

订阅DeepL Pro以编辑此演示文稿。  
访问[www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)，了解更多信息。

// 2015 etcd 作者版权所有

//

// 根据 Apache 许可 2.0 版（"许可"）授权；

// 除非遵守许可协议，否则不得使用本文件。

// 您可以从以下网址获取许可证副本

//

// http://www.apache.org/licenses/LICENSE-2.0

//

// 除非适用法律要求或书面同意，软件

根据本许可发布的 // 均以 "原样 "为基础发布、

// 不附带任何明示或暗示的保证或条件。

// 请参阅许可协议，了解有关权限和

// 许可中的限制。

/\*

软件包筏以协议缓冲区格式发送和接收信息

的定义。

Raft 是一种协议，节点集群可以利用它来维护一个复制的状态机。

状态机通过使用复制日志保持同步。

有关 Raft 的更多详情，请参阅 "寻找可理解的共识算法"。

(https://ramcloud.stanford.edu/raft.pdf），作者为 Diego Ongaro 和 John Ousterhout。

使用方法

raft 中的主要对象是节点。您可以从头开始创建一个节点

使用 raft.StartNode 或使用 raft.RestartNode 从某个初始状态启动节点。

从零开始启动节点

存储 := raft.NewMemoryStorage()

c := &Config{

ID: 0x01、

ElectionTick: 10、

HeartbeatTick: 1、

存储：存储、

}

n := raft.StartNode(c, []raft.Peer{ID: 0x02}, {ID: 0x03}})

从先前的状态重新启动节点：

存储 := raft.NewMemoryStorage()

// 从持久性存储中恢复内存存储

// 快照、状态和条目。

存储.ApplySnapshot（快照）

存储.SetHardState（状态）

storage.Append(entries)

c := &Config{

ID: 0x01、

ElectionTick: 10、

HeartbeatTick: 1、

存储：存储、

最大飞行任务数256,

}

// 在没有同行信息的情况下重新启动木筏。

// 存储中已包含同行信息。

n := raft.RestartNode(c)

既然你已经拥有了一个节点，你就有了一些责任：

首先，您必须从 Node.Ready() 频道读取数据并处理更新

它所包含的内容。这些步骤可以并行执行，除非在步骤

2.

1.将 HardState、条目和快照写入持久性存储（如果它们是

不为空。注意，在写入索引为 i 的条目时，任何

必须丢弃索引 >= i 的先前存在的条目。

2.向收件人字段中指定的节点发送所有信息。重要的是

在最新的 HardState 保存到磁盘之前，不发送任何信息、

以及之前任何就绪批次写入的所有条目（信息可在以下情况下发送

同一批次中的条目将被持久化）。

注意：整理报文不是线程安全的；重要的是，您必须

确保在 marshalling 时没有新条目被持久化。

最简单的方法是直接在

你的主筏环。

3.将快照（如有）和 CommittedEntries 应用于状态机。

如果任何已提交的条目具有 EntryType\_EntryConfChange 类型，则调用 Node.ApplyConfChange()

将其应用到节点。此时可以取消配置更改

将 NodeId 字段设置为零，然后再调用 ApplyConfChange

(但 ApplyConfChange 必须被调用，而决定取消

必须完全基于状态机，而不是外部信息，例如

观测到的节点健康状况）。

4.调用 Node.Advance()，发出下一批更新准备就绪的信号。

这可以在步骤 1 之后的任何时间进行，但必须处理所有更新

的顺序排列。

其次，所有持久化日志条目都必须通过一个

存储接口的实现。所提供的 MemoryStorage

类型可用于此目的（如果您在

重启），或者你也可以提供自己的磁盘支持实现。

第三，当收到来自其他节点的消息时，将其传递给 Node.Step：

func recvRaftRPC(ctx context.Context, m eraftpb.Message) {

n.Step(ctx, m)

}

最后，您需要定期调用 Node.Tick()（可能是

Ticker）。Raft 有两个重要的超时：心跳和

选举超时。但是，筏包内部的时间是

用一个抽象的 "勾 "来表示。

整个状态机处理循环将是这样的

为 {

选择 {

case <-s.Ticker：

n.Tick()

case rd := <-s.Node.Ready()：

saveToStorage(rd.State, rd.Entries, rd.Snapshot)

发送（rd.信息）

if !raft.IsEmptySnap(rd.Snapshot) {

processSnapshot(rd.Snapshot)

}

for \_, entry := range rd.CommittedEntries {

处理(条目)

if entry.Type == eraftpb.EntryType\_EntryConfChange {

var cc eraftpb.ConfChange

cc.Unmarshal（条目数据）

s.Node.ApplyConfChange(cc)

}

}

s.Node.Advance()

案例 <-s.done：

返回

}

}

要从节点对状态机进行修改，请将应用程序

数据，将其序列化为字节片，然后调用

n.Propose(data)

如果提案已提交，数据将出现在提交的条目中，类型为

eraftpb.EntryType\_EntryNormal.不能保证所建议的命令会被

已提交；超时后可能需要重新提交。

要在集群中添加或删除节点，请构建 ConfChange 结构 "cc "并调用：

n.ProposeConfChange(cc)

配置更改提交后，一些类型为

eraftpb.EntryType\_EntryConfChange 将被返回。您必须将其应用于节点：

var cc eraftpb.ConfChange

cc.Unmarshal(data)

n.ApplyConfChange(cc)

注：ID 代表集群中一直唯一的节点。A

即使旧节点已被删除，给定的 ID 也只能使用一次。

这意味着，例如 IP 地址就不能作为节点 ID，因为它们

可以重复使用。节点 ID 必须非零。

实施说明

本实施方案与最终的 Raft 论文保持一致

(https://ramcloud.stanford.edu/~ongaro/thesis.pdf)，尽管我们的

成员变更协议的执行与

第 4 章介绍了这一点。成员资格变化的关键不变因素

但在我们的实施过程中，"......

成员资格更改在应用其条目时生效，而不是在

会被添加到日志中（因此条目会在旧的

而不是新成员）。这在安全性方面是相同的、

因为新旧配置一定会重叠。

为了确保我们不会在

匹配日志位置一次（这是不安全的，因为它们

法定人数要求应有所不同），我们干脆不允许任何

建议的会籍变更，而任何未承诺的变更则出现在

领导日志。

当您尝试删除成员时，这种方法会带来一个问题

从双成员群组中删除：如果其中一个成员在

另一个成员接收到 confchange 条目的提交，则该成员

不能再删除，因为该群组无法取得进展。

因此，强烈建议在

每个群组。

消息类型

软件包筏以协议缓冲区格式（定义为

在 eraftpb 软件包中）。每个状态（追随者、候选者、领导者）实现其

自己的 "步骤 "方法（"步骤追随者"、"步骤候选人"、"步骤领导者"）时

前进。每个步骤由其

eraftpb.MessageType.请注意，每个步骤都由一个通用方法进行检查

步骤"，对节点和传入信息的术语进行安全检查，以防止

过期日志条目：

MessageType\_MsgHup" 用于选举。如果节点是追随者或候选人，则

筏 "结构中的 "tick "函数设置为 "tickElection"。如果追随者或

如果候选者在选举超时前没有收到任何心跳，它就会

将 "MessageType\_MsgHup "传递给它的步骤方法，并成为（或继续成为）"MessageType\_MsgHup "的候选对象。

开始新的选举。

MessageType\_MsgBeat "是一种内部类型，用于向领导者发送以下心跳信号

消息类型\_MsgHeartbeat "类型。如果节点是领导者，则

筏 "结构设置为 "tickHeartbeat"，并触发领导者

定期向其追随者发送 "MessageType\_MsgHeartbeat "信息。

MessageType\_MsgPropose "建议在日志条目中附加数据。这是一个特殊的

类型将建议重定向给领导者。因此，发送方法会覆盖

eraftpb.Message 的术语与其 HardState 的术语相一致，以避免附加其

本地术语为 "MessageType\_MsgPropose"。当 "MessageType\_MsgPropose "传递给领导者的 "Step

方法，领导者首先调用 "appendEntry "方法来添加条目。

到日志中，然后调用 "bcastAppend "方法将这些条目发送到

其对等程序。当传递给候选者时，"MessageType\_MsgPropose "会被丢弃。当传递给

从者，"MessageType\_MsgPropose "被发送到从者的邮箱（msgs）中。

方法。它与发送者的 ID 一起存储，随后通过以下方式转发给领导者

rafthttp 软件包。

MessageType\_MsgAppend "包含要复制的日志条目。领导者调用 bcastAppend、

调用 sendAppend，以 "MessageType\_MsgAppend "格式发送即将被复制的日志

类型。当 "MessageType\_MsgAppend "被传递给候选者的步骤方法时，候选者会恢复 "MessageType\_MsgAppend"。

返回给追随者，因为这表明有一个有效的领导者在向追随者发送信息。

MessageType\_MsgAppend "信息。候选人和追随者以

MessageType\_MsgAppendResponse' 类型。

MessageType\_MsgAppendResponse "是对日志复制请求（"MessageType\_MsgAppend"）的响应。当

如果 "MessageType\_MsgAppend "被传递给候选人或追随者的步骤方法，它的响应方式是

调用 "handleAppendEntries "方法，该方法会向筏发送 "MessageType\_MsgAppendResponse"。

邮箱

MessageType\_MsgRequestVote "请求选举投票。当节点是追随者或

候选节点，并将 "MessageType\_MsgHup "传递给其步骤方法，然后节点调用

运动 "的方法，使自己成为领导者。一旦'竞选

方法被调用时，节点将成为候选节点，并向对等节点发送 "MessageType\_MsgRequestVote

在群组中请求投票。当传递给领导者或候选人的步骤

方法，且信息的任期低于领导者或候选人的任期、

MessageType\_MsgRequestVote "将被拒绝（"MessageType\_MsgRequestVoteResponse "将返回 Reject true）。

如果领导者或候选人收到 "MessageType\_MsgRequestVote"（信息类型\_信息请求投票），且任期较长，则会返回

返回给跟随者。当 "MessageType\_MsgRequestVote "被传递给跟随者时，它将为以下内容投票

仅当发件人的最后期限大于 MessageType\_MsgRequestVote 的期限或

发件人的最后一个任期等于 MessageType\_MsgRequestVote 的任期，但发件人最后承诺的

索引大于或等于追随者的索引。

MessageType\_MsgRequestVoteResponse'包含投票请求的响应。当'MessageType\_MsgRequestVoteResponse'为

传递给候选人，候选人计算自己赢得了多少票。如果

超过多数（法定人数），则成为领导并调用 "bcastAppend"。

如果候选人获得多数否决票，则返回到

追随者。

MessageType\_MsgSnapshot "请求安装快照消息。当节点刚刚

成为领导者或领导者收到 "MessageType\_MsgPropose "信息时，会调用

bcastAppend "方法，然后调用 "sendAppend "方法向每个

从者。在 "sendAppend "中，如果领导者未能获得任期或条目、

领导者通过发送 "MessageType\_MsgSnapshot "类型的信息来请求快照。

MessageType\_MsgHeartbeat'从领导者发送心跳。当通过'MessageType\_MsgHeartbeat'时

而信息的任期高于候选人的任期，则候选人

恢复为跟随者，并更新其提交的索引，从

这个心跳。然后它将信息发送到邮箱。当

信息类型\_MsgHeartbeat "被传递给追随者的步骤方法，信息的术语是

高于追随者，则追随者会用 ID

从信息中

MessageType\_MsgHeartbeatResponse'是对'MessageType\_MsgHeartbeat'的响应。当'MessageType\_MsgHeartbeatResponse'是对'MessageType\_MsgHeartbeat'的响应时

被传递给领导者的步骤方法，领导者就知道哪个追随者

作出了回应。

\*/

包筏