# vue培训

**一：MVVM (框架设计思想）**

1. 解析字母M、V、 VM的含义
2. 结合模型图理解MVVM模型
3. MVVM模型的优势（理论，后面结合vue框架来更好理解）
4. **vue组件**
5. 认识组件
6. 两个vue组件demo为例（过程中进一步说明MVVM模式）

（讲解demo前： 先介绍demo中用到的vue标签属性等）

1. 上面两个demo，用JQ实现一遍：从而对比他们优劣势

更进一步对比他们优势，

（1）普通的列表demo:

（2）使用vue实现项目中某一个功能，从代码量、复用性、移植成本（包括开发者的能力）



（3）拿现有项目的某一个复杂功能（车辆类型选择），触发大家的思考（不上代码，直接让大家思考如何封装组件）

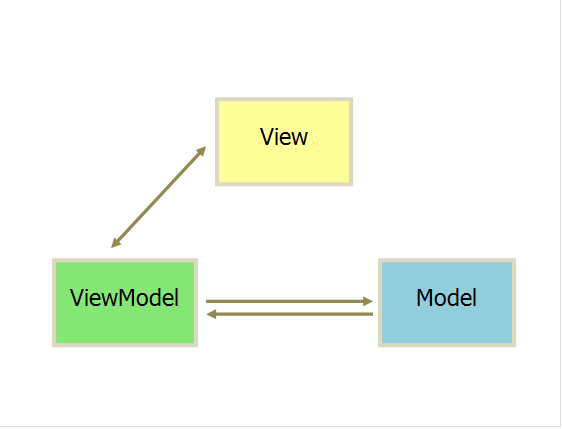
1. 引出：数据驱动代替DOM操作
2. **传统开发模式如何向工程化/模块化开发模式转型：规范化，自动化，模块化，组件化**
3. 目录结构规划，以及如此规划目录的好处

2、模块化/组件化开发

3、自动化

**一：MVVM（Model-View-ViewModel）**

1. **MVVM模型介绍**



**Model层：对应数据层的域模型,**

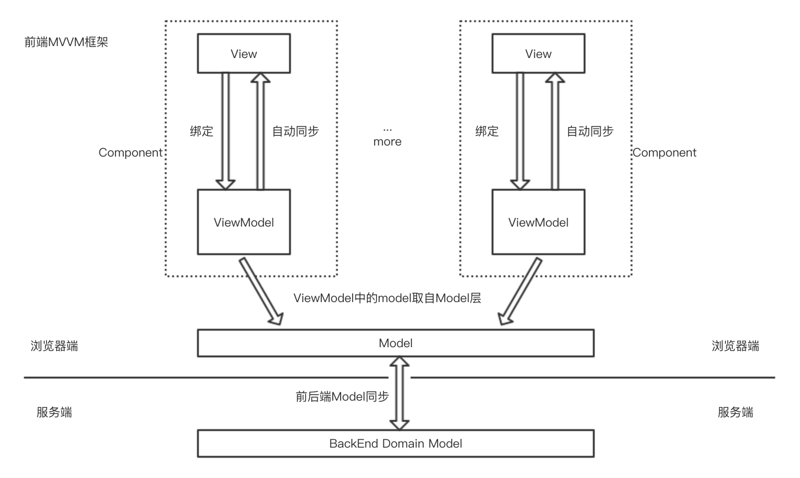
**View层：是视图模板层**。这一层除了定义结构、布局外，她展示的是ViewModel层的数据和状态。View层不负责处理状态，View层做的是数据绑定的声明、指令的声明、事件绑定的声明。

**ViewModel层：作为视图的模型，为视图服务**。连接了M层和V层。

ViewModel 层把 View 需要的层数据暴露，并对 View 层的 数据绑定声明、 指令声明、 事件绑定声明 负责，也就是处理 View 层的具体业务逻辑。ViewModel 底层会做好绑定属性的监听。当 ViewModel 中数据变化，View 层会得到更新；而当 View 中声明了数据的双向绑定（通常是表单元素），框架也会监听 View 层（表单）值的变化。一旦值变化，View 层绑定的 ViewModel 中的数据也会得到自动更新。

比如表单按钮点击提交，这些自定义事件的注册和处理逻辑都写在ViewModule里面了

1. **前端MVVM运作模式：（Viewmodel 和model之间的箭头似乎不太恰当）**



1. **MVVM模型的优势（具体优势将在下一节的组件使用中体现）**

开发人员只要考虑和处理Model（数据模型）的变化即可，不用考虑Model和View之间的数据绑定同步，更不用花精力用大量的代码获取DOM元素改变DOM元素的值来完成界面数据的变化，所有工作交给VM(ViewModel）来处理。

数据的双向绑定，开发人员不用再把精力放在DOM的修改和更新上，只要通过模板引擎将数据模型和用户界面绑定，框架会实时同步双方数据的变化。**减轻了开发人员的负担，也减少了DOM操作逻辑导致业务逻辑混乱的可能性。**

1. 低耦合。视图（View）可以独立于Model变化和修改，一个ViewModel可以绑定到不同的"View"上，当View变化的时候Model可以不变，当Model变化的时候View也可以不变。

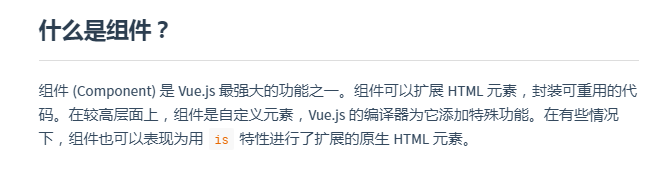
2. 可重用性。你可以把一些视图逻辑放在一个ViewModel里面，让很多view重用这段视图逻辑。

3. 独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计，使用Expression Blend可以很容易设计界面并生成xaml代码。

4. 可测试。界面素来是比较难于测试的，而现在测试可以针对ViewModel来写。

**二、vue组件**

1. **认识组件**
2. vue官方文档的解析:



通俗点理解就是： vue组件就好比我们UI框架里的JS框架，以“提示框”组件为例，提示框有六种不同类型，他们除了提示框背景色不一样外，其他都一模一样。这时候我们就可以把他们封装起来，提高复用率。它好比一个公用方法，我们并不需要了解方法体内的具体实现，只需要传入恰当的参数便可以。

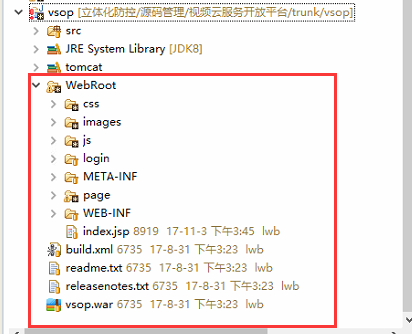
1. Vue组件demo
2. **传统开发模式如何向工程化/模块化开发模式转型：规范化，自动化，模块化，组件化**
3. 目录结构规划，以及如此规划目录的好处

挑几个目录中比较重要的配置文件进行讲解

顺带介绍：webpack、反向代理（跨域）、入口文件。。。。

（https://segmentfault.com/a/1190000010345844）

传统的开发目录：



现在的目录：

├── index.html 入口页面

├── build 构建脚本目录

│ ├── build-server.js 运行本地构建服务器，可以访问构建后的页面

│ ├── build.js 生产环境构建脚本

│ ├── dev-client.js 开发服务器热重载脚本，主要用来实现开发阶段的页面自动刷新

│ ├── dev-server.js 运行本地开发服务器

│ ├── utils.js 构建相关工具方法

│ ├── webpack.base.conf.js wabpack基础配置

│ ├── webpack.dev.conf.js wabpack开发环境配置

│ └── webpack.prod.conf.js wabpack生产环境配置

├── config 项目配置

│ ├── dev.env.js 开发环境变量

│ ├── index.js 项目配置文件

│ ├── prod.env.js 生产环境变量

│ └── test.env.js 测试环境变量

├── mock mock数据目录

│ └── hello.js

├── package.json  npm包配置文件，里面定义了项目的npm脚本，依赖包等信息

├── src 项目源码目录

│ ├── main.js 入口js文件

│ ├── app.vue 根组件

│ ├── components 公共组件目录

│ │ └── title.vue

│ ├── assets 资源目录，这里的资源会被wabpack构建

│ │ └── images

│ │ └── logo.png

│ ├── routes 前端路由

│ │ └── index.js

│ ├── store 应用级数据（state）

│ │ └── index.js

│ └── views 页面目录

│ ├── hello.vue

│ └── notfound.vue

├── static 纯静态资源，不会被wabpack构建。

└── test 测试文件目录（unit&e2e）

└── unit 单元测试

├── index.js 入口脚本

├── karma.conf.js karma配置文件

└── specs 单测case目录

└── Hello.spec.js

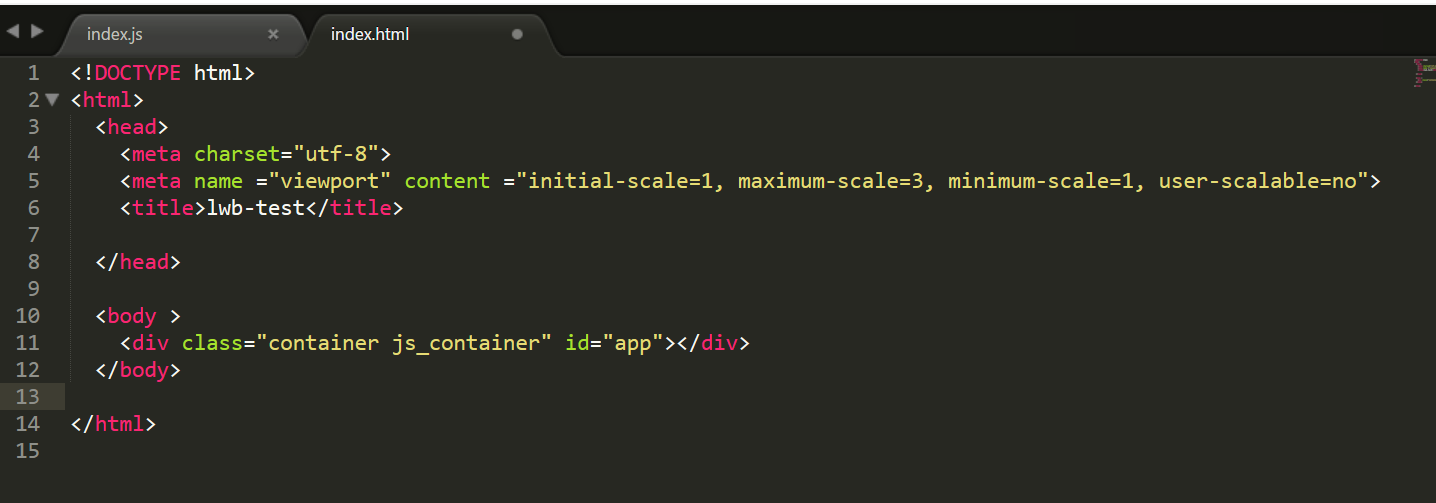
1. 传统的开发认为：将静态资源文件（html、css、js、图片）等，按照文件类型进行目录规划，会更方便管理。这种划分方式简单粗暴。

现在的目录： 目录主要是根据功能和所扮演的角色进行划分，弱化了html、js、css的分离，（在react里，css和html都是使用js来编写）此时，通过目录可以看出，现在前端更偏向工程化，能做的事情也更多了。

1. 主要介绍几个关键的目录、文件（先从熟悉的开始）
2. package.json: npm包配置文件，在这个文件我们可以看见项目中引入了那些依赖及其版本号以及npm的脚本（scripts对象下）。该配置在项目初始化自动生成，且在我们使用npm进行插件依赖的下载时候，package.json也会同步更新。建议，我们不要手动操作package.json文件。

2）index.html:

由于vue是单页面应用，所以他只有一个html文件。（我们传统开发都是多页面应用）发布版本的index.html，webpack会将压缩后的css、js都引入到html，这时候便符合我们正常开发思维。



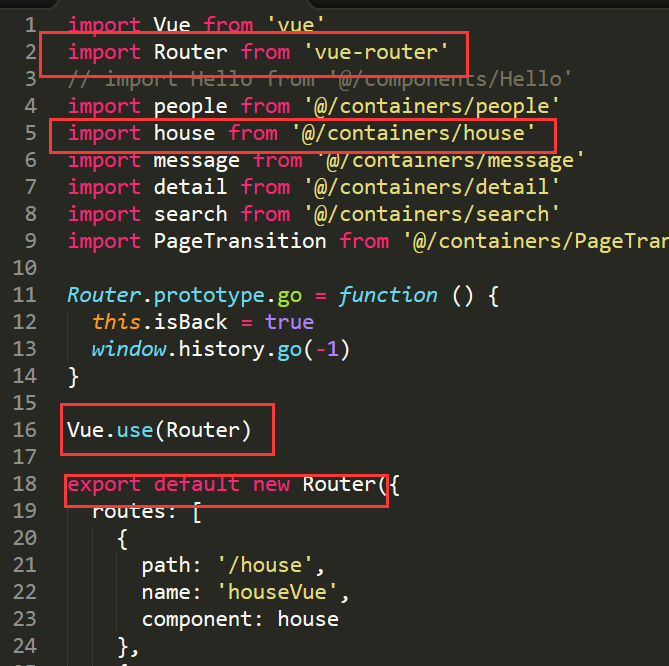
3)src： 生产环境下的静态资源目录。它相当于传统目录下的WebRoot

Main.js: 项目的入口文件，在这个文件中，可以引入全局资源以及定义全局定函数等，这样在项目中的各个组件都可以使用到。这也是webpack打包的入口文件

Router: 路由的配置文件（路由使用可看vue-router官方文档）

要使用路由，首先使用ES6的语法，import进来vue-router插件，并通过Vue.use()来使用插件。若想导入一个已经写好的house组件，首先也是要使用import引入该vue文件，然后通过配置路由的path（也就是在浏览器访问的路径），并将house赋值给component。此时，我们可以在浏览器上通过路径： localhost:端口号/

不再使用iframe嵌套页面，iframe不好访问外部数据，页面多的时候不容易管理



1. mode\_modules: 项目所需的依赖包，使用npm install下载的依赖都将在这个文件夹里保存。可以粗暴理解为，里面保存了我们项目所需的插件。
2. Build: 运行环境的配置文件夹。以压缩html + 2JS文件为例，简单讲解压缩打包原理
3. dist文件是最总发布的前端资源。这就相当于我们传统开发，部署时候我们前端只要提供webroot下的资源文件便可，而这里我们只需要提供dist文件。此外，node-modules下的文件也npm run build发布命令时候，也被打包进来。可能大家会疑惑node-modules文件那么大，最后打包出来的内容会不会很大。其实并不会，就比如项目中用到echarts插件，echarts的图标类型很多，但是在打包过程中我们只会打包我们项目中引用到的文件，没引用到的最终都是不会被打包进去的。
4. Config： 配置文件，主要讲下和代理相关的index.js文件。（详细讲解）