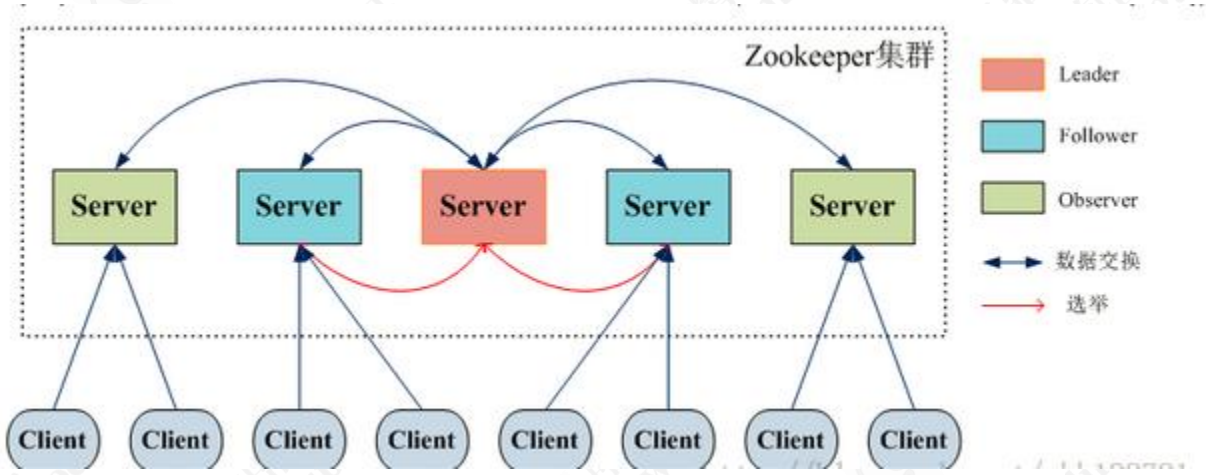
zookeeper 的名词复盘

集群角色



数据模型

zookeeper 的视图结构和标准的文件系统非常类似，每一个节点称之为 ZNode，是 zookeeper 的最小单元。每个 znode 上都可以保存数据以及挂载子节点。构成一个层次化的树形结构

持久节点（PERSISTENT）

创建后会一直存在 zookeeper 服务器上，直到主动删除

持久有序节点（PERSISTENT_SEQUENTIAL）

每个节点都会为它的一级子节点维护一个顺序

临时节点(EPHEMERAL)

临时节点的生命周期和客户端的会话绑定在一起，当客户端会话失效该节点自动清理

临时有序节点(EPHEMERAL)

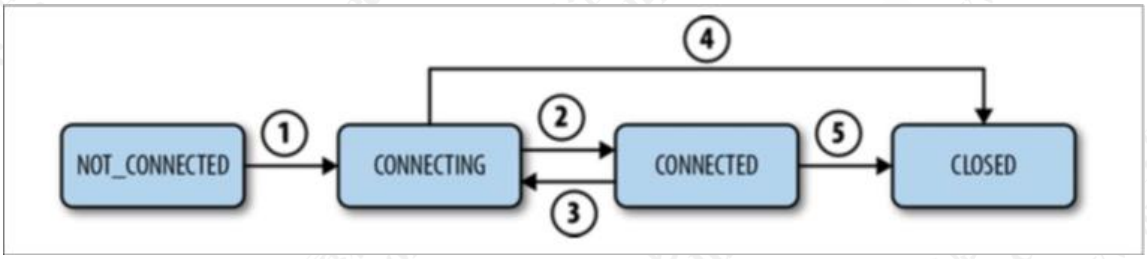
在临时节点的基础上多了一个顺序性

CONTAINER 当子节点都被删除后，Container 也随即删除

PERSISTENT_WITH_TTL 超过 TTL 未被修改，且没有子节点

PERSISTENT_SEQUENTIAL_WITH_TTL 客户端断开连接后不会自动删除 Znode，如果该 Znode 没有子 Znode 且在给定 TTL 时间内无修改，该 Znode 将会被删除；TTL 单位是毫秒，必须大于 0 且小于或等于 EphemeralType.MAX_TTL

会话



1. Client 初始化连接，状态转为 CONNECTING(①)
2. Client 与 Server 成功建立连接，状态转为 CONNECTED(②)
3. Client 丢失了与 Server 的连接或者没有接受到 Server 的响应，状态转为 CONNECTING(③)

4. Client 连上另外的 Server 或连接上了之前的 Server，状态转为 CONNECTED(②)
5. 若会话过期(是 Server 负责声明会话过期，而不是 Client)，状态转为 CLOSED(⑤)，状态转为 CLOSED
6. Client 也可以主动关闭会话(④)，状态转为 CLOSED

Stat 状态信息

每个节点除了存储数据内容以外，还存储了数据节点本身的一些状态信息，通过 get 命令可以获得状态信息的详细内容

状态属性	说　明
czxid	即 Created ZXID，表示该数据节点被创建时的事务 ID
mzxid	即 Modified ZXID，表示该节点最后一次被更新时的事务 ID
ctime	即 Created Time，表示节点被创建的时间
mtime	即 Modified Time，表示该节点最后一次被更新的时间
version	数据节点的版本号。关于 ZooKeeper 中版本相关的内容，将在 7.1.3 节中做详细讲解
cversion	子节点的版本号
aversion	节点的 ACL 版本号
ephemeralOwner	创建该临时节点的会话的 sessionID。如果该节点是持久节点，那么这个属性值为 0
dataLength	数据内容的长度
numChildren	当前节点的子节点个数
pzxid	表示该节点的子节点列表最后一次被修改时的事务 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

版本-保证分布式数据原子性

zookeeper 为数据节点引入了版本的概念，每个数据节点都有三类版本信息，对数据节点任何更新操作都会引起版本号的变化

版本类型	说　明
version	当前数据节点数据内容的版本号
cversion	当前数据节点子节点的版本号
aversion	当前数据节点 ACL 变更版本号

版本有点和我们经常使用的乐观锁类似。这里有两个概念说一下，一个是乐观锁，一个是悲观锁

悲观锁：是数据库中一种非常典型且非常严格的并发控制策略。假如一个事务 A 正在对数据进行处理，那么在整个处理过程中，都会将数据处于锁定状态，在这期间其他事务无法对数据进行更新操作。

乐观锁：乐观锁和悲观锁正好想法，它假定多个事务在处理过程中不会彼此影响，因此在事务处理过程中不需要进行加锁处理，如果多个事务对同一数据做更改，那么在更新请求提交之前，每个事务都会首先检查当前事务读取数据后，是否有其他事务对数据进行了修改。如果有修改，则回滚事务再回到 zookeeper，version 属性就是用来实现乐观锁机制的“写入校验”

Watcher

zookeeper 提供了分布式数据的发布/订阅功能，zookeeper 允许客户端向服务端注册一个 watcher 监听，当服务端的一些指定事件触发了 watcher，那么服务端就会向客户端发送

一个事件通知。

值得注意的是，Watcher 通知是一次性的，即一旦触发一次通知后，该 Watcher 就失效了，因此客户端需要反复注册 Watcher，即程序中在 process 里面又注册了 Watcher，否则，将无法获取 c3 节点的创建而导致子节点变化的事件。

Zookeeper 基于 Java 访问

针对 zookeeper，比较常用的 Java 客户端有 zkclient、curator。由于 Curator 对于 zookeeper 的抽象层次比较高，简化了 zookeeper 客户端的开发量。使得 curator 逐步被广泛应用。

1. 封装 zookeeper client 与 zookeeper server 之间的连接处理
2. 提供了一套 fluent 风格的操作 api
3. 提供 zookeeper 各种应用场景 (共享锁、leader 选举) 的抽象封装

依赖 jar 包

```
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-framework</artifactId>
```

```
    <version>4.0.0</version>
</dependency>
```

建立连接

```
CuratorFramework
curatorFramework=CuratorFrameworkFactory.builder().

connectString(CONNECTION_STR).sessionTimeoutMs(5000)
.
        retryPolicy(new
ExponentialBackoffRetry(1000,3)).namespace(“curator
”).build();
```

重试策略：Curator 内部实现的几种重试策略:

- ExponentialBackoffRetry:重试指定的次数, 且每一次重试之间停顿的时间逐渐增加.
- RetryNTimes:指定最大重试次数的重试策略
- RetryOneTime:仅重试一次
- RetryUntilElapsed:一直重试直到达到规定的时间

namespace: 值得注意的是 session2 会话含有隔离命名空间，即客户端对 Zookeeper 上数据节点的任何操作都是相对/curator 目录进行的，这有利于实现不同的 Zookeeper 的业务之间的隔离

节点的增删改查

《参考源码》

节点权限设置

Zookeeper 作为一个分布式协调框架，内部存储了一些分布式系统运行时的状态的数据，比如 master 选举、比如分布式锁。对这些数据的操作会直接影响到分布式系统的运行状态。因此，为了保证 zookeeper 中的数据的安全性，避免误操作带来的影响。Zookeeper 提供了一套 ACL 权限控制机制来保证数据的安全。

权限控制的案例演示

- 给节点赋权

```
List<ACL> acls=new ArrayList<>();
Id id1=new Id("digest",
DigestAuthenticationProvider.generateDigest("u1:us
```

```
"));
Id id2=new Id("digest",
DigestAuthenticationProvider.generateDigest("u2:us
"));
acls.add(new ACL(ZooDefs.Perms.ALL,id1));//针对
u1，有 read 权限， 针对 u2 有读和删除权限
acls.add(new ACL(ZooDefs.Perms.DELETE |
ZooDefs.Perms.READ,id2));
curatorFramework.create().creatingParentsIfNeeded(
).withMode(CreateMode.PERSISTENT).

withACL(acls,false).forPath("/auth","sc".getBytes(
));
```

- 访问授权的节点
- AuthInfo authInfo=new

```
AuthInfo("digest","u1:us".getBytes());
List<AuthInfo> authInfos=new ArrayList<>();
authInfos.add(authInfo);
CuratorFramework curatorFramework=
CuratorFrameworkFactory.builder().
```

```
connectString("192.168.13.102:2181").sessionTimeou
tMs(5000).
        retryPolicy(new
ExponentialBackoffRetry(1000,3)).authorization(aut
hInfos).
        namespace("curator").build();
```

- 修改已经存在节点的权限

curatorFramework.setACL().withACL().forPath()

权限模式

Ip 通过 ip 地址粒度来进行权限控制，例如配置[ip:192.168.0.1]，或者按照网段 ip:192.168.0.1/24；

Digest：最常用的控制模式，类似于 username:password ；设置的时候需要 DigestAuthenticationProvider.generateDigest() SHA-加密和 base64 编码

World： 最开放的控制模式，这种权限控制几乎没有任何作用，数据的访问权限对所有用户开放。 world:anyone

Super： 超级用户，可以对节点做任何操作

授权对象

指权限赋予的用户或一个指定的实体，不同的权限模式下，授权对象不同

权限模式	授权对象
IP	通常是一个 IP 地址或是 IP 段，例如“192.168.0.110”或“192.168.0.1/24”
Digest	自定义，通常是“username:BASE64(SHA-1(username:password))”，例如“foo:kWN6aNSbjcKWPqjiV7cg0N24raU=”
World	只有一个 ID：“anyone”
Super	与 Digest 模式一致

Id ipId1 = new Id("ip", "192.168.190.1");

Id ANYONE_ID_UNSAFE = new Id("world", "anyone");

权限

指通过权限检查后可以被允许的操作， create /delete /read/write/admin

Create 允许对子节点 Create 操作

Read 允许对本节点 GetChildren 和 GetData 操作

Write 允许对本节点 SetData 操作

Delete 允许对子节点 Delete 操作

Admin 允许对本节点 setAcl 操作

权限模式：Schema 和授权对象：比如 ip 地址、username:passwrod

用来确定权限验证过程中使用的验证策略。

IP：通过 ip 地址来做权限控制，比如 ip:192.168.1.1 表示权限控制都是针对这个 ip 地址的。也可以针对网段 ip:192.168.1.1/24

Digest：最常用的权限控制模式，类似于 username:password 形式来进行权限控制；

World：开放的权限控制模式，数据节点的访问权限对所有用户开放，也可以看作是一种特殊的 Digest 模式 world:anyone

super：表示超级用户，可以对任意 zookeeper 上的数据节点进行操作

节点事件监听

Watcher 监听机制是 Zookeeper 中非常重要的特性，我们基于 zookeeper 上创建的节点，可以对这些节点绑定监听事件，

比如可以监听节点数据变更、节点删除、子节点状态变更等事件，通过这个事件机制，可以基于 zookeeper 实现分布式锁、集群管理等功能

zookeeper 事件	事件含义
EventType.NodeCreated	当 node-x 这个节点被创建时，该事件被触发
EventType.NodeChildrenChanged	当 node-x 这个节点的直接子节点被创建、被删除、
EventType.NodeDataChanged	当 node-x 这个节点的数据发生变更时，该事件被
EventType.NodeDeleted	当 node-x 这个节点被删除时，该事件被触发。
EventType.None	当 zookeeper 客户端的连接状态发生变更时，即 K KeeperState.SyncConnected、KeeperState.Autl

watcher 机制有一个特性：当数据发生改变的时候，那么 zookeeper 会产生一个 watch 事件并发送到客户端，但是客户端只会收到一次这样的通知，如果以后这个数据再发生变化，那么之前设置 watch 的客户端不会再次收到消息。因为他是一次性的；如果要实现永久监听，可以通过循环注册来实现

curator 对节点事件监听提供了很完善的 api，接下来简单演

示一下 curator 事件监听的基本使用

```
<dependency>
    <groupId>org.apache.curator</groupId>
    <artifactId>curator-recipes</artifactId>
    <version>4.0.0</version>
</dependency>
```

Curator 提供了三种 Watcher 来监听节点的变化

- PathChildCache：监视一个路径下孩子结点的创建、删除、更新。
- NodeCache：监视当前结点的创建、更新、删除，并将结点的数据缓存在本地。
- TreeCache：PathChildCache 和 NodeCache 的“合体”，监视路径下的创建、更新、删除事件，并缓存路径下所有孩子结点的数据。

对于 RPC 的改造

参考课程源码