

Vue SSR和API Proxy实践

—— 前端工程构建实践

i5ting(狼叔)

机票事业部/前端架构组

i5ting: 一个开源爱好者



大纲

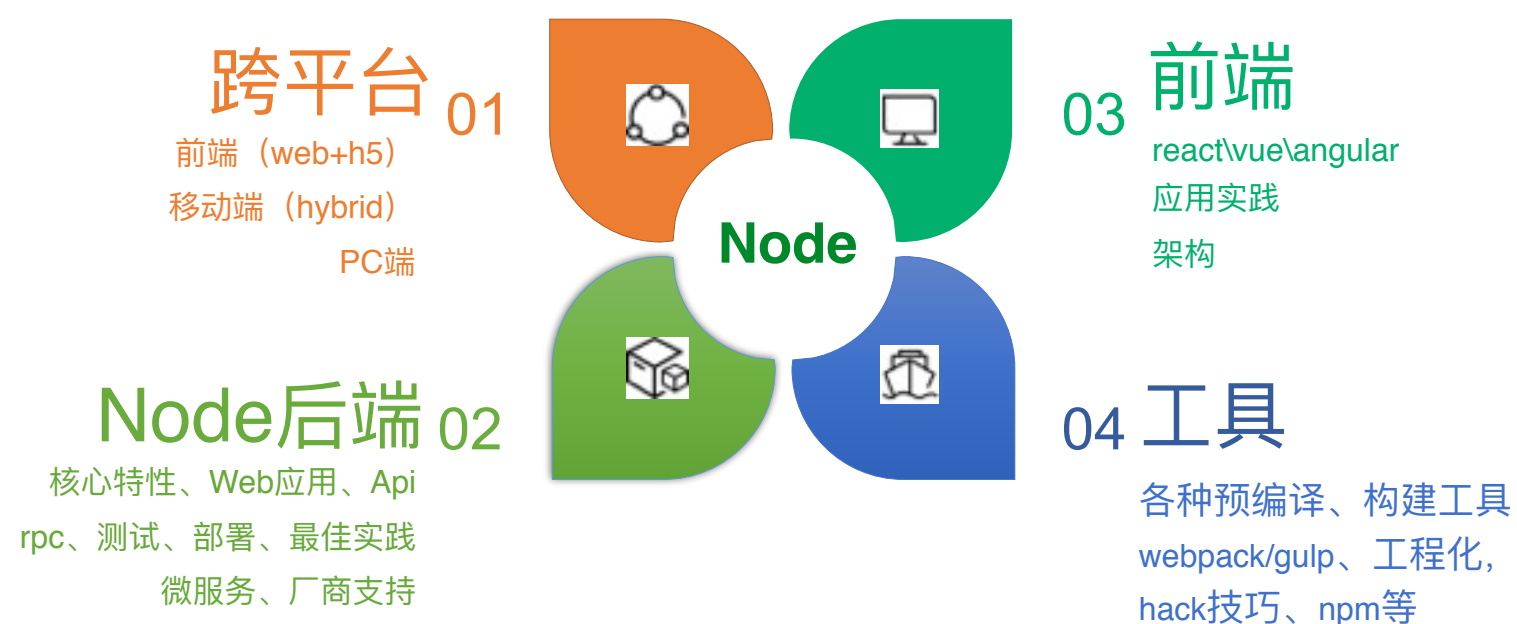
1. 分析Node.js现状，以及2017年趋势预测
2. Vue SSR原理
3. 举例并引出Api层诸多思考
4. API Proxy层深度实践
5. Vue 与 API Proxy如何完美组合

1、分析Node.js现状，以及2017年 趋势预测

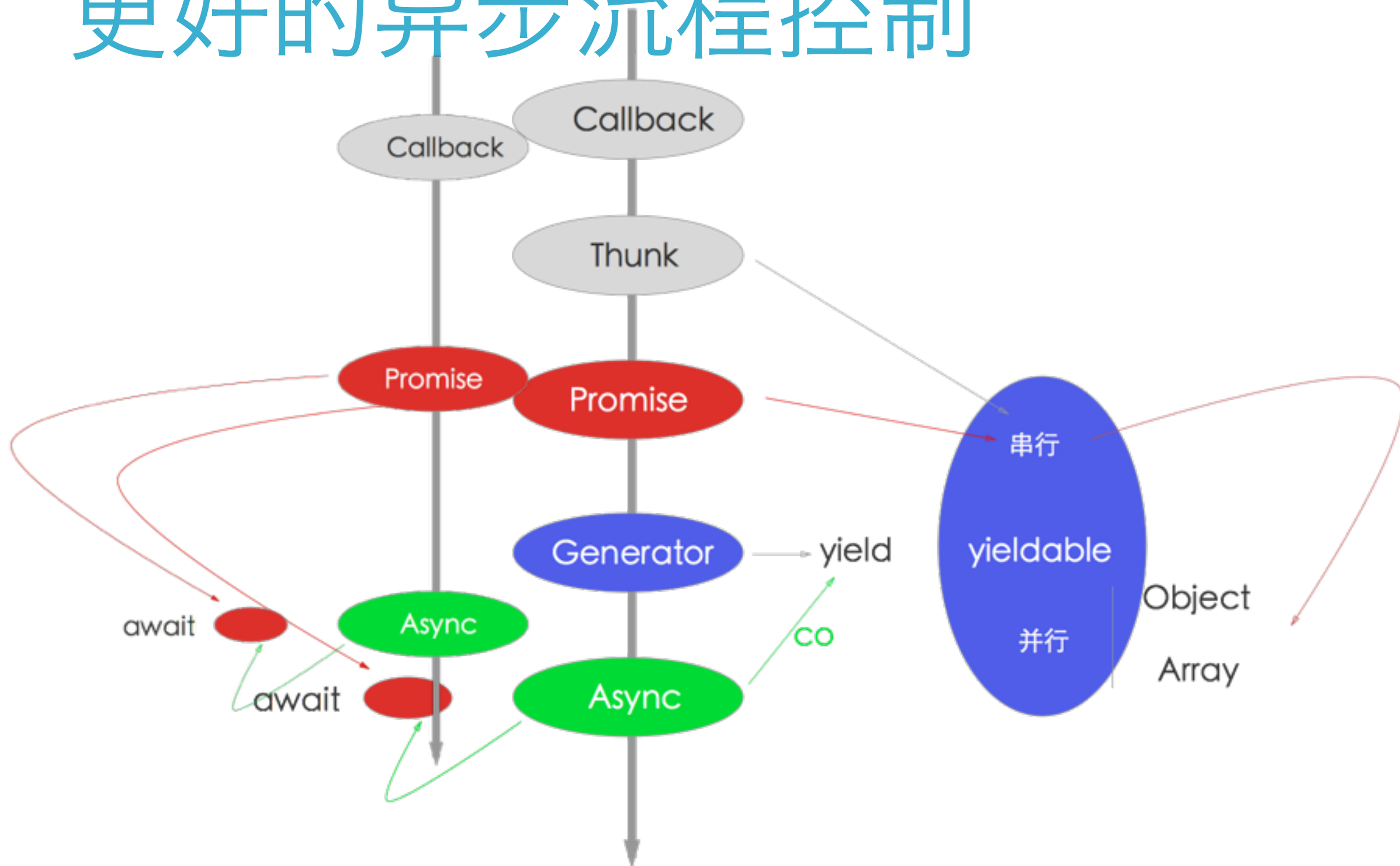
这是前端还是Node ?



更了不起的Node.js



更好的异步流程控制



2017年流行趋势



- PWA
- SSR
- API Proxy
- Isomorphic

Node.js将继续火爆

2、Vue SSR原理

Vue的ssr

- webpack插件内置
- BundleRenderer (Hot-reload && source map support)
- streaming/bigpipe
- cache
- 内置service-worker.js

Server Side Render

- Webpack
- Bigpipe
- Cache
- Isomorphic

Better SEO

Faster time-to-content

Webpack



脚本加载

```
1 <script src="jquery.js"></script>  
2 <script>$.uiBackCompat = false;</script>  
3 <script src="jquery-ui.js"></script>
```

window

eval

iframe

\$.getScript()

依赖管理

- 下载某个库或插件
- 下载它的依赖，以及依赖的依赖
- 无穷尽...

如果升级版本呢?

模块系统

- 使用标准的模块系统来处理依赖和导出
- 每个文件是一个模块
- 使用模块加载器或打包器进行处理

AMD, CommonJS, ES6 Modules

```
<script src="jquery.min.js"></script>  
<script src="jquery.some.plugin.js"></script>  
<script src="main.js"></script>
```

index.html

前端写法演进

```
var $ = require('jquery');  
var otherModule = require('other-module');
```

main.js

```
<script src="bundle.js"></script>
```

index.html

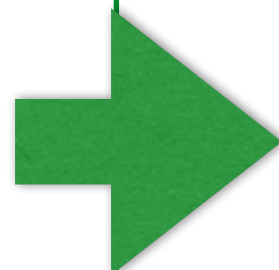
开发环境 module loader

- runs in the browser and loads modules when they are requested
- easy to use
- less optimized for production usage



grunt/gulp

require.js
systemjs



产品环境 module bundler

- runs in preparation and bundles modules into static files
- needs a preparation/build step
- more optimized for production usage

r.js\browsersify\
webpackjspm

Feature - 一切皆模块

在 Webpack 中各种资源都是模块

- JS / JSX / Coffee / Vue
- CSS / SASS / LESS
- PNG / HTML



```
var foo = require('./foo.js');  
var bar = require('./bar.js');  
require('./style.scss');
```

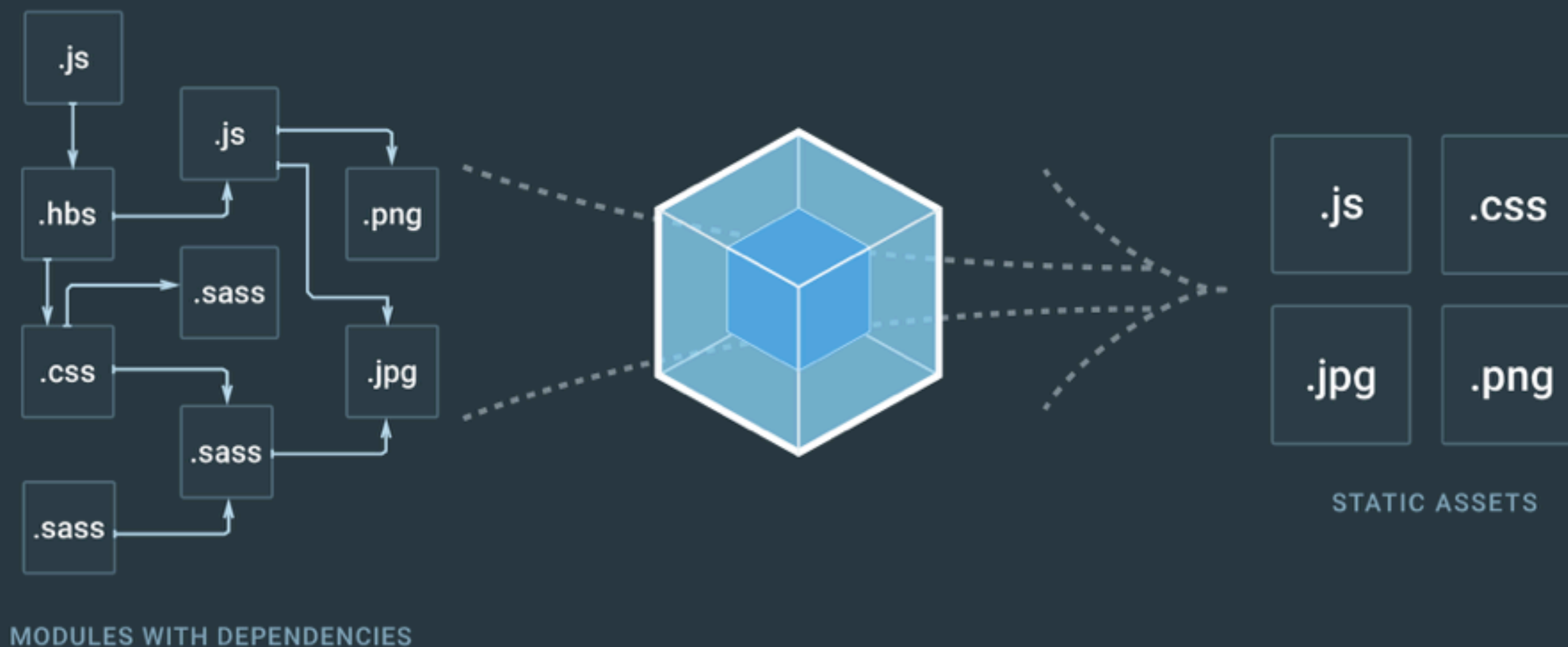
main.js

Feature - Code Splitting

在恰当的时机加载需要的模块 (when & what)

- 第三方库代码分离 (Vendor)
- 样式分离
- 异步加载分离

bundle your assets



这代表可以将任何资源进行模块化
使其[可控&可重用](#)，也可以自定义模块

Webpack打包

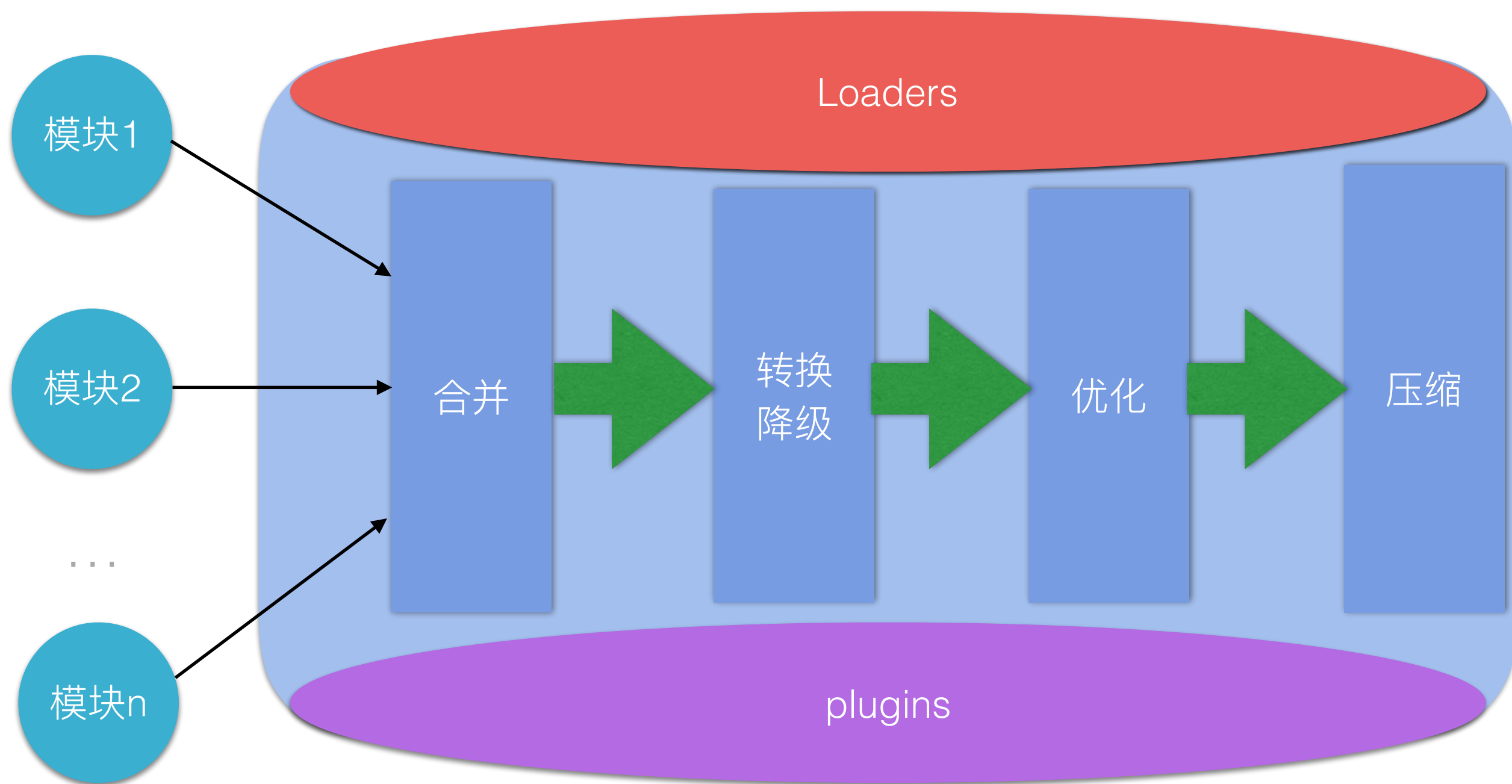
- 从配置文件里找到entry point
- 解析模块系统，解决依赖
- 模块依赖处理（读取，解析，解决）
- 合并所有使用的模块
- 合并模块系统的运行时环境
- 产生打包后的文件

浏览器加载步骤

- 通过<script>加载webpack打包后的文件
- 加载模块运行时环境
- 加载entry point
- 读取依赖
- 解决依赖
- 执行（带有依赖的）entry point

演进过程

- `<script>` 混乱加载
- 各种模块系统标准, `commonjs/amd/es6 module`
- 模块加载器, `requirejs/sea/systemjs`
- 模块打包器, `webpack/r.js/jspm/browserify`
 - A. 合并入口, 对外暴露的只有 entry point
 - B. 提供浏览器运行环境 (内置模块加载器)
 - C. 优化 (tree-shaking、DCE 无用代码移除等)



webpack

你只管写就好了

其他的webpack来

你能写好，就好了。。

你能写，就好了。。

你能，就好了。。

你，就好了。。

打包器演进

- browserify
- webpack 1
- rollup
- webpack 2

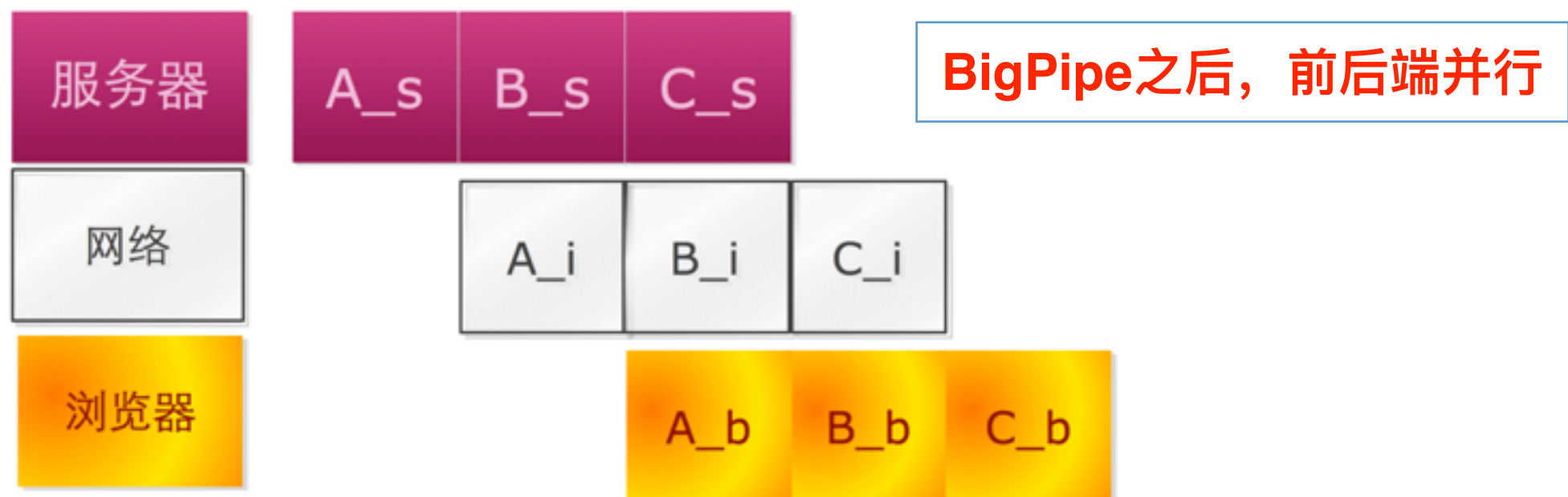
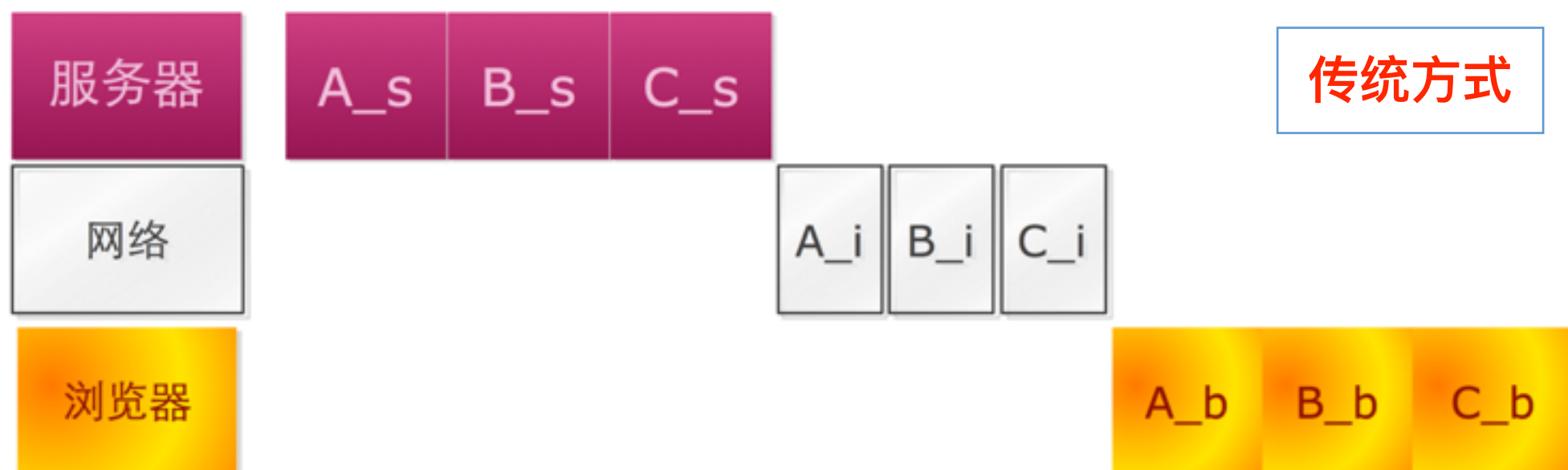
commonjs规范

多种模块

利用es6模块能静态分析语法树的特性，只将需要的代码提取出来打包，能大大减小代码体积

Bigpipe 分块加载技术

- Transfer-Encoding: chunked
- Nodejs自动开启 chunked encoding
 - res.write()

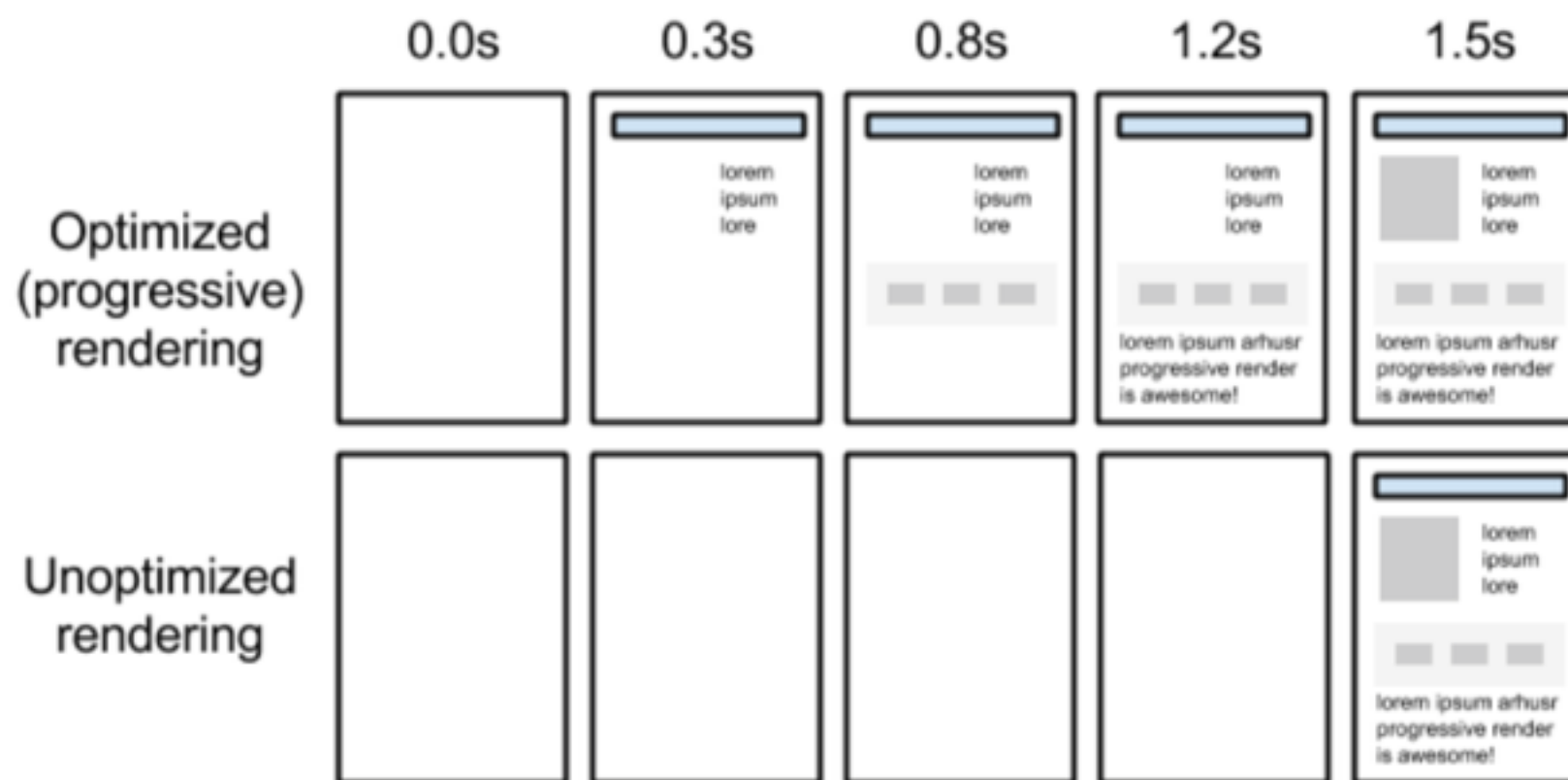


The screenshot displays a flight booking interface for the route from Beijing to Singapore. It lists four flight options with their respective departure times, durations, arrival times, and含税总价 (Total Price including tax). The flight details are as follows:

Flight Option	Departure Time	Duration	Arrival Time	含税总价 (Total Price)
02:15 首都机场 马来西亚亚洲航空长途公司 亚洲航空	02:15	8小时55分	11:10 樟宜机场	¥1448
00:05 首都机场 新加坡航空	00:05	6小时15分	06:20 樟宜机场	¥2724
16:35 首都机场 新加坡航空	16:35	6小时25分	23:00 樟宜机场	¥2750
15:35 首都机场T3 中国国航	15:35	6小时5分	21:40 樟宜机场T1	¥2858

Below the flight list, a network analysis tool (likely Chrome DevTools) is shown, displaying the Network tab. It provides a detailed view of the network requests, including a timeline and a list of requests. The requests are categorized by type (XHR, JS, CSS, etc.) and show the time taken for each request. The first request, 'flight_list?startCity=%E5%8C%97...', is highlighted, showing a response time of 1.5s.

首屏渲染时间



TTFB (Time To First Byte)

Cache缓存

- 编译缓存
- LRU (least recently used) 最近最少使用
- 越来越后端

Isomorphic同构

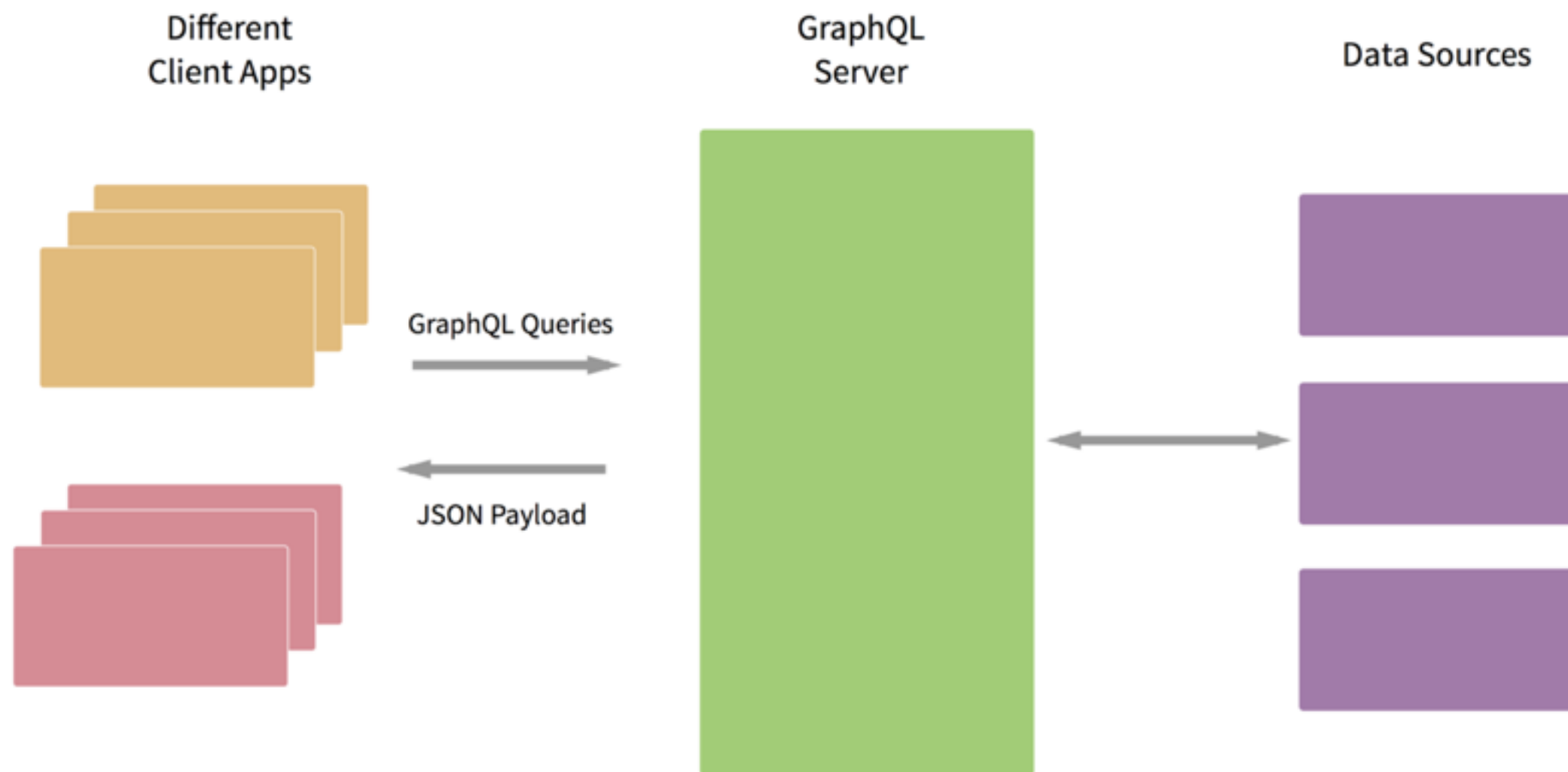
- 模糊了前后端边界
- 全栈是个好词儿

看上去很美



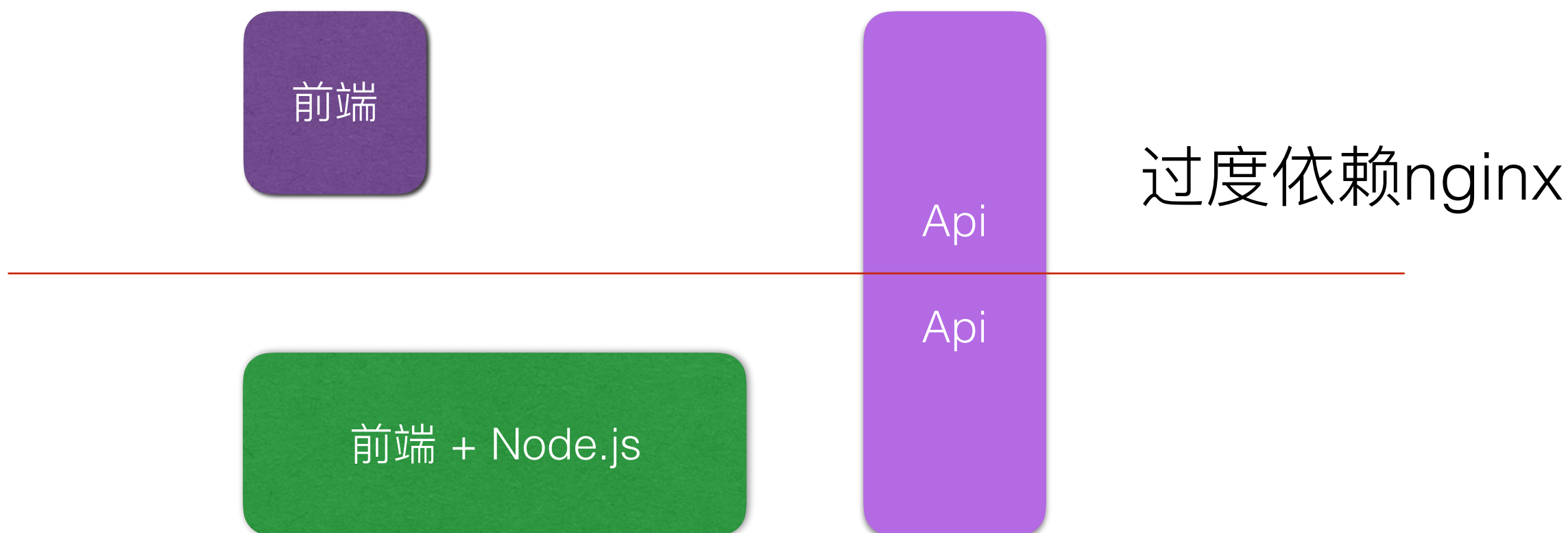
API

复杂应用



前后端分离

- 纯静态化 (http server)
- Node.js (node server)



3、举例并引出Api层诸多思考

出票完成

根据规定, 在航班起飞前7天后将不能打印行程单, 若需要请及时与代理商联系

订单总价 ¥1296 ?

订单号 xyz3698763582344566

下单时间 2016-11-25 12:23:48

报销跟踪 报销凭证尚未支付快递费用 立即支付 ¥20

改签

退票

改签需要航司订单号, 您的航司订单号为: 0123456789
紧急退票改签, 请直接联系国航旗舰店服务热线: 10101234

索要报销 打印订单

航班信息

11-24 北京 - 上海

南方航空

CZ6132 空客320 (中) 共享
头等舱

00:20

首都机场T2

1小时50分

11-26 01:50

周水子机场

航班动态 延误 值机柜台 20-30 登机口 102 行李转盘 32F



- 航班信息
- 退改签说明
- 乘机人信息
- 保险信息
- 报销信息
- 专享优惠

退改签费用标准及特殊票务说明

退改签费用标准

类型	退改签时间点	退票扣费	改期加手续费	转签
成人票	起飞前168小时之前	¥348/人	¥1688/人	不可转签
	起飞前168小时之前	¥348/人	¥1688/人	
	起飞前168小时之前	¥348/人	¥1688/人	

特殊票务说明

- 行程单价格低于实际支付价格, 差价可提供发票, 差价不退还
- 机场无法打印行程单, 如需报销请选择快递行程单 [查看全部](#)

乘机人

隋小隋六七个字

成人套餐

头等/商务舱

身份证 110302 **** * 3322

票号 9992 6543 33232

联系人 隋小隋

卢小鱼

儿童票

经济舱

身份证 110302 **** * 3687

票号 9992 6543 33232

手机号 (+86)136 ****2234

保险

隋小隋

航意险

已支付

保单号 129207 27654 89270 [查询](#)

延误险

已支付

保单号 129207 27654 89036 [查询](#)

航意险打印保单中国人寿保险股份有限公司

卢小鱼

航意险

已支付

保单号 129207 27654 89270 [查询](#)

延误险

已支付

保单号 129207 27654 89036 [查询](#)

延误险打印保单永呈财产保险股份有限公司

业务特点



可选

模块不是必须都显示的，某些不显示没太大影响



可并行

并行，是最快的，Node.js强项



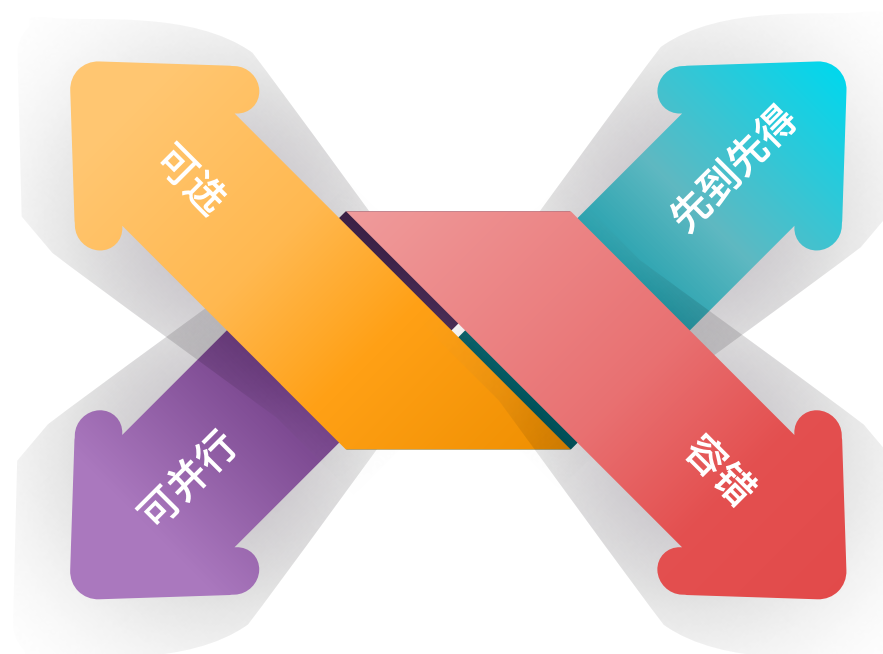
先到先得



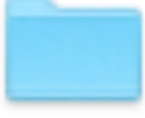






模块的加载速度取决于模块自己身的执行速度。如果都是静态模板，那就是静态

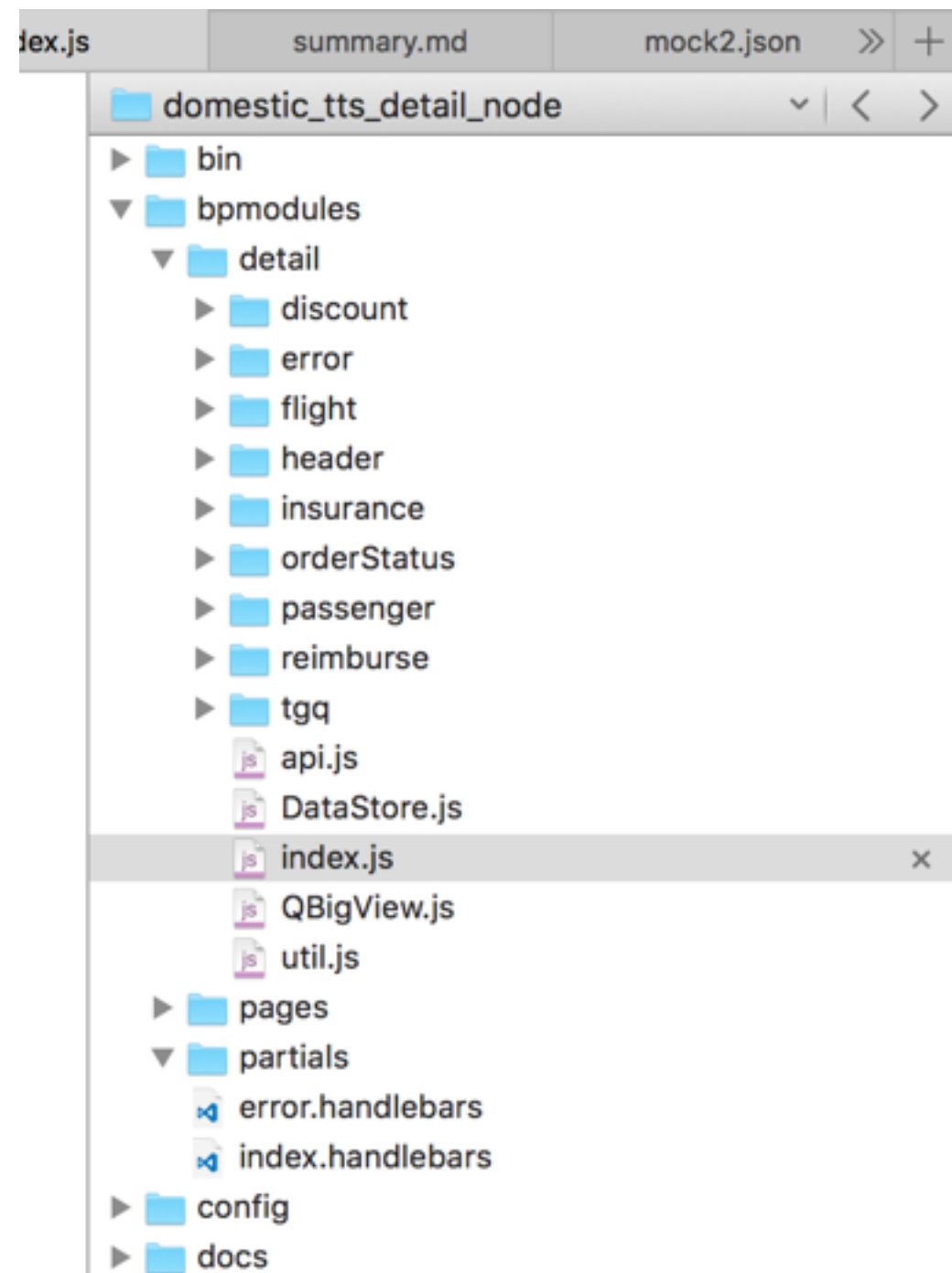


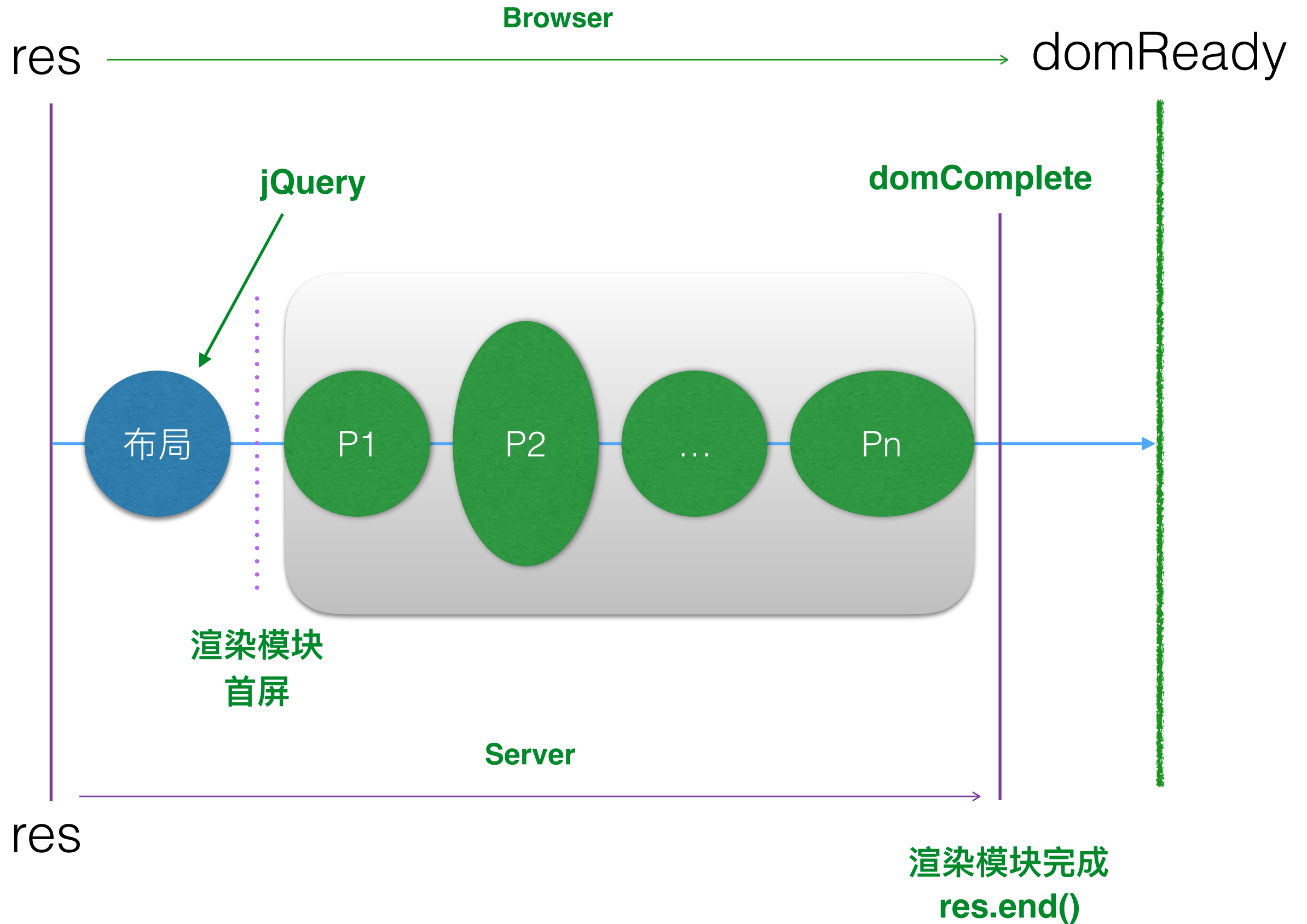
容错

模块错误，显示或不显示，显示成啥都有模块自己决定



ve	
1	▶  discount
2	
3	▶  error
4	
5	▶  flight
6	
7	
8	
9	▶  header
10	
11	
12	
13	▶  insurance
14	
15	
16	▶  orderStatus
17	
18	
19	
20	▶  passenger
21	
22	
23	
24	▶  reimburse
25	
26	
27	▶  tgq





Bigview是Node.js模块

使用Node.js编写，高效稳定，并且对前端友好。尤其是模块化上，可以做的更多，将bigview抽象为bigview和biglet，可以更好的控制。
npm强大的生态，对于模块、依赖管理、测试等有非常大的方便

模块化

模块化，独立发布
具有可测试性

01

子模块

支持子模块和子模块渲染模式
可以无限级嵌套

02



渲染模式

支持5种bigpipe渲染模式
完美兼容各种模式

03

布局

支持动态布局和静态布局
定制更加容易

04

Api问题

- 一个页面的Api非常多
- 跨部门
- 跨域
- Api返回的数据对前端不友好
- 需求决定Api, Api不一定给的及时

Api Proxy层

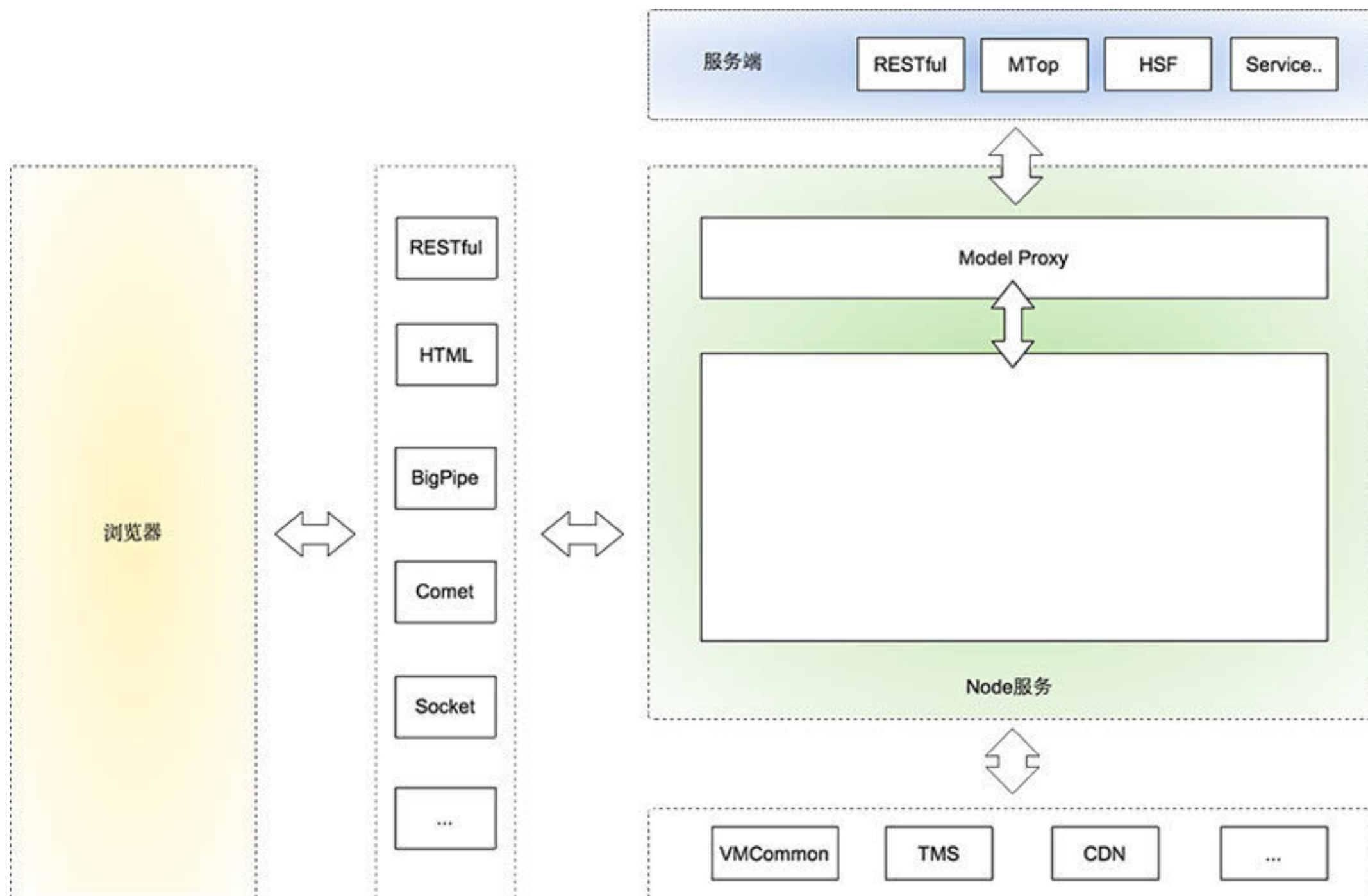
- 产品需要应变，后端不好变，前端更容易
- 后端讨厌（应付）前端，几种api都懒得根据ui/ue去定制，能偷懒就偷懒



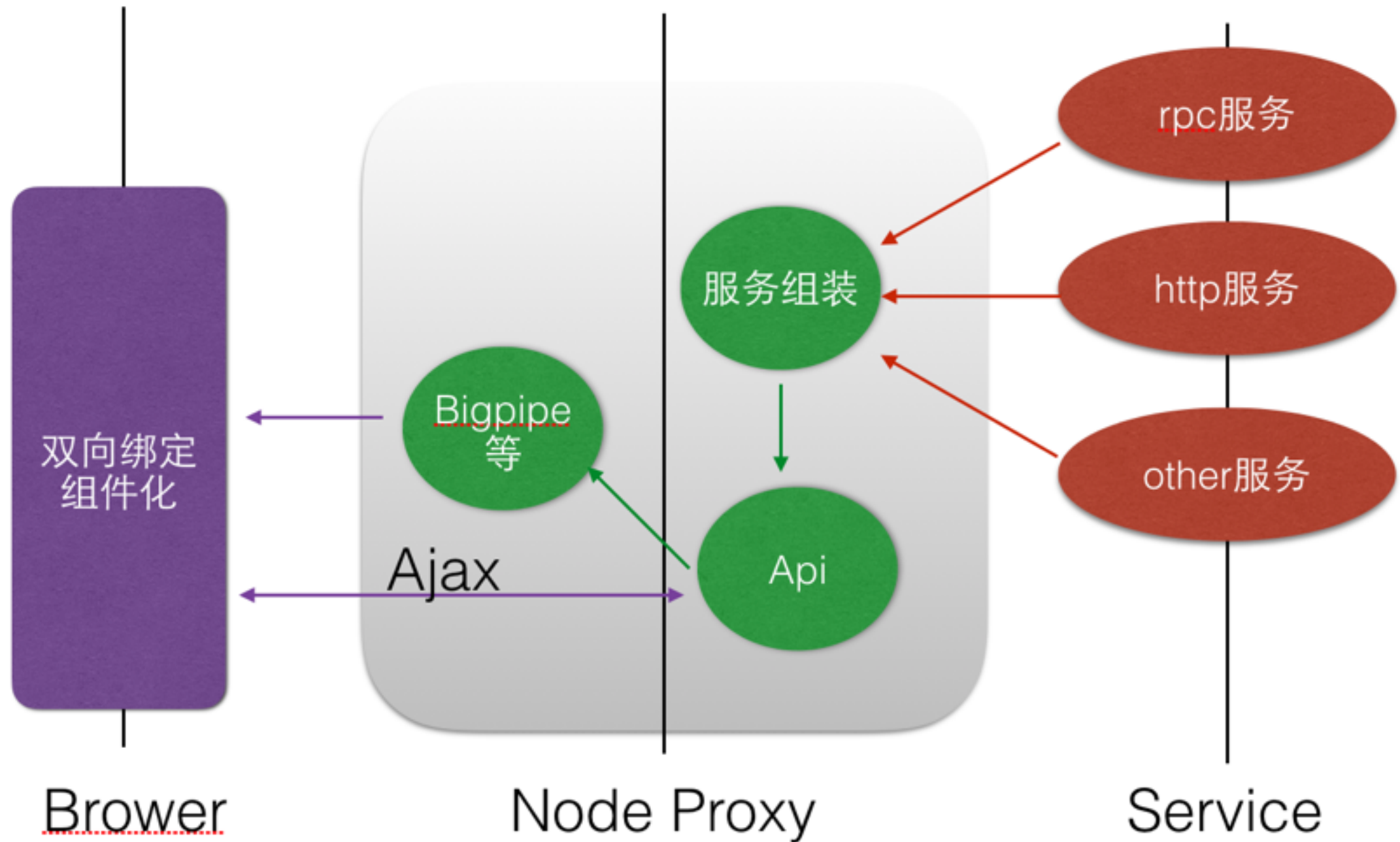
可怜的前端

4、API Proxy层深度实践

淘宝的做法



Api Proxy层



时间呢?

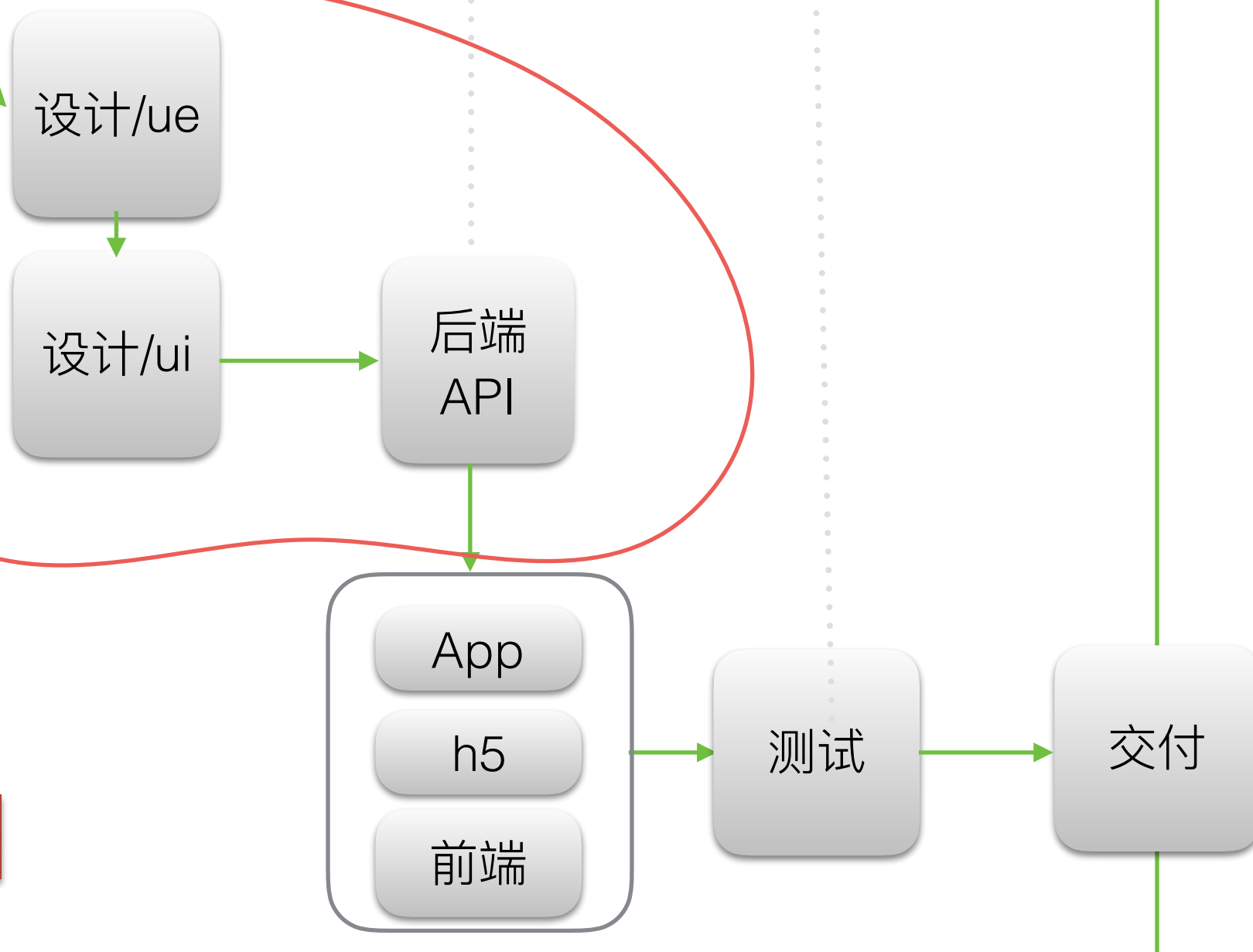
常规



改进前



相互依赖、扯皮?

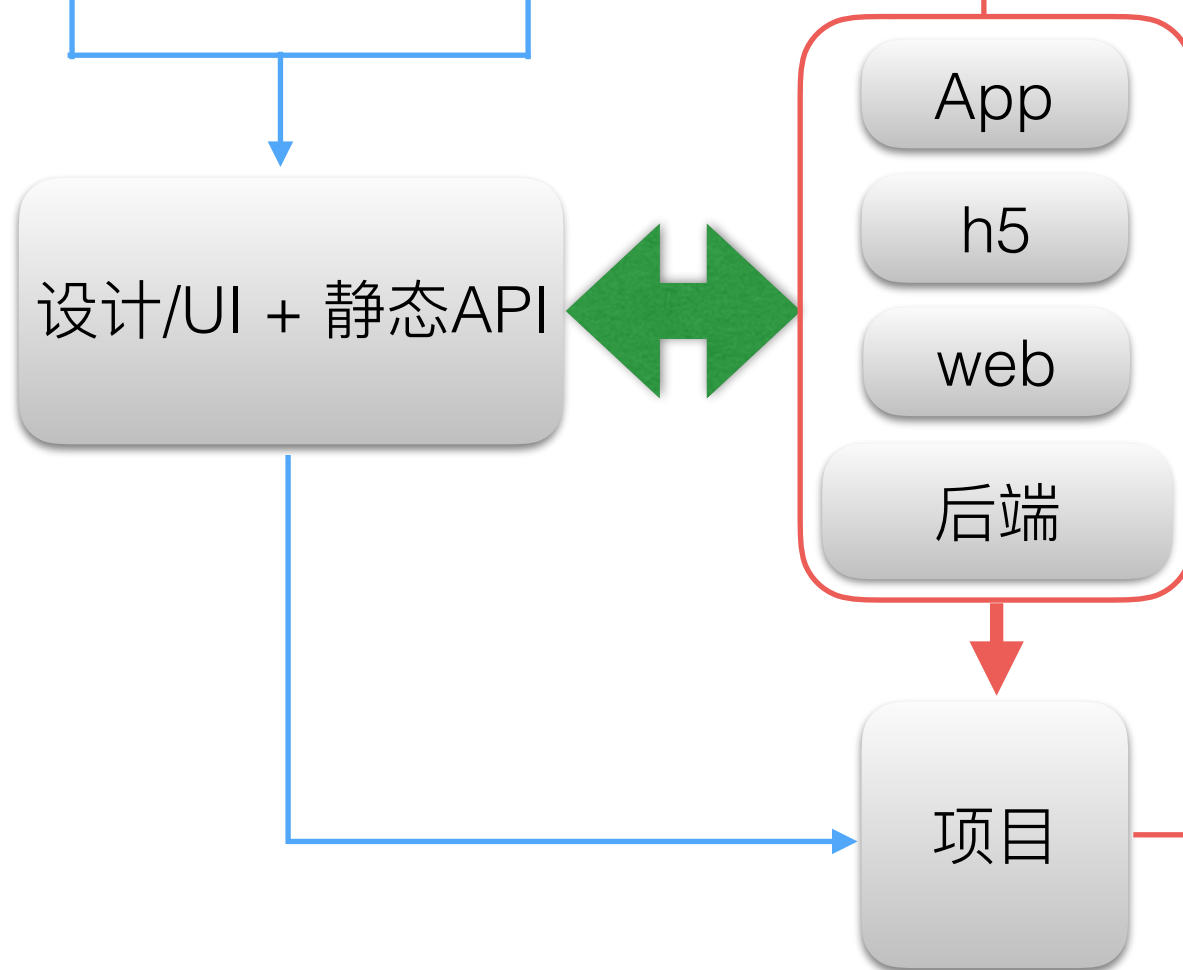


并行开发流程改进

常规



改进后



记录请求过程

- 类似于postman
- 记录request和response
- req和重复使用
- res可以做api反向检查
- 生产网络请求代码

解决了网络请求部分代码复用已经很多了

代码生成器

- 有了请求过程
- 约定代码结构
- 编写脚手架

vdom

双向绑定

data store

网络请求

Case问题

- 难以重现
- 无法复用
- 环境依赖严重
- 反复找qa, 捞日志等

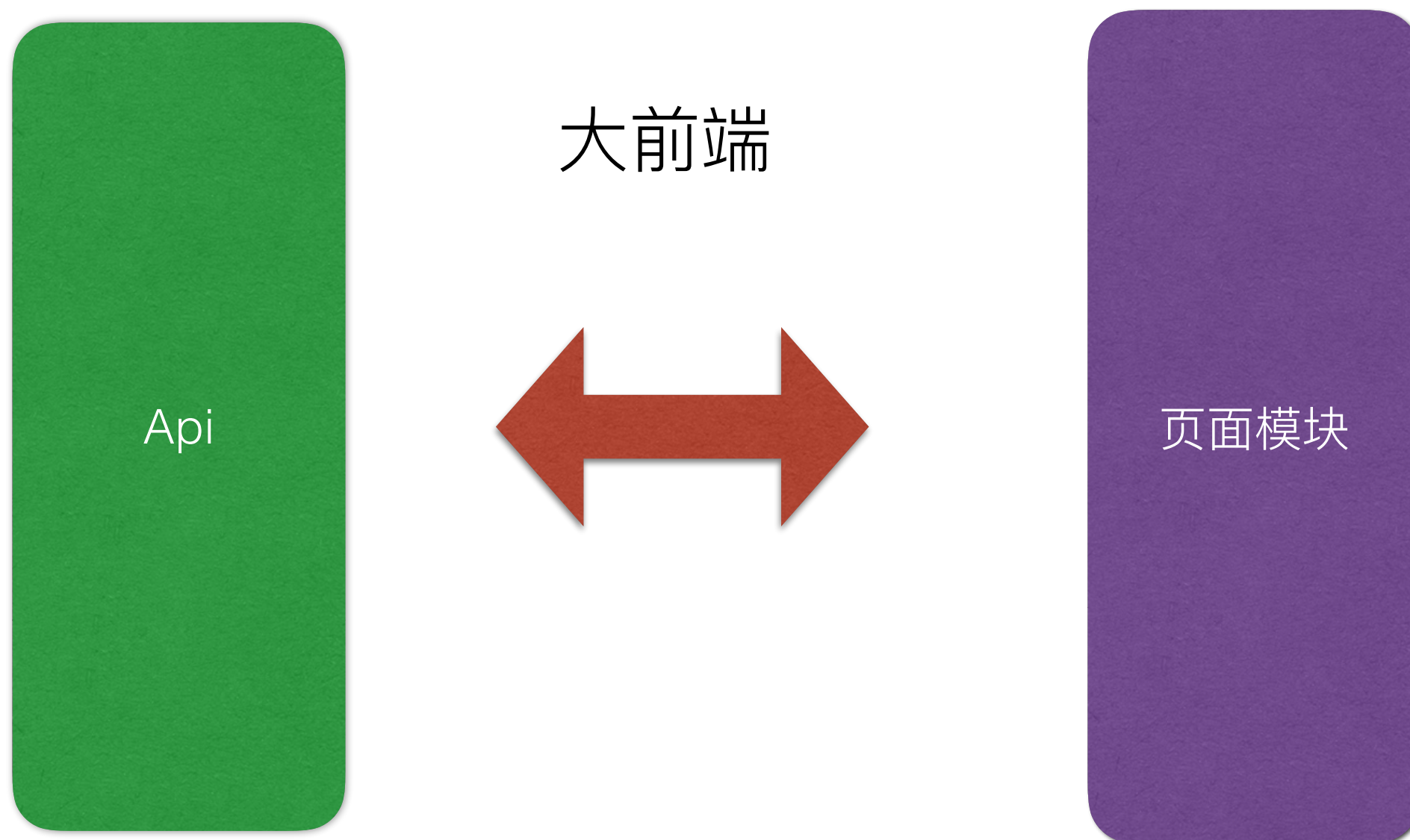
场景多
接口多
人多
部门多

复杂即协调

开发的问题

- 反复和server端确认接口
- 测试依赖环境严重，跨部分沟通更麻烦
- 人员变动、需求变动，找不到人
- 多项任务切换，没有一个事儿是痛快结束的
- 为业务而业务，心累。。。

如果模块化，还剩下什么



没用的mock api



mock api变了，谁维护？

只增加工作量，为啥要维护2份？

fe方便了，后端没方便啊。。。

它可以有用



作为需求和验收标准

反向测试server api

5、Vue 与 API Proxy如何完美组合

3) 生成网络请求代码

4) 反向压测api和检查

1) vue特性

vdom
双向绑定
组件化

Bigpipe
等

Ajax

服务组装

Api

rpc服务

http服务

other服务

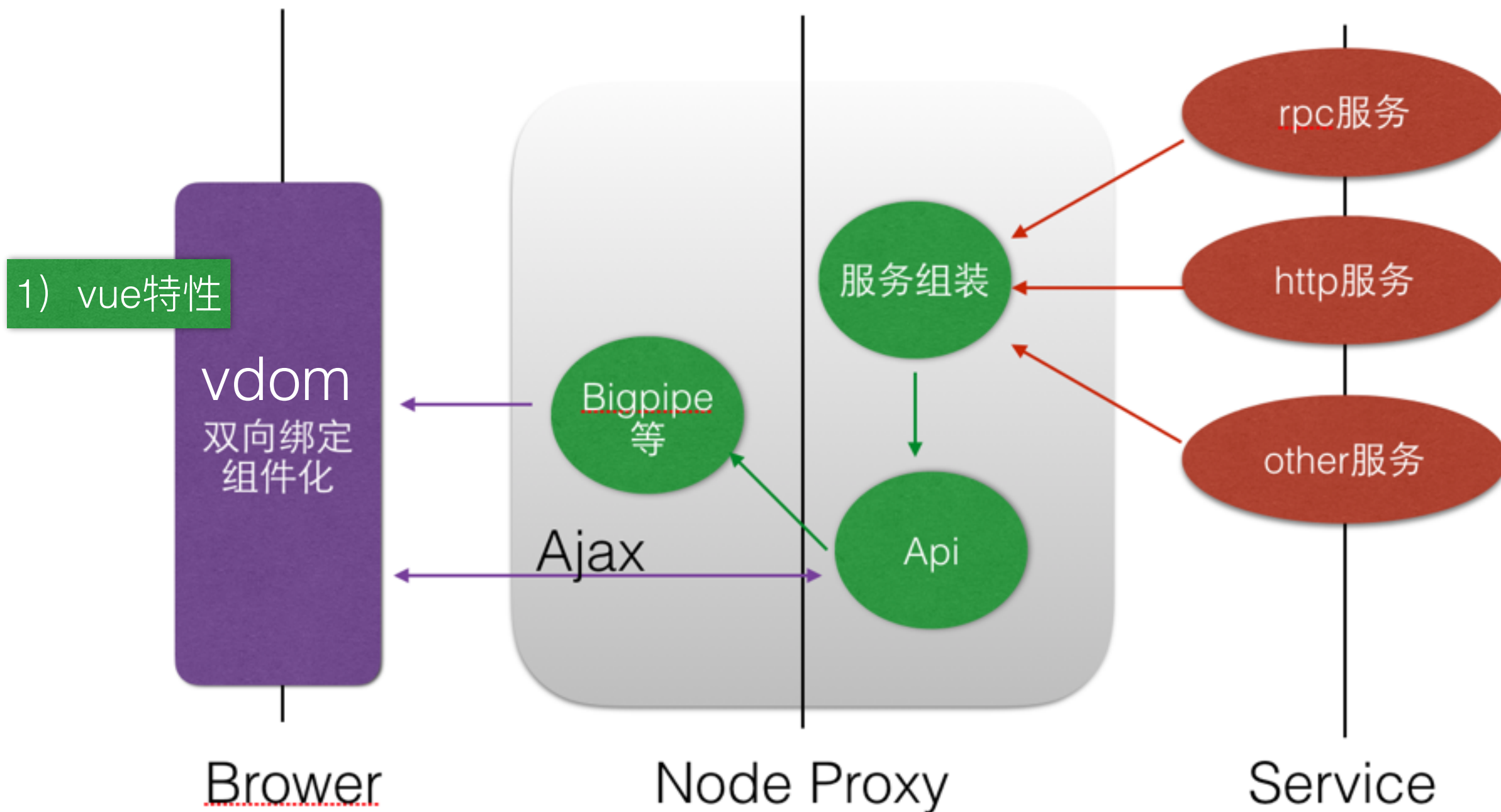
Brower

Node Proxy

Service

2) mock提高开发效率

5) 对store进行封装



谢谢大家

Q&A

狼叔说：少抱怨，多思考，未来更美好