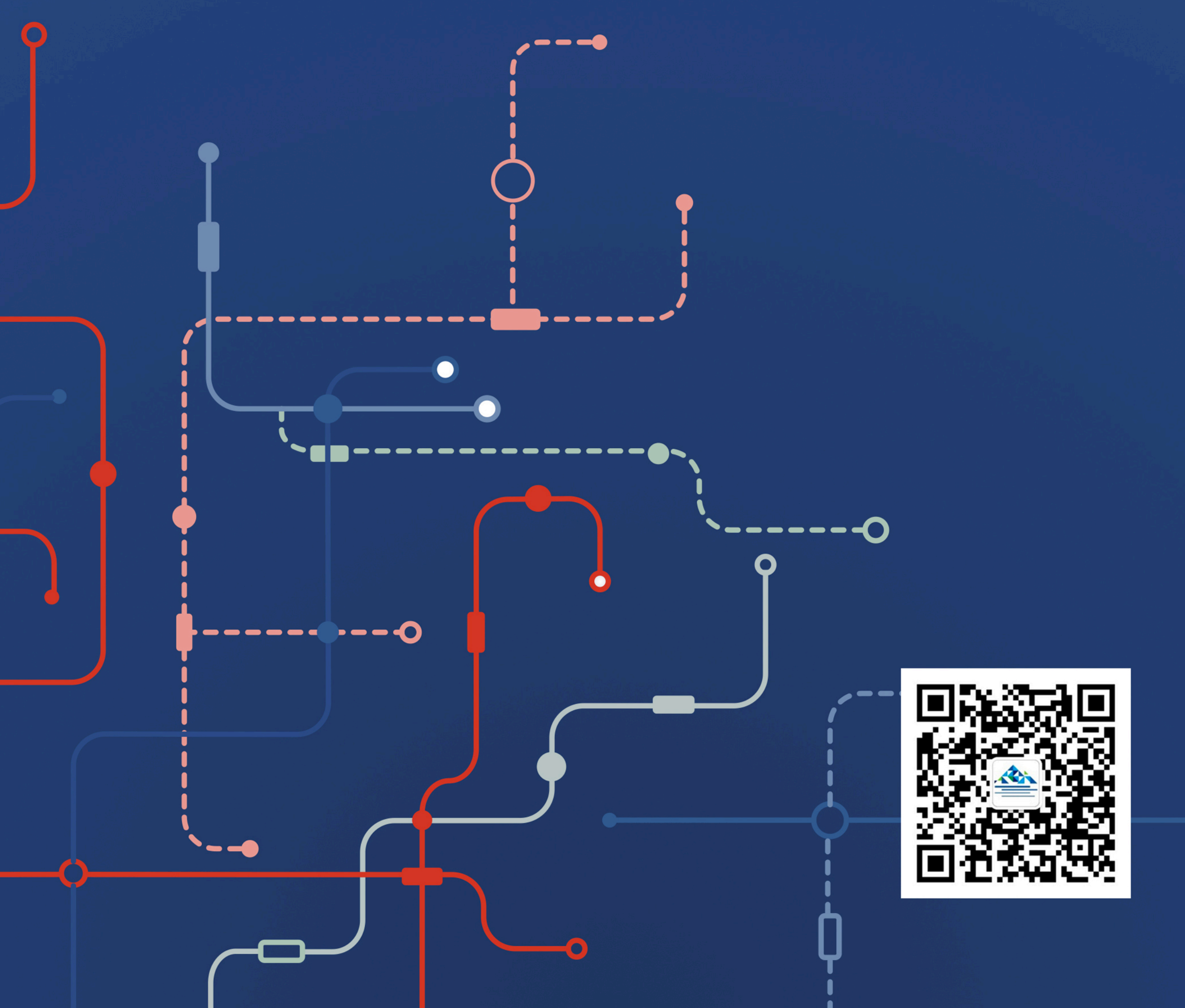
小程序开发实践

Geekbang. 极客邦科技 InfoQ



卷首语：小程序的战场，谁能脱颖而出？

作者：覃云

2018 年，可以说是小程序最火爆的一年，BAT、头条全都进军小程序战场，其中，微信以先发制人的优势继续保持领先的地位，百度最“高调”，从上线到月活过亿用了不到半年的时间，还联合爱奇艺、快手、58 同城等 12 家企业成立了小程序开源联盟，去年年底百度智能小程序也顺利开源了。而阿里也不甘落后，截止至今年 1 月，支付宝小程序日活突破 2.3 亿，支付宝小程序也因此被称为“蚂蚁金服未来三年最重要的战略之一”。

相比之下，头条就相对“低调”多了，除了公布上线的消息和推出一个开发者平台，其他相关技术都没有对外公布，我们曾试图采访他们的专家，但是对方以暂不方便对外透露拒绝了，所以本迷你书只围绕 BAT 小程序的内容进行。

下面来说说，策划本迷你书时的一些思路：

微信小程序已经上线两年有余，不管是在开发还是使用上都相对成熟了，网上的资源也很多了，故我们没有选择将微信小程序的原生开发过程放进迷你书中，但会有第三方框架开发微信小程序的内容。

由于市面上多端小程序的出现，开发者需要针对不同的端去编写多套代码，为降低成本，只编写一套代码就能适配多端的能力显得尤为重要，此时，小程序第三方框架应运而生，市面上此类框架有 wepy、mpvue、Taro 和 Chameleon 等，他们都支持多端小程序，其中，前两者是基于 Vue 或类 Vue 的规范，在市面上已经有很多成熟的应用，而 Taro 是基于 React 标准，于 2018 年 9 月才推出 1.0 版本，Chameleon 是滴滴于 2019 年 1 月最新开源的，它和 Taro 有很多相似之处。

在这本迷你书中，我们选择了 Taro 和 Chameleon 来为大家做详细介绍，为此，InfoQ 特地采访了 Taro 团队京东凹凸实验室，拿到了关于 Taro 深度实践的一手资料，而滴滴方面也给了我们关于 Chameleon 背后的开发理念和方案。

最后，也正如上文所说，2018 年是小程序激战的一年，百度和支付宝小程序接连上线，开发者一时看花了眼，不知该如何选择，虽然有第三方框架支持代码多端运行，但各类小程序最基本的原生开发方式略有不同，开发者还是有必要了解的。因此，InfoQ 分别从百度和支付宝小程序技术专家那里获得了其小程序的技术架构特点和开发过程，希望给那些在做技术选型的开发者带来一些参考。

大前端的下一站

GMTC 2019 全球大前端技术大会

主办方 Geekbang 极客邦科技 InfoQ

大会主题

GMTC TOPICS

前端工程化	性能优化	跨平台专场
前端框架	UI与图形编程	Node专场
未来移动技术	小程序	编程语言
质量保证专场	移动AI	架构演进专场
前端团队管理和个人成长		

会议：6月20-21日 培训：6月22-23日

地址：北京·国际会议中心

7折报名中，立减1440元

咨询热线：18514549229

目录

Chameleon：滴滴开源的跨平台统一 MVVM 框架 1

京东 Taro 框架深度实践 8

支付宝小程序技术架构全解析 22

百度新发布的智能小程序是什么？ 38

苏宁：我们开发百度小程序遇到的那些“坑” 46

Chameleon：滴滴开源的跨平台统一 MVVM 框架

作者 Conan

近日，滴滴在 GitHub 上开源了跨端解决方案 Chameleon，简写 CML，中文名卡梅龙；中文意思变色龙，意味着就像变色龙一样能适应不同环境的跨端整体解决方案，具有易用、开发快、高性能等特点。下文将详细介绍 Chameleon 项目的研发背景和性能特点。

背景

研发同学在端内既追求 h5 的灵活性，也要追求性能趋近于原生。面对入口扩张，主端、独立端、微信小程序、支付宝小程序、百度小程序、Android 厂商联盟快应用，单一功能在各平台都要重复实现，开发和维护成本成倍增加。迫切需要维护一套代码可以构建多入口的解决方案，历经近 20 个月打磨，滴滴跨端解决方案 Chameleon 终于发布，真正专注于让一套代码运行多

端。

设计理念

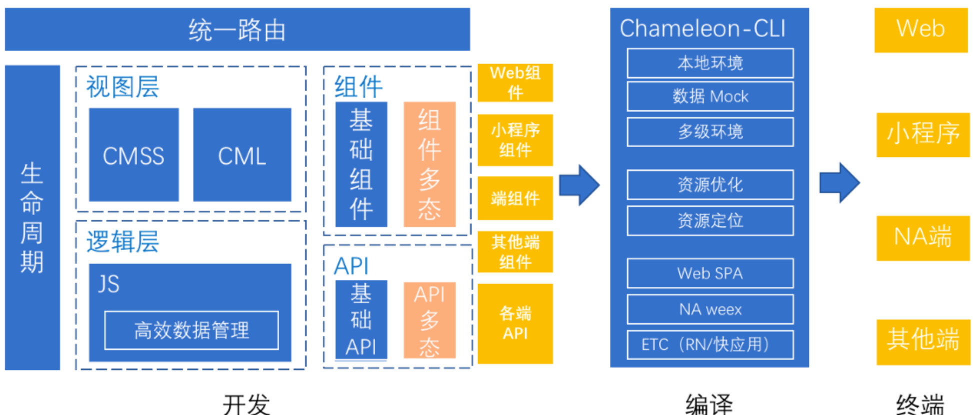
软件架构设计里面最基础的概念“拆分”和“合并”，拆分的意义是“分而治之”，将复杂问题拆分成单一问题解决，比如后端业务系统的”微服务化“设计；“合并”的意义是将同样的业务需求抽象收敛到一块，达成高效率高质量的目的，例如后端业务系统中的“中台服务”设计。

而 Chameleon 属于后者，通过定义统一的语言框架 + 统一多态协议，从多端（对应多个独立服务）业务中抽离出自成体系、连续性强、可维护强的“前端中台服务”。

跨端目标

虽然不同各端环境千变万化，但万变不离其宗的是 MVVM 架构思想，Chameleon 目标是让 MVVM 跨端环境大统一。

学习全景图



开发语言

从事过网页编程的人知道，网页编程采用的是 HTML + CSS + JS 这样的组合，同样道理，chameleon 中采用的是 CML + CMSS + JS。

JS 语法用于处理页面的逻辑层，与普通网页编程相比，本项目目标定义标准 MVVM 框架，拥有完整的生命周期，watch，computed，数据双向绑定等优秀的特性，能够快速提高开发速度、降低维护成本。

CML（Chameleon Markup Language）用于描述页面的结构，我们知道 HTML 是有一套标准的语义化标签，例如文本是 < span> 按钮是 < button>。CML 同样具有一套标准的标签，我们将标签定义为组件，CML 为用户提供了一系列组件。同时 CML 中还支持模板语法，例如条件渲染、列表渲染，数据绑定等等。同时，CML 支持使用类 VUE 语法，让你更快入手。

CMSS(Chameleon Style Sheets) 用于描述 CML 页面结构的样式语言，其具有大部分 CSS 的特性，并且还可以支持各种 css 的预处语言 less stylus。

通过以上对于开发语言的介绍，相信你看到只要是有过网页编程知识的人都可以快速的上手 chameleon 的开发。

丰富的组件

在用 CML 写页面时，chameleon 提供了丰富的组件供开发者使用，内置的有 button switch radio checkbox 等组件，扩展的有 c-picker c-dialog c-loading 等等，覆盖了开发工作中常用的组件。

详情请查看：https://cmljs.org/doc/component/component.html

丰富的 API

为了方便开发者的高效开发，chameleon 提供了丰富的 API 库，发布为 npm 包 chameleon-api，里面包括了网络请求、数据存储、地理位置、系统信息、动画等方法。

详情请查看：https://cmljs.org/doc/api/api.html

自由定制 API 和组件

基于强大的多态协议，可自由扩展任意 API 和组件，不强依赖框架的更新。各端原始项目中已积累大量组件，也能直接引入到跨端项目中使用。

智能规范校验

代码规范校验，当出现不符合规范要求的代码时，编辑器会展示智能提示，不用挨个调试各端代码，同时命令行启动窗口也会提示代码的错误位置。

详情请查看：https://cmljs.org/doc/framework/linter.html

渐进式跨端

很多人已经开发小程序了，又不愿意大多阔斧重新改造，也希望使用 CML？当然可以，2 种方式使用 CML：

说明/类型	整个项目使用CML	公用组件使用CML
关系图	公用业务层 "多态" 协议层 基础库 多态组件扩展 android iOS Web 小程序	普通小程序页面 普通vue页面 普通weex页 CML公用组件层
说明	业务层需求在各端环境高度类似， 原本需要针对不同端重复开发、重复测试， 那么使用Chameleon将整个项目"从上至下"都用一套代码运行。	原本公用组件需要重复开发、重复测试， 使用一套代码开发公用组件， 各个端可以直接使用公用组件
举例	首页、详情页、订单	分享组件、支付组件、地图组件、picker组件

先进前端开发体验

Chameleon 不仅仅是跨端解决方案。基于优秀的前端打包工具 Webpack，吸收了业内多年来积累的最有用的工程化设计，提供了前端基础开发脚手架命令工具，帮助端开发者从开发、联调、测试、上线等全流程高效的完成业务开发。

框架

Chameleon 不仅仅是跨端解决方案，让开发者高效、低成本开发多端原生应用。基于优秀的前端打包工具 Webpack，吸收了业内多年来积累的最有用的工程化设计，提供了前端基础开发脚手架命令工具，帮助端开发者从开发、联调、测试、上线等全流程高效的完成业务开发。

框架提供了自己的视图层描述语言 CML 和 CMSS，以及基于 JavaScript 的逻辑层框架，并在视图层与逻辑层间提供了数据传输和事件系统，让开发者能够专注于数据与逻辑。

脚手架工具

基于 node 开发的脚手架工具，提供简洁的命令，进行初始化与构建项目。

目录结构

提供规范化的项目结构，适合于企业级大型应用的开发，CML 单文件组件的开发模式更有利于提高开发效率与优化文件组织结构。

视图层与逻辑层

视图层由 CML 与 CMSS 编写，逻辑层由 JS 编写，chameleon 的核心是一个标准响应式数据驱动视图更新的 MVVM 框架。

多态协议

提供了跨端时各端底层组件与接口统一的解决方案，使开发者可以自由扩展原生 api 与组件。

规范校验

为了提高开发的效率与代码的可维护性，chameleon 提供了全面的代码规范与校验，帮助开发者能够得到更好的开发体验。

后续规划

方向	子方向	执行项目	进行中	下一个进行
易用性加强	语法检查 编译速度 前端工程化	A 、检查能力加强：潜在错误阻断在编辑时 B、编辑器插件语法检查：Sublime text、Visual Studio Code、Web,Webstorm C、Chameleon playground：Debug工具加强 D、编辑器插件：代码提示 E、图形化界面创建和管理项目	A	B
框架优化	包大小 运行性能 web前端模块服务化	A、包大小：优化各包大小到70%（uglily后当前weex 136k wx 99.3k web 143k） B、多端界面一致性加强：组件创建Web Component化 C、多端并行编译速度优化 D、统一内置能力加强：Canvas、地图、音频等 E、静态资源关系依赖：服务端按依赖自动加载资源包	A	B
端品类扩展	各类小程序 React-Native Flutter	A、支付宝小程序：能力支持（测试中） B、百度小程序：能力支持（测试中） C、快应用：能力支持 D、端扩展协议标准化：用户自由扩展新端	A B	C
组件扩展	c-design 内置组件加强	A、c-design：“开箱即食”的组件库 c-design，任意端用户直接安装可用 B、垂直类组件库：金融、电商类型组件库	A	B
端能力扩展	Native能力 NAtive 内置组件加强	A、Native API：Chameleon Native SDK能力向小程序靠齐	A	
流程优化	XEditor ChameleonShow	A、ChameleonShow：开源Chameleon后台管理平台，解决移动端页面碎片化问题 B、XEditor：让非研发直接发布任意终端的简单页面，无需重复开发	A	
服务扩展	多端服务能力统一	A、统一云服务：统一后端服务接口能力，如分享、支付、消息推送		

有关安装、使用过程以及常见问题解答，请查看以下链接：

FAQ：https://cmljs.org/doc/framework/faq.html

GitHub：https://github.com/didi/chameleon

chameleon 官网：cml.js.org

京东 Taro 框架深度实践

作者 凹凸实验室

前言

Taro 是凹凸实验室遵循 React 语法规范的多端开发方案，Taro 目前已对外开源一段时间，受到了前端开发者的广泛欢迎和关注。截止目前 star 数已经突破 11.2k，还在开启的 Issues 达 200 多个，已经关闭 700 多个，可见使用并参与讨论的开发者是非常多的。Taro 目前已经支持微信小程序、H5、RN、支付宝小程序、百度小程序，持续迭代中的 Taro，也正在兼容更多的端以及增加一些新特性的支持。

回归正题，本篇文章主要讲的是 Taro 深度开发实践，综合我们在实际项目中使用 Taro 的一些经验和总结，首先会谈谈 Taro 为什么选择使用 **React** 语法，然后再从 **Taro** 项目的代码组织、数

据状态管理、性能优化以及多端兼容等几个方面来阐述 Taro 的深度开发实践体验。

为什么选择使用 React 语法？

这个要从两个方面来说，一是小程序原生的开发方式不够友好，或者说不够工程化，在开发一些大型项目时就会显得很吃力，主要体现在以下几点：

- 一个小程序页面或组件，需要同时包含 4 个文件，以至开发一个功能模块时，需要多个文件间来回切换；
- 没有自定义文件预处理，无法直接使用 Sass、Less 以及较新的 ES Next 语法；
- 字符串模板太过孱弱，小程序的字符串模板仿的是 **Vue**，但是没有提供 **Vue** 那么多的语法糖，当实现一些比较复杂的处理时，写起来就非常麻烦，虽然提供了 `wxs` 作为补充，但是使用体验还是非常糟糕；
- 缺乏测试套件，无法编写测试代码来保证项目质量，也就不能进行持续集成，自动化打包。

原生的开发方式不友好，自然就想要有更高效的替代方案。所以我们将目光投向了市面上流行的三大前端框架 **React**、**Vue**、**Angular**。**Angular** 在国内的流行程度不高，我们首先排除了这种语法规范。而类 Vue 的小程序开发框架市面上已经有一些优秀的开源项目，同时我们部门内的技术栈主要是 React，那么 **React** 语法规范也自然成为了我们的第一选择。除此之外，我们还有以下几点的考虑：

- React 一门非常流行的框架，也有广大的受众，使用它也能降低小程序开发的学习成本；
- 小程序的数据驱动模板更新的思想与实现机制，与 React 类似；
- React 采用 JSX 作为自身模板，JSX 相比字符串模板来说更加自由，更自然，更具表现力，不需要依赖字符串模板的各种语法糖，也能完成复杂的处理；
- React 本身有跨端的实现方案 ReactNative，并且非常成熟，社区活跃，对于 Taro 来说有更多的多端开发可能性。

综上所述，Taro 最终采用了 **React** 语法来作为自己的语法标准，配合前端工程化的思想，为小程序开发打造了更加优雅的开发体验。

Taro 项目的代码组织

要进行 Taro 的项目开发，首先自然要安装 taro-cli，具体的安装方法可参照文档，这里不做过多介绍了，默认你已经装好了 taro-cli 并能运行命令。

然后我们用 cli 新建一个项目，得到的项目模板如下：

```
1    ├── dist                     编译结果目录
2    ├── config                   配置目录
3    |   ├── dev.js               开发时配置
4    |   ├── index.js             默认配置
5    |   └── prod.js              打包时配置
6    ├── src                      源码目录
7    |   ├── pages                页面文件目录
8    |   |   ├── index            index 页面目录
9    |   |   |   ├── index.js     index 页面逻辑
10   |   |   |   └── index.css    index 页面样式
11   |   ├── app.css              项目总通用样式
12   |   └── app.js               项目入口文件
13   └── package.json
```

如果是十分简单的项目，用这样的模板便可以满足需求，在 index.js 文件中编写页面所需要的逻辑。

假如项目引入了 redux，例如我们之前开发的项目，目录则是这样的：

```
1    ├── dist                     编译结果目录
2    ├── config                   配置目录
3    |   ├── dev.js               开发时配置
4    |   ├── index.js             默认配置
5    |   └── prod.js              打包时配置
6    ├── src                      源码目录
7    {1} |   ├── actions              redux 里的 actions
8    |   ├── asset                图片等静态资源
9    |   ├── components           组件文件目录
10   |   ├── constants            存放常量的地方，例如 api、一些配置项
11   |   ├── reducers             redux 里的 reducers
12   |   ├── store                redux 里的 store
```

```
13  │   ├── utils               存放工具类函数
14  │   ├── pages               页面文件目录
15  │   │   ├── index           index 页面目录
16  │   │   │   ├── index.js    index 页面逻辑
17  │   │   │   └── index.css   index 页面样式
18  │   ├── app.css             项目总通用样式
19  │   └── app.js              项目入口文件
20  └── package.json
21  {1}
```

我们之前开发的一个电商小程序，整个项目大概 3 万行代码，数十个页面，就是按上述目录的方式组织代码的。比较重要的文件夹主要是 pages、components 和 actions。

- **pages** 里面是各个页面的入口文件，简单的页面就直接一个入口文件可以了，倘若页面比较复杂那么入口文件就会作为组件的聚合文件，redux 的绑定一般也是此页面里进行。
- 组件都放在 **components** 里面。里面的目录是这样的，假如有个 coupon 优惠券页面，在 pages 自然先有个 coupon，作为页面入口，然后它的组件就会存放在 components/coupon 里面，就是 **components** 里面也会按照页面分模块，公共的组件可以建一个 components/public 文件夹，进行复用。

  这样的好处是页面之间互相独立，互不影响。所以我们几个开发人员，也是按照页面的维度来进行分工，互不干扰，大大提高了我们的开发效率。
- **actions** 这个文件夹也是比较重要，这里处理的是拉取数据，数据再处理的逻辑。可以说，数据处理得好，流动清晰，整个项目就成功了一半，具体可以看下面*数据状态管理*的部分。如上，假如是 coupon 页面的 actions，那么就会放在 actions/coupon 里面，可以再一次见到，所有的模块都是以页面的维度来区分的。

除此之外，**asset** 文件用来存放的静态资源，如一些 icon 类的图片，但建议不要存放太多，毕竟程序包有限制。而 **constants** 则是一些存放常量的地方，例如 api 域名，配置等等。

项目搭建完毕后，在根目录下运行命令行 npm run build:weapp 或者 taro build --type weapp --watch 编译成小程序，然后就可以打开小程序开发工具进行预览开发了。编译成其他端的话，只需指定 type 即可（如编译 H5：taro build --type h5 --watch）。

使用 Taro 开发项目时，代码组织好，遵循规范和约定，便成功了一半，至少会让开发变得更有效率。

数据状态管理

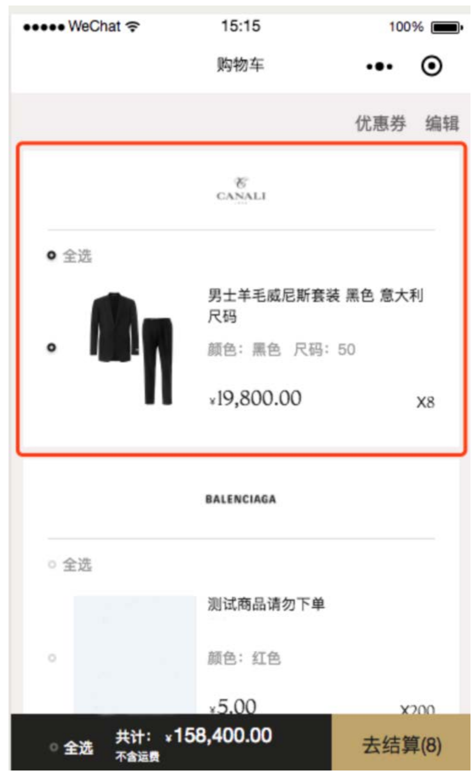
上面说到，会用 redux 进行数据状态管理。

说到 redux，相信大家早已耳熟能详了。在 Taro 中，它的用法和平时在 React 中的用法大同小异，先建立 store、reducers，再编写 actions；然后通过@tarojs/redux，使用 Provider 和 connect，将 store 和 actions 绑定到组件上。基础的用法大家都懂，下面我给大家介绍下如何更好地使用 redux。

数据预处理

相信大家都遇到过这种时候，接口返回的数据和页面显示的数据并不是完全对应的，往往需要再做一层预处理。那么这个业务逻辑应该在哪里管理，是组件内部，还是 redux 的流程里？

举个例子：



例如上图的购物车模块，接口返回的数据是：

```
{
        code: 0,
        data: {
        shopMap: {...}, // 存放购物车里商品的店铺信息的 map
        goods: {...}, // 购物车里的商品信息
        ...
        }
        ...
}
```

对的，购车里的商品店铺和商品是放在两个对象里面的，但视图要求它们要显示在一起。这时候，如果直接将返回的数据存到 store，然后在组件内部 render 的时候东拼西凑，将两者信息匹配，再做显示的话，会显得组件内部的逻辑十分的混乱，不够纯粹。

所以，我个人比较推荐的做法是，在接口返回数据之后，直接将其处理为与页面显示对应的数据，然后再 dispatch 处理后的数据，相当于做了一层拦截，像下面这样：

```
const data = result.data // result 为接口返回的数据
const cartData = handleCartData(data) // handleCartData 为处理数据的函数
dispatch({type: 'RECEIVE_CART', payload: cartData}) // dispatch 处理过后的函数

...
// handleCartData 处理后的数据
{
    commoditys: [{
        shop: {...}, // 商品店铺的信息
        goods: {...}, // 对应商品信息
    }, ...]
}
```

可以见到，处理数据的流程在 render 前被拦截处理了，将对应的商品店铺和商品放在了一个对象了。

这样做有如下几个好处：

- 一个是组件的渲染更纯粹，在组件内部不用再关心如何将数据修改而满足视图要求，只需关心组件本身的逻辑，例如点击事件，用户交互等。
- 二是数据的流动更可控，后台数据 ——> 拦截处理 ——> 期望的数据结构 ——> 组件，假如后台返回的数据有变动，我们要做的只是改变 `handleCartData` 函数里面的逻辑，不用改动组件内部的逻辑。

实际上，不只是后台数据返回的时候，其它数据结构需要变动的时候都可以做一层数据拦截，拦截的时机也可以根据业务逻辑调整，重点是要让组件内部本身不关心数据与视图是否对应，只专注于内部交互的逻辑，这也很符合 `React` 本身的初衷，数据驱动视图。

用 Connect 实现计算属性

计算属性？这不是响应式视图库才会有的么，其实也不是真正的计算属性，只是通过一些处理达到模拟的效果而已。因为很多时候我们使用 redux 就只是根据样板代码复制一下，改改组件各自的 `store`、`actions`。实际上，我们它可以做更多的事情，例如：

```
1  export default connect(({
2    cart,
3  }) => ({
4    couponData: cart.couponData,
5    commoditys: cart.commoditys,
6    editSkuData: cart.editSkuData
7  }), (dispatch) => ({
8    // ...actions 绑定
9  }))(Cart)
10
11 // 组件里
12 render () {
13        const isShowCoupon = this.props.couponData.length !== 0
14     return isShowCoupon && <Coupon />
15 }
16
```

上面是很普通的一种 connect 写法，然后 render 函数根据 couponData 里是否数据来渲染。这时候，我们可以把 this.props.couponData.length !== 0 这个判断丢到 connect 里，达成一种 computed 的效果，如下：

```
1  export default connect(({
2    cart,
3  }) => {
4    const { couponData, commoditys, editSkuData  } = cart
5    const isShowCoupon = couponData.length !== 0
6    return {
7      isShowCoupon,
8      couponData,
9      commoditys,
10     editSkuData
11 }}, (dispatch) => ({
12   // ...actions 绑定
13 }))(Cart)
14
15 // 组件里
16 render () {
17     return this.props.isShowCoupon && <Coupon />
18 }
19
```

可以见到，在 connect 里定义了 isShowCoupon 变量，实现了根据 couponData 来进行 computed 的效果。

实际上，这也是一种数据拦截处理。除了 computed，还可以实现其它的功能，具体就由各位看官自由发挥了。

性能优化

关于数据状态处理，我们提到了两点，主要都是关于 redux 的用法。接下我们聊一下关于性能

优化的。

setState 的使用

其实在小程序的开发中，最大可能的会遇到的性能问题，大多数出现在 setData（具体到 Taro 中就是调用 setState 函数）上。这是由小程序的设计机制所导致的，每调用一次 setData，小程序内部都会将该部分数据在逻辑层（运行环境 JSCore）进行类似序列化的操作，将数据转换成字符串形式传递给视图层（运行环境 WebView），视图层通过反序列化拿到数据后再进行页面渲染，这个过程下来有一定性能开销。

所以关于 setState 的使用，有以下几个原则：

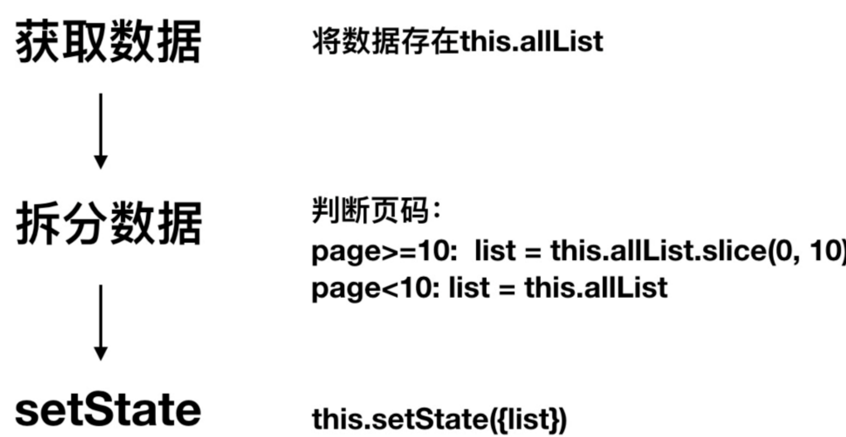
- 避免一次性更新巨大的数据。这个更多的是组件设计的问题，在平衡好开发效率的情况下尽可能地细分组件。
- 避免频繁地调用 setState。实际上在 Taro 中 setState 是异步的，并且在编译过程中会帮你做了这层优化，例如一个函数里调用了两次 setState，最后 Taro 会在下一个事件循环中将两者合并，并剔除重复数据。
- 避免后台态页面进行 setState。这个更有可能是因为在定时器等异步操作中使用了 setState，导致后台态页面进行了 setState 操作。要解决问题该就在页面销毁或是隐藏时进行销毁定时器操作即可。

列表渲染优化

在我们开发的一个商品列表页面中，是需要有无限下拉的功能。

因此会存在一个问题，当加载的商品数据越来越多时，就会报错，invokeWebviewMethod 数据传输长度为 1227297 已经超过最大长度 1048576。原因就是我们上面所说的，小程序在 setData 的时候会将该部分数据在逻辑层与视图层之间传递，当数据量过大时就会超出限制。

为了解决这个问题，我们采用了一个大分页思想的方法。就是在下拉列表中记录当前分页，达到 10 页的时候，就以 10 页为分割点，将当前 this.state 里的 list 取分割点后面的数据，判断滚动向前滚动就将前面数据 setState 进去，流程图如下：



可以见到，我们先把商品所有的原始数据放在 this.allList 中，然后判断根据页面的滚动高度，在页面滚动事件中判断当前的页码。页码小于 10，取 this.allList.slice 的前十项，大于等于 10，则取后十项，最后再调用 this.setState 进行列表渲染。这里的核心思想就是，把看得见的数据才渲染出来，从而避免数据量过大而导致的报错。

同时为了提前渲染，我们会预设一个 500 的阈值，使整个渲染切换的流程更加顺畅。

多端兼容

尽管 Taro 编译可以适配多端，但有些情况或者有些 API 在不同端的表现差异是十分巨大的，这时候 Taro 没办法帮我们适配，需要我们手动适配。

process.env.TARO_ENV

使用 process.env.TARO_ENV 可以帮助我们判断当前的编译环境，从而做一些特殊处理，目前它的取值有 weapp、swan、alipay、h5、rn 五个。可以通过这个变量来书写对应一些不同环境下的代码，在编译时会将不属于当前编译类型的代码去掉，只保留当前编译类型下的代码，从而达到兼容的目的。例如想在微信小程序和 H5 端分别引用不同资源：

```
1  if (process.env.TARO_ENV === 'weapp') {
2    require('path/to/weapp/name')
```

```
3 } else if (process.env.TARO_ENV === 'h5') {
4   require('path/to/h5/name')
5 }
```

我们知道了这个变量的用法后，就可以进行一些多端兼容了，下面举两个例子来详细阐述。

滚动事件兼容

在小程序中，监听页面滚动需要在页面中的 onPageScroll 事件里进行，而在 H5 中则是需要手动调用 window.addEventListener 来进行事件绑定，所以具体的兼容我们可以这样处理：

```
1  class Demo extends Component {
2    constructor() {
3      super(...arguments)
4      this.state = {
5      }
6      this.pageScrollFn = throttle(this.scrollFn, 200, this)
7    }
8
9    scrollFn = (scrollTop) => {
10     // do something
11   }
12
13   // 在 H5 或者其它端中，这个函数会被忽略
14   onPageScroll (e) {
15     this.pageScrollFn(e.scrollTop)
16   }
17
18   componentDidMount () {
19     // 只有编译为 h5 时下面代码才会被编译
20         if (process.env.TARO_ENV === 'h5') {
21       window.addEventListener('scroll', this.pageScrollFn)
```

```
22        }
23    }
24  }
```

可以见到，我们先定义了页面滚动时所需执行的函数，同时外面做了一层节流的处理

不了解函数节流的可以看这里。

然后，在 onPageScroll 函数中，我们将该函数执行。同时的，在 componentDidMount 中，进行环境判断，如果是 h5 环境就将其绑定到 window 的滚动事件上。

通过这样的处理，在小程序中，页面滚动时就会执行 onPageScroll 函数（在其它端该函数会被忽略）；在 h5 端，则直接将滚动事件绑定到 window 上。因此我们就达成小程序，h5 端的滚动事件的绑定兼容（其它端的处理也是类似的）。

canvas 兼容

假如要同时在小程序和 H5 中使用 canvas，同样是需要进行一些兼容处理。canvas 在小程序和 H5 中的 API 基本都是一致的，但有几点不同：

- canvas 上下文的获取方式不同，h5 中是直接从 dom 中获取；而小程序里要通过调用 Taro.createCanvasContext 来手动创建；
- 绘制时，小程序里还需在手动调用 CanvasContext.draw 来进行绘制。

所以做兼容处理时就围绕这两个点来进行兼容：

```
1   componentDidMount () {
2       // 只有编译为 h5 下面代码才会被编译
3       if (process.env.TARO_ENV === 'h5') {
4           this.context = document.getElementById('canvas-id').getContext('2d')
5       // 只有编译为小程序下面代码才会被编译
6       } else if (process.env.TARO_ENV === 'weapp') {
7           this.context = Taro.createCanvasContext('canvas-id', this.$scope)
8           }
9   }
10
11  // 绘制的函数
```

```
12 draw () {
13     // 进行一些绘制操作
14         // .....
15
16     // 兼容小程序端的绘制
17     typeof this.context.draw === 'function' && this.context.draw(true)
18 }
19
20 render () {
21     // 同时标记上 id 和 canvas-id
22         return <Canvas id='canvas-id' canvas-id='canvas-id'/>
23 }
24
```

可以见到，先是在 componentDidMount 生命周期中，分别针对不同的端的方法而取得 CanvasContext 上下文，在小程序端是直接通过 `Taro.createCanvasContext` 进行创建，同时需要在第二个参数传入 `this.$scope`；在 H5 端则是通过 `document.getElementById(id).getContext('2d')`来获得 CanvasContext 上下文。

获得上下文后，绘制的过程是一致的，因为两端的 API 基本一样，而只需在绘制到最后时判读上下文是否有 draw 函数，有的话就执行一遍来兼容小程序端，将其绘制出来。

我们内部用 Canvas 写了一个弹幕挂件，正是用这种方法来进行两端的兼容。

上述两个具体例子总结起来，就是先根据 Taro 内置的 `process.env.TARO_ENV` 环境变量来判断当前环境，然后再对某些端进行单独适配。因此具体的代码层级的兼容方式会多种多样，完全取决于你的需求，希望上面的例子能对你有所启发。

总结

本文先谈了 Taro 为什么选择使用 **React** 语法，然后再从 **Taro** 项目的代码组织、数据状态管理、性能优化以及多端兼容这几个方面来阐述了 Taro 的深度开发实践体验。整体而言，都是一些较为深入的，偏实践类的内容，如有什么观点或异议，欢迎加入开发交流群，一起参与讨论。

支付宝小程序技术架构全解析

作者 白招拒

在轻应用混战的当下，小程序已经成为巨头们角逐的焦点，阿里自然也不甘落后。据阿里官方的数据，截止到今年 1 月 28 日为止，支付宝小程序应用数已经达到 12 万，总用户数突破 5 亿，日活跃用户数突破 2.3 亿，用户通过支付宝首页下拉入口进入小程序的日人均打开次数为 4 次，支付宝小程序也因此被称为“蚂蚁金服未来三年最重要的战略之一”。

然而，支付宝公开的信息更多面向的是普通用户，开发者能获知的信息少之又少，为此，InfoQ 采访了支付宝小程序首席架构师白招拒，为大家解读支付宝小程序的技术架构和开发特点，以下是采访的全部内容。

支付宝小程序从 2016 年开始立项算起，到现在也快 3 年的时间，在这 3 年的过程中，小程序的技术架构也是不断的升级和演进，在满足业务发展的同时对于小程序整体的高可用、性能优

化、多端输出方面做了大量的工作。今天给大家分享下我们在支付宝小程序技术这块所做的一些工作。

小程序技术架构主要分成四个方面来讲：

1. 系统架构，主要给大家说下小程序的架构，以及其中的一些关键技术；
2. 性能体验，讲下我们在性能体验这块做的几个 case；
3. 开发者工具，怎么更好的帮助开发者开发和管理小程序，和保障线上小程序的质量；
4. 多端 inside，将支付宝小程序的技术输出给集团和外部的商户，让他们具备运行小程序的能力。

系统架构

支付宝小程序不是从零开始建设的一个产品，而是依托于蚂蚁技术部多年来的技术沉淀，再结合小程序的业务场景，逐步的发展起来的。

以上是支付宝小程序架构的示意图，最上面是支付宝钱包提供的主要的七个场景入口，开发者可以根据自己的业务场景运营这些场景入口，把这些入口的流量充分利用起来。中间框内的是小程序的核心引擎，上面是对开发者提供的基础组件和基础 API 能力，开发者根据这些组件和 API 来开发自己的小程序，满足用户的需求。

小程序前端框架这块借鉴了主流前端框架 React 的设计思路，从小程序的应用形态，提供了简

洁的编程模型，定义了一套组件和 API 接口的规范，降低了学习门槛，方便开发者快速开发小程序。在小程序框架内部提供了小程序的生命周期管理，通过事件的方式把小程序每个阶段都注入到小程序里面，开发者可以通过这些事件来处理小程序每个阶段需要完成的业务逻辑。同时框架内部使用了虚拟 DOM 来处理页面的每次更新，提升了页面的渲染性能。

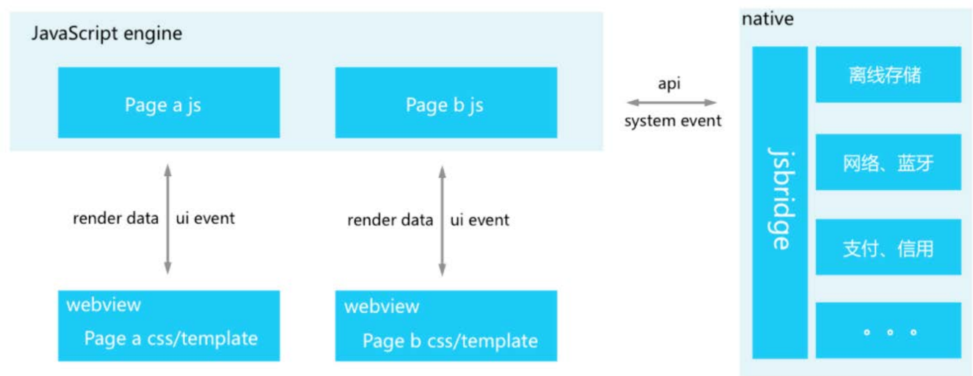
前端框架下面是小程序 native 引擎，包括了小程序容器、渲染引擎和 JavaScript 引擎，这块主要是把客户端 native 的能力和前端框架结合起来，给开发者提供系统底层能力的接口。在渲染引擎上面，支付宝小程序不仅提供 JavaScript+Webview 的方式，还提供 JavaScript+Native 的方式，在对性能要求较高的场景，可以选择 Native 的渲染模式，给用户更好的体验。

示意图左边和右边分别是面对开发者提供的研发支撑和运维支撑服务，可以帮助开发者更有效率的开发小程序，在上线后也提供众多的工具帮助开发者管理和运营线上的小程序。

运行时架构

小程序编程模型是分为多个页面，每个页面有自己的 template、CSS 和 JS，实际在运行的时候，业务逻辑的 JS 代码是运行在独立的 JavaScript 引擎中，每个页面的 template 和 CSS 是运行在各自独立的 webview 里面，页面之间是通过函数 navigateTo 进行页面的切换。

每个 webview 里面的页面和公共的 JavaScript 引擎里面的逻辑的交互方式是通过消息服务，页面的一些事件都会通过这个消息通道传给 JavaScript 引擎运行环境，这个运行环境会响应这个事件，做一些 API 调用，可调到客户端支付宝小程序提供的一些能力，处理之后会把这个数据再重新发送给对应的页面渲染容器来处理，把数据和模板结合在一起来，在产生最终的用户界面。

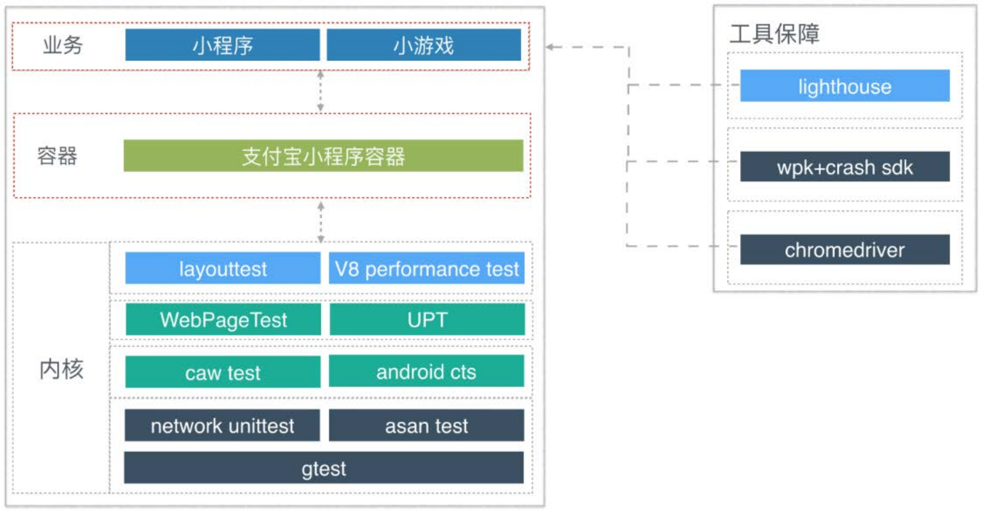


浏览器内核

小程序在 web 上的渲染引擎是浏览器内核，作为小程序的核心组件，经过多方面的考虑，我们采用的是 UC 提供的浏览器内核，UC 的同学在浏览器内核的性能、稳定性和兼容性上做了大量的工作，比系统提供的 webview 提升了不少。

- 稳定性：crash 率只有系统 webview 的三分之一到五分之一；
- 兼容性：不存在各种系统 webview 上的兼容性问题；
- 性能：针对内核启动逻辑，v8 引擎 codecache 深度优化，使得 js 代码解析和编译的时间减少 40%左右；
- 工具：提供了丰富的工具保障 UC 内核的稳定性和性能；

下图是 UC 内核的稳定性保障体系：



同时 UC 内核针对内存做了大量的优化，主要分为几方面：

1. 图片内存：针对低端机，做了更严格的图片缓存限制，在保持性能体验的情况下，进一步限制图片缓存的使用；多个 webview 共用图片缓存池；全面支持 webp、apng 这种更节省内存和 size 的图片格式。
2. 渲染内存：Webview 在不可见的状态下，原生的内存管理没有特殊处理，UC 内核会将不可见 webview 的渲染内存释放；渲染内存的合理设置与调优，避免滚动性能的下降和占用过

多内存。

3. **JS** 内存：更合理地处理 v8 内存 gc，在启动时延时执行 fullgc，避免影响启动的耗时。
4. 峰值内存管理：系统在内存紧张时，会通知内核，UC 内核能够在系统低内存时释放非关键内存占用的模块，避免出现 oom，也避免过度释放带来的渲染黑块；在部分 oom 的情况，规避原生内核主动崩溃的逻辑，在内存极低的情况，部分功能不可用，而不是崩溃。

性能体验

Google 的统计表明，页面打开时间超过 3 秒用户会流失 13%，超过 6 秒用户会流失 60%。反过来，打开时间每减少 1 秒可提升 27%的转化率，给用户带来更好的用户体验一直是支付宝努力在做的事情。

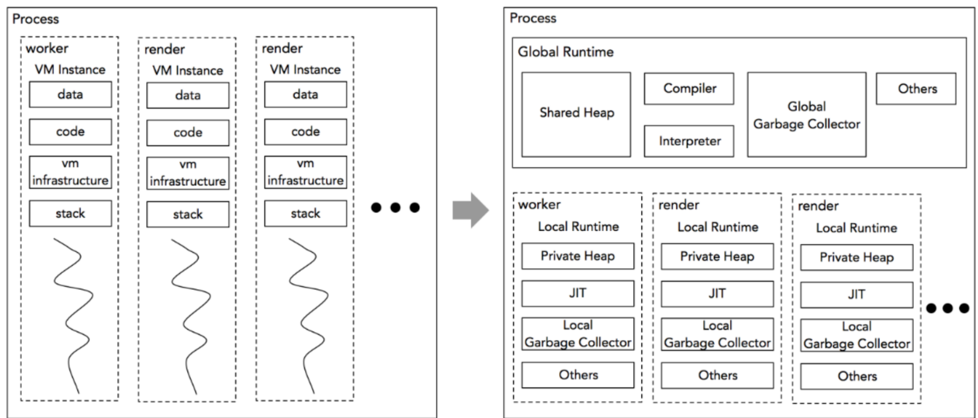
支付宝 app 不同于社交类的 app，属于低频类的应用，所以在小程序的优化方式上会不同于高频的应用，由于高频的应用长期在系统层面是活跃的状态，所以高效的优化方式就是预加载，在后台把小程序相关的资源尽可能的提前加载好，在用户使用小程序时可以快速的启动起来。

而对于低频应用，更多的是冷启动，所以在这种情况下，我们更多的是从技术的角度来优化每一个环节的性能，在小程序用户体验上可以达到高频应用，下面我会分享几个我们性能优化方面的工作。

render 和 worker 交互优化

为了优化小程序的交互体验，目前传统的做法是把 render 层和 woker 层在两个不同的线程里面执行，可以让页面在渲染的时候不会因为业务逻辑的执行而产生卡顿，提升了渲染的速度。

通常的做法是在 webview 里面运行 render 的代码，然后另起一个线程运行 serviceworker，当 serviceworker 需要更新 dom 的时候把事件和数据通过 messagechannel 发送给 render 线程来执行，当业务需要传递到 render 层数据量较大，对象较复杂时，交互的性能就会比较差，因此针对这种情况我们提出一个优化的解决方案。



该方案将原始的 JS 虚拟机实例(即 Isolate)重新设计成了两个部分：Global Runtime 和 Local Runtime。

- Global Runtime 部分是存放共享的装置和数据，全局一个实例。
- Local Runtime 是存放实例自身相关的模块和私有数据，这些不会被共享。

在小程序里面需要做的事情包含两个部分：

1. 轻量级的 js 线程替换 serviceworker 来执行小程序业务逻辑的代码；
2. 更高效的 worker 层和 render 层交互方式。

对于这两个目标我们重新设计了现有的 JS 虚拟机 V8，提出了一种优化的隔离模型（Optimized isolation model,OIM）。OIM 的主要思路是共享 JS 虚拟机实例中与线程执行环境无关的数据和基础设施，以及不可变或不易变的 JS 对象，使得在保持 JS 层逻辑隔离的前提下，节省多实例场景下在内存和功耗上的开销。尽管有些实例间共享的数据会带来同步的开销，但是在隔离模型下，本方案所共享的数据、对象、代码和虚拟机基础设施都是不可变或者不易变的，所以很少发生竞争。

在新的隔离模型下，webview 里面的 v8 实例就是一个 Local Runtime，worker 线程里面的 v8 实例也是一个 Local Runtime，在 worker 层和 render 层交互时，setData 对象的会直接创建在 Shared Heap 里面，因此 render 层的 Local Runtime 可以直接读到该对象，并且用于 render 层的渲染，减少了对象的序列化和网络传输，极大的提升了启动性能和渲染性能。

首页离线缓存优化

首页的加载和渲染对于冷启动是非常关键的，为了减少用户在首页显示前的等待时间，我们采用离线缓存的方式来优化加载的流程。对于正常的加载逻辑，用户在点击小程序图标后就开始启动的过程，下载并解压小程序离线包，找到入口的页面 index.html，作为参数传给浏览器内核开始加载小程序页面。

在浏览器开始加载小程序页面时会先出现三个圆点的 Loading 页，然后在开始加载小程序的前端框架，在前端框架加载过程中会启动异步的 worker 线程加载业务的 js 逻辑代码，前端框架则继续加载小程序的页面，并渲染出首页展现给用户。

为了尽快的把首页展现给用户，在用户首次展现首页后我们会把首页的 UI 页面保存下来，在用户下次重新打开小程序的时候，会首先渲染上次保存下面的首页 UI 页面，把首页展现给用户，然后在后台继续加载前端框架和业务的代码，加载完成后再和离线缓存的首页 UI 进行合并，给用户展现动态的首页。

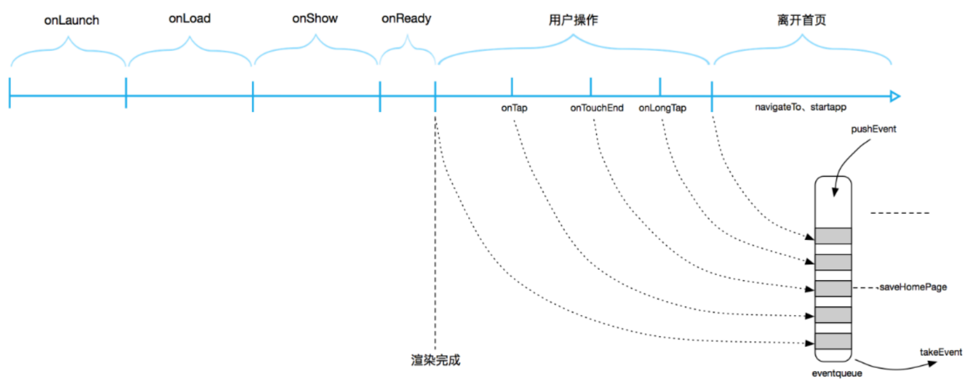
由于在渲染完离线缓存的首页 UI 到真正的业务代码加载完成，这个之间的时间大概在 1 秒左右，所以在用户看到首页并做出反应时动态的首页已经合并完成，并可以对用户的操作做出响应。

在实现首页离线缓存这个特性中，我们面临两个技术上的挑战：

1. 首页离线缓存页面保存的时机

由于小程序启动是受到生命周期的控制，从 onLaunch -> onLoad -> onShow -> onReady -> 用户操作 -> 离开首页这个流程，在这个过程中的任意一个环节都有可能被客观或者主观的原因打断，也就有可能导致保存的离线页面不准确，在启动的时候给用户呈现错误的页面。

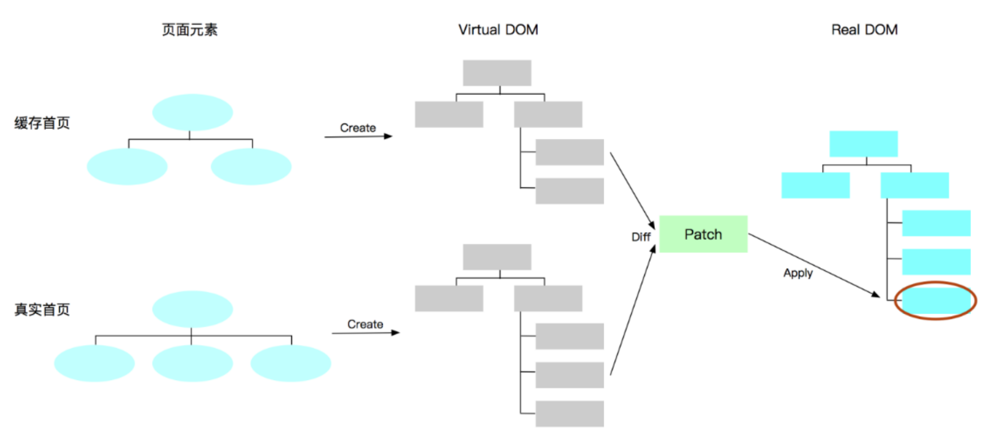
所以对于首页离线缓存渲染的效果，保存页面的时机很重要，我们提供让开发者可以配置的时机，配置的时机有两个：渲染完成和离开首页前。对于渲染完成就是首页渲染完成，用户还未执行任何的操作前把页面保存下来作为离线缓存的页面。离开首页前就是指用户在首页执行了一系列的操作后，跳转到其他页面前用户看到的页面保存下来作为离线缓存的页面。



针对离开首页前保存页面的问题，我们设计了一个事件的队列，小程序生命周期中可能对首页改动的事件都会被捕捉，同时放入到一个队列里面，异步线程会定时的从队列里面拿事件，然后延迟执行保存首页的操作，由于经常对浏览器内核执行保存操作，对性能是有影响的，所以会对这些事件进行合并处理，最终会以最后一个正确保存的首页为准。

2. 离线缓存首页和动态渲染首页替换时的闪屏

对于闪屏问题发生的场景是因为缓存页面和真实渲染的页面是分离的，是两个独立的页面，缓存页面是静态的页面，真实的页面是通过 js 动态创建的页面，所以常规的做法就是当真实页面创建完成后替换缓存的页面，这样的情况下就会发生闪屏。



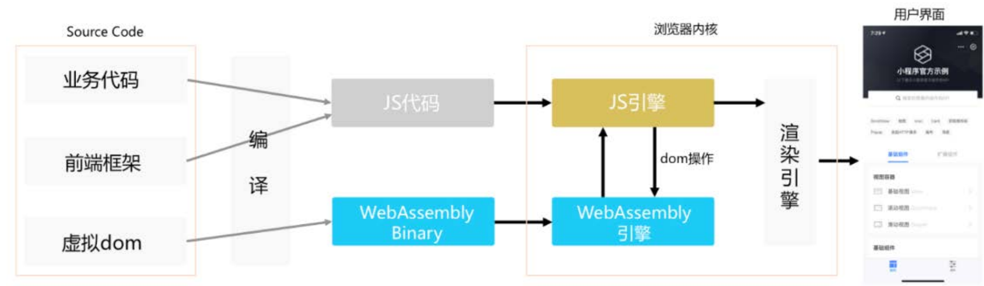
针对这个问题，我们是采用虚拟 dom 来解决，在加载缓存页面的时候把缓存页面放入初始的虚拟 dom 里面，真实页面创建后产生的虚拟 dom 跟缓存页面的虚拟 dom 进行 dom diff，把变化的内容通过 patch 传给浏览器内核，渲染对应的页面，这样就可以只更新局部有变化的页面内容，避免了整个页面的更新，也保证内容的准确性和实时性。

通过实测数据显示，这个优化可以将小程序的冷启动实现秒开。

虚拟 dom 优化

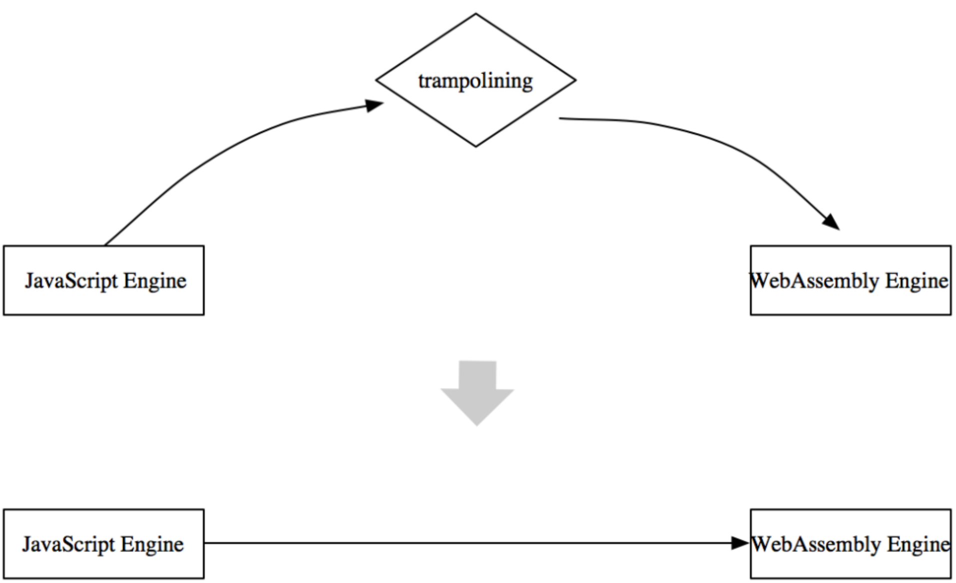
小程序的页面渲染采用的也是业界普遍在使用的虚拟 dom 技术，该技术可以保障在更新页面时只更新变动的部分，提升了更新的效率。不足的地方就是虚拟 dom 也是用 js 来实现，在运算时会大量消耗 cpu，执行的效率不高。

JavaScript 是一种弱动态类型的语言，不同于静态类型的 C 和 Java 语言，相较而言 JS 的运行性能会差一些，因为类型的不确定性限制了 JIT 优化编译器生成代码的质量。



针对这种情况，我们选择 WebAssembly 作为虚拟 dom 的实现方向，WebAssembly 是一个新的 Web 标准，它定义了网页中的可执行代码的二进制格式和相应的类似汇编语言格式。他的目标是使执行代码几乎与本地机器代码一样快，它被用来作为 JavaScript 的补充，以加速 Web 应用程序的性能关键部分,所以我们使用 WebAssembly 技术重新实现了虚拟 dom 这块的核心代码，提升了小程序的页面渲染。

在做这个优化的时候，我们面临 js 代码桥接到 WebAssembly 的性能较差的挑战，因为 js 引擎和 WebAssembly 是两个独立的引擎，他们之间的交互比 js 到 js 的性能要差了不少，针对这个问题，我们参考了业界的一些实现，对 V8 的代码进行了优化，解决 js < -> WebAssembly 交互性能差的问题。



在做这个优化前，我们需要先了解下到底是什么原因导致了 js 和 WebAssembly 交互性能差。由于 JS 和 WebAssembly 是两种不同类型的语言，所以引擎执行过程中遇到语言切换的时候，需要做一些“翻译”工作。而这些翻译工作需要考虑各种情况，需要跳转到一个专门的 trampoline stub 处理。

由于在小程序前端框架的实现代码是 TypeScript 来开发的，所以框架在调用虚拟 dom 的 WebAssembly 的函数时是可以传入具体的参数类型，并且参数的顺序也是固定的，但是这些参数类型和参数顺序在到 js 引擎的时候就丢失了，所以需要做一些额外的“翻译”工作，降低了

交互的性能。

我们的思路就是精简这些翻译的工作，在开发层面把框架和 WebAssembly 的交互代码的参数类型和顺序都固定下来，不让其变动。同时我们让 js 引擎支持了参数类型和参数顺序的传入，在编译期把代码的参数类型和参数顺序保存下来，运行期把 js 代码和类型文件一起传给 js 引擎，让 js 引擎可以直接识别该函数的参数类型，这样就可以直接进行参数转化的工作然后调用 WebAssembly 的方法，避免跳转到一个通用的参数转换的 trampoline stub 上。

通过实测数据表明，相比于以前的实现，新的实现代码执行效率有 50%的提升。

开发者工具

支付宝小程序的目标就是为用户提供高品质的服务，这些服务是靠我们的开发者来实现的，所以如何帮助开发者提供提供高品质的小程序，如何保障线上小程序的质量，就是我们一直努力在做的事情。支付宝小程序提供从开发、调试、发布到运维整个链路的工具，这些工具也在不断的完善和增强，让开发者可以更高效的开发出高品质的小程序。

开发者工具 IDE 支持 mac 和 windows 两个平台的运行，通过打通接入研发平台、数据监控、日志收集等系统，进一步为桌面客户端的稳定性提供保障。提供多端开发能力，通过整合通用能力，适配各端差异，帮助开发者实现代码的多端调试运行，同时可以一键发布到多端。

对于开发新手来说搭建一套完整的后端应用过于复杂，涉及到服务器的购买，域名购买，环境配置等等一系列问题，每一个问题都可能阻碍开发者进行下一步操作。为此我们提供了以下两套一站式云服务方案让开发者能够快速高效搭建一套完整的后端服务：

云函数，将服务器购买，配置，发布，运维等完全解决，让开发者只用关心自己的代码逻辑部分的编写，并且开发语言是 js，对于前端开发者非常友好。相比云应用，更适合编写轻量级的

小程序，但是每个云函数只能在绑定的小程序中调用。

- 云应用，将服务器购买，配置，发布的问题解决，相比云函数，云应用更加灵活，适合编写较复杂的后端应用，并且一个云应用可以支撑多个小程序同时调用。我们提供了两种后端语言 nodeJs 和 java，用户可以自行选择。
- 小程序云测试服务，可以帮助开发者更全面的检测小程序缺陷，评估产品质量，提高审核通过率。我们提供了一套完整的小程序云真机自动化检测方案，在 IDE 申请云测试服务，执行完成后自动生成测试报告。

云测服务提供“快速检测”、“深度检测”两种检测模式，满足多纬度测试需求，并且提供性能检测及优化建议，开发者可根据优化建议优化小程序代码，提供更好的用户体验。

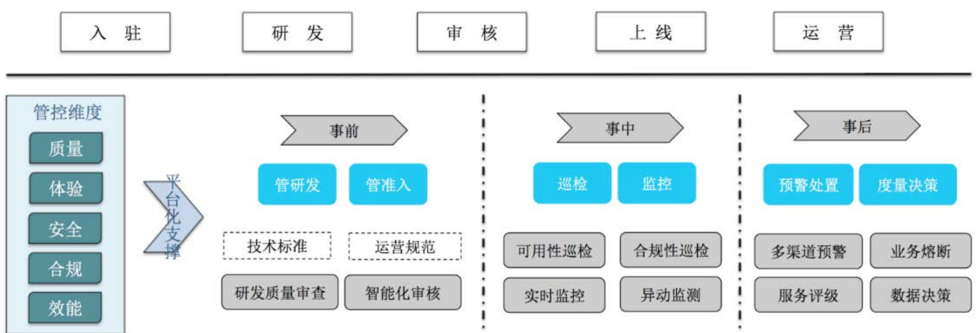
线上巡检

目前支付宝小程序拥有几十万生态合作伙伴，随着小程序生态的不断壮大，合作伙伴的数量也在急剧增加，如何对生态伙伴提供的服务形成有效的管控，如何对小程序的质量进行保障，这是我们面临的新挑战。面对这个问题，我们在制定相应技术标准和运营规范的同时，对小程序从入驻到运营，从质量、体验、安全、合规、效能等维度建设了平台化的质量与风险管控能力。

巡检是开发者生态质量与风险保障重要的一环，是识别问题的重要手段。小程序为开发者提供的服务场景非常丰富而且复杂，为解决这一系列问题，我们通过自建识别引擎，并整合蚂蚁、阿里云等多项基础检测单元的服务能力，以“技术 + 一体化 + 平台化”的方式，建设主动巡检（稽查）的能力，即巡检平台。

在平台建设过程中，我们面临的挑战有：

1. 开发者提供的服务场景非常丰富且复杂，如：缴费、医疗、保险、旅行等服务，产品呈现多样化；
2. 小程序提供的是一套前端框架，服务内容是由服务端动态呈现，随时变化，甚至并且千人千面；
3. 小程序技术的灵活性因素，比如允许内嵌 webview，Js 动态加载等；
4. 小程序体量庞大，百万应用，数千万 page 并且不断增长。



巡检平台具有以下特点：

- 功能全面：可用性、内容合规、信息泄露、图片识别、资源流耗问题的稽查；
- 主动检测：主动访问，非被动监控；先于用户发现，尽可能提前将问题暴露；
- 动态渲染：支动态加载和页面渲染；
- 高频巡检：分钟级高频巡检，快速发现问题；
- 多重保障机制：双引擎检测、智能复查、智能恢复；
- 多渠道灵活的预警决策：多渠道、多阶梯预警，工单决策、故障熔断、事后处罚等完备的业务闭环能力；
- 实时数据大屏：巡检、故障、预警决策实时监控；
- 多维数据度量：多视角、多维护数据大盘；
- 智能高效：预警决策环节，加入大数据 + 算法运用，更智能和高效。

小程序巡检平台从上线以来，实现智能化提效 94%，将小程序审核平均时长从 70.59 小时下降到 4.27 小时并实现 0 积压。根据业务诉求进行不同频率的巡检，目前已累计发现和处理了上万个有问题的小程序，提升了小程序线上服务的品质。

多端 inside

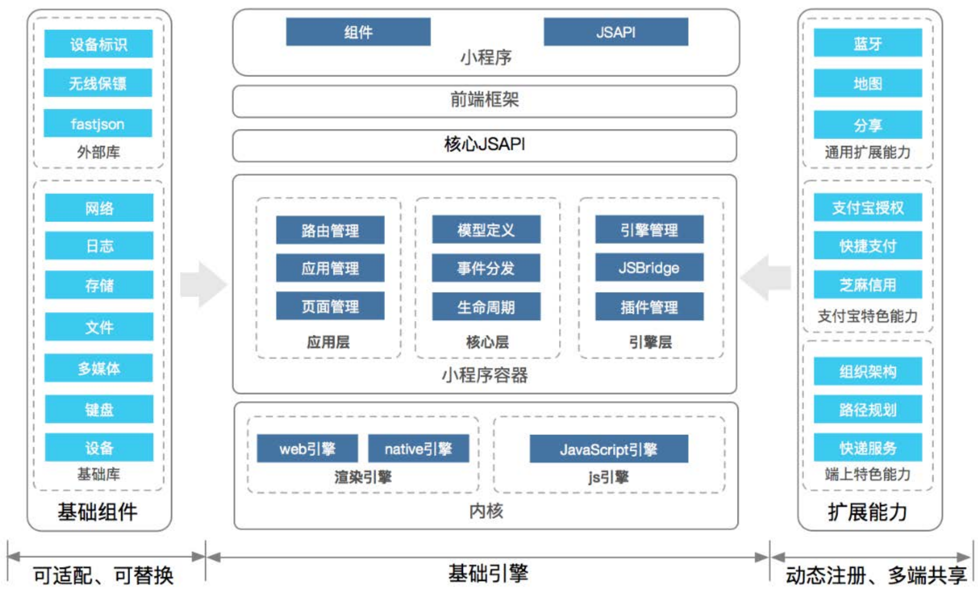
在支付宝小程序发展的过程中，集团内的 BU 也有很强的诉求需要在他们的 app 端运行小程序，扩展他们的商业场景，增加用户的活跃度。为了避免重复造轮子，大家共享小程序生态，也就需要我们从业务和技术上打通小程序的技术栈，输出支付宝小程序技术，帮助集团内的 BU 具

备小程序的运行能力。

目前支付宝小程序正逐步打通阿里生态，开发者可一次开发，阿里各大 app 多端运行，通过小程序连接阿里经济体。小程序对外输出的技术主要包含两个部分，一个是小程序运行时的 SDK，这个需要集成到接入的客户端里面，另一个是小程序的互通，这块需要接入的平台和小程序平台打通，大家共享同一个小程序生态。

小程序 SDK

小程序输出的 SDK 包含两个部分，基础引擎和能力插件，基础引擎是必须的，不可替换的，它承载了小程序的基础能力，包括前端框架和容器的核心能力，以及提供渲染的内核。它提供了小程序核心的运行时和基础的核心组件和 JSAPI，同时提供了能力插件的插件容器，插件容器有良好的隔离性，不会因为插件的 crash 导致容器的 crash，保障了小程序核心运行时的稳定性。



小程序互通

小程序的技术栈除了前端的框架和客户端的运行时，还包括开发者的入驻，小程序的创建，开发和上线，以及后续的运维和运营等管理，为了给用户和开发者较好的体验，小程序的互通是小程序技术输出的必须环节。

平台互通：开发者可以在入驻的开放平台管理投放到所有端的小程序，包括小程序的开发、调试、测试、发布、运维和管理等一系列的工作。

研发平台互通基于支付宝的开放平台能力，统一开发和发布流程，通过门户接入互通、开发者体系接入互通、审核能力接入互通、小程序研发链路接入互通、小程序运行链路接入互通，实现开发者一次开发、多端投放的能力。

运营管理平台通过统一埋点 SDK 提供多端小程序自动化埋点能力，输出标准化行为、异常与性能数据模型，通过数据分析平台，提供小程序在各端实时数据分析能力，并进一步提供用户特征分析、页面分析、用户留存分析支持小程序研发可视化数据自运营的能力。同时也支持小程序研发自定义数据采集点配置，并开放分析管理支持小程序内的用户行为做精细化跟踪、分析，满足除页面访问等标准统计以外的个性化分析需求。

工具平台提供给开发者统一的开发者工具，帮助开发者更好的开发和测试小程序，同时接入的端可以扩展开发者工具的模拟器和特色的 jsapi 接口，方便开发者做端内的特色能力调试。

能力互通：支付宝特色能力支付、会员、卡券、信用等能力可以通过扩展 jsapi 或者插件的方式输出到接入的客户端里面，同样的，接入的端也可以把自己的特色能力输出到小程序联盟的其他端里面，为更多的用户服务。基础能力所有端保持一致，客户端特色能力可以通过扩展 jsapi 的方式集成到小程序 api 里面，也可以通过插件的形式发布到插件市场，用户在使用的时候动态下载插件，屏蔽端上的差异。

用户互通：投放到多端的小程序，需要账号绑定，用户无需登录，给用户提供一致的用户体验。账户通 SDK 通过提供一套完整的注册、登录、授权、账号绑定管理等基础功能来完成多个 APP 间账户互通的功能，并保障整个过程安全可控。通过账户通可以拓展小程序服务覆盖边界，将支付能力、X 服务能力覆盖更多的客户，让服务通行便利、多端权益打通、多端体验统一。

小程序的 inside 技术栈不只是针对阿里集团内输出，也可以输出到外部的 app 商户，帮助 app 商户丰富业务场景，给用户提供更多有价值的服务。欢迎加入支付宝小程序联盟，通过小程序连接到阿里经济体，共同壮大小程序生态。

作者简介

白招拒，支付宝小程序首席架构师，2010 年加入蚂蚁金服，拥有 10 多年的互联网研发经验，对分布式系统，v8 引擎，客户端，前端等领域有深刻的理解。

百度新发布的智能小程序是什么？

作者 覃云

从 4 月份开始，江湖上就有传言说百度要上线小程序了，但那时百度官方既没有“辟谣”，也没有出面解释，直到今年 5 月 22 日，百度 App 业务部总经理平晓黎百度联盟峰会上给出了官方回应，宣布百度智能小程序将于 7 月正式上线。

7 月 5 日，在百度 AI 开发者大会上，百度副总裁沈抖正式对外发布了百度智能小程序。百度称，智能小程序不仅全面接入百度大脑的 AI 能力，更将在今年 12 月份全面开源，但是主题演讲上并没有过多地透露智能小程序技术的细节。为此，小编赶赴百度小程序分论坛为大家挖掘了智能小程序在技术和应用上的特点。

从运行方式、程序架构来看，百度的智能小程序其实是一个跨小程序/Web 的框架，最终会生成两套页面，在百度平台使用小程序，在浏览器加载 H5 页面。

以下为记者从会场的分享整理而来：

生态服务

百度认为，近两年，App 推广成本居高不下，H5 转化漏斗损耗大，H5 下运营方式单一，而且又难以将百度 AI 和大数据的能力结合起来，为了充分发挥自身的优势和提升运营质量，所以智能小程序诞生了。

百度智能小程序和微信小程序原理相似，可以运行在百度的平台，另外还可以运行在合作浏览器、合作 App、Dueros、Apollo 等上。

流量如何分发？

据了解，百度智能小程序在百度信息流中通过“主动分发 + 个性化推荐”触达用户。这也意味着信息流会通过对用户兴趣点的探索、数据的分析等，把用户最感兴趣、最需要的智能小程序

个性化、场景化地推送给用户。不同类型用户刷新信息流时，可以看到不同类型的智能小程序被主动推荐，这样的分发不仅更高效，也更精准。

昨天，百度副总裁沈抖曾宣布，将开放包括搜索、信息流等优势产品在内的千亿流量给开发者，让开发者充分享受超级入口带来的流量红利。

所以在百度搜索上，不管是在已有 H5 站点基础上改造的智能小程序，还是新开发的智能小程序，只要体验好，足够优质，都可以从搜索中获得流量。但是相比 **H5**，百度搜索会优先分发智能小程序。目前，百度 App 日活跃用户已突破 1.5 亿，这巨大的流量将会给开发者带来巨大的红利。7 月下旬，新版的百度 App 会开放智能小程序入口，其他的普通浏览器和其合作伙伴 App 也会陆续开放。

开发者如何接入？

如果开发者希望智能小程序接到搜索引擎上，开发者需要做什么？

众所周知，“熊掌号”是内容和服务提供者入驻百度内容生态的统一认证帐号，那么搜索也是通过熊掌号来连接包括智能小程序在内的互联网优质资源的。

开发者在智能小程序的平台注册以后会自动开通或者绑定一个熊掌号的帐户，在提交和开发智能小程序之后，注意，开发者需要把站长的二级域名映射到智能小程序服务器上，做这个映射以后，开发者就拥有了在自己域名下的智能小程序。百度会自动索引智能小程序，开发者也可以通过阿拉丁结构化数据提交方式在百度搜索和展现。

搜索引擎会自动抓取和理解、索引智能小程序，并且最终给智能小程序导流。如果是在百度 APP 环境下，会通过 Native 加载智能小程序，如果是苹果浏览器和谷歌浏览器，会通过急速框架加载智能小程序 Web 版。

如果已有 H5 站点，开发者只需一行代码就可迁移到智能小程序上。

开发者只需要在智能小程序页面里面增加一个 Canonical 标签，Canonical 标签指向原来的 H5 页面表达一个寄存关系。这里需要注意的是，智能小程序页面和 H5 页面需要一一对应，也就是 A 内容对应 A 内容，B 内容对应 B 内容。两个对应页面内容，主体内容一定是一致的，智能小程序和原来的 H5 网站必须是属于同一个组域，因为在同一个组域下才可以做这样的传递和继承。

H5 和智能小程序属于同一组域下，搜索会帮助开发者自动的把 H5 网站之前建立的 Seo 的优化，搜索积累的权重都无缝继承给智能小程序。

而且智能小程序的访问必须是安全的，必须遵循 https 的协议，这是一个最基本的要求，可以帮助智能小程序以更安全的方式来保护用户隐私不被窃取，减少智能小程序被劫持和假冒的风险。

如何开发智能小程序？

在智能小程序的开发技术上，百度 App 业务部前端架构师雷志兴给我们做了详细介绍：

从技术上来看，智能小程序和 Web 开发没有什么本质的区别，Web 分三个部分，HTML，JS、CSS，为了尽量做到和 Web 一致，降低开发成本提高运行效率，智能小程序做了一个自定义视图，SWAN、JS 和 CSS 在语法上没有什么区别，这个时候就不能使用 JS 的生命周期和事件，而是要使用智能小程序的生命周期和事件。

以上是智能小程序的代码，它定义了一个按纽，在一个 Will 中，下面绑定一个事件，接着，JS 使用的是自定义的事件，有一个对应关系，在 JS 中获取数据，把数据写在 JS 中，而智能小程序 CSS 和普通的 CSS 没有什么区别。

智能小程序采用的是 MVVM 的编程逻辑，整个应用有两层，逻辑层和视图层，逻辑层和视图层是分离的，JS 属于逻辑层，它包括智能小程序的生命周期、响应用户的事件，管理请求的数据，

还有各种各样的页面，视图的核心里面定义了容器，并且用 EFR 这样的语句控制展现，里面所有的数据都是从逻辑层获取的。

智能小程序技术架构

智能小程序的技术架构分成两部分，开发和运行，开发是包括管理平台开发者工具，开发者工具中主要是来调试看效果的。还有一个功能是搬家工具，搬家工具能够快速地将其他的小程序生态迁移到智能小程序生态上。

运行包括运行环境和赋能，运行环境的核心是浏览框架，浏览框架保证智能小程序的流畅运行，运行环境还包括别的基础能力，如存储网络等智能小程序必不可少的基础性能力。

上层的赋能分成两大块，第一块是组件，第二块是能力。

在组件上，百度智能小程序提供了 icon、表单、图片察看、进度条等减少开发工作的组件，还有可以通过客户端技术实现流畅的组件，如音频视频直播、地图画布等。

在能力上，智能小程序提供了地理位置、蓝牙网络等基本的 API 能力，也通过百度 AI 以及百度大数据提供的百度平台能力，如百度统计，百度统计目前有 PC 和移动版，接下来会有小程序版本，在智能小程序开发者后台中看到用户的来源、黏性、传播效果，这都是百度的平台赋予小程序云端的能力。

智能小程序的开发过程和 Web 开发完全一样，而运行部分分成上下两层，下面是运行环境，运行环境是保证小程序流畅运行的基础，在 Web 中实现对应的版本，在浏览器中实现了一套与 Web 等价的框架，在 Web 上智能小程序负责客户端的 AI 应用，而在上层实现了等价的 API 能力和云能力。

一次开发，多端运行是如何实现的？

正如上文所说，智能小程序和 Web 是相似的，它使用的技术都是 Web 技术，可以在浏览器上运行，由于智能小程序有逻辑层和视图层，逻辑层最核心的一点是 JS 引擎，不管客户端还是浏览器，都有 JS 引擎，它可以在不同的平台甚至不同的设备上运行。上面的设计层可以在客户端

部分使用 Web，部分使用客户端技术渲染，在别的平台使用别的渲染技术渲染，中间可以用别的链接，这就保证大家写的代码一次开发在不同平台得到运行。

在实际运行之前，智能小程序的源码，SWAN、JS 和 CSS 会被编成运行代码，一种是 JS，另外一种是 CSS，下一层是逻辑环境，包括业务框架、业务代码，这里面没有视图的逻辑，将数据发送给视图环境，视图环境中包括智能小程序基本框架和第三方组件以及 CSS。

这意味着我们通过分离逻辑环境和视图环境，能够用不同的技术在不同的平台上，让视图环境运行的更加流畅，也让体验和感受更好。JS 引擎是跨平台的，在开发的过程中，开发者使用的是百度提供的智能小程序开发者工具，在本地调试，看效果，开发完成以后，通过开发者工具上传按纽，将智能小程序上传到管理平台上，在管理平台中针对用户不同的环境，下发不同的代码包到对应平台，在 APP 中下载到的是完整的运行包，这个运行包里包括了逻辑环境和视图环境的所有东西。

如在百度 APP 中，拿到这个包以后直接把小程序运行起来，后续的操作就不需要下载了，只需要下载数据就可以了。在 Web 中同样生成一个包，这是部署在服务器上的，在浏览器打开链接

的时候，会用 H5 的方式渲染这个页面，能够达到在 H5 上较好的浏览和体验效果，有些东西在浏览器中并不能实现，比如 AR 技术，百度建议开发者可以下载 APP 来获得更加完整的体验。

关于开源

2018 年 12 月底，百度已经在 GitHub 上正式开源了智能小程序。大家也可以在 Github 中找到小程序前端核心框架 SAN。这是小程序在客户端运行环境最核心的部分，它是 MVVM 的框架，它具有体积小、性能高、兼容性好的特点，它不仅适用于小程序单页应用，也适用于传统的 Web 应用。

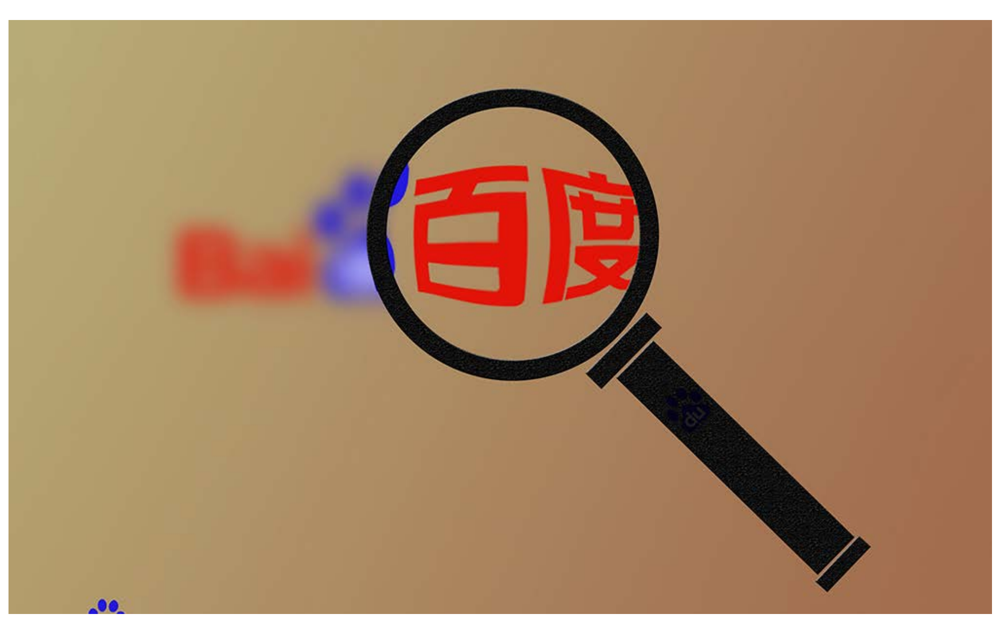
San 地址：https://github.com/baidu/san

百度智能小程序 GitHub 链接：https://github.com/swan-team

最后，百度表示，通过开放和开源建立智能小程序的技术生态，他们希望有越来越多的 APP 集成智能小程序的运行环境，他们也会在更多的设备和系统中集成智能小程序的运行环境，比如合作伙伴的 APP，智能语音操作系统，车载系统阿波罗，真正做到一次开发多次运行智能小程序。

苏宁：我们开发百度小程序遇到的那些“坑”

作者 马建旭



2018 年 5 月，受百度邀请，苏宁首批入驻百度小程序平台。作为微信小程序的开发工程师，我接手了“开荒”百度智能小程序（以下简称百度小程序）的任务，自 5 月初开发，一个多月就完成了苏宁易购百度小程序的开发工作，并在 7 月初的“百度开发者大会”上作为首批小程序对外提供服务。

从 5 月至今，作为百度小程序的主力开发，总结了以下的一些开发经验。

百度小程序的初探

入驻百度小程序

百度小程序的入驻方式，与微信小程序几乎相同，准备一个百度账号，登录百度智能小程序后台，选择入驻申请，选择适合的类型，填写表单信息，提交审核，一般 24 小时内可以审核通过。当然，如果有百度小程序的邀请码，也可以走邀请码通道进行入驻。这里特别需要注意的是，目前仅面向企业、政府、媒体及其它组织等非个人主体开放申请，个人用户是没有办法入驻的。入驻完成后，进入平台，可以看到自己的小程序。

开发前的准备

用注册得到的管理员账号登录后台，在成员管理设置项目组成员的权限，百度小程序的用户权限主要分为：开发者权限、开发管理、开发设置、暂停服务设置、推广设置、流量主、数据统计、小程序简介、小程序头像。在设置 - 开发设置，获取小程序的 AppID、AppKey 和 AppSecret，并设置服务器域名、业务域名和代理域名（小程序 web 化使用）。在设置 - 基础设置设置小程序名称、头像和简介等信息。

开发者工具

在从百度小程序官网点击文档 - 开发 - 左侧导航栏工具 - 界面下载开发者工具，开发者工具启动后，有和微信开发者工具差不多的界面，不同的地方如下：

- 百度小程序没有集成类似于微信小程序 Tgit、腾讯云，云开发平台等功能。开发者需要在自备代码管理工具，服务端所需要的服务器等资源。
- 百度小程序 IDE 目前的新建功能只支持新建一个 DEMO，开发者需要在这个基础上进行修改来得到自己的项目，或者使用搬家具将微信小程序转换为百度小程序，不能从头开始新建。微信小程序的新建功能是从输入 appID 开始的，百度的 appid 只能在建好项目后手动的修改。
- 百度小程序 IDE 不需要独立设置代理。微信小程序 IDE 需要单独配置代理，在复杂网络环境下可能会出现内外网不能同时访问的情况，但是百度小程序不需要设置单独代理。

开发文档

百度小程序有丰富的文档，详细的介绍了从申请入驻到小程序发布、从组件到 API 的百度小程序的方方面面，大家可以自行到官网上去阅读。

前端开发者眼中的百度小程序

因为同时开发百度小程序和微信小程序，我将对比微信小程序，讲述百度小程序开发过程中遇到的一些问题。

组件和 API 几乎和微信相同，框架上分为逻辑层、视图层、自定义组件和基础能力，也支持分包加载等能力。百度小程序的组件也分为视图组件、基础组件、表单组件、导航组件、媒体组件、地图组件、画布组件和开放能力相关组件。

在 API 方面，百度小程序也分为网络、媒体、文件、数据存储、位置、界面（包含绘图）、设备和开放接口等大类。以下是我们发现的一些百度小程序与微信小程序的不同。

组件的不同

百度小程序的列表渲染和微信不一致：

```
1 <view wx:for="{{array}}" wx:for-index="idx" wx:for-item="itemName">
2   {{idx}}: {{itemName.message}}
3 </view>
4 // 也可以简写为
5 <view wx:for="{{array}}">
6   {{index}}: {{item.message}}
7 </view>
8
9
```

百度小程序的列表渲染：

```
1   <view>
2           <view s-for="p in persons">
3                   {{p.name}}
```

```
4         </view>
5  </view>
```

百度小程序的判断和循环不能再同一个组件上。

例如微信小程序可以这么写：

```
1  <view wx:for="{{array}}" wx:if="{{item.isWx}}">
2    {{index}}: {{item.message}}
3  </view>
4  // 或者这样子
5  <view wx:for="{{array}}" wx:if="{{isWx}}">
6    {{index}}: {{item.message}}
7  </view>
8
9
```

而百度小程序则必须写成这边这个样子：

```
1    <view s-for="(index,item) in array">
2      <block s-if="item.isBd">
3        {{index}}: {{item.message}}
4      </block>
5    </view>
6    // 或者这样子
7    <block s-if="isBd">
8      <view s-for="p in persons">
9           {{p.name}}
10     </view>
11   </block>
12
```

API 的不同

- 百度小程序提供了 AI 的能力，可以实现文字识别、文本审核、语音合成、图像审核、图像识别和语音识别功能。

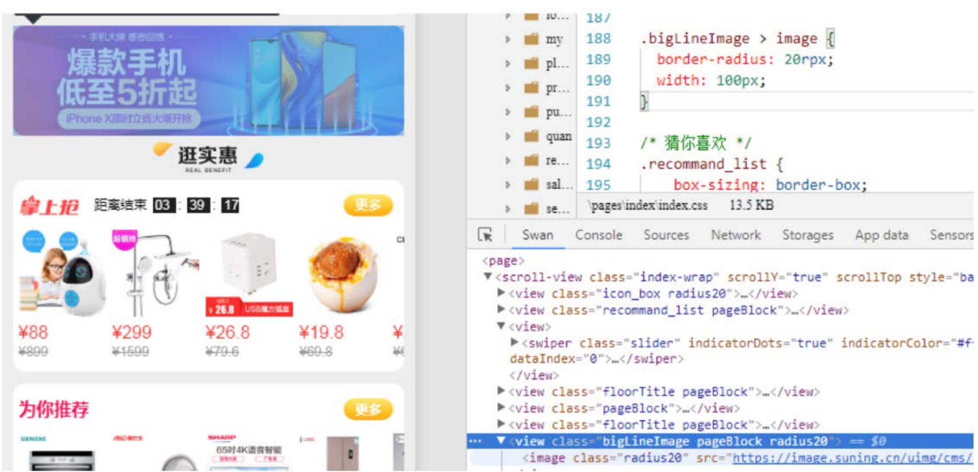
- Swan.request 能力。request 的问题主要体现在对单引号，以及 URL 的汉字的兼容上。百度小程序 request 能力并未对请求 UR 中的汉字做 encodeURIComponent 处理，导致手百客户端发送请求直接失败，这边建议开发者自己对 get 请求的入参做 encodeURIComponent 处理。单引号的问题也是这样，具体表现出来，请求发送出去之后，通过抓包可以看到请求正常的发送了，也正常返回了，但是小程序既不会走进 success 分支，也不会进入 error 分支，报错也不能被 catch，这个问题后期百度官方已经修复了。

分包体积限制不同

微信小程序目前的限制规则是，每个包不超过 2M，总包不超过 8M，而百度小程序目前的限制规则是主包不得超过 4M，每个分包不得超过 2M，总包不得超过 8M，这个规则和目前微信程序的限制规则差别较大，在方便了开发者的同时，可能会在性能方便有所损耗。

在 CSS 和 JS 处理的上的细微差异

百度小程序不支持 css 的 > 选择器，建议样式直接使用 class 选择器。



百度小程序支持的长度单位是 CSS3 的 vw，当然也支持微信小程序的 rpx。

使用伪元素实现的 0.5px 边框在百度小程序下会有异常，建议不要使用这种方法。

百度小程序不支持类似于微信小程序 WXS 的写法。百度小程序没有提供类似于微信小程序 WXS 的写法，但是提供了 Filter 过滤器。Filer 代码可以编写在 swan 文件中的标签内，或以 .filter.js 为后缀名的文件内。

登录方式的区别

因为微信客户端是强制要求用户登录的，但是手百不需要，所以在联合登录时，针对此场景，百度小程序需要作出特别的开发。在百度小程序联合登录前，需要使用 swan.isloginsync api 进行手百客户端的登录状态判断，手百没有登录的，不能使用联合登录，所以建议开发者还需要准备一套独立账号登录体系。

支付方式的区别

微信小程序使用的是微信支付，而百度小程序使用的是百度聚合收银台，在接入流程和开发流程上都和微信小程序不一样，当然百度小程序官方也贴心的提供了微信直联和支付宝直联的能力，大家可以根据自己的需要选用适合自己的付款方式。

打包方式及发布的区别

百度小程序点击预览按钮生成的开发版小程序是可以给其他用户扫码查看的，并未像微信一样严格要求开发者权限才能打开开发版小程序，也没有体验者这一角色，这一点对于测试同学来说还是很方便的。

百度小程序 IDE 提供了两种编译模式：依赖分析、普通编译。依赖分析模式：无用文件不会被打包到产出中，支持 node_modules 的使用；普通编译模式：不支持 node_modues 的使用，打包全部文件。

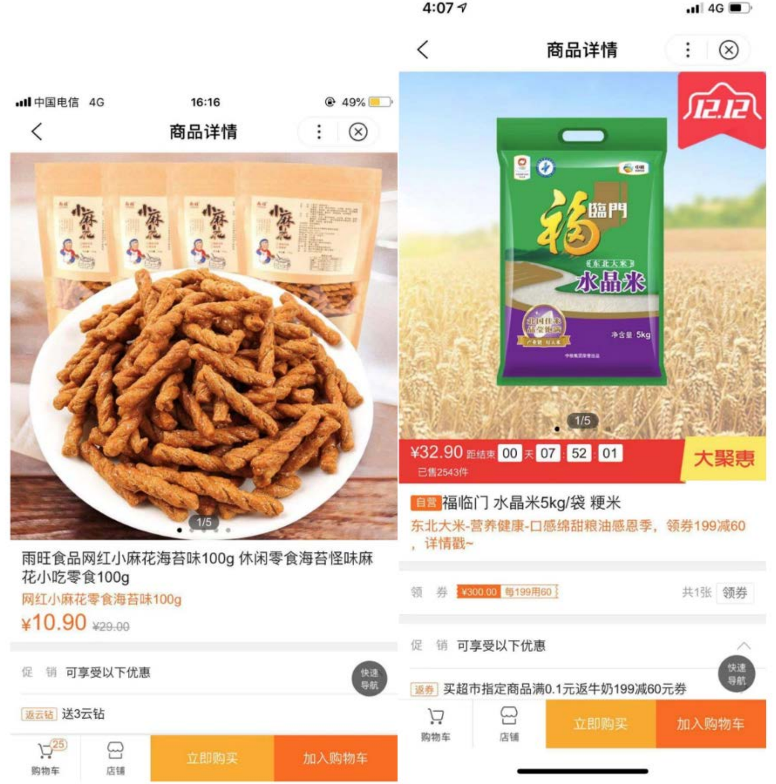
PhoneX 等机型样式适配

目前微信小程序并没有提供对 iphoneX，iphoneXS 等机型底部 bar 的适配，但是百度小程序提供了这样一个适配方案。

```
1   .swan-security-padding-bottom {
2
3   padding-bottom:34px;
4
5   }
6
7   .swan-security-margin-bottom {
8
9   margin-bottom: 34px;
```

```
10
11   }
12   .swan-security-fixed-bottom {
13
14   bottom: 34px;
15
16   }
17
```

该组样式会自动在需要适配安全区的场景动态注入，开发者不需要自行添加，只要在.swan 文件中使用这组类名即可。效果如下图所示：



如果你之前有开发微信小程序开发，百度小程序提供了一个搬家工具。搬家工具，是基于

Abstract Syntax Tree 开发的辅助工具，可以帮助您把微信小程序的部分代码迁移到百度智能小程序上。

工具可进行静态语法上的转换，根据一些规则去转换代码，抹平微信小程序语法和百度智能小程序语法上的差异，为大家减少因平台差异带来的苦恼。需要注意的是：工具做不到运行时 diff 的抹平，也做不到一个 API 从无到有的过程。所以，需要大家根据转换 log，进行二次开发。

网络的虚拟性导致信任关系难以建立，交易决策困难。对于小程序这种轻便、易传播的特性，如何让用户能够更容易的决策，并且给用户适合他的商品。这无疑是十分重要的一环，而与百度合作的手百小程序中，百度的基于用户操作行为的大数据商品推荐系统可以处理信任评估中的主观因素，提高交易预测的准确性。

基于双方协同过滤的思想建立一个模糊信誉管理系统，突出对商品信息的处理与个性化推荐。上线以来，百度推荐模块的数据对比于原有苏宁内部的推荐来看，更贴合于用户在基于百度强大的搜索端的行为数据。对垂直行业的定制化推荐，满足不同行业的需求，让我们看到了百度推荐的智能化、专业化。

以上是我在苏宁开发百度小程序的一些经验，各公司也可以尝试一下，百度小程序也在不断的迭代中，每一次迭代都能感觉到进步，感觉百度小程序开发组的同学，远程帮我们解决了很多问题，特别感谢百度方的李嘉辉来苏宁驻场提供技术支持，没有你，我们的开发之路不会这么顺利，以及百度的 QA 同学，对我们小程序提出宝贵的建议。

作者简介

马建旭，苏宁易购前端技术经理，苏宁易购小程序产品线技术负责人。曾任途牛旅游网在线预订前端负责人，具有丰富的前端开发经验，在 nodeJs、reactjs、vue 等多个领域均有技术实践。对前后端分离、前端组件化、小程序开发等方面，有独特的见解和丰富的实践经验。