浙江大学

大规模信息系统构建技术导论

分布式 MiniSQL 系统报告

2022 学年 春 学期

学号	3190103313
学生姓名	钱星屹
所在专业	软件工程
所在班级	软工 1901

1　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 ETCD 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、负载均衡、副本管理、容错容灾等功能。

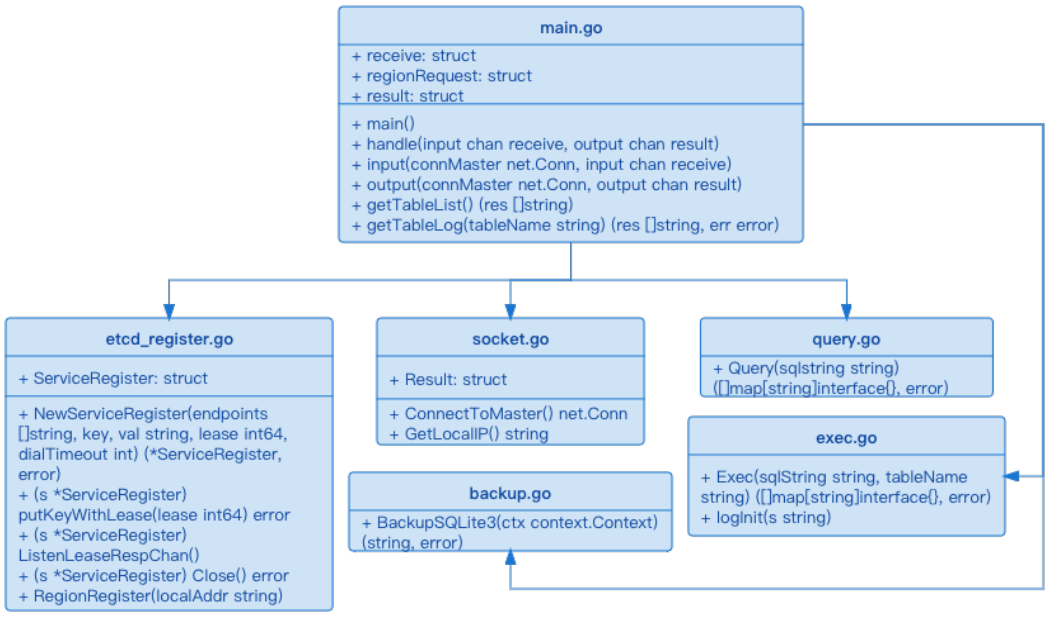
本系统使用 Go 语言开发，并使用 Github 进行版本管理和协作开发，由小组内的五名成员共同完成，每个人都有自己的突出贡献。

2　个人设计

2.1 Region 设计

2.1.1 架构设计

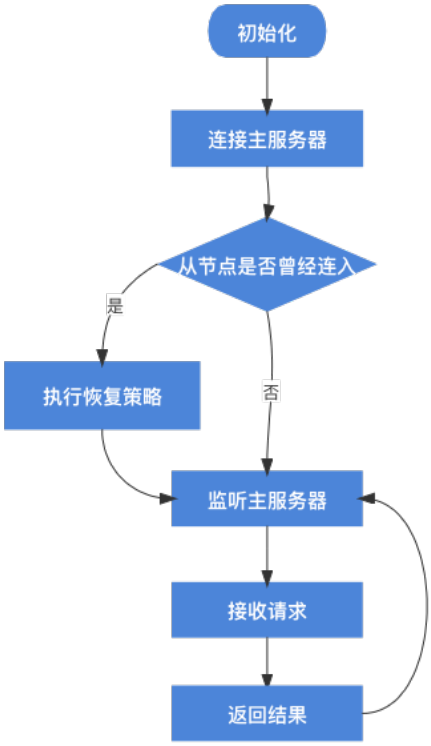
Region Server 设计如下图所示：



从服务器管理自己的数据库，并和主服务器通信：

- main.go 中定义了 Region 的主要操作，处理来自 Master 的请求、返回请求处理结果，其中调用了其他文件中的函数实现对本地数据库的操作，同时包含对表信息的 Log 登记，便于从节点宕机时实行备份策略。核心是开启了三个线程一个用于对 region 接受到的信息做一个初步的处理;第二个线程则是根据前一个现场的处理结果选择语句的处理方式,具体调用 sqlite 包中的函数;第三个线程则是得到结果，返回信息。
- etcd_register.go 负责使用 etcd 进行服务注册，便于主节点监听当前节点，实现集群管理。
- socket.go 给出了与主节点建立连接的函数，负责从节点通讯。
- query.go 负责对本地数据库进行查询、修改等操作。query 的操作与 exec 不同，有可能会返回大量的数据，因此在这里要对查询的结果进行一个处理。
- backup.go 管理了本地数据库中每个表的操作记录，便于从节点宕机时实行备份策略。
- exec.go 在接受到来自主节点的备份命令后，根据传来的相关表的操作记录，逐条实现操作，完成表的备份。在所有有关 exec 的操作都会影响到数据库的数据，所以进行 exec 的语句都会备份到 backup 文件夹下的对应表名.txt 文件中，为之后的宕机恢复做准备。

2.4.2 工作流程



2.4.3 关键代码

```go
func main() {
	defer sqlite.Close()
	StatementChannel := make(chan receive, 500)
	OutputChannel := make(chan result, 500)
	QuitChan = make(chan string)
	fmt.Println( a...: ">Region: 启动中...")
	connMaster := sqlite.ConnectToMaster()
	//sqlite.Exec("delete from sqlite_master where type in ('table', 'index', 'trigger');", "deleteAll")
	go sqlite.RegionRegister(connMaster.LocalAddr().String())
	fmt.Println(connMaster.LocalAddr())
	defer connMaster.Close()
	go input(connMaster, StatementChannel)
	go handle(StatementChannel, OutputChannel)
	go output(connMaster, OutputChannel)
	for {
	}

}
```

main.go

主要包含创建 channel 和线程运行的过程，具体的代码作为附件提交，此处不做展示。

```go
	defer backFile.Close()
	if strings.Contains(strings.ToLower(sqlString), substr: "create") {
		logInit(tableName, t: 0)
	} else {
		logInit(tableName, t: 1)
	}
	_, err := db.Exec(sqlString) // ignore_security_alert
	if err != nil : nil, err 
	write := bufio.NewWriter(backFile)
	write.WriteString(sqlString + "\n")
	write.Flush()
```

exec.go

执行 exec 类型的语句会在执行完成后写入对应的文件用于恢复

```go
for rows.Next() {
        s := make([]interface{}, l)
        ps := make([]*interface{}, l)
        c := make([]interface{}, l)
        for i := 0; i < l; i++ {
                ps[i] = &s[i]
                c[i] = ps[i]
        }
        err = rows.Scan(c...)
        if err != nil : nil, err
        temp := make(map[string]interface{})
        for i, Col := range input {
                temp[Col] = s[i]
        }
        m = append(m, temp)
        //resString = fmt.Sprintf("%s\n%v", resString, s)
}
```

query.go

query 的核心处理如下，涉及到指针和 interface 之间的一个转换

3 总结

本次工作主要负责 region 部分，工作独立性比较高，代码实现较为自由，独立的设计了 region 运行的架构，收获颇丰。同时与 master 通信对接，锻炼了我的团队合作能力，对 go 语言有了更深的理解，收益匪浅。