携程Node.js技术应用实践

付文平

1 前后端分离

2 Node.js与RestfulAPI

3 RestfulAPI->GraphQL

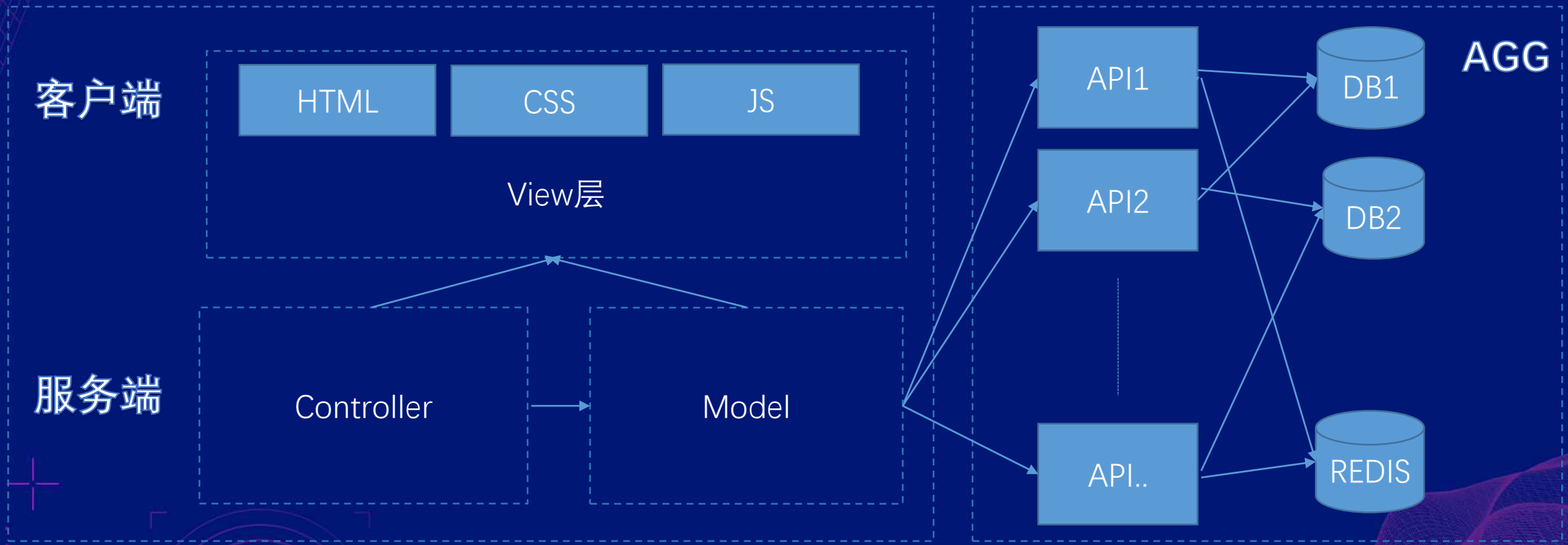
1

前后端分离

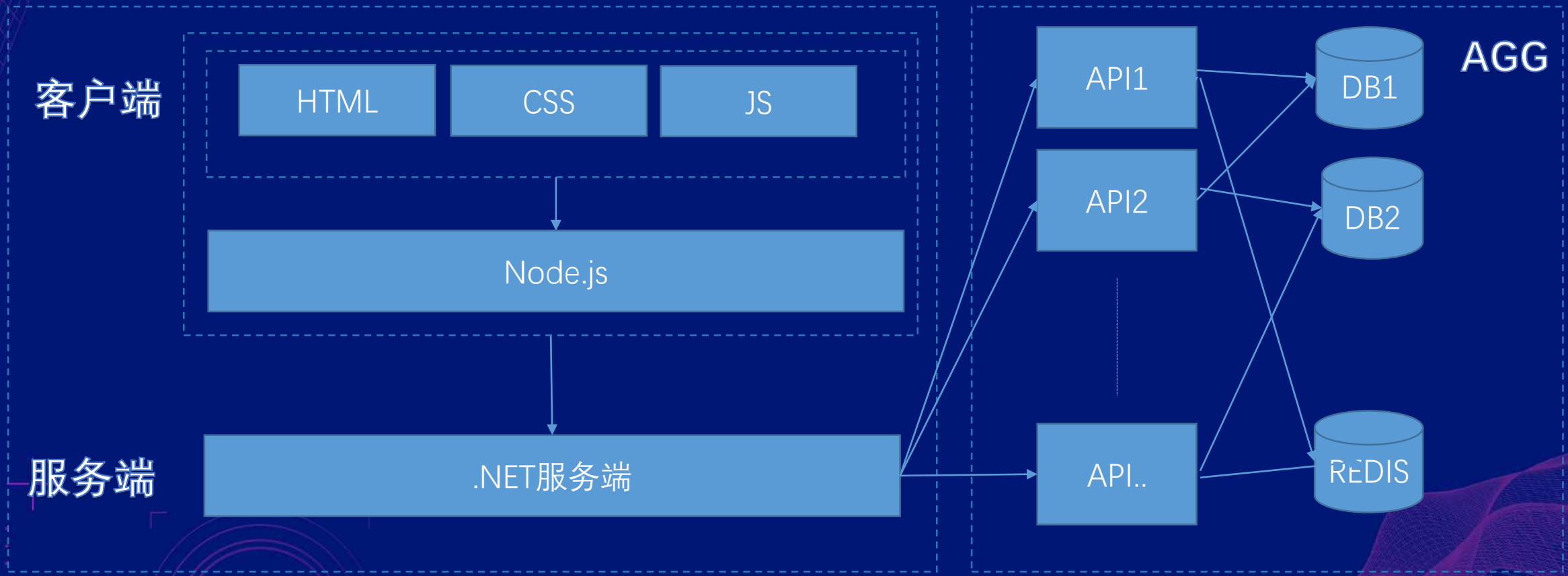
前后端分离的技术痛点

- 机票H5代码历史久远，传统.NET MVC开发模式
- 前后端代码耦合在一起，维护成本大，无法完成项目快速迭代上线
- 展示逻辑和业务逻辑混杂，前后端开发同学职责不明确
- 整体项目后续的扩展性较差，可维护性较低

历史架构


客户端
HTML
CSS
JS
View层
服务端
Controller
Model
AGG
API1
API2
API..
DB1
DB2
REDIS

架构提升


客户端
HTML
CSS
JS
Node.js
服务端
.NET服务端
AGG
API1
API2
API..
DB1
DB2
REDIS

架构组成


PM2+Node.js+Express
Redis
Abtest
Cdn
SSR
Fplus
Ctriputil
Front-end
Vue
LizardLite
Unitest
webpack
vuex
component
router
model
store
user
mocha
eslint
dev server

项目目录结构

```
├── dev     开发目录          服务器、打包脚本
├── dist    发布目录
├── build   调试目录
└── src     源码目录
    ├── controllers           页面控制器
    ├── models                单独定义的Models
    ├── stores                单独定义的Stores
    ├── assets                静态资源
    │   ├── built             编译后的代码，不在源码目录
    │   ├── fonts
    │   ├── images
    │   ├── scripts
    │   └── styles
    ├── fue
    │   ├── components        UI组件
    │   │   ├── alert
    │   │   ├── layer
    │   │   └── toast
    │   └── plugins           Vue插件
    └── vstore                Vuex Store
        ├── actions           共用的 Actions
        ├── getters           共用的 Getters
        ├── modules           子模块 可包含该模块自身的 actions/getters/mutations/plugins
        ├── mutation-types    变异类型
        ├── mutations         变异
        └── plugins           插件
```

改造后架构

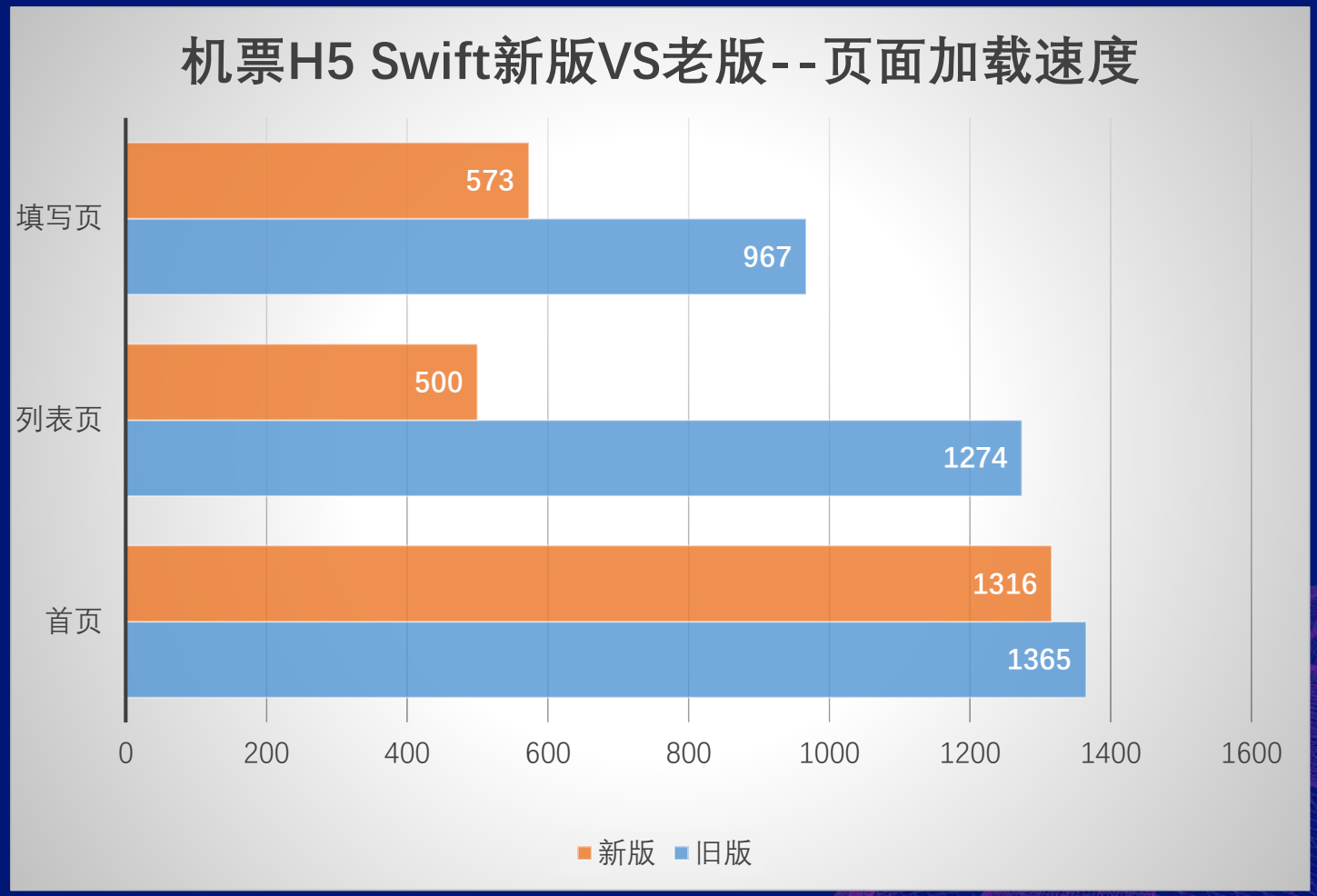
- 模块化
  ES6 import + System.import + vue单文件组件
- 单页路由控制
  vue-router + async component
- 服务器通信
  同构的businiess model（LizardLite.AbsModel）
- 状态管理
  vuex store
- 代码质量
  standardjs + eslint + mocha + chai
- 构建发布
  webpack dev server + npm scripts + html-minifier/uglify js/clean css

改造后架构

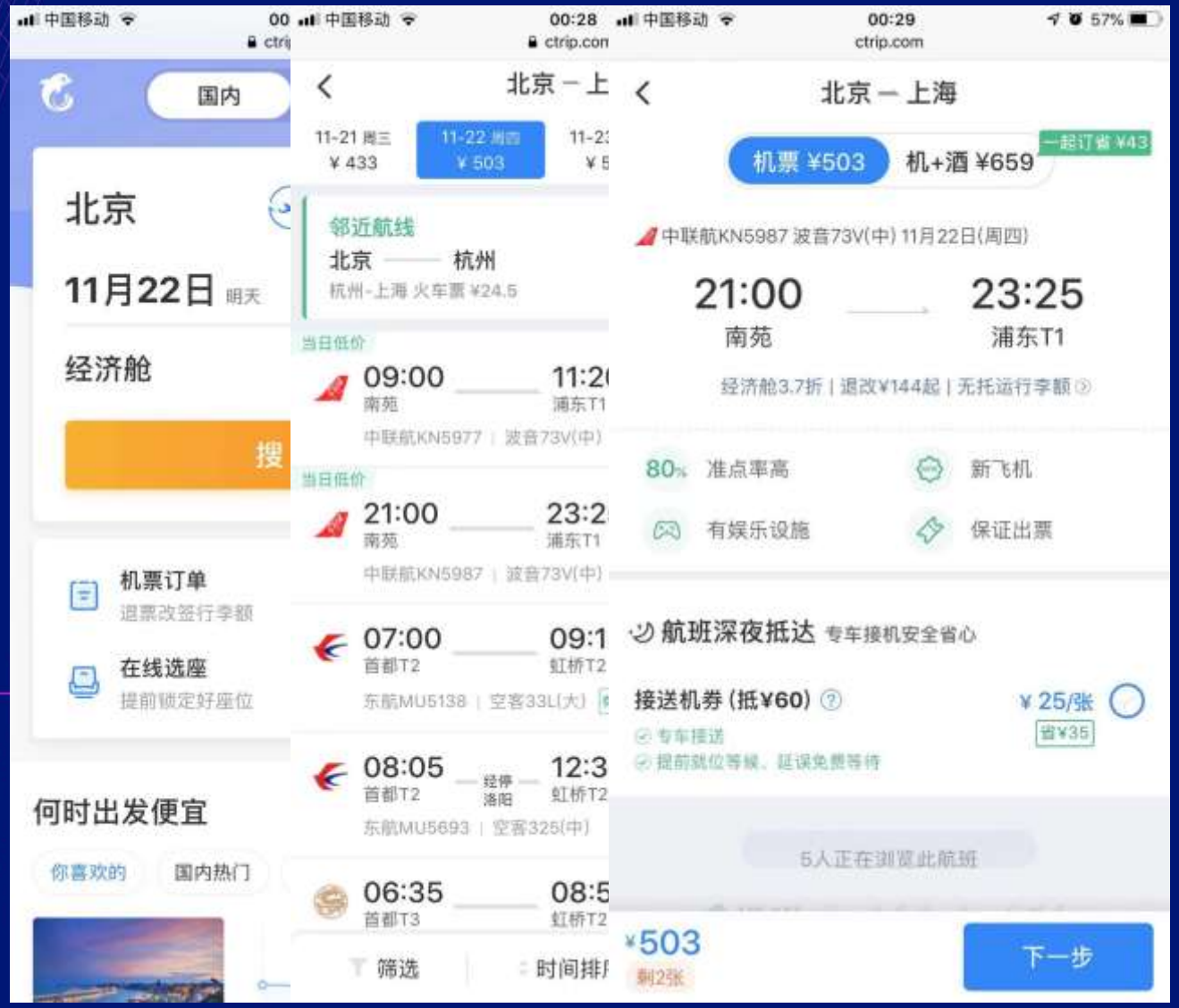
- SPA，单页模式，app-like式的体验
- 针对SEO，支持SSR模式
- 代码复用，客户端、服务端共用
- 资源和数据的双重预加载

上线效果

- 查询首页加载时长缩短49ms，提速3.6%
- 列表页加载时长缩短774ms，提速60.8%
- 填写页加载时长缩短394ms，提速40.7%



H5体验


国内
北京
11月22日 明天
经济舱
搜
机票订单
退票改签行李额
在线选座
提前锁定好座位
何时出发便宜
你喜欢的
国内热门
北京 — 上
11-21 周三 ¥433
11-22 周四 ¥503
邻近航线
北京 杭州
杭州-上海 火车票 ¥24.5
09:00 南苑
11:2 浦东T1
中联航KN5977 | 波音73V(中)
21:00 南苑
23:2 浦东T1
中联航KN5987 | 波音73V(中)
07:00 首都T2
09:1 虹桥T2
东航MU5138 | 空客33L(大)
08:05 首都T2
经停 洛阳
12:3 虹桥T2
东航MU5693 | 空客325(中)
06:35 首都T3
08:5 虹桥T2
筛选
时间排
北京 — 上海
机票 ¥503
机+酒 ¥659
一起订省 ¥43
中联航KN5987 波音73V(中) 11月22日(周四)
21:00 南苑
23:25 浦东T1
经济舱3.7折 | 退改¥144起 | 无托运行李额
80% 准点率高
新飞机
有娱乐设施
保证出票
航班深夜抵达 专车接机安全省心
接送机券 (抵¥60)
¥ 25/张
省¥35
专车接送
提前就位等候，延误免费等待
5人正在浏览此航班
¥503
剩2张
下一步

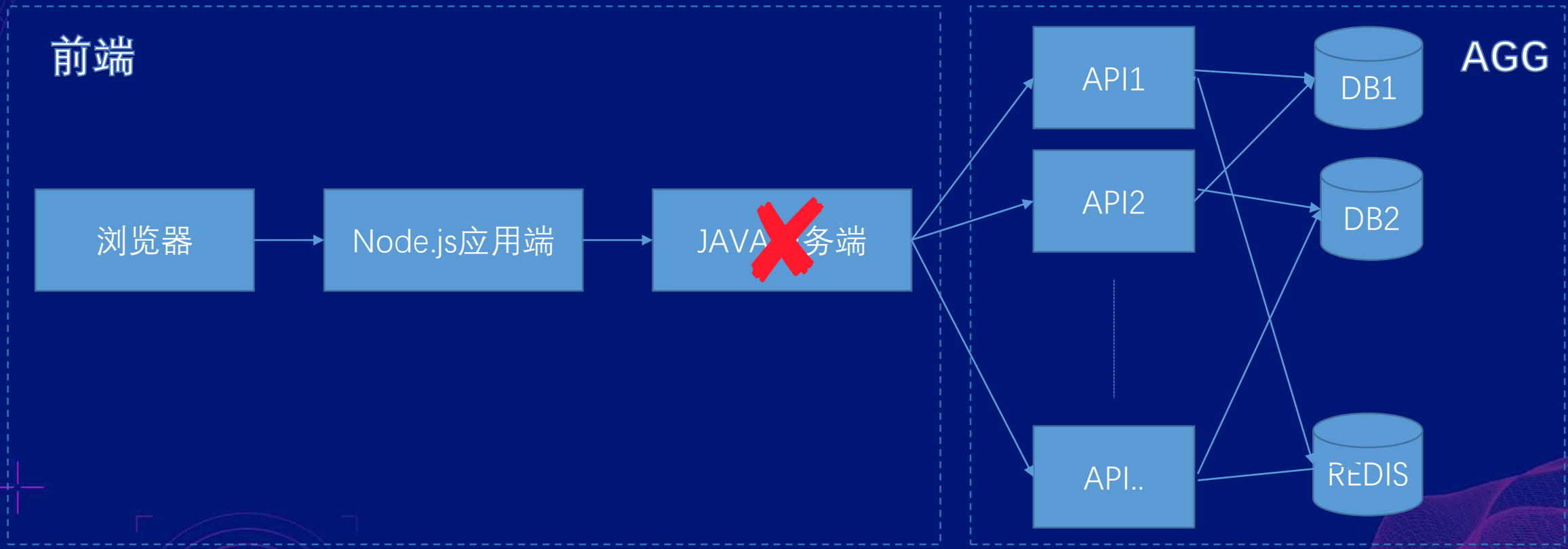
2

Node.js与RestfulAPI

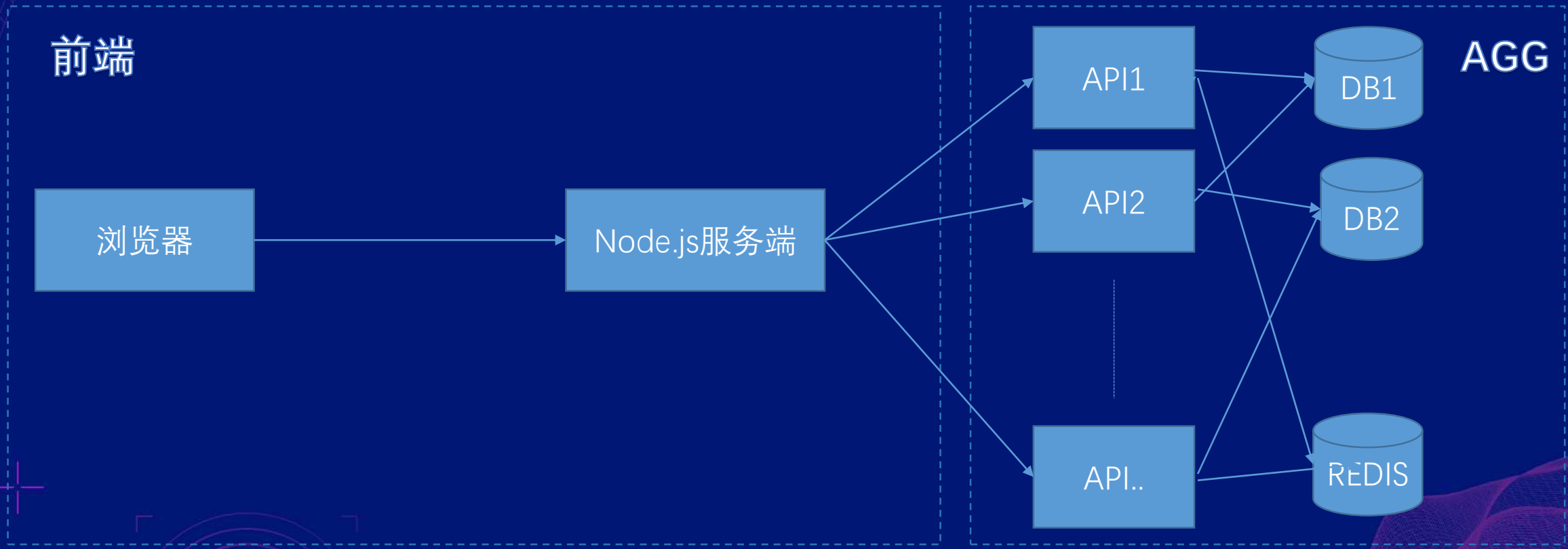
传统方式面临的问题

- 端到端之间调用链太深
- Scrum团队内资源协调困难
- 服务端资源短缺
- 打造全栈工程师

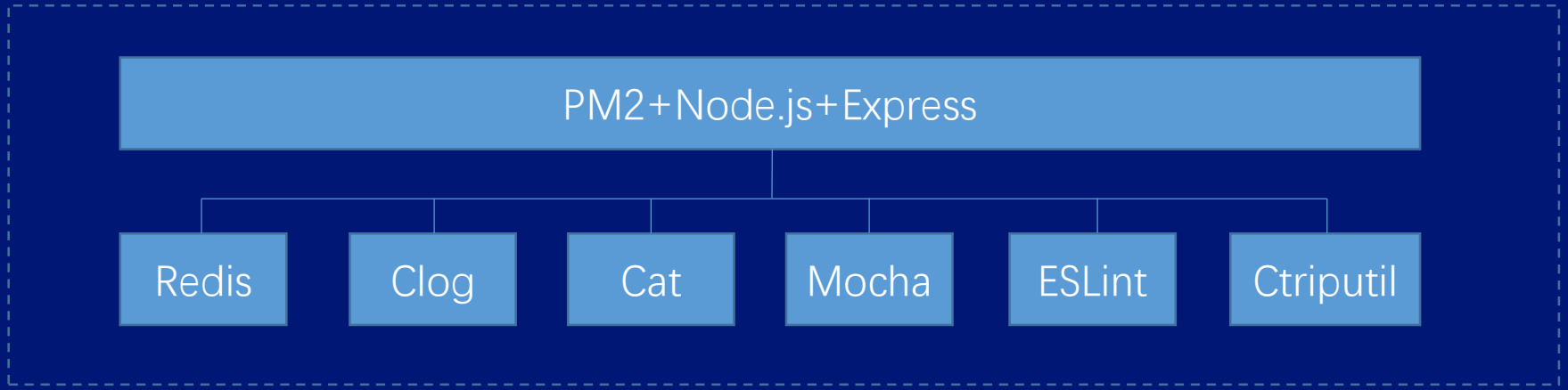
之前的模式



架构改造



技术实现



- RestfulAPI自动化注册方式
- 封装底层接口调用的请求、响应报文、异常处理
- 引入mocha进行接口单元测试覆盖，分支覆盖93%
- 接入ESLint和Sonar检查
- 实现与JAVA相同的报文，并接入契约管理平台

代码实现细节

```
module.exports = function restapi ({ app, vd }) {
  const path = require('path')
  const REST_API_ROOT = path.join(path.resolve('.'), 'restapi')
  const wrap = require('../utils/wrap.js')
  const dir = require('../utils/dir.js')
  const router = require('express').Router()

  const operations = []

  dir(REST_API_ROOT, (route, module) => {
    const { Request, Response } = module

    operations.push({
      Name: route,
      RequestMessage: { Json: JSON.stringify(Request || {}) },
      ResponseMessage: { Json: JSON.stringify(Response || {}) },
    })

    switch (typeof module) { // 注册路由
      case 'object': // 如果模块导出对象是一个对象，则判断 是否有 validate/process
        if (module && (typeof module.validate === 'function') && (typeof module.process === 'function')) {
          module = wrap(module)
          router.post('/' + route, module)
          router.post('/json/' + route, module) // 注册 json 类型的 SOA 调用入口
        }
        break
      case 'function': // 如果是个函数，则直接将该函数做为路由的处理函数
        router.post('/' + route, module)
        router.post('/json/' + route, module)
        break
    }

  })

  app.use(vd, router)

  app.get(vd + '/_operationInfo', (req, res) => {
    res.json(operations)
  })
```

自注册路由

```
/**
 * 发送SOA请求
 * serviceCode/url
 * soa method
 * params
 */
soaAgent: function (serviceCode, method, params, query) {
  const start = Date.now() // 记录请求开始的时间

  serviceCode = this.getConfig(`${SOA_PREFIX}_${serviceCode}_${method}`) || this.getConfig(`${SOA_PREFIX}_${serviceCode}`) || serviceCode

  let preifx = `${serviceCode}_${method}_`
  this.logTime(`${preifx}Start`)

  const loginfo = {guid: this.Guid, ClientIP: this.getRemoteIpAddress(), ClientId: this.ClientId, serviceCode, method}

  // 记录发送的请求参数
  clog.info(Object.assign({type:'request'}, loginfo), `soa request: ${serviceCode}\t${method}`, JSON.stringify(params), `开始时间:${start}`)

  params.rejectIncludeResponse = true

  return CtripUtil.SoaAgent(serviceCode).invoke(method, params, true, query).then((res) => {
    this.logTime(`${preifx}End`)

    const end = Date.now()
    const used = end - start

    clog.info(Object.assign({type:'response'}, loginfo), `soa response: ${serviceCode}\t${method}`, `结束时间: ${end}\t耗时:${used}`)

    return res
  }).catch(([e, params, result]) => {
    this.logTime(`${preifx}Error`)

    const end = Date.now()
    const used = end - start

    clog.warn(Object.assign({type:'response'}, loginfo), `soa response: ${serviceCode}\t${method}`, e, JSON.stringify(params), JSON.stringify(result),

    return Promise.reject(e)
  })
},
```

SOA调用封装

上线效果

18个

对外提供的API有18个，打通公司SOA注册服务，可通过gateway路由

50ms

服务平均耗时50ms，cpu使用率5%，内存使用在2GB左右

50%

上线后只需要4台服务器，比原先8台.NET服务器缩减50%

93.26%

单测用例数883，代码单测分支覆盖率93.26%，行覆盖98%

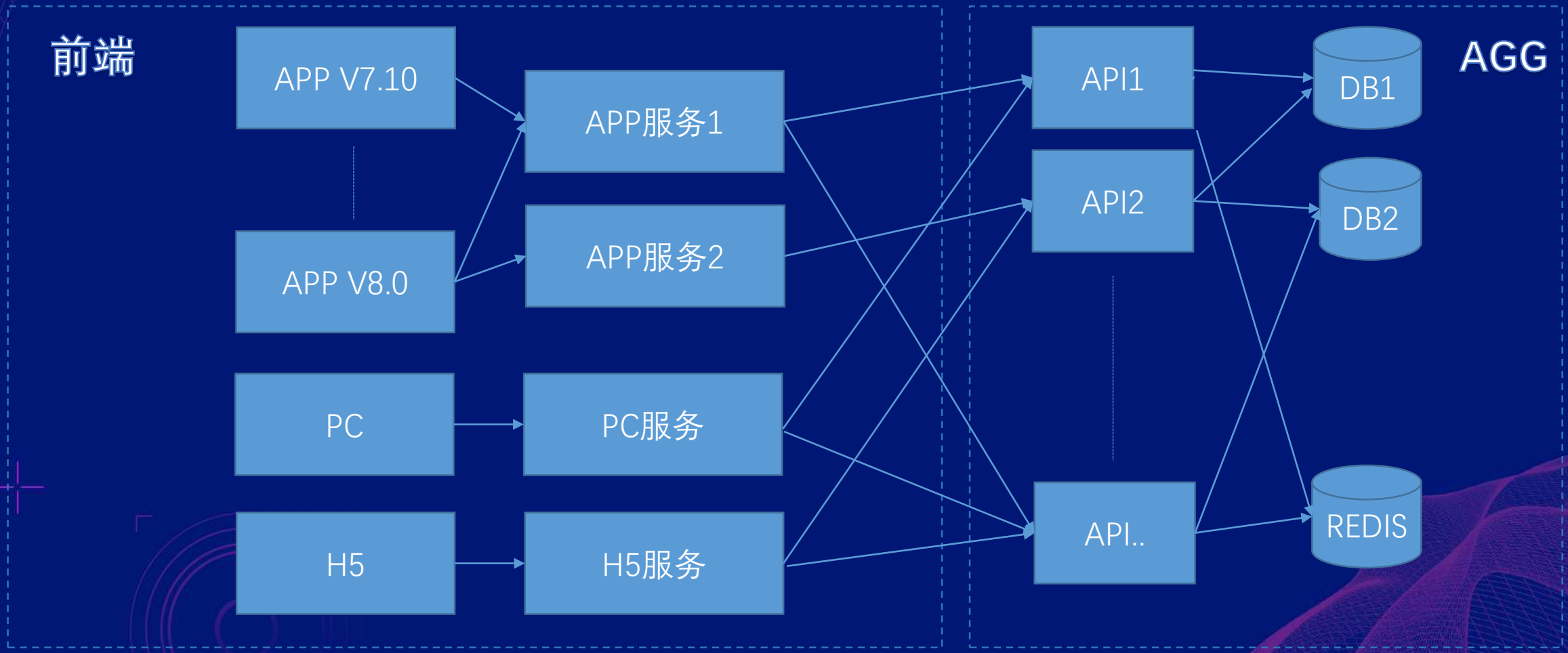
3

RestfulAPI->GraphQL

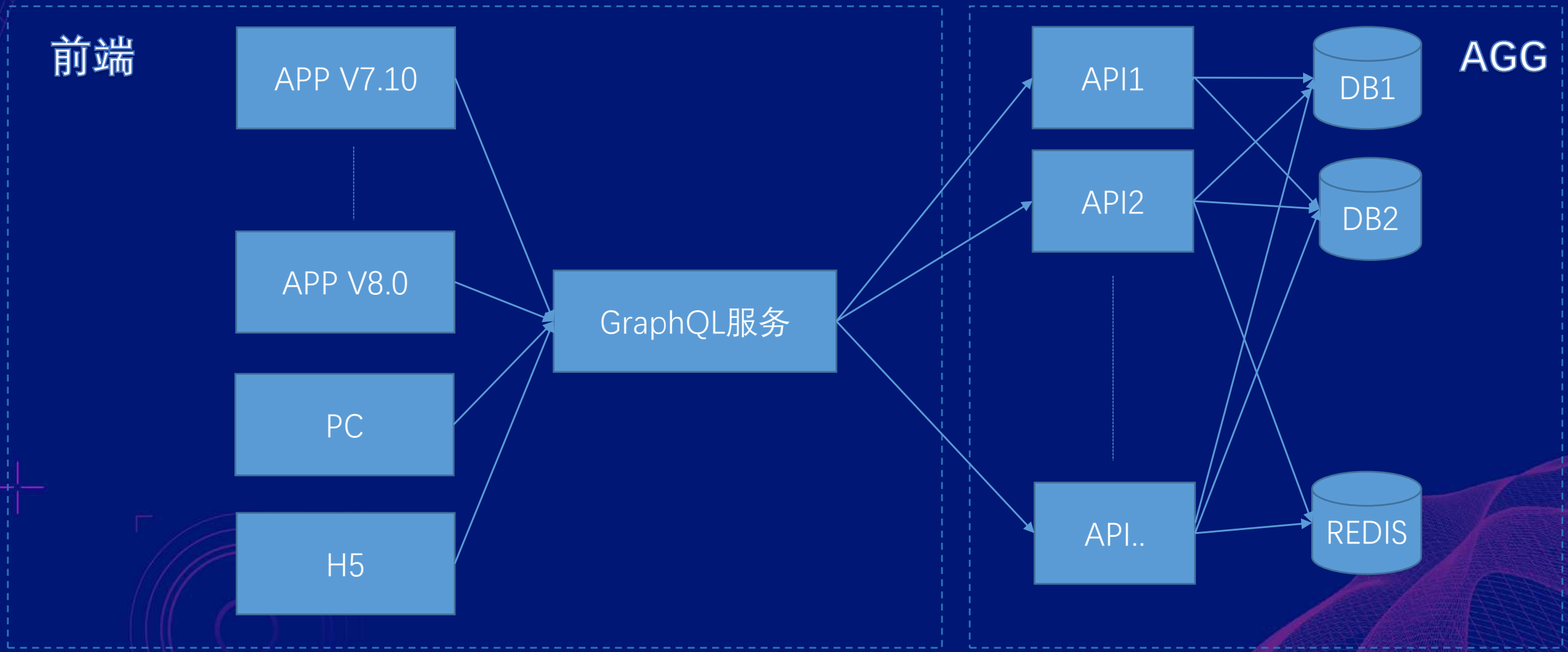
RestfulAPI所存在的问题

- 客户端需求不同，接口内部定制化逻辑太多
- 不同客户端对数据的需求不一致
- 随着版本的迭代，数据存在较多冗余
- 客户端需要进行大量的API聚合
- 接口在完成新的功能时，还需要对之前的版本做兼容
- 后端接口定义的字段前端预期不匹配

RestfulAPI架构的调用模式



GraphQL架构的调用模式



技术实现

- 统一调用模式，/graphql？query=*
- 采用Express-GraphQL核心中间件
- 客户端按需加载，并自定义所需字段
- 合并多次请求为一个调用，减少网络开销

基于GraphQL接口调用实例

```
{
  city: getCityInfo(id: 1) {
    ID
    name: Name
    Code
  }
}
```

Requst

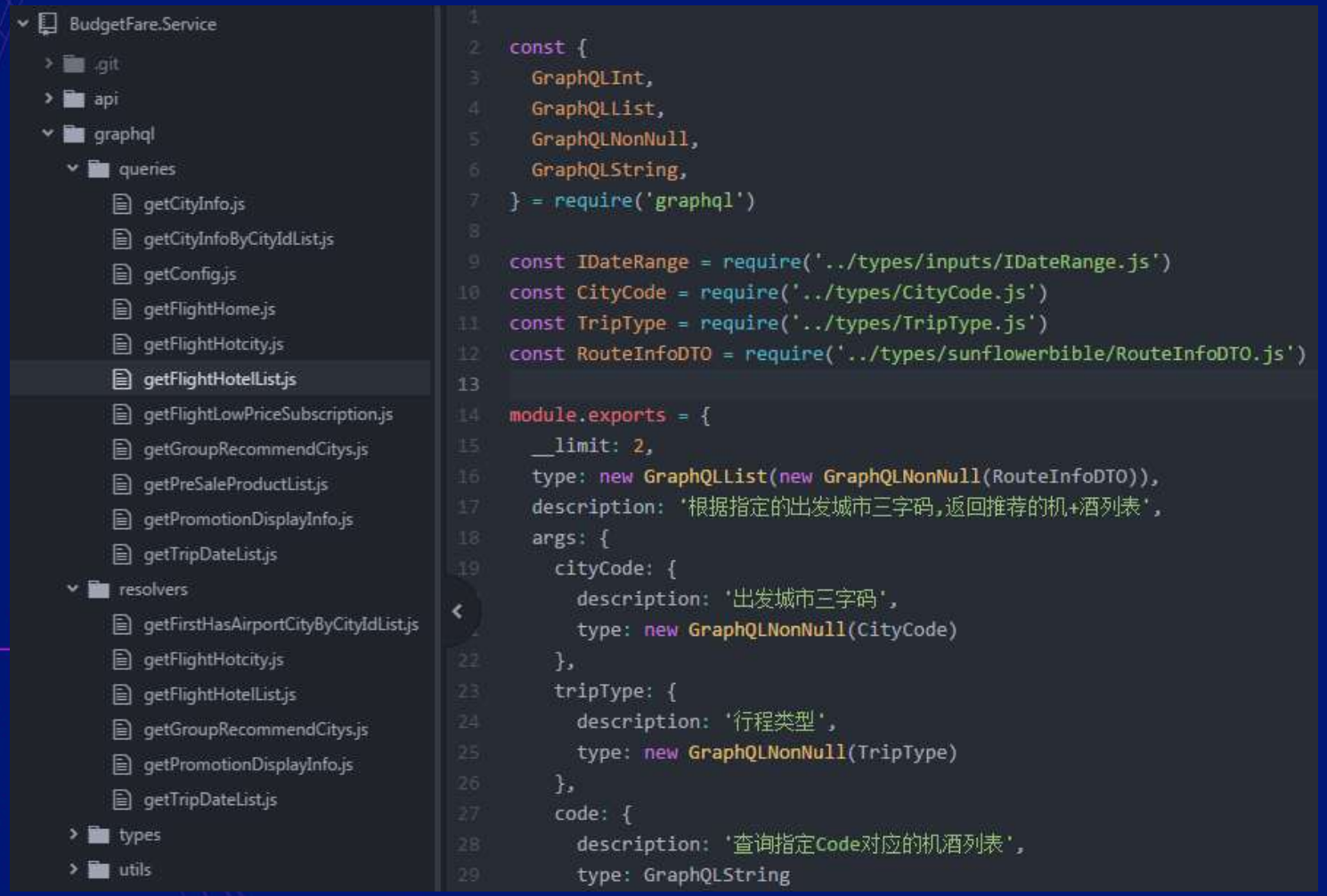
```
{
  "data": {
    "city": {
      "ID": 1,
      "name": "北京",
      "Code": "BJS"
    }
}
```

Response

```
module.exports = new GraphQLObjectType({
  'name': 'CityInfo',
  'description': '城市实体',
  'fields': {
    'Code': {
      'description': '城市三字码',
      'type': GraphQLString
    },
    'ID': {
      'description': '城市ID',
      'type': GraphQLInt
    },
    'Name': {
      'description': '城市名称',
      'type': GraphQLString
    }
  }
})
```

Schema

我们的使用

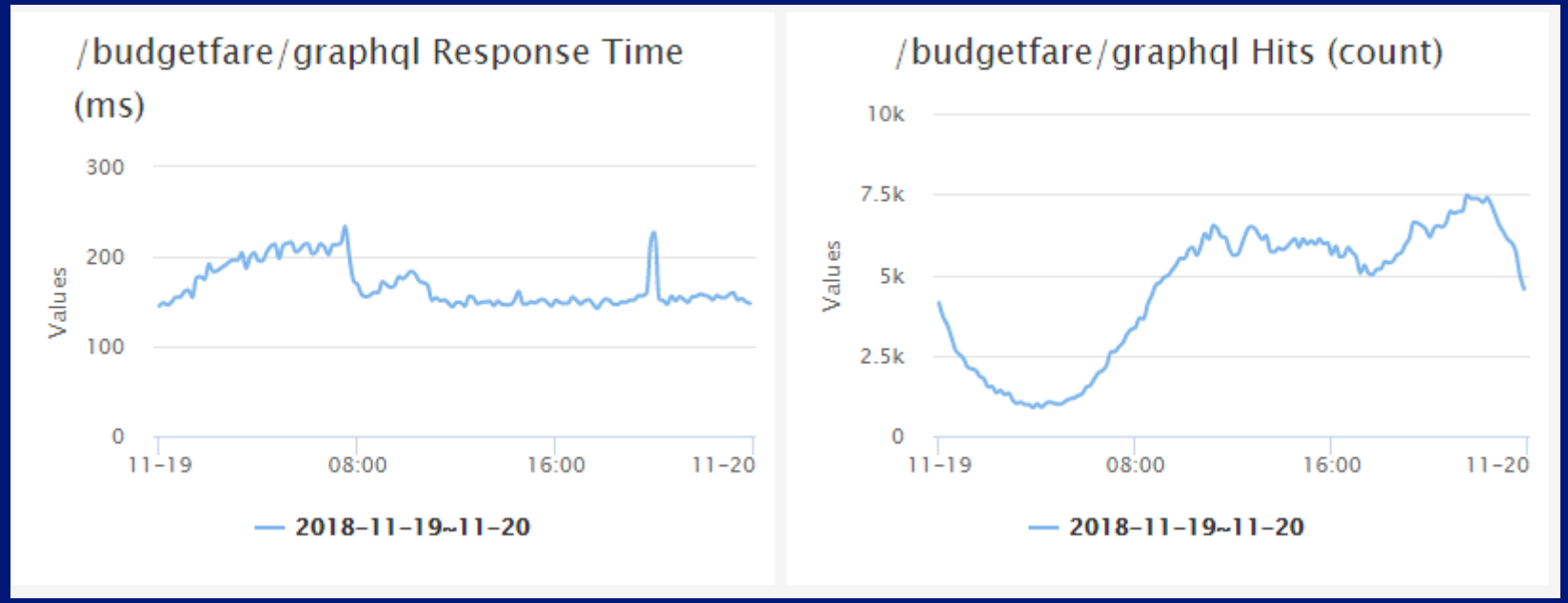


- 应用于特价机票核心模块
- 提供11个内部接口
- 支持多个APP版本
- 支持H5版本
- 统一前后端开发技术栈

GraphQL与RestfulAPI对比

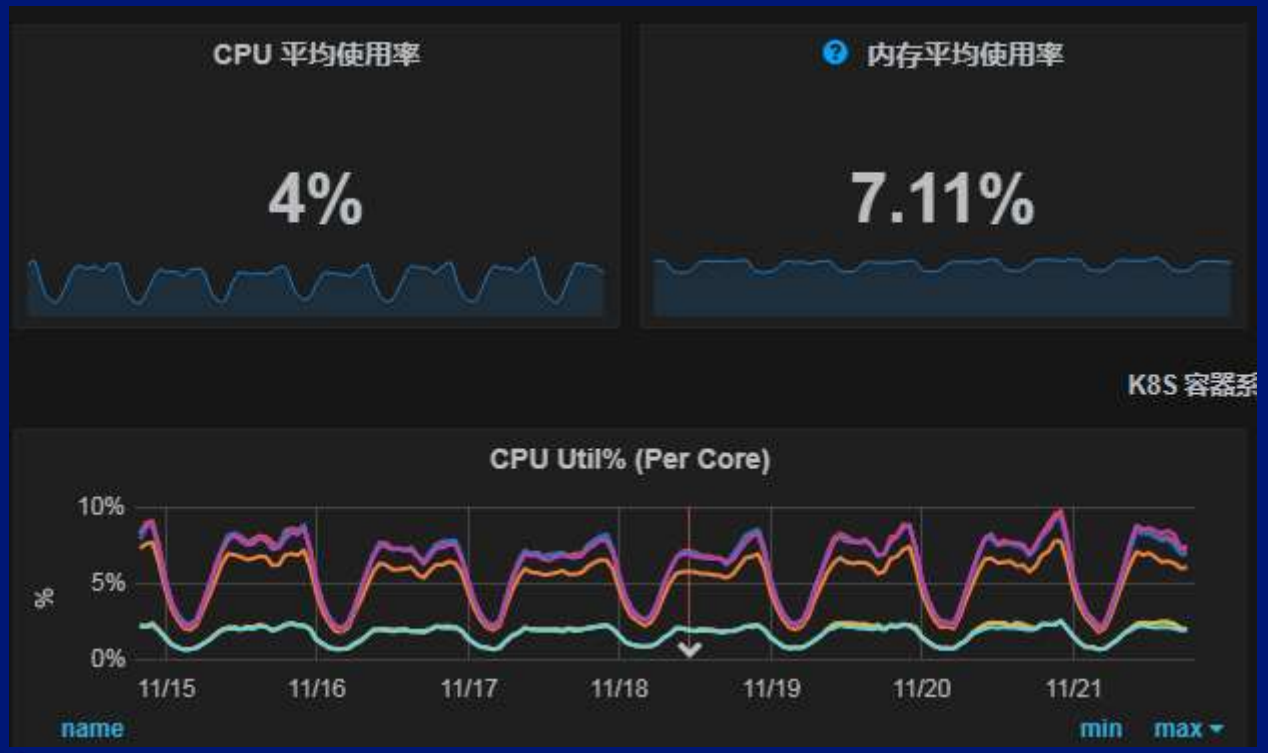
- **数据获取**：GraphQL可以按需获取，通过调用方指定schema返回不同报文，RestfulAPI则是下发相同的结构
- **URL入口**：Rest不同的请求入口不同，在请求的URL上需要做区分，GraphQL则是一个入口(/graphql?query=)，通过调用的request来区分
- **调用方式**：Rest获取多个不同接口数据时，需要并发调用多次，而GraphQL可以合并查询，降低网络开销

基于GraphQL接口指标1



- 日均流量近100W左右，
- 平均耗时在150ms（受制于AGG服务耗时）
- 平均的内部处理耗时在20ms左右

基于GraphQL接口指标2



- CPU平均使用率4%
- 内存平均使用率在7%

本PPT来自2018携程技术峰会

更多技术干货，请关注“携程技术中心”微信公众号