分布式数据库项目

——大规模信息系统构建技术导论课程项目

总体设计报告

指导教师：鲍凌峰

组长：马良

组员：王伊斓、罗依民

日期：2023 年 5 月 21 日

1 引言

1.1 系统目标

本项目是《大规模信息系统构建技术导论》的课程项目。项目内容为：在大二春夏学期《数据库系统》课程的基础上，结合《大规模信息系统构建技术导论》所学知识，实现一个分布式关系型简易数据库系统，实现数据分布、集群管理、分布式查询、副本管理、容错容灾、负载均衡等功能。

1.2 设计说明与任务分工

本系统由小组 3 位成员合作编写完成，使用 Java 开发，VScode 作为集成开发环境，Github 进行合作开发，每个组员都完成了目标任务，具体的分工安排如下：

表格 1 成员信息及分工

成员姓名	学号	分工职责
马良	3200105143	Master Server、Client、Region Server 之间的 Socket 通讯 Client、Master Server 代码 最终程序的集成与功能调试 视频录制
罗依民	3200105307	Region Server 代码 最终程序的集成与功能调试 视频录制
王伊斓	3200105318	Master Server 中的 Zookeeper 部分，负载均衡，FTP 操作 报告撰写、汇报 PPT 制作

2 系统设计与实现

2.1 系统的总体架构

本系统的总体架构设计如下图所示：

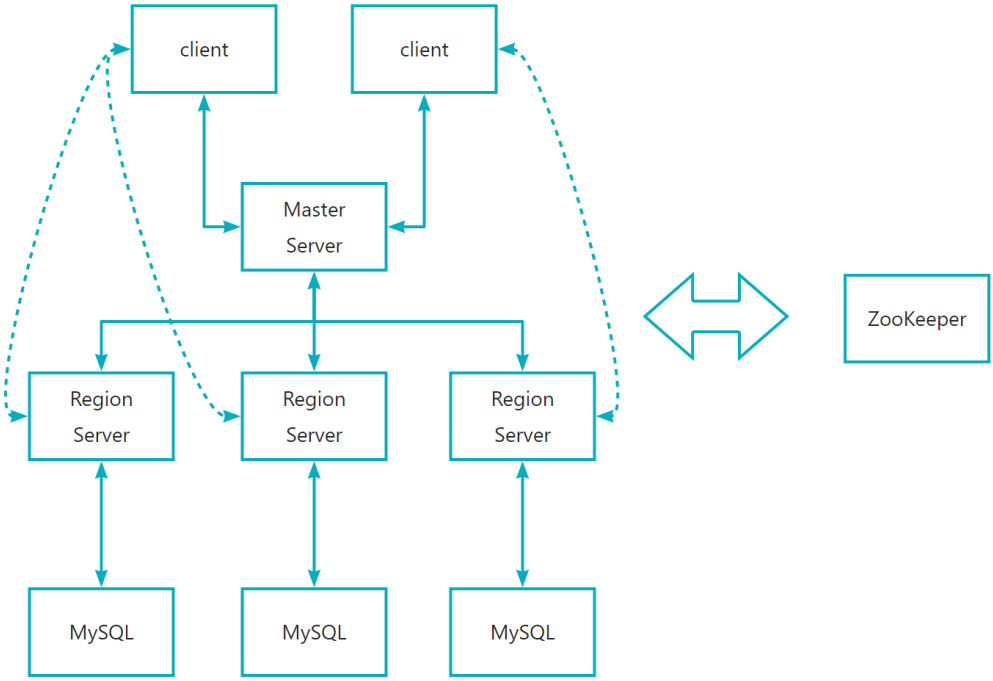


图 1 系统总体架构图

系统主要分为三个部分：Client 客户端、Master Server 主节点、Region Server 从节点，三个部分之间通过 Socket 进行两两通信，而主节点对于从节点的状态监听则是通过 Zookeeper 进行管理。

2.2 Client 设计

Client 模块是分布式数据库系统中直接与用户进行交互的模块，主要功能为：

1. 接收用户输入的 SQL 指令；
2. 执行 SQL 指令，获取语句执行结果，将执行结果呈现给用户。

具体执行方式：

创建数据表：向 Master Server 节点询问是否存在同名数据表，如不存在，则 Master

Server 返回适合创建该表的最佳 Region Server IP 地址；

对表的查询操作：向 Master Server 节点询问数据表存在的 Region Server 地址后，与对应 Region Server 建立 Socket 通讯连接，接收由 Region Server 处理语句后发送回 Client 的消息。

2.2.1 架构设计

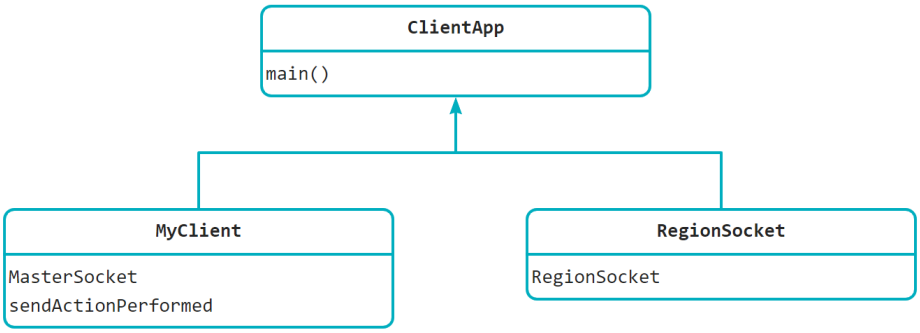


图 2 Client 模块架构设计

Client 的架构设计如图 2 所示，ClientApp 负责 Client 窗口的启动，MyClient 负责调用 MasterSocket 与 RegionSocket 来实现 Client 与 Master、Region 服务器的连接，另外还负责将 Client 端的输入指令传输给 Master 和 Region，RegionSocket 则主要包含了与 Region Server 进行 Socket 连接的操作。

2.2.2 工作流程

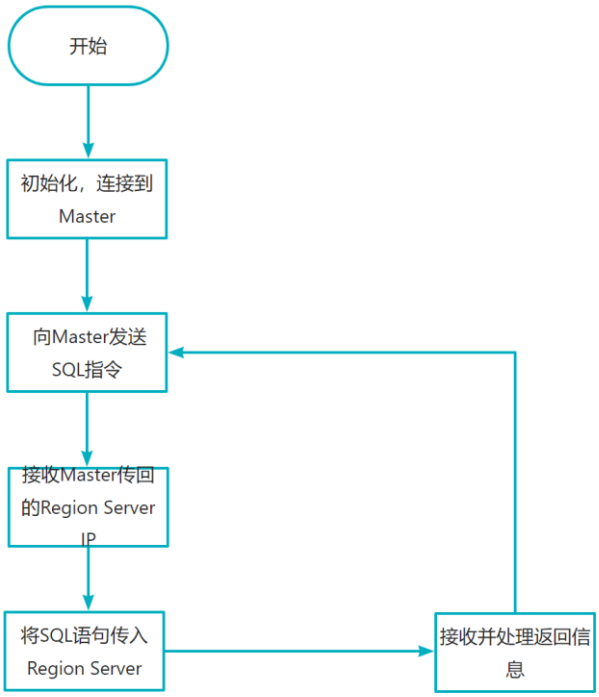


图 3 Client 模块流程图

2.3 Master Server 设计

Master Server 主服务器主要负责进行信息管理、信息接收、负载均衡、容错容灾等功能的实现。其中信息管理指的是使用 zookeeper 监听管理 Region 节点，从节点上线目录，管理结点与对应 ip、对应表信息。

信息接收指的是：接收 client 查询、改动表信息，并返回对应的 Region Server 地址，接收 region 表目录变动信息。

负载均衡指的是：接收 client 创建表的指令信息，根据当前各节点的使用情况，返回合适的 Region Server ip。

容错容灾指的是：监听 Region Server 消息，在其出现异常情况时，在当前服务器中找到最佳的服务器并发送指令让对应 Region 进行数据恢复。

2.3.1 架构设计

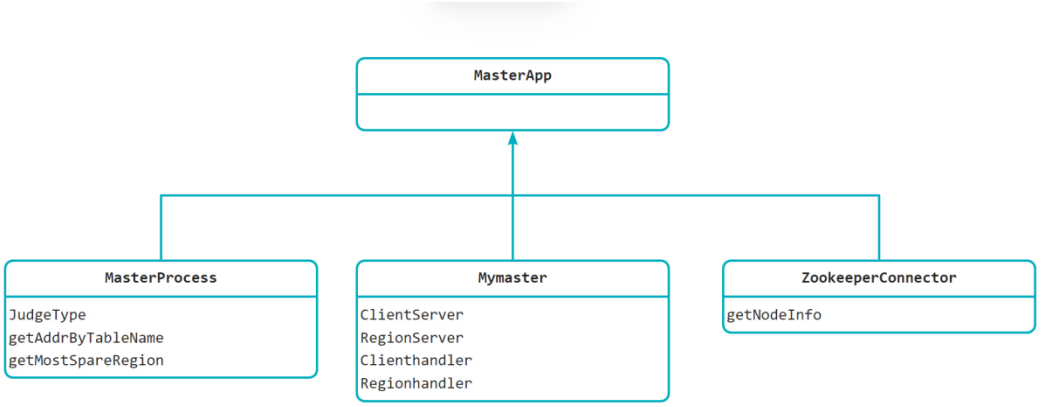


图 4 Master Server 架构设计图

Master Server 的架构设计如上所示，其中 MasterProcess 主要负责解析 Client 传入的 SQL 语句，提取其中的表名，利用 GetAddrByTableName 在缓存表中找到该表对应的 Region Server，如表不存在，则利用 getMostSpareRegion 实现负载均衡，找到目前最空闲的 Region Server 并将其 IP 返回给 Client；Mymaster 中的 ClientServer 和 RegionServer 分别负责对客户端和从节点服务器的监听，并对监听消息应用 Clienthandler 和 Regionhandler 来处理信息；ZookeeperConnector 负责监听各个 RegionServer 的连接状态和所属表的列表。

2.3.2 工作流程

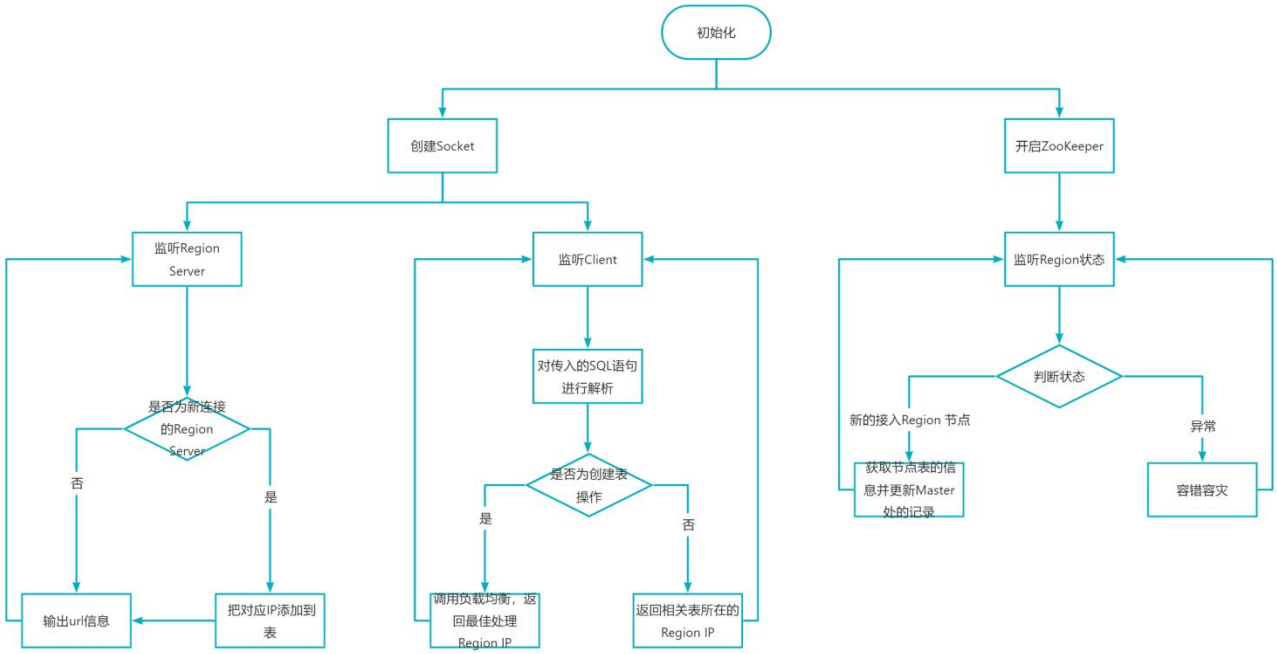


图 5 Master Server 工作流程

2.4 Region Server 设计

Region Server 主要负责实现日志管理、通过 MySQL 处理 client 发来的 sql 语句，副本备份等功能。在实际使用中，Region Server 监听端口，并在收到信息后的处理相关事项。

2.4.1 架构设计

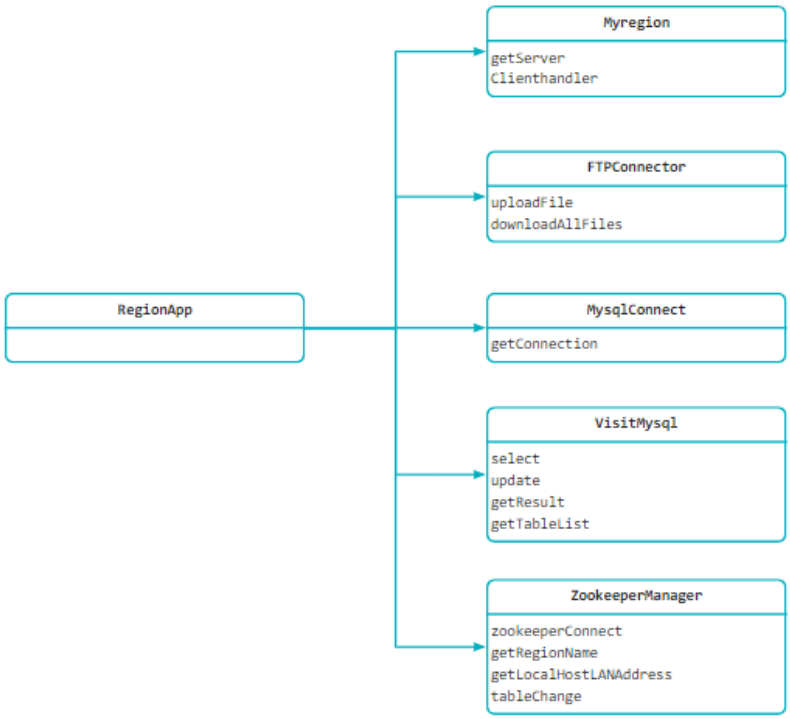


图 6 Region Server 架构设计图

Region Server 架构设计如图 6，其中，RegionApp 负责调用其子模块以启动 Region Server，MyRegion 负责监听客户端的连接以及接收客户端发出的信息；FTPConnector 负责 Region Server 与 FTP 的连接，以及上传和下载文件的操作；MysqlConnect 负责连接本机 MySQL；

VisitMysql 中，select 为执行查询操作，update 以执行非查询语句，getResult 负责解析 sql 语句执行结果，getTableList 为获取数据库中的表；ZookeeperManager 负责了与 Master Server 处的 Zookeeper 建立连接，并向 Master 报告自己的 IP 与表信息。

2.4.2 工作流程

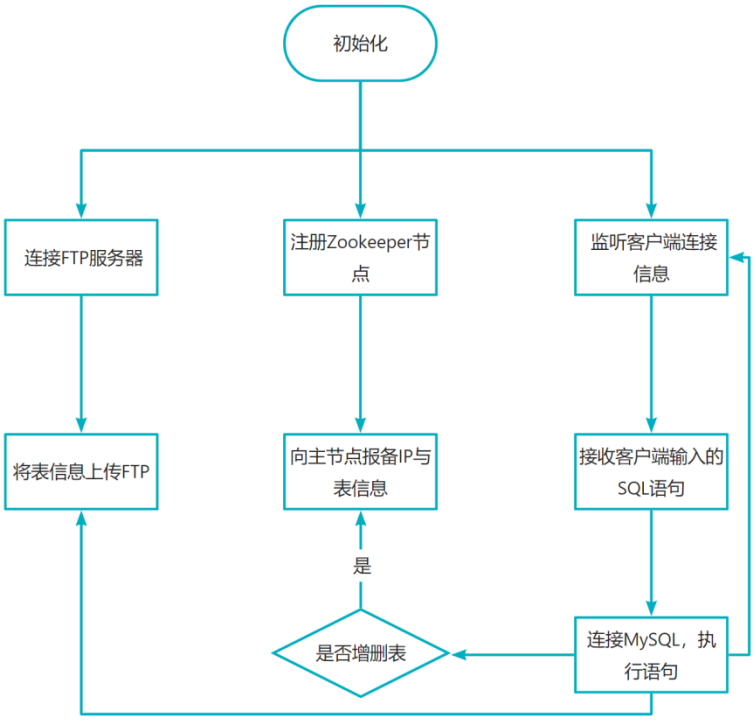


图 7 Region Server 工作流程

2.5 数据库部分设计

分布式数据库的 Region 模块中，需要对由客户端传来的 SQL 语句进行解析处理，由于数据库部分不是本次实验的重点，因此在这里我们直接调用了 Region Server 本机的 MySQL，来进行对 SQL 语句的处理，并返回执行结果。

3 核心功能模块的设计与实现

3.1 Socket 通信架构

3.1.1 架构设计

本系统实现了 Client，Master Server 和 Region Server 之间的两两通信，各部分通过 Socket 来建立稳定可靠的通信和异步的 IO 流，通信采用自定义的协议进行必要信息的传输，保证系统的可靠、准确，系统的通信架构如图所示:

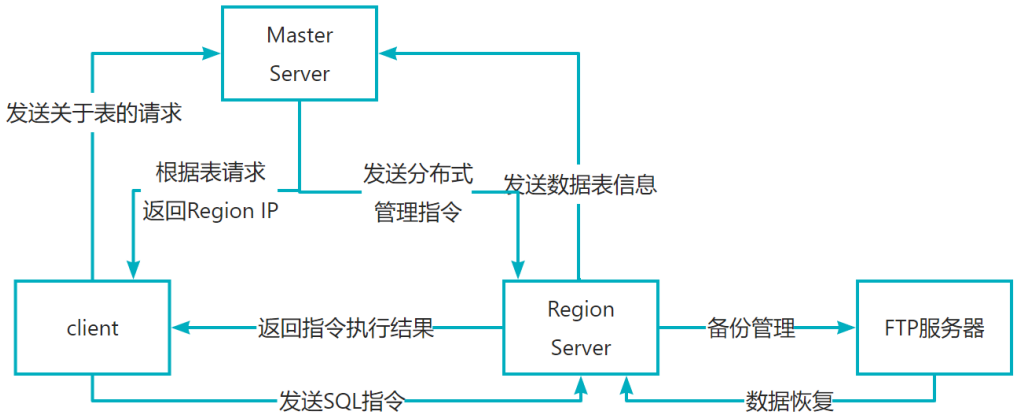


图 8 通讯架构设计

连接采用监听机制，Master 和 Region 的连接始终存在，Client 和 Master 的通信在每次请求执行 SQL 语句后得到连接上的 Region 后就断开，并与对应的 Region 建立连接，执行完 SQL 语句后断开，总体而言，三种不同的模块之间的通信都是全双工的。

3.2 数据分布与集群管理

本项目中，数据以表的单位进行存储，每个表分属于不同的 Region Server。每次客户端发送新建表指令时，Master Server 会寻找当前最空闲的 Region 节点，之后进行建表操作。当每个 Region 节点工作时，需要在 Master Server 完成注册，获得本机编号。Master Server 通过 ZooKeeper 进行集群管理时，能够监听到 Region 的增加或失效，并维护 Region 对应的表信息。

3.3 负载均衡

本系统中实现了创建表时的负载均衡。当客户端需要创建一个新的数据表时，主节点会选择表数量最少的节点进行创建。容错容灾时选择替代服务器时的思路与之一致。

3.4 副本管理

本系统对副本管理功能的设计思路为，执行对表的内容进行增删改查操作成功之后，并通过 FTP 的方式，上传到 FTP 服务器中与 Region 相对应的文件夹下，使得更新后的数据得到备份。

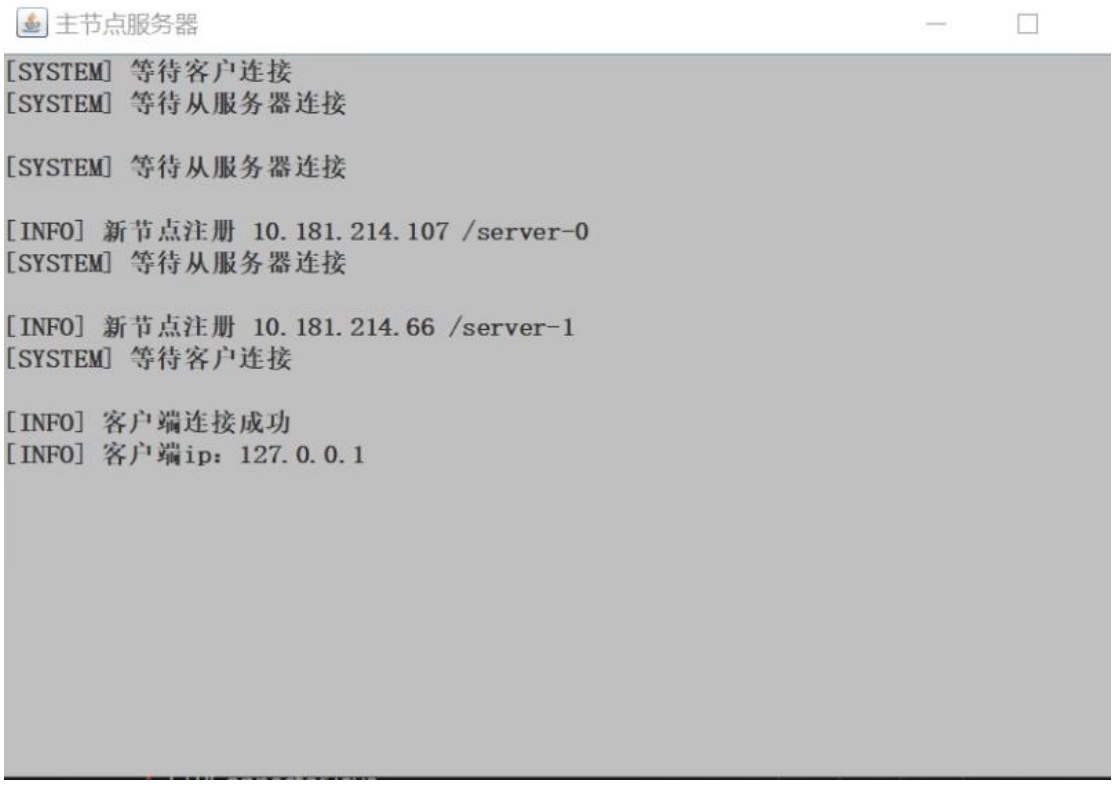
4 系统测试

在项目的测试部分，我们使用了 2 台计算机，一共运行了 1 个 Master，2 个 Region 和 1 个 Client，最终的测试截图如下

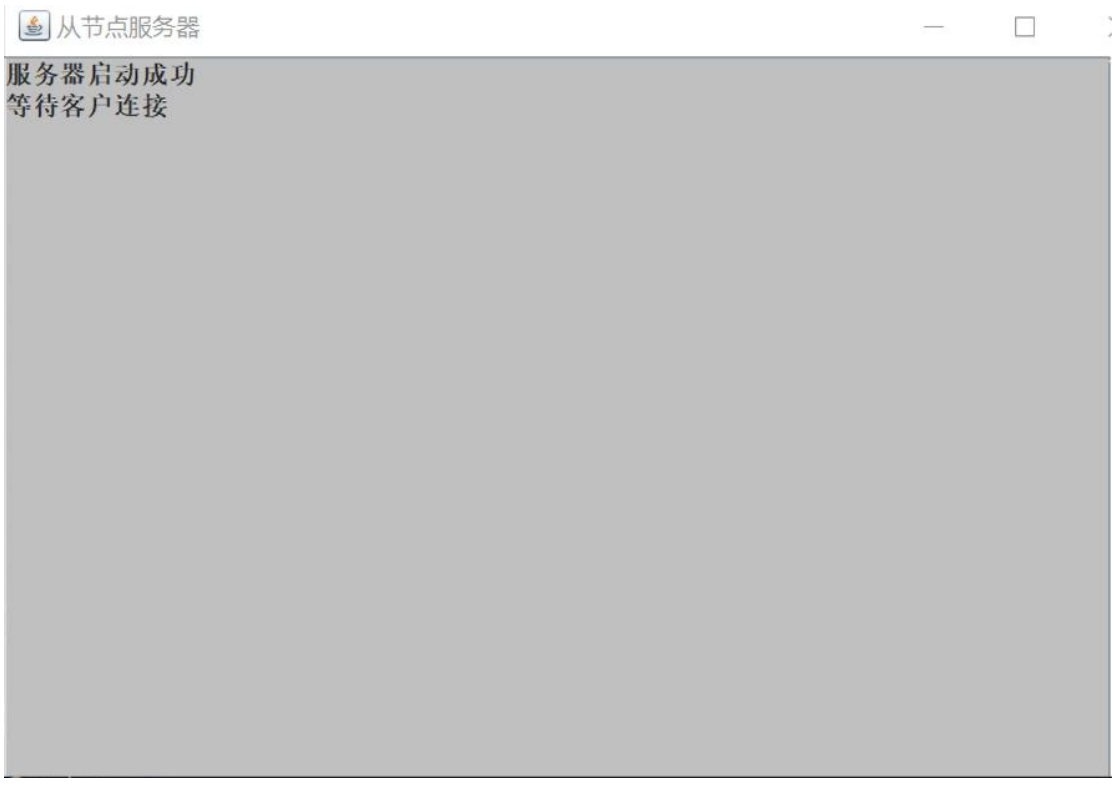
4.1 初始化

当运行所有准备好 Master，Region 和 Client 时，整个系统运行初始化的结果是：

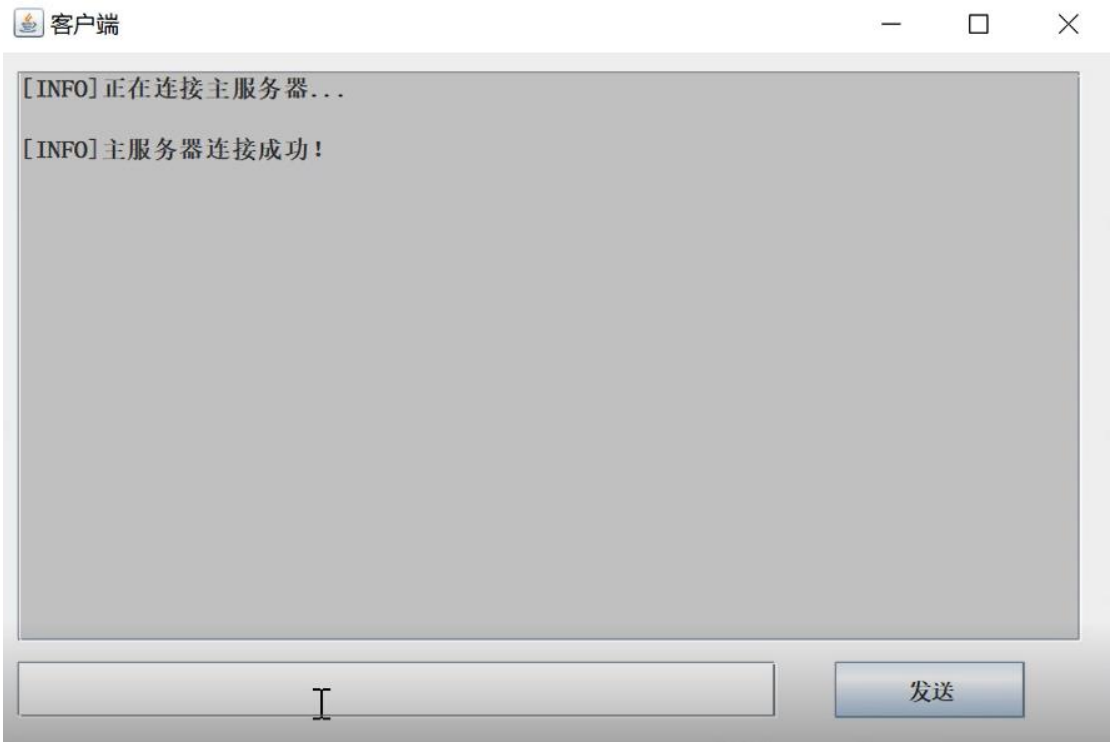
- 主节点：


主节点服务器
[SYSTEM] 等待客户连接
[SYSTEM] 等待从服务器连接
[SYSTEM] 等待从服务器连接
[INFO] 新节点注册 10.181.214.107 /server-0
[SYSTEM] 等待从服务器连接
[INFO] 新节点注册 10.181.214.66 /server-1
[SYSTEM] 等待客户连接
[INFO] 客户端连接成功
[INFO] 客户端ip：127.0.0.1

- 从节点:


从节点服务器
服务器启动成功
等待客户连接

- 客户端:



4.2 创建表

```
发送的sql语句为：create table test1 (id int, name varchar(20))
从节点IP地址为：

[INFO]正在连接从服务器...
[INFO]从服务器连接成功!

操作成功，执行结果如下：
本次操作改变了0行
```

4.3 插入数据

```
发送的sql语句为：insert into test1 values(1,"xxx"),(2,"lym"),(3,"roy")
从节点IP地址为：      10.181.214.107

[INFO]正在连接从服务器...
[INFO]从服务器连接成功!

操作成功，执行结果如下：
本次操作改变了3行
```

4.4 删除数据

```
发送的sql语句为：delete from test1 where id =2
从节点IP地址为：      10.181.214.107

[INFO]正在连接从服务器...
[INFO]从服务器连接成功！

操作成功，执行结果如下：
本次操作改变了1行
```

4.5 查询数据

全部查询:

```
发送的sql语句为：select * from teacher
从节点IP地址为：    10.181.214.66

[INFO]正在连接从服务器...
[INFO]从服务器连接成功！

操作成功，执行结果如下：
id tele text
0 1613 iaxbai
1 4798 xnao
2 168 xnai
3 7616 xnqq
4 7617 wiecpoq
5 1646 xqno
6 8791 wqoq
7 19156 qdni
```

条件查询:

```
发送的sql语句为：select name,sex from student where id < 6
从节点IP地址为：      10.181.214.107

[INFO]正在连接从服务器...
[INFO]从服务器连接成功！

操作成功，执行结果如下：
name sex
roy 男
小明 男
小红 女
小刚 男
小芳 女
```

4.6 更新数据

```
发送的sql语句为：update test1 set name = "lym" where id = 1
从节点IP地址为：      10.181.214.107

[INFO]正在连接从服务器...
[INFO]从服务器连接成功！

操作成功，执行结果如下：
本次操作改变了1行
```

4.7 删除表

```
发送的sql语句为：drop table test1
从节点IP地址为：      10.181.214.107

[INFO]正在连接从服务器...
[INFO]从服务器连接成功！

操作成功，执行结果如下：
本次操作改变了0行
```

4.8 副本管理与容灾容错

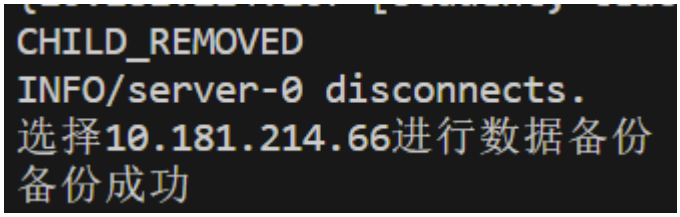
每一次更新操作，我们将采用 FTP 进行数据备份完成副本管理。并在从节点故障时使用 FTP 进行数据恢复

FTP 服务器:

FTP 根位于 10.181.214.107

05/30/2023 08:20下午	122	index_catalog
05/25/2022 08:37下午	4,096	student
05/25/2022 08:37下午	28	student_index.index
05/30/2023 08:20下午	187	table_catalog
05/25/2022 08:37下午	4,096	teacher
05/25/2022 08:37下午	24	teacher_index.index
05/25/2022 08:37下午	4,096	test0
05/30/2023 06:51下午	23	test0_index.index

数据备份:



5 总结

5.1 优点

本次课程项目中，小组三人通力合作，完成了一个分布式的关系型数据库系统，同时在项目实现的时间规划上较为合理，设定好各个时间节点的目标，整个项目推进相对顺畅，每个成员在完成项目的过程中，巩固了课堂知识，也锻炼了代码能力。

5.2 反思与不足

在小组沟通方面，沟通频率与效率还有上升空间，能够更高效地实现项目。另外，本次实验中基本功能都已经实现，但部分功能模块还能够进一步完善改善，如副本管理时，仅使用了一个 FTP 服务器，而没有考虑到该 FTP 服务器出现错误运行状态后的处理方式；另外，本系统中每次客户端与 Region Server 的联系都要通过 Master Server 的介入，如果能够在客户端维护一个关于表与 Region Server IP 的缓存，则能够更高效的查询，同时也一定程度上避免了 Master Server 的故障导致的查询错误。