## 一、client模块

### 1.1 架构设计

Client相关的代码处在client/client.go中。 client向上负责与用户进行直接交互，向下与region和master进行连接。其主要的功能包括持有循环预处理输入语句（这里的预处理并非miniSQL中的interpreter部分），根据解析调用master或者region中接口进行执行语句，维护两个缓存表以供查询对应table所在的region。

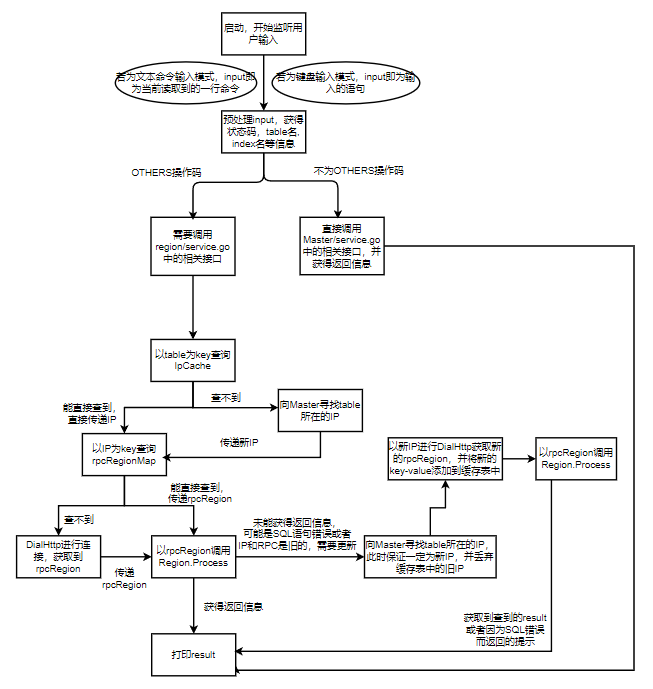
* 预处理输入：根据输入提取要执行的动作和必要参数，函数签名为func (client \*Client) preprocessInput(input string) (TableOp, string, string, error)，返回值分别为：要执行的操作状态码、涉及到的table名称、涉及到的index名称、预处理过程中是否出现了error。
* 调用接口：根据preprocessInput的返回值选择调用master或者region的接口，状态码一共有以下几种选择：

|  |
| --- |
| type TableOp int  const (      CREATE\_TBL = 0      DROP\_TBL   = 1      SHOW\_TBL   = 2      CREATE\_IDX = 3      DROP\_IDX   = 4      SHOW\_IDX   = 5      OTHERS     = 6  ) |

其中除了OTHERS以外其他的操作都会调用master的接口，OTHERS状态码下的操作 调用region的接口。

* 缓存机制：函数签名为func (client \*Client) updateCache(table string) string。为了加快大规模频繁操作时的速度，维护两个map，分别为table-IP表和IP-rpcRegion表，前一个表存储某一个表在哪个IP地址的机器上，后一个表表示IP地址及对应的rpcClient，用于直接和region进行信息的交互。前者可能为多对一，后者必为一对一。若某些IP地址对应的机器挂掉，表中的旧IP不会马上清除，而是通过将来可能有的二次查询而检测到并删除。

### 1.2 工作流程



## 二、 通信简介

### 2.1 RPC通信简介

远程过程调用（Remote Procedure Call，RPC）是一个计算机通信协议，该协议允许运行于一台计算机的程序调用另一台计算机的子程序。

在架构中，client与Master进行多对一的连接，Region与Master进行多对一的连接，而client与Region的连接不定（正常操作下随着SQL语句的执行应该会逐步变成一对多）。

建立rpcClient的过程与socket类似，只需要指定IP地址和端口号即可通过DialHttp函数获取到rpcClient，随后便可使用rpcClient对region/master的函数进行调用。

以client/client.go中建立同master进行交互的rpcClient为例:

|  |
| --- |
| rpcMas, err := rpc.DialHTTP("tcp", masterIP+MASTER\_PORT)  if err != nil {  ​ fmt.Printf("SYSTEM HINT>>> client connect error: %v", err)  }  client.rpcMaster = rpcMas |

在common.go中，专门写了保证不阻塞的封装好RPC调用函数TimeoutRPC，client等中的函数只需要调用该函数并指定需要调用的master/region中的函数名，调用该函数所需要的参数和返回值的地址，以及持续时间便可进行调用。

以client/client.go中调用Master的CreateTable为例：

|  |
| --- |
| call, err := TimeoutRPC(client.rpcMaster.Go("Master.CreateTable", &args, &ip, nil), TIMEOUT)  ​if err != nil { //超时错误  fmt.Println("SYSTEM HINT>>> timeout")  }  if call.Error != nil { //操作错误  fmt.Println("RESULT>>> create table failed")  } else { //操作正确  fmt.Println("RESULT>>> create table succeed, table in ip: " + ip)  } |

### 2.2 通信协议设计

由于封装了非阻塞的RPC调用函数TimeoutRPC，master/service.go和region/service.go中的大部分函数都是基于传入两个指针参数，一个作为封装好的函数固有参数，另一个作为返回值的策略。因此以下解释这些函数的作用，便基本可以解释清楚通信情况。

#### 2.2.1 master/service.go中的函数

master/service.go中的函数都是由client.go进行调用，自身再调用Region中的函数或者使用自己的缓存表（得到返回值）而返回的，其涉及的操作包含：

|  |  |  |
| --- | --- | --- |
| 函数签名 | 作用简介 | 返回值含义 |
| func (master \*Master) CreateTable(args \*TableArgs, ip \*string) error | 根据传入的table参数，在一个region上创建table | 新建table所在机器IP |
| func (master \*Master) DropTable(args \*TableArgs, dummyReply \*bool) error | 根据传入的table参数，调用相关region，删掉某一table | 无含义，不使用 |
| func (master \*Master) ShowTables(dummyArgs \*bool, tables \*[]string) error | 通过缓存表，返回所有已建的table | 缓存表中所有的table |
| func (master \*Master) CreateIndex(args \*IndexArgs, ip \*string) error | 根据传入的index参数，在某一table上创建索引 | 新建index所在机器IP |
| func (master \*Master) DropIndex(args \*IndexArgs, dummyReply \*bool) error | 根据传入的index参数，丢弃某一table上的索引 | 无含义，不使用 |
| func (master \*Master) ShowIndices(dummyArgs \*bool, indices \*map[string]string) error | 通过缓存表，返回所有已建的index和对应的table | 已建的index和对应的table |
| func (master \*Master) TableIP(table \*string, ip \*string) error | 根据table名，通过缓存表返回其所在IP | table所在机器IP |

#### 2.2.2 region/service.go中的函数

该部分的函数由master和client进行调用，起到备份策略和处理select, insert, delete操作并获得返回值的作用。

|  |  |  |
| --- | --- | --- |
| 函数签名 | 作用简介 | 返回值含义 |
| func (region \*Region) Process(input \*string, reply \*string) error | client调用，使用本机或者备份机器最终获得SQL操作的结果，过程中调用了processSQL | SQL操作得到的结果 |
| func (region \*Region) AssignBackup(ip \*string, dummyReply \*bool) error | 备份，调用DownloadSnapshot下载备份 | 未使用，无意义 |
| func (region \*Region) RemoveBackup(dummyArgs, dummyReply \*bool) error | 删除本region的备份IP | 删除成功与否 |
| func (region \*Region) DownloadSnapshot(ip \*string, dummyReply \*bool) error | 调用ftp中的函数下载文件到新IP上 | 未使用，无意义 |
| func (region \*Region) processSQL(sql string) (string, error) | 向下面向minisql的API部分，根据sql命令获得结果 | SQL操作获得的结果 |