微医 WeDoctor

微医前端技术沙龙

WEDOCTOR FED CONFERENCE

2019.5.25

在 Node.js 微服务方向的探索

高翔

微医集团-用户技术部

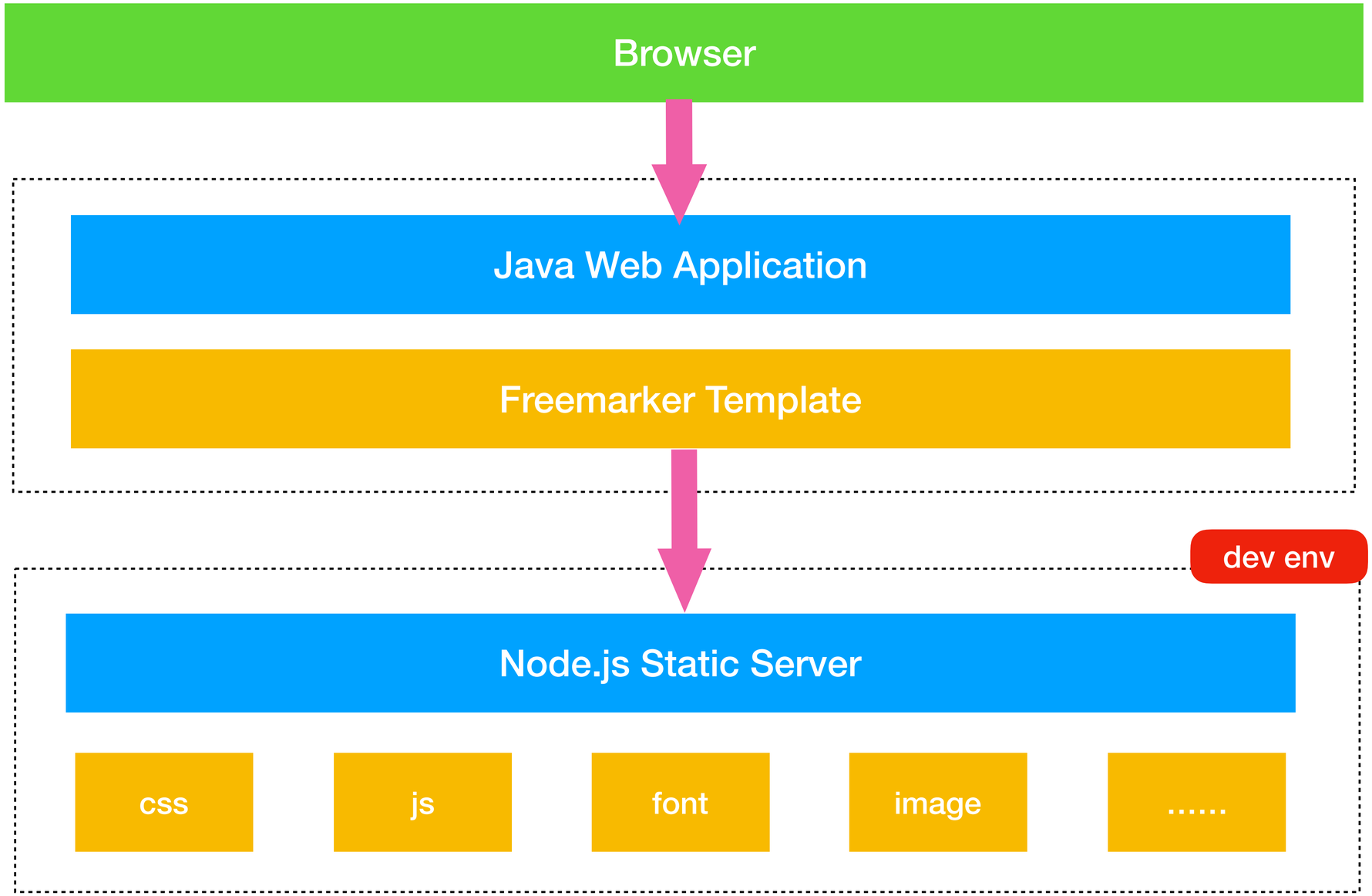
前端开发工程师、95后

主要负责业务：智能对话平台、健康号社区

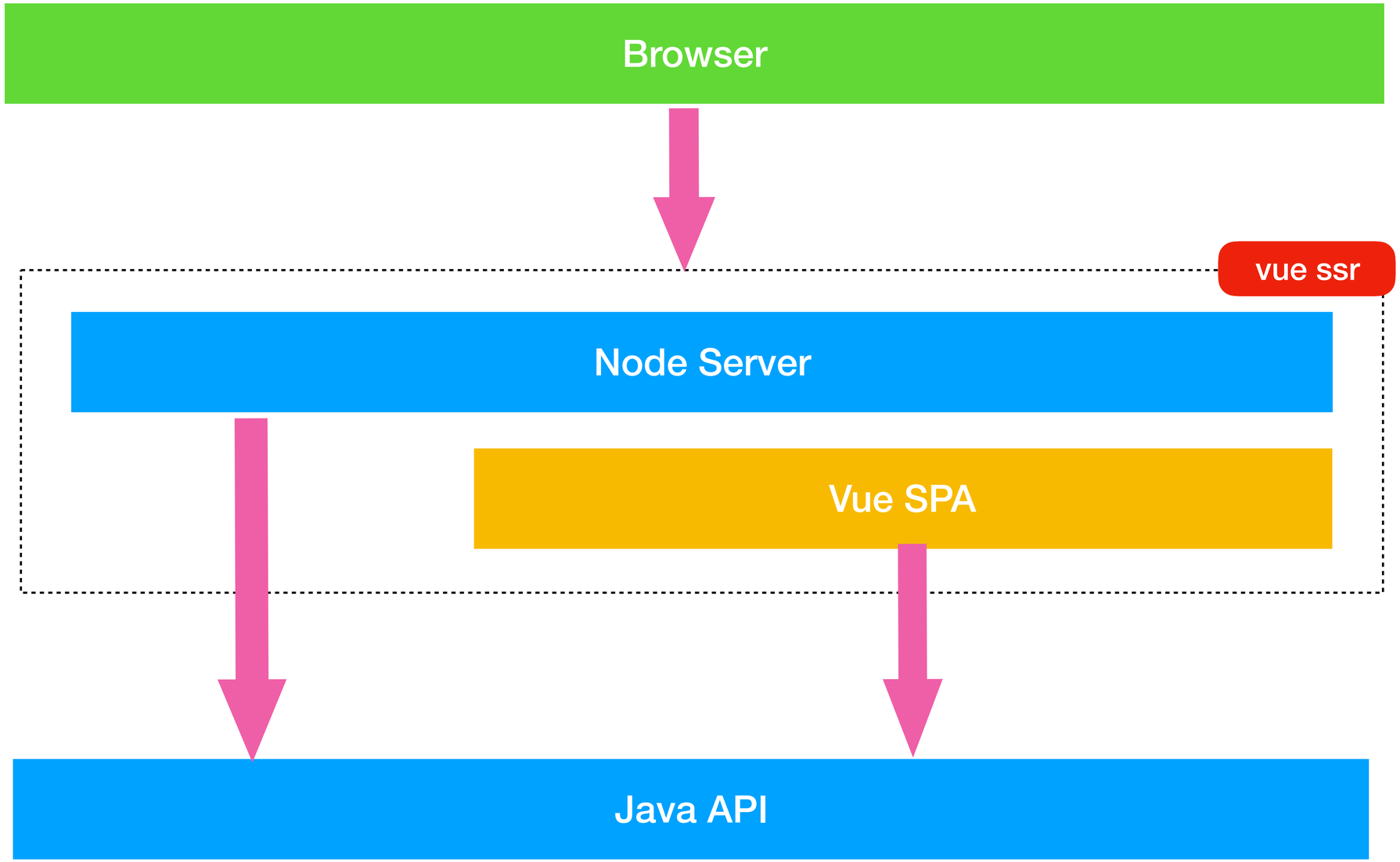
目录

Node.js 在微医的发展

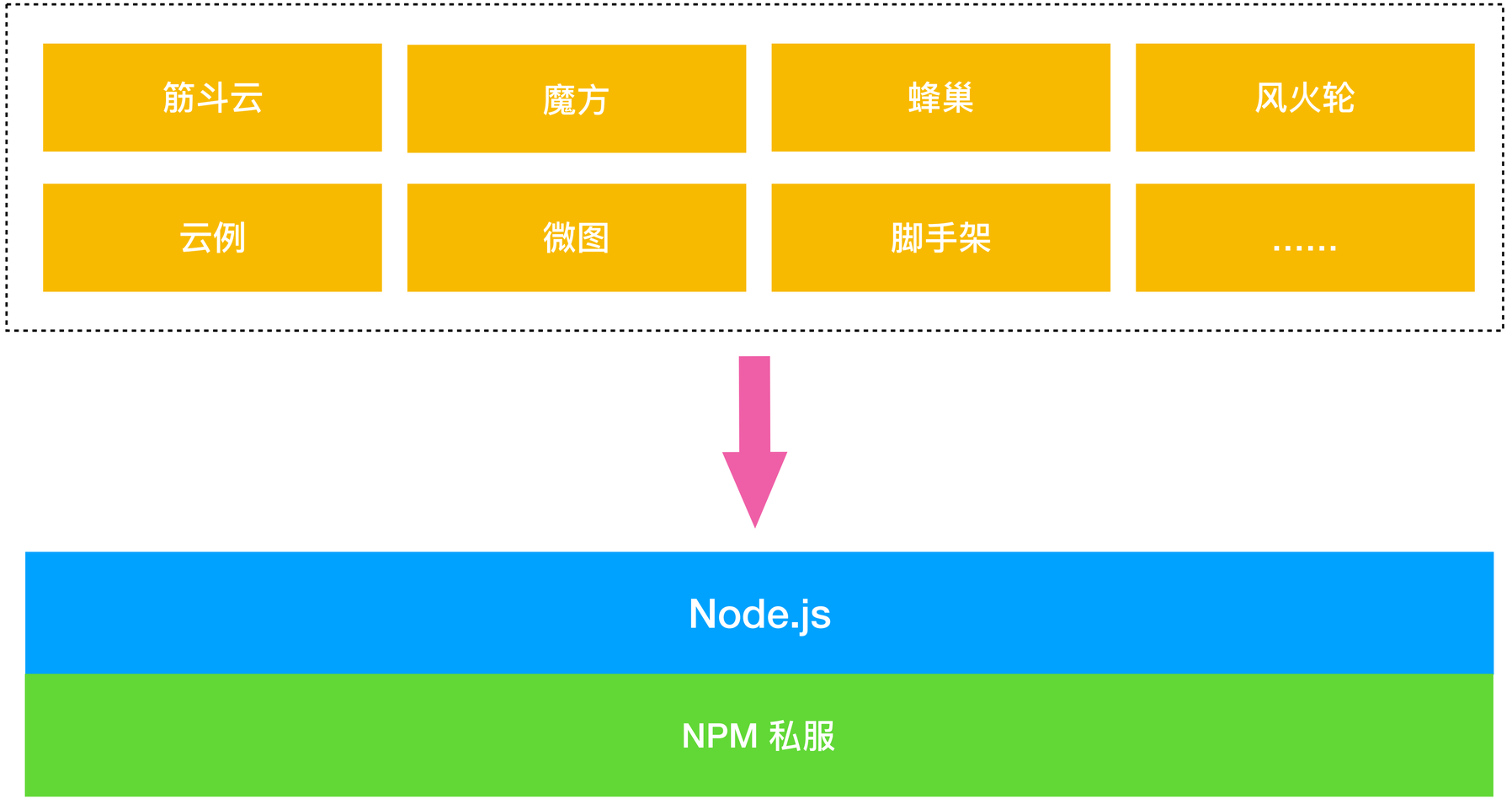
Node.js在微医的发展 - 作为开发模式下的静态资源服务器



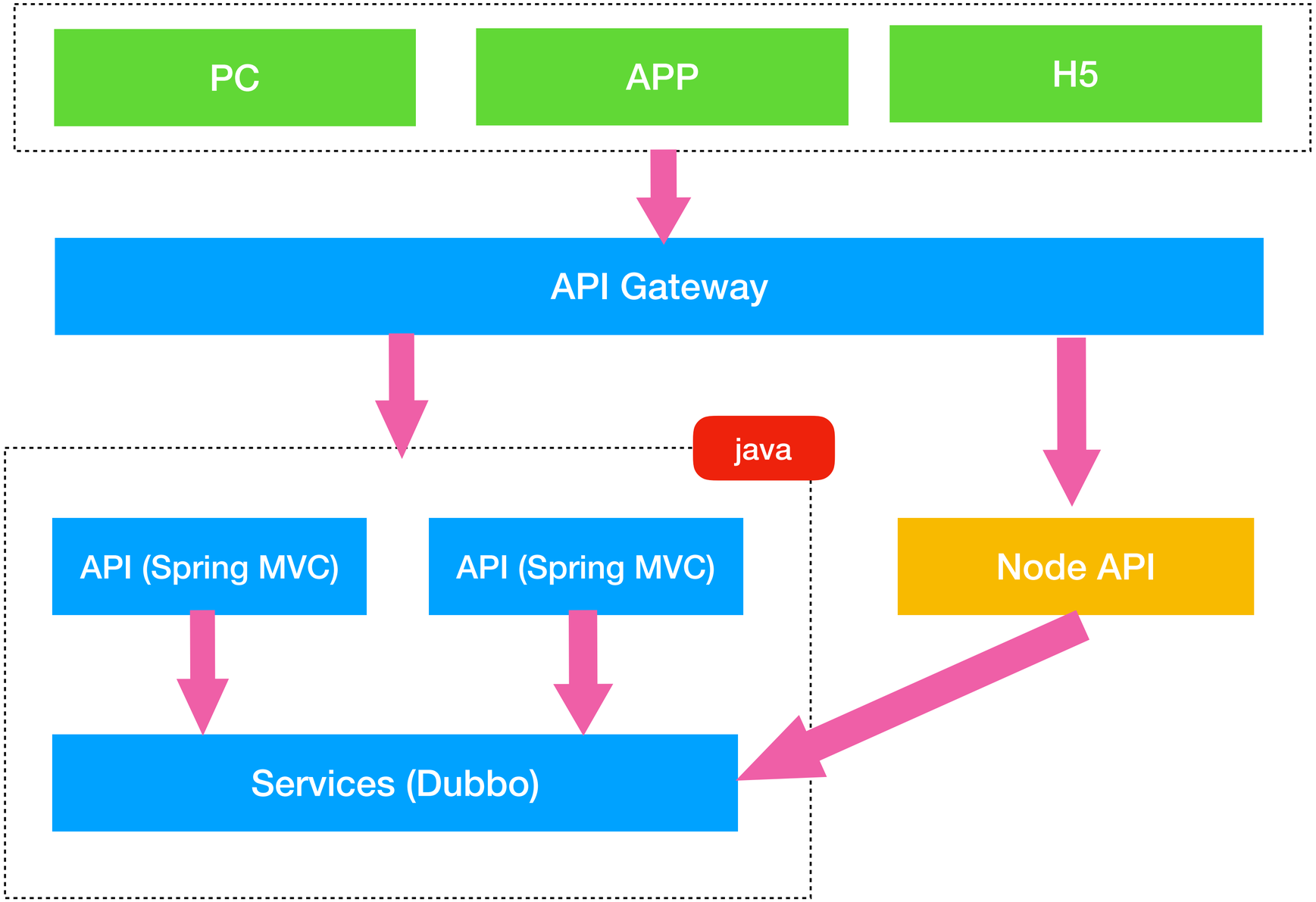
Node.js在微医的发展 - 作为 Vue SSR 的容器



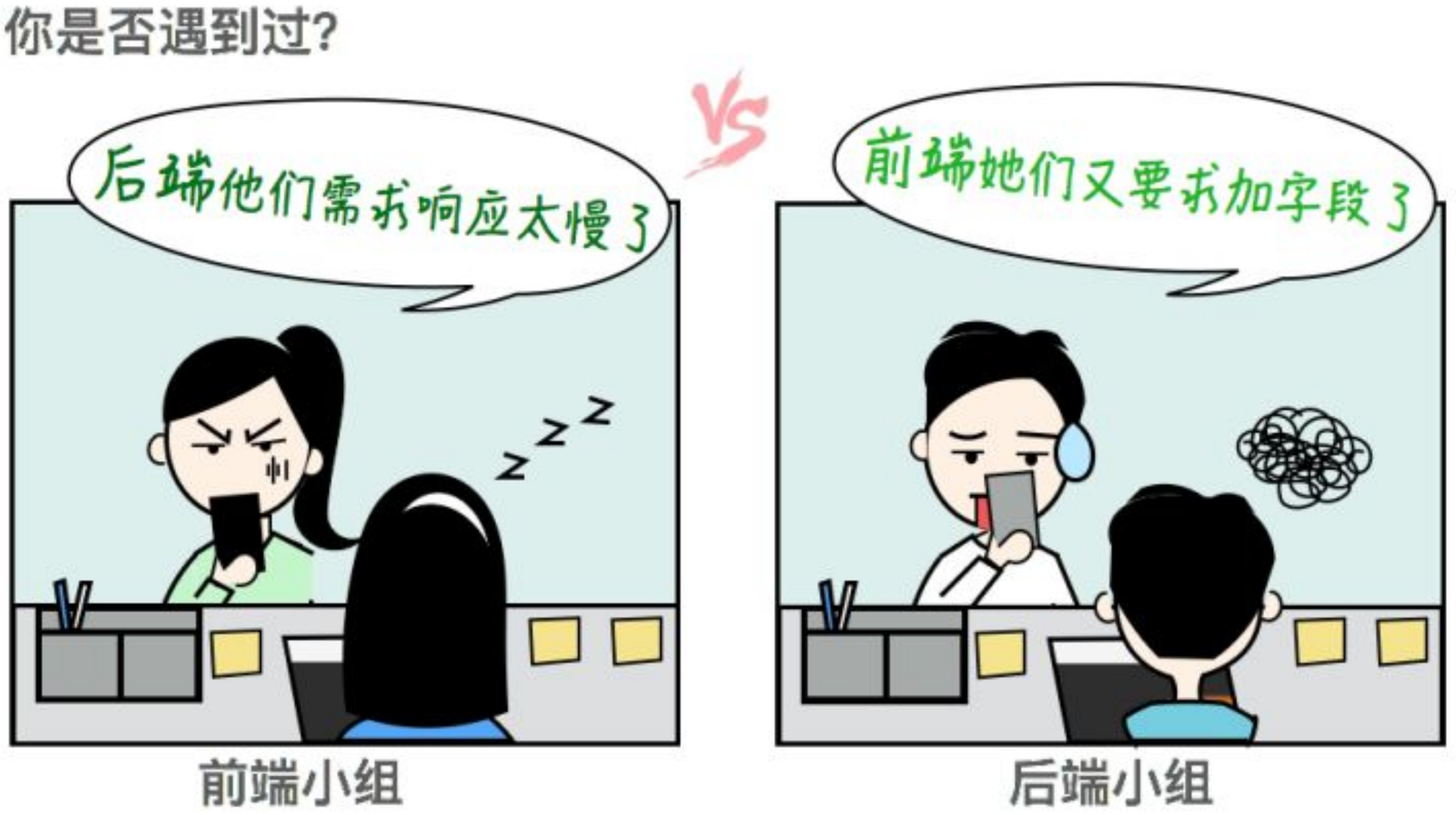
Node.js在用户技术部的发展 - 开发内部效能工具



Node.js在用户技术部的发展 - Node.js真正的开始做一些java的事



为什么使用 Node.js 做 API 层?



- 「你自己请求 2 个接口再组装不就行了？」 - 后端同学追求服务下沉和解耦。
- 「少一次 HTTP 啊，加一个接口有那么难么？」 - 前端同学离用户最近，需要考虑用户体验灵活性。

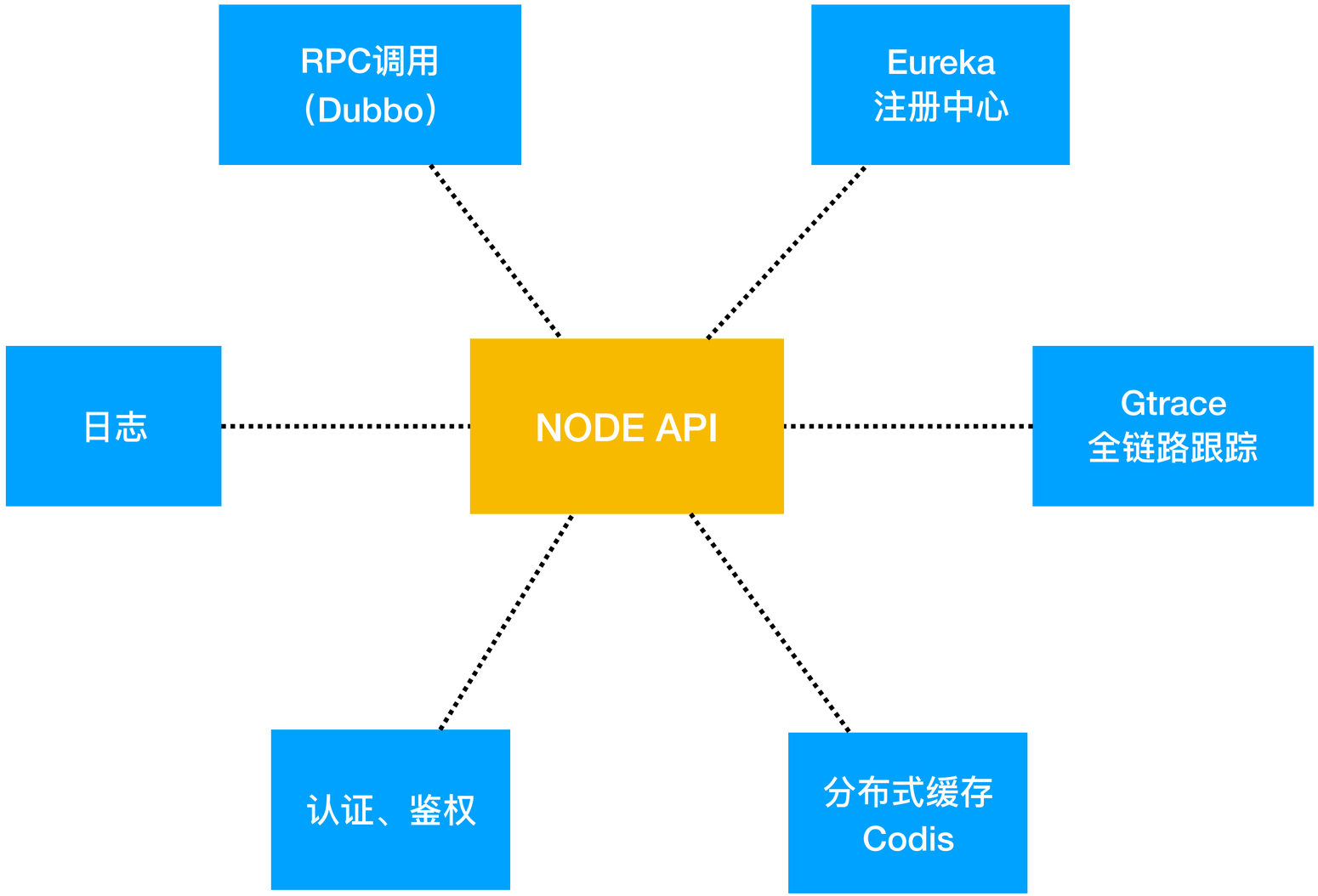
后端同学可能真的不喜欢写 API 层

- 后端资源紧缺
- 数据转换很无聊，没有什么技术含量
- 基于静态语言做数据聚合会不太灵活
- 后端不清楚什么样的数据格式和字段命名是对前端友好的

干脆前端来做这一层吧

前端同学来写 API 层的优势

- 减少没有必要的接口沟通。
- 前端同学可以离业务逻辑更近一步。
- JS天生是JSON友好的，用Node.js做数据聚合很方便。
- JS基于Promise的并行调用很友好。


RPC调用
（Dubbo）
Eureka
注册中心
日志
NODE API
Gtrace
全链路跟踪
认证、鉴权
分布式缓存
Codis

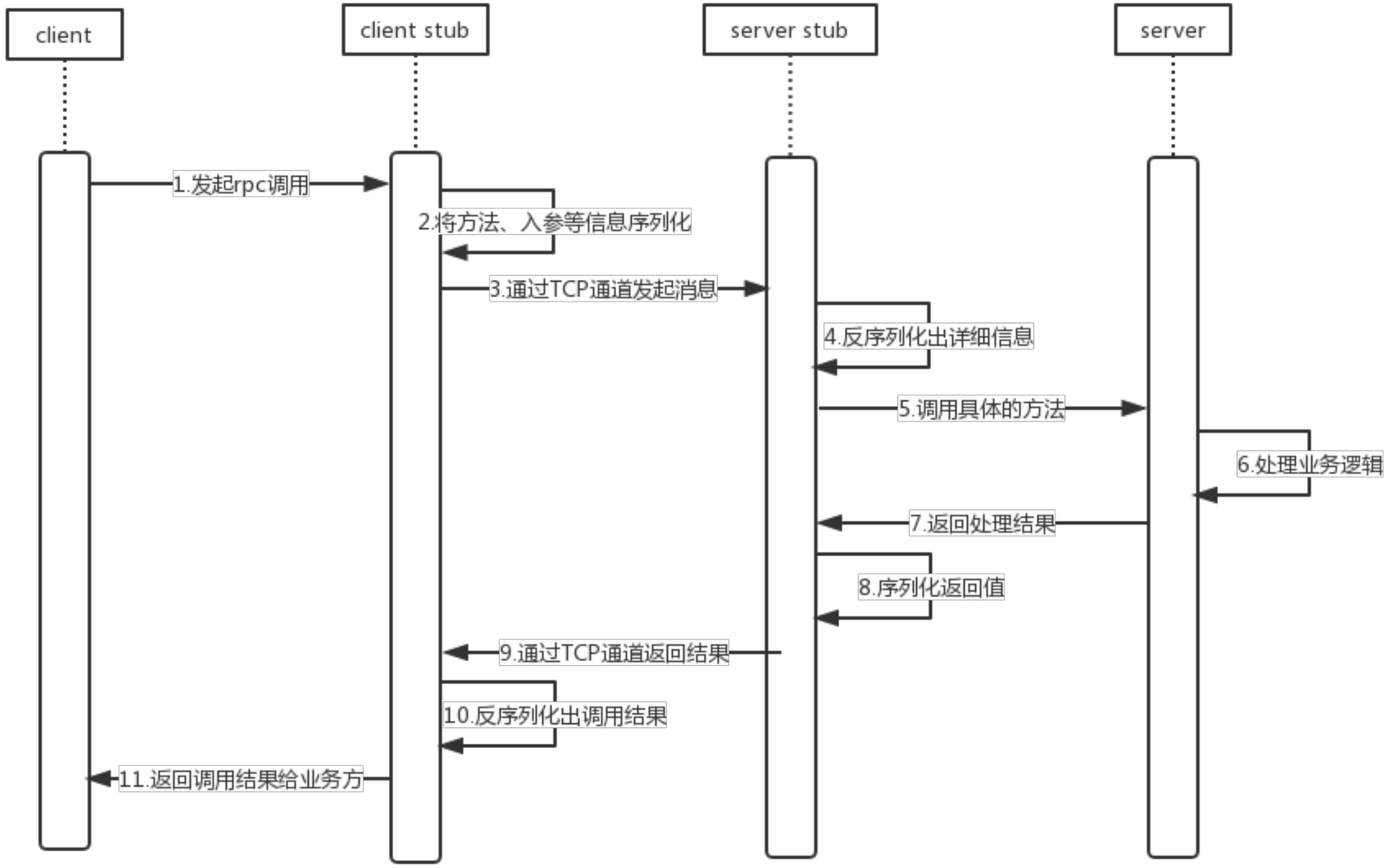
企业级 Node.js 框架 - Egg.js

- 遵循“约定优于配置”，减少学习成本。
- 完善的配套设施，阿里加持，社区活跃。
- 高度可扩展的插件机制，与业务解耦。
- 基于 Typescript，效率更高，开发更爽，错误更少。
- 基于 Koa 中间件，性能优异。

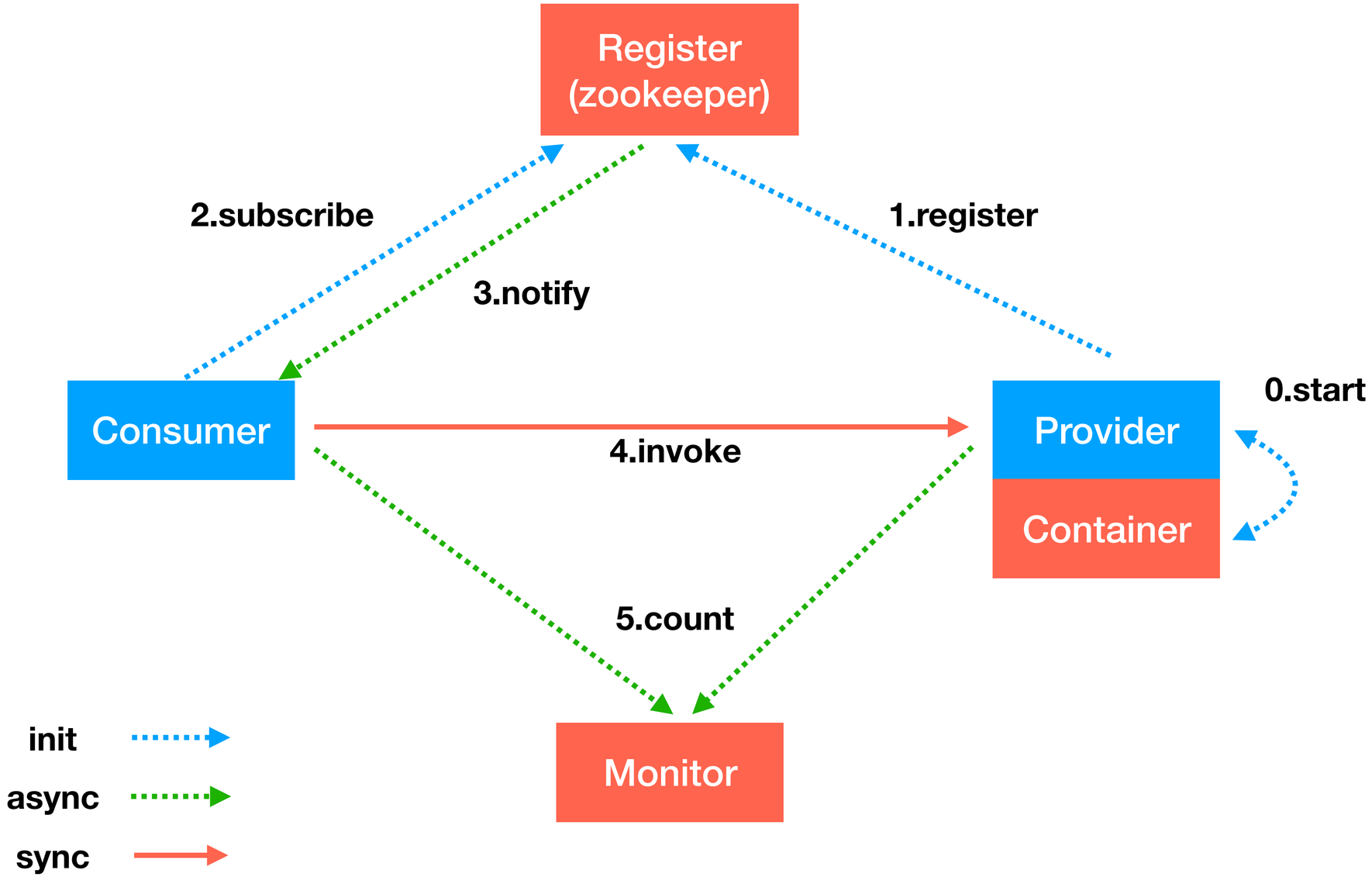
RPC（Remote Procedure Call）

远程过程调用

RPC调用简单时序图


client
client stub
server stub
server
1.发起rpc调用
2.将方法、入参等信息序列化
3.通过TCP通道发起消息
4.反序列化出详细信息
5.调用具体的方法
6.处理业务逻辑
7.返回处理结果
8.序列化返回值
9.通过TCP通道返回结果
10.反序列化出调用结果
11.返回调用结果给业务方

Dubbo简单架构图


Register
(zookeeper)
2.subscribe
3.notify
1.register
Consumer
4.invoke
Provider
Container
0.start
5.count
Monitor
init
async
sync

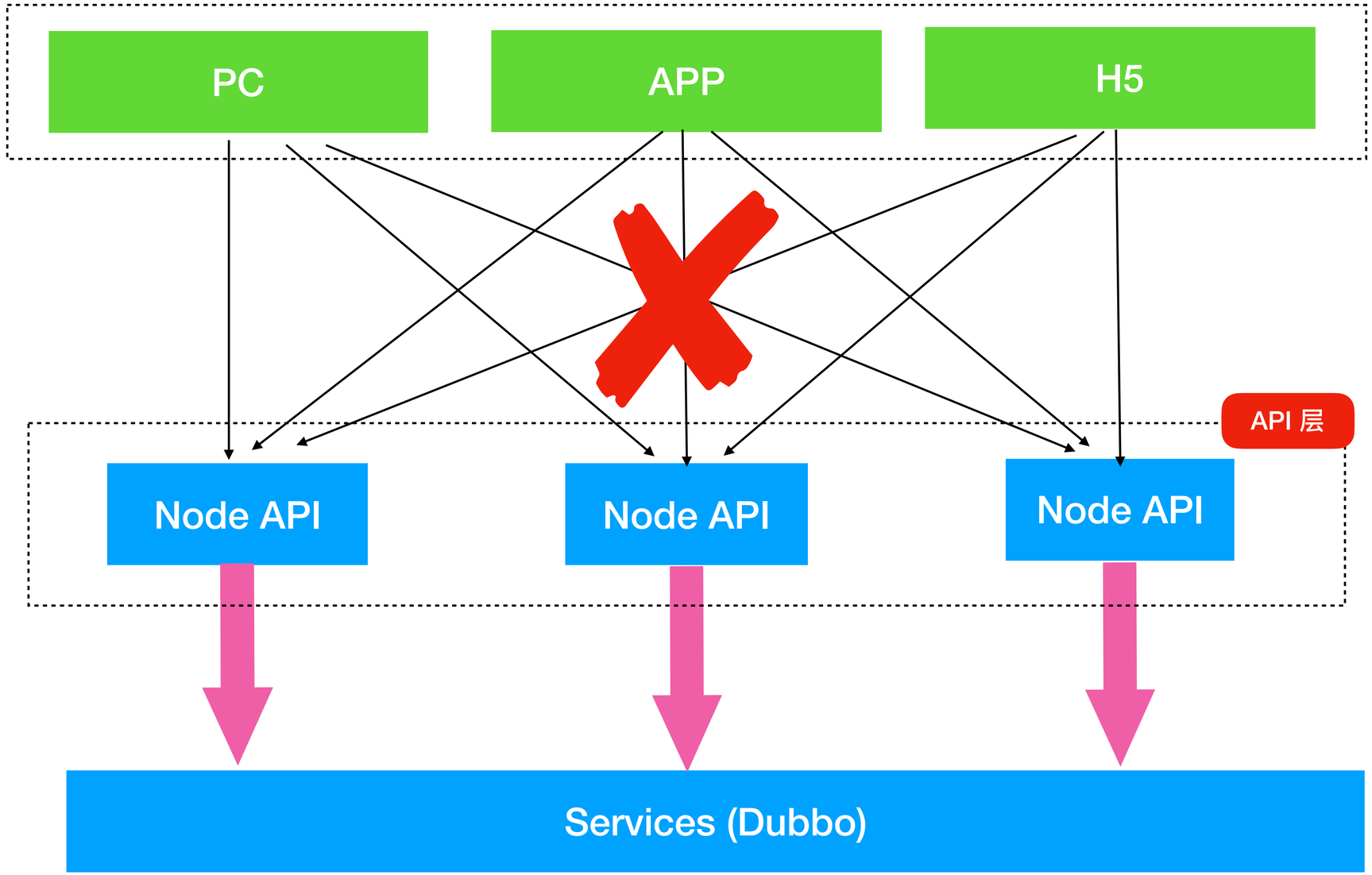
dubbo2.js

- 支持原生的dubbo协议
- 支持dubbo直连
- 支持typescript声明
- 不支持dubbo隔离。

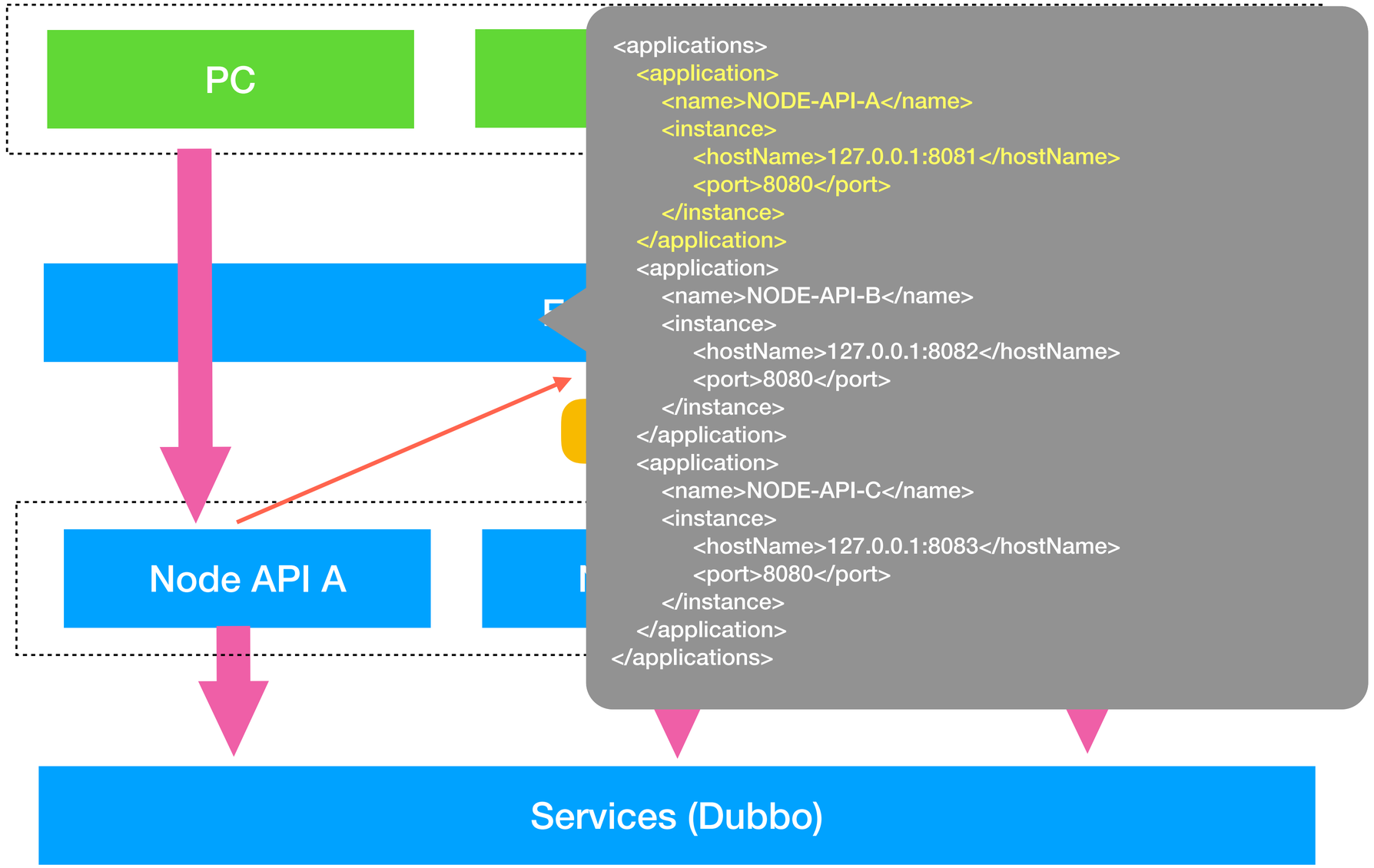
```
import { Dubbo } = from('dubbo2.js')

const testDemoService = (dubbo) => dubbo.proxyService({
  dubboInterface: 'com.alibaba.test.demo',
  version: '0.0.0',
  methods: {
    sayHello: (...params) => params
  }
})

// register service
const dubbo = new Dubbo({
  application: {
    name: 'bff-node'
  },
  register: 'localhost:2181',
  service: {
    testDemoService
  }
});

// dubbo call
(async () => {
  const ret = await dubbo.service.testDemoService.sayHello()
})()
```

Eureka注册中心

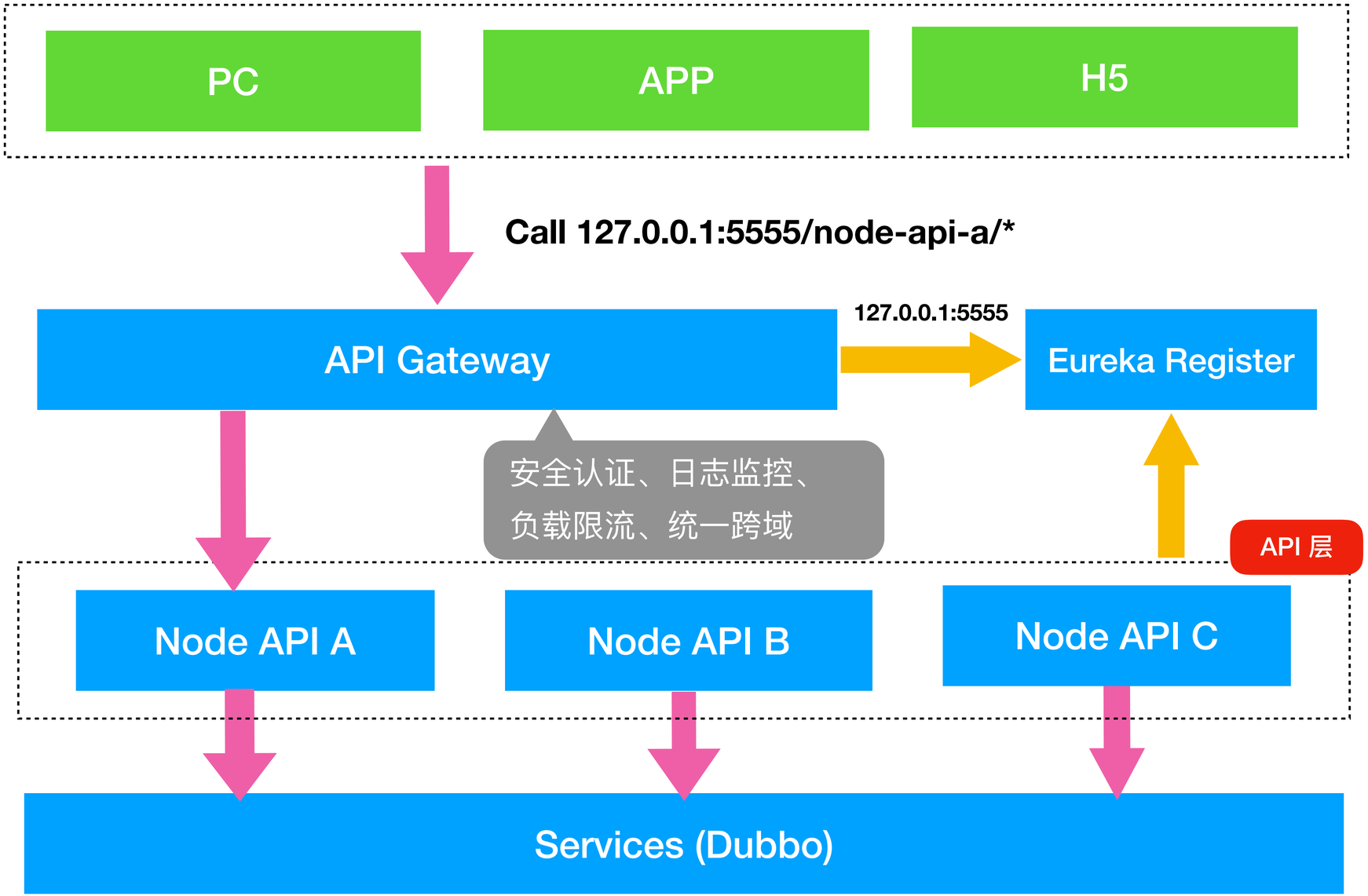
Eureka 注册中心



Eureka 注册中心


PC
<applications>
<application>
<name>NODE-API-A</name>
<instance>
<hostName>127.0.0.1:8081</hostName>
<port>8080</port>
</instance>
</application>
<application>
<name>NODE-API-B</name>
<instance>
<hostName>127.0.0.1:8082</hostName>
<port>8080</port>
</instance>
</application>
<application>
<name>NODE-API-C</name>
<instance>
<hostName>127.0.0.1:8083</hostName>
<port>8080</port>
</instance>
</application>
</applications>
Node API A
Services (Dubbo)

Eureka 注册中心 - API Gateway的作用



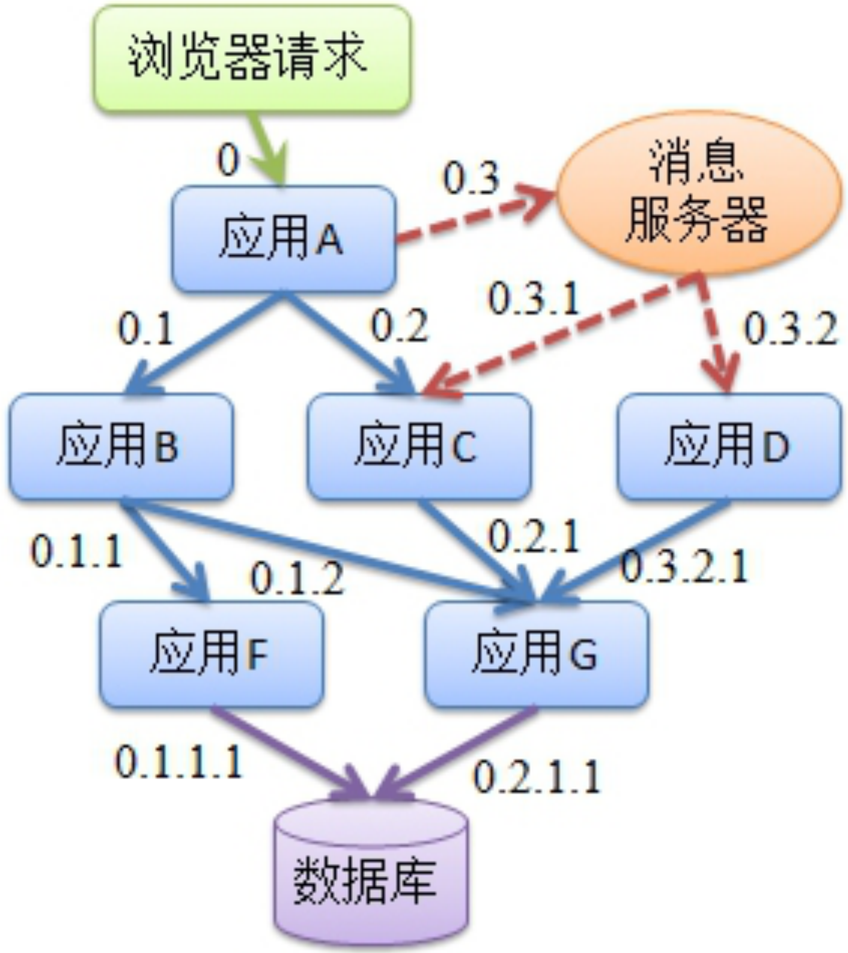
应用接入到 Eureka

```
import Eureka from 'eureka-js-client'

const eureka = new Eureka({
  enable: false,
  instance: {
    app: 'test-node-web',
    dataCenterInfo: {
      '@class': 'com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo',
      'name': 'MyOwn'
    },
    port: {
      '$': 7001,
      '@enabled': true
    }
  },
  register: {
    host: '127.0.0.1',
    port: 9011,
    servicePath: '/eureka/apps/'
  }
})

eureka.start()
```

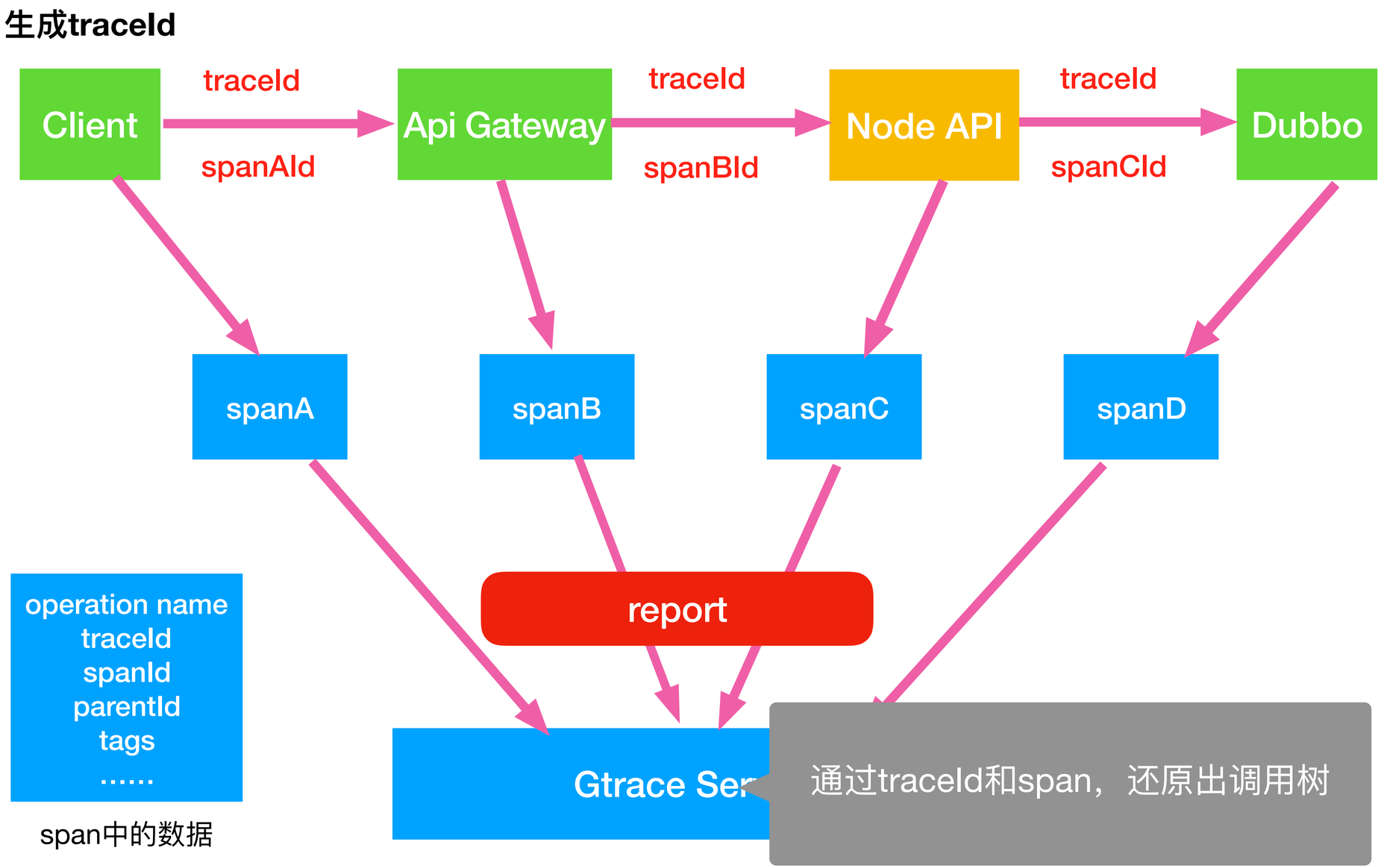
Gtrace全链路跟踪

为什么需要全链路跟踪系统

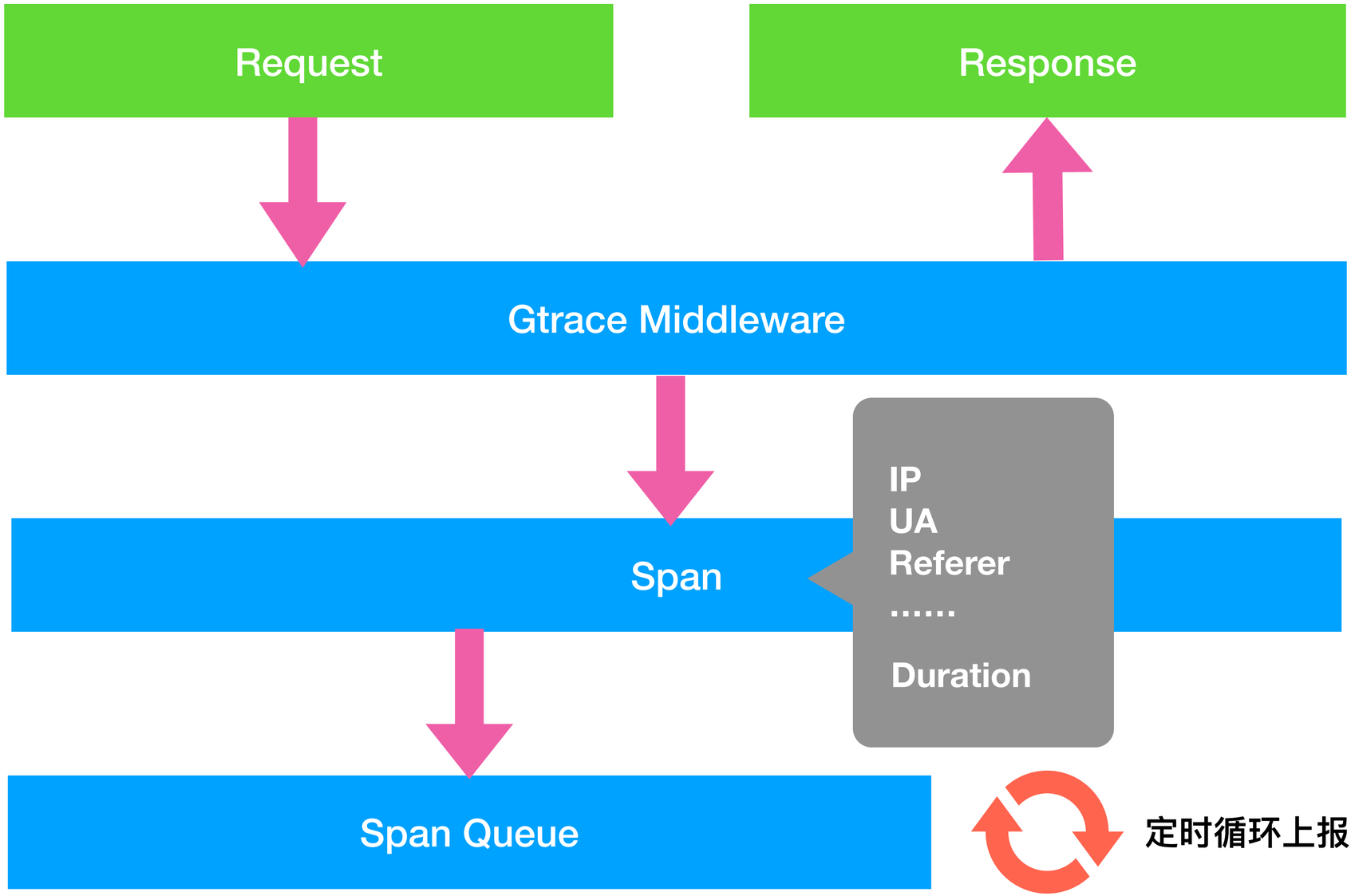
- 微服务系统下应用的调用链路错综复杂。
- 出现异常时，如何快速定位?
- 如何定位应用的瓶颈所在?



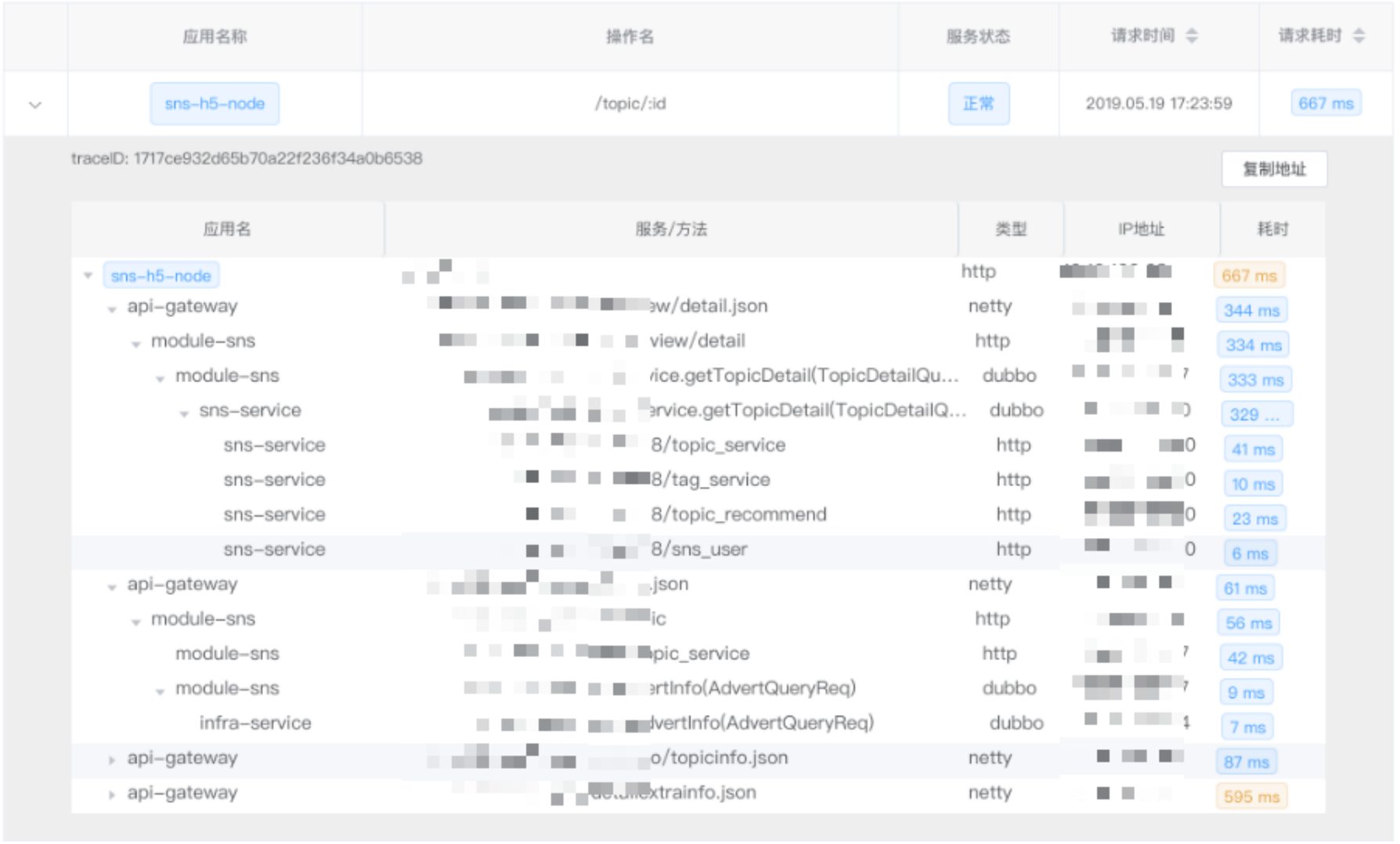
Gtrace全链路跟踪



Gtrace全链路跟踪 - Gtrace Node.js 插件



Gtrace 可视化平台

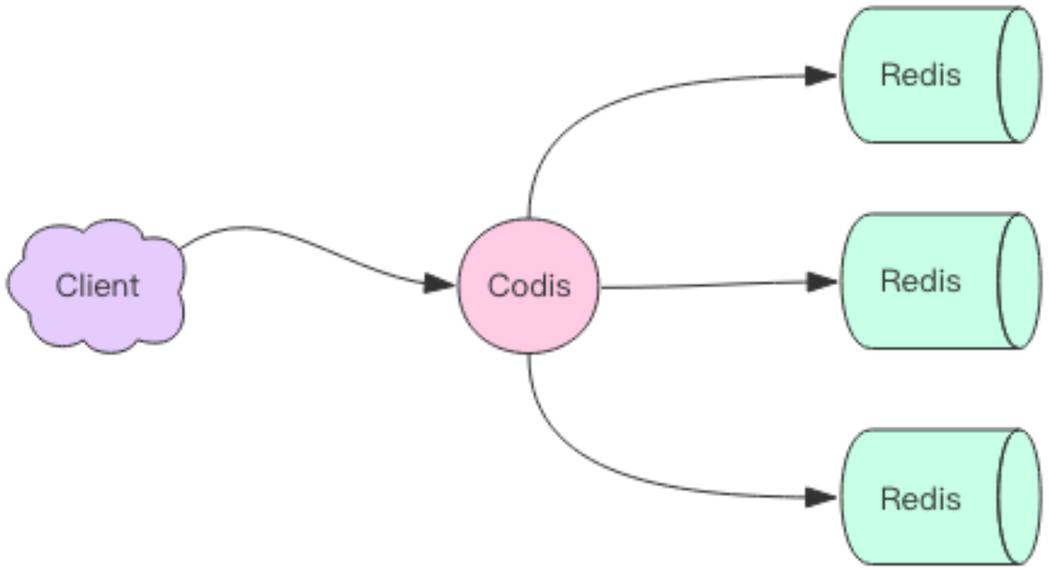


Redis & Codis

Redis的一些缺点

- 单个 Redis 的节点实例，当存储的数据量变大、并发变高时，内存很容易就暴涨。

- 单个 Redis 的节点，内存是受限的。



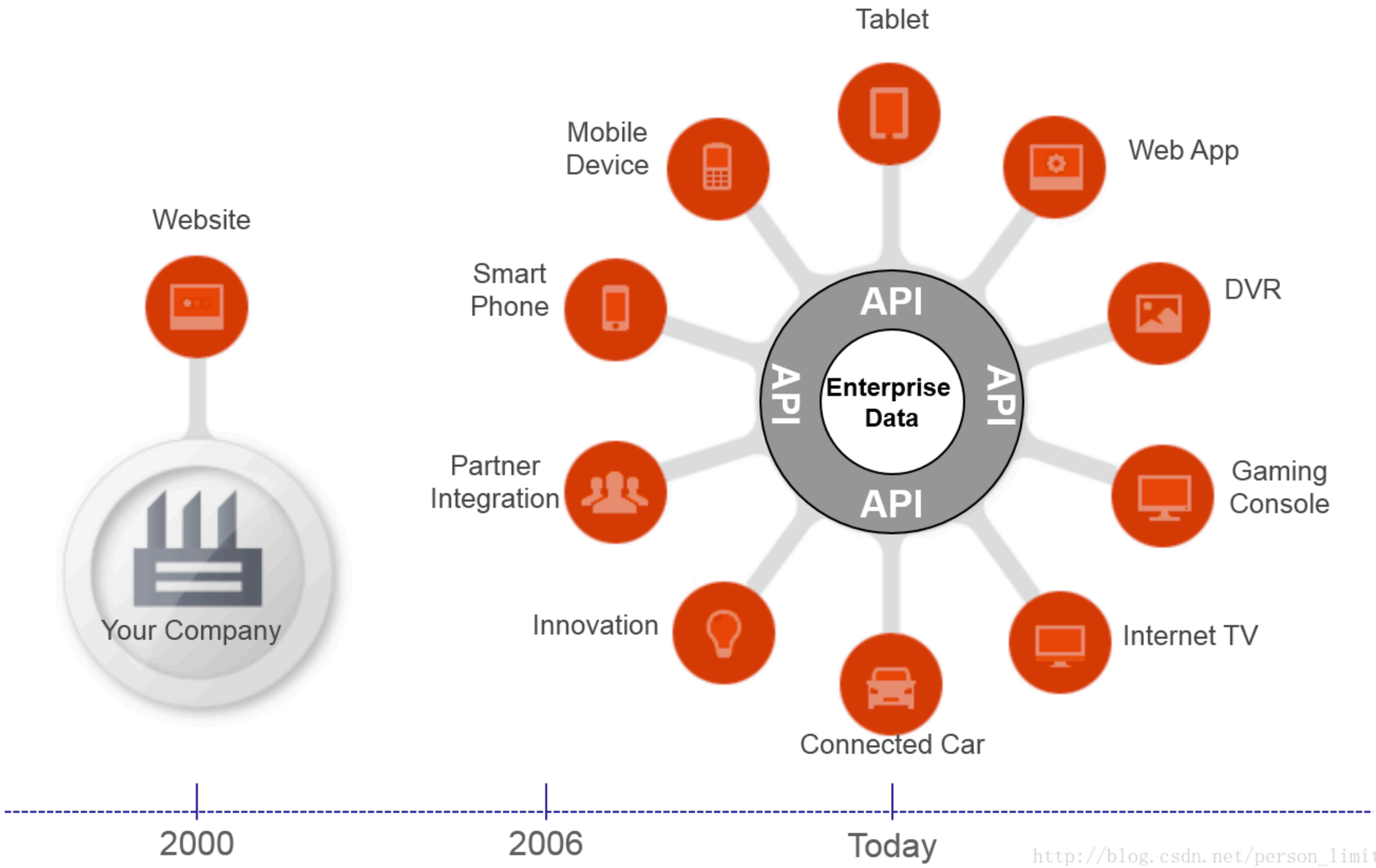
```
const { NodeCodis } = require('node-codis')

const nodeCodis = new NodeCodis({
  zkServers: '127.0.0.1:6701, 127.0.0.1:6702',
  zkCodisProxyDir: '/zk/codis/db_test_node/proxy',
  codisPassword: 'your_codis_password'
})

nodeCodis.on('connected', (err, client) => {
  if (err) {
    console.log(err)
    return
  }

  // Expires after 100 seconds
  client.SETEX('node-codis:test', 100, 'hello world', NodeCodis.print)
  client.GET('node-codis:test', (err, data) => {
    console.log(data) // hello world
  })
})
```

Node-Codis (https://github.com/wefront/node-codis)

我们不生产服务
我们只做服务的搬运工

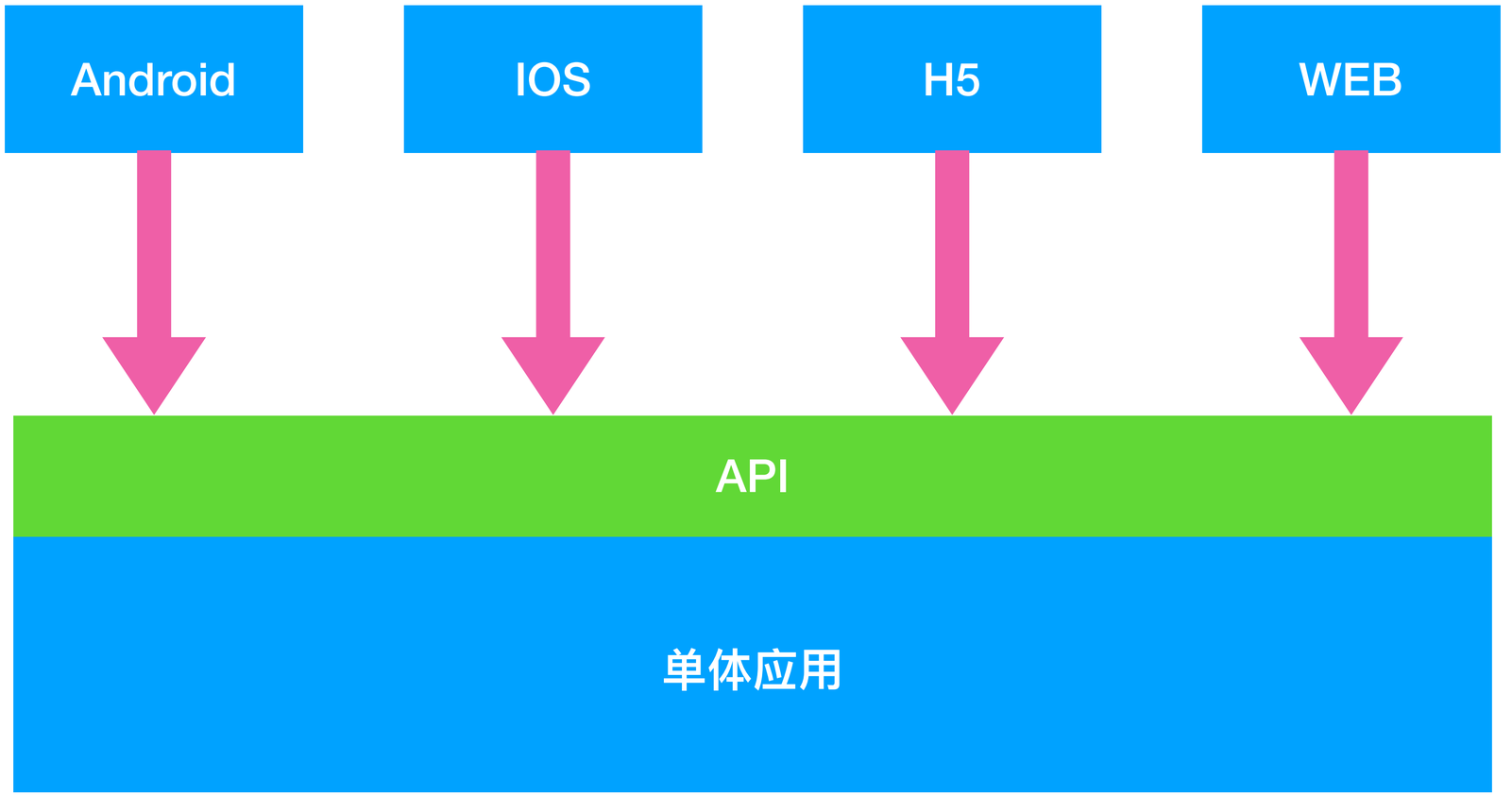
BFF（Backend For Frontend）

服务于前端的后端 || 用户体验适配层



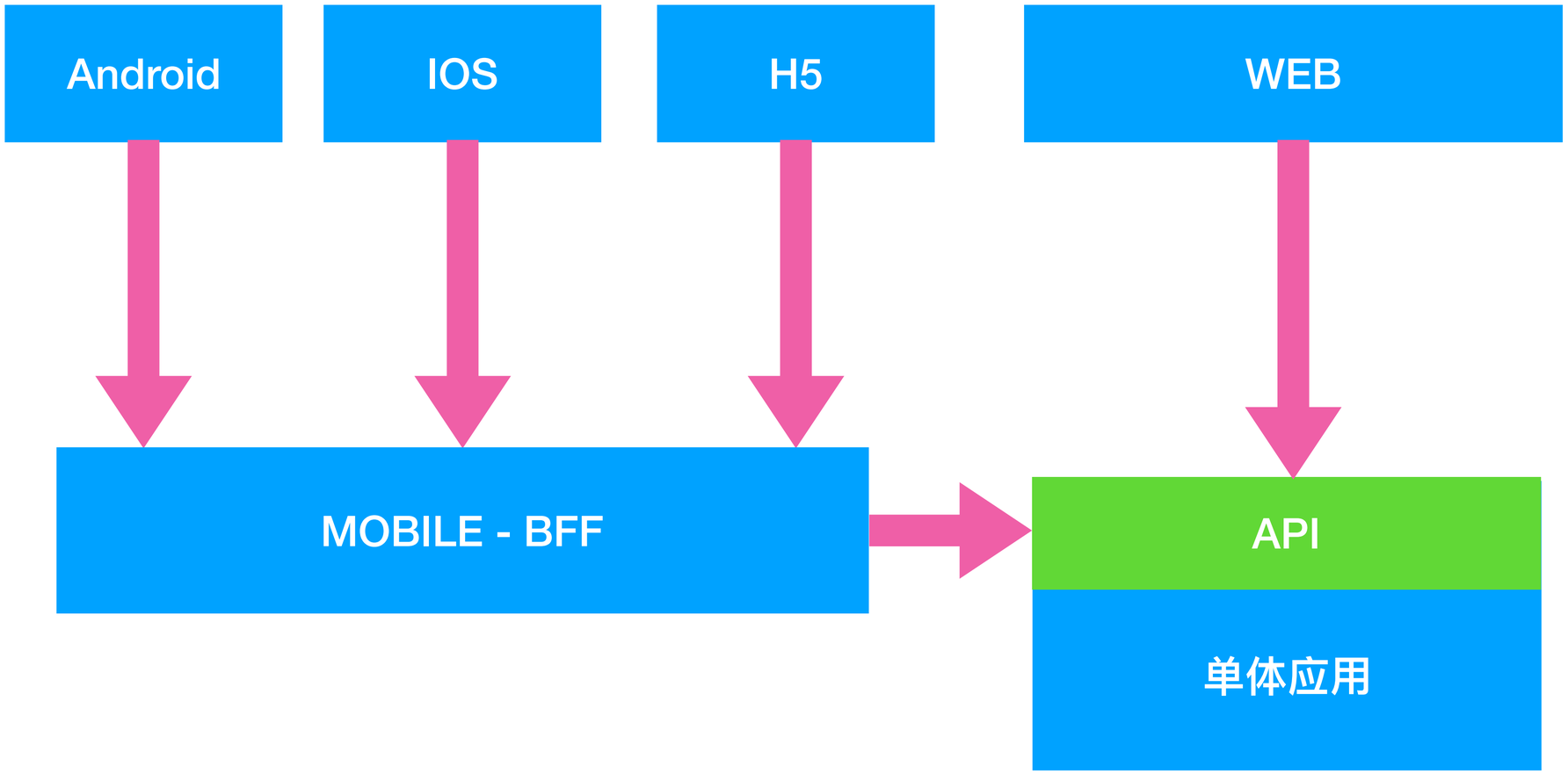
- 互联网应用已经从单一Web浏览器时代演进到以API驱动的**无线优先**和**面向全渠道体验**时代。

BFF的发展



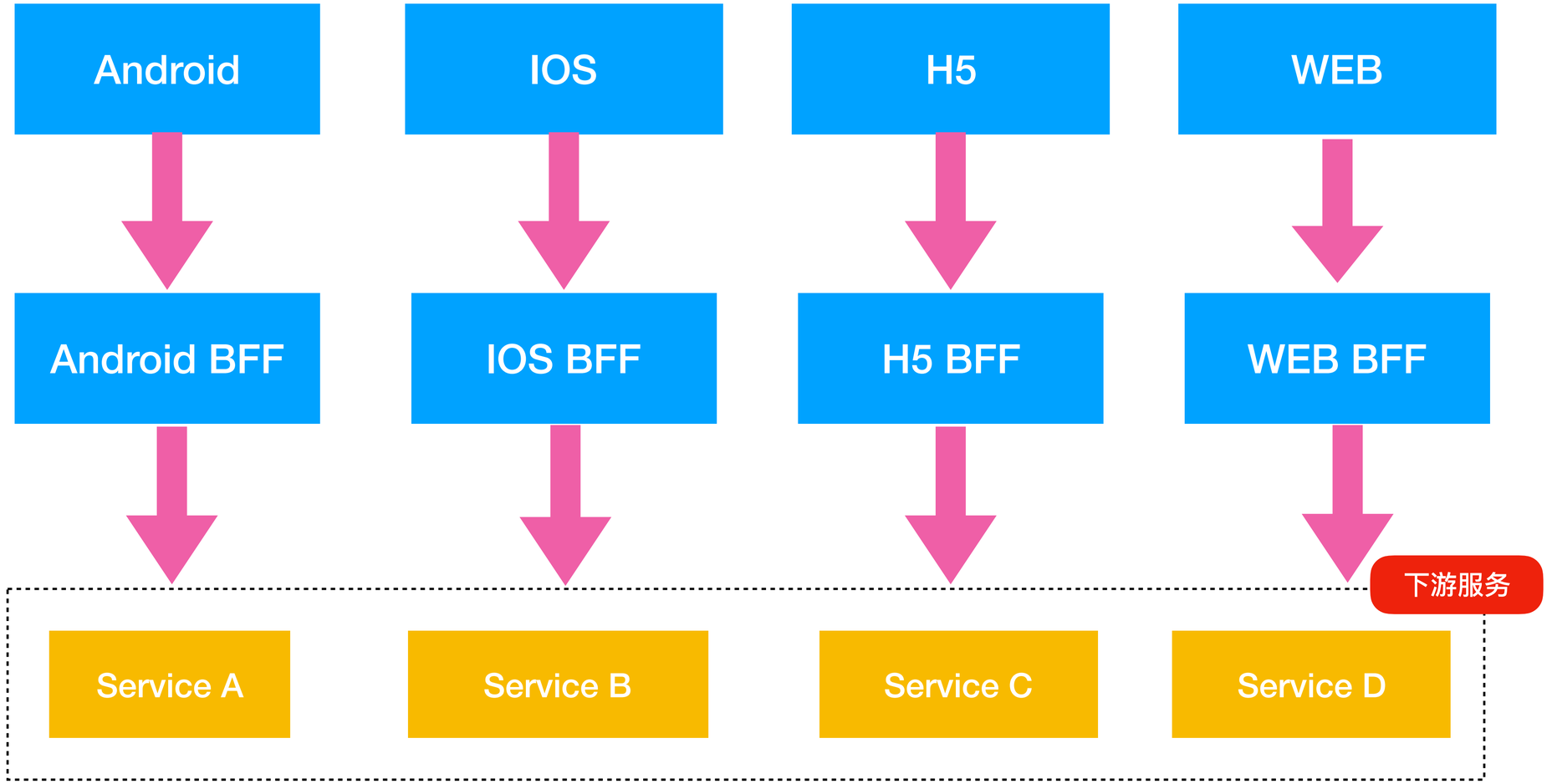
- 面向通用服务的API无法应对复杂的多客户端架构

BFF的发展



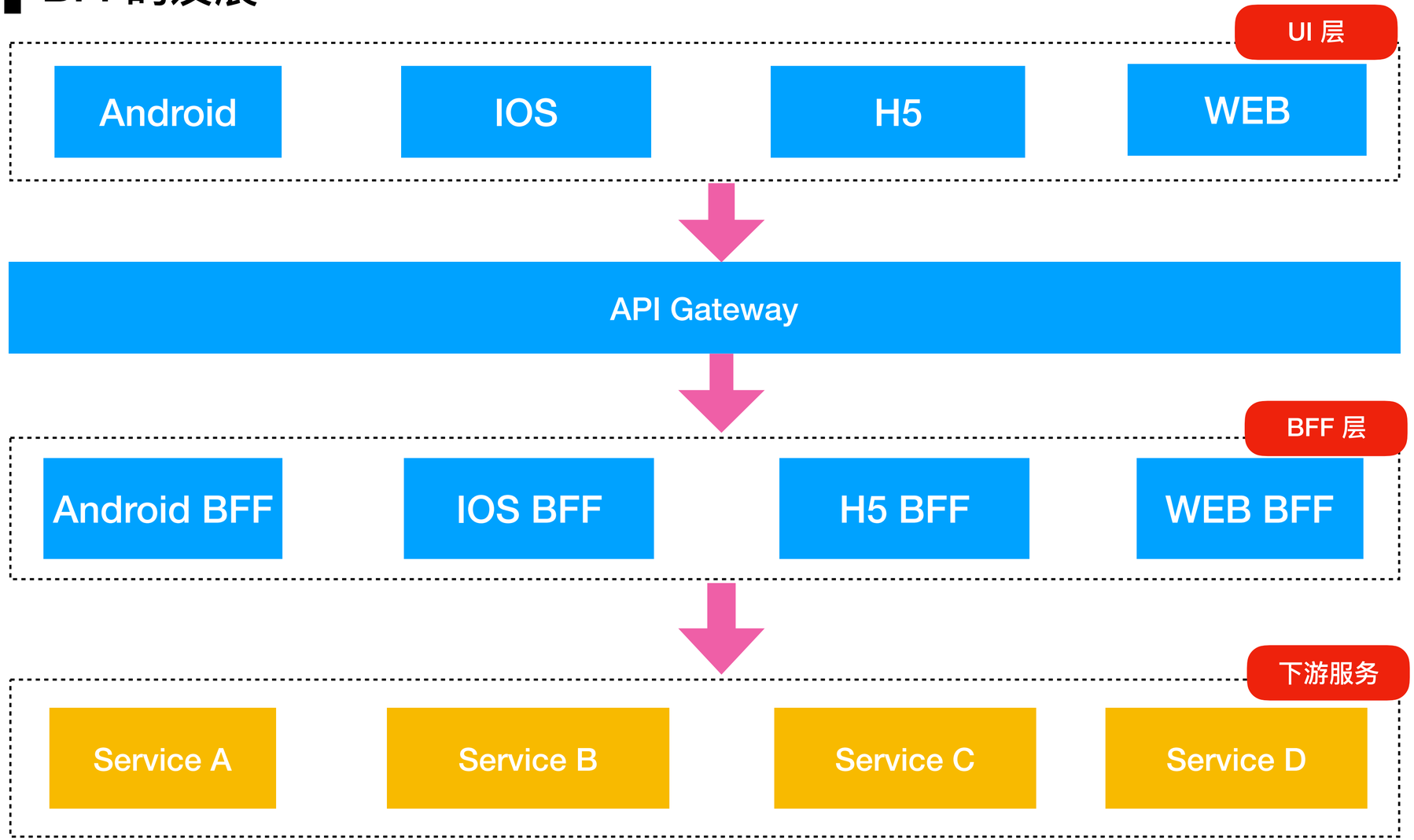
- MOBILE - BFF层调用后端API，再进行裁剪、聚合、适配，输入面向移动设备的接口。

BFF的发展



- 理想情况下对于每一个用户体验层都有一个对应的BFF。

BFF的发展



- 加一层独立网关负责公共的横切面逻辑（安全认证，限流监控...）

革命尚未成功
同志仍需努力

End, Q & A