Apusic 金蝶天燕

生成时间：2026-05-19

用户手册

金蝶Apusic分布式配置中心V1.0 for zookeeper

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

- 1 版本更新说明
- 2 简介
  - 2.1 产品简介
  - 2.2 范围和读者
  - 2.3 约定与术语
- 3 功能清单
- 4 部署架构及技术架构
  - 4.1 部署架构
  - 4.2 数据模型和分层命名空间
- 5 产品安装
  - 5.1 部署前准备
    - 5.1.1 获取安装包
    - 5.1.2 环境要求
  - 5.2 授权认证
    - 5.2.1 普通授权
    - 5.2.2 集中授权
    - 5.2.3 获取特征码
  - 5.3 单机模式
    - 5.3.1 安装
    - 5.3.2 连接到ADCC for zk
  - 5.4 集群模式
    - 5.4.1 安装
    - 5.4.2 配置
    - 5.4.3 验证
    - 5.4.4 更新配置
  - 5.5 云部署
    - 5.5.1 build镜像
    - 5.5.2 验证镜像
    - 5.5.3 运行容器
- 6 快速开始
  - 6.1 简单使用
- 7 常用命令

- 7.1 addauth
- 7.2 close
- 7.3 config
- 7.4 connect
- 7.5 create
- 7.6 delete
- 7.7 deleteall
- 7.8 delquota
- 7.9 get
- 7.10 getAcl
- 7.11 getAllChildrenNumber
- 7.12 getEphemerals
- 7.13 history
- 7.14 listquota
- 7.15 ls
- 7.16 printwatches
- 7.17 quit
- 7.18 reconfig
- 7.19 redo
- 7.20 removewatches
- 7.21 set
- 7.22 setAcl
- 7.23 setquota
- 7.24 stat
- 7.25 sync
- 7.26 version
- 7.27 whoami
- 7.28 命令指南
- 8 客户端SDK
- 9 WEB管理监控的使用
  - 9.1 首页
    - 9.1.1 命令列表
  - 9.2 明细页
    - 9.2.1 配置信息

- 9.2.2 连接统计重置
- 9.2.3 连接信息
- 9.2.4 目录信息
- 9.2.5 dump信息
- 9.2.6 环境信息
- 9.2.7 tracemask信息
- 9.2.8 hash信息
- 9.2.9 初始化信息
- 9.2.10 只读信息
- 9.2.11 最新快照信息
- 9.2.12 leader信息
- 9.2.13 监控信息
- 9.2.14 观察者统计重置
- 9.2.15 观察者信息
- 9.2.16 服务器状态信息
- 9.2.17 服务器统计信息
- 9.2.18 监视信息
- 9.2.19 路径监视信息
- 9.2.20 监视概要信息
- 9.2.21 Tracemask设置
- 9.2.22 统计重设
- 9.2.23 系统属性信息
- 9.2.24 选举视图信息
- 9.2.25 统计信息
- 9.2.26 Zab统计信息

- 10 配额说明
  - 10.1 设置配额
  - 10.2 查看配额
  - 10.3 删除配额
- 11 配置说明
  - 11.1 基本配置
  - 11.2 高级配置
  - 11.3 集群配置
  - 11.4 安全性配置

- 11.5 性能调优选项
- 11.6 调试可观察性配置
- 11.7 AdminServer配置
- 11.8 指标提供方
- 11.9 其他配置

- 12 常见问题
  - 12.1 高并发性能调优

1 版本更新说明

本文档根据实际进行更新，最新版本包含历史修改记录。

日期	手册版本	适用产品	更新说明
2025年4月	V1Z01F02	金蝶Apusic分布式配置中心V1.0 for zookeeper	添加配置文件说明 添加常见问题说明
2024年11月	V1Z01F01	金蝶Apusic分布式配置中心V1.0 for zookeeper	修正文档描述
2024年5月	V1Z01F01	金蝶Apusic分布式配置中心V1.0 for zookeeper	更新授权模块

2 简介

2.1 产品简介

金蝶Apusic分布式配置中心软件for zookeeper （Apusic Distributed Config Center for zookeeper，简称：ADCC for zk）为分布式应用系统提供高性能协调服务的产品，针对分布式技术而产生的一种新型的配置手段，通过简单的接口能够对分布式应用系统的配置进行统一的管理、实时更新，并支持配置更改的订阅，可实现不需要重启服务器而动态的修改配置文件的内容，有效提高工作效率。

2.2 范围和读者

本手册介绍ADCC for nacos产品相关的内容，主要适用于用户、实施人员，维护人员等。

2.3 约定与术语

一些约定的缩略词诠释：

- ADCC

金蝶Apusic分布式配置中心软件（Apusic Distributed Config Center）

- ADCC_HOME

金蝶Apusic分布式配置中心软件安装目录

3 功能清单

一级功能模块	二级功能模块	功能说明	备注
配置管理			
	配置信息存储	提供一个集中式存储配置数据的地方，允许应用程序将配置信息（如数据库连接字符串、服务器端口、日志级别等）存储为 ADCC数据节点	
	配置更新通知	通过 Watcher 机制，使客户端能够实时感知配置信息的更新。当配置发生变化时，订阅该节点变化的客户端会收到通知，从而能够及时更新本地配置	
	分层配置结构	支持类似于文件系统的层次化配置结构，方便组织和管理复杂的配置信息	
命名服务			
	名称注册与解析	为分布式系统中的各种资源（如服务器、服务、进程等）提供统一的命名空间，资源可以在其中注册自己的名称，并将名称与实际的资源地址（如 IP 地址和端口号）关联起来	
	全局唯一命名	确保在整个分布式系统中名称的唯一性，便于识别和定位资源	
	动态名称更新	允许资源在运行过程中更新其关联的名称或地址信息，并且这种更新能够被其他依赖该资源的组件及时感知	
分布式锁			
	互斥锁实现	支持多个客户端在分布式环境中实现互斥访问共享资源。通过在 Zookeeper 中创建顺序临时节点来实现锁机制，只有获取到特定顺序节点的客户端才能获得锁	
	读写锁实现	可以区分读锁和写锁，允许多个客户端同时获取读锁，但在有客户端获取写锁时，其他客户端（无论是读锁还是写锁）都需要等待，以实现对共享资源的并发控制	
	锁的公平性与性能平衡	提供公平锁和非公平锁的实现方式。公平锁保证锁的获取按照请求的顺序进行，非公平锁则可能允许后来的请求先获取锁，通过这种方式在锁的公平性和系统性能之间进行平衡	

集群管理			
	成员管理	能够维护分布式集群中各个成员的状态信息，包括成员的加入、离开以及故障检测	
	领导者选举	在集群中提供领导者选举机制，当集群需要一个领导者来协调工作时，多个节点可以通过ADCC提供的选举算法来竞争领导地位。选举过程具有可靠性和高效性，确保集群能够快速确定领导者并继续工作	
	分布式栅栏	用于协调多个分布式进程，使它们能够在某个条件满足之前等待	
分布式队列服务			
	简单队列实现	以作为一个简单的分布式队列使用。生产者可以将消息（数据）存储在ADCC的数据节点中，消费者可以从这些 数据节点中获取消息进行处理	
	顺序队列	保证队列中消息的顺序性，消息按照生产者存入的顺序被消费者获取和处理	
	持久化队列与非持久化队列	支持创建持久化队列（消息在处理后仍然保留在ADCC 中）和非持久化队列（消息在被消费后自动删除），以满足不同的应用需求	
数据发布			
	数据发布	允许一个或多个节点将数据发布到ADCC的特定数据节点中	
数据订阅			
	数据订阅	其他节点可以通过 Watcher 机制订阅这些数据节点的变化，当数据发生更新时，订阅节点能够及时收到通知并获取最新的数据，实现数据的实时传播和共享	
元数据管理			
	存储系统元数据	在分布式存储系统中，用于存储文件系统、数据库等的元数据信息	
		支持元数据的动态更新，并且通过 Zookeeper 的一致性保证机制（如 ZAB 协议）确保元数据在分布式环境中的一致性，使不同节点对元数据的认知保持同步	

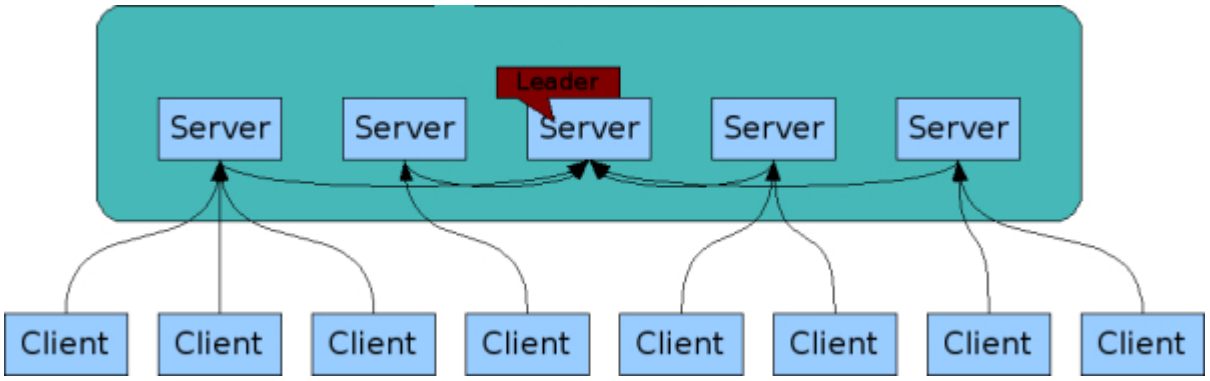
4 部署架构及技术架构

4.1 部署架构

ADCC for zk 支持单机部署和集群部署模式，单机部署模式适用于开发、测试环境，或者对数据一致性要求不高、访问量较小的场景；集群部署模式适用于生成环境，实现高可用。

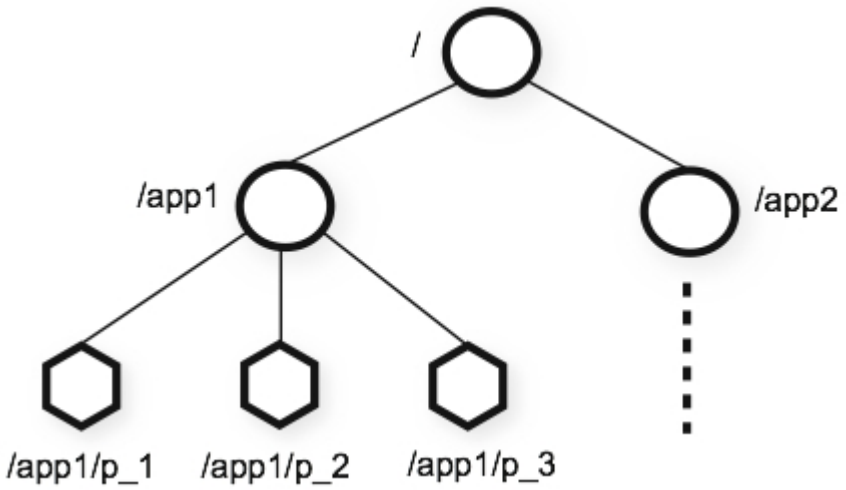
ADCC for zk 通常采用集群部署方式来保证高可用性和数据一致性，通常由多个（一般推荐奇数个，如 3、5、7 等）ADCC 服务器节点组成。节点之间遵循 Zookeeper Atomic Broadcast（ZAB）协议进行通信和协作。在集群启动或者领导者出现故障时，会通过选举机制选出一个领导者（Leader）节点，领导者负责接收客户端的写请求，然后将写操作广播给其他跟随者（Follower）节点，跟随者节点接收并执行写操作，以此保证数据在整个集群中的一致性。而对于客户端的读请求，领导者和跟随者节点都可以处理，这样可以分担负载，提高系统整体的响应效率。

客户端（如各种分布式应用程序、服务等）通过网络与ADCC服务器集群进行交互。客户端可以向 ADCC服务器发送配置管理、获取分布式锁、进行集群成员管理等相关请求，比如将配置信息存储到 ADCC节点中或者申请获取某个分布式锁。同时，客户端也可以注册 Watcher（观察者）来监听ADCC 中特定节点的变化情况，当被监听的节点数据发生变更或者节点状态改变时（例如配置信息被更新或者分布式锁状态变化），ADCC服务器会主动通知相应的客户端，以便客户端做出相应的响应动作，如更新本地配置或者再次尝试获取锁等操作。



4.2 数据模型和分层命名空间

ADCC for zk提供的名称空间与标准文件系统的名称空间非常相似。名称是由斜杠（/）分隔的路径元素序列。ADCC for zk命名空间中的每个节点都由路径标识。ADCC for zk的分层命名空间架构如下：



- 节点和临时节点

与标准文件系统不同，ADCC for zk命名空间中的每个节点都可以有与之关联的数据以及子节点。这就像是拥有一个允许文件同时也是一个路径的文件系统。（ADCC for zk被设计用来存储协调数据：状态信息、配置、位置信息等，因此每个节点存储的数据通常很小，通常在字节到千字节范围内。）使用术语znode来表示ADCC for zk 的数据节点。

znode维护一个stat结构，包括数据更改、ACL更改和时间戳的版本号，以允许缓存验证和协调更新。每当znode的数据发生变化时，版本号就会增加。例如，每当客户端检索数据时，它也会收到数据的版本。

命名空间中每个znode存储的数据都是以原子方式读写的。读取获取与znode关联的所有数据字节，而写入则替换所有数据。每个节点都有一个访问控制列表（ACL），它限制了谁可以执行什么操作。

ADCC for zk还有临时节点的概念。这些znode只要创建它们的会话处于活动状态就存在。当会话结束时，znode就会被删除。

- 条件更新和监视

ADCC for zk 支持监视的概念。客户端可以在 znode 上设置一个监视器。当 znode 发生变化时，监视器会被触发并移除。当监视器被触发时，客户端会收到一个表明 znode 已经改变的数据包。如果客户端与 ZooKeeper 服务器之间的连接断开，客户端将收到一个本地通知。

5 产品安装

5.1 部署前准备

5.1.1 获取安装包

从http://www.apusic.com/下载金蝶Apusic分布式配置中心软件安装包，或从金蝶Apusic分布式配置中心软件产品光盘中获得相应的安装包文件。

安装包通常命名为`ADCC-VXXX-Zookeeper-XXX.tar.gz`，如`ADCC-V1.0.371-Zookeeper-20241219.tar.gz`。

5.1.2 环境要求

类型	要求
操作系统	国产操作系统如银河麒麟系列、中标麒麟系列、普华、中科红旗、深度等； Windows系列； Linux Red Hat 5.2或以上。
CPU	国产架构：鲲鹏、海光、飞腾、龙芯、兆芯等； 国外架构：如Intel等。
内存、硬盘	建议内存4G或以上；可用空间100G或以上
JDK	JDK8、JDK11、JDK12、JDK17稳定版本

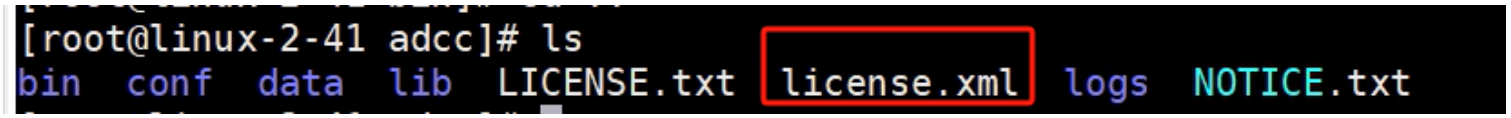
5.2 授权认证

ADCC需要有对应的许可证才能正常使用，通常情况下，金蝶天燕会根据用户购买的产品版本配套对应的许可证。

产品授权方式分为普通授权和集中授权。

5.2.1 普通授权

普通授权指根据IP、域名等方式生成`license.xml`文件，将授权文件放置安装目录下，`${ADCC_HOME}/license.xml`。



5.2.2 集中授权

集中授权指连接授权中心，进行统一授权。需要先搭建金蝶Apusic授权中心，操作方式可参考《金蝶Apusic许可授权中心用户手册》，或联系金蝶天燕技术支持人员。

在服务器中配置环境变量，或在ADCC安装目录根目录创建`acls.properties`文件，添加以下参数：

```
apusic_acls_enable=true
apusic_acls_authUrls=172.24.4.166:6886
apusic_acls_ns=apusic
apusic_acls_tenant=ApusicTest
```

连接参数说明：

参数名	参数值说明
apusic_acls_enable	是否开启授权中心认证，取值为true或false，为true则表示开启授权中心认证。没有该参数或该参数值为false，都表示没有开启授权中心认证；
apusic_acls_authUrls	授权中心的地址，可设置多个授权地址，格式为ip1:port1,ip2:port2，如果一个授权地址链接失败，会轮询其他的地址；如果开启授权中心认证，则为必填参数，其中端口为授权中心的https端口；
apusic_acls_ns	设置该实例所属的命名空间名称，可选参数；默认值为public，具体的命名空间可以在授权中心管理控制台-系统管理-授权管理查看。
apusic_acls_tenant	设置该实例所属的租户名称，可选参数。

ADCC启动时将会自动连接到Apusic授权中心。

5.2.3 获取特征码

如果在使用过程中出现许可证过期或无效等问题，建议优先联系对接的天燕服务人员，重新申请对应许可证。重新申请对应许可证时，需要将产品的特征码(auth code)提供到天燕对接人员。

在`${ADCC_HOME}/bin`，执行`adccServer.sh -ac host`，host取值为ip地址或者网卡名称，类似如下：

```
./adccServer.sh -ac 172.20.140.17
```

打印特征码信息，类似如下，Auth Code=右边则为特征码内容：

```
ADCC JMX enabled by default
Auth Code=SZTY777387783
```

获取特征码后再提供特征码申请授权文件。

5.3 单机模式

5.3.1 安装

在单机模式下设置ADCC for zk服务器非常简单。服务器包含在一个JAR文件中，因此安装时包括创建配置。

获取到稳定的ADCC for zk版本后，如`ADCC-V1.0.371-Zookeeper-20241219.tar.gz`；拷贝到目录，如`/opt`；执行解压，得到有目录`adcc/`，此时`/opt/adcc/`就是`${ADCC_HOME}`。

```
tar -zxvf ADCC-V1.0.371-Zookeeper-20241219.tar.gz
```

启动ADCC for zk前，需要确认data目录，默认为`../data`。可以在`conf/zoo.cfg`中创建：

```
tickTime=2000
dataDir=../data
clientPort=2181
```

更改dataDir的值以指定一个现有的（首先为空）目录。以下是每个字段的含义：

tickTime：ADCC for zk使用的基本时间单位（毫秒）。它用于控制 ADCC服务器的心跳机制和会话超时等多个重要功能的时间间隔。简单来说，`tickTime`是 ADCC 进行各种时间相关操作的基本节拍。

dataDir：存储内存中数据库快照的位置，除非另有规定，还存储数据库更新的事务日志。

clientPort：侦听客户端连接的端口。

启动ADCC for zk：

```
bin/adccServer.sh start
```

ADCC for zk使用log4j记录消息。根据log4j配置，您将看到控制台（默认）和/或日志文件中的日志消息。

此处步骤以单机模式运行ADCC for zk，没有复制，因此如果ADCC for zk进程失败，服务将关闭。

注：对于长期运行的生产系统，ADCC for zk存储必须由外部管理（dataDir和日志）。

5.3.2 连接到ADCC for zk

```
\$ bin/adccCli.sh -server 127.0.0.1:2181
```

5.4 集群模式

在单机模式下运行ADCC for zk便于评估、某些开发和测试。但在生产中，建议在集群模式下运行ADCC for zk。同一应用程序中的一组复制服务器称为仲裁，在集群模式下，仲裁中的所有服务器都具有相同配置文件的副本。

通常情况下，集群需要保持半数以上的存活节点，因此集群节点至少需要3个及以上。

5.4.1 安装

获取到稳定的ADCC for zk版本后，如`ADCC-V1.0.371-Zookeeper-20241219.tar.gz`；拷贝到目录，如`/opt`；执行解压，得到有目录`adcc/`，此时`/opt/adcc/`就是`${ADCC_HOME}`。

```
tar -zxvf ADCC-V1.0.371-Zookeeper-20241219.tar.gz
```

在集群的每个节点服务器都需要安装ADCC；如果节点在同一服务器，需要注意端口不能冲突。

5.4.2 配置

集群模式的`conf/zoo.cfg`文件与单机模式中使用的类似，但有一些区别。下面以在172.20.140.17、172.20.140.13和172.20.12的机器部署集群为环境进行介绍，在每一个环境上ADCC for zk的`zoo.cfg`的集群配置文件，需要增加如下的最后3行（如果存在端口冲突，则需要修改端口）：

```
tickTime=2000
dataDir=../data
clientPort=2181
initLimit=10
syncLimit=5
server.1=172.20.140.17:2888:3888
server.2=172.20.140.13:4888:5888
server.3=172.20.140.12:6888:7888
```

表单的条目`server.X`列出组成ADCC for zk集群服务的服务器。

```
dataDir=../data
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
# increase this if you need to handle more clients
#maxClientCnxns=60
#
# Be sure to read the maintenance section of the
# administrator guide before turning on autopurge.
#
#
# The number of snapshots to retain in dataDir
autopurge.snapRetainCount=10
# Purge task interval in hours
# Set to "0" to disable auto purge feature
autopurge.purgeInterval=12

## Metrics Providers
#
# https://prometheus.io Metrics Exporter
#metricsProvider.className=org.apache.zookeeper.metrics.prometheus.PrometheusMetricsProvider
#metricsProvider.httpPort=7000
#metricsProvider.exportJvmInfo=true

#
# the port of adminserver,default is 8080
#admin.serverPort=8080
#
#the config of cluster,must create myid file in dataDir and set value respectively
server.1=172.20.140.17:2888:3888
server.2=172.20.140.13:4888:5888
server.3=172.20.140.12:6888:7888
~
~
```

当服务器启动时，它通过在数据目录中查找文件 `myid` 来知道它是哪个服务器。该文件包含服务器号 `X` ，所以集群配置还需要如下的修改：

在172.20.140.12中的 `dataDir` 目录创建 `myid` 文件，即在 `../data` 下创建 `myid` 文件，并输入上面配置中 `server.1` 中的序号 `1` ，并保存；

同理在172.20.140.17中的 `dataDir` 目录创建 `myid` 文件，即在 `../data` 下创建 `myid` 文件，并输入上面配置中 `server.2` 中的序号 `2` ，并保存；

同理在172.20.140.43中的 `dataDir` 目录创建 `myid` 文件，即在 `../data` 下创建 `myid` 文件，并输入上面配置中 `server.3` 中的序号 `3` ，并保存；

如下图为 `server.1` 服务器的 `myid` 配置：

```
[root@myRabbitA data]# vim myid
[root@myRabbitA data]# ls
myid  version-2
[root@myRabbitA data]# cat myid
1
```

到这里，3个节点的集群就已经配置完成，然后分别启动即可。注意：在3个节点未完全启动完前，会答应连接不上的信息，这是正常的。

上面配置的其他说明：

initLimit：ADCC for zk用来限制quorum中ADCC for zk服务器必须连接到领导者的时间长度的超时。条目syncLimit限制了服务器与Leader之间的过期距离。

对于这两种超时，您可以使用tickTime指定时间单位。在本例中，initLimit的超时为5次，每次为2000毫秒，即10秒。

最后，请注意每个服务器名称后面的两个端口号："2888"和"3888"。对等体使用前一个端口连接到其他对等体。这样的连接是必要的，以便对等体可以进行通信，例如商定更新的顺序。更具体地说，ADCC for zk服务器使用此端口将follower连接到Leader。当出现新的Leader时，follower使用此端口打开与Leader的TCP连接。由于默认Leader选举也使用TCP，因此我们当前需要另一个端口进行Leader选举。这是服务器条目中的第二个端口。

5.4.3 验证

依次启动ADCC集群节点。

```
./adccServer.sh start
```

查看ADCC节点状态。

```
./adccServer.sh status
```

下图为示例参考图，主要查看Mode属性值和运行状态，其他值可能会与实际略不同。

可以看到有一个是leader，两个是follower。

```
[root@tck-server bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
[root@tck-server bin]#
```

```
[root@test3 bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
[root@test3 bin]#
```

```
[root@myRabbitA bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
[root@myRabbitA bin]#
```

先把其中一个follower的节点停止，查看其他节点的状态。可以看到其他节点的状态依旧正常，集群可以正常运行。

```
[root@myRabbitA bin]# ./adccServer.sh stop
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Stopping ADCC ... STOPPED
[root@myRabbitA bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Error contacting service. It is probably not running.
[root@myRabbitA bin]#
```

```
[root@tck-server bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
[root@tck-server bin]#
```

```
[root@test3 bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
[root@test3 bin]#
```

再停止一个节点，查看结果。可以看到节点停止后，此时运行的节点少于集群总数量的一半，剩下的一个节点也停止了。

停止运行的节点：

```
[root@tck-server bin]# ./adccServer.sh stop
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Stopping ADCC ... STOPPED
[root@tck-server bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Error contacting service. It is probably not running.
[root@tck-server bin]#
```

没有执行停止运行命令的节点：

```
[root@test3 bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Error contacting service. It is probably not running.
[root@test3 bin]#
```

上一步停止运行的节点：

```
[root@myRabbitA bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Error contacting service. It is probably not running.
[root@myRabbitA bin]#
```

再把其中一个已停止的节点启动，查看结果。会看到集群的节点运行数量过半后，集群又恢复正常运行状态，并且会重新推举出一个leader。

执行启动命令的节点：

```
[root@tck-server bin]# ./adccServer.sh start
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Starting ADCC ... STARTED
[root@tck-server bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: follower
[root@tck-server bin]#
```

没有执行停止命令的节点：

```
[root@test3 bin]# ./adccServer.sh status
ADCC JMX enabled by default
Using config: /opt/ADCC/adcc-server-1.0/bin/../conf/zoo.cfg
Client port found: 2181. Client address: localhost. Client SSL: false.
Mode: leader
[root@test3 bin]#
```

此时再加入节点也是follower节点，不会影响到现任的leader节点。

5.4.4 更新配置

ADCC集群支持在运行时更改集群的成员身份，可以通过reconfig命令更新配置。

执行reconfig前，需要在**每一台集群成员节点**的 `zoo.cfg` 中添加 `reconfigEnabled=true` 、`standaloneEnabled=false` 。

如果是新增集群成员，需要先安装有ADCC，配置文件改为与集群中其他成员的一致。

示例在服务器172.20.140.17添加新节点，新安装ADCC，在ADCC新安装目录中的 `dataDir` 目录创建 `myid` 文件，即在 `../data` 下创建 `myid` 文件，并输入上面配置中 `server.4` 中的序号 `4` ，并保存。如果端口冲突，需要更新端口，如 `clientPort=2184` 。配置文件 `zoo.cfg` 如下：

```
autopurge.purgeInterval=12
initLimit=10
```

```
syncLimit=5
autopurge.snapRetainCount=10
clientPort=2184
tickTime=2000
dataDir=../data
reconfigEnabled=true
standaloneEnabled=false
server.1=172.20.140.17:2888:3888
server.2=172.20.140.13:4888:5888
server.3=172.20.140.12:6888:7888
server.4=172.20.140.17:2781:2786:observer;2184
```

连接集群的其中一个节点。

```
\$ bin/adccCli.sh -server 172.20.140.17:2181
```

需要添加一个超级用户；或者关闭ACL，在 `zoo.cfg` 中设置 `skipACL=yes`

1、添加超级用户

```
[adcc: 172.20.140.17:2181(CONNECTED) 0] addauth digest super:apusic
```

2、执行reconfig命令更新配置。

示例：

1）将follower节点 2更改为observer,并且将端口2182改为12182；

2）将节点5添加到集群，设置角色为observer；

```
[adcc: 127.0.0.1:2181(CONNECTED) 1] reconfig -add
2=172.20.140.13:4888:5888:observer;12182 -add
4=172.20.140.17:2781:2786:observer;2184 -remove 4
Committed new configuration:
server.1=172.20.140.17:2888:3888:participant;0.0.0.0:2181
server.2=172.20.140.13:4888:5888:observer;0.0.0.0:12182
server.3=172.20.140.12:6888:7888:participant;0.0.0.0:2183
server.4=172.20.140.17:2781:2786:observer;0.0.0.0:2185
version=2000000003
```

3）将节点4从集群中移除。

```
[adcc: 127.0.0.1:2181(CONNECTED) 1] reconfig -remove 4
Committed new configuration:
server.1=172.20.140.17:2888:3888:participant;0.0.0.0:2181
server.2=172.20.140.13:4888:5888:observer;0.0.0.0:12182
server.3=172.20.140.12:6888:7888:participant;0.0.0.0:2183
version=2000000004
```

5.5 云部署

ADCC支持云部署。

5.5.1 build镜像

先获取到ADCC的安装包以及有效的授权文件。

创建Dockerfile。示例如下，操作系统、安装包名称需要根据实际情况修改。

```
FROM harbor.apusic.com/public/jdk1.8:v241
#For ARM Architect
#FROM harbor.apusic.com/arm64/openjdk:8u342-jdk-oraclelinux7

ENV ADCCZK_VERSION 1.0
COPY adcc-server-1.0.tar.gz /home/apusic/

RUN mkdir -p /home/apusic/adcc/data /home/apusic/adcc/wal
/home/apusic/adcc/log
RUN cd /home/apusic &&  \
tar xf adcc-server-1.0.tar.gz -C /home/apusic && \
rm -rf adcc-server-1.0.tar.gz

ADD license.xml /home/apusic/adcc/license.xml
ADD zoo.cfg /home/apusic/adcc/conf/zoo.cfg
COPY entrypoint.sh /home/apusic/adcc/bin/

ENV PATH=/home/apusic/adcc/bin:${PATH} \
    ZOO_LOG_DIR=/home/apusic/adcc/log \
```

```
    ZOO_LOG4J_PROP="INFO, CONSOLE, ROLLINGFILE" \
    JMXPORT=9010

RUN chmod +x /home/apusic/adcc/bin/*.sh
ENTRYPOINT ["entrypoint.sh"]
CMD [ "/home/apusic/adcc/bin/adccServer.sh", "start-foreground" ]
EXPOSE 2181 2888 3888 9010
```

执行`docker build`命令，如

```
docker build -t apusic/adcc-zk:v1.0 .
```

5.5.2 验证镜像

使用`docker images`验证镜像。

```
docker images
```

5.5.3 运行容器

执行`docker run`命令运行容器，如:

```
docker run --name adcczk -p 2181:2181 -d apusic/adcc-zk:v1.0
```

执行`docker ps -a`命令查看容器。

6 快速开始

使用ADCC for zk的自带命令行工具，可以进行基础的功能使用。

6.1 简单使用

启动ADCC for zk服务器端后，使用如下命令连接到服务器：

```
\$ bin/adccCli.sh -server 127.0.0.1:2181
```

连接服务器后，就可以执行一下简单的操作。

连接成功后，会看到如下类似的提示：

```
Connecting to 127.0.0.1:2181
...
...
...
[adcc: 127.0.0.1:2181(CONNECTED) 0]
```

从命令行窗口，输入help，可以获得可操作的命令列表，如下：

```
[adcc: 127.0.0.1:2181(CONNECTED) 0] help
adccCli -server host:port -client-configuration properties-file cmd
args
        addWatch [-m mode] path # optional mode is one of
[PERSISTENT, PERSISTENT_RECURSIVE] - default is PERSISTENT_RECURSIVE
        addauth scheme auth
        close
        config [-c] [-w] [-s]
        connect host:port
        create [-s] [-e] [-c] [-t ttl] path [data] [acl]
        delete [-v version] path
        deleteall path [-b batch size]
        delquota [-n|-b|-N|-B] path
        get [-s] [-w] path
```

```
        getAcl [-s] path
        getAllChildrenNumber path
        getEphemerals path
        history
        listquota path
        ls [-s] [-w] [-R] path
        printwatches on|off
        quit
        reconfig [-s] [-v version] [[-file path] | [-members
serverID=host:port1:port2;port3[,...]*]] | [-add
serverId=host:port1:port2;port3[,...]]* [-remove serverId[,...]*]
        redo cmdno
        removewatches path [-c|-d|-a] [-l]
        set [-s] [-v version] path data
        setAcl [-s] [-v version] [-R] path acl
        setquota -n|-b|-N|-B val path
        stat [-w] path
        sync path
        version
        whoami
```

从这里开始，可以尝试几个简单的命令来了解这个简单的命令行界面。

通过运行create/adcc_test my_data创建一个新的znode。这将创建一个新的znode，并将字符串"my_data"与节点相关联。您应该看到：

```
[adcc: 127.0.0.1:2181(CONNECTED) 9] create /adcc_test my_data
Created /adcc_test
```

发出另一个ls/命令以查看目录的内容：

```
[adcc: 127.0.0.1:2181(CONNECTED) 11] ls /
[adcc_test]
```

注意，现在已经创建了adcc_test目录。

接下来，通过运行get命令验证数据是否与znode关联，如所示：

```
[adcc: 127.0.0.1:2181(CONNECTED) 12] get -s /adcc_test
# 节点的值
my_data
# 节点创建时分配的id
cZxid = 0xf00000002
# 节点创建的时间
ctime = Fri Nov 01 10:52:44 CST 2024
# 修改后分配的id
mZxid = 0xf00000002
# 节点的修改时间
mtime = Fri Nov 01 10:52:44 CST 2024
# 子节点的id
pZxid = 0xf00000002
# 子节点的版本
cversion = 0
# 当前节点数据的版本号
dataVersion = 0
# 权限的版本
aclVersion = 0
# session的id,如果是0x0表示为永久节点，否则就是临时节点
ephemeralOwner = 0x0
# 数据长度
dataLength = 7
# 子节点个数
numChildren = 0
```

我们可以通过发出set命令来更改与adcc_test相关的数据，如下所示：

```
[adcc: 127.0.0.1:2181(CONNECTED) 16] set /adcc_test apusic
[adcc: 127.0.0.1:2181(CONNECTED) 17] get -s /adcc_test
apusic
cZxid = 0xf00000002
ctime = Fri Nov 01 10:52:44 CST 2024
mZxid = 0xf00000005
mtime = Fri Nov 01 11:05:16 CST 2024
```

```
pZxid = 0xf00000002
cversion = 0
dataVersion = 3
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

最后，让我们通过发出以下命令来删除节点：

```
[adcc: 127.0.0.1:2181(CONNECTED) 18] delete /adcc_test
```

7 常用命令

7.1 addauth

增加一个用户到ACL列表中。

ACL全称为Access Control List，访问控制列表。ADCC的ACL由三部分组成，即`Schema:id:permission`，其中：

Schema是验证过程中使用的检验策略，有5个类型，分别是world、auth、digest、IP、super；

id的权限被赋予的对象，比如某个IP或者用户；

permission为可以操作的权限，有5个值：crdwa，分别表示create、read、delete、write、admin，admin是修改权限。

```
# 添加用户user1，密码为123456
[adcc: 127.0.0.1:2181(CONNECTED) 10] addauth digest user1:123456

# 设置ACL权限
[adcc: 127.0.0.1:2181(CONNECTED) 11] setAcl /adcc_test
auth:user1:123456:r

# 查看ACL权限
[adcc: 127.0.0.1:2181(CONNECTED) 12] getAcl -s /adcc_test
'digest,'user1:x
: r
cZxid = 0xf0000000c
ctime = Fri Nov 01 14:23:46 CST 2024
mZxid = 0xf0000000c
mtime = Fri Nov 01 14:23:46 CST 2024
pZxid = 0xf0000000c
cversion = 0
dataVersion = 0
aclVersion = 1
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

7.2 close

关闭当前会话。

```
[adcc: 127.0.0.1:2181(CONNECTED) 45] close

WATCHER::

WatchedEvent state:Closed type:None path:null
2024-11-01 14:36:30,189 [myid:] - INFO  [main:ZooKeeper@1288] -
Session: 0x301e0100dee0000 closed
2024-11-01 14:36:30,189 [myid:] - INFO  [main-
EventThread:ClientCnxn$EventThread@568] - EventThread shut down for
session: 0x301e0100dee0000
```

7.3 config

显示仲裁成员的配置。

```
[adcc: 127.0.0.1:2181(CONNECTED) 0] config
server.1=172.20.140.17:2888:3888:participant
server.2=172.20.140.13:4888:5888:participant
server.3=172.20.140.12:2888:3888:participant
version=0
```

7.4 connect

连接到一个ADCC for zk服务器。

```
# 连接远程服务.格式为connect remoteIP:2181
[adcc: 127.0.0.1:2181(CONNECTED) 2] connect 172.20.140.12:2181
```

7.5 create

创建一个znode。

```
# 新建持久节点
[adcc: 127.0.0.1:2181(CONNECTED) 4] create /persistent_node
Created /persistent_node

# 新建临时节点
[adcc: 127.0.0.1:2181(CONNECTED) 5] create -e /ephemeral_node mydata
Created /ephemeral_node

# 新建持久顺序节点
[adcc: 127.0.0.1:2181(CONNECTED) 6] create -s
/persistent_sequential_node mydata
Created /persistent_sequential_node0000000010

# 新建临时顺序节点
[adcc: 127.0.0.1:2181(CONNECTED) 7] create -s -e
/ephemeral_sequential_node mydata
Created /ephemeral_sequential_node0000000011

# 新建符合模式的节点
[adcc: 127.0.0.1:2181(CONNECTED) 8] create /zk-node-create-schema
mydata digest:user1:+owfoSBn/am19roBPzR1/MfCblE=:crwad
Created /zk-node-create-schema
[adcc: 127.0.0.1:2181(CONNECTED) 9] addauth digest user1:12345
[adcc: 127.0.0.1:2181(CONNECTED) 10] getAcl /zk-node-create-schema
'digest,'user1:+owfoSBn/am19roBPzR1/MfCblE=
: cdrwa

# 新建容器节点，当容器的最后一个节点被删除时，该容器节点随之被删除
[adcc: 127.0.0.1:2181(CONNECTED) 11] create -c /container_node
mydata
Created /container_node
[adcc: 127.0.0.1:2181(CONNECTED) 12] create -c
/container_node/child_1 mydata
Created /container_node/child_1
[adcc: 127.0.0.1:2181(CONNECTED) 13] create -c
```

```
/container_node/child_2 mydata
Created /container_node/child_2
[adcc: 127.0.0.1:2181(CONNECTED) 14] delete /container_node/child_1
[adcc: 127.0.0.1:2181(CONNECTED) 15] delete /container_node/child_2
[adcc: 127.0.0.1:2181(CONNECTED) 17] get /container_node
Node does not exist: /container_node

# 新建ttl节点
# 需要在zoo.cfg中添加extendedTypesEnabled=true
# 否则/ttl_node 会返回KeeperErrorCode = Unimplemented
[adcc: 127.0.0.1:2181(CONNECTED) 0] create -t 3000 /ttl_node mydata
Created /ttl_node
# 3秒之后
[adcc: 127.0.0.1:2181(CONNECTED) 1] get /ttl_node
Node does not exist: /ttl_node
```

7.6 delete

根据具体的路径删除一个节点。

```
[adcc: 127.0.0.1:2181(CONNECTED) 11] delete /adcc_node1/test1
[adcc: 127.0.0.1:2181(CONNECTED) 12] ls /adcc_node1/test1
Node does not exist: /adcc_node1/test1
```

7.7 deleteall

删除一个具体路径下的所有节点。

```
[adcc: 127.0.0.1:2181(CONNECTED) 17] ls /adcc_node1
[test1, test2, test3]
[adcc: 127.0.0.1:2181(CONNECTED) 18] deleteall /adcc_node1
[adcc: 127.0.0.1:2181(CONNECTED) 19] ls /adcc_node1
Node does not exist: /adcc_node1
```

7.8 delquota

删除一个路径下的配额。

```
[adccshell: 1] delquota /quota_test
[adccshell: 2] listquota /quota_test
" absolute path" is /zookeeper/quota/quota_test/zookeeper_limits
" quota" for /quota_test does not exist.
[adccshell: 3] delquota -n /c1
[adccshell: 4] delquota -N /c2
[adccshell: 5] delquota -b /c3
[adccshell: 6] delquota -B /c4
```

7.9 get

获取具体路径下的数据。

```
# 获取具体路径的数据
[adcc: 127.0.0.1:2181(CONNECTED) 42] get /adcc_node1
mydata1

# -s 获取具体路径的详细信息
[adcc: 127.0.0.1:2181(CONNECTED) 43] get -s /adcc_node1
mydata1
cZxid = 0x1100000024
ctime = Fri Nov 01 17:20:56 CST 2024
mZxid = 0x1100000025
mtime = Fri Nov 01 17:22:17 CST 2024
pZxid = 0x1100000024
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 7
numChildren = 0
```

```
# -w 设置监视数据变化，注意：打开printwatches
[adcc: 127.0.0.1:2181(CONNECTED) 60] get -w /adcc_node1
mydata1
```

7.10 getAcl

获取路径下的ACL权限。

```
[adcc: 127.0.0.1:2181(CONNECTED) 69] create /adcc_node4 mydata
ip:172.20.140.12:crwda
Created /adcc_node4
[adcc: 127.0.0.1:2181(CONNECTED) 70] getAcl /adcc_node4
'ip,'172.20.140.12
: cdrwa
```

7.11 getAllChildrenNumber

获取指定路径下的所有子节点数目。

```
[adcc: 127.0.0.1:2181(CONNECTED) 71] getAllChildrenNumber /
27
```

7.12 getEphemerals

获取当前会话创建的所有临时节点。

```
[adcc: 127.0.0.1:2181(CONNECTED) 75] create -e /adcc_eph mydata
Created /adcc_eph
[adcc: 127.0.0.1:2181(CONNECTED) 76] getEphemerals /
[/adcc_eph]
```

7.13 history

显示最近执行的命令，最多显示11个。

```
[adcc: 127.0.0.1:2181(CONNECTED) 7] history
0 - close
1 - close
2 - ls /
3 - ls /
4 - connect
5 - ls /
6 - ll
7 - history
```

7.14 listquota

显示路径下的配额。

```
[adcc: 127.0.0.1:2181(CONNECTED) 1] listquota /adcc_test
absolute path is /zookeeper/quota/adcc_test/zookeeper_limits
Output quota for /adcc_test
count=2,bytes=5=;byteHardLimit=-1;countHardLimit=-1
Output stat for /adcc_test count=5,bytes=10
```

7.15 ls

显示路径下的子节点。

```
[adcc: 127.0.0.1:2181(CONNECTED) 22] ls /adcc
[test1, test2, test3, test4]

# -s 查看当前路径下详细信息
[adcc: 127.0.0.1:2181(CONNECTED) 23] ls -s /adcc
[test1, test2, test3, test4]
cZxid = 0x1100000015
ctime = Mon Nov 04 09:56:28 CST 2024
mZxid = 0x1100000018
mtime = Mon Nov 04 09:57:51 CST 2024
pZxid = 0x110000001c
```

```
cversion = 4
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 10
numChildren = 4

# -R 查看当前路径下所有子节点
[adcc: 127.0.0.1:2181(CONNECTED) 24] ls -R /adcc
/adcc
/adcc/test1
/adcc/test2
/adcc/test3
/adcc/test4

# -w 查看当前路径下子节点的变化。需要开启监视：printwatches
[adcc: 127.0.0.1:2181(CONNECTED) 25] ls -w /adcc
[test1, test2, test3, test4]
```

7.16 printwatches

打开或关闭打印监视的开关。

```
[adcc: 127.0.0.1:2181(CONNECTED) 0] printwatches
printwatches is on
[adcc: 127.0.0.1:2181(CONNECTED) 1] printwatches off
[adcc: 127.0.0.1:2181(CONNECTED) 2] printwatches
printwatches is off
[adcc: 127.0.0.1:2181(CONNECTED) 3] printwatches on
[adcc: 127.0.0.1:2181(CONNECTED) 4] printwatches
printwatches is on
```

7.17 quit

退出CLI窗口。

```
[adcc: 127.0.0.1:2181(CONNECTED) 1] quit
```

7.18 reconfig

在运行时更改集合的成员身份。

执行reconfig前，需要在 `zoo.cfg` 中添加 `reconfigEnabled=true` 、 `standaloneEnabled=false` 。

```
# 需要添加一个超级用户；或者关闭ACL，在zoo.cfg中设置skipACL=yes
# 添加超级用户
[adcc: 127.0.0.1:2181(CONNECTED) 0] addauth digest super:apusic

# 将follower节点2更改为observer,并且将端口2182改为12182
# 将节点5添加到集群，设置角色为observer
# 将节点4从集群中移除
[adcc: 127.0.0.1:2181(CONNECTED) 1] reconfig -add
2=localhost:2781:2786:observer;12182 -add
5=localhost:2781:2786:observer;2185 -remove 4
Committed new configuration:
server.1=localhost:2780:2785:participant;0.0.0.0:2181
server.2=localhost:2781:2786:observer;0.0.0.0:12182
server.3=localhost:2782:2787:participant;0.0.0.0:2183
server.5=localhost:2784:2789:observer;0.0.0.0:2185
version=2000000003

# -members 任命集群成员
[adcc: 127.0.0.1:2181(CONNECTED) 2] reconfig -members
server.1=localhost:2780:2785:participant;0.0.0.0:2181,server.2=localhos
Committed new configuration:
server.1=localhost:2780:2785:participant;0.0.0.0:2181
server.2=localhost:2781:2786:observer;0.0.0.0:12182
server.3=localhost:2782:2787:participant;0.0.0.0:12183
version=2000000004

# 将当前配置文件改为使用 newconfig.txt中的配置
```

```
# 需要指定当前的配置文件版本，如2000000004
[adcc: 127.0.0.1:2181(CONNECTED) 3] reconfig -file /opt/ADCC/adcc-
server/conf/newConfig.txt -v 2000000004
Committed new configuration:
server.1=localhost:2780:2785:participant;0.0.0.0:2181
server.2=localhost:2781:2786:observer;0.0.0.0:12182
server.3=localhost:2782:2787:participant;0.0.0.0:2183
server.5=localhost:2784:2789:observer;0.0.0.0:2185
version=2000000005
```

7.19 redo

使用历史记录中的索引重做cmd。

```
[adcc: 172.0.0.1:2181(CONNECTED) 4] history
0 - ls /
1 - get /consumers
2 - get /hbase
3 - ls /hbase
4 - history
[adcc: 172.0.0.1:2181(CONNECTED) 5] redo 3
[backup-masters, draining, flush-table-proc, hbaseid,
master-maintenance, meta-region-server, namespace, online-snapshot,
replication, rs, running, splitWAL, switch, table, table-lock]
```

7.20 removewatches

移除一个节点下的监视器。

```
[adcc: 172.0.0.1:2181(CONNECTED) 1] get -w /brokers
null
[adcc: 172.0.0.1:2181(CONNECTED) 2] removewatches /brokers
WATCHER::
WatchedEvent state:SyncConnected type:DataWatchRemoved path:/brokers
```

7.21 set

设置或更新一个路径下的数据。

```
[adcc: 172.0.0.1:2181(CONNECTED) 27] set /adcc newdata

# 显示该节点的状态
[adcc: 127.0.0.1:2181(CONNECTED) 28] set -s /adcc newdata
cZxid = 0x1100000015
ctime = Mon Nov 04 09:56:28 CST 2024
mZxid = 0x110000001e
mtime = Mon Nov 04 10:23:26 CST 2024
pZxid = 0x110000001c
cversion = 4
dataVersion = 3
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 7
numChildren = 4

# -v使用CAS设置数据，可以使用stat从dataVersion中找到版本
[adcc: 127.0.0.1:2181(CONNECTED) 29] set -v 0 /adcc newdata
version No is not valid : /adcc
```

7.22 setAcl

设置一个节点的ACL权限。

```
# 添加用户user1，密码为123456
[adcc: 127.0.0.1:2181(CONNECTED) 4] addauth digest user1:123456

# 为节点 /mynode/test1设置认证用户为user1，密码为123456，权限为crwad
[adcc: 127.0.0.1:2181(CONNECTED) 5] setAcl /mynode/test1
auth:user1:123456:crwad
```

```
# 查看节点权限
[adcc: 127.0.0.1:2181(CONNECTED) 6] getAcl /mynode/test1
'digest,'user1:HYGa7IZRm2PUBFiFFu8xY2pPP/s=
: cdrwa

# -R 递归设置权限
[adcc: 127.0.0.1:2181(CONNECTED) 8] ls /mynode
[test1, test2]
[adcc: 127.0.0.1:2181(CONNECTED) 9] getAcl /mynode/test2
'world,'anyone
: cdrwa
[adcc: 127.0.0.1:2181(CONNECTED) 10] setAcl -R /mynode/test2
auth:user1:123456:crwad
[adcc: 127.0.0.1:2181(CONNECTED) 11] getAcl /mynode/test2
'digest,'user1:HYGa7IZRm2PUBFiFFu8xY2pPP/s=
: cdrwa


# -v 使用Acl版本设置Acl，该版本可以使用stat从aclVersion中找到，权限版本每次修改该版本号加1
[adcc: 127.0.0.1:2181(CONNECTED) 12] stat /mynode
cZxid = 0x1100000024
ctime = Tue Nov 05 10:04:55 CST 2024
mZxid = 0x1100000024
mtime = Tue Nov 05 10:04:55 CST 2024
pZxid = 0x1100000027
cversion = 2
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 2
[adcc: 127.0.0.1:2181(CONNECTED) 18] setAcl -v 1 /mynode/test2
auth:user1:123456:crwad
```

7.23 setquota

设置一个路径下的配额。在设置配额前，需要在配置文件 `zoo.cfg` 添加 `enforceQuota=true` 。

```
# -n 限制子节点的个数
[adcc: 127.0.0.1:2181(CONNECTED) 18] setquota -n 2 /quota_test
[adcc: 127.0.0.1:2181(CONNECTED) 19] create /quota_test/child_1
Created /quota_test/child_1
[adcc: 127.0.0.1:2181(CONNECTED) 20] create /quota_test/child_2
Created /quota_test/child_2
[adcc: 127.0.0.1:2181(CONNECTED) 21] create /quota_test/child_3
Created /quota_test/child_3
# 注意：-s是没有硬约束，只需要记录警告信息，因而只会在日志打印警告信息
2024-03-07 11:22:36,680 [myid:1] - WARN[SyncThread:0:DataTree@374] -
Quota exceeded: /quota_test count=3
limit=2
2024-03-07 11:22:41,861 [myid:1] - WARN[SyncThread:0:DataTree@374] -
Quota exceeded: /quota_test count=4
limit=2
# -b 限制路径字节数（数据长度）
[adcc: 127.0.0.1:2181(CONNECTED) 22] setquota -b 5 /brokers
[adcc: 127.0.0.1:2181(CONNECTED) 23] set /brokers "I_love_zookeeper"
# 注意：-b是没有硬约束，只需要记录警告信息，因而只会在日志打印警告信息
WARN [CommitProcWorkThread-7:DataTree@379] - Quota exceeded:
/brokers bytes=4206 limit=5
# -N 子节点硬配额
[adcc: 127.0.0.1:2181(CONNECTED) 3] create /c1
Created /c1
[adcc: 127.0.0.1:2181(CONNECTED) 4] setquota -N 2 /c1
[adcc: 127.0.0.1:2181(CONNECTED) 5] listquota /c1
absolute path is /zookeeper/quota/c1/zookeeper_limits
Output quota for /c1
count=-1,bytes=-1=;byteHardLimit=-1;countHardLimit=2
Output stat for /c1 count=2,bytes=0
[adcc: 127.0.0.1:2181(CONNECTED) 6] create /c1/ch-3
Count Quota has exceeded : /c1/ch-3
```

```
# -B 字节硬配额
[adcc: 127.0.0.1:2181(CONNECTED) 3] create /c2
[adcc: 127.0.0.1:2181(CONNECTED) 4] setquota -B 4 /c2
[adcc: 127.0.0.1:2181(CONNECTED) 5] set /c2 "foo"
[adcc: 127.0.0.1:2181(CONNECTED) 6] set /c2 "foo-bar"
Bytes Quota has exceeded : /c2
[adcc: 127.0.0.1:2181(CONNECTED) 7] get /c2
foo
```

7.24 stat

显示一个节点的统计和元数据。

```
[adcc: 127.0.0.1:2181(CONNECTED) 20] stat /mynode
cZxid = 0x1100000024
ctime = Tue Nov 05 10:04:55 CST 2024
mZxid = 0x1100000024
mtime = Tue Nov 05 10:04:55 CST 2024
pZxid = 0x1100000027
cversion = 2
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 2
```

7.25 sync

在leader和follower之间同步一个节点的数据（异步同步）

```
[adcc: 127.0.0.1:2181(CONNECTED) 14] sync /
Sync is OK
```

7.26 version

显示ADCC for zk客户端/CLI的版本。

```
[adcc: 127.0.0.1:2181(CONNECTED) 22] version
ZooKeeper CLI version: 3.7.1-
54a15b10cb6e89430bd2ea712b4b032723cb20a1, built on 2024-03-13 08:25
UTC
```

7.27 whoami

提供添加到当前会话中的所有身份验证信息。

```
[adcc: 127.0.0.1:2181(CONNECTED) 23] whoami
Auth scheme: User
ip: 127.0.0.1
digest: user1
```

7.28 命令指南

命令基本语法及解释

基本命令	说明
help	显示所有操作命令
ls	查看当前节点信息 -s 展示附加次级信息 -w 监听子节点变化
create	普通创建节点 -s 创建含有序列的节点 -e 创建临时节点，重启或超时会消失
get	获得节点的值 -w 监听节点内容变化 -s 附加次级信息
getAcl	获得节点的Acl权限值 -s 附加次级信息
getAllChildrenNumber	获取特定路径下的所有子节点数量

getEphemerals	获取此会话创建的所有临时节点
set	设置节点的值
setAcl	设置节点的Acl权限
setquta	设置指定目录下的配额
stat	查看节点的状态
delete	删除节点
quit	退出CLI窗口
close	关闭当前客户端或会话
config	显示法定人数成员的配置信息
connect	连接adcc服务
deleteall	删除指定目录下的所有节点
delquota	删除指定目录下的配额
history	显示最近执行的11个命令的历史记录
listquota	展示指定目录下的配额
printwatches	是否打印监听事件
reconfig	在运行时更改集成的成员资格
redo	使用历史记录中的索引重新执行cmd
removewatches	移除节点下的监听事件
sync	在leader和follower之间同步一个节点的数据（异步同步）
version	查看adcc版本
whoami	提供添加到当前会话中的所有身份验证信息

8 客户端SDK

ADCC for zk兼容ZooKeeper3.7.X软件的客户端访问协议，ZooKeeper的客户端SDK可以用于访问ADCC for zk服务器，使用ZooKeeper客户的应用，不需要修改应用代码。

9 WEB管理监控的使用

ADCC for zk服务器启动的时候，会启动8080端口服务，用于对服务器环境的统计信息进行展示。

如果要修改端口，可以在配置文件 `zoo.cfg` 中进行设置，如下配置表示管控端口修改为8082端口：

```
admin.serverPort=8082
```

可以通过在启动脚本中增加参数 `-Dadcc.admin.enableServer=false` 禁用管理控制台。

9.1 首页

9.1.1 命令列表

部署完成后，打开浏览器（推荐Chrome、firefox）输入地址进入到控制台登录页面：
http://serverIP:serverPort/commands，会出现如下的界面：

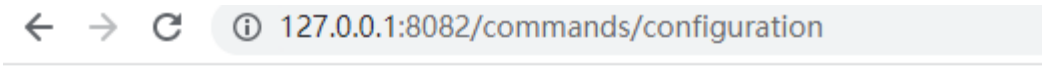
configuration
connection_stat_reset
connections
dirs
dump
environment
get_trace_mask
hash
initial_configuration
is_read_only
last_snapshot
leader
monitor
observer_connection_stat_reset
observers
ruok
server_stats
set_trace_mask
stat_reset
stats
system_properties
voting_view
watch_summary
watches
watches_by_path
zabstate

点击每一个连接，会出现对应的统计和监控数据。

9.2 明细页

9.2.1 配置信息

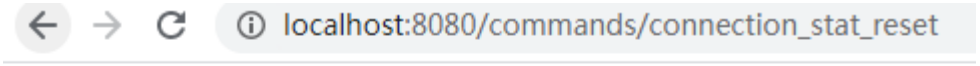
点击"configuration"连接，会出现服务器的基础配置信息，如下图：




```
{
  "client_port" : 2181,
  "data_dir" : "D:\\tmp\\adcc\\version-2",
  "data_log_dir" : "D:\\tmp\\adcc\\version-2",
  "tick_time" : 2000,
  "max_client_cnxns" : 60,
  "min_session_timeout" : 4000,
  "max_session_timeout" : 40000,
  "server_id" : 0,
  "client_port_listen_backlog" : -1,
  "command" : "configuration",
  "error" : null
}
```


9.2.2 连接统计重置

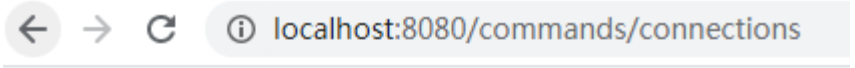
点击"connection_stat_reset"连接，会执行连接统计重置，并返回执行响应信息，如下图：




```
{
  "command" : "connection_stat_reset",
  "error" : null
}
```


9.2.3 连接信息

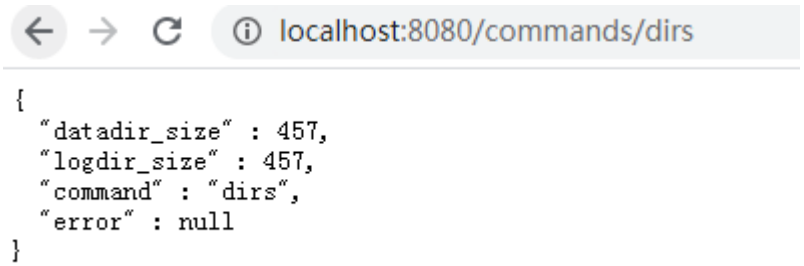
点击"connections"连接，响应信息包括显示已有的安全连接信息和非安全连接信息，如下图：




```
{
  "connections" : [ ],
  "secure_connections" : [ ],
  "command" : "connections",
  "error" : null
}
```

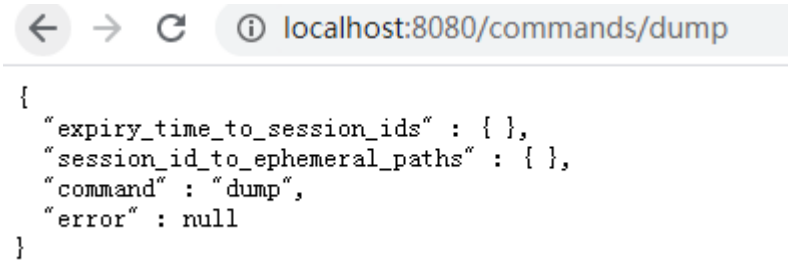

9.2.4 目录信息

点击"dirs"连接，响应信息包括数据目录信息和日志目录、命令执行状态，如下图：



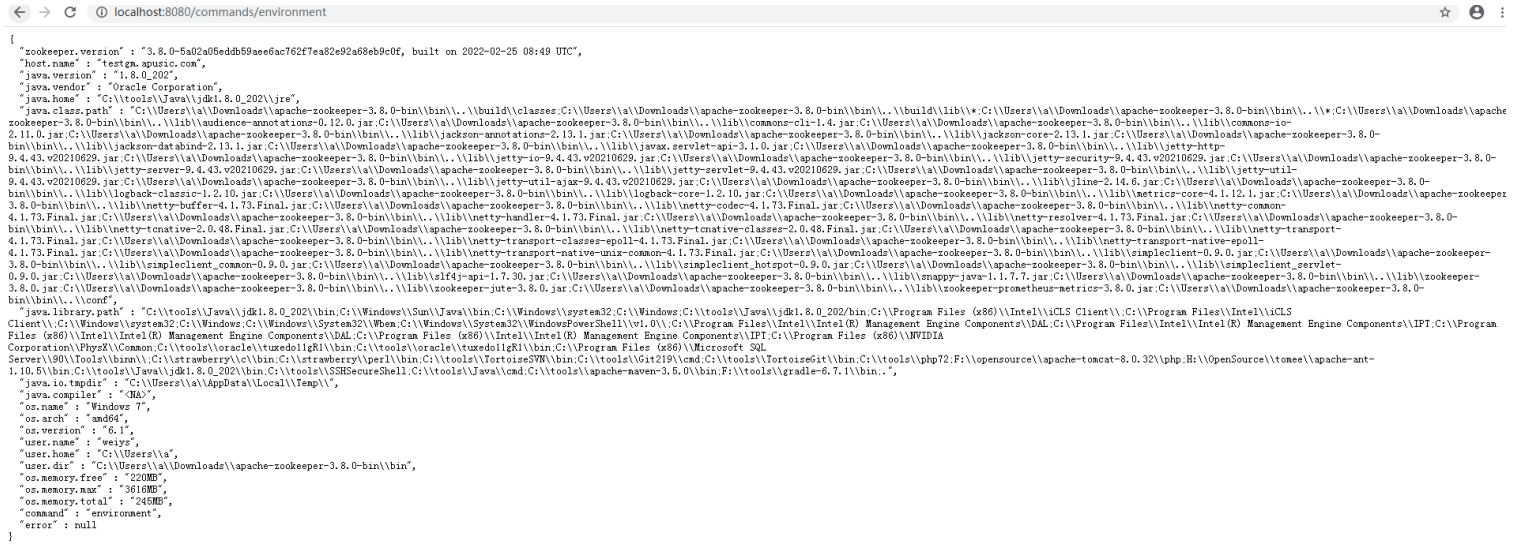
9.2.5 dump信息

点击"dump"连接，响应信息包括未经处理的会话及临时节点、命令执行状态，如下图：



9.2.6 环境信息

点击"environment"连接，响应信息Java运行环境属性、classPath等、命令执行状态，如下图：



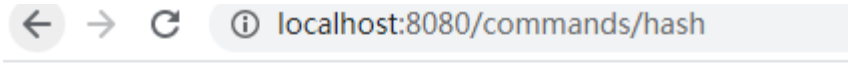
9.2.7 tracemask信息

点击"get_trace_mask"连接，响应信息为tracemask内容、命令执行状态，如下图：


```
{
  "tracemask" : 306,
  "command" : "get_trace_mask",
  "error" : null
}
```


9.2.8 hash信息

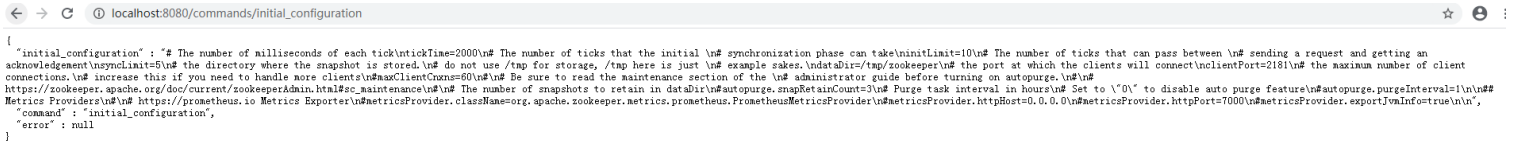
点击"hash"连接，响应信息为hash内容、命令执行状态，如下图：




```
{
  "digests" : [ ],
  "command" : "hash",
  "error" : null
}
```


9.2.9 初始化信息

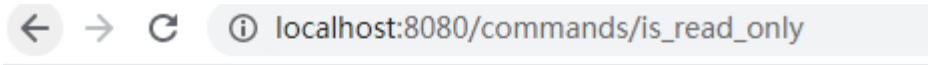
点击"initial_configuration"连接，响应信息为服务器初始化配置内容、命令执行状态，如下图：




```
{
  "initial_configuration" : "# The number of milliseconds of each tick\ntickTime=2000\n# The number of ticks that the initial \n# synchronization phase can take\ninitLimit=10\n# The number of ticks that can pass between \n# sending a request and getting an acknowledgement\nsyncLimit=5\n# the directory where the snapshot is stored.\n# do not use /tmp for storage, /tmp here is just \n# example sakes.\ndataDir=/tmp/zookeeper\n# the port at which the clients will connect\nclientPort=2181\n# the maximum number of client connections.\n# increase this if you need to handle more clients\n#maxClientCnxns=60\n#\n# Be sure to read the maintenance section of the \n# administrator guide before turning on autopurge.\n#\n# https://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_maintenance\n#\n# The number of snapshots to retain in dataDir\n#autopurge.snapRetainCount=3\n# Purge task interval in hours\n# Set to \"0\" to disable auto purge feature\n#autopurge.purgeInterval=1\n\n## Metrics Providers\n#\n# https://prometheus.io Metrics Exporter\n#metricsProvider.className=org.apache.zookeeper.metrics.prometheus.PrometheusMetricsProvider\n#metricsProvider.httpHost=0.0.0.0\n#metricsProvider.httpPort=7000\n#metricsProvider.exportJvmInfo=true\n\n",
  "command" : "initial_configuration",
  "error" : null
}
```


9.2.10 只读信息

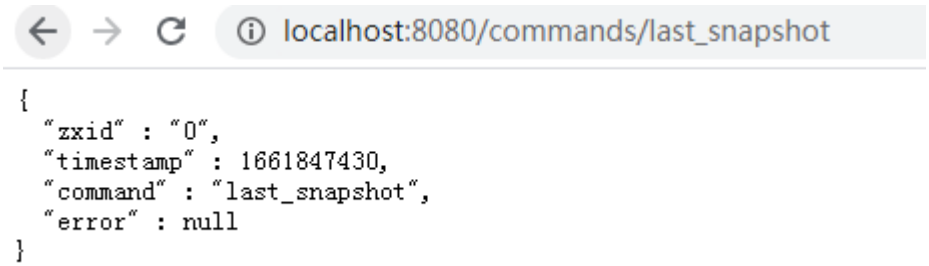
点击"is_read_only"连接，响应信息为服务器是否为只读、命令执行状态，如下图：




```
{
  "read_only" : false,
  "command" : "is_read_only",
  "error" : null
}
```

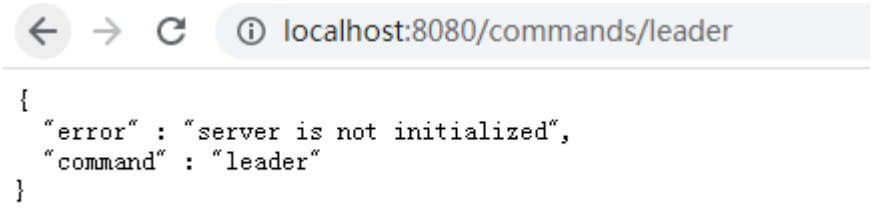

9.2.11 最新快照信息

点击"last_snapshot"连接，响应信息为服务器快照最新时间、命令执行状态，如下图：



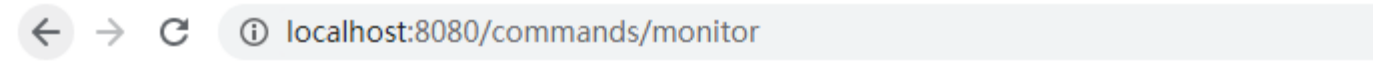
9.2.12 leader信息

点击"leader"连接，响应信息为leader的信息，如果是单机模式，则返回错误提示信息" server is not initialized"、命令执行状态，如下图：



9.2.13 监控信息

点击"monitor"连接，响应信息为服务器相关相关指标内容、命令执行状态，如下图：

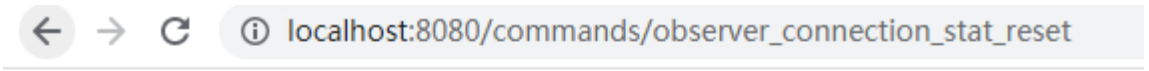



```
{
  "version" : "3.8.0-5a02a05eddb59aee6ac762f7ea82e92a68eb9c0f, built on 2022-02-25 08:49 UTC",
  "server_state" : "standalone",
  "ephemerals_count" : 0,
  "num_alive_connections" : 0,
  "avg_latency" : 0.0,
  "outstanding_requests" : 0,
  "znode_count" : 5,
  "global_sessions" : 0,
  "non_mtls_remote_conn_count" : 0,
  "last_client_response_size" : -1,
  "packets_sent" : 0,
  "packets_received" : 0,
  "max_client_response_size" : -1,
  "connection_drop_probability" : 0.0,
  "watch_count" : 0,
  "auth_failed_count" : 0,
  "min_latency" : 0,
  "max_file_descriptor_count" : -1,
  "approximate_data_size" : 44,
  "open_file_descriptor_count" : -1,
  "local_sessions" : 0,
  "uptime" : 1002907,
  "max_latency" : 0,
  "outstanding_tls_handshake" : 0,
  "min_client_response_size" : -1,
  "non_mtls_local_conn_count" : 0,
  "proposal_count" : 0,
  "watch_bytes" : 0,
  "outstanding_changes_removed" : 0,
  "throttled_ops" : 0,
  "stale_requests_dropped" : 0,
  "large_requests_rejected" : 0,
  "insecure_admin_count" : 0,
  "connection_rejected" : 0,
  "cnxn_closed_without_zk_server_running" : 0,
  "sessionless_connections_expired" : 0,
  "looking_count" : 0,
  "dead_watchers_queued" : 0,
  "stale_requests" : 0,
  "connection_drop_count" : 0,
  "learner_proposal_received_count" : 0,
  "digest_mismatches_count" : 0,
  "dead_watchers_cleared" : 0,
  "response_packet_cache_hits" : 0,
  "bytes_received_count" : 0,
  "add_dead_watcher_stall_time" : 0,
  "request_throttle_wait_count" : 0,
  "requests_not_forwarded_to_commit_processor" : 0,
  "response_packet_cache_misses" : 0,
  "ensemble_auth_success" : 0,
  "prep_processor_request_queued" : 0,
  "learner_commit_received_count" : 0,
  "stale_replies" : 0,
  "connection_request_count" : 0,
  "response_bytes" : 0,
  "ensemble_auth_fail" : 0,
  "diff_count" : 0,
  "response_packet_get_children_cache_misses" : 0,
  "connection_revalidate_count" : 0,
  "quit_leading_due_to_disloyal_voter" : 0,
  "snap_count" : 0,
  "unrecoverable_error_count" : 0,
```

```
  "unsuccessful_handshake" : 0,
```


9.2.14 观察者统计重置

点击"observer_connection_stat_reset"连接，充值观察者统计相关的数据，响应信息为服务器是否为只读、命令执行状态，如下图：




```
{
  "command" : "observer_connection_stat_reset",
  "error" : null
}
```

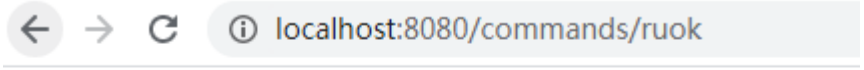

9.2.15 观察者信息

点击"observers"连接，响应信息为观察者信息、命令执行状态，如下图：


```
{
  "synced_observers" : 0,
  "observers" : [ ],
  "command" : "observers",
  "error" : null
}
```


9.2.16 服务器状态信息

点击"ruok"连接，响应信息为服务器状态信息、命令执行状态，错误为null则表示正常，如下图：




```
{
  "command" : "ruok",
  "error" : null
}
```

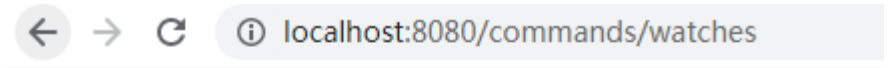

9.2.17 服务器统计信息

点击"server_stats"连接，响应信息为服务器状态统计信息、命令执行状态，错误为null则表示正常，如下图：


```
{
  "version" : "3.8.0-5a02a05eddb59aee6ac762f7ea82e92a68eb9c0f, built on 2022-02-25 08:49 UTC",
  "read_only" : false,
  "server_stats" : {
    "packets_sent" : 0,
    "packets_received" : 0,
    "fsync_threshold_exceed_count" : 0,
    "client_response_stats" : {
      "last_buffer_size" : -1,
      "min_buffer_size" : -1,
      "max_buffer_size" : -1
    },
    "num_alive_client_connections" : 0,
    "auth_failed_count" : 0,
    "non_mtlsremote_conn_count" : 0,
    "non_mtlslocal_conn_count" : 0,
    "provider_null" : false,
    "server_state" : "standalone",
    "avg_latency" : 0.0,
    "max_latency" : 0,
    "min_latency" : 0,
    "uptime" : 1131029,
    "last_processed_zxid" : 0,
    "outstanding_requests" : 0,
    "data_dir_size" : 457,
    "log_dir_size" : 457
  },
  "client_response" : {
    "last_buffer_size" : -1,
    "min_buffer_size" : -1,
    "max_buffer_size" : -1
  },
  "node_count" : 5,
  "command" : "server_stats",
  "error" : null
}
```


9.2.18 监视信息

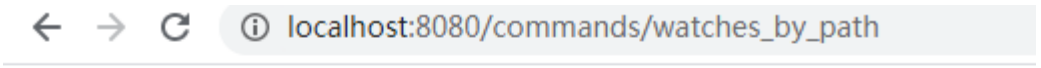
点击"watches"连接，响应信息为监视的相关路径信息、命令执行状态，如下图：




```
{
  "session_id_to_watched_paths" : { },
  "command" : "watches",
  "error" : null
}
```


9.2.19 路径监视信息

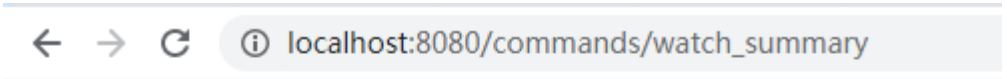
点击"watches_by_path"连接，响应信息为以路径为主显示监视信息、命令执行状态，如下图：




```
{
  "path_to_session_ids" : { },
  "command" : "watches_by_path",
  "error" : null
}
```


9.2.20 监视概要信息

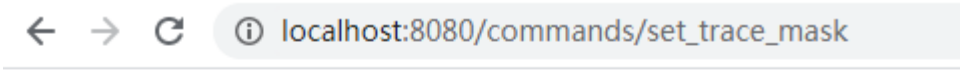
点击"watch_summary"连接，响应信息为服务器监视概要信息（连接数、路径数以及总共的监视数量）、命令执行状态，如下图：




```
{
  "num_connections" : 0,
  "num_paths" : 0,
  "num_total_watches" : 0,
  "command" : "watch_summary",
  "error" : null
}
```


9.2.21 Tracemask设置

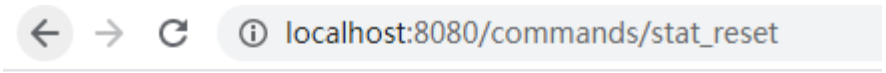
点击"set_trace_mask"连接，响应信息为是否成功设置服务器tracemask、命令执行状态，如下图：




```
{
  "error" : "setTraceMask requires long traceMask argument",
  "command" : "set_trace_mask"
}
```


9.2.22 统计重设

点击"stat_reset"连接，响应信息为服务器统计重设是否成功、命令执行状态，如下图：




```
{
  "command" : "stat_reset",
  "error" : null
}
```

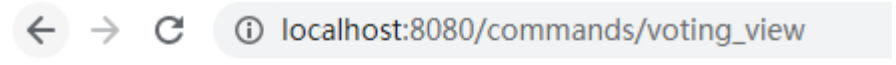

9.2.23 系统属性信息

点击"system_properties"连接，响应信息为服务器系统相关属信息、命令执行状态，如下图：


```
{
  "awt.toolkit" : "sun.awt.windows.WToolkit",
  "file.encoding" : "GBK",
  "file.encoding.pkg" : "sun.io",
  "file.separator" : "\\",
  "java.awt.graphicsenv" : "sun.awt.Win32GraphicsEnvironment",
  "java.awt.printerjob" : "sun.awt.windows.WPrinterJob",
  "java.class.path" : "C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\build\\classes;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\build\\lib\\*;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-
bin\\bin\\..\\*;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\audience-annotations-0.12.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\commons-cli-1.4.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-
bin\\bin\\..\\lib\\commons-io-2.11.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jackson-annotations-2.13.1.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jackson-core-2.13.1.jar;C:\\Users\\a\\Downloads\\apache-
zookeeper-3.8.0-bin\\bin\\..\\lib\\jackson-databind-2.13.1.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\javax.servlet-api-3.1.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jetty-http-
9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jetty-io-9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jetty-security-9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-
bin\\bin\\..\\lib\\jetty-server-9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jetty-servlet-9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jetty-util-
9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jetty-util-ajax-9.4.43.v20210629.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\jline-2.14.6.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-
bin\\bin\\..\\lib\\logback-classic-1.2.10.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\logback-core-1.2.10.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\metrics-core-4.1.12.1.jar;C:\\Users\\a\\Downloads\\apache-
zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-buffer-4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-codec-4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-common-
4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-handler-4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-resolver-4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-
bin\\bin\\..\\lib\\netty-tcnative-2.0.48.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-tcnative-classes-2.0.48.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-transport-
4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-transport-classes-epoll-4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-transport-native-epoll-
4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\netty-transport-native-unix-common-4.1.73.Final.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\simpleclient-0.9.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-
3.8.0-bin\\bin\\..\\lib\\simpleclient_common-0.9.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\simpleclient_hotspot-0.9.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\simpleclient_servlet-
0.9.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\slf4j-api-1.7.30.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\snappy-java-1.1.7.7.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\zookeeper-
3.8.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\zookeeper-jute-3.8.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-bin\\bin\\..\\lib\\zookeeper-prometheus-metrics-3.8.0.jar;C:\\Users\\a\\Downloads\\apache-zookeeper-3.8.0-
bin\\bin\\..\\conf",
  "java.class.version" : "52.0",
  "java.endorsed.dirs" : "C:\\tools\\Java\\jdk1.8.0_202\\jre\\lib\\endorsed",
  "java.ext.dirs" : "C:\\tools\\Java\\jdk1.8.0_202\\jre\\lib\\ext;C:\\Windows\\Sun\\Java\\lib\\ext",
  "java.home" : "C:\\tools\\Java\\jdk1.8.0_202\\jre",
  "java.io.tmpdir" : "C:\\Users\\a\\AppData\\Local\\Temp\\",
  "java.library.path" : "C:\\tools\\Java\\jdk1.8.0_202\\bin;C:\\Windows\\Sun\\Java\\bin;C:\\Windows\\system32;C:\\Windows;C:\\tools\\Java\\jdk1.8.0_202/bin;C:\\Program Files (x86)\\Intel\\iCLS Client\\;C:\\Program Files\\Intel\\iCLS
Client\\;C:\\Windows\\system32;C:\\Windows;C:\\Windows\\System32\\Wbem;C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\;C:\\Program Files\\Intel\\Intel(R) Management Engine Components\\DAL;C:\\Program Files\\Intel\\Intel(R) Management Engine Components\\IPT;C:\\Program
Files (x86)\\Intel\\Intel(R) Management Engine Components\\DAL;C:\\Program Files (x86)\\Intel\\Intel(R) Management Engine Components\\IPT;C:\\Program Files (x86)\\NVIDIA
Corporation\\PhysX\\Common;C:\\tools\\oracle\\tuxedo11gR1\\bin;C:\\tools\\oracle\\tuxedo11gR1\\bin;C:\\Program Files (x86)\\Microsoft SQL
Server\\90\\Tools\\binn\\;C:\\strawberry\\c\\bin;C:\\strawberry\\perl\\bin;C:\\tools\\TortoiseSVN\\bin;C:\\tools\\Git219\\cmd;C:\\tools\\TortoiseGit\\bin;C:\\tools\\php72;F:\\opensource\\apache-tomcat-8.0.32\\php;H:\\OpenSource\\tomee\\apache-ant-
1.10.5\\bin;C:\\tools\\Java\\jdk1.8.0_202\\bin;C:\\tools\\SSHSecureShell;C:\\tools\\Java\\cmd;C:\\tools\\apache-maven-3.5.0\\bin;F:\\tools\\gradle-6.7.1\\bin;.",
  "java.runtime.name" : "Java(TM) SE Runtime Environment",
  "java.runtime.version" : "1.8.0_202-b08",
```


9.2.24 选举视图信息

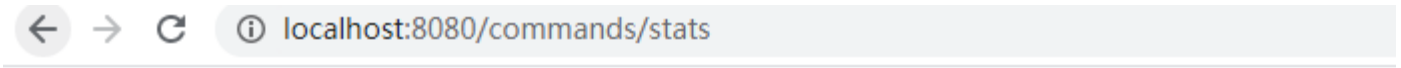
点击"voting_view"连接，响应信息为服务器当前视图配置，单机模式下为空、命令执行状态，如下图：




```
{
  "current_config" : { },
  "command" : "voting_view",
  "error" : null
}
```


9.2.25 统计信息

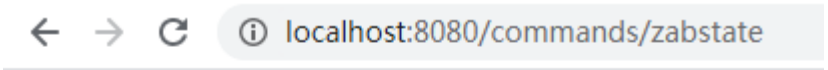
点击"stats"连接，响应信息为服务器请求响应的统计信息、命令执行状态，如下图：




```
{
  "version" : "3.8.0-5a02a05eddb59aee6ac762f7ea82e92a68eb9c0f, built on 2022-02-25 08:49 UTC",
  "read_only" : false,
  "server_stats" : {
    "packets_sent" : 0,
    "packets_received" : 0,
    "fsync_threshold_exceed_count" : 0,
    "client_response_stats" : {
      "last_buffer_size" : -1,
      "min_buffer_size" : -1,
      "max_buffer_size" : -1
    },
    "num_alive_client_connections" : 0,
    "auth_failed_count" : 0,
    "non_mtlsremote_conn_count" : 0,
    "non_mtlslocal_conn_count" : 0,
    "provider_null" : false,
    "server_state" : "standalone",
    "avg_latency" : 0.0,
    "max_latency" : 0,
    "min_latency" : 0,
    "uptime" : 1169462,
    "last_processed_zxid" : 0,
    "outstanding_requests" : 0,
    "data_dir_size" : 457,
    "log_dir_size" : 457
  },
  "client_response" : {
    "last_buffer_size" : -1,
    "min_buffer_size" : -1,
    "max_buffer_size" : -1
  },
  "node_count" : 5,
  "connections" : [ ],
  "secure_connections" : [ ],
  "command" : "stats",
  "error" : null
}
```


9.2.26 Zab统计信息

点击"zabstate"连接，响应信息为服务器zab执行的统计信息、命令执行状态，如下图:




```
{
  "voting" : false,
  "zabstate" : "",
  "command" : "zabstate",
  "error" : null
}
```

10 配额说明

ADCC for zk支持命名空间和字节进行配额，限制路径（包括其自身）下的子节点数量，或限制字节长度。可以使用ZooKeeperMain类来设置配额。如果用户超过分配给他们的配额，ADCC将会在日志中打印警告消息。

首选连接到服务器：

```
\$ bin/adccCli.sh -server 127.0.0.1:2181
```

连接服务器后，就可以执行操作。

10.1 设置配额

可以使用`set quota`在节点上设置配额。它可以选择使用`-n`（用于命名空间/计数）和`-b`（用于字节/数据长度）设置配额。

为了防止他人任意更改配额，用户可以设置ACL，限制访问操作。

配额用户集的范围是指指定路径下的所有节点，包括其自身。

在设置配额前，需要在配置文件`zoo.cfg`添加`enforceQuota=true`。

操作参考以下说明：

```
# -n 限制子节点的个数
[adcc: 127.0.0.1:2181(CONNECTED) 18] setquota -n 2 /quota_test
[adcc: 127.0.0.1:2181(CONNECTED) 19] create /quota_test/child_1
Created /quota_test/child_1
[adcc: 127.0.0.1:2181(CONNECTED) 20] create /quota_test/child_2
Created /quota_test/child_2
[adcc: 127.0.0.1:2181(CONNECTED) 21] create /quota_test/child_3
Created /quota_test/child_3
# 注意：-s是没有硬约束，只需要记录警告信息，因而只会在日志打印警告信息
2024-03-07 11:22:36,680 [myid:1] - WARN[SyncThread:0:DataTree@374] -
Quota exceeded: /quota_test count=3
limit=2
2024-03-07 11:22:41,861 [myid:1] - WARN[SyncThread:0:DataTree@374] -
Quota exceeded: /quota_test count=4
```

```
limit=2

# -b 限制路径字节数（数据长度）
[adcc: 127.0.0.1:2181(CONNECTED) 22] setquota -b 5 /brokers
[adcc: 127.0.0.1:2181(CONNECTED) 23] set /brokers "I_love_zookeeper"
# 注意：-b是没有硬约束，只需要记录警告信息，因而只会在日志打印警告信息
WARN [CommitProcWorkThread-7:DataTree@379] - Quota exceeded: /brokers
bytes=4206 limit=5

# -N 子节点硬配额
[adcc: 127.0.0.1:2181(CONNECTED) 3] create /c1
Created /c1
[adcc: 127.0.0.1:2181(CONNECTED) 4] setquota -N 2 /c1
[adcc: 127.0.0.1:2181(CONNECTED) 5] listquota /c1
absolute path is /zookeeper/quota/c1/zookeeper_limits
Output quota for /c1
count=-1,bytes=-1=;byteHardLimit=-1;countHardLimit=2
Output stat for /c1 count=2,bytes=0
[adcc: 127.0.0.1:2181(CONNECTED) 6] create /c1/ch-3
Count Quota has exceeded : /c1/ch-3

# -B 字节硬配额
[adcc: 127.0.0.1:2181(CONNECTED) 3] create /c2
[adcc: 127.0.0.1:2181(CONNECTED) 4] setquota -B 4 /c2
[adcc: 127.0.0.1:2181(CONNECTED) 5] set /c2 "foo"
[adcc: 127.0.0.1:2181(CONNECTED) 6] set /c2 "foo-bar"
Bytes Quota has exceeded : /c2
[adcc: 127.0.0.1:2181(CONNECTED) 7] get /c2
foo
```

10.2 查看配额

使用 `listquota` 查看节点的配额。

```
[adcc: 127.0.0.1:2181(CONNECTED) 1] listquota /adcc_test
absolute path is /zookeeper/quota/adcc_test/zookeeper_limits
```

```
Output quota for /adcc_test
count=2,bytes=5=;byteHardLimit=-1;countHardLimit=-1
Output stat for /adcc_test count=5,bytes=10
```

10.3 删除配额

使用 deletequota 删除节点的配额。

```
[adccshell: 1] delquota /quota_test
[adccshell: 2] listquota /quota_test
" absolute path" is /zookeeper/quota/quota_test/zookeeper_limits
" quota" for /quota_test does not exist.
[adccshell: 3] delquota -n /c1
[adccshell: 4] delquota -N /c2
[adccshell: 5] delquota -b /c3
[adccshell: 6] delquota -B /c4
```

11 配置说明

11.1 基本配置

下表为ADCC配置文件中的基本配置项。

配置项	说明	默认值
clientPort	侦听客户端连接的端口	2181
secureClientPort	指定ADCC服务器监听安全客户端连接的端口。clientPort指定明文连接的端口，而secureClientPort指定SSL连接的端口。同时指定这两个端口可以启用混合模式，而忽略其中一个则会禁用该模式	
observerMasterPort	用于监听观察者连接的端口；即观察者尝试连接的端口。如果设置了该属性，则服务器在从属模式下除了在领导模式下外还会承载观察者连接，并且在观察者模式下会尝试连接任何投票对等体	
tickTime	ADCC for zk使用的基本时间单位（毫秒）。它用于控制 ADCC服务器的心跳机制和会话超时等多个重要功能的时间间隔。简单来说，tickTime是 ADCC 进行各种时间相关操作的基本节拍。例如最小会话超时时间为两个心跳	2000
dataDir	存储内存中数据库快照的位置，除非另有规定，还存储数据库更新的事务日志。存放事务日志时建议选择专用的事务日志设置以确保性能稳定	../data

11.2 高级配置

下表的配置是可选项，可根据需要进行配置。

配置项	说明	默认值	
dataLogDir	储存事务日志的目录。此选项将指示计算机将事务日志写入dataLogDir而不是dataDir。这样可以使用专用的日志设备，并有助于避免日志记录和快照之间的竞争		
globalOutstandingLimit	客户端提交请求的速度可能超过ZooKeeper处理这些请求的速度，尤其是在客户端数量较多	1000	

	时。为了防止ZooKeeper因排队请求而耗尽内存，ZooKeeper会限制客户端的请求量，确保系统中未处理的请求数量不超过全局未处理请求限制		
preAllocSize	为避免查找，ADCC 在事务日志文件中以预分配大小（preAllocSize）千字节为单位分配空间。默认块大小为64M。改变块大小的一个原因是如果更频繁地进行快照，则减小块的大小。（另外，请参阅snapCount和snapSizeLimitInKb）。	64M	
snapCount	使用快照和事务日志（类似于预写日志）记录其事务。在可以进行快照（并滚动事务日志）之前，事务日志中记录的事务数量由 snapCount 确定。为了防止所有机器同时创建快照，当事务日志中的事务数量达到运行时生成的随机值[snapCount/2+1，snapCount]范围内时，每个ADCC服务器将创建一个快照	100000	
commitLogCount *	ADCC在跟随者没有严重落后的情况下，维护一个内存中的已提交请求列表，以便快速与跟随者同步。这在快照较大（>100,000）时提高了同步性能	500	
snapSizeLimitInKb	ADCC使用快照和事务日志（类似于预写日志）来记录其事务。在快照生成（以及事务日志滚动）之前，事务日志中记录的事务集合允许的最大总大小由snapSize决定。为了防止所有机器同时进行快照操作，当事务日志中的事务集合大小达到运行时生成的随机值[snapSize/2+1，snapSize]范围内的某个值时，每个ZooKeeper服务器将生成一个快照。每个文件系统都有一个最小标准文件大小，为了使此功能正常工作，所选数值必须大于该值。默认的snapSizeLimitInKb是4194304（4GB）。非正数将禁用此功能	4194304	
txnLogSizeLimitInKb	ADCC事务日志文件也可以通过txnLogSizeLimitInKb更直接地控制。较大的事务日志会导致使用事务日志同步时，从者同步速度变慢。这是因为领导者需要扫描磁盘上的相应日志文件以找到要开始同步的事务。此功能默认是关闭的，snapCount和snapSizeLimitInKb是唯一限制事务日志大小的值。启用后，当任何限制被		

	达到时，ADCC将滚动日志。请注意，实际的日志大小可能会超出此值，超出部分取决于序列化事务的大小。另一方面，如果此值设置得太接近（或小于）预分配大小，可能会导致ADCC对每个事务都滚动日志。虽然这不是一个正确性问题，但可能会导致性能严重下降。为了避免这种情况并充分利用此功能，建议将值设置为N*预分配大小，其中N >= 2		
maxCnxns	限制可以连接到ADCC服务器的并发连接总数（每个服务器的每个客户端的端口）。这用于防止某些类型的拒绝服务攻击。默认值为0，将其设置为0将完全取消对并发连接总数的限制。服务器CnxnFactory和安全服务器CnxnFactory的连接数分别计算，因此只要类型合适，一个节点最多可容纳2*maxCnxns个连接	0	
maxClientCnxns	限制由 IP 地址标识的单个客户端对 ADCC 集群的单个成员可能建立的并发连接数（在套接字级别）。这用于防止某些类型的拒绝服务攻击，包括文件描述符耗尽。将此设置为0将完全取消对并发连接的限制	60	
clientPortAddress	侦听客户端连接的地址（ipv4、ipv6 或主机名）；也就是说，客户端尝试连接到的地址。这是可选的，默认情况下，我们以这样一种方式绑定，即服务器上任何地址 / 接口 /nic 的任何与 clientPort 的连接都将被接受		
minSessionTimeout	服务器允许客户端协商的最小会话超时，单位为毫秒。默认值为tickTime的两倍		
maxSessionTimeout	服务器允许客户端协商的最大会话超时时间，单位为毫秒。默认值为tickTime的20倍		
fsync.warningthresholdms	每当事务日志（WAL）中的同步操作（fsync）花费的时间超过此值时，将向日志输出一条警告消息。该值以毫秒为单位指定，默认为 1000。此值只能作为系统属性进行设置。	1000	
maxResponseCacheSize	当设置为正整数时，它决定了存储最近读取记录的序列化形式的缓存的大小。有助于节省流行znode上的序列化成本。指标response_packet_cache_hits和response_packet_cache_misses可用于将此值调整为给定的工作负载。该功能	400	

	默认打开，值为400，设置为0或负整数以关闭该功能		
maxGetChildrenResponseCacheSize	类似于maxResponseCacheSize，但适用于获取子请求。response_packet_get_children_cache_hits和response_packet_get_children_cache_misses的指标可用于将此值调整为给定的工作负载。默认情况下，该功能打开，值为400，设置为0或负整数以关闭该功能	400	
autopurge.snapRetainCount	启用后，ADCC自动清除功能会分别在dataDir和dataLogDir中保留autopurge. snapRetainCount最近的快照和相应的事务日志，并删除其余部分。默认为3。最小值为3	3	
autopurge.purgeInterval	触发清除任务所需的时间间隔，以小时为单位。设置为正整数（1及以上）可启用自动清除	0	
syncEnabled	观察者现在默认像参与者一样将事务和写入快照记录到磁盘。这减少了观察者在重新启动时的恢复时间。将此设置为“false”可禁用此功能	true	
extendedTypesEnabled	定义为true以启用扩展功能，例如创建TTL节点。重要提示：启用时，由于内部限制，服务器ID必须小于255	false	
watchManagerName	定义watch manager的名称。使用WatchManagerOptimized，用于优化大量使用watch的场景下的内存开销时设置		
watcherCleanThreadsNum	管理器WatchManagerOptimized会延迟清理已死亡的观察者。此配置用于决定观察者清理器中使用的线程数量。更多的线程通常意味着更高的清理吞吐量。默认值为2，即使在大量且连续的会话关闭/重建情况下也足够好	2	
watcherCleanThreshold	管理器WatchManagerOptimized将延迟清理已死亡的watcher，清理过程相对耗时，批量处理可以降低成本并提高性能。此设置用于决定批量大小，默认值为1000，如果没有内存或清理速度问题，无需更改	1000	
watcherCleanIntervalInSeconds	管理器WatchManagerOptimized会延迟清理已死亡的观察者，清理过程相对复杂，批量处理可以降低成本并提高性能。除了watcherCleanThreshold外，此设置还用于在特定时间后	10分钟	

	清理已死亡的观察者，即使这些观察者未超过watcherCleanThreshold，这样可以防止它们长时间存在。默认设置为10分钟，通常无需更改		
maxInProcessingDeadWatchers	此设置用于控制WatcherCleaner中可以拥有的积压数量，当达到这个数值时，它会减缓将死掉的观察者添加到WatcherCleaner的速度，进而减缓观察者的添加和关闭速度，从而避免内存不足问题。默认情况下没有限制，可以将其设置为类似watcherCleanThreshold * 1000的值		
bitHashCacheSize	这是用于在 BitHashSet 实现中决定 HashSet 缓存大小的设置。如果没有 HashSet，我们需要使用 O(N)的时间来获取元素，N 是 elementBits 中的位数。但是我们需要保持较小的尺寸，以确保它在内存方面不会消耗太多，在内存和时间复杂度之间存在一种权衡。默认值是10，这是一个相对合理的缓存大小。		
fastleader.minNotificationInterval	领导选举中两次连续通知检查之间时间长度的下限。此间隔决定了对等节点等待检查选举投票集的时间，影响选举解决的速度。对于长时间的选举，该间隔遵循从配置的最小值（此）到最大值（fastleader.maxNotificationInterval）的退避策略		
fastleader.maxNotificationInterval	领导者选举中两次连续通知检查之间时间长度的上限。此间隔决定了对等节点等待检查选举投票集的时间，影响选举解决的速度。对于长时间的选举，该间隔遵循从配置的最小值（fastleader.minNotificationInterval）到最大值（此值）的退避策略		
connectionMaxTokens	用于调整服务器端连接限制器的参数之一，该限制器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。此参数定义了token-bucket中的最大token数。当设置为0时，将禁用限制	0	
connectionTokenFillTime	用于调整服务器端连接限流器的参数之一，该限流器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。该参数定义token-	1	

	bucket以毫秒为单位重新填充 connectionTokenFillCount token的时间间隔		
connectionTokenFillCount	用于调整服务器端连接限制器的参数之一，该限制器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。此参数定义每connectionTokenFillTime毫秒要添加到token-bucket的token数量	1	
connectionFreezeTime	用于调整服务器端连接限制器的参数之一，该限制器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。此参数定义了以毫秒为单位的时间间隔，在此时间间隔内调整丢弃概率。当设置为 -1 时，禁用概率性丢弃。	-1	
connectionDropIncrease	用于调整服务器端连接限制器的参数之一，该限制器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。此参数定义要增加的丢弃概率。限制器每connectionFreezeTime毫秒检查一次，如果token-bucket为空，丢弃概率将增加connectionDropIncrease	0.02	
connectionDropDecrease	用于调整服务器端连接限制器的参数之一，该限制器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。 此参数定义要减少的丢弃概率。限制器每connectionFreezeTime毫秒检查一次，如果token-bucket的token数量超过阈值，则丢弃概率将减少connectionDropDecrease。阈值为connectionMaxTokens * connectionDecreaseRatio	0.02	
connectionDecreaseRatio	用于调整服务器端连接限制器的参数之一，该限制器是一种基于token的速率限制机制，具有可选的概率性丢弃功能。此参数定义了降低丢弃概率的阈值。	0	
zookeeper.connection_throttle_weight_enabled	在进行限流时是否考虑连接权重。仅在启用连接限流时有用，即 connectionMaxTokens 大于 0。	false	
zookeeper.connection_throttle_global_session_weight	全局会话的权重。它是全局会话请求通过连接限流器所需的token数量。它必须是一个不小于本地会话权重的正整数。	3	

zookeeper.connection_throttle_local_session_weight	本地会话的权重。它是本地会话请求通过连接限流器所需的token数量。它必须是一个不大于全局会话或更新会话权重的正整数。	1	
zookeeper.connection_throttle_renew_session_weight	更新会话的权重。它也是重新连接请求通过限流器所需的token数量。它必须是一个不小于本地会话权重的正整数。	2	
clientPortListenBacklog	ADCC服务器套接字的套接字队列长度。此值控制将由ADCC服务器处理的服务器端请求队列长度。超过此长度的连接将收到网络超时（30秒），这可能会导致ADCC会话过期问题。默认情况下，此值未设置（-1），在Linux上使用50的队列长度。此值必须是正数		
serverCnxnFactory	指定serverCnxnFactory实现。若要使用基于TLS的服务器通信，应将其设置为NettyServerCnxnFactory	NIOServerCnxnFactory	
flushDelay	延迟提交日志刷新的时间，单位为毫秒。不会影响由maxBatchSize定义的限制。默认情况下禁用（值为0）。具有高写入速率的集合可能通过使用10-20 ms的值来提高吞吐量	0	
maxWriteQueuePollTime	如果启用了flushDelay，则此属性确定在没有新请求排队时等待以进行刷新的时间，单位为毫秒。默认情况下，该属性设置为flushDelay/3（默认情况下隐式禁用）		
maxBatchSize	在触发提交日志刷新之前，服务器允许的事务数。不会影响flushDelay定义的限制	1000	
enforceQuota	强制执行配额检查。当启用此功能且客户端超过某个znode下的总字节数或子节点数量硬性配额时，服务器将拒绝请求，并强制向客户端返回QuotaExceededException	false	
requestThrottleLimit	请求限流器开始停滞之前允许的未处理请求数的总数。设置为0时，将禁用限流	0	
requestThrottleStallTime	线程等待被通知可以继续处理请求的最长时间（以毫秒为单位）	100	
requestThrottleDropStale	启用此选项后，限流器将丢弃过期请求，而不是将其发送到请求管道。过期请求是指由已关	true	

	闭的连接发送的请求，和/或请求延迟时间超过会话超时时间的请求		
requestStaleLatencyCheck	如果启用，当请求延迟高于其关联会话超时时间时，将认为该请求已过期。默认情况下禁用	disabled	
requestStaleConnectionCheck	启用后，如果请求的连接已关闭，则该请求将被视为过期。默认启用。	enabled	
zookeeper.request_throttler.shutdownTimeout	请求限流器在关闭期间等待请求队列排空的时间（以毫秒为单位），超过该时间后它将强制关闭。	10000	
advancedFlowControlEnabled	在Netty中基于ADCC管道状态使用精确的流量控制，以避免直接缓冲区内存溢出。这将禁用Netty中的AUTO_READ。		
enableEagerACLCheck	设置为“true”时，可在将写入请求发送到定额之前，在每个本地服务器上对写入请求启用即时ACL检查	false	
maxConcurrentSnapSyncs	领导者或跟随者可以同时服务的最大快照同步数	10	
maxConcurrentDiffSyncs	领导者或跟随者同时可以处理的差异同步的最大数量	100	
digest.enabled	增加摘要功能是为了在从磁盘加载数据库、追赶并跟随领导者时，检测ADCC内部的数据不一致情况，它会根据上述提到的adHash论文对数据树进行增量哈希检查。	true	
snapshot.compression.method	此属性控制ADCC在将快照存储到磁盘之前是否对其进行压缩。可以设置为： "" :禁用（不压缩快照）。默认值 "gz"：将打包成gzip格式 "Snappy":使用snappy压缩		
audit.enable	是否启用审计日志功能，默认情况下审计日志处于禁用状态，设置为"true"可以开启	false	
audit.impl.class	实现审核记录器的类。默认情况下使用基于log4j的审核记录器org.apache.zookeeper.audit		
largeRequestMaxBytes	所有正在处理的大型请求的最大字节数。如果即将到来的大型请求导致超过此限制，连接将被关闭。	100*1024*1024	

largeRequestThreshold	设置请求被视为大请求的大小阈值。如果该值为 -1，则所有请求都被视为小请求，从而有效地关闭大请求限制。	-1	
outstandingHandshake.limit	ADCC中可能存在的最大飞行中TLS握手连接数，超过此限制的连接将在开始握手前被拒绝。此设置并不限制最大TLS并发数，但有助于避免在飞行中TLS握手过多时因TLS握手超时而产生的羊群效应。将其设置为250左右就足以避免羊群效应。		
throttledOpWaitTime	请求在RequestThrottler队列中等待的时间超过此时间后，请求将被标记为已限制。被限制的请求除了被送入其所属服务器的处理流程以保持所有请求的顺序外，不会被进一步处理。FinalProcess将为这些未消化的请求发出错误响应（新错误代码：ZTHROTTLEDOP）。目的是让客户端不要立即重试它们。当设置为0时，不会限制任何请求。	0	
learner.closeSocketAsync	启用后，学习者将异步关闭多数派套接字。这在TLS连接中非常有用，因为关闭套接字可能需要很长时间，会阻塞关闭过程，可能导致新的领导者选举延迟，并使多数派不可用。异步关闭套接字可以避免阻塞关闭过程，即使关闭时间较长，也可以在关闭套接字的同时启动新的领导者选举	false	
leader.closeSocketAsync	启用后，领导者会异步关闭法定人数套接字。这在TLS连接中非常有用，因为关闭套接字可能需要很长时间。如果由于SyncLimitCheck失败而在ping（）中断开跟随者，则长时间的套接字关闭时间将阻止向其他跟随者发送心跳。如果没有收到心跳，其他跟随者将不会向领导者发送会话信息，导致会话过期。将此标志设置为true可确保定期发送心跳	false	
learner.asyncSending	Learner中的发送和接收数据包是在临界区同步完成的。不合时宜的网络问题可能会导致跟随者挂起。将Learner中的发送数据包移动到单独的线程并异步发送数据包。使用此参数启用设计（learner. asyncSending）		

forward_learner_requests_to_commit_processor_disabled	当设置此属性时，来自学习者的请求将不会被排入提交处理器队列，这将有助于节省领导者上的资源和垃圾回收时间。	false	

11.3 集群配置

下表中的配置项针对集群环境，可在集群中进行配置。

配置项	说明	默认值	
maxTimeToWaitForEpoch	激活领导者时，等待投票者提供时期信息的最长时间。如果领导者从其某个投票者处收到 “LOOKING”（寻找）通知，并且在 “maxTimeToWaitForEpoch”（等待时期的最长时间）内未从大多数选民处收到时期数据包，那么它将进入 “LOOKING”（寻找）状态并重新选举领导者。可以对此进行调整，以减少仲裁或服务器不可用时间，该时间可以设置为远小于 “initLimit * tickTime”。在跨数据中心环境中，可以将其设置为 2 秒左右。		
initLimit	允许追随者连接和同步到领导者的时间量，以跳动刻度为单位（参见tickTime）。如果ADCC管理的数据量很大，可根据需要增大此值。		
connectToLearnerMasterLimit	允许追随者在领导者选举后连接到领导者的时长，以跳动刻度为单位（参见tickTime）。默认为 initLimit 的值。当initLimit值较高时使用，这样连接到learner主节点时不会导致更长的超时时间		
leaderServes	领导者接受客户端连接。默认值为 "yes"。领导者机器协调更新。为了提高更新吞吐量，尽管会略微降低读取吞吐量，可以配置领导者不接受客户端并专注于协调。此选项的默认设置为"yes"，这意味着领导者将接受客户端连接。当集群中包含超过三个ADCC服务器时，强烈建议启用领导者选择	yes	
server.x=[hostname]:nnnnn[:nnnnn] etc	组成 ADCC 集群的服务器。当服务器启动时，它通过查找数据目录中的myid文件来确定自身的位置。该文件包含以ASCII编码表示的服务器编号，应与此设置左侧的server.x中的x匹配。客户端使用的ADCC服务器列表必须与每个AD		

	CC服务器拥有的ADCC服务器列表相匹配。有两个端口号nnnnn。第一个用于跟从者连接到领导者，第二个用于领导者选举。示例： server.1=172.20.1.1:2888:3888 server.2=172.20.1.2:2888:3888 server.3=172.20.1.3:2888:3888 如果想在同一台机器上测试多个服务器，则可以为每个服务器使用不同的端口。同时，ADCC服务器支持指定多个地址。当服务器使用多个物理网络接口时，ADCC能够在所有接口上绑定，并在发生网络错误时切换到工作接口。不同的地址可以在配置文件中使用管道（'|'）字符指定。示例： server.1=adcc1-net1:2888:3888|adcc1-net2:2889:3889 server.2=adcc2-net1:2888:3888|adcc2-net2:2889:3889 server.3=adcc3-net1:2888:3888|adcc3-net2:2889:3889		
syncLimit	设置集群中的 Follower 服务器与 Leader 服务器之间请求和应答之间能容忍的最多心跳数（tickTime的数量）。如果Follower落后于Leader太多，它们将被删除	5	
group.x=nnnnn[:nnnnn]	启用层次级联构建。“x”是组标识符，而“=”符号后的数字对应服务器标识符。赋值左侧是一系列用冒号分隔的服务器标识符。请注意，组必须互不相交，所有组的并集必须构成ADCC集合		
weight.x=nnnnn	与"group"一起使用时，它会在形成仲裁时为服务器分配权重。这个值就是在投票时服务器的权重。ADCC有几个部分需要进行投票，比如领导者选举和原子广播协议。默认情况下，服务器的权重为1。如果配置定义了组，但没有定义权重，那么所有服务器都将被赋予值1。		
cnxTimeout	设置用于打开连接以接收领导者选举通知的超时值。仅适用于选举算法3	5秒	
quorumCnxnTimeoutMs	设置用于领导者选举通知的连接的读取超时值。仅适用于选举算法3。默认值为-1，然后将使用syncLimit * tickTime作为超时	7-1	

standaloneEnabled	当设置为 false 时，单个服务器可以以复制模式启动，单个参与者可以与观察者一起运行，集群可以重新配置为单节点，也可以从单节点重新配置。出于向后兼容性考虑，默认值为 true。可以使用 QuorumPeerConfig 的 setStandaloneEnabled 方法进行设置，也可以在服务器的配置文件中添加 “standaloneEnabled=false” 或 “standaloneEnabled=true”。	true	
reconfigEnabled	是否启用动态重置配置功能，启用该功能后，用户可以通过 ADCC客户端 API 或通过 ADCC命令行工具执行重新配置操作，假设用户被授权执行此类操作。当该功能被禁用时，任何用户，包括超级用户，都不能执行重新配置。任何重新配置的尝试都会返回错误。“reconfigEnabled” 选项可以设置为 “reconfigEnabled=false” 或 “reconfigEnabled=true” 到服务器的配置文件，或使用QuorumPeerConfig的setReconfigEnabled方法。如果设置该属性，则该值在集群中的每台服务器上应保持一致。如果集群中存在一些服务器上的值设置为 true，一些服务器上的值设置为 false 的情况，将导致根据选举出的领导者不同而产生不一致的行为。如果领导者的设置为 “reconfigEnabled=true”，集群将启用重新配置功能。如果领导者的设置为 “reconfigEnabled=false”，集群将禁用重新配置功能。因此，建议在集群中的服务器之间为 “reconfigEnabled” 设置一致的值	false	
4lw.commands.whitelist	用户想要使用的以逗号分隔的四个字母单词命令列表。有效的四字母单词命令必须包含在此列表中，否则ADCC服务器将不会启用该命令。默认情况下，白名单仅包含adccServer.sh使用的“srvr”命令。其余的四字单词命令默认是禁用的：尝试使用它们将得到响应“……未执行，因为它不在白名单中。”的提示。下面是启用stat、ruok、conf和isro命令同时禁用其余四个字母单词命令的配置示例: 4lw.commands.whitelist=stat, ruok, conf, isro 如果确实需要默认启用所有四个字母单词命令，可以使用星号*		
tcpKeepAlive	设置为true可以在用于执行选举的quorum成员的套接字上设置TCP keepAlive标志。中断网络	false	

	基础设施的情况下确保quorum成员之间的连接保持畅通。一些网络地址转换（NAT）设备和防火墙可能会终止长时间运行或空闲的连接，或丢失其状态。 启用此选项依赖于操作系统级别的设置才能正常工作，有关更多信息，请查看操作系统中与TCP保活相关的选项。		
clientTcpKeepAlive	设置为true可以在客户端套接字上启用TCP keepAlive标志。一些损坏的网络基础设施可能会丢失从正在关闭的客户端发送的FIN数据包。这些从未关闭的客户端套接字会导致操作系统资源泄漏。启用此选项通过空闲检查终止这些僵尸套接字。启用此选项依赖于操作系统级别的设置才能正常工作，请参阅操作系统的TCP keepAlive设置以获取更多信息。默认值为false。请注意它与TCP keepAlive的区别。它适用于客户端套接字，而TCP keepAlive则用于由quorum成员使用的套接字。目前，此选项仅在使用默认NIOServerCnxnFactory时可用	false	
electionPortBindRetry	设置ADCC服务器绑定领导者选举端口失败时的最大重试次数。在出现短暂错误的情况下，此属性可以提高ADCC服务器的可用性并帮助其自我恢复。在容器环境中，尤其是在Kubernetes中，应将此值增加或设置为0（无限重试），以克服与DNS名称解析相关的问题。	3	
observer.reconnectDelayMs	当观察者与领导者失去连接时，它会等待指定的时长，然后再尝试重新连接领导者，这样整个观察者集群就不会同时尝试进行领导者选举并重新连接领导者。	0ms	
observer.election.DelayMs	断开连接时，延迟观察者参与领导者选举，以防止在该过程中给投票对等节点带来意外的额外负载。	200ms	
localSessionsEnabled and localSessionsUpgradingEnabled	通过设置 localSessionsEnabled=true 打开本地会话功能。打开 localSessionsUpgradingEnabled 可以根据需要自动将本地会话升级为全局会话（例如创建临时节点），这仅在启用 localSessionsEnabled 时才重要	false	

11.4 安全性配置

下表为允许控制服务执行的加密/身份验证/授权安全相关的配置。

配置项	说明	默认值	
DigestAuthenticationProvider.enabled	确定是否启用摘要身份验证提供程序	true	
DigestAuthenticationProvider.superDigest	允许 ADCC 集群管理员以"超级"用户身份访问 znode 层级结构。特别地，对于通过超级用户身份验证的用户，不会进行 ACL 检查。org.apache.zookeeper.server.auth.DigestAuthenticationProvider可用于生成superDigest，并用一个参数"super："调用它。在启动集群中每个服务器时，提供生成的"super："作为系统属性值。当向ADCC服务器（从ADCC客户端）进行身份验证时，传递方案为"digest"和身份数据为"super："。请注意，digest身份验证会将身份数据以明文形式传递给服务器，因此仅应在本地主机（不通过网络）或加密连接上使用此身份验证方法		
DigestAuthenticationProvider.digestAlg	设置ACL摘要算法		
X509AuthenticationProvider.superUser	通过SSL支持的方式，使ADCC集群管理员能够以"超级"用户身份访问znode层次结构。当此参数设置为X500主体名称时，只有具有该主体的经过身份验证的客户端才能绕过ACL检查，并对所有znode拥有完全权限。		
zookeeper.superUser	类似于zookeper.X509AuthenticationProvider.SuperUser，但对于基于SASL的登录是通用的。它将可以访问znode层次结构的用户的名称存储为"super"用户。可以使用zookeper.SuperUser.[后缀]表示法指定多个SASL超级用户，例如：zookeer.SuperUser.1=...		
ssl.authProvider	指定用于安全客户端身份验证的org. apache.zookeper.auth.X509AuthenticationProvider的子类。这在不使用JKS的证书密钥基础结构中很有用。可能需要扩展javax.net.ssl.X509KeyManager和javax.net.ssl.X509TrustManager以从SSL堆栈中获取所需的行为。要将ADCC服务器配置为使用自定义提供程序进行身份验证，请为自定义AuthenticationProvider选择方案名		

	称，并将属性zookeeper.AuthProvider.[scheme]设置为自定义实现的完全限定类名。这会将提供程序加载到ProviderRegister中。然后设置此属性zookeeper.ssl.AuthProvider=[scheme]，该提供程序将用于安全身份验证		
zookeeper.ensembleAuthName	指定一个由逗号分隔的集群有效names/aliases列表。客户端可以提供其打算连接的集群名称，作为 “ensemble” 方案的凭证。EnsembleAuthenticationProvider 将根据接收连接请求的集群的names/aliases列表检查该凭证。如果凭证不在列表中，连接请求将被拒绝。这可防止客户端意外连接到错误的集群。		
sessionRequireClientSASLAuth	当设置为true时，ADCC服务器将只接受来自已通过SASL向服务器进行身份验证的客户端的连接和请求。未配置SASL身份验证，或配置了SASL但身份验证失败（即凭据无效）的客户端将无法与服务器建立会话。在这种情况下，将返回一个类型错误代码（-124），此后Java和C客户端都将关闭与服务器的会话，而不会进一步尝试重新尝试重新连接。此配置是enforce.auth.enabled=true 和 enforce.auth.scheme=sasl的简写。默认情况下，此功能被禁用。想要选择加入的用户可以通过将sessionRequireClientSASLAuth设置为true来启用该功能。此功能覆盖了zookeeper.allowSaslFailedClients选项，因此即使服务器配置为允许未能通过SASL身份验证的客户端登录，如果启用此功能，客户端也将无法与服务器建立会话		
enforce.auth.enabled	当设置为true时，ADCC服务器仅接受通过配置的认证方案与服务器身份验证过的客户端连接和请求。认证方案可以通过属性enforce.auth.schemes配置。未配置任何服务器上已配置的认证方案或配置但认证失败（即使用无效凭证）的客户端将无法与服务器建立会话。在这种情况下，将返回一个类型错误代码（-124），Java和C客户端随后将关闭与服务器的会话，不再尝试重新连接。默认情况下，此功能处于禁用状态。可以通过将enforce.auth.enabled设置为true来启用此功能。当enforce.auth.enabled=true和enforce.auth.schemes=sasl配置被覆盖时，即使服务器配置为允许未能通过		

	SASL身份验证的客户端登录，如果启用了SASL作为身份验证方案，客户端也无法与服务器建立会话		
enforce.auth.schemes	以逗号分隔的认证方案列表。客户端必须使用至少一种认证方案进行身份验证，才能执行ADCC操作。此属性仅在enforce.auth.enabled为true时使用		
sslQuorum	启用加密的quorum通信。启用此功能时，请同时考虑启用leader.closeSocketAsync和learner.closeSocketAsync，以避免在关闭SSL连接时可能出现的可能较长的套接字关闭时间相关问题。值为true或false	false	
ssl.keyStore.location	指定密钥库的文件路径		
ssl.keyStore.password	配置密钥库的密码		
ssl.trustStore.location	指定信任库的文件路径		
ssl.trustStore.password	设置信任库的密码		
serverCnxnFactory	指定使用 Netty 作为服务器连接工厂，以支持 SSL 通信，通常用于集群环境。值为org.apache.zookeeper.server.NettyServerCnxnFactory		
sslQuorum	启用服务器之间的 SSL 通信，通信，通常用于集群环境。值为true或false		
sslQuorum.keyStore.location	指定服务器之间通信的密钥库路径，通常用于集群环境		
sslQuorum.keyStore.password	指定服务器之间通信的密钥库密码，通常用于集群环境		
sslQuorum.trustStore.location	指定服务器之间通信的信任库路径，通常用于集群环境		
sslQuorum.trustStore.password	指定服务器之间通信的信任库密码，通常用于集群环境		
ssl.trustStore.type 和 ssl.quorum.trustStore.type	指定客户端或集群环境下的节点的信任存储的文件格式。取值：JKS、PEM、PKCS12 或 null（通过文件名检测）。	null	

ssl.protocol 和ssl.quorum.protocol	指定客户端与 ADCC服务器或ADCC 集群中各个节点（Quorum 节点）之间建立安全连接时所采用的 SSL/TLS 协议。它明确了在客户端和服务器进行通信时，用于加密和身份验证的协议版本	TLSv1.2	
ssl.enabledProtocols和ssl.quorum.enabledProtocols	配置客户端与 ADCC 服务器或集群中各个节点（Quorum 节点）之间通信时所启用的 SSL/TLS 协议版本。该参数决定了在建立安全连接时，客户端和服务器能够协商使用哪些协议版本。默认值为协议属性的值		
ssl.ciphersuites和ssl.quorum.ciphersuites	配置客户端与 ADCC服务器或ADCC 集群中各个节点（Quorum 节点）之间建立 SSL/TLS 连接时允许使用的加密套件列表。加密套件定义了在 SSL/TLS 握手过程中使用的加密算法、密钥交换算法、消息认证码（MAC）算法等，它决定了数据在传输过程中的加密方式和安全级别	启用的密码套件取决于所使用的Java运行时版本	
ssl.context.supplier.class和ssl.quorum.context.supplier.class	配置客户端与 ADCC服务器或ADCC 集群中各个节点（Quorum 节点）之间建立 SSL/TLS 连接时创建SSL上下文的类。当遇到以下问题时可以使用自定义的类: 1.使用硬件密钥库，使用PKCS11或类似的东西加载。 2.无权访问软件密钥库，但可以从其容器中检索已构建的SSLContext。默认值：空		
ssl.hostnameVerification和ssl.quorum.hostnameVerification	客户端与 ADCC服务器或ADCC 集群中各个节点（Quorum 节点）之间建立SSL/TLS 连接时是否启用主机名验证。建议在测试环境中才禁用此功能	true	
ssl.crl和ssl.quorum.crl	客户端与 ADCC服务器或ADCC 集群中各个节点（Quorum 节点）之间建立SSL/TLS 连接时是否启用证书吊销列表	false	
ssl.ocsp 和 ssl.quorum.ocsp	客户端与 ADCC服务器或ADCC 集群中各个节点（Quorum 节点）之间建立SSL/TLS 连接时是否启用在线证书状态协议	false	
ssl.clientAuth和ssl.quorum.clientAuth	指定用于验证客户端SSL连接的选项。可以设置为: "none":服务器不会请求客户端身份验证	"need"	

	"want":服务器将请求客户端身份验证 "need":服务器将要求客户端身份验证		
client.portUnification	指定客户端端口应接受SSL连接（使用与安全客户端端口相同的配置）	false	
authProvider	可以为 ADCC指定多个身份验证提供程序类。使用此参数来指定 SASL 身份验证提供程序，如： authProvider.1=org.apache.zookeeper.server.auth.SASLAuthenticationProvider	false	
kerberos.removeHostFromPrincipal	可以指示 ADCC 在身份验证期间从客户端主体名称中删除主机。（例如，adcc/myhost@EXAMPLE.COM 客户端主体在ADCC中进行身份验证时将变为 adcc@EXAMPLE.COM）	false	
kerberos.canonicalizeHostNames	指示ADCC规范化从server. x行提取的服务器主机名。这允许使用例如CNAME记录在配置文件中引用服务器，同时仍然启用集群中各个节点（Quorum 节点）之间的SASLKerberos身份验证	false	
multiAddress.enabled	可以为每个 ADCC 服务器实例指定多个地址（当集群中可以并行使用多个物理网络接口时，这可以提高可用性）	false	
multiAddress.reachabilityCheckTimeoutMs	可以为每个 ADCC 服务器实例指定多个地址（当集群中可以并行使用多个物理网络接口时，这可以提高可用性）。ADCC 将执行 ICMP ECHO 请求或尝试在目标主机的端口 7（Echo）上建立传输控制协议，以便找到可到达的地址。只有在配置中提供多个地址时才会发生这种情况。在此属性中，可以设置可达性检查的超时时间（以毫秒为单位）。不同地址的检查是并行进行的，因此这里设置的超时时间是检查所有地址可达性所需的最长时间。默认值为1000。此参数无效，除非您通过设置multiAddress.enabled=true来启用MultiAddress功能		

11.5 性能调优选项

对于 NIO 通信子系统和请求处理管道（提交处理器）这两个系统而言，若要达成读取吞吐量的最大值，就必须保证它们拥有充足数量的线程。只有当线程数量足够时，系统才能充分发挥其性能，高效处理大量的读取请求，从而实现读取

吞吐量达到峰值状态。比如在高并发的场景下，如果线程数量过少，NIO 通信子系统无法及时处理众多客户端的请求，或者请求处理管道不能同时处理多个读取请求，就会导致系统响应变慢，读取吞吐量无法达到最佳水平。下面选项根据需要调整。

配置项	说明	默认值
zookeeper.nio.numSelectorThreads	NIO选择器线程数。至少需要1个选择器线程。对于大量客户端连接，建议使用多个选择器。默认值为sqrt（cpu核数/ 2）	
zookeeper.nio.numWorkerThreads	NIO工作线程数。如果配置为0个工作线程，选择器线程将直接执行套接字I/O操作。默认值是cpu核心数的两倍	
zookeeper.commitProcessor.numWorkerThreads	提交处理器工作线程数。如果配置为0个工作线程，则主线程将直接处理请求。默认值为cpu核数	
zookeeper.commitProcessor.maxReadBatchSize	在切换到处理提交之前，从排队请求中要处理的最大读取次数。如果该值小于0（默认值），那么只要有本地写入和待处理的提交，就进行切换。较高的读取批次大小会延迟提交处理，导致提供陈旧数据。如果已知读取以固定大小的批次到达，则将此属性的值与该批次大小匹配可以平滑队列性能。由于读取是并行处理的，建议将此属性设置为与zookeeper.commitProcessor.numWorkerThread匹配（默认值是cpu核心数）或更低	
zookeeper.commitProcessor.maxCommitBatchSize	在处理读取操作之前要处理的最大提交数。在达到此数量之前，我们将尽可能多地处理远程/本地提交。较大的提交批次大小会在处理更多提交时延迟读取操作，而较小的提交批次大小则有利于读取操作。建议仅在集群处理高提交率的工作负载时设置此属性。如果已知写入操作以固定数量的批次到达，那么将此属性的值设置为与该批次大小匹配可以优化队列性能。一种通用方法是将此值设置为与集群大小相等，这样在处理每个批次时，当前服务器就有可能处理与其某个直接客户端相关的写入操作。默认值为“1”。不支持负数和零值。	
znode.container.checkIntervalMs	每次检查候选容器和生存时间（TTL）节点的时间间隔（单位为毫秒）	60000

znode.container.maxPerMinute	每分钟可删除的容器和生存时间（TTL）节点的最大数量。这可以防止在容器删除期间出现集群效应	10000
znode.container.maxNeverUsedIntervalMs	从未有过任何子项的容器保留的最长时间间隔（以毫秒为单位）。该时间间隔应足够长，以便客户端创建容器、完成任何必要的工作，然后创建子项。默认值为"0"，表示从未有过任何子项的容器永远不会被删除。	

11.6 调试可观察性配置

下表为调试的配置项。

配置项	说明	默认值
zookeeper.messageTracker.BufferSize	控制MessageTracker中存储的最大消息数。值应该是正整数。MessageTracker用于在服务器与领导者断开连接时记录服务器（追随者或观察者）和领导者之间的最后一组消息。然后，这些消息集将被转储到ADCC的日志文件中，这将有助于重建服务器在断开连接时的状态，并将有助于调试目的	10
zookeeper.messageTracker.Enabled	当设置为"true"时，将启用MessageTracker来跟踪和记录消息	false

11.7 AdminServer配置

下表为AdminServer的配置。

配置项	说明	默认值
admin.forceHttps	强制AdminServer使用SSL,覆盖admin.portUnification设置	false
admin.portUnification	启用管理端口以接受HTTP和HTTPS通信	false
admin.enableServer	是否启用AdminServer	true
admin.serverAddress	嵌入式Jetty服务器监听的地址	0.0.0.0
admin.serverPort	嵌入式Jetty服务器监听的端口	8080

admin.idleTimeout	设置连接在发送或接收数据前可以等待的最大空闲时间，单位为毫秒	30000ms
admin.commandURL	用于列出和发出与根URL相关命令的URL	/commands

11.8 指标提供方

下表为指标提供方配置。

配置项	说明	默认值
metricsProvider.className	设置为“org.apache.zookeeper.metrics.prometheus.PrometheusMetricsProvider”以启用 Prometheus.io 导出器	
metricsProvider.httpPort	Prometheus.io 导出器将启动一个 Jetty 服务器并绑定到该端口，默认端口为 7000。Prometheus 的URL是 http://hostname:httPort/metrics	
metricsProvider.exportJvmInfo	如果此属性设置为truePrometheus.io将导出有关JVM的有用指标	true

11.9 其他配置

以下选项可能很有用，但使用时要小心。每个选项的风险以及变量的作用都会进行说明。

配置项	说明	默认值	
forceSync	在完成更新处理之前，需要将更新同步到事务日志的存储介质。如果此选项设置为“no”，ADCC将不要求将更新同步到存储介质。		
jute.maxbuffer	1.此选项只能设置为Java系统属性。它没有zookeeper前缀。它指定可以存储在znode中的数据的最大大小。单位是：字节。默认值为0xfffff（1048575）字节，或接近1M 2.如果更改此选项，则必须在所有服务器和客户端上设置系统属性，否则会出现问题 3.当客户端的jute.maxbuffer大于服务器端时，客户端想要写入的数据超过了服务器端的jute.maxbuffer，服务器端将出现java.io.IOException: Len error 4.当客户端的jute.maxbuffer小于服务器端时，客户端想要读取的数据超过了客户端的jute.ma		

	xbuffer，客户端将收到java.io.IOException: Unreasonable length 或 Packet len is out of range! 5.这实际上是一种合理性检查.ADCC 旨在存储大小为千字节级别的数据。在生产环境中，出于以下原因，不建议将此属性增加到超过默认值。 1)大型znode会导致不必要的延迟峰值，从而降低吞吐量 大型znode会导致领导者与跟随者之间的同步时间不可预测且不收敛（有时会超时），从而致使仲裁不稳定		
jute.maxbuffer.extrasize	在处理客户端请求时，ADCC服务器会在持久化为事务之前向请求中添加一些额外信息。此前，这些额外信息的大小固定为1024字节。对于许多场景，特别是当jute.maxbuffer值超过1MB并且请求类型为多选时，这个固定大小是不够的。为了应对所有场景，额外信息的大小已从1024字节增加到与jute.maxbuffer大小相同，并且通过jute.maxbuffer.extrasize进行配置。通常不需要配置此属性，默认值是最优选择		
skipACL	跳过ACL检测。这会有助于提高吞吐量，但会让所有用户都能完全访问数据树。		
quorumListenOnAllIPs	当设置为true时，ADCC服务器将监听来自其对等节点的所有可用IP地址的连接，而不仅仅是配置文件的服务器列表中配置的地址。它会影响处理ZAB协议和快速领导者选举协议的连接。	false	
multiAddress.reachabilityCheckEnabled	可以为每个ADCC服务器实例指定多个地址（这可以在集群中并行使用多个物理网络接口时提高可用性）。ADCC将执行ICMP ECHO请求或尝试在目标主机的端口7（Echo）上建立TCP连接，以查找可访问的地址。这仅在配置中提供了多个地址时才会发生。如果在单台机器上启动一个大型（例如11个及以上）集群成员的集群进行测试时，遇到某些ICMP速率限制（例如在MacOS上），可达性检查可能会失败。默认值为true。设置为“false”，可以禁用可达性检查。请注意，禁用可达性检查将导致	true	

	集群在网络问题期间无法正确重新配置自身，因此仅建议在测试期间禁用。此参数无效，除非您通过设置multiAddress.enabled=true来启用MultiAddress功能	

12 常见问题

12.1 高并发性能调优

在进行高并发时，可以参考以下配置进行调优。

1、修改zoo.cfg文件，内容如下：

```
#取消单个IP的最大连接数限制（默认60），设置0表示无限制
maxClientCnxns=0
#保存3个快照，3个日志文件
autopurge.snapRetainCount=3
#每隔1个小时执行一次清理
autopurge.purgeInterval=1
```

2、修改运行内存，可以在启动脚本如 adccServer.sh中添加参数：

```
JVMFLAGS="-Xms4096m -Xmx4096m"
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com