蚂蚁财富的BFF实践

D2 - 2016

— 汤尧

蚂蚁金服
ANT FINANCIAL

汤尧

Tags: Java, 前端, Node.js, 大数据，金融

Chair contributor

蚂蚁-体验技术部-财富&保险前端负责人

@coolme200

业务现状

什么是BFF

BFF 实战

总结

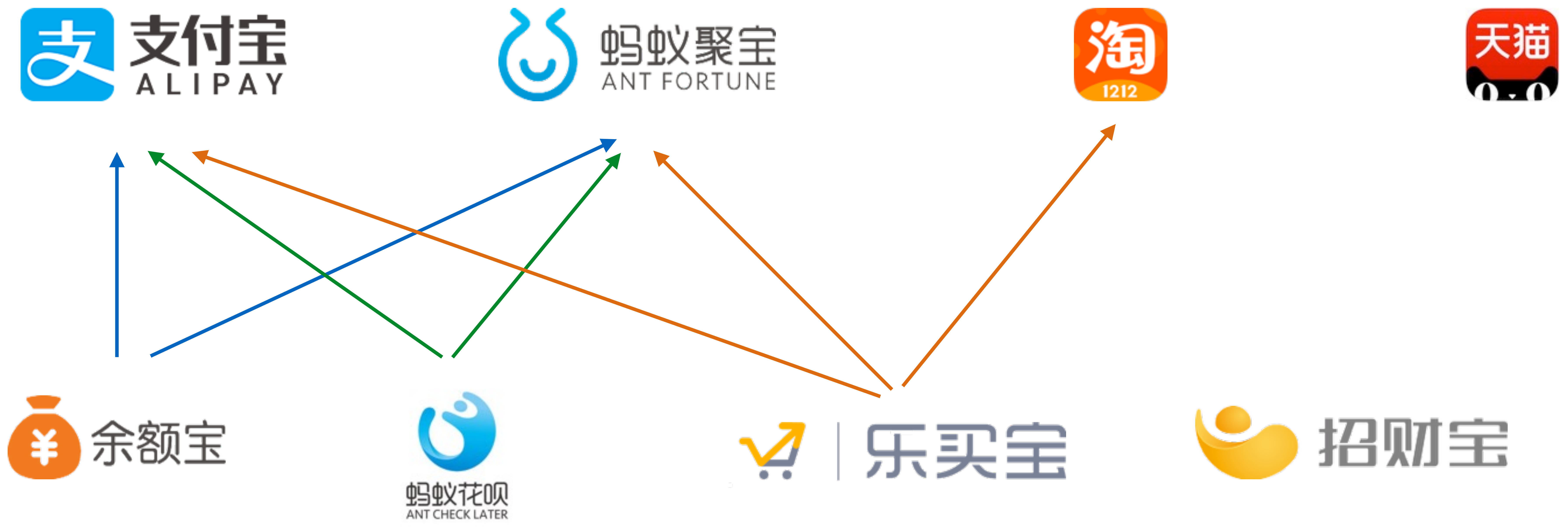
业务现状

什么是BFF

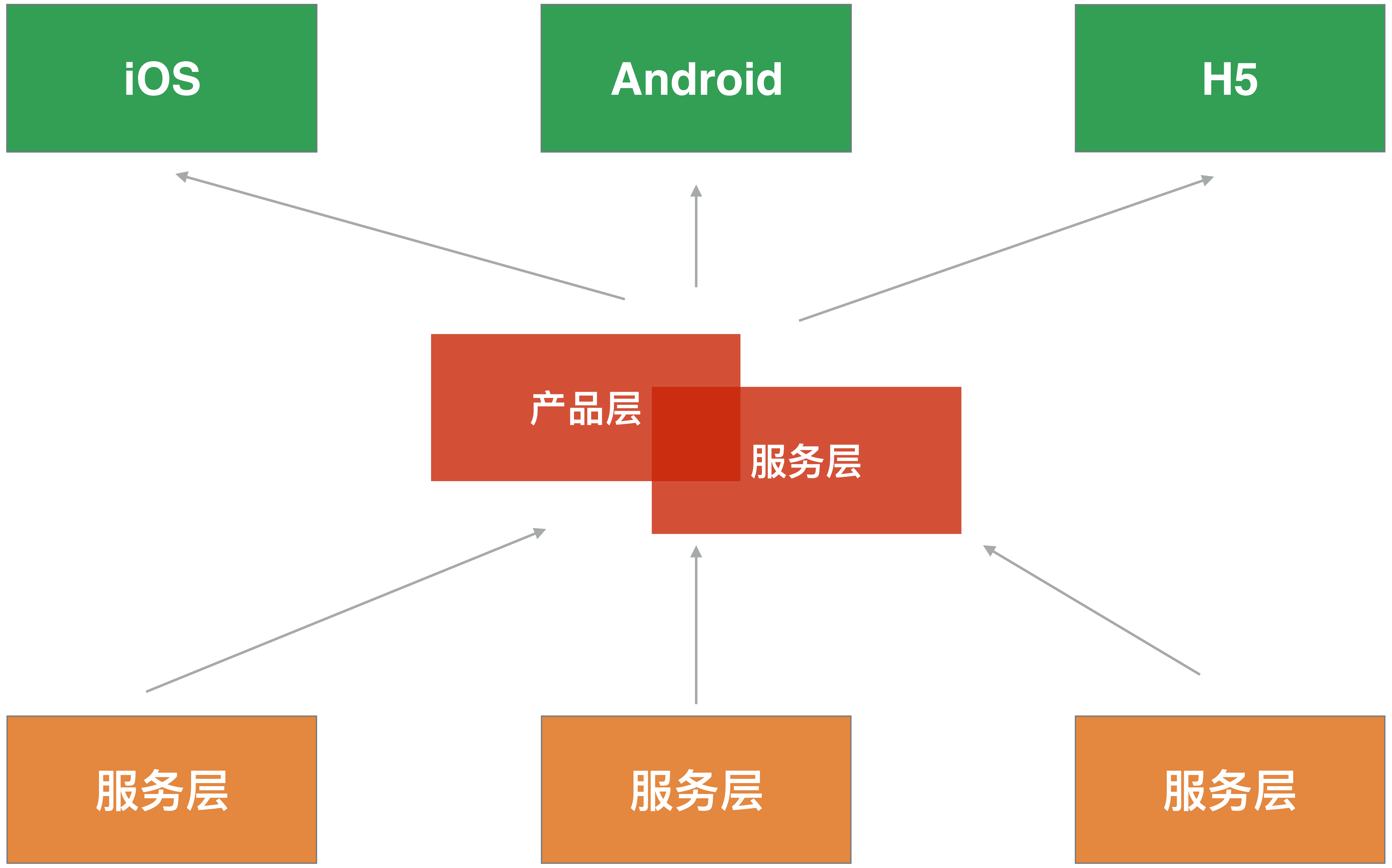
BFF 实战

总结

业务现状


支付宝
ALIPAY
蚂蚁聚宝
ANT FORTUNE
淘
1212
天猫
余额宝
蚂蚁花呗
ANT CHECK LATER
乐买宝
招财宝

业务现状



业务现状

- 服务层 API 相对稳定
- 体验者 API 经常变化
  - 服务端设计的接口究竟是面向 UI，还是只是通用服务?
- 跨团队低效协作
- 资源协调困难

有没有更好的方法来解决上面的问题?

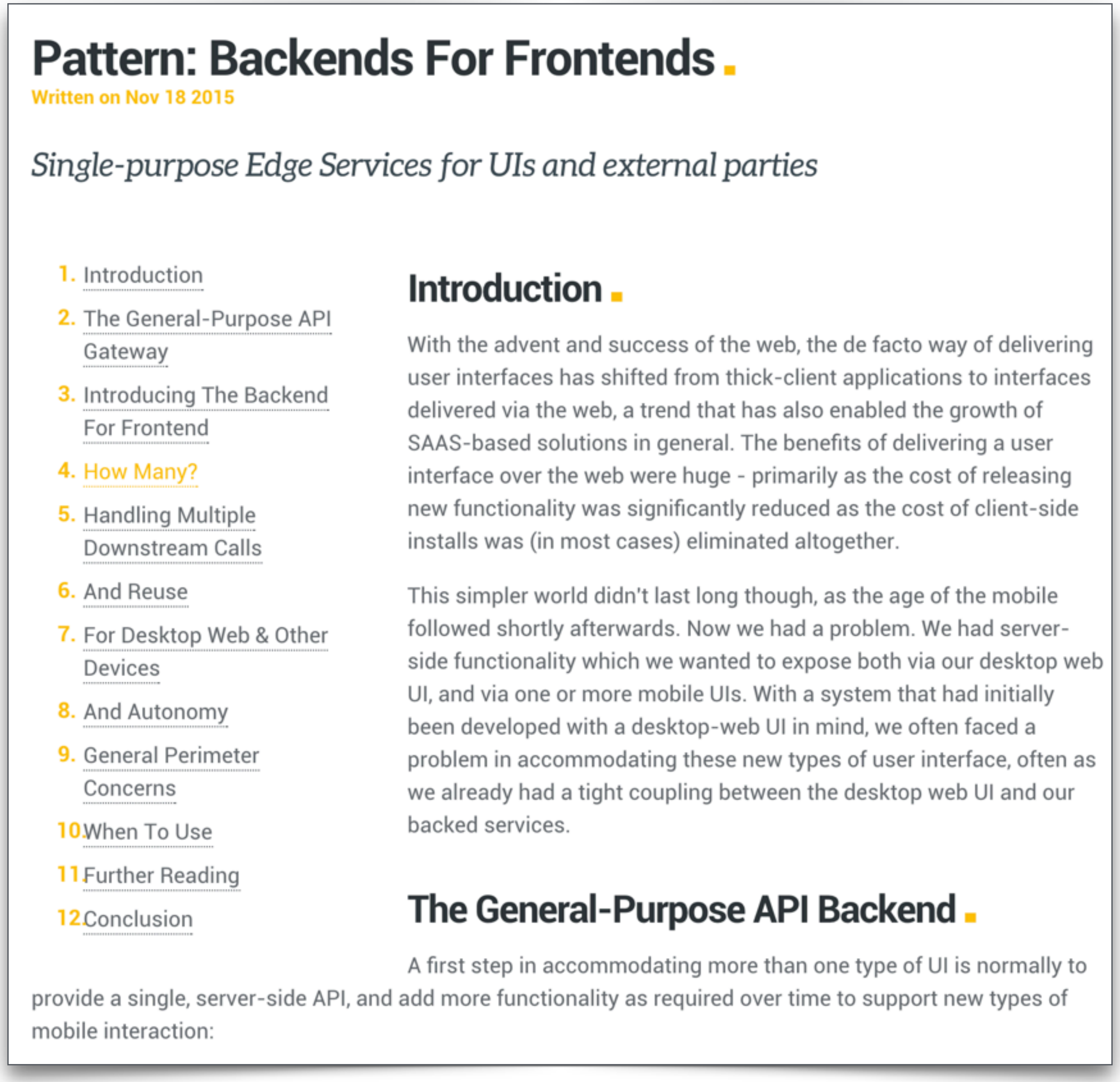
业务现状

什么是BFF

BFF 实战

总结

什么是BFF



Pattern: Backends For Frontends

Written on Nov 18 2015

Single-purpose Edge Services for UIs and external parties

1. Introduction
2. The General-Purpose API Gateway
3. Introducing The Backend For Frontend
4. How Many?
5. Handling Multiple Downstream Calls
6. And Reuse
7. For Desktop Web & Other Devices
8. And Autonomy
9. General Perimeter Concerns
10. When To Use
11. Further Reading
12. Conclusion

Introduction

With the advent and success of the web, the de facto way of delivering user interfaces has shifted from thick-client applications to interfaces delivered via the web, a trend that has also enabled the growth of SAAS-based solutions in general. The benefits of delivering a user interface over the web were huge - primarily as the cost of releasing new functionality was significantly reduced as the cost of client-side installs was (in most cases) eliminated altogether.

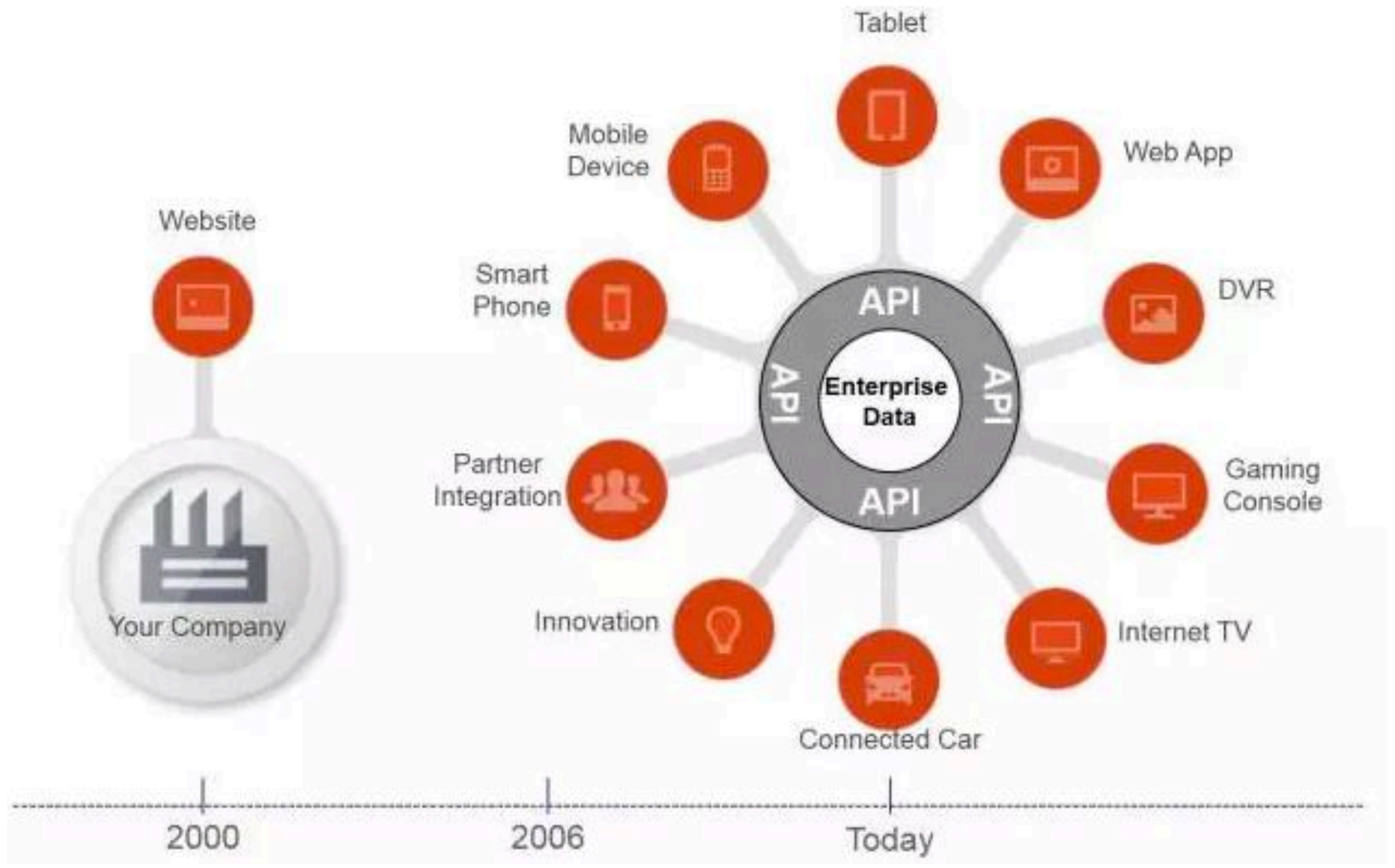
This simpler world didn't last long though, as the age of the mobile followed shortly afterwards. Now we had a problem. We had server-side functionality which we wanted to expose both via our desktop web UI, and via one or more mobile UIs. With a system that had initially been developed with a desktop-web UI in mind, we often faced a problem in accommodating these new types of user interface, often as we already had a tight coupling between the desktop web UI and our backed services.

The General-Purpose API Backend

A first step in accommodating more than one type of UI is normally to provide a single, server-side API, and add more functionality as required over time to support new types of mobile interaction:

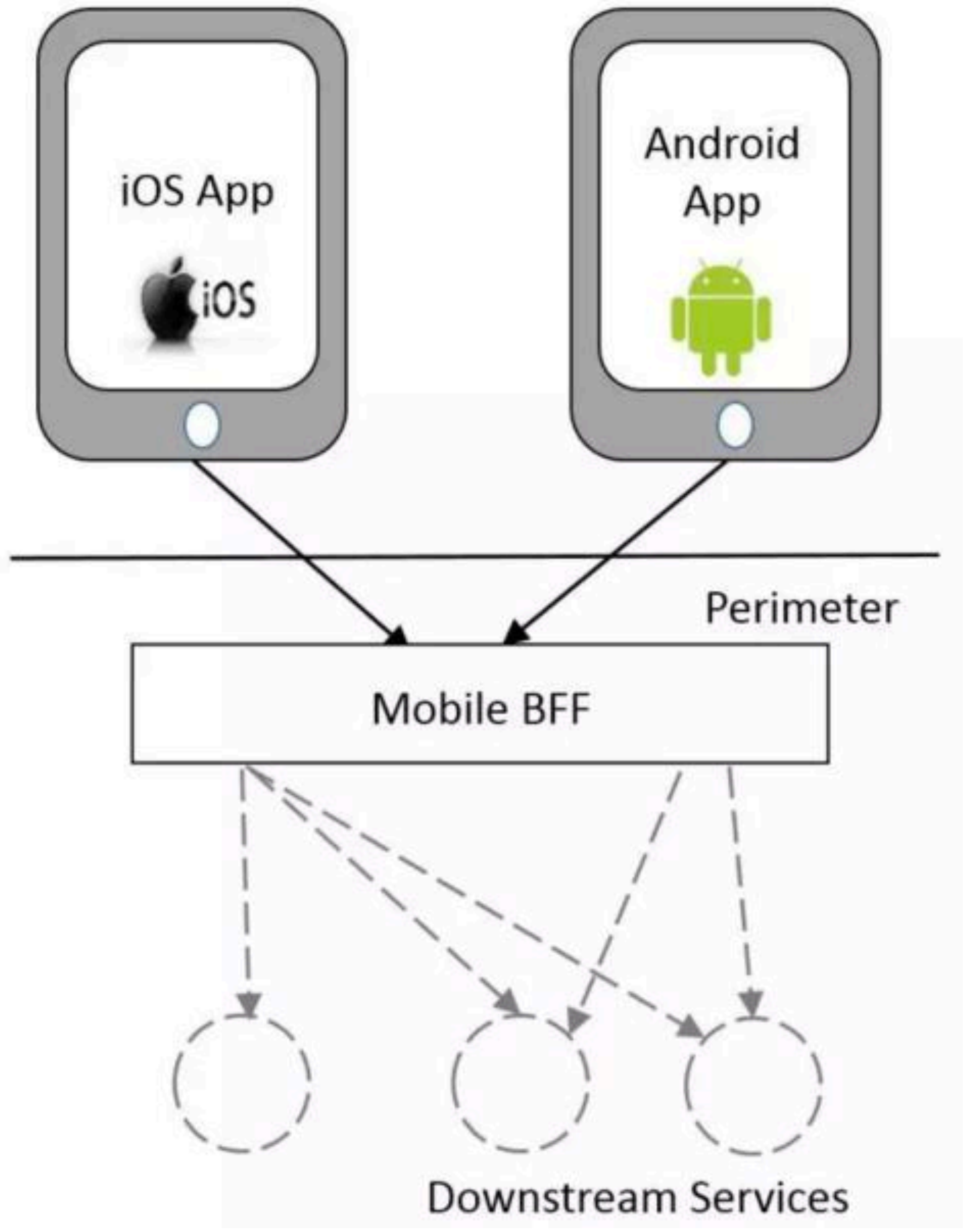
Sam Newman 发表了一篇文章，讲述了这种体验者专用API的方式，并将其称为 **BFF（Backends for Frontends）**模式

什么是BFF



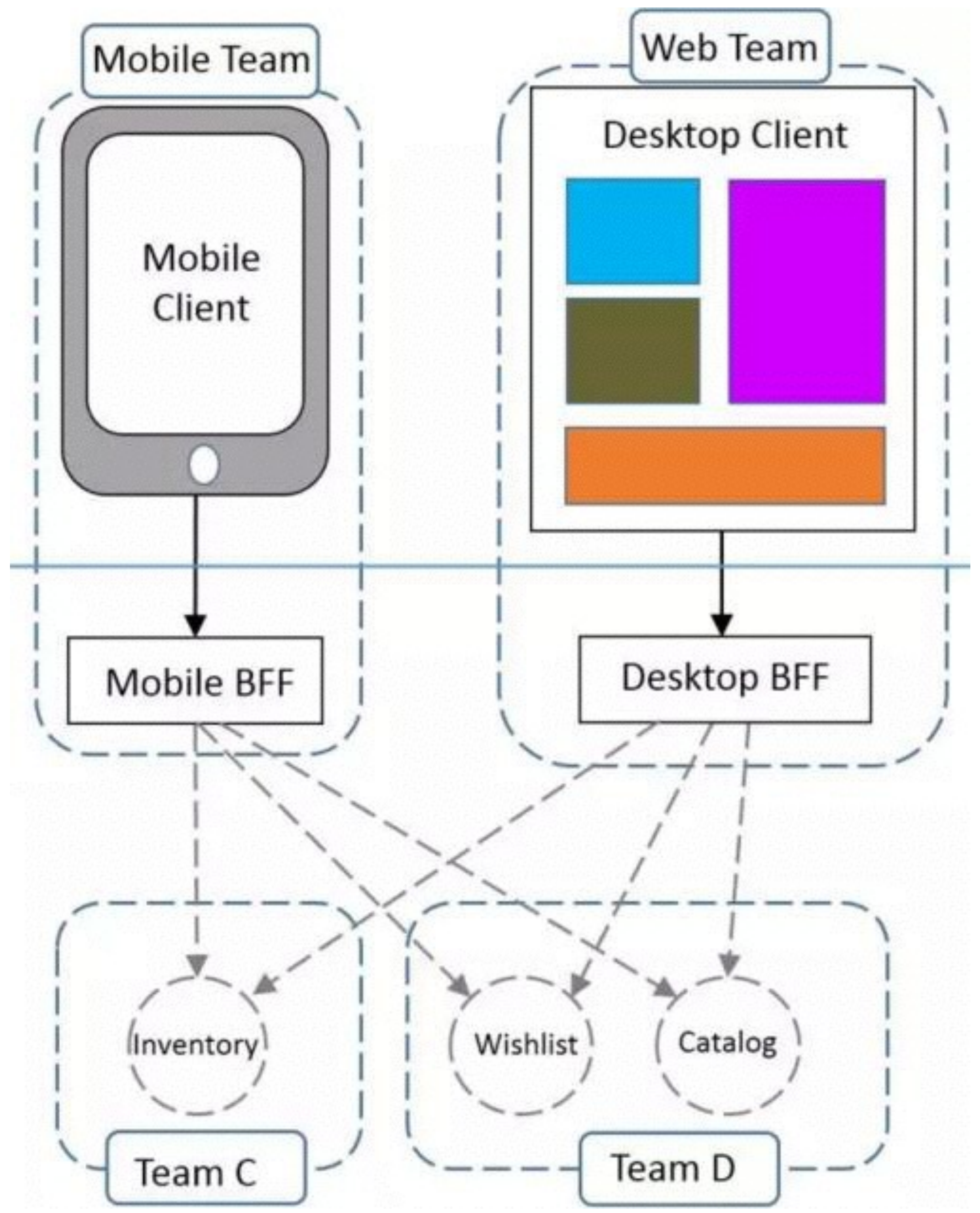
- 用户接入形式的多样性
- 设备不同，需要设计不一样的 API

什么是BFF



- 裁剪和格式化
- 聚合编排

什么是BFF



这是一个理想模型，但是需要考虑实际情况！

业务现状

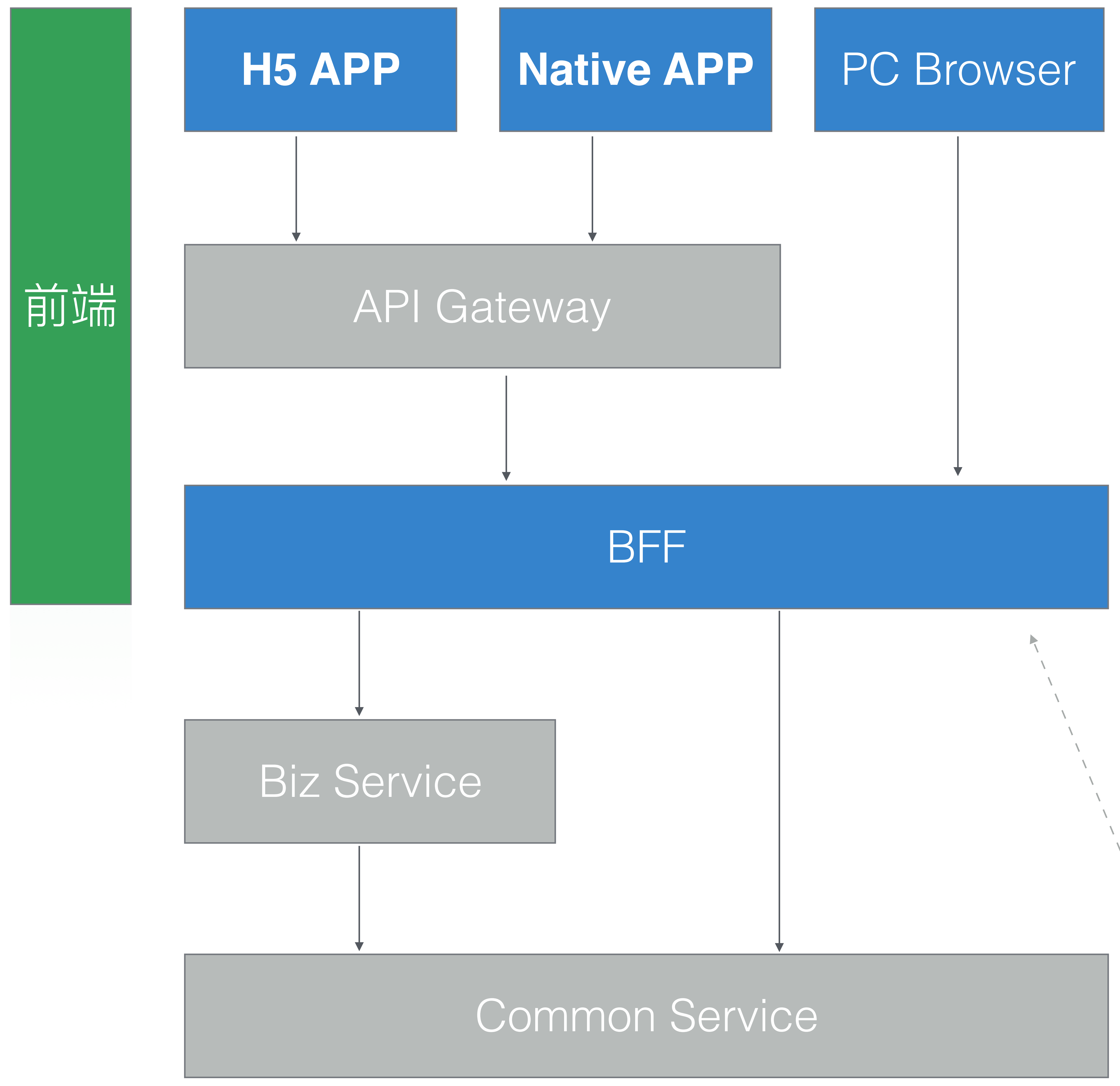
什么是BFF

BFF 实战

总结

Newman 的 BFF 是顺应他的复杂环境的必然产物

在我们的环境中如何做 BFF?



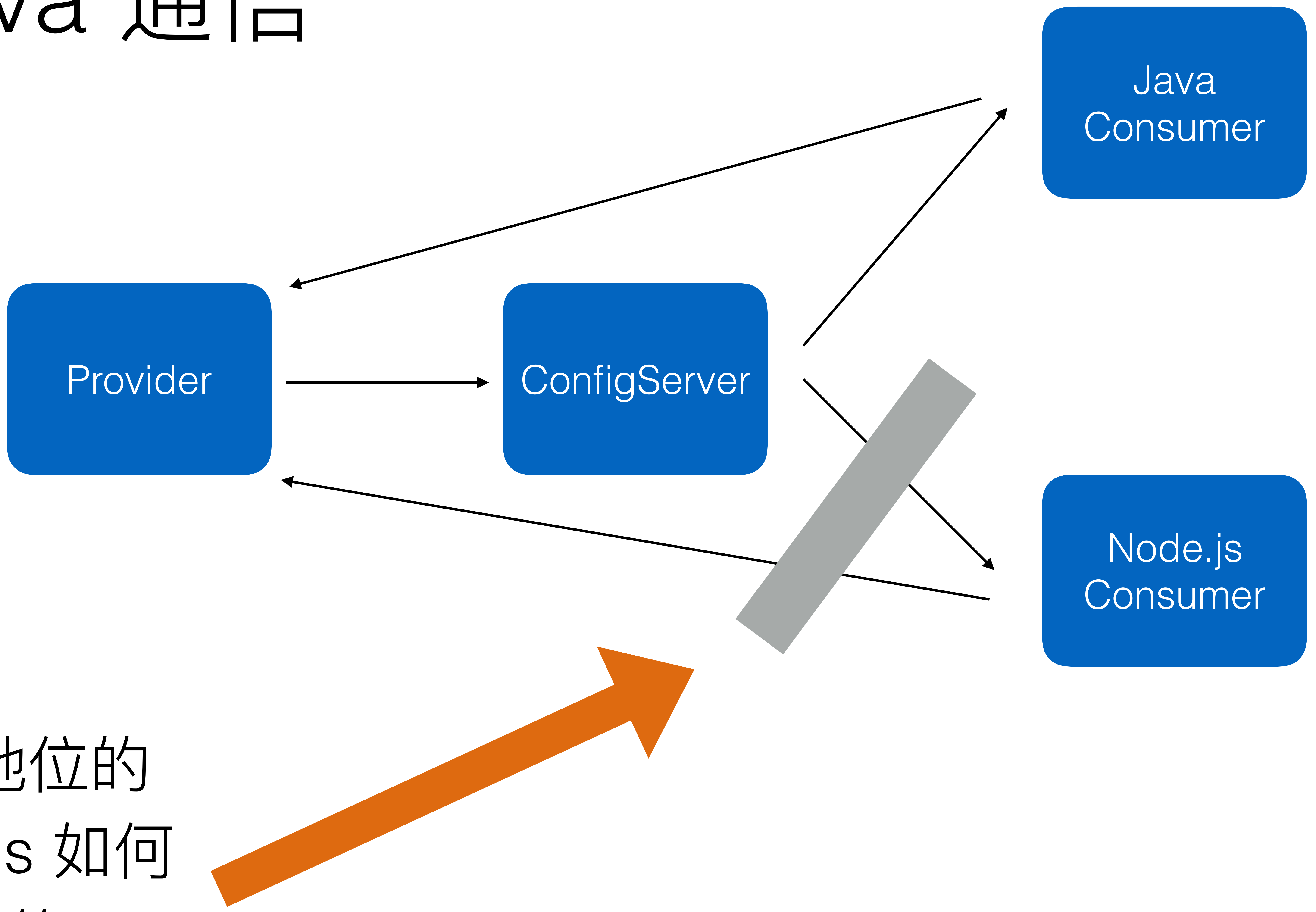
BFF 针对多端不能做到完全隔离，也不能完全复用，需要适当权衡。

Node.js

BFF 实战

1.Node.js 与 Java 通信
2.多 App 适配
3.聚合
4.接口设计

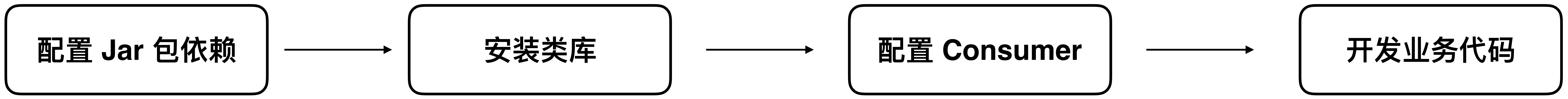
Node.js 与 Java 通信



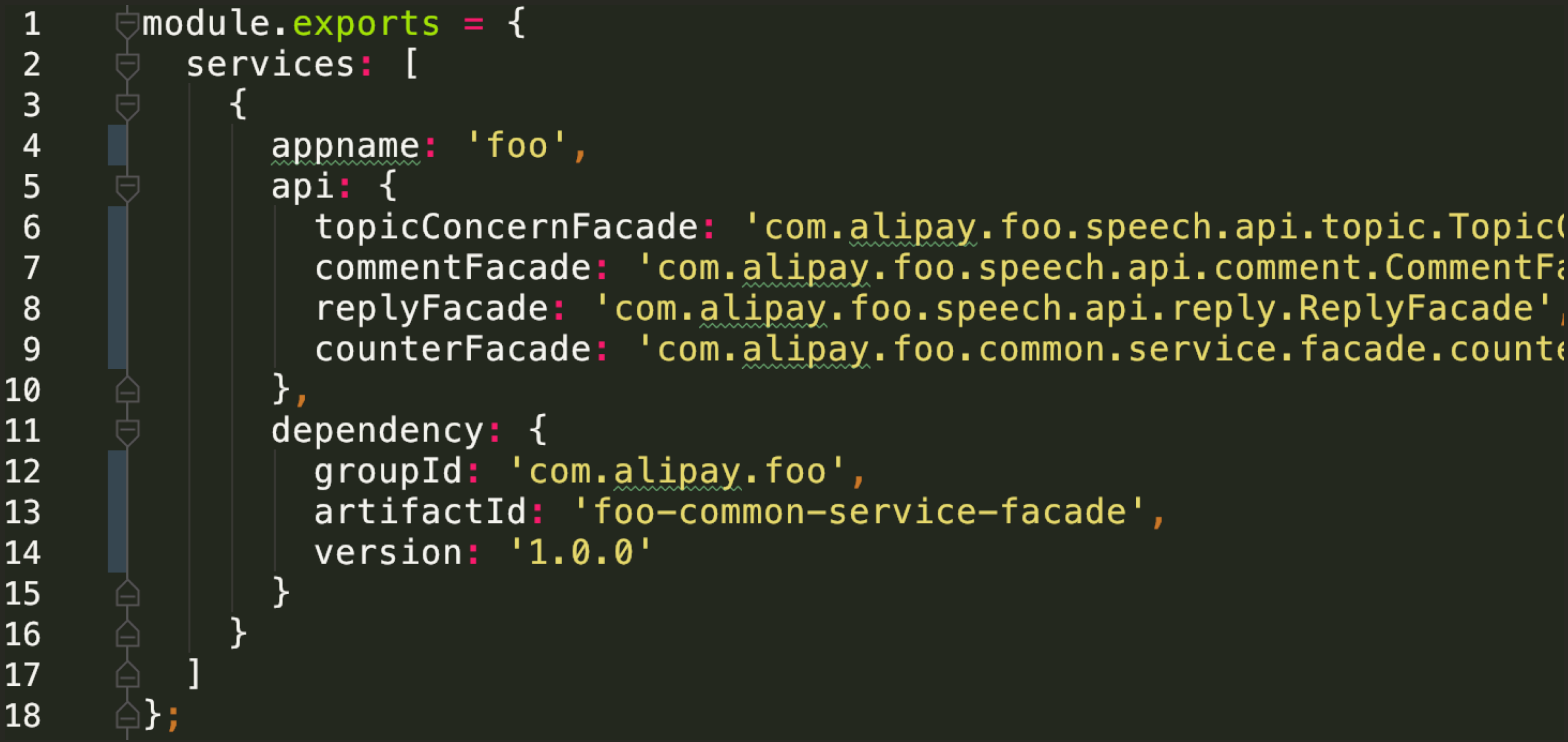
在 Java 占主导地位的应用环境中，Node.js 如何接入，并且保持良好的开发体验?

Node.js 与 Java 通信

- 跨语言序列化协议 hessian
- 在弱类型的 Node.js 中如何调强类型的 Java 服务?



Node.js 与 Java 通信

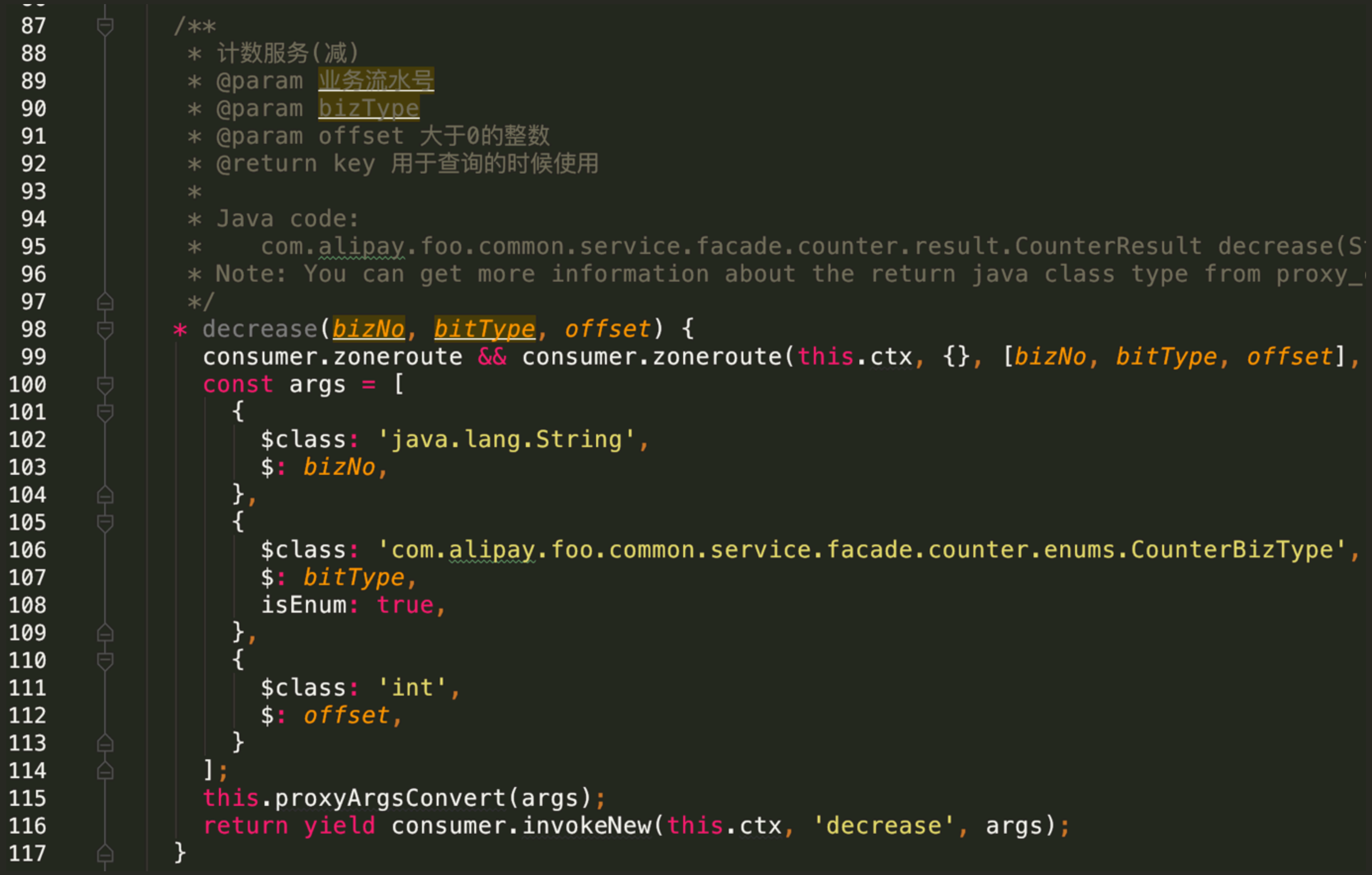


```
module.exports = {
  services: [
    {
      appname: 'foo',
      api: {
        topicConcernFacade: 'com.alipay.foo.speech.api.topic.Topic
        commentFacade: 'com.alipay.foo.speech.api.comment.CommentFa
        replyFacade: 'com.alipay.foo.speech.api.reply.ReplyFacade',
        counterFacade: 'com.alipay.foo.common.service.facade.counte
      },
      dependency: {
        groupId: 'com.alipay.foo',
        artifactId: 'foo-common-service-facade',
        version: '1.0.0'
      }
    }
  ]
};
```

Node.js 与 Java 通信

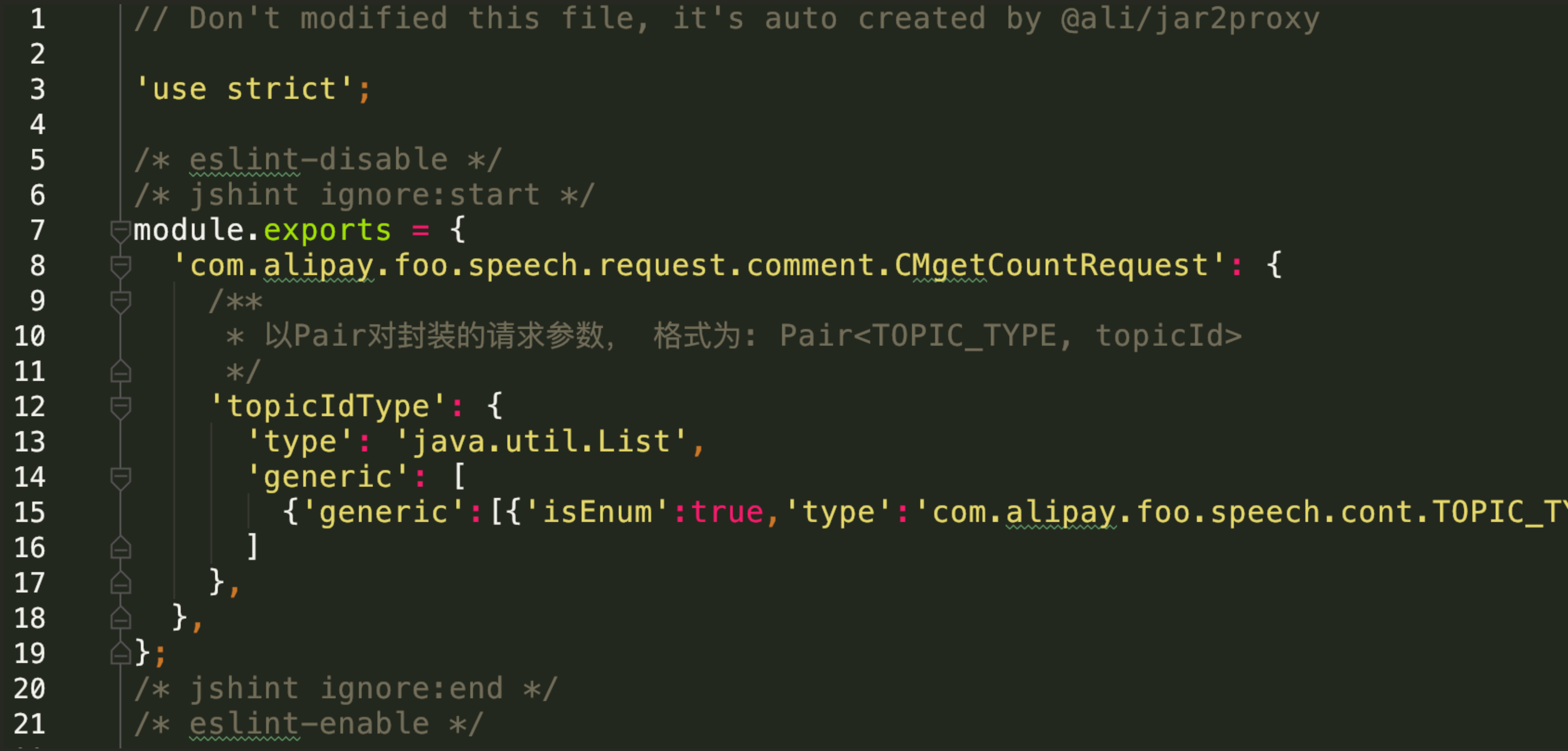
```
$ tnpm run proxy

// 等价于   $ mvn install
```

Node.js 与 Java 通信



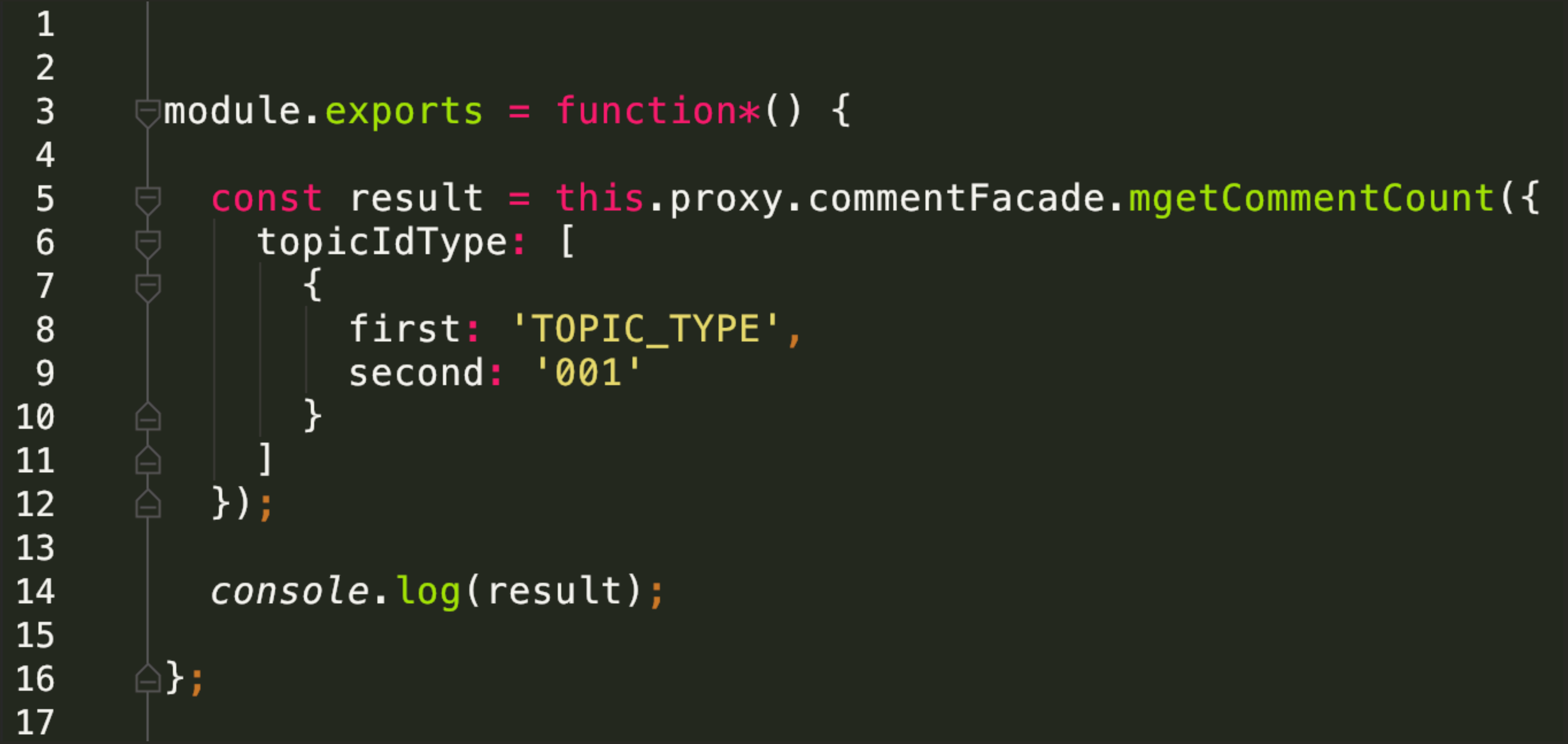
```
/**
 * 计数服务(减)
 * @param 业务流水号
 * @param bizType
 * @param offset 大于0的整数
 * @return key 用于查询的时候使用
 *
 * Java code:
 *    com.alipay.foo.common.service.facade.counter.result.CounterResult decrease(S
 * Note: You can get more information about the return java class type from proxy_
 */
* decrease(bizNo, bitType, offset) {
  consumer.zoneroute && consumer.zoneroute(this.ctx, {}, [bizNo, bitType, offset],
  const args = [
    {
      $class: 'java.lang.String',
      $: bizNo,
    },
    {
      $class: 'com.alipay.foo.common.service.facade.counter.enums.CounterBizType',
      $: bitType,
      isEnum: true,
    },
    {
      $class: 'int',
      $: offset,
    }
  ];
  this.proxyArgsConvert(args);
  return yield consumer.invokeNew(this.ctx, 'decrease', args);
}
```

Node.js 与 Java 通信



```
// Don't modified this file, it's auto created by @ali/jar2proxy

'use strict';

/* eslint-disable */
/* jshint ignore:start */
module.exports = {
  'com.alipay.foo.speech.request.comment.CMgetCountRequest': {
    /**
     * 以Pair对封装的请求参数， 格式为: Pair<TOPIC_TYPE, topicId>
     */
    'topicIdType': {
      'type': 'java.util.List',
      'generic': [
        {'generic':[{'isEnum':true,'type':'com.alipay.foo.speech.cont.TOPIC_T
      ]
    },
  },
};
/* jshint ignore:end */
/* eslint-enable */
```

Node.js 与 Java 通信



```javascript


module.exports = function*() {

  const result = this.proxy.commentFacade.mgetCommentCount({
    topicIdType: [
      {
        first: 'TOPIC_TYPE',
        second: '001'
      }
    ]
  });

  console.log(result);

};

```

安全

缓存

消息服务

网关

限流

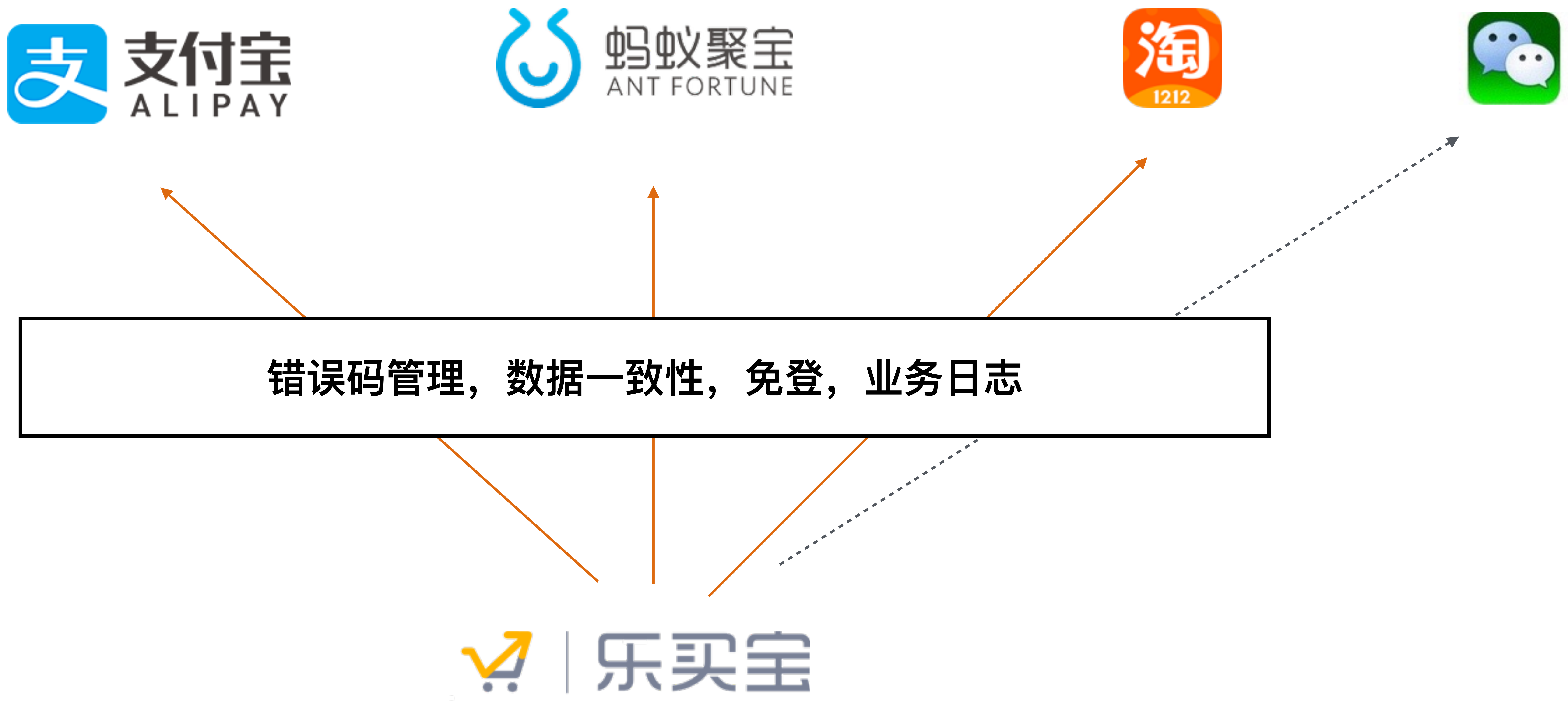
权限

日志

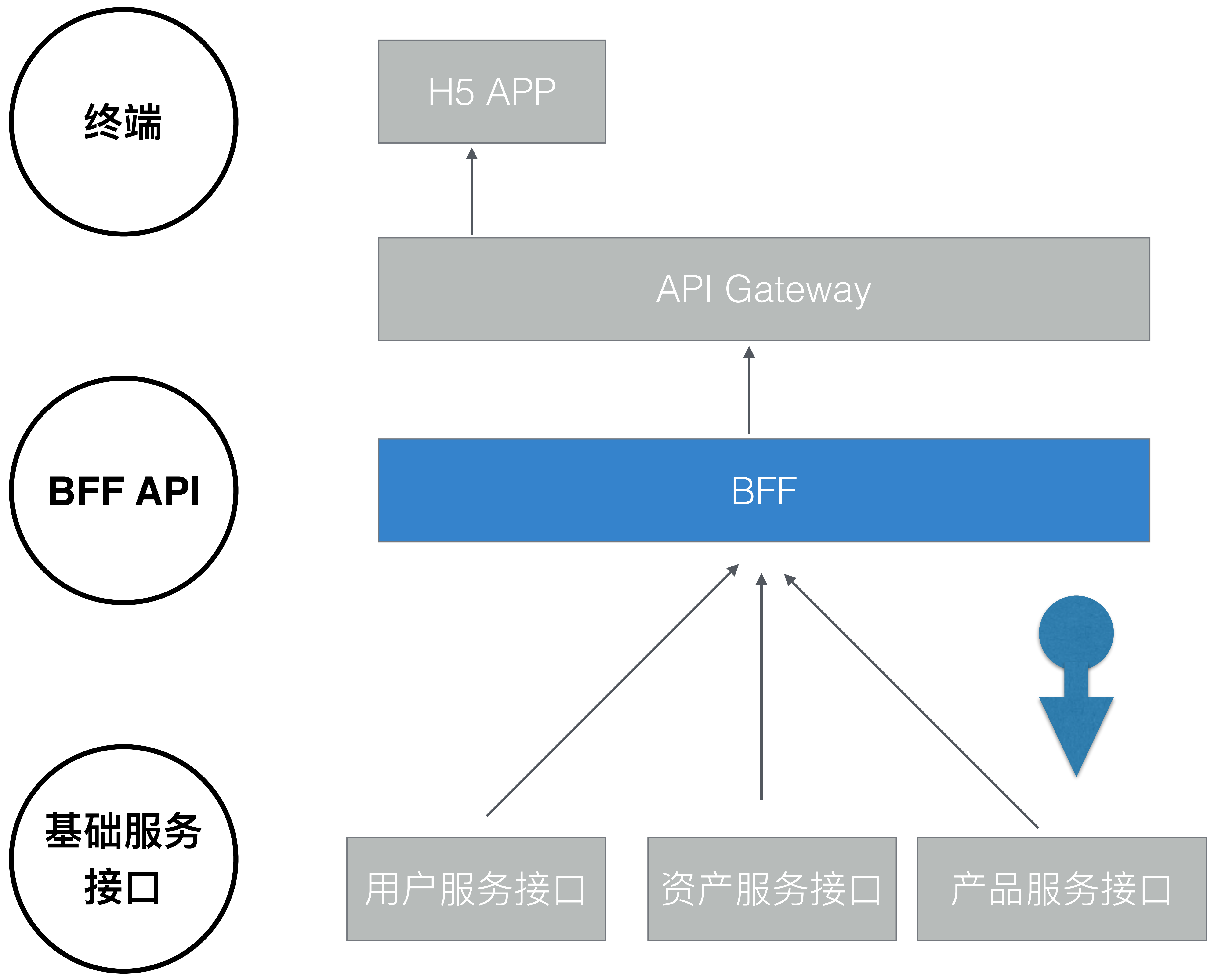
逻辑数据中心路由

虚拟IP

多 App 适配

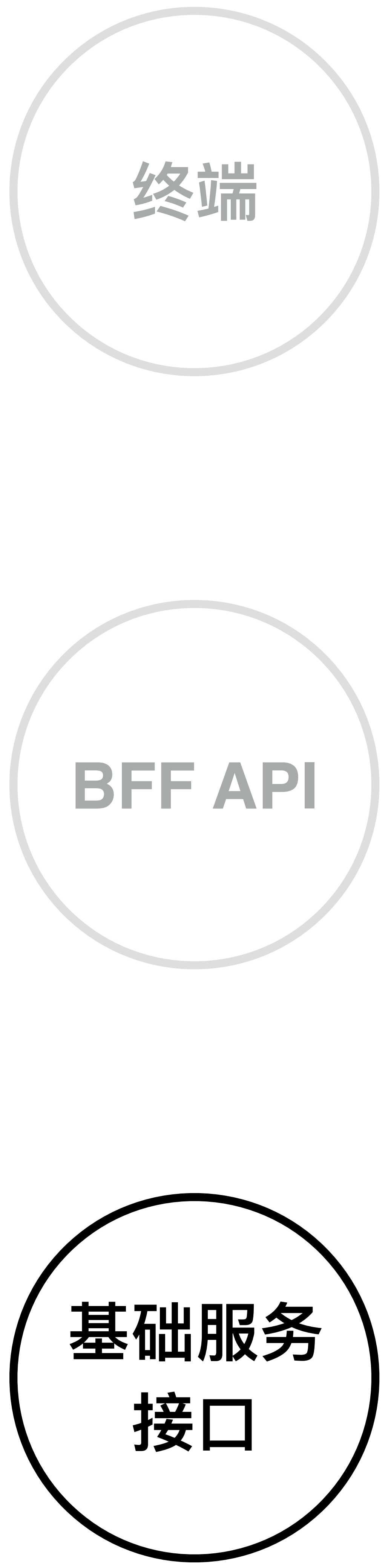


聚合



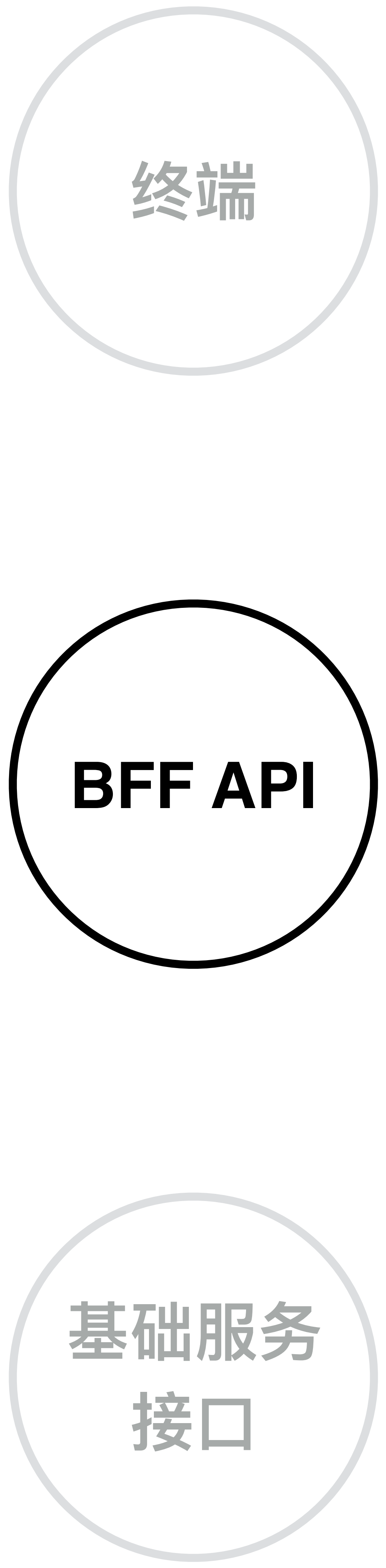
- 简化客户端逻辑，减少网络开销
- 避免无意义的透传
- 敏感信息过滤

接口设计



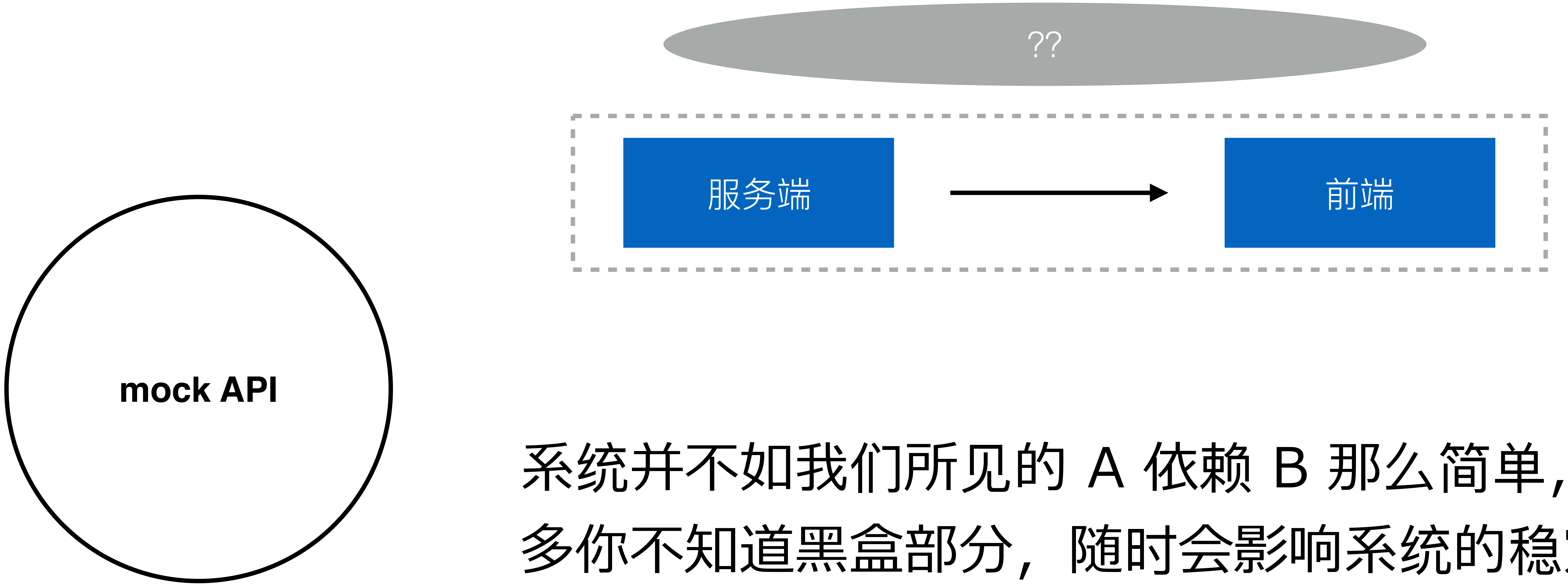
- 细粒度
- 通用的功能, 可能会被多个 BFF 用到
- 提供含各种状态的 mock **真实**数据，易于同步开发

接口设计



- 合理设计接口数量，太多不易维护
- 提供含各种状态的 mock **真实**数据，页面不依赖 server 开发
- 多协议发布
- 规范数据格式

接口设计



系统并不如我们所见的 A 依赖 B 那么简单，还有很多你不知道黑盒部分，随时会影响系统的稳定性，导致你的开发无法正常进行。

业务现状

什么是BFF

BFF 实战

总结

技术

- 前端和 BFF 由同一人完成
- 前端需要具备服务端技能
- 快速的应用发布能力 （docker?）

开发者

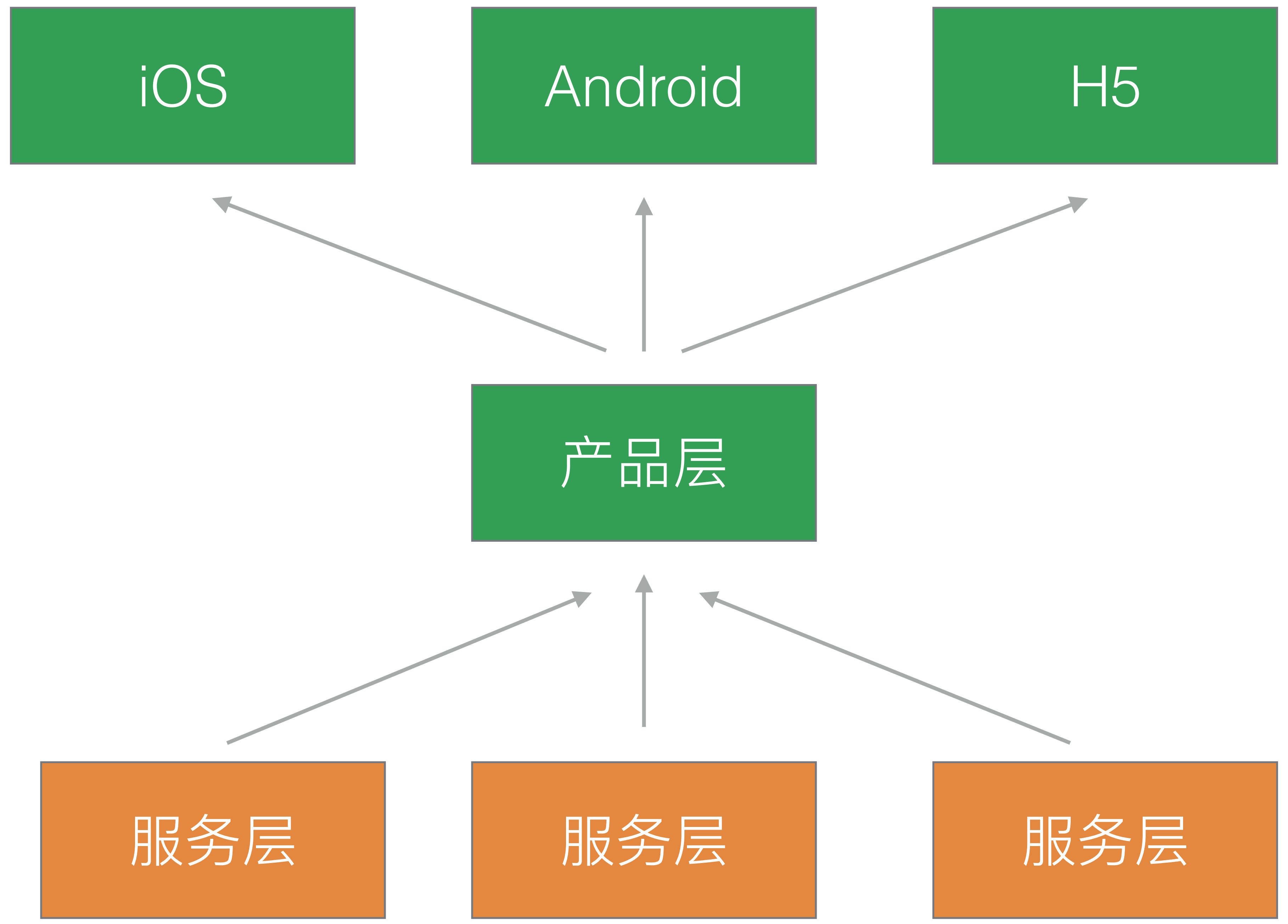
- 全栈是为了更合理的分工
- 做 BFF 可以拓展知识面
- 提升沟通协调能力

好处

业务支持变多

沟通协作变少

解决问题变快



坏处

- 组织决定了架构的复杂度
- 前期学习成本高，短期成为资源瓶颈

THANK YOU