



# ZooKeeper的操作与编程

# ZooKeeper简介

从前面的原理章节中，我们知道分布式程序需要一定的协同功能，以便能够在多个运行的进程之间建立联系

- 其中一个重要的协同功能就是进行分布式锁，这样就可以在多个应用程序进行共享资源访问的时候起到保护作用

**ZooKeeper**就是一个为分布式应用程序进行协调的服务，这样的话，每一个分布式的应用程序如果需要进行协调的话就可以直接使用**ZooKeeper**所提供的服务

**ZooKeeper**提供了一系列分布式系统的基本服务或者可以基于**ZooKeeper**完成分布式系统的基本服务：同步、配置管理、分组和命名

**ZooKeeper**提供了一个易于编程的环境，实现了一个简化的文件系统，提供类似的目录树结构

**ZooKeeper**使用**Java**编写，支持了**Java**以及**C**语言绑定 分布式的协调服务

**coordination**非常容易出错，出错之后也很难恢复，例如死锁状态，或者出现资源竞争状态，通过**ZooKeeper**可以以良好的编程接口将程序员从自己构造协调服务的负担中解放出来

# ZooKeeper的特性

## 结构简单

- ZooKeeper提供了文件系统的树状结构

## 数据备份

- 数据一致性，快照+WAL (write ahead log)

## 有序性

- 有序的事务编号 `zxid`

## 高效性

- 所有的server都提供读服务

# 安装过程概述

下载并将zookeeper解压缩到任意一个目录 修

改conf目录下的zoo\_sample.cfg为zoo.cfg

修改配置参数

启动ZooKeeper

# ZooKeeper程序的启动

在完成配置之后

1,将ZooKeeper拷贝到多个需要执行的对应节点

2,注意需要在每个运行的节点中的数据存储目录中创建一个myid文件，在其中写入一个id号

```
hadoop@master:~/zkdata$ pwd
/home/hadoop/zkdata
hadoop@master:~/zkdata$ ls
myid  version-2
hadoop@master:~/zkdata$ cat myid
1
hadoop@master:~/zkdata$
```

分别在多个节点中启动运行多个ZooKeeper服务器的实例

```
hadoop@master:~$ cat startzookeeper
zookeeper-3.4.3/bin/zkServer.sh start
ssh slave1 /home/hadoop/zookeeper-3.4.3/bin/zkServer.sh start
ssh slave2 /home/hadoop/zookeeper-3.4.3/bin/zkServer.sh start
hadoop@master:~$ █
```

# ZooKeeper启动成功

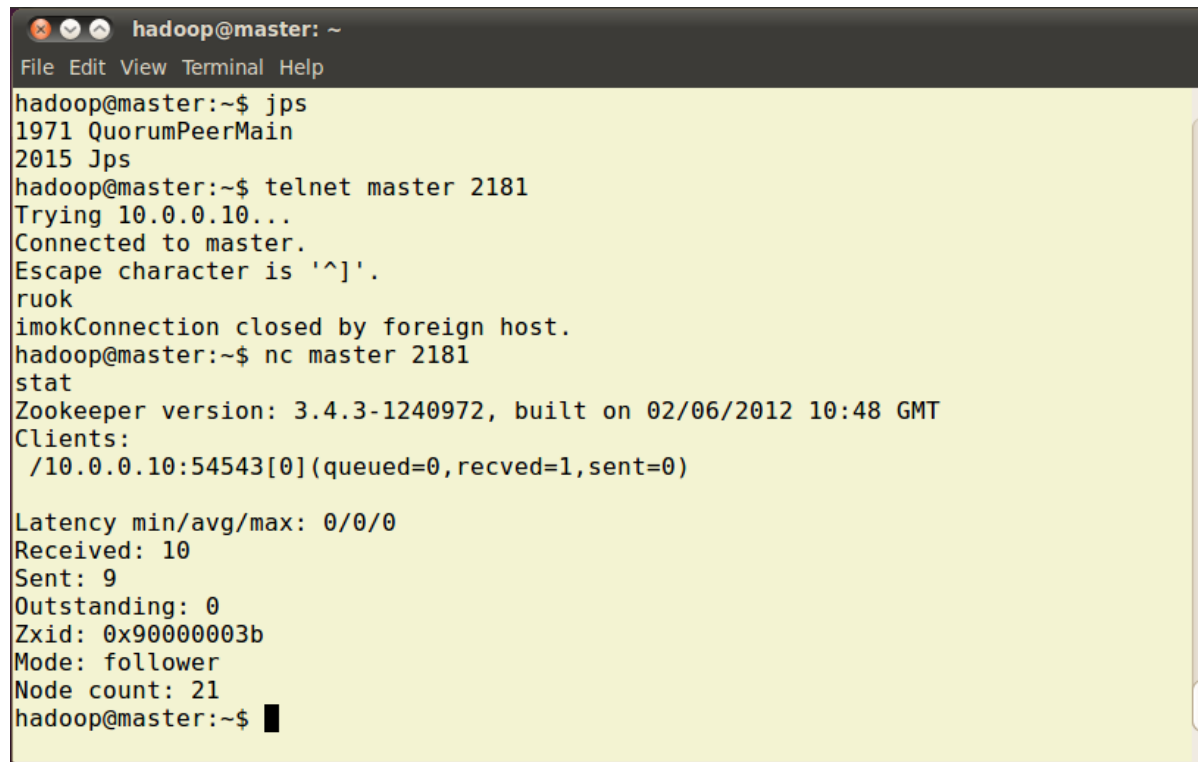
```
hadoopgsiaste r:-$ cat start zookeeper
zookeepe r -3. 4. 37b±n7 zkserve r .sh sta
rt
ssh stave1 7hosie7hadoop7 zookeeper -3. 4. 37b±n7zkServe r .sh sta
rt ssh stave2 7hosie7hadoop7 zookeeper -3. 4. 37b:tn7 zkServe r .sh sta
rt hadoopgaaster:-$ .7start zookeeper
JHX enabled by default
Us ±ng conf ±g: 7hosie7hadoop7 zookeeper -3. 4. 37b±n7 . .7 conf 7 zoo
.cfg
Sta rt±ng zookeeper . . .
STARTED
JHX enabled by default
Us ±ng conf ±g: 7hone7hadoop7 zookeeper -3. 4. 37b±n7 . .7 conf 7zoo
.cfg
Start:Ing zookeeper ... STARTED
JHX enabled by default
Us ±ng conf ±g: 7hoise7hadoop7 zookeeper -3. 4. 37b±n7 . .7 conf 7zoo
.cfg
Start:Ing zookeeper ... STARTED
hadoopgsiaste r:-$
```

```
hadoop@master:~$ jps
1971 QuorumPeerMain
2015 Jps
hadoop@master:~$ telnet master 2181
Trying 10.0.0.10...
Connected to master.
Escape character is '^]'.
ruok
imnknconnection closed by foreign host.
hadoop@master:~$
```

# ZooKeeper的四字命令

如同前面一页看到的命令ruok一样，ZooKeeper支持一系列的四个字母的命令，可以询问ZooKeeper的运行状态，用nc工具就可以打印状态，或者如同前一页一样使用telnet工具

用echo命令可以直接将命令行输入给nc（尚未试验成功）



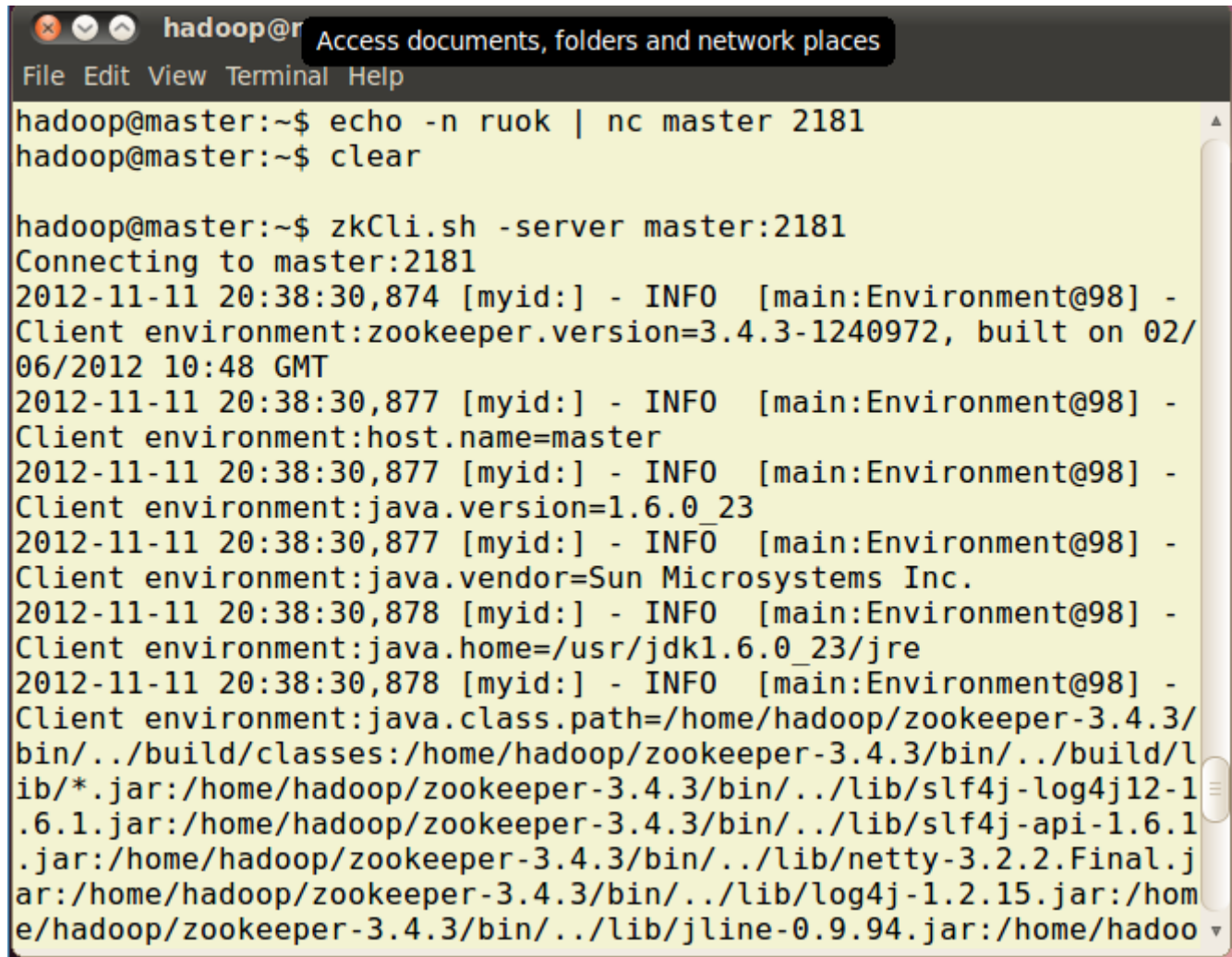
```
hadoop@master: ~  
File Edit View Terminal Help  
hadoop@master:~$ jps  
1971 QuorumPeerMain  
2015 Jps  
hadoop@master:~$ telnet master 2181  
Trying 10.0.0.10...  
Connected to master.  
Escape character is '^]'.  
ruok  
imokConnection closed by foreign host.  
hadoop@master:~$ nc master 2181  
stat  
Zookeeper version: 3.4.3-1240972, built on 02/06/2012 10:48 GMT  
Clients:  
 /10.0.0.10:54543[0](queued=0,recved=1,sent=0)  
  
Latency min/avg/max: 0/0/0  
Received: 10  
Sent: 9  
Outstanding: 0  
Zxid: 0x90000003b  
Mode: follower  
Node count: 21  
hadoop@master:~$
```

# ZooKeeper四字命令举例

ZooKeeper四字命令		功能描述
<b>conf</b>		输出相关服务配置的详细信息。
<b>cons</b>		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。
<b>dump</b>		列出未经处理的会话和临时节点。
<b>envi</b>		输出关于服务环境的详细信息（区别于 conf 命令）。
<b>reqs</b>		列出未经处理的请求
<b>ruok</b>		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。
<b>stat</b>		输出关于性能和连接的客户端的列表。
<b>wchs</b>		列出服务器 watch 的详细信息。
<b>wchc</b>		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。
<b>wchp</b>		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。



# ZooKeeper的命令行工具

A terminal window titled 'hadoop@r' with a subtitle 'Access documents, folders and network places'. The window has a menu bar with 'File', 'Edit', 'View', 'Terminal', and 'Help'. The terminal shows the following commands and output:

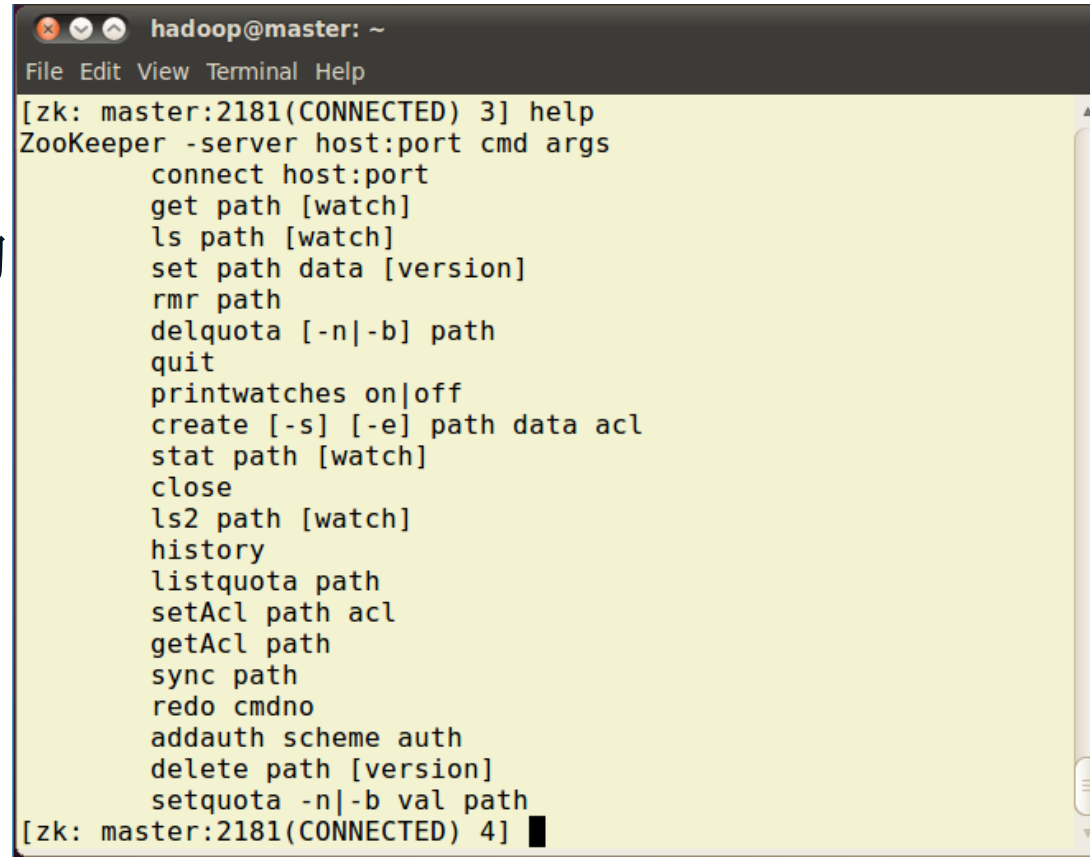
```
hadoop@master:~$ echo -n ruok | nc master 2181
hadoop@master:~$ clear

hadoop@master:~$ zkCli.sh -server master:2181
Connecting to master:2181
2012-11-11 20:38:30,874 [myid:] - INFO [main:Environment@98] -
Client environment:zookeeper.version=3.4.3-1240972, built on 02/
06/2012 10:48 GMT
2012-11-11 20:38:30,877 [myid:] - INFO [main:Environment@98] -
Client environment:host.name=master
2012-11-11 20:38:30,877 [myid:] - INFO [main:Environment@98] -
Client environment:java.version=1.6.0_23
2012-11-11 20:38:30,877 [myid:] - INFO [main:Environment@98] -
Client environment:java.vendor=Sun Microsystems Inc.
2012-11-11 20:38:30,878 [myid:] - INFO [main:Environment@98] -
Client environment:java.home=/usr/jdk1.6.0_23/jre
2012-11-11 20:38:30,878 [myid:] - INFO [main:Environment@98] -
Client environment:java.class.path=/home/hadoop/zookeeper-3.4.3/
bin/./build/classes:/home/hadoop/zookeeper-3.4.3/bin/./build/l
ib/*.jar:/home/hadoop/zookeeper-3.4.3/bin/./lib/slf4j-log4j12-1
.6.1.jar:/home/hadoop/zookeeper-3.4.3/bin/./lib/slf4j-api-1.6.1
.jar:/home/hadoop/zookeeper-3.4.3/bin/./lib/netty-3.2.2.Final.j
ar:/home/hadoop/zookeeper-3.4.3/bin/./lib/log4j-1.2.15.jar:/hom
e/hadoop/zookeeper-3.4.3/bin/./lib/jline-0.9.94.jar:/home/hadoo
```

# ZooKeeper命令行工具

zkCli.sh -server  
master 2181

Help命令可以列出支持的  
命令



```
hadoop@master: ~  
File Edit View Terminal Help  
[zk: master:2181(CONNECTED) 3] help  
ZooKeeper -server host:port cmd args  
connect host:port  
get path [watch]  
ls path [watch]  
set path data [version]  
rmr path  
delquota [-n|-b] path  
quit  
printwatches on|off  
create [-s] [-e] path data acl  
stat path [watch]  
close  
ls2 path [watch]  
history  
listquota path  
setAcl path acl  
getAcl path  
sync path  
redo cmdno  
addauth scheme auth  
delete path [version]  
setquota -n|-b val path  
[zk: master:2181(CONNECTED) 4]
```

# ZooKeeper的命令操作1

使用 ls 命令来查看  
当前ZooKeeper 中  
所包含的内容

```
[zk: master:2181(CONNECTED) 4] ls /  
[pppp, hbase, zookeeper, mydata]  
[zk: master:2181(CONNECTED) 5] █
```

## ZooKeeper的操作命令2

创建一个新的 **znode**，使用 `create /mydata IntelData`。这个命令创建了一个新的 **znode** 节点 “mydata” 以及与它关联的字符串 **IntelData**。

运行 `get` 命令获得数据

```
[zk: master:2181(CONNECTED) 17] create /mydata IntelData
Created /mydata
[zk: master:2181(CONNECTED) 18] get /mydata
IntelData
cZxid = 0xb000000008
ctime = Sun Nov 11 20:59:35 CST 2012
mZxid = 0xb000000008
mtime = Sun Nov 11 20:59:35 CST 2012
pZxid = 0xb000000008
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 9
numChildren = 0
[zk: master:2181(CONNECTED) 19] ls /
[hbase, zookeeper, mydata]
[zk: master:2181(CONNECTED) 20] █
```

# ZooKeeper的命令操作3

通过 `set` 命令来对 `mydata` 所关联的字符串进行设置

```
[zk: master:2181(CONNECTED) 20] set /mydata NewIntelData
cZxid = 0xb000000008
ctime = Sun Nov 11 20:59:35 CST 2012
mZxid = 0xb000000009
mtime = Sun Nov 11 21:04:45 CST 2012
pZxid = 0xb000000008
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 12
numChildren = 0
[zk: master:2181(CONNECTED) 21] get /mydata
NewIntelData
cZxid = 0xb000000008
ctime = Sun Nov 11 20:59:35 CST 2012
mZxid = 0xb000000009
mtime = Sun Nov 11 21:04:45 CST 2012
pZxid = 0xb000000008
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 12
numChildren = 0
[zk: master:2181(CONNECTED) 22] █
```

# ZooKeeper的命令操作4

## 删除命令

```
[zk: master:2181(CONNECTED) 22] delete /mydata  
[zk: master:2181(CONNECTED) 23] ls /  
[hbase, zookeeper]  
[zk: master:2181(CONNECTED) 24] █
```

# ZooKeeper的编程接口

ZooKeeper API 包含5个包： `org.apache.zookeeper` ，  
`org.apache.zookeeper.data` ， `org.apache.zookeeper.server` ，  
`org.apache.zookeeper.server.quorum` 和 `org.apache.zookeeper.server.upgrade` 。  
其中 `org.apache.zookeeper` 包含 `ZooKeeper` 类是编程时最常用的类文件

为了使用 `ZooKeeper` 服务，应用程序首先创建一个`Zookeeper` 实例，与`ZooKeeper` 服务建立起连接， `ZooKeeper` 系统将会分配给此连接回话一个 ID 值，客户端会周期地向服务器发送心跳来维持会话的连接，并调用 `ZooKeeper API` 来做相应的处理

与命令行提供的功能类似，**API**也提供类似的功能

功能	描述
<code>create</code>	在本地目录树中创建一个节点
<code>delete</code>	删除一个节点
<code>exists</code>	测试本地是否存在目标节点
<code>get/set data</code>	从目标节点上读取 / 写数据
<code>get/set ACL</code>	获取 / 设置目标节点访问控制列表信息
<code>get children</code>	检索一个子节点上的列表
<code>sync</code>	等待要被传送的数据

# ZooKeeper的客户端工作

- 1 与ZooKeeper服务端进行通信，包括：连接，发送消息，接受消息。
- 2 发送心跳信息，保持与ZooKeeper服务端的有效连接与Session的有效性。
- 3 错误处理，如果客户端当前连接的ZooKeeper服务端失效，自动切换到另一台有效的ZooKeeper服务端。
- 4 管理Watcher，处理异常调用和Watcher。



# ZooKeeper代码举例1

```
import java.io.*;
import org.apache.zookeeper.CreateMode;
import org.apache.zookeeper.KeeperException;
import org.apache.zookeeper.Watcher;
import org.apache.zookeeper.ZooDefs.Ids;
import org.apache.zookeeper.ZooKeeper;

public class Demo {
    private static final int SESSION_TIMEOUT=30000;
    ZooKeeper zk;

    Watcher wh=new Watcher(){
        public void process(org.apache.zookeeper.WatchedEvent event)
        {
            System.out.println(event.toString());
        }
    };

    private void createZKInstance() throws IOException
    {
        zk=new ZooKeeper("master:2181", Demo.SESSION_TIMEOUT, this.wh);
    }
}
```

```

private void ZKOperations() throws IOException, InterruptedException, KeeperException
{
    System.out.println(" create nodes on /mydata2 data IntelData2 ACL:OPEN_ACL_UNSAFE, Persistent");
    zk.create("/mydata2", "IntelData2".getBytes(), Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);

    System.out.println("check if exist");
    System.out.println(new String(zk.getData("/mydata2", false, null)));

    System.out.println("do modification");
    zk.setData("/mydata2", "IntelData3".getBytes(), -1);

    System.out.println("check modification");
    System.out.println(new String(zk.getData("/mydata2", false, null)));

    System.out.println("delete node");
    zk.delete("/mydata2", -1);

    System.out.println("check delete");
    System.out.println("nodes exists: ["+zk.exists("/mydata2", false)+""]");
}

private void ZKClose() throws InterruptedException
{
    zk.close();
}

/**
 * @param args
 */
public static void main(String[] args) throws IOException, InterruptedException, KeeperException{
    Demo dm=new Demo();
    dm.createZKInstance();
    dm.ZKOperations();
    dm.ZKClose();
}
}

```

# ZooKeeper示例代码的执行结果

```
hadoop@master:~/temp$ java -cp ../zookeeper-3.4.3/zookeeper-3.4.3.jar:../zookeeper-3.4.3/lib/slf4j-api-1.6.1.jar Demo
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
  create nodes on /mydata2 data IntelData2 ACL:OPEN_ACL_UNSAFE, Persistent
WatchedEvent state:SyncConnected type:None path:null
check if exist
IntelData2
do modification
check modification
IntelData3
delete node
check delete
nodes exists: [null]
hadoop@master:~/temp$ █
```

# ZNode

ZooKeeper树中的节点称作znode。 znode会维护一个包含数据修改和ACL修改版本号的Stat结构体，这个结构体还包含时间戳字段。

版本号和时间戳让ZooKeeper可以校验缓存，协调更新。每次修改znode数据的时候，版本号会增加。

客户端获取数据的同时，也会取得数据的版本号。执行更新或者删除操作时，客户端必须提供版本号。如果提供的版本号与数据的实际版本不匹配，则更新操作失败。

# ZNode上的观察器

客户端可以在znode上设置观察器。对znode的修改将触发观察器，然后移除观察器。观察器被触发时，ZooKeeper向客户端发送一个通知。

观察器用来让应用程序可以得到异步的通知

# ZooKeeper的Stat结构体

czxid: 创建节点的事务的zxid

mzxid: 对znode最近修改的zxid

ctime: 毫秒数表示的znode创建时间（自从创建开始）

mtime: 毫秒数表示的znode最近修改时间（自从修改开始）

version: znode数据的修改次数 cversion: znode子节点修

改次数 aversion: znode的ACL修改次数

ephemeralOwner: 如果znode是临时节点，则指示节点所有者的会话ID；如果不是临时节点，则为零。

dataLength: znode数据长度。

numChildren: znode子节点个数。

# 会话过期的处理

会话过期由**ZooKeeper**集群，而不是客户端来管理。

客户端与集群建立会话时会提供上面讨论的超时值。

集群使用这个值来确定客户端会话何时过期。 集群在指定的超时时间内没有得到客户端的消息时发生会话过期。

会话过期时集群将删除会话的所有临时节点，立即通知所有(观察器节点的)客户端。

此时已过期会话的客户端还是同集群断开连接的，不会被通知会话已经过期，直到(除非)客户端重新建立到集群的连接，这时候已过期会话的观察器才会收到“会话已过期”通知。

# ZooKeeper的观察器Watches

- 观察器是ZooKeeper中一个非常重要的概念，实际是一个观察器，ZooKeeper中的所有读操作：getData()、getChildren()和exists()，都有一个设置观察器作为进行触发的选项。
- ZooKeeper对观察器的定义是：观察器事件是在被观察器数据发生变化时，发送给建立观察器的客户端的一次性观察器。
  - 一次触发：触发一次之后，观察器会被删除
  - 发送给客户端：观察器事件是异步地发送给观察器者（客户端）的。ZooKeeper会保证顺序：在收到观察器事件之前，客户端不会看到已经为之设置观察器的节点的改动。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器事件和更新操作的返回码。但是不同客户端看到的事情都有一致的顺序。



# ZooKeeper关于观察器Watches的保证

观察器与其他事件、其他观察器和异步回应是顺序的。

ZooKeeper客户端库保证一切都是按顺序分发的。

客户端将在看到znode的新数据之前收到其观察器事件。

观察器事件的次序与ZooKeeper服务看到的更新次序一致。

# ZooKeeper中的权限

- ZooKeeper使用ACL控制对节点的访问，ACL指定一个ID集合，以及这些ID相关联的权限。
- ACL仅仅用于某特定节点，ACL不会应用到子节点。比如说，/app 只能被ip:172.16.16.1读取，/app/status可以被所有用户读取。ACL不是递归的。
- ZooKeeper支持下述权限：
  - CREATE：可创建子节点
  - READ：可获取节点数据和子节点列表
  - WRITE：可设置节点数据
  - DELETE：可删除子节点
  - ADMIN：可设置节点权限

# 节点的权限设置（创建节点）

```
List<ACL> acls = new ArrayList<ACL>(2);
```

```
Id id1 = new Id("digest", DigestAuthenticationProvider.generateDigest("admin:admin123"));
```

```
ACL acl1 = new ACL(ZooDefs.Perms.ALL, id1);
```

```
Id id2 = new Id("digest", DigestAuthenticationProvider.generateDigest("guest:guest123"));
```

```
ACL acl2 = new ACL(ZooDefs.Perms.READ, id2);
```

```
acls.add(acl1);
```

```
acls.add(acl2);
```

```
ZooKeeper zk = new ZooKeeper("127.0.0.1:2181", 10000, new DefaultWatcher());
```

```
zk.create("/test", new byte[0], acls, CreateMode.PERSISTENT);
```

# ZooKeeper的一致性保证1

Zookeeper 是一种高性能、可扩展的服务。Zookeeper 的读写速度非常快，并且读的速度要比写的速度更快

在进行读操作的时候，ZooKeeper 依然能够为旧的数据提供服务。这些都是由于 ZooKeeper 所提供的一致性保证

- 顺序一致性：客户端的更新顺序与它们被发送的顺序一致
- 原子性：更新操作要么成功，要么失败，没有第三种结果
- 单系统镜像：无论客户端连接到哪一个客户端，客户端将看到相同的ZooKeeper视图

# ZooKeeper一致性保证2

可靠性：一旦一个更新操作被应用，那么在客户端再次更新它之前，它的值将不会改变

- 1 . 如果客户端成功地获得了正确的返回代码，那么说明更新已经成果。如果不能获得返回代码（由于通信错误、超时等等），那么客户端将不知道更新操作是否生效。

- 2 . 当从故障恢复的时候，任何客户端能够看到的执行成功的更新操作将不会被回滚。

实时性：在特定的一段时间内，客户端看到的系统需要被保证是实时的（在十几秒的时间里）。在此时间段内，任何系统的改变将被客户端看到，或者被客户端侦测到。