2019

企业级Node及Serverless应用实践

金炳

网易-严选事业部

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

Geekbang 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

个人介绍

Name: 金炳
Web developer @yanxuan
Github: stone-jin
知乎：金炳

Doing

目前致力于Node应用框架开发及生态建设，
实践Node应用在Serverless、FaaS场景下的迁移和落地，
探索service mesh在Node应用中的价值。

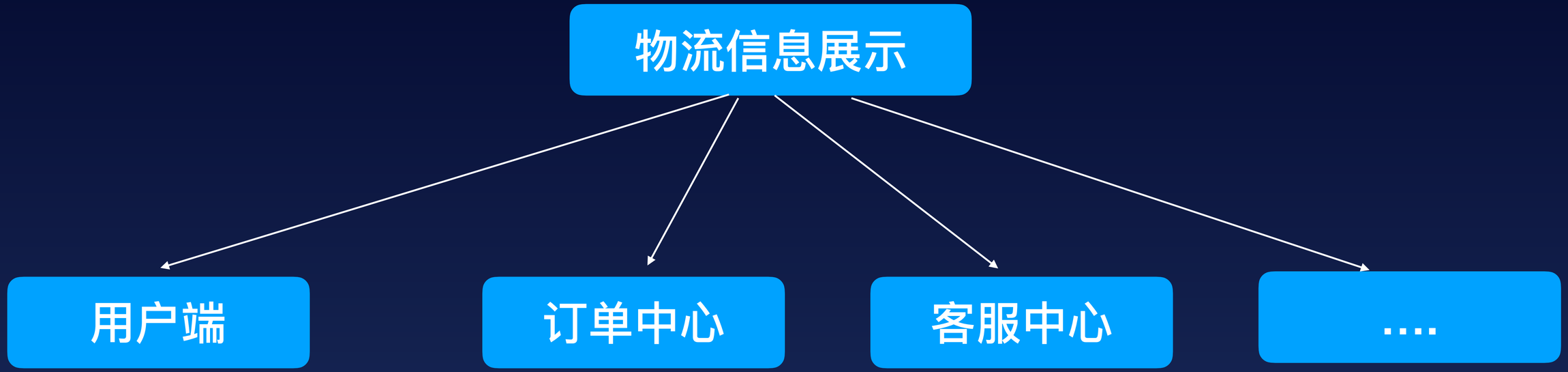
目录

一、业务场景

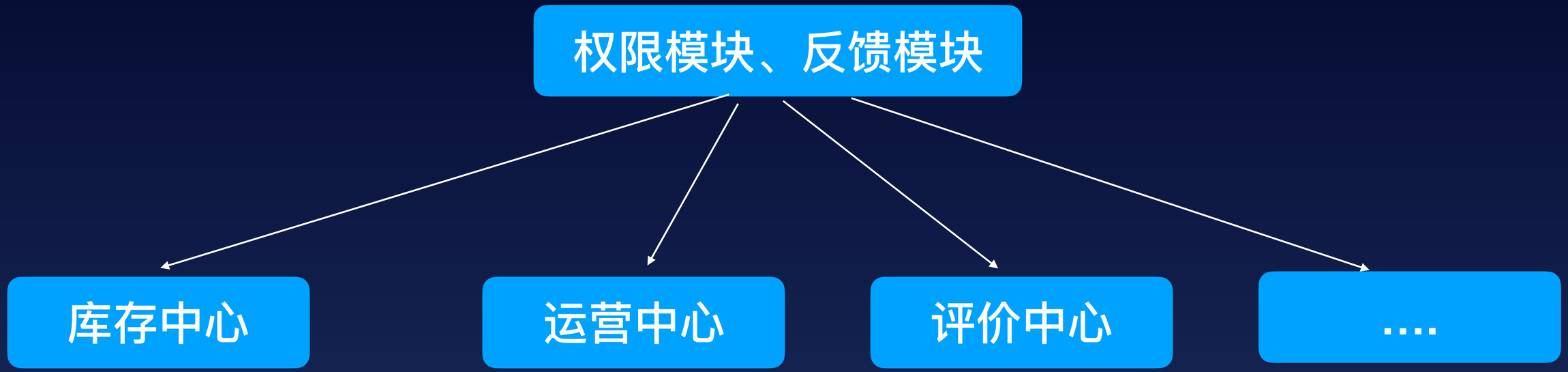
网易严选
网易旗下原创生活类自营电商品牌

产品架构（简化版）


电商前台业务
APP/H5商城
线下店
小程序/PC
渠道平台
....
支撑
电商后台业务
商品中心
订单中心
库存中心
会员中心
促销中心
CRM系统
评价中心
采购中心
WMS
物流中心
风控中心
客服系统
财务管理
营销中心
数据中心
支付中心



关联性强



关联性强

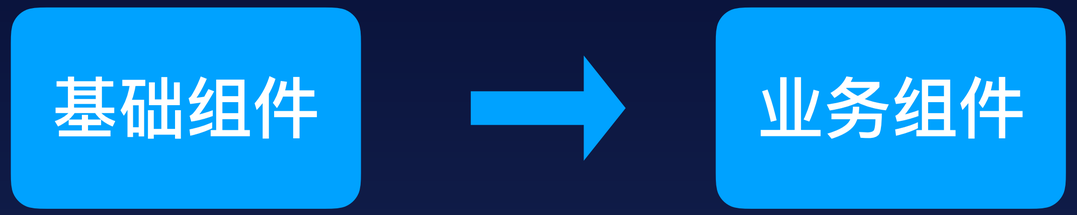
如何处理跨系统 关联性强 的业务场景？？

前端

基础组件

Node

前端



Node

业务组件-举个例子

多系统通用，与业务紧密关联的组件

二、Node应用

前端

基础组件 → 业务组件

Node

业务组件Node

前端业务组件：

```
<yx-logistics-trace [orderId]="orderId"></yx-logistics-trace>
```

Node承载数据服务：

```
import { LogisticsModule } from '@yx-module/logistics-node';

app.use(LogisticsModule.routes());
```

```
import { BaseModule, TgModule } from '@tiger/boot';
import { LogisticsController } from './logistics.controller';

@TgModule({
  imports: [],
  controllers: [LogicsController],
  middlewares: [],
  tasks: []
})
export class LogisticsModule extends BaseModule{
}
```

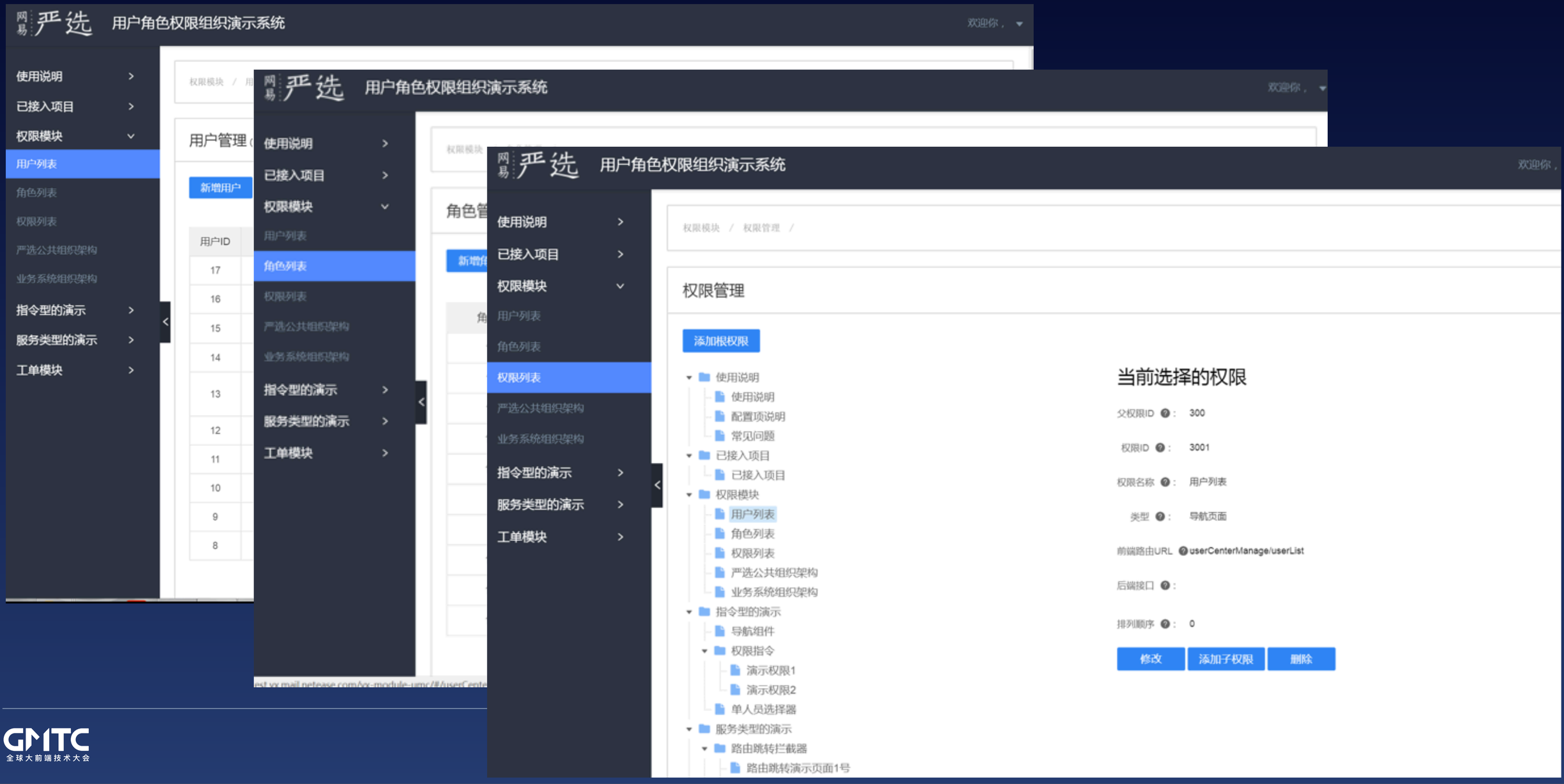
数据服务承载层

```
import { RestController, RequestMapping, GetMapping } from '@tiger/boot';
import { Context } from 'koa';
import { LogisticsService } from './logistics.service';
import { valid } from '@tiger/validator';
import { LogisticsVO } from './logistics.vo';

@RestController
@RequestMapping("/logistics")
export class LogisticController{
    constructor(private logisticsService: LogisticsService){
    }

    /**
     * 获取物流信息
     */
    @GetMapping("/getInfo.do", [valid(LogisticsVO)])
    async getInfo(ctx: Context, next: any){
        // 调用service层，对接后端服务
        ctx.body = await this.logisticsService.getInfo();
    }
}
```

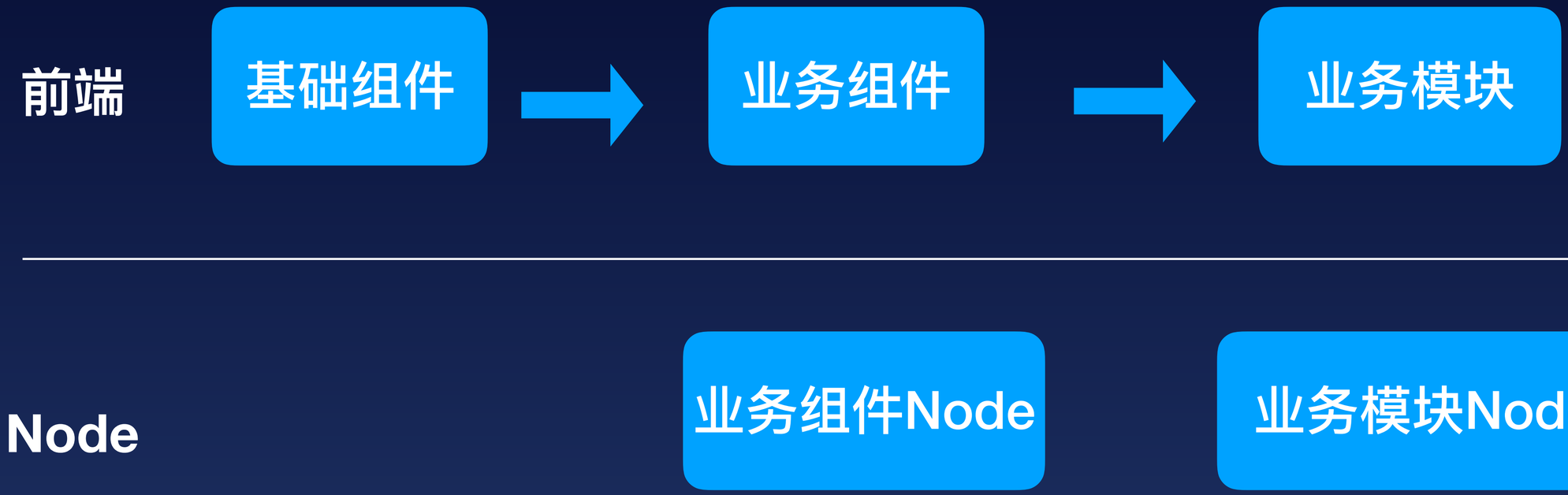
接口承载方

模块的复用?

业务模块-举个例子


网易严选
用户角色权限组织演示系统
欢迎你，
使用说明
已接入项目
权限模块
用户列表
角色列表
权限列表
严选公共组织架构
业务系统组织架构
指令型的演示
服务类型的演示
工单模块
权限模块 / 权限管理 /
权限管理
添加根权限
当前选择的权限
父权限ID： 300
权限ID： 3001
权限名称： 用户列表
类型： 导航页面
前端路由URL userCenterManage/userList
后端接口：
排列顺序： 0
修改
添加子权限
删除

业务模块：包含多个业务功能的集合。


前端
基础组件
业务组件
业务模块
Node
业务组件Node
业务模块Node

业务模块-使用

前端

```
import { UserCenterManageModule } from '@yx-module/user-center-manage';

@NgModule({
    imports: [
        ...
        UserCenterManageModule
        ...
    ]
})
export class AppModule {}
```

Node数据承载层:

```
import { UserCenterManageModule } from '@yx-module/user-center-manage-node';

app.use(UserCenterManageModule.routes());
```

```
import { TgModule, BaseModule } from '@tiger/boot';
import { UserModule } from './user/user.module';
import { RoleModule } from './role/role.module';
import { PermModule } from './perm/perm.module';
import { OrgModule } from './org/org.module';

@TgModule({
    prefix: '',
    imports: [UserModule, RoleModule, PermModule, OrgModule],
    controllers: [],
    middlewares: [],
    tasks: []
})
export class UserCenterManageModule extends BaseModule{
}
```

数据服务承载层

```
import { TgModule, BaseModule} from '@tiger/boot';
import { UserController } from './user.controller';

@TgModule({
  imports: [],
  controllers: [UserController],
  middlewares: [],
  tasks: []
})
export class UserModule extends BaseModule{
}
```

数据服务承载层
子模块

```ts
import { RestController, RequestMapping, GetMapping, PostMapping } from '@tiger/boot';
import { Context } from 'koa';
import { UserService } from './user.service';

@RestController
@RequestMapping("/user")
export class UserController{
  constructor(private userService: UserService){}

  // 查询用户列表
  @GetMapping("/list.do")
  async list(ctx: Context, next: any){}

  // 更新用户信息
  @PostMapping("/update.do")
  async update(ctx: Context, next: any){}

  // 新增用户信息
  @PostMapping("/add.do")
  async add(ctx: Context, next: any){}

  // 锁定用户
  @PostMapping("/lock.do")
  async lock(ctx: Context, next: any){}

  // 导出excel
  @GetMapping("/export.do")
  async export(ctx: Context, next: any){}
}
```

接口集

业务模块建设

多功能模块的集合

多页面的聚合

路由的集成

业务模块建设

多功能模块的集合

多页面的聚合

路由的集成

业务组件与业务模块的区别

点-面

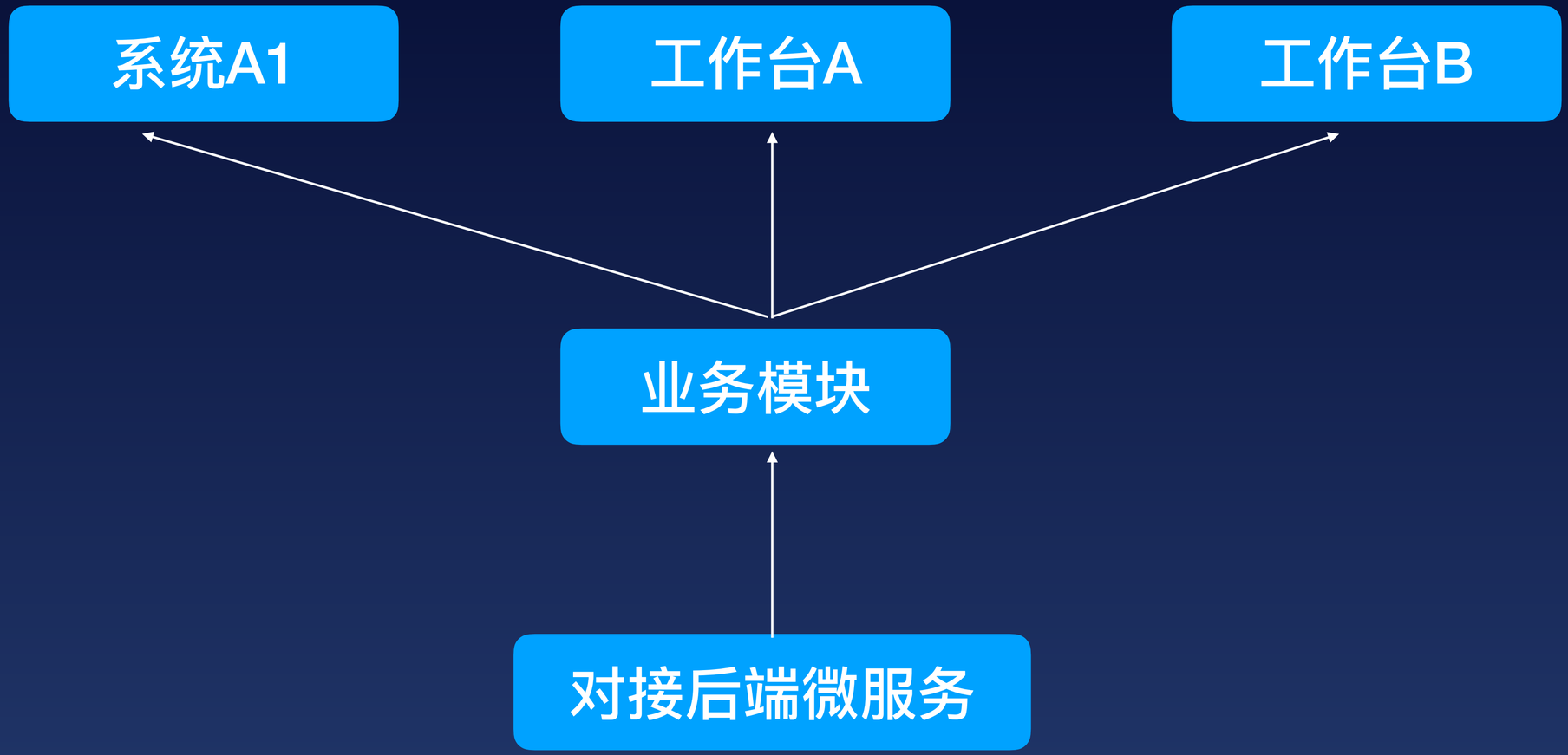
工作台?

```
import { UserCenterManageModule } from '@yx-module/user-center-manage';
import { FeedBackModule } from '@yx-module/feed-back';
import { GoodsManageModule } from '@yx-module/good-manage';

@NgModule({
  imports: [
    UserCenterManageModule, //权限模块
    FeedBackModule, // 反馈模块
    GoodsManageModule   //商品管理模块
    // ... 其他模块
  ]
})
class AppModule{
}
```

前端

```
import { TgModule, BaseModule } from '@tiger/boot';
import { UserCenterManageModule } from '@yx-module/user-center-manage-node';
import { FeedBackModule } from '@yx-module/feed-back-node';
import { GoodsManageModule } from '@yx-module/good-manage-node';

@TgModule({
  imports: [
    UserCenterManageModule, //权限模块
    FeedBackModule, // 反馈模块
    GoodsManageModule   //商品管理模块
    // ... 其他模块
  ]
})
class AppModule extends BaseModule{
}

app.use(AppModule.routes());
```

Node


系统A1
工作台A
工作台B
业务模块
对接后端微服务

Node的使用

业务组件　　业务模块　　工作台

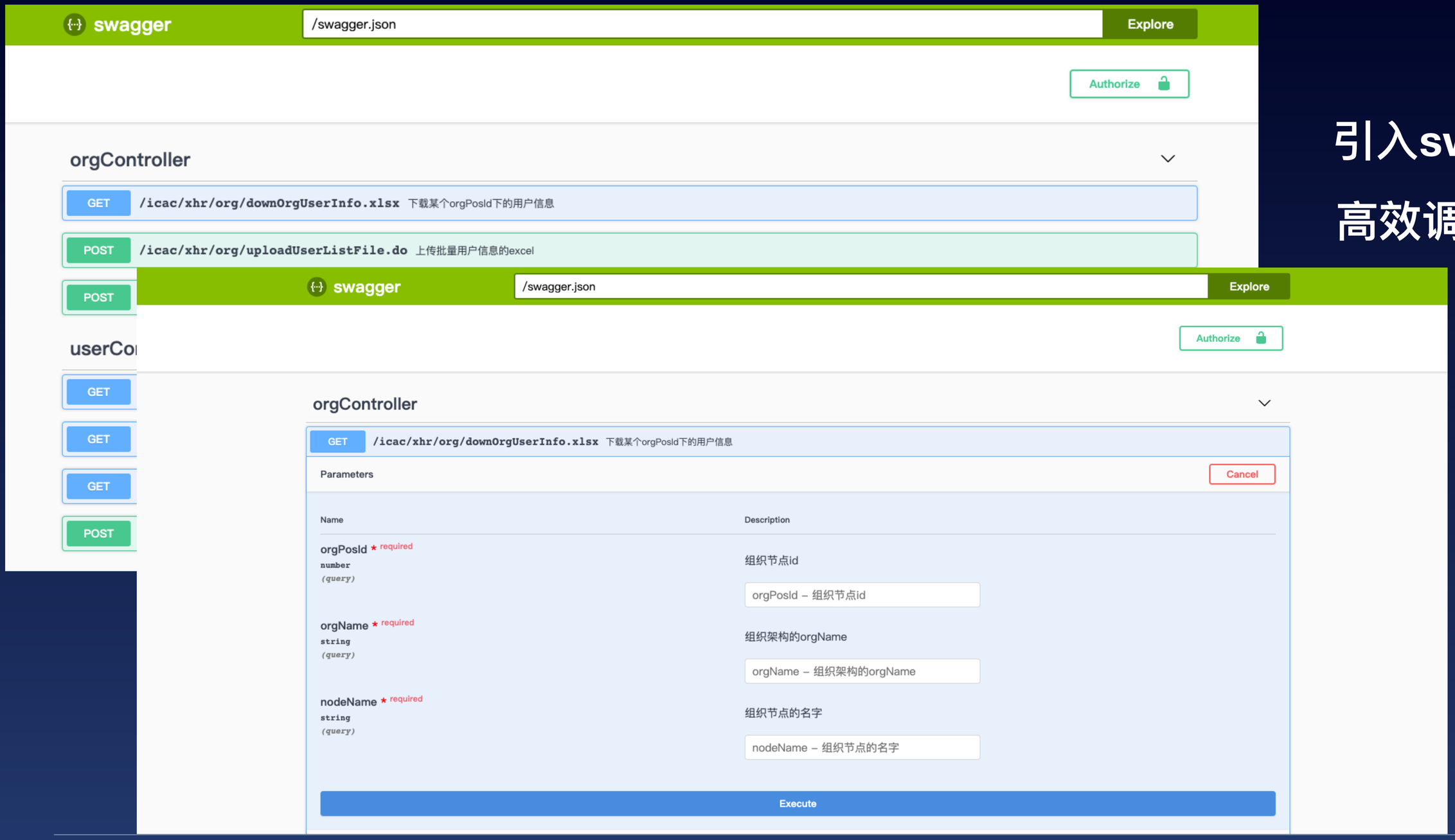
Node使用-角色转变

Node使用-角色转变

业务组件、业务模块

接口的使用方 → 接口的提供方

如何高效调试我们的接口？curl、 postman、swagger?



引入swagger

高效调试接口

传统swagger方式

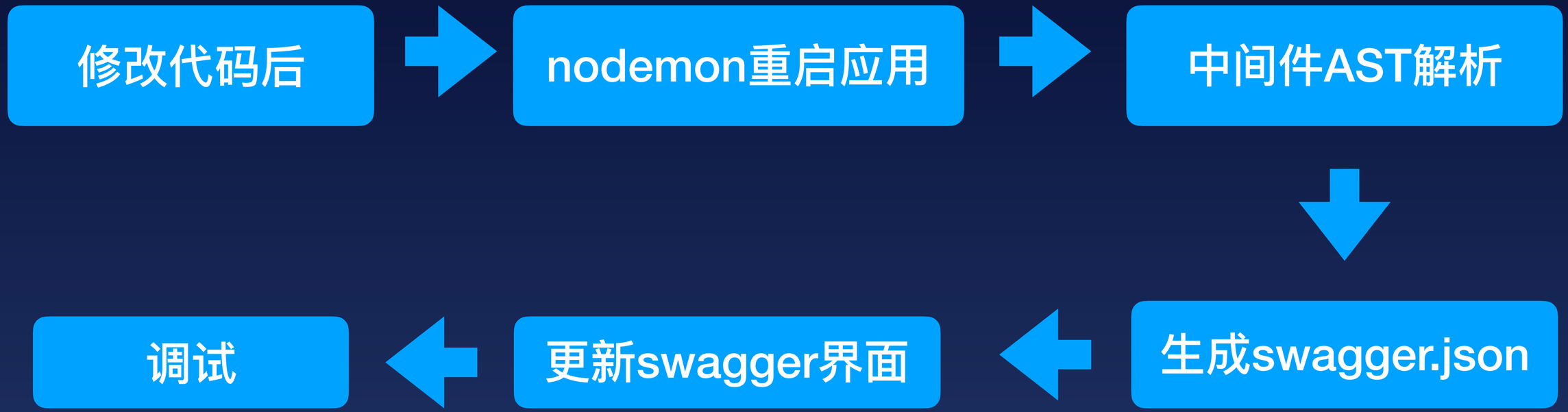
swagger中间件开发

解析ast

生成swagger.json

UI

Node使用-开发调试流程



Node使用-反哺前端

加速业务组件、业务模块、业务的前端开发

```
/**
 * 新增模板
 */
export function postTemplateUsingPosing(
    requestBodyVO: RequestBodyVO,
    cache: boolean | number = false
 ): Promise<ResultVO<String>>{
}
```

前端SDK
-typescript

如何进行权限拦截、参数校验?

权限校验

前端

```
<button type="button" *perm="'user_add'" (click)="addUser()">新增用户</button>
```

Node

```
// 新增用户
@PostMapping("/add.do", [perm("user_add")])
async add(ctx: Context, next: any){}
```

参数校验

```
@PostMapping("/add.do", [valid(UserVO)])
async add(ctx: RequestContext<UserVO, ResultVO>, next: any){ }
```

```
export class NameVO{
    // 用户名称
    @Validation(Type.string().max(10).min(5))
    name: string;
}
```

```
export class NameVO{
    // 用户名称
    @IsEmail()
    @MinLength(16)
    name: string;
}
```

单元测试？依赖注入？

IOC注入

```
@Service
export class UserService{
    getUserName(name: string){
        return name + ' world!'
    }
}
```

```
@Service
export class UserServiceTest{
    constructor(private userService: UserService){}

    @Test
    getUserName(){
        expect(this.userService.getUserName('hello')).toEqual('hello world!');
    }
}
```

三、Node保障

稳定性？性能？监控？

基建保障

1、日志平台

- 日志聚合
- 日志查询

2、分布式链路跟踪

- 链路串联
- 错误链路排查
- 慢链路排查

3、监控中心

- 内存、CPU等监控
- 流量等监控
- 服务状态等监控

4、配置中心

- 动态配置

```
@EnableApolloConfig('application')
@Service
export class NpmConfig{

  @Value('scope', ['@yx-module']
  scope!: String[];

  @ApplloConfigChangeListener("application")
  filedChange(field: String[]){
      // 部分配置修改需要做一些系统之间的联动
  }
}
```

5、分布式事务调度

- 事务
- 回滚

6、CI/CD

- 加速上线流程

四、Serverless与Knative

服务部署？服务器资源？流量扩缩容？

什么是Serverless?

Serverless = FaaS + BaaS

无服务器的思想，研发无需关心服务器，关注业务本身

云上的Serverless模式

FaaS：函数计算，自动缩放容
BaaS：云服务，自动缩放容

云服务提供了统一的SDK，
写函数即可，使用云计算服务

起飞

我们的Serverless怎么搞

都要用FaaS吗?

我们的BaaS在哪儿?

我们的BaaS层不是云服务

Serverless + 微服务

先跑起来

适合我们的框架-Knative

工作负载

和标准化的FaaS不同，
Knative期望能够运行所有的工作负载：

- Function
- Microservice
- Traditional Application
- Container

平台支撑

Knative建立在K8S和istio之上

K8S容器管理能力

Istio网络管理功能

knative对严选的价值

✓支持 传统应用+微服务+函数计算服务，混合模式
✓多云战略，不会被某个云提供商锁定，可在不同云平台之间移植

Knative是云原生中三个领域的最佳实践的结合

knative是一个Serverless平台，包含三大组件

Knative原理

从使用Serving组件开始

镜像  服务

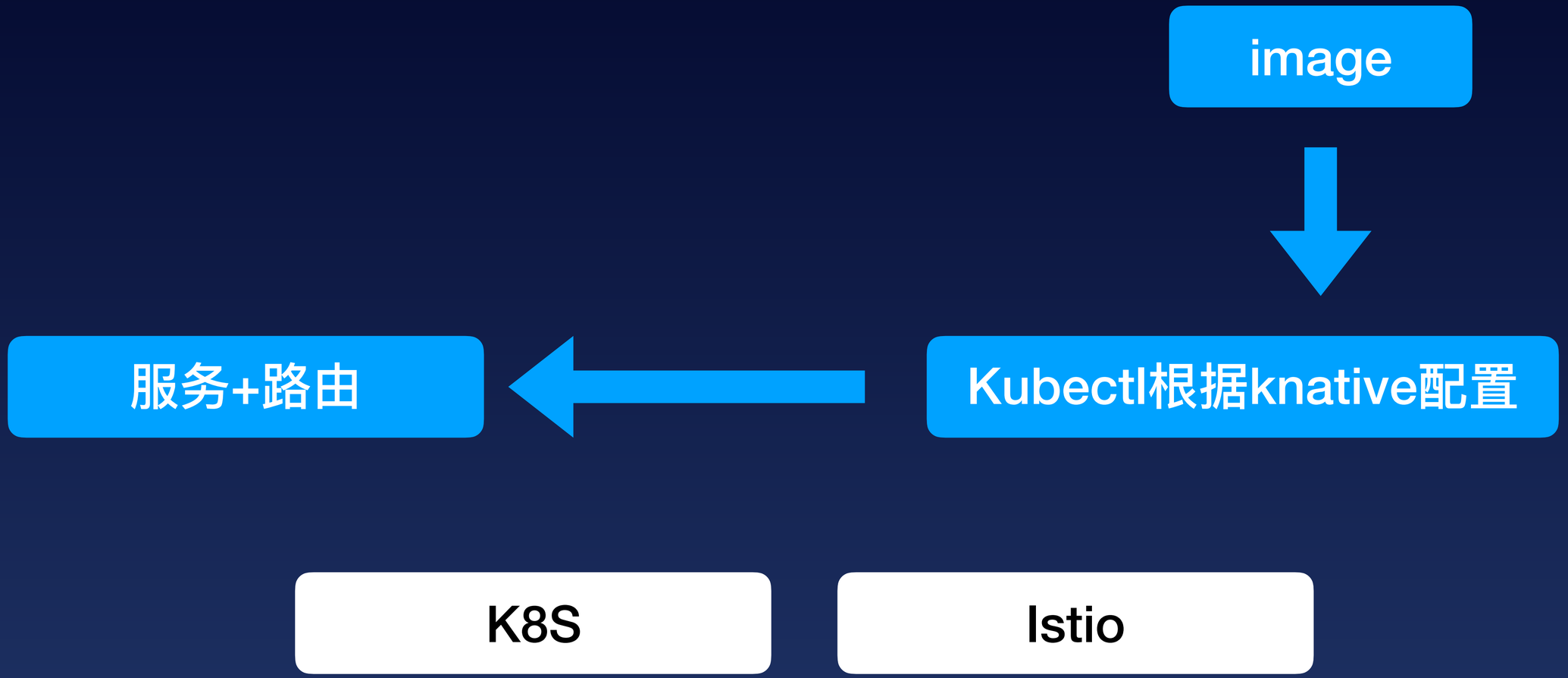


image
Kubectl根据knative配置
服务+路由
K8S
Istio

依赖于K8s的动态scale

利用Istio的服务治理功能

使用Build组件

源码

镜像

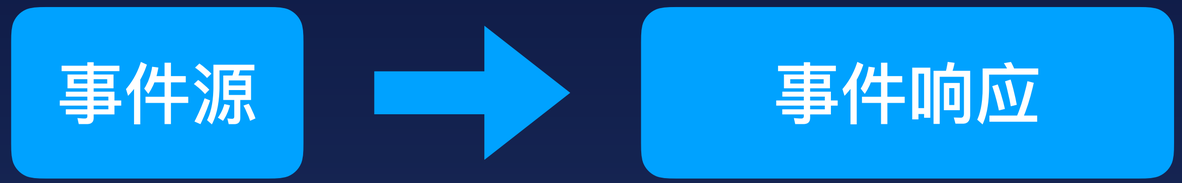

源码
Buildtemplate
镜像

开发Buildtemplate

```yaml
apiVersion: build.knative.dev/v1alpha1
kind: BuildTemplate
metadata:
    name: yyyyy-template
spec:
    paramters:
        - name: IMAGE
          description: Where to publish the result image
    steps:
        - name: node-hello
          image: xxx/yyyyyy
          command:
          - bash
          args:
          - -c
          - |…
        - name: export
          image: gcr.io/kaniko-project/executor@sha256:30ba460a034a8051b3222a32e20cb6049741e58384e3adf8c8987c004e2f2ab9
          args:
          - --context=/workspace/dist
          - --dockerfile=/workspace/dist/Dockerfile
          - --destination=${IMAGE}
```


steps
拉代码
构建
生成镜像
推送到仓库

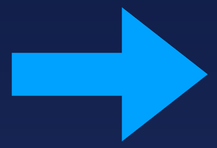
使用Event组件


事件源
事件响应

构建严选的Serverless Framework


产品化
UI
开发者工具
CLI
WEB IDE
基础设施层,serverless runtime

knative



应用级别

模块级别

FaaS

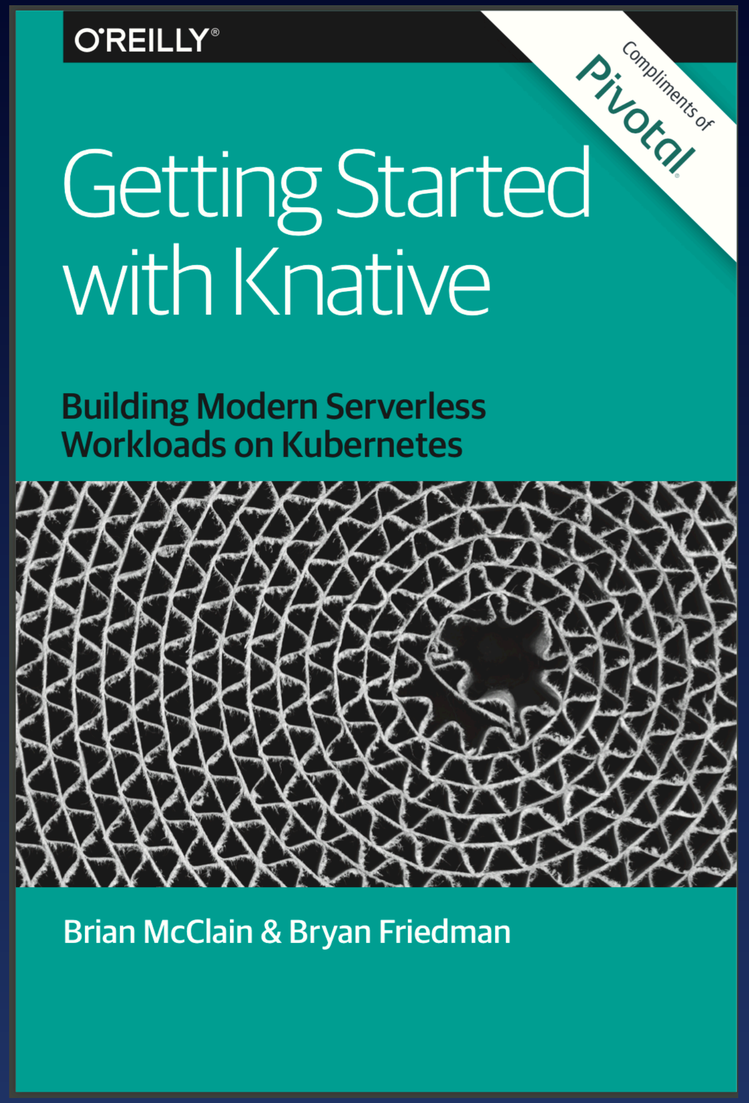
前途不可限量

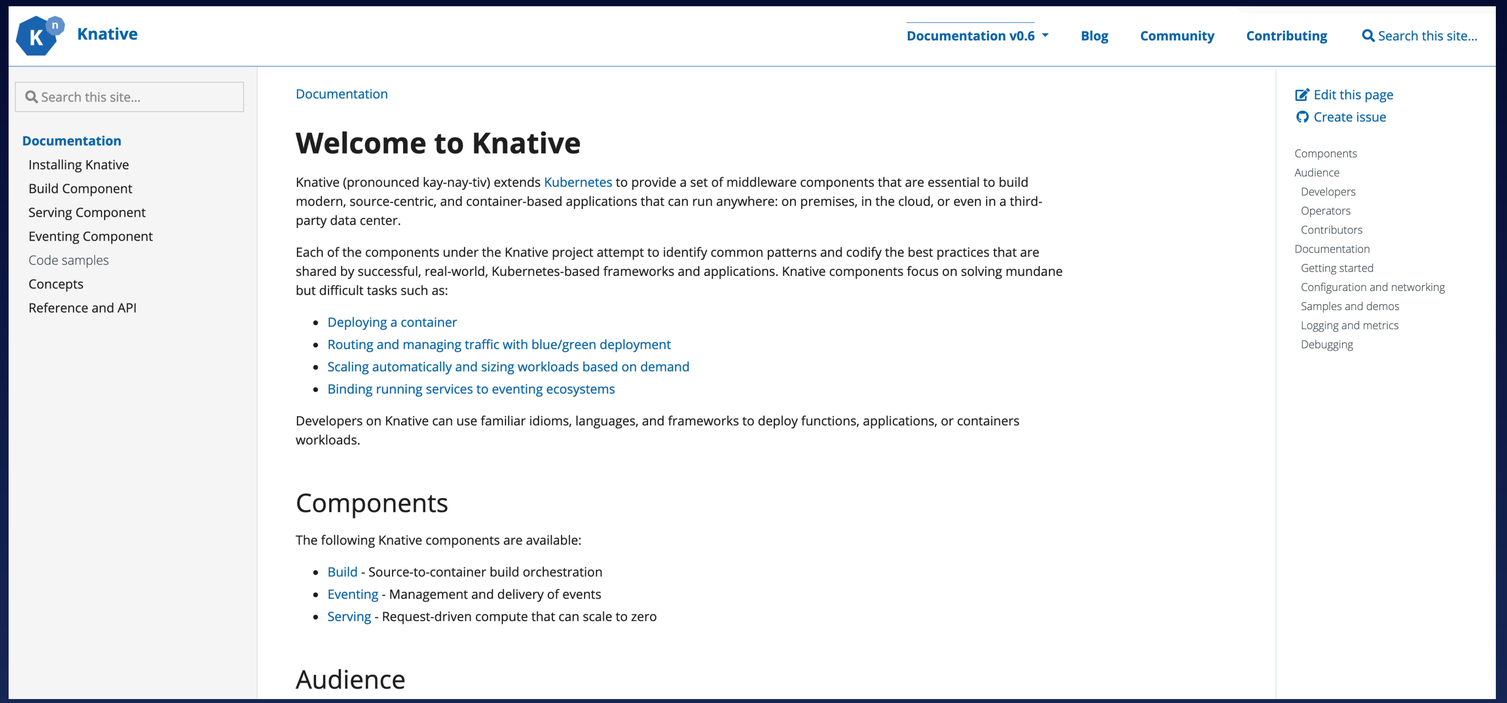
✓Knative的优势

- 不拘泥于FaaS，支持BaaS和传统应用，适用性更广泛
- 平台化，标准化，不锁定于某一个云提供商

✓存在的问题

- 项目发展不久，当前V0.6版本
- 文档这块较为欠缺


O'REILLY®
Compliments of Pivotal
Getting Started with Knative
Building Modern Serverless Workloads on Kubernetes
Brian McClain & Bryan Friedman


Knative
Documentation v0.6
Blog
Community
Contributing
Search this site...
Search this site...
Documentation
Installing Knative
Build Component
Serving Component
Eventing Component
Code samples
Concepts
Reference and API
Documentation
Welcome to Knative
Knative (pronounced kay-nay-tiv) extends Kubernetes to provide a set of middleware components that are essential to build modern, source-centric, and container-based applications that can run anywhere: on premises, in the cloud, or even in a third-party data center.
Each of the components under the Knative project attempt to identify common patterns and codify the best practices that are shared by successful, real-world, Kubernetes-based frameworks and applications. Knative components focus on solving mundane but difficult tasks such as:
Deploying a container
Routing and managing traffic with blue/green deployment
Scaling automatically and sizing workloads based on demand
Binding running services to eventing ecosystems
Developers on Knative can use familiar idioms, languages, and frameworks to deploy functions, applications, or containers workloads.
Components
The following Knative components are available:
Build - Source-to-container build orchestration
Eventing - Management and delivery of events
Serving - Request-driven compute that can scale to zero
Audience
Edit this page
Create issue
Components
Audience
Developers
Operators
Contributors
Documentation
Getting started
Configuration and networking
Samples and demos
Logging and metrics
Debugging

knative.dev

五、Service Mesh的价值

服务治理

传统的微服务治理

service mesh构想的微服务治理

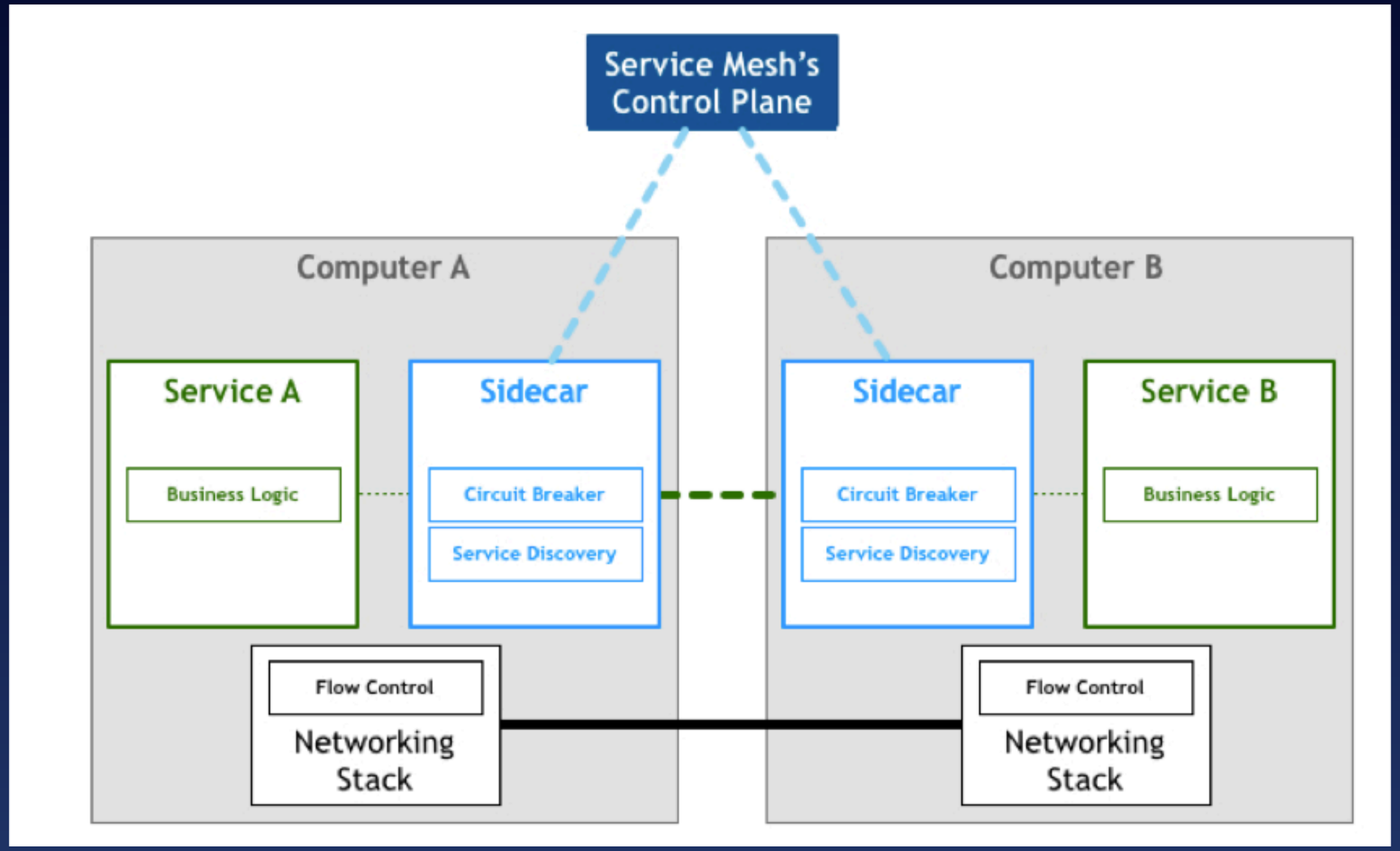
语言层面 → 语言无关

service mesh的微服务治理

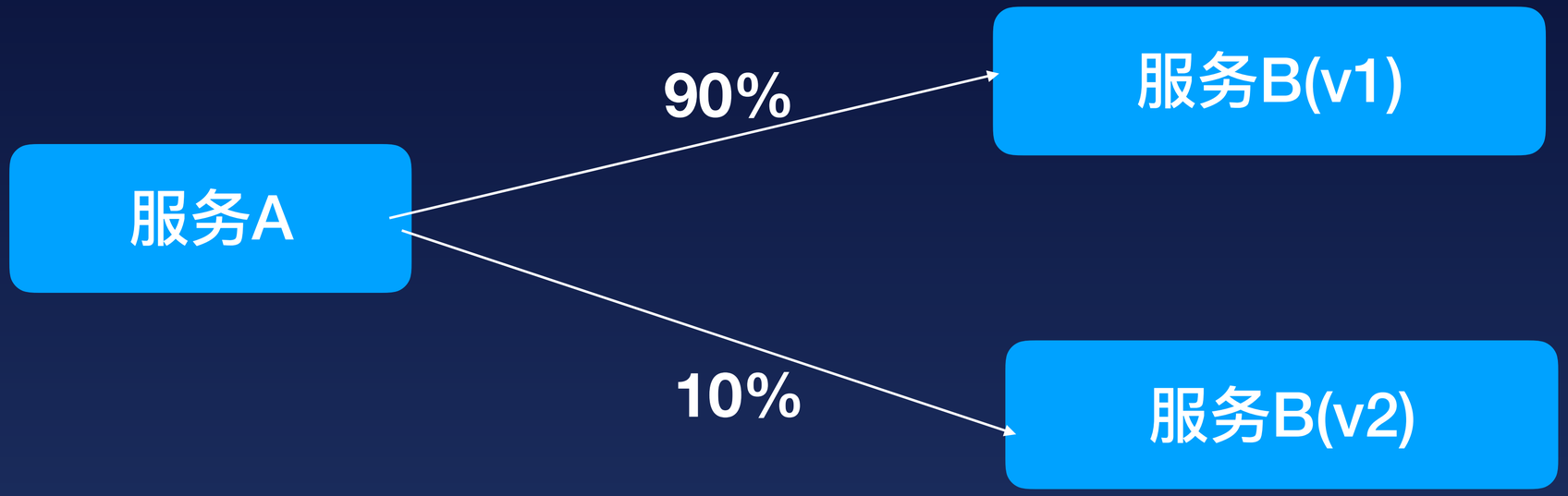
下沉为通讯层、并利用底层基础设施

中文：”服务网格”

基础设施层
处理服务间通信
轻量级网络代理



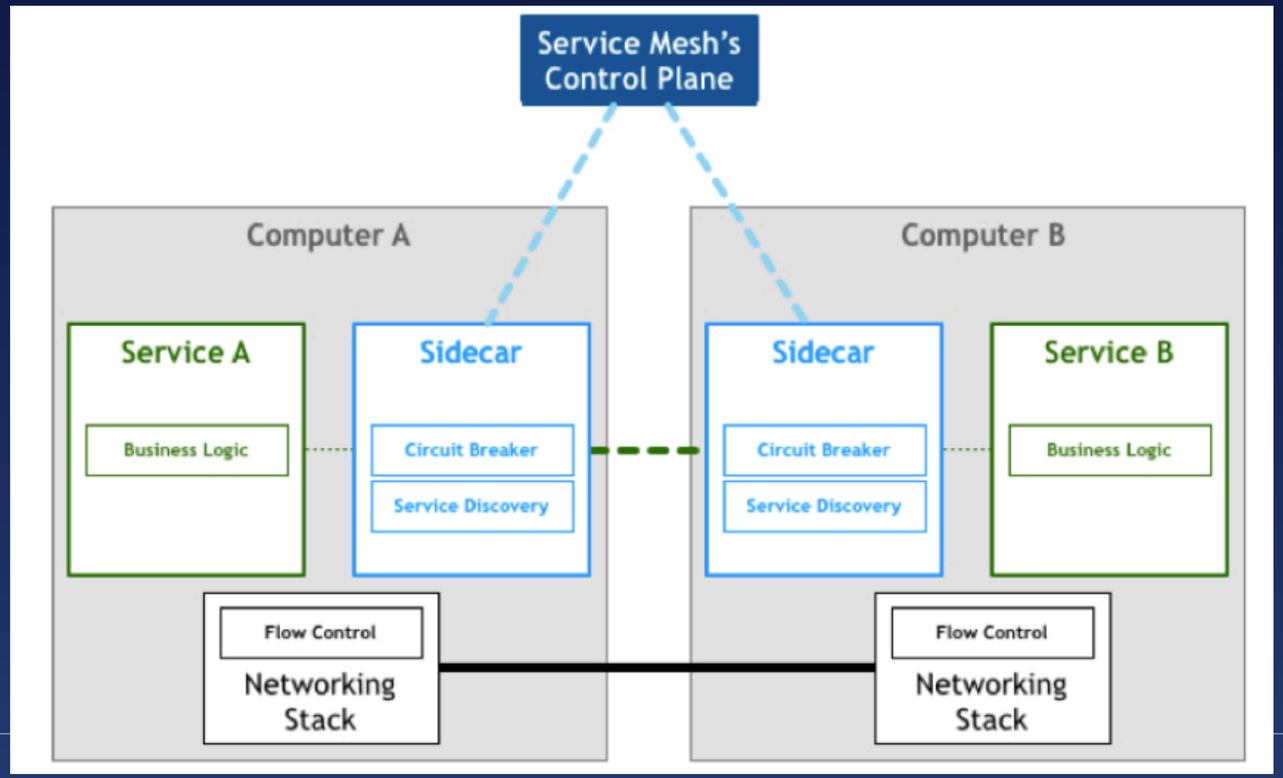
举个例子-流量管理



举个例子-流量管理

控制面

数据面



前途不可限量

- 路由能力
- 安全：认证，加密，限流，熔断
- 请求转发
- 链路跟踪
- ....

是不是万能的?

不是万能的

- 链路跟踪这类细粒度
- 性能消耗，对于业务的影响
- 应用架构决定
- ....

总结

业务  前端架构

THANKS

GMTC
全球大前端技术大会

Geekbang 极客邦科技 | TGO 鲲鹏會

TGO鲲鹏会

汇聚全球科技领导者的高端社群

全球12大城市

850+高端科技领导者

使命 Mission

为社会输送更多优秀的科技领导者

愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

极客大学

前端训练营

用3个月时间，彻底学透前端开发必备技能

- 线下线上混合式学习
- 名师手把手教学
- 一线大厂项目实操
- 毕业即享内推服务

了解详情

讲师：程劭非（winter）

前手机淘宝前端负责人

Q&A