腾讯课堂-图灵学院
主讲老师：郭嘉（QQ:2790284115）
课程资料：小十（QQ: 895900002）

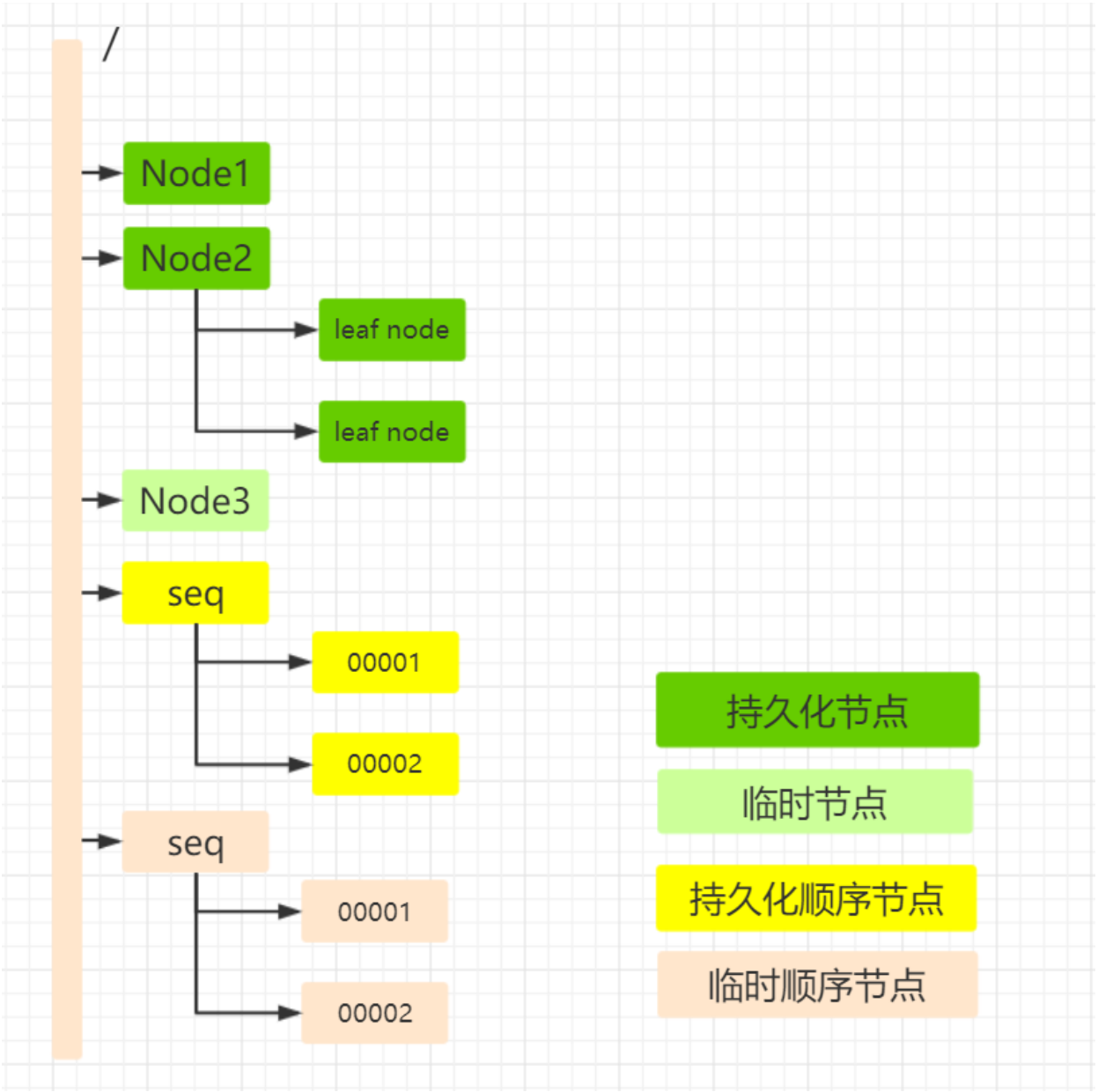
Zookeeper 是什么

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。

上面的解释有点抽象，简单来说zookeeper=文件系统+监听通知机制。

1、 文件系统

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)**，和文件系统一样，我们能够自由的增加、删除znode，在一个znode下增加、删除子znode，唯一的不同在于znode是可以存储数据的。

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

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

2、 监听通知机制

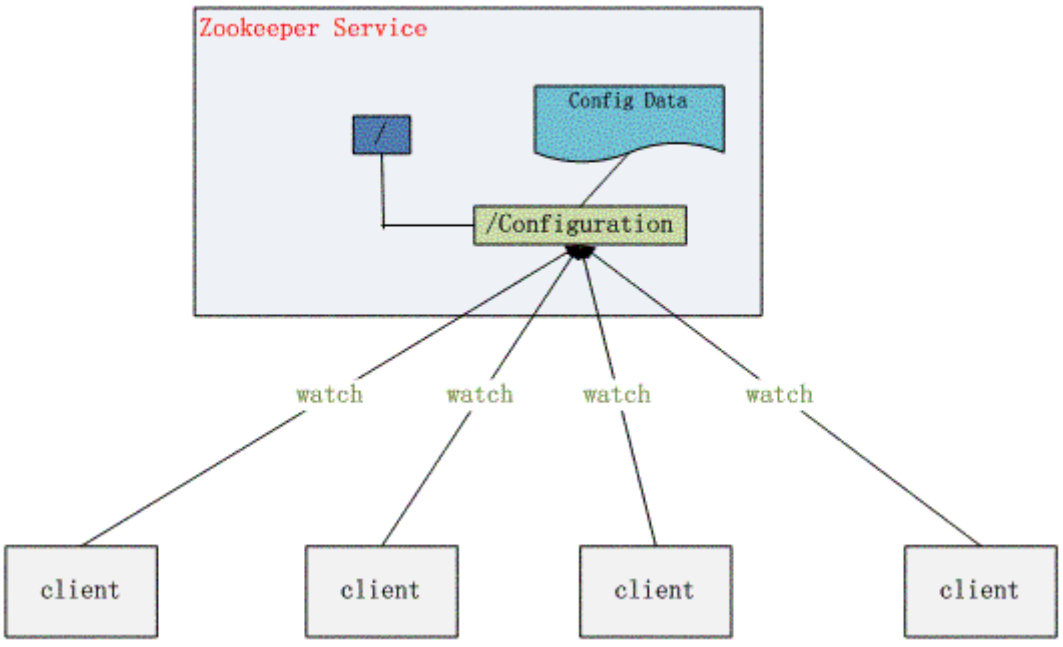
客户端注册监听它关心的目录节点，当目录节点发生变化（数据改变、被删除、子目录节点增加删除）时，zookeeper会通知客户端。

就这么简单，下面我们看看Zookeeper能做点什么呢？

Zookeeper 能做什么

zookeeper功能非常强大，可以实现诸如分布式应用配置管理、统一命名服务、状态同步服务、集群管理等功能，我们这里拿比较简单的分布式应用配置管理为例来说明。

假设我们的程序是分布式部署在多台机器上，如果我们要改变程序的配置文件，需要逐台机器去修改，非常麻烦，现在把这些配置全部放到zookeeper上去，保存在 zookeeper 的某个目录节点中，然后所有相关应用程序对这个目录节点进行监听，一旦配置信息发生变化，每个应用程序就会收到 zookeeper 的通知，然后从 zookeeper 获取新的配置信息应用到系统中。



如上，你大致应该了解zookeeper是个什么东西，大概能做些什么了，我们马上来学习下zookeeper的安装及使用，并开发一个小程序来实现zookeeper这个配置管理的功能

zookeeper安装

Step1：配置JAVA环境，检验环境：

```
java -version
```

Step2: 下载解压 zookeeper

```
wget http://archive.apache.org/dist/zookeeper/zookeeper-3.4.12/zookeeper-3.4.12.tar.gz
tar -zxvf zookeeper-3.4.12.tar.gz
cd zookeeper-3.4.12
```

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
```

Step5: 检测是否启动成功

```
bin/zkCli.sh start
```

Zookeeper使用

使用命令行操作zookeeper

1、使用 ls 命令来查看当前 ZooKeeper 中所包含的内容

```
[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper]
```

2、创建一个新的 znode ，使用 create /zkPro myData

```
[zk: localhost:2181(CONNECTED) 1] create /zkPro myData
Created /zkPro
```

3、再次使用 ls 命令来查看现在 zookeeper 中所包含的内容：

```
[zk: localhost:2181(CONNECTED) 3] ls /
[zookeeper, zkPro]
```

4、下面我们运行 get 命令来确认第二步中所创建的 znode 是否包含我们所创建的字符串：

```
[zk: localhost:2181(CONNECTED) 4] get /zkPro
myData
cZxid = 0x100000014
ctime = Wed Jul 11 22:21:07 CST 2018
mZxid = 0x100000014
mtime = Wed Jul 11 22:21:07 CST 2018
pZxid = 0x100000014
cversion = 0
dataVersion = 0
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 6
numChildren = 0
```

5、下面我们通过 set 命令来对 zk 所关联的字符串进行设置：

```
[zk: localhost:2181(CONNECTED) 5] set /zkPro myData123
cZxid = 0x100000014
ctime = Wed Jul 11 22:21:07 CST 2018
mZxid = 0x100000015
mtime = Wed Jul 11 22:24:47 CST 2018
pZxid = 0x100000014
cversion = 0
dataVersion = 1
aclVersion = 0
ephemeralOwner = 0x0
dataLength = 9
numChildren = 0
```

6、下面我们将刚才创建的 znode 删除

```
[zk: localhost:2181(CONNECTED) 6] delete /zkPro
[zk: localhost:2181(CONNECTED) 7]
```

Stat

- cZxid：创建znode的事务ID。
- mZxid：最后修改znode的事务ID。
- pZxid：最后修改添加或删除子节点的事务ID。
- 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。

Zookeeper集群模式安装

本例搭建的是伪集群模式，即一台机器上启动三个zookeeper实例组成集群，真正的集群模式无非就是实例IP地址不同，搭建方法没有区别

Step1：配置JAVA环境，检验环境：保证是jdk7 及以上即可

```
java -version
```

Step2：下载并解压zookeeper

```
wget http://archive.apache.org/dist/zookeeper/zookeeper-3.4.12/zookeeper-3.4.12.tar.gz
tar -zxvf zookeeper-3.4.12.tar.gz
cd zookeeper-3.4.12
```

Step3：重命名 zoo_sample.cfg文件

```
 cp conf/zoo_sample.cfg conf/zoo-1.cfg
```

Step4：修改配置文件zoo-1.cfg，原配置文件里有的，修改成下面的值，没有的则加上

```
# vim conf/zoo-1.cfg
dataDir=/usr/local/data/zookeeper-1
clientPort=2181
server.1=127.0.0.1:2888:3888
server.2=127.0.0.1:2889:3889
server.3=127.0.0.1:2890:3890
```

配置说明

- tickTime：这个时间是作为 Zookeeper 服务器之间或客户端与服务器之间维持心跳的**时间间隔**，也就是每个 tickTime 时间就会发送一个心跳。
- initLimit：这个配置项是用来配置 Zookeeper 接受客户端（这里所说的客户端不是用户连接 Zookeeper 服务器的客户端，而是 Zookeeper 服务器集群中连接到 Leader 的 Follower 服务器）初始化连接时最长能忍受多少个心跳时间间隔数。当已经超过 10个心跳的时间（也就是 tickTime）长度后 Zookeeper 服务器还没有收到

客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是 10*2000=20秒

- syncLimit：这个配置项标识 Leader 与 Follower 之间发送消息，请求和应答时间长度，最长不能超过多少个 tickTime 的时间长度，总的时间长度就是 5*2000=10秒
- dataDir：顾名思义就是 Zookeeper 保存数据的目录，默认情况下，Zookeeper 将写数据的日志文件也保存在这个目录里。
- clientPort：这个端口就是客户端连接 Zookeeper 服务器的端口，Zookeeper 会监听这个端口，接受客户端的访问请求。
- server.A=B：C：D：其中 A 是一个数字，表示这个是第几号服务器；B 是这个服务器的 ip 地址；C 表示的是这个服务器与集群中的 Leader 服务器交换信息的端口；D 表示的是万一集群中的 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。如果是伪集群的配置方式，由于 B 都是一样，所以不同的 Zookeeper 实例通信端口号不能一样，所以要给它们分配不同的端口号。

Step4：再从zoo-1.cfg复制两个配置文件zoo-2.cfg和zoo-3.cfg，只需修改dataDir和clientPort不同即可

```
cp conf/zoo-1.cfg conf/zoo-2.cfg
cp conf/zoo-1.cfg conf/zoo-3.cfg
vim conf/zoo-2.cfg
dataDir=/usr/local/data/zookeeper-2
clientPort=2182
vim conf/zoo-2.cfg
dataDir=/usr/local/data/zookeeper-3
clientPort=2183
```

Step5：标识Server ID

创建三个文件夹/usr/local/data/zookeeper-1，/usr/local/data/zookeeper-2，/usr/local/data/zookeeper-3，在每个目录中创建文件myid 文件，写入当前实例的server id，即1.2.3

```
cd /usr/local/data/zookeeper-1
vim myid
1
cd /usr/local/data/zookeeper-2
vim myid
2
```

```
cd /usr/local/data/zookeeper-3
vim myid
3
```

Step6：启动三个zookeeper实例

```
bin/zkServer.sh start conf/zoo-1.cfg
bin/zkServer.sh start conf/zoo-2.cfg
bin/zkServer.sh start conf/zoo-3.cfg
```

Step7：检测集群状态，也可以直接用命令“zkCli.sh -server IP:PORT”连接zookeeper服务端检测

```
[root@centos-new zookeeper-3.4.12]# bin/zkServer.sh status conf/zoo-1.cfg
ZooKeeper JMX enabled by default
Using config: conf/zoo-1.cfg
Mode: follower
[root@centos-new zookeeper-3.4.12]# bin/zkServer.sh status conf/zoo-2.cfg
ZooKeeper JMX enabled by default
Using config: conf/zoo-2.cfg
Mode: leader
[root@centos-new zookeeper-3.4.12]# bin/zkServer.sh status conf/zoo-3.cfg
ZooKeeper JMX enabled by default
Using config: conf/zoo-3.cfg
Mode: follower
```

```
bin/zkCli.sh -server 192.168.6.134:2181
```

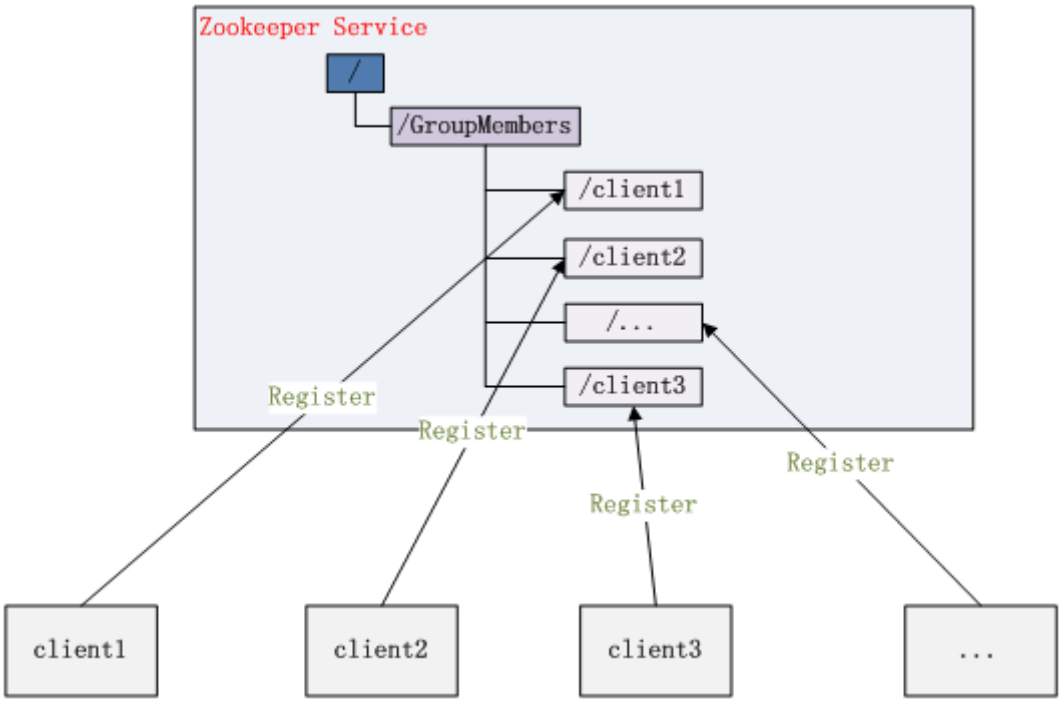
Zookeeper其它应用介绍

1、 集群管理

所谓集群管理无在乎两点：是否有机器退出和加入、选举master。

对于第一点，所有机器约定在父目录GroupMembers下创建**临时目录节点**，然后监听父目录节点的子节点变化消息。一旦有机器挂掉，该机器与 zookeeper的连接断开，其所创建的临时目录节点被删除，所有其他机器都收到通知：某个兄弟目录被删除，于是，所有人都知道：它上船了。新机器加入 也是类似，所有机器收到通知：新兄弟目录加入，highcount又有了。

对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为master就好。

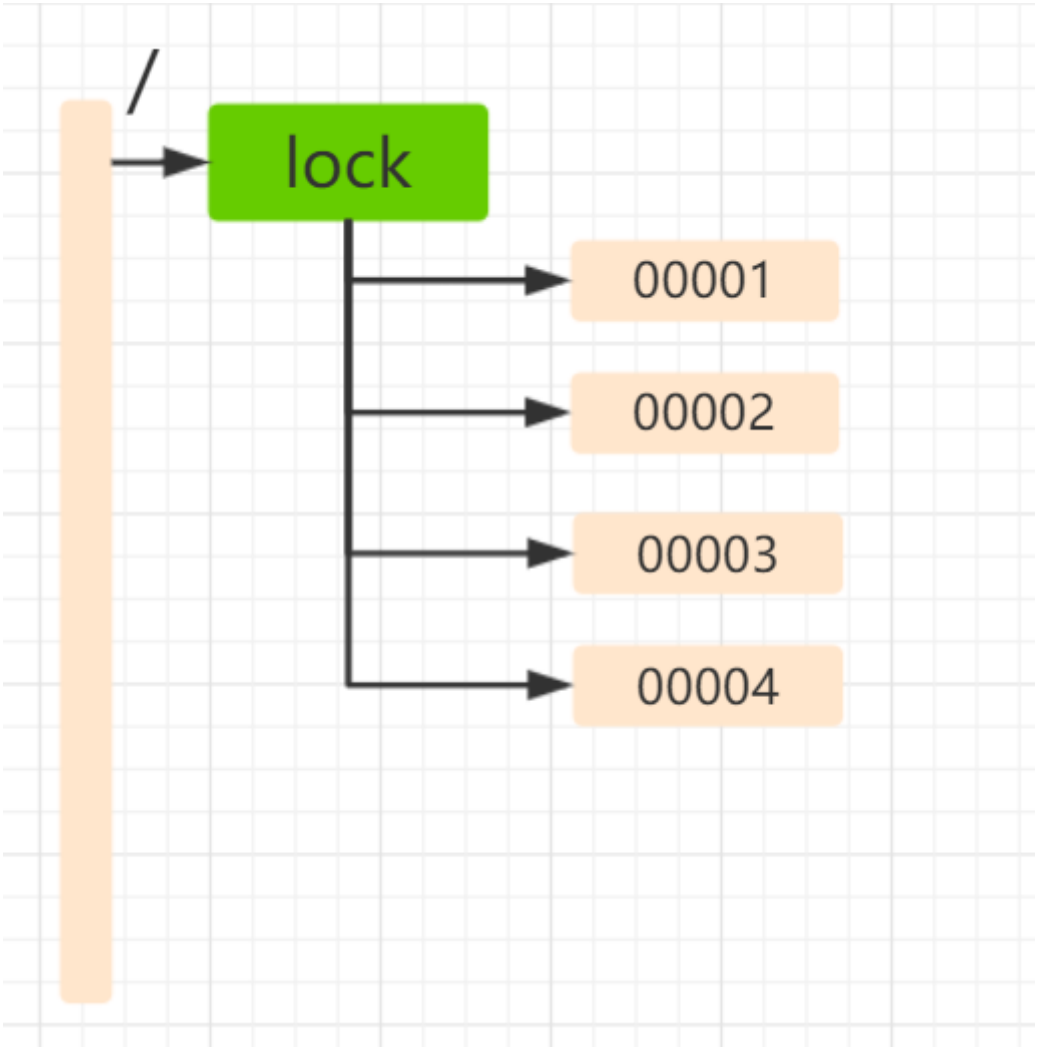


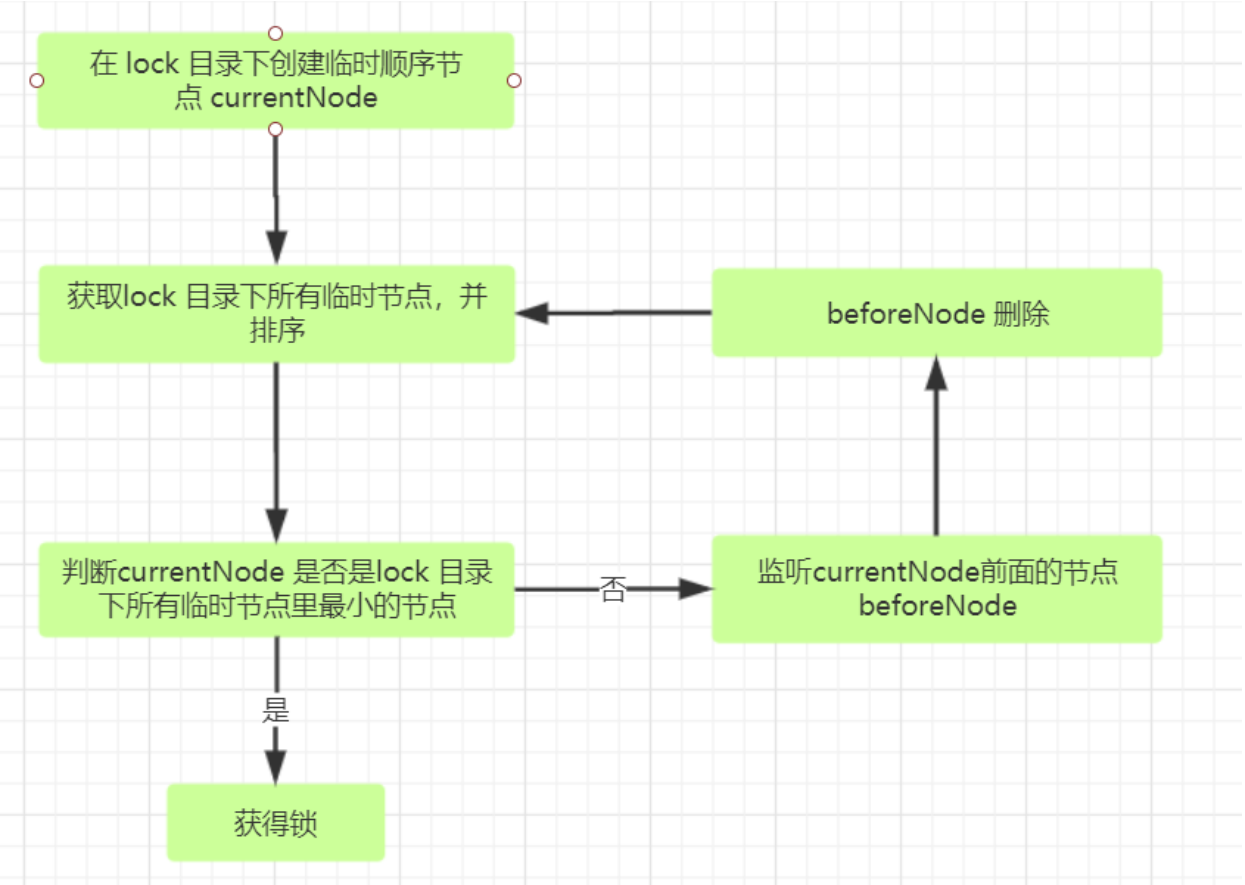
2、 分布式锁

有了zookeeper的一致性文件系统，锁的问题变得容易。锁服务可以分为两类，一个是保持独占，另一个是控制时序。

对于第一类，我们将zookeeper上的一个znode看作是一把锁，通过createznode的方式来实现。所有客户端都去创建 /distribute_lock 节点，最终成功创建的那个客户端也即拥有了这把锁。厕所有言：来也冲冲，去也冲冲，用完删除掉自己创建的distribute_lock 节点就释放出锁。

对于第二类， /distribute_lock 已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选master一样，编号最小的获得锁，用完删除，依次方便。


/
lock
00001
00002
00003
00004



3、队列管理

两种类型的队列：

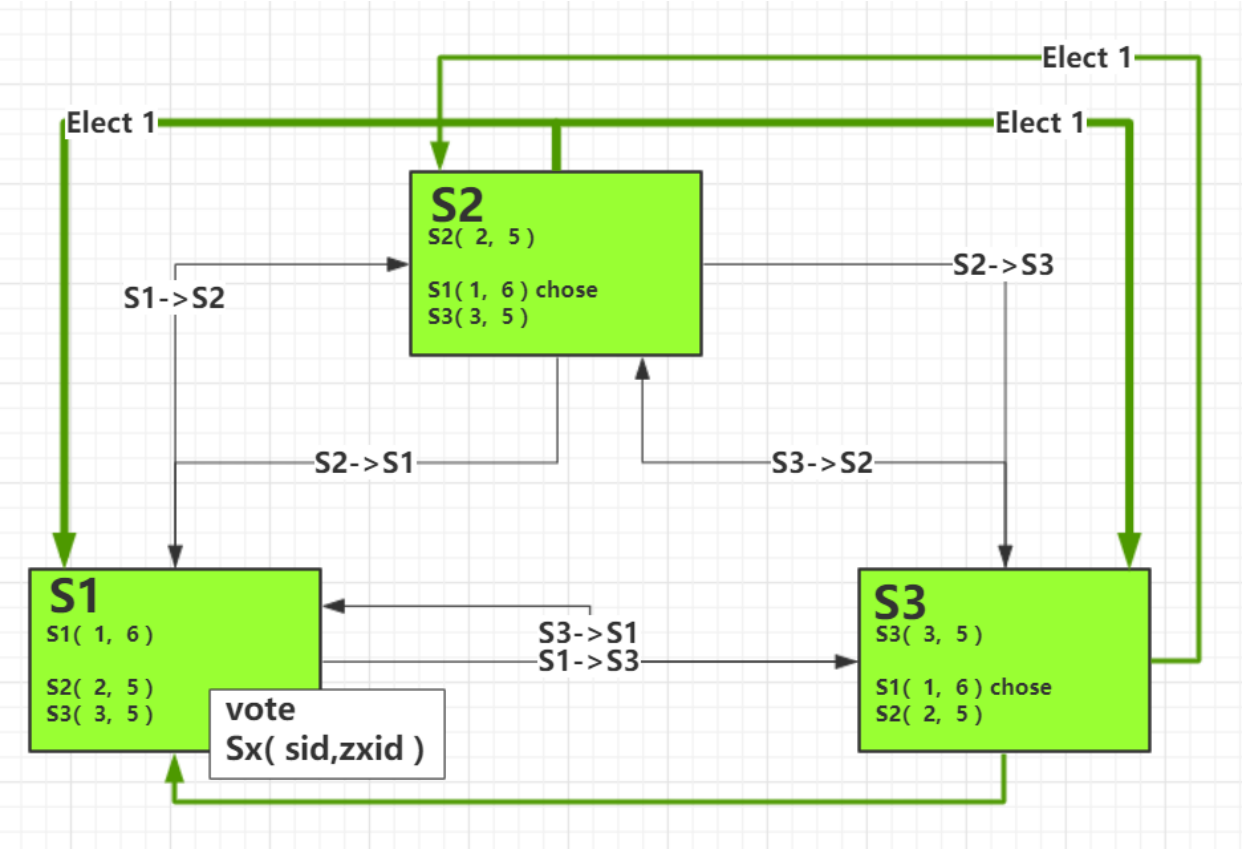
1、 同步队列，当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达。

2、队列按照 FIFO 方式进行入队和出队操作。

第一类，在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。

第二类，和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。

选举原理（FastLeaderElection）



选举时，各节点**首先**将 选举群首的票投给自己， 即服务器会将自身的 zxid 和 sid 发生给集群中的其他节点， **sid**表示这张选票投的服务器id，**zxid**表示这张选票投的服务器上最大的事务id，同时也会接收到其他服务器的选票，接收到其他服务器的选票后，可以根据选票信息中的zxid来与自己当前所投的服务器上的最大zxid来进行比较，如果其他服务器的选票中的zxid较大，则表示自己当前所投的机器数据没有接收到的选票所投的服务器上的数据新，所以本节点需要**改票**，改成投给和刚刚接收到的选票一样。

原则 1： 数据最新原则，zxid 表示数据的新旧，一个节点最新的zxid越大则该节点的数据越新,所以 Zookeeper选举时会根据zxid的大小来作为投票的基本规则。

原则 2： 当zxid 一致时，选sid大的。

原则 3： 过半原则，获得过半票数的节点，就是主节点。