Raft结构建议

笔记本：	Distributed System		
创建时间：	2018/11/26 22:34	**更新时间：**	2018/11/27 11:33
作者：	956318206@qq.com		

- # 状态管理

  每个Raft实例都有一堆（a bunch of）状态（日志，当前索引，等等），这些必须被更新以响应并发goroutine中发生（arising）的事件。
  Go文档指出（1）goroutine可以使用共享数据结构和锁直接执行更新，（2）或者通过传递消息到channels，（3）或者通过中央（central）goroutine来管理状态变化。
  每一种都能很好地工作（work well），但是我们相信**对Raft来说使用共享数据和锁是最直截了当的（straightforward）**。

- # 两种时间驱动的（time-driven）活动

  Raft实例有两种时间驱动的活动：**（1）领导者必须发送心跳，（2）以及其他（对等点）自收到领导者消息以来（since hearing from the the leader），如果太长时间过去（if too much time has passed），开始一次选举。最好使用一个专门的（dedicated）、长期运行的（long-runing）的gorountine来驱动这两者中的每个活动**，而不是将多个活动组合到单个goroutine中。

- # 管理选举超时

  管理选举超时是一个常见的头痛问题。也许**最简单的计划（plan）是在Raft结构中维护一个变量，其包含了该对等点最后一次从领导者那里听到消息的时间（the last time at which the peer heard from the leader），并且让选举超时goroutine（the election timeout goroutine）定期进行检查，看看自那时起的时间（the time since then）是否大于超时周期。**
  使用带有一个小的常量参数的time.Sleep()来驱动定期检查（periodic checks）是最容易的，time.Ticker和time.Timer很难正确使用。

- # 应用提交的（committed）日志条目

  你会想**有一个单独的（separate）长期运行的goroutine来按顺序（in order）发送已提交的（committed）日志条目到`applyCh`。**
  它必须是**单独的（separate）**，因为在`applyCh`上发送消息可能阻塞（block）；
  并且它也必须**单个的goroutine**，因为否则可能很难确保你可以**按日志顺序（in log order）**发送日志条目。
  提升（advances）`commitIndex`的代码将需要踢（kick）该应用goroutine（apply goroutine）；使用条件变量（condition variable）（Go's sync.Cond）做这个可能是最简单。

- # RPC管理

  在同一个goroutine里进行RPC回复（reply）处理是最简单的，而不是通过（over）channel发送回复信息。