快应用引入第三方 DSL Vue 的实践之路

1. 背景介绍

大家好，这里是快应用联盟的前端研发团队；

自去年 3 月 ,快应用联盟成立之后 ,已经有很多开发者使用快应用的标准 DSL（以 ux 文件后缀的项目形式）上线了对应产品。

以“旅游出行”的品类为例，就有：携程、去哪儿、高铁管家等；

关于更多的快应用产品与体验 ,读者可以在 Android 手机的应用商店 -> 分类 -> 快应用 栏目中查看；

今天呢，研发团队带来一个好消息，就是：**快应用开放平台接口，可以支持第三方的 DSL 啦**！

接下来分享主题 :以流行的 vue 框架为例 ,让快应用支持第三方 DSL 的开发能力；

那么为什么做这个事呢？主要还是为了满足前端同学的开发习惯，提升开发者的体验与效率。所以借助这种契机与接口开放的能力 ,快应用可以支持其他更

多的 DSL。

1.1 过往回溯

自从微信小程序从 16 年 10 月内测以来，前端开发在端适配上迎来了巨大的变化；开发者写的代码，不仅要满足 WEB 平台、原生渲染平台（RN 开发），而且还要增加对小程序的支持；

18 年 3 月成立了国内手机厂商成立了快应用联盟，随后又涌现了百度/阿里/头条等多个小程序生态，给开发者提出了更高要求，从"面向模块的开发"到"面向多端适配"；

可喜的是：在这种背景下，前端圈子里逐渐衍生出新的框架，就是希望能够提供统一的 DSL，让开发者编写一套代码，完成多端自适应的运行效果；

面对市场上众多的框架，新手开发者如何粗略了解与选择呢？

1.2 当前现状

从历史发展与职责目标的角度看，当前市场上的 DSL 框架可以分为以下 4 类：

1) WEB 型：

比如：React、Vue、Angular 等轻量级的数据驱动框架；

简述：主要用于浏览器的页面开发，因为语法简单、容易上手、调试方便而备受开发者的喜欢；

发展：因为拥有广泛的用户基础，逐渐发展多个子方向，如：UI 组件库方向（如 :Ant Design、ElementUI),简化版方向(移动端性能好 ,扩展更多能力)；

职责：提供开发者钟爱的语法，解决前端项目中组件化、分层架构、工程组织、数据流等问题，让开发者以最优雅的方式管理项目；

2) 平台型

比如：Weex 初期的 we 文件语法，微信小程序的 wxml 语法，快应用提供的 ux 文件、百度智能小程序的 swan 文件、Flutter 的 dart；

简述：主流的大互联网公司都会推出自有的渲染平台，从而为满足初期自身平台的渲染而提供一套自己实现的前端框架；

发展：每个平台拥有独特的语法，让开发者水土不服，针对该平台重新开发一套产品代码，学习成本较大；

职责：这类前端框架的存在主要是为了满足初版与迭代，更多服务于平台的系统能力提供，研发方向着重于技术深度的底层渲染（绘制、合成）；对于前端框架而言，仅满足开发需求，期望培养开发者习惯，并引领开发潮流较难，除非这类 DSL 可以同时生成到移动端等的适配；

3) 适配型

比如：Weex 上支持 Vue 语法，微信小程序中使用 Wepy 和 mpvue，以及本次介绍的快应用平台上引入 Vue 开发方式；

简述：介于诸多开发者对于上述平台型 DSL 不适应，从而引入前端受欢迎的

WEB 型 DSL；

发展：这里的发展思路差异比较大，有的是平台自身开发支持的，有的是通过DSL爱好者移植适配完成的 ,经历二次编译(WEB型DSL先转成平台型DSL，然后平台型 DSL 再转换成可以直接运行的平台编译代码)；

二次编译的优势在于：它不需要了解平台内部是如何实现的，仅需要根据官方提供的 DSL 能力进行能力适配即可；缺点在于：比较依赖平台型 DSL 能力，如果不支持某个特性则适配困难，或者容易造成性能瓶颈；

如果是平台自身支持的，那么开发者代码，直接就可以完成对 UI 的操作，跳过官方标准 DSL 的模型，减少中间调用，完成加速渲染；这种方式的难点在于：平台自身需要提供稳定的 UI 操作接口，做到向后兼容；

毫无疑问，平台自身的支持，能够比二次转换，带来更好的效果；

职责：尽管各自思路不同，但是目标一致，完成开发者从 WEB 到具体平台的顺利过渡；

4) 全能型

比如：滴滴的 chameleon、去哪儿的 nanachi、京东的 taro 等；

简述：该类型从上面的适配型开始萌芽，围绕如何解决多端适配问题，但这仅仅只是表象问题；对于后续壮大发展，需要思考面更广，对抽象概念理解更深刻，如： APP 容器管理，渲染设计，系统功能调用，动态加载等概念；

发展：尽量抹平 WEB、原生、快应用、小程序等渲染的差异 ,抽象应用模型，完成页面渲染设计，最后适配到各平台；当然适配时如果能够直接完成对平台 API 的操作，要比二次转换效果要好的多；

职责：完成较全面的多端适配，达到一套代码多端适配的目标；

那么本文讲述的快应用引入 Vue DSL ,属于上述的适配型 ,让平台自身支持，同时开发平台接口，为往后出现的全能高效型框架服务；

1.3 近期趋势

在作者看来，新的 19 年，全能型框架会逐渐取代适配型，并且从规范、架构、设计等角度上，提出新的理念与原则；基于此，各平台通过自身或者开源爱好者完成适配转换；

当然 ,各平台负责方(快应用、小程序)也会加深对统一的认识 ,借助于 W3C 研讨会、兴趣组、前端会议 ,促教交流 ,考虑抽象出自己的渲染 API 与能力通道，让更友好的全能型框架完成高效适配，这块敬请期待吧；

中间也必定会产生一些兼容性类库，完成 polyfill 的辅助角色；

所以基于这种趋势，快应用采用了这样的路线：开放页面渲染接口，轻松支持第三方的 DSL；

2. 实现方案

那么快应用本次支持 Vue 的 DSL 能力，都做了哪些事情呢？

接下来我们从渲染流程、架构设计、开发体验、项目代码、加载过程、平台解耦、测试保证的多个角度阐述。

2.1 渲染流程

要想完成适配，首先需要对比两方平台的页面渲染过程是否相似；经过抽象

汇总，得出主体过程都是这样的：

步骤 1. 工程化工具编译开发者使用某种 DSL 而编写的业务代码；

步骤 2. JS 引擎运行时加载完 DSL 框架后，执行开发者的业务代码；

步骤 3. 基于 DSL 的核心逻辑，生成 MVVM 的模型；

步骤 4. 业务中对数据的操作，触发对 DOM 节点的更新；

步骤 5. DOM 更新后渲染引擎，发出 VSync 申请，标记脏值节点；

步骤 6. 遍历待更新节点，依次样式布局计算、绘制合成，完成渲染；

当前两者实现的区别主要在于：线程的工作分配与协调机制、渲染实现的具体逻辑；然而这些对于 DSL 框架而言，是不需要关注深度实现的；

同时快应用自身会构建一套页面 UI 的 DOM 树，因此抽象出了一套 DOM 的 API 提供给 DSL；DSL 只需要调用快应用提供的节点操作接口，即可轻易完成适配；

为了方便理解 ,我们在 Github 上增加了 Vue 版本的 TodoMVC 的示例项目 quickappcn/todomvc-vue。

实际效果可以访问下面地址：https://github.com/quickappcn/todomvc-vue/raw/master/preview.gif

既然流程一致，那接下来就看如何架构设计，分层组织了。

2.2 架构设计

以当前支持 Vue 的适配为例，主要工作在于：编译时、运行时两方面；

- **编译时**

提供针对 DSL 的项目模板化、DSL 的解析编译能力，期间可以使用快应用组件与样式的校验解析接口；

当前快应用项目的开发，使用的是官方 hap-toolkit 工具，这是一个基于 NodeJS 的 npm 库；

关于项目的构建打包、调试等非 DSL 专有能力的，均已模块独立；项目结构采用模块化的开发方式，借助于 lerna 完成耦合分离；

DSL 开发者只需要开发对应的 DSL 模块，增加模板化、语法解析，即可完成适配。

- **运行时**

负责执行开发者的业务代码，管理 DSL 中的驱动模型，完成数据更新到 DOM 操作的转换；

快应用平台运行时会提供 DOM 的 API，针对每个页面提供一个 document 节点；

DSL 层除了包含官方 Vue 的源码逻辑之外，还有两部分：

✓ **DOM API 调用**：完成对节点的操作；

✓ **容器适配模块**：提供针对应用/页面概念的适配；

针对这块，快应用在 Github 提供了以下几个项目：

项目 quickappcn/Vue

从官方 Vue 站点克隆而来，保存 Vue 核心源码、以及针对快应用 DOM API 的适配；

项目 quickappcn/quickapp-dsl-vue

负责 DSL 在快应用平台上的应用容器适配，如：生命周期、事件通知等；

有了编译时/运行时的核心支持，其它工作（如：IDE 支持）就是相对较小的任务拆解了。

2.3 开发体验

对于使用 DSL Vue 的快应用开发者来说，会不会与标准 DSL(标准 DSL：ux 作为后缀名)开发方式，差别很大呢？

其实开发过程，与快应用标准的 DSL 项目开发方式基本完全一致，标准 DSL 的项目中展示的 ux 文件，DSL Vue 中展示的是 vue 文件；

使用方式如下：

步骤 1：全局安装 npm 库：*npm install -g hap-toolkit*

步骤 2：初始化项目：*hap init --vue*

步骤 3：构建项目：*npm run build*

步骤 4：运行在快应用的 APK 平台，开发者可以选择“本地安装“或者” 在线更新“的方式，与标准开发方式一致。

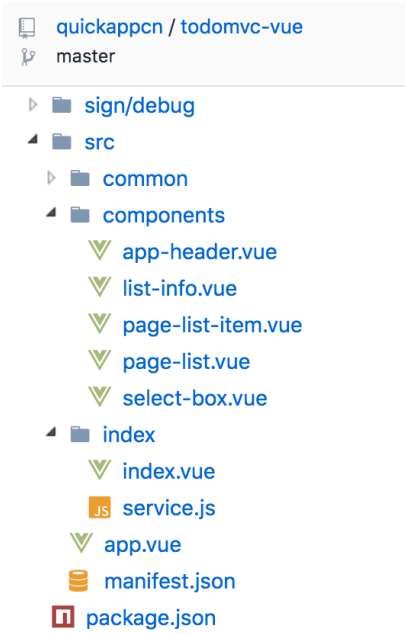
2.4 项目代码

总结一下，本次快应用为引入 Vue DSL 而提供的项目：

- **项目 quickappcn/todomvc-vue**

展示在快应用平台上运行该 DSL 项目的实际开发示例；

项目使用了组件化的开发方式，完成展示与表单的页面交互，文件组织结构如下图所示：



熟悉快应用开发的读者，会发现与标准 DSL 一样，这样方便快速上手。

- **项目 quickappcn/Vue**

从官方 Vue 站点克隆而来，提供针对快应用 DOM API 的适配；

项目中新建了一个 quickapp-initial 的分支，放置适配代码；

- **项目 quickappcn/quickapp-dsl-vue**

提供了 DSL 在平台上的应用容器适配，如：生命周期、事件通知等；同时包含针对上一个核心 DSL 源码项目的构建后代码；

为了辅助开发，开发者可以补充测试用例，完成单元测试、项目测试的功能保证；

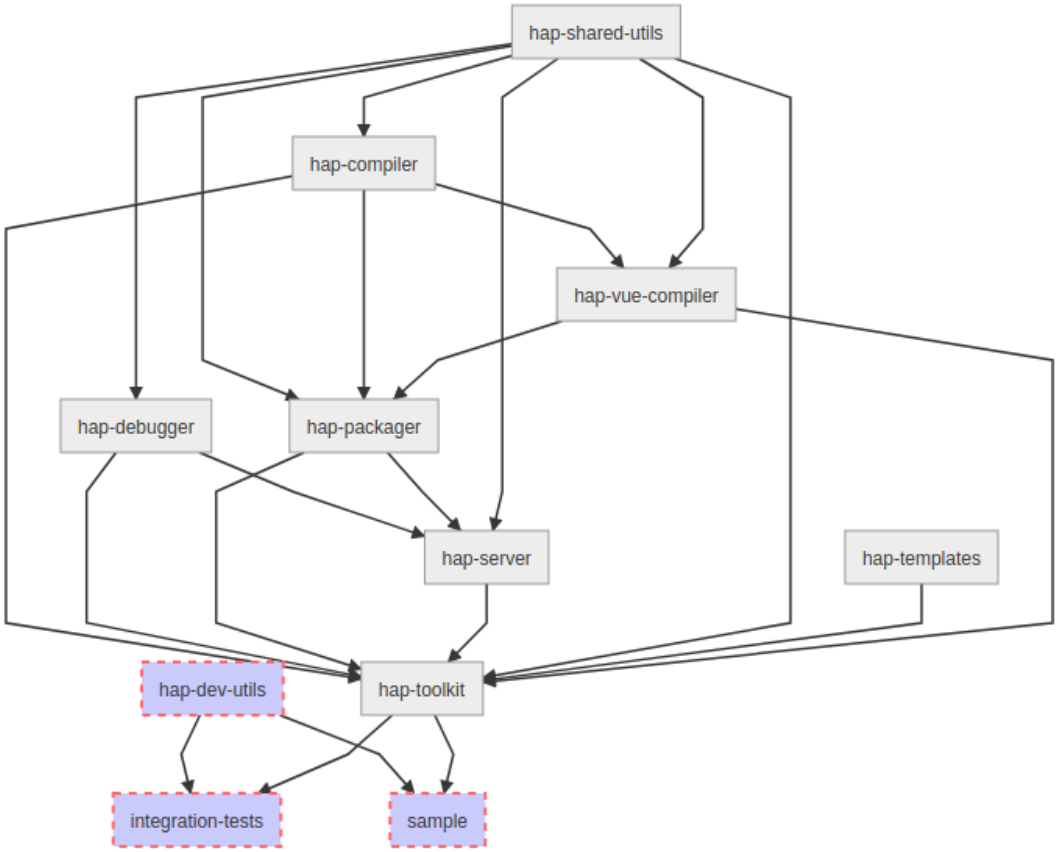
其中的单元测试：测试 Vue 的自身功能表现正常；

其中的项目测试：测试 Vue 在基于 NodeJS 的快应用模拟平台上，是否表现正常；

- **项目 quickappcn/hap-toolkit**

提供对开发者写的 DSL 的模板化、语法校验、项目打包等功能；

采用 lerna 模块化改造后，目前划分的模块的依赖关系如下图所示：



对于 DSL 开发者来说，只需要关注：hap-dsl-vue 的模块即可，这块的代码以源码的形式保存；其中的 templates 文件夹存放项目模板，src 文件夹存放相关的编译解析能力；

对于其他的部分模块，比如：hap-compiler，hap-server 属于所有的 DSL 共用模块，开发者一般无需更新；同时部分模块并未仅提供了编译后代码，如需开放源码，开发者可以下来联系；

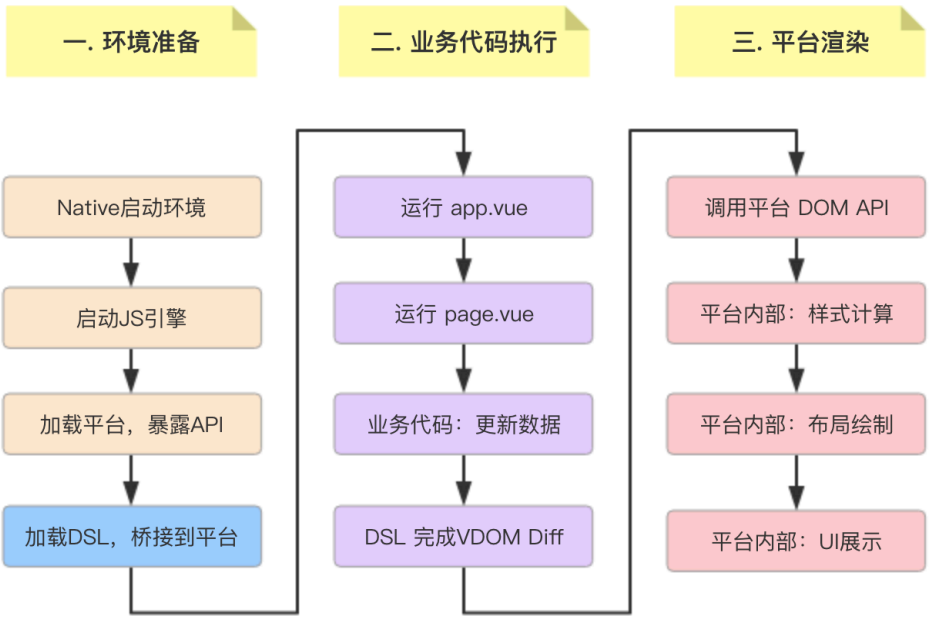
为了保证稳定性，也可以增加测试用例（当前使用的是 Jest），完成单元测试与项目测试的编译功能确认。

hap-toolkit@0.3.0 版本上增加了对 Vue DSL 的支持，不过并未采用 lerna 管理，后续发布的 0.4 版本以后会用这种方式；

注意：由于新业务功能的开发，当前模块化组织结构可能还会继续调整。

2.5 加载过程

在快应用完成编译时/运行时的开发后，DSL 是如何加载并调用渲染的呢？大家看下下面的图例就明白了；



快应用的运行可以分为三个阶段：

第一阶段：环境准备

底层平台启动，暴露 DOM 等相关 API，加载 DSL 代码，并完成与平台的桥接通讯；

第二阶段：业务代码执行

加载并执行开发者项目中的 vue 业务代码（编译后转换为 JS），建立驱动模型，完成 VDOM 的对比；

第三阶段：平台渲染

上一层 VDOM 对比的结果，转换成对平台的 DOM API 的实际调用，平台

线程然后做布局计算、绘制等完成界面的展示；

上图所示，可以得出：DSL 与平台的解耦与交互发生在第一阶段的最后一步，即：平台接口暴露之后，业务代码执行之前；因此整个运行，DSL 框架仅会加载一次。

2.6 平台解耦

那么 DSL 与平台需要考虑哪些方面的解耦事项呢？主要分为三个部分：

1) 容器管理

快应用是一个应用形态，包含多个页面，这点不同于浏览器，所以就会存在应用/页面的生命周期；

开发者需要监听这些生命周期，用于完成：数据请求、统计、性能监控；

为了保持解耦合，平台使用了 Publish/Subscribe 模型，DSL 只需要订阅相关的事件，即可暴露给开发者；

开发者可以从项目 quickappcn/quickapp-dsl-vue 的 src/shared 文件夹中得到提示；

2) 页面渲染

对于每个页面来说，页面的渲染依赖于组件树的构建，为了方便对组件进行操作，平台提供了一套类似浏览器的组件操作接口，称为：平台 DOM API；

为了提升 DSL 适配的简易性，快应用的 DOM 与浏览器中的 DOM 非常相似；

比如：创建节点的 API（*document.createElement()*）、节点增删的 API（*element.appendChild()*、*element.insertBefore()*）

在 Vue DSL 中，开发者都会使用哪些接口进行节点操作呢？

可以从项目 quicappcn/vue 的 src/platforms/quickapp/node-ops.js 文件中得到提示；

3) 接口功能

业务开发中，开发者肯定需要调用系统功能，如：fetch 请求：require('@sysem.fetch')、地理位置：require('@system.geolocation')

这方面的语法与平台的标准 DSL 语法保持一致，会在编译时进行转换，如：fetch 请求转换为：$app_require$("@app-module/system.fetch")$；

平台执行开发者的 JS 代码时，会自动注入一个全局函数$app_require$；那么 JS 执行时就会通过该函数完成系统功能的获取；

所以关于这块，DSL 适配不会有实际工作量；可以项目 quickappcn/quickapp-dsl-vue 的 src/dsls/vue/page/interface.js 文件中得到提示；

2.7 测试保证

对于 DSL 开发者来说，通过测试用例保证功能稳定是必不可缺的；

针对编译时，测试相对简单，查看项目 quickapp/hap-toolkit 即可读懂；

针对运行时，如果每次对源码修改后，都需要在手机设备上测试运行，确认功能的话，将会很浪费时间；

为了解决这一难题，快应用平台的前端层面，提供了在 NodeJS 环境上的平台模拟能力；因此开发者可以通过两方面的测试用例来保证稳定：

- **单元测试**

完成对 DSL 的基本功能进行测试，如：指令、filter、数据驱动等各种 DSL 自身特性；

相关代码请参考：项目 quickappcn/quickapp-dsl-vue 的 test/suite/dsls/vue/unit 文件夹；

测试命令请参考 项目 quickappcn/quickapp-dsl-vue 的 package.json 中的"test:suite:framework:main:vue:unit"命令；

- **项目测试**

开发者像开发正式的快应用项目一样，编写 DSL 页面；模拟平台会将测试项目编译打包，然后逐个执行开发者的页面代码；

开发者可以在这里，测试 DOM 树的结构一致性、生命周期、接口功能等；

相关代码请参考：项目 quickappcn/quickapp-dsl-vue 的 test/suite/dsls/vue/project 文件夹；

测试命令请参考 :项目 quickappcn/quickapp-dsl-vue 的 package.json 中的"test:suite:framework:main"命令；

3. 合作交流

如果您是快应用的开发者，或者其他角色，对于前端开发生态感兴趣，欢迎提出各类建议，或合作意向。

快应用平台对 DSL 能力的支持，前端与底层研发团队做了很多耦合分离工作，开源工作得以推进；同时感谢联盟内各家厂商研发的鼎力支持，大家合作共同管理项目源码与规范制定。

3.1 使用 DSL 开发

如果您也是 Vue 框架深度爱好者的话，可以考虑使用 Vue 来做快应用产品的开发；

围绕 Vue DSL 的最新能力与运行体验，我们将会在项目 quickappcn/quickapp-dsl-vue 中保持更新，您可以向这里提交 issue 反馈需求；

快应用的 Vue DSL 会在 1050 版本邀请内测，待功能稳定后，平台将内置正式版本的 Vue DSL；

注意：由于当前内测期间，Vue DSL 暂不支持华为设备，联盟内所有剩余厂商均可以无缝支持；

3.2 其它 DSL 接入

如果您是某个 DSL（如：React）的爱好者，或者某个全能型框架的设计者，有意向接入到快应用平台中，让更多的开发者受益，欢迎洽谈垂询；

您可以通过联盟的任何成员/各种渠道联系前端团队，或者向我们发送邮件：dongyongqing#xiaomi.com；

谢谢！

3.3 简历推荐

如果您对快应用的研发工作感兴趣，有意提升对平台的架构/设计能力，或者能够跨越浏览器的限制提出更多的规范思路，欢迎联系我们；