前端框架方案

目 录

[第一章 Web前端 5](#_Toc446800116)

[第一节 概述 5](#_Toc446800117)

[第二节 前沿思想 5](#_Toc446800118)

[一、 组件化开发 5](#_Toc446800119)

[二、 SPA 6](#_Toc446800120)

[三、 REST 6](#_Toc446800121)

[四、 工程化 6](#_Toc446800122)

[五、 FLUX 7](#_Toc446800123)

[第三节 前端框架 7](#_Toc446800124)

[第四节 主流前端框架 7](#_Toc446800125)

[一、 Backbone.js 7](#_Toc446800126)

[二、 ReactJs 8](#_Toc446800127)

[三、 Vue.js 8](#_Toc446800128)

[第二章 方案一——AngularJs 8](#_Toc446800129)

[第一节 概述 8](#_Toc446800130)

[第二节 核心思想 9](#_Toc446800131)

[一、 MVVM 9](#_Toc446800132)

[二、 模块化和依赖注入 9](#_Toc446800133)

[三、 双向绑定 10](#_Toc446800134)

[第三节 主要功能 10](#_Toc446800135)

[一、 语义化标签 10](#_Toc446800136)

[二、 路由 10](#_Toc446800137)

[三、 模板引擎 10](#_Toc446800138)

[四、 国际化和本地化 11](#_Toc446800139)

[五、 支持单元测试和端对端测试 11](#_Toc446800140)

[第四节 目录结构 12](#_Toc446800141)

[第五节 优点 12](#_Toc446800142)

[一、 完整的前端框架 12](#_Toc446800143)

[二、 强规范 13](#_Toc446800144)

[三、 全面和强大的社区 13](#_Toc446800145)

[四、 高维护性 13](#_Toc446800146)

[五、 高可测试性 13](#_Toc446800147)

[第六节 缺点 13](#_Toc446800148)

[一、 学习曲线陡峭 13](#_Toc446800149)

[二、 SEO 14](#_Toc446800150)

[三、 断崖升级 14](#_Toc446800151)

[第三章 方案二——规范型框架 14](#_Toc446800152)

[第一节 概述 14](#_Toc446800153)

[第二节 MVP模式和代码结构 14](#_Toc446800154)

[一、 MVP模式 14](#_Toc446800155)

[二、 代码组织结构 15](#_Toc446800156)

[第三节 主要功能 15](#_Toc446800157)

[一、 模块化 15](#_Toc446800158)

[二、 View层控制 16](#_Toc446800159)

[三、 模板引擎 16](#_Toc446800160)

[四、 路由 16](#_Toc446800161)

[第四节 优缺点 16](#_Toc446800162)

[一、 优点 16](#_Toc446800163)

[二、 缺点 16](#_Toc446800164)

[第四章 前端重构 17](#_Toc446800165)

[第一节 框架选型 17](#_Toc446800166)

[一、 方案对比 17](#_Toc446800167)

[二、 总结 17](#_Toc446800168)

[第二节 多版本支持 18](#_Toc446800169)

[一、 文字描述不同： 18](#_Toc446800170)

[二、 页面级的功能模块增减： 18](#_Toc446800171)

[三、 功能细节不同： 18](#_Toc446800172)

[四、 总结 18](#_Toc446800173)

[第三节 重构计划 18](#_Toc446800174)

# Web前端

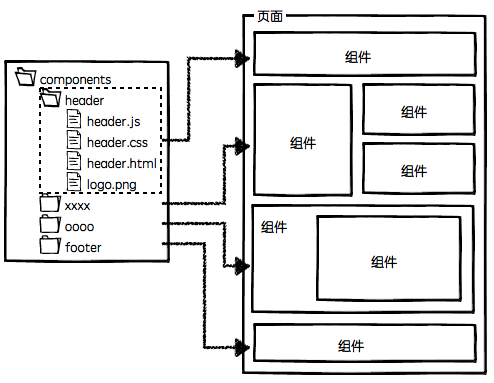
## 概述

随着时代的发展，前端从最初单纯的页面展示，转变为了具有多样化展现方式、交互更人性化和更重视用户体验的应用。这也导致前端代码量大增、团队扩大，随之而来的团队协作、代码管理、代码质量等问题也越来越影响开发效率。那么使用框架也是顺理成章的事情，通过框架来引入类似模块化、mvc等编程思想，完善代码规范，进行单元测试，使前端更容易管理和优化。

## 前沿思想

### 组件化开发

组件化包括对数据逻辑层业务梳理，形成不同层级的能力封装。它不一定是一个自定义语义标签；它可以是一个包含逻辑(js)、样式(css)、模版(html)的功能完备的结构单元，也就是我们常“口口相传”的模块；它也可以是一个单纯的js，比如http组件这种纯逻辑单元。



简单理解组件化开发即是把整个页面切为一个一个的组件，每个组件相互独立，并且有对应的目录存放组件的资源。

由于组件间相互独立，可以根据需求随意替换调整组件，不会对其他部分造成影响。而且这种“分而治之”的思想也更容易分配工作，提高团队协作的能力。

### SPA

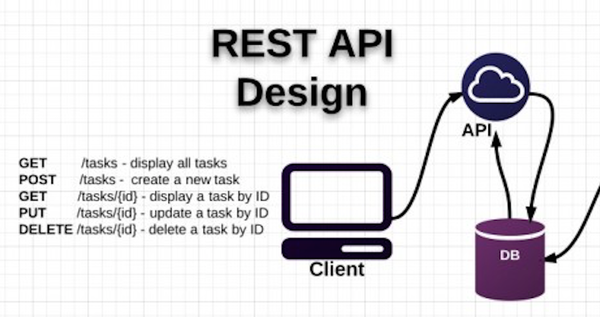
SPA（Single Page Application），即是单页面应用。只有一个页面，采用js控制，不重载页面，只刷新部分页面元素。

SPA在切换页面只需要加载部分html模板和json字符串，大大增加了响应速度，还减少了服务器的压力和带宽消耗。另外SPA更好使前后端高度分离，同一套后端能应对多个平台。

但SPA不利于SEO。

### REST

Representational State Transfer，这是一种架构风格，目标是构建可扩展的web service。REST规范可以提高架构的性能和可维护性。RESTful（采用REST架构规范的）系统通常是通过HTTP协议，并且使用HTTP的GET,POST,PUT,DELETE等动词来收发数据。



### 工程化

前端工程化就是以系统化、 规范化、 数量化等工程原则和方法去进行前端开发和维护。

首先，采用能最大程度帮助代码管理和团队协作的项目结构构建项目。

其次，统一编码风格和规范。

第三，在部署前可以对代码进行质量检测和单元测试。

最后，进行代码优化处理，采用利于运行的目录结构，部署项目。

### FLUX

Flux是Facebook用来构建用户端的web应用的应用程序体系架构。它通过利用数据的单向流动为可复用的视图组件提供了补充。Flux应用主要包括四部分：dispatcher、Action、store和views。



FLUX的核心特性是单向数据流，解决了传统 MVC框架下View和Model双向绑定导致的关系混乱。

## 前端框架

框架是一种特殊的、已经实现了的WEB应用，你只需要对它填充具体的业务逻辑。这里框架是起主导作用的，由它来根据具体的应用逻辑来调用你的代码。

框架有三个优点：首先，框架可以让程序员更专注于业务；其次，框架有较为完善的规范，能拉平整个团队的技术水平；最后，框架会在底层处理部分浏览器的差异，能提高web app的浏览器兼容性。

## 主流前端框架

### Backbone.js

Backbone顾名思义，只是mvc架构的骨干，与其他mv\*框架相比，也相对原始，更纯粹，开发人员可干预的东西也相对多一些，但是view层的实现也相对比较落后的。

### ReactJs

React 是一个Facebook和Instagram用来创建用户界面的JavaScript库，用于解决构建随着时间数据不断变化的大规模应用程序。React的核心内容：

* virtual dom：react中的组件跟页面真实dom之间会有一层虚拟dom。
* 单向数据流：flux架构下的数据流呈现出一种单向、闭环的流动路线，使得一切行为变的可预测，也能更好的定位错误发生点。
* 状态机：每个组件都是状态机，组件关注的只应该是状态，不同的状态呈现不同的表现形式。
* 组件：一切都是组件。
* Immutable object：使用只读数据来建立数据模型，每次更新都是new object。

### Vue.js

Vue.js是国内一个用于创建 web 交互界面的库，是一个精简的MVVM。vue专注于 MVVM 模型的 ViewModel 层，通过双向数据绑定把 View 层和 Model 层连接了起来。

相较其他mv\*框架，vue轻量、易用、更优异的性能及面向组件化的设计。而且阿里手淘团队发布了名为“无线电商动态化解决方案”的 Weex，该方案就是基于vue.js。

# 方案一——AngularJs

## 概述

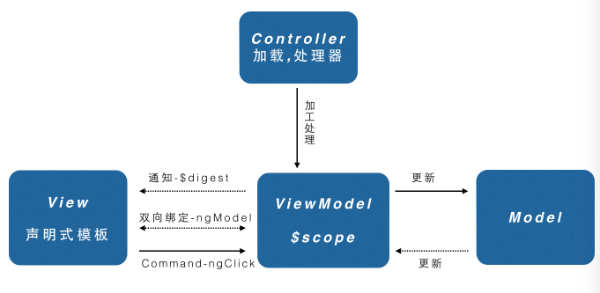
AngularJs是一款开源JavaScript库，由Google维护，用来协助单一页面应用程序运行的。框架采用并扩展了传统HTML，通过双向的数据绑定来适应动态内容，双向的数据绑定允许模型和视图之间的自动同步。因此，AngularJS使得对DOM的操作不再重要并提升了可测试性。

AngularJs更适合CRUD应用，例如阿里云管理控制台。而对于dom操作频繁且复杂的应用，像游戏或图形界面编辑器，就不太适合使用AngularJs。

## 核心思想

### MVVM

MVVM模式是Model-View-ViewMode（模型-视图-视图模型）模式的简称，是MVC模式的衍生物。



View负责于界面的显示和渲染，在angular中则是包含一堆声明式Directive的html模板。

Model是与应用程序的业务逻辑相关的数据的封装载体，Model并不关心会被如何显示或操作，所以Model也不会包含任何界面显示相关的逻辑。在Angular中的Service则是封装和处理这些与Model相关的业务逻辑的场所，这类的业务服务是可以被多个Controller或者其他Service复用的领域服务。

ViewModel是View和Model的粘合体，负责View和Model的交互和协作，它负责给View提供显示的数据，以及提供了View中Command事件操作Model的途径；在angular中$scope对象充当了这个ViewModel的角色；

Controller虽然不是MVVM模式的核心元素，但它负责ViewModel对象的初始化，它将组合一个或者多个Service来获取业务领域Model放在ViewModel对象上，使得应用界面在启动加载的时候达到一种可用的状态。

### 模块化和依赖注入

模块化是一种将系统分离成独立功能部分的方法，可将系统分割成独立的功能部分，严格定义模块接口、模块间具有透明性。模块化使js开发关注分离，提升代码复用性，利于测试和团队协作。

依赖注入是一个在组件中给出的替代了硬的组件内的编码它们的依赖关系的软件设计模式。这减轻一个组成部分，从定位的依赖，依赖配置。这有助于使组件可重用，维护和测试。

### 双向绑定

双向绑定，是从界面的操作能实时反映到数据模型，数据模型的变更能实时展现到界面。

双向绑定减少了大量的DOM操作，使前端开发从dom驱动开发，变为数据驱动开发。使视图和模型分离解耦，更容易测试。

## 主要功能

### 语义化标签

AngularJs采用指令（directive）实现自定义标签，例如<taskList></taskList>这样有实际意义名称的标签。在directive内部指定改标签对应html模板，当页面被浏览器加载后，自定义的标签将被替换为模板。

语义化标签能极大增加html的可读性，并且更利于实现组件化开发。

### 路由

路由就是页面跳转控制。基于AngularJs路由功能，就可以实现SPA应用。通过拦截符合规则的url，来切换页面，并且可以只切换部分页面。

路由功能可以使前后端分离更彻底，页面跳转由前端控制，后端只负责数据。

### 模板引擎

由于ajax请求的一般是数据，而不是html，所以前端需要在js里面进行html字符串拼接，来生成对应的视图。大量的字符串拼接不但严重降低了代码的可读性，更增加了维护难度。

对于处理这类问题，就产生了模板引擎。模板引擎通过在html模板是声明对应的数据模型，运行过程只需要传入数据模型，即可生成dom，大大减少了代码量和代码可读性。

### 国际化和本地化

国际化（i18n）是指在设计软件和编码的过程中，将软件与特定语言及地区脱钩的过程。当软件被移植到不同的语言及地区时，软件本身不需要做内部工程上的修改或者修正。

本地化(l10n)是指将一个软件产品按照特定国家/地区或语言市场的需要进行加工，例如语言，数字、日期时间格式和姓名顺序等。

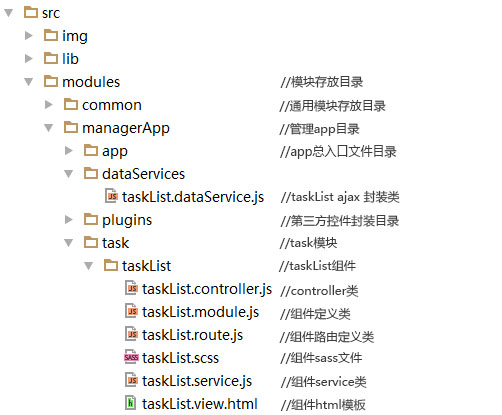
AngularJs通过angular-translate扩展，将界面语言从代码中抽离成单独的文件，实现多语言切换。通过filter对日期、数字和货币进行过滤改变其格式。

### 支持单元测试和端对端测试

AngularJs本身就具有很高的测试性，并且设计团队还提供了angular-mocks和protractor来辅助进行测试。

自动化的单元测试和端对端测试可以在一定程度上保证产品的质量。

## 目录结构



* 目录结构采用app – 模块 – 组件。
* app文件夹内放该app的配置文件和html页面。
* dataService是专门用于封装ajax请求，类似于DAO层。
* 采用组件化思想，一个组件的资源放在一个目录下。

## 优点

### 完整的前端框架

angularjs是一个全面的框架，不需要太多的工作，就可以使用于大部分的业务场景。 module和依赖注入系统，controller中定义的数据和事件，service实现不同组件之间共享数据，filter处理筛选数据，forms支持表单和复杂的表单验证，简单的动画模块animations，强大的directive实现指令和指令的嵌套。ui组件有bootstrap for angular，路由有ui-router，还有promise模块$q，还有原生的$resource模块直接支持标准的restful接口，集成的单元测试，等等。

### 强规范

Javascript作为脚本语言，没有比较完善的规范，导致js的上下限都非常高。不同人的代码，不仅风格可能不一样，而且代码健壮性都会差很多。

而angularJs从代码组织到规范都有很强的约束性，弥补了js在这方面的不足。

### 全面和强大的社区

AngularJs面世几年，已经非常成熟，不仅有google和社区的支持，在github上的项目和StackOverflow的问答都很多。

### 高维护性

前端代码逻辑分层，分而治之，能快速地位定位问题，部分修改也不容易影响其他部分代码。

视图和模型分离。哪怕页面被重新设计，也只需要修改视图层，并不会影响模型层代码。

### 高可测试性

AngularJs能完美支持单元测试和端对端测试。

## 缺点

### 学习曲线陡峭

AngularJs在开发思路上和Jquery有本质的区别。Jquery是以DOM驱动开发的，所有逻辑都围绕着DOM，甚至数据都绑定在DOM上。而AngularJs是以模型驱动开发的，只关心数据模型的变化，而操作DOM的事情由AngularJs干。

另外AngularJs引入很多编程思想和使用了很多新名词，都加大了学习难度。

### SEO

由于很多爬虫不会运行js，导致爬过去的是空页面，所以AngularJs不利于SEO。

### 断崖升级

AngularJs一直是断崖升级，1.08时兼容IE6-8，到1.3时就完全不支持了。而AngularJs2直接和1代完全不同。

# 方案二——规范型框架

## 概述

不使用现成的框架，而是通过文档规范去约束代码书写和组织方式，形成合理的架构。

重要的核心功能采用第三方库实现，例如模块化采用RequireJs、视图层控制采用Jquery。

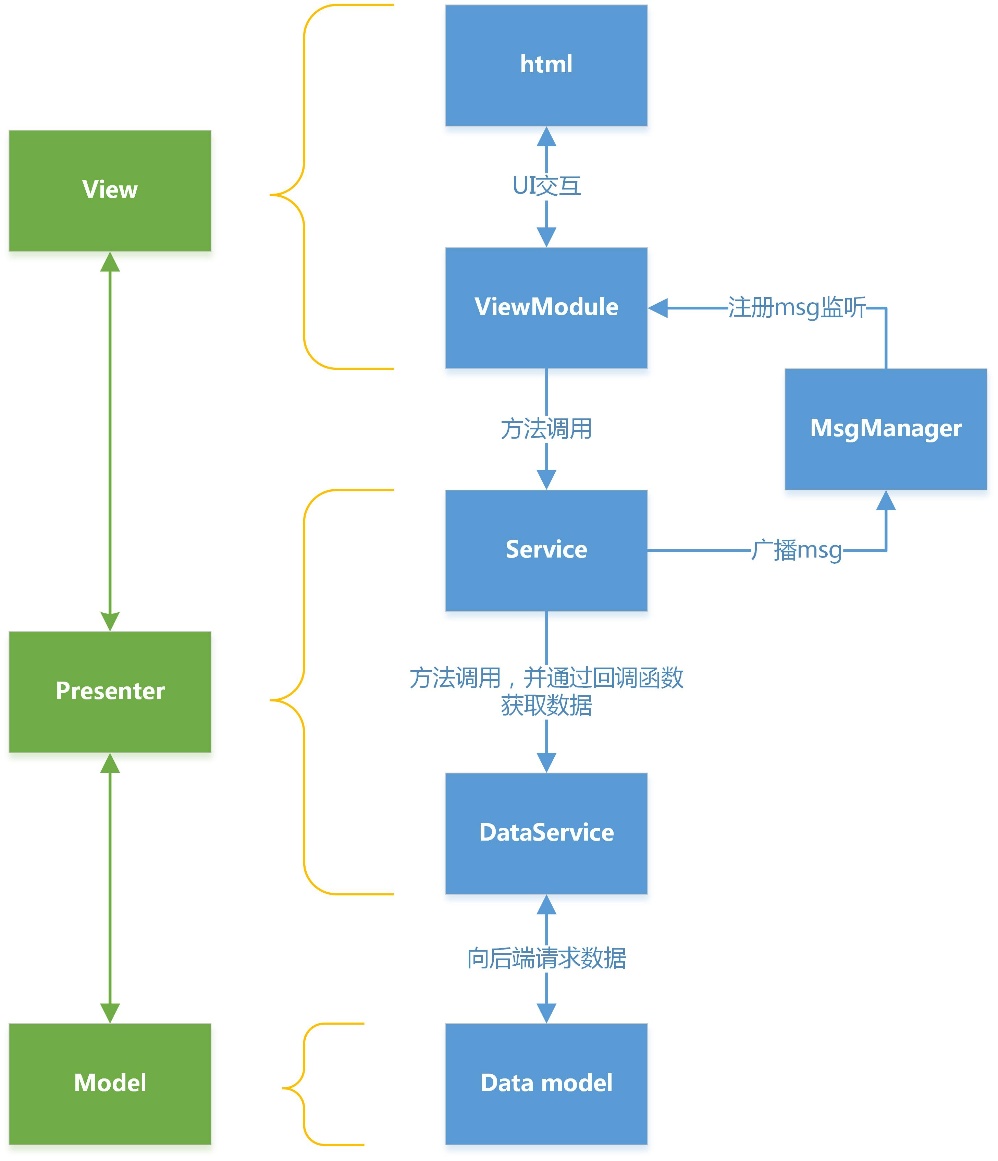
## MVP模式和代码结构

### MVP模式



MVP模式是由MVC模式衍生出的。其中Presenter负责逻辑的处理，Model提供数据，View负责显示和交互。而区别于MVC，MVP中View和Model不能直接关联，所有的通信必须通过Presenter。

### 代码组织结构



* DataService：类似于DAO层，只负责从后端请求数据，并且对请求结果进行处理，然后通过回调函数将数据模型传回Service。
* Service：负责业务逻辑和保存数据模型。
* ViewModule：负责视图层控制，例如事件绑定和处理。
* MsgManager：统一只有一个，用于分发消息，防止ViewModule和Service相互依赖，Service可以通过MsgManager广播msg，通知多个ViewModel；ViewModel通过注册，监听msg。

## 主要功能

### 模块化

模块化采用RequireJS实现，模块定义遵从AMD规范。RequireJs还可以用于管理第三js插件。

### View层控制

View层的视图变化，事件响应采用jQuery。

### 模板引擎

artTemplate是由腾讯团队出的一款JavaScript模板引擎。artTemplate除了有较高的性能，还能定位模板错误位置。

注：[各种js模板引擎性能对比](http://blog.csdn.net/wuchengzhi82/article/details/8938122)。

### 路由

路由采用StateMan，一个用于创建复杂层级状态SPA的路由库。

## 优缺点

### 优点

* 学习难度低。
* 更灵活，更容易集成各种第三方插件。
* 由于约束很弱，并且尽可能少的引用其他插件，更容易移植到其他框架内。

### 缺点

* 因为未引入框架，所以代码量完全没减少。

# 前端重构

## 框架选型

### 方案对比

|  |  |  |
| --- | --- | --- |
|  | [AngularJs](#_方案一——AngularJs) | [规范型框架](#_方案二——规范型框架) |
| 代码约束力 | 强 | 弱 |
| 核心功能 | 齐全，且风格统一 | 需要找各种第三库或者自己实现，代码风格迥异。 |
| 扩展插件 | 社区大，扩展插件多。 | 依托jQuery，扩展插件非常多。 |
| 第三方库兼容性 | 除非是由AngularJs写的库，否则兼容较差，而且可能和AngularJs核心思想冲突。 | 兼容性强。 |
| 编程思想 | 有很多编程思想和设计模式，而且比较接近java。 | DOM驱动开发略落后。 |
| 灵活度 | 中。 | 高。 |
| 学习曲线 | 入门简单，深入高，曲线陡峭。 | 低。 |
| 代码量 | 减少视图控制部分的代码量。 | 不减少代码量 |
| 浏览器兼容性 | IE9+ | IE8+ |

### 总结

结合公司目前状况，采用方案二更合适。

方案二采用的两个主要库(jQuery和requireJs)已经在使用，学习难度低很多。其次目前项目使用的jQuery控件，也无需再封装，重构工作量也要相对小。另外由于方案二代码级约束弱，也方便移植到其他框架内。

## 多版本支持

### 文字描述不同：

利用国际化插件，将页面文字独立在文件中，不同平台的描述存放在不同的描述文件中。加载对应的文件，页面将显示对应的描述。

### 页面级的功能模块增减：

不同平台支持不同的功能模块，从两个方面控制。第一通过配置文件动态控制页面元素生成，在视图层控制功能模块入口显示。第二通过java Controller层ajax接口，屏蔽不同平台的功能请求。

### 功能细节不同：

类似支付状态之类的显示细节变化，通过css控制。

类似支付情况控制下载的部分，是js内的细节变化。利用配置全局变量，在js内写判断语句处理。（但这种处理会导致js代码性能和可读性降低，尽可能在设计层面规避）。

### 总结

基于上述三点多版本的情况，是可以统一在一套代码内，通过解析url地址，使用不同配置文件动态应对不同的版本。

对于页面布局变动较大的版本，只能重做Html和ViewModule。功能和数据模型没变的话，其他代码可重用。

## 重构计划

* 重构不影响新功能开发。
* 重构以页面为级别重构。
* 对于老页面进行功能增强时，时间允许可重构顺带将功能增强完成。