浙江大学

题　　目　分布式 MiniSQL 模块设计报告

授课教师　鲁伟明

学　　院　计算机科学与技术学院

专　　业　软件工程

姓　　名

学　　号

2023 年 5 月 17 号

目录

1 系统总体介绍

1.1 项目背景

本项目是一个分布式 MiniSQL 系统，属于《大规模信息系统构建技术导论》的课程项目。

本项目包含 Client，Master，Region Server 和 Zookeeper 共四个模块，实现一个在多主机上共享数据资源访问的分布式数据库系统，还实现了前端可视化界面，能够对简单 SQL 语句进行处理和解析，实现了基本分布式数据库的功能，包括容错容灾、负载均衡、副本管理等

本项目的 Master、Region Server、Zookeeper 使用 Java 语言开发，使用 Maven 作为项目管理工具，client 和前端使用 Javascript、CSS、HTML 基于 Vue 框架进行开发。整体项目并使用 Github 进行版本管理和协作开发，由小组内的五名成员共同完成。

1.2 系统目标

本项目开发之初针对分布式数据库的特点设计了以下具体功能目标，均得到了较好的实现：

- **负载均衡**

  为了避免单个数据点负载过重，我们设计了热点机制，将频繁访问的数据分散在不同服务器上；同时，在 Master 服务器的统一协调下，对数据的访问会按照轮询规则分摊到不同的 Slave 服务器上。

- **容错容灾**

我们小组针对容错容灾设计了投票机制、多活动中心、冗余和备份、基于临时节点的心跳机制、故障监测与恢复共五项措施，有效保证系统在面对异常情况时的正常运行。

- **副本管理**

对于副本的管理，服务器之间我们使用主从复制来协调多个服务器之间的数据关系，同时在多个 Zookeeper 节点之间我们使用 Master 来统一进行副本一致性的管理。而对于数据库内的数据，我们使用了 CRC32 来完成数据完整性的校验。

- **数据分布和集群管理**

本项目系统中包含多个 Region，每个 Region 中都有一个 Master 和多个 Slave，由 Master 来统一进行集群管理和数据分布管理。

- **分布式查询**

本项目支持基本 SQL 操作，基于 Master 路由功能实现了分布式查询。

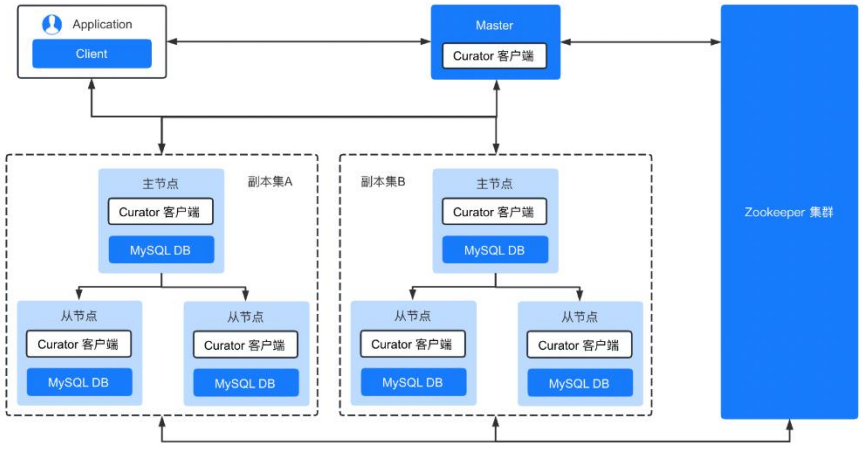
2 个人工作介绍

在本项目开发过程中，本人负责了总体的技术选型与架构设计，同时完成了 ZooKeeper 节点结构设计。对 Curator API 进行二次封装以实现 Master / Region Server 项目中的 ZooKeeper 工具类。同时完成了故障检测与恢复、集群管理的核心逻辑。

2.1 整体架构与技术选型

我们小组开发的分布式数据库系统包含了以下四个模块：Master（路由器）、Region Server、Zookeeper Server 和 Client。

- Master Server 负责维护集群的 Meta 信息并对用户请求进行路由，以实现全局管理和调度
- Region Server 实际处理用户操作，对本地数据库进行修改。
- Zookeeper 提供消息订阅与通知服务，以实现分布式协调和故障检测
- Client 负责与分布式系统进行交互，实现友好的用户界面

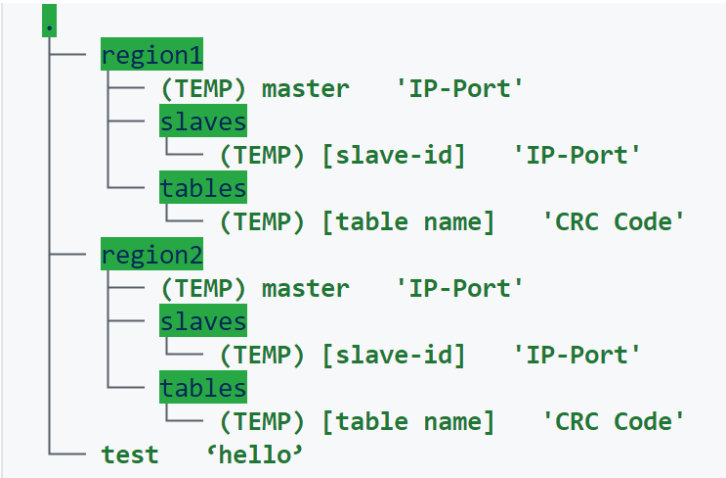


MiniSQL 项目整体架构图

本项目采用 3.8.0 版本的 ZooKeeper，基于 B/S 架构进行开发：Master 与 Region Server 均为 SpringBoot 后端项目，分别使用 JDBC 与 Curator 对 MySQL 与 ZooKeeper 进行操作；Client 为 Vue 前端项目，综合使用了 Vuex、PublSub-JS、ElementUI 组件库以提供更友好的用户交互服务。在 Region Server 上线时使用 MySQLdump 进行远程数据同步。

当用户试图操作数据库时，Client 首先会向 Master 请求具体执行操作的 Region Server 地址，随后再与具体的 Region Server 建立连接，并对返回结果进行渲染，整个过程对用户透明。

2.2 ZooKeeper 节点设计



ZooKeeper 节点结构示意图

各持久化节点在 Master 项目启动时被自动初始化，各节点功能如下：

- /test 持久化节点，内容为固定字符串“hello”

  用于在创建连接时读取，用以测试是否正确连接

- /region[n] 持久化节点，内容为空

  各子目录下存储对应 Region 的相关信息

  - /master 临时节点，内容为 “IP:port”

    由成功当选 Master 的 Region Server 自主填写

  - /tables 持久化节点，内容为空

    - /[table name] 临时节点，内容为数据表内容 CRC 校验码

      由当选 Master 的 Region Server 进行编辑

  - /slaves 持久化节点，内容为空

    - /[slave-id] 临时节点，内容为“IP:pot”

      由未能当选 Master 的 Region Server 自主填写

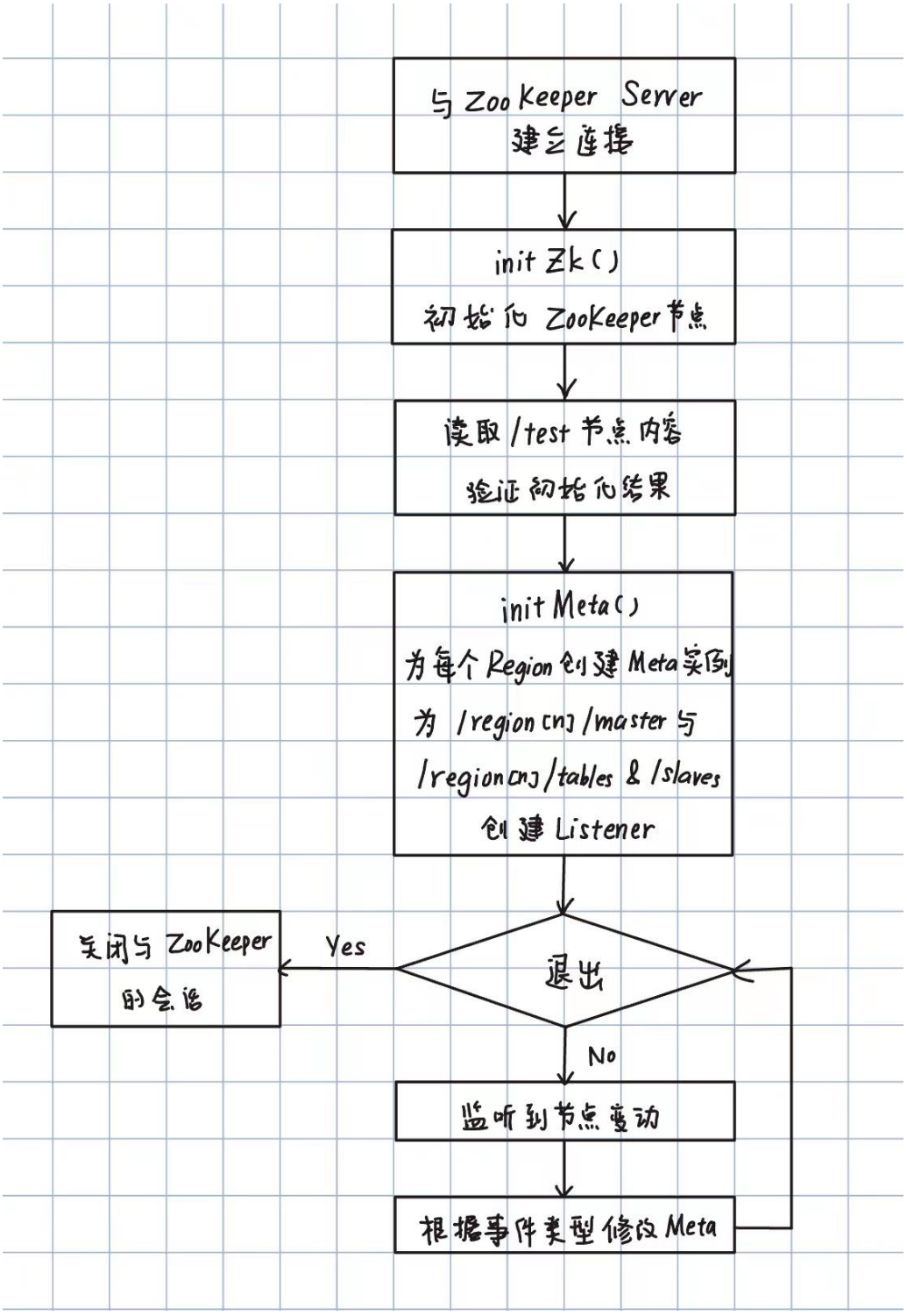
2.3 ZooKeeper 工具类实现

对 Curator 提供的 API 进行二次封装，并对原有的 NodeCacheListener 和 TreeCacheListener 进行拓展以满足不同的业务需要。

2.3.1 Master 中的 ZooKeeper 工具类

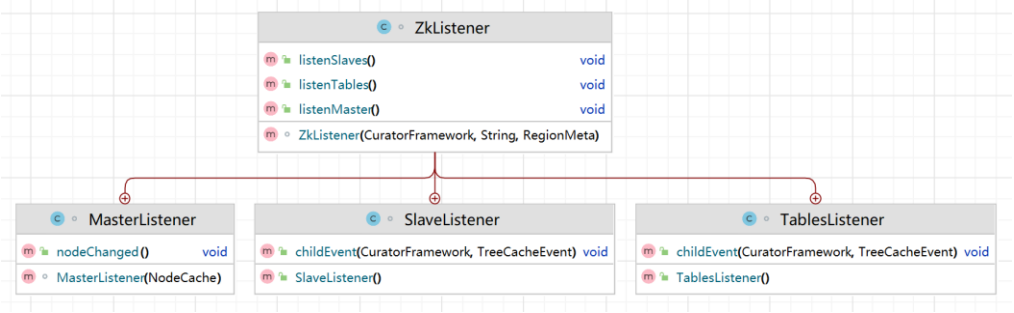
Master 项目中的 ZooKeeper 工具类图

1. 使用 connect 与 disconnect 对与 ZooKeeper 建立/断开行为进行了封装，其中连接的基本流程如下：



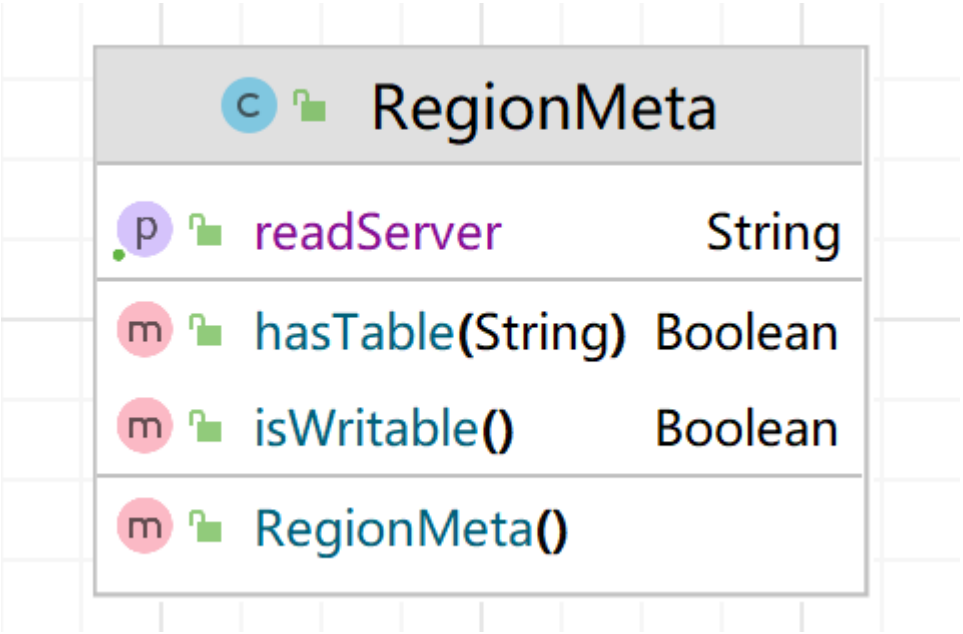
2. 针对监听的三类节点，我们对原有的 Listener 进行继承，分别实现了

MasterListener，SlaveListener 与 TableListner 以执行个性化的回调：



- MasterListner：监听 /region[n]/master 节点的创建/删除事件，并对 Meta 中的 MasterAddr 信息进行修改
- TableListner 与 SlaveListener：分别监听 /region[n]/tables 与 /region[n]/slaves 的所有子节点事件，并对 Meta 中的 tables 列表与 slaves 列表做对应修改

3. 该类使用一个 ArrayList 维护所有 Region 的 Meta 信息，每一个元素都是一个 RegionMeta 实例，结构如下：



4. 该类对 Curator 提供的一些 API 进行二次封装，使得调用过程更加简洁，如 exist()，createNode() 允许传入 path，data 等必要参数后自动进行相关操作；

getChildsData() 则用于获取指定路径下所有子节点包含的数据。

5. ZooKeeper 工具类同时为启动类提供了一系列基础以快速获取整个集群的总 Meta 信息与各 Region 状态：

   - getRegionMaster(int) 用于获取指定 Region 的主副本地址

   - getServers(int) 用于获取指定 Region 下所有 Region Server 的地址

   - getTables(int) 用于获取指定 Region 下所有表名构成的列表

   - isWritable(String) 用于判断指定 Table 所在 Region 当前是否可写

   - isAvailable(int) 用于判断指定 Region 是否可用——至少包含一台活跃 RegionServer

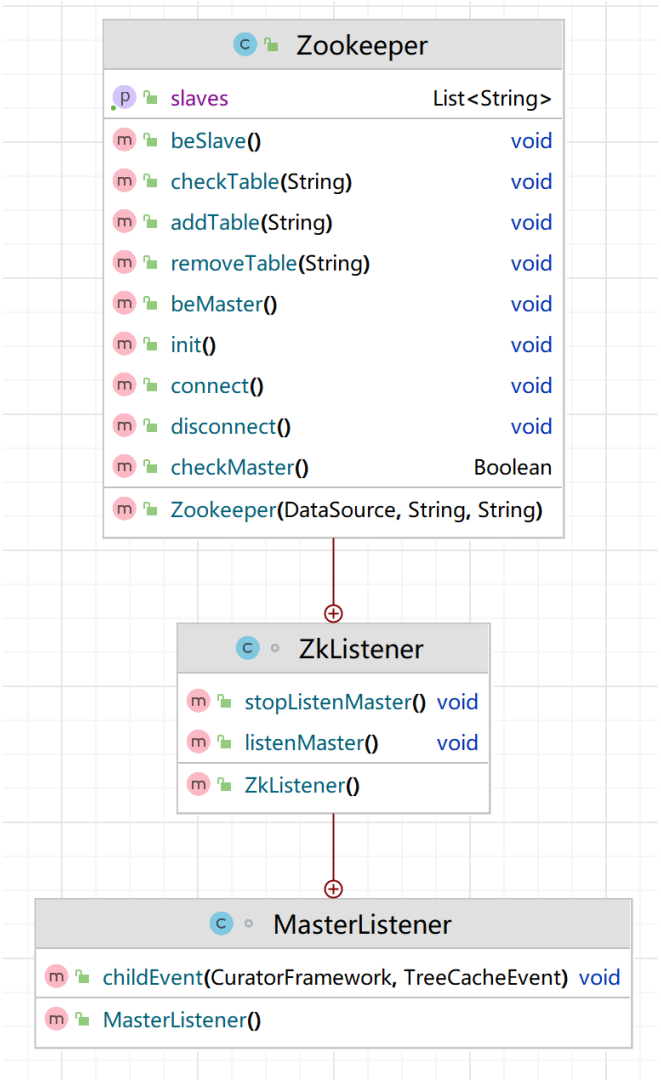
   - hasTable(String) 用于判断集群中是否存在同名表

   - hasWritable() 确定集群内是否有可写 Region

   - getWriteServer(String) 用于获取 Table 所在 Region 的主副本地址

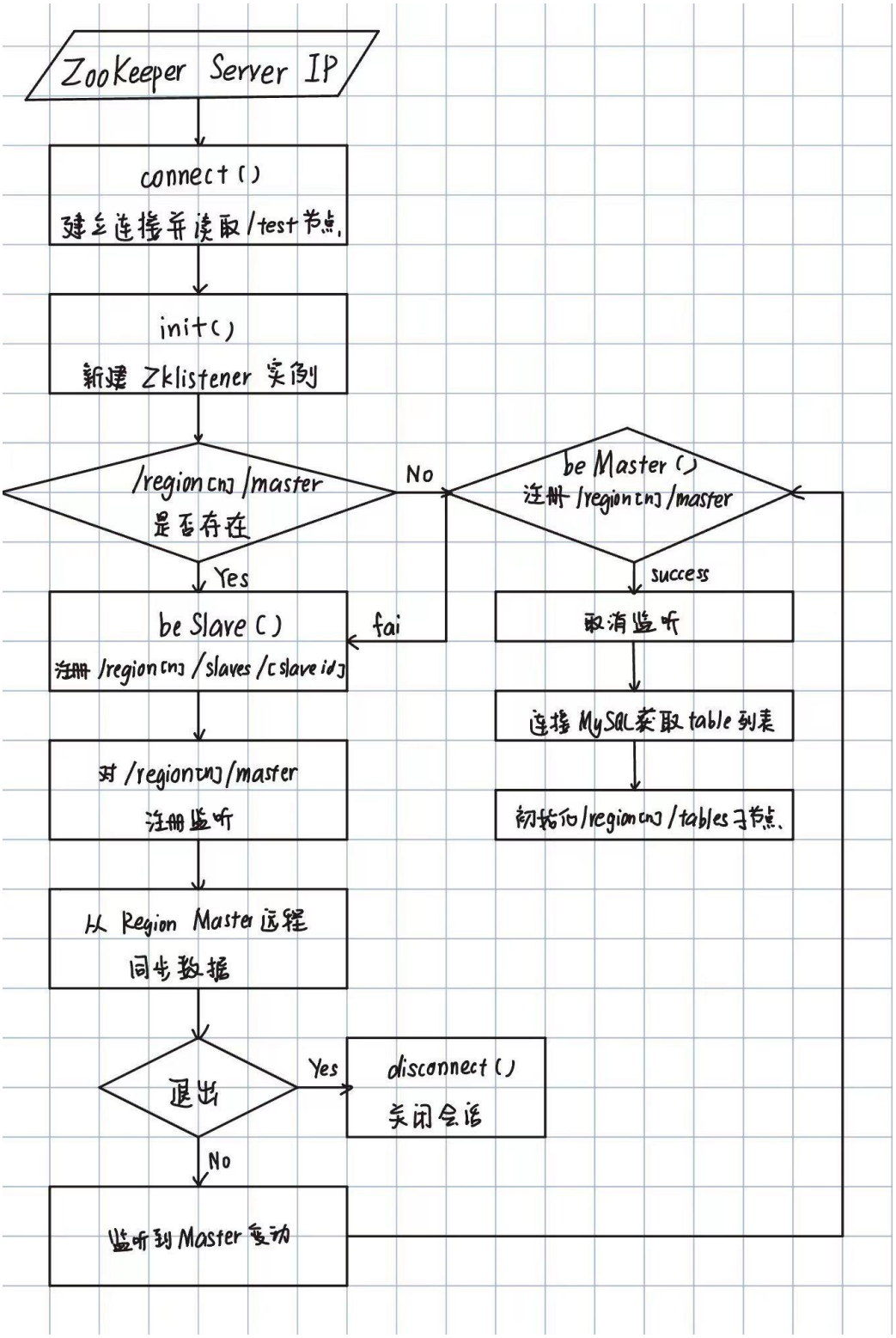
   - getReadServer(String) 返回用于读取指定 Table 的 Region Server 地址：首先在 salves 间轮询，仅有一台 Region Server 可用时路由至主副本

2.3.2 Region Server 中的 ZooKeeper 工具类



Region Server 项目中的 ZooKeeper 工具类图

1. 使用 connect 与 disconnect 对与 ZooKeeper 建立/断开行为进行了封装，其中连接的基本流程如下：



2. 由于 Region Server 仅在作为 Slave 时需要对 /region[n]/master 的相关事件进行监听，此处仅继承实现 MasterListener 类，用于在触发 /region[n]/master 节点的删除事件时自动调用 beMaster() 函数以自动实现 Master 选举。

   同时提供 stopListenMaster() 方法以便在 Slave 转变为 Master 时取消啊对节点消息的订阅。
3. beMaster() 与 beSlave() 函数分别对主从节点进行初始化操作，具体逻辑如上图所示。
4. checkMaster() 是向 Controller 暴露的方法，用于检测当前节点是否为主副本，已决定是否需要向子节点同步 SQL 操作。
5. addTable() 与 removeTable() 对 Curator API 进行了二次封装，用于在 ZooKeeper Server 的本 Region 的 tables 路径下添加/删除指定 Table 记录。
6. checkTable() 是对 CheckSum 工具类提供方法的封装，用于计算指定 Table 数据的校验和，并将结果写入 ZooKeeper Server 的对应节点中。

2.4 核心功能设计与实现

2.4.1 故障检测与恢复

- **故障检测**

本系统的故障检测功能基于 ZooKeeper 中临时节点会随会话关闭而删除的特性与 ZooKeeper 本身提供的消息订阅机制实现，通过“订阅-判断事件类型-执行回调操作”的过程对 Meta 信息进行动态维护。具体而言：

Master 项目对/region[n]/master，/region[n]/tables 子节点，/region[n]/slaves 子节点的创建和删除事件进行订阅与响应，根据触发的事件类型对内存中维护的 Meta 信息进行插入或删除操作，以便即时为 Client 提供正确的路由信息。

Region Server 中的从节点对本 Region 的 /master 节点删除事件进行订阅和响应。一旦检测到 /master 的删除事件就尝试通过注册 /master 以竞选成为主节点，成功则取消消息订阅，反之则继续订阅 /master 节点消息以实现自动 Master 选举机制。

- **故障恢复**

本系统当且仅当一个 Region 中存在大于等于 3 台（1*主节点+2*从节点）可用 Region Server 时方为可写 Region，因此在只有少数节点可用时数据将不会产生变动，以保证数据的一致性。

当节点重新上线时，首先检测本 Region 中是否存在主节点：

- 若不存在，则认为本地既存数据为最新数据副本，并注册成为主节点。
- 若已存在，则从 /region[n]/tables 子节点中读取主节点中存储的

Table 与其对应的校验和：

若本地不存在对应表，则从 /region[n]/master 节点中读取主节点地址，并通过 MySQLdump 直接从远端同步该表；

若本地已存在对应表，则对比本地数据与 ZooKeeper 节点中存储的校验和，仅当两者不等时通过 MySQLdump 从远端更新该表数据。

2.4.2 集群管理

本系统通过 Master 项目+ ZooKeeper Server 实现集群管理，包括：故障检测与可用节点信息维护、Table-Region 映射关系维护、用户请求路由实现以 Table 为单位的数据分布式存储。

故障检测与 Meta 信息维护基于 ZooKeeper 临时节点特性与消息订阅发布机制实现，具体逻辑已在 2.3 中说明，此处不做赘述。

用户请求路由基于 Master 在内存中维护的 Meta 信息实现，具体逻辑如下：

- 建表请求

1. Master 检查系统中是否存在同名表，若已存在则返回错误提示。
2. Master 检查系统中是否存在可写 Region：
   - 若不存在，返回错误提示。
   - 若存在，返回当前维护 Table 数量**最少**的 Region（若数量相等则返回首个 Region）的主节点地址，实现数据的分布式存储。

- Read 请求（SELECT 请求）

1. Master 检查系统中是否存在指定表，若不存在则返回错误提示。
2. 为避免主副本压力过大，Master 优先以轮询算法返回对应 Region 下的从

节点地址供用户执行 Read 操作。当前仅当该 Region 中不存在可用从节点时将 Read 请求路由至主节点。

- Write 请求（DROP/INSERT/UPDATE/DELETE 等修改数据请求）

1. Master 检查系统中是否存在待操作的表，若不存在则返回错误提示。
2. Master 检查 Table 所在的 Region 是否可写：
   - 若不可写，返回错误提示。
   - 若可写，返回该 Region 的主节点地址。

3 功能测试

3.1 ZooKeeper 连接与路径初始化

```
Master & Zookeeper are @10.181.215.240
Master listening Port: 9090
Trying to connect Zk Sever @10.181.215.240:2181
2023-05-18T04:17:00.769+08:00  INFO 12948 --- [           main] o.a.c.f.imps.CuratorFrameworkImpl
2023-05-18T04:17:00.775+08:00  INFO 12948 --- [           main] org.apache.zookeeper.ZooKeeper
2023-05-18T04:17:00.775+08:00  INFO 12948 --- [           main] org.apache.zookeeper.ZooKeeper
```

成功获取本机局域网 IP

```
initZk
2023-05-18T04:17:00.813+08:00  INFO 12948 --- [1.215.240:2181)] org.apache.zookeeper.ClientCnxn
2023-05-18T04:17:00.818+08:00  INFO 12948 --- [ain-EventThread] o.a.c.f.state.ConnectionStateManager
2023-05-18T04:17:00.827+08:00  INFO 12948 --- [ain-EventThread] o.a.c.framework.imps.EnsembleTracker
2023-05-18T04:17:00.827+08:00  INFO 12948 --- [ain-EventThread] o.a.c.framework.imps.EnsembleTracker
data @ /test = hello!
if you can see the return value, then you're successfully connected.
```

成功连接 ZooKeeper 、完成路径初始化并读取 /test 节点值

3.2 节点信息监听与 Meta 信息更新

```
/region1's MASTER is @10.181.215.240:9091
new TABLE for /region1 :password
new TABLE for /region2 :product
new TABLE for /region1 :user
new TABLE for /region2 :sales
new SLAVE for /region2 @10.181.215.240:9091, 1 SLAVES available now
2023-05-18T04:25:12.856+08:00  INFO 16536 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
```

启动后自动获取可用的 region Server 与 table 信息

```
2023-05-18T04:29:54.679+08:00  INFO 10068 --- [           main] main.MasterApplication
region1 - 0 times
region2 - 0 times
new SLAVE for /region1 @10.181.215.240:9091, 1 SLAVES available now
new SLAVE for /region1 @10.181.215.240:9091, 2 SLAVES available now
Checking HOT POINT...
```

监听到 Slave 节点添加事件

3.3 用户请求路由

```
Direct to 10.181.215.240:9091
new TABLE for /region1 :test
```

Master 收到请求，并路由至 Region1-Master

3.4 自定义路径 ZooKeeper 连接

```
Please input zkServer IP: 10.181.215.240
Current Server is @10.181.215.240:9091
Trying to connect Zk Sever @10.181.215.240:2181
2023-05-18T04:21:12.823+08:00  INFO 12716 --- [           main] o.a.c.f.imps.CuratorFrameworkImpl        : Starti
```

指定 Zookeeper Serve 地址并尝试连接

```
2023-05-18T04:21:12.865+08:00  INFO 12716 --- [ain-EventThread] o.a.c.framework.imps.EnsembleTracker     : New co
data @ /test = hello!
if you can see the return value, then you're successfully connected.
```

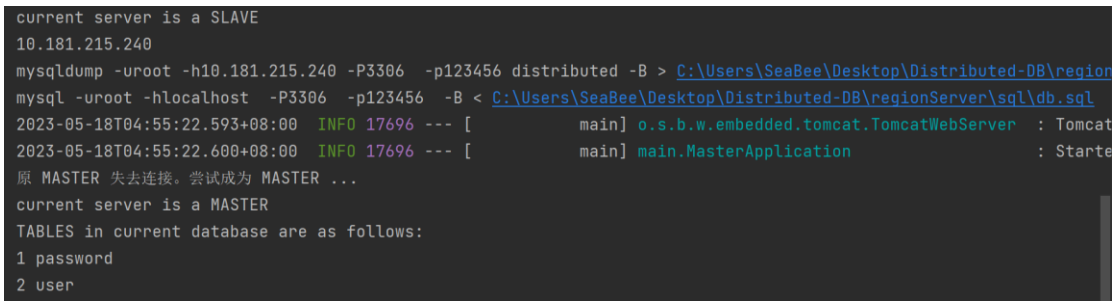
成功连接并读取测试数据 /test

3.5 beMaster()

```
current server is a MASTER
TABLES in current database are as follows:
1 password
2 user
```

当选 Master，获取 table 列表信息

3.6 BeSlave()



Master 已存在，从远端同步数据库

Master 已失效，竞选成功并修改 meta 数据