百度外卖前端那些事

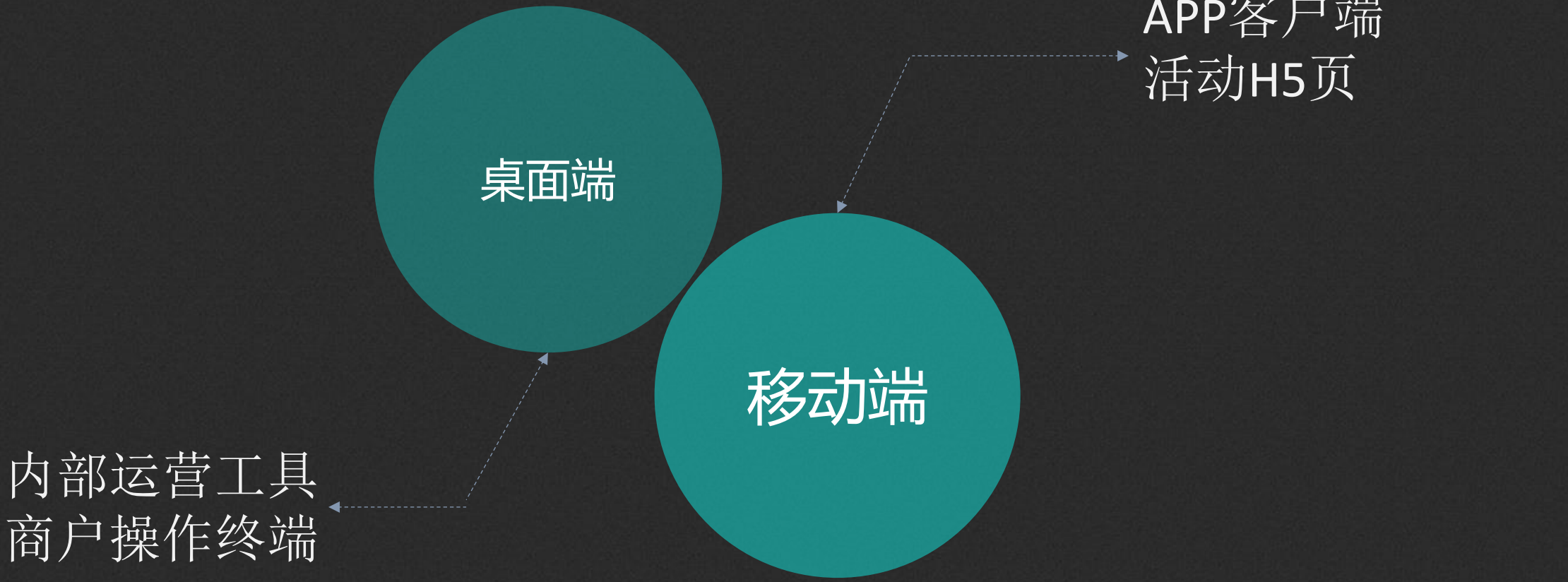
在前端方面的经验和尝试

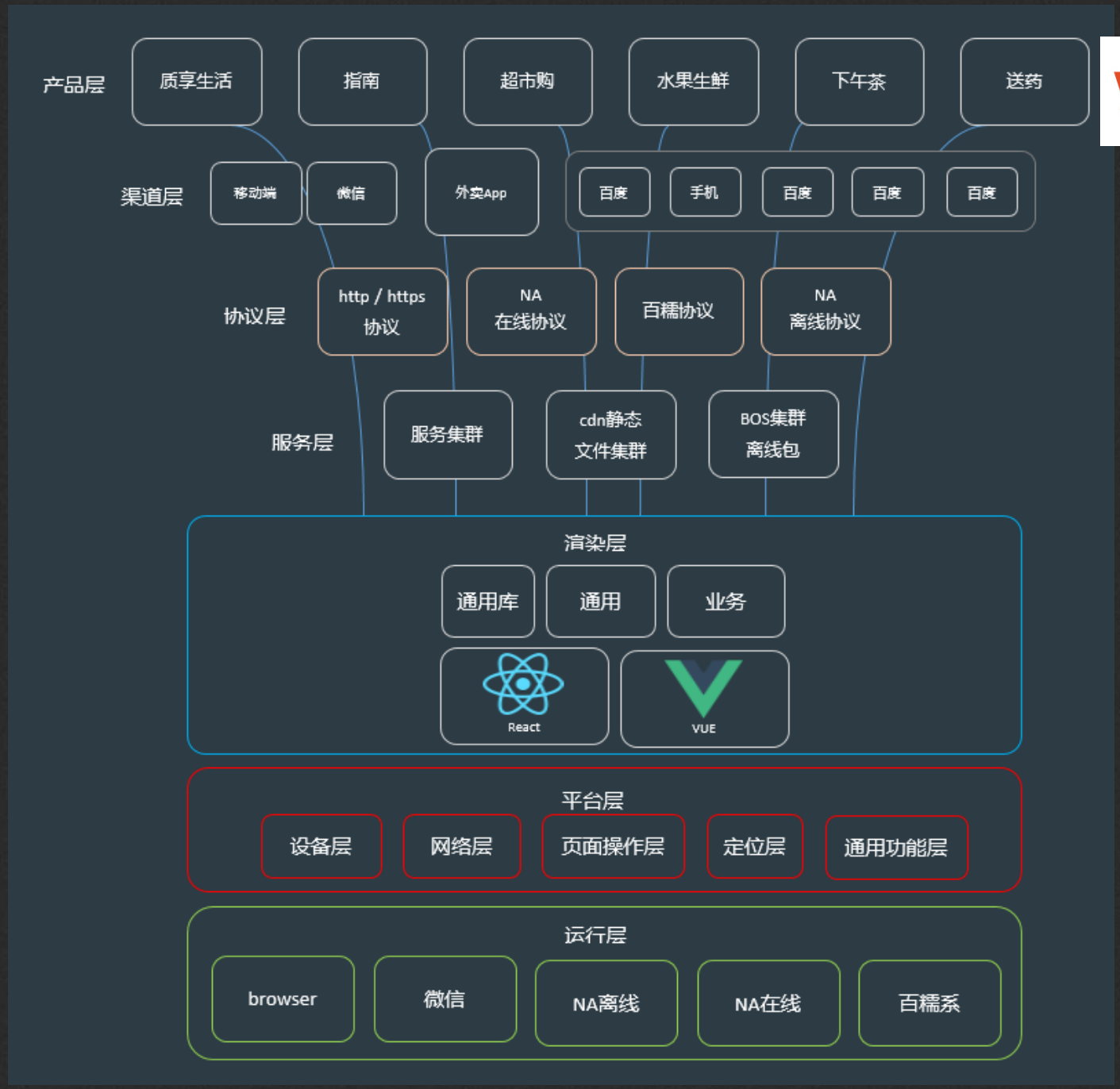
目录
CONTENT

移动端前端架构

原生与H5合力支持快速发展变化的市场需求

百度外卖前端产品结构




产品层
质享生活
指南
超市购
水果生鲜
下午茶
送药
渠道层
移动端
微信
外卖App
百度
手机
百度
百度
百度
协议层
http / https
协议
NA
在线协议
百糯协议
NA
离线协议
服务层
服务集群
cdn静态
文件集群
BOS集群
离线包
渲染层
通用库
通用
业务
React
VUE
平台层
设备层
网络层
页面操作层
定位层
通用功能层
运行层
browser
微信
NA离线
NA在线
百糯系

原生与H5混合

现在移动端H5已经能达到接近原生的用户体验
而H5的最大优势是跨平台且迭代周期短

一个典型频道页的技术框架

每个频道页都是一个独立的前端项目
独立开发、测试、上线

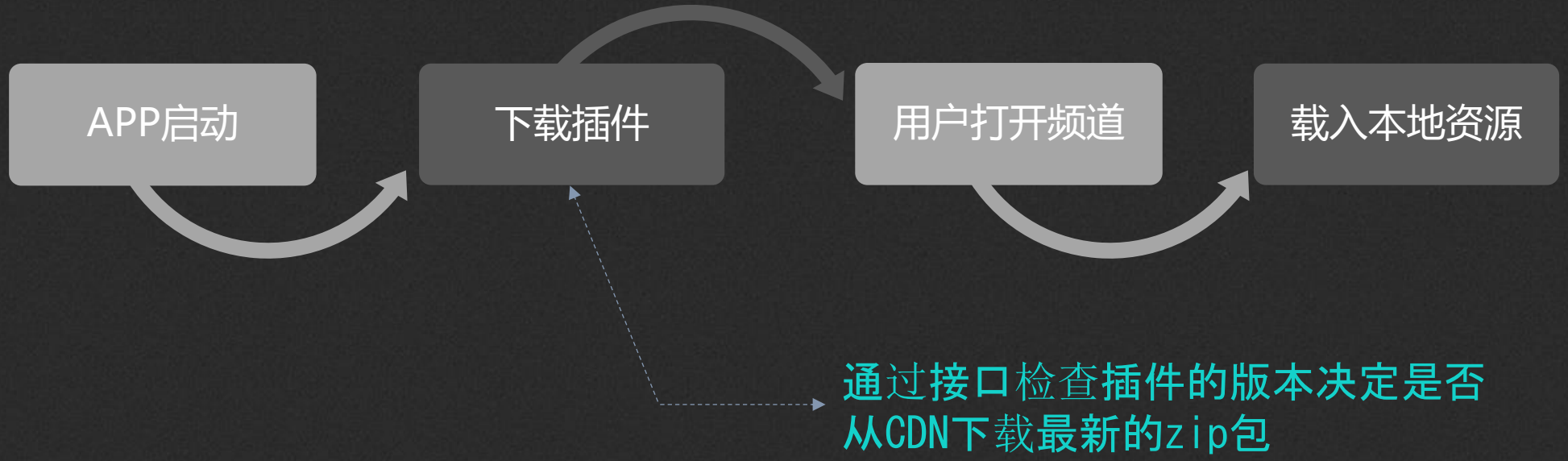
离线化插件模式（一）

现代前端框架之痛：

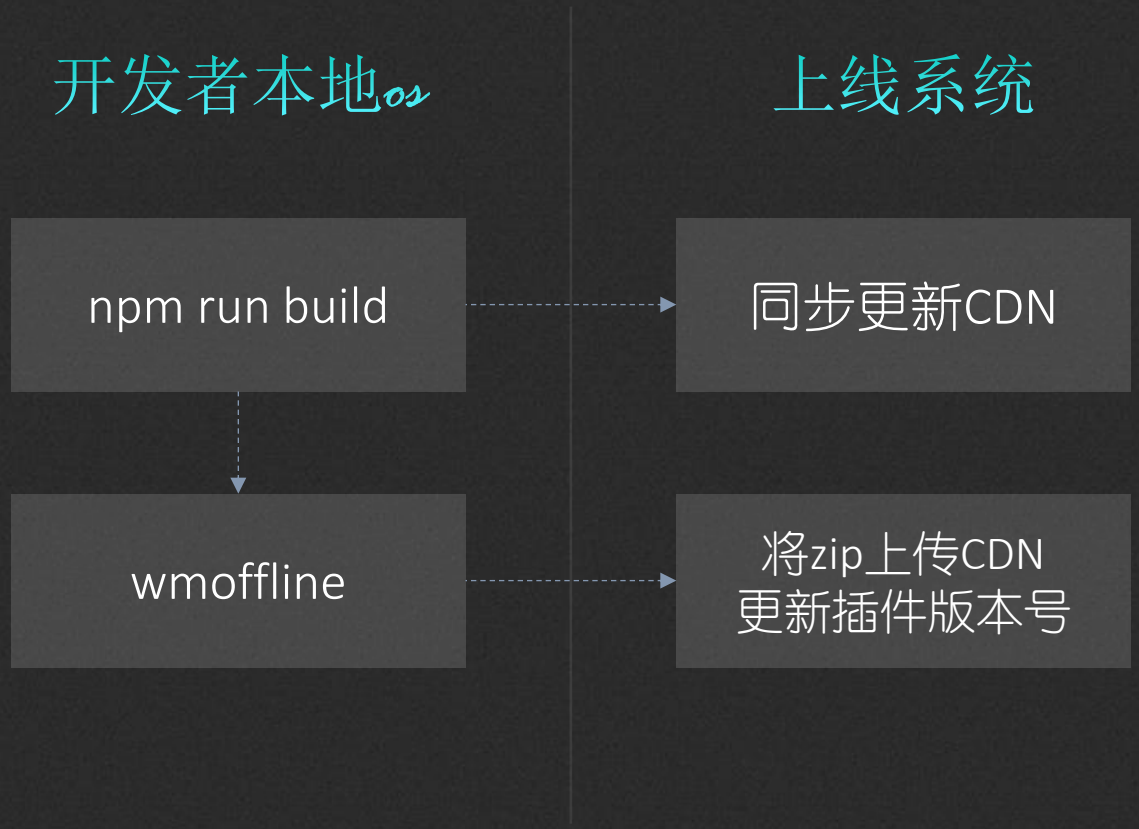
页面必须等JS执行才会展示，JS文件又比较大

白屏时间

离线化插件模式（二）



发布上线方式


开发者本地os
上线系统
npm run build
同步更新CDN
wmoffline
将zip上传CDN
更新插件版本号

开发调试方式（一）

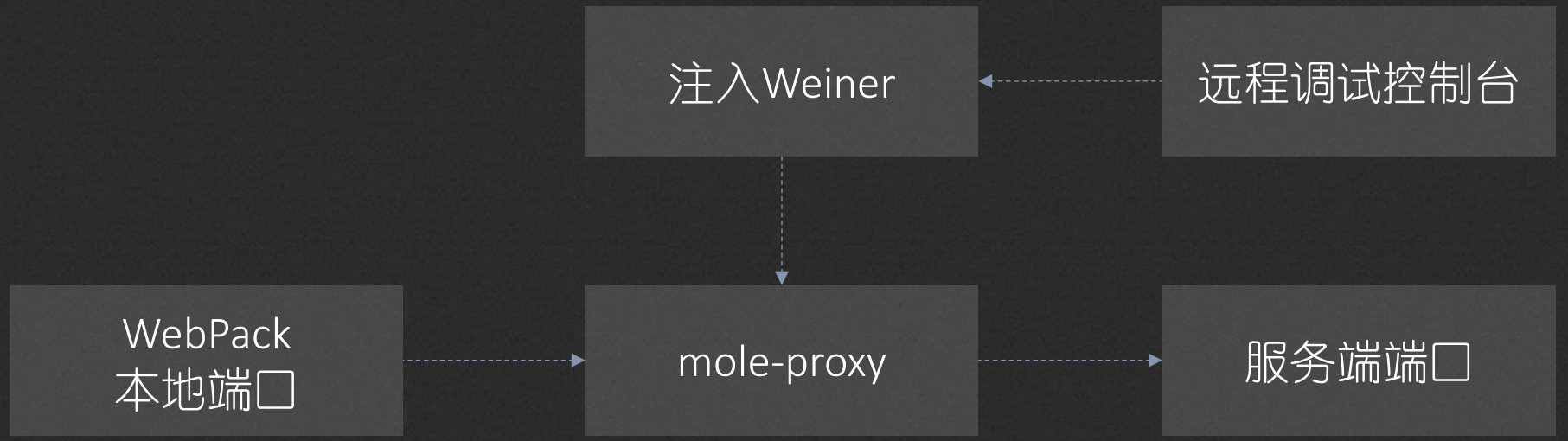
问题：

在PC上开发一个页面
如何在手机上预览效果

开发调试方式（二）

自己开发工具，将本地监听端口映射到服务端
并生成二维码，手机扫描二维码进入开发预览
还可以享受WebPack的热更新功能

开发调试方式（三）

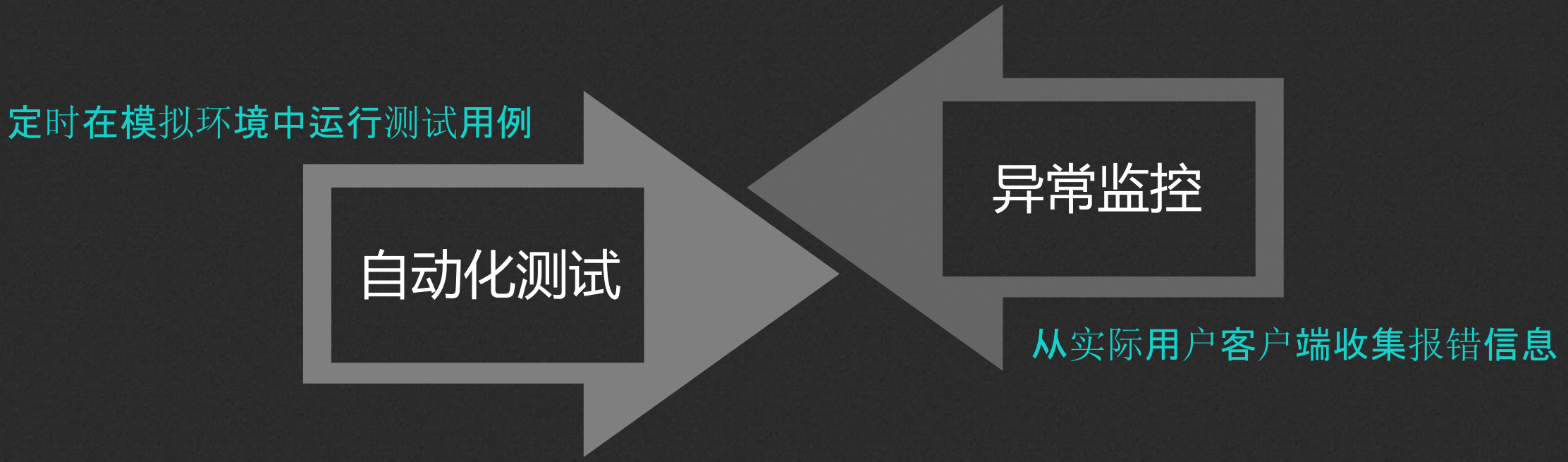


映射端口就不需要手机与开发者电脑必须同网
在代理过程中还可自动注入远程调试

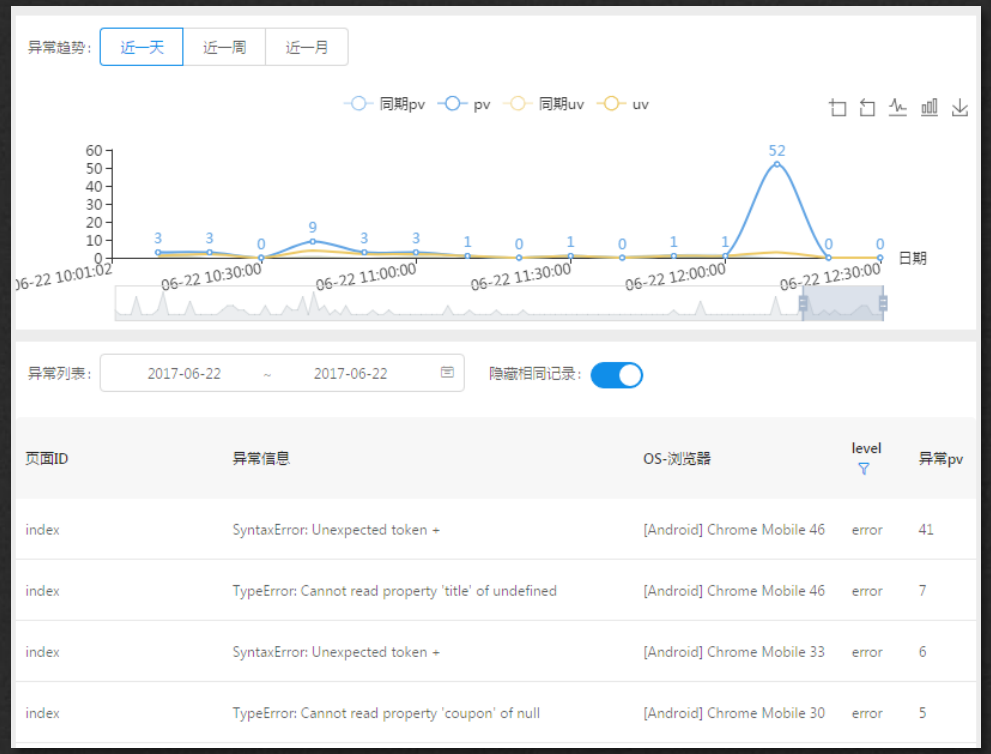
稳定性保障

稳定性是用户体验的第一保障

稳定性的两大平台



异常监控



▶ 问题背景

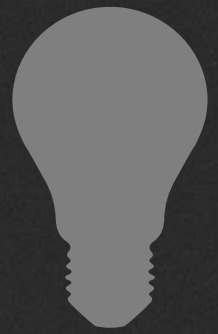
QA很难覆盖所有用户可能的手机机型、网络情况，需要有效的方式收集实际用户遇到的问题

▶ 平台收益

能够实时地获知产品在实际运行过程中，在什么时间点、什么机型遇到了哪些JS报错

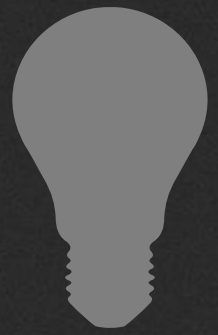
聚合同类错误，并记录错误发生的代码位置，方便开发者修复问题

异常监控技术点



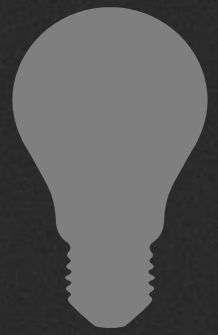
捕获错误信息

利用window.onerror、Error.stack等获取错误信息及位置信息
注意要给<script>设置crossorigin，否则无法回去错误信息



错误位置还原

通过上传sourcemap，平台可以将编译后代码的出错位置还原成源代码中的位置



错误信息聚合

以日志的形式保存客户端上报的信息，利用kafka、zookeeper、ES等分布式框架搭建实时日志数据聚合

自动化测试

问题背景

普通的自动化单元测试很难检查样式是否正确，关键元素是否丢失，需要QA人工确认

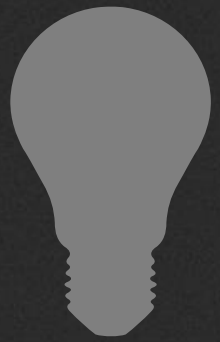
平台收益

自动化执行，降低QA人力消耗

定时执行，先于用户发现线上事故

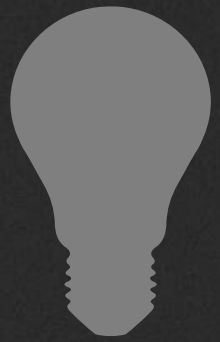

Numen | 百度外卖守护
numen.inwaimai.baidu.com/#/app/develop/ui2
Numen 1.5
anqin
自动化测试
用例设计
API测试
UI测试
用例管理
装配车间
执行历史
手动测试
手动测试
监控中心
线上监控
代码安全
数据监控
Step 1: 基本信息
Step 2: 参数配置
Step 3: 用例设计
Step 4: 用例预览
用例设计
步骤名称
打开网页
目标网址url
http://wmaudit.baidu.com
删除此步骤
始
打开网页
终
点击倒三角形添加步骤
打开网址
在虚拟浏览器的地址栏里输入网址并打开该网址所指向的页面。网址必须以http/https开头
保存

自动化测试控技术点



一套能运行于服务端的浏览器

Ubuntu Debootstrap：提供统一的ubuntu系统环境
Xvfb：提供虚拟GUI环境
Google Chrome：可控制的浏览器
Selenium WebDriver：将Chrome封装成服务



直播被自动操作的浏览器

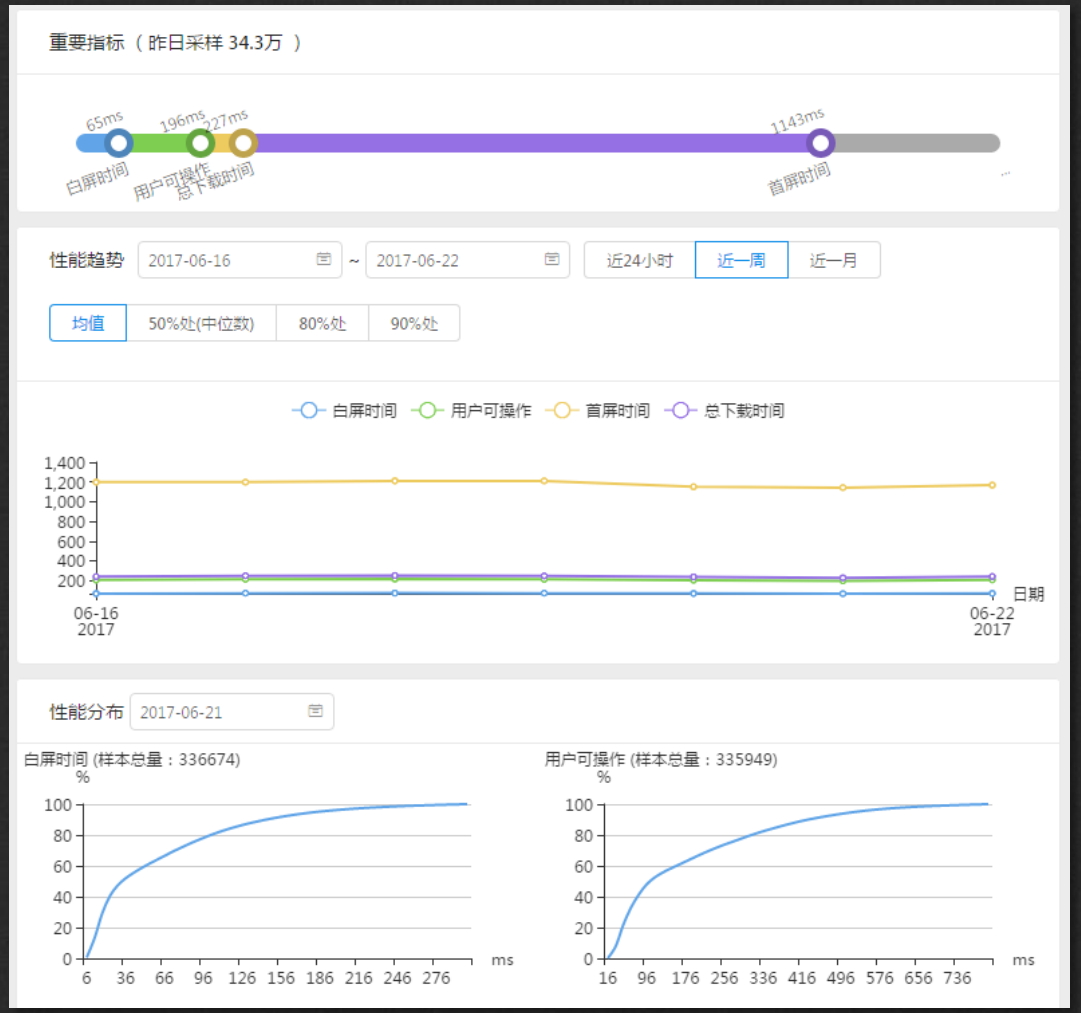
通过SSE实时将执行任务的浏览器画面、console、network等信息发送给直播播放器，使用者可以更直观地把握用例的执行情况

03

性能优化

一切为了更流畅的用户体验

性能监控平台



问题背景

优化性能需要了解当前的性能状况如何，尤其是在实际用户环境中性能表现情况

平台收益

数据可视化后，各时间占比一目了然。在哪里进行优化可能收益最高，以及优化后效果如何，都提供了有力的数据支撑

性能优化点概览

- 离线化
- 分包按需加载
- 图片格式优化
- 提前Ajax请求
- 减少依赖库

分包与按需加载

WebPack

CommonsChunkPlugin

require.ensure

将node_modules中的文件与业务相关代码分开打包
不常变动的内容可以单独被浏览器缓存

按需延迟加载，以最快速度优先加载第一屏

图片格式优化

单色或色彩简单的图标

色彩较复杂的logo或提示图片

不支持WebP的设备

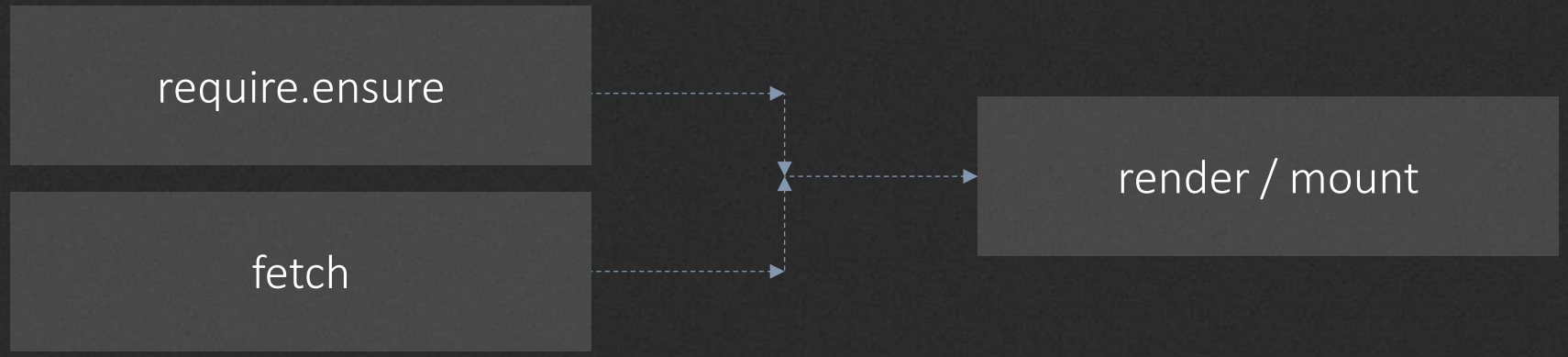
实景拍摄的图片

渲染前进行Ajax请求

串行：

第一次加载的JS比较大，而且在组件开始渲染后再调用
接口会有一次额外的渲染

并行：



接口调用与组件加载同步进行，都完成才渲染

减少依赖库

大而全的第三方库严重占用最终包大小
避免使用或手动提取需要的部分

04

组件化

提升人效与生产力

移动端组件库（目前内部使用）

可与外界共享

slideshow	headline	affix	tab
scroll	input	select	datepicker

与业务或样式强耦合

loading	shoplist

桌面端开源组件库：XCUI

- 基于Vue 2.0
- 主要面向PC端
- 通过npm安装

https://wmfe.github.io/xcui/

运营平台通用模型

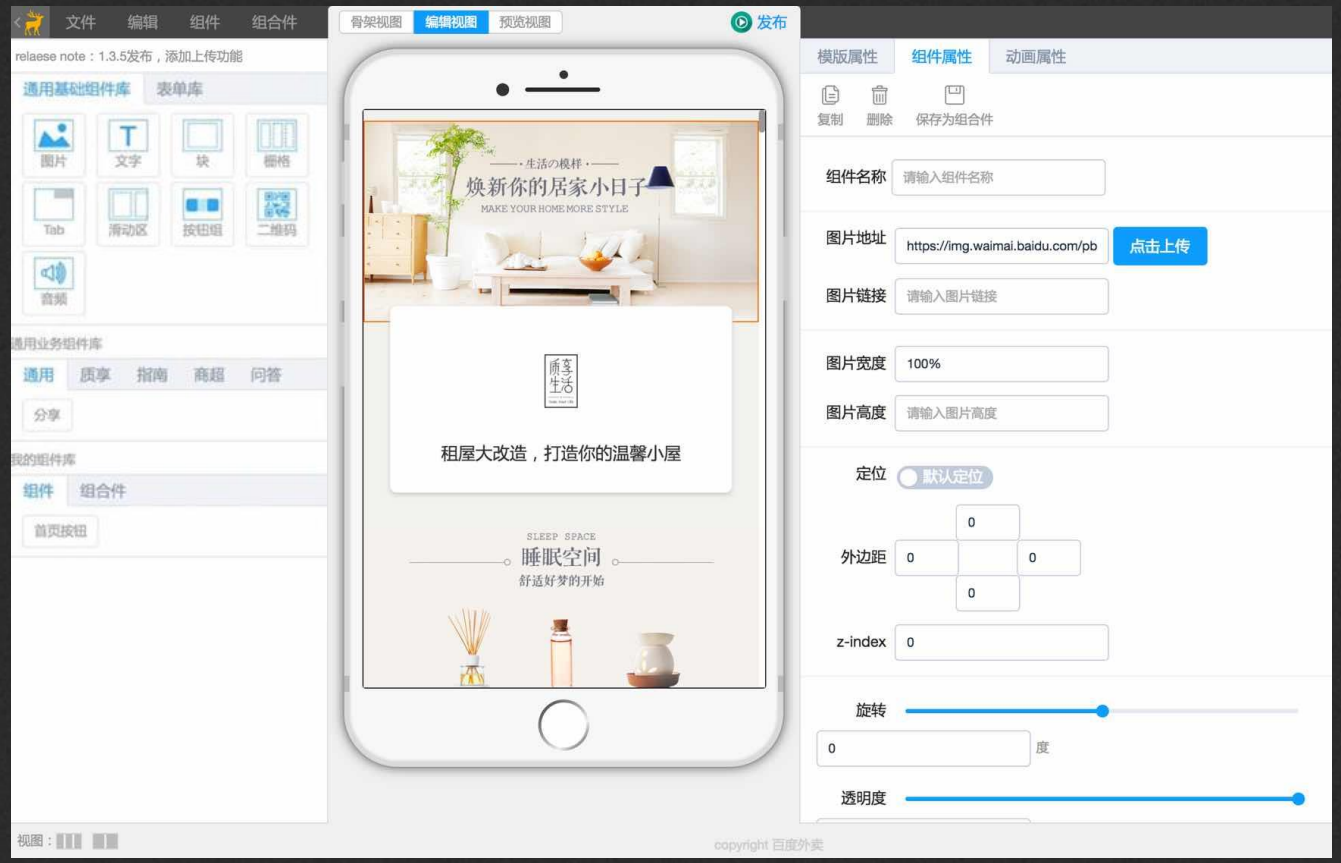
筛选框

操作按钮

表格列表

将大量重复的筛选+列表模型抽象成一个业务模型组件
通过简单配置即可生成一个完整页面

拖拽式H5页面生成平台



▸ 问题背景

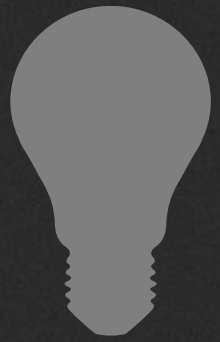
所有的需求都必须经过运营、产品、研发一整套流程，即使是小调整也需要走流程

▸ 平台收益

拖拽操作，即使是运营、产品也能上手制作或改动页面

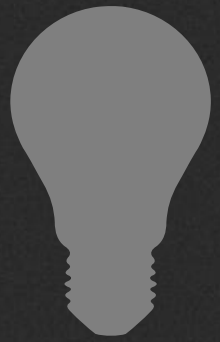
一键发布，即使对研发也比写代码方便

H5页面生成平台技术点



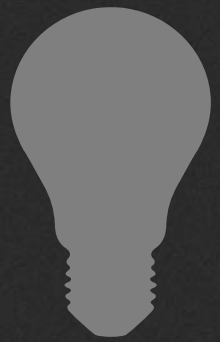
嵌套复用

组件不只是调换顺序，一个组件可以嵌入另一个组件或者出现多次，实现更加丰富高级的复用



组件扩展

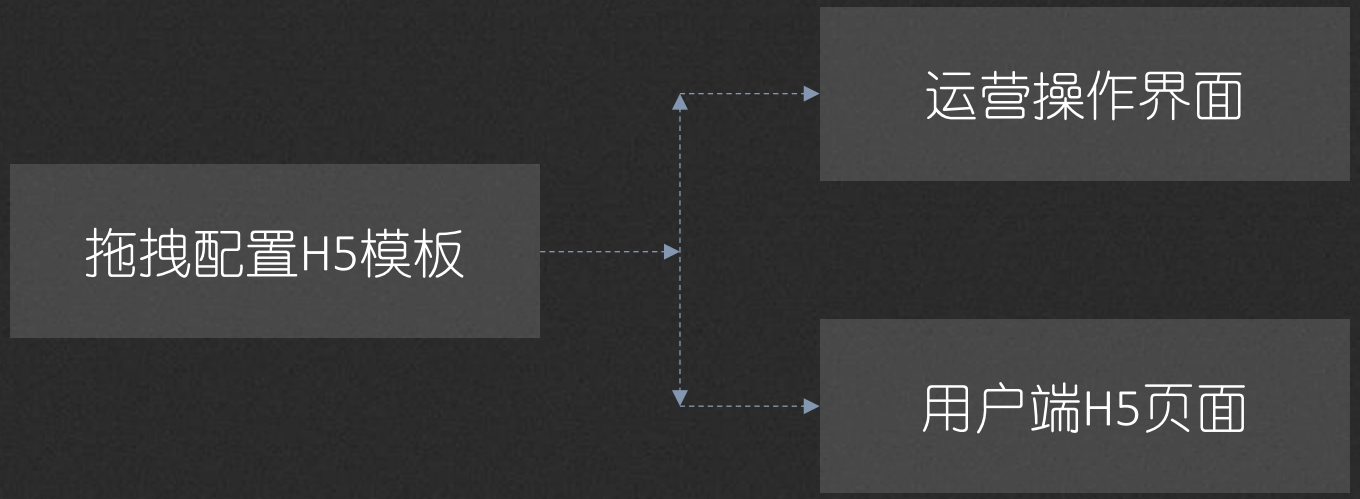
开发者可自主按照规范开发组件并上传到平台，组件的维护与平台本身分离，单个组件稳定性不影响整个平台稳定性



即时编译

配置结果是个JSON，但是发布上线的不是光JSON，然后依赖一个沉重的通用客户端进行渲染，而是直接生成实际运行的代码

后续目标：平台全流程组件化



最终目标是将组件的组合及配套数据接口全部配置化
开发者只专注独立的组件模型，从繁琐的低技术含量开发中解脱出来

感 谢 聆 听

THANKS