

大规模信息系统构建技术导论

分布式MiniSQL系统最终报告

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 |
| 3180103162 | 张琦 |
| 3180103501 | 聂俊哲 |
|  |  |

2021年 6 月 8日

目 录

[一、引言 2](#_Toc74045470)

[1.1系统目标 3](#_Toc74045471)

[1.2设计说明与任务分工 3](#_Toc74045472)

[二、系统设计与实现 3](#_Toc74045473)

[2.1系统总体架构 4](#_Toc74045474)

[2.2 Client设计 4](#_Toc74045475)

[2.2.1架构设计 4](#_Toc74045476)

[2.2.2工作流程 5](#_Toc74045477)

[2.3 Master Server设计 6](#_Toc74045478)

[2.4 Region Server设计 6](#_Toc74045479)

[2.5 miniSQL架构设计 7](#_Toc74045480)

[三、核心功能模块设计与实现 7](#_Toc74045481)

[3.1 Socket通信架构与通信协议 7](#_Toc74045482)

[3.1.1系统通信架构 7](#_Toc74045483)

[3.1.2通信协议设计 8](#_Toc74045484)

[3.2客户端缓存与分布式查询 8](#_Toc74045485)

[3.3数据分布与集群管理 9](#_Toc74045486)

[3.4均衡负载 9](#_Toc74045487)

[3.5副本管理与容错容灾 9](#_Toc74045488)

[四、系统测试 9](#_Toc74045489)

[五、总结 9](#_Toc74045490)

# 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

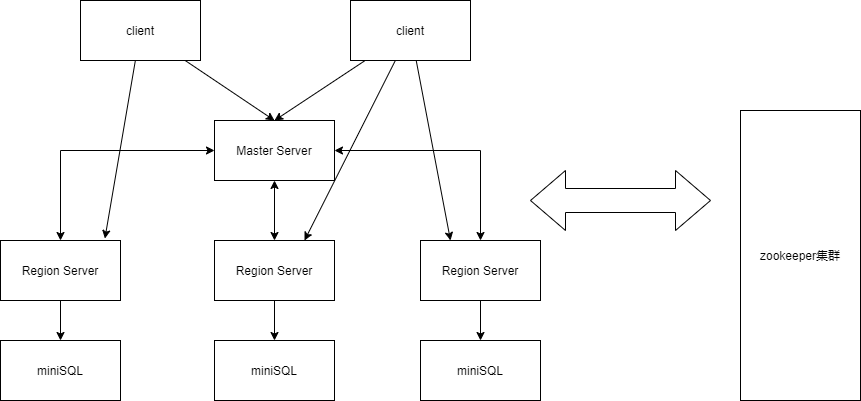
本系统由小组3位成员合作编写完成，使用Java开发，IDEA作为集成开发环境，Maven作为包管理工具，Github进行合作开发，每个组员都完成了目标任务，具体的分工安排如下：

|  |  |  |
| --- | --- | --- |
| 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数据分区、缓存机制的开发，miniSQL的开发 |
| 张琦 | 3180103162 | 从节点的开发，副本管理和容错容灾的开发，miniSQL的开发 |
| 聂俊哲 | 3180103501 | 主节点的开发，Zookeeper服务的开发，容错容灾和均衡负载功能的开发，miniSQL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：

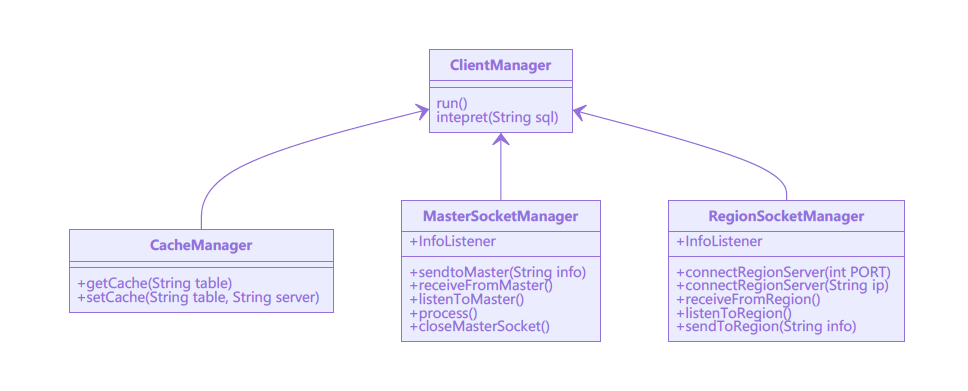


系统的源代码中将整个maven项目划分成三个模块，分别是Client，Master Server和Region Server，分别对应分布式MiniSQL系统的客户端、主节点和从节点，并且三者之间都可以在一定的通信框架下进行通信，同时**主节点和从节点通过Zookeeper集群，对数据表的信息进行统一的管理**。

## 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：

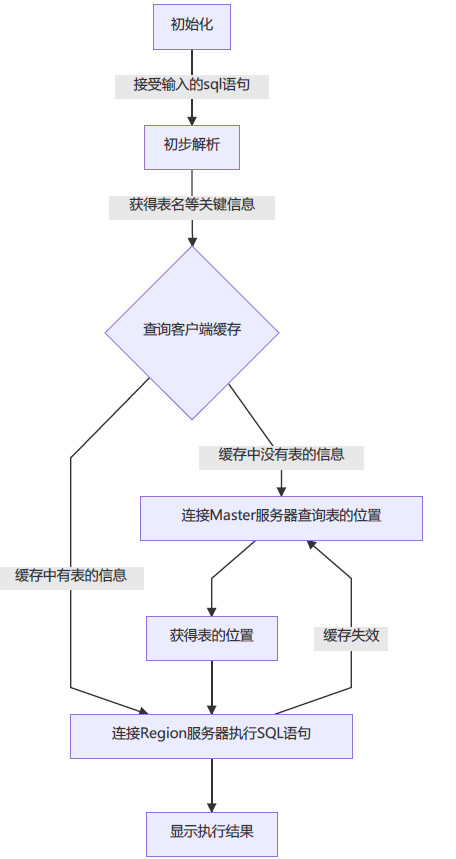


Client由一个ClientManager类负责管理，该类下面还有三个子功能的类，分别是：

* 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：

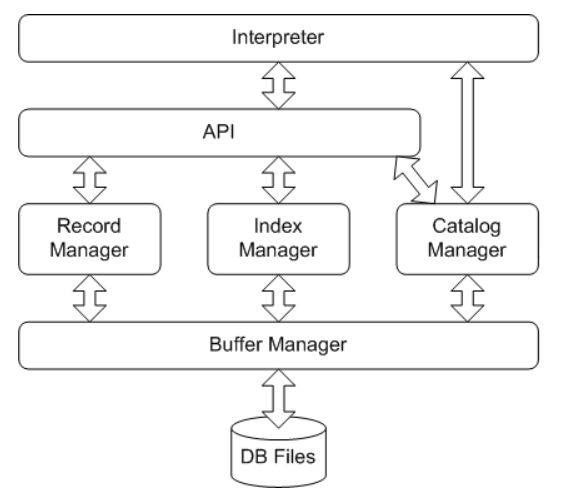


## 2.3 Master Server设计

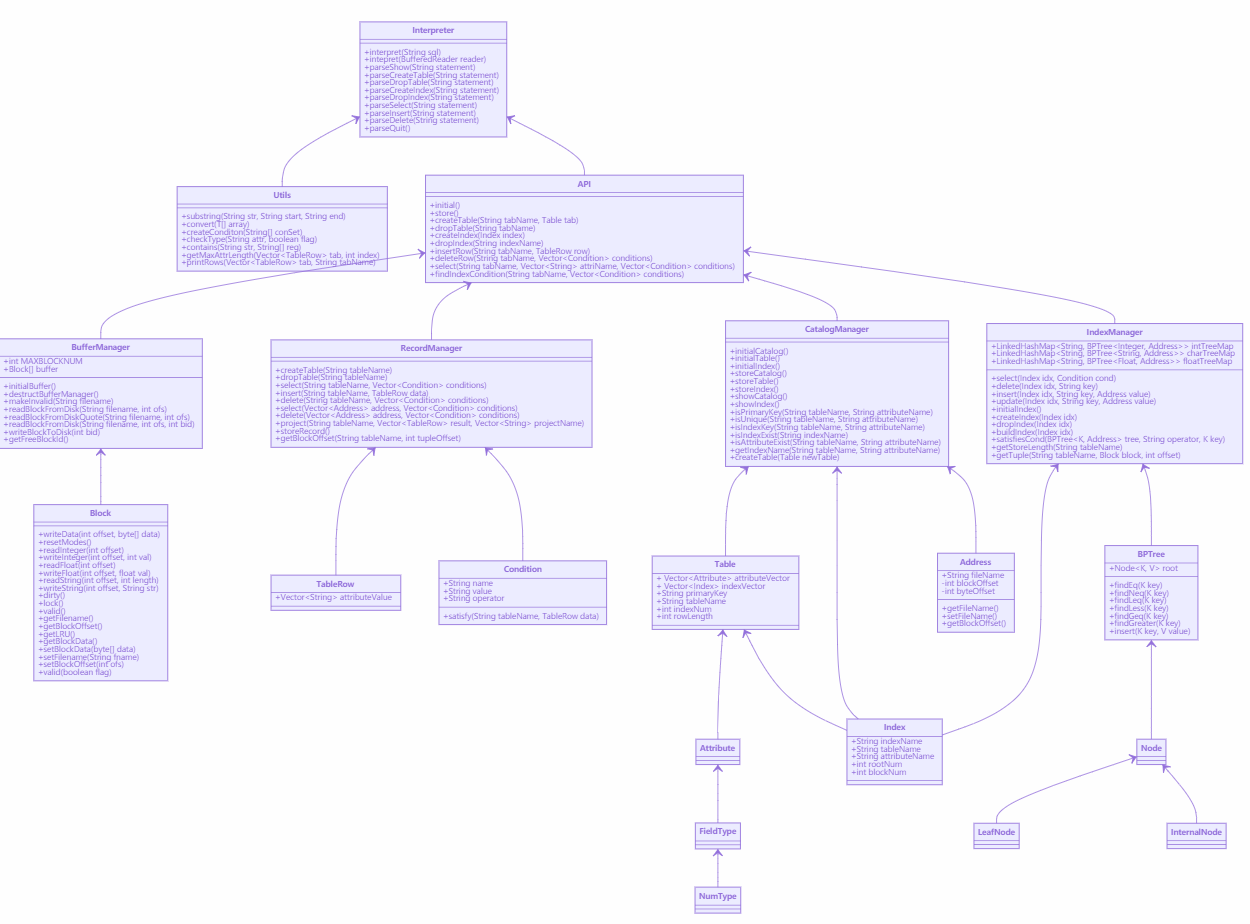
## 2.4 Region Server设计

## 2.5 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：



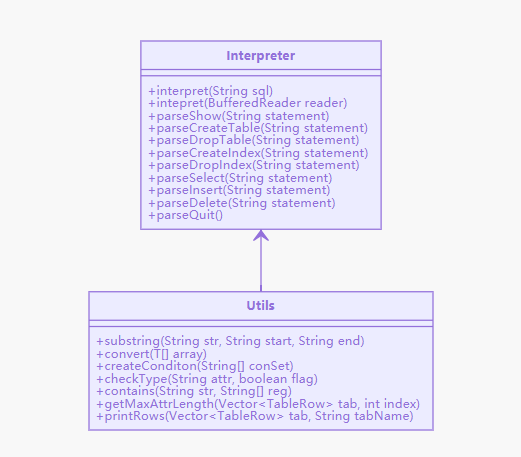
下面我们来具体介绍每个模块的设计思路和组成部分。

### 2.5.1 Interpreter层

Interpreter模块直接与用户交互，主要实现以下功能：

* 程序流程控制，即启动并初始化→【接收命令、处理命令、显示命令结果】循环→退出流程。
* 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。

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：

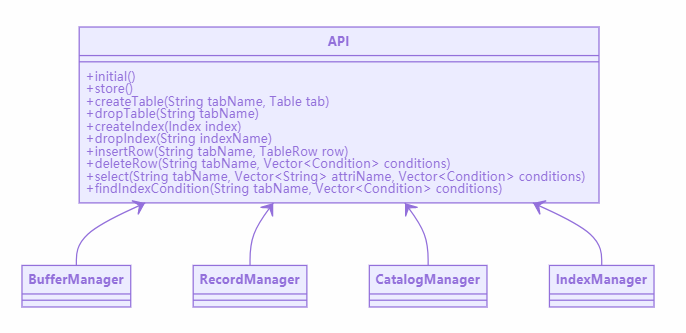


### 2.5.2 API层

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：



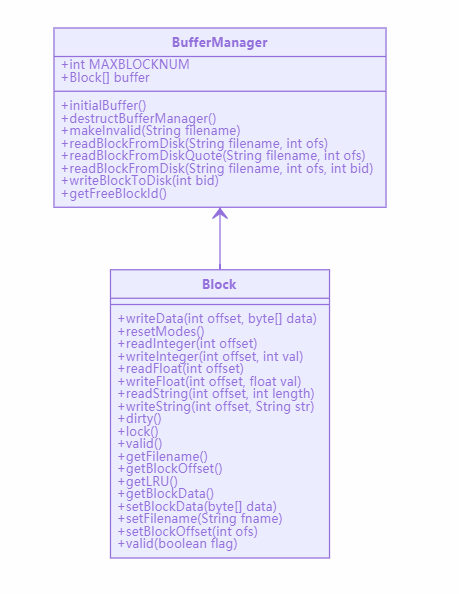
### 2.5.3 Buffer Manager

Buffer Manager负责**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
* 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，本项目中采用4096KB作为块的大小。

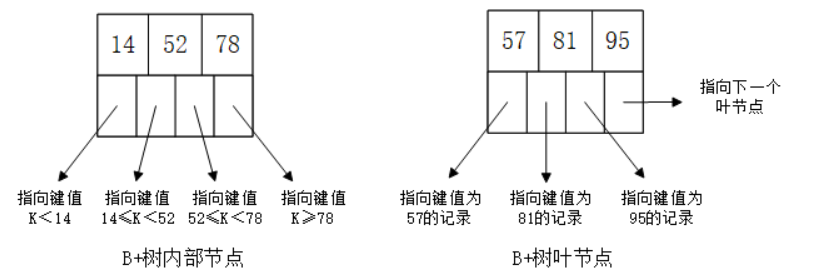
BufferManager模块的类图如下，该模块包含一个Block类和一个管理调度缓冲区的BufferManager：



### 2.5.4 Index Manager

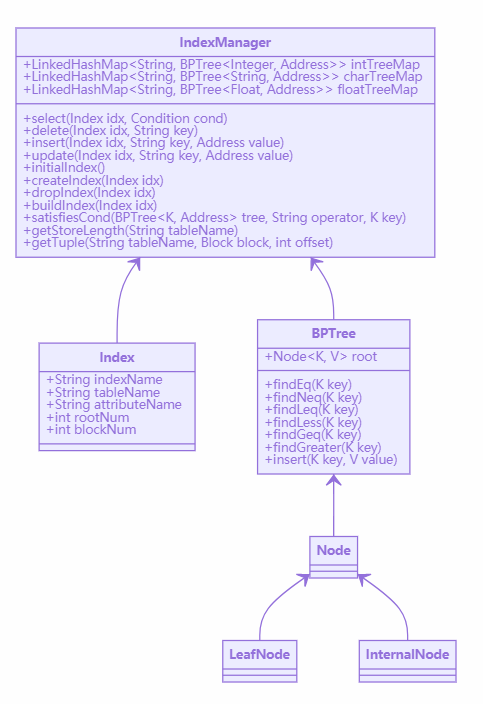
Index Manager负责B+树索引的实现，实现B+树的创建和删除（由索引的定义与删除引起）、等值查找、插入键值、删除键值等操作，并对外提供相应的接口。

B+树中索引节点大小应与缓冲区的块大小相同(4096KB)，B+树的叉数由节点大小与索引键大小计算得到。B+树的结构可以用下面的图来表示：



当然上面只是n=3的情况，在本系统的miniSQL模块开发中，一个节点的大小和Buffer模块的块大小是相同的，因此n会比3大得多。

整个IndexManager模块可以用如下类图来表示：



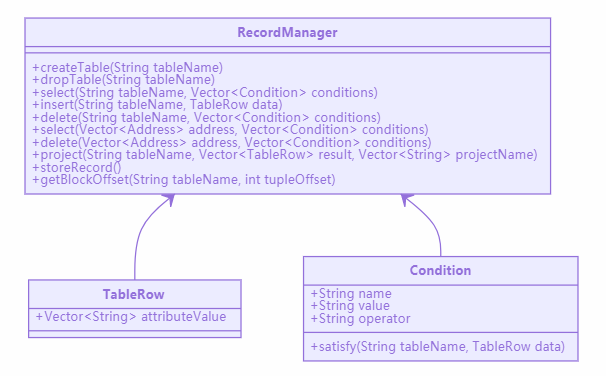
其中Index类是一个表示索引的数据结构，而BPTree具体实现了B+树的泛型，由一系列内部节点和外部节点组合而成。

### 2.5.5 Record Manager

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）。

数据文件由一个或多个数据块组成，块大小应与缓冲区块大小相同。一个块中包含一条至多条记录，为简单起见，只要求支持定长记录的存储，且不要求支持记录的跨块存储。

RecordManager与BufferManager有一定的联动，当需要对记录进行增删查改的时候，会在Buffer中对Block的数据进行修改，具体的类图如下：

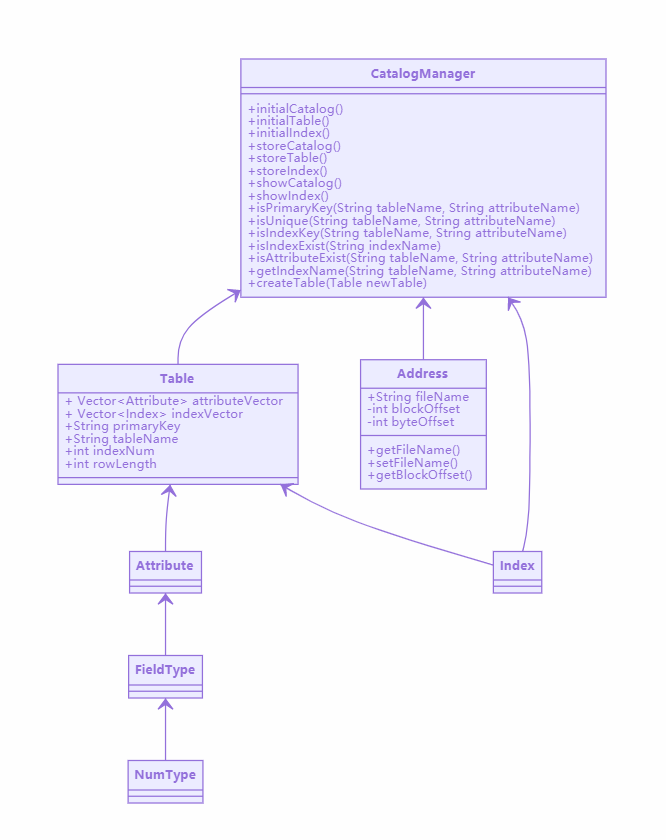


### 2.5.6 Catalog Mananger

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

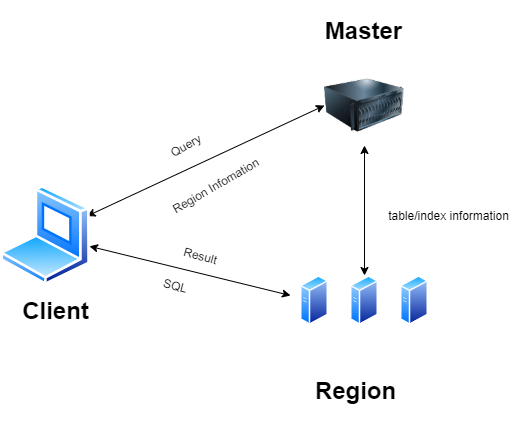
DB Files也就是数据库的文件系统，指构成数据库的所有数据文件，主要由记录数据文件、索引数据文件和Catalog数据文件组成。

# 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 | 用途 | 含义 |
| <client>[1] tableName | 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 | 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 |
|  |  |  |
|  |  |  |

## 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）

## 3.3数据分布与集群管理

## 3.4均衡负载

## 3.5副本管理与容错容灾

# 四、系统测试

# 五、总结