浙江大学

大规模信息系统构建技术导论

分布式 MySQL 系统设计报告

小组成员

学号	姓名
3200105872	庄毅非
3200104657	李予谦
3200100836	黄政

2023 年 5 月 22 日

目录

一. 系统简介

本项目是《大规模信息系统构建技术导论》的课程项目。在已经掌握的数据库基本知识与《大规模信息系统构建技术导论》课程上学习的分布式系统与大规模软件系统的构建的知识，通过小组合作，完成一个分布式的数据库管理系统，包含 Zookeeper 集群，Master，Region Server，Client 等模块，其基本功能，可以完成一个分布式的关系型数据库的各类基本操作，SQL 语句的执行，并且具有副本维护、负载均衡、容错容灾等功能。

本系统依托MySQL进行单机存储，使用JDBC简化数据库部分的功能实现，同时保证数据库存储上的可靠性和稳定性，使得我们能够有更多的精力实现分布式的存储功能。

本系统的 Region 和 Master 使用 Java 作为编程语言进行构建，Zookeeper 进行节点管理，在分布式集群上实现表的建立/删除、表记录的插入/删除/查找的基本功能的分布式关系数据库，通过基本功能的实现满足用户对于一个基本数据库功能的需要，同时考虑处理相关的，包括数据一致性等在内的、可能遇到的特殊情况。

本系统实现了两种类型的 Client 客户端，分别为使用 Python 的命令行工具客户端以及使用 React＋Flask 编写的网页客户端，提供了较为完整的用户交互功能，能够比较完整地呈现系统中的各项数据信息。

命令行工具客户端采用 Python 程序设计语言，在 Windows10 平台下编辑、编译与调试。与Zookeeper交流使用的库为kazoo。与Region交流的库为requests。

网页版客户端使用 React 作为前端，使用 axios 组件进行 HTTP 请求，使用 Flask 作为后端，标定网页客户端的 UID，辅助处理部分未在网页版客户端中实现的功能。

1.1 系统达成的目标

分布式 MySQL 是一个分布式数据库，允许用户通过字符界面输入 SQL 语句，在分布式集群上实现表的建立/删除、表记录的插入/删除/查找的基本功能的

分布式关系数据库，同时考虑处理相关的，包括数据一致性等在内的、可能遇到的特殊情况。具体功能如下：

（1）能够进行表的建立、删除

（2）能够进行记录的插入、删除

（3）能够进行记录的查找

（4）能够保证数据的一致性

（5）能够保证一定程度的容灾容错

对于 Master 模块，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。

对于 Region 模块，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。

定时和 Zookeeper 集群通信，保障本地缓存有效。

对于 Client 模块，具体功能如下：

（1）设置 Zookeeper 数据缓存，时刻与 Zookeeper 内元数据同步

（2）响应用户指令，发送 SQL 语句到缓存中对应的 Region 执行

（3）缓存线程安全、手动刷新缓存、缓存查看功能

（4）Zookeeper 数据变更时，通过监视器回调函数自动同步更新

（5）新建表时根据主表数量选择 Region 建立主表达成负载均衡

对于 Client（Web）模块，具体功能如下：

（1）能够进行表的建立、删除

（2）能够进行记录的插入、删除

（3）能够进行记录的查找

（4）能够保证数据的一致性

（5）能够显示数据表和有关查询结果以及报错信息

1.2 设计说明

本大程采用 Java、Python 以及 JavaScript 程序设计语言，在 Windows、Linux、MacOS 平台下编辑、编译与调试。具体程序由 3 人组成的小组开发而成。小组成员的具体分工如表 1 所示：

表 1 各成员分工表

成员姓名	学号	分工
庄毅非	3200105872	Master 和 Region，Region 的部署和调试
李予谦	3200104657	Client 的实现，Region 的部署
黄政	3200100836	网页版 Client 的实现，Region 的部署

（第一行为组长，分工需与后面的模块、功能等相对应）

二. 总体设计

2.1 总体架构设计

本系统主要分为以下几个模块：

（1）Master（主节点）

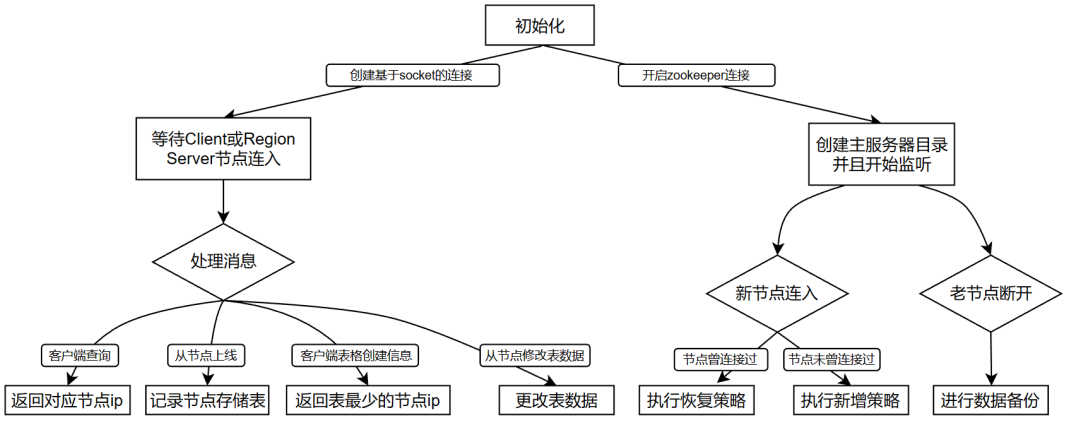
（2）Region（从节点）

（3）Zookeeper 集群

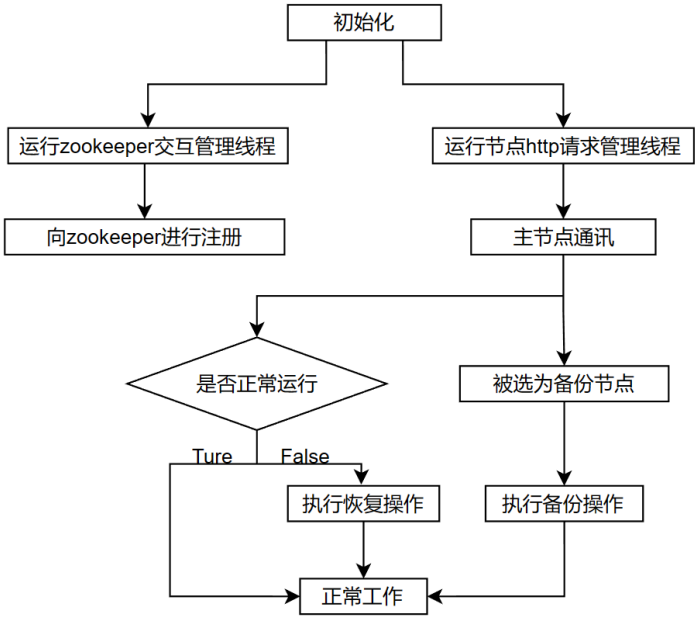
（4）Client（客户端）

2.2 流程图设计

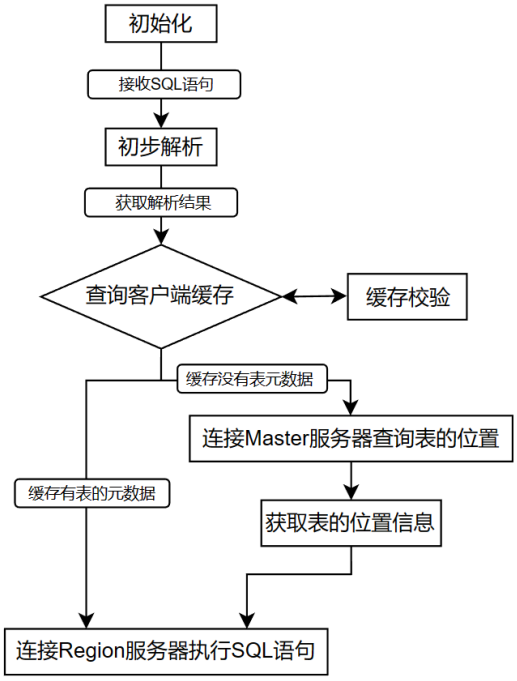
主节点（Master）工作逻辑：



从节点（Region Server）工作逻辑：



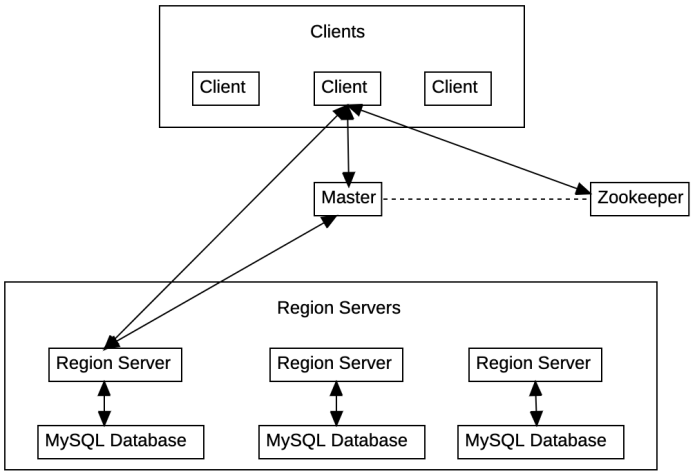
客户端（Client）工作逻辑：



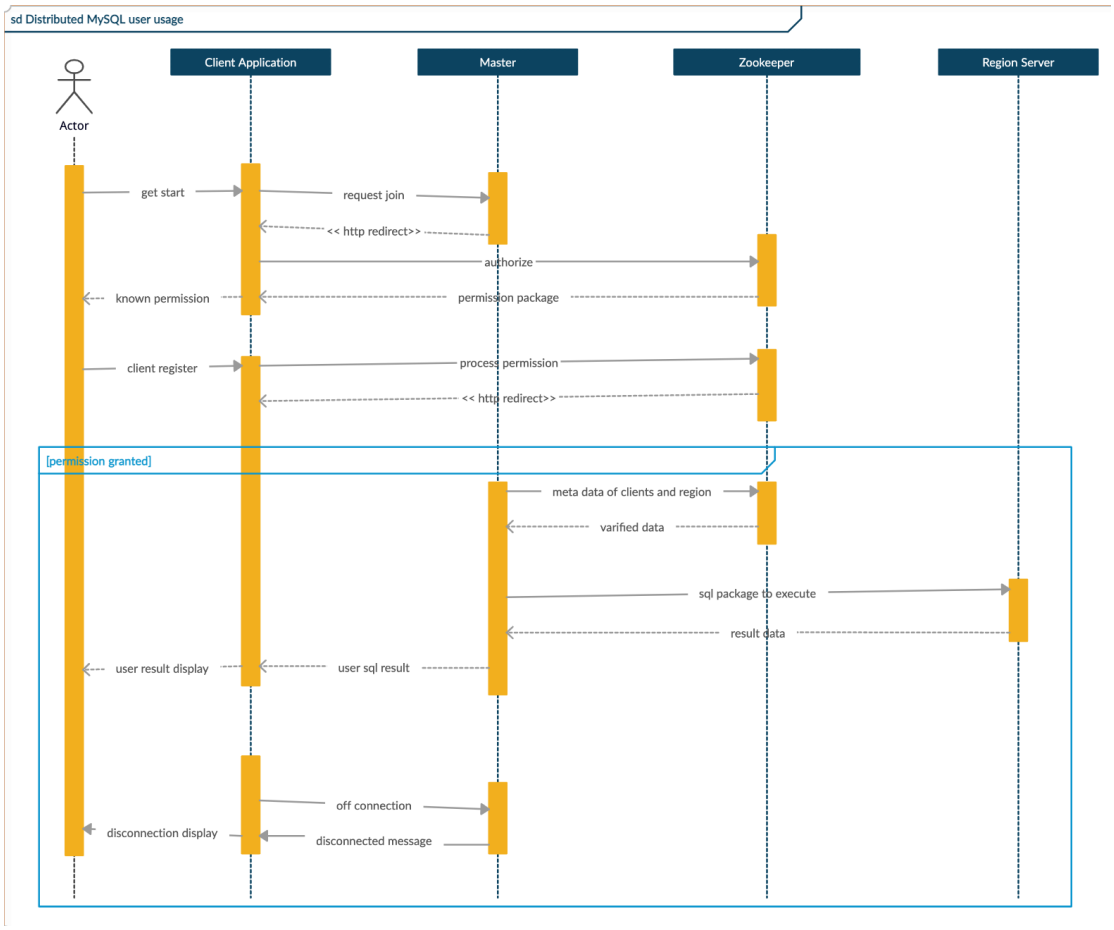
三．详细设计

3.1 分布式架构

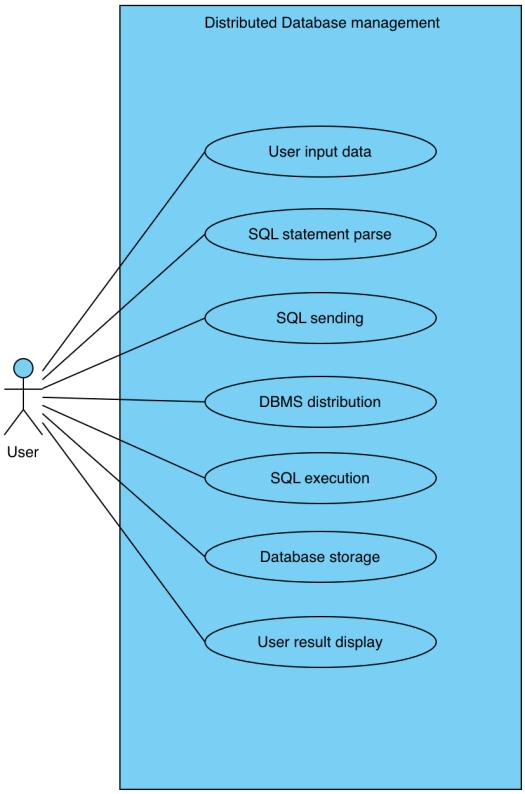
3.1.1 项目整体架构图


Clients
Client
Client
Client
Master
Zookeeper
Region Servers
Region Server
Region Server
Region Server
MySQL Database
MySQL Database
MySQL Database

3.1.2 时序图

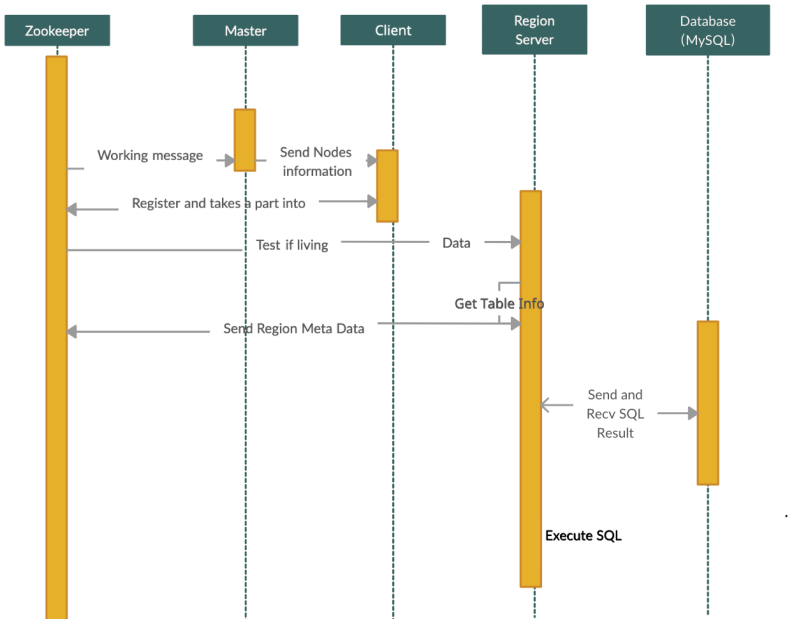

sd Distributed MySQL user usage
Client Application
Master
Zookeeper
Region Server
Actor
get start
request join
<< http redirect>>
authorize
known permission
permission package
client register
process permission
<< http redirect>>
[permission granted]
meta data of clients and region
varified data
sql package to execute
result data
user result display
user sql result
off connection
disconnection display
disconnected message

3.1.3 用例图


Distributed Database management
User input data
SQL statement parse
SQL sending
DBMS distribution
SQL execution
Database storage
User result display
User

3.2 集群管理

3.2.1 时序图



3.3 容错容灾

在分布式数据库系统中，容灾容错是非常重要的一部分，其详细设计需要考虑以下几个方面：

数据备份和恢复：系统需要定期对数据进行备份，以防止数据丢失或损坏。备份的频率和方式需要根据数据的重要性和实时性进行合理设置。同时，系统也需要能够快速恢复备份数据，以保证系统的可用性和稳定性。

数据冗余和副本管理：系统需要对数据进行冗余和副本管理，以防止单点故障和数据丢失。例如，系统可以采用数据分片和副本存储等方式，将数据分散到多个节点上，同时保证数据的一致性和可靠性。

故障检测和处理：系统需要实时检测节点和网络的故障，及时处理故障，保证系统的可用性和稳定性。例如，系统可以采用心跳机制等方式，检测节点的存活状态，同时对故障节点进行自动迁移和恢复等处理。

自动化运维：系统需要具备自动化运维的能力，例如自动化部署、配置管理、监控报警等，以保证系统的可靠性和稳定性。系统应该能够自动发现和处理各种

故障，自动调整资源分配，提高系统的响应速度和负载能力。

安全和权限管理：系统需要实现安全和权限管理，以防止未经授权的访问和数据泄露等风险。例如，系统应该能够提供数据加密、访问控制等功能，保护数据的安全和隐私。

以上是分布式数据库系统中容灾容错部分的一些详细设计考虑。在实际应用中，还需要根据不同的场景和需求进行适当的调整和优化。

在此次的开发工作中，针对容错容灾的不同方面的需求，我们将采取节点下线之后，使得 Master 选择一个没有该表的节点作为新的 slave，进行表的迁移复制，从而保障副本数量为 2（总数为 3）。

3.4 负载均衡

在分布式数据库系统中，负载均衡是一个重要的组成部分，它可以帮助系统更好地处理客户端请求，并将负载分配到系统中的多个节点。以下是一些关于负载均衡的详细设计考虑：

负载检测和监控：负载均衡器应该能够实时监测系统各个节点的负载情况，包括 CPU 使用率、内存使用率、磁盘使用率等指标。这些监测指标可以通过监控工具、日志分析等方式来获取。

负载分配策略：负载均衡器应该采用适当的负载分配策略来确保系统的高可用性和性能。常用的策略包括轮询、最少连接数、IP 哈希等。这些策略应该能够根据不同的场景和业务需求进行配置和调整。

动态负载均衡：负载均衡器应该能够根据实时的负载情况和节点状态进行动态负载均衡。例如，在节点出现故障或负载过高时，应该能够自动将请求分配到其他可用节点上，以避免系统崩溃或服务不可用。

负载均衡器的高可用性：负载均衡器本身也需要具备高可用性，以避免成为系统的瓶颈。因此，应该采用集群部署、备份等方式来保证负载均衡器的可靠性和容错性。

安全性：负载均衡器应该能够保证系统的安全性和防止恶意攻击。例如，应该采用防火墙、访问控制等手段来防止未经授权的访问和攻击。

以上是分布式数据库系统中负载均衡部分的一些详细设计考虑。在实际应用

中，还需要根据不同的场景和需求进行适当的调整和优化。在我们本次的项目开发中，我们着重考虑的是动态负载均衡，部分考虑负载均衡器的可用性。

在本次实现中，Region 在 Zookeeper 集群中维护各个 Region 的元信息（包括 host，port，表的个数以及表的列表等内容），在创建表的时候，Client 请求的 Region 会先创建数据表，然后更新该 Region 的元信息，Master 在监听性的监听并且接收到之后，判断是新表创建，就会随机选两个节点作为 slave 节点。

之后，当 Client 需要请求对应表的 Master Region，当该 Region 执行完之后，进行判断 SQL 是修改表的 SQL 语句（比如 insert，update，delete），如果判定为是，那么就会进行主从同步。

节点下线之后，master 会选择一个没有这个表的节点作为新的 slave，保障副本数量为 2。

3.5 客户端

在分布式数据库系统中，客户端是用户与系统之间的接口，需要考虑以下几个方面的详细设计：

客户端与服务器的通信：客户端需要能够与系统中的服务器进行通信，常用的通信协议包括 TCP/IP 和 HTTP 等。客户端应该能够与服务器进行安全的通信，例如采用加密通信等方式保证数据的安全性和保密性。

用户权限管理：客户端需要能够进行用户权限管理，包括用户认证、授权等。客户端应该能够验证用户的身份，并根据用户的权限和角色授予相应的权限，以保证数据的安全性和保密性。

查询语言支持：客户端需要能够支持系统所使用的查询语言，例如 SQL 等。客户端应该能够提供易用的查询界面和工具，帮助用户更方便地进行数据查询和分析。

多平台支持：客户端应该能够支持多种平台和操作系统，例如 Windows、Linux、macOS 等。客户端应该能够提供相应的客户端软件和驱动程序，以便用户能够在不同的平台上使用系统。

数据可视化支持：客户端应该能够支持数据可视化，包括图表、报表等。客户端应该能够提供易用的数据可视化工具，以帮助用户更好地理解和分析数据。

异常处理和错误日志记录：客户端应该能够进行异常处理和错误日志记录，例如记录用户的操作记录和错误信息，以便系统管理员进行分析和排查问题。

以上是分布式数据库系统中客户端部分的一些详细设计考虑。在实际应用中，还需要根据不同的场景和需求进行适当的调整和优化。

在此次的功能开发中，针对客户端不同方面的需求，我们将采用不同的技术和实现去完成不同方面的功能：

客户端与服务器的通信方面，我们将使用 HTTP 请求实现 Client 和 Master 的通讯，使用 HTTP 请求可以保证通讯的跨平台性，同时由于可以高效复用现有的模块组件，我们能够保证通讯的可靠性，并且通过报错及时处理通讯传输中遇到的各种问题。

多平台支持方面，由于我们基于 Python 实现 Client 客户端，尽可能减少对高计算量功能的搭载，在客户端上只要实现基本的一些 HTTP 请求发送和接收的功能，可以很好地保障客户端平台的的跨平台特性，保证用户能够在各大操作系统平台上运行我们的程序，并且获得流畅可靠的体验。

四．Master 模块实现说明

4.1 模块组件设计

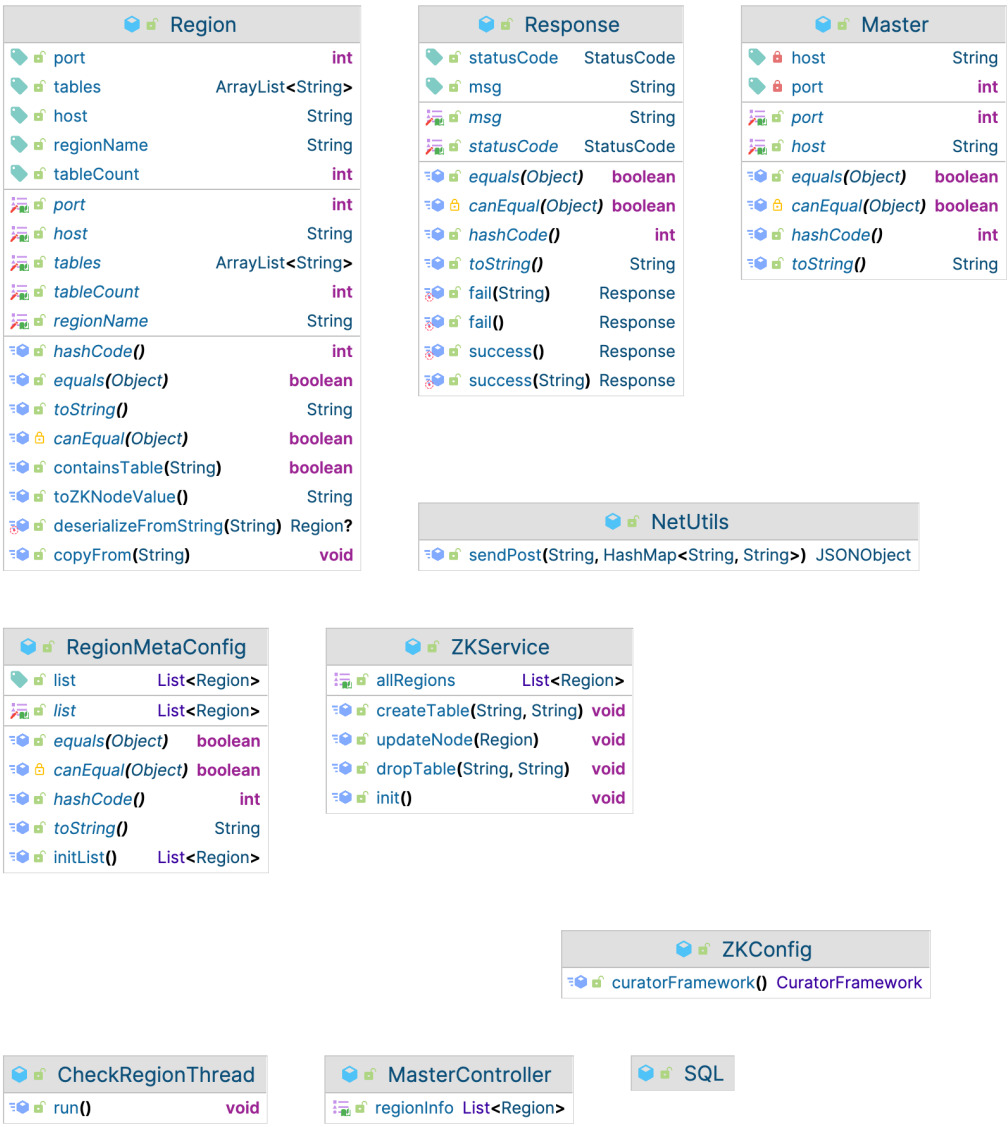
该类下的主要功能类是 ZKService、CheckRegionThread、ZKConfig 和 MasterController 四个。

- 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。

4.2 主要数据结构

4.2.1 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。

4.2.2 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

4.2.3 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

4.2.5 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。

4.2.6 CheckRegionThread 类

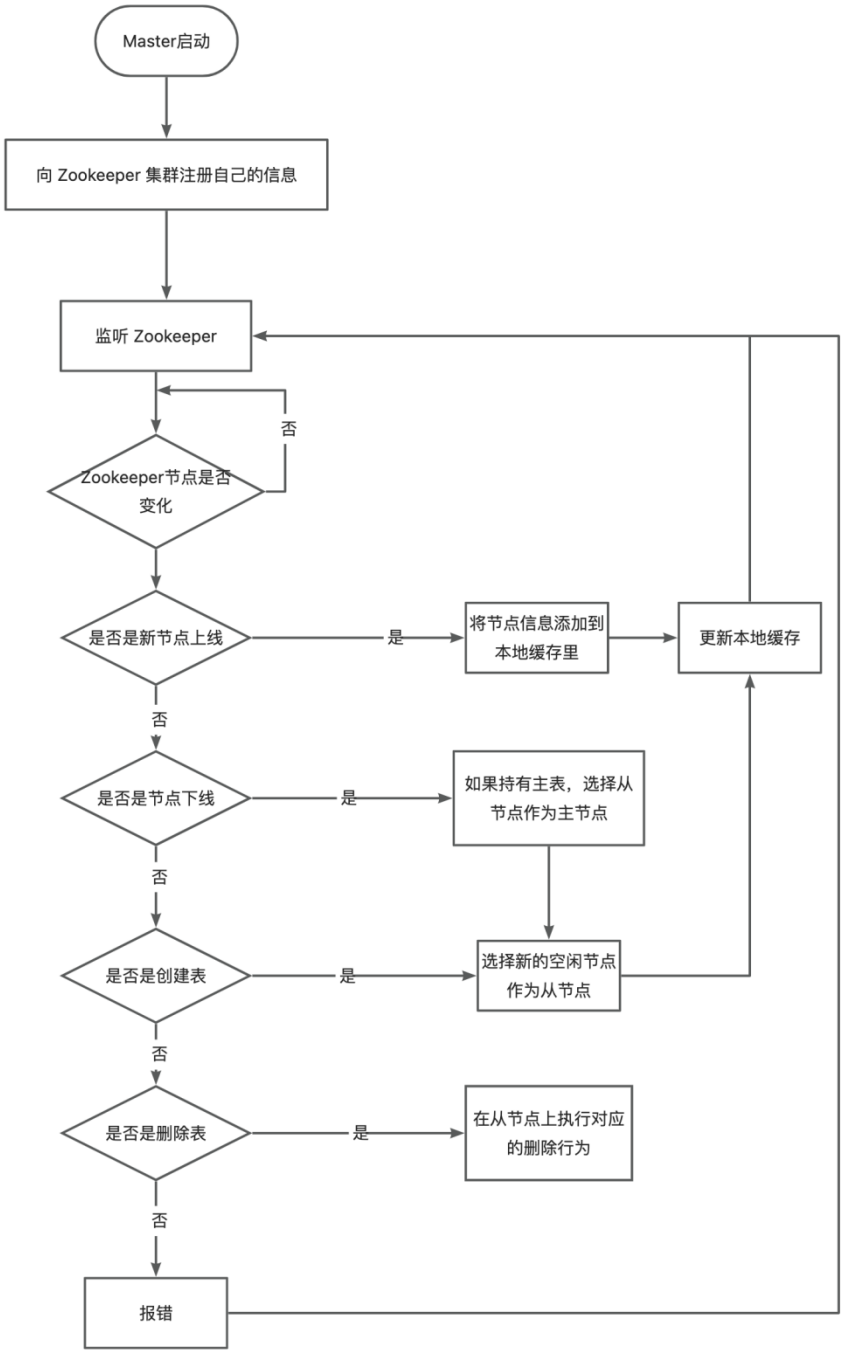
函数名	返回类型	功能说明

Run		定时任务，负责刷新 region 缓存

4.2.7 NetUtils 类

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

4.3 流程图设计

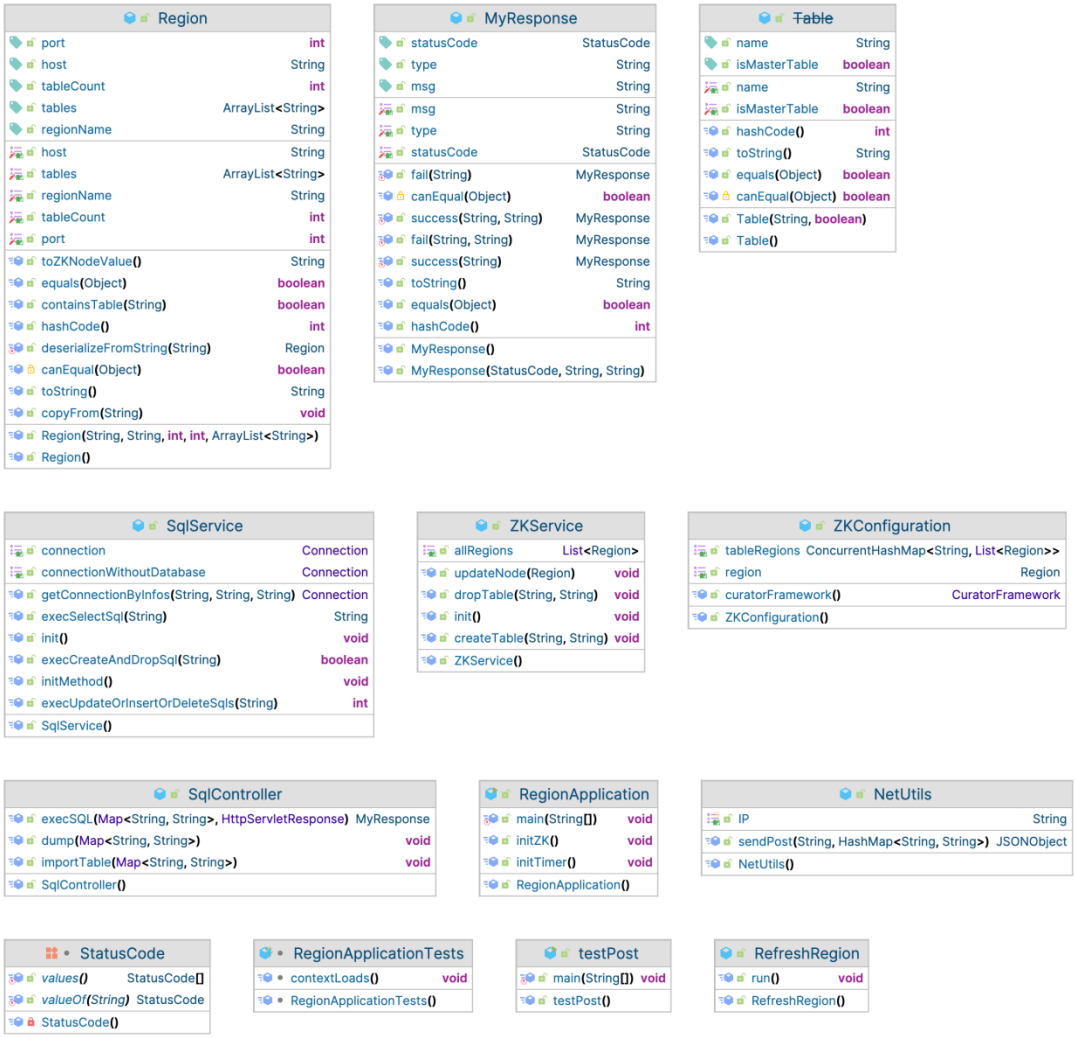


五．Region 模块实现说明

5.1 模块组件设计

5.2 主要数据结构

5.2.1 Region 类图



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。

5.2.2 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

5.2.3 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。

5.3.4 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。

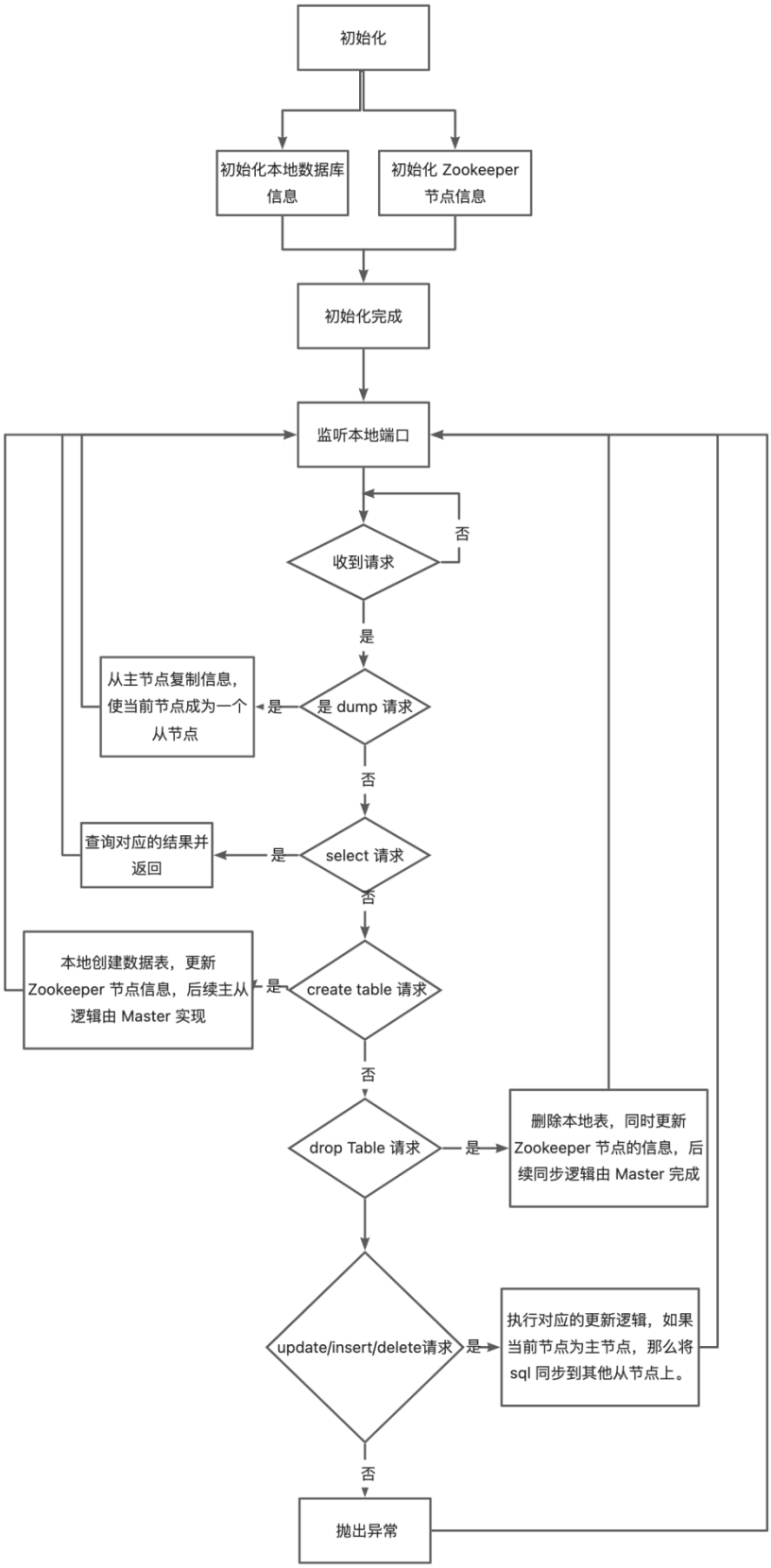
5.3.5 ZKService 类

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

5.3.6 RefreshRegion 类

函数名	返回类型	功能说明
Run		定时任务，刷新本地缓存

5.3 流程图设计


初始化
初始化本地数据库信息
初始化 Zookeeper 节点信息
初始化完成
监听本地端口
收到请求
否
是
是 dump 请求
是
从主节点复制信息，使当前节点成为一个从节点
否
select 请求
是
查询对应的结果并返回
否
create table 请求
是
本地创建数据表，更新 Zookeeper 节点信息，后续主从逻辑由 Master 实现
否
drop Table 请求
是
删除本地表，同时更新 Zookeeper 节点的信息，后续同步逻辑由 Master 完成
update/insert/delete请求
是
执行对应的更新逻辑，如果当前节点为主节点，那么将 sql 同步到其他从节点上。
否
抛出异常

六. 重要功能实现

6.1 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。

6.2 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）。

6.3 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。

七．Client 模块实现说明

7.1 模块组件设计

本模块包括以下几个部分：

（1）Client 类

Client 类负责：

- 维护与 Zookeeper 的连接
- 响应用户输入的 SQL 指令
- 结合 Buffer 信息选择执行指令的 Region

每次启动主程序时将会启动一个 Client 单例。该 Client 实例将会连接指定的 Zookeeper 服务，并开启缓存。该 Client 单例将会启动一个线程安全锁并传递给 Buffer，保证在 Client 与 Buffer 中所有对缓存数据的操作都是线程安全的，从而确保 SQL 信息发送的准确性。

Client 将会根据缓存信息和从用户 SQL 中提取的表信息来确定向哪个 Region 发送 SQL 语句。如果 Client 分析出用户的 SQL 语句是建表命令，则会从缓存信息中提取所有主表信息，并与 Zookeeper 沟通，获取当前所含主表最少的 Region，并向该 Region 发送建表信息来达成负载均衡。

Client 也采取了多种措施以预防出现未知的问题。例如良好的 try-catch、请求发送的超时设置、关键操作时的缓存检查与刷新。

（2）Buffer 类

Buffer 类负责：

- 缓存 Zookeeper 中的 Region List
- 缓存所有 Region 到其 hosts+port 的对应
- 缓存所有主表到其所在 Region 的对应

- 向 Zookeeper 注册监听器，动态监听 Zookeeper 的节点列表
- 向 Zookeeper 注册监听器，动态监听所有 Region 的节点内容

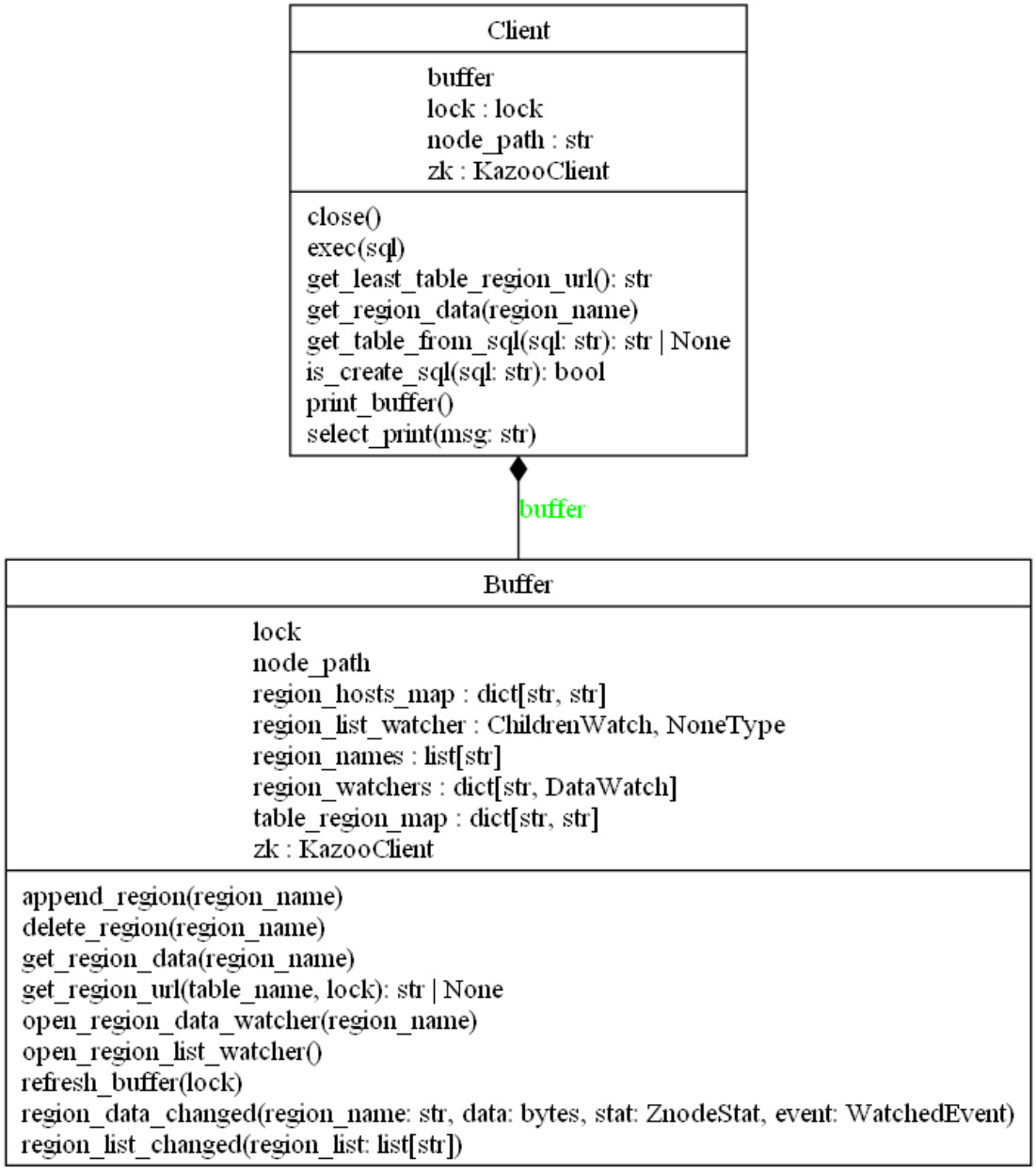
Buffer 将会在 Client 启动时完成自动创建，并接受来自 Client 的线程安全锁。由于涉及到多线程编程，Buffer 中所有对于缓存内容的操作均需要保证线程安全。

在初次启动的时候，Buffer 会向 Zookeeper 请求所需要的数据，并针对整个 Region List 和各个 Region 注册监听器。当 Zookeeper 的 Region 下线或者上线或者节点数据发生改变时，会触发对应的监听器回调函数，对 Buffer 进行部分更新。

同时 Buffer 留出了手动强制刷新的借口，刷新时会清除之前留下的所有数据和监听器，重新进行 Buffer 的初始化。该功能仅用于发生未知错误的情况，正常情况下 Buffer 的所有数据将会自动与 Zookeeper 同步。

7.2 主要数据结构

7.2.1 类图



7.2.2 Client 类

类成员	类型	说明
buffer	Buffer	Client 对应的 Buffer

lock	threading.Lock	线程安全锁
node_path	Str	Zookeeper 中存储数据的节点路径
zk	kazoo.client.KazooClinet	与 Zookeeper 沟通的实例

函数名	返回类型	功能说明
close	None	关闭客户端和 Zookeeper 的链接
exec	None	执行制定的 SQL 语句
get_least_table_region_url	Str	获取主表最少的 Region 的 url
get_region_data	Str	获取指定 region 节点的内容
get_table_from_sql	Str|None	返回从 SQL 中解析出的第一张表
is_create_sql	bool	判断 SQL 是否为 CREATE 语句
print_buffer	None	格式化打印目前的 Buffer 数据
select_print	None	格式化打印 SELECT 语句返回的数据

7.2.3 Buffer 类

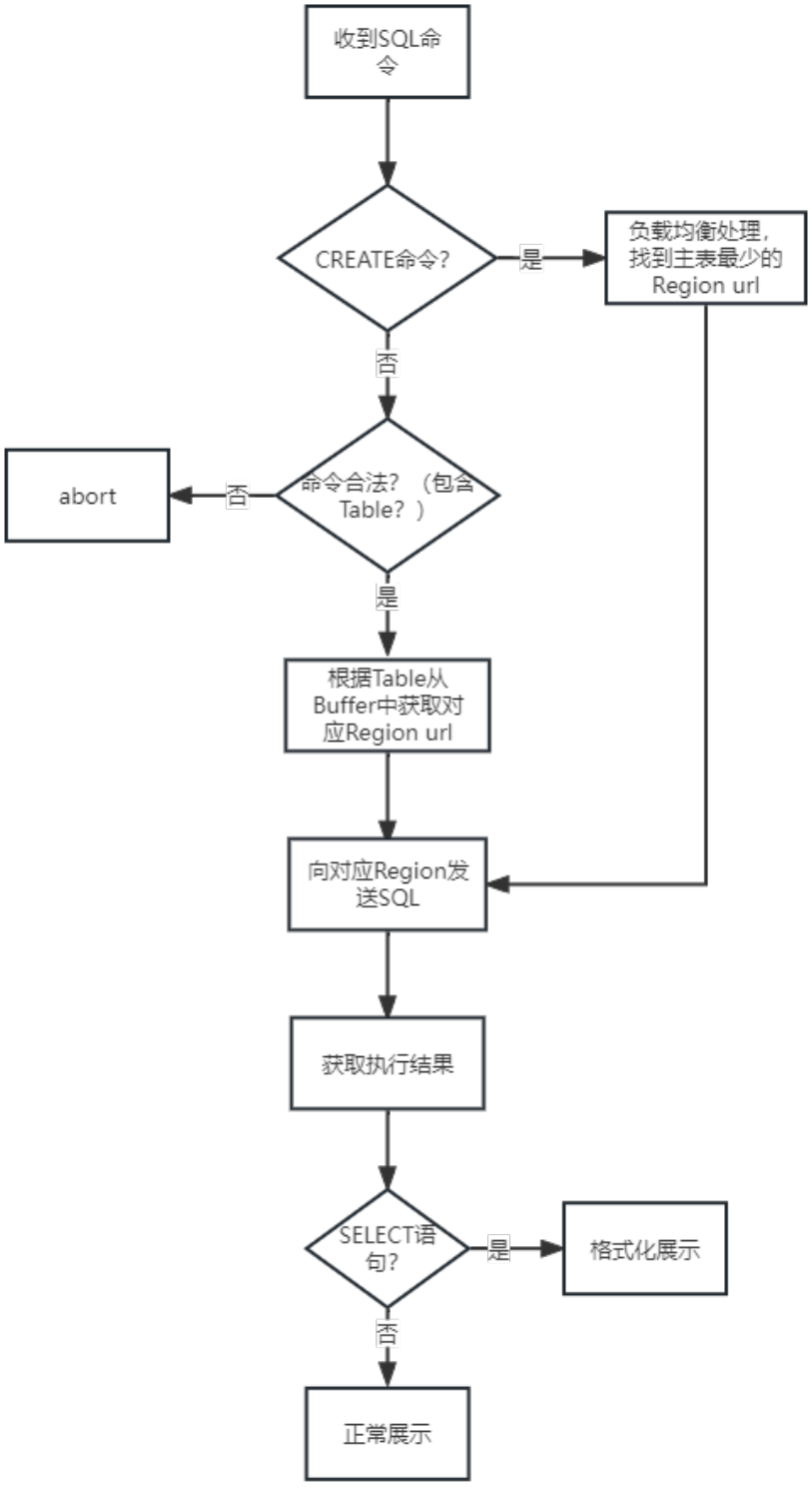
类成员	类型	说明
lock	threading.Lock	线程安全锁
node_path	Str	Zookeeper 中存储数据的节点路径
zk	KazooClinet	与 Zookeeper 沟通的实例
region_host_map	dict[str, str]	储存 region_name -> hosts+port 的映射
region_list_watcher	kazoo.recipe.watchers.ChildrenWatch|NoneType	Region List 的 children_list 的监听器

region_names	list[Str]	储存所有的 Region 节点名称
region_watchers	dict[Str, kazoo.recipe.watchers.DataWatch]	储存 region_name -> region_watcher 的映射
table_region_map	dict[str,str]	储存 table_name -> region_name 的映射
zk	kazoo.client.KazooClinet	与 Zookeeper 沟通的实例

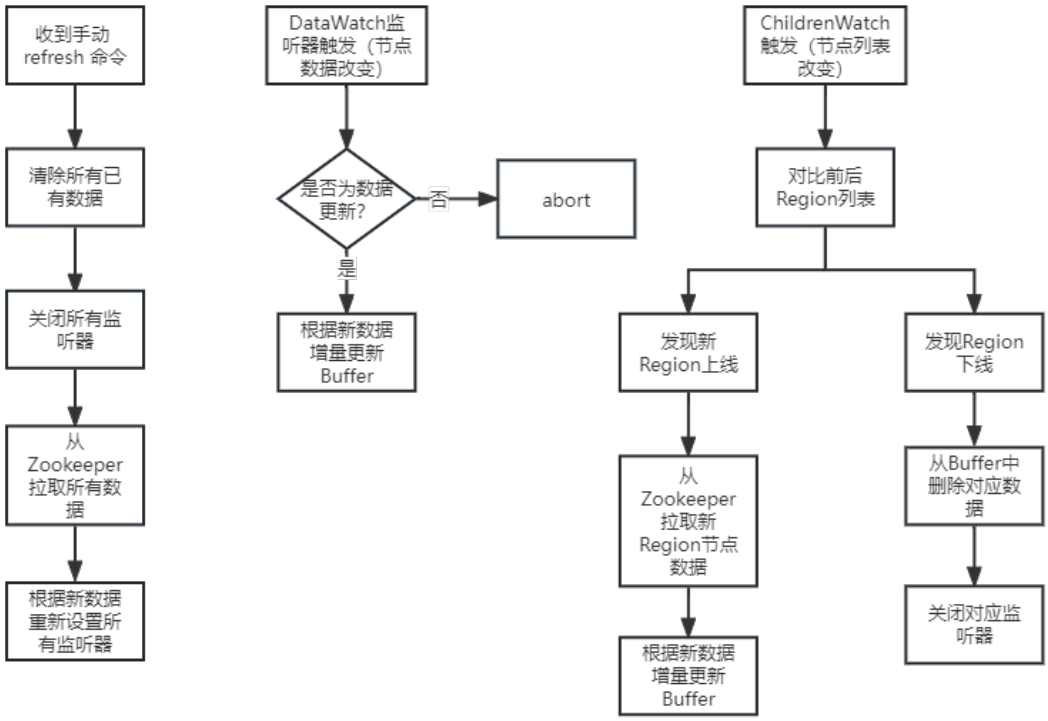
函数名	返回类型	功能说明
append_region	None	在整个缓存中新增一个 Region
delete_region	None	在整个缓存中删除有关该 Region 的所有信息
get_region_data	Str	获取指定 Region 节点的内容
get_region_url	Str	获取指定 Region 的 url
open_region_data_watcher	None	开启指定 Region 的数据监听器
open_region_list_watcher	None	开启 Region List 监听器
refresh_buffer	None	重置整个 buffer
region_data_changed	None	region_data_watcher 的回调函数,只处理 CHANGE 事件,节点被删除等事件由 region_list_watcher 来处理
region_list_changed	None	region_list_watcher 的回调函数,查看是否有节点被删除或新增

7.3 流程图设计

7.3.1 Client 执行 SQL 流程图



7.3.2 Buffer 更新流程图



八．Client (Web)模块实现说明

8.1 模块组件设计

本模块包括以下几个部分：

（1）前端 UI 界面

（2）前端网页的生存周期检测

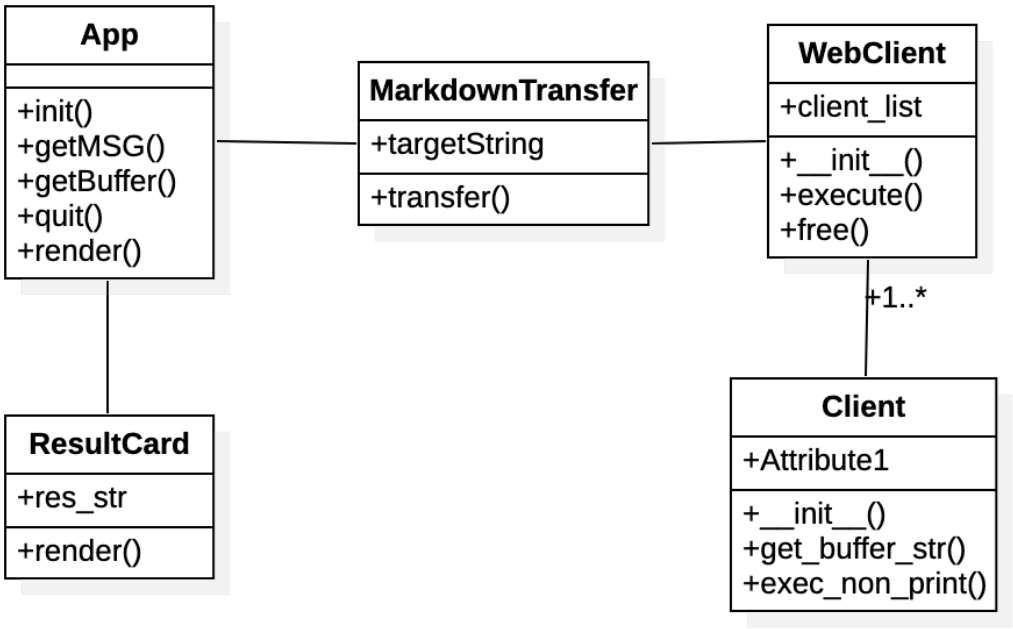
（3）前端查询和连接请求

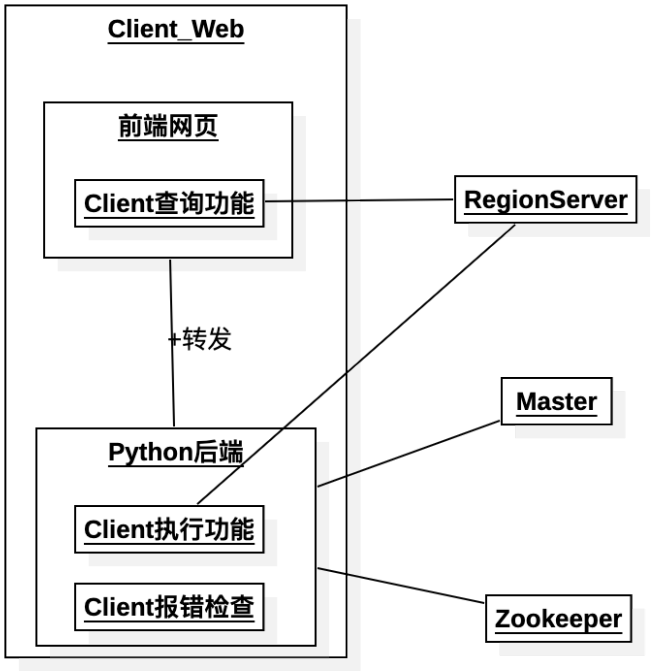
（4）后端对接入请求的响应

（5）后端转发查询结果、报错信息

相比于原有的单机客户端，本客户端在其基础上做了网页版的移植和重写，使得其适应于 BS 架构的分布式查询。

8.2 主要数据结构

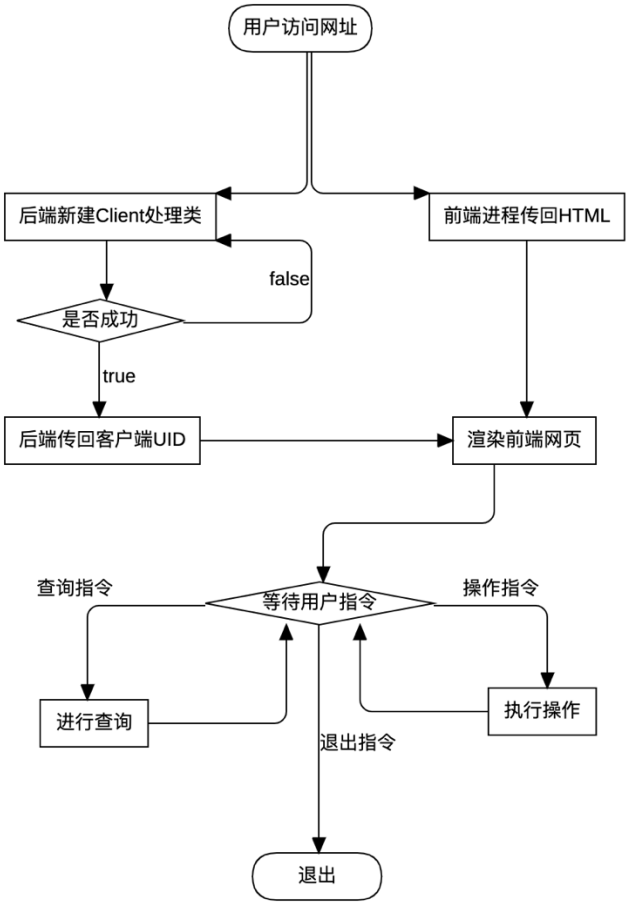




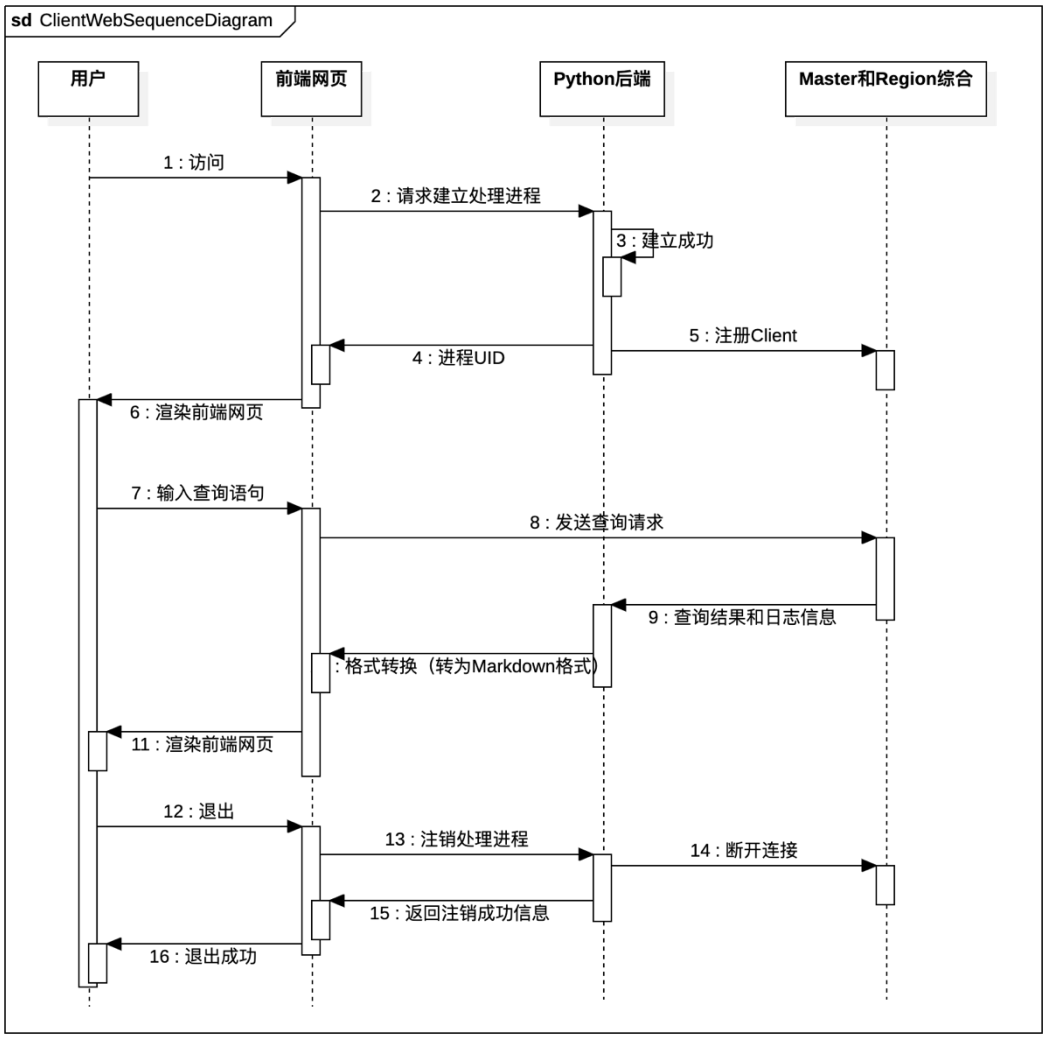
在网页版的客户端中，在用户输入网址请求时，网页 APP 会想 Python 后端进行一个申请和注册，Python 后端会分配相应的处理线程给前端网页，并且代为向 Master 和 zookeeper 进行注册。

对于前端网页获取用户查询和执行的信息，通过后端客户端进行语法检查，客户端将结果返回，随后网页端向对应的 Region 发送执行和请求信息，执行和操作结果返回至前端网页，随后再请求后端进行字符处理，将表格数据转换为 Markdown 格式，使用网页端现成组件渲染为表格。

8.3 流程图设计



时序图：



九. 测试用例设计

9.1 SQL 功能测试

9.1.1 测试用例 1

测试用例 1

描述	考察 int 类型上的等值条件查询
输入信息	select from student2 where id=1080100245;

假设	无异常
输出	返回查询结果（无该条目或所有符合条件的条目）
备注	需要人工比对核实

9.1.2 测试用例 2

测试用例 2

描述	考察 float 类型上的等值条件查询，观察数量
输入信息	select from student2 where score=98.5;
假设	无异常
输出	返回查询结果（无该条目或所有符合条件的条目）
备注	需要人工对比核查

9.1.3 测试用例 3

测试用例 3

描述	考察 float 类型上的等值条件查询，观察数量
输入信息	select from student2 where score=98.5;
假设	无异常
输出	返回查询结果（无该条目或所有符合条件的条目）
备注	需要人工对比核查

9.1.4 测试用例 4

测试用例 4

描述	考察 int 类型上的不等条件查询，观察数量
输入信息	select from student2 where id<>1080109998;
假设	无异常
输出	返回查询结果（无该条目或所有符合条件的条目）
备注	需要人工对比核查

9.1.5 测试用例 5

测试用例 5

描述	考察 char 类型上的不等条件查询，观察数量
输入信息	select from student2 where name<>'name9998';
假设	无异常
输出	返回查询结果（无该条目或所有符合条件的条目）
备注	需要人工对比核查

9.1.6 测试用例 6

测试用例 6

描述	考察 char 类型上的不等条件查询，观察数量
输入信息	select from student2 where name<>'name9998';
假设	无异常
输出	返回查询结果（无该条目或所有符合条件的条目）
备注	需要人工对比核查

9.1.7 测试用例 7

测试用例 7

描述	考察 primary key 约束冲突（或 unique 约束冲突）
输入信息	insert into student2 values(1080100245,'name245',100);
假设	无异常
输出	报 primary key 约束冲突
备注	需要人工对比核查

9.1.8 测试用例 8

测试用例 8

描述	考察索引的创建，unique key 才能建立索引

输入信息	create index stuidx on student2 (name);
假设	无异常
输出	如果 name 是 unique，则创建成功
备注	需要人工对比核查，需要同时对单机存储数据库和分布式数据库进行检查

9.1.9 测试用例 9

测试用例 9

描述	考察索引创建后再插入数据，索引是否有正确插入数据
输入信息	insert into student2 values(1080197996,'name97996',100);
假设	无异常
输出	返回插入结果
备注	需要人工对比核查，需要同时对单机存储数据库和分布式数据库进行检查

9.1.10 测试用例 10

测试用例 10

描述	考察索引创建后再删除数据，索引是否有正确删除数据
输入信息	delete from student2 where name='name97996';
假设	无异常
输出	返回删除结果
备注	需要人工对比核查，需要同时对单机存储数据库和分布式数据库进行检查

9.2 集群管理

集群管理是分布式数据库系统中非常重要的一部分，它涉及到系统的稳定性和可靠性。为了确保分布式数据库系统的集群管理能力，需要进行充分的测试。下面是一些可能的测试用例设计，用于测试分布式数据库系统的集群管理部分：

1. 节点加入和退出测试：测试系统能否正确处理节点加入和退出的情况，并保持系统的稳定性和可靠性。

2. 节点故障测试：模拟某个节点发生故障的情况，检查系统是否能够正确识别和处理节点故障，并从故障中恢复。

3. 节点负载均衡测试：测试系统的负载均衡算法是否正常，能否均衡地分配负载。

4. 节点状态监测测试：测试系统的节点状态监测功能是否正常，能否及时检测到节点的状态变化。

5. 节点间通信测试：测试系统的节点间通信功能是否正常，包括数据传输、数据同步等。

6. 系统可扩展性测试：测试系统的可扩展性，包括节点数量、数据存储容量、并发访问量等。

7. 性能测试：测试系统的性能指标，包括响应时间、吞吐量、并发数等。

8. 安全测试：测试系统的安全性，包括防止数据泄露、攻击和入侵的能力。

需要注意的是，这些测试用例只是一个示例，实际的测试用例需要根据后续具体的开发进度和开发成果进行适当的修改和再设计。同时，在进行集群管理测试时，应该考虑到系统的复杂性和不可预测性，尽可能地完整模拟可能遇到的场景，确保系统的稳定性和可靠性。

9.2.1 集群管理测试用例 1

测试用例 1

描述	节点加入和退出测试
输入信息	Start Client (or Region Server)
假设	无异常

输出	节点元数据增加新增节点的数据
备注	无

9.2.2 集群管理测试用例 2

测试用例 2

描述	节点故障测试
输入信息	Stop -f Client (or Region Server)
假设	无异常
输出	节点元数据信息更新，相关节点显示已经断联
备注	无

9.3 容错容灾

容灾容错是分布式系统中非常重要的一部分，它涉及到系统的可靠性和稳定性。为了确保分布式数据库系统的容灾容错能力，需要进行充分的测试。测试用例设计如下，用于测试分布式数据库系统的容灾容错部分：

1. 网络断开测试：模拟网络断开的情况，检查系统能否正确处理和恢复。

2. 节点故障测试：模拟某个节点发生故障的情况，检查系统是否能够正确识别和处理节点故障，并从故障中恢复。

3. 数据一致性测试：模拟数据同步的情况，检查系统能否保持数据一致性。

4. 备份和恢复测试：测试备份和恢复系统的功能是否正常。

5. 负载均衡测试：测试负载均衡系统的功能是否正常，包括负载均衡算法的正确性和负载均衡效果的好坏。

6. 容量测试：测试系统的容量上限，包括节点数、数据存储量、并发访问量等。

7. 性能测试：测试系统的性能指标，包括响应时间、吞吐量、并发数等。

8. 安全测试：测试系统的安全性，包括防止数据泄露、攻击和入侵的能力。

9.3.1 容错容灾测试用例 1

测试用例 1

描述	节点故障测试
输入信息	Stop -f Region Server
假设	无异常
输出	该断联节点数据表发生迁移，并且保证数据表备份数量
备注	无

9.3.2 容错容灾测试用例 2

测试用例 2

描述	数据一致性测试
输入信息	Insert into student2(id = Random())
假设	无异常
输出	插入成果
备注	应当人工检查各个备份表插入的随机数数值是否相等

9.4 负载均衡

在分布式数据库系统中，负载均衡部分的测试用例设计需要考虑以下几个方面：

1. 负载均衡策略的测试：测试负载均衡策略在不同负载下的表现，例如在高负载时是否能够正确地分配请求到不同的节点上，从而保证系统的可扩展性和性能。

2. 节点失效和故障转移的测试：测试系统在节点失效或出现故障时的表现，例如测试系统是否能够及时检测到节点的失效并将请求转移到其他节点上，从而保证系统的可靠性和容错性。

3. 负载监控和调整的测试：测试负载监控和调整的功能是否正常工作，例如测试系统是否能够及时监测负载变化并作出相应的调整，从而保证系统的负载均

衡能力。

4. 配置管理的测试：测试系统的配置管理功能是否正常工作，例如测试系统是否能够正确地读取和解析配置文件，并根据配置文件中的参数进行负载均衡策略的调整。

5. 性能测试：测试负载均衡部分的性能表现，例如测试系统的响应时间、吞吐量等指标是否满足需求，并发现性能瓶颈和优化点。

总之，测试用例需要从负载均衡的各个方面出发，考虑各种可能的情况和异常情况，并根据实际开发情况和需求设计相应的测试用例。同时，需要结合自动化测试和手动测试，保证测试的全面性和有效性

9.4.1 负载均衡测试用例 1

测试用例 1

描述	节点负载均衡测试
输入信息	（大量增加对于某个节点所拥有的数据表的访问）
假设	无异常
输出	该节点数据表发生迁移，各节点数据表访问情况接近
备注	无

9.5 客户端

分布式数据库中客户端部分的测试用例设计应该包括以下几个方面：

1. 输入验证：测试应该确保客户端输入的 SQL 语句是有效的，不能包含任何不良数据，例如 SQL 注入攻击。

2. 功能测试：测试应该验证客户端是否能够成功连接到分布式数据库，并能够执行基本操作，例如创建/删除表，插入/删除/查询数据等。

3. 负载测试：测试应该模拟多个客户端同时连接到分布式数据库并执行各种操作，以验证系统的负载能力。

4. 失败测试：测试应该模拟各种故障场景，例如断电、网络故障等，以验证系统的容错性。

5. 安全测试：测试应该验证客户端的身份验证和访问控制机制是否有效，以保证数据的安全性和保密性。

6. 兼容性测试：测试应该确保客户端能够与不同类型的数据库服务器兼容，并能够正确地解析和执行 SQL 语句。

在设计测试用例时，需要考虑以上因素，并确保测试用例覆盖尽可能多的场景和情况，以保证分布式数据库系统的稳定性和可靠性。

9.5.1 客户端测试用例 1

测试用例 1

描述	客户端加入和退出测试
输入信息	Start Client
假设	无异常
输出	节点元数据增加新增节点的数据
备注	无

9.5.2 客户端测试用例 2

测试用例 2

描述	节点故障测试
输入信息	Stop -f Client
假设	无异常
输出	节点元数据信息更新，相关节点显示已经断联
备注	无

参考文献

[1] 《大规模信息系统构建导论》课程 PPT（大规模信息系统基础知识 I-VIII）

[2] Designing Data-Intensive Applications The Big Ideas Behind Reliable, Scalable, and

Maintainable Systems”

[3] "Spanner: Google's Globally-Distributed Database" by Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J., ... and Wilkins, R. (2012)

[4] "Apache Kafka: A Distributed Streaming Platform" by Kreps, J. (2011)

[5] React 官方文档 https://react.docschina.org

[6] Flask 官方中文文档 https://flask.net.cn

[7] https://kazoo.readthedocs.io/en/latest/

[8]https://stackoverflow.com/questions/40153340/how-to-stop-datawatch-on-zookeeper-node-using-kazoo

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

[10] https://spring.io

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