

大规模信息系统构建技术导论

分布式 MiniSQL 系统个人报告

2021 学年 第一学期（上）

组员信息（第一行请写组长信息）

学号	姓名
3180103772	张溢弛

2021 年 6 月 10 日

目 录

一、引言	4
1.1 系统目标	4
1.2 设计说明与任务分工	4
二、系统设计与实现	4
2.1 系统总体架构	5
2.2 Client 设计	5
2.2.1 架构设计	5
2.2.2 工作流程	6
2.2 miniSQL 架构设计	7
2.2.1 Interpreter 层	9
2.2.2 API 层	10
2.2.3 Catalog Mananger	11
2.5.7 数据文件	12
三、核心功能模块设计与实现	13
3.1 Socket 通信架构与通信协议	13
3.1.1 系统通信架构	13
3.1.2 通信协议设计	13
3.2 客户端缓存与分布式查询	14
四、系统测试	15
4.1 初始化	15
4.2 创建表	16
4.3 插入记录	17
4.4 查询记录	17
4.4.1 条件查询	17
4.4.2 全部查询	18
4.5 删除记录	18

4.6 删除表.....	19
4.7 均衡负载.....	19
4.8 副本管理与容错容灾.....	19
五、总结.....	20

一、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，在大二春夏学期学习的《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》所学知识实现的一个分布式关系型简易数据库系统。

该系统包含 Zookeeper 集群，客户端，主从节点等多个模块，可以实现对简单 SQL 语句的处理解析和分布式数据库的功能，并具有数据分区，均衡负载，客户端缓存，副本管理，容错容灾等功能。

本系统使用 Java 语言开发，并使用 Maven 作为项目管理工具，使用 Github 进行版本管理和协作开发，由小组内的三名成员共同完成，每个人都有自己的突出贡献。

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，IDEA 作为集成开发环境，Maven 作为包管理工具，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：

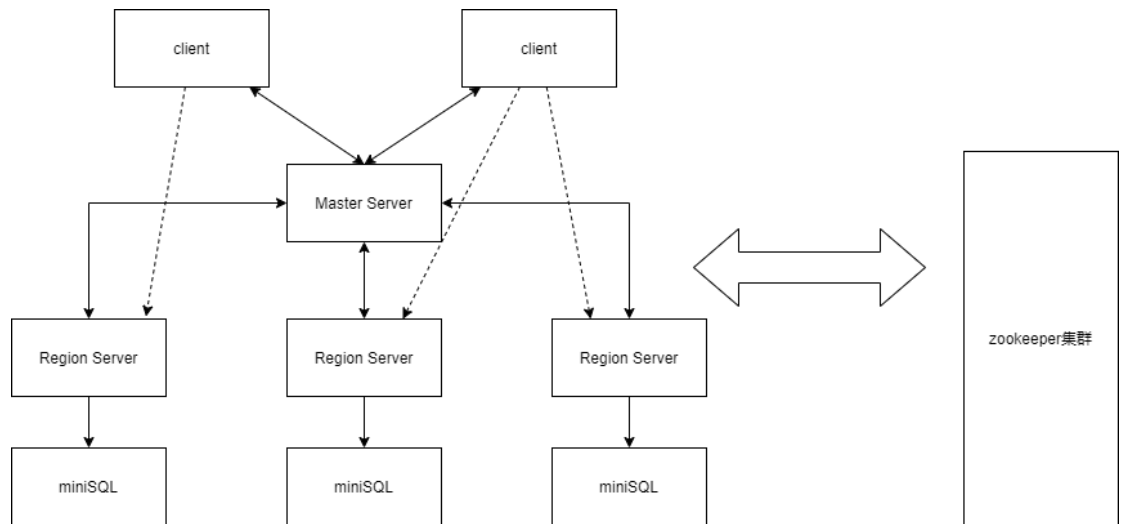
成员姓名	学号	分工职责
张溢弛	3180103772	总体架构的设计和搭建，客户端和通信协议模块以及分布式存储、缓存机制的开发，miniSQL 的 Interpreter，CatalogManager 和 API 模块的开发

- 本文档是张溢弛的个人工作文档，仅包含我个人在本次课程项目中完成的一系列工作
- 我在本项目中主要负责了基础架构的搭建，Client 的开发，Socket 通信框架和通信协议的开发、客户端缓存的开发和 miniSQL 中部分模块的开发。

二、系统设计与实现

2.1 系统总体架构

本系统的总体架构设计如下图所示：

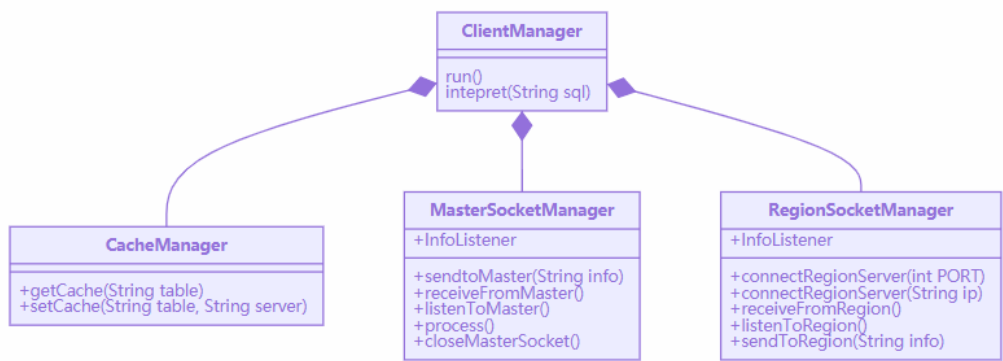


系统的源代码中将整个 maven 项目划分成三个模块，分别是 Client, Master Server 和 Region Server，分别对应分布式 MiniSQL 系统的客户端、主节点和从节点，并且三者之间都可以在一定的通信框架下进行通信，同时主节点和从节点通过 Zookeeper 集群，对数据表的信息进行统一的管理。

2.2 Client 设计

2.2.1 架构设计

本系统中的 Client 设计如下图所示：



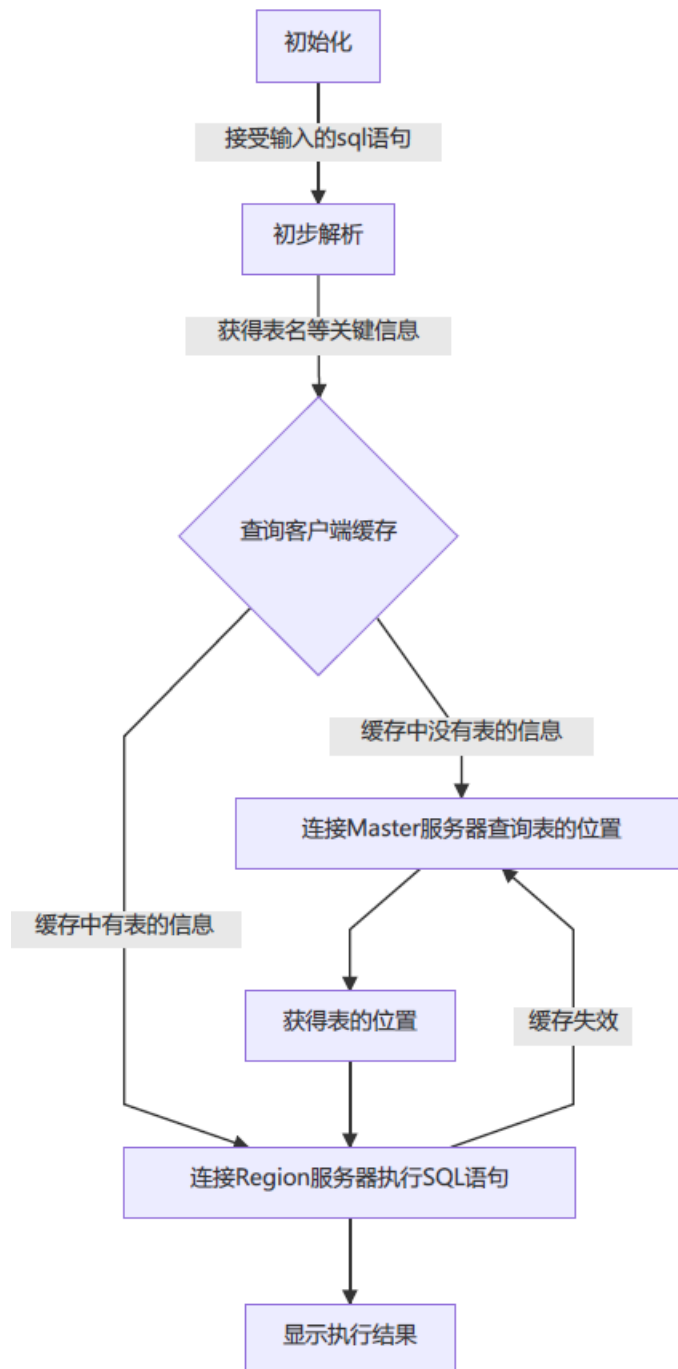
Client 由一个 ClientManager 类负责管理，该类下面还有三个子功能的类，分别

是：

- **CacheManager**，负责客户端的缓存服务，使用一个 **HashMap** 来对缓存进行管理，每次开启客户端就会创建而退出客户端就会清除，使用 **key-value** 的方式存储了一些表名-从节点 **IP** 地址的对，并提供了增删查改的接口
- **MasterSocketManager** 负责和主服务器进行通信，通过建立 **Socket** 连接实现异步的 **IO** 流，在一个单独的线程中运行，包括查询数据表所在的 **Region** 等功能，具体的通信协议在第三部分介绍
- **RegionSocketManager** 负责和 **Region Server** 进行通信，**Client** 在获得了某个表对应的 **Region** 的 **IP** 地址之后会和这个 **Region** 建立 **Socket** 连接，并且监听端口位于一个单独的线程中，可以执行 **SQL** 语句的发送和接收 **SQL** 语句执行结果等功能

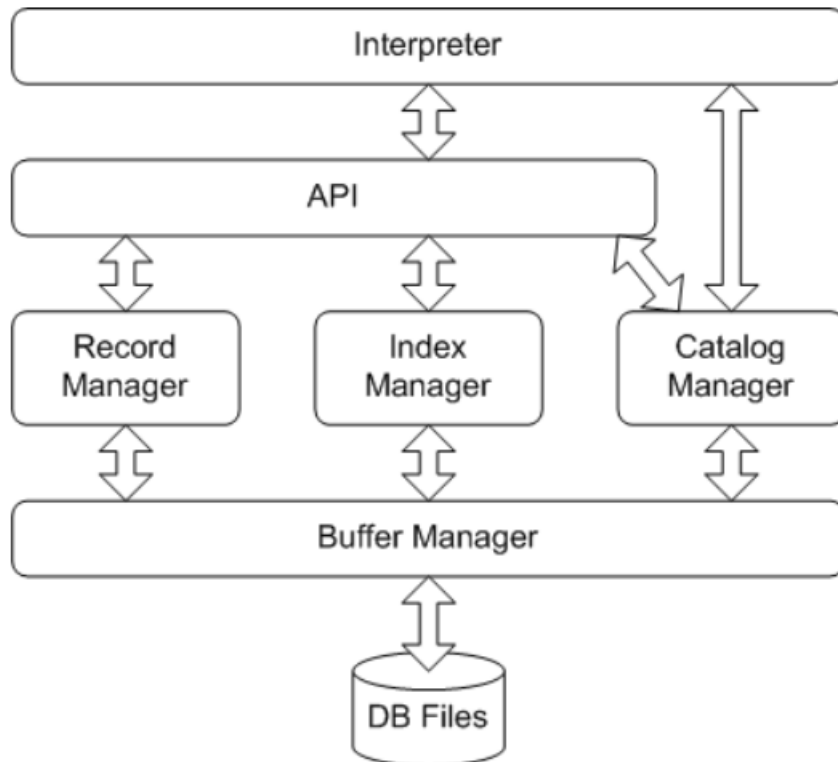
2.2.2 工作流程

客户端的工作流程可以用下面的流程图来表示：



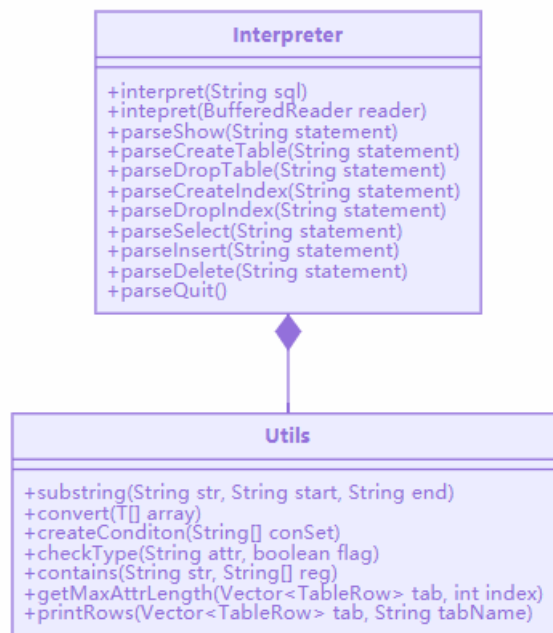
2.2 miniSQL 架构设计

miniSQL 是 Region 中很重要的一个模块，是一个关系型数据库引擎，支持多种简单的 SQL 语句的解析和处理，同时对外提供 RESTful 的接口供 Region 进行调用。其总体架构设计如下：



其支持的 SQL 语句类型主要有：创建数据表，插入记录，修改记录，删除记录，创建索引，删除索引，删除数据表，退出运行等等。我们在原本的 C++miniSQL 基础上对其功能进行了扩展，完成了一个 Java 版本的 miniSQL，其具体的功能模块设计如下，其中我主要负责了 Interpreter，API 和 Catalog 模块，他们的架构设计如下：

我们使用一个类 `Interpreter` 完成了这些功能，并编写了一个工具类 `Utils` 提供一系列辅助方法，具体的类图如下：

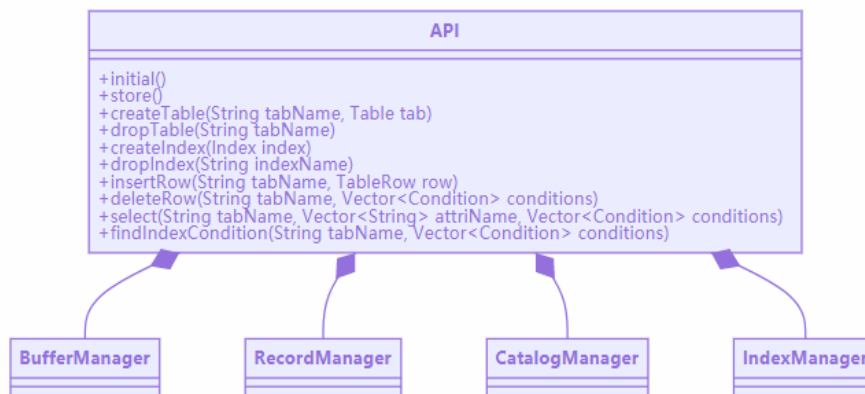


2.2.2 API 层

API 模块是整个系统的核心，其主要功能为提供执行 SQL 语句的接口，供 Interpreter 层调用。

该接口以 Interpreter 层解释生成的命令内部表示为输入，根据 Catalog Manager 提供的信息确定执行规则，并调用 Record Manager、Index Manager 和 Catalog Manager 提供的相应接口进行执行，最后返回执行结果给 Interpreter 模块。

API 层是连接顶层的 SQL 解释器和底层各个主模块的中间层，对 Interpreter 产生的解析结果调用不同的底层 API 来实现对应的功能，其类图如下：

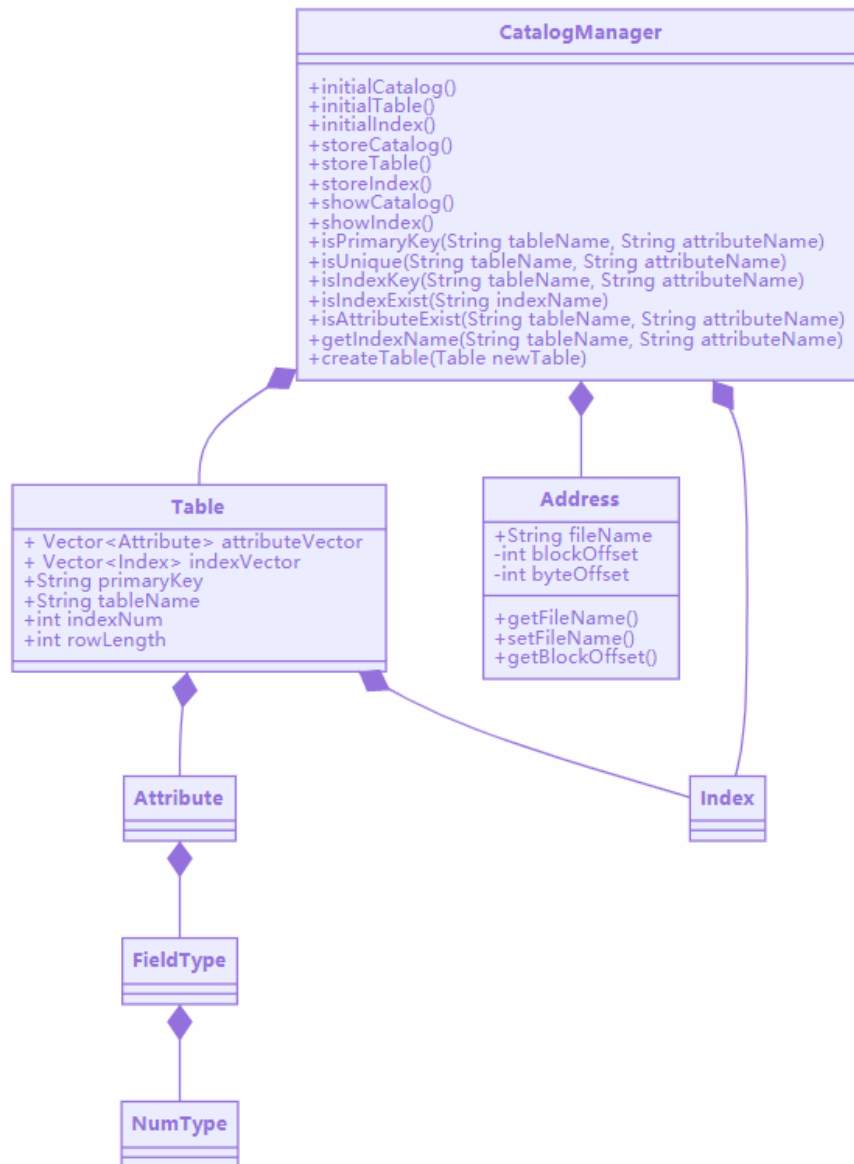


2.2.3 Catalog Manager

Catalog Manager 负责管理数据库的所有模式信息，包括：

- 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。
- 表中每个字段的定义信息，包括字段类型、是否唯一等。
- 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

Catalog Manager 还必需提供访问及操作上述信息的接口，供 Interpreter 和 API 模块使用。同时 Catalog Manager 中完成了一系列 Table 和 Address 等元信息存储结构，具体的类图如下：



2.5.7 数据文件

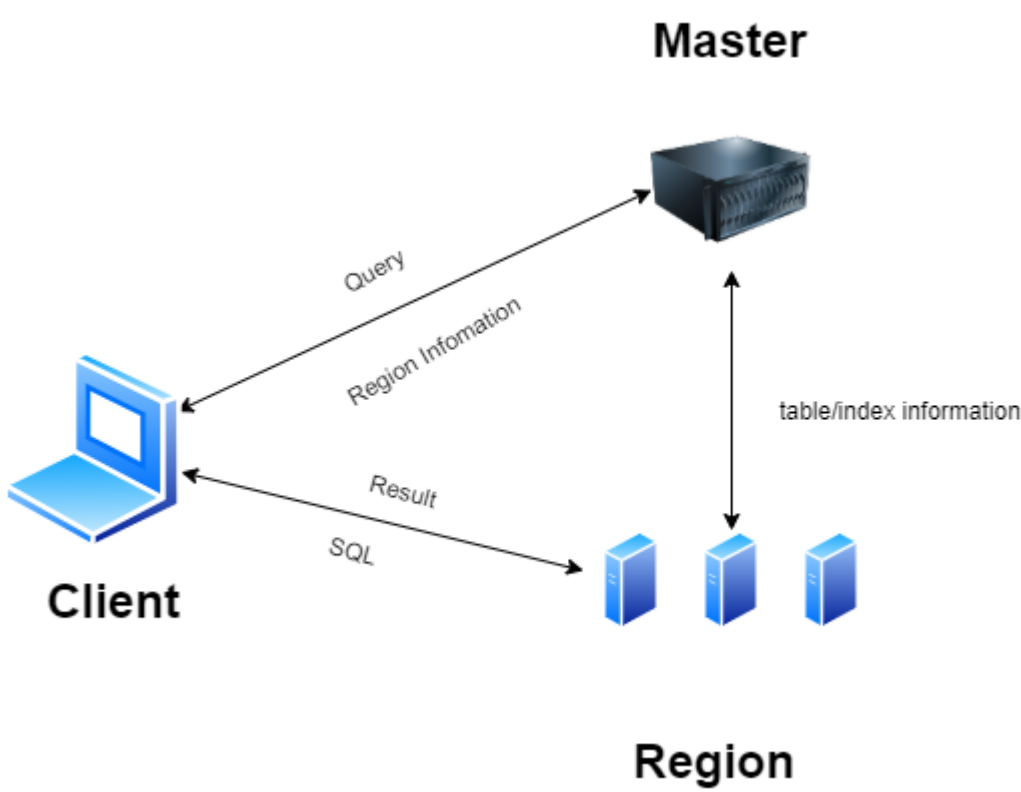
DB Files 也就是数据库的文件系统，指构成数据库的所有数据文件，主要由记录数据文件、索引数据文件和 Catalog 数据文件组成。

三、核心功能模块设计与实现

3.1 Socket 通信架构与通信协议

3.1.1 系统通信架构

本系统实现了 Client, Master 和 Region 之间的两两通信，通过 Socket 连接建立稳定可靠的通信与异步的 IO 流，同时制定了一系列通信协议，用于确保通信的可靠性和准确性，系统的总体通信架构如下图所示：



其中 Client 和 Master 以及 Master 和 Region 的连接是一直存在的，而 Region 和 Client 的通信在每次执行 SQL 语句需要换 Region 时会重新建立和目标 Region 对应的服务器，总体而言，三种不同的模块之间的通信都是全双工的。

3.1.2 通信协议设计

协议格式	返回格式	用途	含义
------	------	----	----

<client>[1] tableName	<master>[1]ip	用于客户端和主服务器的通信	向主服务器查询该表所在的 Region 服务器
<client>[2] tableName	<master>[2]ip	用于客户端和主服务器的通信	向主服务器发送一个建表的请求，并由主服务器使用负载均衡算法分配一个 Region
<master>[3]ip#name@name	<region>[3] Complete disaster recovery	用于主服务器和从节点的通信	容错容灾策略，主节点给负载小的从节点发送挂掉从节点的 ip 与所有的表，从节点根据这些信息去 ftp 服务器上下载相应备份
<master>[4]recover	<region>[4] Online	用于主服务器和从节点的通信	恢复策略，从节点重新上线，主节点给从节点发消息，让该从节点删除所有旧的表

3.2 客户端缓存与分布式查询

本系统中我们实现了客户端缓存，可以大大降低查询所需的时间，提高 IO 的效率，缓存通过一个 HashMap 实现，并且会定期更新和清除缓存的内容，客户端在向服务器发送 SQL 语句之前先对输入的 SQL 语句进行一个简单的解析，提取出要处理的表名或者索引名，然后在缓存中先进行查询，如果查到了对应的 Region 地址就直接和 Region 进行连接，如果没有查到就和 Master 先连接并获取

Region 的地址之后再和 Region 连接。

因此本系统实现了所谓的分布式查询，即客户端执行 SQL 语句时，并不是所有时候都依赖 Master 提供的元信息，而是使用一套分布式的缓存功能（缓存分布在各个客户端上），有的时候客户端可以直接实现和 Region 的连接以及通信，并不完全依赖主服务器，因此也具有一定的容错容灾能力（当然主要的容错容灾能力并不依赖于缓存）

同时，如果客户端的缓存发生了失效，那么客户端会向主节点发起新的请求获取 Region 的最新位置之后，再建立和 Region 的 Socket 连接并进行通信，并将缓存进行更新。


四、系统测试

在项目的测试部分，我们使用了 3 台计算机，在本地运行了 1 个 Master，3 个 Region 和 2 个 Client(其中一个后面才打开)，由于是分布式的系统，因此最终的测试截图如下：

4.1 初始化

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

- 客户端



```
新消息>>>客户端的主服务器监听线程启动!  
Distributed-MiniSQL客户端启动!
```

- 从节点

```

从节点开始运行!
新消息>>>从节点的主服务器监听线程启动!
服务端建立了新的客户端子线程: 59594
服务器监听客户端消息中/10.181.241.13359594
要处理的命令: create table stu (id int, name char(10), score float, primary key(id)) ;
-->Create table stu successfully
服务端建立了新的客户端子线程: 59651
服务器监听客户端消息中/10.181.241.13359651
要处理的命令: insert into stu values (1, 'zq', 100.0) ;

stu
success

```

● 主节点

```

2021-06-09 23:17:10,063 WARN [MasterManagers.utils.ServiceMonitor] - 服务器目录新增节点: /db/Region_1
2021-06-09 23:17:10,064 WARN [MasterManagers.utils.ServiceMonitor] - 新增服务器节点: 主机名 Region_1, 地址 10.181.241.133
2021-06-09 23:17:10,064 WARN [MasterManagers.utils.ServiceMonitor] - 对该服务器Region_1执行新增策略
2021-06-09 23:17:10,802 WARN [MasterManagers.SocketManager.SocketThread] - <region>[1]
服务端建立了新的客户端子线程:/10.162.11.58:57603
服务端建立了新的客户端子线程:/10.181.241.133:59548

```

4.2 创建表

```

新消息>>>请输入你想要执行的SQL语句:
create table stu (id int, name char(10), score float, primary key(id)) ;
create table stu (id int, name char(10), score float, primary key(id)) ;
新消息>>>请输入你想要执行的SQL语句:
存入缓存: 表名stu 端口号: stu
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动!
新消息>>>从服务器收到的信息是: <result>-->Create table stu successfully!

```


4.3 插入记录

```
insert into stu values (1, 'zq', 100.0) ;
insert into stu values (1, 'zq', 100.0) ;
新消息>>>客户端缓存中存在该表! 其对应的服务器是: 10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动!
新消息>>>请输入你想要执行的SQL语句:
新消息>>>从服务器收到的信息是: <result>-->Insert successfully
insert into stu values (2, 'zyc', 100.1) ;
insert into stu values (2, 'zyc', 100.1) ;
新消息>>>客户端缓存中存在该表! 其对应的服务器是: 10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动!
新消息>>>请输入你想要执行的SQL语句:
新消息>>>从服务器收到的信息是: <result>-->Insert successfully
```

4.4 查询记录

4.4.1 条件查询

```
select * from stu where score>100 ;
select * from stu where score>100 ;
新消息>>>客户端缓存中存在该表! 其对应的服务器是: 10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动!
新消息>>>请输入你想要执行的SQL语句:

新消息>>>从服务器收到的信息是: <result>|id|name|score|
新消息>>>从服务器收到的信息是: -----
新消息>>>从服务器收到的信息是: |2 |zyc |100.1|
新消息>>>从服务器收到的信息是: -->Query ok! 1 rows are selected
新消息>>>从服务器收到的信息是: Finished in 0.003 s
```

4.4.2 全部查询

```
select * from stu ;
select * from stu ;
新消息>>>客户端缓存中存在该表！其对应的服务器是：10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动！
新消息>>>请输入你想要执行的SQL语句：
新消息>>>从服务器收到的信息是：<result>|id|name|score|
新消息>>>从服务器收到的信息是：-----
新消息>>>从服务器收到的信息是：|1 |zq |100.0|
新消息>>>从服务器收到的信息是：|2 |zyc |100.1|
新消息>>>从服务器收到的信息是：-->Query ok! 2 rows are selected
新消息>>>从服务器收到的信息是：Finished in 0.0 s
```

4.5 删除记录

```
delete from stu where id=1 ;
delete from stu where id=1 ;
新消息>>>客户端缓存中存在该表！其对应的服务器是：10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动！
新消息>>>请输入你想要执行的SQL语句：
新消息>>>从服务器收到的信息是：<result>-->Delete 1 row(s).
select * from stu ;
select * from stu ;
新消息>>>客户端缓存中存在该表！其对应的服务器是：10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动！
新消息>>>请输入你想要执行的SQL语句：
新消息>>>从服务器收到的信息是：<result>|id|name|score|
新消息>>>从服务器收到的信息是：-----
新消息>>>从服务器收到的信息是：|2 |zyc |100.1|
新消息>>>从服务器收到的信息是：-->Query ok! 1 rows are selected
新消息>>>从服务器收到的信息是：Finished in 0.0 s
```

4.6 删除表

```
drop table stu ;
drop table stu ;
新消息>>>客户端缓存中存在该表！其对应的服务器是：10.181.241.133
connectRegionServer : 10.181.241.133
新消息>>>与从节点10.181.241.133建立Socket连接
新消息>>>客户端的从服务器监听线程启动！
新消息>>>请输入你想要执行的SQL语句：
新消息>>>从服务器收到的信息是：<result>-->Drop table stu successfully!
```

4.7 均衡负载

- 主服务器消息记录：

```
2021-06-09 23:17:10,063 WARN [MasterManagers.utils.ServiceMonitor] - 服务器目录新增节点: /db/Region_1
2021-06-09 23:17:10,064 WARN [MasterManagers.utils.ServiceMonitor] - 新增服务器节点: 主机名 Region_1, 地址 10.181.241.133
2021-06-09 23:17:10,064 WARN [MasterManagers.utils.ServiceMonitor] - 对该服务器Region_1执行新增策略
2021-06-09 23:17:10,802 WARN [MasterManagers.SocketManager.SocketThread] - <region>[1]
服务端建立了新的客户端子线程:/10.162.11.58:57603
服务端建立了新的客户端子线程:/10.181.241.133:59548
2021-06-09 23:17:28,899 WARN [MasterManagers.SocketManager.SocketThread] - <client>[2]stu
2021-06-09 23:17:32,729 WARN [MasterManagers.SocketManager.SocketThread] - <region>[2]stu add
2021-06-09 23:17:36,625 WARN [MasterManagers.SocketManager.SocketThread] - <client>[2]test
2021-06-09 23:17:39,897 WARN [MasterManagers.SocketManager.SocketThread] - <region>[2]test add
```

4.8 副本管理与容错容灾

- 主服务器执行失效时候的备份和恢复策略

```
2021-06-09 23:18:26,570 WARN [MasterManagers.utils.ServiceMonitor] - 服务器目录新增节点: /db/Region_2
2021-06-09 23:18:26,571 WARN [MasterManagers.utils.ServiceMonitor] - 新增服务器节点: 主机名 Region_2, 地址 10.181.247.33
2021-06-09 23:18:26,571 WARN [MasterManagers.utils.ServiceMonitor] - 对该服务器Region_2执行新增策略
2021-06-09 23:18:27,385 WARN [MasterManagers.SocketManager.SocketThread] - <region>[1]
2021-06-09 23:18:43,643 WARN [MasterManagers.utils.ServiceMonitor] - 服务器目录删除节点: /db/Region_1
2021-06-09 23:18:43,644 WARN [MasterManagers.utils.ServiceMonitor] - 服务器节点失效: 主机名 Region_1, 地址 10.181.241.133
2021-06-09 23:18:43,644 WARN [MasterManagers.utils.ServiceMonitor] - 对该服务器Region_1执行失效策略
2021-06-09 23:18:43,644 WARN [MasterManagers.utils.ServiceStrategyExecutor] - bestInet:10.181.247.33
2021-06-09 23:18:47,215 WARN [MasterManagers.SocketManager.SocketThread] - <region>[3]Complete disaster recovery
2021-06-09 23:18:47,217 WARN [MasterManagers.SocketManager.RegionProcessor] - 完成从节点的数据转移
```

```
2021-06-09 23:19:47,317 WARN [MasterManagers.utils.ServiceMonitor] - 服务器目录新增节点: /db/Region_3
2021-06-09 23:19:47,319 WARN [MasterManagers.utils.ServiceMonitor] - 新增服务器节点: 主机名 Region_3, 地址 10.181.241.133
2021-06-09 23:19:47,319 WARN [MasterManagers.utils.ServiceMonitor] - 对该服务器Region_3执行恢复策略
2021-06-09 23:19:47,896 WARN [MasterManagers.SocketManager.SocketThread] - <region>[1]test
2021-06-09 23:19:48,901 WARN [MasterManagers.SocketManager.SocketThread] - <master>[4]OnLine
```

五、总结

本次课程项目中，我们小组三人通力合作，完成了一个分布式的关系型数据库系统，在数据库系统课程所做的 miniSQL 基础之上为其扩展出了 Java 版本，并在 miniSQL 的基础上引入了 Zookeeper 集群管理，将其扩展成了一个具备分布式存储，数据分区，均衡负载，副本管理和容错容灾的分布式数据库系统，在这一过程中我们收获良多。