三、go-kit 与 grpc 结合实现注册发现与负载均衡

作者：450370050
原文链接：https://ld246.com/article/1524894068545
来源网站：链滴

介绍

grpc提供了简单的负载均衡，需要自己实现服务发现resolve。我们既然要使用go-kit来治理微服务那么我们就使用go-kit的注册发现、负载均衡机制。

go-kit官方【stringsvc3】例子中使用的负载均衡方案是通过服务端转发进行，翻找下源码go-kit的务注册发现、负载均衡在【sd】包中。下面我们介绍怎么通过go-kit进行客户端负载均衡。

go-kit提供的注册中心

1、 etcd

2、 consul

3、 eureka

4、 zookeeper

go-kit提供的负载均衡

1、 random[随机]

2、 roundRobin[轮询]

只需实现Balancer接口，我们可以很容易的增加其它负载均衡机制

```
type Balancer interface {
   Endpoint() (endpoint.Endpoint, error)
}
```

etcd注册发现

etcd和zookeeper类似是一个高可用、强一致性的存储仓库，拥有服务发现功能。 我们就通过go-kit供的etcd包来实现服务注册发现

服务端代码

服务注册

1、连接注册中心

2、注册当前服务

```
var (
   //etcd服务地址
   etcdServer = "127.0.0.1:2379"
   //服务的信息目录
   prefix     = "/services/book/"
   //当前启动服务实例的地址
   instance   = "127.0.0.1:50052"
   //服务实例注册的路径
   key        = prefix + instance
```

```
    //服务实例注册的val
    value       = instance
    ctx         = context.Background()
    //服务监听地址
    serviceAddress = ":50052"
)

//etcd的连接参数
options := etcdv3.ClientOptions{
    DialTimeout: time.Second * 3,
    DialKeepAlive: time.Second * 3,
}
//创建etcd连接
client, err := etcdv3.NewClient(ctx, []string{etcdServer}, options)
if err != nil {
    panic(err)
}

// 创建注册器
registrar := etcdv3.NewRegistrar(client, etcdv3.Service{
    Key:    key,
    Value: value,
}, log.NewNopLogger())

// 注册器启动注册
registrar.Register()
```

完整代码

```
package main

import (
    "grpc-test/pb"
    "context"
    grpc_transport "github.com/go-kit/kit/transport/grpc"
    "github.com/go-kit/kit/endpoint"
    "google.golang.org/grpc"
    "net"
    "github.com/go-kit/kit/sd/etcdv3"
    "github.com/go-kit/kit/log"
    "time"
)

type BookServer struct {
    bookListHandler  grpc_transport.Handler
    bookInfoHandler  grpc_transport.Handler
}

//通过grpc调用GetBookInfo时,GetBookInfo只做数据透传, 调用BookServer中对应Handler.ServeG
PC转交给go-kit处理
func (s *BookServer) GetBookInfo(ctx context.Context, in *book.BookInfoParams) (*book.Book
nfo, error) {
    _, rsp, err := s.bookInfoHandler.ServeGRPC(ctx, in)
    if err != nil {
```

```
    return nil, err
  }
  return rsp.(*book.BookInfo),err
}

//通过grpc调用GetBookList时,GetBookList只做数据透传, 调用BookServer中对应Handler.ServeGRC转交给go-kit处理
func (s *BookServer) GetBookList(ctx context.Context, in *book.BookListParams) (*book.BookList, error) {
  _, rsp, err := s.bookListHandler.ServeGRPC(ctx, in)
  if err != nil {
    return nil, err
  }
  return rsp.(*book.BookList),err
}

//创建bookList的EndPoint
func makeGetBookListEndpoint() endpoint.Endpoint {
  return func(ctx context.Context, request interface{}) (interface{}, error) {
    //请求列表时返回 书籍列表
    bl := new(book.BookList)
    bl.BookList = append(bl.BookList, &book.BookInfo{BookId:1,BookName:"21天精通php"})
    bl.BookList = append(bl.BookList, &book.BookInfo{BookId:2,BookName:"21天精通java"})
    return bl,nil
  }
}

//创建bookInfo的EndPoint
func makeGetBookInfoEndpoint() endpoint.Endpoint {
  return func(ctx context.Context, request interface{}) (interface{}, error) {
    //请求详情时返回 书籍信息
    req := request.(*book.BookInfoParams)
    b := new(book.BookInfo)
    b.BookId = req.BookId
    b.BookName = "21天精通php"
    return b,nil
  }
}

func decodeRequest(_ context.Context, req interface{}) (interface{}, error) {
  return req, nil
}

func encodeResponse(_ context.Context, rsp interface{}) (interface{}, error) {
  return rsp, nil
}

func main() {
  var (
    //etcd服务地址
    etcdServer = "127.0.0.1:2379"
    //服务的信息目录
    prefix     = "/services/book/"
    //当前启动服务实例的地址
```

```
    instance    = "127.0.0.1:50052"
    //服务实例注册的路径
    key         = prefix + instance
    //服务实例注册的val
    value       = instance
    ctx         = context.Background()
    //服务监听地址
    serviceAddress = ":50052"
  )

  //etcd的连接参数
  options := etcdv3.ClientOptions{
    DialTimeout: time.Second * 3,
    DialKeepAlive: time.Second * 3,
  }
  //创建etcd连接
  client, err := etcdv3.NewClient(ctx, []string{etcdServer}, options)
  if err != nil {
    panic(err)
  }

  // 创建注册器
  registrar := etcdv3.NewRegistrar(client, etcdv3.Service{
    Key:    key,
    Value: value,
  }, log.NewNopLogger())

  // 注册器启动注册
  registrar.Register()


  bookServer := new(BookServer)
  bookListHandler := grpc_transport.NewServer(
    makeGetBookListEndpoint(),
    decodeRequest,
    encodeResponse,
  )
  bookServer.bookListHandler = bookListHandler


  bookInfoHandler := grpc_transport.NewServer(
    makeGetBookInfoEndpoint(),
    decodeRequest,
    encodeResponse,
  )
  bookServer.bookInfoHandler = bookInfoHandler


  ls, _ := net.Listen("tcp", serviceAddress)
  gs := grpc.NewServer(grpc.UnaryInterceptor(grpc_transport.Interceptor))
  book.RegisterBookServiceServer(gs, bookServer)
  gs.Serve(ls)
}
```

客户端代码

客户端流程

1、 连接注册中心

2、 获取提供的服务

3、 监听服务目录变化，目录变化更新本地缓存

4、 创建负载均衡器

5、 获取请求的 endPoint

完整代码

```
package main

import (
  "context"
  "github.com/go-kit/kit/sd/etcdv3"
  "time"
  "github.com/go-kit/kit/sd"
  "github.com/go-kit/kit/log"
  "github.com/go-kit/kit/endpoint"
  "io"
  "github.com/go-kit/kit/sd/lb"
  "grpc-test/pb"
  "fmt"
  "google.golang.org/grpc"
)

func main() {

  var (
    //注册中心地址
    etcdServer = "127.0.0.1:2379"
    //监听的服务前缀
    prefix     = "/services/book/"
    ctx        = context.Background()
  )
  options := etcdv3.ClientOptions{
    DialTimeout: time.Second * 3,
    DialKeepAlive: time.Second * 3,
  }
  //连接注册中心
  client, err := etcdv3.NewClient(ctx, []string{etcdServer}, options)
  if err != nil {
    panic(err)
  }
  logger := log.NewNopLogger()
  //创建实例管理器, 此管理器会Watch监听etc中prefix的目录变化更新缓存的服务实例数据
  instancer, err := etcdv3.NewInstancer(client, prefix, logger)
```

```
    if err != nil {
        panic(err)
    }
    //创建端点管理器， 此管理器根据Factory和监听的到实例创建endPoint并订阅instancer的变化动
更新Factory创建的endPoint
    endpointer := sd.NewEndpointer(instancer, reqFactory, logger)
    //创建负载均衡器
    balancer := lb.NewRoundRobin(endpointer)

    /**
    我们可以通过负载均衡器直接获取请求的endPoint，发起请求
    reqEndPoint,_ := balancer.Endpoint()
    */

    /**
    也可以通过retry定义尝试次数进行请求
    */
    reqEndPoint := lb.Retry(3, 3*time.Second, balancer)

    //现在我们可以通过 endPoint 发起请求了
    req := struct{}{}
    if _, err = reqEndPoint(ctx, req); err != nil {
        panic(err)
    }
}


//通过传入的 实例地址  创建对应的请求endPoint
func reqFactory(instanceAddr string) (endpoint.Endpoint, io.Closer, error) {
    return func(ctx context.Context, request interface{}) (interface{}, error) {
        fmt.Println("请求服务: ", instanceAddr)
        conn, err := grpc.Dial(instanceAddr, grpc.WithInsecure())
        if err != nil {
            fmt.Println(err)
            panic("connect error")
        }
        defer conn.Close()
        bookClient := book.NewBookServiceClient(conn)
        bi,_:=bookClient.GetBookInfo(context.Background(),&book.BookInfoParams{BookId:1})
        fmt.Println("获取书籍详情")
        fmt.Println("bookId: 1", " => ", "bookName:", bi.BookName)

        bl,_ := bookClient.GetBookList(context.Background(), &book.BookListParams{Page:1, Limit
10})
        fmt.Println("获取书籍列表")
        for _,b := range bl.BookList {
            fmt.Println("bookId:", b.BookId, " => ", "bookName:", b.BookName)
        }
        return nil,nil
    },nil,nil
}
```

测试

请求测试

```
请求服务：  127.0.0.1:50052

获取书籍详情
bookId: 1  =>  bookName: 21天精通php
获取书籍列表
bookId: 1  =>  bookName: 21天精通php
bookId: 2  =>  bookName: 21天精通java
```

负载均衡测试

1、 修改server的注册监听端口，启动多个server

```
instance    = "127.0.0.1:50052"
serviceAddress = ":50052"
```

2、client发起多次请求

```
req := struct{}{}
for i := 1; i <= 8; i++ {
   if _, err = reqEndPoint(ctx, req); err != nil {
      panic(err)
   }
}
```

通过返回结果中记录的请求地址，我们可以看到已经按照轮询的方式请求不同的微服务实例。

```
请求服务:  127.0.0.1:50051

        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50052
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50051
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50052
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50051
```

```
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50052
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50051
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java
请求服务:  127.0.0.1:50052
        获取书籍详情
        bookId: 1  =>  bookName: 21天精通php
        获取书籍列表
        bookId: 1  =>  bookName: 21天精通php
        bookId: 2  =>  bookName: 21天精通java

Process finished with exit code 0
```