组件标准化改造指导

一、流量入口组件

1.1、grpc/http service

使用chassis框架提供的grpc服务、rest服务和gin服务。

依赖

go.mod

```
require git.garena.com/shopee/bg-logistics/go/chassis v0.4.1-b.2.master0.3.1.r.10
```

配置

无

示例

grpc服务

略，详见

rest服务

略，详见

gin服务

略，详见

限流

限流组件是面向chassis框架内置服务提供的内置组件。目前只有当使用chassis框架的内置服务作为流量入口时，才能使用限流组件。

限流组件除了常规的限流功能外，还对全链路压测提供了特殊支持：保证正常流量的高优先级，正常流量不会受到压测流量的影响。

举例说明：一个配置了1000QPS的限流组件，在压测流量达到或者超过1000QPS的情况下，只要正常流量的平均流量不超过1000QPS且瞬时流量（1ms内的访问量）不超过100（QPS阈值的10%），正常流量不会被限流。

使用该功能需进行配置：

chassis.yaml

```
---
chassis:
  plugins: #请求处理过程中，各种插件的配置
    rateLimit: #频限插件
      service: #服务端入口流量的频限
        __total: #服务端所有入口流量的频限配置
          enabled: true #启用频限，默认为false
          local: #本地频限
            burst: 1000 # 限制并发，如果设置并发配置，同级的qps配置会失效, 目前最高并发限制为100000, 超过会取100000
            qps: 100000
```

以上是服务维度的限流。burst表示并发限制，也就是限制该资源同时访问的次数；qps是则是限制该资源每秒的访问次数。在同级下两种类型只能选其一，同时设置只有burst生效。

补充点：不同的接口不同的限流如何配？配置能否动态生效，还是需要服务重启？

1.2、saturn 定时任务

依赖

go.mod
require git.garena.com/shopee/bg-logistics/service/saturn-rpc-job v1.3.0-a6

配置

无

示例

影子定时任务

任务名以 _shadow 结尾都会认为是影子定时任务，在影子定时任务执行时，流量上会带上影子标签，使得整个调用链路都会走影子流量。

影子定时任务不需要额外调用 RegisterRpcJob 做任务的注册，框架层面检测到当前是影子任务会自动使用真实任务的handler自动做一次任务的注册。

唯一需要做的，则是在saturn任务中心去创建一个影子定时任务，任务名有特定要求：原任务名+"_shadow" 。影子任务和真实任务的配置可以一致，也可以不一致，视真实需求而定。

影子消息任务

针对全链路压测中，消息任务目前支持两种流量标记的手段

1. 创建独立的影子topic，影子topic中的消息自动标记为影子流量
2. 跟正常的业务消息公用一个topic，通过消息中header部分来区分是影子流量的消息

影子消息任务（影子topic）

其实这种情况相当于完全复制一个任务，topic是额外的，任务也是额外的。为了与真实任务对应得起来，对于任务名有特定要求：原任务名+"_shadow" 。并且任务中的配置topic是对应的影子topic。

影子任务和真实任务的配置可以一致，也可以不一致，视真实需求而定。

影子消息复用业务topic

这种情况，我们不需要做任何修改，是否影子任务完全靠消息中的标识来确定。

1.3、saturn 消息队列consumer

依赖

go.mod

```
require git.garena.com/shopee/bg-logistics/go/chassis-saturn-server v1.0.1
```

配置

chassis.yaml

```
---
chassis:
  plugins:
    mq:
      consumer:
        default:
          shadowTopic:
            enabled: true #默认为false。表示是否自动监听shadow topic。
          expiration: 1646733041 #当影子消息的入队时间早于该时间戳，则该影子消息被自动丢弃。对正式消息无影响。
```

以上配置主要是针对影子消息，配置是否开启监控听和过期时间。

示例

无

二、流量出口组件

2.1、mysql

mysql的客户端标准组件为scorm。对于全链路压测，mysql的解决方案是使用影子库。

依赖

使用chassis框架的服务，升级到相应的版本即包含了scorm。

go.mod

```
require git.garena.com/shopee/bg-logistics/go/chassis v0.4.1-b.2.master0.3.1.r.10
```

没有使用chassis框架的服务，需要额外引入scorm库，并且在初始化scorm句柄的时候也需要手动显式处理。(该方式不推荐使用，目前我们应该绝大部分都是使用chassis的服务吧)

go.mod

```
require git.garena.com/shopee/bg-logistics/go/scorm v0.1.0-alpha.4.master0.0.23
```

配置

chassis.yaml

```
---
chassis:
  plugins:
    scorm:
      default: #默认的数据库配置，必须包含一个名为default的数据库，用于scorm.Context方法从context中读取
      driver: mysql #数据库的驱动
      autoReport: false #mysql语句是否自动上报cat
      logDisabled: false #是否打开日志模式，默认打开
      maxIdleConns: 10 #最大空闲连接数
      maxOpenConns: 100 #最大连接数
      connMaxLifetime: 10000 #连接最大生命周期，单位毫秒
      groups: #集群下的主从复制组，可以配置多个复制组
        - masterDsn: "root:123456@(127.0.0.1:3306)/scorm_db"
      shadowGroups: #对应的影子集群
        - masterDsn: "root:123456@(127.0.0.1:3306)/shadow_scorm_db"
```

示例

scorm 的日常使用说明书？

2.2、redis

redis的客户端标准组件为gocommon中的redis客户端。对于全链路压测，redis的解决方案是使用影子key。

依赖

go.mod
require git.garena.com/shopee/bg-logistics/go/chassis v0.4.1-b.2.master0.3.1.r.10

配置

无

示例

redis的日常使用说明书？

2.3、grpc调用

grpc调用需使用chassis 提供的 rpc invoker（grpc客户端）。

依赖

go.mod
require git.garena.com/shopee/bg-logistics/go/chassis v0.4.1-b.2.master0.3.1.r.10

配置

无

示例

正常调用

如果使用GRPC调用仅用于传递影子标记，则GRPC调用无需进行任何改动，像原来一样正常使用即可。(需要详细使用文档)

```
func (s *Server) SayHello(b *rf.
Context) {
   invoker := chassis.NewRPCInvoker()
   reply := helloworld.HelloReply{}
   err := invoker.Invoke(
      b.Ctx,       //context.Context
ctx
      "inject-grpc-local-sg",     //
      "helloworld.Greeter",       //
      "SayHello",                 //
      &helloworld.HelloRequest{},
//grpc
      &reply,
//grpcreply
      //option
      chassis.WithProtocol("grpc"),
//
   )
   if err != nil {
      println("grpc invoke failed:",
err.Error())
      b.JSON(http.StatusOK, &respData
{Data: err.Error()})
   } else {
      println("grpc invoke success:"
, reply.String())
      b.JSON(http.StatusOK, &respData
{Data: reply.String()})
   }
}
```

mock调用

如果需要屏蔽某些GRPC调用，控制压测范围，则需要使用mock功能，mock功能需要进行配置改动。

step1：在配置文件中定义屏蔽的调用

chassis.yaml

```
---
chassis:
  plugins:
    router:
      mock:
        - serviceName: XXXSvc #访问该目标服务的所有请求都屏蔽
        - serviceName: XXXSvc #访问该目标服务的SayHello接口的请求才会被屏蔽
          interfaceName: SayHello
```

step2：在chassis初始化时注册一个mock的handler

example.go

```
package main

import (
    pb "chassis-example/plugin/logger/grpc/helloworld"
    "context"
    "git.garena.com/shopee/bg-logistics/go/chassis"
    "git.garena.com/shopee/bg-logistics/go/chassis/core/invocation"
    "git.garena.com/shopee/bg-logistics/go/chassis/handler"
)

func main() {
    handler.RegisterInjectHandler(handler.WithInjection(func(ctx context.Context, serviceName string, interfaceName string,

        headers map[string]string, req interface{}, rsp interface{}) (status int, mockRsp interface{}) {
        if _, ok := req.(*pb.HelloRequest); ok {
            return invocation.StatusOk, &pb.HelloReply{
                Message:            "MockMessage",
            }
        }
        return invocation.StatusOk, rsp
    }))

    err := chassis.Init(chassis.WithDefaultProviderHandlerChain(handler.InjectConsumerName))
    if err != nil {
        panic(err)
    }
   ......
}
```

step3：参考正常调用，略。

2.4、http调用

http 调用需使用chassis 提供的 rest invoker（http客户端）。

依赖

go.mod

```
require git.garena.com/shopee/bg-logistics/go/chassis v0.4.1-b.2.master0.3.1.r.10
```

配置

主要配置http客户端的连接数及超时信息，由于其底层使用的是net/http 的客户端，故配置信息基本为该客户端的信息，下面有对照说明

chassis.yaml

```
$CHASSIS.client.rest.dailTimeoutMs: 60000 // 60秒
cse.transport.maxIdleCon.rest: 100
$CHASSIS.client.rest.idleConnTimeoutMs: 120000 //120秒
cse.isolation.Consumer.timeoutInMilliseconds:60000 //60秒
```

各配置的对应关系如下:

```
var defaultClient = http.Client{
   Transport: &http.Transport{
      DialContext: (&net.Dialer{
         Timeout:   60 * time.Second,
//$CHASSIS.client.rest.dailTimeoutMs
         KeepAlive: 60 * time.Second, //60s
      }).DialContext,
      MaxIdleConns:        100,
// cse.transport.maxIdleCon.rest
      MaxIdleConnsPerHost: 8,
// cse.transport.maxIdleCon.rest
      IdleConnTimeout:     120 * time.Second,
//$CHASSIS.client.rest.idleConnTimeoutMs
   },
   Timeout: time.Second * 60, //cse.isolation.
Consumer.timeoutInMilliseconds
}
```

示例

正常调用

如果http client 调用仅用于传递影子标记，则调用无需进行任何改动，像原来一样正常使用即可。(需要详细使用文档)

```
func (s *Server) SayHai(b *rf.Context) {
   invoker := chassis.NewRestInvoker()
   request, err := chassis.NewRestRequest(
      "POST", //HTTP methodmethod
      "http://127.0.0.1:7000/openapi/ops/sayhi", //URL
ip:port
      []byte(`{"name": "test name1"}`),          //POST
POSTdataGETnil
   )
   if err != nil {
      println("err: ", err)
      return
   }
   //
   ctx, cancel := context.WithTimeout(b.Ctx, time.
Duration(2)*time.Second)
   defer cancel()
   //header
   request.Header.Set("Content-Type", "application/json"
)
   request.Header.Set("X-Account", "celestial")
   response, err := invoker.Invoke(
      ctx,   //context.Contextctx
      request, //request
   )
   //httputil.ReadBodyresponse
   data := httputil.ReadBody(response)
   //body
   defer response.Body.Close()
   if err != nil {
      println("rest invoke failed:", err.Error())
   } else {
      println("rest invoke success:", data)
   }
}
```

mock调用

如果需要屏蔽某些http调用，控制压测范围，则需要使用mock功能，mock功能需要进行配置改动。

step1：在配置文件中定义屏蔽的调用

```
chassis.yaml
```

```
---
chassis:
  plugins:
    router:
      mock:
        - serviceName: https://celestial.ssc.test.shopee.io  #可以填请求的域名
        - serviceName: https://celestial.ssc.test.shopee.io
          mockServiceEndpoint: /api/fault_inject/whitelist/list?size=10&offset=1
```

step2：在chassis初始化时注册一个mock的handler

example.go

```
package main

import (
    pb "chassis-example/plugin/logger/grpc/helloworld"
    "context"
    "git.garena.com/shopee/bg-logistics/go/chassis"
    "git.garena.com/shopee/bg-logistics/go/chassis/core/invocation"
    "git.garena.com/shopee/bg-logistics/go/chassis/handler"
)


type HttpResponse struct {
  Retcode int         `json:"retcode"`
  Message string      `json:"message"`
  Data    interface{} `json:"data,omitempty"`
}


func main() {
    handler.RegisterInjectHandler(handler.WithInjection(func(ctx context.Context, serviceName string, interfaceName string,

        headers map[string]string, req interface{}, rsp interface{}) (status int, mockRsp interface{}) {
        if serviceName == "https://celestial.ssc.test.shopee.io " {
            return invocation.StatusOk, &HttpResponse{
                Retcode: 0,
                Message: "Success",
                Data:    "ok",
            }
        }
        return invocation.StatusOk, rsp
    }))

    err := chassis.Init(chassis.WithDefaultProviderHandlerChain(handler.InjectConsumerName))
    if err != nil {
        panic(err)
    }
   ......
}
```

step3：参考正常调用，略。

2.5、异步调用客户端

saturn producer不支持全链路压测，建议更改为使用esarama的producer。

Sarama Producer支持自动丢弃影子消息（实现类似于mock）的效果。

依赖

go.mod

require git.garena.com/shopee/bg-logistics/go/chassis v0.4.0

github.com/Shopify/sarama => sarama git.garena.com/shopee/bg-logistics/go/chassis/pkg/mq/esarama

配置

chassis.yaml

```
---
chassis:
  plugins:
    mq:
      producer:
        default:
          shadowTopic:
            enabled: true #默认为false。表示是否自动监听shadow topic。
          mock:
            enabled: true #默认为false。表示是否自动丢弃影子消息
```

示例

对example 不太明白，需要请教一下

三、辅助组件

3.1、cat监控

依赖

配置

示例

3.2、promethues监控

依赖

配置

示例

3.3、Spex mock

依赖

配置

示例

四、业务改造

为了保障调用链上的影子标识不丢失，除了在流量出入口使用标准组件之处，在业务处理时也需要遵守规则，以保障标识不丢失。

rule 1：一个context原则

流量标记在服务内部是通过Context进行传递的。

每一个请求对应一个原始的Context。

- 对于chassis的grpc服务、rest服务和gin服务来说，Context由chassis框架提供；
- 对于定时任务来说，原始Context由开发人员自己生成；
- 对于consumer来说，原始Context从Header中提取，或者由consumer给出。

所以在业务处理过程中需要用到Context的地方都应该使用该原始Context或者其衍生出的Context，不允许重新生成全新的context！

如果使用任意Context可能会造成影子标记丢失，从而使影子流量污染正式库。

> **Context检测**
>
> chassis框架内部关键的流量出口，会对Context进行检查，如果该Context不安全，则会产生一个CAT Event上报，Event的Type为UnsafeContext。
>
> 同时，会按照1%的采样比例，打印出调用堆栈，方便定位。