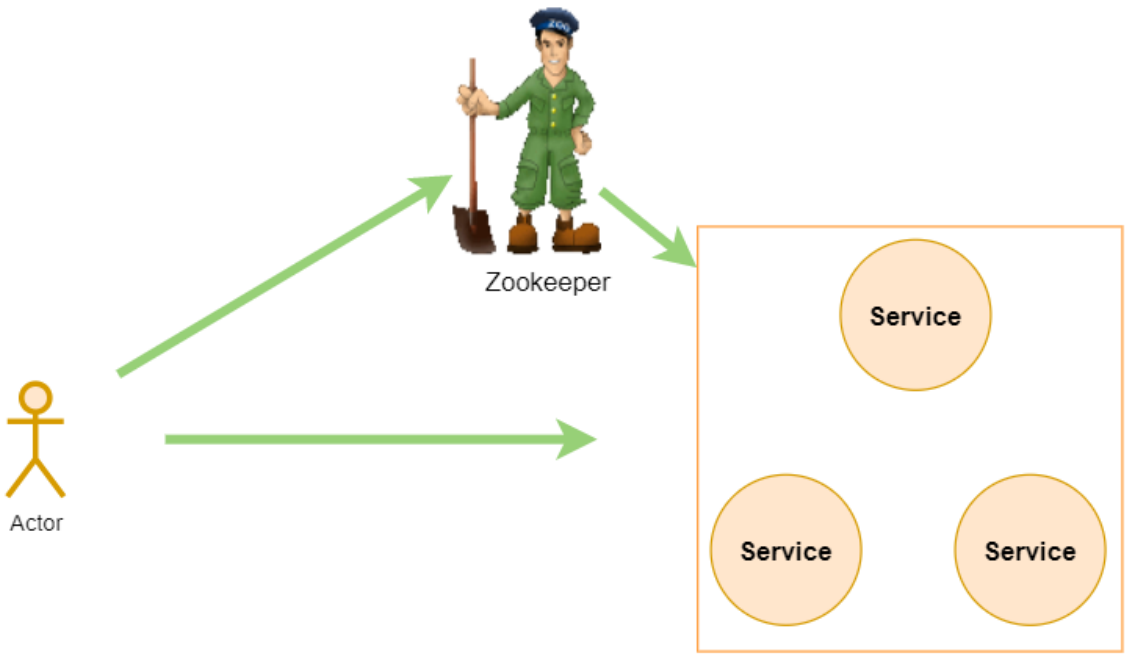
1. 什么是Zookeeper
2. Zookeeper核心概念
3. Zookeeper实操
4. Zookeeper ACLs权限控制
5. ZooKeeper 内存数据和持久化

在了解Zookeeper之前，需要对分布式相关知识有一定了解，什么是分布式系统呢？通常情况下，单个物理节点很容易达到性能，计算或者容量的瓶颈，所以这个时候就需要多个物理节点来共同完成某项任务，一个分布式系统的本质是分布在不同网络或计算机上的程序组件，彼此通过信息传递来协同工作的系统，而Zookeeper正是一个分布式应用协调框架，在分布式系统架构中有广泛的应用场景。

1. 什么是Zookeeper?

官方文档上这么解释zookeeper，它是一个分布式协调框架，是Apache Hadoop 的一个子项目，它主要是用来解决分布式应用中经常遇到的一些数据管理问题，如：统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。

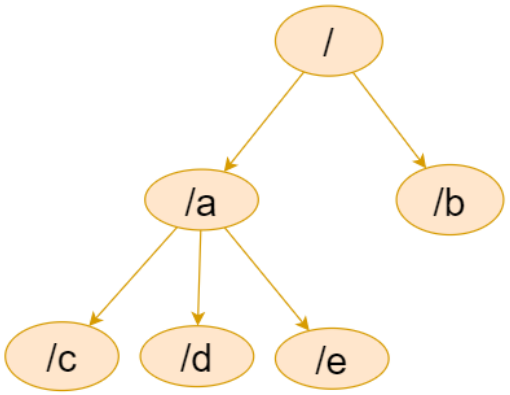


2. Zookeeper 核心概念

上面的解释有点抽象，同学们暂时可以理解为 Zookeeper 是一个用于存储少量数据的基于内存的数据库，主要有如下两个核心的概念：文件系统数据结构+监听通知机制。

2.1、 文件系统数据结构

Zookeeper维护一个类似文件系统的数据结构：



每个子目录项都被称作为 **znode(目录节点)**，和文件系统类似，我们能够自由的增加、删除znode，在一个znode下增加、删除子znode。
有四种类型的znode：

1、PERSISTENT-持久化目录节点
客户端与zookeeper断开连接后，该节点依旧存在，只要不手动删除该节点，他将永远存在

2、 PERSISTENT_SEQUENTIAL-持久化顺序编号目录节点
客户端与zookeeper断开连接后，该节点依旧存在，只是Zookeeper给该节点名称进行顺序编号

3、EPHEMERAL-临时目录节点
客户端与zookeeper断开连接后，该节点被删除

4、EPHEMERAL_SEQUENTIAL-临时顺序编号目录节点
客户端与zookeeper断开连接后，该节点被删除，只是Zookeeper给该节点名称进行顺序编号

5. Container 节点（3.5.3 版本新增，如果Container节点下面没有子节点，则Container节点在未来会被Zookeeper自动清除,定时任务默认60s 检查一次）

6. TTL 节点(默认禁用，只能通过系统配置 *zookeeper.extendedTypesEnabled=true* 开启，不稳定)

2.2、监听通知机制

客户端注册监听它关心的任意节点，或者目录节点及递归子目录节点

1. 如果注册的是对某个节点的监听，则当这个节点被删除，或者被修改时，对应的客户端将被通知
2. 如果注册的是对某个目录的监听，则当这个目录有子节点被创建，或者有子节点被删除，对应的客户端将被通知
3. 如果注册的是对某个目录的递归子节点进行监听，则当这个目录下面的任意子节点有目录结构的变化（有子节点被创建，或被删除）或者根节点有数据变化时，对应的客户端将被通知。

注意：所有的通知都是一次性的，及无论是对节点还是对目录进行的监听，一旦触发，对应的监听即被移除。递归子节点，监听是对所有子节点的，所以，每个子节点下面的事件同样只会被触发一次。

2.3、Zookeeper 经典的应用场景

1. 分布式配置中心

2. 分布式注册中心
3. 分布式锁
4. 分布式队列
5. 集群选举
6. 分布式屏障
7. 发布/订阅

3. Zookeeper 实战

3.1. zookeeper安装

Step1：配置JAVA环境，检验环境：

```
java -version
```

Step2: 下载解压 zookeeper

```
wget https://mirror.bit.edu.cn/apache/zookeeper/zookeeper-3.5.8/apache-zookeeper-3.5.8-bin.tar.gz
tar -zxvf apache-zookeeper-3.5.8-bin.tar.gz
cd apache-zookeeper-3.5.8-bin
```

Step3: 重命名配置文件 zoo_sample.cfg

```
 cp zoo_sample.cfg zoo.cfg
```

Step4: 启动zookeeper

```
# 可以通过 bin/zkServer.sh 来查看都支持哪些参数
bin/zkServer.sh start conf/zoo.cfg
```

Step5: 检测是否启动成功

```
echo stat | nc 192.168.109.200 // 前提是配置文件中中讲 stat 四字命令设置了了白名单
如:
4lw.commands.whitelist=stat
```

Step6: 连接服务器

```
bin/zkCli.sh -server ip:port
```

3.2. 使用命令行操作zookeeper

输入命令 help 查看zookeeper所支持的所有命令：

```
[zk: localhost:2181(CONNECTED) 80] help
ZooKeeper -server host:port cmd args
 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
 delquota [-n|-b] path
 get [-s] [-w] path
 getAcl [-s] path
 history
 listquota path
 ls [-s] [-w] [-R] path
 ls2 path [watch]
 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]
 rmr path
 set [-s] [-v version] path data
 setAcl [-s] [-v version] [-R] path acl
 setquota -n|-b val path
 stat [-w] path
 sync path
```

1. 创建zookeeper 节点命令

```
create [-s] [-e] [-c] [-t ttl] path [data] [acl]

```

中括号为可选项，没有则默认创建持久化节点

-s: 顺序节点

-e: 临时节点

-c: 容器节点

-t: 可以给节点添加过期时间，默认禁用，需要通过系统参数启用

（-D*zookeeper.extendedTypesEnabled=true*，*znode.container.checkIntervalMs*：(Java system property only) **New in 3.5.1:** The time interval in milliseconds for each check of candidate container and ttl nodes. Default is "60000".）

创建节点：

```
create /test-node some-data
```

如上，没有加任何可选参数，创建的就是持久化节点

```
[zk: localhost:2181(CONNECTED) 112] create  /test-node  some-data
Created /test-node
```

查看节点：

```
get /test-node
```

```
[zk: localhost:2181(CONNECTED) 113] get /test-node
some-data
```

修改节点数据：

```
set /test-node some-data-changed
```

```
[zk: localhost:2181(CONNECTED) 114] set /test-node some-data-changed
[zk: localhost:2181(CONNECTED) 115] get /test-node
some-data-changed
```

查看节点状态信息：

```
stat /test-node
```

```
[zk: localhost:2181(CONNECTED) 116] stat /test-node
cZxid = 0xa5
ctime = Tue Sep 29 13:32:52 CST 2020
mZxid = 0xa6
mtime = Tue Sep 29 13:33:23 CST 2020
pZxid = 0xa5
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 17
numChildren = 0
```

Stat

- cZxid：创建znode的事务ID（Zxid的值）。
- mZxid：最后修改znode的事务ID。
- pZxid：最后添加或删除子节点的事务ID（子节点列表发生变化才会发生改变）。
- ctime：znode创建时间。
- mtime：znode最近修改时间。
- dataVersion：znode的当前数据版本。
- cversion：znode的子节点结果集版本（一个节点的子节点增加、删除都会影响这个版本）。
- aclVersion：表示对此znode的acl版本。
- ephemeralOwner：znode是临时znode时，表示znode所有者的 session ID。 如果znode不是临时znode，则该字段设置为零。
- dataLength：znode数据字段的长度。

- numChildren：znode的子znode的数量。

查看节点状态信息同时查看数据

```
[zk: localhost:2181(CONNECTED) 117] get -s /test-node
some-data-changed
cZxid = 0xa5
ctime = Tue Sep 29 13:32:52 CST 2020
mZxid = 0xa6
mtime = Tue Sep 29 13:33:23 CST 2020
pZxid = 0xa5
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 17
numChildren = 0
```

根据状态数据中的版本号有并发修改数据实现乐观锁的功能

比如： 客户端首先获取版本信息， get -s /node-test

```
[zk: localhost:2181(CONNECTED) 121] get -s /test-node
some-data-changed
cZxid = 0xa5
ctime = Tue Sep 29 13:32:52 CST 2020
mZxid = 0xa6
mtime = Tue Sep 29 13:33:23 CST 2020
pZxid = 0xa7
cversion = 1
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 17
numChildren = 1
```

/test-node 当前的数据版本是 1， 这时客户端 用 set 命令修改数据的时候可以把版本号带上

```
[zk: localhost:2181(CONNECTED) 123] set -v 1 /test-node  changed
```

如果在执行上面 set命令前， 有人修改了数据，zookeeper 会递增版本号， 这个时候，如果再用以前的版本号去修改，将会导致修改失败，报如下错误

```
[zk: localhost:2181(CONNECTED) 123] set -v 1 /test-node  changed
version No is not valid : /test-node
```

创建子节点， 这里要注意，zookeeper是以节点组织数据的，没有相对路径这么一说，所以，所有的节点一定是以 / 开头。

```
create /test-node/test-sub-node
```

```
[zk: localhost:2181(CONNECTED) 119] create /test-node/test-sub-node
Created /test-node/test-sub-node
```

查看子节点信息，比如根节点下面的所有子节点， 加一个大写 R 可以查看递归子节点列表

```
ls /
```

```
[zk: localhost:2181(CONNECTED) 118] ls /
[test-node, zookeeper]
```

查看 /test-node 下面所有的子节点

```
[zk: localhost:2181(CONNECTED) 120] ls /test-node
[test-sub-node]
```

创建临时节点

```
create -e /ephemeral data
```

create 后跟一个 -e 创建临时节点， 临时节点不能创建子节点

```
[zk: localhost:2181(CONNECTED) 124] create -e /ephemeral data
Created /ephemeral
[zk: localhost:2181(CONNECTED) 125] get /ephemeral
data
[zk: localhost:2181(CONNECTED) 126] set /ephemeral  ddd
[zk: localhost:2181(CONNECTED) 127] get /ephemeral
ddd
[zk: localhost:2181(CONNECTED) 128] create /ephemeral/sub-node
Ephemerals cannot have children: /ephemeral/sub-node
```

创建序号节点，加参数 -s

```
create /seq-parent data // 创建父目录，单纯为了分类，非必须
create -s /seq-parent/ data // 创建顺序节点。顺序节点将再seq-parent 目录下面，顺序递增
```

为了容纳子节点，先创建个父目录 /seq-parent

```
[zk: localhost:2181(CONNECTED) 141] create /seq-parent
Created /seq-parent
[zk: localhost:2181(CONNECTED) 142] create -s /seq-parent/   seq-data
Created /seq-parent/0000000000
```

也可以再序号节点前面带一个前缀

```
Created /seq-parent/0000000000
[zk: localhost:2181(CONNECTED) 143] create -s /seq-parent/x   seq-data
Created /seq-parent/x0000000001
[zk: localhost:2181(CONNECTED) 144] create -s /seq-parent/y   seq-data
Created /seq-parent/y0000000002
[zk: localhost:2181(CONNECTED) 145] create -s /seq-parent/z   seq-data
Created /seq-parent/z0000000003
```

创建临时顺序节点,其它增删查改和其他节点无异，不再贴图

```
create -s -e /ephemeral-node/前缀-
```

创建容器节点

```
create -c /container
```

容器节点主要用来容纳字节点，如果没有给其创建子节点，容器节点表现和持久化节点一样，如果给容器节点创建了子节点，后续又把子节点清空，容器节点也会被zookeeper删除。

2. 事件监听机制：

针对节点的监听：一定事件触发，对应的注册立刻被移除，所以事件监听是一次性的

```
get -w /path // 注册监听的同时获取数据
stat -w /path // 对节点进行监听，且获取元数据信息
```

```
[zk: localhost:2181(CONNECTED) 6] get -w /test
null
[zk: localhost:2181(CONNECTED) 7] set /test xxx

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/test
[zk: localhost:2181(CONNECTED) 8] set /test xxx
[zk: localhost:2181(CONNECTED) 9]
[zk: localhost:2181(CONNECTED) 9]
[zk: localhost:2181(CONNECTED) 9] set /test xxx
[zk: localhost:2181(CONNECTED) 10] get -w /test
xxx
[zk: localhost:2181(CONNECTED) 11] set /test  xxx

WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/test
[zk: localhost:2181(CONNECTED) 12]
[zk: localhost:2181(CONNECTED) 12]
[zk: localhost:2181(CONNECTED) 12]
[zk: localhost:2181(CONNECTED) 12] set /test  xxx
```

针对目录的监听，如下图，目录的变化，会触发事件，且一旦触发，对应的监听也会被移除，后续对节点的创建没有触发监听事件

```
ls -w /path
```

```
[zk: localhost:2181(CONNECTED) 21] ls /test
[sub1]
[zk: localhost:2181(CONNECTED) 22] ls -w /test
[sub1]
[zk: localhost:2181(CONNECTED) 23] delete /test/sub1

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/test
[zk: localhost:2181(CONNECTED) 24] create /test/sub0
Created /test/sub0
[zk: localhost:2181(CONNECTED) 25]
```

针对递归子目录的监听

```
ls -R -w /path :  -R 区分大小写，一定用大写
```

如下对/test 节点进行递归监听，但是每个目录下的目录监听也是一次性的，如第一次在/test 目录下创建节点时，触发监听事件，第二次则没有，同样，因为时递归的目录监听，所以在/test/sub0下进行节点创建时，触发事件，但是再次创建/test/sub0/subsub1节点时，没有触发事件。

```
[zk: localhost:2181(CONNECTED) 25] ls -R -w  /test
/test
/test/sub0
[zk: localhost:2181(CONNECTED) 26] create /test/sub1

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/test
Created /test/sub1
[zk: localhost:2181(CONNECTED) 27] create /test/sub2
Created /test/sub2
[zk: localhost:2181(CONNECTED) 28] create /test/sub0/subsub0

WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/test/sub0
Created /test/sub0/subsub0
[zk: localhost:2181(CONNECTED) 29] create /test/sub0/subsub1
Created /test/sub0/subsub1
[zk: localhost:2181(CONNECTED) 30]
```

Zookeeper事件类型：

None：连接建立事件
NodeCreated： 节点创建
NodeDeleted： 节点删除
NodeDataChanged：节点数据变化
NodeChildrenChanged：子节点列表变化
DataWatchRemoved：节点监听被移除
ChildWatchRemoved：子节点监听被移除

4. Zookeeper 的 ACL 权限控制(Access Control　List)

Zookeeper 的ACL 权限控制,可以控制节点的读写操作,保证数据的安全性，Zookeeper ACL 权限设置分为 3 部分组成，分别是：**权限模式**（Scheme）、**授权对象**（ID）、**权限信息**（Permission）。最终组成一条例如“scheme:id:permission”格式的 ACL 请求信息。下面我们具体看一下这 3 部分代表什么意思：

Scheme（权限模式）：用来设置 ZooKeeper 服务器进行权限验证的方式。ZooKeeper 的权限验证方式大体分为两种类型：

一种是**范围验证**。所谓的范围验证就是说 ZooKeeper 可以针对一个 IP 或者一段 IP 地址授予某种权限。比如我们可以让一个 IP 地址为“ip：192.168.0.110”的机器对服务器上的某个数据节点具有写入的权限。或者也可以通过“ip:192.168.0.1/24”给一段 IP 地址的机器赋权。

另一种权限模式就是**口令验证**，也可以理解为用户名密码的方式。在 ZooKeeper 中这种验证方式是 Digest 认证，而 Digest 这种认证方式首先在客户端传送“username:password”这种形式的权限表示符后，ZooKeeper 服务端会对密码 部分使用 SHA-1 和 BASE64 算法进行加密，以保证安全性。

还有一种Super权限模式，Super可以认为是一种特殊的 Digest 认证。具有 Super 权限的客户端可以对 ZooKeeper 上的任意数据节点进行任意操作。

授权对象（ID）

授权对象就是说我们要把权限赋予谁，而对应于 4 种不同的权限模式来说，如果我们选择采用 IP 方式，使用的授权对象可以是一个 IP 地址或 IP 地址段；而如果使用 Digest 或 Super 方式，则对应于一个用户名。如果是 World 模式，是授权系统中所有的用户。

权限信息（Permission）

权限就是指我们可以在数据节点上执行的操作种类，如下所示：在 ZooKeeper 中已经定义好的权限有 5 种：

数据节点（c: create）创建权限，授予权限的对象可以在数据节点下创建子节点；
数据节点（w: wirte）更新权限，授予权限的对象可以更新该数据节点；
数据节点（r: read）读取权限，授予权限的对象可以读取该节点的内容以及子节点的列表信息；
数据节点（d: delete）删除权限，授予权限的对象可以删除该数据节点的子节点；
数据节点（a: admin）管理者权限，授予权限的对象可以对该数据节点体进行 ACL 权限设置。

命令:

getAcl：获取某个节点的acl权限信息

setAcl：设置某个节点的acl权限信息

addauth: 输入认证授权信息，相当于注册用户信息，注册时输入明文密码，zk将以密文的形式存储

可以通过系统参数zookeeper.skipACL=yes进行配置，默认是no,可以配置为true, 则配置过的ACL将不再进行权限检测

生成授权ID的两种方式:

a.代码生成ID:

```
@Test
public void generateSuperDigest() throws NoSuchAlgorithmException {
 String sId = DigestAuthenticationProvider.generateDigest("gj:test");
 System.out.println(sId);// gj:X/NSthOB0fD/OT6iilJ55WJVado=
}
```

b.在xshell 中生成

```
echo -n <user>:<password> | openssl dgst -binary -sha1 | openssl base64
```

设置ACL有两种方式

节点创建的同时设置ACL

create [-s] [-e] [-c] path [data] [**acl**]

```
create /zk-node datatest digest:gj:X/NSthOB0fD/OT6iilJ55WJVado=:cdrwa
```

或者用setAcl 设置

```
setAcl /zk-node digest:gj:X/NSthOB0fD/OT6iilJ55WJVado=:cdrwa
```

添加授权信息后，不能直接访问，直接访问将报如下异常

```
get /zk-node
异常信息:
org.apache.zookeeper.KeeperException$NoAuthException: KeeperErrorCode = NoAuth for /zk-node
```

访问前需要添加授权信息

```
addauth digest gj:test
```

```
get /zk-node
datatest
```

另一种授权模式： auth 明文授权

使用之前需要先

addauth digest username:password 注册用户信息，后续可以直接用明文授权

如：

```
addauth digest u100:p100
create /node-1 node1data auth:u100:p100:cdwra
这是u100用户授权信息会被zk保存，可以认为当前的授权用户为u100
get /node-1
node1data
```

IP授权模式：

```
setAcl /node-ip ip:192.168.109.128:cdwra
create /node-ip data ip:192.168.109.128:cdwra
```

多个指定IP可以通过逗号分隔， 如 setAcl /node-ip ip:IP1:rw,ip:IP2:a

Super 超级管理员模式

这是一种特殊的Digest模式， 在Super模式下超级管理员用户可以对Zookeeper上的节点进行任何的操作。

需要在启动了上通过JVM 系统参数开启：

```
DigestAuthenticationProvider中定义
-Dzookeeper.DigestAuthenticationProvider.superDigest=super:
<base64encoded(SHA1(password))
```

5. ZooKeeper 内存数据和持久化

Zookeeper数据的组织形式为一个类似文件系统的数据结构，而这些数据都是存储在内存中的，所以我们可以认为，Zookeeper是一个基于内存的小型数据库

内存中的数据：

```
public class DataTree {
 private final ConcurrentHashMap<String, DataNode> nodes =
 new ConcurrentHashMap<String, DataNode>();


 private final WatchManager dataWatches = new WatchManager();
```

```
7   private final WatchManager childWatches = new WatchManager();
8
```

DataNode 是Zookeeper存储节点数据的最小单位

```
1 public class DataNode implements Record {
2  byte data[];
3  Long acl;
4  public StatPersisted stat;
5  private Set<String> children = null;
```

事务日志

针对每一次客户端的事务操作，Zookeeper都会将他们记录到事务日志中，当然，Zookeeper也会将数据变更应用到内存数据库中。我们可以在zookeeper的主配置文件zoo.cfg 中配置内存中的数据持久化目录，也就是事务日志的存储路径 dataLogDir. 如果没有配置dataLogDir（非必填），事务日志将存储到dataDir （必填项）目录，
zookeeper提供了格式化工具可以进行数据查看事务日志数据
org.apache.zookeeper.server.LogFormatter

```
1 java -classpath .:slf4j-api-1.7.25.jar:zookeeper-3.5.8.jar:zookeeper-jute-
3.5.8.jar org.apache.zookeeper.server.LogFormatter /usr/local/zookeeper/apache-zo
okeeper-3.5.8-bin/data/version-2/log.1
```

如下是我本地的日志文件格式化效果

```
ZooKeeper Transactional Log File with dbid 0 txnlog format version 2
11/13/20 5:28:28 PM CST session 0x1000ea2bb620000 cxid 0x0 zxid 0x1 createSession 30000

11/13/20 5:28:38 PM CST session 0x1000ea2bb620000 cxid 0x2 zxid 0x2 create '/test,,v{s{31,s{'world,'anyone}}},F,1

11/13/20 7:36:52 PM CST session 0x1000ea2bb620000 cxid 0x4 zxid 0x3 setData '/test,#787878,1
```

从左到右分别记录了操作时间，客户端会话ID，CXID,ZXID,操作类型，节点路径，节点数据（用#+ascii 码表示），节点版本。

Zookeeper进行事务日志文件操作的时候会频繁进行磁盘IO操作，事务日志的不断追加写操作会触发底层磁盘IO为文件开辟新的磁盘块，即磁盘Seek。因此，为了提升磁盘IO的效率，Zookeeper在创建事务日志文件的时候就进行文件空间的预分配- 即在创建文件的时候，就向操作系统申请一块大一点的磁盘块。这个预分配的磁盘大小可以通过系统参数zookeeper.preAllocSize 进行配置。

事务日志文件名为： log.<当时最大事务ID>，应为日志文件时顺序写入的，所以这个最大事务ID也将是整个事务日志文件中，最小的事务ID，日志满了即进行下一次事务日志文件的创建

数据快照

数据快照用于记录Zookeeper服务器上某一时刻的全量数据，并将其写入到指定的磁盘文件中。
可以通过配置snapCount配置每间隔事务请求个数，生成快照，数据存储在dataDir 指定的目录中，
可以通过如下方式进行查看快照数据（ 为了避免集群中所有机器在同一时间进行快照，实际的快照生成时机为事务数达到 [snapCount/2 + 随机数(随机数范围为1 ~ snapCount/2)] 个数时开始快照）

```
java -classpath .:slf4j-api-1.7.25.jar:zookeeper-3.5.8.jar:zookeeper-jute-3.5.8.jar org.apache.zookeeper.server.SnapshotFormatter /usr/local/zookeeper/apache-zookeeper-3.5.8-bin/data-dir/version-2/snapshot.0
```

```
ZNode Details (count=6):
----
/
  cZxid = 0x00000000000000
  ctime = Thu Jan 01 08:00:00 CST 1970
  mZxid = 0x00000000000000
  mtime = Thu Jan 01 08:00:00 CST 1970
  pZxid = 0x00000000000002
  cversion = 1
  dataVersion = 0
  aclVersion = 0
  ephemeralOwner = 0x00000000000000
  dataLength = 0
----
/zookeeper
  cZxid = 0x00000000000000
  ctime = Thu Jan 01 08:00:00 CST 1970
  mZxid = 0x00000000000000
  mtime = Thu Jan 01 08:00:00 CST 1970
  pZxid = 0x00000000000000
  cversion = 0
  dataVersion = 0
```

快照事务日志文件名为： snapshot.<当时最大事务ID>，日志满了即进行下一次事务日志文件的创建

有了事务日志，为啥还要快照数据。
快照数据主要时为了快速恢复， 事务日志文件是每次事务请求都会进行追加的操作，而快照是达到某种设定条件下的内存全量数据。所以通常快照数据是反应当时内存数据的状态。事务日志是更全面的数据，所以恢复数据的时候，可以先恢复快照数据，再通过增量恢复事务日志中的数据即可。

文档：VIP-01 Zookeeper特性与节点数据类型详...

链接：http://note.youdao.com/noteshare?id=16ab5da92ee89d93f948dc57762b7ae6&sub=D37AC1F910A04A2EAEC83881AB0F32A1