框架设计

**Vue框架概述**

1. **Vue发展历史简介**

1.1重要版本发布

* 2013年，在 Google 工作的尤雨溪，受到 Angular 的启发，开发出了一款轻量框架，最初命名为 Seed 。
* 2013年12月，更名为 Vue，图标颜色是代表勃勃生机的绿色，版本号是 0.6.0。
* 2014.01.24，Vue 正式对外发布，版本号是 0.8.0。
* 2014.02.25，0.9.0 发布，有了自己的代号：Animatrix，此后，重要的版本都会有自己的代号。
* 2015.06.13，0.12.0，代号Dragon Ball，Laravel 社区（一款流行的 PHP 框架的社区）首次使用 Vue，Vue 在 JS 社区也打响了知名度。
* 2015.10.26，1.0.0 Evangelion 是 Vue 历史上的第一个里程碑。同年，vue-router、vuex、vue-cli 相继发布，标志着 Vue从一个视图层库发展为一个渐进式框架。
* 2016.10.01，2.0.0 是第二个重要的里程碑，它吸收了 React 的虚拟 Dom 方案，还支持服务端渲染。自从Vue 2.0 发布之后，Vue 就成了前端领域的热门话题。
* 2019.02.05，Vue 发布了 2.6.0 ，这是一个承前启后的版本，在它之后，将推出 3.0.0。
* 2019.12.05，在万众期待中，尤雨溪公布了 Vue 3 源代码，目前 Vue 3 处于 Alpha 版本。

1.2 Vue 1.0

Vue最初的目标是成为大型项目的一个良好补充。

设计思想是一种“渐进式框架”，淡化框架本身的主张，降低框架作为工具的复杂度，从而降低对使用者的要求。

1.2.1 主要改进

1. 提供指令的缩写
2. 针对v-bind和v-on提供缩写形式：

<!-- v-bind -->

<div v-bind:class="box">全写</div>

<div :class="box">缩写</div>

<!-- v-on -->

<button v-on:click="btn">全写</button>

<button @click="btn">缩写</button>

1. 清理精简所提供的接口
2. 提高初始化的渲染效率
3. 将v-repeat指令换成了v-for指令。同时优化了这个指令的渲染，效率提升了一倍。
4. 两个官方工具的增强：vue-loader和vueify。

1.3 Vue 2.0

2.0版本对Vue做了大幅度的重构，性能有了很大的提高，也为日后的跨端发展打下了基础。

1.3.1 性能

2.0 用一个 fork 自 snabbdom 的轻量 Virtual DOM 实现对渲染层进行了重写。

在其上层，Vue 的模板编译器能够在编译时做一些智能的优化处理，例如分析并提炼出静态子树以避免界面重绘时不必要的比对。

新的渲染层带来了巨大的性能提升，也让 Vue 2.0 成为了最快速的框架之一。

1.3.2服务端渲染

Vue 2.0 支持服务端渲染 (SSR)，并且是流式的，可以做组件级的缓存，这使得极速渲染成为可能。

1.4 Vue 3.0

3.0 是非常大的重构，源码使用 TypeScript 重写，目前的代码 98% 以上使用 TypeScript 编写。

根据 Vue 官方时间表，要等到今年第一季度才有可能发布 3.0 正式版。

**2.什么是Vue**

Vue.js是一套构建用户界面的渐进式框架。与其他重量级框架不同的是，Vue 采用自底向上增量开发的设计。Vue 的核心库只关注视图层，并且非常容易学习，非常容易与其它库或已有项目整合。另一方面，Vue 完全有能力驱动采用单文件组件和Vue生态系统支持的库开发的复杂单页应用。

Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

Vue.js 自身不是一个全能框架——它只聚焦于视图层。因此它非常容易学习，非常容易与其它库或已有项目整合。另一方面，在与相关工具和支持库一起使用时，Vue.js 也能完美地驱动复杂的单页应用。

**3. Vue的优点**

（1）轻量级框架

只关注视图层,是一个构建数据的视图集合,大小只有几十kb

Vue.js通过简洁的API提供高效的数据绑定和灵活的组件系统

（2）简单易学

国人开发,中文文档,不存在语言障碍,易于理解和学习

（3）双向数据绑定

也就是所谓的响应式数据绑定。这里的响应式不是@media 媒体查询中的响应式布局，而是指vue.js会自动对页面中某些数据的变化做出同步的响应。

也就是说，vue.js会自动响应数据的变化情况，并且根据用户在代码中预先写好的绑定关系，对所有绑定在一起的数据和视图内容都进行修改。而这种绑定关系，就是以input 标签的v-model属性来声明的，因此你在别的地方可能也会看到有人粗略的称vue.js为声明式渲染的模版引擎。

这也就是vue.js最大的优点，通过MVVM思想实现数据的双向绑定，让开发者不用再操作dom对象，有更多的时间去思考业务逻辑。

（4）组件化

在前端应用，我们是否也可以像编程一样把模块封装呢？这就引入了组件化开发的思想。

Vue.js通过组件，把一个单页应用中的各种模块拆分到一个一个单独的组件（component）中，我们只要先在父级应用中写好各种组件标签（占坑），并且在组件标签中写好要传入组件的参数（就像给函数传入参数一样，这个参数叫做组件的属性），然后再分别写好各种组件的实现（填坑），然后整个应用就算做完了。

（5）视图,数据,结构分离

使数据的更改更为简单,不需要进行逻辑代码的修改,只需要操作数据就能完成相关操作。

（6）虚拟DOM

现在的网速越来越快了，很多人家里都是几十甚至上百M的光纤，手机也是4G起步了，按道理一个网页才几百K，而且浏览器本身还会缓存很多资源文件，那么几十M的光纤为什么打开一个之前已经打开过，已经有缓存的页面还是感觉很慢呢？这就是因为浏览器本身处理DOM也是有性能瓶颈的，尤其是在传统开发中，用JQuery或者原生的JavaScript DOM操作函数对DOM进行频繁操作的时候，浏览器要不停的渲染新的DOM树，导致页面看起来非常卡顿。

而Virtual DOM则是虚拟DOM的英文，简单来说，他就是一种可以预先通过JavaScript进行各种计算，把最终的DOM操作计算出来并优化，由于这个DOM操作属于预处理操作，并没有真实的操作DOM，所以叫做虚拟DOM。最后在计算完毕才真正将DOM操作提交，将DOM操作变化反映到DOM树上。

（7）运行速度更快

像比较与react而言,同样都是操作虚拟dom,就性能而言,vue存在很大的优势。

**4. Vue的MVVM架构模式**

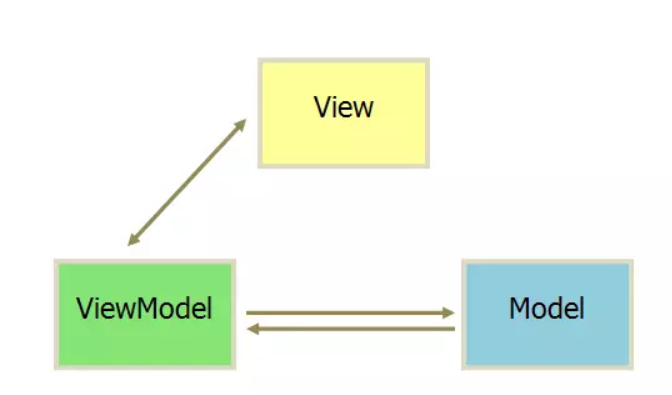
4.1 MVVM的简介

MVVM 由 Model,View,ViewModel 三部分构成，Model 层代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑；View 代表UI 组件，它负责将数据模型转化成UI 展现出来，ViewModel 是一个同步View 和 Model的对象。

在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。

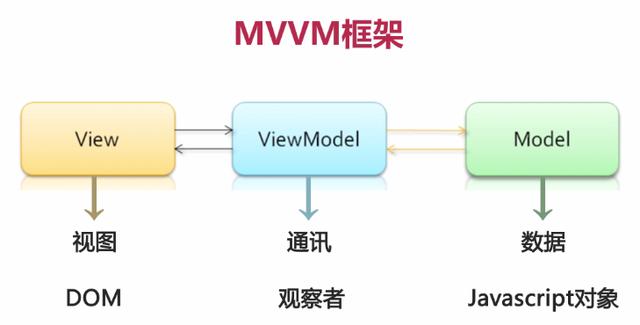
ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

4.2 MVVM模式图



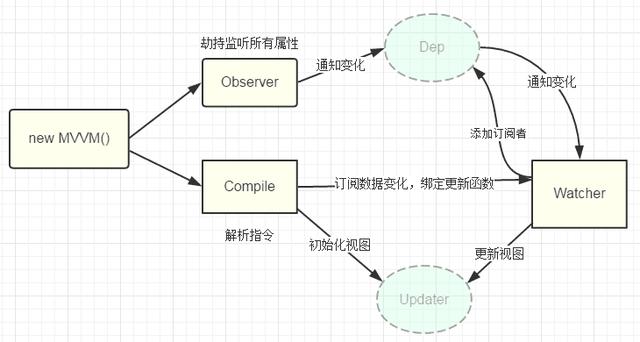
4.3 应用场景

* 针对具有复杂交互逻辑的前端应用
* 提供基础的架构抽象
* 提供ajax数据持久化，保证前端用户体验



4.4 MVVM框架数据分层情况

* Model 层代表数据模型，可以在Model中定义数据修改和操作的业务逻辑
* View 代表展示层，它负责将数据模型转化成UI 展现出来
* ViewModel 是一个同步View 和 Model的对象



图例介绍：

Observer 数据监听器，能够对数据对象的所有属性进行监听，如有变动可拿到最新值并通知订阅者，内部采用Object.defineProperty的getter和setter来实现。

Compile 指令解析器，它的作用对每个元素节点的指令进行扫描和解析，根据指令模板替换数据，以及绑定相应的更新函数。

Watcher 订阅者， 作为连接 Observer 和 Compile 的桥梁，能够订阅并收到每个属性变动的通知，执行指令绑定的相应回调函数。

Dep 消息订阅器，内部维护了一个数组，用来收集订阅者（Watcher），数据变动触发notify 函数，再调用订阅者的 update 方法。

当执行 new Vue() 时，Vue 就进入了初始化阶段，一方面Vue 会遍历 data 选项中的属性，并用 Object.defineProperty 将它们转为 getter/setter，实现数据变化监听功能；另一方面，Vue 的指令编译器Compile 对元素节点的指令进行扫描和解析，初始化视图，并订阅Watcher 来更新视图， 此时Wather 会将自己添加到消息订阅器中(Dep),初始化完毕。

当数据发生变化时，Observer 中的 setter 方法被触发，setter 会立即调用Dep.notify()，Dep 开始遍历所有的订阅者，并调用订阅者的 update 方法，订阅者收到通知后对视图进行相应的更新。

1. **Vue的目录结构**

├── build # 构建相关

├── static # 静态资源

│ │── img # 图片

│ │── favicon.ico # favicon图标

│ └── index.html # html模板

│ │── Tinymce # 富文本

├── src # 源代码

│ ├── api # 所有请求

│ ├── assets # 主题 字体等静态资源

│ ├── components # 全局公用组件

│ ├── directive # 全局指令

│ ├── icons # 项目所有 svg icons

│ ├── lang # 国际化 language

│ ├── layout # 全局 layout

│ ├── router # 路由

│ ├── store # 全局 store管理

│ ├── styles # 全局样式

│ ├── utils # 全局公用方法

│ ├── views # views 所有页面

│ ├── App.vue # 入口页面

│ ├── main.js # 入口文件 加载组件 初始化等

│ └── permission.js # 权限管理

├── tests # 测试

├── .env.xxx # 环境变量配置

├── .eslintrc.js # eslint 配置项

├── .gitignore # git 配置项

├── .babelrc # babel-loader 配置

├── .travis.yml # 自动化CI配置

├── vue.config.js # vue-cli 配置

└── package.json # package.json

此为常见的目录结构，在实际开发过程中，会因为项目的差异，目录有些小的改动。

### 