

ZooKeeper系列之一：ZooKeeper简介

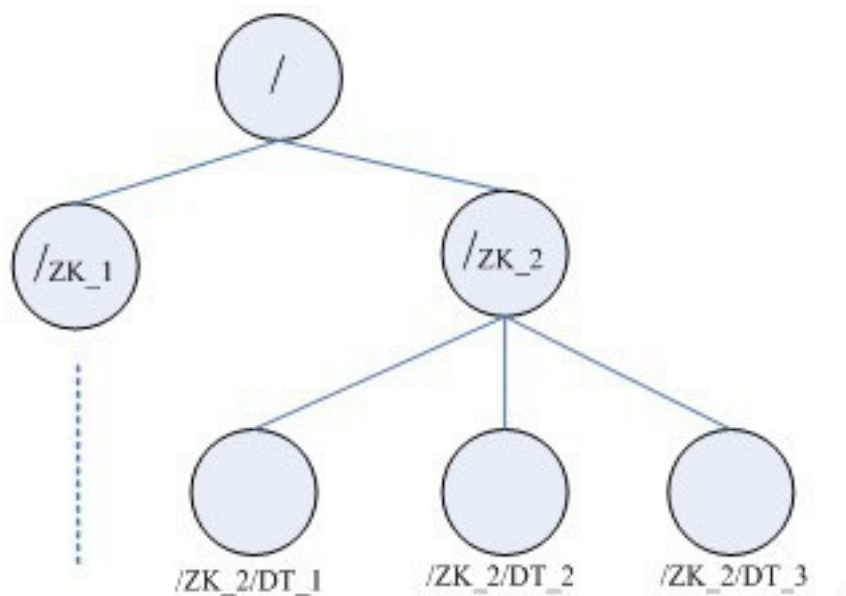
ZooKeeper 是一个为分布式应用所设计的分布的、开源的协调服务。分布式的应用可以建立在同步、配置管理、分组和命名等服务的更高级别的实现的基础之上。ZooKeeper 意欲设计一个易于编程的环境，它的文件系统使用我们所熟悉的目录树结构。ZooKeeper 使用 Java 所编写，但是支持 Java 和 C 两种编程语言。

众所周知，协调服务非常容易出错，但是却很难恢复正常，例如，协调服务很容易处于竞态以至于出现死锁。我们设计 ZooKeeper 的目的是为了减轻分布式应用程序所承担的协调任务。

ZooKeeper系列之二：ZooKeeper数据模型、命名空间以及节点的概念

ZooKeeper数据模型和层次命名空间

提供的命名空间与标准的文件系统非常相似。一个名称是由通过斜线分隔开的路径名序列所组成的。ZooKeeper中的每一个节点是都通过路径来识别。



下图是Zookeeper中节点的数据模型，这种树形结构的命名空间操作方便且易于理解。

图：ZooKeeper层次命名空间

ZooKeeper中节点和临时节点

ZooKeeper的节点是通过像树一样的结构来进行维护的，并且每一个节点通过路径来标示以及访问。除此之外，每一个节点还拥有自身的一些信息，包括：数据、数据长度、创建时间、修改时间等等。从这样一类既含有数据，又作为路径表标示的节点的特点中，可以看出，ZooKeeper的节点既可以被看做是一个文件，又可以被看做是一个目录，它同时具有二者的特点。为了便于表达，今后我们将使用Znode来表示所讨论的ZooKeeper节点。

具体地说，Znode维护着数据、ACL（access control list，访问控制列表）、时间戳等交换版本号等数据结构，它通过对这些数据的管理来让缓存生效并且令协调更新。每当Znode中的数据更新后它所维护的版本号将增加，这非常类似于数据库中计数器时间戳的操作方式。

另外Znode还具有原子性操作的特点：命名空间中，每一个Znode的数据将被原子地读写。读操作将读取与Znode相关的所有数据，写操作将替换掉所有的数据。除此之外，每一个节点都有一个访问控制列表，这个访问控制列表规定了用户操作的权限。ZooKeeper中同样存在临时节点。这些节点与session同时存在，当session生命周期结束，这些临时节点也将被删除。临时节点在某些场合也发挥着非常重要的作用。

ZooKeeper系列之三：ZooKeeper的安装

ZooKeeper的安装模式分为三种，分别为：单机模式（stand-alone）、集群模式和集群伪分布模式。ZooKeeper 单机模式的安装相对比较简单，如果第一次接触ZooKeeper的话，建议安装ZooKeeper单机模式或者集群伪分布模式。

1) 单机模式

首先，从Apache官方网站下载一个ZooKeeper 的最近稳定版本。

<http://hadoop.apache.org/zookeeper/releases.html>

ZooKeeper 要求 JAVA 的环境才能运行，并且需要 JAVA6 以上的版本，可以从 SUN 官网上下载，并对 JAVA 环境变量进行设置。除此之外，为了今后操作的方便，我们需要对 ZooKeeper 的环境变量进行配置，方法如下，在 /etc/profile 文件中加入如下的内容：

```
#Set ZooKeeper Enviroment
export ZOOKEEPER_HOME=/root/hadoop-0.20.2/zookeeper-3.3.1
export PATH=$PATH:$ZOOKEEPER_HOME/bin:$ZOOKEEPER_HOME/conf
```

ZooKeeper 服务器包含在单个 JAR 文件中，安装此服务需要用户创建一个配置文档，并对其进行设置。我们在 ZooKeeper-*.*. 目录（我们以当前 ZooKeeper 的最新版 3.3.1 为例，故此下面的“ZooKeeper-*.*. ”都将写为“ZooKeeper-3.3.1”）的 conf 文件夹下创建一个 zoo.cfg 文件，它包含如下的内容：

```
tickTime=2000
dataDir=/var/zookeeper
clientPort=2181
```

在这个文件中，我们需要指定 dataDir 的值，它指向了一个目录，这个目录在开始的时候需要为空。下面是每个参数的含义：

tickTime：基本事件单元，以毫秒为单位。它用来指示心跳，最小的 session 过期时间为两倍的 tickTime。

dataDir：存储内存中数据库快照的位置，如果不设置参数，更新事务日志将被存储到默认位置。

clientPort：监听客户端连接的端口

使用单机模式时用户需要注意：这种配置方式下没有 ZooKeeper 副本，所以如果 ZooKeeper 服务器出现故障，ZooKeeper 服务将会停止。

以下代码清单 A 是我们的根据自身情况所设置的 zookeeper 配置文档：zoo.cfg

代码清单 A：zoo.cfg

```
# The number of milliseconds of each tick
tickTime=2000
```

```
# the directory where the snapshot is stored.
dataDir=/root/hadoop-0.20.2/zookeeper-3.3.1/snapshot/data
```

```
# the port at which the clients will connect
```

```
clientPort=2181
```

2) 集群模式

为了获得可靠的 ZooKeeper 服务，用户应该在一个集群上部署 ZooKeeper。只要集群上大多数的 ZooKeeper 服务启动了，那么总的 ZooKeeper 服务将是可用的。另外，最好使用奇数台机器。如果 zookeeper 拥有 5 台机器，那么它就能处理 2 台机器的故障了。

之后的操作和单机模式的安装类似，我们同样需要对 JAVA 环境进行设置，下载最新的 ZooKeeper 稳定版本并配置相应环境变量。不同之处在于每台机器上 conf/zoo.cfg 配置文件的参数设置，参考下面的配置：

```
tickTime=2000
dataDir=/var/zookeeper/
clientPort=2181
initLimit=5
syncLimit=2
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
```

“server.id=host:port:port.”指示了不同的 ZooKeeper 服务器的自身标识，作为集群的一部分的机器应该知道 ensemble 中的其它机器。用户可以从“server.id=host:port:port.”中读取相关的信息。在服务器的 data（dataDir 参数所指定的目录）目录下创建一个文件名为 myid 的文件，这个文件中仅含有一行的内容，指定的是自身的 id 值。比如，服务器“1”应该在 myid 文件中写入“1”。这个 id 值必须是 ensemble 中唯一的，且大小在 1 到 255 之间。这一行配置中，第一个端口（port）是从（follower）机器连接到主（leader）机器的端口，第二个端口是用来进行 leader 选举的端口。在这个例子中，每台机器使用三个端口，分别是：clientPort，2181；port，2888；port，3888。

我们在拥有三台机器的 Hadoop 集群上测试使用 ZooKeeper 服务，下面代码清单 B 是我们根据自身情况所设置的 ZooKeeper 配置文档：

代码清单 B：zoo.cfg

```
# The number of milliseconds of each tick
tickTime=2000
```

```
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
```

```
# The number of ticks that can pass between
```

```
# sending a request and getting an acknowledgement
syncLimit=5
```

```
# the directory where the snapshot is stored.
dataDir=/root/hadoop-0.20.2/zookeeper-3.3.1/snapshot/d1
```

```
# the port at which the clients will connect
clientPort=2181
```

```
server.1=IP1:2887:3887
server.2=IP2:2888:3888
server.3=IP3:2889:3889
```

清单中的 IP 分别对应的配置分布式 ZooKeeper 的 IP 地址。当然，也可以通过机器名访问 zookeeper，但是需要在 ubuntu 的 hosts 环境中进行设置。读者可以查阅 Ubuntu 以及 Linux 的相关资料进行设置。

3)集群伪分布

简单来说，集群伪分布模式就是在单机下模拟集群的ZooKeeper服务。

那么，如何对配置 ZooKeeper 的集群伪分布模式呢？其实很简单，在 zookeeper 配置文档中，clientPort 参数用来设置客户端连接 zookeeper 的端口。server.1=IP1:2887:3887 中，IP1 指示的是组成 ZooKeeper 服务的机器 IP 地址，2887 为用来进行 leader 选举的端口，3887 为组成 ZooKeeper 服务的机器之间通信的端口。集群伪分布模式我们使用每个配置文档模拟一台机器，也就是说，需要在单台机器上运行多个 zookeeper 实例。但是，我们必须要保证各个配置文档的 clientPort 不能冲突。

下面是我们所配置的集群伪分布模式，通过 zoo1.cfg，zoo2.cfg，zoo3.cfg 模拟了三台机器的 ZooKeeper 集群。详见代码清单 C：

代码清单C：

zoo1.cfg：

```
# The number of milliseconds of each tick
tickTime=2000
```

```
# The number of ticks that the initial
# synchronization phase can take
```

```
initLimit=10

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5

# the directory where the snapshot is stored.
dataDir=/root/hadoop-0.20.2/zookeeper-3.3.1/d_1

# the port at which the clients will connect
clientPort=2181

server.1=localhost:2887:3887
server.2=localhost:2888:3888
server.3=localhost:2889:3889
```

zoo2.cfg :

```
# The number of milliseconds of each tick
tickTime=2000

# The number of ticks that the initial
# synchronization phase can take
initLimit=10

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5

# the directory where the snapshot is stored.
dataDir=/root/hadoop-0.20.2/zookeeper-3.3.1/d_2

# the port at which the clients will connect
clientPort=2182

#the location of the log file
dataLogDir=/root/hadoop-0.20.2/zookeeper-3.3.1/logs

server.1=localhost:2887:3887
server.2=localhost:2888:3888
server.3=localhost:2889:3889
```

zoo3.cfg :

```
# The number of milliseconds of each tick
tickTime=2000

# The number of ticks that the initial
# synchronization phase can take
initLimit=10

# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5

# the directory where the snapshot is stored.
dataDir=/root/hadoop-0.20.2/zookeeper-3.3.1/d_2

# the port at which the clients will connect
clientPort=2183

#the location of the log file
dataLogDir=/root/hadoop-0.20.2/zookeeper-3.3.1/logs

server.1=localhost:2887:3887
server.2=localhost:2888:3888
server.3=localhost:2889:3889
```

从上述三个代码清单中可以看到，除了 clientPort 不同之外，dataDir 也不同。另外，不要忘记在 dataDir 所对应的目录中创建 myid 文件来指定对应的 zookeeper 服务器实例。

这里 ZooKeeper 的安装已经说完了，下一节我们来谈一谈对 ZooKeeper 的参数配置 的理解。

ZooKeeper系列之四：ZooKeeper的配置

ZooKeeper 的功能特性通过 ZooKeeper 配置文件来进行控制管理（zoo.cfg 配置文件）。ZooKeeper 这样的设计其实是有它自身的原因的。通过前面对 ZooKeeper 的配置可以看出，对 ZooKeeper 集群进行配置的时候，它的配置文档是完全相同的（对于集群伪分布模式来说，只有很少的部分是不同的）。这样的配置方使得在部署 ZooKeeper 服务的时候非常地方便。另外，如果服务器使用不同的配置文件，必须要确保不同配置文件中的服务器列表相匹配。

在设置 ZooKeeper 配置文档的时候，某些参数是可选的，但是某些参数是必须的。这些必须的参数就构成了 ZooKeeper 配置文档的最低配置要求。

下面是在最低配置要求中必须配置的参数：

1) 最低配置

`clientPort`

监听客户端连接的端口；

`dataDir`

存储内存中数据库快照的位置；

注意 应该谨慎地选择日志存放的位置，使用专用的日志存储设备能够大大地提高系统的性能，如果将日志存储在比较繁忙的存储设备上，那么将会在很大程度上影响系统的性能。

`tickTime`

基本事件单元，以毫秒为单位。它用来控制心跳和超时，默认情况下最小的会话超时时间为两倍的 `tickTime` 。

2) 高级配置

下面是高级配置要求中可选的配置参数，用户可以使用下面的参数来更好地规定 ZooKeeper 的行为：

`dataLogDir`

这个操作将管理机器把事务日志写入到“`dataLogDir`”所指定的目录，而不是“`dataDir`”所指定的目录。这将允许使用一个专用的日志设备并且帮助我们避免日志和快照之间的竞争。配置如下：

```
#the location of the log file
dataLogDir=/root/hadoop-0.20.2/zookeeper-3.3.1/log/data_log
```

`maxClientCnxns`

这个操作将限制连接到 ZooKeeper 的客户端的数量，限制并发连接的数量，它通过 IP 来区分不同的客户端。此配置选项可以用来阻止某些类别的 Dos 攻击。将它设置为 0 或者忽略而不进行设置将会取消对并发连接的限制。

例如，此时我们将 `maxClientCnxns` 的值设置为 1，如下所示：

```
#set maxClientCnxns
maxClientCnxns=1
```

启动 ZooKeeper 之后，首先用一个客户端连接到 ZooKeeper 服务器之上。然后，当第二个客户端尝试对 ZooKeeper 进行连接，或者某些隐式的对客户端的连接操作，将会触发 ZooKeeper 的上述配置。系统会提示相关信息，如下图 1 所示：

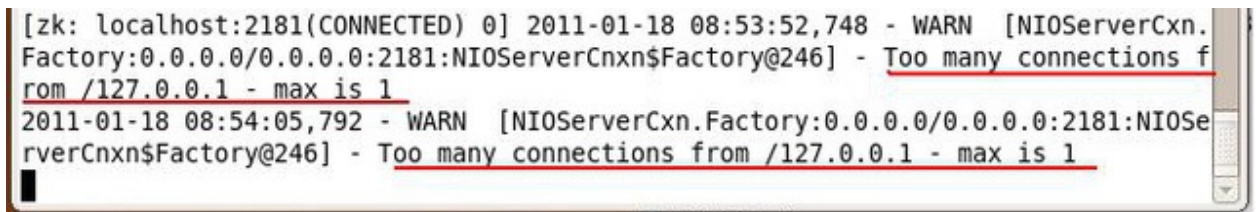


图 1： ZooKeeper maxClientCnxns 异常

minSessionTimeout 和 maxSessionTimeout

最小的会话超时时间以及最大的会话超时时间。其中，最小的会话超时时间默认情况下为 2 倍的 tickTime 时间，最大的会话超时时间默认情况下为 20 倍的会话超时时间。在启动时，系统会显示相应信息，见下图 2 所示，默认会话超时时间：



图2：默认会话超时时间

从上图中可以看书， minSessionTimeout 以及 maxSessionTimeout 的值均为 -1，现在我们来设置系统的最小会话超时时间以及最大会话超时时间，如下所示：

```
#set minSessionTimeout  
minSessionTimeout=1000
```

```
#set maxSessionTimeout  
maxSessionTimeout=10000
```

在配置 minSessionTmeout 以及 maxSessionTimeout 的值的时候需要注意，如果将此值设置的太小的话，那么会话很可能刚刚建立便由于超时而不得不退出。一般情况下，不能将此值设置的比 tickTime 的值还小。

3) 集群配置

initLimit

此配置表示，允许 follower（相对于 leader 而言的“客户端”）连接并同步到 leader 的初始化连接时间，它以 tickTime 的倍数来表示。当超过设置倍数的 tickTime 时间，则连接失败。

syncLimit

此配置表示，leader 与 follower 之间发送消息，请求和应答时间长度。如果 follower 在设置的时间内不能与 leader 进行通信，那么此 follower 将被丢弃。

ZooKeeper系列之五：ZooKeeper的运行

我们这里所介绍的是对应 [ZooKeeper系列之三：ZooKeeper的安装](#) 模式的运行。

1) 单机模式

用户可以通过下面的命令来启动 ZooKeeper 服务：

```
zkServer.sh start
```

这个命令默认情况下执行 ZooKeeper 的 conf 文件夹下的 zoo.cfg 配置文件。当运行成功用户会看到类似如下的提示界面：

```
root@ubuntu:~# zkServer.sh start
JMX enabled by default
Using config: /root/hadoop-0.20.2/zookeeper-3.3.1/bin/../conf/zoo.cfg
Starting zookeeper ...
STARTED
... ..

2011-01-19 10:04:42,300 - WARN [main:QuorumPeerMain@105] - Either no config or no
quorum defined in config, running in standalone mode

... ..

2011-01-19 10:04:42,419 - INFO [main:ZooKeeperServer@660] - tickTime set to 2000
2011-01-19 10:04:42,419 - INFO [main:ZooKeeperServer@669] - minSessionTimeout set to -1
2011-01-19 10:04:42,419 - INFO [main:ZooKeeperServer@678] - maxSessionTimeout set to -1
2011-01-19 10:04:42,560 - INFO [main:NIOServerCnxn$Factory@143] - binding to port
0.0.0.0/0.0.0.0:2181

2011-01-19 10:04:42,806 - INFO [main:FileSnap@82] - Reading snapshot /root/hadoop-0.20.2/
zookeeper-3.3.1/data/version-2/snapshot.2000000036

2011-01-19 10:04:42,927 - INFO [main:FileSnap@82] - Reading snapshot /root/hadoop-0.20.2/
zookeeper-3.3.1/data/version-2/snapshot.2000000036

2011-01-19 10:04:42,950 - INFO [main:FileTxnSnapLog@208] - Snapshotting: 4000000058
```

从上面可以看出，运行成功后，系统会列出 ZooKeeper 运行的相关环境配置信息。

2) 集群模式

集群模式下需要用户每台 ZooKeeper 机器上运行第一部分的命令，这里不再赘述。

3) 集群伪分布模式

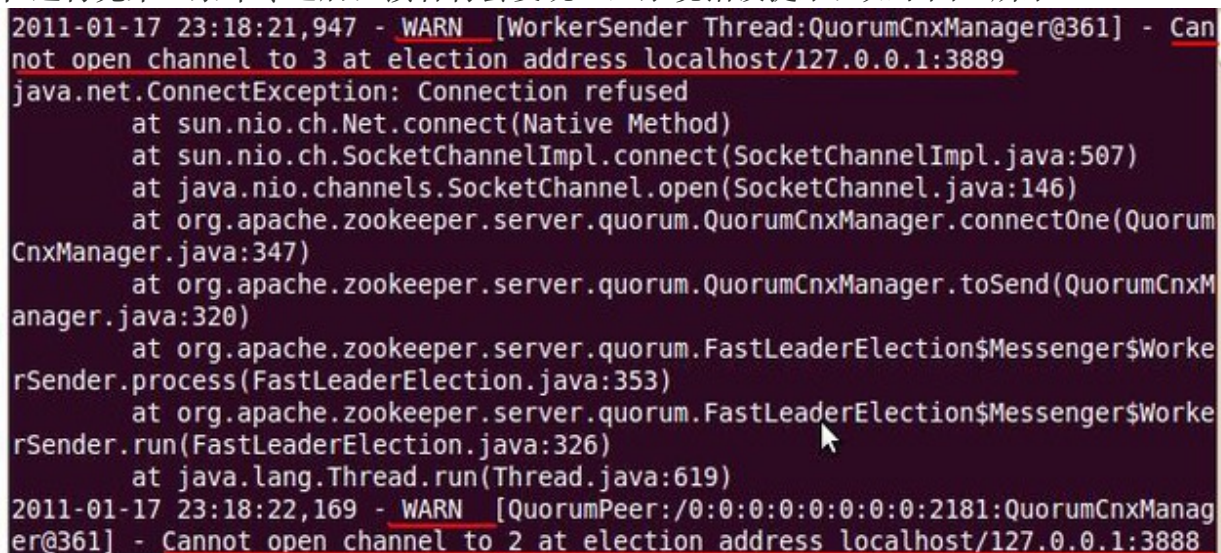
在集群伪分布模式下，我们只有一台机器，但是要运行三个 ZooKeeper 服务实例。此时，如果再使用上述命令肯定行不通的。这里，我们通过下面三条命令来运行 ZooKeeper 系列之三：ZooKeeper 的安装 中我们配置的 ZooKeeper 服务。如下所示：

```
zkServer.sh start zoo1.cfg
```

```
zkServer.sh start zoo2.cfg
```

```
zkServer.sh start zoo3.cfg
```

在运行完第一条命令之后，读者将会发现一些系统错误提示，如下图 1 所示：



```
2011-01-17 23:18:21,947 - WARN [WorkerSender Thread:QuorumCnxManager@361] - Cannot open channel to 3 at election address localhost/127.0.0.1:3889
java.net.ConnectException: Connection refused
    at sun.nio.ch.Net.connect(Native Method)
    at sun.nio.ch.SocketChannelImpl.connect(SocketChannelImpl.java:507)
    at java.nio.channels.SocketChannel.open(SocketChannel.java:146)
    at org.apache.zookeeper.server.quorum.QuorumCnxManager.connectOne(QuorumCnxManager.java:347)
    at org.apache.zookeeper.server.quorum.QuorumCnxManagerToSend(QuorumCnxManager.java:320)
    at org.apache.zookeeper.server.quorum.FastLeaderElection$Messenger$WorkerSender.process(FastLeaderElection.java:353)
    at org.apache.zookeeper.server.quorum.FastLeaderElection$Messenger$WorkerSender.run(FastLeaderElection.java:326)
    at java.lang.Thread.run(Thread.java:619)
2011-01-17 23:18:22,169 - WARN [QuorumPeer:/0:0:0:0:0:0:0:0:2181:QuorumCnxManager@361] - Cannot open channel to 2 at election address localhost/127.0.0.1:3888
```

图 1：集群伪分布异常提示

产生如上图所示的异常信息是由于 ZooKeeper 服务的每个实例都拥有全局的配置信息，它们在启动的时候需要随时地进行 Leader 选举操作（此部分内容下面将会详细讲述）。此时第一个启动的 Zookeeper 需要和另外两个 ZooKeeper 实例进行通信。但是，另外两个 ZooKeeper 实例还没有启动起来，因此将会产生上述所示的异常信息。

我们直接将其忽略即可，因为当把图示中的“2号”和“3号” ZooKeeper 实例启动起来之后，相应的异常信息就自然而然地消失。

ZooKeeper系列之六：ZooKeeper四字命令

ZooKeeper 支持某些特定的四字命令字母与其的交互。它们大多是查询命令，用来获取 ZooKeeper 服务的当前状态及相关信息。用户在客户端可以通过 telnet 或 nc 向 ZooKeeper 提交相应的命令。ZooKeeper 常用四字命令见下表 1 所示：

表 1：ZooKeeper 四字命令

ZooKeeper 四字命令	功能描述
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
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。
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。

下图 1 是 ZooKeeper 四字命令的简单用例：

```
root@ubuntu-laptop:~# echo ruok | nc 10.77.20.23 2181
imokroot@ubuntu-laptop:~# echo conf | nc 10.77.20.23 2181
clientPort=2181
dataDir=/root/hadoop-0.20.2/zookeeper-3.3.1/d_1/version-2
dataLogDir=/root/hadoop-0.20.2/zookeeper-3.3.1/d_1/version-2
tickTime=2000
maxClientCnxns=10
minSessionTimeout=4000
maxSessionTimeout=40000
serverId=1
initLimit=10
syncLimit=5
electionAlg=3
electionPort=3887
quorumPort=2887
peerType=0
root@ubuntu-laptop:~#
```

图 1： ZooKeeper 四字命令用例

ZooKeeper系列之七：ZooKeeper命令行工具

当启动 ZooKeeper 服务成功之后，输入下述命令，连接到 ZooKeeper 服务：

```
zkCli.sh -server 10.77.20.23:2181
```

连接成功后，系统会输出 ZooKeeper 的相关环境以及配置信息，并在屏幕输出“Welcome to ZooKeeper”等信息。

输入 help 之后，屏幕会输出可用的 ZooKeeper 命令，如下图 1 所示：


```
[zk: 10.77.20.23:2181(CONNECTED) 1] help
ZooKeeper -server host:port cmd args
  connect host:port
  get path [watch]
  ls path [watch]
  set path data [version]
  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
```

图 1： ZooKeeper 命令

ZooKeeper系列之八：ZooKeeper的简单操作

1) 使用 ls 命令来查看当前 ZooKeeper 中所包含的内容：

```
[zk: 10.77.20.23:2181(CONNECTED) 1] ls /
[zookeeper]
```

2) 创建一个新的 znode ，使用 create /zk myData 。这个命令创建了一个新的 znode 节点“ zk ”以及与它关联的字符串：

```
[zk: 10.77.20.23:2181(CONNECTED) 2] create /zk myData
Created /zk
```

3) 再次使用 ls 命令来查看现在 zookeeper 中所包含的内容：

```
[zk: 10.77.20.23:2181(CONNECTED) 3] ls /
[zk, zookeeper]
```

此时看到， zk 节点已经被创建。

4) 下面我们运行 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: 10.77.20.23:2181(CONNECTED) 4] get /zk  
myData  
Zxid = 0x40000000c  
time = Tue Jan 18 18:48:39 CST 2011  
Zxid = 0x40000000c  
mtime = Tue Jan 18 18:48:39 CST 2011  
pZxid = 0x40000000c  
cversion = 0  
dataVersion = 0  
aclVersion = 0  
ephemeralOwner = 0x0  
dataLength = 6  
numChildren = 0
```

5) 下面我们通过 set 命令来对 zk 所关联的字符串进行设置:

```
[zk: 10.77.20.23:2181(CONNECTED) 5] set /zk shenlan211314  
cZxid = 0x40000000c  
ctime = Tue Jan 18 18:48:39 CST 2011  
mZxid = 0x40000000d  
mtime = Tue Jan 18 18:52:11 CST 2011  
pZxid = 0x40000000c  
cversion = 0  
dataVersion = 1  
aclVersion = 0  
ephemeralOwner = 0x0  
dataLength = 13  
numChildren = 0
```

6) 下面我们将刚才创建的 znode 删除:

```
[zk: 10.77.20.23:2181(CONNECTED) 6] delete /zk
```

7) 最后再次使用 ls 命令查看 ZooKeeper 所包含的内容:

```
[zk: 10.77.20.23:2181(CONNECTED) 7] ls /  
[zookeeper]
```

经过验证, zk 节点已经被删除。

ZooKeeper系列之九：ZooKeeper API简介及编程

1) ZooKeeper API 简介

ZooKeeper API 共包含 5 个包，分别为：

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，它我们编程时最常用的类文件。

这个类是 ZooKeeper 客户端库的主要类文件。如果要使用 ZooKeeper 服务，应用程序首先必须创建一个 Zookeeper 实例，这时就需要使用此类。一旦客户端和 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，并且客户端将会周期地向服务器发送心跳来维持会话的连接。只要连接有效，客户端就可以调用 ZooKeeper API 来做相应的处理。

它提供了表 1 所示几类主要方法：

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

2) ZooKeeper API 的使用

这里，笔者通过一个例子来简单介绍，如何使用 ZooKeeper API 编写自己的应用程序，见代码清单 1：

代码清单 1： ZooKeeper API 的使用

```
import java.io.IOException;

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 {
    // 会话超时时间, 设置为与系统默认时间一致
    private static final int SESSION_TIMEOUT=30000;

    // 创建 ZooKeeper 实例
    ZooKeeper zk;

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

    // 初始化 ZooKeeper 实例
    private void createZKInstance() throws IOException
    {
        zk=new ZooKeeper("localhost:2181",demo.SESSION_TIMEOUT,this.wh);
    }

    private void ZKOperations() throws
    IOException,InterruptedException,KeeperException
    {
        System.out.println("/n1. 创建 ZooKeeper 节点 (znode : zoo2, 数据: myData2 ,
权限: OPEN_ACL_UNSAFE , 节点类型: Persistent");
        zk.create("/zoo2","myData2".getBytes(), Ids.OPEN_ACL_UNSAFE,
CreateMode.PERSISTENT);
        System.out.println("/n2. 查看是否创建成功: ");
        System.out.println(new String(zk.getData("/zoo2",false,null)));

        System.out.println("/n3. 修改节点数据 ");
        zk.setData("/zoo2", "shenlan211314".getBytes(), -1);

        System.out.println("/n4. 查看是否修改成功: ");
        System.out.println(new String(zk.getData("/zoo2", false, null)));

        System.out.println("/n5. 删除节点 ");
        zk.delete("/zoo2", -1);

        System.out.println("/n6. 查看节点是否被删除: ");
        System.out.println(" 节点状态: ["+zk.exists("/zoo2", false)+"]");
    }

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

    public static void main(String[] args) throws
    IOException,InterruptedException,KeeperException {
        demo dm=new demo();
        dm.createZKInstance( );
    }
}

```

```
        dm.ZKOperations();
        dm.ZKClose();
    }
```

此类包含两个主要的 ZooKeeper 函数，分别为 createZKInstance（）和 ZKOperations（）。其中 createZKInstance（）函数负责对 ZooKeeper 实例 zk 进行初始化。ZooKeeper 类有两个构造函数，我们这里使用“ZooKeeper（String connectionString, int sessionTimeout, Watcher watcher）”对其进行初始化。因此，我们需要提供初始化所需的，连接字符串信息，会话超时时间，以及一个 watcher 实例。17 行到 23 行代码，是程序所构造的一个 watcher 实例，它能够输出所发生的事件。

ZKOperations（）函数是我们所定义的对节点的一系列操作。它包括：创建 ZooKeeper 节点（33 行到 34 行代码）、查看节点（36 行到 37 行代码）、修改节点数据（39 行到 40 行代码）、查看修改后节点数据（42 行到 43 行代码）、删除节点（45 行到 46 行代码）、查看节点是否存在（48 行到 49 行代码）。另外，需要注意的是：在创建节点的时候，需要提供节点的名称、数据、权限以及节点类型。此外，使用 exists 函数时，如果节点不存在将返回一个 null 值。关于 ZooKeeper API 的更多详细信息，读者可以查看 ZooKeeper 的 API 文档，如下所示：

<http://hadoop.apache.org/zookeeper/docs/r3.3.1/api/index.html>

ZooKeeper系列之十：ZooKeeper的一致性保证及Leader选举

1) 一致性保证

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。另外，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，它具有如下特点：

顺序一致性

客户端的更新顺序与它们被发送的顺序相一致。

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服务器，客户端将看到相同的 *ZooKeeper* 视图。

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。。这个保证将会产生下面两种结果：

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。如果不能够获得返回代码（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。

给予这些一致性保证，*ZooKeeper* 更高级功能的设计与实现将会变得非常容易，例如：leader 选举、队列以及可撤销锁等机制的实现。

2) Leader选举

ZooKeeper 需要在所有的服务（可以理解为服务器）中选举出一个 Leader，然后让这个 Leader 来负责管理集群。此时，集群中的其它服务器则成为此 Leader 的 Follower。并且，当 Leader 故障的时候，需要 *ZooKeeper* 能够快速地在 Follower 中选举出下一个 Leader。这就是 *ZooKeeper* 的 Leader 机制，下面我们将简单介绍在 *ZooKeeper* 中，Leader 选举（Leader Election）是如何实现的。

此操作实现的核心思想是：首先创建一个 EPHEMERAL 目录节点，例如“/election”。然后。每一个 *ZooKeeper* 服务器在此目录下创建一个 SEQUENCE|EPHEMERAL 类型的节点，例如“/election/n_”。在 SEQUENCE 标志下，*ZooKeeper* 将自动地为每一个 *ZooKeeper* 服务器分配一个比前一个分配的序号要大的序号。此时创建节点的 *ZooKeeper* 服务器中拥有最小序号编号的服务器将成为 Leader。

在实际的操作中，还需要保障：当 Leader 服务器发生故障的时候，系统能够快速地选出下一个 *ZooKeeper* 服务器作为 Leader。一个简单的解决方案是，让所有的 follower 监视 leader 所对应的节点。当 Leader 发生故障时，Leader 所对应的临时节点将会自动地被删除，此操作将会触发所有监视 Leader 的服务器的 watch。这样这些服务器将会收到 Leader 故障的消息，并进而进行下一次的 Leader 选举操作。但是，这种操作将会导致“从众效应”的发生，尤其当集群中服务器众多并且带宽延迟比较大的时候，此种情况更为明显。

在 Zookeeper 中，为了避免从众效应的发生，它是这样来实现的：每一个 follower 对 follower 集群中对应的比自己节点序号小一号的节点（也就是所有序号比自己小的节点中的序号最大的节点）设置一个 watch。只有当 follower 所设置的 watch 被触发的时候，它才进行 Leader 选举操作，一般情况下它将成为集群中的下一个 Leader。很明显，此 Leader 选举操作的速度是很快的。因为，每一次 Leader 选举几乎只涉及单个 follower 的操作。