

大规模信息系统构建技术导论

分布式MiniSQL系统模块实现与测试报告

|  |  |
| --- | --- |
| 姓名： | 庄毅非 |
| 学院： | 计算机科学与技术学院 |
| 系： | 软件工程 |
| 专业： | 软件工程 |
| 学号： | 3200105872 |

2023 年 5 月 7 日

# 一．系统模块简介

本人在分布式MiniSQL系统中负责研发Master、Region模块的开发，采用Java程序设计语言，在Ubuntu 20.04 LTS平台下编辑、编译与调试，通过了所有测试。

Master模块的具体功能如下：

（1）对 Zookeeper 集群注册自己的地址，方便调试

（2）监听 lss节点下的所有子节点的状态，注册对应的回调函数执行对应业务逻辑

（3）当新表创建的时候，在所有节点中随机选择两个节点创建从表，通知对应的主 Region 节点进行主从同步

（4）当主节点的表被删除的时候，通知从表执行相同的删除表操作

（5）当从节点下线的时候，选择另一个非从节点以及非主节点的节点作为新的从节点，保证任何时候都有两个从节点

（6）当主节点宕机的时候，选择一个从节点切换为主节点，然后选择另一个相对该表空闲的节点作为新的从节点，保障数据始终可用。

Region模块的具体功能如下：

1. 启动的时候，连接本地数据库，然后将节点信息注册到 zookeeper 的 lss 父节点下，保障 master 和 client 能够执行正确的逻辑。
2. 在出现表的创建的时候，负责执行创建从节点的具体逻辑。
3. 在出现表的修改的时候（比如插入，删除，更新等），将对应的修改行为同步到从节点上。
4. 在主节点宕机的时候，能够接受 Master 的信号，切换为主节点。
5. 定时和 Zookeeper 集群通信，保障本地缓存有效。

具体设计逻辑详见后续设计报告。

# 二．Master模块实现说明

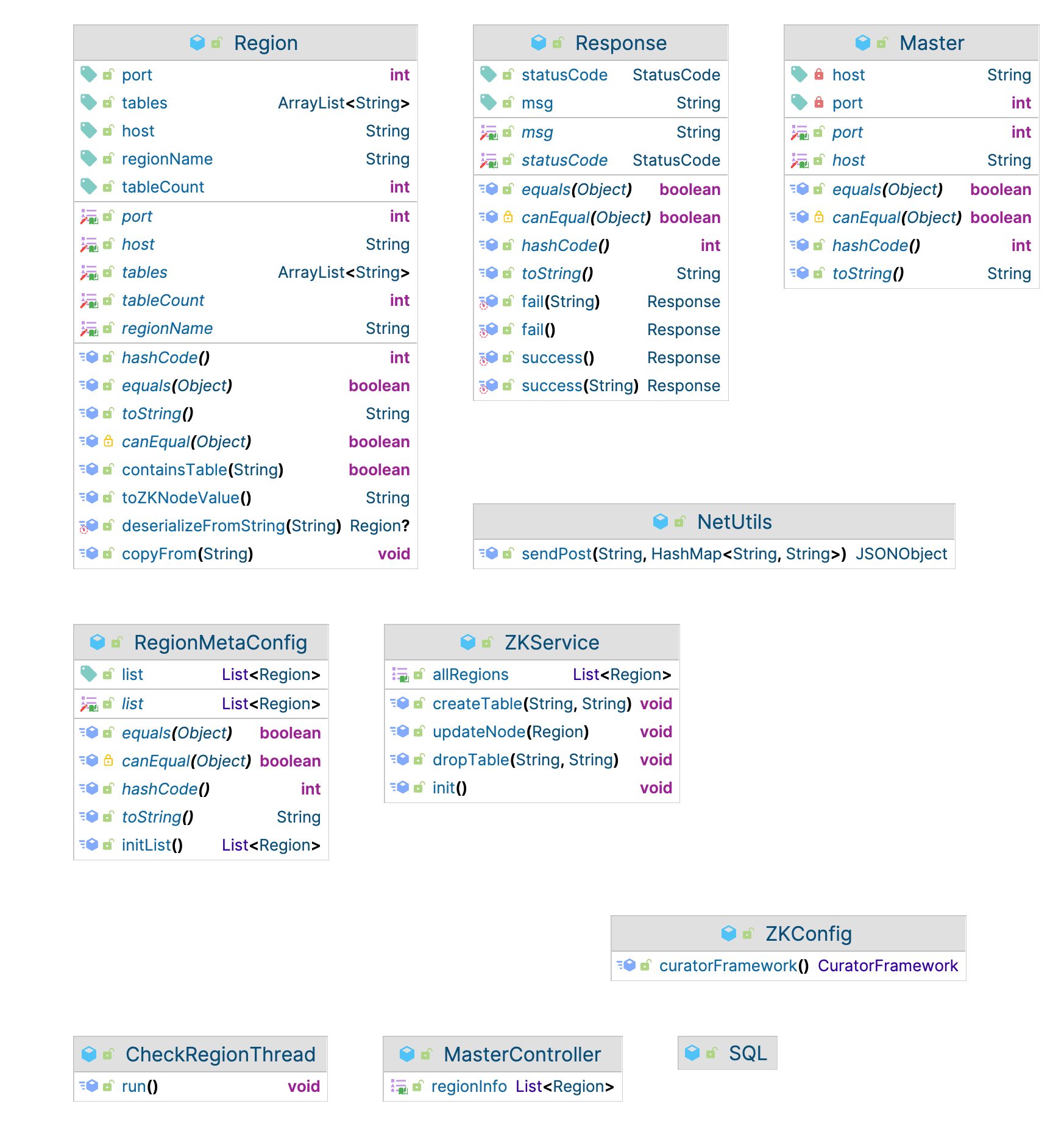
## 2.1 模块组件设计

该类下的主要功能类是ZKService、CheckRegionThread、ZKConfig和 MasterController 四个。

* ZKService 类主要负责使用curator 框架执行对应的操作，对外提供一个能够获取 zk 节点状态的接口，方便调试使用，除此之外，ZKService 类也会使用curatorCache类注册增加、删除、更新节点和表所对应的回调函数，方便执行表间同步逻辑以及节点切换逻辑。
* CheckRegionThread 类主要负责定时和 Zookeeper 集群通信，尽可能保障本地缓存的一致性。
* ZKConfig 主要负责在 Master 启动的时候对 Zookeeper 集群注册Master 的信息，方便 Client 使用。
* MasterController 主要负责接受 Client 的请求，返回对应的结果。

## 2.2 主要数据结构

### 2.2.1 Master类图



图一 Master 相关类图

Master 服务器使用 SSM 框架开发，该类下的主要功能类是ZKService、CheckRegionThread、ZKConfig 和 MasterController 四个。

* ZKService 类主要负责使用curator 框架执行对应的操作，对外提供一个能够获取 zk 节点状态的接口，方便调试使用，除此之外，ZKService 类也会使用curatorCache类注册增加、删除、更新节点和表所对应的回调函数，方便执行表间同步逻辑以及节点切换逻辑。
* CheckRegionThread 类主要负责定时和 Zookeeper 集群通信，尽可能保障本地缓存的一致性。
* ZKConfig 主要负责在 Master 启动的时候对 Zookeeper 集群注册Master 的信息，方便 Client 使用。
* MasterController 主要负责接受 Client 的请求，返回对应的结果。

### 2.2.2 RegionMetaDataConfig 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| initList | List<Region> | 初始化一个 list，供项目中其他模块使用 |

### 2.2.3 ZKConfig类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| curatorFramework | CuratorFrame | 初始化 curatorFrameworll类，和 Zookeeper集群通信，注册自身信息。 |

### 2.2.4 MasterController 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| getRegionInfgo | List<Region> | 获取所有 Region 的信息，方便调试 |

### 2.2.5 ZKService类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| ZKService |  | 构造函数 |
| Init |  | 执行初始化逻辑，对 Zookeeper 集群注册信息 |
| getAllRegions | List<Region> | 对外提供获取所有 Region 信息的接口 |
| sendPost | JSONObject | 发送 Post 请求 |
| refreshRegion |  | 刷新本地 Region 缓存列表 |
| getRegionNotHaveTable | List<Region> | 获取没有对应表的region 的列表，执行从节点创建逻辑 |
| registerCallback |  | 注册对应的回调函数，执行必要的节点间同步逻辑。 |
| getRegion | Region | 通过指定的region 的名称，获取对应的 region 对象 |
| getRegionByPath | Region | 使用执行的 Region 的路径，获取对应的 Region |
| writeRegion |  | 更新 Zookeeper 集群上的region的信息。 |

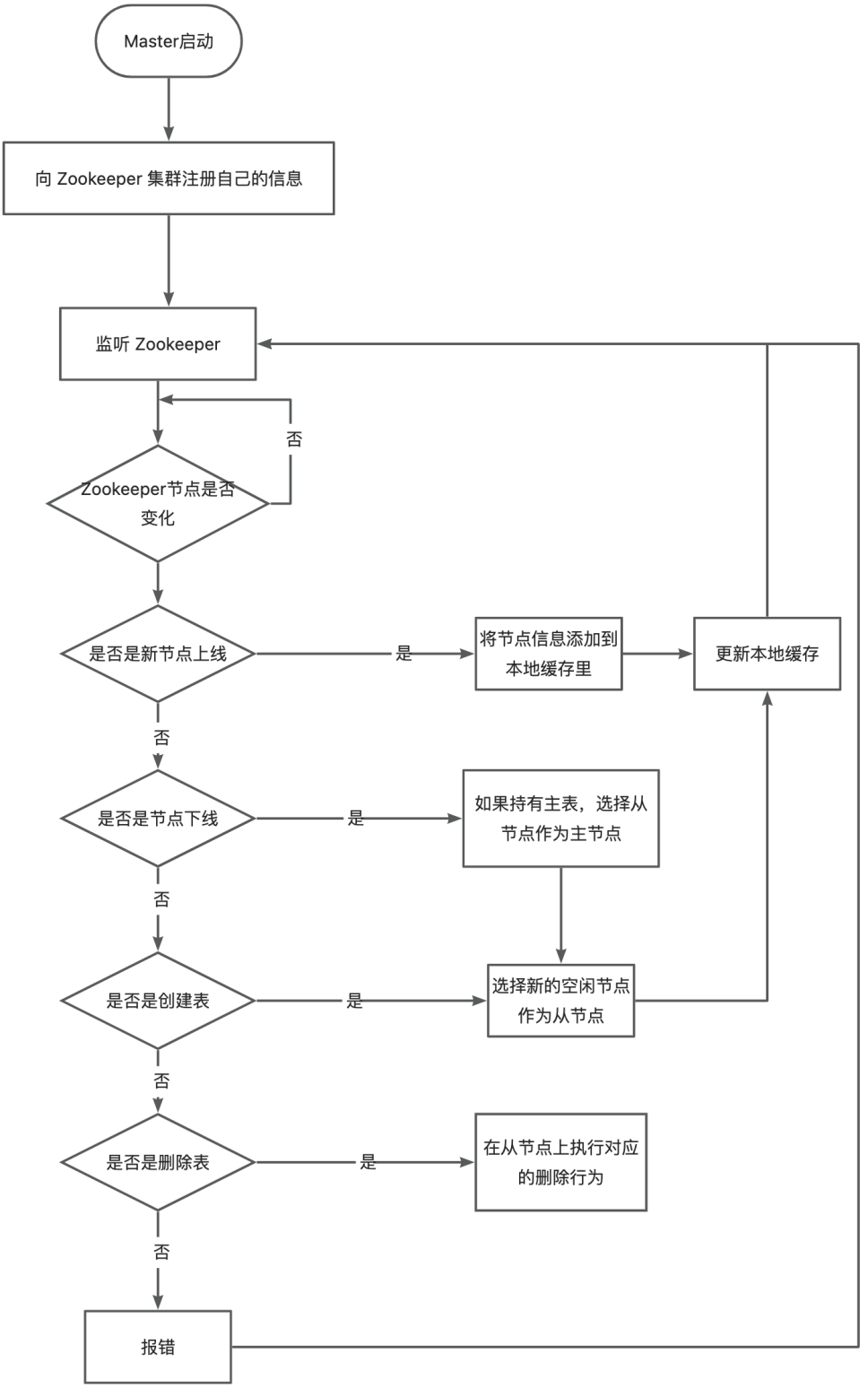
### 2.2.6 CheckRegionThread 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| Run |  | 定时任务，负责刷新 region 缓存 |

### 2.2.7 NetUtils 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| sendPost | JSONObject | 发送Post 请求 |

## **2.3 流程图设计**



# 三．Region模块实现说明

## 3.1 模块组件设计

## 3.2 主要数据结构

### 3.2.1 Region 类图

### Region

Region使用 SSM 框架开发，该类下的主要功能类是ZKConfiguration、SqlController、SqlService、ZKService、RefreshRegion 五个类

* ZKService类主要负责使用curator 框架执行对应的操作，比如获取当前 region 信息，更新 region 信息，获取所有region 的信息等。
* CheckRegionThread 类主要负责定时和 Zookeeper 集群通信，尽可能保障本地缓存的一致性。
* SqlService主要负责执行具体的 sql 函数，包括表的创建和删除，行的插入、删除、更新和查询等。
* ZKConfiguration主要负责在 Master 启动的时候对 Zookeeper 集群注册Region的信息，方便 Client和 Master使用。
* SqlController主要负责接受 Client 和 Master的请求，返回对应的结果。

### 3.2.2 ZKConfiguration 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| curatorFramework | curatorFramework | 注册节点信息，返回curatorFramework对象 |
| getRegion |  | 构造全局 Region对象 |

### 3.2.3 SqlController 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| execSQL | MyResponse | 执行对应的 SQL，返回结果 |
| Dump |  | 执行对应的同步逻辑，将当前节点作为主节点的从节点 |
| import |  | 和 Dump 类似，区别在于使用本地 sql 文件执行同步逻辑。 |

### 3.3.4 SqlService 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| initMethod |  | 弃用 |
| init |  | 初始化方法，将本地数据表清空，使当前节点成为一个空节点 |
| getConnectionWithoutDataBase | Connection | 通过配置文件中的数据库元信息，使用 JDBC 建立对整个 Mysql 的链接 |
| getConnection | Connection | 通过配置文件中的数据库元信息，使用 JDBC 建立对单个 Mysql 数据库的链接 |
| getConnectionByInfos | Connection | 通过输入的JDBCURL，用户名和密码，建立对应的链接 |
| execUpdateOrInsertOrDeleteSqls | int | 执行更新、插入和删除行的SQL，在本地执行完毕之后，如果当前节点是主节点，那么会根据 zookeeper 中存储的从节点信息，将对应的 sql 同步给其他节点。 |
| execSelectSQL | String | 执行查询 SQL |
| execCreateAndDropSQL | Boolean | 执行创建或者删除表的 SQL，如果是创建表，那么会在本地创建成功之后，更新 Zookeeper 中的信息，以便 Master 创建从节点；如果是删除表，那么会在本地删除成功之后，更新 Zookeeper 集群中存储的信息，方便 Master 执行对应的同步逻辑。 |

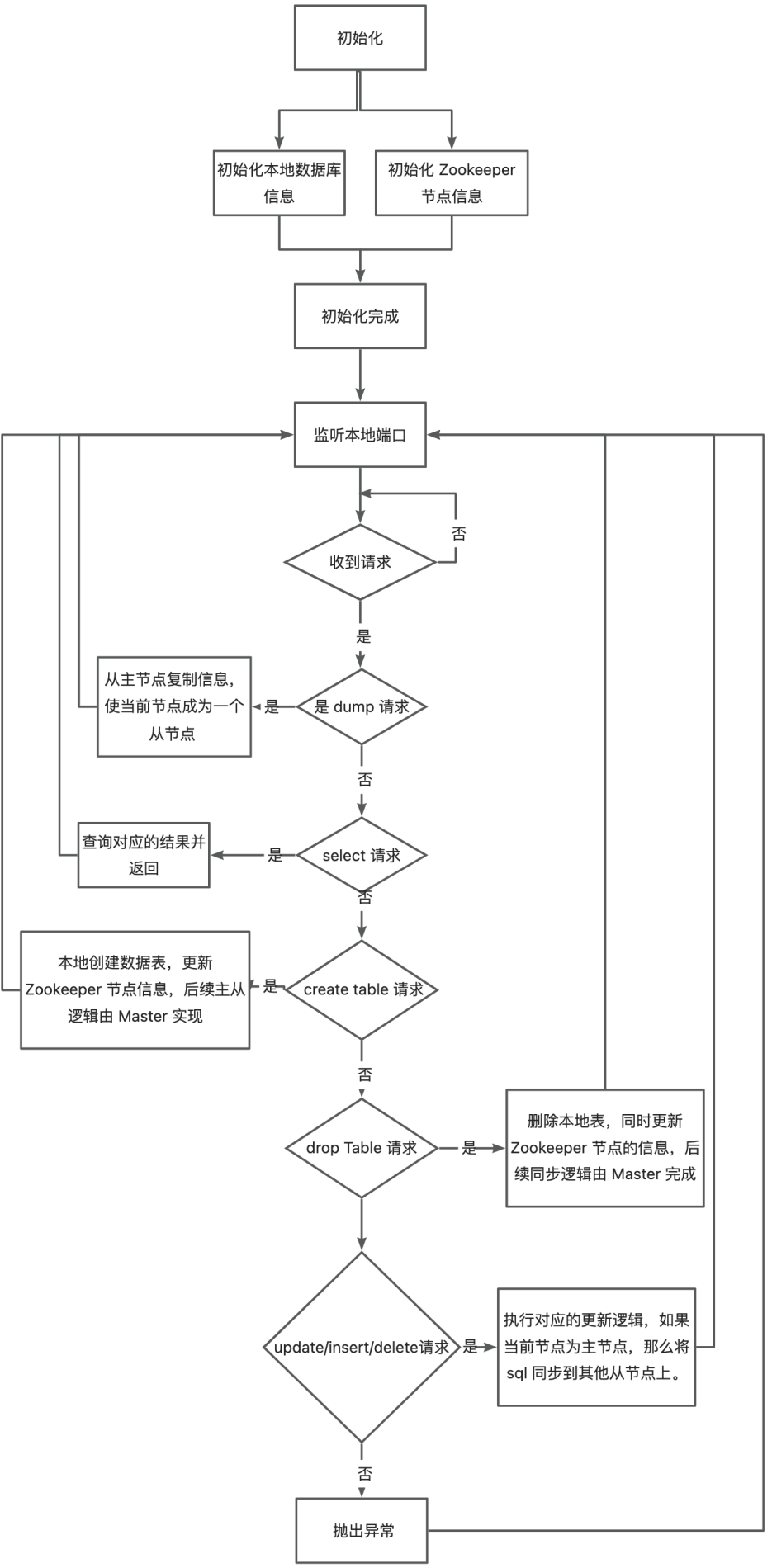
### 3.3.5 ZKService 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| Init |  | 执行初始化逻辑，向 Zookeeper 集群注册自己的信息 |
| getAllRegions | List<Region> | 获取所有 Region 的信息 |
| updateNode |  | 更新 Zookeeper 集群中对应的节点的信息 |
| createTable |  | 在 Zookeeper 集群中，将创建的表的信息加到节点上，方便 Client 和 Master查询 |
| dropTable |  | 在 Zookeeper 集群中，将删除的表的信息从节点上移除 |

### 3.3.6 RefreshRegion 类

|  |  |  |
| --- | --- | --- |
| 函数名 | 返回类型 | 功能说明 |
| Run |  | 定时任务，刷新本地缓存 |

## 3.3 流程图设计



# 重要功能实现

## 4.1 副本管理

本项目对容灾容错的实现如下：在创建新的数据表的时候，将当前创建数据表的节点作为主节点，该节点会更新自身在 Zookeeper 集群中的信息，此更新行为会被 master 监听到，并启动对应的回调函数。在该回调函数中，master 会从当前的所有节点中随机选择两个节点作为从节点，在这两个从节点上创建对应表的副本，之后发送请求让这两个节点执行对应的同步操作，这样一张表就拥有了一个主表和两个从表。

在任何修改数据表的sql 发送到主节点的时候，主节点会处理这个 sql，然后根据 Zookeeper 上的信息，将sql 转发给对应的从节点，从而实现节点之间的数据同步。

在删除表的时候，主节点会执行这个删除动作，然后将自己在 Zookeeper 集群中的节点信息更新，master 会监听到这个更新请求，然后将删除的 sql 同步到对应的从节点上，从而实现同步。

## 4.2 容灾容错

本项目对容灾容错的实现如下：在节点下线的时候，如果其拥有某个数据表的主表，那么 master 会从其从节点中选择一个晋升为主节点，然后为了保障任何数据表都有三个节点存储的要求，master 会随机从剩下的未持有这个数据表的节点中选择一个节点作为新的从节点，这样就实现了主从切换和保证从节点至少有两个（如果没有，那么系统不能正常运行）。

## 4.3 节点管理

本项目对节点的管理如下：在新的节点加入 Zookeeper 集群的时候，master 会监听到这个请求，然后将这个节点添加到自己维护的一个本地缓存里；在节点退出集群的时候，那么 master 会执行上述的容灾容错逻辑，然后将对应的节点的信息从自己的本地缓存中删除；如果是节点信息的更新，这一般是新表创建或者旧表删除，那么 master 也会执行对应的同步逻辑。

# 模块测试

## 5.1 Master测试

Master 模块目录结构如下。

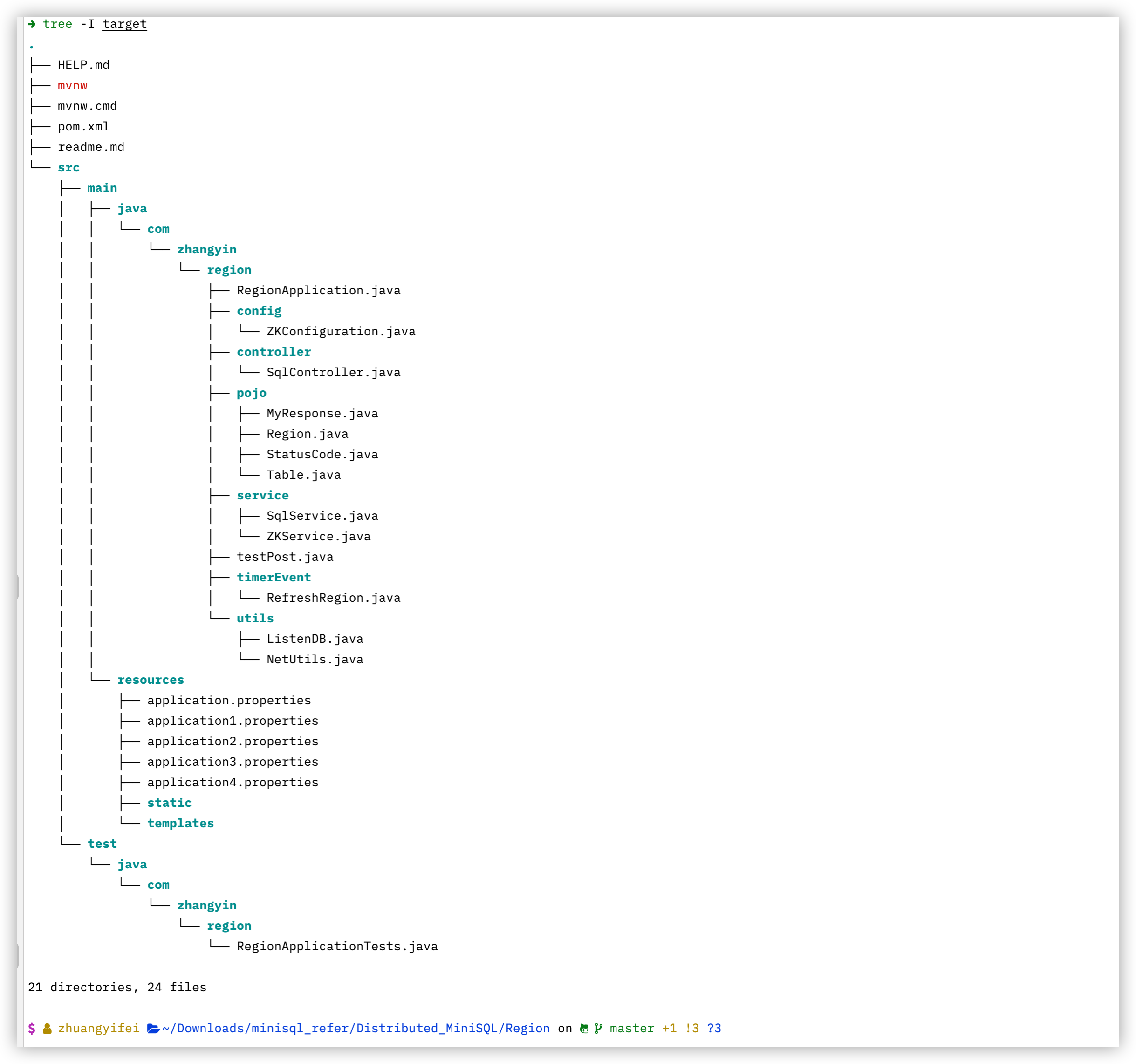


启动成功截图如下。

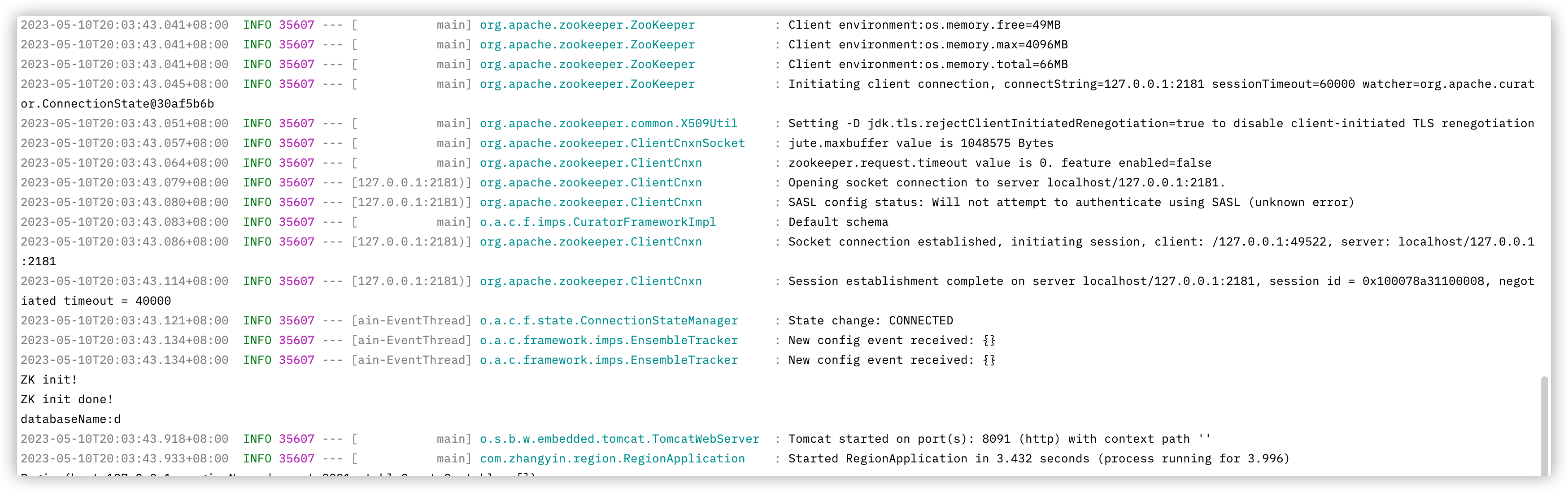


## 5.2 Region 测试

Region目录如下。



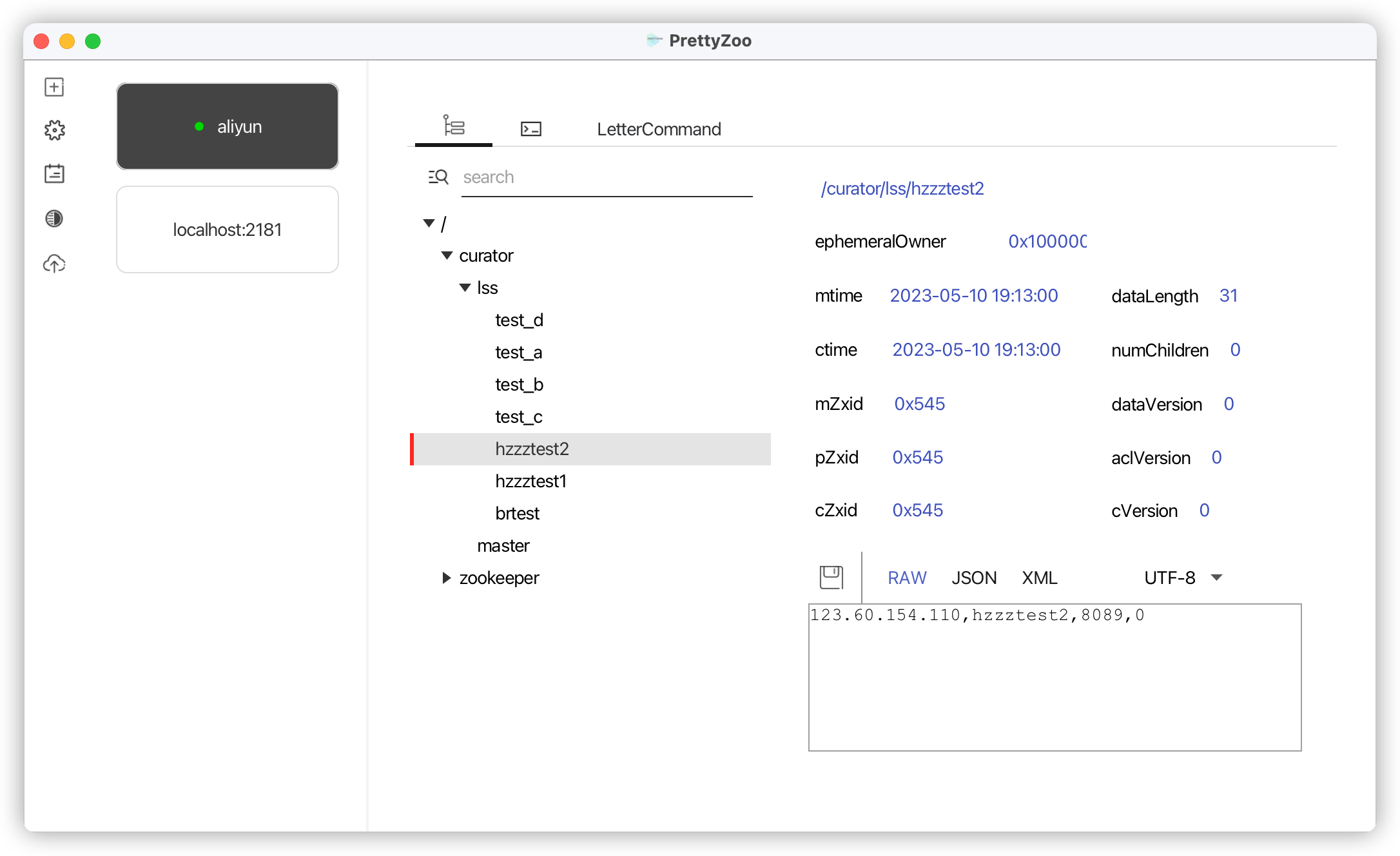
Region 启动成功截图如下。

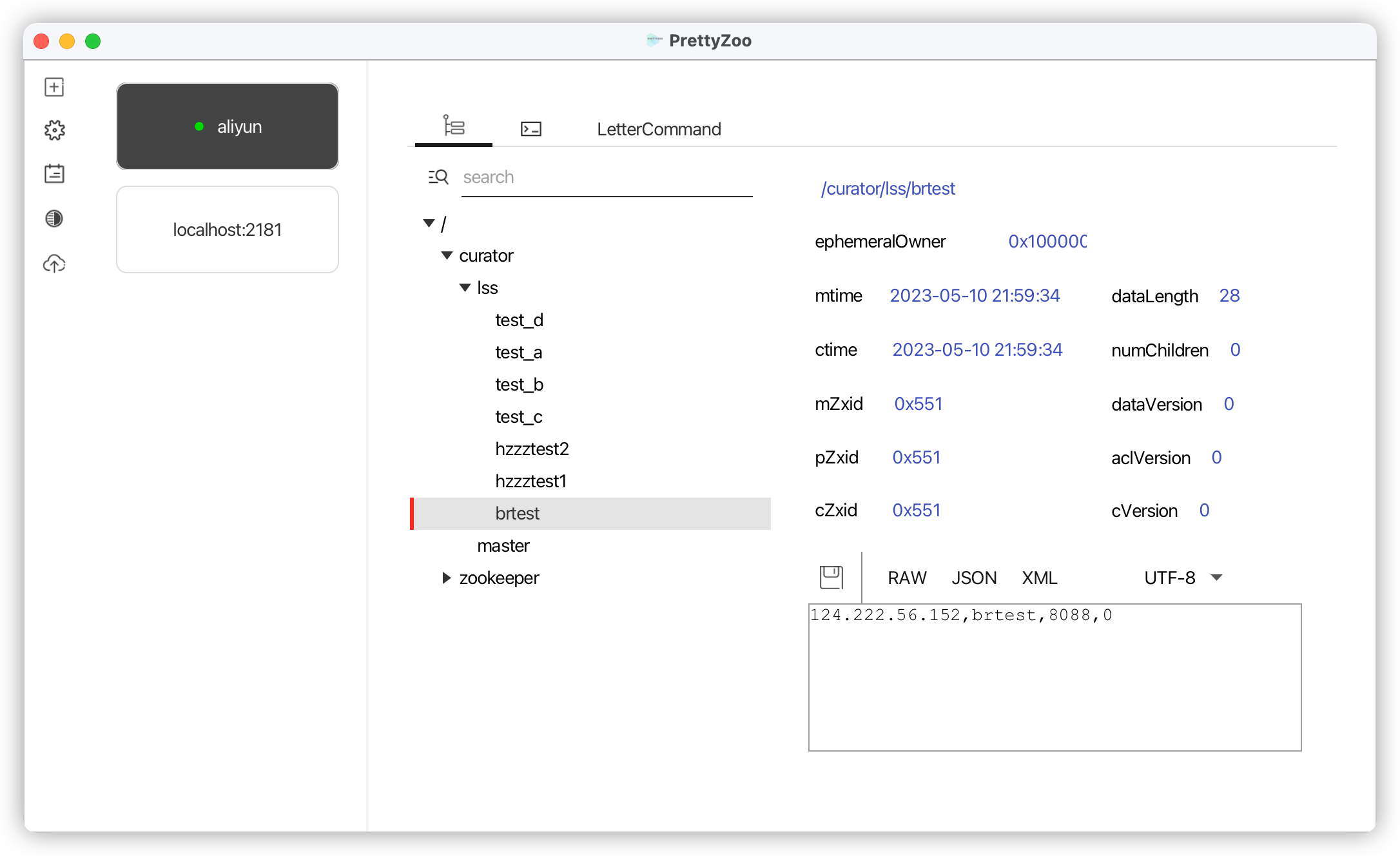


## 5.3 创建表

这里启动了七个实例作为 region Server，ip 分别是120.26.195.57，123.60.154.110和124.222.56.152。

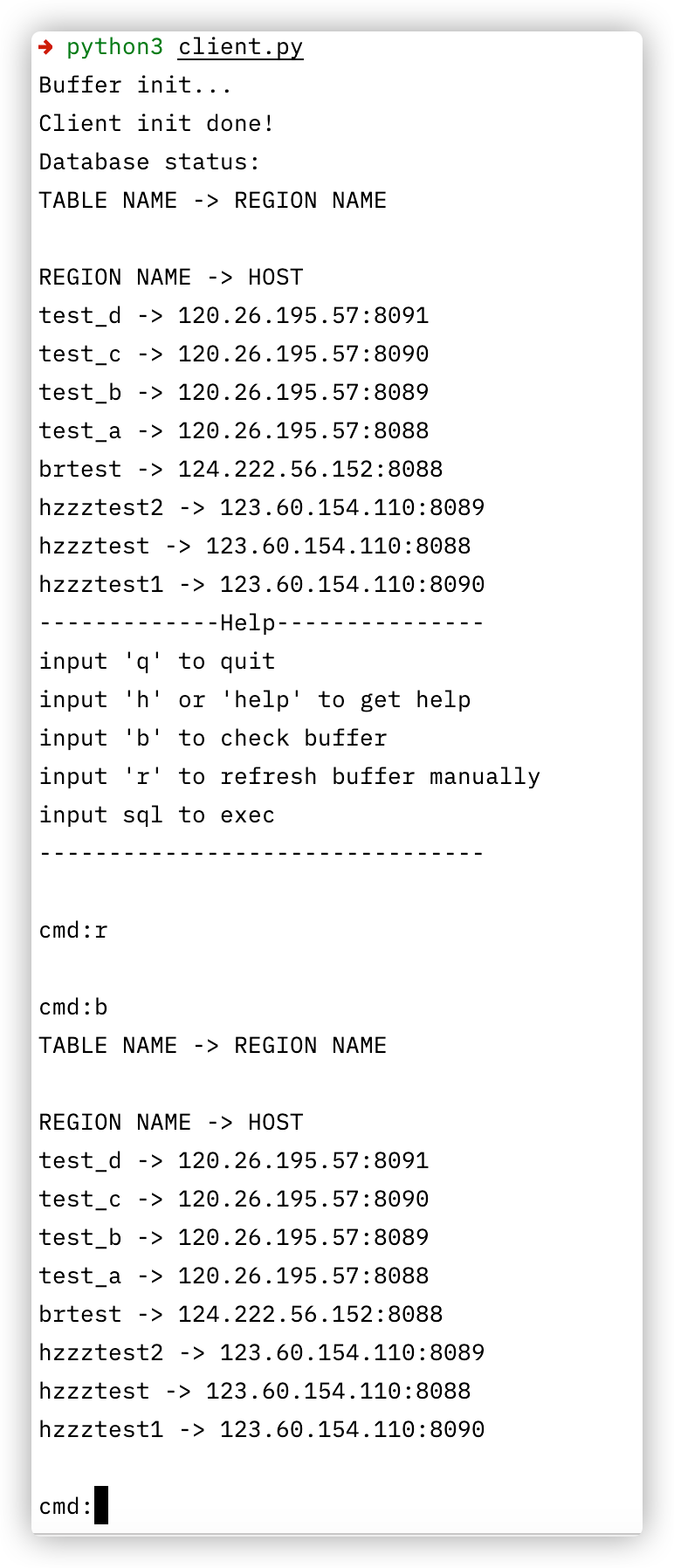




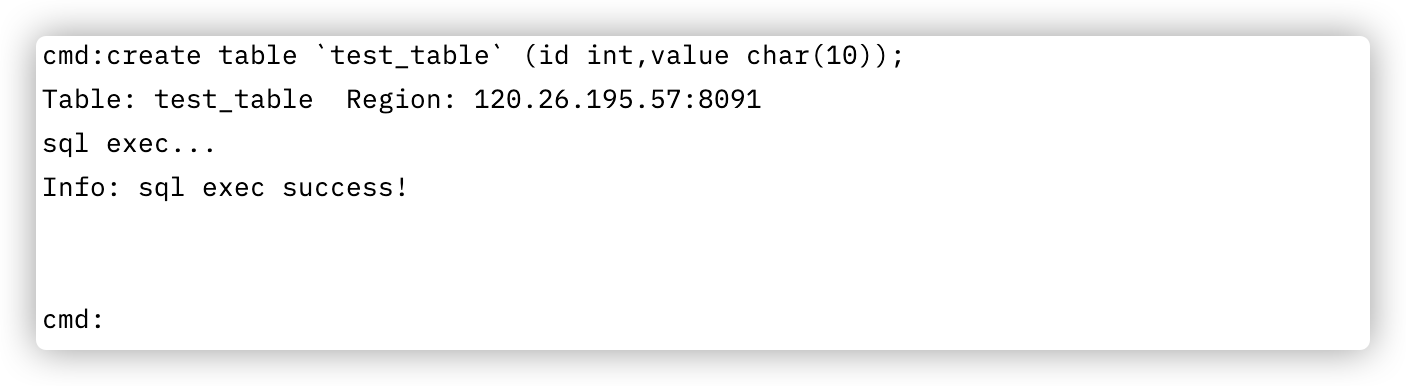


图中每个节点属性的第一个字段是 ip。

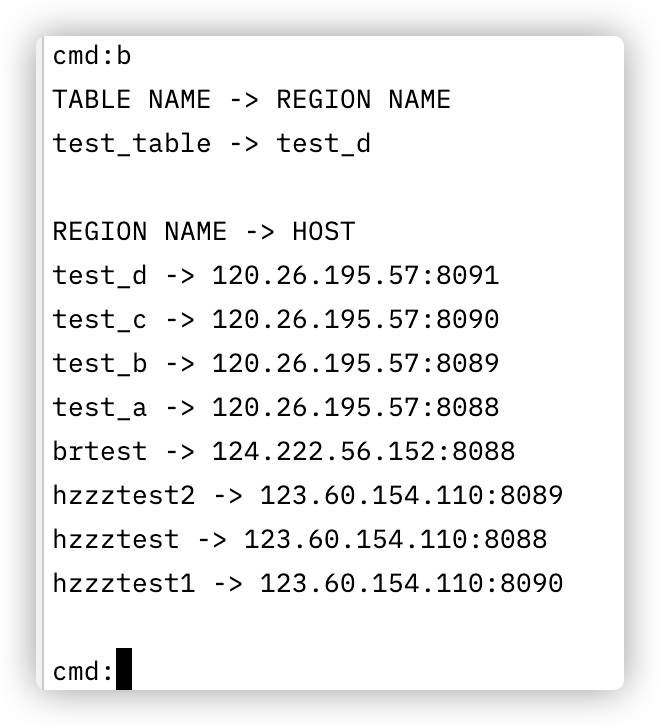
启动 Client，刷新缓存。



发送创建表test\_table命令。

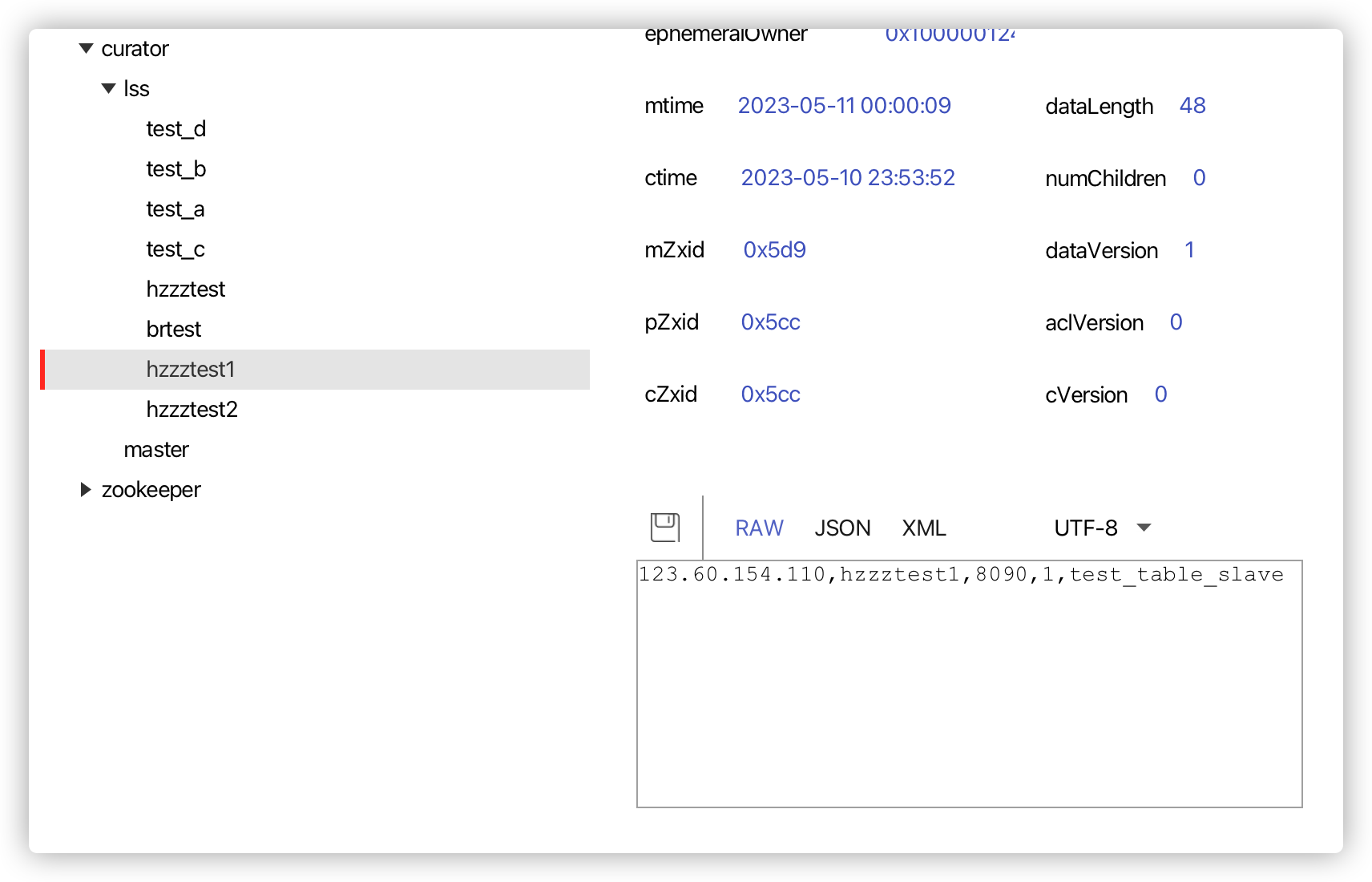


刷新 Client 缓存，可以看到现在主表test\_table 存储在节点test\_d上。

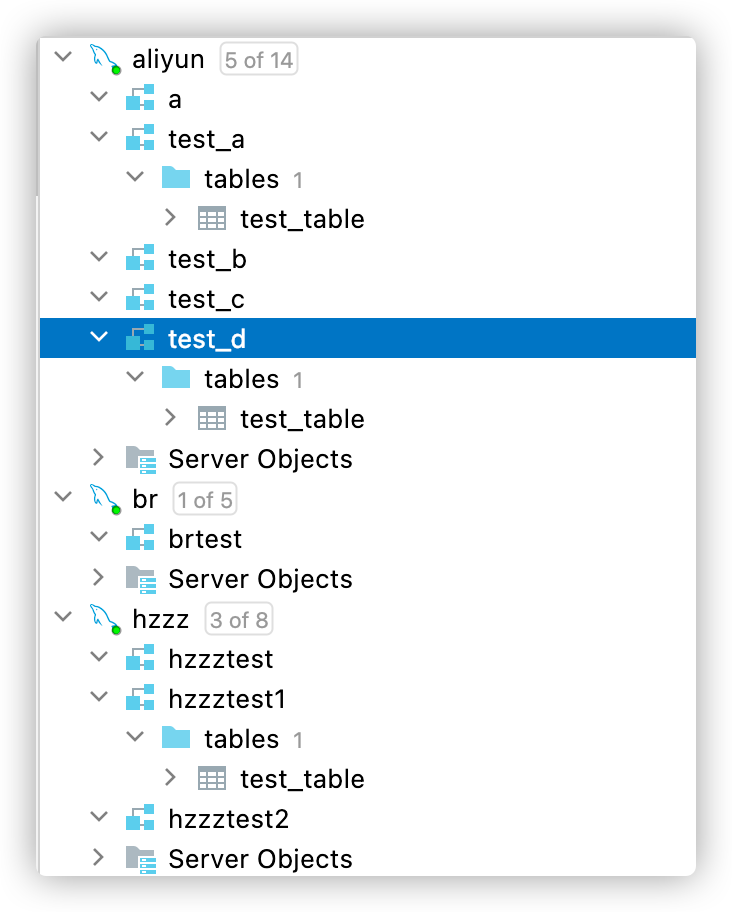


观察 ZK 集群的节点信息，可以看到从节点test\_a和hzzztest1存储了从表



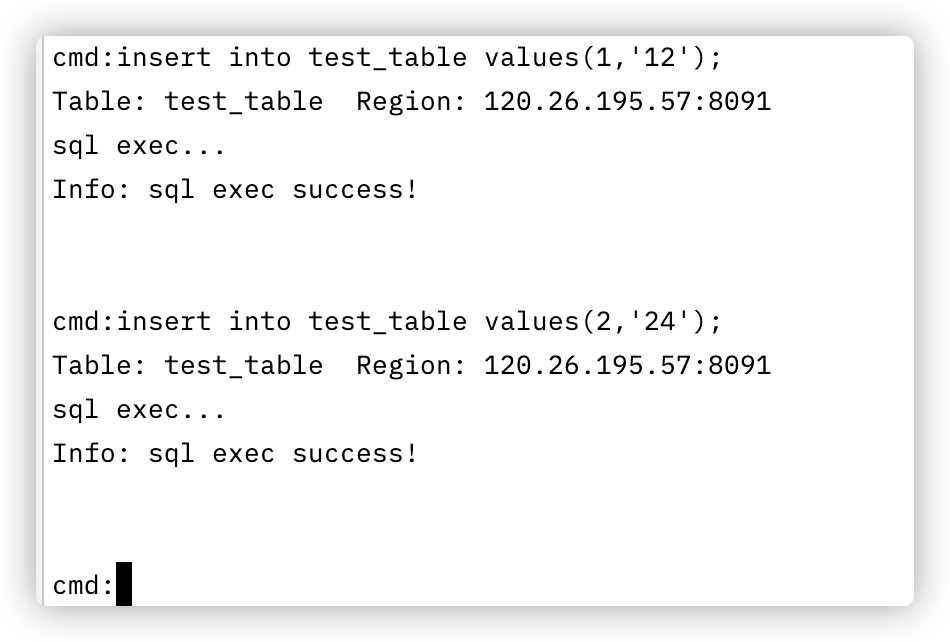


查看三个节点的数据库信息，可以发现表均正确地存储在数据库里。

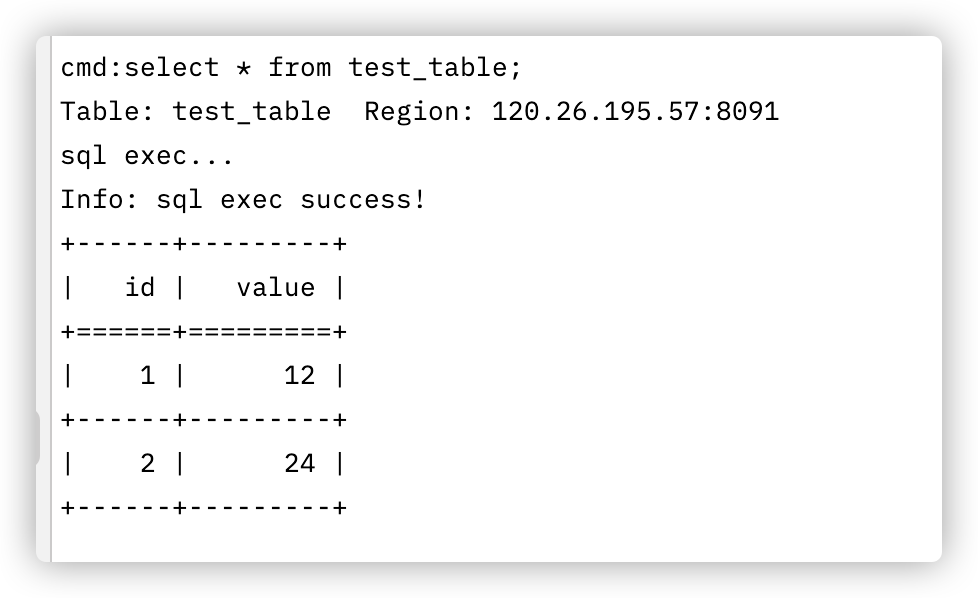


## 5.4 插入数据

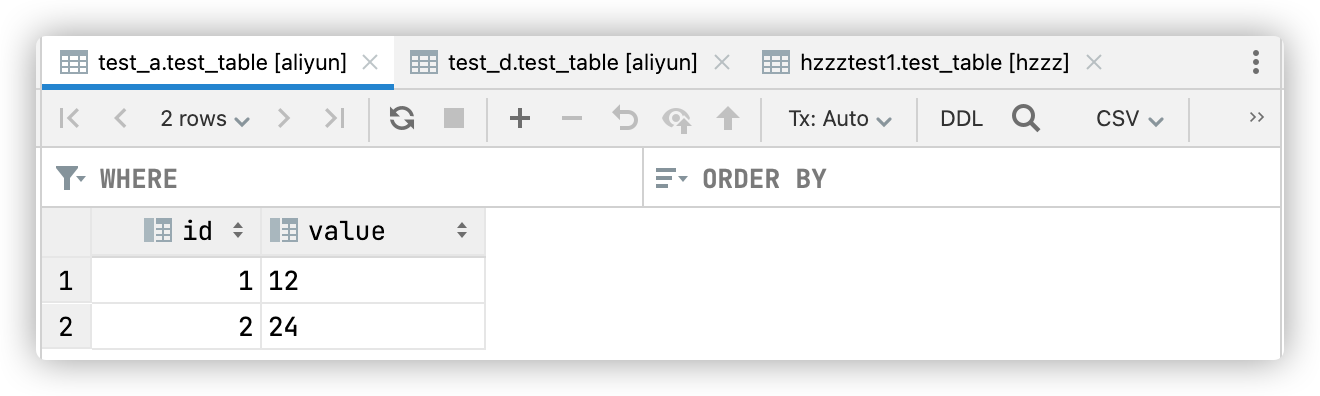
Client 对创建的test\_table发送插入命令。

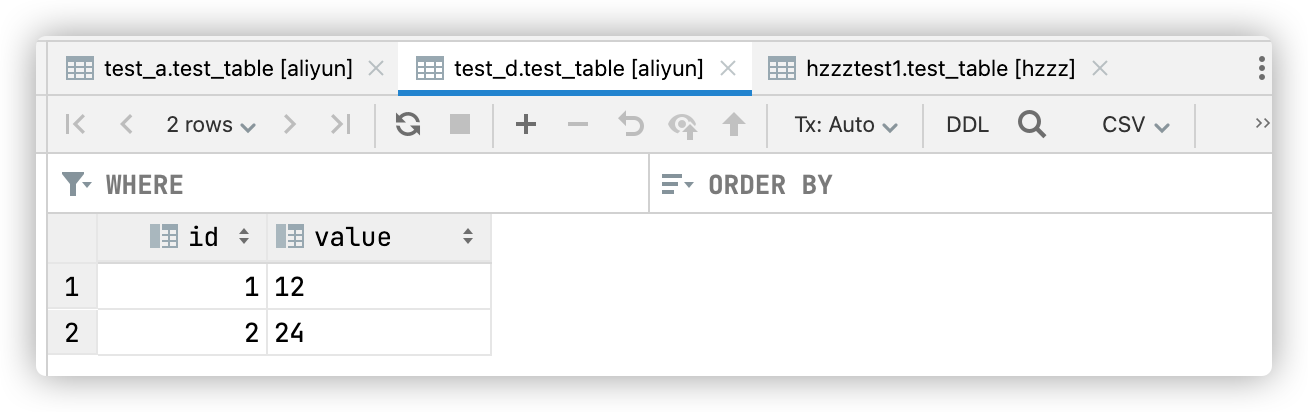


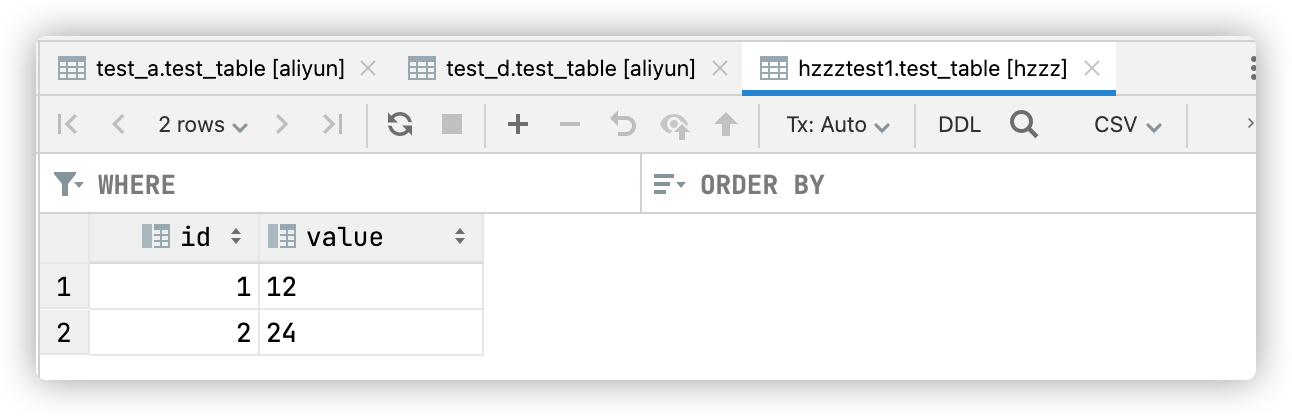
使用查询语句，发现返回了正确结果。



观察三个节点的数据库，发现信息均被正确存储。

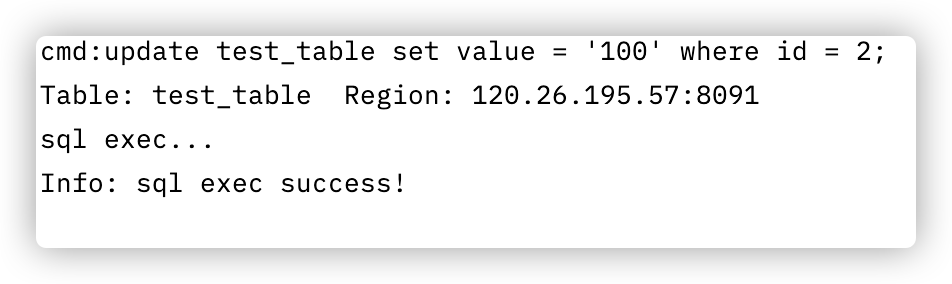




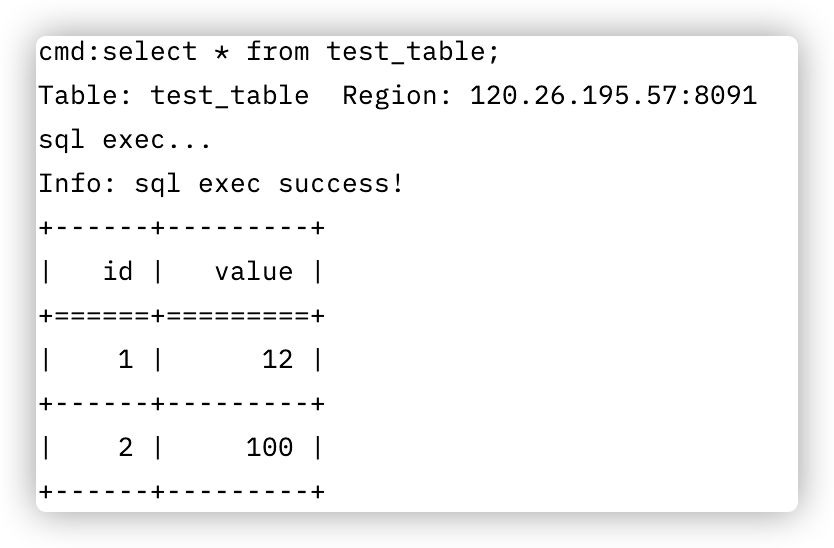


## 5.5 更新数据

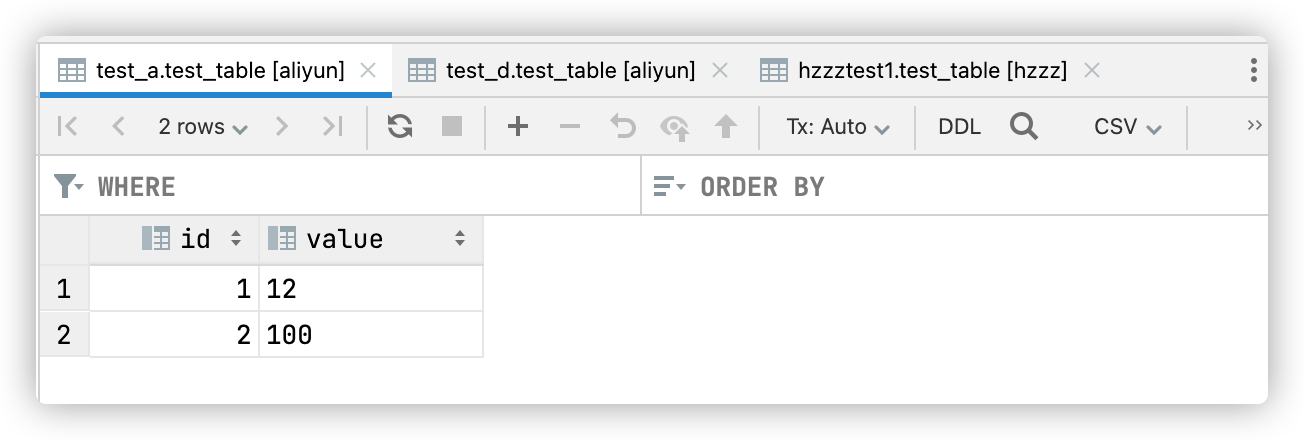
Client 对创建的test\_table发送更新命令。

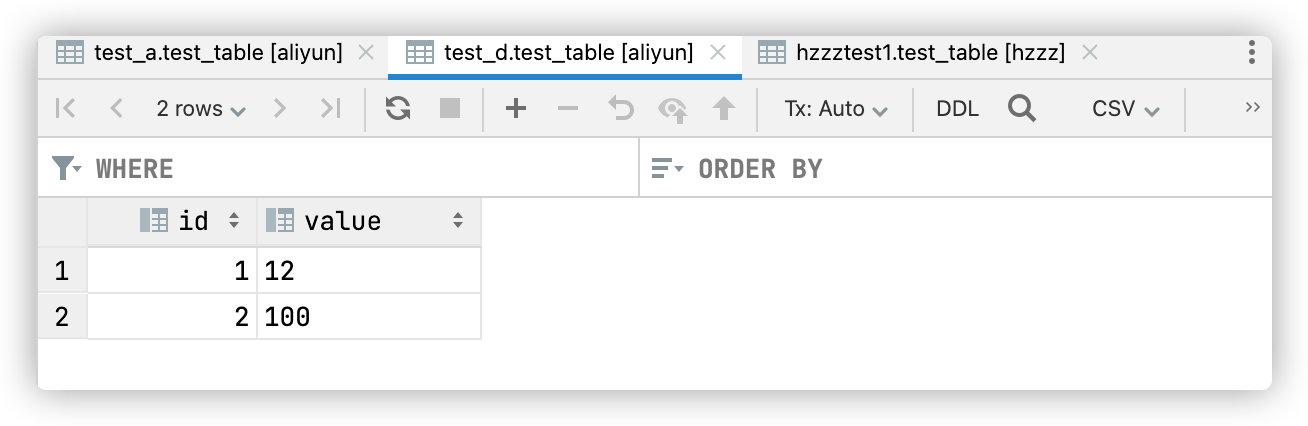


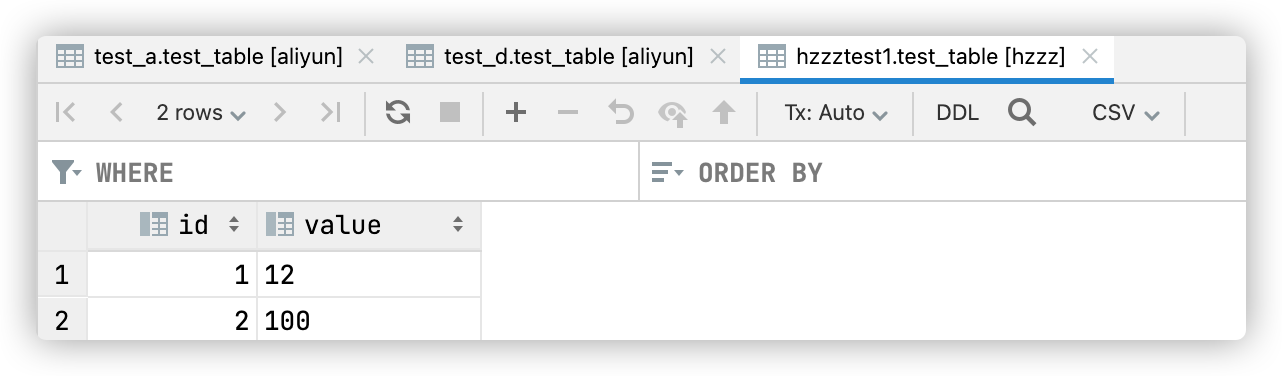
使用查询语句，发现返回了正确结果。



观察三个节点的数据库，发现信息均被正确存储。

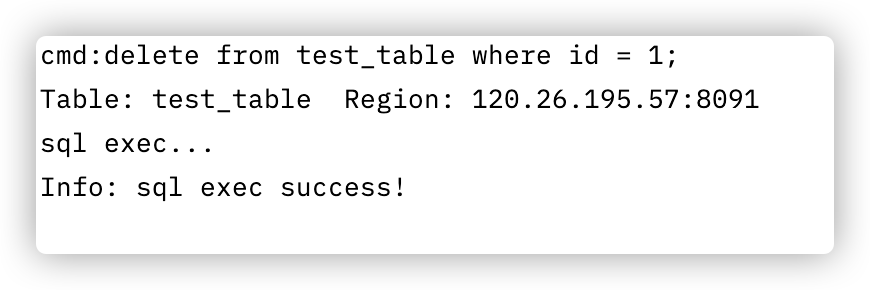




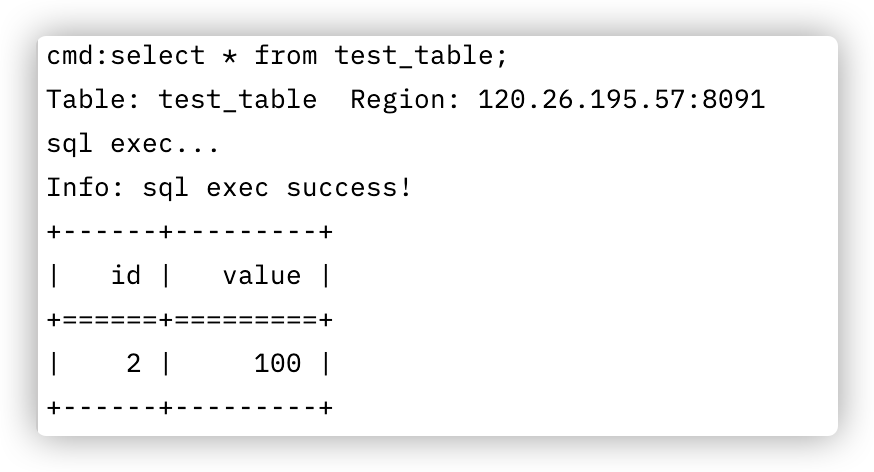


## 5.6 删除数据

Client 对创建的test\_table发送删除行命令。

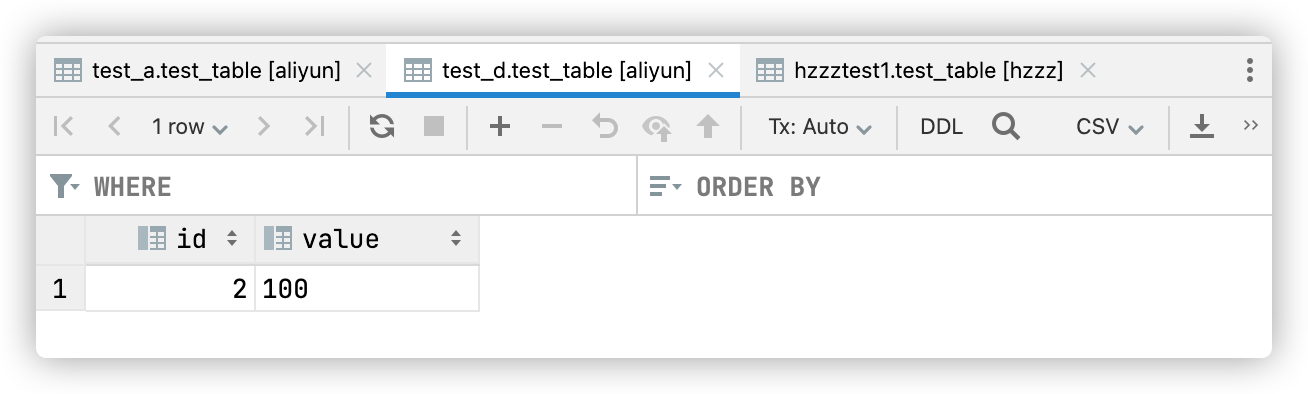


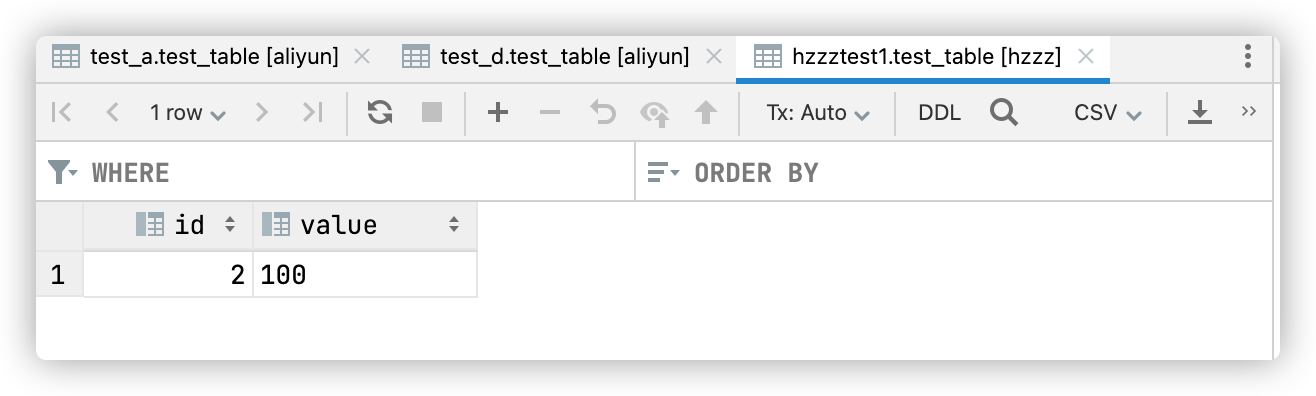
使用查询语句，发现返回了正确结果。



观察三个节点的数据库，发现信息均被正确删除。







## 5.7 主从切换

我们让主节点test\_d下线，观察其他节点的变化。



可以看到，这时候test\_a成为了test\_table的主节点。



可以发现原来的从节点test\_a变成了主节点，同时Master 从剩下的空闲节点里面选择了一个brtest，作为新的从节点，所以现在还是一主二从的模式，这是我们对容灾的实现。



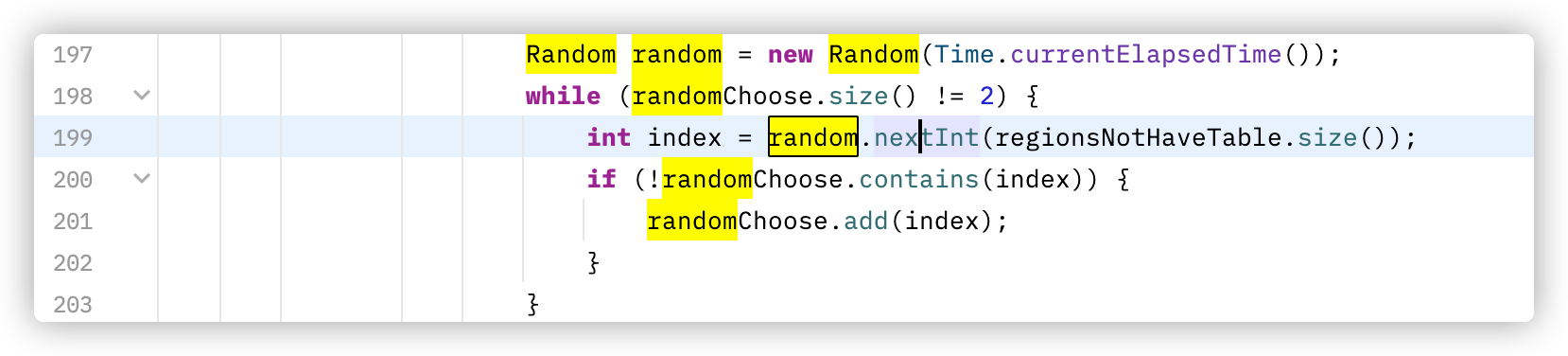
## 5.8 负载均衡

我们对负载均衡的实现如下。

在创建表的时候，Client 根据 zookeeper 节点的信息，选择主表最少的节点作为表的主节点（我们将主表的数量作为负载的主要衡量指标）



在主表选择从节点的时候，我们使用生成随机数的策略选择节点，从而实现负载均衡。



# 开发体会

通过这次大作业的开发，我对于分布式系统的开发更加熟悉，对 Java 语言和相关工具使用也更加熟练，对于团队协作开发也更加熟练。在完成项目之后，我们也进行了回顾总结，分析出我们在沟通、协调、技术、流程等方面的优点和不足，并制定了改进计划。这些经验使我深刻认识到一个好的团队是成功的关键所在，而与团队合作相比，技术本身的掌握虽然重要，但是远远不够。本次的大作业提升了我的架构设计能力，为之后的开发实践打下了坚实的基础。

# 参考文献

[1] https://curator.apache.org/apidocs/org/apache/curator/

[2] [https://spring.io](https://spring.io/)

[3] https://dev.mysql.com/doc/refman/8.0/en/