# DisEX01 Raft Leader Election

刘凡维 MF1933058

1. 实验环境

Ubuntu 18.4.1下使用Go语言完成



1. 问题分析

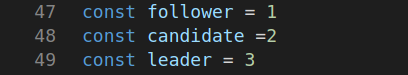
首先下载NJU-DisSys-2017-master文件，在src/raft目录下的raft.go文件中通过补全代码来实现Raft Leader Election功能。主要任务分为以下三个方面：

* 补全RequestVoteArgs和RequestVoteReply的数据结构
* 修改Make()函数，实现当出现超时的情况时，通过发送RequestVote RPC的方式来开始一次Election
* 定义AppendEntries的数据结构，并实现AppendEntries RPC handler

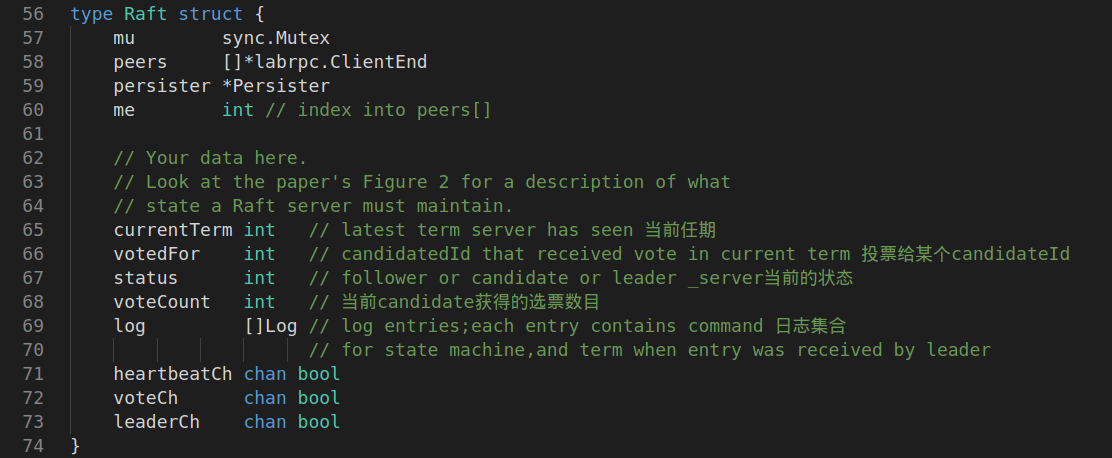
1. Raft 结构与算法设计

代码的最终实现主要参考了In Search of Understandable Consensus Algorithm的Section 5的部分

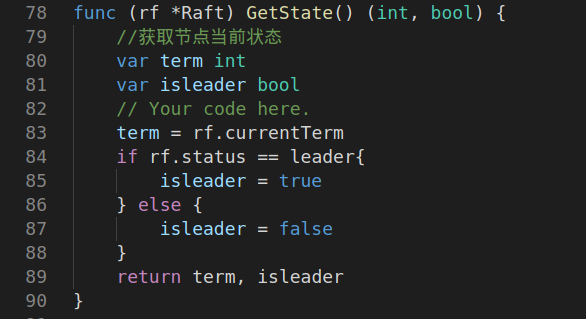
**定义follower、candidate、leader三种状态**

****

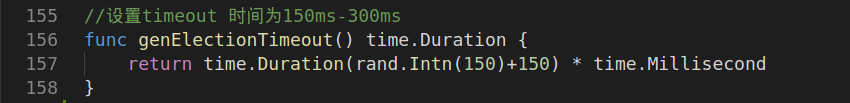
**补全Raft结构体**



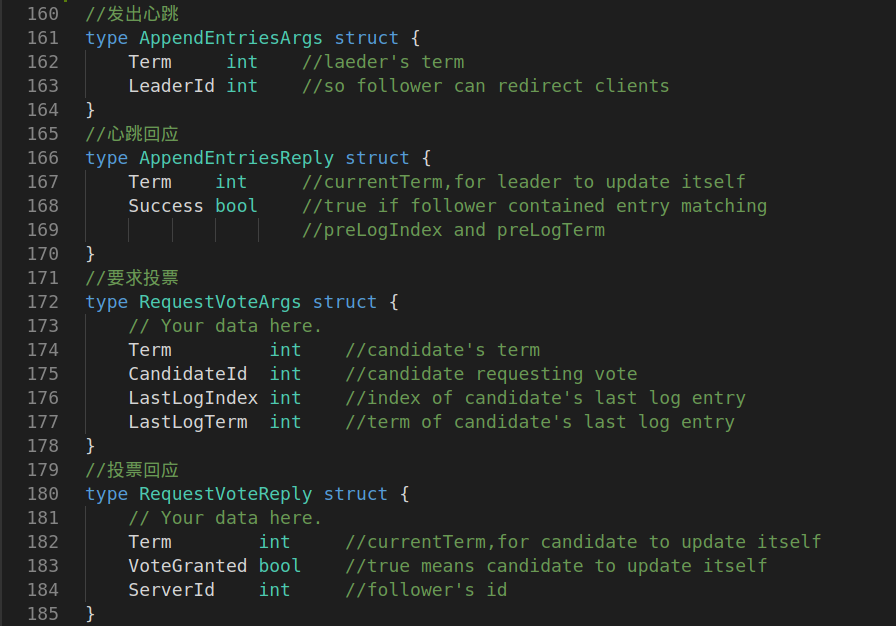
**获取节点当前状态（判断是否为leader）**



**随机生成timeout，150ms-300ms**



**RequestVote结构体与AppendEntries结构体设计**



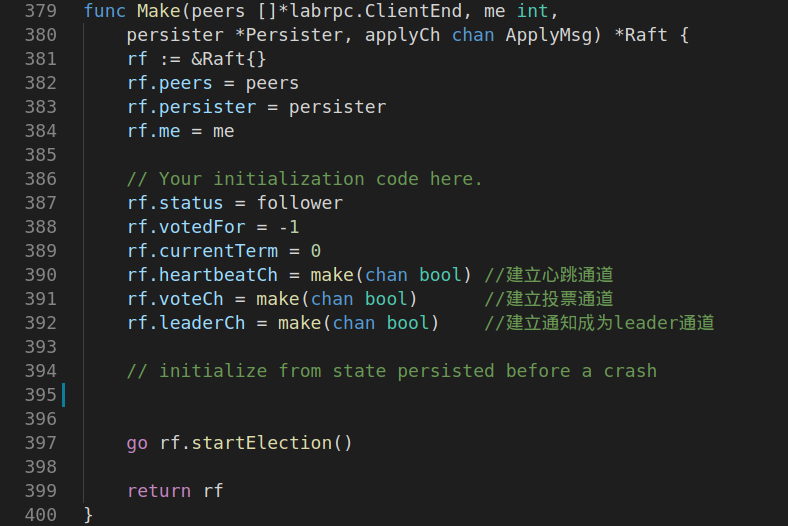
**Make()函数设计**

一个server存在三种可能的状态，分别为leader、candidate和follower。

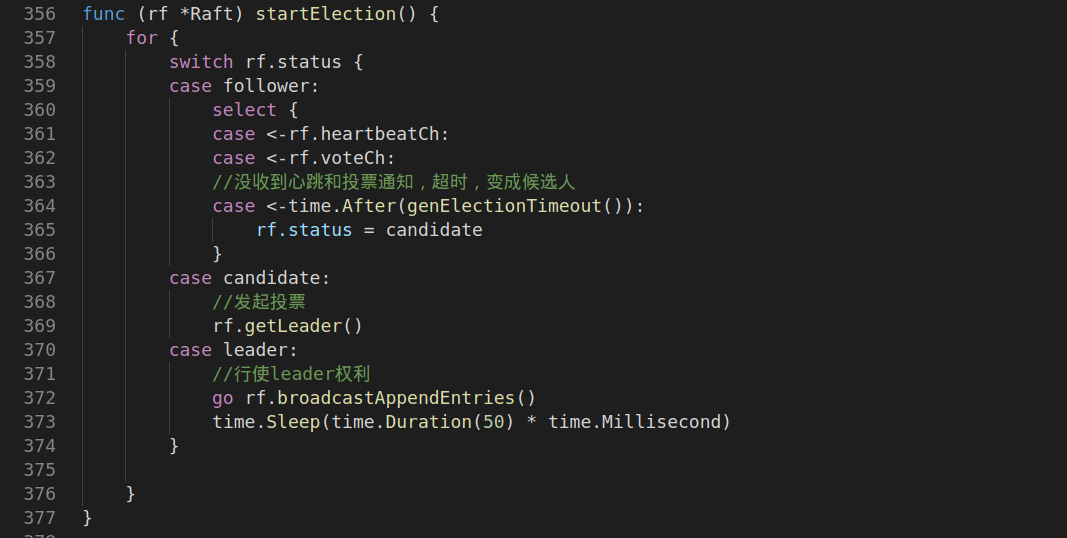
*leader*：向所有节点发送heartbeat，并且在每一个新周期重新发送heartbeat以维持自己的leader身份。对于所有收到heartbeat的server存在两种情况，若leader的term不小于当前server的term则该server则为follower，若leader的term小于当前server的term则server成为candidate状态，并开始发起一个Election。

*candidate*：每一个server从follower状态转换为candidate时都发起一次Election，并令term加1。选举时首先投自己一票，同时像其他节点发送 RequestVote RPC 请求，如果收到的选票数超过server数的一般，则成为leader。如果选举超时，则重新开始新一轮选举。如果收到了leader发来的heartbeat，则判断leader的term与currentTerm，如果leader的term不小于自己发起选举时的term则该server从candidate转换为follower状态，反之则不承认该leader，继续选举。

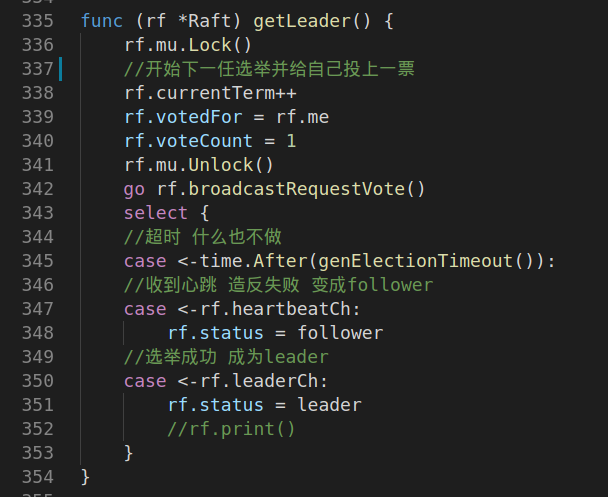
*follower*:响应来自leader的heartbeat以及来自candidate的RequestVote，在发出响应后，更新自己的响应周期，若在出现响应周期的timeout时既未收到leader发来的heartbeat，也未收到candidate发来的投票请求，则转换为candidate状态。



**startElection函数实现（三种状态的转换）**

****

**getLeader函数（对Election过程candidate行为的实现）**

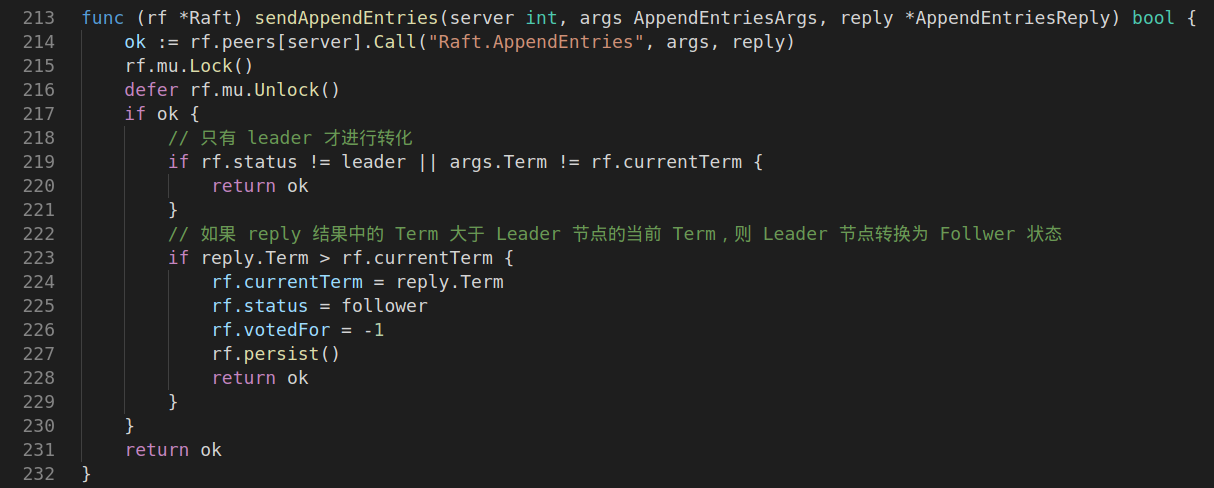


**AppendEntries RPC handler的实现**

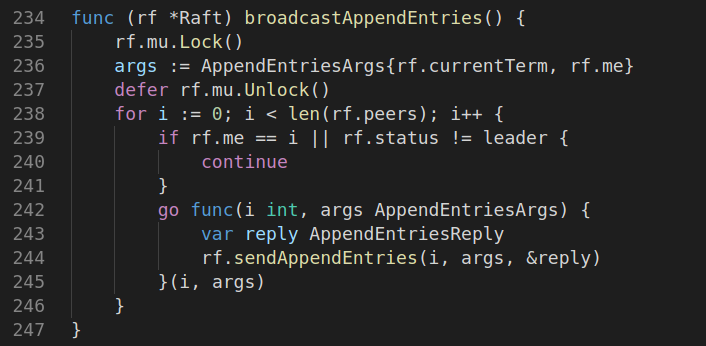
AppendEntries函数首先判断leader的term是否过期，若没有过期，则更新term，各个server角色不变。若leader的term已经过期，则该server从leader状态转换为follower状态。



sendAppendEntries函数实现回应heartbeat的过程，如果一个leader发现CurrentTerm大于自己的term，则leader状态会转换为follower状态。



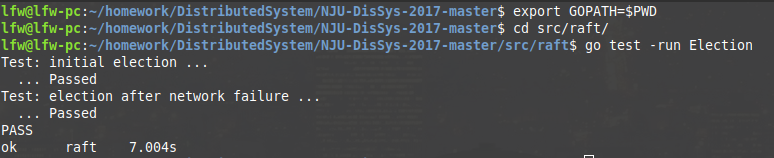
broadcastAppendEntries函数的作用为leader通过该函数实现对其余server发送heartbeat的功能



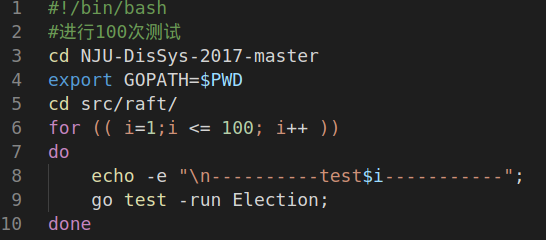
1. 实验演示

首先使用export GOPATH=$PWD 命令设置环境变量，之后在./src/raft/目录下执行测试命令测试代码

实验运行截图：

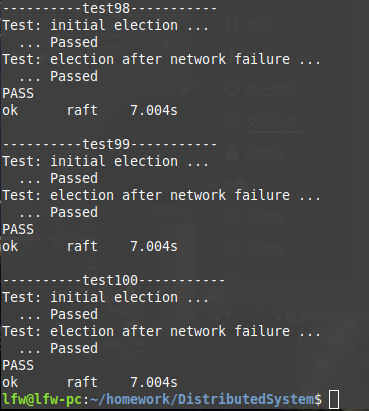


更进一步，通过编写脚本的方式来进行100次测试，测试脚本test代码如下：



在脚本目录下通过./test命令进行了100次测试。结果显示100次测试均成功，且耗时范围为7.003s-7.005s。

实验部分结果如图：



1. 总结

本次实验中，首先学习了Go语言相关知识，对于Go语言的并发性有了一定了解。通过观看动画以及阅读相关论文

学习了Raft一致性算法，最终通过论文的Section 5部分完成了Raft Leader Election的实验，并利用脚本进行了100次测试并全部通过。

实验过程中，由于对于leader、candidate、follower三种状态间转换条件的考虑不全，导致了有时候实验会失败，尤其应注意若出现leader收到heartbeatReply的CurrentTerm大于自己term的情况时，说明产生了新leader，该server应从leader状态转换为follower。而从follower状态转换为candidate状态时，只有产生timeout一种可能。

编写脚本时创建的脚本文件test，umask的值决定了文件的默认权限设置。对于新创建的文件只有属组和属主才有读/写权限。因此在第一次执行脚本文件时需要通过chmod u+x test命令赋予test文件属主执行文件的权限，否则会执行文件失败。