六、go-kit 微服务请求跟踪介绍

作者：450370050
原文链接：https://ld246.com/article/1525401758789
来源网站：链滴

介绍

go-kit 提供了两种tracing请求跟踪

1、opentracing【跟踪标准】

2、zipkin【zipkin的go封装】

我们下面来介绍下zipkin在go-kit中的使用方法。

zipkin安装启动

1、 java -version <font color=red>现在安装zipkin，必须使用java8（即java-1.8.0-openjdk）</fnt>

2、 wget -O zipkin.jar 'https://search.maven.org/remote_content?g=io.zipkin.java&a=zipkin-server&v=LATEST&c=exec'

3、 java -jar zipkin.jar

哈哈， 只是大概介绍下zipkin的安装 出现问题需要自己解决了。

go-kit的zipkin

服务端trace

```
//创建zipkin上报管理器
reporter := http.NewReporter("http://localhost:9411/api/v2/spans")

//运行结束，关闭上报管理器的for-select协程
defer reporter.Close()

//创建trace跟踪器
zkTracer, err := opzipkin.NewTracer(reporter)

//添加grpc请求的before after finalizer 事件对应要处理的trace操作方法
zkServerTrace := zipkin.GRPCServerTrace(zkTracer)

//通过options的方式运行trace
bookListHandler := grpctransport.NewServer(
   bookListEndPoint,
   decodeRequest,
   encodeResponse,
   zkServerTrace,
)
```

完整代码

我们还在之前的代码中加入，trace的代码

```
package main
```

```
import (
   "grpc-test/pb"
   "context"
   grpctransport
   "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"
   "golang.org/x/time/rate"
   "github.com/go-kit/kit/ratelimit"
   opzipkin "github.com/openzipkin/zipkin-go"
   "github.com/openzipkin/zipkin-go/reporter/http"
   "github.com/go-kit/kit/tracing/zipkin"
   "math/rand"
)

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

//通过grpc调用GetBookInfo时,GetBookInfo只做数据透传, 调用BookServer中对应Handler.ServeGPC转交给go-kit处理
func (s *BookServer) GetBookInfo(ctx context.Context, in *book.BookInfoParams) (*book.BookInfo, 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) {
      rand.Seed(time.Now().Unix())
      randInt := rand.Int63n(200)
      time.Sleep( time.Duration(randInt) * time.Millisecond)
      //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) {
        rand.Seed(time.Now().Unix())
        randInt := rand.Int63n(200)
        time.Sleep( time.Duration(randInt) * time.Microsecond)
        //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   = "127.0.0.1:50051"
        //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 = ":50051"
    )

    //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()

    reporter := http.NewReporter("http://localhost:9411/api/v2/spans")
    defer reporter.Close()
    zkTracer, err := opzipkin.NewTracer(reporter)
    zkServerTrace := zipkin.GRPCServerTrace(zkTracer)
    bookServer := new(BookServer)
    bookListEndPoint := makeGetBookListEndpoint()
    //创建限流器 1r/s  limiter := rate.NewLimiter(rate.Every(time.Second * 1), 100000)
    //通过DelayingLimiter中间件，在bookListEndPoint的外层再包裹一层限流的endPoint
    bookListEndPoint = ratelimit.NewDelayingLimiter(limiter)(bookListEndPoint)

    bookListHandler := grpctransport.NewServer(
        bookListEndPoint,
        decodeRequest,
        encodeResponse,
        zkServerTrace,
    )
    bookServer.bookListHandler = bookListHandler


    bookInfoEndPoint := makeGetBookInfoEndpoint()
    //通过DelayingLimiter中间件，在bookListEndPoint的外层再包裹一层限流的endPoint
    bookInfoEndPoint = ratelimit.NewDelayingLimiter(limiter)(bookInfoEndPoint)
    bookInfoHandler := grpctransport.NewServer(
        bookInfoEndPoint,
        decodeRequest,
        encodeResponse,
        zkServerTrace,
    )
    bookServer.bookInfoHandler = bookInfoHandler

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

客户端trace

首先改造请求方式，之前我们是通过grpc的idl生成的代码直接发起的请求。我们要改成go-kit endPo
int的方式，这样利于增加其它middleware或options扩展功能，通过grpctransport.NewClient().End

oint()可以获取到请求的endPoint

原来的代码

```
//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, "当前时间: ", time.Now().Format("2006-01-02 15:04:
5.99"))
      conn, err := grpc.Dial(instanceAddr, grpc.WithInsecure())
      if err != nil {
         fmt.Println(err)
         panic("connect error")
      }
      defer conn.Close()
      bookClient := book.NewBookServiceClient(conn)
      bi,err := bookClient.GetBookInfo(context.Background(),&book.BookInfoParams{BookId:1})

      fmt.Println(bi)
      fmt.Println("       ", "获取书籍详情")
      fmt.Println("       ", "bookId: 1", " => ", "bookName:", bi.BookName)
      return nil,nil
   },nil,nil
}
```

改造后

```
//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, "当前时间: ", time.Now().Format("2006-01-02 15:04:
5.99"))
      conn, err := grpc.Dial(instanceAddr, grpc.WithInsecure())
      if err != nil {
         fmt.Println(err)
         panic("connect error")
      }
      bookInfoRequest := grpctransport.NewClient(
         conn,
         "BookService",
         "GetBookInfo",
         func(_ context.Context, in interface{}) (interface{}, error) { return nil, nil },
         func(_ context.Context, out interface{}) (interface{}, error) {
            return out, nil
         },
         book.BookInfo{},
      ).Endpoint()

      bookListRequest := grpctransport.NewClient(
         conn,
         "BookService",
         "GetBookList",
         func(_ context.Context, in interface{}) (interface{}, error) { return nil, nil },
         func(_ context.Context, out interface{}) (interface{}, error) {
```

```
            return out, nil
        },
        book.BookList{},
    ).Endpoint()

    infoRet ,_:= bookInfoRequest(ctx, request)
    bi := infoRet.(*book.BookInfo)
    fmt.Println("获取书籍详情")
    fmt.Println("bookId: 1", " => ", "bookName:", bi.BookName)

    listRet,_ := bookListRequest(ctx, request)
    bl := listRet.(*book.BookList)
    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
}
```

增加trace代码

与服务端trace的区别在于kitzipkin.GRPCClientTrace

```
reporter := http.NewReporter("http://localhost:9411/api/v2/spans")
defer reporter.Close()

zkTracer, err := opzipkin.NewTracer(reporter)
zkClientTrace := zipkin.GRPCClientTrace(zkTracer)
```

可以通过span组装span结构树

```
parentSpan := zkTracer.StartSpan("bookCaller")
defer parentSpan.Flush()
ctx = opzipkin.NewContext(context.Background(), parentSpan)
```

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"
  "fmt"
  "google.golang.org/grpc"
  "github.com/afex/hystrix-go/hystrix"
  "github.com/go-kit/kit/circuitbreaker"
```

```
    opzipkin "github.com/openzipkin/zipkin-go"
    "github.com/go-kit/kit/tracing/zipkin"
    grpctransport "github.com/go-kit/kit/transport/grpc"
    "grpc-test/pb"
    "github.com/openzipkin/zipkin-go/reporter/http"
)

func main() {
    commandName := "my-endpoint"
    hystrix.ConfigureCommand(commandName, hystrix.CommandConfig{
        Timeout: 1000 * 30,
        ErrorPercentThreshold: 1,
        SleepWindow: 10000,
        MaxConcurrentRequests: 1000,
        RequestVolumeThreshold: 5,
    })
    breakerMw := circuitbreaker.Hystrix(commandName)


    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, 100*time.Second, balancer)

    //增加熔断中间件
    reqEndPoint = breakerMw(reqEndPoint)
    //现在我们可以通过 endPoint 发起请求了

    req := struct{}{}
    for i := 1; i <= 1; i++ {
        if _, err = reqEndPoint(ctx, req); err != nil {
            fmt.Println(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, "当前时间: ", time.Now().Format("2006-01-02 15:04:
5.99"))
        conn, err := grpc.Dial(instanceAddr, grpc.WithInsecure())
        if err != nil {
            fmt.Println(err)
            panic("connect error")
        }

        reporter := http.NewReporter("http://localhost:9411/api/v2/spans")
        defer reporter.Close()

        zkTracer, err := opzipkin.NewTracer(reporter)
        zkClientTrace := zipkin.GRPCClientTrace(zkTracer)

        bookInfoRequest := grpctransport.NewClient(
            conn,
            "BookService",
            "GetBookInfo",
            func(_ context.Context, in interface{}) (interface{}, error) { return nil, nil },
            func(_ context.Context, out interface{}) (interface{}, error) {
                return out, nil
            },
            book.BookInfo{},
            zkClientTrace,
        ).Endpoint()

        bookListRequest := grpctransport.NewClient(
            conn,
            "BookService",
            "GetBookList",
            func(_ context.Context, in interface{}) (interface{}, error) { return nil, nil },
            func(_ context.Context, out interface{}) (interface{}, error) {
                return out, nil
            },
            book.BookList{},
            zkClientTrace,
```

```
    ).Endpoint()

    parentSpan := zkTracer.StartSpan("bookCaller")
    defer parentSpan.Flush()

    ctx = opzipkin.NewContext(ctx, parentSpan)
    infoRet ,_:= bookInfoRequest(ctx, request)
    bi := infoRet.(*book.BookInfo)
    fmt.Println("获取书籍详情")
    fmt.Println("bookId: 1", " => ", "bookName:", bi.BookName)

    listRet,_ := bookListRequest(ctx, request)
    bl := listRet.(*book.BookList)
    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
}
```

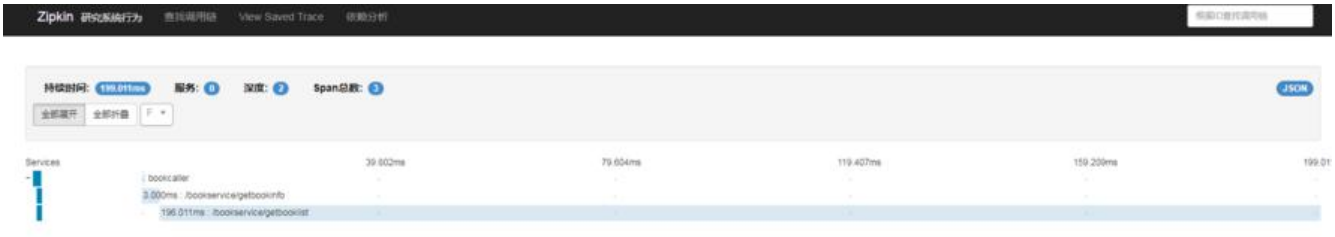
运行

```
请求服务:  127.0.0.1:50051 当前时间:  2018-05-04 10:34:55.91

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
```

访问http://localhost:9411/zipkin/ 查看我们的zipkin中的记录



总结

至此基于go-kit的功能基本都介绍了，log模块和其它middleware差不多就不再描述。