zookeeper 运维管理

# 部署

本章节主要讲述如何部署ZooKeeper，包括以下三部分的内容：

1. 系统环境

2. 集群模式的配置

3. 单机模式的配置

系统环境和集群模式配置这两节内容大体讲述了如何部署一个能够用于生产环境的ZK集群。如果仅仅是想在单机上将ZK运行起来，进行一些开发与测试，那么第三部分或许是你的菜。

## 系统环境

### 平台支持

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **平 台** | 运行client | 运行server | 开发环境 | 生产环境 |
| **GNU/Linux** | √ | √ | √ | √ |
| **Sun Solaris** | √ | √ | √ | √ |
| **FreeBSD** | √ | ⅹ，对nio的支持不好 | √ | √ |
| **Win32** | √ | √ | √ | ⅹ |
| **MacOSX** | √ | √ | √ | ⅹ |

**注**：运行client是指作为客户端，与server进行数据通信，而运行server是指将ZK作为服务器部署运行。

### 软件环境

ZooKeeper Server是一个Java语言实现的分布式协调服务框架，因此需要6或更高版本的JDK支持。集群的机器数量方面，宽泛的讲，其实是任意台机器都可以部署运行的，注意，这里并没有说一定要奇数台机器哦！通常情况下，建议使用3台独立的Linux服务器构成的一个ZK集群。

## 集群模式的配置

为了确保ZooKeeper服务的稳定与可靠性，通常是搭建成一个ZK集群来对外提供服务。关于ZooKeeper，需要明确一个很重要的特性：集群中只要有过半的机器是正常工作的，那么整个集群对外就是可用的（本文下面就用“过半存活即可用”来代替这个特性吧^-^）。正是基于这个特性，建议是将ZK集群的机器数量控制为奇数较为合适。为什么选择奇数台机器，我们可以来看一下，假如是4台机器构成的ZK集群，那么只能够允许集群中有一个机器down掉，因为如果down掉2台，那么只剩下2台机器，显然没有过半。而如果是5台机器的集群，那么就能够对2台机器down掉的情况进行容灾了。

你可以按照以下步骤来配置一个ZK机器，更多详细步骤请查看《[ZooKeeper快速搭建](http://nileader.blog.51cto.com/1381108/795230)》：

**1**. 安装JDK。相关链接：<http://java.sun.com/javase/downloads/index.jsp>

**2**. 设置Java heap 大小。避免内存与磁盘空间的交换，能够大大提升ZK的性能，设置合理的heap大小则能有效避免此类空间交换的触发。在正式发布上线之前，建议是针对使用场景进行一些压力测试，确保正常运行后内存的使用不会触发此类交换。通常在一个物理内存为4G的机器上，最多设置-Xmx为3G。

**3**. 下载安装ZooKeeper，相关链接：<http://zookeeper.apache.org/releases.html>

**4**. 配置文件zoo.cfg。初次使用zookeeper，按照如下这个简单配置即可：

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

本文后续章节会对这些参数进行详细的介绍，这里只是简单说几点：

**A**. 集群中的每台机器都需要感知整个集群是由哪几台机器组成的，在配置文件中，可以按照这样的格式，每行写一个机器配置：server.id=host:port:port. 关于这个id，我们称之为Server ID，标识host机器在集群中的机器序号，在每个ZK机器上，我们需要在数据目录（数据目录就是dataDir参数指定的那个目录）下创建一个myid文件，myid中就是这个Server ID数字。

**B**. 在ZooKeeper的设计中，集群中任意一台机器上的zoo.cfg文件的内容都是一致的。因此最好是用SVN把这个文件管理起来，保证每个机器都能共享到一份相同的配置。

**5**. 关于myid文件。myid文件中只有一个数字，即一个Server ID。例如，server.1 的myid文件内容就是“1”。注意，请确保每个server的myid文件中id数字不同，并且和server.id=host:port:port中的id一致。另外，id的范围是1~255。

**6**. 至此，配置文件基本ok，可以尝试使用如下命令来启动zookeeper了：

1. $ java -cp zookeeper.jar:lib/slf4j-api-1.6.1.jar:lib/slf4j-log4j12-1.6.1.jar:lib/log4j-1.2.15.jar:conf \ org.apache.zookeeper.server.quorum.QuorumPeerMainzoo.cfg

**注意**，不同的ZK版本，依赖的log4j和slf4j版本也是不一样的，请看清楚自己的版本后，再执行上面这个命令。QuorumPeerMain类会启动ZooKeeper Server，同时，JMX MB也会被启动，方便管理员在JMX管理控制台上进行ZK的控制。这里有对ZK JMX的详细介绍：<http://zookeeper.apache.org/doc/r3.4.3/zookeeperJMX.html>.  另外，完全可以有更简便的方式，直接使用%ZK\_HOME%/bin 中的脚本启动即可。

1. ./zkServer.sh start

**7**. 连接ZK host来检验部署是否成功。

**A**. Java语言的话，可以通过运行这个命令来检测：

1. $ java -cp zookeeper.jar:lib/slf4j-api-1.6.1.jar:lib/slf4j-log4j12-1.6.1.jar:lib/log4j-1.2.15.jar:conf:src/java/lib/jline-0.9.94.jar \ org.apache.zookeeper.ZooKeeperMain -server 127.0.0.1:2181

**B**. 如果是C语言的话，方法如下：

1. $ make cli\_st
2. $ make cli\_mt

然后按照的这样的方式连接ZK：$ cli\_mt 127.0.0.1:2181。无论运行哪种客户端，最终都是一个类似于文件系统的命令行操作。

**注意**：除了上面这种检测方法，其实%ZK\_HOME%/bin也有其它脚本，下面这个命令执行后，就进入了zookeeper树状结构的文件系统中。

1. ./zkCli.sh

另外，还有一种方式，能够查看ZK服务器当前状态，如下，这个能够很好的看出目前这个机器的运行情况了：

1. $ echo stat|nc localhost 2181
2. Zookeeper version: 3.4.3-1240972, built on 02/06/2012 10:48 GMT
3. Clients:
4. /127.0.0.1:40293[0](queued=0,recved=1,sent=0)
6. Latency min/avg/max: 1/2/3
7. Received: 4
8. Sent: 3
9. Outstanding: 0
10. Zxid: 0×200000006
11. Mode: leader
12. Node count: 4

## 单机模式的配置

如果你想安装一个ZooKeeper来进行开发测试，通常可以使用单机模式来启动ZK。大体的步骤和上面说的是一样了，除了配置文件会更加简单一些。详细的配置方法可以查看这里：[http://zookeeper.apache.org/doc/r3.4.3/zookeeperStarted.html#sc\_InstallingSingleMode](http://zookeeper.apache.org/doc/r3.4.3/zookeeperStarted.html" \l "sc_InstallingSingleMode" \t "_blank)

# 运 维

本章节主要要讲述如何更好地运维ZooKeepr，大致包含以下几部分内容：

**2.1**. 部署方案的设计

**2.2**. 日常运维

**2.3**. Server的自检恢复

**2.4**. 监控

**2.5**. 日志管理

**2.6**. 数据加载出错

**2.7**. 配置参数详解

**2.8**. 常用的四字命令

**2.9**. 数据文件管理

**2.10**. 注意事项

## 部署方案的设计

我们常说的ZooKeeper能够提供高可用分布式协调服务，是要基于以下两个条件：

**1**. 集群中只有少部分的机器不可用。这里说的不可用是指这些机器或者是本身down掉了，或者是因为网络原因，有一部分机器无法和集群中其它绝大部分的机器通信。例如，如果ZK集群是跨机房部署的，那么有可能一些机器所在的机房被隔离了。

**2**. 正确部署ZK server，有足够的磁盘存储空间以及良好的网络通信环境。

下面将会从集群和单机两个维度来说明，帮助zookeeper管理员尽可能地提高ZK集群的可用性。

### 集群维度

在上面提到的“过半存活即可用”特性中已经讲到过，整个集群如果对外要可用的话，那么集群中必须要有过半的机器是正常工作并且彼此之间能够正常通信。基于这个特性，那么如果想搭建一个能够允许F台机器down掉的集群，那么就要部署一个由2xF+1 台机器构成的ZK集群。因此，一个由3台机器构成的ZK集群，能够在down掉一台机器后依然正常工作，而5台机器的集群，能够对两台机器down掉的情况容灾。**注意**，如果是一个6台机器构成的ZK集群，同样只能够down掉两台机器，因为如果down掉3台，剩下的机器就没有过半了。基于这个原因，ZK集群通常设计部署成奇数台机器。

所以，为了尽可能地提高ZK集群的可用性，应该尽量避免一大批机器同时down掉的风险，换句话说，最好能够为每台机器配置互相独立的硬件环境。举个例子，如果大部分的机器都挂在同一个交换机上，那么这个交换机一旦出现问题，将会对整个集群的服务造成严重的影响。其它类似的还有诸如：供电线路，散热系统等。其实在真正的实践过程中，如果条件允许，通常都建议尝试跨机房部署。毕竟多个机房同时发生故障的机率还是挺小的。

### 单机维度

对于ZK来说，如果在运行过程中，需要和其它应用程序来竞争磁盘，CPU，网络或是内存资源的话，那么整体性能将会大打折扣。

首先来看看磁盘对于ZK性能的影响。客户端对ZK的更新操作都是永久的，不可回退的，也就是说，一旦客户端收到一个来自server操作成功的响应，那么这个变更就永久生效了。为做到这点，ZK会将每次更新操作以事务日志的形式写入磁盘，写入成功后才会给予客户端响应。明白这点之后，你就会明白磁盘的吞吐性能对于ZK的影响了，磁盘写入速度制约着ZK每个更新操作的响应。为了尽量减少ZK在读写磁盘上的性能损失，不仿试试下面说的几点：

**A**、使用单独的磁盘作为事务日志的输出（比如我们这里的ZK集群，使用单独的挂载点用于事务日志的输出）。事务日志的写性能确实对ZK性能，尤其是更新操作的性能影响很大，所以想办法搞到一个单独的磁盘吧！ZK的事务日志输出是一个顺序写文件的过程，本身性能是很高的，所以尽量保证不要和其它随机写的应用程序共享一块磁盘，尽量避免对磁盘的竞争。

**B、**尽量避免内存与磁盘空间的交换。如果希望ZK能够提供完全实时的服务的话，那么基本是不允许操作系统触发此类swap的。因此在分配JVM堆大小的时候一定要非常小心，具体在本文最后的“注意事项”章节中有讲到。

## 日常运维

对zookeeper运维是一个长期积累经验的过程，希望以下几点对广大ZK运维人员有一定的帮助：

### 清理数据目录

上文中提到dataDir目录指定了ZK的数据目录，用于存储ZK的快照文件（snapshot）。另外，默认情况下，ZK的事务日志也会存储在这个目录中。在完成若干次事务日志之后（在ZK中，凡是对数据有更新的操作，比如创建节点，删除节点或是对节点数据内容进行更新等，都会记录事务日志），ZK会触发一次快照（snapshot），将当前server上所有节点的状态以快照文件的形式dump到磁盘上去，即snapshot文件。这里的若干次事务日志是可以配置的，默认是100000，具体参看下文中关于配置参数“snapCount”的介绍。

考虑到ZK运行环境的差异性，以及对于这些历史文件，不同的管理员可能有自己的用途（例如作为数据备份），因此默认ZK是不会自动清理快照和事务日志，需要交给管理员自己来处理。这里是我们用的清理方法，保留最新的66个文件，将它写到crontab中，每天凌晨2点触发一次：

1. #!/bin/bash
3. #snapshot file dir
4. dataDir=/home/yinshi.nc/test/zk\_data/version-2
5. #tran log dir
6. dataLogDir=/home/yinshi.nc/test/zk\_log/version-2
7. #zk log dir
8. logDir=/home/yinshi.nc/test/logs
9. #Leave 66 files
10. count=66
11. count=$[$count+1]
12. ls -t $dataLogDir/log.\* | tail -n +$count | xargs rm -f
13. ls -t $dataDir/snapshot.\* | tail -n +$count | xargs rm -f
14. ls -t $logDir/zookeeper.log.\* | tail -n +$count | xargs rm -f
16. #find /home/yinshi.nc/taokeeper/zk\_data/version-2 -name “snap\*” -mtime +1 | xargs rm -f
17. #find /home/yinshi.nc/taokeeper/zk\_logs/version-2 -name “log\*” -mtime +1 | xargs rm -f
18. #find /home/yinshi.nc/taokeeper/logs/ -name “zookeeper.log.\*” -mtime +1 | xargs rm –f

其实，仅管ZK没有自动帮我们清理历史文件，但是它的还是提供了一个叫PurgeTxnLog的 工具类，实现了一种简单的历史文件清理策略，可以在这里看一下他的使用方法：<http://zookeeper.apache.org/doc/r3.4.3/api/index.html> 简单使用如下：

1. java -cp zookeeper.jar:lib/slf4j-api-1.6.1.jar:lib/slf4j-log4j12-1.6.1.jar:lib/log4j-1.2.15.jar:conf org.apache.zookeeper.server.PurgeTxnLog**<dataDir><snapDir>** -n **<count>**

最后一个参数表示希望保留的历史文件个数，注意，count必须是大于3的整数。可以把这句命令写成一个定时任务，以便每天定时执行清理。

**注意**： 从3.4.0版本开始， zookeeper提供了自己清理历史文件的功能了，相关的配置参数是autopurge.snapRetainCount和autopurge.purgeInterval，在本文后面会具体说明。更多关于zookeeper的日志清理，可以阅读这个文章[《ZooKeeper日志清理》](http://nileader.blog.51cto.com/1381108/932156)。

### ZK程序日志

这里说两点，ZK默认是没有向ROLLINGFILE文件输出程序运行时日志的，需要我们自己在conf/log4j.properties中配置日志路径。另外，没有特殊要求的话，日志级别设置为INFO或以上，我曾经测试过，日志级别设置为DEBUG的话，性能影响很大！

## Server的自检恢复

ZK运行过程中，如果出现一些无法处理的异常，会直接退出进程，也就是所谓的快速失败（fail fast）模式。在上文中有提到，“过半存活即可用”的特性使得集群中少数机器down掉后，整个集群还是可以对外正常提供服务的。另外，这些down掉的机器重启之后，能够自动加入到集群中，并且自动和集群中其它机器进行状态同步（主要就是从Leader那里同步最新的数据），从而达到自我恢复的目的。

因此，我们很容易就可以想到，是否可以借助一些工具来自动完成机器的状态检测与重启工作。回答是肯定的，这里推荐两个工具： Daemontools(<http://cr.yp.to/daemontools.html>) 和 SMF（<http://en.wikipedia.org/wiki/Service_Management_Facility>），能够帮助你监控ZK进程，一旦进程退出后，能够自动重启进程，从而使down掉的机器能够重新加入到集群中去~

## 监控

有几种方法：

**1、** ZK提供一些简单但是功能强大的4字命令，通过对这些4字命令的返回内容进行解析，可以获取不少关于ZK运行时的信息。

**2**、用jmx也能够获取一些运行时信息，详细可以查看这里：<http://zookeeper.apache.org/doc/r3.4.3/zookeeperJMX.html>

**3**、淘宝网已经实现的一个ZooKeeper监控——TaoKeeper，已开源，在这里：<http://rdc.taobao.com/team/jm/archives/1450>，主要功能如下:

**A**、机器CPU/MEM/LOAD的监控

**B**、ZK日志目录所在磁盘空间监控

**C**、单机连接数的峰值报警

**D**、单机Watcher数的峰值报警

**E**、节点自检

**F**、ZK运行时信息展示

## 日志管理

ZK使用log4j作为日志系统，conf目录中有一份默认的log4j配置文件，注意，这个配置文件中还没有开启ROLLINGFILE文件输出，配置下即可。其它关于log4j的详细介绍，可以移步到log4j的官网：[http://logging.apache.org/log4j/1.2/manual.html#defaultInit](http://logging.apache.org/log4j/1.2/manual.html" \l "defaultInit)

## 加载数据出错

ZK在启动的过程中，首先会根据事务日志中的事务日志记录，从本地磁盘加载最后一次提交时候的快照数据，如果读取事务日志出错或是其它问题（通常在日志中可以看到一些IO异常），将导致server将无法启动。碰到类似于这种数据文件出错导致无法启动服务器的情况，一般按照如下顺序来恢复：

**1**、确认集群中其它机器是否正常工作，方法是使用“stat”这个命令来检查：echo stat|nc ip 2181

**2**、如果确认其它机器是正常工作的（这里要说明下，所谓正常工作还是指集群中有过半机器可用），那么可以开始删除本机的一些数据了，删除$dataDir/version-2和$dataLogDir/version-2 两个目录下的所有文件。

重启server。重启之后，这个机器就会从Leader那里同步到最新数据，然后重新加入到集群中提供服务。

## 配置参数详解(主要是%ZOOKEEPER\_HOME%/conf/zoo.cfg文件)

|  |  |
| --- | --- |
| 参数名 | 说明 |
| clientPort | 客户端连接server的端口，即对外服务端口，一般设置为2181吧。 |
| dataDir | 存储快照文件snapshot的目录。默认情况下，事务日志也会存储在这里。建议同时配置参数dataLogDir, 事务日志的写性能直接影响zk性能。 |
| tickTime | ZK中的一个时间单元。ZK中所有时间都是以这个时间单元为基础，进行整数倍配置的。例如，session的最小超时时间是2\*tickTime。 |
| dataLogDir | 事务日志输出目录。尽量给事务日志的输出配置单独的磁盘或是挂载点，这将极大的提升ZK性能。 （No Java system property） |
| globalOutstandingLimit | 最大请求堆积数。默认是1000。ZK运行的时候， 尽管server已经没有空闲来处理更多的客户端请求了，但是还是允许客户端将请求提交到服务器上来，以提高吞吐性能。当然，为了防止Server内存溢出，这个请求堆积数还是需要限制下的。 (Java system property:?**zookeeper.globalOutstandingLimit.**) |
| preAllocSize | 预先开辟磁盘空间，用于后续写入事务日志。默认是64M，每个事务日志大小就是64M。如果ZK的快照频率较大的话，建议适当减小这个参数。(Java system property:**zookeeper.preAllocSize**) |
| snapCount | 每进行snapCount次事务日志输出后，触发一次快照(snapshot), 此时，ZK会生成一个snapshot.\*文件，同时创建一个新的事务日志文件log.\*。默认是100000.（真正的代码实现中，会进行一定的随机数处理，以避免所有服务器在同一时间进行快照而影响性能）(Java system property:**zookeeper.snapCount**) |
| traceFile | 用于记录所有请求的log，一般调试过程中可以使用，但是生产环境不建议使用，会严重影响性能。(Java system property:**requestTraceFile**) |
| maxClientCnxns | 单个客户端与单台服务器之间的连接数的限制，是ip级别的，默认是60，如果设置为0，那么表明不作任何限制。请注意这个限制的使用范围，仅仅是单台客户端机器与单台ZK服务器之间的连接数限制，不是针对指定客户端IP，也不是ZK集群的连接数限制，也不是单台ZK对所有客户端的连接数限制。指定客户端IP的限制策略，这里有一个patch，可以尝试一下：<http://rdc.taobao.com/team/jm/archives/1334>（No Java system property） |
| clientPortAddress | 对于多网卡的机器，可以为每个IP指定不同的监听端口。默认情况是所有IP都监听**clientPort**指定的端口。**New in 3.3.0** |
| minSessionTimeout  maxSessionTimeout | Session超时时间限制，如果客户端设置的超时时间不在这个范围，那么会被强制设置为最大或最小时间。默认的Session超时时间是在2 \***tickTime ~ 20 \* tickTime这个范围** **New in 3.3.0** |
| fsync.warningthresholdms | 事务日志输出时，如果调用fsync方法超过指定的超时时间，那么会在日志中输出警告信息。默认是1000ms。(Java system property:**fsync.warningthresholdms**) **New in 3.3.4** |
| autopurge.purgeInterval | 在上文中已经提到，3.4.0及之后版本，ZK提供了自动清理事务日志和快照文件的功能，这个参数指定了清理频率，单位是小时，需要配置一个1或更大的整数，默认是0，表示不开启自动清理功能。(No Java system property) **New in 3.4.0** |
| autopurge.snapRetainCount | 这个参数和上面的参数搭配使用，这个参数指定了需要保留的文件数目。默认是保留3个。(No Java system property) **New in 3.4.0** |
| electionAlg | 在之前的版本中， 这个参数配置是允许我们选择leader选举算法，但是由于在以后的版本中，只会留下一种“TCP-based version of fast leader election”算法，所以这个参数目前看来没有用了，这里也不详细展开说了。(No Java system property) |
| initLimit | 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F在**initLimit**时间内完成这个工作。通常情况下，我们不用太在意这个参数的设置。如果ZK集群的数据量确实很大了，F在启动的时候，从Leader上同步数据的时间也会相应变长，因此在这种情况下，有必要适当调大这个参数了。(No Java system property) |
| syncLimit | 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。如果L发出心跳包在syncLimit之后，还没有从F那里收到响应，那么就认为这个F已经不在线了。注意：不要把这个参数设置得过大，否则可能会掩盖一些问题。(No Java system property) |
| leaderServes | 默认情况下，Leader是会接受客户端连接，并提供正常的读写服务。但是，如果你想让Leader专注于集群中机器的协调，那么可以将这个参数设置为no，这样一来，会大大提高写操作的性能。(Java system property: zookeeper.**leaderServes**)。 |
| server.x=[hostname]:nnnnn[:nnnnn] | 这里的x是一个数字，与myid文件中的id是一致的。右边可以配置两个端口，第一个端口用于F和L之间的数据同步和其它通信，第二个端口用于Leader选举过程中投票通信。 (No Java system property) |
| group.x=nnnnn[:nnnnn]weight.x=nnnnn | 对机器分组和权重设置，可以 [参见这里](http://zookeeper.apache.org/doc/r3.4.3/zookeeperHierarchicalQuorums.html)(No Java system property) |
| cnxTimeout | Leader选举过程中，打开一次连接的超时时间，默认是5s。(Java system property: zookeeper.**cnxTimeout**) |
| zookeeper.DigestAuthenticationProvider .superDigest | ZK权限设置相关，具体参见[《](http://nileader.blog.51cto.com/1381108/930635)**[使用super身份对有权限的节点进行操作](http://nileader.blog.51cto.com/1381108/930635)**[》](http://nileader.blog.51cto.com/1381108/930635) 和 [《**ZooKeeper权限控制**》](http://nileader.blog.51cto.com/1381108/795525) |
| skipACL | 对所有客户端请求都不作ACL检查。如果之前节点上设置有权限限制，一旦服务器上打开这个开头，那么也将失效。(Java system property:**zookeeper.skipACL**) |
| forceSync | 这个参数确定了是否需要在事务日志提交的时候调用[FileChannel](http://nileader.blog.51cto.com/%5C/java%5C/jdk1.6.0_22%5C/jre%5C/lib%5C/rt.jar%3Cjava.nio.channels(FileChannel.class%E2%98%83FileChannel).force来保证数据完全同步到磁盘。(Java system property:**zookeeper.forceSync**) |
| jute.maxbuffer | 每个节点最大数据量，是默认是1M。这个限制必须在server和client端都进行设置才会生效。(Java system property:**jute.maxbuffer**) |

## 常用的四字命令

|  |  |
| --- | --- |
| 参数名 | 说明 |
| conf | 输出server的详细配置信息。**New in 3.3.0**   1. $**>**echo conf|nc localhost 2181 2. clientPort=2181 3. dataDir=/home/test/taokeeper/zk\_data/version-2 4. dataLogDir=/test/admin/taokeeper/zk\_log/version-2 5. tickTime=2000 6. maxClientCnxns=1000 7. minSessionTimeout=4000 8. maxSessionTimeout=40000 9. serverId=2 10. initLimit=10 11. syncLimit=5 12. electionAlg=3 13. electionPort=3888 14. quorumPort=2888 15. peerType=0 |
| cons | 输出指定server上所有客户端连接的详细信息，包括客户端IP，会话ID等。 **New in 3.3.0**类似于这样的信息：   1. $**>**echo cons|nc localhost 2181 2. /1.2.3.4:43527[1](queued=0,recved=152802, 3. sent=152806,sid=0x2389e662b98c424,lop=PING, 4. est=1350385542196,to=6000, 5. lcxid=0×114,lzxid=0xffffffffffffffff,lresp=1350690663308,llat=0,minlat=0,avglat=0,maxlat=483) 6. …… |
| crst | 功能性命令。重置所有连接的**统计**信息。**New in 3.3.0** |
| dump | 这个命令针对Leader执行，用于输出所有等待队列中的会话和临时节点的信息。 |
| envi | 用于输出server的环境变量。包括操作系统环境和Java环境。 |
| ruok | 用于测试server是否处于无错状态。如果正常，则返回“imok”,否则没有任何响应。 注意：ruok不是一个特别有用的命令，它不能反映一个server是否处于正常工作。“stat”命令更靠谱。 |
| stat | 输出server简要状态和连接的客户端信息。 |
| srvr | 和stat类似，**New in 3.3.0**   1. $**>**echo stat|nc localhost 2181 2. Zookeeper version: 3.3.5-1301095, built on 03/15/2012 19:48 GMT 3. Clients: 4. /10.2.3.4:59179[1](queued=0,recved=44845,sent=44845) 6. Latency min/avg/max: 0/0/1036 7. Received: 2274602238 8. Sent: 2277795620 9. Outstanding: 0 10. Zxid: 0xa1b3503dd 11. Mode: leader 12. Node count: 37473 13. $**>**echo srvr|nc localhost 2181 14. Zookeeper version: 3.3.5-1301095, built on 03/15/2012 19:48 GMT 15. Latency min/avg/max: 0/0/980 16. Received: 2592698547 17. Sent: 2597713974 18. Outstanding: 0 19. Zxid: 0xa1b356b5b 20. Mode: follower 21. Node count: 37473 |
| srst | 重置server的统计信息。 |
| wchs | 列出所有watcher信息概要信息，数量等：**New in 3.3.0**   1. $**>**echo wchs|nc localhost 2181 2. 3890 connections watching 537 paths 3. Total watches:6909 |
| wchc | 列出所有watcher信息，以watcher的session为归组单元排列，列出该会话订阅了哪些path：**New in 3.3.0**   1. $**>**echo wchc|nc localhost 2181 2. 0x2389e662b97917f 3. /mytest/test/path1/node1 4. 0x3389e65c83cd790 5. /mytest/test/path1/node2 6. 0x1389e65c7ef6313 7. /mytest/test/path1/node3 8. /mytest/test/path1/node1 |
| wchp | 列出所有watcher信息，以watcher的path为归组单元排列，列出该path被哪些会话订阅着：**New in 3.3.0**   1. $**>**echo wchp|nc localhost 2181 2. /mytest/test/path1/node 3. 0x1389e65c7eea4f5 4. 0x1389e65c7ee2f68 5. /mytest/test/path1/node2 6. 0x2389e662b967c29 7. /mytest/test/path1/node3 8. 0x3389e65c83dd2e0 9. 0x1389e65c7f0c37c 10. 0x1389e65c7f0c364   **注意**，wchc和wchp这两个命令执行的输出结果都是针对session的，对于运维人员来说可视化效果并不理想，可以尝试将cons命令执行输出的信息整合起来，就可以用客户端IP来代替会话ID了，具体可以看这个实现：[http://rdc.taobao.com/team/jm/archives/1450](http://rdc.taobao.com/team/jm/archives/1450" \t "_blank) |
| mntr | 输出一些ZK运行时信息，通过对这些返回结果的解析，可以达到监控的效果。**New in 3.4.0**   1. $ echo mntr | nc localhost 2185 2. zk\_version 3.4.0 3. zk\_avg\_latency 0 4. zk\_max\_latency 0 5. zk\_min\_latency 0 6. zk\_packets\_received 70 7. zk\_packets\_sent 69 8. zk\_outstanding\_requests 0 9. zk\_server\_state leader 10. zk\_znode\_count 4 11. zk\_watch\_count 0 12. zk\_ephemerals\_count 0 13. zk\_approximate\_data\_size 27 14. zk\_followers 4 – only exposed by the Leader 15. zk\_synced\_followers 4 – only exposed by the Leader 16. zk\_pending\_syncs 0 – only exposed by the Leader 17. zk\_open\_file\_descriptor\_count 23 – only available on Unix platforms 18. zk\_max\_file\_descriptor\_count 1024 – only available on Unix platforms |

## 数据文件管理

默认情况下，ZK的数据文件和事务日志是保存在同一个目录中，建议是将事务日志存储到单独的磁盘上。

### 数据目录

ZK的数据目录包含两类文件：

**A**、myid – 这个文件只包含一个数字，和server id对应。

**B**、snapshot. - 按zxid先后顺序的生成的数据快照。

集群中的每台ZK server都会有一个用于惟一标识自己的id，有两个地方会使用到这个id：myid文件和zoo.cfg文件中。myid文件存储在dataDir目录中，指定了当前server的server id。在zoo.cfg文件中，根据server id，配置了每个server的ip和相应端口。Zookeeper启动的时候，读取myid文件中的server id，然后去zoo.cfg 中查找对应的配置。

zookeeper在进行数据快照过程中，会生成 snapshot文件，存储在dataDir目录中。文件后缀是zxid，也就是事务id。（这个zxid代表了zk触发快照那个瞬间，提交的最后一个事务id）。注意，一个快照文件中的数据内容和提交第zxid个事务时内存中数据近似相同。仅管如此，由于更新操作的幂等性，ZK还是能够从快照文件中恢复数据。数据恢复过程中，将事务日志和快照文件中的数据对应起来，就能够恢复最后一次更新后的数据了。

**2.9.2事务日志目录**

dataLogDir目录是ZK的事务日志目录，包含了所有ZK的事务日志。正常运行过程中，针对所有更新操作，在返回客户端“更新成功”的响应前，ZK会确保已经将本次更新操作的事务日志写到磁盘上，只有这样，整个更新操作才会生效。每触发一次数据快照，就会生成一个新的事务日志。事务日志的文件名是log.，zxid是写入这个文件的第一个事务id。

**2.9.3文件管理**

不同的zookeeper server生成的snapshot文件和事务日志文件的格式都是一致的（无论是什么环境，或是什么样的zoo.cfg 配置）。因此，如果某一天生产环境中出现一些古怪的问题，你就可以把这些文件下载到开发环境的zookeeper中加载起来，便于调试发现问题，而不会影响生产运行。另外，使用这些较旧的snapshot和事务日志，我们还能够方便的让ZK回滚到一个历史状态。

另外，ZK提供的工具类LogFormatter能够帮助可视化ZK的事务日志，帮助我们排查问题，关于事务日志的可以化，请查看这个文章[《可视化zookeeper的事务日志》](http://nileader.blog.51cto.com/1381108/926753).

需要注意的一点是，zookeeper在运行过程中，不断地生成snapshot文件和事务日志，但是不会自动清理它们，需要管理员来处理。（ZK本身只需要使用最新的snapshot和事务日志即可）关于如何清理文件，上面章节“日常运维”有提到。

## 注意事项

### 保持Server地址列表一致

**A**、客户端使用的server地址列表必须和集群所有server的地址列表一致。（如果客户端配置了集群机器列表的子集的话，也是没有问题的，只是少了客户端的容灾。）

**B**、集群中每个server的zoo.cfg中配置机器列表必须一致。

### 独立的事务日志输出

对于每个更新操作，ZK都会在确保事务日志已经落盘后，才会返回客户端响应。因此事务日志的输出性能在很大程度上影响ZK的整体吞吐性能。强烈建议是给事务日志的输出分配一个单独的磁盘。

### 配置合理的JVM堆大小

确保设置一个合理的JVM堆大小，如果设置太大，会让内存与磁盘进行交换，这将使ZK的性能大打折扣。例如一个4G内存的机器的，如果你把JVM的堆大小设置为4G或更大，那么会使频繁发生内存与磁盘空间的交换，通常设置成3G就可以了。当然，为了获得一个最好的堆大小值，在特定的使用场景下进行一些压力测试。