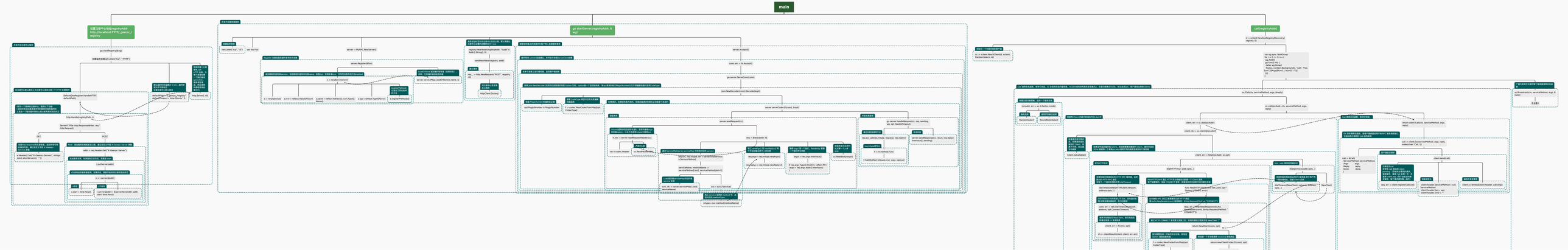

并发开启服务端服务

并发开启注册中⼼服务

在注册中⼼默认路径上为注册中⼼消息注册⼀个 HTTP处理程序

默认超时时间设置为5 min，超时即
视为不可⽤状态
设置注册中⼼默认路径

在监听器l上接
受传⼊的
HTTP连接，为
每个连接创建
⼀个新的服务
goroutine。
服务读取请
求，然后调⽤
处理程序来回
复它们。

r即为⼀个简单的注册中⼼，提供以下功能：
1.返回所有活动服务器并同时删除死服务器同步
2.添加⼀个服务器并接收⼼跳以使其保持活动状态

设置http response的头部信息。返回所有可⽤
的服务列表，通过⾃定义字段 X-Geerpc-
Servers承载

Post：添加服务实例或发送⼼跳，通过⾃定义字段 X-Geerpc-Server承载

添加服务实例，如果服务已经存在，则更新 start

s为该地址的服务器实例。如果存在，更新开始时间以保持活动状态

s存在 s不存在

创建监听连接

服务启动时定时向注册中⼼发送⼼跳，默认周期⽐
注册中⼼设置的过期时间少 1 min 接受侦听器上的连接并为每个传⼊连接提供请求

Register注册在服务器中发布的⽅法集

返回映射的结构体service。包括映射的结构体名称name、类型typ、实例本身rcvr、所有符合条件的⽅法method

LoadOrStore返回键的现有值（如果存在），
否则，它存储并返回给定的值

registerMethods
过滤出了符合条件
的⽅法

建⽴请求

向注册中⼼发送请
求⼼跳包

循环等待socket连接建⽴，并开启⼦协程ServerConn处理

在单个连接上运⾏服务器，直到客户端挂断

使⽤json.NewDecoder反序列化连接请求得到 Option实例。option是⼀个选项结构体，有rpc请求的标记MagicNumber以及不同编解码器的选项CodeType

检查MagicNumber的值是否正确

根据CodeType得到对应的消息编解
码器函数 处理请求。处理请求是并发的，但是回复请求的报⽂必须是逐个发送的

读取请求 并发处理请求

request结构体包含请求头部h，请求的参数argv
和回复replyv，以及⽅法类型mtype与服务svc

读请求头部

通过ServiceMethod从serviceMap中找到对应的 service

通过newArgv()和newReplyv()两
个⽅法创建出两个⼊参实例

确保argvi是⼀个指针，ReadBody需要
⼀个指针作为参数

将请求报⽂反序列
化为第⼀个⼊参
argv

Load返回键serviceMap对应的值
service实例

再从service实例的method中，找
到对应的methodType

通过反射值调⽤⽅法 发送回复

req.mtype即为m

初始化⼀个负载均衡的客户端

Call调⽤命名函数，等待它完成。xc会选择合适的服务器。XClient结构体有服务发现模式d，负载均衡模式mode，协议选项opt，客户端地址映射clients

⼴播为发现中注册的每个服务器调⽤命名函
数

根据负载均衡策略，选择⼀个服务实例

随机选择 使⽤罗宾算法选择
将复⽤Client的能⼒封装在⽅法 dial中 Call调⽤命名函数，等待它完成，

检查是否是可⽤状
态，如果是则返回
缓存的Client，如
果不可⽤，则从缓
存中删除

如果没有返回缓存的Client，则说明需要创建新的 Client，缓存并返回
XDial根据第⼀个参数rpcAddr调⽤不同的函数连接到RPC服务器

若为HTTP协议 tcp、unix或其他传输协议

连接到指定⽹络地址的HTTP RPC服务器，监听
默认的HTTP RPC路径
实现了⼀个超时处理的外壳 dialTimeout

NewHTTPClient通过HTTP 作为传输协议新建⼀个 Client实例
客户端要做的，发起CONNECT请求，检查返回状态码即可成功建⽴连接

DialTimeout的作⽤类似于 Dial，但有超时处
理,如果连接创建超时，将返回错误

使⽤⼦协程执⾏NewClient，执⾏完成后
则通过信道ch发送结果

在切换到RPC协议之前需要成功的 HTTP 响应
读:bufio.NewReader(conn)返回响应：&http.Request{Method: "CONNECT"}

通过HTTP CONNECT请求建⽴连接之后，后续的通信过程就交给 NewClient了

⾸先需要完成⼀开始的协议交换，即发送
Option信息给服务端 再创建⼀个⼦协程调⽤receive()接收响应

连接到指定⽹络地址的RPC服务器,便于⽤户传
⼊服务端地址，创建Client实例

Go异步调⽤该函数。是客户端暴露给⽤户的 RPC服务调⽤接⼝
它返回表示调⽤的Call结构实例

客户端发送请求

注册这次call
将参数call添加到client.
pending （存储未处理完的请求，
键是编号，值是Call实例）中，并
更新client.seq（⽤于给发送的请
求编号，每个请求拥有唯⼀编号） 准备请求头 编码并发送请求

main

设置注册中⼼地址registryAddr

http://localhost:9999/_geerpc_/

registry

go startServer(registryAddr, &

wg)

call(registryAddr)

go startRegistry(&wg)

创建监听连接net.Listen("tcp", ":9999")

DefaultGeeRegister.HandleHTTP(
defaultPath)

http.Handle(registryPath, r)

ServeHTTP(w http.ResponseWriter, req *
http.Request)

GET

w.Header().Set("X-Geerpc-Servers", strings.
Join(r.aliveServers(), ","))

POST

addr:= req.Header.Get("X-Geerpc-Server")

r.putServer(addr)

s:= r.servers[addr]

s.start = time.Now() r.servers[addr] = &ServerItem{Addr: addr,
start: time.Now()

defaultPath = "/_geerpc_/registry"
defaultTimeout = time.Minute * 5

http.Serve(l, nil)

net.Listen("tcp", ":0") server:= MyRPC.NewServer()

server.Register(&foo)

var foo Foo

s:= newService(rcvr)

s:= new(service) s.rcvr = reflect.ValueOf(rcvr) s.name = reflect.Indirect(s.rcvr).Type(). Name()

s.typ = reflect.TypeOf(rcvr) s.registerMethods()

server.serviceMap.LoadOrStore(s.name, s)

registry.Heartbeat(registryAddr, "tcp@"+l. Addr().String(), 0)

sendHeartbeat(registry, addr)

req, _:= http.NewRequest("POST", registry,
nil)

httpClient.Do(req)

server.Accept(l)

conn, err:= lis.Accept()

go server.ServeConn(conn)

json.NewDecoder(conn).Decode(&opt)

opt.MagicNumber!= MagicNumber f:= codec.NewCodecFuncMap[opt. CodecType]

server.serveCodec(f(conn), &opt)

server.readRequest(cc)

h, err:= server.readRequestHeader(cc)

var h codec.Header cc.ReadHeader(&h)

req := &request{h: h}

req.svc, req.mtype, err = server.findService(
h.ServiceMethod)

serviceName, methodName :=
serviceMethod[:dot], serviceMethod[dot+1:]

svci, ok:= server.serviceMap.Load(
serviceName)

svc = svci.(*service)

mtype = svc.method[methodName]

req.argv = req.mtype.newArgv()

req.replyv = req.mtype.newReplyv()

argvi:= req.argv.Interface()

if req.argv.Type().Kind() != reflect.Ptr {
argvi = req.argv.Addr().Interface()
}

cc.ReadBody(argvi)

go server.handleRequest(cc, req, sending, wg, opt.HandleTimeout)

req.svc.call(req.mtype, req.argv, req.replyv) server.sendResponse(cc, req.h, req.replyv.
Interface(), sending)

f:= m.method.Func

f.Call([]reflect.Value{s.rcvr, argv, replyv})

d:= xclient.NewGeeRegistryDiscovery(
registry, 0)

xc:= xclient.NewXClient(d, xclient. RandomSelect, nil)

var wg sync.WaitGroup
for i:= 0; i < 5; i++ {
wg.Add(1)
go func(i int) {
defer wg.Done()
foo(xc, context.Background(), "call", "Foo.
Sum", &Args{Num1: i, Num2: i * i})
}(i)
}

xc.Call(ctx, serviceMethod, args, &reply) xc.Broadcast(ctx, serviceMethod, args, &
reply)

⼦主题1
rpcAddr, err:= xc.d.Get(xc.mode)

RandomSelect RoundRobinSelect

xc.call(rpcAddr, ctx, serviceMethod, args,
reply)

client, err:= xc.dial(rpcAddr)

client, ok:= xc.clients[rpcAddr]

!client.IsAvailable() client, err = XDial(rpcAddr, xc.opt)

DialHTTP("tcp",addr,opts...) Dial(protocol,addr,opts...)

dialTimeout(NewHTTPClient,network,
address,opts...)

conn, err:= net.DialTimeout(network,
address, opt.ConnectTimeout)

client, err:= f(conn, opt)

ch <- clientResult{client: client, err: err}

func NewHTTPClient(conn net.Conn, opt *
Option) (*Client, error)

resp, err:= http.ReadResponse(bufio. NewReader(conn), &http.Request{Method: "
CONNECT"})

return NewClient(conn, opt)

f:= codec.NewCodecFuncMap[opt. CodecType]

return newClientCodec(f(conn), opt)

li i ()

dialTimeout(NewClient, network, address,
opts...)

NewClient

return client.Call(ctx, serviceMethod, args,
reply)

call:= client.Go(serviceMethod, args, reply, make(chan *Call, 1))

call:= &Call{

ServiceMethod: serviceMethod, Args: args, Reply: reply, Done: done,

}

client.send(call)

seq, err:= client.registerCall(call) client.header.ServiceMethod = call.
ServiceMethod
client.header.Seq = seq
client.header.Error = ""

client.cc.Write(&client.header, call.Args)