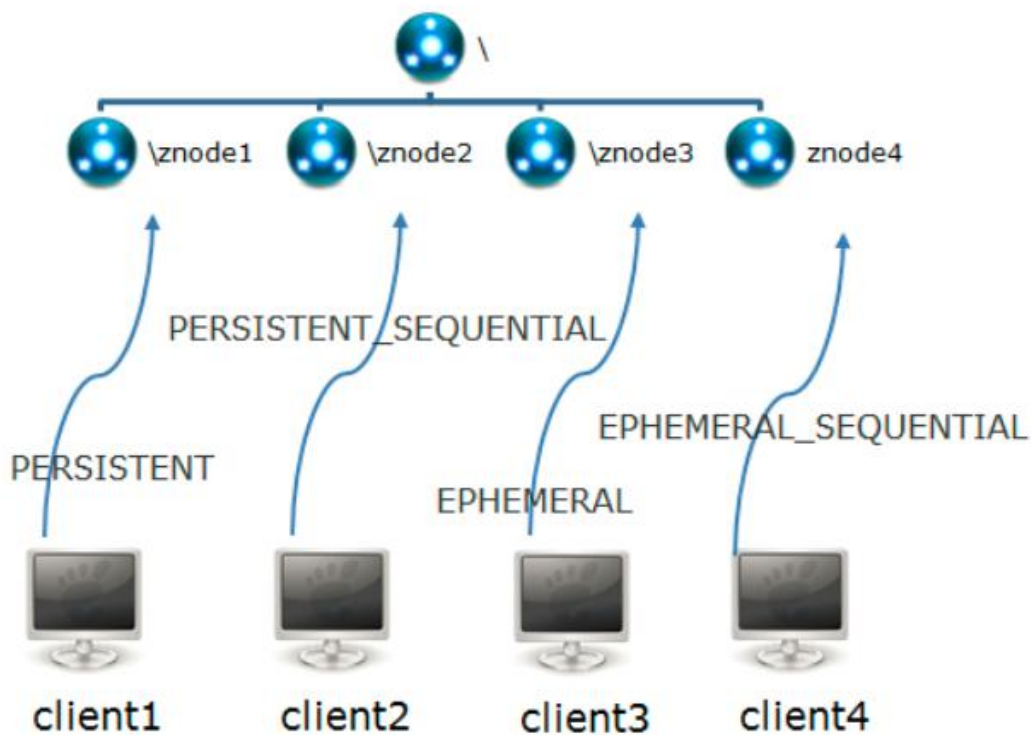
1.3 总结

ZooKeeper 内部维护了一套类似 UNIX 的树形数据结构：由 Znode 构成的集合，

Znode 的集合又是一个树形结构，每一个 Znode 又有很多属性进行描述。

Znode = path + data + Stat

2. Znode 中的存在类型



Znode 是由客户端创建的，它和创建它的客户端的内在联系，决定了它的存在性：

PERSISTENT-持久化节点：

创建这个节点的客户端在与 zookeeper 服务的连接断开后，这个节点也不会被删除（除非您使用 API 强制删除）。

PERSISTENT_SEQUENTIAL-持久化顺序编号节点：

当客户端请求创建这个节点 A 后，zookeeper 会根据 parent-znode 的 zxid 状态，为这个 A 节点编写一个全目录唯一的编号（这个编号只会一直增长）。当客户端与 zookeeper 服务的连接断开后，这个节点也不会被删除。

EPHEMERAL-临时目录节点：

创建这个节点的客户端在与 zookeeper 服务的连接断开后，这个节点（还有涉及到的子节点）就会被删除。

EPHEMERAL_SEQUENTIAL-临时顺序编号目录节点：

当客户端请求创建这个节点 A 后，zookeeper 会根据 parent-znode 的 zxid 状态，为这个 A 节点编写一个全目录唯一的编号（这个编号只会一直增长）。当创建这个节点的客户端与 zookeeper 服务的连接断开后，这个节点被删除。

另外，无论是 EPHEMERAL 还是 EPHEMERAL_SEQUENTIAL 节点类型，在 zookeeper 的 client 异常终止后，节点也会被删除

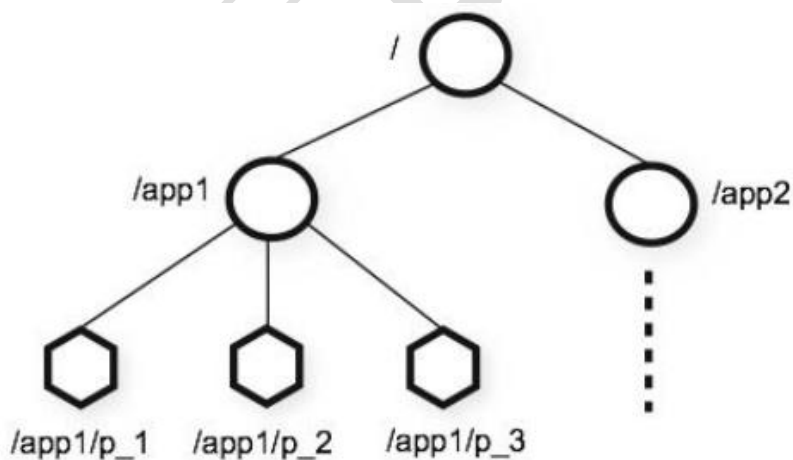
第 4 章基础命令和 Java 客户端操作

1. 常用操作命令

一句话：和 redis 的 KV 键值对类似，只不过 key 变成了一个路径节点值，v 就是 data

Zookeeper 表现为一个分层的文件系统目录树结构

不同于文件系统之处在于：zk 节点可以有自己的数据，而 unix 文件系统中的目录节点只有子节点



zookeeper 数据模型结构图

一个节点对应一个应用/服务，节点存储的数据就是应用需要的配置信息。

命令	功能描述
help	不是命令，但会列出帮助信息
ls	使用 ls 命令来查看当前 znode 中所包含的内容
ls2	查看当前节点数据并能看到更新次数等数据
stat	查看节点状态
set	设置节点的具体值, set 节点 值
get	获得节点的值, get 节点
create	创建持久节点: create 节点 值 创建带序号持久节点: create -s 节点 值 创建临时节点: create -e 节点 值 创建带序号临时节点: create -s -e 节点 值
delete	删除不带子节点的节点
rmr	递归删除节点

2. 四字命令

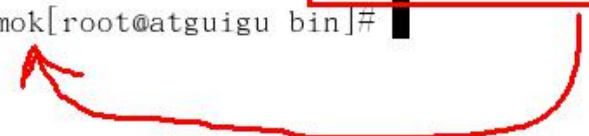
2.1 是什么

zookeeper 支持某些特定的四字命令，他们大多是用来查询 ZK 服务的当前状态及相关信息的，

通过 telnet 或 nc 向 zookeeper 提交相应命令，如：echo ruok | nc 127.0.0.1 2181

运行公式：echo 四字命令 | nc 主机 IP zookeeper 端口

```
[root@atguigu bin]# echo ruok | nc localhost 2181
imok[root@atguigu bin]#
```



2.2 常用命令

命令	功能描述
ruok	测试服务是否处于正确状态。如果确实如此，那么服务返回“imok”，否则不做任何相应
stat	输出关于性能和连接的客户端的列表
conf	输出相关服务配置的详细信息
cons	列出所有连接到服务器的客户端的完全的连接 / 会话的详细信息。包括“接受 / 发送”的包数量、会话 id、操作延迟、最后的操作执行等等信息
dump	列出未经处理的会话和临时节点
envi	输出关于服务环境的详细信息（区别于 conf 命令）
reqs	列出未经处理的请求
wchs	列出服务器 watch 的详细信息
wchc	通过 session 列出服务器 watch 的详细信息，它的输出是一个与 watch 相关的会话的列表
wchp	通过路径列出服务器 watch 的详细信息。它输出一个与 session 相关的路径

2.3 总结

Category	Command	Description
Server status	ruok	Prints imok if the server is running and not in an error state.
	conf	Prints the server configuration (from zoo.cfg).
	envi	Prints the server environment, including ZooKeeper version, Java version, and other system properties.
	svr	Prints server statistics, including latency statistics, the number of znodes, and the server mode (standalone, leader, or follower).
	stat	Prints server statistics and connected clients.
	srst	Resets server statistics.
	isro	Shows whether the server is in read-only (ro) mode (due to a network partition) or read/write mode (rw).
Client connections	dump	Lists all the sessions and ephemeral znodes for the ensemble. You must connect to the leader (see svr) for this command.
	cons	Lists connection statistics for all the server' s clients.
	crst	Resets connection statistics.
Watches	wchs	Lists summary information for the server' s watches.
	wchc	Lists all the server' s watches by connection. Caution: may impact server performance for a large number of watches.
	wchp	Lists all the server' s watches by znode path. Caution: may impact server performance for a large number of watches.
Monitoring	mntr	Lists server statistics in Java properties format, suitable as a source for monitoring systems such as Ganglia and Nagios.

3. Java 客户端操作

Maven 工程和配置 POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.atguigu.mall</groupId>
  <artifactId>zk0919</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>zk0919</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!-- https://mvnrepository.com/artifact/com.101tec/zkclient -->
    <dependency>
      <groupId>com.101tec</groupId>
      <artifactId>zkclient</artifactId>
      <version>0.10</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.zookeeper/zookeeper -->
```



```
<dependency>

    <groupId>org.apache.zookeeper</groupId>

    <artifactId>zookeeper</artifactId>

    <version>3.4.9</version>

</dependency>

<dependency>

    <groupId>log4j</groupId>

    <artifactId>log4j</artifactId>

    <version>1.2.17</version>

</dependency>

<dependency>

    <groupId>junit</groupId>

    <artifactId>junit</artifactId>

    <version>4.9</version>

    <scope>test</scope>

</dependency>

</dependencies>

</project>
```

程序代码

```
package com.atguigu.zk3;

import java.io.IOException;

import org.apache.log4j.Logger;

import org.apache.zookeeper.CreateMode;

import org.apache.zookeeper.KeeperException;
```

```
import org.apache.zookeeper.WatchedEvent;

import org.apache.zookeeper.Watcher;

import org.apache.zookeeper.ZooDefs.Ids;

import org.apache.zookeeper.ZooKeeper;

import org.apache.zookeeper.data.Stat;


public class HelloZK
{
    /**
     * Logger for this class
     */
    private static final Logger logger = Logger.getLogger(HelloZK.class);

    private static final String CONNECTSTRING = "192.168.67.167:2181";
    private static final String PATH = "/atguigu";
    private static final int    SESSION_TIMEOUT = 50*1000;


    public ZooKeeper startZK() throws IOException
    {
        return new ZooKeeper(CONNECTSTRING, SESSION_TIMEOUT, new Watcher() {

            @Override

            public void process(WatchedEvent event)

            {

            }

        });
    }
}
```

```
}

public void stopZK(ZooKeeper zk) throws InterruptedException
{
    if(zk != null)
    {
        zk.close();
    }
}

public void createZNode(ZooKeeper zk,String path,String nodeValue) throws KeeperException, InterruptedException
{
    zk.create(path,nodeValue.getBytes(),Ids.OPEN_ACL_UNSAFE,CreateMode.PERSISTENT);
}

public String getZNode(ZooKeeper zk,String path) throws KeeperException, InterruptedException
{
    byte[] byteArray = zk.getData(path, false, new Stat());
    return new String(byteArray);
}

public static void main(String[] args) throws IOException, KeeperException, InterruptedException
{
    HelloZK hello = new HelloZK();

    ZooKeeper zk = hello.startZK();
}
```

```
Stat stat = zk.exists(PATH, false);

if(stat == null)
{
    hello.createZNode(zk, PATH, "zk1014");

    String result = hello.getZNode(zk, PATH);

    System.out.println("*****result: "+result);
}else{
    System.out.println("*****znode has already ok*****");
}

hello.stopZK(zk);
}
}
```

第 5 章 通知机制

1. 通知机制简介

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。

1.1 是什么

ZooKeeper 支持 watch(观察)的概念。客户端可以在每个 znode 结点上设置一个观察。如果被观察服务端的 znode

结点有变更，那么 watch 就会被触发，这个 watch 所属的客户端将接收到一个通知包被告知结点已经发生变化，把相应的事件通知给设置过 Watcher 的 Client 端。

Zookeeper 里的所有读取操作：`getData()`、`getChildren()`和 `exists()`都有设置 watch 的选项。

一句话：异步回调的触发机制

2. Watch 事件

ZooKeeper Watches

All of the **read operations** in ZooKeeper - `getData()`, `getChildren()`, and `exists()` - have the option of setting a watch as a side effect. Here is ZooKeeper's definition of a watch: a watch event is one-time trigger, sent to the client that set the watch, which occurs when the data for which the watch was set changes. There are three key points to consider in this definition of a watch:

- **One-time trigger**
One watch event will be sent to the client when the data has changed. For example, if a client does a `getData("/znode1", true)` and later the data for `/znode1` is changed or deleted, the client will get a watch event for `/znode1`. If `/znode1` changes again, no watch event will be sent unless the client has done another read that sets a new watch.
- **Sent to the client**
This implies that an event is on the way to the client, but may not reach the client before the successful return code to the change operation reaches the client that initiated the change. Watches are sent asynchronously to watchers. ZooKeeper provides an ordering guarantee: a client will never see a change for which it has set a watch until it first sees the watch event. Network delays or other factors may cause different clients to see watches and return codes from updates at different times. The key point is that everything seen by the different clients will have a consistent order.
- **The data for which the watch was set**
This refers to the different ways a node can change. It helps to think of ZooKeeper as maintaining two lists of watches: data watches and child watches. `getData()` and `exists()` set data watches. `getChildren()` sets child watches. Alternatively, it may help to think of watches being set according to the kind of data returned. `getData()` and `exists()` return information about the data of the node, whereas `getChildren()` returns a list of children. Thus, `setData()` will trigger data watches for the znode being set (assuming the set is successful). A successful `create()` will trigger a data watch for the znode being created and a child watch for the parent znode. A successful `delete()` will trigger both a data watch and a child watch (since there can be no more children) for a znode being deleted as well as a child watch for the parent znode.

Watches are maintained locally at the ZooKeeper server to which the client is connected. This allows watches to be lightweight to set, maintain, and dispatch. When a client connects to a new server, the watch will be triggered for any session events. Watches will not be received while disconnected from a server. When a client reconnects, any previously registered watches will be reregistered and triggered if needed. In general this all occurs transparently. There is one case where a watch may be missed: a watch for the existence of a znode not yet created will be missed if the znode is created and deleted while disconnected.

2.1 一次触发

当数据有了变化时 zkserver 向客户端发送一个 watch,它是一次性的动作，即触发一次就不再有效，类似一次性纸杯。

只监控一次。如果想继续 Watch 的话，需要客户端重新设置 Watcher。因此如果你得到一个 watch 事件且想在将来的变化得到通知，必须新设置另一个 watch。

2.2 发往客户端

Watches 是异步发往客户端的，Zookeeper 提供一个顺序保证：在看到 watch 事件之前绝不会看到变化，这样不同客户端看到的是一致性的顺序。

在（导致观察事件被触发的）修改操作的成功返回码到达客户端之前，事件可能在去往客户端的路上，但是可能不会到达客户端。观察事件是异步地发送给观察者（客户端）的。ZooKeeper 会保证次序：在收到观察事件之前，客户端不会看到已经为之设置观察的节点的改动。网络延迟或者其他因素可能会让不同的客户端在不同的时间收到观察事件和更新操作的返回码。这里的要点是：不同客户端看到的事情都有一致的次序。

2.3 为数据设置 watch

节点有不同的改动方式。可以认为 ZooKeeper 维护两个观察列表：数据观察和子节点观察。`getData()`和 `exists()` 设置数据观察。`getChildren()`设置子节点观察。此外，还可以认为不同的返回数据有不同的观察。`getData()`和 `exists()` 返回节点的数据，而 `getChildren()`返回子节点列表。所以，`setData()`将为 `znode` 触发数据观察。成功的 `create()`将为新创建的节点触发数据观察，为其父节点触发子节点观察。成功的 `delete()`将会为被删除的节点触发数据观察以及子节点观察（因为节点不能再有子节点了），为其父节点触发子节点观察。

观察维护在客户端连接到的 ZooKeeper 服务器中。这让观察的设置、维护和分发是轻量级的。客户端连接到新的服务器时，所有会话事件将被触发。同服务器断开连接期间不会收到观察。客户端重新连接时，如果需要，先前已经注册的观察将被重新注册和触发。通常这都是透明的。有一种情况下观察事件将丢失：对还没有创建的节点设置存在观察，而在断开连接期间创建节点，然后删除。

2.4 时序性和一致性

Watches 是在 client 连接到 Zookeeper 服务端的本地维护，这可让 watches 成为轻量的，可维护的和派发的。当一个 client 连接到新 server，watch 将会触发任何 session 事件，断开连接后不能接收到。当客户端重连，先前注册的 watches 将会被重新注册并触发。

关于 watches，Zookeeper 维护这些保证：

- （1）Watches 和其他事件、watches 和异步恢复都是有序的。Zookeeper 客户端保证每件事都是有序派发
- （2）客户端在看到新数据之前先看到 watch 事件
- （3）对应更新顺序的 watches 事件顺序由 Zookeeper 服务所见

3. 程序代码

3.1 一次性

```
package com.atguigu.zk3;

import java.io.IOException;

import org.apache.log4j.Logger;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.ZooDefs.Ids;
import org.apache.zookeeper.data.Stat;

public class WatchOne
{
    /**
     * Logger for this class
     */
    private static final Logger logger = Logger.getLogger(HelloZK.class);

    //定义常量
    private static final String CONNECTSTRING = "192.168.67.167:2181";
    private static final String PATH = "/atguigu";
    private static final int    SESSION_TIMEOUT = 50*1000;
```

```
//定义实例变量

private ZooKeeper zk = null;


//以下为业务方法

public ZooKeeper startZK() throws IOException
{
    return new ZooKeeper(CONNECTSTRING, SESSION_TIMEOUT, new Watcher() {

        @Override
        public void process(WatchedEvent event)
        {
        }

    });
}

public void stopZK() throws InterruptedException
{
    if(zk != null)
    {
        zk.close();
    }
}

public void createZNode(String path,String nodeValue) throws KeeperException, InterruptedException
{
    zk.create(path,nodeValue.getBytes(),Ids.OPEN_ACL_UNSAFE,CreateMode.PERSISTENT);
}
```

```
public String getZNode(String path) throws KeeperException, InterruptedException
{
    byte[] byteArray = zk.getData(path,new Watcher() {
        @Override
        public void process(WatchedEvent event)
        {
            try
            {
                triggerValue(path);
            }catch (KeeperException | InterruptedException e) {
                e.printStackTrace();
            }
        }
    }, new Stat());
    return new String(byteArray);
}

public String triggerValue(String path) throws KeeperException, InterruptedException
{
    byte[] byteArray = zk.getData(path,false, new Stat());
    String retValue = new String(byteArray);
    System.out.println("*****triggerValue: "+retValue);
    return retValue;
}

public static void main(String[] args) throws IOException, KeeperException, InterruptedException
```

```
{

    WatchOne watchOne = new WatchOne();

    watchOne.setZk(watchOne.startZK());

    if(watchOne.getZk().exists(PATH, false) == null)
    {
        watchOne.createZNode(PATH,"BBB");

        System.out.println("*****>: "+watchOne.getZNode(PATH));

        Thread.sleep(Long.MAX_VALUE);
    }else{
        System.out.println("i have znode");
    }
}

//setter---getter
public ZooKeeper getZk()
{
    return zk;
}

public void setZk(ZooKeeper zk)
{
    this.zk = zk;
}
```



```
}
```

3.2 多次

```
package com.atguigu.zk3;

import java.io.IOException;

import org.apache.log4j.Logger;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.WatchedEvent;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooKeeper;
import org.apache.zookeeper.ZooDefs.Ids;
import org.apache.zookeeper.data.Stat;

public class WatchMoreTest
{
    /**
     * Logger for this class
     */
    private static final Logger logger = Logger.getLogger(WatchMoreTest.class);

    //定义常量
    private static final String CONNECTSTRING = "192.168.67.167:2181";
    private static final String PATH = "/atguigu";
```

```
private static final int    SESSION_TIMEOUT = 50*1000;

//定义实例变量

private ZooKeeper zk = null;

private String lastValue = "";


//以下为业务方法

public ZooKeeper startZK() throws IOException
{
    return new ZooKeeper(CONNECTSTRING, SESSION_TIMEOUT, new Watcher() {

        @Override

        public void process(WatchedEvent event)

        {}

    });
}

public void stopZK() throws InterruptedException
{
    if(zk != null)
    {
        zk.close();
    }
}

public void createZNode(String path,String nodeValue) throws KeeperException, InterruptedException
{
    zk.create(path,nodeValue.getBytes(),Ids.OPEN_ACL_UNSAFE,CreateMode.PERSISTENT);
}
```

```
public String getZNode(String path) throws KeeperException, InterruptedException
{
    byte[] byteArray = zk.getData(path, new Watcher() {
        @Override
        public void process(WatchedEvent event)
        {
            try
            {
                triggerValue(path);
            } catch (KeeperException | InterruptedException e) {
                e.printStackTrace();
            }
        }
    }, new Stat());

    return new String(byteArray);
}

public boolean triggerValue(String path) throws KeeperException, InterruptedException
{
    byte[] byteArray = zk.getData(path, new Watcher() {
        @Override
        public void process(WatchedEvent event)
        {
            try
            {
```

```
        triggerValue(path);
    }catch (KeeperException | InterruptedException e) {
        e.printStackTrace();
    }
}

}, new Stat());

String newValue = new String(byteArray);

if(lastValue.equals(newValue))
{
    System.out.println("there is no change~~~~~");
    return false;
}else{
    System.out.println("lastValue: "+lastValue+"\t"+"newValue: "+newValue);
    this.lastValue = newValue;
    return true;
}
}

public static void main(String[] args) throws IOException, KeeperException, InterruptedException
{
    WatchMoreTest watch = new WatchMoreTest();

    watch.setZk(watch.startZK());
}
```

```
if(watch.getZk().exists(PATH, false) == null)
{
    String initValue = "0000";
    watch.setLastValue(initValue);
    watch.createZNode(PATH,initValue);
    System.out.println("*****>: "+watch.getZNode(PATH));
    Thread.sleep(Long.MAX_VALUE);
}else{
    System.out.println("i have znode");
}
}

//setter---getter
public ZooKeeper getZk()
{
    return zk;
}

public void setZk(ZooKeeper zk)
{
    this.zk = zk;
}

public String getLastValue()
{

```

```
        return lastValue;
    }

    public void setLastValue(String lastValue)
    {
        this.lastValue = lastValue;
    }
}
```

第 6 章 集群

1. 伪分布式单机配置

说明：

服务器名称与地址：集群信息（服务器编号，服务器地址，LF 通信端口，选举端口）

这个配置项的书写格式比较特殊，规则如下：**server.N=YYY:A:B** 其中，

N 表示服务器编号，

YYY 表示服务器的 IP 地址，

A 为 LF 通信端口，表示该服务器与集群中的 **leader** 交换的信息的端口。

B 为选举端口，表示选举新 **leader** 时服务器间相互通信的端口（当 **leader** 挂掉时，其余服务器会相互通信，选择出新的 **leader**）

一般来说，集群中每个服务器的 A 端口都是一样，每个服务器的 B 端口也是一样。

下面是一个集群的例子：

```
server.0=233.34.9.144:2008:6008
```

```
server.1=233.34.9.145:2008:6008
```

```
server.2=233.34.9.146:2008:6008
```

```
server.3=233.34.9.147:2008:6008
```

但是当所采用的为伪集群时，IP 地址都一样，只能是 A 端口和 B 端口不一样。

下面是一个伪集群的例子：

```
server.0=127.0.0.1:2008:6008
```

```
server.1=127.0.0.1:2007:6007
```

```
server.2=127.0.0.1:2006:6006
```

```
server.3=127.0.0.1:2005:6005
```

initLimit 是 Zookeeper 用它来限定集群中的 Zookeeper 服务器连接到 Leader 的时限。

syncLimit 限制了 follower 服务器与 leader 服务器之间请求和应答之间的时限。

2. 配置步骤

2.1 复制 3 个 ZooKeeper

zookeeper-3.4.9.tar.gz 解压后拷贝到/myzookeeper 目录下并重新名为 zk01，再复制 zk01 形成 zk02、zk03，共计 3 份

2.2 新增目录

进入 zk01/02/03 分别新建文件夹，mydata、mylog

```
[root@atguigu mydata]# pwd
/myzookeeper/zk01/mydata
[root@atguigu mydata]#
[root@atguigu mydata]#
[root@atguigu mydata]#
[root@atguigu mydata]# cd /myzookeeper/zk01/mylog/
[root@atguigu mylog]# pwd
/myzookeeper/zk01/mylog
[root@atguigu mylog]#
[root@atguigu mylog]#
```

2.3 新建配置文件

分别进入 zk01-zk03 各自的 conf 文件夹新建 zoo.cfg

```
[root@atguigu conf]# pwd
/myzookeeper/zk01/conf
[root@atguigu conf]# ls -l
总计 16
-rw-r--r-- 1 root root 535 03-23 23:21 configuration.xml
-rw-r--r-- 1 root root 2161 03-23 23:21 log4j.properties
-rw-r--r-- 1 root root 1057 03-23 23:42 zoo.cfg
-rw-r--r-- 1 root root 922 03-23 23:21 zoo_sample.cfg
[root@atguigu conf]#
```

拷贝 zoo_sample.cfg ----> zoo.cfg

2.4 编辑配置文件

设置自己的数据和 log 路径

```
dataDir=/myzookeeper/zk01/mydata
dataLogDir=/myzookeeper/zk01/mylog
```

修改各自的 clientPort

在最后面添加 server 的列表

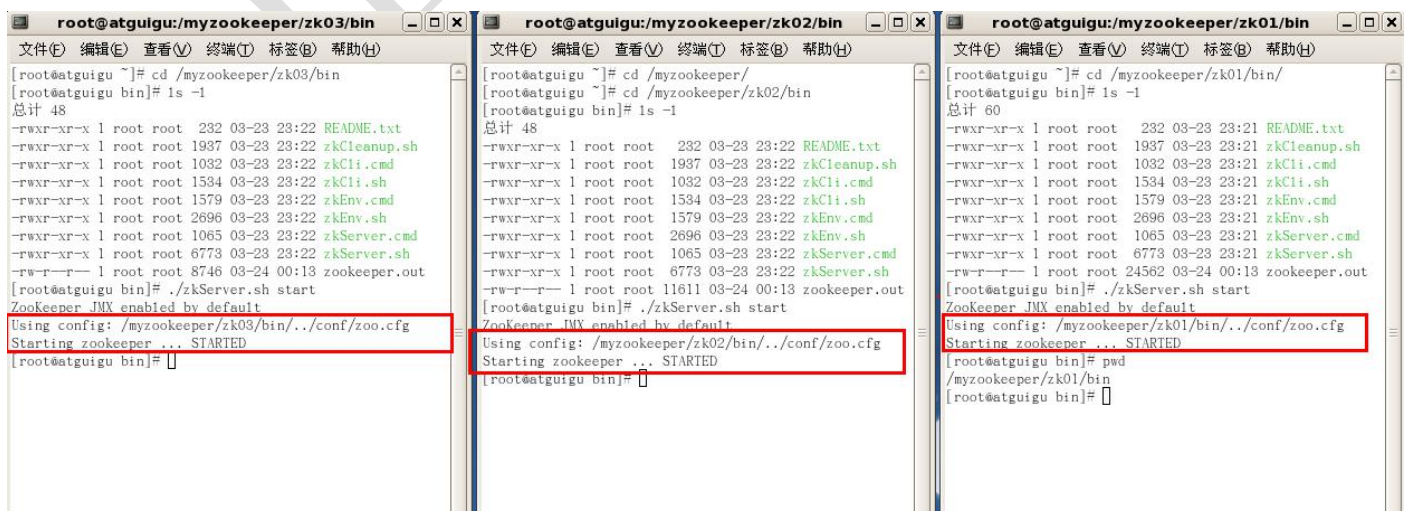
2.5 新增服务编号

在各自 mydata 下面创建 myid 的文件，在里面写入 server 的数字

```
[root@atguigu mydata]# pwd
/myzookeeper/zk01/mydata
[root@atguigu mydata]# ls -l myid
-rw-r--r-- 1 root root 2 03-23 23:58 myid
[root@atguigu mydata]# cat myid
1
[root@atguigu mydata]# cd /myzookeeper/zk02/mydata/
[root@atguigu mydata]# ls -l myid
-rw-r--r-- 1 root root 2 03-23 23:59 myid
[root@atguigu mydata]# cat myid
2
You have new mail in /var/spool/mail/root
[root@atguigu mydata]# cd /myzookeeper/zk03/mydata/
[root@atguigu mydata]# ls -l myid
-rw-r--r-- 1 root root 2 03-23 23:57 myid
[root@atguigu mydata]# cat myid
3
[root@atguigu mydata]#
```

2.6 启动服务

分别启动三个服务器



```
root@atguigu: /myzookeeper/zk03/bin
[root@atguigu ~]# cd /myzookeeper/zk03/bin
[root@atguigu bin]# ls -l
总计 48
-rwxr-xr-x 1 root root 232 03-23 23:22 README.txt
-rwxr-xr-x 1 root root 1937 03-23 23:22 zkCleanup.sh
-rwxr-xr-x 1 root root 1032 03-23 23:22 zkCli.cmd
-rwxr-xr-x 1 root root 1534 03-23 23:22 zkCli.sh
-rwxr-xr-x 1 root root 1579 03-23 23:22 zkEnv.cmd
-rwxr-xr-x 1 root root 2696 03-23 23:22 zkEnv.sh
-rwxr-xr-x 1 root root 1065 03-23 23:22 zkServer.cmd
-rwxr-xr-x 1 root root 6773 03-23 23:22 zkServer.sh
-rw-r--r-- 1 root root 8746 03-24 00:13 zookeeper.out
[root@atguigu bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /myzookeeper/zk03/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@atguigu bin]#

root@atguigu: /myzookeeper/zk02/bin
[root@atguigu ~]# cd /myzookeeper/
[root@atguigu ~]# cd /myzookeeper/zk02/bin
[root@atguigu bin]# ls -l
总计 48
-rwxr-xr-x 1 root root 232 03-23 23:22 README.txt
-rwxr-xr-x 1 root root 1937 03-23 23:22 zkCleanup.sh
-rwxr-xr-x 1 root root 1032 03-23 23:22 zkCli.cmd
-rwxr-xr-x 1 root root 1534 03-23 23:22 zkCli.sh
-rwxr-xr-x 1 root root 1579 03-23 23:22 zkEnv.cmd
-rwxr-xr-x 1 root root 2696 03-23 23:22 zkEnv.sh
-rwxr-xr-x 1 root root 1065 03-23 23:22 zkServer.cmd
-rwxr-xr-x 1 root root 6773 03-23 23:22 zkServer.sh
-rw-r--r-- 1 root root 11611 03-24 00:13 zookeeper.out
[root@atguigu bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /myzookeeper/zk02/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@atguigu bin]#

root@atguigu: /myzookeeper/zk01/bin
[root@atguigu ~]# cd /myzookeeper/zk01/bin/
[root@atguigu bin]# ls -l
总计 60
-rwxr-xr-x 1 root root 232 03-23 23:21 README.txt
-rwxr-xr-x 1 root root 1937 03-23 23:21 zkCleanup.sh
-rwxr-xr-x 1 root root 1032 03-23 23:21 zkCli.cmd
-rwxr-xr-x 1 root root 1534 03-23 23:21 zkCli.sh
-rwxr-xr-x 1 root root 1579 03-23 23:21 zkEnv.cmd
-rwxr-xr-x 1 root root 2696 03-23 23:21 zkEnv.sh
-rwxr-xr-x 1 root root 1065 03-23 23:21 zkServer.cmd
-rwxr-xr-x 1 root root 6773 03-23 23:21 zkServer.sh
-rw-r--r-- 1 root root 24562 03-24 00:13 zookeeper.out
[root@atguigu bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /myzookeeper/zk01/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@atguigu bin]# pwd
/myzookeeper/zk01/bin
[root@atguigu bin]#
```

2.7 访问集群环境

zkCli 连接 server，带参数指定-server

```
[root@atguigu bin]# pwd
/myzookeeper/zk03/bin
[root@atguigu bin]# ./zkCli.sh -server 127.0.0.1:2193
Connecting to 127.0.0.1:2193
```

2191/2192/2193 任意用客户端链接一台，会发现只需要有一个改变了，整个集群的内容自动一致性同步。