

关于我

九十 (a.k.a 王光) @淘宝
Twitter: @GuangWong

2014 年开始在淘宝 FED 做一些 Node.js 相关的工作
现在手头在做一些 Node.js 全栈项目和底层容器的工作



Node.js 服务前端数据接口的演进

九十 @ 淘宝

我们经历了什么

多终端开始爆发



桌面 Web 时代

2014

BFF 的思想开始萌芽

2015

中心化的视图数据服务

2016

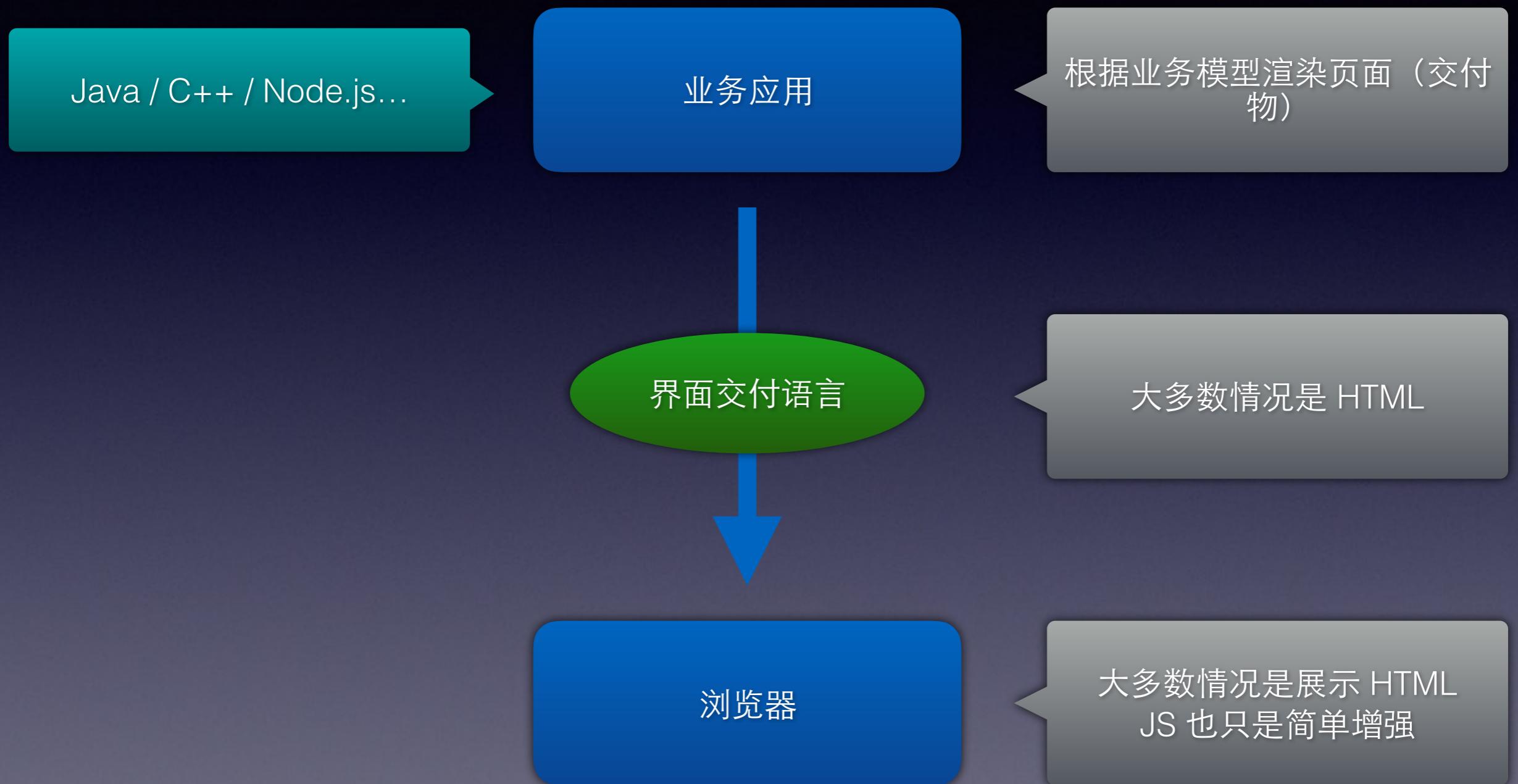


桌面 Web 时代

我们赶上了一个时代的尾巴

拱宸桥 (1631年) @ 杭州 / 京杭大运河的终点也是起点

这个时代的特点



我们是怎么应用 Node.js 的

Java / C++...

业务应用-业务逻辑

根据业务需求交付数据接口

也许 JSON Over HTTP
也许 Hessian Over HSF

纯粹数据

对数据做约束、定义 Schema
增强协作、异步开发效率

前端在服务器和客户端的基础技术方案得以统一

业务应用- Node.js 部分

拼装（渲染）HTML

界面交付语言

浏览器

所谓半栈

终端开始爆发



多端不止是端种类的爆发
更是端能力的爆发
一不小心把 HTML 革命没了



weex

都没有界面交付语言... 半栈玩不下去了...



BFF 的思想开始萌芽

多端时代的开始

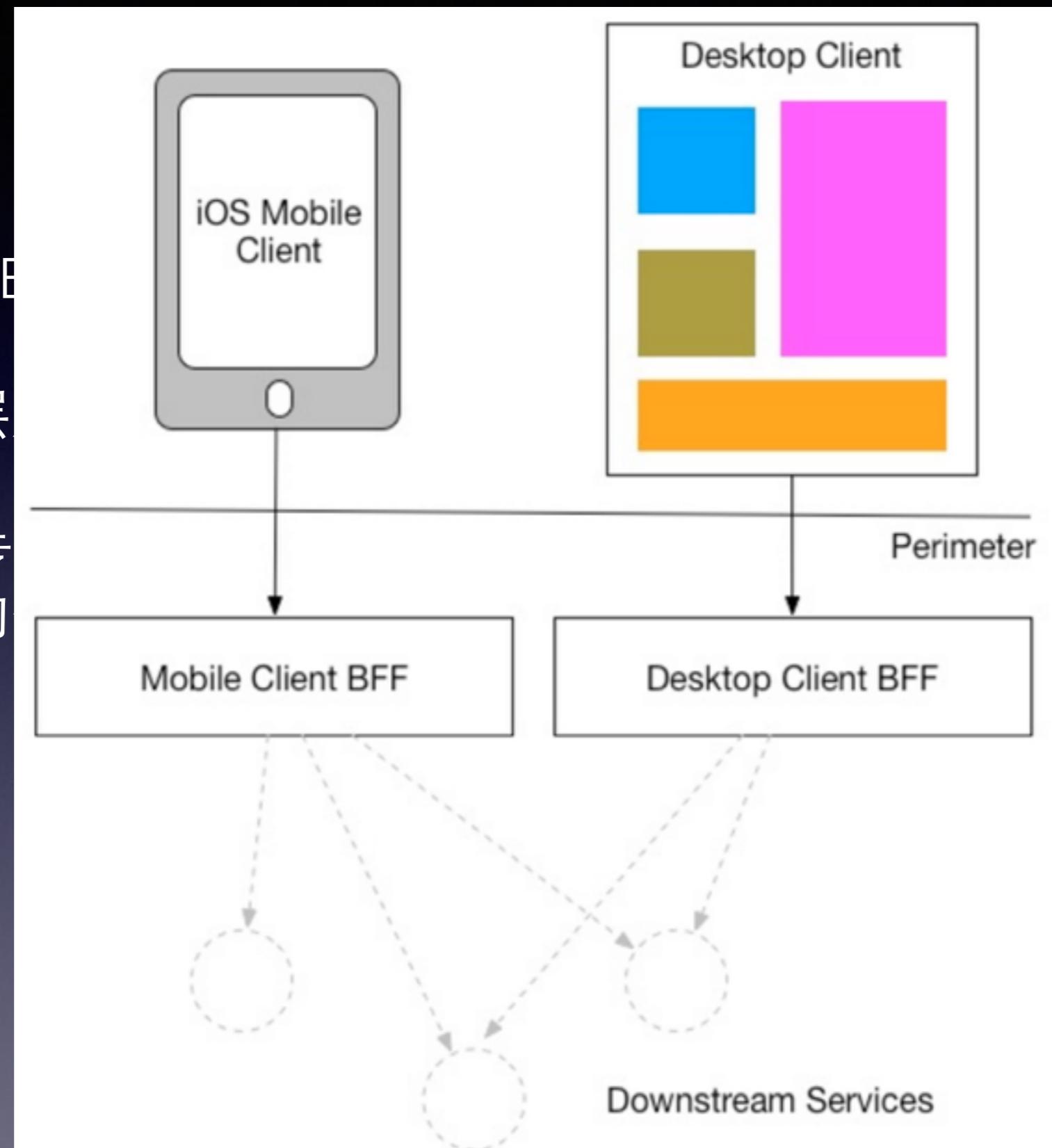
Iron Bridge (1779年) @ UK / 世界首座铸铁大桥，结构却是木工的结构

术语解释 BFF (Backend for Frontend)

1. BFF (Backend for Frontend) 意为前端专设一个服务端
2. 从底层服务抓取数据、组装、裁剪，然后提供给前端
3. BFF 专为前端服务，要能够快速响应前端的变化，提供各种各样的数据

术语解释

1. BFF (Backend for Frontend)
2. 从底层
3. BFF 专
各样的

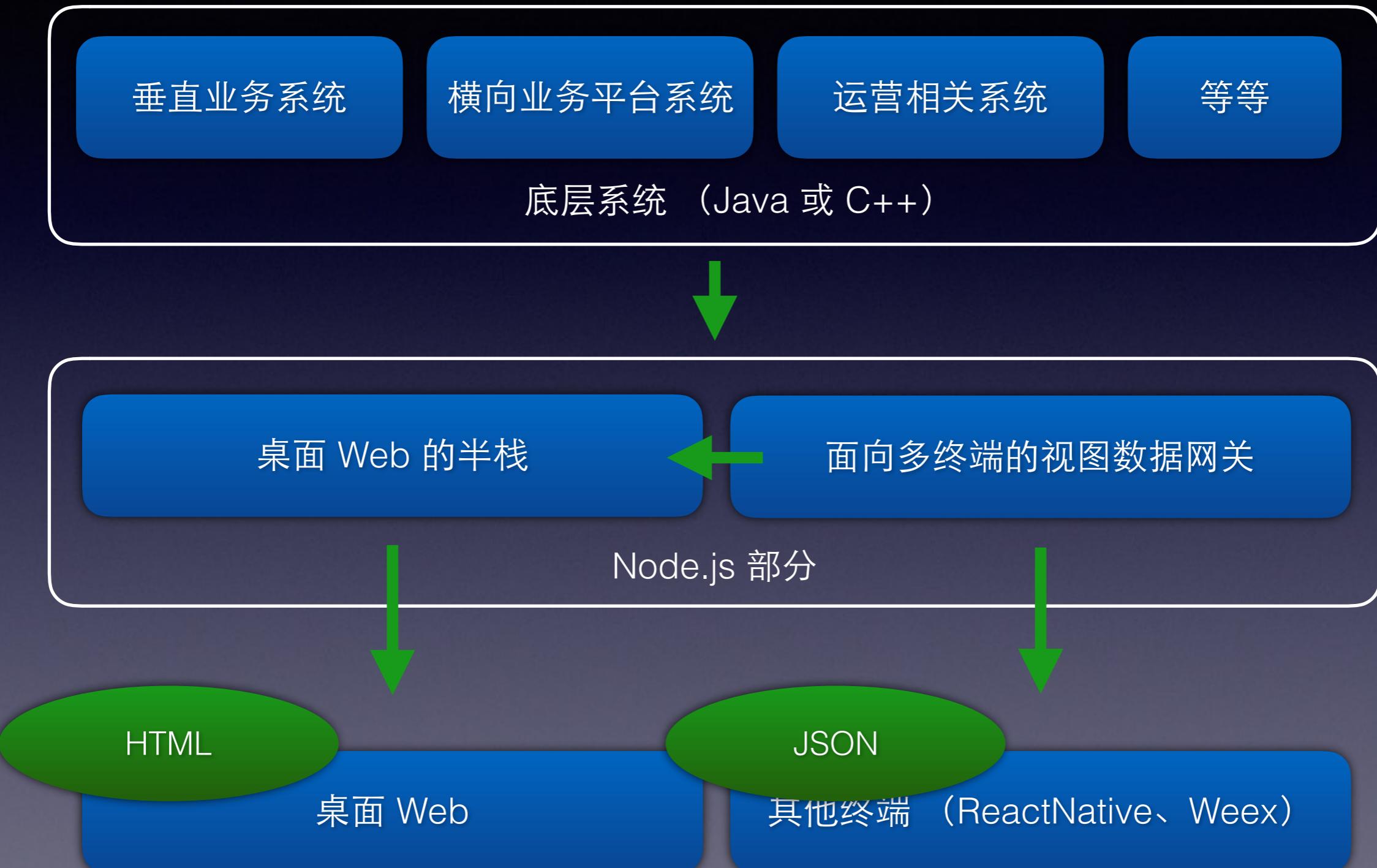


服务端

端

提供各种

BFF 开始萌芽



BFF 开始萌芽

垂直业务系统

横向业务平台系统

运营相关系统

等等

底层系统 (Java 或 C++)

桌面 Web 的半栈

Node.js 部分

HTML

桌面 Web

怎么调用?

阿里巴巴内部早有一个服务框架



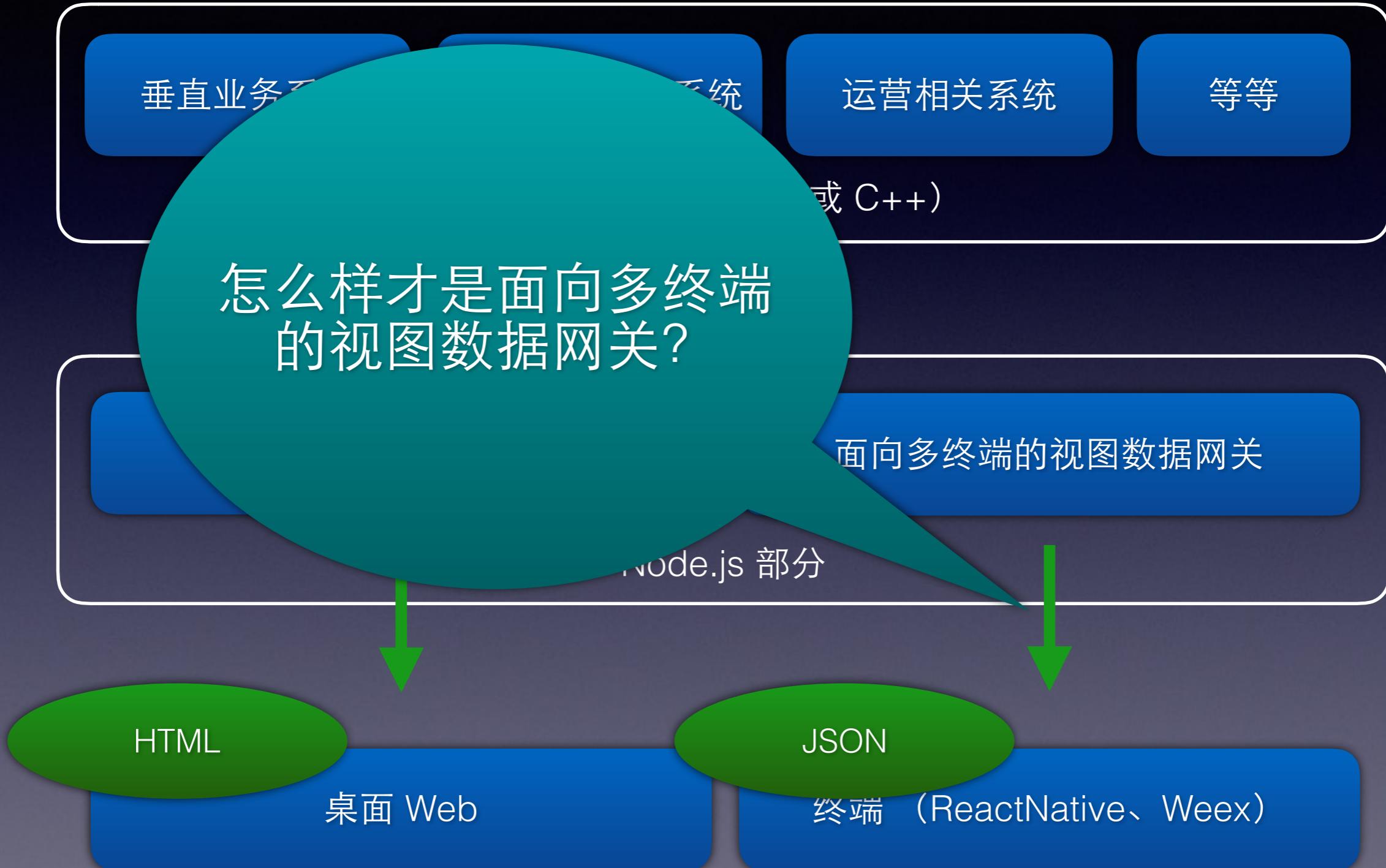
基本上也覆盖了多语言调用的问题

- 早期 Node.js 有 C++ 版本的 Binding
- 现在也有纯 Node.js 的版本

不过依然有一个小问题

- 那些后端应用大部分都是 Java 的，用的最多的序列化是 Hessian !
- Node.js 想调用 HSF 需要手拼 Hessian !
 - Java 反射拿到的类型信息自动做了 Mapping
 - 分析 Jar 自动做了 Mapping
 - 所以也算解决了

BFF 开始萌芽



面向多终端的数据网关

一些尝试：



GraphQL

- 良好的查询语言
 - 对多终端的碎片化版本支持很赞
 - 和 React 体系兼容性好
- 完善的大问题建模语言
 - 可以实现将多个系统、多个领域的模型统一表示
 - 客户端只需要使用简单的查询语言便可覆盖各种问题



了解更多请至淘宝 FED 博客

面向多终端的数据网关

一些尝试：



GraphQL

- 良好的查询语言
- {
 - 对多终端的碎片化版本支持很赞
 - react 体系兼容性好
 - 完善的大问题建模语言
 - 可以实现将多个系统、多个领域的模型统一表示
 - 可以实现将多个系统、多个领域的模型统一表示
 - width,
 - height
- } 客户端只需要使用简单的查询语言便可覆盖各种问题
- }
- }



了解更多请至淘宝 FED 博客

但是

GraphQL 对下层的数据体系的领域模型抽象要求很高

一些简单的方案比如

JSON-RPC

覆盖这个问题成本更低

不过能沉淀下的价值也相应变小

这就要看投入产出了

当一个应用的

桌面 Web 的半栈

被彻底拿掉

就剩一个包数据的网关了

每个业务搞一个集群包视图数据？







开始尝试
中心化的视图数据服务

中心化的视图数据服务

就像 55144 根钢丝吊起来的金门大桥

金门大桥 (1937年) @ 旧金山 / 著名的悬索桥，20 世纪桥梁工程奇迹

中心化的视图数据服务

垂直业务系统

横向业务平台系统

运营相关系统

等等

数据源层：统一收拢接入

业务脚本沙箱

CDN 容灾

限流策略

面向多端的网关

异常警报

慢接口治理

中心化的视图服务 Powered By Node.js

JSON

多终端 (ReactNative、Weex)

问题：业务脚本沙箱

业务脚本沙箱

- 安全执行第三方脚本
- require('vm').runInNewContext()

Resolved

高可用 · 服务脚本小结

Concurrency Level:	100	请求信息
Time taken for tests:	31.543 seconds	
Complete requests:	1000	
Failed requests:	0	
Total transferred:	75000 bytes	
HTML transferred:	0 bytes	
Requests per second:	31.70 [#/sec] (mean)	
Time per request:	3154.342 [ms] (mean)	
Time per request:	31.543 [ms] (mean, across all concurrent requests)	
Transfer rate:	2.32 [Kbytes/sec] received	

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	1	1.5	0
Processing:	57	3000	572.0	3074
Waiting:	55	3000	572.0	3074
Total:	64	3001	570.6	3669

PID	COMMAND	%CPU	TIME	#TH	#WQ	#POR	MEM	PURG	CMPR	PGRP	PPID
37997	node	0.0	00:30.54	7	0	29	853M	0B	0B	37997	24163

问题：业务脚本沙箱

src/node_contextify.cc

```
Persistent<Context> context_;
```

基本只有在 **Full GC** 时才能清理

1. 然而 Node.js 分代内存管理策略，让费时费力的 Full GC 很少使用（2/8 法则是 GC 领域非常重要的发现）
2. 弱减少 new-space-size 的大小倒是能频繁触发 Full GC 可程序效率就感人了
3. 如果 keep 较大的 new-space-size，进行一次大型的 Full GC 也感人
4. 所以怎么样都不靠谱



了解更多请至淘宝 FED 博客

问题：业务脚本沙箱 那回到 PL 的思路来处理吧

```
const ds = require('@ali/ds');
const moment = require('moment');
const result = ds.invoke('setNotice', {
    time: moment().add(1, 'days').format(),
    message: $params.message,
    userId: $context.userId
});
module.exports = result;
```

问题：业务脚本沙箱 那回到 PL 的思路来处理吧

@ali/sealing

```
const ds = __sealing_unbinding('require', typeof require !== 'undefined' ? require : undefined)('@ali/ds');

const moment = __sealing_unbinding('require', typeof require !== 'undefined' ? require : undefined)('moment');

const result = ds.invoke('setNotice', {
    time: moment().add(1, 'days').format(),
    message: __sealing_unbinding('$params', typeof $params !== 'undefined' ? $params : undefined).message,
    userId: __sealing_unbinding('$context', typeof $context !== 'undefined' ? $context : undefined).userId
});

module.exports = result;
```

这只是一个简单例子，还有很多场景要处理

问题：自动容灾

还记得那次鞋厂挂掉的事情么？

页面直接报错了！空窗了！



吓得我瓜子都掉了

虽然挂掉之后股价竟然涨了...

我宝要求严格多了...

问题：自动容灾

首先我们要定义接口和用户的关系

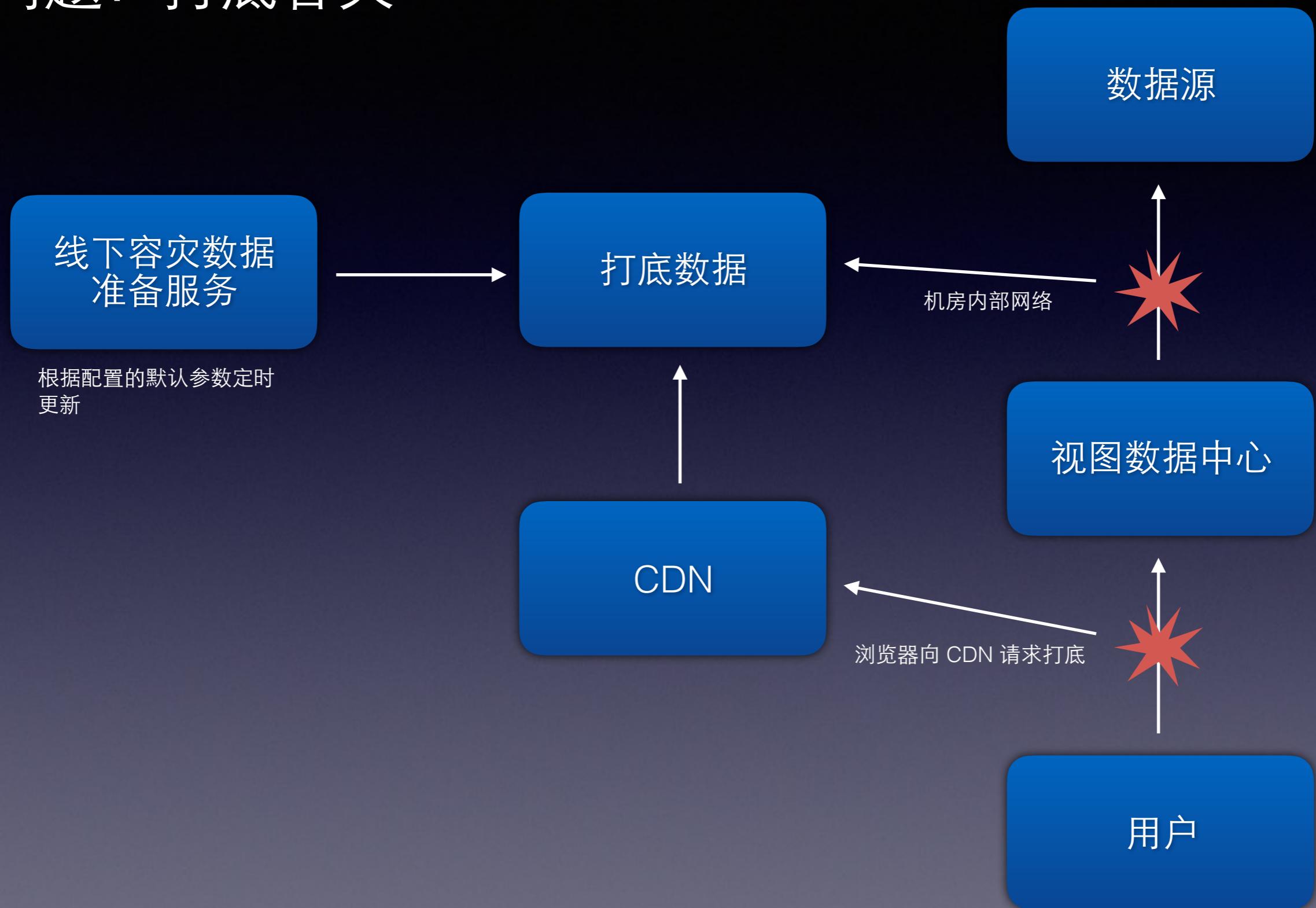
	用户隐私	用户弱关联	无用户关联
打底 (默认参数容灾)	×	√	√
动态容灾 (动态参数容灾)	×	×	√

如个人中心

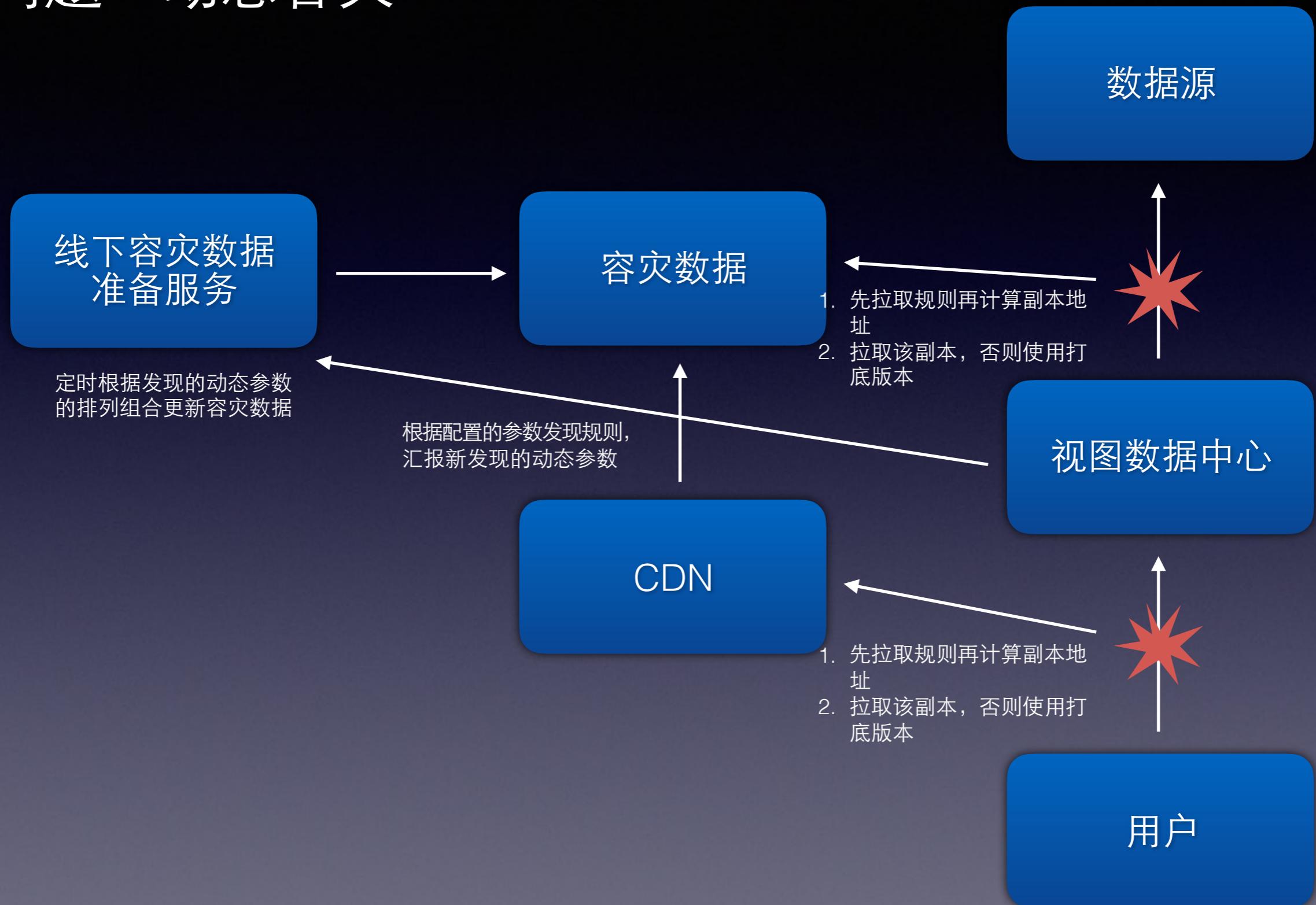
如个性化推荐

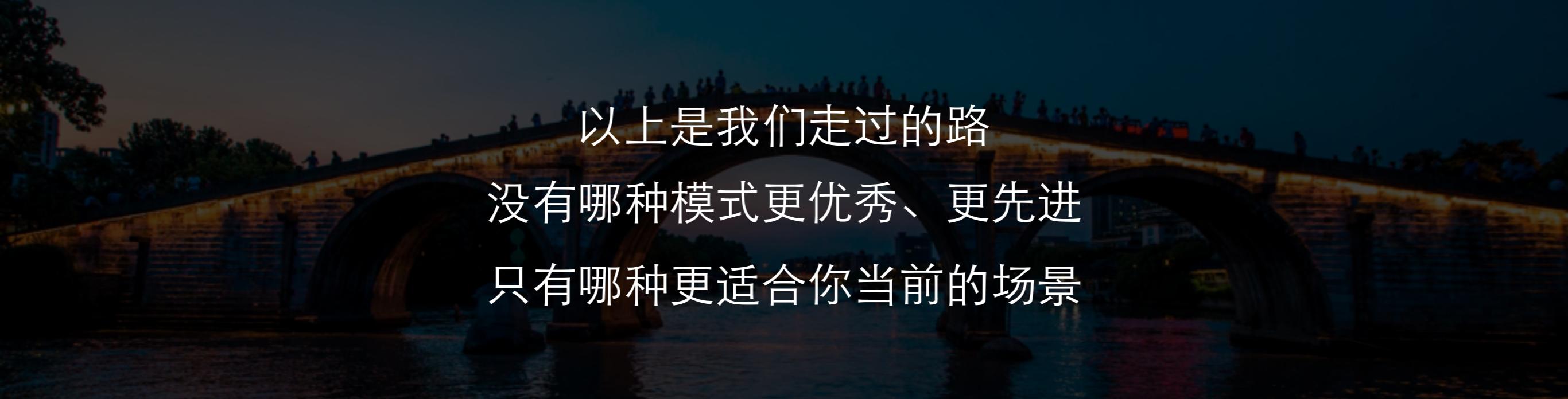
纯展示

问题：打底容灾



问题：动态容灾

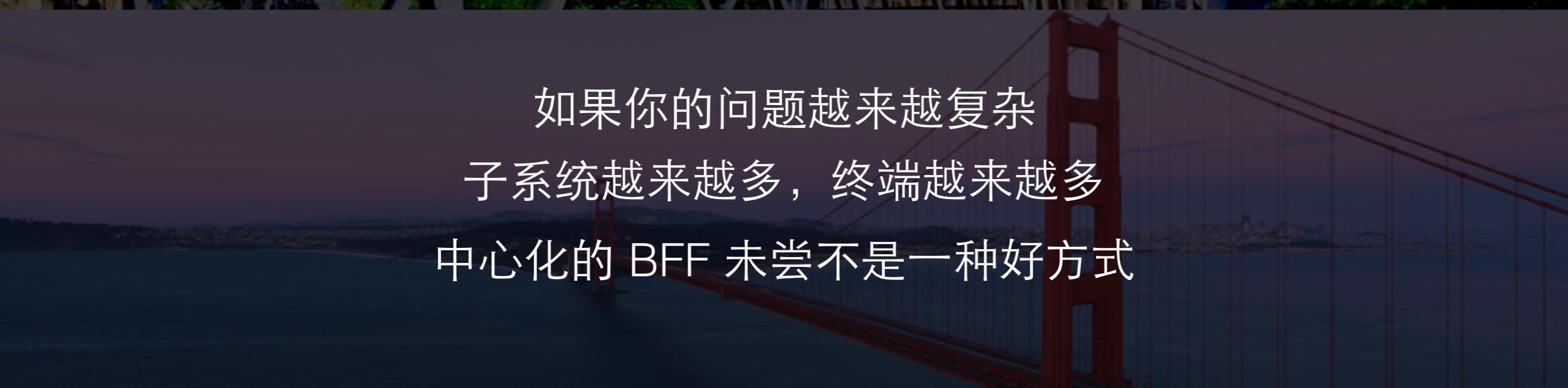




以上是我们走过的路
没有哪种模式更优秀、更先进
只有哪种更适合你当前的场景



如果你是一个初创团队
那些简单实用的方案是最有效的



如果你的问题越来越复杂
子系统越来越多，终端越来越多
中心化的 BFF 未尝不是一种好方式

还有很多想要分享，可是时间短暂
欢迎来 taobaofed.org 一起交流

THX

Follow me on Twitter @GuangWong