**gRPC服务发现&负载均衡**

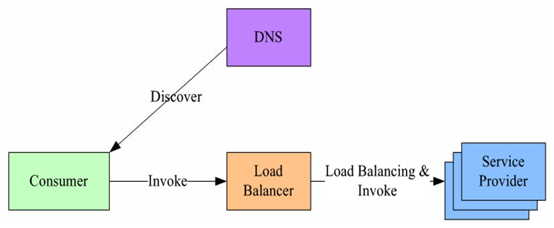
**目录 [−]**

1. [1、集中式LB（Proxy Model）](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#1%E3%80%81%E9%9B%86%E4%B8%AD%E5%BC%8FLB%EF%BC%88Proxy_Model%EF%BC%89)
2. [2、进程内LB（Balancing-aware Client）](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#2%E3%80%81%E8%BF%9B%E7%A8%8B%E5%86%85LB%EF%BC%88Balancing-aware_Client%EF%BC%89)
3. [3、独立 LB 进程（External Load Balancing Service）](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#3%E3%80%81%E7%8B%AC%E7%AB%8B_LB_%E8%BF%9B%E7%A8%8B%EF%BC%88External_Load_Balancing_Service%EF%BC%89)
4. [gRPC服务发现及负载均衡实现](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#gRPC%E6%9C%8D%E5%8A%A1%E5%8F%91%E7%8E%B0%E5%8F%8A%E8%B4%9F%E8%BD%BD%E5%9D%87%E8%A1%A1%E5%AE%9E%E7%8E%B0)
   1. [1）命名解析实现：resolver.go](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#1%EF%BC%89%E5%91%BD%E5%90%8D%E8%A7%A3%E6%9E%90%E5%AE%9E%E7%8E%B0%EF%BC%9Aresolver-go)
   2. [2）服务发现实现：watcher.go](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#2%EF%BC%89%E6%9C%8D%E5%8A%A1%E5%8F%91%E7%8E%B0%E5%AE%9E%E7%8E%B0%EF%BC%9Awatcher-go)
   3. [3）服务注册实现：register.go](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#3%EF%BC%89%E6%9C%8D%E5%8A%A1%E6%B3%A8%E5%86%8C%E5%AE%9E%E7%8E%B0%EF%BC%9Aregister-go)
   4. [4）接口描述文件：helloworld.proto](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#4%EF%BC%89%E6%8E%A5%E5%8F%A3%E6%8F%8F%E8%BF%B0%E6%96%87%E4%BB%B6%EF%BC%9Ahelloworld-proto)
   5. [5）实现服务端接口：helloworldserver.go](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#5%EF%BC%89%E5%AE%9E%E7%8E%B0%E6%9C%8D%E5%8A%A1%E7%AB%AF%E6%8E%A5%E5%8F%A3%EF%BC%9Ahelloworldserver-go)
   6. [6）实现客户端接口：helloworldclient.go](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#6%EF%BC%89%E5%AE%9E%E7%8E%B0%E5%AE%A2%E6%88%B7%E7%AB%AF%E6%8E%A5%E5%8F%A3%EF%BC%9Ahelloworldclient-go)
   7. [7）运行测试](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#7%EF%BC%89%E8%BF%90%E8%A1%8C%E6%B5%8B%E8%AF%95)
5. [参考](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/#%E5%8F%82%E8%80%83)

原文出处: [gRPC服务发现&负载均衡](https://segmentfault.com/a/1190000008672912" \t "_blank), 作者: [softfn](https://segmentfault.com/u/softfn" \t "_blank)。

构建高可用、高性能的通信服务，通常采用服务注册与发现、负载均衡和容错处理等机制实现。根据负载均衡实现所在的位置不同，通常可分为以下三种解决方案：

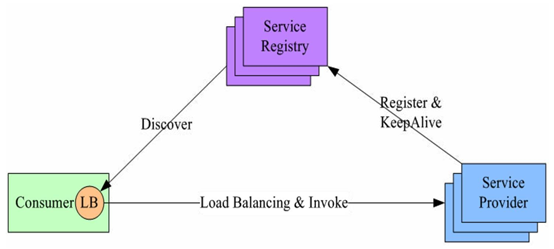
***1、集中式LB（Proxy Model）***

[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/1.png)

在服务消费者和服务提供者之间有一个独立的LB，通常是专门的硬件设备如 F5，或者基于软件如 LVS，HAproxy等实现。LB上有所有服务的地址映射表，通常由运维配置注册，当服务消费方调用某个目标服务时，它向LB发起请求，由LB以某种策略，比如轮询（Round-Robin）做负载均衡后将请求转发到目标服务。LB一般具备健康检查能力，能自动摘除不健康的服务实例。 该方案主要问题：

1. 单点问题，所有服务调用流量都经过LB，当服务数量和调用量大的时候，LB容易成为瓶颈，且一旦LB发生故障影响整个系统；
2. 服务消费方、提供方之间增加了一级，有一定性能开销。

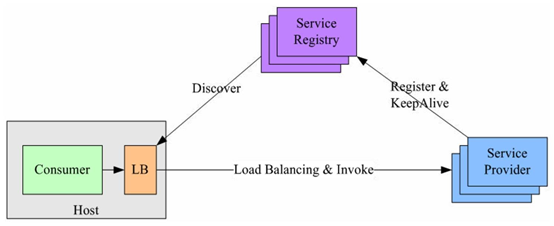
***2、进程内LB（Balancing-aware Client）***

[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/2.png)

针对第一个方案的不足，此方案将LB的功能集成到服务消费方进程里，也被称为软负载或者客户端负载方案。服务提供方启动时，首先将服务地址注册到服务注册表，同时定期报心跳到服务注册表以表明服务的存活状态，相当于健康检查，服务消费方要访问某个服务时，它通过内置的LB组件向服务注册表查询，同时缓存并定期刷新目标服务地址列表，然后以某种负载均衡策略选择一个目标服务地址，最后向目标服务发起请求。LB和服务发现能力被分散到每一个服务消费者的进程内部，同时服务消费方和服务提供方之间是直接调用，没有额外开销，性能比较好。该方案主要问题：

1. 开发成本，该方案将服务调用方集成到客户端的进程里头，如果有多种不同的语言栈，就要配合开发多种不同的客户端，有一定的研发和维护成本；
2. 另外生产环境中，后续如果要对客户库进行升级，势必要求服务调用方修改代码并重新发布，升级较复杂。

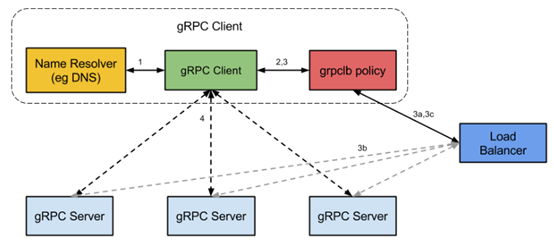
***3、独立 LB 进程（External Load Balancing Service）***

[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/3.png)

该方案是针对第二种方案的不足而提出的一种折中方案，原理和第二种方案基本类似。  
不同之处是将LB和服务发现功能从进程内移出来，变成主机上的一个独立进程。主机上的一个或者多个服务要访问目标服务时，他们都通过同一主机上的独立LB进程做服务发现和负载均衡。该方案也是一种分布式方案没有单点问题，一个LB进程挂了只影响该主机上的服务调用方，服务调用方和LB之间是进程内调用性能好，同时该方案还简化了服务调用方，不需要为不同语言开发客户库，LB的升级不需要服务调用方改代码。  
该方案主要问题：部署较复杂，环节多，出错调试排查问题不方便。

***gRPC服务发现及负载均衡实现***

gRPC开源组件官方并未直接提供服务注册与发现的功能实现，但其设计文档已提供实现的思路，并在不同语言的gRPC代码API中已提供了命名解析和负载均衡接口供扩展。

[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/4.png)

其基本实现原理：

1. 服务启动后gRPC客户端向命名服务器发出名称解析请求，名称将解析为一个或多个IP地址，每个IP地址标示它是服务器地址还是负载均衡器地址，以及标示要使用那个客户端负载均衡策略或服务配置。
2. 客户端实例化负载均衡策略，如果解析返回的地址是负载均衡器地址，则客户端将使用grpclb策略，否则客户端使用服务配置请求的负载均衡策略。
3. 负载均衡策略为每个服务器地址创建一个子通道（channel）。
4. 当有rpc请求时，负载均衡策略决定那个子通道即grpc服务器将接收请求，当可用服务器为空时客户端的请求将被阻塞。

根据gRPC官方提供的设计思路，基于进程内LB方案（即第2个案，阿里开源的服务框架 Dubbo 也是采用类似机制），结合分布式一致的组件（如Zookeeper、Consul、Etcd），可找到gRPC服务发现和负载均衡的可行解决方案。接下来以GO语言为例，简单介绍下基于Etcd3的关键代码实现：

**1）命名解析实现：resolver.go**

resolver.go

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package etcdv3  import (  "errors"  "fmt"  "strings"  etcd3 "github.com/coreos/etcd/clientv3"  "google.golang.org/grpc/naming"  )  // resolver is the implementaion of grpc.naming.Resolver  type resolver struct {  serviceName string // service name to resolve  }  // NewResolver return resolver with service name  func NewResolver(serviceName string) \*resolver {  return &resolver{serviceName: serviceName}  }  // Resolve to resolve the service from etcd, target is the dial address of etcd  // target example: "http://127.0.0.1:2379,http://127.0.0.1:12379,http://127.0.0.1:22379"  func (re \*resolver) Resolve(target string) (naming.Watcher, error) {  if re.serviceName == "" {  return nil, errors.New("grpclb: no service name provided")  }  // generate etcd client  client, err := etcd3.New(etcd3.Config{  Endpoints: strings.Split(target, ","),  })  if err != nil {  return nil, fmt.Errorf("grpclb: creat etcd3 client failed: %s", err.Error())  }  // Return watcher  return &watcher{re: re, client: \*client}, nil  } |

**2）服务发现实现：watcher.go**

watcher.go

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | package etcdv3  import (  "fmt"  etcd3 "github.com/coreos/etcd/clientv3"  "golang.org/x/net/context"  "google.golang.org/grpc/naming"  "github.com/coreos/etcd/mvcc/mvccpb"  )  // watcher is the implementaion of grpc.naming.Watcher  type watcher struct {  re \*resolver // re: Etcd Resolver  client etcd3.Client  isInitialized bool  }  // Close do nothing  func (w \*watcher) Close() {  }  // Next to return the updates  func (w \*watcher) Next() ([]\*naming.Update, error) {  // prefix is the etcd prefix/value to watch  prefix := fmt.Sprintf("/%s/%s/", Prefix, w.re.serviceName)  // check if is initialized  if !w.isInitialized {  // query addresses from etcd  resp, err := w.client.Get(context.Background(), prefix, etcd3.WithPrefix())  w.isInitialized = true  if err == nil {  addrs := extractAddrs(resp)  //if not empty, return the updates or watcher new dir  if l := len(addrs); l != 0 {  updates := make([]\*naming.Update, l)  for i := range addrs {  updates[i] = &naming.Update{Op: naming.Add, Addr: addrs[i]}  }  return updates, nil  }  }  }  // generate etcd Watcher  rch := w.client.Watch(context.Background(), prefix, etcd3.WithPrefix())  for wresp := range rch {  for \_, ev := range wresp.Events {  switch ev.Type {  case mvccpb.PUT:  return []\*naming.Update{ {Op: naming.Add, Addr: string(ev.Kv.Value)} }, nil  case mvccpb.DELETE:  return []\*naming.Update{ {Op: naming.Delete, Addr: string(ev.Kv.Value)} }, nil  }  }  }  return nil, nil  }  func extractAddrs(resp \*etcd3.GetResponse) []string {  addrs := []string{}  if resp == nil || resp.Kvs == nil {  return addrs  }  for i := range resp.Kvs {  if v := resp.Kvs[i].Value; v != nil {  addrs = append(addrs, string(v))  }  }  return addrs  } |

**3）服务注册实现：register.go**

register.go

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | package etcdv3  import (  "fmt"  "log"  "strings"  "time"  etcd3 "github.com/coreos/etcd/clientv3"  "golang.org/x/net/context"  "github.com/coreos/etcd/etcdserver/api/v3rpc/rpctypes"  )  // Prefix should start and end with no slash  var Prefix = "etcd3\_naming"  var client etcd3.Client  var serviceKey string  var stopSignal = make(chan bool, 1)  // Register  func Register(name string, host string, port int, target string, interval time.Duration, ttl int) error {  serviceValue := fmt.Sprintf("%s:%d", host, port)  serviceKey = fmt.Sprintf("/%s/%s/%s", Prefix, name, serviceValue)  // get endpoints for register dial address  var err error  client, err := etcd3.New(etcd3.Config{  Endpoints: strings.Split(target, ","),  })  if err != nil {  return fmt.Errorf("grpclb: create etcd3 client failed: %v", err)  }  go func() {  // invoke self-register with ticker  ticker := time.NewTicker(interval)  for {  // minimum lease TTL is ttl-second  resp, \_ := client.Grant(context.TODO(), int64(ttl))  // should get first, if not exist, set it  \_, err := client.Get(context.Background(), serviceKey)  if err != nil {  if err == rpctypes.ErrKeyNotFound {  if \_, err := client.Put(context.TODO(), serviceKey, serviceValue, etcd3.WithLease(resp.ID)); err != nil {  log.Printf("grpclb: set service '%s' with ttl to etcd3 failed: %s", name, err.Error())  }  } else {  log.Printf("grpclb: service '%s' connect to etcd3 failed: %s", name, err.Error())  }  } else {  // refresh set to true for not notifying the watcher  if \_, err := client.Put(context.Background(), serviceKey, serviceValue, etcd3.WithLease(resp.ID)); err != nil {  log.Printf("grpclb: refresh service '%s' with ttl to etcd3 failed: %s", name, err.Error())  }  }  select {  case <-stopSignal:  return  case <-ticker.C:  }  }  }()  return nil  }  // UnRegister delete registered service from etcd  func UnRegister() error {  stopSignal <- true  stopSignal = make(chan bool, 1) // just a hack to avoid multi UnRegister deadlock  var err error;  if \_, err := client.Delete(context.Background(), serviceKey); err != nil {  log.Printf("grpclb: deregister '%s' failed: %s", serviceKey, err.Error())  } else {  log.Printf("grpclb: deregister '%s' ok.", serviceKey)  }  return err  } |

**4）接口描述文件：helloworld.proto**

helloworld.proto

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | syntax = "proto3";  option java\_multiple\_files = true;  option java\_package = "com.midea.jr.test.grpc";  option java\_outer\_classname = "HelloWorldProto";  option objc\_class\_prefix = "HLW";  package helloworld;  // The greeting service definition.  service Greeter {  // Sends a greeting  rpc SayHello (HelloRequest) returns (HelloReply) {  }  }  // The request message containing the user's name.  message HelloRequest {  string name = 1;  }  // The response message containing the greetings  message HelloReply {  string message = 1;  } |

**5）实现服务端接口：helloworldserver.go**

helloworldserver.go

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | package main  import (  "flag"  "fmt"  "log"  "net"  "os"  "os/signal"  "syscall"  "time"  "golang.org/x/net/context"  "google.golang.org/grpc"  grpclb "com.midea/jr/grpclb/naming/etcd/v3"  "com.midea/jr/grpclb/example/pb"  )  var (  serv = flag.String("service", "hello\_service", "service name")  port = flag.Int("port", 50001, "listening port")  reg = flag.String("reg", "http://127.0.0.1:2379", "register etcd address")  )  func main() {  flag.Parse()  lis, err := net.Listen("tcp", fmt.Sprintf("0.0.0.0:%d", \*port))  if err != nil {  panic(err)  }  err = grpclb.Register(\*serv, "127.0.0.1", \*port, \*reg, time.Second\*10, 15)  if err != nil {  panic(err)  }  ch := make(chan os.Signal, 1)  signal.Notify(ch, syscall.SIGTERM, syscall.SIGINT, syscall.SIGKILL, syscall.SIGHUP, syscall.SIGQUIT)  go func() {  s := <-ch  log.Printf("receive signal '%v'", s)  grpclb.UnRegister()  os.Exit(1)  }()  log.Printf("starting hello service at %d", \*port)  s := grpc.NewServer()  pb.RegisterGreeterServer(s, &server{})  s.Serve(lis)  }  // server is used to implement helloworld.GreeterServer.  type server struct{}  // SayHello implements helloworld.GreeterServer  func (s \*server) SayHello(ctx context.Context, in \*pb.HelloRequest) (\*pb.HelloReply, error) {  fmt.Printf("%v: Receive is %s\n", time.Now(), in.Name)  return &pb.HelloReply{Message: "Hello " + in.Name}, nil  } |

**6）实现客户端接口：helloworldclient.go**

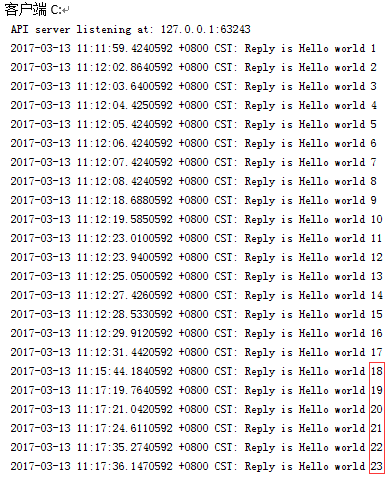
helloworldclient.go

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package main  import (  "flag"  "fmt"  "time"  grpclb "com.midea/jr/grpclb/naming/etcd/v3"  "com.midea/jr/grpclb/example/pb"  "golang.org/x/net/context"  "google.golang.org/grpc"  "strconv"  )  var (  serv = flag.String("service", "hello\_service", "service name")  reg = flag.String("reg", "http://127.0.0.1:2379", "register etcd address")  )  func main() {  flag.Parse()  r := grpclb.NewResolver(\*serv)  b := grpc.RoundRobin(r)  ctx, \_ := context.WithTimeout(context.Background(), 10\*time.Second)  conn, err := grpc.DialContext(ctx, \*reg, grpc.WithInsecure(), grpc.WithBalancer(b))  if err != nil {  panic(err)  }  ticker := time.NewTicker(1 \* time.Second)  for t := range ticker.C {  client := pb.NewGreeterClient(conn)  resp, err := client.SayHello(context.Background(), &pb.HelloRequest{Name: "world " + strconv.Itoa(t.Second())})  if err == nil {  fmt.Printf("%v: Reply is %s\n", t, resp.Message)  }  }  } |

**7）运行测试**

1、运行3个服务端S1、S2、S3，1个客户端C，观察各服务端接收的请求数是否相等？  
[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/5.png)

2、关闭1个服务端S1，观察请求是否会转移到另外2个服务端？  
[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/6.png)

3、重新启动S1服务端，观察另外2个服务端请求是否会平均分配到S1？  
[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/7.png)  
[](https://colobu.com/2017/03/25/grpc-naming-and-load-balance/8.png)

4、关闭Etcd3服务器，观察客户端与服务端通信是否正常？ 关闭通信仍然正常，但新服务端不会注册进来，服务端掉线了也无法摘除掉。

5、重新启动Etcd3服务器，服务端上下线可自动恢复正常。

6、关闭所有服务端，客户端请求将被阻塞。

***参考***

1. <http://www.grpc.io/docs/>
2. <https://github.com/grpc/grpc/blob/master/doc/load-balancing.md>