

# Lab3 raft选举说明文档

补充相关结构体

主要函数实现

ticker()函数

RequestVote()函数

sendRequestVote()函数

sendAppendEntry()函数与AppendEntry()函数

运行结果

## 补充相关结构体

raft

```
// A Go object implementing a single Raft peer.
type Raft struct {
    mu          sync.Mutex           // Lock to protect shared access to this peer's state
    peers      []*labrpc.ClientEnd // RPC end points of all peers
    persister  *Persister          // Object to hold this peer's persisted state
    me         int                 // this peer's index into peers[]
    dead       int32              // set by Kill()

    // Your data here (2A, 2B, 2C).
    // Look at the paper's Figure 2 for a description of what
    // state a Raft server must maintain.
    currentTerm int //当前term
    voteFor     int //投票对象
    log         []LogEntry

    commitIndex int
    lastApplied int

    nextIndex []int
    matchIndex []int

    myStatus    int //当前raft状态
    voteTimeout time.Duration //变成candidate的超时时间, 不同raft不同
    timer       *time.Ticker
    applyChan   chan ApplyMsg
}
```

RequestVoteArgs与RequestVoteReply

```

168
169 type RequestVoteArgs struct {
170     // Your data here (2A, 2B).
171     Term          int //candidate's term
172     CandidateId    int //candidate requesting vote
173     LastLogIndex  int
174     LastLogTerm   int
175 }
176
177 // example RequestVote RPC reply structure.
178 // field names must start with capital letters!
179 type RequestVoteReply struct {
180     // Your data here (2A).
181     Term          int //currentTerm, for candidate to update itself
182     VoteGranted    bool //true means candidate received vote
183 }
184

```

## AppendEntryArgs与AppendEntryReply

```

184
185 // 心跳结构体
186 type AppendEntryArgs struct {
187     Term          int //Leader's term
188     LeaderId       int
189     PreLogIndex    int
190     PreLogTerm     int
191     LeaderCommit   int
192 }
193
194 type AppendEntryReply struct {
195     Term          int //current term, for leader to update itself
196     Success        bool
197 }
198

```

是根据论文中表2进行补充的

## 主要函数实现

### ticker()函数

这个函数是主入口，根据rf的当前状态是follower、leader、candidate来做出对应的响应，若是follower，则等待electionTimeOut，变为candidate；若是candidate，则广播选举，选出新的leader；若是leader，则定时发送心跳。

具体的实现详见代码

### RequestVote()函数

分两种情况讨论，请求term小于当前term，说明是过期的请求；请求term大于当前term，则投出自己一票；若请求term等于当前term，则判断当前是否已经投过票，若已经投过票，不做任何处理，若还未投票，则投出自己一票

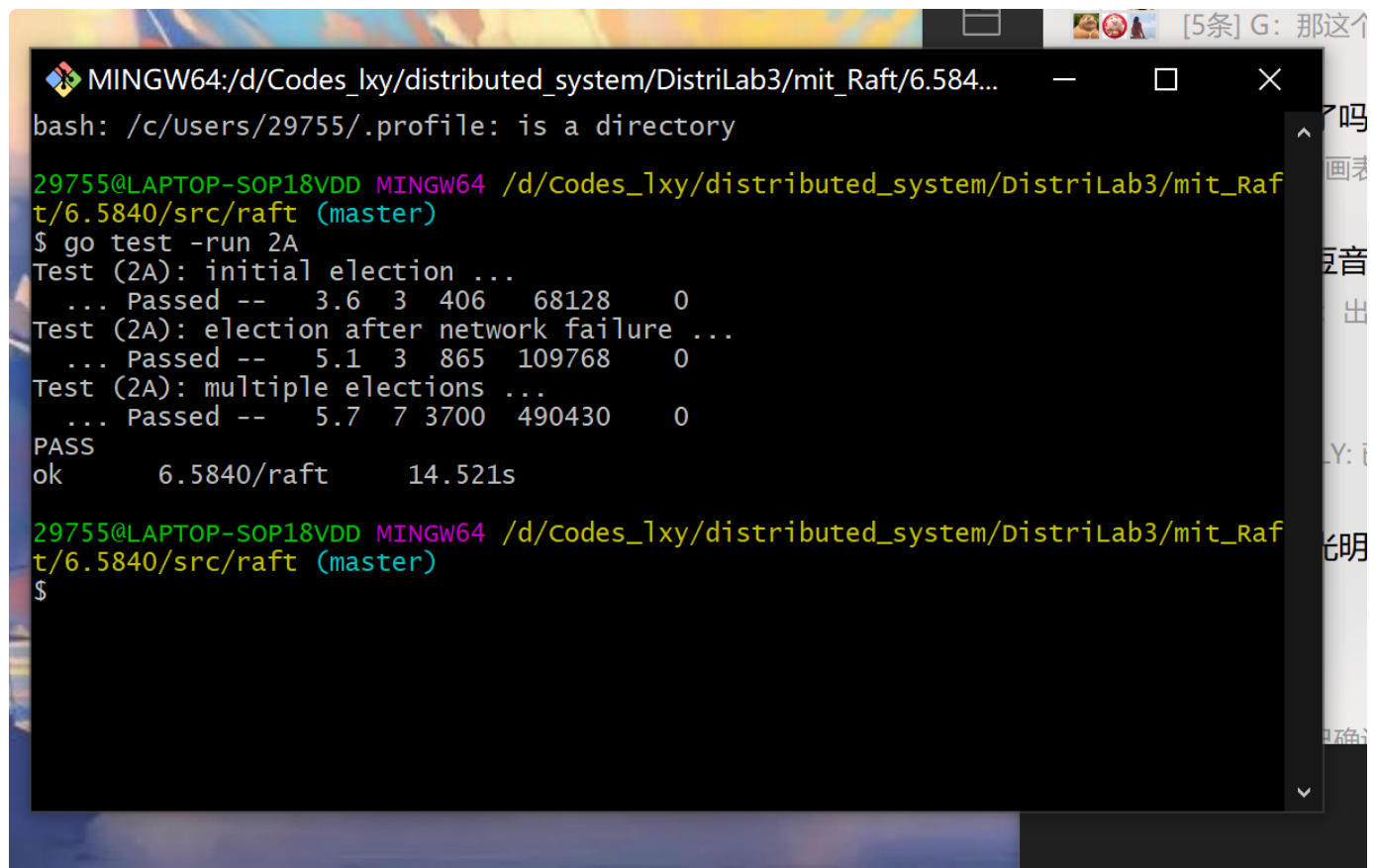
## sendRequestVote()函数

若当前term为最新，则根据接收到的投票结果判断投票数是否已到达最大投票数的一半，若达到，成为新的leader，否则接收到新leader心跳，变成follower

## sendAppendEntry()函数与AppendEntry()函数

逻辑比较简单，不再赘述

## 运行结果



```
MINGW64:/d/Codes_lxy/distributed_system/DistriLab3/mit_Raft/6.5840...
bash: /c/Users/29755/.profile: is a directory

29755@LAPTOP-SOP18VDD MINGW64 /d/Codes_lxy/distributed_system/DistriLab3/mit_Raft/6.5840/src/raft (master)
$ go test -run 2A
Test (2A): initial election ...
... Passed -- 3.6 3 406 68128 0
Test (2A): election after network failure ...
... Passed -- 5.1 3 865 109768 0
Test (2A): multiple elections ...
... Passed -- 5.7 7 3700 490430 0
PASS
ok      6.5840/raft      14.521s

29755@LAPTOP-SOP18VDD MINGW64 /d/Codes_lxy/distributed_system/DistriLab3/mit_Raft/6.5840/src/raft (master)
$
```

测试全部通过