# Zookeeper and ZNode and Recipes

ZNode是以目录结构存储的；

可以在ZNode上设置Watches，数据变更时Watches会得到通知，并清除这个Watches。

Data Access ZK是作为协调服务使用的，别做数据库。（内存，数据节点）

Ephemeral Nodes 和创建节点的session保持同样生命周期（active）的ZNode。

Sequence Node -- Unique Naming

在同一个节点根据创建顺序创建以整数作为名字的节点

3.5.3版本后

TTL节点 在所创建PERSISTENT或者PERSISTENT\_SEQUENTIAL的节点上设置TTL

Container节点 作为leader选举和实现锁的节点

Zookeeper中的时间

Zxid 递增的zk事务ID，ID小的修改表示发生的时间在ID大修改的之前。指修改zk状态。

Version numbers 指node的版本好，递增。三个版本好，version/cversion（子节点）/aversion（修改acl）

Tick 超时时间内平均两个tick

Real time，不用real time

zk原语

Barriers

使一组processes可以同步任务的开始和结束。在某一个节点下创建节点，数据可以是创建者的id，当有足够多的子节点时，任务开始。任务完成时，删除对应子节点，并等待所有子节点删除后正式结束任务。

Producer-Consumer Queues

按照顺序创建递增数字的节点，消费者端负责删除，只有一个C可以成功删除。个人不提倡这种用法。

LOCK

Lock1

Clients wishing to obtain a lock do the following:

1. Call **create( )** with a pathname of "locknode/guid-lock-" and the sequence and ephemeral flags set. The guid is needed in case the create() result is missed. See the note below. - **guid保证信息（指这个数据是我创建的）不会丢失。**
2. Call **getChildren( )** on the lock node without setting the watch flag (this is important to avoid the herd effect). - **不用watch，否则可能接受到无用的信息-herd effect。**
3. If the pathname created in step **1** has the lowest sequence number suffix, the client has the lock and the client exits the protocol. - **是我第一个（序号最低）创建的，那说明我得到了lock**
4. The client calls **exists( )** with the **watch flag set** on the path in the lock directory with the next lowest sequence number. - **如果第一个不是我，看看第二个是不是我。**
5. if **exists( )** returns null, go to step **2**. Otherwise, wait for a notification for the pathname from the previous step before going to step **2**. **如果第二个不存在，那继续查下一个（有可能是我）。如果存在，getChildren之后数据变动可以通过通知获取，不需要立刻去继续跳转第2步。如果之前的lock holder变动，应该有通知来。**

| **Obtaining a read lock:** | **Obtaining a write lock:** |
| --- | --- |
| 1. Call **create( )** to create a node with pathname "**guid-/read-**". This is the lock node use later in the protocol. Make sure to set both the sequence and ephemeral flags. | 1. Call **create( )** to create a node with pathname "guid-/write-". This is the lock node spoken of later in the protocol. Make sure to set both sequence and ephemeral flags. |
| 2. Call **getChildren( )** on the lock node without setting the watch flag - this is important, as it avoids the herd effect. | 2. Call **getChildren( )** on the lock node without setting the watch flag - this is important, as it avoids the herd effect. |
| 3. If there are **no children with a pathname starting with "write-" and having a lower sequence number than the node created in step **1****, the client has the lock and can exit the protocol. **我是否在writeLock前面** | 3. If there are no children with a lower sequence number than the node created in step **1**, the client has the lock and the client exits the protocol. **我是第一个，wlock了** |
| 1. Otherwise, call **exists( )**, with watch flag, set on the node in lock directory with pathname staring with "write-" having the next lowest sequence number. **是否是writer lock了，看队列前面一个是否是wrLock，盯住前一个wrLock，等它完成。** | 4. Call **exists( ),** with watch flag set, on the node with the pathname that has **the next lowest sequence number**. |
| 5. If **exists( )** returns false, goto step **2**. | 5. If **exists( )** returns false, goto step **2**. Otherwise, wait for a notification for the pathname from the previous step before going to step **2**. **直接查下一个是不是我（跳转步骤2）** |
| 6. Otherwise, wait for a notification for the pathname from the previous step before going to step **2。**有wrLock变动时继续查。**** |  |

#### Revocable Shared Locks

With minor modifications to the Shared Lock protocol, you make shared locks revocable by modifying the shared lock protocol:

In step **1**, of both obtain reader and writer lock protocols, call **getData( )** with watch set, immediately after the call to **create( )**. **If the client subsequently receives notification for the node it created in step **1****, it does another **getData( )** on that node, with watch set and looks for the string "unlock", which signals to the client that it must release the lock. This is because, according to this shared lock protocol, you can request the client with the lock give up the lock by calling **setData()** on the lock node, writing "unlock" to that node.

Note that this protocol requires the lock holder to consent to releasing the lock. Such consent is important, especially if the lock holder needs to do some processing before releasing the lock. Of course you can always implement Revocable Shared Locks with Freaking Laser Beams by stipulating in your protocol that the revoker is allowed to delete the lock node if after some length of time the lock isn't deleted by the lock holder.

### Two-phased Commit

A two-phase commit protocol is an algorithm that lets all clients in a distributed system agree either to commit a transaction or abort.

In ZooKeeper, you can implement a two-phased commit by **having a coordinator create a transaction node, say "/app/Tx"**, and one child node per participating site, say "/app/Tx/s\_i". When coordinator creates the child node, it leaves the content undefined. Once each site involved in the transaction receives the transaction from the coordinator, t**he site reads each child node and sets a watch**. **Each site then processes the query and votes "commit" or "abort" by writing to its respective node.** Once the write completes, the other sites are notified, and as soon as all sites have all votes, they can decide either "abort" or "commit". Note that a node can decide "abort" earlier if some site votes for "abort". **在文件夹子节点里写commit或者abort。如有任何一方abort，则事务abort；所有参与方都同意，则commit。**

An interesting aspect of this implementation is that the **only role of the coordinator is to decide upon the group of sites, to create the ZooKeeper nodes, and to propagate the transaction to the corresponding sites.** In fact, even propagating the transaction can be done through ZooKeeper by writing it in the transaction node.

There are two important drawbacks of the approach described above. One is the message complexity, which is O(n²). The second is the impossibility of detecting failures of sites through ephemeral nodes. To detect the failure of a site using ephemeral nodes, it is necessary that the site create the node.

To solve the first problem, you can have only the coordinator notified of changes to the transaction nodes, and then notify the sites once coordinator reaches a decision. Note that **this approach is scalable, but it's is slower too,** as it requires all communication to go through the coordinator.

To address the second problem, **you can have the coordinator propagate the transaction to the sites, and have each site creating its own ephemeral node**. **这不是正常做法吗？**

### Leader Election

A simple way of doing leader election with ZooKeeper is to use the **SEQUENCE|EPHEMERAL** flags when creating znodes that represent "proposals" of clients. The idea is to have a znode, say "/election", such that **each znode creates a child znode "/election/guid-n\_" with both flags SEQUENCE|EPHEMERAL**. With the sequence flag, ZooKeeper automatically appends a sequence number that is greater than any one previously appended to a child of "/election". **The process that created the znode with the smallest appended sequence number is the leader. - 多数派呢？**

That's not all, though. It is important to watch for failures of the leader, so that a new client arises as the new leader in the case the current leader fails. A trivial solution is to have all application processes watching upon the current smallest znode, and checking if they are the new leader when the smallest znode goes away (note that the smallest znode will go away if the leader fails because the node is ephemeral). **But this causes a herd effect**: upon a failure of the current leader, all other processes receive a notification, and execute getChildren on "/election" to obtain the current list of children of "/election". If the number of clients is large, it causes a spike on the number of operations that ZooKeeper servers have to process. **To avoid the herd effect, it is sufficient to watch for the next znode down on the sequence of znodes**. If a client receives a notification that the znode it is watching is gone, then it becomes the new leader in the case that there is no smaller znode（**顺位继承，我看我前一个有没有挂掉，没有挂掉，则永无指望**）. Note that this avoids the herd effect by not having all clients watching the same znode.

Here's the pseudo code:

Let ELECTION be a path of choice of the application. To volunteer to be a leader:

1. Create znode **z with path "ELECTION/guid-n\_"** with both SEQUENCE and EPHEMERAL flags;
2. Let C be the children of "ELECTION", and i be the sequence number of z;
3. Watch for changes on "ELECTION/guid-n\_j", where j is the largest sequence number such that j < i and n\_j is a znode in C; **盯住前一个**

Upon receiving a notification of znode deletion:

1. Let C be the new set of children of ELECTION;
2. If z is the smallest node in C, then execute leader procedure;
3. Otherwise, watch for changes on "ELECTION/guid-n\_j", where j is the largest sequence number such that j < i and n\_j is a znode in C;

Notes:

Note that the znode having no preceding znode on the list of children does not imply that the creator of this znode is aware that it is the current leader. Applications may consider creating a separate znode to acknowledge that the leader has executed the leader procedure.

See the [note for Locks](http://zookeeper.apache.org/doc/r3.5.6/recipes.html" \l "sc_recipes_GuidNote) on how to use the guid in the node.

1. <http://zookeeper.apache.org/doc/r3.5.6/recipes.html#sc_recipes_GuidNote>