

大规模信息系统构建技术导论

分布式 MiniSQL 系统总体报告

2021 学年 第一学期（上）

组员信息

学号	姓名
3190106087	裴睿韬
3190103179	赵文琛
3190105239	张柏涵

2022 年 5 月 20 日

目 录

1 引言.....	1
1.1 项目背景.....	1
1.2 系统目标.....	1
1.3 设计说明.....	1
1.4 组员分工.....	2
2 总体设计.....	3
2.1 总体架构设计.....	3
3.核心模块设计.....	4
3.1 Master 层.....	4
3.2 Client 层.....	5
3.3 Region 层.....	7
3.4 Zookeeper 相关.....	9
4 分布式设计.....	12
4.1 负载均衡设计.....	12
4.2 容错容灾设计.....	13
4.3 副本管理设计.....	14
5 测试以及结果.....	16
6.总结.....	21

1 引言

1.1 项目背景

本次的项目开发为一个分布式 MiniSQL 系统，根据《数据库系统》课程所学的基本知识以及本课程所学的大规模信息技术系统构建的相关知识来进行项目的开发，通过 Client，Master，Region Server 以及 Zookeeper 集群等模块的设计，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。

1.2 系统目标

此次的实验项目为一个分布式 MiniSQL 系统，项目整体采用 Java 语言进行编写，需要实现的具体功能如下：

- (1) 数据库查询：支持基本的 SQL 语句查询，包括数据库数据的插入、查询、修改、删除等数据库的基本功能的实现
- (2) 副本管理：采用主从复制策略，选择其中的一个表作为主副本，负责副本的复制操作。
- (3) 负载均衡：Master Server 可以对其他 Region Server 进行管理，当某一 Region Server 出现繁忙时，Master Server 通过设计好的负载均衡算法将某些 Region 重新分配到其他的 Region Server 上；同时各 Region Server 可以实现数据传输，等待负载均衡之后修改 Table 的定位信息。
- (4) 容错容灾：当某个 Region Server 失效后，Master Server 可以将表的信息升级为主表并创建新的备份。

1.3 设计说明

本程序采用 Java 程序设计语言，使用 Maven 作为包管理工具，在 IDEA 开发环境下进行项目的编写、调试与测试等，使用 Zookeeper 作为集群管理工具，支持在不同的操作系统与语言环境下跨平台使用。

1.4 组员分工

表 1 各成员分工表

成员姓名	学号	分工
裴睿韬	3190106087	Zookeeper 模块封装, Client 端编写
赵文琛	3190103179	从节点 (Region Server) 开发, Record 模块开发, Buffer 模块开发
张柏涵	3190105239	客户端开发, Catalog 模块开发, Buffer 模块开发

2 总体设计

2.1 总体架构设计

本系统为分布式 MiniSQL 系统，主要架构特点如下：

(1) 该分布式 MiniSQL 系统主要由 Client，Master，Region Server 与 Zookeeper 集群模块组成，其中 Client，Master，Region Server 分别对应客户端，主节点，节点，通过 Zookeeper 的帮助对数据表信息进行管理。

(2) MiniSQL 为 Region Server 层提供技术服务。

(3) Client 层作为应用端实现用户交互。

系统总体架构图如下所示：

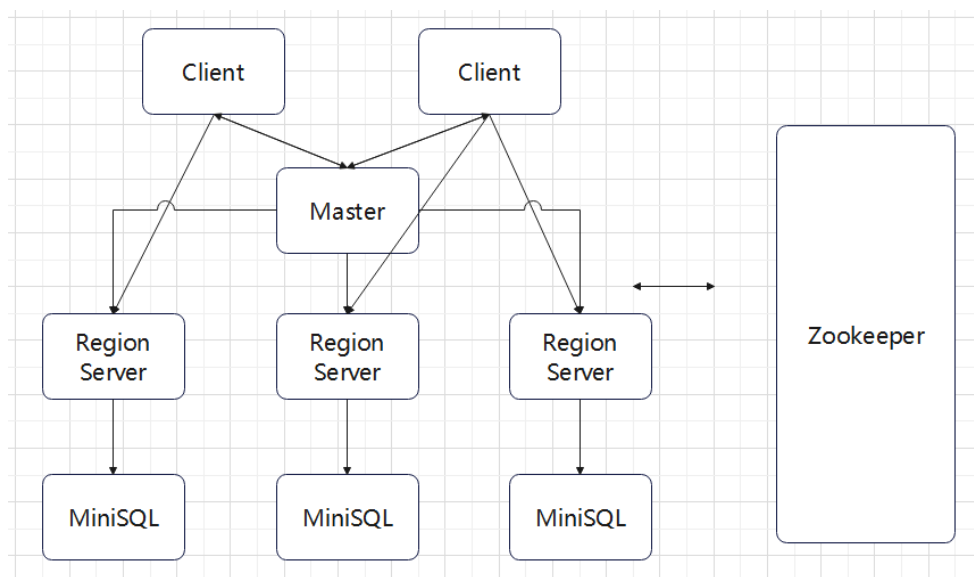


图 1 系统总体架构图

由于该项目重心更偏向于分布式系统处理，minisql 部分由功能更加成熟完善的 MySQL 代替，项目中由 MySQL 提供 sql 技术支持

3.核心模块设计

3.1 Master 层

Master 层概述

该项目中 Master 层的功能由两个子类实现：

(1) 建立与 Zookeeper 的连接，实现数据集群管理，对节点的状态改变进行监听，基于此可以实现容错容灾等功能；储存维护主节点的重要信息。例如

Region Server 列表以及 ip 的相关信息等，基于此可实现容错容灾，副本管理等功能。

(2) 建立与 Client 的通信，通过 Client 的请求查找相关 Region Server 信息，并将结果返回至 Client。整体工作流程图如下：

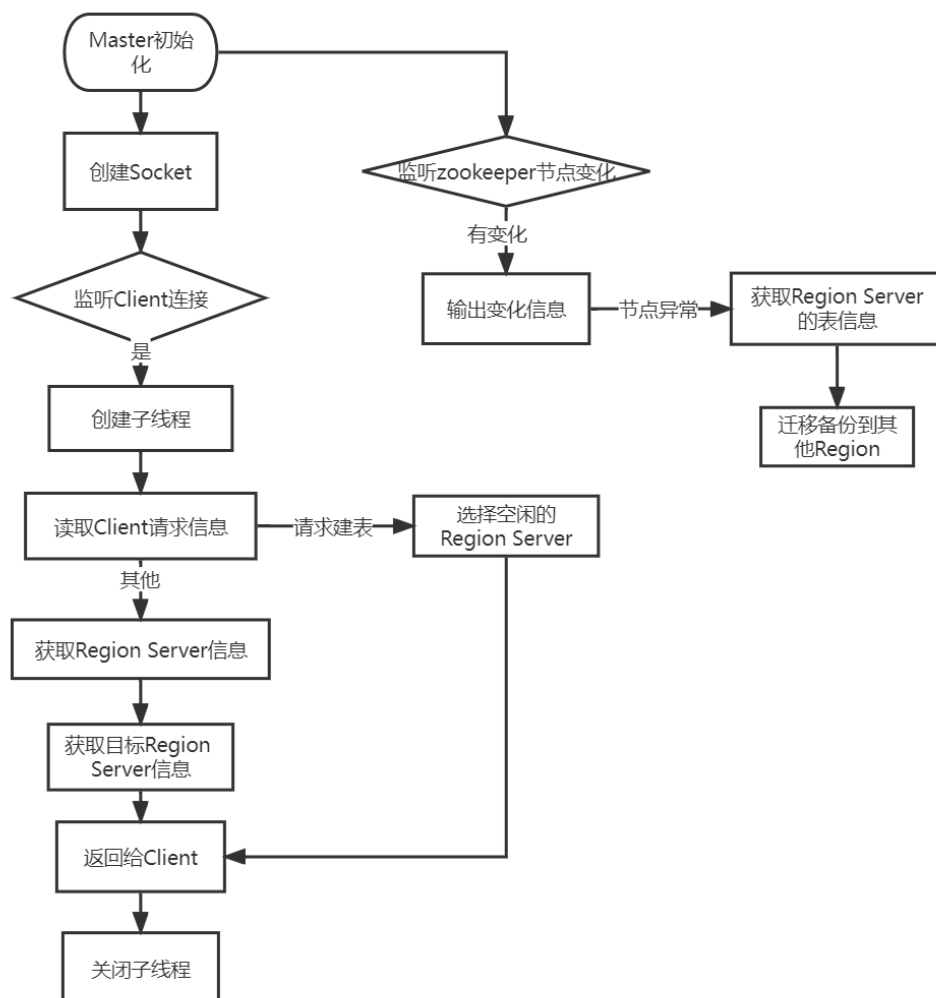
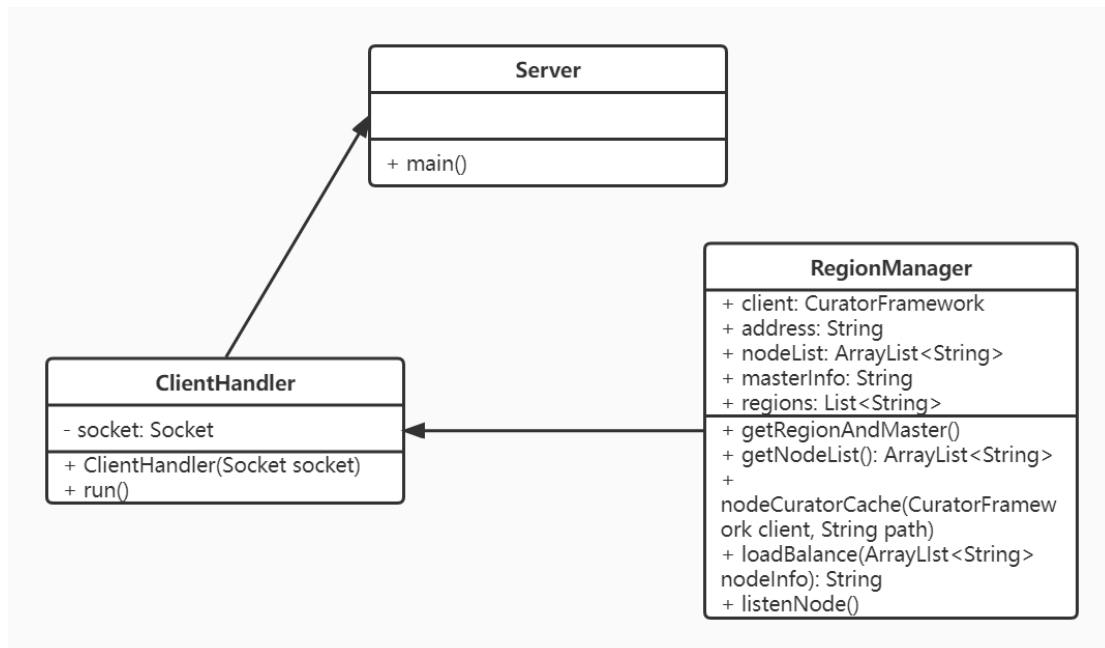


图 3 Master 整体工作流程图

Master 结构的实现

Master 的功能实现用到了以下的类的设计：



各个类所实现的功能如下：

(1) **Server**：负责创建 Socket 监听指定端口的连接请求，接收到来自 Client 的访问请求后调用 **ClientHandler** 类，为 Client 创建单独的线程进行数据通信。

(2) **ClientHandler**：继承于 **Thread**，调用时创建新的线程。运行时，首先调用 **RegionManager** 类的 **getNodeList()**方法获取当前所有 **Region** 的信息，根据 Client 发送的 Socket 请求内容，将对应的 **Region Server** 信息从列表中提取出来，并发送给 Client。

(3) **RegionManager**：负责 **Region Server** 的信息获取以及监听。通过调用封装好的 **Zookeeper** 实例，获取到当前 **Region Server** 的相关信息以及；并监听 **zookeeper** 节点的变化，实现容错容灾、负载均衡。

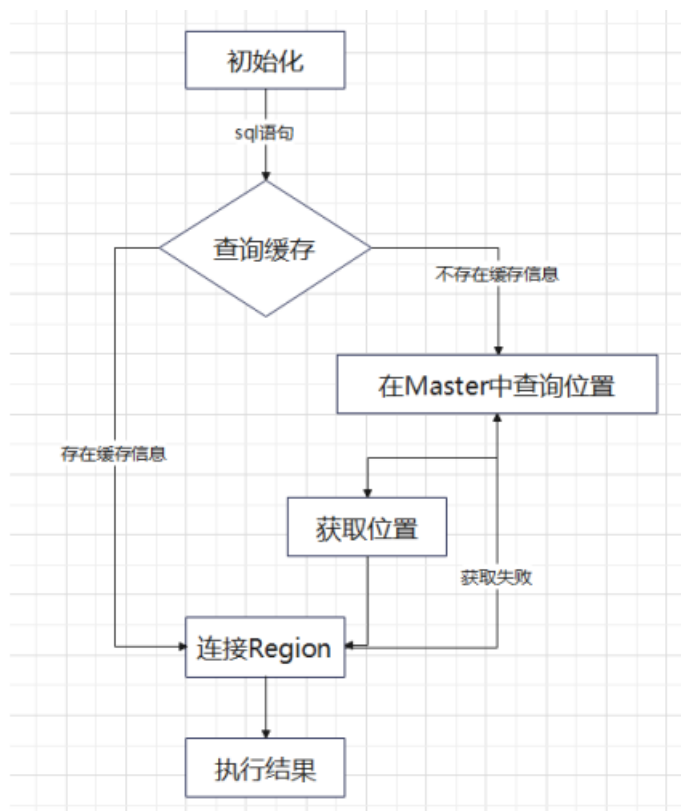
3.2 Client 层

Client 层概述

该项目中 Client 层主要由 getConnection() 方法, Communication 类以及主类构成, 主要实现了以下功能:

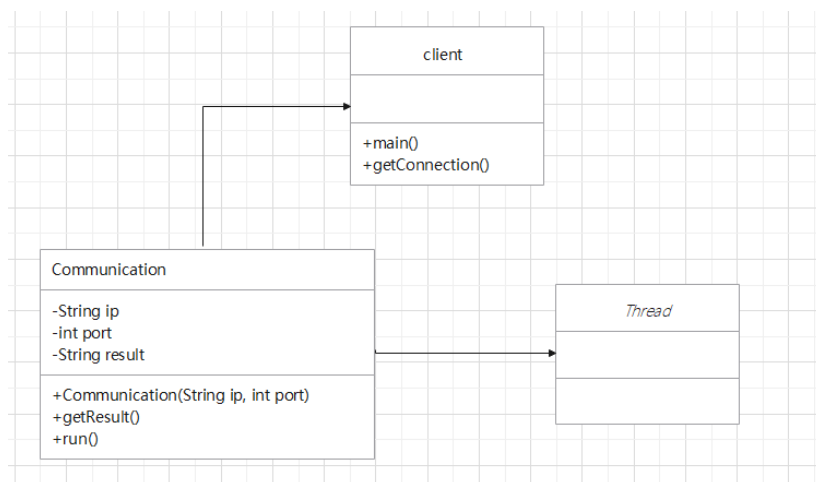
- (1) 实现了向指定端口进行 socket 连接的功能
- (2) 实现了向服务端输入 sql 语句并获取其返回的字符串的功能
- (3) 继承 Thread 类实现了多线程的功能
- (4) 实现了将返回字符串进行处理并存储到 HashMap 并打印出来的功能

整体流程图如下:



Client 结构的实现

Client 的功能的实现用到了以下类以及方法设计



相关的类及方法所实现的功能如下：

(1) getConnection() 方法包含两个形参 ip 和 port, 该方法接受了 ip 与 port 之后会与之建立 socket 连接, 连接建立后会立即输出 CONNECT CLIENT 以表明身份, 随后读入键盘输入的 sql 语句并向服务器端传输, 获取服务器返回的字符串后将其返回。

(2) Communication 类服务于多线程, 继承了 Thread 类, 包含 ip, port 与 result 三个成员变量以及构造函数与 getResult() 两个函数。getResult 函数通过调用 getConnection 方法获取服务器的返回字符串。

(3) 主类中实现了如下所示流程图的相关流程, 并将服务端返回的字符串进行处理后储存于 HashMap 并打印。

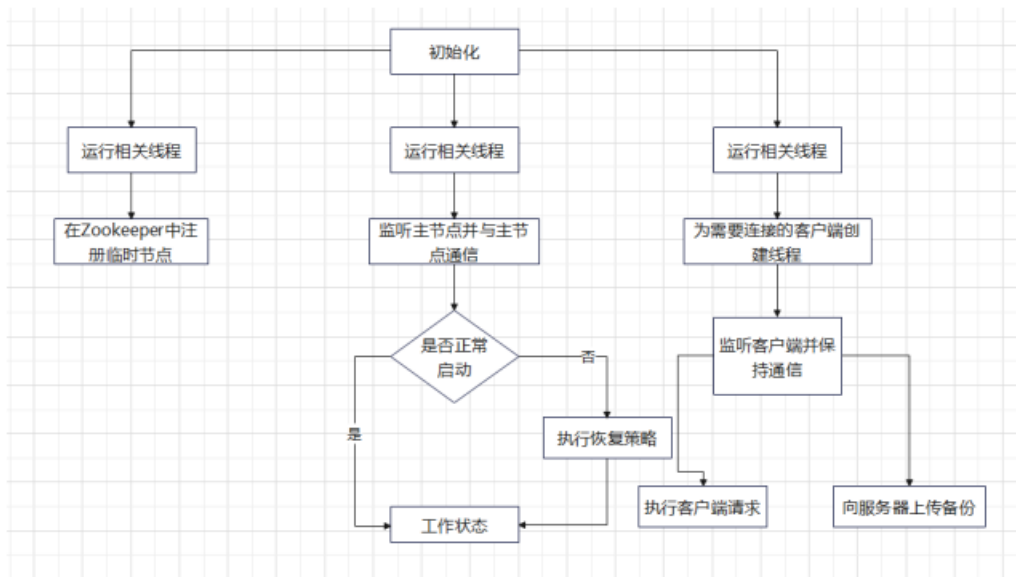
3.3 Region 层

Region 层概述

该项目中 Region 层主要实现了如下功能：

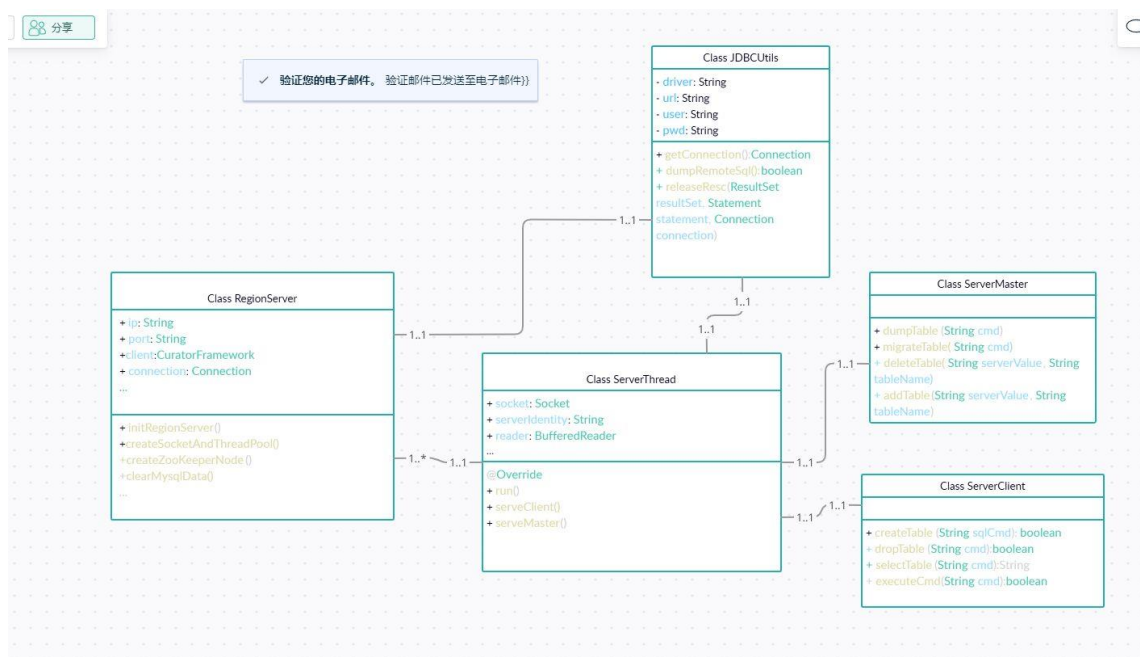
- (1) 处理 client 端发送的 sql 语句
- (2) 对表进行增删等操作并对 ZNODE 进行同步的修改
- (3) 调用 MySQL 的相关功能
- (4) 负载均衡以及容错容灾的相关功能

整体流程图如下所示



Region 结构的实现

Region 的功能实现用到了以下类设计：



相关的类以及所实现的功能如下：

- (1) RegionServer 类实例化了一个 RegionServer 对象
- (2) ServerThread 实现了多线程操作功能
- (3) JDBCUtils 实现了数据库相关操作的接口封装，以便于其他类调用接口对数据库进行操作。
- (4) ServerMaster 实现了与 Master 之间的交互，实现了使用 MySqldump 直接远程 dump 目标数据库的功能同时接受到 Master 迁移命令后迁移的 RegionServer

调用 `java runtime msyqldump` 命令，远程 dump 出某个表的创建插入语句等等，并在本地保存为 `sql`，紧接着使用 `mysql` 导入指令，执行 `sql` 文件，将数据库同步到本地 MySQL，然后调整 zookeeper 节点然后被迁移的 RegionServer 删除表，并调整 Zookeeper 节点

(1)

3.4 Zookeeper 相关

Zookeeper 节点设计：

```
├─lss
│  ├─master
│  │  ├─ip
│  │  └─port
│  └─region_servers
│     ├─server_1
│     │  ├─ip
│     │  ├─mysql
│     │  │  ├─port
│     │  │  ├─pwd
│     │  │  └─user
│     │  └─tables
│     │     ├─table_1
│     │     │  └─payload(number of item)
│     │     └─table_2
│     │        └─payload(number of item)
│     │  └─to_master_port
│     │  └─to_region_server_port
│     ├─server_2
│     │  ├─ip
│     │  ├─mysql
│     │  │  ├─port
│     │  │  ├─pwd
│     │  │  └─user
│     │  └─tables
│     │     └─table_1_slave
│     │        └─payload
│     │  └─to_master_port
│     │  └─to_region_server_port
│     └─server_3
│        ├─ip
│        ├─mysql
│        │  ├─port
│        │  ├─pwd
│        │  └─user
│        └─to_master_port
│        └─to_region_server_port
└─zookeeper
```

Zookeeper 接口封装

(1)创建客户端

```

/**
 * 创建客户端，默认连接本地2181端口
 * @return CuratorFramework
 */
public static CuratorFramework createAndStartClient() {
    ExponentialBackoffRetry exponentialBackoffRetry = new ExponentialBackoffRetry( baseSleepTimeMs: 1000, maxRetries: 3, maxSleepMs: 3);
    //创建客户端
    CuratorFramework client = CuratorFrameworkFactory.newClient( connectString: "127.0.0.1:2181", sessionTimeoutMs: 3000, connectionTimeoutMs: 3000, exponentialBackoffRetry);
    //启动客户端
    client.start();

    return client;
}

```

(2)创建客户端

```

/**
 * 创建客户端，默认远程2181端口
 * @param ip 远程zookeeper服务器ip
 * @return CuratorFramework
 */
public static CuratorFramework createAndStartClient(String ip) {
    ExponentialBackoffRetry exponentialBackoffRetry = new ExponentialBackoffRetry( baseSleepTimeMs: 1000, maxRetries: 3, maxSleepMs: 3);
    //创建客户端
    CuratorFramework client = CuratorFrameworkFactory.newClient( connectString: ip + ":2181", sessionTimeoutMs: 3000, connectionTimeoutMs: 3000, exponentialBackoffRetry);
    //启动客户端
    client.start();

    return client;
}

```

(3)关闭客户端

```

/**
 * 关闭客户端
 */
public static void closeClient(CuratorFramework client) { client.close(); }

```

(4)获取子节点

```

/**
 *
 * 获取子节点列表 打印
 * @param client 传入客户端
 * @param parentPath 节点路径
 * @return List<String>
 */
public static List<String> nodesList(CuratorFramework client, String parentPath) throws Exception {
    Stat stat = client.checkExists().forPath(parentPath);
    if (stat == null) {
        return null;
    }
    return client.getChildren().forPath(parentPath);
}

```

(5)创建无默认值的节点

```

/**
 * 递归创建一个节点，没有默认值
 * @param client 传入客户端
 * @param path 节点路径
 * @return void
 */
public static Boolean createNode(CuratorFramework client, String path) throws Exception{
    Stat stat = client.checkExists().forPath(path);
    if (stat == null) {
        client.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL).forPath(path);
    } else {
        System.out.println("节点目录已存在");
        return false;
    }
    return true;
}

```

(6)创建有默认值的节点

```
/**
 * 递归创建一个节点，有默认值
 * @param client 传入客户端
 * @param path 节点路径
 * @return void
 */
public static Boolean createNode(CuratorFramework client, String path, String value) throws Exception {
    Stat stat = client.checkExists().forPath(path);
    if (stat == null) {
        client.create().creatingParentsIfNeeded().withMode(CreateMode.EPHEMERAL).forPath(path, value.getBytes());
    } else {
        System.out.println("节点已存在");
        return false;
    }
    return true;
}
```

(7)获取节点信息

```
/**
 * 获取一个节点的信息，返回一个字符串
 * @param client 传入客户端
 * @param path 节点路径
 * @return String
 */
public static String getDataNode(CuratorFramework client, String path) throws Exception {
    Stat stat = client.checkExists().forPath(path);
    if (stat == null) {
        return null;
    }
    byte[] data = client.getData().forPath(path);
    return new String(data, StandardCharsets.UTF_8);
}
```

(8)设置节点信息

```
/**
 * 设置节点中的信息
 * @param client 传入客户端
 * @param path 节点路径
 * @param message 传入信息
 * @return boolean
 */
public static Boolean setDataNode(CuratorFramework client, String path, String message) throws Exception {
    Stat stat = client.checkExists().forPath(path);
    if (stat == null) {
        return false;
    }
    client.setData().withVersion(stat.getVersion()).forPath(path, message.getBytes());
    return true;
}
```

(9)删除节点以及其子节点

```
/**
 * 删除该节点以及该节点下子节点
 * @param client 传入客户端
 * @param path 节点路径
 * @return boolean
 */
public static Boolean delNodeAndChild(CuratorFramework client, String path) throws Exception {
    Stat stat = client.checkExists().forPath(path);
    if (stat == null) {
        return false;
    }
    client.delete().deletingChildrenIfNeeded().forPath(path);
    return true;
}
```

(10)删除某节点

```

/**
 * 删除当前节点
 * @param client 传入客户端
 * @param path 节点路径
 * @return boolean
 */
public static Boolean delCurrentNode(CuratorFramework client, String path) throws Exception {
    Stat stat = client.checkExists().forPath(path);
    if (stat == null) {
        return false;
    }
    client.delete().forPath(path);
    return true;
}

```

(11)检查节点是否存在

```

/**
 * 检查节点是否存在
 * @param client 传入客户端
 * @param path 节点路径
 * @return boolean
 */
public static Boolean checkNode(CuratorFramework client, String path) throws Exception {
    Stat stat = client.checkExists().forPath(path);
    return stat != null;
}

```

(12)监听某一节点

```

/**
 * 监听某个节点
 * @param client
 * @param path
 */
public static void nodeCuratorCache(CuratorFramework client, String path) {
    CuratorCache curatorCache = CuratorCache.build(client, path);
    CuratorCache cache = CuratorCache.build(client, path);
    CuratorCacheListener listener = CuratorCacheListener.builder()
        .forCreates(node -> System.out.println(String.format("Node created: [%s]", node)))
        .forChanges(oldNode, node) -> System.out.println(String.format("Node changed. Old: [%s] New: [%s]", oldNode, node)))
        .forDeletes(oldNode -> System.out.println(String.format("Node deleted. Old value: [%s]", oldNode)))
        .forInitialized(() -> System.out.println("Cache initialized"))
        .build();

    cache.listenable().addListener(listener);

    cache.start();
}

```

4 分布式设计

4.1 负载均衡设计

在该项目中我们通过 master 的调度实现了负载均衡的设计，在创建一张表时，总是会选择负载最少的节点进行创建，同时会对一定时间内某个节点的访问次数进行统计，若次数过高会将表迁移，负载均衡相关的代码如下：

```

public static String loadBalance(ArrayList<String> nodeInfo){
    if(nodeInfo.size() == 1){
        return nodeInfo.get(0);
    }
    else {
        ArrayList<String[]> node = new ArrayList<>();
        int min1 = 0;
        for (String s : nodeInfo) {
            node.add(s.split( regex: " "));
            if (s.split( regex: " ").length > min1) {
                min1 = s.split( regex: " ").length;
            }
        }
        int min2 = min1;
        int index1 = 0, index2 = 0;
        for (String[] arr : node) {
            if (arr.length <= min2) {
                if (arr.length >= min1) {
                    min2 = arr.length;
                    index2 = node.indexOf(arr);
                } else {
                    min2 = min1;
                    min1 = arr.length;
                    index2 = index1;
                    index1 = node.indexOf(arr);
                }
            }
        }
        return nodeInfo.get(index1) + ";" + nodeInfo.get(index2);
    }
}

```

4.2 容错容灾设计

在该项目中我们进行了容错容灾处理，每当创建表时都会创建相应的副表（table_slave），对表的属性进行修改时也会对其副表进行相应的修改。当主表下线时副表会升级成主表并创建相应的备份，当副表下线时会重新创建新的副表。容错容灾相关代码如下：

```

public static void listenNode() throws Exception {
    String brokenRegion = null;
    List<String> regionsNew = client.getChildren().forPath("/lss/region_servers");
    try{
        for(String region : regions){
            if(!regionsNew.contains(region)){
                brokenRegion = region;
                int index = regions.indexOf(brokenRegion);
                String[] nodeInfo = nodeList.get(index).split(regex: " ");
                nodeList.remove(index);
                ArrayList<String> tableList = new ArrayList<>(Arrays.asList(nodeInfo).subList(6, nodeInfo.length));
                for(String table : tableList){
                    if(Objects.equals(table.split(regex: " ")[table.split(regex: " ").length - 1], b: "slave")){...}
                    else{...}
                }
                break;
            }
        }
    }catch (Exception e){
        e.printStackTrace();
    }
}

```

4.3 副本管理设计

在该项目中我们对数据表进行了副本管理设计，这样当少数数据表出现问题时整个系统依然能正常工作，当创建或者对某个表进行增删改时，都会创建或者修改这个表的相应副表，起到备份的作用。副表的存在使得主表下线时该系统有处理应对的方法。相关代码如下所示：

```

public static int updateSlave(String table){
    StringBuilder tableName = new StringBuilder();
    String[] op = table.split(regex: " ");
    for(int i = 0; i < op.length - 1; i++){
        tableName.append(op[i]);
        if(i < op.length - 2){
            tableName.append("_");
        }
    }
    int min = 100;
    int index = -1;
    for(String node : nodeList){
        if(!(Arrays.asList(node.split(regex: " ")).contains(tableName.toString())
            || Arrays.asList(node.split(regex: " ")).contains(table))){
            if(node.split(regex: " ").length < min){
                min = node.split(regex: " ").length;
                index = nodeList.indexOf(node);
            }
        }
    }
    return index;
}

```

Master 的各功能测试如下：

(1) Master 启动:

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...  
Master is waiting for connection
```

(2) Client 向 Master 发送连接请求:

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...  
Master is waiting for connection  
New client connected, address: /127.0.0.1, port: 60019
```

(3) Client 请求查询:

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...  
select * from school
```

Master 端输出信息:

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...  
Master is waiting for connection  
New client connected, address: /127.0.0.1, port: 60019  
CLIENT REQUEST: select * from school  
TARGET REGION SERVER: 127.0.0.1,1001,3306,123,root,3,school,student,teacher_slave
```

Client 接收到 Master 的反馈:

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...  
select * from school  
REGION IP: 127.0.0.1, PORT: 1001
```

5 测试以及结果

1. 表的创建

① . 启动 regionserver

pc1

```
21:14:20.980 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 15
21:14:22.214 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 51
21:14:23.451 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 15
21:14:24.673 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 57
21:14:26.310 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 16
21:14:27.544 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 47
21:14:28.966 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 13
21:14:30.199 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 38
21:14:31.636 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 14
21:14:32.840 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 5m
21:14:34.292 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 12
21:14:35.510 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 5m
21:14:36.854 [main-SendThread(192.168.246.136:2181)] DEBUG org.apache.zookeeper.ClientCnxn - Got ping response for session id: 0x10004d9183f0026 after 4m
```

pc2

```
C:\Users\123\.jdk\openjdk-16.0.1-1\bin\java.exe ...
Local HostAddress: 192.168.246.136
Local host name: DESKTOP-4V8K83F
21:14:17.679 [main] INFO org.apache.curator.framework.imps.CuratorFrameworkImpl - Starting
21:14:17.688 [main] DEBUG org.apache.curator.CuratorZookeeperClient - Starting
21:14:17.689 [main] DEBUG org.apache.curator.ConnectionState - Starting
21:14:17.689 [main] DEBUG org.apache.curator.ConnectionState - reset
21:14:17.699 [main] INFO org.apache.zookeeper.ZooKeeper - Client environment:zookeeper.version=3.5.6
```

pc3

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...
Local HostAddress: 192.168.246.10
Local host name: LAPTOP-A0NEUGJO
```

启动 client

```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.1.12-hotspot\bin\java.exe" ...
```

启动 Master

```
C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...
Master is waiting for connection
```

Client 与 Master 进行通信

```

create table student(id int, name char(20))
create table student(id int, name char(20))
192.168.246.136,1001,root,123,3306,0;192.168.246.168,1001,root,123,3306,0
192.168.246.136,1001,root,123,3306,0
information of region_server is as follows
{mysql_pwd=123, region_port=1001, mysql_port=3306, mysql_user=root, table_num=0, region_ip=192.168.246.136}
information of region_server_slave is as follows
{mysql_port_slave=3306, mysql_user_slave=root, mysql_pwd_slave=123, region_ip_slave=192.168.246.168, region_port_slave=1001, table_num_slave=0}
|

```

Master 端

```

C:\Users\lenovo\.jdk\openjdk-17.0.1\bin\java.exe ...
Master is waiting for connection
New client connected, address: /192.168.246.168, port: 50442
CLIENT REQUEST: create table student(id int, name char(20))

```

Client 创建主副表命令

```

create table student(id int, name char(20))
create table student(id int, name char(20))
true
true
create table student_slave(id int, name char(20))
create table student_slave(id int, name char(20))
true
true

```

数据库结果展示:

```

Empty set (0.00 sec)
n]
n] mysql> show tables;
n] +-----+
n] | Tables_in_1ss |
n] +-----+
n] | student |
n] +-----+
n] 1 row in set (0.00 sec)

```



RegionServer:

pc2:

```
21:19:43.797 [main-SendThread(192.168.246.136:2181)] DEBUG c
CLIENT已连接
Client服务
Client请求创建表
create table student(id int, name char(20))
student
21:19:44.522 [main-SendThread(192.168.246.136:2181)] DEBUG c
```

pc3:

```
CLIENT已连接
Client服务
Client请求创建表
create table student_slave(id int, name char(20))
```

2. 向表中插入数据

Client 插入数据命令

```
insert into student value(3,'zwc')
insert into student value(3,'zwc')
true
true
```

```
insert into student_slave value(3,'zwc')
insert into student_slave value(3,'zwc')
true
true
```

RegionServer:

pc2:

```

21:19:59.289 [main-SendThread(192.
CLIENT已连接
Client服务
Client请求插入语句
21:20:00.632 [main-SendThread(192.

```

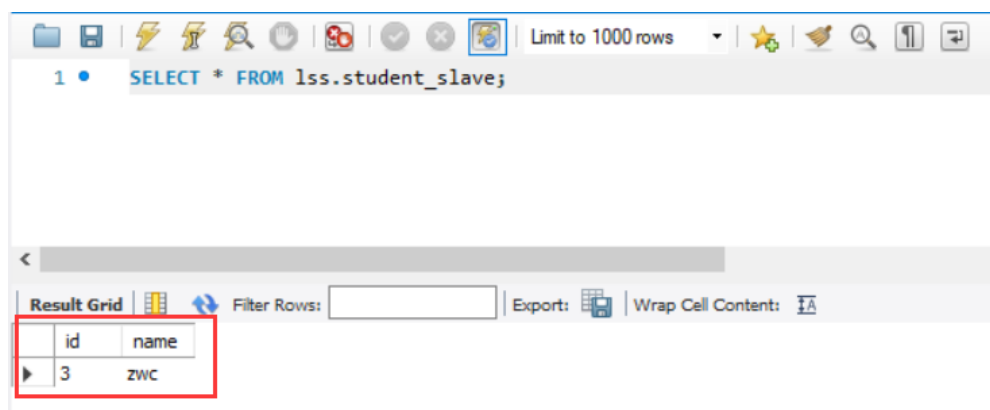
pc3:

```

CLIENT已连接
Client服务
Client请求插入语句

```

数据库展示



```

mysql> select * from student;
+----+-----+
| id | name |
+----+-----+
| 3  | zwc  |
+----+-----+
1 row in set (0.00 sec)

```

3. 删除表

Client 删除表命令

```

drop table student
drop table student
true
true

```

```
drop table student_slave
drop table student_slave
true
true
```

RegionServer

pc2:

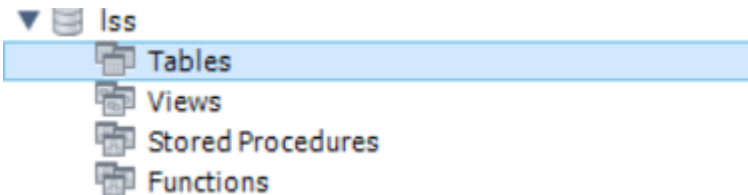
```
21:21:17.085 [main-SendThread(19
CLIENT已连接
Client服务
Client请求删除表
21:21:18.281 [main-SendThread(19
21:21:18.283 [main-SendThread(19
```

pc3:

```
CLIENT已连接
Client服务
Client请求删除表
```

数据库展示:

```
-S
mysql> show tables;
Empty set (0.00 sec)
mysql>
```



4. 整个过程中 zookeeper 节点的变化:

```
Node does not exist: /lss/region_servers/server_1
[zk: localhost:2181(CONNECTED) 111] ls /lss/region_servers
[]
[zk: localhost:2181(CONNECTED) 112]
```



```

server_1, server_2, server_3]
zk: localhost:2181(CONNECTED) 124] ls /lss/region_servers
server_1, server_2, server_3]
zk: localhost:2181(CONNECTED) 125]

192.168.246.168,1001,root,123,3306,0
[zk: localhost:2181(CONNECTED) 127] get /lss/region_servers/server_2
192.168.246.136,1001,root,123,3306,1,student
[zk: localhost:2181(CONNECTED) 128] _
INFO org.apache.zookeeper.ZooKeeper - Client environment: java.vendor=Or
in] INFO org.apache.zookeeper.ZooKeeper - Client environment: java.home=C:\U

192.168.246.10,1001,root,123,3306,0
[zk: localhost:2181(CONNECTED) 129] get /lss/region_servers/server_1
192.168.246.168,1001,root,123,3306,1,student_slave
[zk: localhost:2181(CONNECTED) 130]

INFO org.apache.zookeeper.ZooKeeper - Client environment: java.ver

[zk: localhost:2181(CONNECTED) 132] ls /lss/region_servers
i[server_1, server_2, server_3]
[zk: localhost:2181(CONNECTED) 133] get /lss/region_servers/server_1
192.168.246.10,1001,root,123,3306,0
[zk: localhost:2181(CONNECTED) 134] get /lss/region_servers/server_2
192.168.246.168,1001,root,123,3306,0
i[zk: localhost:2181(CONNECTED) 135] get /lss/region_servers/server_1
192.168.246.10,1001,root,123,3306,0
i[zk: localhost:2181(CONNECTED) 136] get /lss/region_servers/server_3_
i192.168.246.136,1001,root,123,3306,0
i[zk: localhost:2181(CONNECTED) 137] _

```

6.总结

到截止日期之前我们组的分布式 minisql 项目的基本功能已经实现,在此过程中大家对分布式系统的整体结构以及相关功能与操作有了较好的认知,并在项目的整合中得到了进一步的加强,需要反思的是容错容灾以及负载均衡相关的内容还不够完善,仍需改进