

## 08 ZooKeeper常用命令+ Curator使用详解

## 08 ZooKeeper常用命令+ Curator使用详解

我们 [guide-rpc-framework <https://github.com/Snailclimb/guide-rpc-framework>](https://github.com/Snailclimb/guide-rpc-framework) 使用了 Zookeeper 来存储服务的相关信息，并且使用的是 ZooKeeper Java客户端 Curator 来对 ZooKeeper 进行增删改查等操作。

所以，本文就简单介绍一下 ZooKeeper常用命令 以及 Curator 的基本使用。

### 1. 前言

这篇文章简单给演示一下 ZooKeeper 常见命令的使用以及 ZooKeeper Java客户端 Curator 的基本使用。介绍到的内容都是最基本的操作，能满足日常工作的基本需要。

如果文章有任何需要改善和完善的地方，欢迎在评论区指出，共同进步！

## 2. ZooKeeper 安装和使用

### 2.1. 使用Docker 安装 zookeeper

#### a.使用 Docker 下载 ZooKeeper

```
1 docker pull zookeeper:3.5.8
```

Shell |  Copy

#### b.运行 ZooKeeper

```
1 docker run -d --name zookeeper -p 2181:2181 zookeeper:3.5.8
```

Shell |  Copy

### 2.2. 连接 ZooKeeper 服务

## a.进入ZooKeeper容器中

先使用 `docker ps` 查看 ZooKeeper 的 ContainerID, 然后使用 `docker exec -it ContainerID /bin/bash` 命令进入容器中。

b.先进入 bin 目录,然后通过 `./zkCli.sh -server 127.0.0.1:2181` 命令连接ZooKeeper 服务

```
1 root@eaf70fc620cb:/apache-zookeeper-3.5.8-bin# cd bin
```

Bash | Copy

如果你看到控制台成功打印出如下信息的话,说明你已经成功连接 ZooKeeper 服务。

```
root@eaf70fc620cb:/apache-zookeeper-3.5.8-bin# ./zkCli.sh -server 127.0.0.1:2181
Connecting to 127.0.0.1:2181
[2020-08-05 06:36:46,416 [myid:]] - INFO [main:Environment@109] - Client environment:zookeeper.version=3.5.8-f439ca583e70862c3068a1f2a7d40d068e33315, built on 05/04/2020 15:07 GMT
[2020-08-05 06:36:46,422 [myid:]] - INFO [main:Environment@109] - Client environment:host.name=eaf70fc620cb
[2020-08-05 06:36:46,422 [myid:]] - INFO [main:Environment@109] - Client environment:java.version=11.0.8
[2020-08-05 06:36:46,422 [myid:]] - INFO [main:Environment@109] - Client environment:java.vendor=N/A
[2020-08-05 06:36:46,428 [myid:]] - INFO [main:Environment@109] - Client environment:java.home=/usr/local/openjdk-11
[2020-08-05 06:36:46,428 [myid:]] - INFO [main:Environment@109] - Client environment:java.class.path=/apache-zookeeper-3.5.8-bin/bin/./zookeeper-server/target/classes:/apache-zookeeper-3.5.8-bin/bin/./build/classes:/apache-zookeeper-3.5.8-bin/bin/./zookeeper-server/target/lib/*:/apache-zookeeper-3.5.8-bin/bin/./build/lib/*:/apache-zookeeper-3.5.8-bin/bin/./lib/zookeeper-jute-3.5.8.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/zookeeper-3.5.8.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/slf4j-log4j12-1.7.25.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/slf4j-api-1.7.25.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-transport-native-unix-common-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-transport-native-epoll-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-transport-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-resolver-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-handler-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-common-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-codec-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/netty-buffer-4.1.48.Final.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/log4j-1.2.17.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/json-simple-1.1.1.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jline-2.11.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jetty-util-9.4.24.v20191120.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jetty-server-9.4.24.v20191120.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jetty-security-9.4.24.v20191120.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jetty-io-9.4.24.v20191120.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jetty-http-9.4.24.v20191120.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/javax.servlet-api-3.1.0.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jackson-databind-2.10.3.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jackson-core-2.10.3.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/jackson-annotations-2.10.3.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/commons-cli-1.2.jar:/apache-zookeeper-3.5.8-bin/bin/./lib/audience-annotations-0.5.0.jar:/apache-zookeeper-3.5.8-bin/bin/./zookeeper-*.jar:/apache-zookeeper-3.5.8-bin/bin/./zookeeper-server/src/main/resources/lib/*:/conf:
[2020-08-05 06:36:46,429 [myid:]] - INFO [main:Environment@109] - Client environment:java.library.path=/usr/java/packages/lib:/usr/lib64:/lib64:/lib:/usr/lib
[2020-08-05 06:36:46,429 [myid:]] - INFO [main:Environment@109] - Client environment:java.io.tmpdir=/tmp
[2020-08-05 06:36:46,429 [myid:]] - INFO [main:Environment@109] - Client environment:java.compiler=<NA>
[2020-08-05 06:36:46,430 [myid:]] - INFO [main:Environment@109] - Client environment:os.name=Linux
[2020-08-05 06:36:46,430 [myid:]] - INFO [main:Environment@109] - Client environment:os.arch=amd64
[2020-08-05 06:36:46,430 [myid:]] - INFO [main:Environment@109] - Client environment:os.version=4.19.76-linuxkit
[2020-08-05 06:36:46,431 [myid:]] - INFO [main:Environment@109] - Client environment:user.name=root
[2020-08-05 06:36:46,431 [myid:]] - INFO [main:Environment@109] - Client environment:user.home=/root
[2020-08-05 06:36:46,431 [myid:]] - INFO [main:Environment@109] - Client environment:user.dir=/apache-zookeeper-3.5.8-bin/bin
[2020-08-05 06:36:46,432 [myid:]] - INFO [main:Environment@109] - Client environment:os.memory.free=25MB
[2020-08-05 06:36:46,438 [myid:]] - INFO [main:Environment@109] - Client environment:os.memory.max=256MB
[2020-08-05 06:36:46,439 [myid:]] - INFO [main:Environment@109] - Client environment:os.memory.total=32MB
[2020-08-05 06:36:46,444 [myid:]] - INFO [main:ZooKeeper@868] - Initiating client connection, connectString=127.0.0.1:2181 sessionTimeout=30000 watcher=org.apache.zookeeper.ZooKeeperMain$MyWatcher@46238e3f
[2020-08-05 06:36:46,455 [myid:]] - INFO [main:X509Util@79] - Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation
[2020-08-05 06:36:46,473 [myid:]] - INFO [main:ClientCnxnSocket@237] - jute.maxbuffer value is 4194304 Bytes
[2020-08-05 06:36:46,486 [myid:]] - INFO [main:ClientCnxn@1653] - zookeeper.request.timeout value is 0. feature enabled=
Welcome to zookeeper!
[2020-08-05 06:36:46,524 [myid:127.0.0.1:2181]] - INFO [main:SendThread(127.0.0.1:2181):ClientCnxn$SendThread@1112] - Opening socket connection to server localhost/127.0.0.1:2181. Will not attempt to authenticate using SASL (unknown error)
```

## 2.3. 常用命令演示

### 2.3.1. 查看常用命令(help 命令)

通过 `help` 命令查看 ZooKeeper 常用命令

### 2.3.2. 创建节点(create 命令)

通过 `create` 命令在根目录创建了 node1 节点, 与它关联的字符串是"node1"

```
1 [zk: 127.0.0.1:2181(CONNECTED) 34] create /node1 "node1"
```

Shell |  Copy

通过 `create` 命令在根目录创建了 node1 节点，与它关联的内容是数字 123

```
1 [zk: 127.0.0.1:2181(CONNECTED) 1] create /node1/node1.1 123
2 Created /node1/node1.1
```

Shell |  Copy

### 2.3.3. 更新节点数据内容(set 命令)

```
1 [zk: 127.0.0.1:2181(CONNECTED) 11] set /node1 "set node1"
```

Shell |  Copy

### 2.3.4. 获取节点的数据(get 命令)

`get` 命令可以获取指定节点的数据内容和节点的状态,可以看出我们通过 `set` 命令已经将节点数据内容改为 "set node1"。

```
1 set node1
2 cZxid = 0x47
3 ctime = Sun Jan 20 10:22:59 CST 2019
4 mZxid = 0x4b
5 mtime = Sun Jan 20 10:41:10 CST 2019
6 pZxid = 0x4a
7 cversion = 1
8 dataVersion = 1
9 aclVersion = 0
10 ephemeralOwner = 0x0
11 dataLength = 9
12 numChildren = 1
```

Shell |  Copy

### 2.3.5. 查看某个目录下的子节点(ls 命令)

通过 `ls` 命令查看根目录下的节点

```
1 [zk: 127.0.0.1:2181(CONNECTED) 37] ls /
2 [dubbo, ZooKeeper, node1]
```

通过 `ls` 命令查看 node1 目录下的节点

```
1 [zk: 127.0.0.1:2181(CONNECTED) 5] ls /node1
2 [node1.1]
```

ZooKeeper 中的 ls 命令和 linux 命令中的 ls 类似，这个命令将列出绝对路径 path 下的所有子节点信息（列出 1 级，并不递归）

### 2.3.6. 查看节点状态(stat 命令)

通过 `stat` 命令查看节点状态

```
1 [zk: 127.0.0.1:2181(CONNECTED) 10] stat /node1
2 cZxid = 0x47
3 ctime = Sun Jan 20 10:22:59 CST 2019
4 mZxid = 0x47
5 mtime = Sun Jan 20 10:22:59 CST 2019
6 pZxid = 0x4a
7 cversion = 1
8 dataVersion = 0
9 aclVersion = 0
10 ephemeralOwner = 0x0
11 dataLength = 11
12 numChildren = 1
```

上面显示的一些信息比如 cversion、aclVersion、numChildren 等等，我在上面“znode(数据节点)的结构”这部分已经介绍到。

### 2.3.7. 查看节点信息和状态(ls2 命令)

`ls2` 命令更像是 `ls` 命令和 `stat` 命令的结合。`ls2` 命令返回的信息包括 2 部分：

1. 子节点列表
2. 当前节点的 stat 信息。

Shell |  Copy

```
1 [zk: 127.0.0.1:2181(CONNECTED) 7] ls2 /node1
2 [node1.1]
3 cZxid = 0x47
4 ctime = Sun Jan 20 10:22:59 CST 2019
5 mZxid = 0x47
6 mtime = Sun Jan 20 10:22:59 CST 2019
7 pZxid = 0x4a
8 cversion = 1
9 dataVersion = 0
10 aclVersion = 0
11 ephemeralOwner = 0x0
12 dataLength = 11
13 numChildren = 1
```

### 2.3.8. 删除节点(delete 命令)

这个命令很简单，但是需要注意的一点是如果你要删除某一个节点，那么这个节点必须无子节点才行。

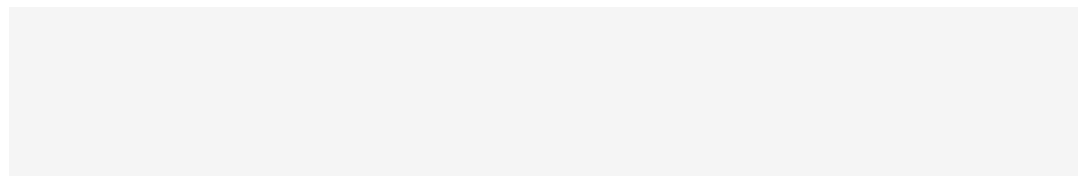
Shell |  Copy

```
1 [zk: 127.0.0.1:2181(CONNECTED) 3] delete /node1/node1.1
```

在后面我会介绍到 Java 客户端 API 的使用以及开源 ZooKeeper 客户端 ZkClient 和 Curator 的使用。

## 3. ZooKeeper Java客户端 Curator简单使用

Curator 是Netflix公司开源的一套 ZooKeeper Java客户端框架，相比于 Zookeeper 自带的客户端 zookeeper 来说，Curator 的封装更加完善，各种 API 都可以比较方便地使用。



下面我们就来简单地演示一下 Curator 的使用吧！

Curator4.0+版本对ZooKeeper 3.5.x支持比较好。开始之前，请先将下面的依赖添加进你的项目。

XML |  Copy

```
1 <dependency>
2     <groupId>org.apache.curator</groupId>
3     <artifactId>curator-framework</artifactId>
4     <version>4.2.0</version>
5 </dependency>
6 <dependency>
7     <groupId>org.apache.curator</groupId>
8     <artifactId>curator-recipes</artifactId>
9     <version>4.2.0</version>
10 </dependency>
```

### 3.1. 连接 ZooKeeper 客户端

通过 `CuratorFrameworkFactory` 创建 `CuratorFramework` 对象，然后再调用 `CuratorFramework` 对象的 `start()` 方法即可！

Java |  Copy

```
1 private static final int BASE_SLEEP_TIME = 1000;
2 private static final int MAX_RETRIES = 3;
3
4 // Retry strategy. Retry 3 times, and will increase the sleep time between
  retries.
5 RetryPolicy retryPolicy = new ExponentialBackoffRetry(BASE_SLEEP_TIME,
  MAX_RETRIES);
6 CuratorFramework zkClient = CuratorFrameworkFactory.builder()
7     // the server to connect to (can be a server list)
8     .connectString("127.0.0.1:2181")
9     .retryPolicy(retryPolicy)
10    .build();
11 zkClient.start();
```

对于一些基本参数的说明：

- `baseSleepTimeMs`：重试之间等待的初始时间
- `maxRetries`：最大重试次数
- `connectString`：要连接的服务器列表
- `retryPolicy`：重试策略

### 3.2. 数据节点的增删改查

### 3.2.1. 创建节点

我们在 [ZooKeeper常见概念解读 <https://snailclimb.gitee.io/javaguide/#/docs/system-design/distributed-system/zookeeper/zookeeper-intro>](https://snailclimb.gitee.io/javaguide/#/docs/system-design/distributed-system/zookeeper/zookeeper-intro) 中介绍到，我们通常是将 znode 分为 4 大类：

- **持久 (PERSISTENT) 节点**：一旦创建就一直存在即使 ZooKeeper 集群宕机，直到将其删除。
- **临时 (EPHEMERAL) 节点**：临时节点的生命周期是与 **客户端会话 (session)** 绑定的，会话消失则节点消失。并且，临时节点 **只能做叶子节点**，不能创建子节点。
- **持久顺序 (PERSISTENT\_SEQUENTIAL) 节点**：除了具有持久 (PERSISTENT) 节点的特性之外，子节点的名称还具有顺序性。比如 `/node1/app0000000001`、`/node1/app0000000002`。
- **临时顺序 (EPHEMERAL\_SEQUENTIAL) 节点**：除了具备临时 (EPHEMERAL) 节点的特性之外，子节点的名称还具有顺序性。

你在使用的 ZooKeeper 的时候，会发现 `CreateMode` 类中实际有 7 种 znode 类型，但是用的最多的还是上面介绍的 4 种。

#### a. 创建持久化节点

你可以通过下面两种方式创建持久化的节点。

```
1 //注意:下面的代码会报错,下文说了具体原因
2 zkClient.create().forPath("/node1/00001");
3 zkClient.create().withMode(CreateMode.PERSISTENT).forPath("/node1/00002");
```

但是，你运行上面的代码会报错，这是因为的父节点 `node1` 还未创建。

你可以先创建父节点 `node1`，然后再执行上面的代码就不会报错了。

```
1 zkClient.create().forPath("/node1");
```

更推荐的方式是通过下面这行代码，`creatingParentsIfNeeded()` 可以保证父节点不存在的时候自动创建父节点，这是非常有用的。

Java |  Copy

```
1 zkClient.create().creatingParentsIfNeeded().withMode(CreateMode.PERSISTENT
```

### b.创建临时节点

Java |  Copy

```
1 zkClient.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL)
```

### c.创建节点并指定数据内容

Java |  Copy

```
1 zkClient.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL)
2 zkClient.getData().forPath("/node1/00001");//获取节点的数据内容，获取到的是 byte
```

### d.检测节点是否创建成功

Java |  Copy

```
1 zkClient.checkExists().forPath("/node1/00001");//不为null的话，说明节点创建成功
```

## 3.2.2. 删除节点

### a.删除一个子节点

Java |  Copy

```
1 zkClient.delete().forPath("/node1/00001");
```

### b.删除一个节点以及其下的所有子节点

Java |  Copy

```
1 zkClient.delete().deletingChildrenIfNeeded().forPath("/node1");
```

## 3.2.3. 获取/更新节点数据内容



Java |  Copy

```
1 zkClient.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL)
2 zkClient.getData().forPath("/node1/00001");//获取节点的数据内容
3 zkClient.setData().forPath("/node1/00001","c++".getBytes());//更新节点数据内容
```

#### 3.2.4. 获取某个节点的所有子节点路径

Java |  Copy

```
1 List<String> childrenPaths = zkClient.getChildren().forPath("/node1");
```

### 3.3 监听器

下面简单演示一下如何给某个节点注册子节点监听器。注册了监听器之后，这个节点的子节点发生变化比如增加、减少或者更新的时候，你可以自定义回调操作。

Java |  Copy

```
1 String path = "/node1";
2 PathChildrenCache pathChildrenCache = new PathChildrenCache(zkClient,
3     path, true);
4 PathChildrenCacheListener pathChildrenCacheListener = (curatorFramework,
5     pathChildrenCacheEvent) -> {
6     // do something
7 };
8 pathChildrenCache.getListenable().addListener(pathChildrenCacheListener);
9 pathChildrenCache.start();
```

如果你要获取节点事件类型的话，可以通过：

Java |  Copy

```
1 pathChildrenCacheEvent.getType()
```

一共有下面几种类型：

```
1      public static enum Type {
2          CHILD_ADDED, //子节点增加
3          CHILD_UPDATED, //子节点更新
4          CHILD_REMOVED, //子节点被删除
5          CONNECTION_SUSPENDED,
6          CONNECTION_RECONNECTED,
7          CONNECTION_LOST,
8          INITIALIZED;
9
10     private Type() {
11     }
12 }
```