浙江大学

大规模技术导论

分布式 MiniSQL 总体设计报告

学生姓名 ： 刘轩铭 学号： 3180106071

学生姓名 ： 蔡灿宇 学号： 3180101972

学生姓名 ： 胡洋凡 学号： 3180103167

学生姓名 ： 杨凌霄 学号： 3180103608

学生姓名 ： 王绍兴 学号： 3180106074

2021 年 6 月

Verse:2.0

目录

1. 引言

本次开发的是分布式 MiniSQL 应用程序，这是一个综合性的题目，可以对 Java 语言中的各项功能有更好的理解和使用，对分布式应用有着更好的理解和使用，同时也能用具体的应用实践让大家对大规模软件设计的理解加深，提高自己的编程水平和设计能力，为以后的工作打下一定的基础。

1.1 系统目标

本应用的目标是设计并实现一个分布式精简型单用户 SQL 引擎(DBMS)MiniSQL，允许用户通过字符界面输入 SQL 语句实现表的建立/删除、索引的建立/删除以及表记录的插入/删除/查找。同时需要实现对分布式数据库集群的管理，实现分布式系统容错容灾、副本复制、负载均衡等功能。

通过对分布式 MiniSQL 的设计与实现，提高学生的系统编程能力，加深对数据库系统原理的理解和分布式应用的理解。

1.2 开发排期安排

- [x] 2021.03.22 - 2021.04.20 项目整体技术架构初步完成
- [x] 2021.04.21 - 2021.04.24 项目成员会议，对整体架构进行完善和责任明确
- [x] 2021.04.25 - 2021.05.02 项目各模块详细设计完成
- [x] 2021.05.06 - 2021.05.14 各模块基本架构等完成
- [x] 2021.05.15 - 2021.05.21 ZooKeeper 模块基本完成、MiniSQL 各部分初步完成、各项目间接口、API 定义完成
- [x] 2021.05.22 - 2021.05.28 MiniSQL 各部分基本完成、Region Server Manager 模块基本完成
- [x] 2021.05.29 - 2021.06.04 项目整体整合和修改
- [x] 2021.06.05 - 2021.06.12 项目文档编写，录制视频并提交

2. 设计分工

本程序基本采用 Java 程序设计语言设计，在 IntelliJ IDEA 平台下编辑、编译与调试。具体程序由 5 人组成的小组开发而成。小组成员的具体分工如下表：

姓名	学号	分工	备注
刘轩铭	3180106071	组长，系统整体设计，负责 ZooKeeper 集群管理和 Master Server Manager 模块编写	**组长**
杨凌霄	3180103608	组员，系统详细设计，负责 Index Manager、Catalog Manager 编写	组员
蔡灿宇	3180101972	组员，系统详细设计，负责 Interpreter 模块编写	组员
胡洋凡	3180103167	组员，系统详细设计，负责 Record Manager 模块编写，MiniSQL Business 整合管理	组员
王绍兴	3180106074	组员，系统详细设计，负责 MiniSQL DAO(Region Server Manager)模块编写	组员

3. 总体设计

3.1 技术字典

- ZooKeeper:

ZooKeeper 是一个开放源代码的分布式应用程序协调服务，由知名互联网公司雅虎创建，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。ZooKeeper 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。它是一个为分布式应用提供一致性服务的软件。

- Thrift:

Thrift 是一种接口描述语言和二进制通讯协议，它被用来定义和创建跨语言的服务。它被当作一个远程过程调用（RPC, Remote Procedure Call）框架来使用，是由 FaceBook 为“大规模跨语言服务开发”而开发的。

• Region:

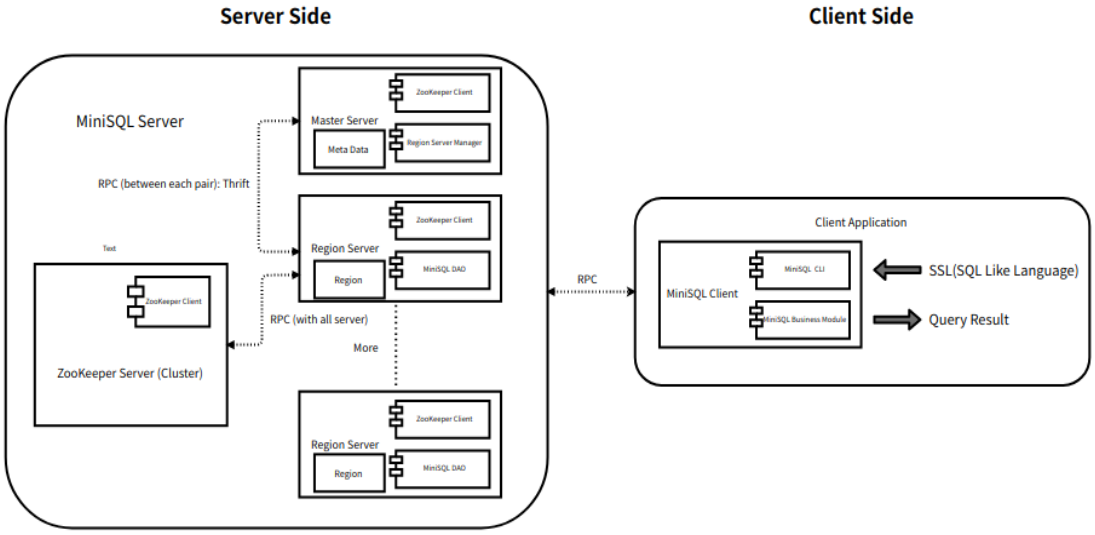
数据库中存储的数据（表格、元组等）的最小单位。为了简单起见，本应用中将 Region 定义为每一个数据表（Table）。按照分布式数据库的设计思想，Region 将被分布在各个 Region Server 上，用户进行 DDL 操作的时候通过查询 Region 的位置信息来与对应 Region Server 进行交互，完成对于 Region 内数据的 DAO 操作。

3.2 总体架构设计

本程序需要实现的主要功能有：

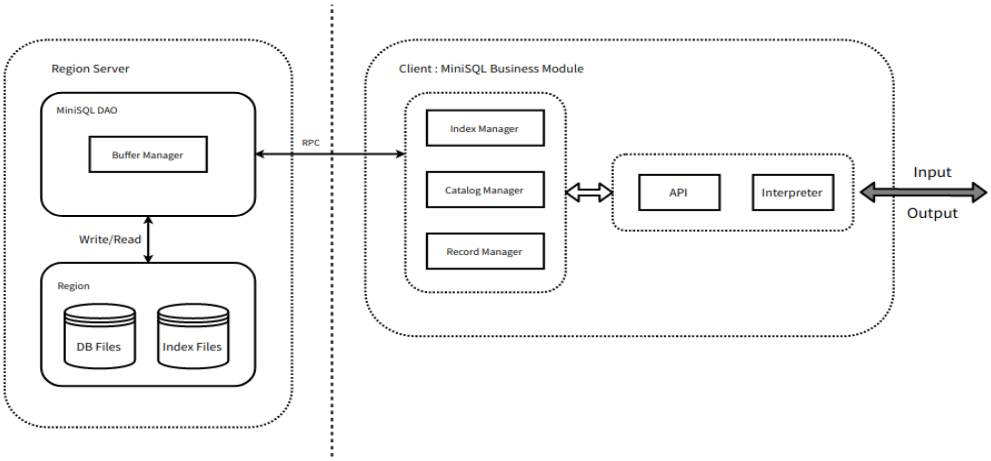
1. 用户输入 SSL（SQL Like Language），系统对其逻辑进行解析，返回查询结果；
2. 客户端与服务端的数据表进行交互，读写源数据；
3. 维护服务端集群的生命活力，对各机器（进程）生命周期进行监测；

程序的总体架构设计图如下：



由上述架构图可以看出，本应用整体分为服务端和客户端两个部分。其中服务端主要涉及到对于分布式系统的管理，而客户端主要涉及到对于用户输入 SQL 的解析和查询结果展示。

对于 MiniSQL 而言，我们将其逻辑层全部放在客户端，在 Region Server 上仅完成对于数据的读写工作。MiniSQL 部分的架构图如下：



由上述架构图可以看出，MiniSQL 分为 BufferManager， IndexManager，Interpreter 等多个模块。为了简便和解耦考虑，将 BufferManager 放置在 Region Server 中，负责对硬盘数据的读写工作。而将其他模块放在 Client 中。在运行的时候，Client 的各个模块只需要访问 BufferManager 给出的接口，即可完成对底层数据的读写操作。

下面，分别介绍各个模块的详细功能。

3.2.1 Server Side

根据以上架构图，分别设计如下的模块：

- **Master Server**：部署在一台物理机上，设计对应的逻辑进行管理。该模块订阅 ZooKeeper 的服务集群，需要设计回调函数对服务机注册、服务机失效等事件进行处理。此外，还需要对于客户端的元数据请求进行响应，返回相应的元数据。
- **Region Server**：挂载在 Master 上（由于容错容灾考虑，Master 服务机有可能需要替换，但在实际操作上该可能性较小，为了简便起见，在本应用中我们假设 Master 机不会发生失效的情况），用于管理 Meta Data 的内容，即对于各 Region 的信息（位置信息、数据保存情况等）进行管理。
- **MiniSQL DAO(Data Access Object)**：通用模块，挂载在所有 Region 机上，用于接收到数据访问请求后，对相应的数据进行读取和写入的功能。此外还需要设计和其他 Region Server 的信息传输功能，从而完成副本拷贝等工作。该部分由许多子模块组成，具体功能由子模块自行设计。子模块包含 BufferManager 等。

3.2.2 Client Side

- ZooKeeper Server: 部署在一台物理机上，设计对应的逻辑进行管理。需要设计回调函数对服务机注册、服务机失效等事件进行处理。
- ZooKeeper Client: 通用模块，挂载在每台服务机上，负责与 Server 进行联系，汇报生命周期情况等。
- Region Server Manager: 挂载在 Master 上（由于容错容灾考虑，Master 服务机有可能需要替换，但在实际操作上该可能性较小，为了简便起见，在本应用中我们假设 Master 机不会发生失效的情况），用于管理 Meta Data 的内容，即对于各 Region 的信息（位置信息、数据保存情况等）进行管理。
- MiniSQL DAO(Data Access Object): 通用模块，挂载在所有 Region 机上，用于接收到数据访问请求后，对相应的数据进行读取和写入的功能。此外还需要设计和其他 Region Server 的信息传输功能，从而完成副本拷贝等工作。该部分由许多子模块组成，具体功能由子模块自行设计。子模块包含 BufferManager 等。

3.2.3 MiniSQL Modules

根据 MiniSQL 架构，分别设计如下的模块：

- Interpreter: 模块直接与用户交互，主要实现程序流程控制、接收并解释用户输入的命令，生成命令的内部数据结构表示，同时检查命令的语法正确性和语义正确性，对正确的命令调用 API 层提供的函数执行并显示执行结果，对不正确的命令显示错误信息。
- API: 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。
- Catalog Manager: 负责管理数据库的所有模式信息。
- Record Manager: 负责管理记录表中数据的数据文件。主要功能为实现数据文件的创建与删除（由表的定义与删除引起）、记录的插入、删除与查找操作，并对外提供相应的接口。其中记录的查找操作要求能够支持不带条件的查找和带一个条件的查找（包括等值查找、不等值查找和区间查找）。

• **Index Manager**：负责 B+树索引的实现，实现 B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。B+树中节点大小应与缓冲区的块大小相同，B+树的叉数由节点大小与索引键大小计算得到。

• **Buffer Manager**：负责磁盘读写和缓冲区的管理，主要功能有根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件、实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换、记录缓冲区中各页的状态，如是否被修改过等、提供缓冲区页的 pin 功能，及锁定缓冲区的页，不允许替换出去、为提高磁盘 I/O 操作的效率，缓冲区与文件系统交互的单位是块，块的大小应为文件系统与磁盘交互单位的整数倍，一般可定为 4KB 或 8KB。

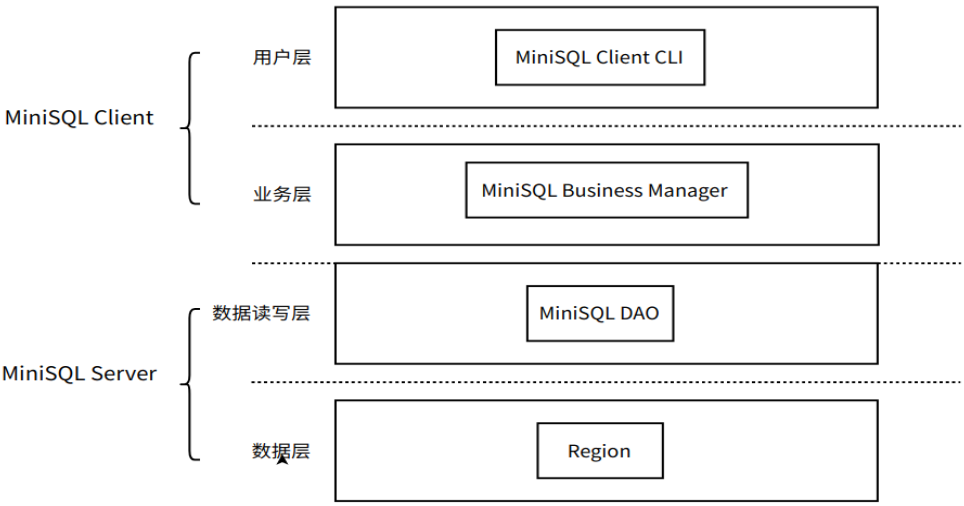
3.2.4 Utils

此外，还需要设计如下通用模块：

• **Thrift Interface**：由于本应用采用 Thrift 作为 RPC 调用的实现方式，所以在服务端与客户端之间、Region Server 和 Master 之间，Region Server 和 Region Server 之间，都需要设计 Thrift 接口，并安装 Thrift 来完成 RPC 调用的功能，该部分应该由对应的模块自行设计。在实现上，我们采用包装 Thrift 服务端和客户端的方法，用 Java 抽象类的方式进行实现，各模块只需要按要求创建相应的实体类，就可以方便的创建和使用 Thrift 服务。

3.3 业务流程设计

根据本小组自行设计，分布式 MiniSQL 的业务流程如下图所示：



对应到模块，其中：

- Master 负责对各 Region Server 进行管理，决定哪些机器是副本机，哪些是主机，同时决定其中保存的数据表内容。
- 用户通过客户端 CLI 输入 SQL 语句后，语句将在客户端的 MiniSQL 引擎中的 Interpreter 进行语法分析、索引查询等操作，之后由 API 调用决定其他模块进行业务逻辑地执行和分析。
- 之后将需要查询的数据实体表以查询的方式提交给 Master，Master 将返回各表格在哪一台 Region Server 上，以及该 Server 的相应位置等元数据。
- 客户端各模块通过 API 向各 Region Server 发起 RPC 调用，请求相应数据。
- Region Server 接收请求后，通过 MiniSQL DAO 对本地数据进行读写，然后将执行结果通过 RPC 调用返回给客户端。
- ZooKeeper 集群对机器或进程进行管理，通过设计的回调函数等对发生的诸如服务器失效等事件进行处理。

3.4 （分布式）技术重点设计

在分布式数据库系统设计中可能有许多问题，以下为本应用的解决方案。

3.4.1 机器情况

- 上述架构中每个模块，其实可以看作一个进程，也就是说所有的模块实际上可以在同一台物理机上，只需要开多个进程来进行模拟即可（这是一种解决方案）。

• 考虑到小组有 5 位成员，至少有 5 台物理机，则可以做如下设计：将 ZooKeeper Server 和 Master 机器集成到统一物理机上，作为 Master Server；其余机器成对形式出现，分别扮演主机和副本机。此外还需要一定机器作为客户端（考虑到可用物理机数量限制，客户端也可以以不同的进程形式存在）或闲置机（即在 Region Server 数量不够的时候被注册到 ZooKeeper 中，转变为 Region Server）。

3.4.2 数据缓存

• 客户端在每次读写操作时都需要向 Master 请求表格元数据，这样会造成大量的网络通信压力。为了解决这一问题，考虑用缓存的方式。

• 客户端首次请求元数据后，会在本地缓存一份元数据文件。之后再次请求数据时，则会先根据本地元数据缓存的信息，想相应的数据服务器请求数据，如果请求失败或者超时，代表元数据有更新，此时重新向 Master 请求元数据并更新缓存。若没有失败，则代表元数据有效，继续保留元数据。

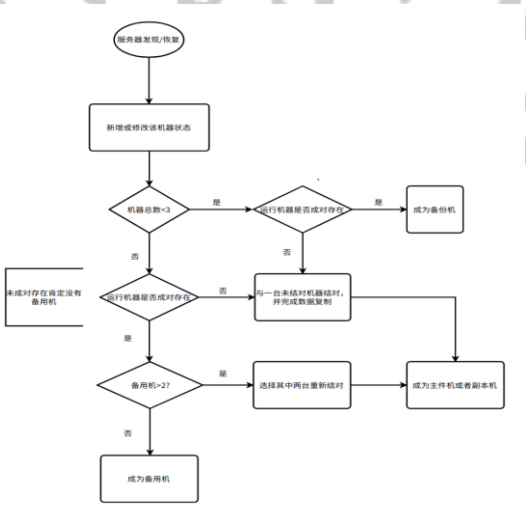
3.4.3 副本维护

• 采用主从复制策略，选择一定量的机器作为副本机器，分别对应其他的主机。

• 在用户进行读写操作的时候，主机和副本机同时进行读写操作，以两台机器均读写完成作为一次读写操作结束的标志。保证强一致性。

• 保证集群中总有至少双数台可用的 Region Server，分别进行配对。

• 服务器新增策略流程如下图所示：



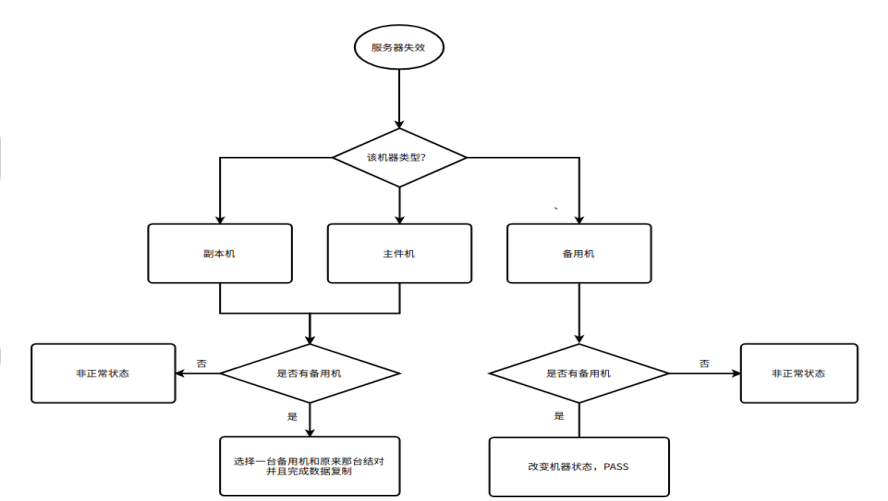
3.4.4 容错容灾

• 当某个 RegionServer 失效后，Master 会将其中的 Region 重新分配到其他的 RegionServer 中。其具体实现方式是读取副本机的内容然后转移到其他机器。

• 暂不考虑 Master 机器失效的情况。

• 由于系统中一直保证有备用机的存在，所以当有服务器失效后，Master 会安排 Region Server 和孤单的数据服务器进行重新配对，保证双数台数据服务器的存在。

• 服务器失效策略流程如下图所示：



3.4.5 负载均衡

• 采用桶装均衡策略

• 假设在系统中各个服务器被访问的频率取决于它们容纳的表格数量

• 每次新增表格的时候，新增的表格都会被安排到当前拥有表格数量最小的服务器上。

4. 模块详细设计

4.1 Interpreter 模块

各模块详细设计报告\3180101972_蔡灿宇_Interpreter 模块设计报告.pdf

4.2 Dataloader & RecordManager 模块

各模块详细设计报告\3180103167_胡洋凡_DataLoader 模块详细设计.pdf

各模块详细设计报告\3180103167_胡洋凡_RecordManager 模块详细设计.pdf

4.3 CatalogManager & IndexManager 模块

各模块详细设计报告\3180103608_杨凌霄_Catalog&Index Manager 模块详细设计.pdf

4.4 MasterManager 模块

各模块详细设计报告\3180106071_刘轩铭_MasterServer 模块详细设计.pdf

4.5 RegionServer 模块

各模块详细设计报告\3180106074_王绍兴_Regionserver 模块详细设计.pdf

5. 功能运行测试

5.1 数据库模块测试

首先运行 Master，并创建三台 Region Server，运行 Client。

图 5-1 其中一台 Region Server 创建成功截图

执行以下代码，创建表格：

```
create table student (
    sid int,
    sname string,
    sage int,
    sgender float,
    primary key (sid)
);
```

图 X-2 创建表格客户端输入代码

向 Master 请求创建表格成功，表格将被提示存储主机上，回复一个请求成功的提示，如下图所示：

图 X-3 创建表格成功提示

执行以下代码，插入数据。

```
insert into student values (1,'y',20,1.20);
insert into student values (2,'y',21,1.21);
insert into student values (3,'y',22,1.22);
insert into student values (4,'y',23,1.23);
insert into student values (5,'y',24,1.24);
```

图 5-4 插入数据客户端输入代码

以其中的一条插入语句为例，输入后得到了以下的返回结果，提示更新表格成功，如下图所示：

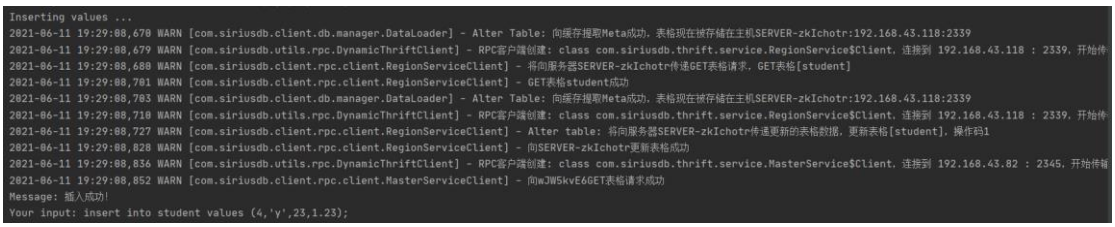


图 5-5 插入数据成功提示

执行以下代码，插入一条不合规范的数据：

```
insert into student values (6,'y');
```

图 5-6 插入违规数据客户端输入代码

因为数据不合规范，导致报错，这个请求是不合法的，提示如下图所示：

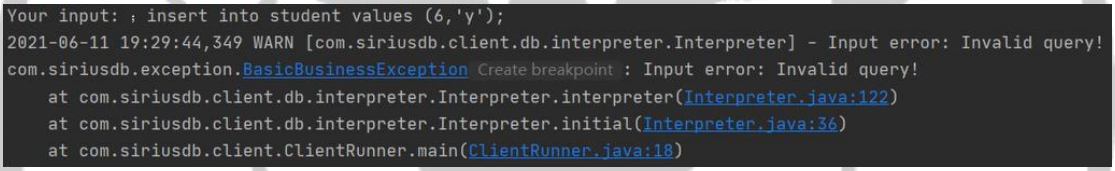


图 5-7 插入不合法数据提示

执行以下代码，继续插入数据：

```
insert into student values (7,'y',22,1.25);
insert into student values (8,'y',22,1.25);
insert into student values (9,'y',22,1.25);
insert into student values (10,'y',22,1.25);
insert into student values (10,'y',22,1.25);
```

图 5-8 继续插入数据客户端输入代码

因为代码格式符合规定，可以正常插入，提示如下图所示，以其中一条插入语句为例：

图 5-9 继续插入数据提示

执行以下代码，实现 select 语句的测试：

```
select * from student;
select sid from student;
select * from student where sage = 22;
select * from student where sage != 22;
select * from student where sage = 22 and sid = 1;
select * from student where sage = 22 or sid = 1;
```

图 5-10 选择数据客户端输入代码

每一条 select 语句均可执行，向服务器 GET 表格成功后进行选择操作成功，如下图所示，以其中一条 select 语句为例：

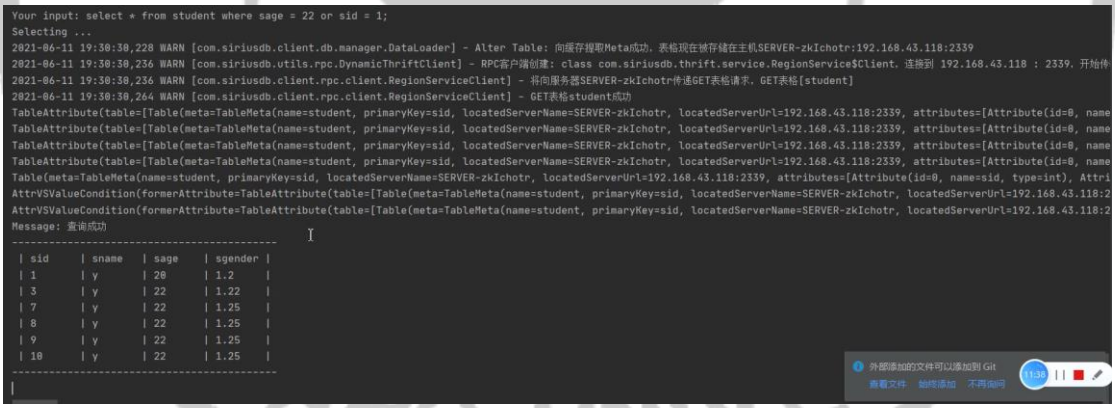


图 5-11 选择数据成功后提示

执行以下代码，实现先更新后查询的操作：

```
update student set sname = 'xxxxx' where sage = 21;
select * from student;
```

图 5-12 更新并重新查询表格客户端输入代码

完成更新后重新 select 该表，得出输出结果，发现已经成功被更改，结果如下图所示：

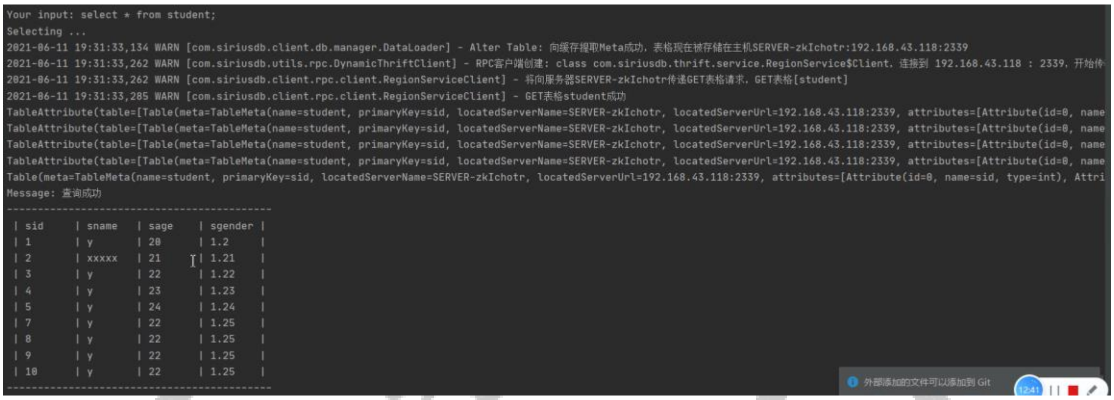


图 5-13 更新并重新查询表格输出结果

执行以下代码，首先删除指定的纪录，再次 select 该表：

```sql
delete from student where sage = 22;
select * from student;
```

图 5-14 删除指定的纪录并 select 该表客户端输入代码

完成删除后重新 select 该表，得出输出结果，发现已经成功被删除，结果如下图所示：

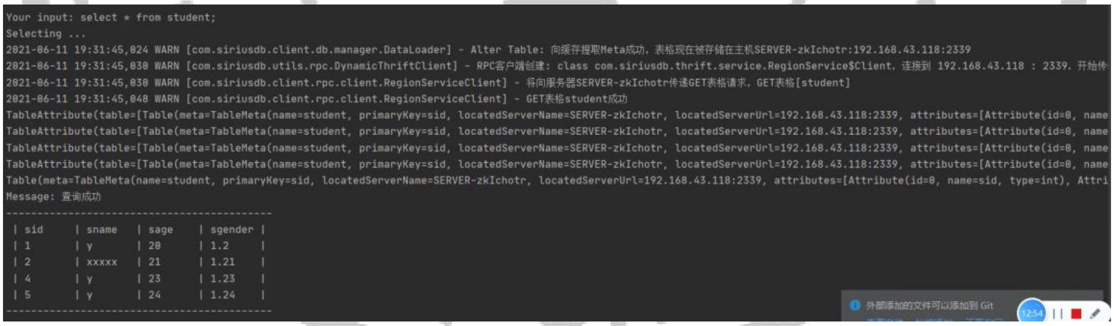


图 5-15 删除纪录并重新查询表格输出结果

执行以下代码，首先创建并随后删除一个索引：

```sql
create index indexName on student(sage);
drop index indexName on student;
```

图 5-16 新建并删除一个 Index 客户端输入代码

```
drop table newstudent;Your input: create index indexName on student(sage);
Creating index ...
Success: Index indexName has been created!
Your input: drop index indexName on student;
Dropping index ...
Success: Index indexName has been dropped!
```

图 5-17 新建并删除一个 Index 输出结果

最后执行以下代码，以检验客户端是否可以发起新建表格的请求，是否可以成功删除一个表格，输入代码如下：

```
create table newstudent (
    sid int,
    sname string,
    sage int,
    sgender float,
    primary key (sid)
);

drop table newstudent;
```

图 5-18 新建并删除一个 table 客户端输入代码

代码符合规定，成功在 Region 服务器中找到可以放置表格的位置，并且也成功删除该表，输出结果如下图所示：

```
Your input: create table newstudent ( sid int, sname string, sage int, sgender float, primary key (sid));
Creating table ...
2021-06-11 19:32:01,317 WARN [com.siriusdb.utils.rpc.DynamicThriftClient] - RPC客户端创建: class com.siriusdb.thrift.service.MasterService$Client, 连接到 192.168.43.82 : 2345, 开始传输
2021-06-11 19:32:01,726 WARN [com.siriusdb.client.rpc.client.MasterServiceClient] - 向wJW5kvE6创建表格请求成功
2021-06-11 19:32:01,726 WARN [com.siriusdb.client.db.manager.DataLoader] - 向Master请求创建表格成功, 表格将被存储在主机SERVER-zkIchotr:192.168.43.118:2339
2021-06-11 19:32:01,732 WARN [com.siriusdb.utils.rpc.DynamicThriftClient] - RPC客户端创建: class com.siriusdb.thrift.service.RegionService$Client, 连接到 192.168.43.118 : 2339, 开始传输
2021-06-11 19:32:01,732 WARN [com.siriusdb.client.rpc.client.RegionServiceClient] - 将向服务器SERVER-zkIchotr传递创建表格数据, 创建表格[newstudent], 操作码0
2021-06-11 19:32:01,995 WARN [com.siriusdb.client.rpc.client.RegionServiceClient] - 向SERVER-zkIchotr创建表格成功
Success: Table newstudent has been created!

Your input: drop table newstudent;
Dropping table ...
2021-06-11 19:32:03,573 WARN [com.siriusdb.utils.rpc.DynamicThriftClient] - RPC客户端创建: class com.siriusdb.thrift.service.MasterService$Client, 连接到 192.168.43.82 : 2345, 开始传输
2021-06-11 19:32:03,579 WARN [com.siriusdb.client.rpc.client.MasterServiceClient] - 向wJW5kvE6GET表格请求成功
2021-06-11 19:32:03,579 WARN [com.siriusdb.client.db.manager.DataLoader] - Alter Table: 向缓存提取Meta成功, 表格现在被存储在主机SERVER-zkIchotr:192.168.43.118:2339
2021-06-11 19:32:03,586 WARN [com.siriusdb.utils.rpc.DynamicThriftClient] - RPC客户端创建: class com.siriusdb.thrift.service.RegionService$Client, 连接到 192.168.43.118 : 2339, 开始传输
2021-06-11 19:32:03,586 WARN [com.siriusdb.client.rpc.client.RegionServiceClient] - 将向服务器SERVER-zkIchotr传递GET表格请求, GET表格[newstudent]
2021-06-11 19:32:03,598 WARN [com.siriusdb.client.rpc.client.RegionServiceClient] - GET表格newstudent成功
2021-06-11 19:32:03,606 WARN [com.siriusdb.utils.rpc.DynamicThriftClient] - RPC客户端创建: class com.siriusdb.thrift.service.MasterService$Client, 连接到 192.168.43.82 : 2345, 开始传输
2021-06-11 19:32:03,613 WARN [com.siriusdb.client.rpc.client.MasterServiceClient] - 向wJW5kvE6删除表格请求成功
2021-06-11 19:32:03,620 WARN [com.siriusdb.utils.rpc.DynamicThriftClient] - RPC客户端创建: class com.siriusdb.thrift.service.RegionService$Client, 连接到 192.168.43.118 : 2339, 开始传输
2021-06-11 19:32:03,680 WARN [com.siriusdb.client.rpc.client.RegionServiceClient] - 向SERVER-zkIchotr删除表格成功
Success: Table newstudent has been dropped!
```

图 5-19 新建并删除一个 table 输出结果

以上，数据库模块功能全部测试完毕，测试结果符合预期。

5.2 分布式模块设计

首先，由于我们组机器数量不够，在测试时用进程来代替 Region Server，通过改变端口和随机生成 ServerName 的方式来构造。

随后运行 Master，再运行 2 台 Region Server，最后运行 Client，此时 Region 和 Master 有相应的 ZooKeeper 信息显示，证明已经完成了服务注册和服务发现，如图，是其中一个 Region Runner。

图 5-20 完成服务注册和服务发现截图

因为首先只创建 2 个 Region Runner，执行以下代码，创建一个表格，发现无法创建，因为不存在闲置机：

```
create table student (
    sid int,
    sname string,
    sage int,
    sgender float,
    primary key (sid)
);
```

图 5-21 创建表格客户端输入代码

然后再次运行 4 台 Region Server，此时应该有两组 Region Server 和两台闲置机，如下图所示，是其中一台拥有副本机的主机：

图 5-22 其中一台 Region 服务器

然后执行以下代码，新建两个表格，测试是否符合负载均衡的策略，分配到不同的机器中：

```
create table student (
    sid int,
    sname string,
    sage int,
    sgender float,
    primary key (sid)
);

create table newstudent (
    sid int,
    sname string,
    sage int,
    sgender float,
    primary key (sid)
);
```

图 5-23 负载均衡测试客户端输入代码

图 5-24 负载均衡测试客户端输出结果

可以清晰地在不同的两台 Region Server 中看到，表格被分别创建，如下二图所示：

图 5-25 table1 被创建

图 5-26 table2 被创建

副本维护已经在数据库测试模块进行测试，接下来是容错容灾的测试，首先执行以下代码：

```sql
select * from student;
```

图 5-27 select 语句客户端输入代码

可以在客户端输出结果中看到，向服务器 A 上查询到了数据，同样，在对应的 Region Server 上也可以看到 queryTableData 中添加了一个表名：

图 5-28 select 语句客户端输出结果

此时我们挂掉该机器 A 的副本机 B，此时应该发生 Master 检测到变化，通知一台备用机 C 和 A 重新完成结对，并且进行表格的信息复制，如下图所示：

图 5-29 备用机上线并复制信息

接下来执行以下代码：

```sql
insert into student values (1,'y',20,1.20);
```

图 5-30 插入语句客户端输入代码

在插入语句执行后，在机器 A 和副本机 C 上都发生了 TableChange，也就是都发生了信息的新增，如下图所示：

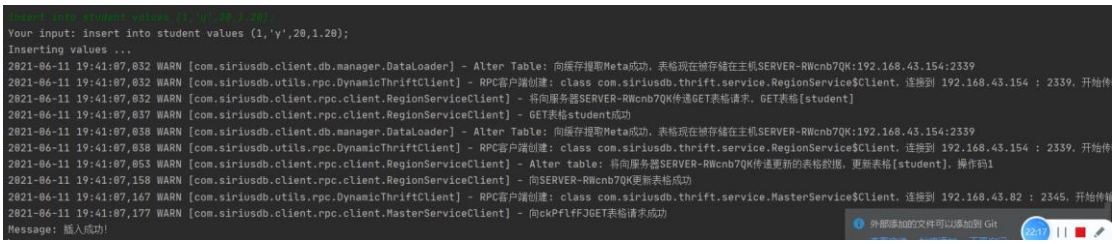


图 5-31 插入语句客户端输出结果

此时我们挂掉机器 A，并重新上线一台机器 D，测试是否新上线的机器可以和机器 C 发生配对，新上线的机器如下图所示：

图 5-32 新机器 D 上线

此时应该发生 Master 检测到变化，通知 C 和 D 重新完成结对，其中 D 为主，C 为副。接下来执行以下代码，测试是否在新上线的机器 D 中查询到表格数据：

```
select * from student;
```

图 5-33 select 语句执行

此时应该发生向机器 D 的数据请求并得到正确的数据，客户端输出结果下图所示：

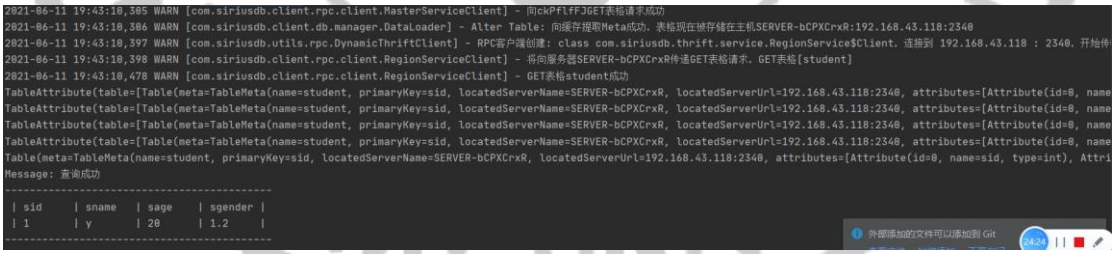


图 5-34 成功向机器 D 查询数据

至此，全部模块功能全部测试完毕，测试结果符合预期。