ZooKeeper Administrator's Guide 中文版

存档 zookeeper

> 本文档主要介绍ZooKeeper部署及日常管理
> 转载请注明出处：cloudnoter.com

安装部署

本章节包含与ZooKeeper部署有关的内容，具体来说包含下面三部分内容：

- 系统软硬件需求
- 集群部署（安装）
- 单机开发环境部署（安装）

前两部分主要介绍如何在数据中心等生产环境上安装部署ZooKeeper，第三部分则介绍如何在非生产环境上（如为了评估、测试、开发等目的）安装部署ZooKeeper。

系统软硬件需求

支持的OS平台

ZooKeeper框架由多个组件组成，有的组件支持全部平台，而还有一些组件只支持部分平台，详细支持情况如下：

- **Client：**它是一个 `Java` 客户端连接库，上层应用系统通过它连接至ZooKeeper集群。
- **Server：**它是运行在ZooKeeper集群节点上的一个 `Java` 后台服务程序。
- **Native Client：**它是一个用 `C` 语言实现的客户端连接库，其与 `Java` 客户端库一样，上层应用（非 `Java` 实现）通过它连接至ZooKeeper集群。
- **Contrib：**它是指多个可选的扩展组件。

操作系统	Client	Server	Native Client	Contrib
GNU/Linux	D + P	D + P	D + P	D + P
Solaris	D + P	D + P	/	/
FreeBSD	D + P	D + P	/	/
Windows	D + P	D + P	/	/
Mac OS X	D	D	/	/

> **D：支持开发环境， P：支持生成环境， /：不支持任何环境**

上表中未显式注明支持的组件在相应平台上可能不能正常运行。虽然ZooKeeper社区会尽量修复在未支持平台上发现的BUG，但并无法保证会修复全部BUG。

软件要求

ZooKeeper需要运行在JDK6或以上版本中。若ZooKeeper以集群模式部署，则推荐的节点数至少为3，同时建议部署在独立的服务器上。在Yahoo!，ZooKeeper通常部署在运行RHEL系统的服务器上（服务器配置：双核CPU、2G内存、80G容量IDE硬盘）。

集群部署（安装）

为了保证ZooKeeper服务的可靠性，您应该以集群模式部署ZooKeeper服务。只要半数以上的集群节点在线，服务将是可用的。因为ZooKeeper需要半数以上节点同意才能选举出Leader，所以建议ZooKeeper集群的节点数为奇数个。举个例子，对于有四个节点的集群只能应付一个节点宕机的异常，如果有两个节点宕机，则剩下两个节点未达到法定的半数以上选票，ZooKeeper服务将变为不可用。而如果集群有五个节点，则集群就可以应付二个节点宕机的异常。

> **提示：**
> 正如《ZooKeeper快速入门》文档中所提到的，至少需要三个节点的ZooKeeper集群才具备容灾特性，因此我们强烈建议集群节点数为奇数。
> 通常情况下，生产环境下，集群节点数只需要三个。但如果为了在服务维护场景下也保证最大的可靠性，您也许会部署五个节点的集群。原因很简单，如果集群节点为三个，当你对其中一个节点进行维护操作，将很有可能因维护操作导致集群异常。而如果集群节点为5个，那你可以直接将维护节点下线，此时集群仍然可正常提供服务（就算四个节点中的任意一个突然宕机）。
> 您的冗余措施应该包括生产环境的各个方面。如果你部署三个节点的ZooKeeper集群，但你却将这三个节点都连接至同一个网络交换机，那么当交换机宕掉时，你的集群服务也一样是不可用的。

关于如何配置服务器使其成为集群中的一个成员节点的详细步骤如下（每个节点的操作类似）：

- 1.安装JDK，你可以使用本地包管理系统安装，也可以从JDK官网下载
- 2.设置Java堆相关参数，这对于避免内存swap来说非常重要。因为频繁的swap将严重降低ZooKeeper集群的性能。您可以通过使用压力负载测试来决定一个合适的值，确保该值刚好低于触发swap的阈值。保守的做法是：当节点拥有4G的内存，则设置-xmx=3G。
- 3.安装ZooKeeper，您可以从官网下载：http://zookeeper.apache.org/releases.html
- 4.创建一个配置文件，这个文件可以随便命名，您可以先使用如下设置参数：

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

> 您可以在配置参数章节中找到上面这些参数及其他参数的解释说明。这里简要说一下：集群中的每个节点都需要知道集群中其它节点成员的连接信息。通过上述配置文件中格式为 `server.id=host:port:port` 的若干行配置信息您就可以实现这个目标。 `host` 与 `port` 意思很简单明了，不多作说明。而 `server.${id}` 代表节点ID，你需要为每个节点创建一个名为 `myid` 的文件，该文件存放于参数 `dataDir` 所指向的目录下。

- 5. `myid` 文件的内容只有一行，其值为配置参数 `server.${id}` 中 `${id}` 的实际值。即服务器1对应的 `myid` 文件的内容为1，这个值在集群中必须保证其唯一性，同时必须处于[1, 255]之间。
- 6.如果你已经创建好配置文件（如zoo.cfg），你就可以启动ZooKeeper服务：

```
$ 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.QuorumPeerMain zoo.cfg
```

> QuorumPeerMain启动一个ZooKeeper服务，JMX管理bean也同时被注册，通过这些JMX管理bean，你就可以在JMX管理控制台上对ZooKeeper服务进行监控管理。ZooKeeper的JMX文档有更详细的介绍信息。同时，你也可以在 `$ZOOKEEPER_HOME/bin` 目录下找到ZooKeeper的启动脚本 `zkServer.sh` 。

- 7.接下来你就可以连接至ZooKeeper节点来测试部署是否成功。
  在 `Java` 环境下，你可以运行下面的命令连接至已启动的ZooKeeper服务节点并执行一些简单的操作

```
$ bin/zkCli.sh -server 127.0.0.1:2181
```

单机开发环境部署介绍

如果你想部署ZooKeeper以便于开发测试，那你可以使用单机部署模式。然后安装Java或C客户端连接库，同时在配置文件中将服务器信息与开发机绑定。
具体安装部署步骤也集群部署模式类似，唯一不同的是zoo.cfg配置文件更简单一些。你可以从《ZooKeeper快速入门》文档中的相关章节获取详细的操作步骤。
关于安装客户端连接库的相关信息，你可以从ZooKeeper Programmer's Guide文件的Bindings章节中获取。

维护管理

这部分包含ZooKeeper运行与维护相关的信息，其包含如下几个主题：

ZooKeeper集群部署规划

ZooKeeper可靠性依赖于两个基本的假设：

- 一个集群中只会有少数服务器会出错，这里的出错是指宕机或网络异常而使得服务与集群中的多数服务器失去联系。
- 已部署的机器可以正常运行，所谓正常运行是指所有代码可以正确的执行，有适当且一致的工作时钟、存储、网络。

为了最大可能的保证上述两个前提假设能够成立，这里有几个点需要考虑。一些是关于跨服务器（节点之间）需求的，而另一些是关于集群中每个节点服务器的考虑。

跨机器要求

为了启用ZooKeeper服务，集群中半数以上的节点需要能够相互通信。为了创建一个可以支撑F个节点异常的集群，你需要部署2xF+1个节点。即一个包含三个节点的集群可以应对一个节点异常的情况，一个包含五个节点的集群则可以应对二个节点异常的情况，依此类推。需要注意的是，一个包含六个节点的集群也只能应对二个节点失败的情况，因为三个节点并未超过半数。因此，ZooKeeper集群中的节点数通常为奇数。

为了实现最大限度的容灾能力，你应该尽力使集群节点发生异常时不影响其它节点（即保持节点部署上尽可能的独立）。举个例子，当大部分的机器共享同一个交换机，若这个交换机挂了将使关联的服务都下线。共享电源电路、冷却系统等也是同样的道理（有同样的单点问题）。

单机要求

如果ZooKeeper服务需要与其它应用竞争存储、CPU、网络、内存等资源时，其性能将显著下降。机器必须为ZooKeeper服务提供持久化保障，因为ZooKeeper需要先使用存储设备来保存变更日志，然后才允许变更操作提交。如果你想确保ZooKeeper操作不会因存储而挂起，那你应该意识到这个依赖关系并重视起来。这里有几个事情可以最小化这类问题所带来的消极影响。

- ZooKeeper的事务日志必须存储在专用的设备上（一个专用分区是不够的）ZooKeeper以串行的方式写入日志。如果与其它进程共享日志存储设备将导致磁盘寻址与IO竞争，这最终将导致几秒的时延。
- 不要将Zookeeper放置在可能引起swap的环境。为了保证Zookeeper的实时性，它几乎不允许swap。因此，请确保分配给ZooKeeper的最大堆内存大小不能大于ZooKeeper可用的真实的内存大小。关于这一点的更多信息，请参数后续章节 Things to Avoid。

Provisioning

无

ZooKeeper的优势与局限

无

管理

无

维护

ZooKeeper的长期维护需求几乎没有，然而你仍然需要注意如下几件事情：

持续的数据目录清理

ZooKeeper的数据目录包含集群上特殊存储结点znode数据的持久化拷贝文件。文件中包含快照与事务日志文件。znode上数据的变更将同时会被追加至事务日志中。当这些日志不断增长时，所有znode的当前状态的快照文件将被写回文件系统。这个快照将取代之前的日志。

ZooKeeper服务器默认情况下不会移除旧的快照与日志文件，删除快照与日志这项工作是管理员的责任。因为每个服务环境是不一样的，所以管理这些快照与文件的需求也是不同一。 `PurgeTxnLog` 工具实现了一个简单的保留策略，管理员可以使用这个工具，API docs 中包含有详细的使用说明。下面的例子的作用为：保留最近 `<count>` 个快照及相关的日志，其它的快照及相关日志则删除。 `<count>` 的值通常要大于3（虽然不是必须的，但提供3个备份时将极大降低日志损坏的可能性）。您可以在ZooKeeper服务器上运行 `cron` 脚本来每天清除过期日志。

```
$ 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>
```

在V3.4.0版本中引入的自动清除快照及相关事务日志的特性可以通过配置参数 `autopurge.snapRetainCount` 与 `autopurge.purgeInterval` 来启用。更多的使用见后续的配置参数（高级配置）章节。

调试日志清理 (log4j)

通过本文档的日志章节可知，ZooKeeper将使用Log4j内置的日志滚动覆盖功能最多生成N个日志文件。Log4j的配置示例可在发行版本tar包的 `conf/log4j.properties` 文件中找到。

Supervision

你需要一个监督进程来管理ZooKeeper服务进程（本质上为一个Java进程）。ZooKeeper服务具有"快速失败"的特性，即只要服务发生不可恢复的错误，则ZK进程直接退出。因为ZK集群是高可用的，所有集群中的一个ZK服务虽然异常退出了，但整个集群仍然是可用的，仍然可对外提供服务。同时由于集群可以自行恢复特性，一旦此前异常退出的服务器重启后，它将自动重新加入集群中而不需要任何的人工干预。

因此，我们就需要一个类似 `daemontools或SMF` 的工具来管理ZK服务，确保进程异常退出后可以自动被重启并快速重新加入集群。

Monitoring

ZK服务可以使用如下两种方式进行监控：

- ZK提供的4个字母的命令工具
- JMX（详见ZK官网提供的JMX相关文档）

日志

ZK使用V1.2版本的Log4j作为其日志组件。ZK默认的日志配置文件位于发行版本的conf目录下。Log4j需要 `log4j.properties` 文件，这个文件要么位于ZK运行时所在的目录，要么位于classpath目录。

更多的信息见 Log4j Default Initialization Procedure 。

疑难解答

文件损坏导致服务器无法启动问题
ZK服务器可能会因无法正常读取数据导致启动失败。出现这个问题可能是因为ZK服务器上的事务日志文件损坏了。你可以从ZK的log4j日志中看到一些与ZK数据加载相关的 `IOException` 异常。在这种情况下，首先确保集群中的其他服务器能正常工作。可以使用 `stat` 命令查看它们是否处于健康状态。当你已经确认集群中其它服务器都在运行中，你可以进一步清理异常中断的服务器上的数据库：删除 `$data_dir/version-2` 目录与 `$data_log_dir/version-2` 目录下的内容，然后重启服务即可。

配置参数（高级配置）

ZooKeeper的行为是由其配置文件 `$ZK_HOME\conf\zoo.cfg` 来控制的。由于这个配置文件是设计好的，因此当ZK集群中每个成员的部署结构（磁盘路径等）都一样时，这个配置文件可以被所有成员共用。如果成员服务器使用不同的配置文件，一定要注意保证各个服务器的配置文件的服务器列表信息是匹配的。

最简（小）配置

下面是部署一个ZK集群时，配置文件必须要设置的参数信息（最简配置）：
clientPort
客户端连接监听端口，即客户端发起连接请求时，都会尝试连接至ZK服务器的该端口。
dataDir
ZK内存数据库快照的保存位置，除非另有规定，否则事务日志也会保存至该位置。

> 一定要注意事务日志的存储位置。一个专用的事务日志设备是保证高性能的关键。如果将事务日志存放于IO比较频繁的设备将产生不利的性能效果。

tickTime

计时时间片长度（单位：毫秒），它是ZK中所有与时间有关参数的基本时间单元（即当某参数的值为X，则其语义可这样理解：某参数的值为tickTime的X倍）。它通常用于日常的心跳发送周期、超时时间等。
比如，当 `tickTime=2000,minSessionTimeout=2` ，
则会话超时的最小值为： `tickTime*minSessionTimeout=4000ms` 。

高级配置

本节所介绍的配置参数是可选的。你可以使用它们进一步调整ZK集群的行为。有些参数也可以通过形如 `zookeeper.keyword` 之类的Java系统属性的方式来配置。下面的参数详细介绍中有标明哪些参数可以通过Java系统属性方式配置。

dataLogDir

这个选项会让机器将事务日志直接写到dataLogDir所指定的位置，而不再是默认的dataDir。这将允许使用专用的日志设备，同时可有效避免快照操作与日志操作所带来的IO竞争。

> 使用专用的日志设备将极大的影响系统吞吐量与系统时延。强烈推荐使用专用的日志设备，并将dataLogDir指向该专用设备的一个目录，然后也要确保dataDir参数所指向的目录未指向专用的日志设备。

globalOutstandingLimit
(Java系统属性: zookeeper.globalOutstandingLimit)

客户端提交的请求数通过会比ZK的处理速度快，特别是存在大量客户端连接的情况。为了防止因队列中的请求数多过导致ZK运行过程中出现内存溢出问题，ZK将控制客户端的请求数，这样将可以保证系统请求数不会多于 `globalOutstandingLimit` 参数所指定的值，系统默认值为1000。

preAllocSize
(Java系统属性: zookeeper.preAllocSize)

To avoid seeks ZooKeeper allocates space in the transaction log file in blocks of preAllocSize kilobytes. The default block size is 64M. One reason for changing the size of the blocks is to reduce the block size if snapshots are taken more often. (Also, see snapCount).

snapCount
(Java系统属性: zookeeper.snapCount)

ZK将事务写入事务日志文件中。当 `snapCount` 个事务被写入日志文件后，ZK将创建一个快照，然后一个新的事务日志文件会被创建。该参数的默认值为:100000。

maxClientCnxns

限制同一个客户端连接（通过IP标识唯一性）至一个ZK集群节点的并发连接数（Socket级别）。这个参数主要是用来防止某些类型的DoS攻击，包括 `file-descriptor exhaustion` 。该参数默认值为60，如果设置为0则意味着关闭并发控制限制功能。

clientPortAddress

V3.3.0引入: 客户端连接监听地址（IPv4,IPv6,主机名），这个参数是可选的。默认情况下，所有连接至 `clientPort` 参数指定端口号的连接都会被接受，不管这些连接是来自于哪个网络接口。

minSessionTimeout

V3.3.0引入: 最小会话超时时间（毫秒），在会话期间服务器将允许来自客户端的协商。该参数默认值为2倍的tickTime。

maxSessionTimeout

V3.3.0引入: 最小会话超时时间（毫秒），在会话期间服务器将允许来自客户端的协商。该参数默认值为20倍的tickTime。

fsync.warningthresholdms
(Java系统属性: zookeeper.fsync.warningthresholdms)

V3.3.4引入: 当事务日志(WAL)中的 `fsync` 操作时间超过该参数所设置的值时，一个告警信息将被输出到log4j日志中。该参数默认值为1000毫秒，该值只能通过系统属性的方式设置。

autopurge.snapRetainCount

V3.4.0引入: 当启用该参数时，ZK将仅保留dataDir与dataLogDir目录下最近的快照及事务日志数据，保留的数量由该参数设置值决定；然后清除其它过期数据。默认值为3，最小值为3。

autopurge.purgeInterval

V3.4.0引入: 以小时为单位的时间间隔，即每隔N小时将触发一次快照及事务日志清理任务。设置一个正数（大于等于1）即可启用自动清理功能，默认值为0。

syncEnabled
(Java系统属性: zookeeper.observer.syncEnabled)

V3.4.6, V3.5.0引入: 集群中角色为observer的成员实时的将事务日志与快照数据写回磁盘，就好像它们是业务的参与者一样。这将减少这些成员的重启恢复时间。如果将这个参数设置为 `false` ，将禁用这个特性。默认为 `true` 。

与集群有关的参数选项
本节所介绍的参数选项主要用于服务器集群中，也就是说当部署一个集群时可以使用这些参数。

electionAlg

配置集群使用的选举算法。0值代表基于UDP的版本，1值代表基于UDP的无认证的快速Leader选举版本，2值代表基于UDP的有认证的快速Leader选举版本，3值代表基于TCP快速Leader选举版本。算法3为默认值。

> 0，1，2三个leader选举的实现方案现已被废弃。我们计划在下一个版本中移除，也就是说后续只有 `FastLeaderElection` 可用。

initLimit

允许集群中follower节点连接及同步数据至leader节点的时间耗时（以tickTime为单位）。如果由ZK管理的数据量比较大，则可以按需增加这个值。

leaderServes
(Java系统属性: zookeeper.leaderServes)

配置集群中的leader节点是否接受客户端连接，默认为 `yes` 。Leader节点主要负责协调集群节点之间数据的更新。对于与协调更新有关的吞吐量远高于客户端的读请求的吞吐量，则leader节点可设置为不接受客户端的连接请求，而专注于协调集群节点间的数据更新。默认值为 `yes` 意味着leader节点默认情况下将会接受客户端的连接请求。

> 当集群超过3个以上的节点时，推荐打开这个开关。

server.x=[hostname]:nnnnn[:nnnnn]

ZK集群由多个服务器节点组成。当服务器节点启动时，ZK到$data_dir目录下找到 `myid` 文件，并根据里面内容确定其身份编号及连接端口信息。 `myid` 文件中包含当前服务器节点的编号（以ASCII格式呈现），这个编号与上述配置参数 `server.x` 中的 `x` 匹配的则为当前服务器节点的IP与监听端口。

配置参数中的服务器列表信息必须与ZK集群服务器节点的一一匹配。

配置参数中有两个端口号，第一个端口用于follower节点与leader节点间的通信，第二个端口用于集群leader选举。只有 `electionAlg` 参数值为非0值时，选举用的端口号（第二个）才是必须的。若 `electionAlg` 参数值为0值时，则第二个端口号是非必要的。如果你想在单机上测试多个服务器，则可以使用不同的端口号。

syncLimit

允许集群follower节点与leader节点进行数据同步的总耗时（以tickTime为单位）。如果follower的数据与leader的数据由于同步不及时导致差异太大，则这些follower节点将被移出集群。

group.x=nnnnn[:nnnnn]

Enables a hierarchical quorum construction."x"是组编号，而"="右边则是用":"分隔的服务器编号。每组的服务器成员编号不能有交集，同时所有组的节点成员的并集刚好是集群的所有成员。

你可在这里找到使用的例子。

weight.x=nnnnn

与"group"参数搭配使用。它为集群中的每个成员提供一个权值，这个权值在进行进行法定人数选举投票时会用到。There are a few parts of ZooKeeper that require voting such as leader election and the atomic broadcast protocol. By default the weight of server is 1. If the configuration defines groups, but not weights, then a value of 1 will be assigned to all servers.

你可在这里找到使用的例子。

cnxTimeout
(Java系统属性: zookeeper.cnxTimeout)

为leader选举通知消息而打开的连接设置一个超时时间。只有 `electionAlg` 值为3时这个参数才有用。默认值为5秒。

4lw.commands.whitelist
(Java系统属性: zookeeper.4lw.commands.whitelist)

V3.4.10引入: 这个属性包含了一系统用逗号分隔的由4个字符组成的命令列表。它之所以被引入是为了提供更灵活的粒度控制机制来决定哪些ZK命令可以执行。因此借助这个参数，用于可以根据需要来关闭某些命令。默认情况下，若未指定该参数，则除了 `wchp` 与 `wchc` 以外的命令都可以执行。如果配置了该参数，则只有参数中列出的命令可以执行（被连接的客户端调用执行）。

下面是一个只启用了 `stat,ruok,conf,isro` 命令的配置:

```
4lw.commands.whitelist=stat, ruok, conf, isro
```

用户也可以使用通配符来进行配置，如:

```
4lw.commands.whitelist=*
```

认证与授权参数选项
本节介绍的内容主要是用于配置认证/授权相关功能的。

zookeeper.DigestAuthenticationProvider.superDigest
(Java系统属性: zookeeper.DigestAuthenticationProvider.superDigest)，只能通过Java系统属性方式配置。

该特性默认是禁用的

V3.2版引入: Enables a ZooKeeper ensemble administrator to access the znode hierarchy as a "super" user. In particular no ACL checking occurs for a user authenticated as super.

org.apache.zookeeper.server.auth.DigestAuthenticationProvider can be used to generate the superDigest, call it with one parameter of `super:<password>`. Provide the generated `super:<data>` as the system property value when starting each server of the ensemble.

When authenticating to a ZooKeeper server (from a ZooKeeper client) pass a scheme of "digest" and authdata of `super:<password>`. Note that digest auth passes the authdata in plaintext to the server, it would be prudent to use this authentication method only on localhost (not over the network) or over an encrypted connection.

ZK命令：4个字母

ZooKeeper可以响应一个小规模的命令集。命令集中的每个命令由4个字母组成。
你可以通过nc或telnet向ZooKeeper服务器的客户端监听端口发送命令。其中有三个命令是比较令人感兴趣的： `stat` 会输出与服务器、已连接的客户端相关的基本信息； `srvr` 与 `cons` 会输出服务器与各个连接的进一步的详细信息。

conf
V3.3.0版引入: 打印ZooKeeper配置的详细信息。
cons
V3.3.0版引入: 列出所有连接至本服务器的客户端连接/会话的详细信息。包括收发数据包数量、会话ID、操作时延（响应时间）、上一次执行的操作等信息。
crst
V3.3.0版引入: 重置所有连接/会话的统计数据。
dump
Lists the outstanding sessions and ephemeral nodes. This only works on the leader.
envi
打印ZK服务环境信息。
ruok
测试ZK服务器是否处于正常运行状态。若服务器正常运行，则会返回imok响应；反之则压根不会返回任何响应信息。
`imok` 响应并不意味着该服务器已经加入ZK法定成员中，而仅代表服务器处于活动状态且与某个特定的客户端口建立了绑定关系。你可以使用 `stat` 命令查看该服务器是否加入法定投票成员及客户端连接等详细状态信息。
srst
重置服务器统计数据。
srvr
V3.3.0版引入: 列出服务器的完整的详细信息。
stat
列出服务器及已连接至该服务器的客户端的简要信息。
wchs
V3.3.0版引入: 列出watch该服务器的客户端的简要信息。
wchc
V3.3.0版引入: 从session视角列出watch该服务器的客户端的详细信息。该命令将输出session(connection)所watch的路径的相关信息。有一点要注意下，执行这个命令的性能损耗与watch的数量有关，请谨慎使用。
wchp
V3.3.0版引入: 从path视角列出watch该path的相关客户端session。有一点要注意下，执行这个命令的性能损耗与watch的数量有关，请谨慎使用。
mntr
V3.4.0版引入: 输出可用于监控集群健康状态的参数列表。

```
$ echo mntr | nc localhost 2185
zk_version 3.4.0
zk_avg_latency 0
zk_max_latency 0
zk_min_latency 0
zk_packets_received 70
zk_packets_sent 69
zk_outstanding_requests 0
zk_server_state leader
zk_znode_count 4
zk_watch_count 0
zk_ephemerals_count 0
zk_approximate_data_size 27
zk_followers 4 - only exposed by the Leader
zk_synced_followers 4 - only exposed by the Leader
zk_pending_syncs 0 - only exposed by the Leader
zk_open_file_descriptor_count 23 - only available on Unix platforms
zk_max_file_descriptor_count 1024 - only available on Unix platforms
```

The output is compatible with java properties format and the content may change over time (new keys added). Your scripts should expect changes.
ATTENTION: Some of the keys are platform specific and some of the keys are only exported by the Leader. The output contains multiple lines with the following format:
`key \t value`

Here's an example of the ruok command:

```
$ echo ruok | nc 127.0.0.1 5111
imok
```

数据文件管理

ZK将其业务数据保存在数据目录，将其事务日志保存在事务日志目录。默认情况下数据目录与事务日志目录是相同的。服务器可以（应该）将事务日志目录与数据目录分开存储。如果将事务日志存储在专用的日志设备上，则吞吐量上升的同时，时延还会下降。

数据目录

这个目录下包含两个文件:

- myid - 包含一个单独的以ASCII形式存储的整数值，这个值代表其所在服务器的ID。
- `snapshot.<zxid>` - 这个文件保存有数据树结构的模糊快照。

每个ZooKeeper服务器都有一个唯一标识ID，这个ID用于两个地方： `myid` 文件与配置文件。位于数据目录下的 `myid` 文件用于标识ZK服务器。配置文件 `zoo.cfg` 中列出了集群节点间相互通信的配置信息（IP或主机名、监听端口等），每行配置信息通过Server ID来标识。当ZooKeeper服务实例启动时，其会从 `myid` 文件中获取它的标识ID，然后再从配置文件 `zoo.cfg` 中读取匹配该ID的服务器配置信息，根据配置信息启动相关端口的监听服务。

The snapshot files stored in the data directory are fuzzy snapshots in the sense that during the time the ZooKeeper server is taking the snapshot, updates are occurring to the data tree. The suffix of the snapshot file names is the zxid, the ZooKeeper transaction id, of the last committed transaction at the start of the snapshot. Thus, the snapshot includes a subset of the updates to the data tree that occurred while the snapshot was in process. The snapshot, then, may not correspond to any data tree that actually existed, and for this reason we refer to it as a fuzzy snapshot. Still, ZooKeeper can recover using this snapshot because it takes advantage of the idempotent nature of its updates. By replaying the transaction log against fuzzy snapshots ZooKeeper gets the state of the system at the end of the log.

日志目录

日志目录包含ZK的事务日志。当任何更新生效前，ZK会确保代表该更新的事务会被写入稳定的存储中。每创建一次快照，会创建一个新的事务日志文件，该日志文件名的后缀为第一个写入该文件的事务日志的zxid。

文件管理

单机模式与集群模式下的快照与日志文件的格式是一样的，因此你可以从集群复制模式环境中获取相关快照及日志文件到单机开发环境中进行问题排查。

通过使用旧的日志及快照文件，你可以分析ZK服务器以前的状态甚至可以恢复至该旧状态。 `LogFormatter` 类允许管理员分析一个日志中的相关事务。

ZK负责快照及日志的创建，但其不会删除这些文件。数据及日志文件的保留策略由外部应用实现。服务器自身只需要最近一次的模糊快照以及从开始创建该快照以来的事务日志文件即可。可以查看“维护”章节来查看更详细的与保留策略相关的配置信息。

> 存储在这些文件中的数据是未加密的。在存储敏感数据的场景下，有必要采取一定的措施来阻止未授权访问请求。这些预防措施都不属于ZK的功能范围，因此相关配置信息也是独立的，本文档不多做介绍。

避免踩坑

下面是几个在配置ZK服务时经常碰到的问题及解决办法：

服务器列表数据不一致

The list of ZooKeeper servers used by the clients must match the list of ZooKeeper servers that each ZooKeeper server has. Things work okay if the client list is a subset of the real list, but things will really act strange if clients have a list of ZooKeeper servers that are in different ZooKeeper clusters. Also, the server lists in each Zookeeper server configuration file should be consistent with one another.

incorrect placement of transasction log

The most performance critical part of ZooKeeper is the transaction log. ZooKeeper syncs transactions to media before it returns a response. A dedicated transaction log device is key to consistent good performance. Putting the log on a busy device will adversely effect performance. If you only have one storage device, put trace files on NFS and increase the snapshotCount; it doesn't eliminate the problem, but it should mitigate it.

Java堆大小配置不正确

You should take special care to set your Java max heap size correctly. In particular, you should not create a situation in which ZooKeeper swaps to disk. The disk is death to ZooKeeper. Everything is ordered, so if processing one request swaps the disk, all other queued requests will probably do the same. the disk. DON'T SWAP.

Be conservative in your estimates: if you have 4G of RAM, do not set the Java max heap size to 6G or even 4G. For example, it is more likely you would use a 3G heap for a 4G machine, as the operating system and the cache also need memory. The best and only recommend practice for estimating the heap size your system needs is to run load tests, and then make sure you are well below the usage limit that would cause the system to swap.

暴露在公网

一个ZK集群应该部署在一个可靠的计算环境中，因此推荐部署在防火墙后面。

最佳实践

For best results, take note of the following list of good Zookeeper practices: For multi-tennant installations see the section detailing ZooKeeper "chroot" support, this canbe very useful when deploying many applications/services interfacing to a single ZooKeeper cluster.