Node.js 的 Dubbo RPC 实践

高晓晨（宗羽）

蚂蚁金服体验技术部 | 基础技术组

专注于 Node.js 基础技术，中间件，Web 框架，Serverless 架构等方向

@gxcsoccer

https://github.com/gxcsoccer

提纲

1. Node.js 在阿里的主要运用场景
2. 如何用 Node.js 实现 Dubbo RPC ？
3. Demo: Node.js 和 Java 用 Dubbo 互相调用
4. Demo: 用 Nacos 做服务发现
5. 我们踩过的一些「坑」

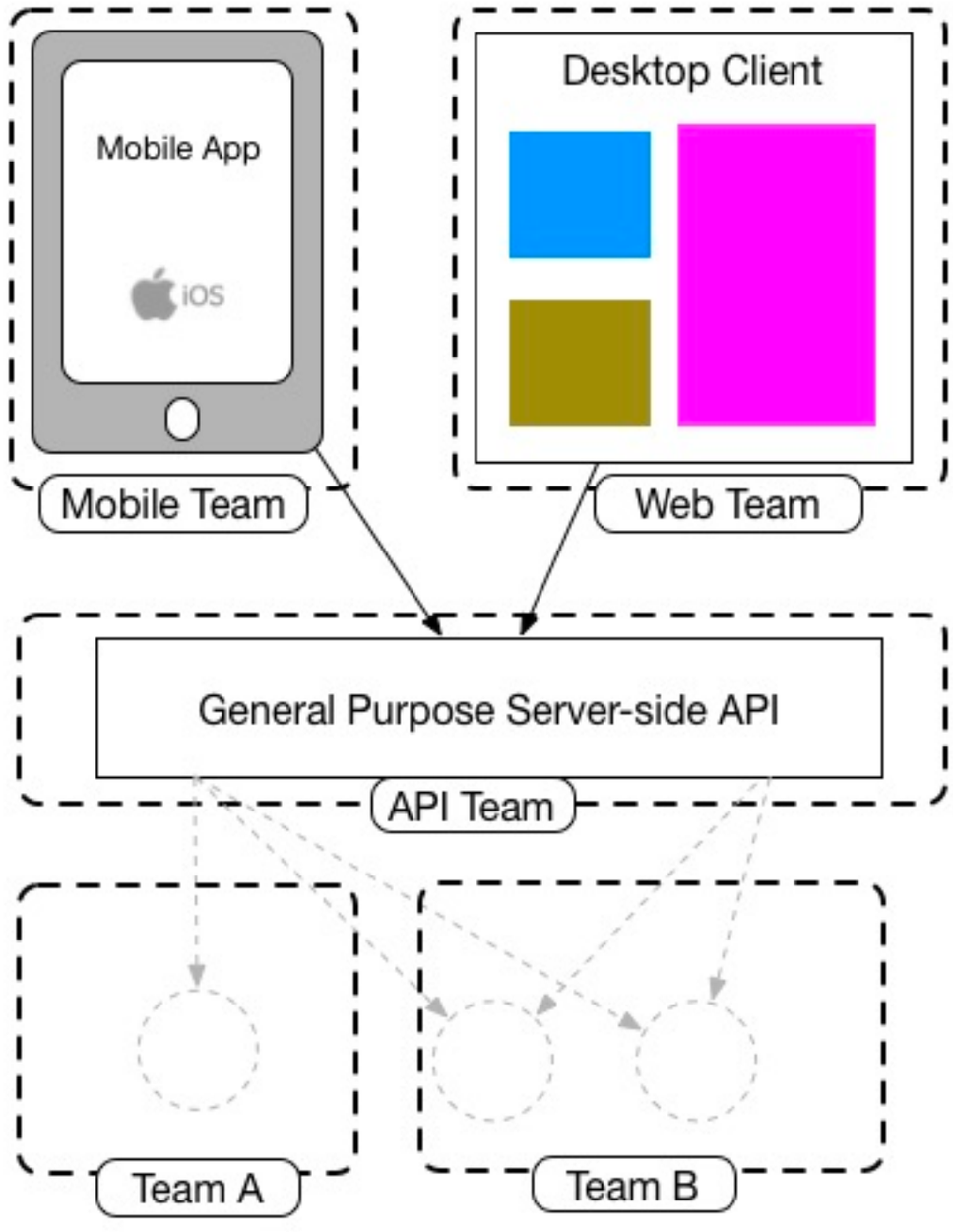
Node.js 在阿里主要场景

1. 全栈（中后台应用）
2. BFF / MVC （Node.js 作为接口聚合层）
3. 其他（前端工具、同构、IoT 等）

重点介绍一下 BFF

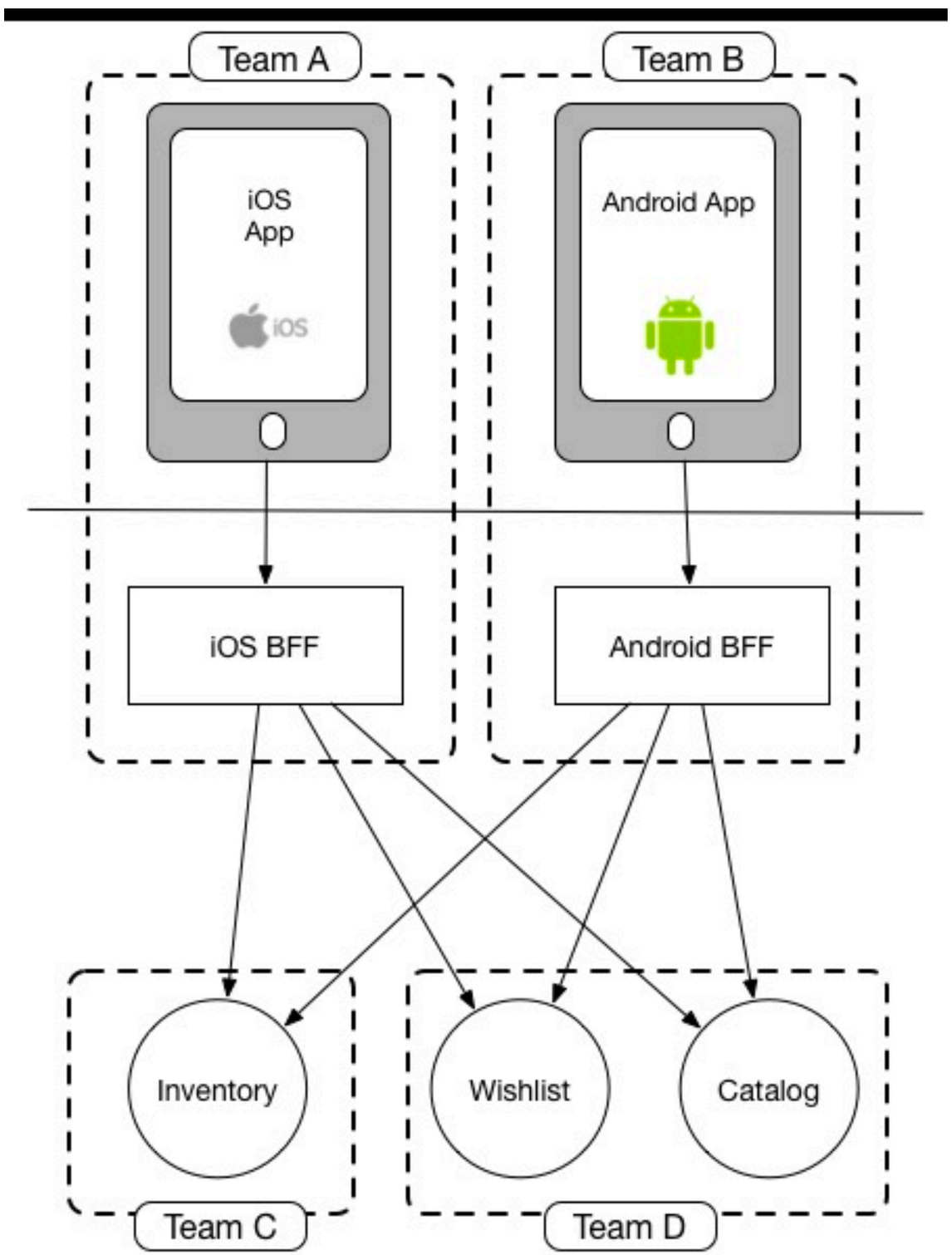
以前的研发模式

- 用户体验灵活性与服务稳定性的矛盾：不同的终端对 API 有不同的诉求
- 团队协作上的问题：通用服务封装层变成研发的瓶颈

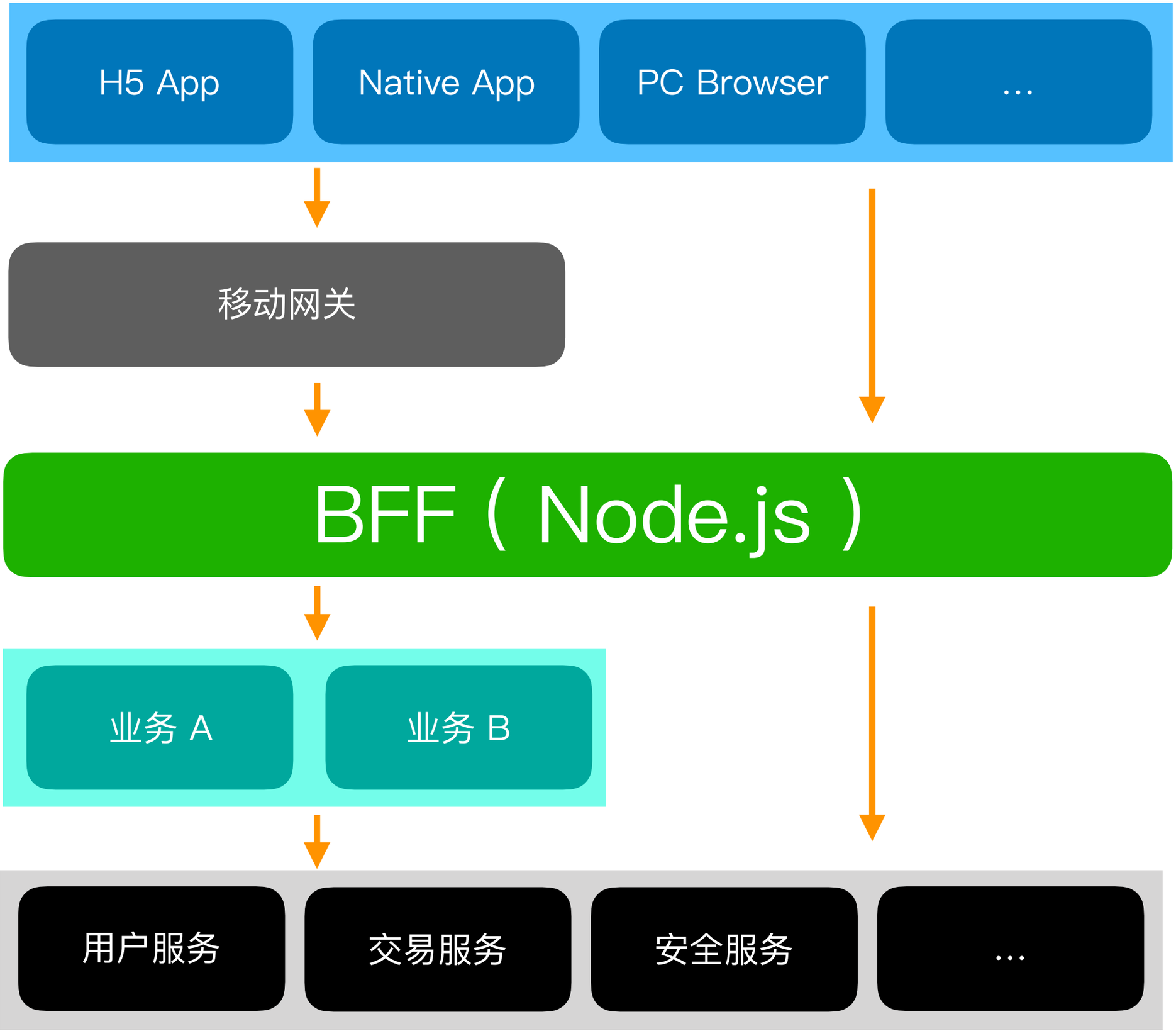


BFF 模式

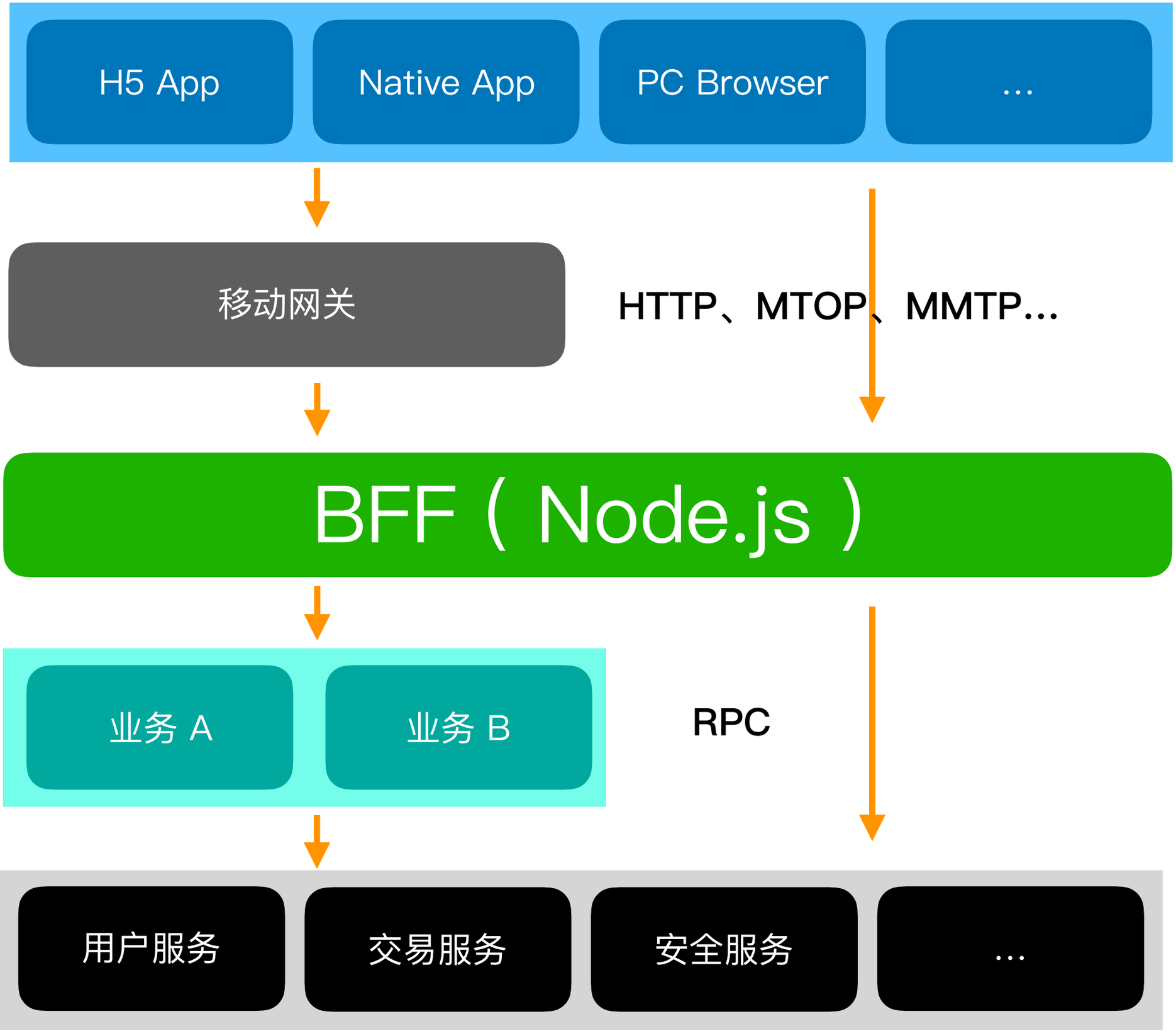
- Backend for Frontend
- 每一个端独立拥有一个接口层，只为这个端服务，实现数据聚合、裁剪和格式化
- BFF 可以根据团队技术栈自由选型
- 服务自治原则：谁使用、谁负责、谁开发



我们用 Node.js 来承载 BFF 层



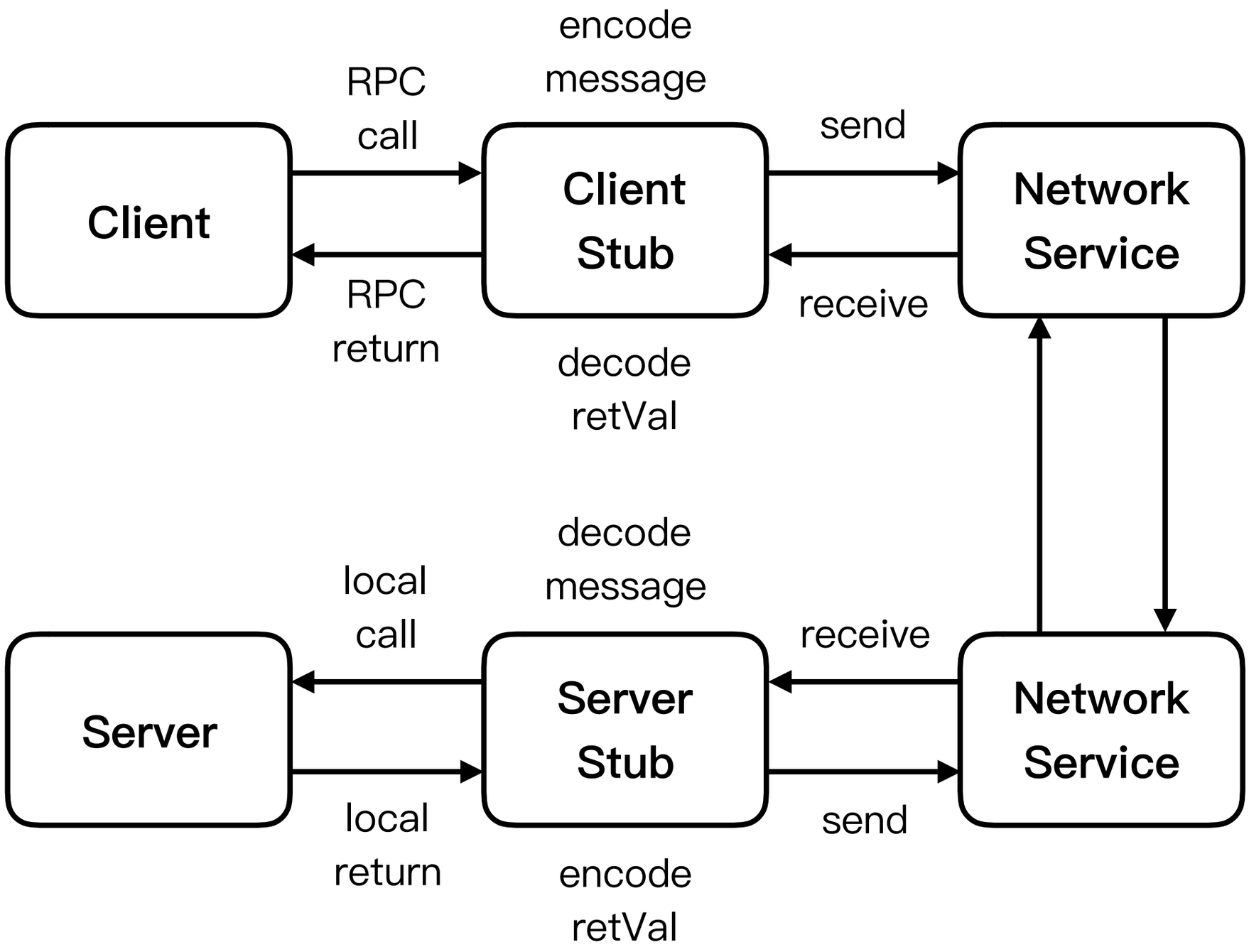
我们用 Node.js 来承载 BFF 层



如何用 Node.js 实现

Dubbo RPC ?

RPC 原理

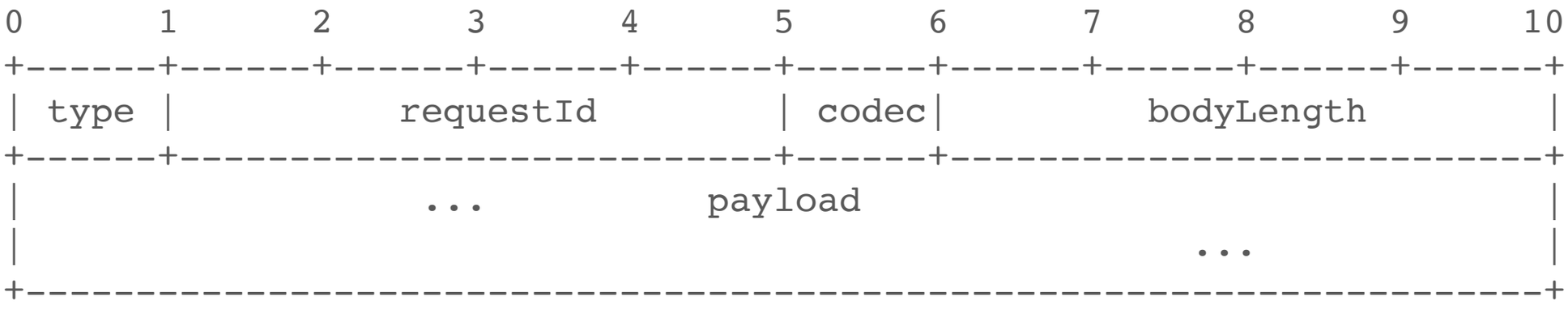

encode
message
RPC
call
send
Client
Client
Stub
Network
Service
RPC
return
receive
decode
retVal
decode
message
local
call
receive
Server
Server
Stub
Network
Service
local
return
send
encode
retVal

RPC 中的协议

数据在网络只能以二进制形式传输，所以我们必须将数据类型转换成二进制传递给对方，然后对方收到以后再还原成原始数据类型，这个转换的规则就叫「协议」，只要调用两端遵守同一份协议，那么就可以愉快的通讯了

- 通讯层协议： 和业务无关的，它的职责是将业务数据打包后，安全、完整的传输给接收方
- 应用层协议： 约定业务数据和二进制的转换规则

最简单的 RPC 通讯协议



应用层协议（序列化 / 反序列化）

应用层协议负责业务数据和二进制的转换，从业务数据转换成二进制这个过程叫「序列化」，相反从二进制还原业务数据的过程叫「反序列化」。常见的序列化、反序列化方式有：

- Hessian
- Protobuf
- JSON
- ...

接口描述

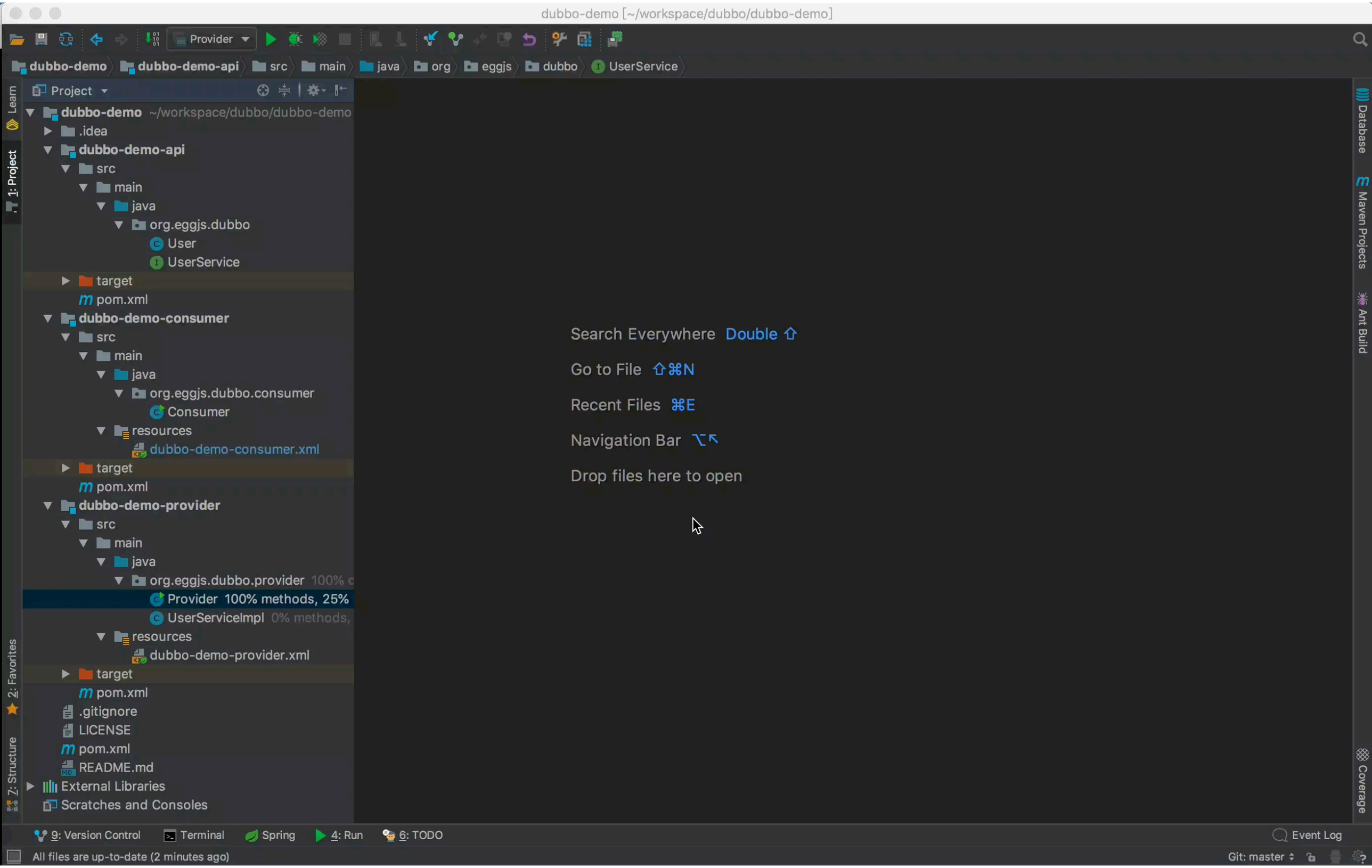
除了协议，要能通讯还需要一种语言无关的接口描述，它定义了接口的元数据，包括：

- 方法签名（名字 + 参数列表）
- 基础数据类型
- 复杂对象类型

接口描述

```
service ProtoService {
    rpc echoObj (EchoRequest) returns (EchoResponse) {}
}

message EchoRequest {
    string name = 1;
    Group group = 2;
}

message EchoResponse {
    int32 code = 1;
    string message = 2;
}
```

DEMO: Node.js 和 Java 用 Dubbo RPC 互相调用


dubbo-demo [~/workspace/dubbo/dubbo-demo]
Provider
dubbo-demo
dubbo-demo-api
src
main
java
org
eggjs
dubbo
UserService
Project
dubbo-demo ~/workspace/dubbo/dubbo-demo
.idea
dubbo-demo-api
src
main
java
org.eggjs.dubbo
User
UserService
target
pom.xml
dubbo-demo-consumer
src
main
java
org.eggjs.dubbo.consumer
Consumer
resources
dubbo-demo-consumer.xml
target
pom.xml
dubbo-demo-provider
src
main
java
org.eggjs.dubbo.provider
Provider 100% methods, 25%
UserServiceImpl 0% methods,
resources
dubbo-demo-provider.xml
target
pom.xml
.gitignore
LICENSE
pom.xml
README.md
External Libraries
Scratches and Consoles
Search Everywhere Double ⇧
Go to File ⇧⌘N
Recent Files ⌘E
Navigation Bar ⌥↖
Drop files here to open
Learn
1: Project
2: Favorites
7: Structure
Database
Maven Projects
Ant Build
Coverage
9: Version Control
Terminal
Spring
4: Run
6: TODO
Event Log
All files are up-to-date (2 minutes ago)
Git: master

服务发现

通讯问题解决了，接下来的问题是服务在哪儿？该往哪里调它？

这里就要引入「服务发现」的概念

DEMO: 用 Nacos 实现服务发现

https://nacos.io/en-us/

NACOS.

HOME DOCS BLOG COMMUNITY DEMO-CONSOLE 中

NACOS.

an easy-to-use dynamic service discovery, configuration and service management platform for building cloud native applications

View on Github | Manual

V0.7.0 Release Note

Released on Dec 14, 2018

使用 egg-cloud
调用 dubbo 服务

使用 egg-cloud
发布 dubbo 服务

实现 Node.js RPC

我们遇到的一些问题

跨语言类型映射问题

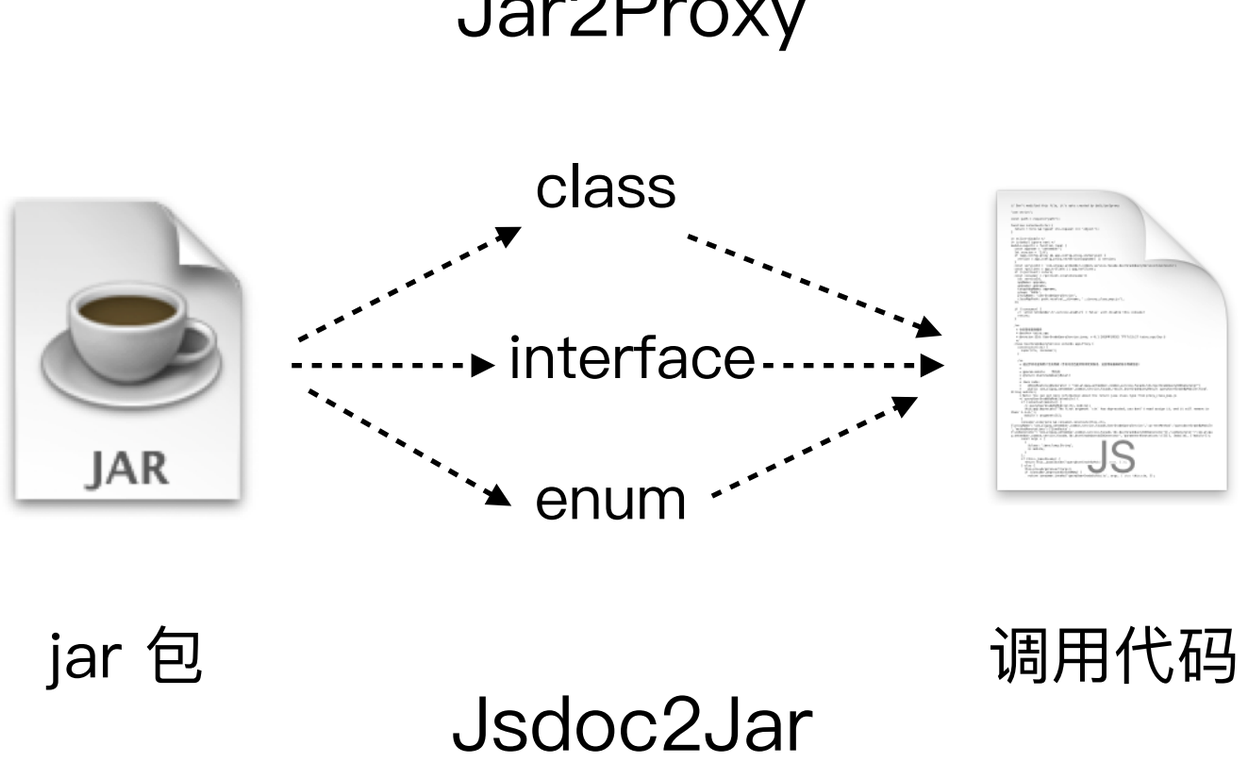
- JS 是弱类型语言，在序列化过程中要转换成强类型的 Java，需要对类型进行额外的描述

```java
package com.alipay.test;

class TestObj implements Serializable {
    private String name;
    private Integer age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

```javascript
const arg = {
  $class: 'com.alipay.test.TestObj',
  $: {
    name: {
      $class: 'java.lang.String',
      $: 'GAO',
    },
    age: {
      $class: 'java.lang.Integer',
      $: 34,
    },
  },
};
```

跨语言类型映射问题

- 通过一个工具来解析 jar 包，从包里面导出接口的元数据，然后根据元数据自动做类型映射



抽象类的问题

- 如果接口入口参数类型是确定，那么我们可以通过工具来做类型映射，但如果接口参数是抽象类，那就需要调用者在调用时显示的指定，这个对于 JavaScript 开发者是很不友好的
- 所以，我们推荐的最佳实践是：RPC 接口类型尽量是确定，并且尽量使用常规的类型

Long 类型的处理

JS 的基本类型里面表示数字的只有 Number，它能够表达的整数范围是 –(2^53 – 1) ~ (2^53 – 1)，而 Java 里面的 Long 类型的范围是 –(2^64 – 1) ~ (2^64 – 1)。那么在 RPC 调用中遇到 Long 类型我们该如何处理呢?

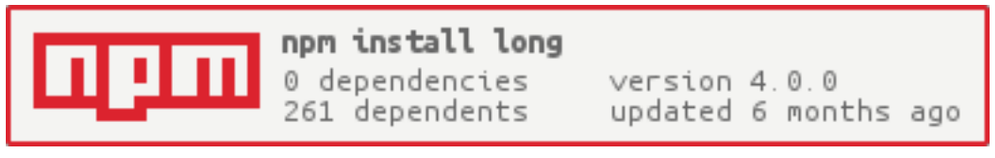
Long 类型的处理

一个 Long 数字占用 8 Bytes，我们可以把它拆分成两个 32 位整数（各占 4 Bytes）来表示，分别称之为「高位」和「低位」，低位存储的是长整形对 2^32 取模后的值，高位储存的是长整形整除 2^32 后的值

```
Long: 45565600000000

High: 10609 Low: 291956736
+-----------+-----------+
|00 00 29 71|11 66 e8 00|
+-----------+-----------+
```

```
Long: 1000

High: 0     Low: 1000
+-----------+-----------+
|00 00 00 00|00 00 03 e8|
+-----------+-----------+
```

```
Long: 4294967296

High: 1     Low: 0
+-----------+-----------+
|00 00 00 01|00 00 00 00|
+-----------+-----------+
```



我们可以用 Long 模块来处理 Long 类型

谢谢！欢迎交流