稀土掘金 CloudWeGo

飞书管理后台平台化

从单体到网关的演进

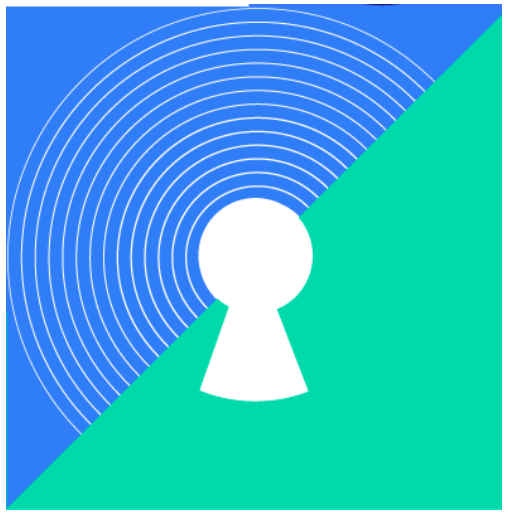
汪浩 飞书业务中台

CloudWeGo | 稀土掘金 出品

2022/06/25

目录

CONTENT



架构和挑战

飞书管理后台单体架构面临的各种挑战

飞书是什么?

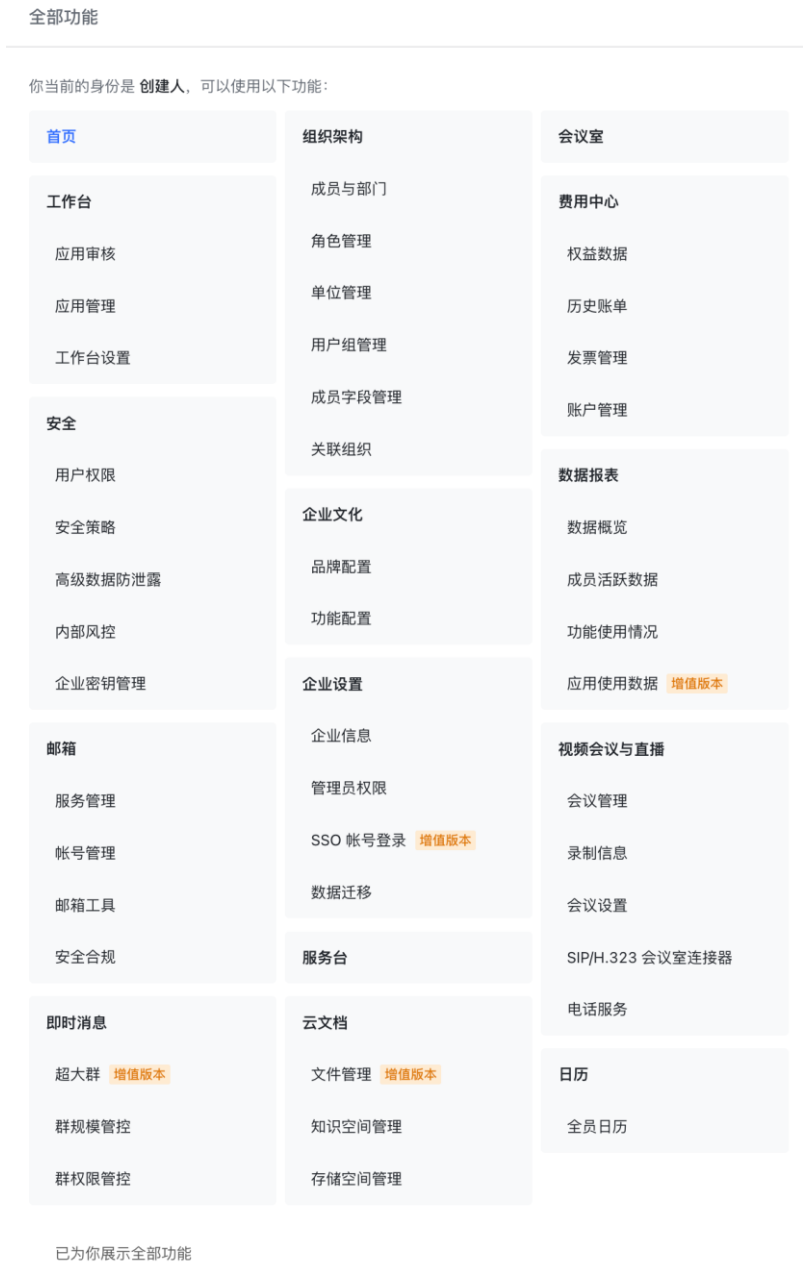
飞书是真正的一站式企业沟通与协作平台，整合视频会议、即时消息、日历、云文档、邮箱、工作台等功能于一体，立志打造高效的办公方式，加速企业成长。

飞书管理后台

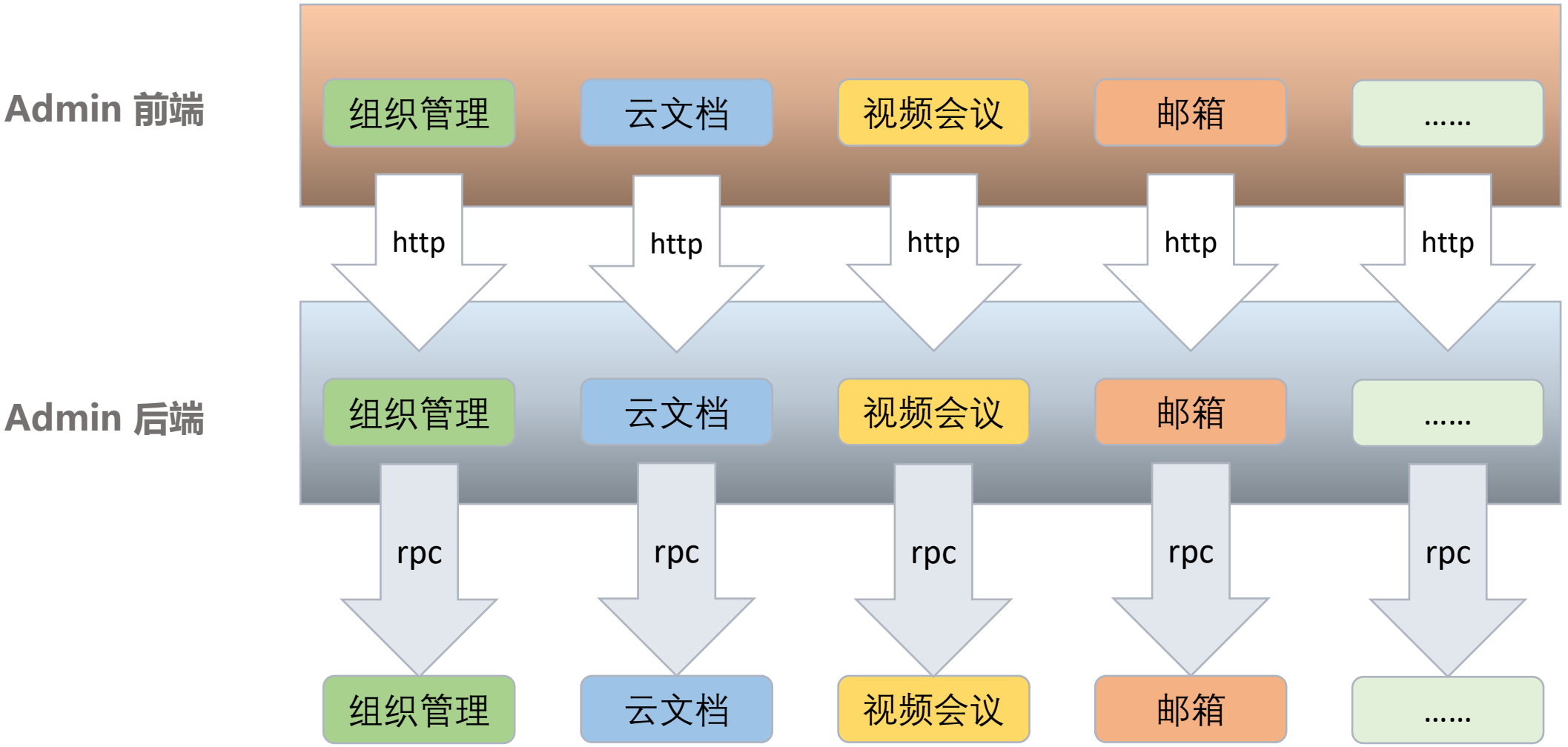
飞书管理后台（以下简称Admin）是飞书套件专为企业管理员提供的信息管理平台，企业管理员可通过后台管理企业设置、组织架构、工作台和会议室等功能。

平台化改造背景

飞书采用的是all-in-one的套件模式，Admin作为整个套件统一的管理后台，承接了包括组织管理、云文档、视频会议、邮箱、开放平台等10多个业务线的管控需求。一直以来的开发模式是各业务方直接在Admin的代码仓库提交代码或者由Admin团队负责web层逻辑的开发。



Admin架构


Admin 前端
组织管理
云文档
视频会议
邮箱
.....
http
http
http
http
http
Admin 后端
组织管理
云文档
视频会议
邮箱
.....
rpc
rpc
rpc
rpc
rpc
组织管理
云文档
视频会议
邮箱
.....

面临的问题

业务迭代慢

Admin的研发资源被过多的耗在各业务的迭代中，无法快速支持自身的业务规划，如组织架构、安全、KA等；发版节奏不一致，研发资源不匹配，导致Admin会成为业务迭代的瓶颈。

研发效率低

由于需要了解各业务的上下文，需要花费大量时间沟通。联调、oncall的链路也很长，双方的责任也划不清楚，导致整体的研发效率偏低。

工程质量差

多个团队共同维护一个代码仓库，代码质量参差不齐，设计规范也各不相同，底层的代码的修改还会相互影响，造成线上问题。

面临的挑战

多环境互通与隔离

飞书不仅有SaaS，还有众多私有化部署环境。我们需要解决不同环境的网络隔离、版本异构问题，还需要灵活支持不同环境运行不同版的配置。

接入业务复杂性

Admin需要集成十几个业务线，接入诉求不统一。接口协议包含http协议和thrift协议，还有各种自定义插件需求和权限校验需求。

安全保障

Admin作为飞书套件的管理配置中心，关系到整个企业的数据安全。为了保障Admin的接口数据安全，需要提供鉴权中间件、管理员权限验证、参数校验、风控、频控等功能，提升业务方的安全能力。

平台化构想

飞书管理后台平台化

构 想 和 架 构 升 级

目标

我们要做什么

通过提供一套统一的标准和通用服务，让有管控诉求的套件业务方能快速实现能力集成，并且能给客户带来一致的体验，最终实现Admin作为企业统一数字化管理平台的愿景。

技术上需要达到的效果

业务方的后端接口和前端页面动态接入，Admin无需代码改造和服务发布即可无缝上线，Admin从单体应用进化为业务网关，是包含UI交互在内的独立产品模块的集成。

我们不要做什么

我们不是要做一个搭建系统。目前很多平台型产品都提供low code / no code 的工具方便开发者快速搭建所需要的功能。但是目前通过我们对客户诉求的调研，没有相关的需求（但是不代表未来也没有，这块我们会持续保持关注）。我们需要做的是制定相关的标准，比如UI、交互、API等，业务按照标准去实现。

我们也不是要做一个API Gateway 或者Service Mesh。API Gateway 的核心是「Exposes your services as managed APIs」，将内部的服务以更加可控可管理的方式暴露出去，可以认为是后端服务的一个代理。Service Mesh 可以看成是API Gateway的去中心化实现方式，用来解决单点、隔离、耦合等问题。我们需要解决的不仅仅是服务路由、协议转换、安全管控等问题，而是包含UI交互在内的独立产品模块的集成。

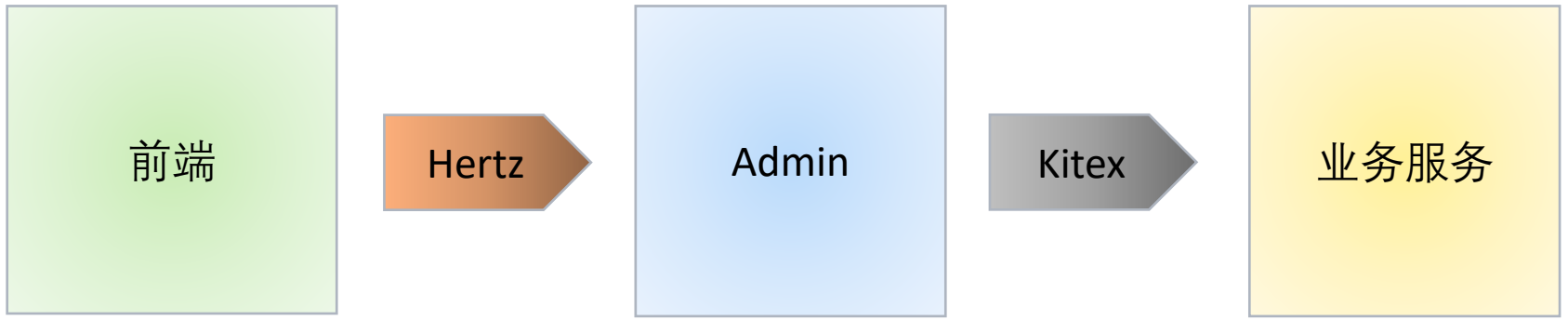
旧的架构

前端架构

前后端分离，node单体项目。

后端架构

后端(Golang实现)采用Hertz框架对前端暴露HTTP接口，Handler层通过Kitex调用依赖的各个业务线的微服务。



框架介绍

Hertz

Hertz[həːts] 是一个 Golang 微服务 HTTP 框架，在设计之初参考了其他开源框架 fasthttp、gin、echo 的优势， 并结合字节跳动内部的需求，使其具有高易用性、高性能、高扩展性等特点，目前在字节跳动内部已广泛使用。 如今越来越多的微服务选择使用 Golang，如果对微服务性能有要求，又希望框架能够充分满足内部的可定制化需求，Hertz 会是一个不错的选择。

Kitex

Kitex[kaɪt’ eks] 字节跳动内部的 Golang 微服务 RPC 框架，具有高性能、强可扩展的特点，在字节内部已广泛使用。如果对微服务性能有要求，又希望定制扩展融入自己的治理体系，Kitex 会是一个不错的选择。

新的架构

Gaia控制面

我们增加了Gaia平台(基于Hertz框架的web服务)来作为我们整个Admin的控制系统，负责整体的发布和管控需求，包括接口的生命周期管理、微应用生命周期管理、监控告警、业务线接入、多环境发布等。

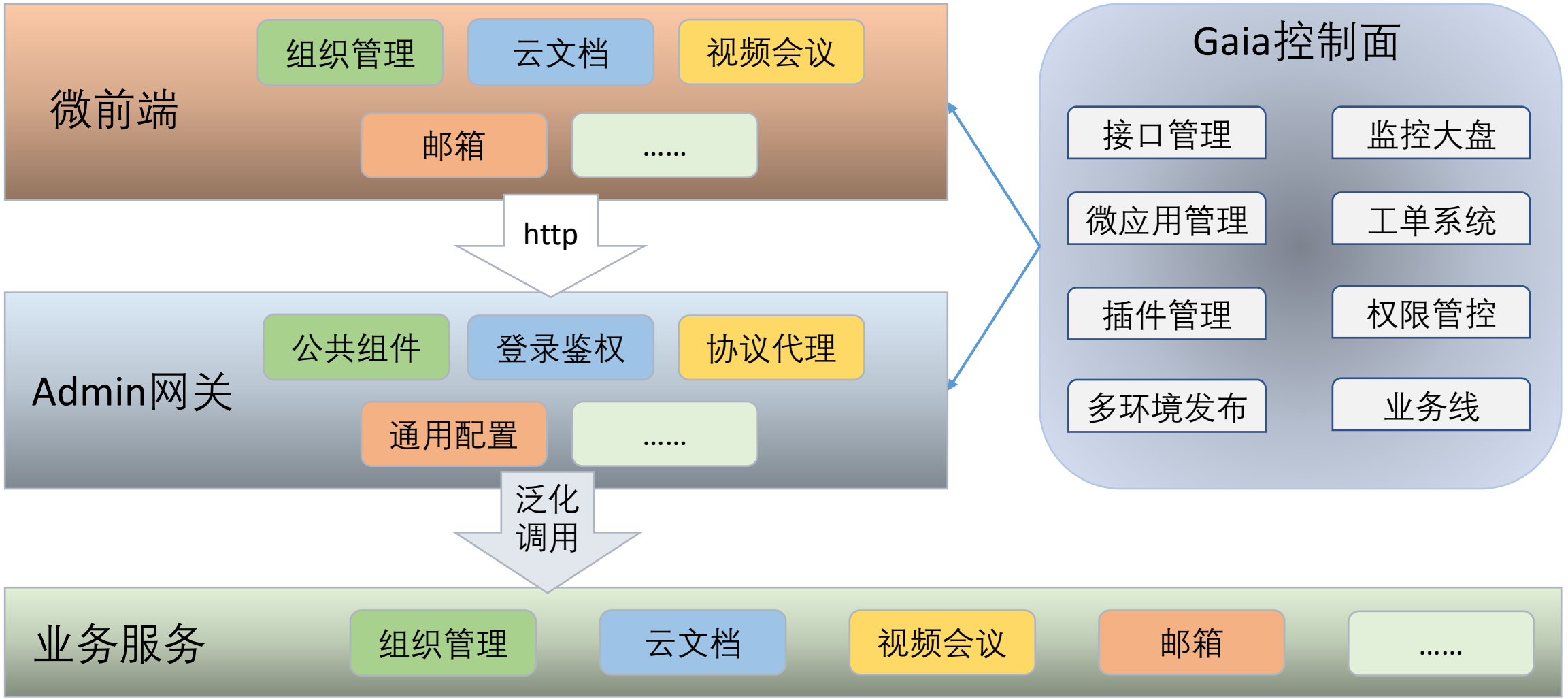
前端架构

前端采用微前端架构，各个业务方通过构建微应用接入Admin基座，使用统一封装好的组件库实现前端页面。

后端架构

后端使用字节通用BAM规范，通过泛化调用的方式打通Admin和各接入业务方服务，并抽象公共组件以插件的方式进行功能扩展。

Admin架构


微前端
组织管理
云文档
视频会议
邮箱
……
http
Admin网关
公共组件
登录鉴权
协议代理
通用配置
……
泛化
调用
业务服务
组织管理
云文档
视频会议
邮箱
……
Gaia控制面
接口管理
监控大盘
微应用管理
工单系统
插件管理
权限管控
多环境发布
业务线

Gaia平台功能

业务线管理

业务线：实现以业务为维度进行接入Admin而提出的概念。通过业务线来聚合业务为维度的所有资源，相关资源包括微应用、菜单、接口、监控等。

接口生命周期管理

包括接口创建、更新、编排、发布、上线、下线、删除等。同时维护接口IDL文件。

微应用生命周期管理

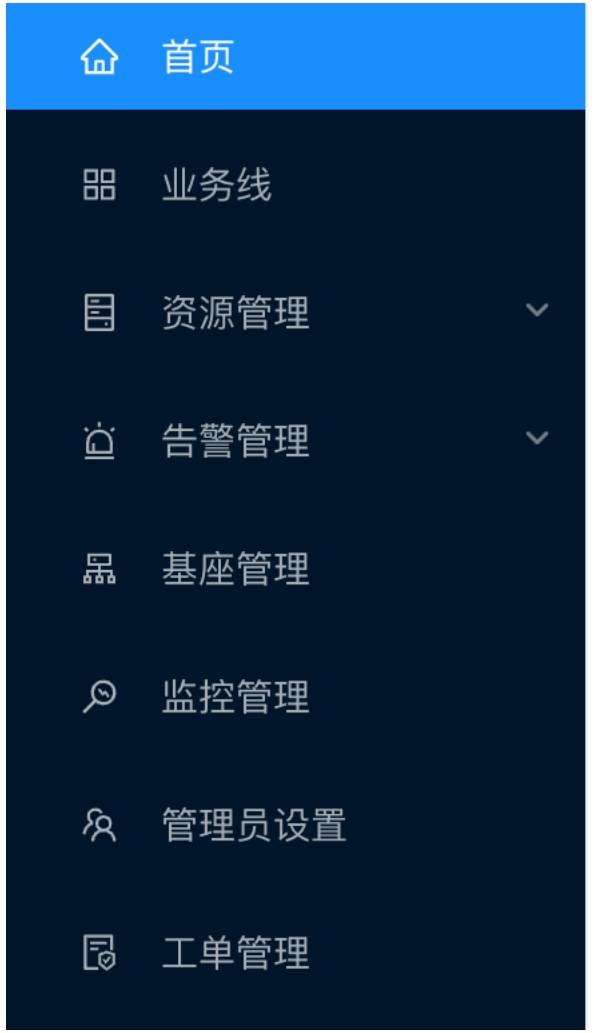
微应用的申请、接入、微应用版本创建、发布、下线等。

监控大盘

业务整体维度和单接口维度的SLA大盘，以及错误告警管理。

插件管理

默认插件和自定义插件的配置管理。



平台化实现

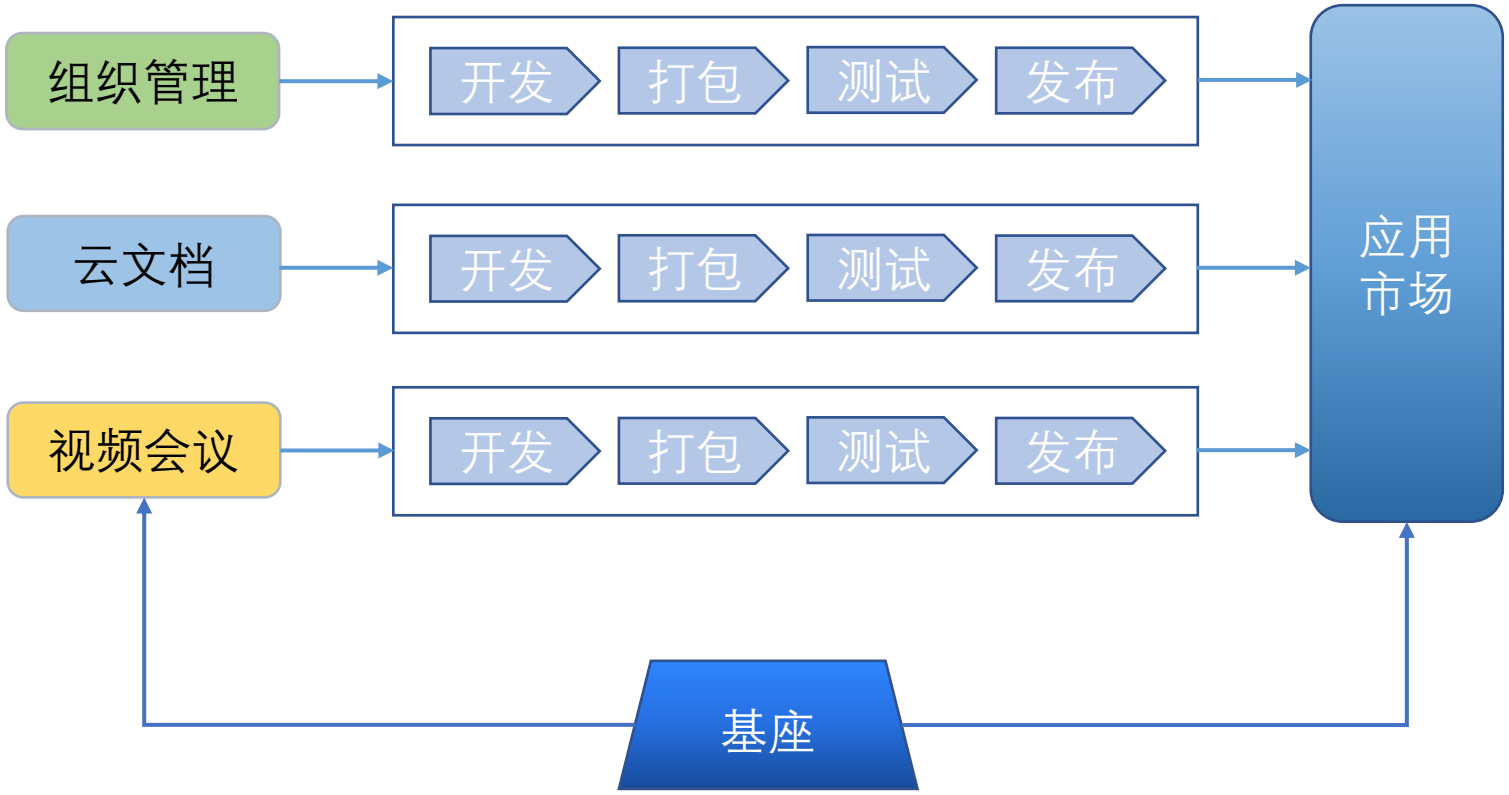
微前端技术架构、泛化

调用实践和功能扩展

微前端技术架构

- **基座** 是指微前端入口模块，负责组装各个模块
- **微应用** 就是指独立的业务模块
- **微应用市场** 负责管理微应用的创建，管理，版本发布等

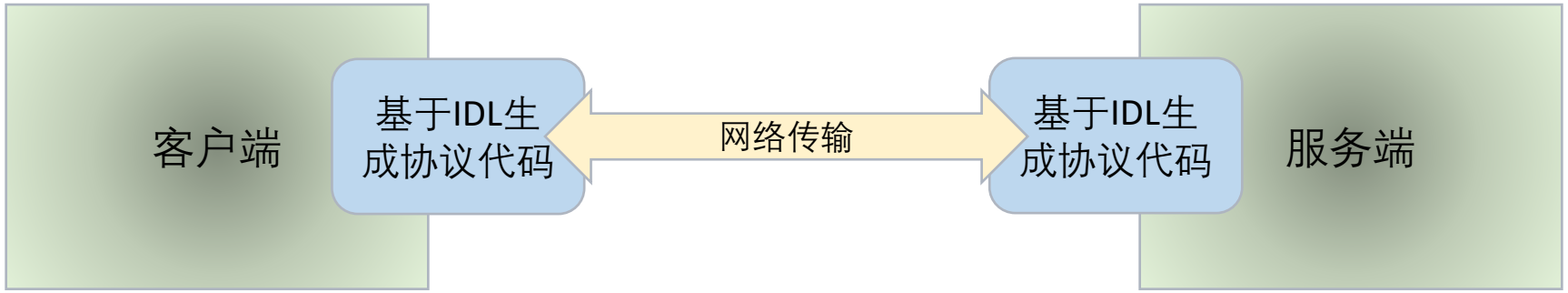
通过微应用市场下发的配置进行微应用组合，将基础能力下放到各个业务方。



泛化调用方案调研

- 为了解决之前的问题，一个比较容易想到的就是把Admin变成一个平台化的网关，不再维护业务逻辑，只处理通用逻辑。由于前后端通信使用json序列化，因此 **泛化调用** 是我们最终的选择。
- 网关与微服务之间使用gRPC、Thrift等协议进行通信，都是通过代码生成实现的协议解析，不能动态更新，都需要生成代码->重新发布。
- 内部Kite(Thrift协议)不支持泛化调用。
- Kitex字节跳动内部的 Golang 微服务 RPC 框架，具有高性能、强可扩展的特点，支持基于Thrift协议的泛化调用。

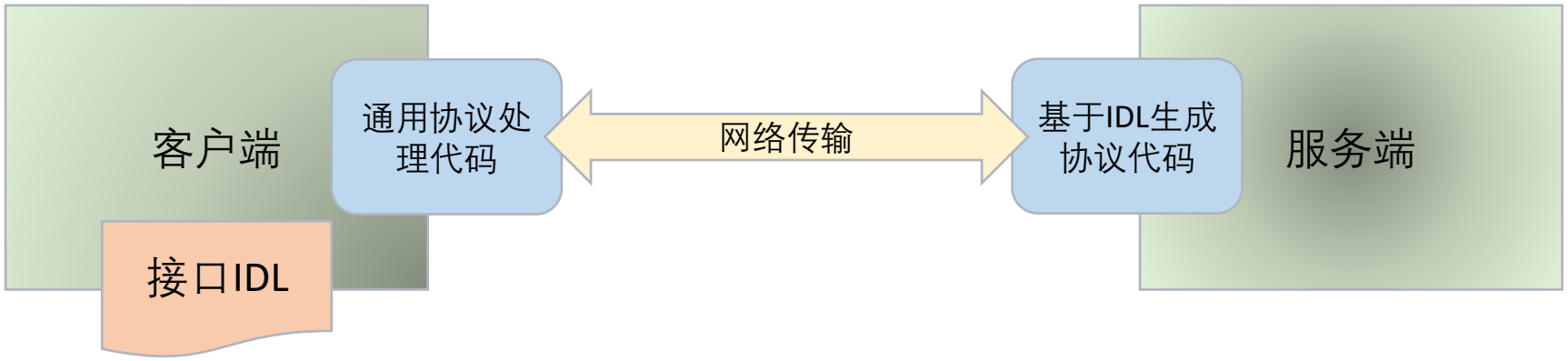
非泛化调用



非泛化调用，服务端和客户端都需要依赖IDL生成静态代码，接口的迭代意味着服务端和客户端都需要升级代码重新发布。

Admin场景下意味着其他业务方的业务迭代，需要我们引入代码依赖并发布服务

Kitex泛化调用



Kitex泛化调用中，服务端 **无需做任何改造**。客户端只有一份通用的协议处理代码，基于已有的IDL信息来动态生成协议字节流；IDL信息可以动态更新，以维护最新的接口协议。

https://github.com/cloudwego/kitex/tree/develop/pkg/generic/thrift

Admin场景下，网关作为客户端，动态维护业务方接口的IDL，通过泛化调用来实现HTTP接口到RPC接口的转换，**不再依赖业务服务客户端代码**。

HTTP协议映射

- Admin网关是基于Hertz对外暴露HTTP协议的接口，Hertz路由支持运行时新增，通过自定义Middleware和HandlerFunc可以实现接口运行时的增删改。

```
func initCLI() genericclient.Client {
  // IDL文件解析
 p, err := generic.NewThriftContentProvider("main", nil)
  if err != nil {
    panic(err)
  }
  // p.UpdateIDL("main", nil) IDL异步更新

  // 构造http类型的泛化调用
 g, err := generic.HTTPThriftGeneric(p)
  if err != nil {
    panic(err)
  }
  cli, err := genericclient.NewClient("destServiceName", g)
  if err != nil {
    panic(err)
  }
  return cli
}
```

HTTP协议映射

```
func initRouter(group *hertz.RouterGroup, cli genericclient.Client, handlerMethods map[string]string) {
  // 注册路由
 for httpMethod, path := range handlerMethods {
    group.Handle(httpMethod, path, func(c *hertz.RequestContext) {
      // hertz request转泛化request
      genericReq, err := generateGenericRequest(c.Request)
      if err != nil {
        c.AbortWithError(http.StatusInternalServerError, err)
        return
      }
      // 泛化调用
   resp, err := cli.GenericCall(getCtxWithBaseInfo(c), "", genericReq)
      if err != nil {
        c.AbortWithError(http.StatusInternalServerError, err)
        return
      }
      realResp, ok := resp.(*generic.HTTPResponse)
      if !ok {
        c.AbortWithStatus(http.StatusInternalServerError)
        return
      }
      // 写回response
      writeResponse(c, realResp)
    })
  }
}
```

功能扩展

自定义注解

Kitex内置了API注解来实现路由解析、参数传递等功能。我们通过自定义注解方式实现了框架未提供的功能，例如文件上传和下载、自定义参数注入、参数校验、自定义鉴权等

```
// HTTPMapping http mapping annotation
type HTTPMapping interface {
  // get value from request
  Request(ctx context.Context, req *HTTPRequest, field *FieldDescriptor) (interface{}, bool, error)
  // set value to response
  Response(ctx context.Context, resp *HTTPResponse, field *FieldDescriptor, val interface{}) error
}
// Route the route annotation
type Route interface {
  Method() string
  Path() string
  Function() *FunctionDescriptor
}
// ValueMapping value mapping annotation
type ValueMapping interface {
  Request(ctx context.Context, val interface{}, field *FieldDescriptor) (interface{}, error)
  Response(ctx context.Context, val interface{}, field *FieldDescriptor) (interface{}, error)
}
```

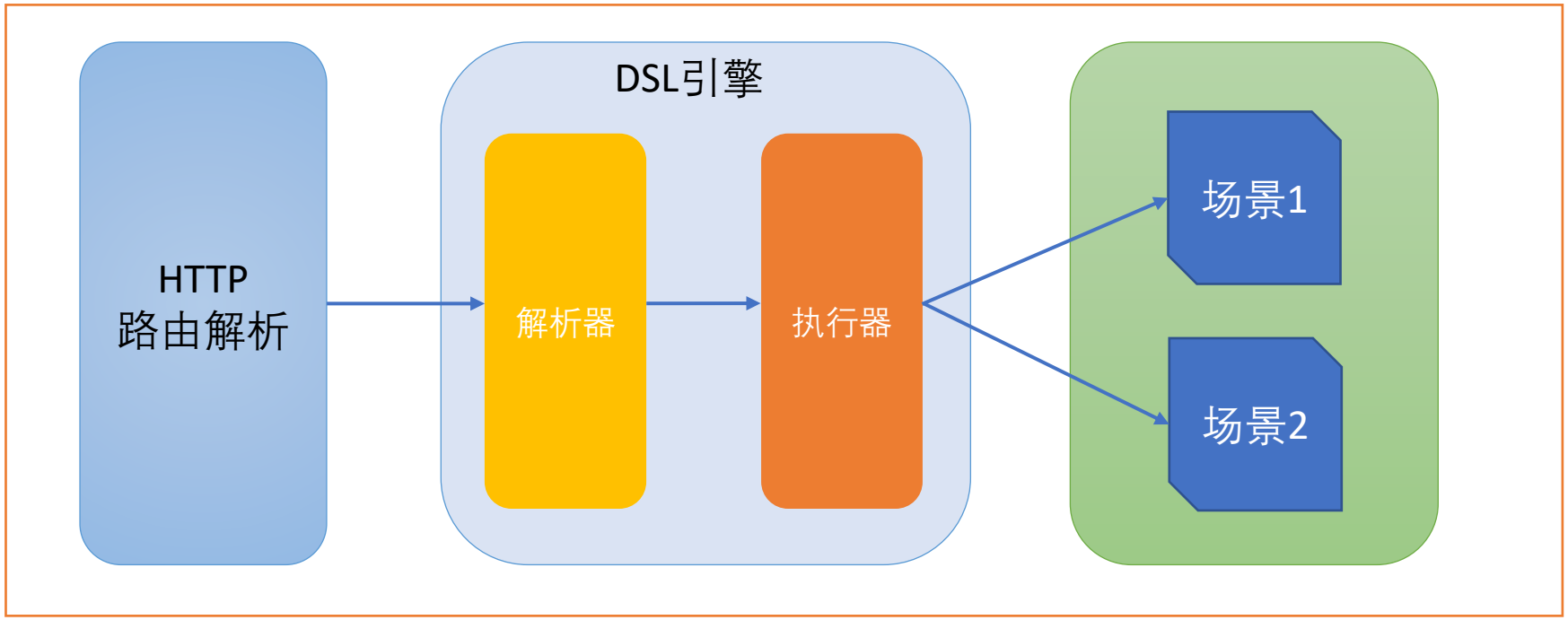
功能扩展

接口编排

已经实现的单接口泛化调用，不能完全满足我们一些复杂场景的使用需求，例如：

- 简单组装两个接口的结果
- 接口有顺序依赖，一个接口结果是其他接口的参数

我们通过自定义DSL引擎来对简单接口进行编排来实现一些复杂场景的接口调用需求。



成果

业务迭代加速

Admin不再关注其他业务线的需求，更加专注于自身的迭代需求。各个业务方发布完全隔离，使得他们不再依赖Admin，加快了Admin整体的业务功能迭代速度。

研发效率提升

丰富的前后端组件和简单的接入方式，使得业务方接入更加便捷，研发效率大大提升。

工程质量提高

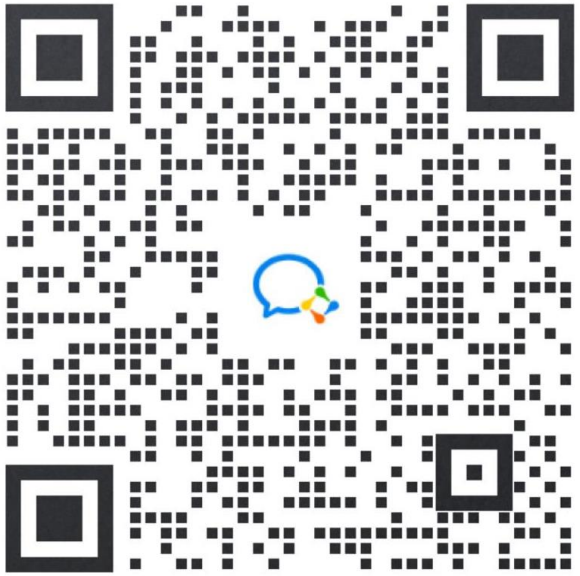
其他团队不再向Admin仓库提交代码，仓库代码风格趋向统一；去除了大量的业务逻辑，聚焦网关通用逻辑，提高了单测覆盖率；bug率显著下降，服务SLA明显提升。

未来规划

- ✓ 开放更多的组件，让接入的业务方聚焦在业务逻辑本身。包括消息中心、任务管理、安全风控、短信邮件等
- ✓ 完善服务治理和运维能力，包括灰度、降级、限流、精细化大盘等
- ✓ 建设通用的静态页面托管解决方案，为开发者提供便捷、稳定、高扩展性的静态页面托管服务
- ✓ 对接集成测试平台，闭环路由管理生命周期，保障接口稳定性和安全性

加入我们

扫描二维码加入飞书



扫描二维码加入群聊

CloudWeGo Day #1

CloudWeGo

从开源、开放到企业落地

2022 / 6 / 25 / 14:00

线上直播

联合主办：CloudWeGo 稀土掘金

合作伙伴：ByteTech InfoQ OSCHINA 51CTO

扫码看直播

THANKS