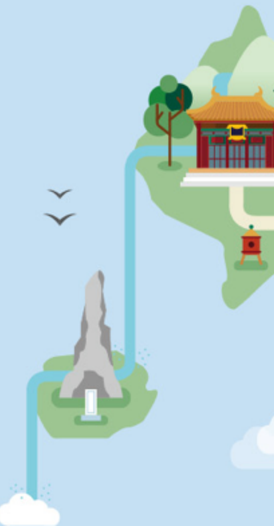
跨境电商的 Go服务治理实践

陈冶 @ ezbuy



目录

1. 背景&前言
2. 开发环境构建 New!
3. 微服务选型 Hot!
4. 分布式追踪 New!
5. 跨数据中心

前言&背景

- 我们正在将业务从 C# 切换到 Go，计划是全部最终均使用 Go 实现。
- 此次分享关注点在于怎么从零打造一整套 Go 服务体系
- 所以我们第一件事就是从规范开发环境做起

开发环境构建: Goflow

- 开发环境统一化
- 第三方依赖方案
- 编译流程一体化

开发环境构建: Goflow

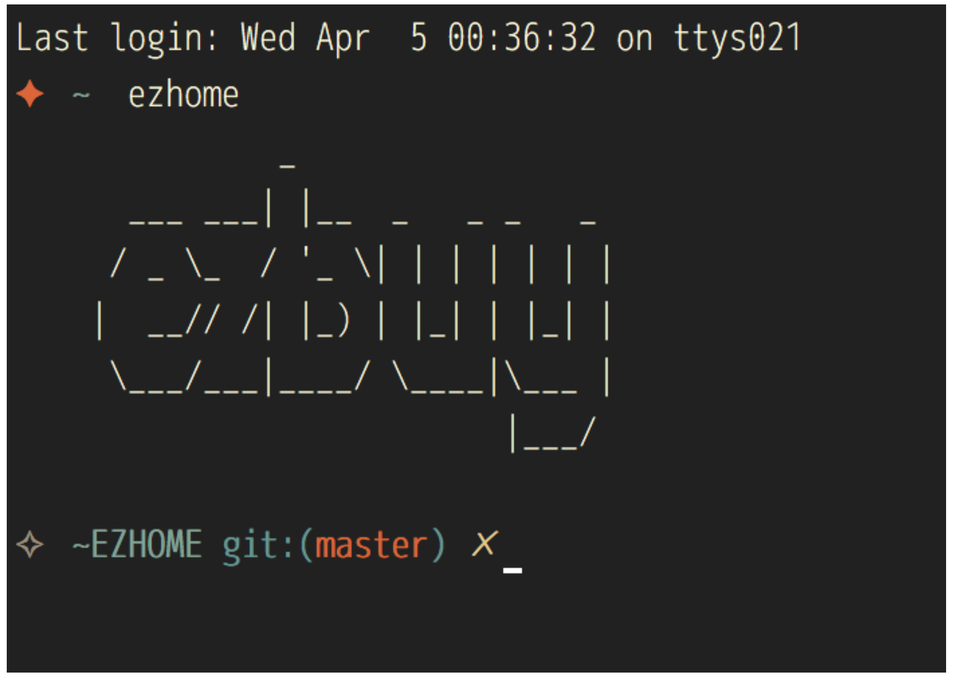
- 设计理念:
  - 没错，我就是 GOPATH
  - 与个人环境 共存 且 相互独立

```
✧ ~EZHOME git:(master) ✗ ls
README.md    bin          bootstrap.sh config       ezbuy.sh     pkg          scripts      src
```

开发环境构建: Goflow

- 修改环境变量 GOPATH，PATH
- 在所在环境注册一些函数
- 负责基础库更新，工具链的编译

当环境被污染时会打印一个Logo



开发环境构建: Goflow - 依赖管理

- 共享依赖！
- 内网缓存，不走小水管
- 和业务代码分开
- “随意”修改第三方包

开发环境构建: Goflow - 依赖管理

- 实现方案
  - 使用官方的 vendor 方案来实现
  - 新建一个仓库来存放所有依赖包
  - 第三方包通过 subtree 加入到仓库内
  - 这个库的名字直接叫 vendor

开发环境构建: Goflow - 依赖管理

- 现在全部第三方库的体积
- 内网下载只要 12 秒 ?!

```
✦ vendor git:(master) du -h -d 1
547M    ./.git
 89M    ./.tmp
8.6M    ./bin
 63M    ./cloud.google.com
202M    ./github.com
516K    ./go4.org
 54M    ./golang.org
 69M    ./google.golang.org
9.2M    ./gopkg.in
296K    ./qiniupkg.com
284K    ./rsc.io
6.4M    ./sourcegraph.com
1.0G    .
```

```
downloading vendor...
Cloning into 'src/[redacted]/ezbuy/vendor'...
remote: Counting objects: 360548, done.
remote: Compressing objects: 100% (112217/112217), done.
Receiving objects:  52% (187494/360548), 123.25 MiB | 21.42 MiB/s
```

开发环境构建: Goflow - 工具链集成

- goflow 是分发平台
- 全部放在vendor里面
- 自我迭代时更新

```
function _install() {
    go install -v xxxxxxxx.com/ezbuy/vendor/$1
}

_install github.com/ezbuy/ezorm
_install github.com/ezbuy/tgen
_install github.com/ezbuy/ezrpc
_install github.com/ezbuy/redis-orm
_install github.com/jteeuwen/go-bindata/...
_install google.golang.org/grpc
_install github.com/golang/protobuf/proto
```

开发环境构建: Goflow - 总结

- 全程自动化 - 个人环境
- 巧妙管理第三方依赖（包括工具链）
- 自我迭代
- （只支持命令行）

开发环境构建: Goflow

微服务选型: gRPC

- 使用pb来描述接口
- 扩展代码生成
- 使用consul用于服务发现和负载均衡

微服务选型: 接口定义

- 包 -> 服务 -> 方法
- 可以拥有接口级别的配置

```
package greeter
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {
    option (google.api.http) = {
      post: "/v1/example/echo"
    };
  }
}
```

微服务选型: 接口扩展

```
syntax = "proto3";
package common;
import "ezbuy/option.proto";

service Product {
    rpc Get(ProductGet) returns (ProductGetResp) {
        option (ezbuy.webapi).enable = true;
        option (ezbuy.webapi).public = true;
    }
    rpc Update(ProductUpdate) returns (ProductUpdateResp);
}
```

```
extend google.protobuf.MethodOptions {
    WebapiOption webapi = 72295728;
}

message WebapiOption {
    // 该方法是否支持webapi
    bool enable = 1;
    // 使用multipart上传文件
    bool upload = 2;
}
```

定义接口特征

微服务选型: gRPC

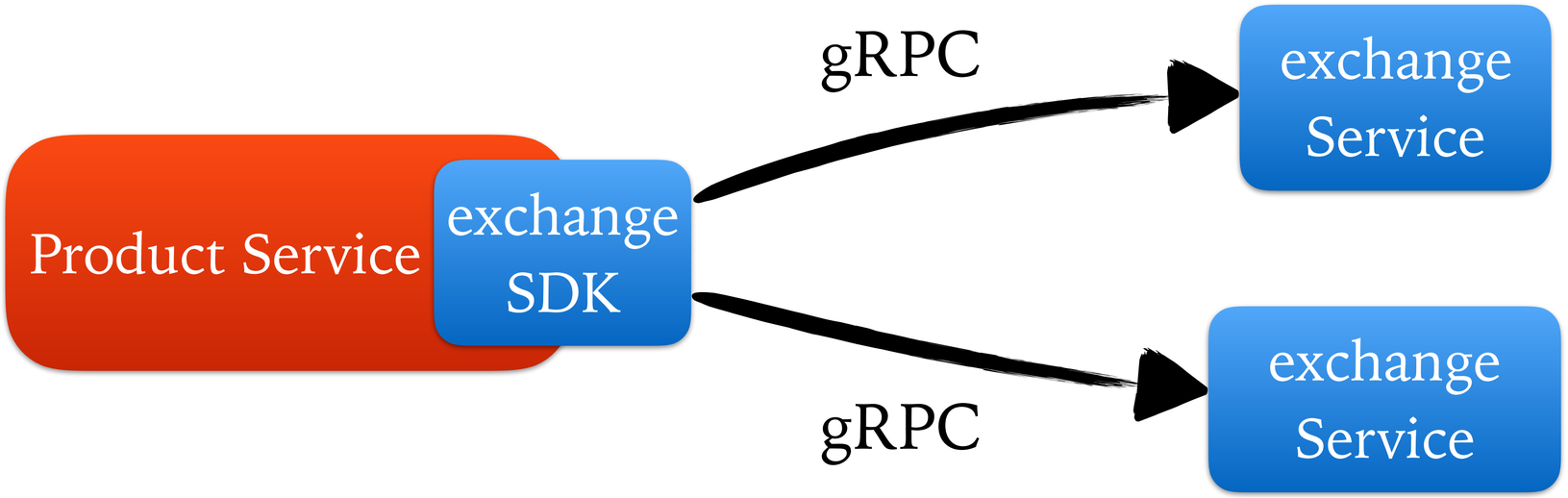
```
syntax = "proto3";
package common;

service Product {
    rpc Get(ProductGet) returns (ProductGetResp);
}
import (
    "ezbuy.com/ezbuy/common/rpc/common"
)

func GetProduct(ctx context.Context) {
    common.GetProduct().Get(ctx, ...)
}
```

```
info := &grpc.UnaryServerInfo{
    Server:     srv,
    FullMethod: "/common.Product/Get",
}
```

```
.
├── rpc
│   └── common
│       └── product.pb.go
└── service
    └── product
        └── product.go
```

微服务选型: SDK vs RPC

- 一个服务提供的内容不仅限于接口



微服务选型: gRPC - 项目改造

- 使用 internal 来隔离资源/函数

```
syntax = "proto3";
package common;

service Product {
    rpc Get(ProductGet) returns (ProductGetResp);
    rpc Update(ProductUpdate) returns (ProductUpdateResp);
    rpc Search(ProductSearch) returns (ProductSearchResp);
}
```

```
service
└── product
    ├── internal
    │   ├── model
    │   │   └── gen_product_mongo.go
    │   ├── product_get
    │   │   └── product_get.go
    │   ├── product_search
    │   │   └── product_search.go
    │   └── product_update
    │       └── product_update.go
    └── product.go
```

微服务选型: gRPC

```
import (
    "ezbuy.com/ezbuy/common/rpc/common"
)

func GetProduct(ctx context.Context) {
    common.GetProduct().Get(ctx, ...)
}
```

微服务选型: gRPC

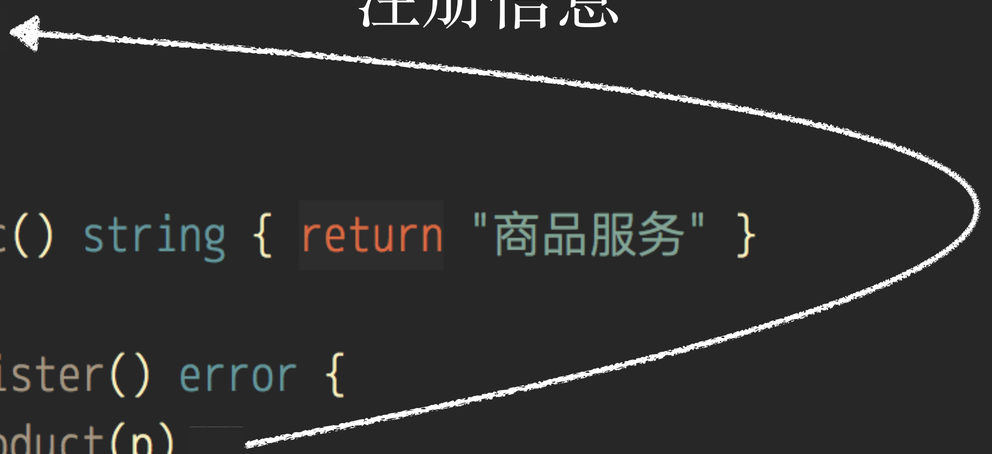
```
func NewClientEx(name string, cfg *ClientConfig) (*Client, error) {
    opts := []grpc.DialOption{
        grpc.WithInsecure(),
        grpc.WithUserAgent(stack.GetServiceName()),
        grpc.WithUnaryInterceptor(clientInterceptor),
        grpc.WithBalancer(RoundRobin(NewResolver(mconsul.DefaultDatacenter))),
    }
    grpcClient, err := grpc.Dial(name, opts...)
    if err != nil {
        return nil, errors.TraceWithField(err, "dial", name)
    }
    return &Client{grpcClient}, nil
}
```

- Consul
- 程序内维护地址列表
- polling 获取更新

```go
func (r *Resolver) watchLoop(addrs []string) error {
    wp, err := watch.Parse(map[string]interface{}{
        "type":       "service",
        "service":    r.name,
        "datacenter": r.dc.Name,
    })
    if err != nil {
        return errors.Trace(err)
    }
    wp.Handler = func(idx uint64, obj interface{}) {
        addrs = r.onUpdate(addrs, obj.([]*api.ServiceEntry))
    }
    if err := wp.Run(r.dc.ConsulAddr()); err != nil {
        return errors.Trace(err)
    }
    return nil
}
```

微服务选型: gRPC - 项目改造

```
type Service struct {
    mservice.Server
}

func (p *Service) Desc() string { return "商品服务" }

func (p *Service) Register() error {
    common.RegisterProduct(p)
    return nil
}

func (p *Service) Get(ctx context.T, req *common.ProductGet) (*common.ProductGetResp, error) {
    return product_get.Get(ctx, req)
}
```



微服务选型: 问题

- 阅读体验... 差
- 调用远程服务的思考成本
- 调用栈无法真实还原

微服务选型: gRPC + Webapi

```
type ProductWebapi struct{ s ProductWebApiRegister }
func RegisterProductWebapiEx(s ProductWebApiRegister) {
    wrap := &ProductWebapi{s}
    s.WebApiRegisterMethod("common.Product", "Get", wrap.Get)
    s.WebApiRegisterMethod("common.Product", "Update", wrap.Update)
    s.WebApiRegisterMethod("common.Product", "Delete", wrap.Delete)
}
func (s *ProductWebapi) Get(ctx *context.T, w http.ResponseWriter, req *http.Request) {
    params := new(ProductGet)
    if err := s.s.WebApiDecode(ctx, req, params); err != nil {
        s.s.WebApiHandleResp(ctx, w, nil, err)
        return
    }
    resp, err := s.s.Get(*ctx, params)
    s.s.WebApiHandleResp(ctx, w, resp, err)
}
```

Webapi生成代码

微服务选型: gRPC + Webapi

- consul-template 自动注册路由
- 找出支持webapi的接口
- 共享一个端口 （github.com/cockroachdb/cmux）

```
mux := cmux.New(ln)
if s.GrpcServer.isGrpcEnable() {
    match := cmux.HTTP2HeaderField("content-type", "application/grpc")
    ln := mux.Match(match)
    go s.GrpcServer.Serve(ln)
}
ln := mux.Match(cmux.Any())
go s.WebServer.Serve(ctx, ln)
```

微服务选型: 总结

- 使用 gRPC 作为微服务框架，与服务发现深度结合
- 让 远程调用 == 本地调用 （形式上）
- 接口路径 == 代码路径
- 通过option定义接口特性
- 用 goflow 解决依赖的工具链(consul)

分布式追踪：调用栈

- sentry，调用栈展示

Exception (most recent call first) **Full** Raw

/ .Lang/Translate:lang_translate.bingRunLoop

```
result not match
```

/srv/deploy/goflow/src/	/ezbuy/	/service/lang/lang.go in Translate at line 49
/srv/deploy/goflow/src/	/ezbuy/	/service/lang/internal/lang_translate/translate.go in Translate at line 300
/srv/deploy/goflow/src/	/ezbuy/	/service/lang/internal/lang_translate/translate.go in bingRunLoop at line 172
/srv/deploy/goflow/src/	/ezbuy/	/service/lang/internal/lang_translate/translate.go in bingRunLoop at line 169

分布式追踪：调用栈

- sentry, 记录的一些上下文信息值

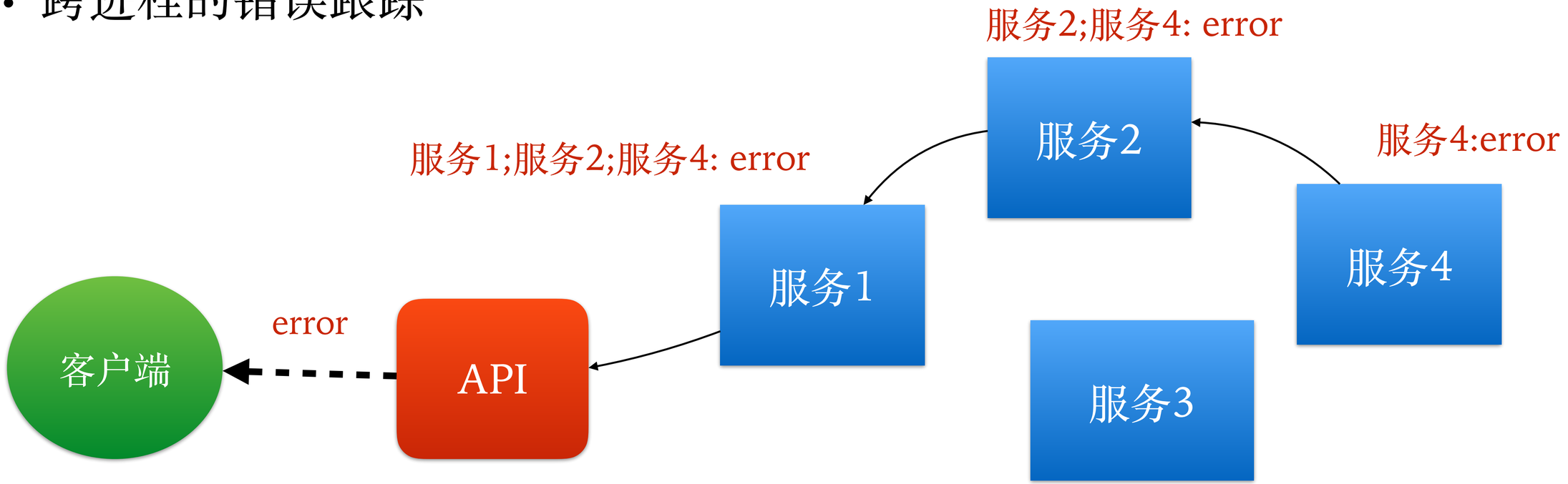
Additional Data

field.entity

```
[
  2*0.6*2蓝300kg,
  1.5*0.5*2灰100kg,
  1.5*0.4*2蓝100kg,
  1.5*0.4*2灰100kg,
  1.2*0.5*2蓝100kg,
  1.2*0.5*2灰100kg,
  1.2*0.4*2蓝100kg,
  1.2*0.4*2灰100kg,
  2*0.4*2蓝200kg,
  2*0.4*2灰200kg,
  2*0.5*2蓝200kg
]
```

分布式追踪：调用栈

- 跨进程的错误跟踪



分布式追踪：调用栈

- 跨进程的错误跟踪

/srv/deploy/goflow/src/	/ezbuy/	/service/oracle/oracle.go in GetProduct at line 62	
... w/src/	/ezbuy/	/service/oracle/internal/oracle_getproduct/oracle_getproduct.go in GetProduct at line 676	
... w/src/	/ezbuy/	/service/oracle/internal/oracle_getproduct/oracle_getproduct.go in getTProduct at line 728	
/srv/deploy/goflow/src/	/ezbuy/	/service/product/product.go in Get at line 95	
... eploy/goflow/src/	/ezbuy/	/service/product/internal/product_get/product_get.go in Get at line 42	

分布式追踪：Context

- Go1.7 加入标准库
- 作用于整个函数调用链
- 请求间的信息传递
  - 通过Header

分布式追踪：调用栈

- Go本身没有Exception支持

```
func OpenFile(fp string) (*File, error) {
    fd, err := os.Open(fp)
    if err != nil {
        return nil, err
    }
    return fd, nil
}
```

分布式追踪：调用栈

- 常见做法

```
func ReadFile(f *os.File) ([]byte, error) {
    body, err := ioutil.ReadAll(f)
    if err != nil {
        return nil, fmt.Errorf("ReadFile: %v", err)
    }
    return body, nil
}
```

分布式追踪：调用栈

- 懒人做法

```go
func ReadFile(f *os.File) ([]byte, error) {
    body, err := ioutil.ReadAll(f)
    if err != nil {
        return nil, errors.Trace(err)
    }
    return body, nil
}
```

```go
type Error struct{
    error
    stack []string
}

func Trace(err error) error {
    return &Error{err, getStackInfo()}
}

func Cause(err error) error {
    return err.(Error).error
}
```

分布式追踪：调用栈

- 有更进一步的做法吗?

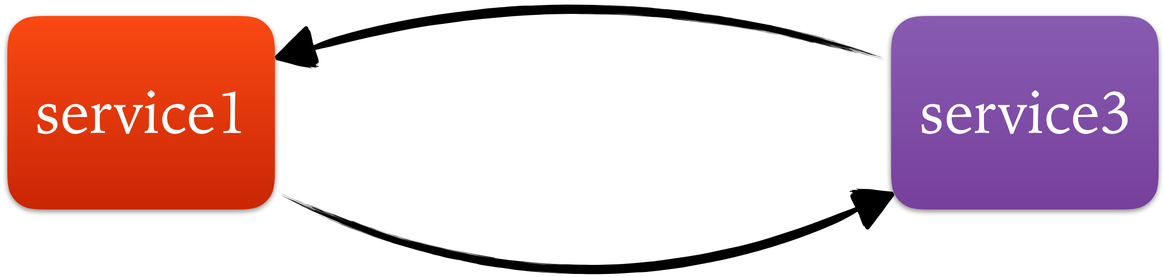
分布式追踪：调用栈

```
func (p *Service) Get(ctx context.T, req *common.ProductGet) (*common.ProductGetResp, error) {
    ctx.SetField("productId", req.Id)
    resp, err := product_get.Get(ctx, req.Id)
    if err != nil {
        return nil, ctx.Trace(err)
    }
    return resp, nil
}
```

```
 ERRO[0023] product not found                                method=/common.Product/Get productId
 =012345 stack=product.(*Service).Get:95;
```

分布式追踪：死循环?

- A 和 B 服务相互调用，滚雪球
  - Context + TTL



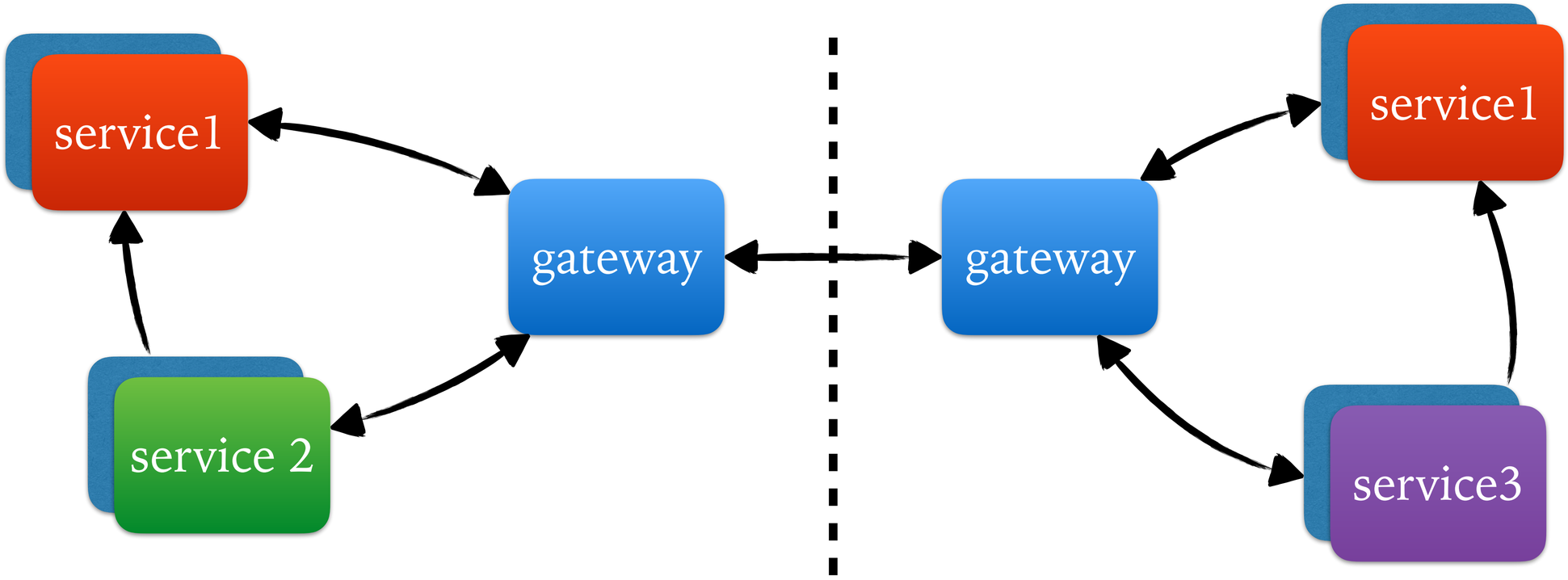
分布式追踪：总结

- 完善错误调用栈和日志打印
- 直接聚合分布式服务跟踪
- 使用 sentry 储存错误日志
- 使用 context + 结构化日志 + Trace 来打印日志

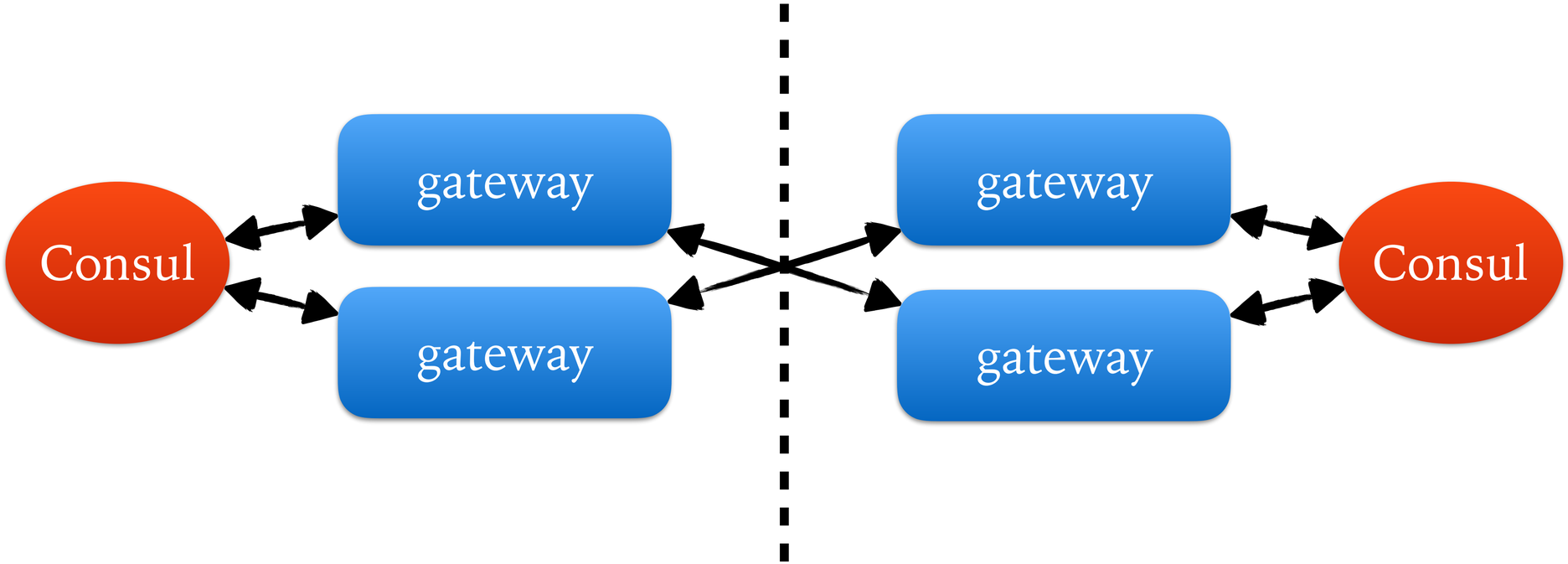
跨数据中心：Gateway

- 数据中心差异化，跨数据中心通讯
- 要求
  - 完全透明
  - 连接内外网
  - 高可用

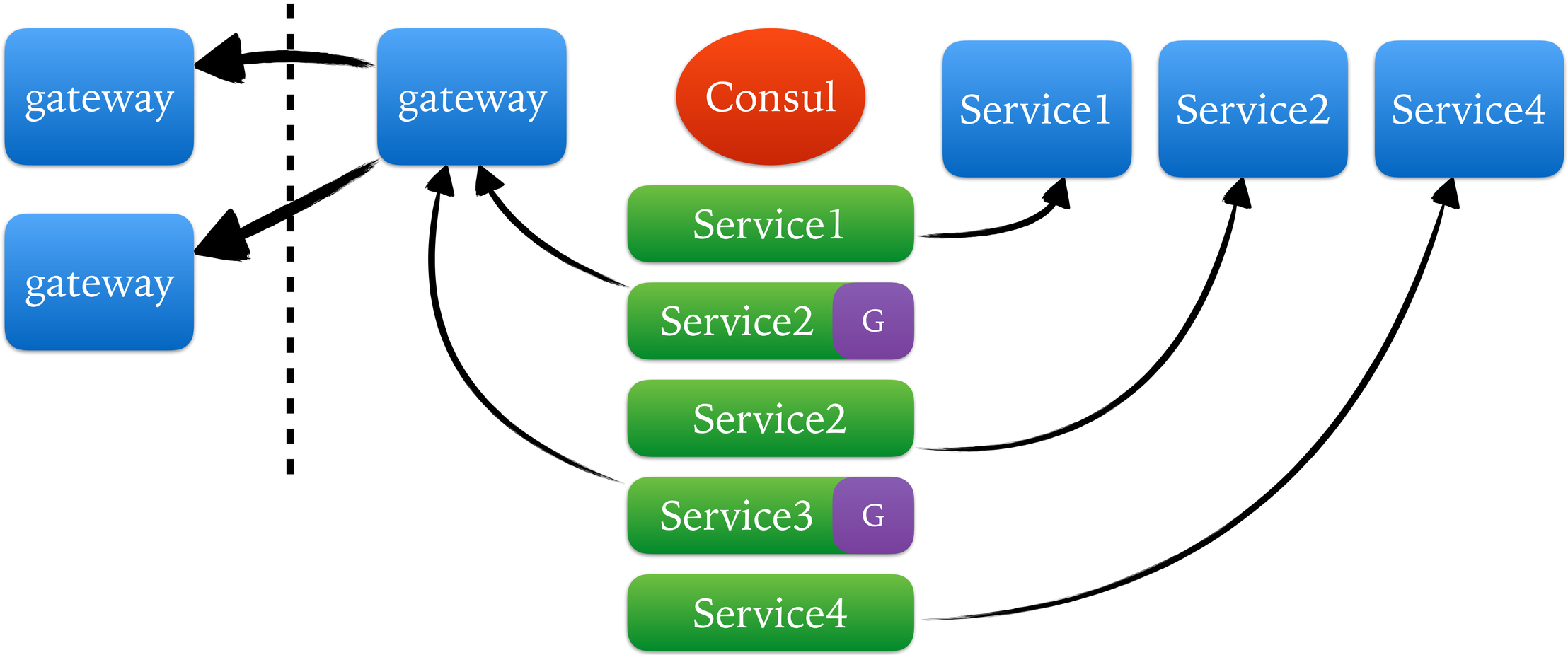
跨数据中心：Gateway



跨数据中心：Gateway - 解决方案



跨数据中心：Gateway - 解决方案



跨数据中心：Gateway - 总结

- Gateway 自己也是个 gRPC 服务
- Gateway 直接的相互感知
- consul 不对外暴露，确保可控
- 客户端完全透明

总结

- 个人开发环境尤其重要
  - 将各种公司内部特殊流程标准化自动化
- 统一使用微服务框架
- 轻量级解决错误分布式跟踪的问题
- gateway 代理 gRPC 请求连接内外网

Q&A