

大规模信息系统构建技术导论

分布式MiniSQL系统个人设计报告

2023学年 第一学期（上）

组员信息

|  |  |
| --- | --- |
| 学号 | 姓名 |
| 3200105109 | 孔郁杰 |
|  |  |

2023年 5月 24日

**1 引言**

* 1. 系统目标

本系统为《大规模信息系统构建技术导论》的课程项目，项目全称为分布式Minsiql系统，该旨在通过课程学习知识将之前数据库课程所制作的Minisql拓展为能够支持大规模数据的**分布式关系型简易数据库系统**

该系统主要包括Zookeeper集群、Client客户端、Master Server主服务器与Region Server区域服务器四个部分，支持包括简单SQL语句的解析与处理、容错容灾、主从备份、负载均衡等功能。

该系统使用Java作为语言开发，同时使用Github进行代码管理和协作开发，由小组五名成员共同协作完成。

1.2 个人任务说明

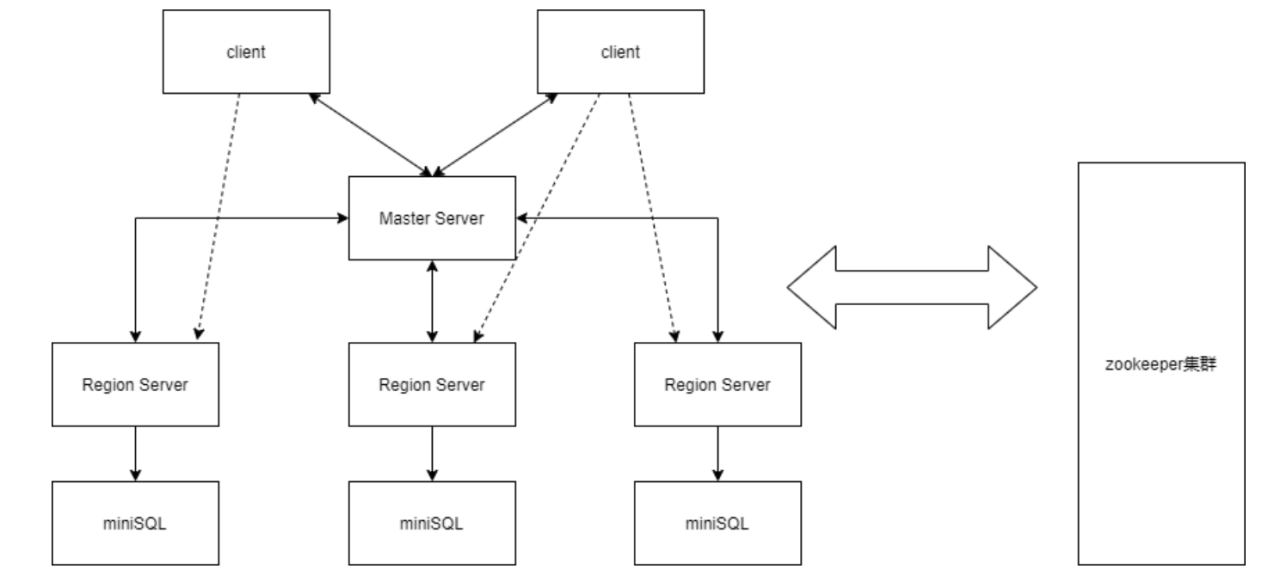
|  |  |  |
| --- | --- | --- |
| 成员姓名 | 学号 | 分工 |
| 孔郁杰 | 3200105109 | 负责Client的全部开发，包括Client与Zookeeper、Master、Region的连接与通信、Client本地缓存、SQL语句预处理、本地SQL文件运行、以及当任意服务器发生损坏时，Client产生的表现。 |

我在本系统开发中主要负责分布式数据库Client的总体开发以及其与Region、Master、Zookeeper三者间的正常、异常通信处理。同时，为了加快通信速度，Client本地会维护一个Cache缓存，用来保存本地已知的表和Region的映射，以此来减少向Master服务器请求的频率，从而降低服务器压力。

除此之外，我参与了整个系统的集成，参与了进度推进和通信规范使我们的整个系统集成性更加优越。

**2 系统设计**

2.1 系统总体架构

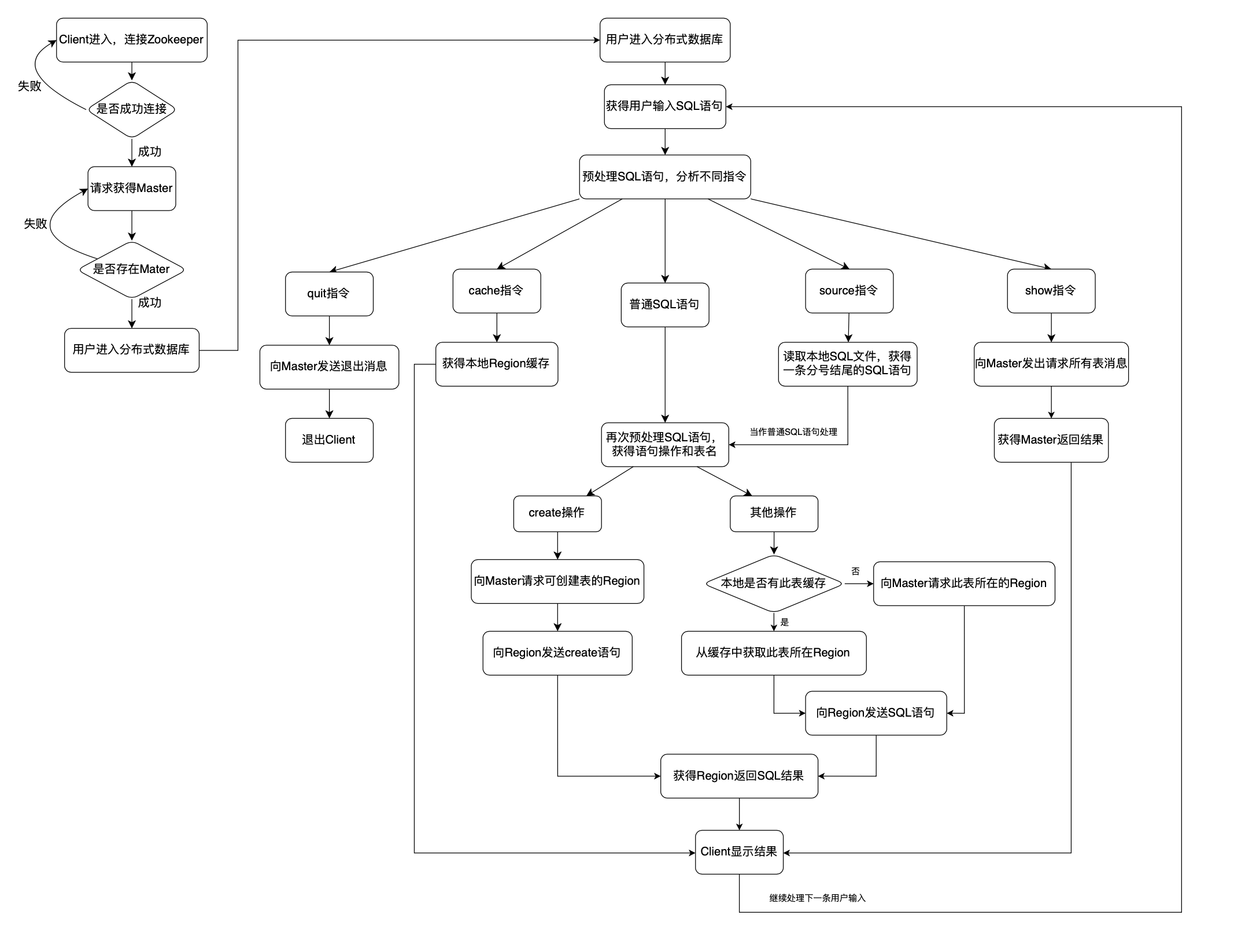


**图1 分布式MiniSQL总体架构**

该系统中Client主要负责处理以下多个部分的功能内容：

1. 主动连接Zookeeper集群，获取Master节点地址信息，用来与Master建立连接。
2. 主动连接Master节点，并维持与Master节点的Socket长连接，用来与Master进行通信。其中通信内容包括：
   1. 退出分布式数据库；
   2. 获得数据库中所有的表（即Master存下的表）；
   3. 获得用户输入SQL语句中的表所在的Region；
   4. 获得可以创建表的Region。
3. 主动连接Region节点，向用户输入表所在的Region发送SQL语句请求，以进行数据库分布式查询。其中在Client端存有一个本地缓存，用来减少查询时间，同时降低Master服务器压力。
4. 实现容错容灾适配，用来降低Client端用户对错、灾的敏感度。
5. 加载本地SQL文件，进行批量语句运行。

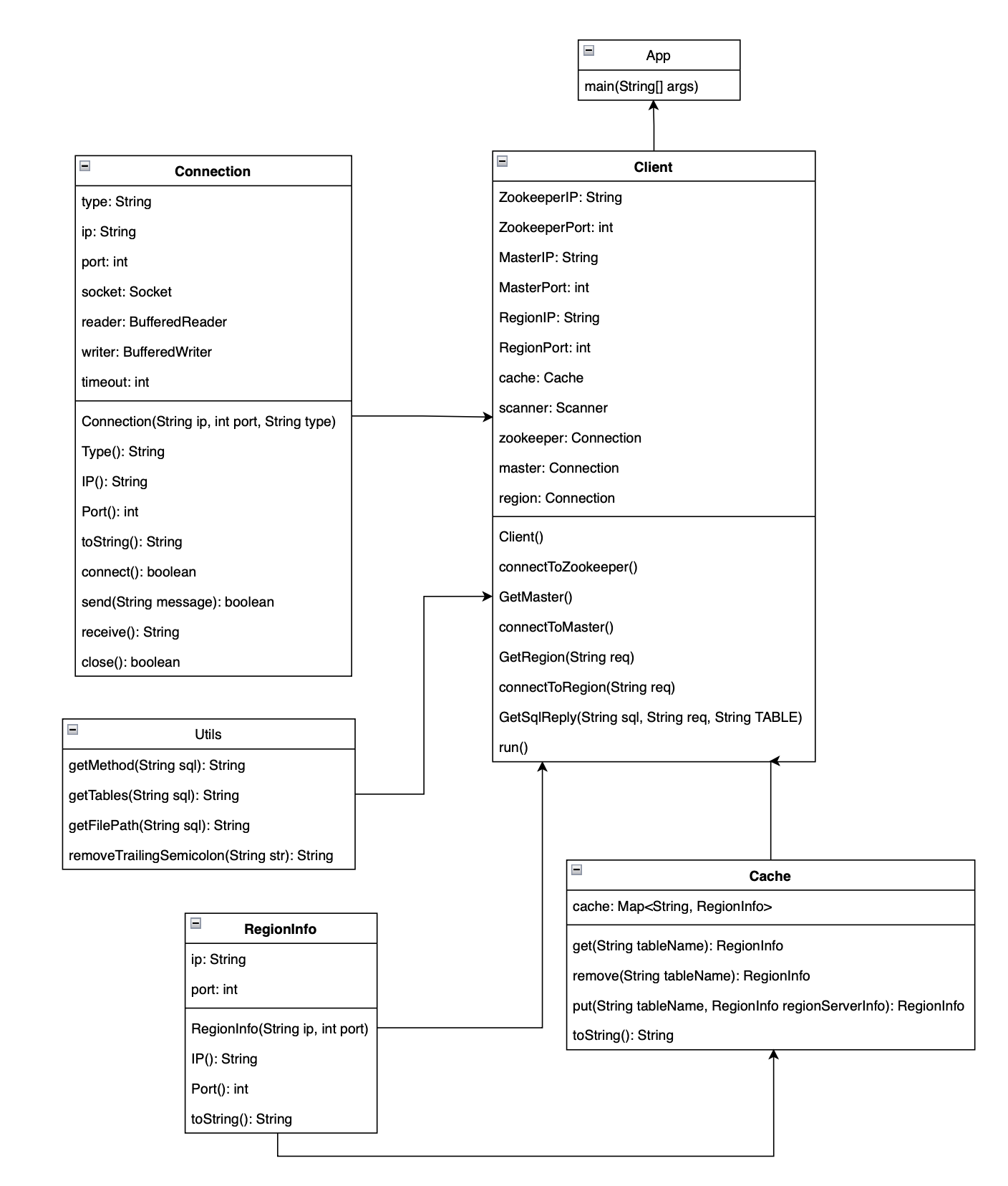
2.2 总体流程图



**图2 Client部分总体流程图**

2.3 Client设计

本系统的Client模块设计如下图所示：



**图3 Client总体设计类图**

* App类：系统的入口点，包含一个main方法。main方法创建一个Client对象并运行它。
* Client类：系统的核心类，负责与Zookeeper、Master和Region服务器进行通信。它包含以下方法：
  + connectToZookeeper()：连接到Zookeeper服务器。
  + GetMaster()：从Zookeeper服务器获取Master服务器的IP和端口。
  + connectToMaster()：连接到Master服务器。
  + GetRegion(String req)：向Master服务器发送请求，获取Region服务器的IP和端口。
  + connectToRegion(String req)：连接到Region服务器。
  + GetSqlReply(String sql, String req, String TABLE)：向Region服务器发送SQL语句，并获取执行结果。同时更新缓存。
  + run()：客户端的主要运行方法，循环接收用户输入的SQL语句，并根据需要与Master和Region服务器进行通信。
* Cache类：缓存类，用于存储表名与Region服务器信息（IP和端口）之间的映射。它包含以下方法：
  + get(String tableName)：根据表名获取Region服务器信息。
  + remove(String tableName)：从缓存中移除表名及其对应的Region服务器信息。
  + put(String tableName, RegionInfo regionServerInfo)：将表名及其对应的Region服务器信息添加到缓存中。
  + toString()：返回缓存的字符串表示。
* RegionInfo类：用于存储Region服务器的IP和端口。它包含以下方法：
  + IP()：返回Region服务器的IP。
  + Port()：返回Region服务器的端口。
  + toString()：返回Region服务器信息的字符串表示。
* Connection类：一个通用的网络连接类，用于与Zookeeper、Master和Region服务器进行通信。它包含以下方法：
  + connect()：连接到服务器。
  + send(String message)：向服务器发送消息。
  + receive()：从服务器接收消息。
  + close()：关闭与服务器的连接。
* Utils类：实用类，用于处理SQL语句。它包含以下方法：
  + getMethod(String sql)：从SQL语句中提取操作方法（如SELECT、INSERT等）。
  + getTables(String sql)：从SQL语句中提取表名。
  + getFilePath(String sql)：从SQL语句中提取文件路径（用于SOURCE操作）。
  + removeTrailingSemicolon(String str)：从字符串中删除末尾的分号。

**3 功能实现**

3.1 通信协议

Client实现了与Zookeeper、Region、Master三者的通信，在此制定针对不同节点的通信规范：

Client与Zookeeper的连接短连接，Client只需要在有获得Master需求时向Zookeeper发出请求。

Client与Master的连接是持久的长连接，因为Client需要向Master获得用户SQL语句中表所在的Region地址。

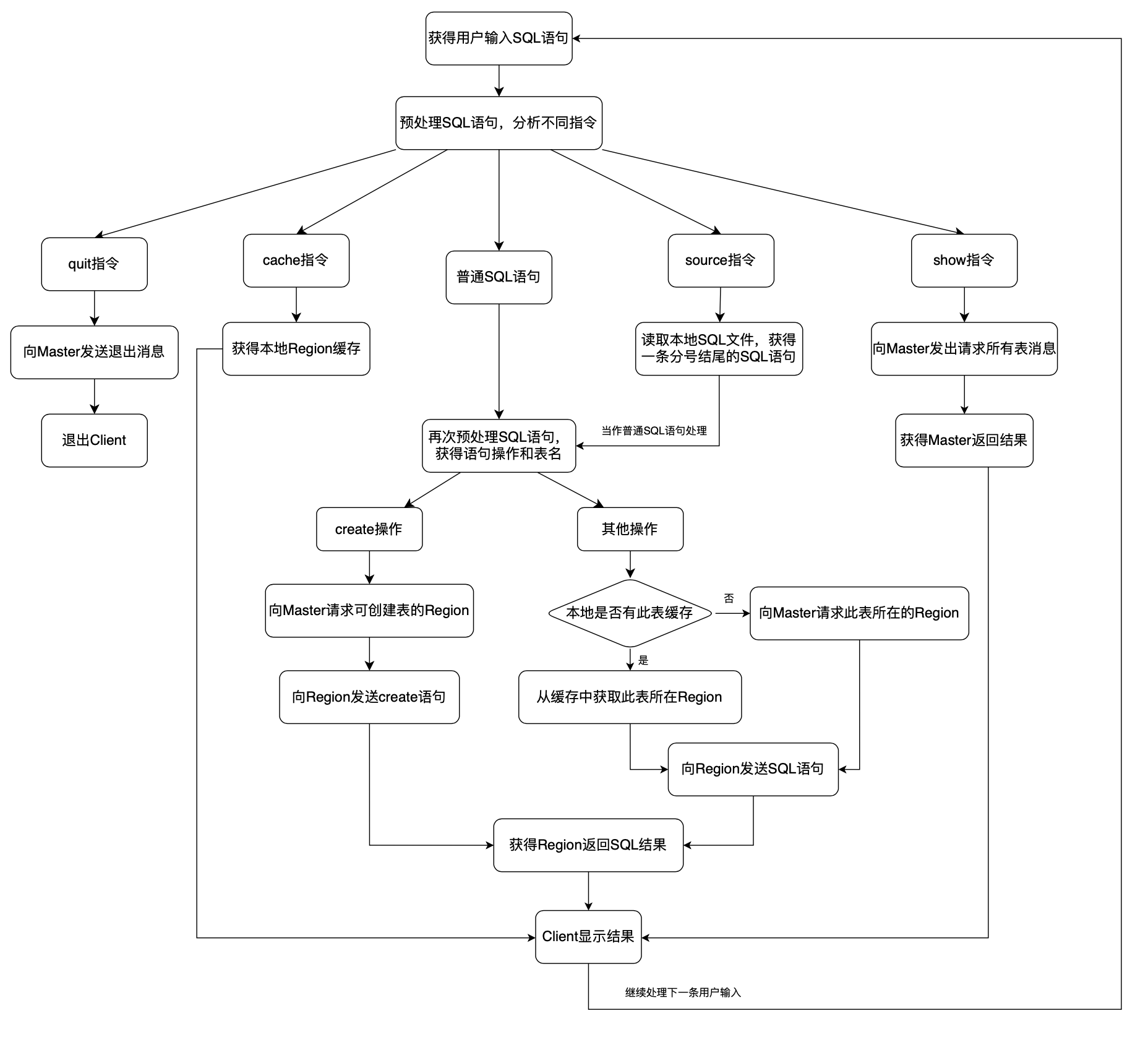
Client与Region的连接是短连接，我们设定为Client每一次的连接只会发送一条完整的SQL语句，Region在处理完该SQL语句并将处理结果发送回给Client后，应当直接关闭对应的socket连接。

基于以上讨论，Client的通信协议设计如下：

**表1 通信协议**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **服务器类型** | **请求类型** | **发送的消息示例** | **响应的消息示例** | **描述** |
| Zookeeper | 获取Master信息 | “client” | “192.168.1.1:1234” | 客户端请求Zookeeper提供Master服务器的IP和端口信息。 |
| Master | 获取Region信息 | “<get>table\_name” | “192.168.1.2:5678” | 客户端请求Master提供指定表所在的Region服务器的IP和端口信息。 |
| Master | 创建表 | “<create>table\_name” | “192.168.1.3:9012” | 客户端请求Master提供创建新表的Region服务器的IP和端口信息。 |
| Master | 显示所有表 | “<show>” | “{table1, table2, table3}” | 客户端请求Master提供当前系统中所有表的列表。 |
| Region | 执行SQL语句 | “SELECT \* FROM table\_name;” | “Result” | 客户端向Region服务器发送SQL语句，Region服务器执行并返回结果。 |

3.2 SQL语句查询



**图4 SQL语句查询流程图**

首先，客户端连接到Master服务器。

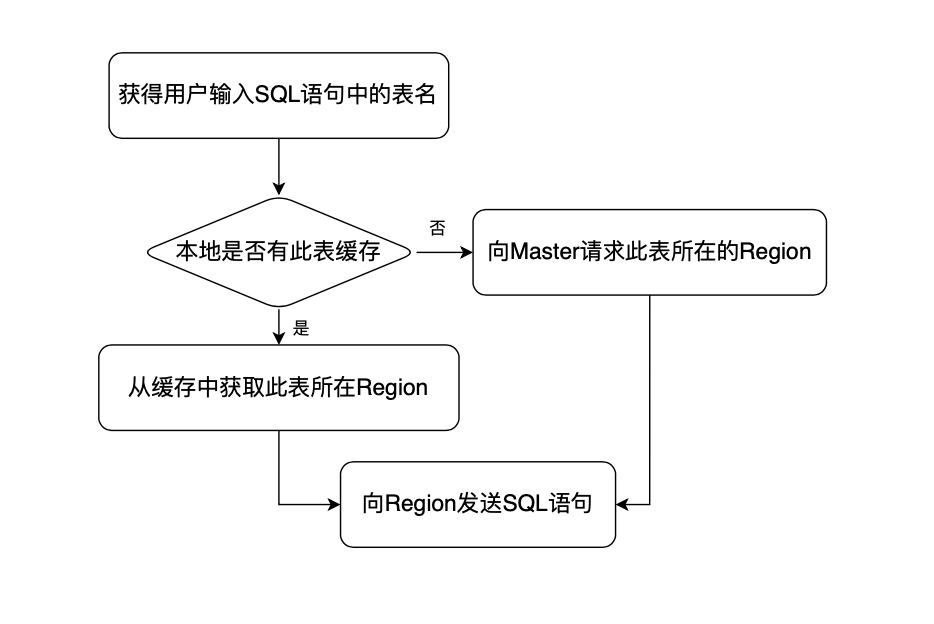
然后，客户端进入一个循环，等待用户输入SQL语句。

对于每个输入的SQL语句，客户端首先去除末尾的分号，然后提取SQL语句的方法（如SELECT、INSERT等）和涉及的表名。

* 如果用户输入的是"quit"，客户端将关闭连接并退出程序。
* 如果用户输入的是"cache"，客户端将打印当前的缓存信息。
* 如果用户输入的是"show"，客户端将向Master服务器发送一个请求，以获取当前的表信息，并将其打印出来。
* 如果用户输入的是一个包含"source"方法的SQL语句，客户端将从指定的文件中读取并执行每个SQL语句。对于文件中的每个SQL语句，客户端将根据其方法和表名进行相应的处理（如下所述）。
* 如果用户输入的是一个"create"方法的SQL语句，客户端将向Master服务器发送一个创建请求，然后连接到相应的Region服务器，并将SQL语句发送给Region服务器。最后，客户端将更新缓存并关闭与Region服务器的连接。
* 对于其他方法（如SELECT、INSERT等），客户端首先检查涉及的表是否存在于缓存中。如果存在，客户端将连接到缓存中指定的Region服务器，并将SQL语句发送给Region服务器。如果Region服务器无法连接，客户端将从缓存中移除该表，并重新连接到一个新的Region服务器。如果表不存在于缓存中，客户端将向Master服务器发送一个请求，以获取相应的Region服务器信息，然后连接到Region服务器并将SQL语句发送给Region服务器。最后，客户端将更新缓存并关闭与Region服务器的连接。

这个流程将不断重复，直到用户输入"quit"退出程序。

3.3 Client本地缓存



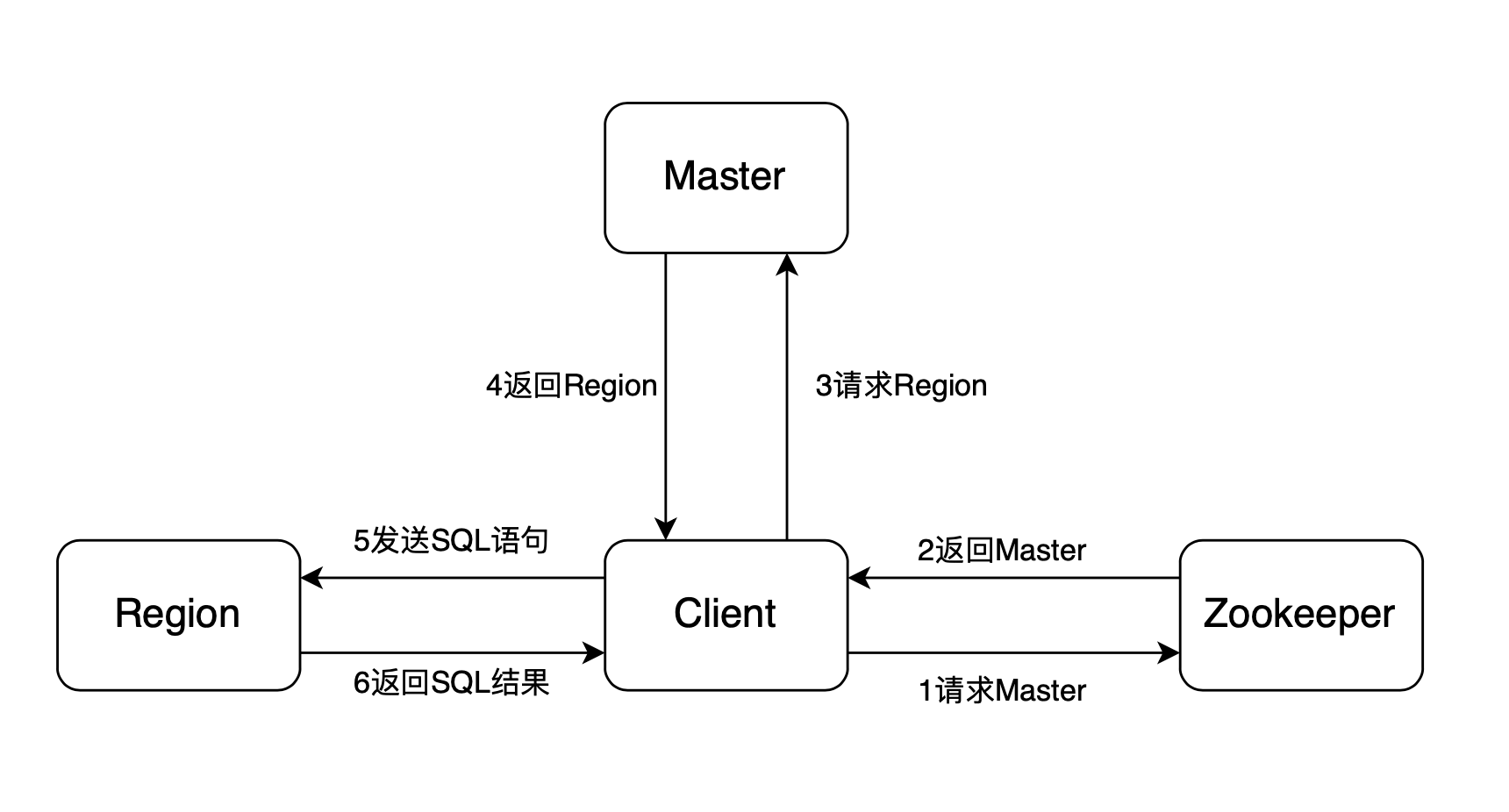
**图5 缓存流程图**

Client在本地会维护一个Cache，用于快速与Region建立连接，从而降低查询速度、减少Master服务器压力。

当用户输入SQL语句时，Client并不会直接向Master索要Region，而是在本地Cache中搜索：如果发现本地有缓存，则不经过Master，直接与Region进行通信，实现分布式请求；如果发现本地没有缓存，则向Master发出请求，获得Region后与Region建立通信，同时更新本地缓存，用来加快下一次查询。

当分布式数据库因为灾祸发生损坏时，Client会自动维护本地Cache，删除已经损坏的Region缓存，同时向Master获取备份后的Region所在地来更新缓存。

3.4 容错容灾



**图6 容错容灾流程图**

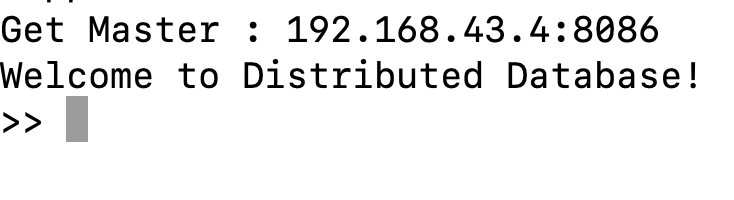
Client对于容错容灾，主要实现于：当对Zookeeper、Master、Region任意一个环节的请求出现错误时，也即上面三个服务器发生损坏时，Client端的表现。而我致力于实现用户无感知的Client端实现，即用户感知不到服务器端发生损坏。具体实现如下：

1. 对Zookeeper请求Master：如果失败，则等待一段时间后再次发出请求。
2. 获得返回Master：如果返回无效Master，则再次发出请求，直到获得有效Master为止。
3. 对Master请求Region：如果失败，则认为Master损坏，向Zookeeper发出请求Master。
4. 获得返回Region：如果返回无效Region，则再次发出请求，直到获得有效Region为止。
5. 对Region发送SQL语句：如果失败，则认为此Region损坏，向Master发出请求Region。同时需要对缓存进行更新，将损坏的Region删除。
6. 获得返回SQL结果：直接打印。
7. 若以上任何一个流程超时，则认为服务器端出现损坏，重新进入流程，等待服务器正确响应。

**4 测试与运行**

本项目测试中，我们一共启动了五个Server服务进程（其中一个注册为Master，四个注册为Region），一个Client服务进程以及一个Zookeeper服务进程。因为Client设计成用户无感知状态，所以测试显示较少。其中与Client相关的功能测试图片如下：

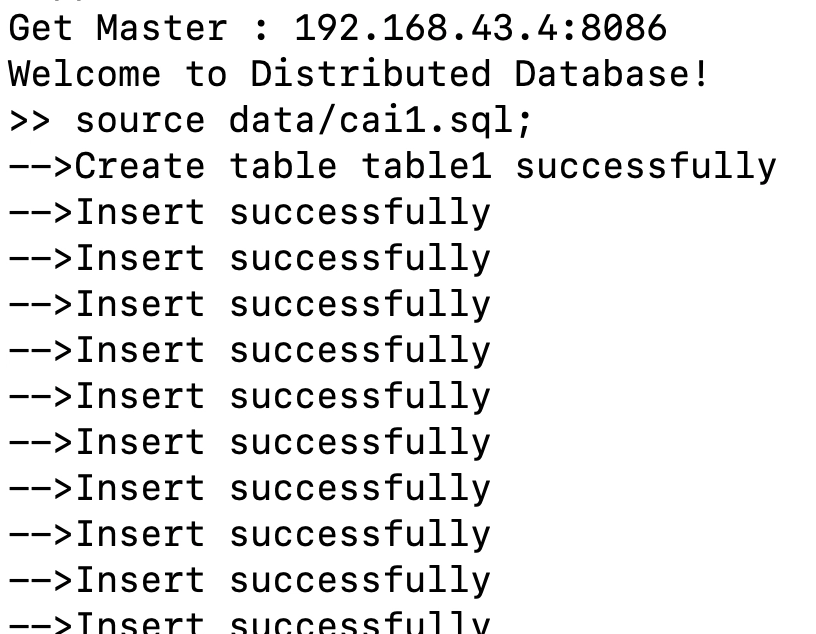
用户进入时，Client会自动请求Zookeeper、Master，然后进入系统：



**图7 登录上线**

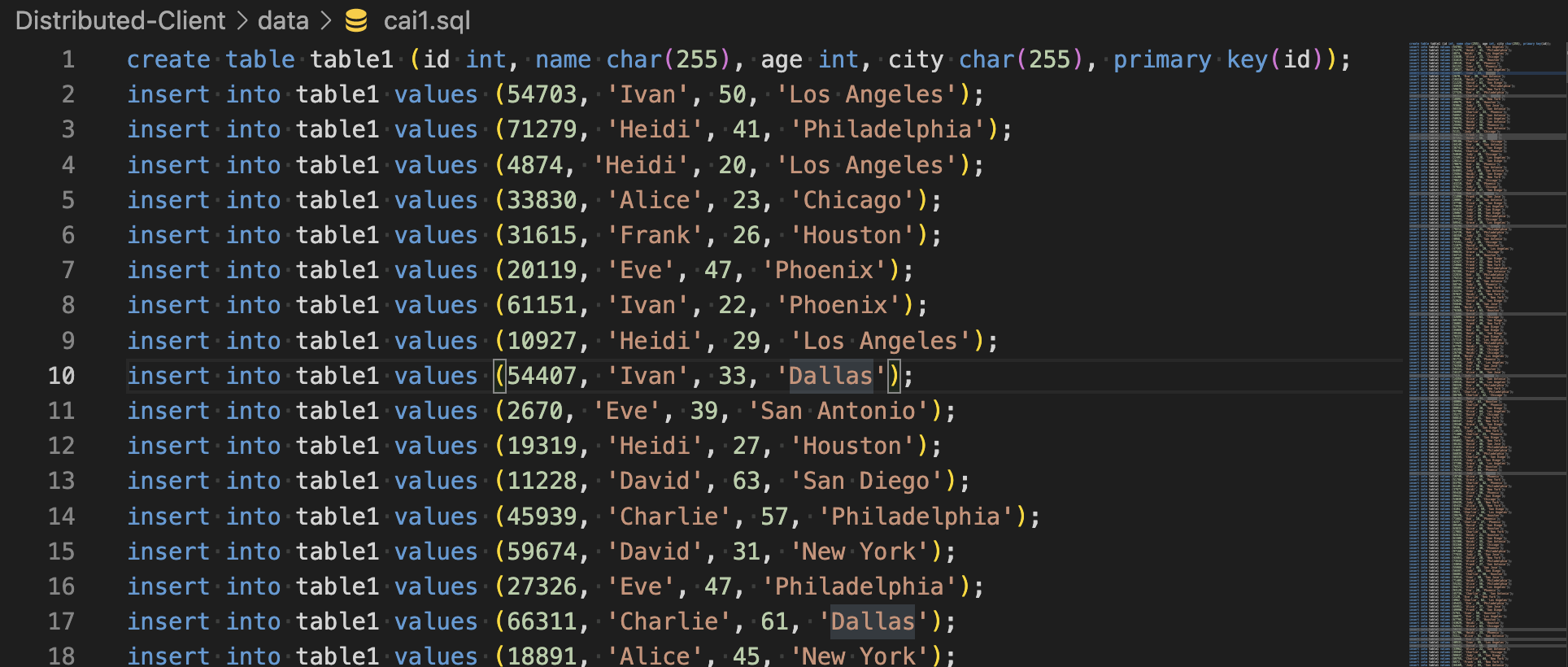
**用户输入source语句：**

成功进入系统后，用户输入source语句，进行SQL文件的批量执行：



**图8 用户输入source语句**

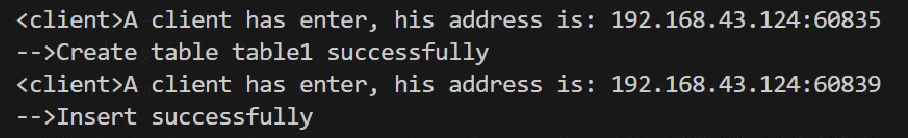
其中cai1.sql文件如下，创建了一个table1表，并且插入了10000条数据：



**图9 cai1.sql**

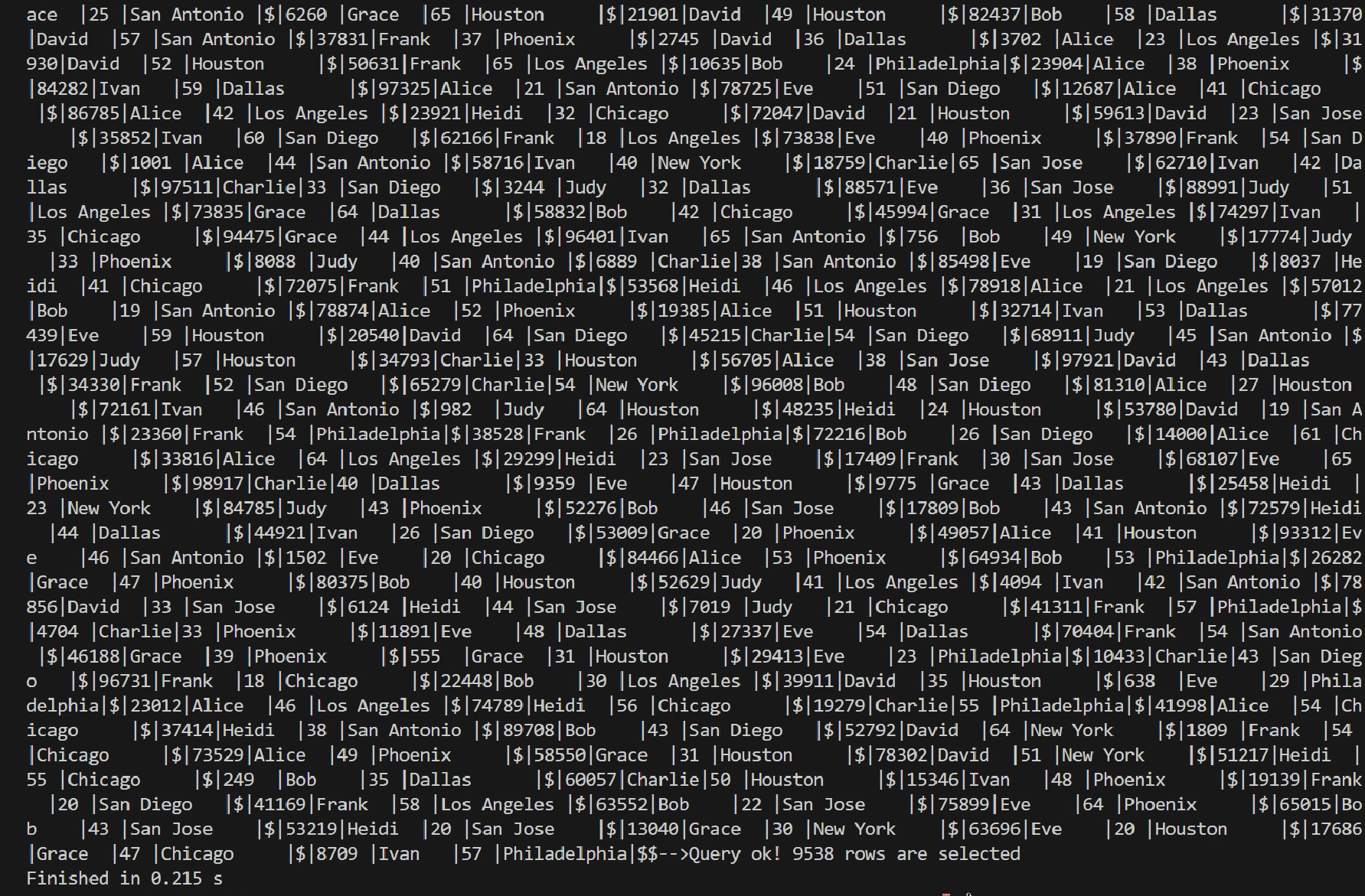
**处理SQL语句：**

Client会主动和Region建立Socket连接，并且发送对应的SQL语句，Region会使用minisql对其进行解析处理，执行相关语句，并且将返回结果打印并且回报给Client

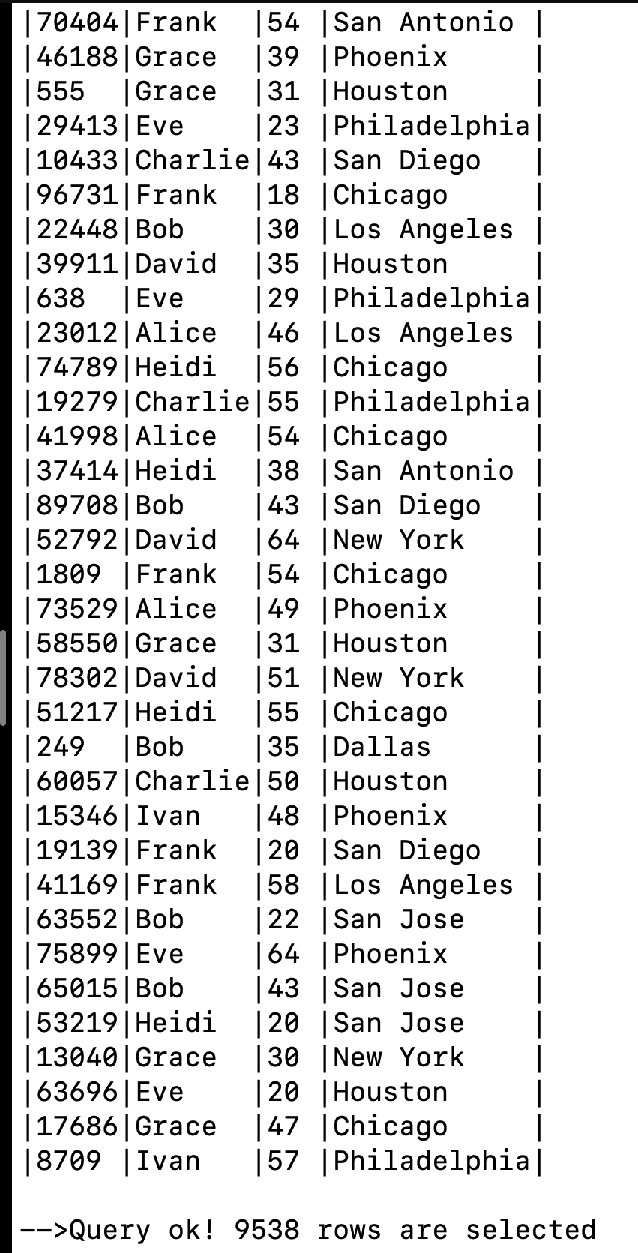


**图10 SQL测试 Region部分**

针对select等语句，考虑到write无法正常传输换行符，我们约定使用$代替\n进行传输。下图是对10000条插入语句（存在重复）的select结果，Client会对其进行解析，将$转换成\n进行显示。



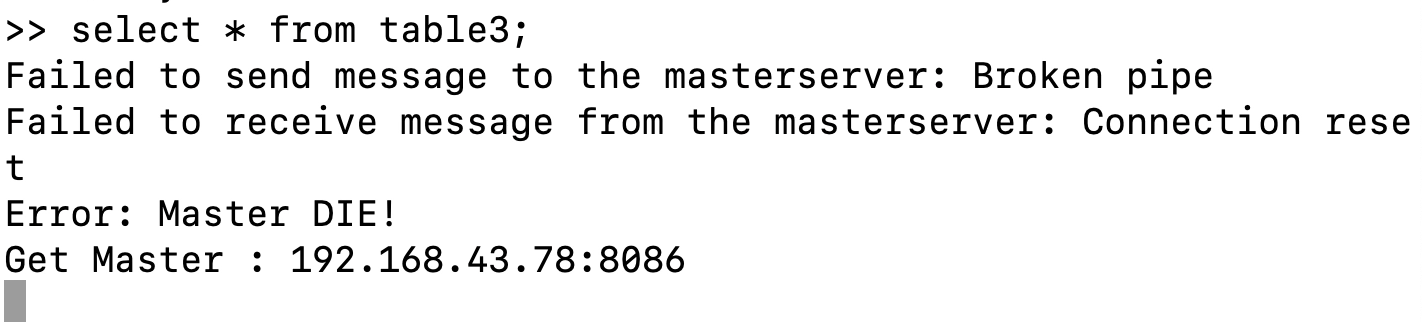
**图11 Select测试Region部分**



**图12 Select测试Client部分**

**容错容灾：**

Region、Master死亡后，Client会感知，并重新获取。



**图13 Master死亡，Client页面**