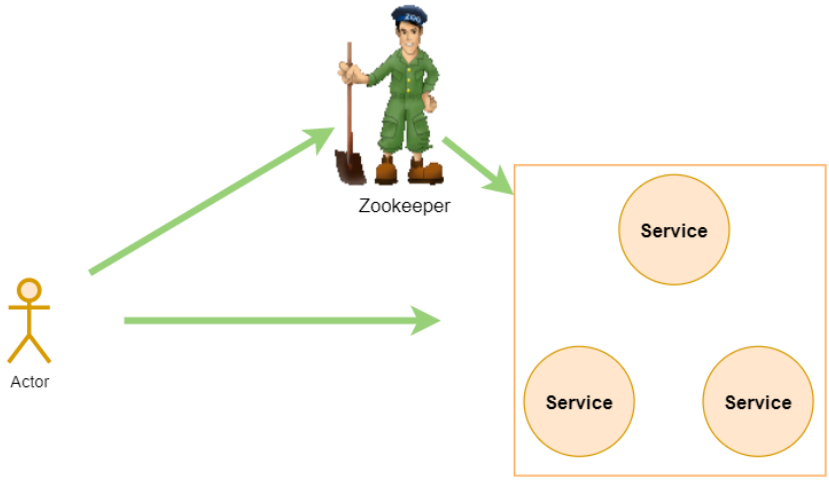
主讲老师： Fox

> 有道笔记地址：https://note.youdao.com/s/TxsbsdPn

1. Zookeeper介绍

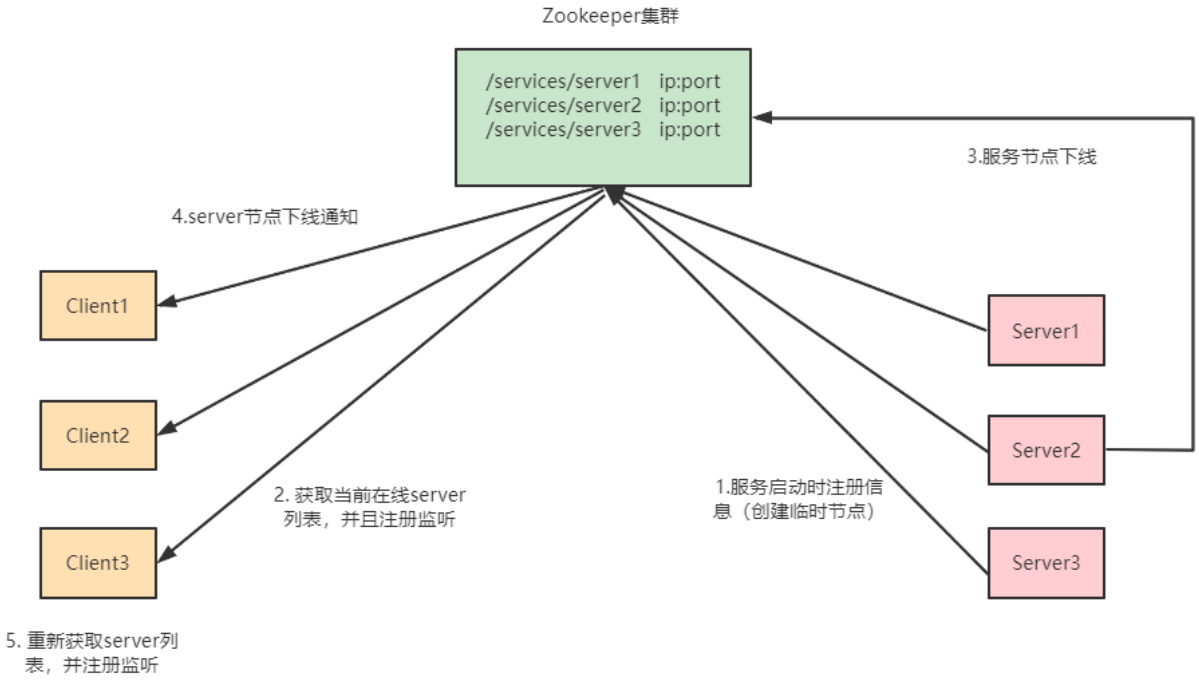
ZooKeeper 是一个开源的分布式协调框架，是Apache Hadoop 的一个子项目，主要用来解决分布式集群中应用系统的一致性问题。Zookeeper 的设计目标是将那些复杂且容易出错的分布式一致性服务封装起来，构成一个高效可靠的原语集，并以一系列简单易用的接口提供给用户使用。

官方：https://zookeeper.apache.org/



ZooKeeper本质上是一个分布式的小文件存储系统（Zookeeper=文件系统+监听机制）。提供基于类似于文件系统的目录树方式的数据存储，并且可以对树中的节点进行有效管理，从而用来维护和监控存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理、统一命名服务、分布式配置管理、分布式消息队列、分布式锁、分布式协调等功能。

Zookeeper从设计模式角度来理解：是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，Zookeeper就将负责通知已经在Zookeeper上注册的那些观察者做出相应的反应。



2. Zookeeper快速开始

2.1 Zookeeper安装

下载地址：https://zookeeper.apache.org/releases.html

运行环境：jdk8

1）修改配置文件

解压安装包后进入conf目录，复制zoo_sample.cfg，修改为zoo.cfg

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

修改 zoo.cfg 配置文件，将 dataDir=/tmp/zookeeper 修改为指定的data目录

zoo.cfg中参数含义：

2）启动zookeeper server

```
# 可以通过 bin/zkServer.sh  来查看都支持哪些参数
# 默认加载配置路径conf/zoo.cfg
```

```
bin/zkServer.sh start
bin/zkServer.sh start conf/my_zoo.cfg

# 查看zookeeper状态
bin/zkServer.sh status
```

3）启动zookeeper client连接Zookeeper server

```
bin/zkCli.sh
# 连接远程的zookeeper server
bin/zkCli.sh -server ip:port
```

2.2 客户端命令行操作

输入命令 help 查看zookeeper支持的所有命令：

常见cli命令

https://zookeeper.apache.org/doc/r3.8.0/zookeeperCLI.html

命令基本语法	功能描述
help	显示所有操作命令
ls [-s] [-w] [-R] path	使用 ls 命令来查看当前 znode 的子节点 [可监听] -w: 监听子节点变化 -s: 节点状态信息（时间戳、版本号、数据大小等） -R: 表示递归的获取
create [-s] [-e] [-c] [-t ttl] path [data] [acl]	创建节点 -s : 创建有序节点。 -e : 创建临时节点。 -c : 创建一个容器节点。 t ttl] : 创建一个TTL节点， -t 时间（单位毫秒）。 data：节点的数据，可选，如果不使用时，节点数据就为null。 acl：访问控制
get [-s] [-w] path	获取节点数据信息

	-s: 节点状态信息（时间戳、版本号、数据大小等） -w: 监听节点变化
set [-s] [-v version] path data	设置节点数据 -s:表示节点为顺序节点 -v: 指定版本号
getAcl [-s] path	获取节点的访问控制信息 -s: 节点状态信息（时间戳、版本号、数据大小等）
setAcl [-s] [-v version] [-R] path acl	设置节点的访问控制列表 -s:节点状态信息（时间戳、版本号、数据大小等） -v:指定版本号 -R:递归的设置
stat [-w] path	查看节点状态信息
delete [-v version] path	删除某一节点，只能删除无子节点的节点。 -v： 表示节点版本号
deleteall path	递归的删除某一节点及其子节点
setquota -n|-b val path	对节点增加限制 n:表示子节点的最大个数 b:数据值的最大长度，-1表示无限制

2.3 GUI工具

- Zookeeper图形化工具：ZooInspector
- Zookeeper图形化工具：开源的prettyZoo
- Zookeeper图形化工具：收费的ZooKeeperAssistant

3. ZooKeeper数据结构

ZooKeeper 数据模型的结构与 Unix 文件系统很类似，整体上可以看作是一棵树，每个节点称做一个 ZNode。

ZooKeeper的数据模型是层次模型，层次模型常见于文件系统。层次模型和key-value模型是两种主流的数据模型。ZooKeeper使用文件系统模型主要基于以下两点考虑:

1. 文件系统的树形结构便于表达数据之间的层次关系
2. 文件系统的树形结构便于为不同的应用分配独立的命名空间(namespace)

ZooKeeper的层次模型称作Data Tree，Data Tree的每个节点叫作Znode。不同于文件系统，每个节点都可以保存数据，每一个 ZNode 默认能够存储 1MB 的数据，每个 ZNode 都可以通过其路径唯一标识，每个节点都有一个版本(version)，版本从0开始计数。

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


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

}

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

3.1 节点分类

zookeeper存在几种不同的节点类型，他们具有不同的生命周期：

类型	生命周期	创建示例
持久节点 （persistent node)	一直存在，一直存储在 ZooKeeper 服务器上，即使创建该节点的客户端与服务端的会话关闭了，该节点依然不会被删除	create /locks
临时节点 (ephemeral node)	当创建该临时节点的客户端会话因超时或发生异常而关闭时，该节点也相应在 ZooKeeper 服务器上被删除。	create -e /locks/DBLock
有序节点 (sequential node)	并不算是一种单独种类的节点，而是在之前提到的持久节点和临时节点特性的基础上，增加了一	create -e -s /jobs/job (有序临时节点)

	个节点有序的性质。在我们创建有序节点的时候会自动使用一个单调递增的数字作为后缀	
容器节点 (container node)	当一个容器节点的最后一个子节点被删除后，容器节点也会被删除	create -c /work
TTL节点 (ttl node)	当一个TTL节点在 TTL 内没有被修改并且没有子节点，会被删除。注意：默认此功能不开启，需要修改配置文件 extendedTypesEnabled=true	create -t 3000 /ttl_node

一个znode可以使持久性的，也可以是临时性的:

1. 持久节点(PERSISTENT): 这样的znode在创建之后即使发生ZooKeeper集群宕机或者client宕机也不会丢失。

2. 临时节点(EPHEMERAL): client宕机或者client在指定的timeout时间内没有给ZooKeeper集群发消息，这样的znode就会消失。

如果上面两种znode具备顺序性，又有以下两种znode :

3. 持久顺序节点(PERSISTENT_SEQUENTIAL): znode除了具备持久性znode的特点之外，znode的名字具备顺序性。

4. 临时顺序节点(EPHEMERAL_SEQUENTIAL): znode除了具备临时性znode的特点之外，zorde的名字具备顺序性。

zookeeper主要用到的是以上4种节点。

5. Container节点 (3.5.3版本新增)：Container容器节点，当容器中没有任何子节点，该容器节点会被zk定期删除（定时任务默认60s 检查一次)。 和持久节点的区别是 ZK 服务端启动后，会有一个单独的线程去扫描，所有的容器节点，当发现容器节点的子节点数量为 0 时，会自动删除该节点。可以用于 leader 或者锁的场景中。

6. TTL节点: 带过期时间节点，默认禁用，需要在zoo.cfg中添加 extendedTypesEnabled=true 开启。 注意：TTL不能用于临时节点

```
#创建持久节点
create /servers  xxx
#创建临时节点
create -e /servers/host  xxx
#创建临时有序节点
create -e -s /servers/host  xxx
```

```
#创建容器节点
create -c /container xxx
# 创建ttl节点
create -t 10 /ttl

```

示例：实现分布式锁

分布式锁要求如果锁的持有者宕了，锁可以被释放。ZooKeeper 的 ephemeral 节点恰好具备这样的特性。

终端1:

```
zkCli.sh
create -e /lock
quit
```

终端2:

```
zkCli.sh
create -e /lock
stat -w /lock
create -e /lock
```

节点状态信息

类似于树状结构，节点下面是可以存储一些信息和属性的。可以通过stat命令来进行查看。

- cZxid ： Znode创建的事务id。
- ctime：节点创建时的时间戳。
- mZxid ： Znode被修改的事务id，即每次对znode的修改都会更新mZxid。

对于zk来说，每次的变化都会产生一个唯一的事务id，zxid（ZooKeeper Transaction Id），通过zxid，可以确定更新操作的先后顺序。例如，如果zxid1小于zxid2，说明zxid1操作先于zxid2发生，zxid对于整个zk都是唯一的，即使操作的是不同的znode。

- pZxid: 表示该节点的子节点列表最后一次修改的事务ID，添加子节点或删除子节点就会影响子节点列表，但是修改子节点的数据内容则不影响该ID（注意: 只有子节点列表变更了才会变更pzxid，子节点内容变更不会影响pzxid）
- mtime：节点最新一次更新发生时的时间戳.
- cversion ： 子节点的版本号。当znode的子节点有变化时，cversion 的值就会增加1。
- dataVersion：数据版本号，每次对节点进行set操作，dataVersion的值都会增加1（即使设置的是相同的数据），可有效避免了数据更新时出现的先后顺序问题。
- ephemeralOwner:如果该节点为临时节点, ephemeralOwner值表示与该节点绑定的session id。如果不是, ephemeralOwner值为0(持久节点)。

在client和server通信之前,首先需要建立连接,该连接称为session。连接建立后,如果发生连接超时、授权失败,或者显式关闭连接,连接便处于closed状态, 此时session结束。

- dataLength： 数据的长度
- numChildren ： 子节点的数量（只统计直接子节点的数量）

示例：zookeeper乐观锁删除

3.2 监听机制详解

watch机制，顾名思义是一个监听机制。Zookeeper中的watch机制，必须客户端先去服务端注册监听，这样事件发送才会触发监听，通知给客户端。

监听的对象是事件，支持的事件类型如下：

- 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

```
#监听节点数据的变化
get -w path
stat -w path
#监听子节点增减的变化
ls -w path

```

特性	说明
一次性触发	watch是一次性的，一旦被触发就会移除，再次使用时需要重新注册
客户端顺序回调	watch回调是顺序串行执行的，只有回调后客户端才能看到最新的数据状态。一个watcher回调逻辑不应该太多，以免影响别的watch执行
轻量级	WatchEvent是最小的通信单位，结构上只包含通知状态、事件类型和节点路径，并不会告诉数据节点变化前后的具体内容
时效性	watcher只有在当前session彻底失效时才会无效，若在session有效期内快速重连成功，则watcher依然存在，仍可接收到通知；

永久性Watch

在被触发之后，仍然保留，可以继续监听ZNode上的变更，是Zookeeper 3.6.0版本新增的功能

```
addWatch [-m mode] path
```

addWatch的作用是针对指定节点添加事件监听，支持两种模式

- PERSISTENT，持久化订阅，针对当前节点的修改和删除事件，以及当前节点的子节点的删除和新增事件。
- PERSISTENT_RECURSIVE，持久化递归订阅(默认)，在PERSISTENT的基础上，增加了子节点修改的事件触发，以及子节点的子节点的数据变化都会触发相关事件（满足递归订阅特性）

示例: 协同服务

设计一个master-worker的组成员管理系统，要求系统中只能有一个master，master能实时获取系统中worker的情况。

保证组里面只有一个master的设计思路

```
#master1
create -e /master "m1:2223"

#master2
create -e /master "m2:2223"    # /master已经存在，创建失败
Node already exists: /master
```

```
#监听/master节点
stat -w /master
#当master2收到/master节点删除通知后可以再次发起创建节点操作
create -e /master "m2:2223"
```

master-slave选举也可以用这种方式

master监控worker状态的设计思路

```
#master服务
create /workers
#让master服务监控/workers下的子节点
ls -w /workers

#worker1
create -e /workers/w1 "w1:2224" #创建子节点，master服务会收到子节点变化通知

#master服务
ls -w /workers
#worker2
create -e /workers/w2 "w2:2224"  #创建子节点，master服务会收到子节点变化通知

#master服务
ls -w /workers
#worker2
quit  #worker2退出，master服务会收到子节点变化通知
```

示例：条件更新

设想用znode /c实现一个counter，使用set命令来实现自增1操作。条件更新场景：

1. 客户端1把/c更新到版本1，实现/c的自增1。
2. 客户端2把/c更新到版本2，实现/c的自增1。

3. 客户端1不知道/c已经被客户端2 更新过了，还用过时的版本1是去更新/c，更新失败。如果客户端1使用的是无条件更新，/c就会更新为2，没有实现自增1。
使用条件更新可以避免出现客户端基于过期的数据进行数据更新的操作。

3.3 节点特性总结

1. 同一级节点 key 名称是唯一的

已存在/lock节点，再次创建会提示已经存在

2.创建节点时，必须要带上全路径

3.session 关闭，临时节点清除

4.自动创建顺序节点

5.watch 机制，监听节点变化

事件监听机制类似于观察者模式，watch 流程是客户端向服务端某个节点路径上注册一个 watcher，同时客户端也会存储特定的 watcher，当节点数据或子节点发生变化时，服务端通知客户端，客户端进行回调处理。特别注意：监听事件被单次触发后，事件就失效了。

6.delete 命令只能一层一层删除。提示：新版本可以通过 deleteall 命令递归删除。

3.4 应用场景详解

ZooKeeper适用于存储和协同相关的关键数据，不适合用于大数据量存储。

有了上述众多节点特性，使得 zookeeper 能开发不出不同的经典应用场景，比如：

- 注册中心
- 数据发布/订阅（常用于实现配置中心）
- 负载均衡
- 命名服务
- 分布式协调/通知
- 集群管理
- Master选举
- 分布式锁

- 分布式队列

统一命名服务

在分布式环境下，经常需要对应用/服务进行统一命名，便于识别。
例如：IP不容易记住，而域名容易记住。

利用 ZooKeeper 顺序节点的特性，制作分布式的序列号生成器，或者叫 id 生成器。（分布式环境下使用作为数据库 id，另外一种是 UUID（缺点：没有规律）），ZooKeeper 可以生成有顺序的容易理解的同时支持分布式环境的编号。

```
/
└── /order
    ├── /order-date1-000000000000001
    ├── /order-date2-000000000000002
    ├── /order-date3-000000000000003
    ├── /order-date4-000000000000004
    └── /order-date5-000000000000005
```

数据发布/订阅

数据发布/订阅的一个常见的场景是配置中心，发布者把数据发布到 ZooKeeper 的一个或一系列的节点上，供订阅者进行数据订阅，达到动态获取数据的目的。
配置信息一般有几个特点:

1. 数据量小的KV
2. 数据内容在运行时会发生动态变化
3. 集群机器共享，配置一致

ZooKeeper 采用的是推拉结合的方式。

1. 推: 服务端会推给注册了监控节点的客户端 Watcher 事件通知
2. 拉: 客户端获得通知后，然后主动到服务端拉取最新的数据

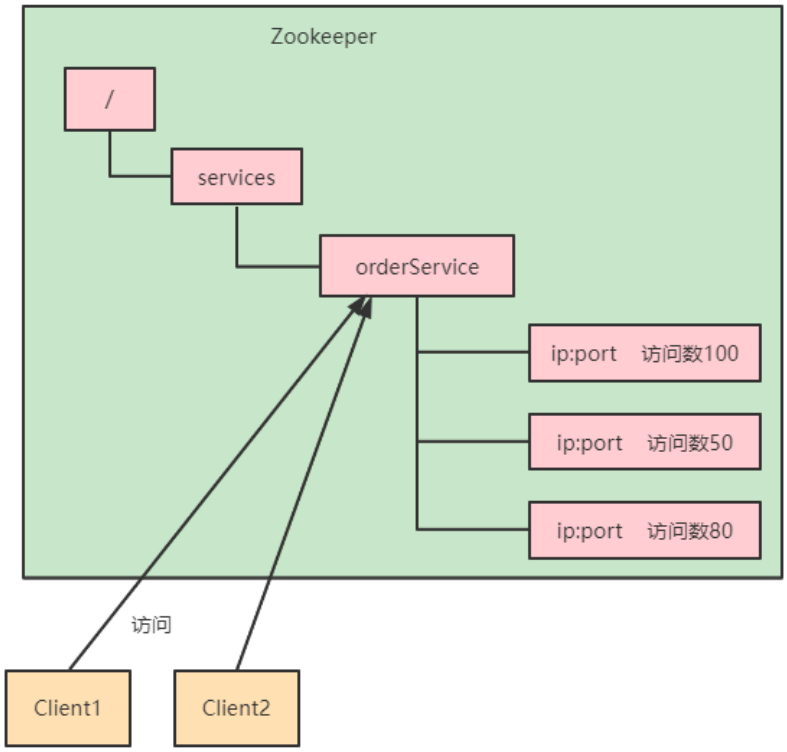
统一集群管理

分布式环境中，实时掌握每个节点的状态是必要的，可根据节点实时状态做出一些调整。
ZooKeeper可以实现实时监控节点状态变化：

- 可将节点信息写入ZooKeeper上的一个ZNode。
- 监听这个ZNode可获取它的实时状态变化。

负载均衡

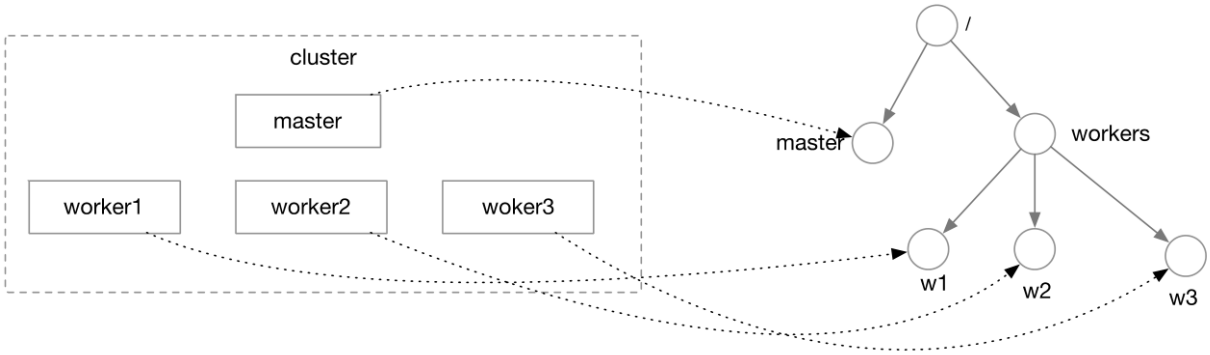
在Zookeeper中记录每台服务器的访问数，让访问数最少的服务器去处理最新的客户端请求



Master-Worker架构

master-work是一个广泛使用的分布式架构。 master-work架构中有一个master负责监控worker的状态，并为
worker分配任务。

- 在任何时刻，系统中最多只能有一个master，不可以出现两个master的情况，多个master共存会导致脑裂。
- 系统中除了处于active状态的master还有一个backup master，如果active master失败了，backup master可以很快的进入active状态。
- master实时监控worker的状态，能够及时收到worker成员变化的通知。master在收到worker成员变化的时候，通常重新进行任务的重新分配。



Zookeeper其他应用场景会在后续课程中详细讲解

3.5 ACL权限控制

zookeeper 的 ACL（Access Control List，访问控制表）权限在生产环境是特别重要的，ACL 权限可以针对节点设置相关读写等权限，保障数据安全性。

ACL 构成

zookeeper 的 acl 通过 **[scheme:id:permissions]** 来构成权限列表。

- **scheme**：授权的模式，代表采用的某种权限机制，包括 world、auth、digest、ip、super 几种。
- **id**：授权对象，代表允许访问的用户。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。
- **permissions**：授权的权限，权限组合字符串，由 cdrwa 组成，其中每个字母代表支持不同权限， 创建权限 create(c)、删除权限 delete(d)、读权限 read(r)、写权限 write(w)、管理权限admin(a)。

模式	描述
world	授权对象只有一个anyone，代表登录到服务器的所有客户端都能对该节点执行某种权限
ip	对连接的客户端使用IP地址认证方式进行认证
auth	使用以添加认证的用户进行认证
digest	使用 用户:密码方式验证

权限类型	ACL简写	描述
read	r	读取节点及显示子节点列表的权限
write	w	设置节点数据的权限
create	c	创建子节点的权限

delete	d	删除子节点的权限
admin	a	设置该节点ACL权限的权限

授权命令	用法	描述
getAcl	getAcl path	读取节点的ACL
setAcl	setAcl path acl	设置节点的ACL
create	create path data acl	创建节点时设置acl
addAuth	addAuth scheme auth	添加认证用户，类似于登录操作

测试

取消节点的读权限后，读取/name节点没有权限

取消节点删除子节点的权限

auth授权模式

创建用户

```
addauth digest fox:123456
```

设置权限

```
setAcl /name auth:fox:123456:cdrwa

# 加密
echo -n fox:123456 | openssl dgst -binary -sha1 | openssl base64
setAcl /name auth:fox:ZsWwgmtnTnx1usRF1voHFJAYGQU=:cdrwa
```

退出客户端，重新连接之后获取/name会没权限，需要添加授权用户。

digest授权模式

```
#设置权限
setAcl /tuling/fox digest:fox:ZsWwgmtnTnx1usRF1voHFJAYGQU=:cdrwa
```

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。
需要在启动脚本上通过添加JVM 参数开启：

```
# DigestAuthenticationProvider中定义
-Dzookeeper.DigestAuthenticationProvider.superDigest=admin:<base64encoded(SHA1(123456))
```

可插拔身份验证接口

ZooKeeper提供了一种权限扩展机制来让用户实现自己的权限控制方式。
要想实现自定义的权限控制机制，需要继承接口AuthenticationProvider，用户通过该接口实现自定义的权限控制。

```
public interface AuthenticationProvider {
 // 返回标识插件的字符串
    String getScheme();
    // 将用户和验证信息关联起来
    KeeperException.Code handleAuthentication(ServerCnxn cnxn, byte authData[]);
    // 验证id格式
    boolean isValid(String id);
    // 将认证信息与ACL进行匹配看是否命中
```

```
    boolean matches(String id, String aclExpr);
    // 是否授权
    boolean isAuthenticated();
}
```

4. Zookeeper集群架构

4.1 集群角色

- Leader： 领导者

事务请求（写操作）的唯一调度者和处理者，保证集群事务处理的顺序性；集群内部各个服务器的调度者。对于create、setData、delete等有写操作的请求，则要统一转发给leader处理，leader需要决定编号、执行操作，这个过程称为事务。

- Follower: 跟随者

处理客户端非事务（读操作）请求（可以直接响应），转发事务请求给Leader；参与集群Leader选举投票。

- Observer: 观察者

对于非事务请求可以独立处理（读操作），对于事务性请求会转发给leader处理。Observer节点接收来自leader的inform信息，更新自己的本地存储，不参与提交和选举投票。通常在不影响集群事务处理能力的前提下提升集群的非事务处理能力。

```
#配置一个ID为3的观察者节点：
server.3=192.168.0.3:2888:3888:observer
```

Observer应用场景：

- 提升集群的读性能。因为Observer和不参与提交和选举的投票过程，所以可以通过往集群里面添加observer节点来提高整个集群的读性能。
- 跨数据中心部署。 比如需要部署一个北京和香港两地都可以使用的zookeeper集群服务，并且要求北京和香港客户的读请求延迟都很低。解决方案就是把香港的节点都设置为observer。

4.2 集群架构

leader节点可以处理读写请求，follower只可以处理读请求。follower在接到写请求时会把写请求转发给leader来处理。

Zookeeper数据一致性保证：

- 全局可线性化(Linearizable)写入：先到达leader的写请求会被先处理，leader决定写请求的执行顺序。
- 客户端FIFO顺序：来自给定客户端的请求按照发送顺序执行。

4.3 三节点Zookeeper集群搭建

环境准备：三台虚拟机

```
192.168.65.163
192.168.65.184
192.168.65.186
```

条件有限也可以在一台虚拟机上搭建zookeeper伪集群

1) 修改zoo.cfg配置，添加server节点配置

```
# 修改数据存储目录
dataDir=/data/zookeeper

#三台虚拟机 zoo.cfg 文件末尾添加配置
server.1=192.168.65.163:2888:3888
server.2=192.168.65.184:2888:3888
server.3=192.168.65.186:2888:3888
```

> server.A=B:C:D
> A 是一个数字，表示这个是第几号服务器； 集群模式下配置一个文件 myid，这个文件在 dataDir 目录下，这个文件里面有一个数据 就是 A 的值，Zookeeper 启动时读取此文件，拿到里面的数据与 zoo.cfg 里面的配置信息比较从而判断到底是哪个server。
> B 是这个服务器的地址；
> C 是这个服务器Follower与集群中的Leader服务器交换信息的端口；
> D 是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，选出一个新的Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

2）创建 myid 文件，配置服务器编号

在dataDir对应目录下创建 myid 文件，内容为对应ip的zookeeper服务器编号

```
cd /data/zookeeper
# 在文件中添加与 server 对应的编号（注意：上下不要有空行，左右不要有空格）
vim myid
```

注意：添加 myid 文件，一定要在 Linux 里面创建，在 notepad++里面很可能乱码

3）启动zookeeper server集群

启动前需要关闭防火墙(生产环境需要打开对应端口)

```
# 分别启动三个节点的zookeeper server
bin/zkServer.sh start
# 查看集群状态
bin/zkServer.sh status
```

常见问题：

如果服务启动出现下面异常：

原因：

1. zoo.cfg配置错误
2. 防火墙没关

```
#centos7
# 检查防火墙状态
systemctl status firewalld
#关闭防火墙
systemctl stop firewalld
systemctl disable firewalld
```

4.4 Zookeeper四字命令使用

用户可以使用Zookeeper四字命令获取 zookeeper 服务的当前状态及相关信息。用户在客户端可以通过 nc（netcat） 向 zookeeper 提交相应的命令。
安装 nc 命令：

```
# centos
yum install nc
```

四字命令格式：

```
echo [command] | nc [ip] [port]
```

ZooKeeper 常用四字命令主要如下：

四字命令	功能描述
conf	3.3.0版本引入的。打印出服务相关配置的详细信息。
cons	3.3.0版本引入的。列出所有连接到这台服务器的客户端全部连接/会话详细信息。包括"接受/发送"的包数量、会话id、操作延迟、最后的操作执行等等信息。
crst	3.3.0版本引入的。重置所有连接的连接和会话统计信息。
dump	列出那些比较重要的会话和临时节点。这个命令只能在leader节点上有用。
envi	打印出服务环境的详细信息。
reqs	列出未经处理的请求
ruok	测试服务是否处于正确状态。如果确实如此，那么服务返回"imok"，否则不做任何相应。
stat	输出关于性能和连接的客户端的列表。
srst	重置服务器的统计。
srvr	3.3.0版本引入的。列出连接服务器的详细信息
wchs	3.3.0版本引入的。列出服务器watch的详细信息。

wchc	3.3.0版本引入的。通过session列出服务器watch的详细信息，它的输出是一个与watch相关的会话的列表。
wchp	3.3.0版本引入的。通过路径列出服务器watch的详细信息。它输出一个与session相关的路径。
mntr	3.4.0版本引入的。输出可用于检测集群健康状态的变量列表

https://zookeeper.apache.org/doc/current/zookeeperAdmin.html#sc_4lw

开启四字命令

方法1： 在zoo.cfg 文件里加入配置项让这些指令放行

```
#开启四字命令
4lw.commands.whitelist=*
```

方法2：在zk的启动脚本zkServer.sh中新增放行指令

```
#添加JVM环境变量-Dzookeeper.4lw.commands.whitelist=*
ZOOMAIN="-Dzookeeper.4lw.commands.whitelist=* ${ZOOMAIN}"
```

stat 命令

stat 命令用于查看 zk 的状态信息，实例如下：

```
 echo stat | nc 192.168.65.186 2181
```

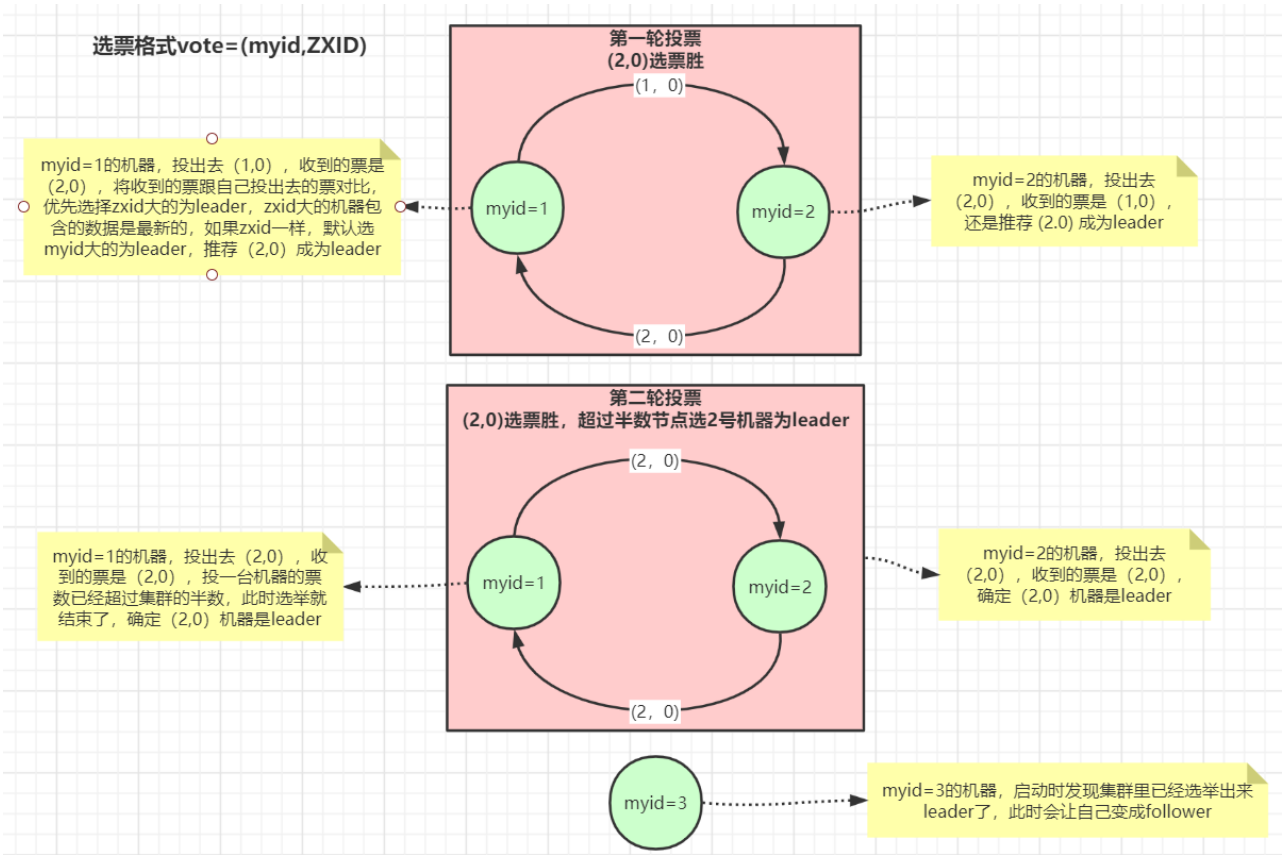
ruok 命令

ruok 命令用于查看当前 zkserver 是否启动，若返回 imok 表示正常。

```
echo ruok | nc 192.168.65.186 2181
```

4.5 Zookeeper选举原理

ZooKeeper的Leader选举过程是基于投票和对比规则的，确保集群中选出一个具有最高优先级的服务器作为Leader来处理客户端请求。以服务启动期间选举为例：



投票对比规则如下：

- 首先比较epoch，选取具有最大epoch的服务器。epoch用于区分不同的选举轮次，每次重新选举时都会增加epoch。
- 如果epoch相同，则比较zxid（事务ID），选取事务ID最大的服务器。zxid表示最后一次提交的事务ID。
- 如果zxid也相同，则比较myid（服务器ID），选取服务器ID最大的服务器。

```
/**
 * Check if a pair (server id, zxid) succeeds our
 * current vote.
 *
 */
protected boolean totalOrderPredicate(long newId, long newZxid, long newEpoch, long curId, long curZxid, long curEpoch) {
    LOG.debug(
        "id: {}, proposed id: {}, zxid: 0x{}, proposed zxid: 0x{}",
        newId,
        curId,
```

```
            Long.toHexString(newZxid),
            Long.toHexString(curZxid));

    if (self.getQuorumVerifier().getWeight(newId) == 0) {
        return false;
    }

    /*
     * We return true if one of the following three cases hold:
     * 1- New epoch is higher
     * 2- New epoch is the same as current epoch, but new zxid is higher
     * 3- New epoch is the same as current epoch, new zxid is the same
     *  as current zxid, but server id is higher.
     */

    return ((newEpoch > curEpoch)
            || ((newEpoch == curEpoch)
                && ((newZxid > curZxid)
                    || ((newZxid == curZxid)
                        && (newId > curId)))));
}
```

zxid的数据结构

根据这个工具类，可以得出zxid的数据结构的一些信息。

1. zxid是一个64位的整数，由高32位的epoch和低32位的counter组成。
2. epoch表示ZooKeeper服务器的逻辑时期（logical epoch），它是一个相对时间的概念，用于区分不同的Leader选举周期。
3. counter是一个在每个时期（epoch）内递增的计数器，用于标识事务的顺序。

```
public class ZxidUtils {

    public static long getEpochFromZxid(long zxid) {
        return zxid >> 32L;
    }
    public static long getCounterFromZxid(long zxid) {
        return zxid & 0xffffffffL;
    }
```

```
9       public static long makeZxid(long epoch, long counter) {
10          return (epoch << 32L) | (counter & 0xffffffffL);
11      }
12      public static String zxidToString(long zxid) {
13          return Long.toHexString(zxid);
14      }
15
16  }
```