开发心得与小结

1 收获

为设计本小组分布式 MiniSQL 的架构爬回去把所有的 PPT 重新过了一遍，参考了若干个现有解决方案的架构，最后发现自己大概率实现不了，只搓了一个最简单的框架出来。但多少对于主流的框架和协议有了更加深刻的认识，并充分认识到自己技术力的不足。

数据一致性保障与存储/工作负载均衡策略设计也导致了长时间的沉思——通过转发 SQL 语句进行同步操作会显著受网络质量影响，错误与丢包都会导致同步失败，或因 SQL 操作本身的非幂等性导致实际数据的不一致（虽然在结果返回阶段通过投票机制保证了返回结果的一致性，但也导致了轮询查询 Slave 节点以分摊压力策略的失效）。但手搓 log 信息又超出了能力范围，最后就选择性忽视了本需求。

此外本次开发过程也加深了本人对于 Java 语言和 SpringBoot 框架使用的熟练度。对于在入口类 main() 函数中无法正常访问成员变量、退出时无法正确关闭 ZooKeeper 会话、工具类无法正常注入使用 yml 配置的 Database 等问题探索了许多新修饰器的使用，收获颇丰。

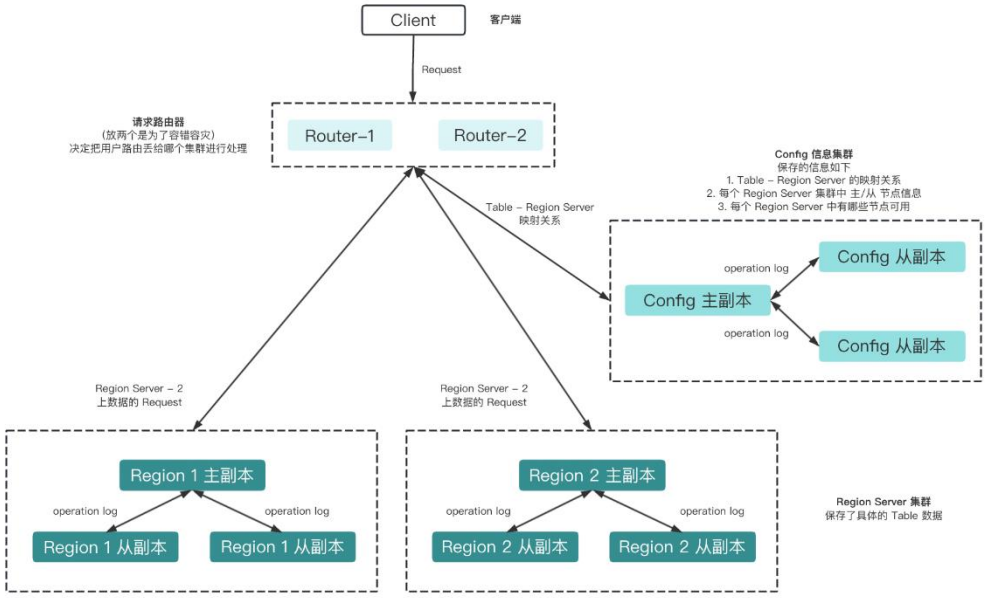
2 不足与反思

本次大规模数据库系统基本覆盖了作业的最低要求，但由于部分策略过于 Naive，实际运行效率并不理想。另外由于 SpringBoot 自动注入的 Controller 无

法很好的通过构造函数传递和访问共享变量，导致代码出现了高耦合性。

架构设计

Master 项目本身仍存在单点失效的问题 —— 由于整个系统的正确运行高度依赖于 Master 提供的请求路由功能，Master 的故障将导致整个系统失效（即便所有 Region Server 均正常工作），这无疑是一个很尴尬的局面。



MongoDB 中的分布式架构示意图

在 Mongo DB 分布式策略中，这一问题通过设置多个 Router（Master）得到解决。但由于在本项目中 Client 并不直接与 ZooKeeper Server 进行交互，因而无从动态获取当前可用的 Master 地址，这一方案最终被废弃。

- 由于本项目最终基于 B/S 架构实现，前端项目代理地址必须在 config 文件中预先配置（单一地址），最后并没有实现多 Master（Router）架构。

数据同步

在技术选型阶段，我们计划通过 MySQL Replication 通过 binlog 信息实现数据的主从备份。但由于在使用 JDBC 实现动态主从副本配置的过程中遇到了较大

困难，最终采用主节点主动向从节点转发 SQL 语句的方案进行实现。

这一 Naïve 的策略不仅加重了主节点的操作负担，同时可能由于 SQL 操作本身的非幂等性导致数据不一致。

多线程

由于各组员对于在 SpringBoot 项目中引入线程池的操作并不熟悉，最终 Master 项目与 Region Server 项目为单线程操作，这在响应高并发操作时显然时即为不利的。Master 项目本身需要定时统计各 Region 访问量以确定热点，而 Region Server 在使用 JDBC 操作数据库时也可能因为线程阻塞导致无法处理异常响应。

前端渲染效率与数据格式

目前 Client 端会将 Select 的全部结果一次性渲染到页面中，这在数据量较大时会导致较差的用户使用体验，若有计划进行迭代最好能手动添加分页查询限制或进行分屏渲染。

此外，为符合 ElementUI 表格组件的数据渲染格式，后端手动将每一条 Record 的每一个字段替换为 Colname-Vaue 的键值对，这显著增加了交换数据的体积，并增加了后端项目的工作量，计划在之后重新实现表单组件以支持更加简洁的数据交换格式。

分布式数据存储

系统目前以 Table 为单位进行分布式存储，底层采用了 MySQL，一定程度上保证了数据存储的可靠性，但在处理非结构化的大量数据时仍存在不足。