前端体系

想要成为真正的“互联网 Java 全栈工程师”还有很长的一段路要走，其中“我大前端”是绕不开的一门必修课。

本阶段课程的主要目的就是带领我 Java 后台程序员认识前端、了解前端、掌握前端，为实现成为 “互联网 Java 全栈工程师” 再向前迈进一步。

前端三要素

> 前端三要素

- HTML（结构）：超文本标记语言（Hyper Text Markup Language），决定网页的结构和内容
- CSS（表现）：层叠样式表（Cascading Style Sheets），设定网页的表现样式
- JavaScript（行为）：是一种弱类型脚本语言，其源代码不需经过编译，而是由浏览器解释运行，用于控制网页的行为

> 结构层（HTML）

太简单，略

> 表现层（CSS）

CSS 层叠样式表是一门标记语言，并不是编程语言，因此不可以自定义变量，不可以引用等，换句话说就是不具备任何语法支持，它主要缺陷如下：

- 语法不够强大，比如无法嵌套书写，导致模块化开发中需要书写很多重复的选择器；
- 没有变量和合理的样式复用机制，使得逻辑上相关的属性值必须以字面量的形式重复输出，导致难以维护；

这就导致了我们在工作中无端增加了许多工作量。为了解决这个问题，前端开发人员会使用一种称之为【CSS 预处理器】 的工具，提供 CSS 缺失的样式层复用机制、减少冗余代码，提高样式代码的可维护性。大大提高了前端在样式上的开发效率。

什么是 CSS 预处理器?

CSS 预处理器定义了一种新的语言，其基本思想是，用一种专门的编程语言，为 CSS 增加了一些编程的特性，将 CSS 作为目标生成文件，然后开发者就只要使用这种语言进行 CSS 的编码工作。转化成通俗易懂的话来说就是“用一种专门的编程语言，进行 Web 页面样式设计，再通过编译器转化为正常的 CSS 文件，以供项目使用”。

常用的 CSS 预处理器有哪些:

- SASS：基于 Ruby，通过服务端处理，功能强大。解析效率高。需要学习 Ruby 语言，上手难度高于 LESS。
- LESS：基于 NodeJS，通过客户端处理，使用简单。功能比 SASS 简单，解析效率也低于 SASS，但在实际开发中足够了，所以我们后台人员如果需要的话，建议使用 LESS。

> 行为层（JavaScript）

JavaScript 一门弱类型脚本语言，其源代码在发往客户端运行之前不需经过编译，而是将文本格式的字符代码发送给浏览器由浏览器解释运行。

Native 原生 JS 开发

原生 JS 开发，也就是让我们按照 **【ECMAScript】** 标准的开发方式，简称是 ES，特点是所有浏览器都支持。ES 标准已发布如下版本：

- ES3
- ES4（内部，未正式发布）
- ES5（全浏览器支持）
- ES6（常用，当前主流版本：webpack打包成为ES5支持！）
- ES7
- ES8
- ES9（草案阶段）

从 ES6 开始每年发布一个版本，以年份作为名称，区别就是逐步增加新特性。

TypeScript 微软的标准

TypeScript 是一种由微软开发的自由和开源的编程语言。它是 JavaScript 的一个超集，而且本质上向这个语言添加了可选的静态类型和基于类的面向对象编程。由安德斯·海尔斯伯格（C#、Delphi、TypeScript 之父；.NET 创立者）主导。

JavaScript 框架

> jQuery库

大家最熟知的 JavaScript 库，优点是简化了 DOM 操作，缺点是 DOM 操作太频繁，影响前端性能；在前端眼里使用它仅仅是为了兼容 IE6、7、8；

> Angular

Google 收购的前端框架，由一群 Java 程序员开发，其特点是将后台的 MVC 模式搬到了前端并增加了**模块化开发**的理念，与微软合作，采用 TypeScript 语法开发；对后台程序员友好，对前端程序员不太友好；最大的缺点是版本迭代不合理（如：1代 -> 2代，除了名字，基本就是两个东西；已推出了 Angular6）

> React

Facebook 出品，一款高性能的 JS 前端框架；特点是提出了新概念 **【虚拟 DOM】** 用于减少真实 DOM 操作，在内存中模拟 DOM 操作，有效的提升了前端渲染效率；缺点是使用复杂，因为需要额外学习一门 **【JSX】** 语言；

> Vue

一款渐进式 JavaScript 框架，所谓渐进式就是逐步实现新特性的意思，如实现模块化开发、路由、状态管理等新特性。其特点是综合了 Angular（模块化） 和 React（虚拟 DOM） 的优点；

> Axios

前端通信框架；因为 Vue 的边界很明确，就是为了处理 DOM，所以并不具备通信能力，此时就需要额外使用一个通信框架与服务器交互；当然也可以直接选择使用 jQuery 提供的 AJAX 通信功能；

JavaScript 构建工具

- Babel：JS 编译工具，主要用于浏览器不支持的 ES 新特性，比如用于编译 TypeScript
- WebPack：模块打包器，主要作用是打包、压缩、合并及按序加载

前端所需后端技术

前端人员为了方便开发也需要掌握一定的后端技术，但我们 Java 后台人员知道后台知识体系极其庞大复杂，所以为了方便前端人员开发后台应用，就出现了 NodeJS 这样的技术。

NodeJS 的作者已经声称放弃 NodeJS（说是架构做的不好再加上笨重的 node_modules，可能让作者不爽了吧），开始开发全新架构的 Deno。

既然是后台技术，那肯定也需要框架和项目管理工具，NodeJS 框架及项目管理工具如下：

- Express：NodeJS 框架
- Koa：Express 简化版
- NPM：项目综合管理工具，类似于 Maven
- YARN：NPM 的替代方案，类似于 Maven 和 Gradle 的关系

UI 框架

> 常用

- Ant-Design：阿里巴巴出品，基于 React 的 UI 框架
- ElementUI、iview、ice：饿了么出品，基于 Vue 的 UI 框架
- Bootstrap：Twitter 推出的一个用于前端开发的开源工具包
- AmazeUI：又叫“妹子 UI”，一款 HTML5 跨屏前端框架
- Layui：轻量级框架

> iView

iview 是一个强大的基于 Vue 的 UI 库，有很多实用的基础组件比 elementui 的组件更丰富，主要服务于 PC 界面的中后台产品。使用单文件的 Vue 组件化开发模式 基于 npm + webpack + babel 开发，支持 ES2015 高质量、功能丰富 友好的 API ，自由灵活地使用空间。

- [官网地址] https://www.iviewui.com/
- [Github] https://github.com/TalkingData/iview-weapp
- [iview-admin] https://github.com/iview/iview-admin

备注：属于前端主流框架，选型时可考虑使用，主要特点是移动端支持较多

> ElementUI

Element 是饿了么前端开源维护的 Vue UI 组件库，组件齐全，基本涵盖后台所需的所有组件，文档讲解详细，例子也很丰富。主要用于开发 PC 端的页面，是一个质量比较高的 Vue UI 组件库。

- [官网地址] http://element-cn.eleme.io/#/zh-CN

- [Github] https://github.com/ElementUI/element-starter
- [vue-element-admin] https://github.com/PanJiaChen/vue-element-admin

备注：属于前端主流框架，选型时可考虑使用，主要特点是桌面端支持较多

> ICE

飞冰是阿里巴巴团队基于 React/Angular/Vue 的中后台应用解决方案，在阿里巴巴内部，已经有 270 多个来自几乎所有 BU 的项目在使用。飞冰包含了一条从设计端到开发端的完整链路，帮助用户快速搭建属于自己的中后台应用。

- [官网地址] https://alibaba.github.io/ice
- [Github] https://github.com/alibaba/ice

备注：主要组件还是以 React 为主，截止 2019 年 02 月 17 日更新博客前对 Vue 的支持还不太完善，目前尚处于观望阶段

> VantUI

Vant UI 是有赞前端团队基于有赞统一的规范实现的 Vue 组件库，提供了一整套 UI 基础组件和业务组件。通过 Vant，可以快速搭建出风格统一的页面，提升开发效率。

- [官网地址] https://youzan.github.io/vant/#/zh-CN/intro
- [Github] https://github.com/youzan/vant

> AtUI

at-ui 是一款基于 Vue 2.x 的前端 UI 组件库，主要用于快速开发 PC 网站产品。 它提供了一套 npm + webpack + babel 前端开发工作流程，CSS 样式独立，即使采用不同的框架实现都能保持统一的 UI 风格。

的 UI 风格。

- [官网地址] https://at-ui.github.io/at-ui/#/zh
- [Github] https://github.com/at-ui/at-ui

> CubeUI

cube-ui 是滴滴团队开发的基于 Vue.js 实现的精致移动端组件库。支持按需引入和后编译，轻量灵活；扩展性强，可以方便地基于现有组件实现二次开发。

- [官网地址] https://didi.github.io/cube-ui/#/zh-CN
- [Github] https://github.com/didi/cube-ui/

> Flutter

Flutter 是谷歌的移动端 UI 框架，可在极短的时间内构建 Android 和 iOS 上高质量的原生级应用。Flutter 可与现有代码一起工作, 它被世界各地的开发者和组织使用, 并且 Flutter 是免费和开源的。

- [官网地址] https://flutter.dev/docs
- [Github] https://github.com/flutter/flutter

备注：Google 出品，主要特点是快速构建原生 APP 应用程序，如做混合应用该框架为必选框架

> Ionic

Ionic 既是一个 CSS 框架也是一个 Javascript UI 库，Ionic 是目前最有潜力的一款 HTML5 手机应用开发框架。通过 SASS 构建应用程序，它提供了很多 UI 组件来帮助开发者开发强大的应用。它使用 JavaScript MVVM 框架和 AngularJS/Vue 来增强应用。提供数据的双向绑定，使用它成为 Web 和移动开发者的共同选择。

- [官网地址] https://ionicframework.com/
- [官网文档] https://ionicframework.com/docs/
- [Github] https://github.com/ionic-team/ionic

> mpvue

mpvue 是美团开发的一个使用 Vue.js 开发小程序的前端框架，目前支持 **微信小程序、百度智能小程序，头条小程序** 和 **支付宝小程序。** 框架基于 Vue.js，修改了的运行时框架 runtime 和代码编译器 compiler 实现，使其可运行在小程序环境中，从而为小程序开发引入了 Vue.js 开发体验。

- [官网地址] http://mpvue.com/
- [Github] https://github.com/Meituan-Dianping/mpvue

备注：完备的 Vue 开发体验，并且支持多平台的小程序开发，推荐使用

> WeUI

WeUI 是一套同微信原生视觉体验一致的基础样式库，由微信官方设计团队为微信内网页和微信小程序量身设计，令用户的使用感知更加统一。包含 button、cell、dialog、toast、article、icon 等各式元素。

- [官网地址] https://weui.io/
- [Github] https://github.com/weui/weui.git

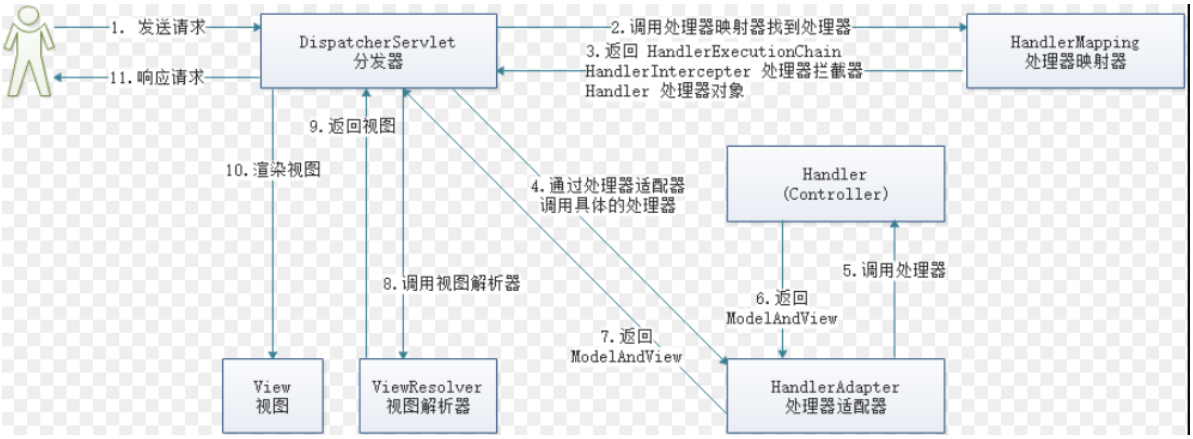
注：以上知识点已将 WebApp 开发所需技能全部梳理完毕

前后分离的演变史

> 后端为主的 MVC 时代

为了降低开发的复杂度，以后端为出发点，比如：Struts、SpringMVC 等框架的使用，就是后端的 MVC 时代;

以 `SpringMVC` 流程为例：



- 发起请求到前端控制器(`DispatcherServlet`)
- 前端控制器请求 `HandlerMapping` 查找 `Handler`，可以根据 `xml` 配置、注解进行查找
- 处理器映射器 `HandlerMapping` 向前端控制器返回 `Handler`
- 前端控制器调用处理器适配器去执行 `Handler`
- 处理器适配器去执行 `Handler`
- `Handler` 执行完成给适配器返回 `ModelAndView`
- 处理器适配器向前端控制器返回 `ModelAndView`，`ModelAndView` 是 `SpringMVC` 框架的一个底层对象，包括 `Model` 和 `View`
- 前端控制器请求视图解析器去进行视图解析，根据逻辑视图名解析成真正的视图(`JSP`)
- 视图解析器向前端控制器返回 `View`
- 前端控制器进行视图渲染，视图渲染将模型数据(在 `ModelAndView` 对象中)填充到 `request` 域
- 前端控制器向用户响应结果

优点：

MVC 是一个非常好的协作模式，能够有效降低代码的耦合度，从架构上能够让开发者明白代码应该写在哪里。为了让 View 更纯粹，还可以使用 Thymeleaf、Freemarker 等模板引擎，使模板里无法写入 Java 代码，让前后端分工更加清晰。

缺点：

前端开发重度依赖开发环境，开发效率低，这种架构下，前后端协作有两种模式：

1、第一种是前端写 DEMO，写好后，让后端去套模板。好处是 DEMO 可以本地开发，很高效。不足是还需要后端套模板，有可能套错，套完后还需要前端确定，来回沟通调整的成本比较大；

2、另一种协作模式是前端负责浏览器端的所有开发和服务器端的 View 层模板开发。好处是 UI 相关的代码都是前端去写就好，后端不用太关注，不足就是前端开发重度绑定后端环境，环境成为影响前端开发效率的重要因素。

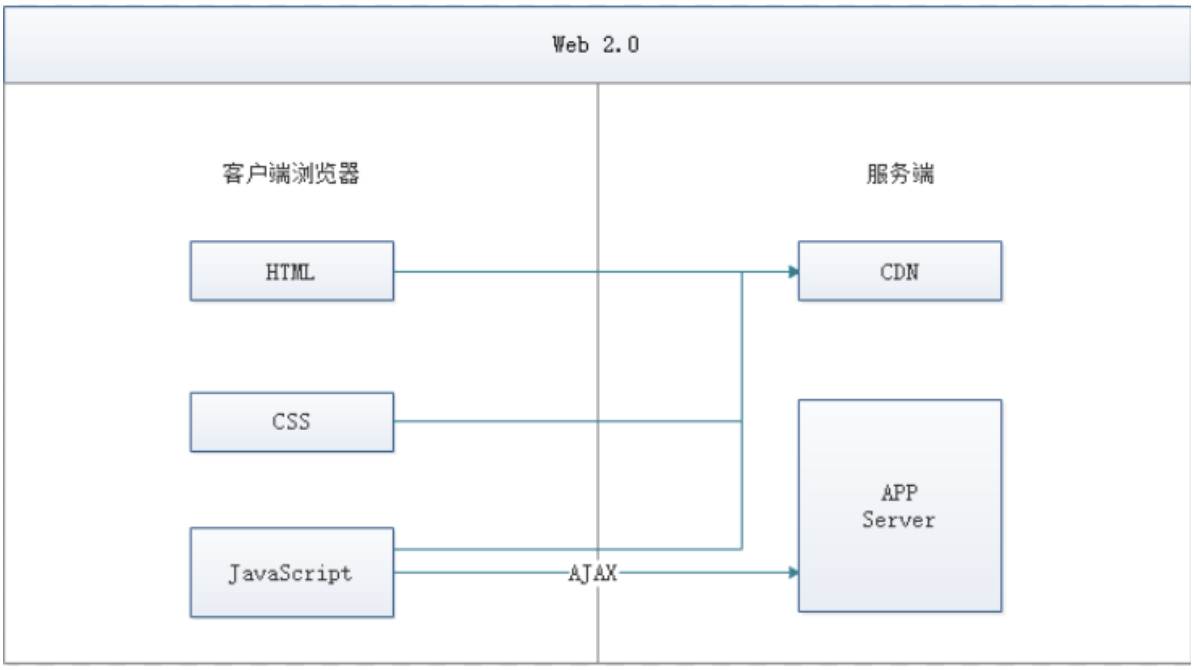
前后端职责纠缠不清：模板引擎功能强大，依旧可以通过拿到的上下文变量来实现各种业务逻辑。这样，只要前端弱势一点，往往就会被后端要求在模板层写出不少业务代码。还有一个很大的灰色地带是 `Controller`，页面路由等功能本应该是前端最关注的，但却是由后端来实现。`Controller` 本身与 `Model` 往往也会纠缠不清，看了让人咬牙的业务代码经常会出现在 `Controller` 层。这些问题不能全归结于程序员的素养，否则 JSP 就够了。

对前端发挥的局限性：性能优化如果只在前端做空间非常有限，于是我们经常需要后端合作；

注：在这期间（2005 年以前），包括早期的 JSP、PHP 可以称之为 Web 1.0 时代。在这里想说一句，如果你是一名 Java 初学者，请你不要再把一些陈旧的技术当回事了，比如 JSP，因为时代在变、技术在变、什么都在变（引用扎克伯格的一句话：唯一不变的是变化本身）；当我们去给大学做实训时，有些同学会认为我们没有讲什么 干货，其实不然，只能说是你认知里的干货对于市场来说早就过时了而已。

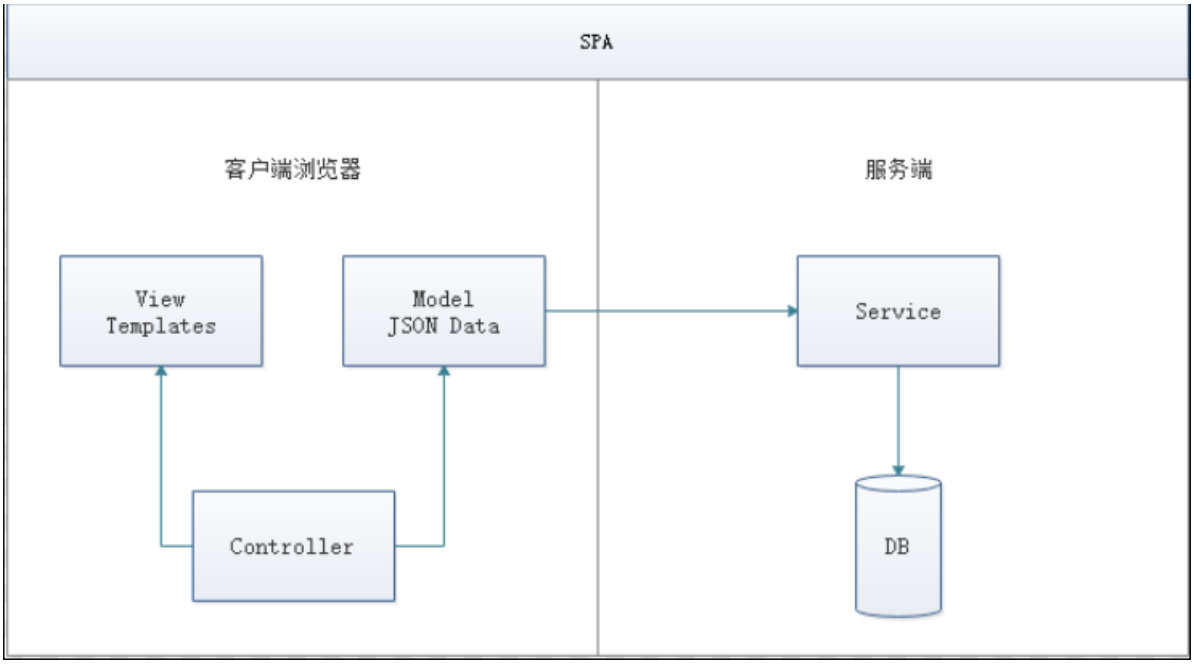
> 基于 AJAX 带来的 SPA 时代

时间回到 2005 年 AJAX（Asynchronous JavaScript And XML，异步 JavaScript 和 XML，老技术新用法） 被正式提出并开始使用 CDN 作为静态资源存储，于是出现了 JavaScript 王者归来（在这之前 JS 都是用来在网页上贴狗皮膏药广告的）的 SPA（Single Page Application）单页面应用时代。



优点：

这种模式下，**前后端的分工非常清晰，前后端的关键协作点是 AJAX 接口。**看起来是如此美妙，但回过头来看看的话，这与 JSP 时代区别不大。复杂度从服务端的 JSP 里移到了浏览器的 JavaScript，浏览器端变得很复杂。类似 Spring MVC，**这个时代开始出现浏览器端的分层架构**：



缺点：

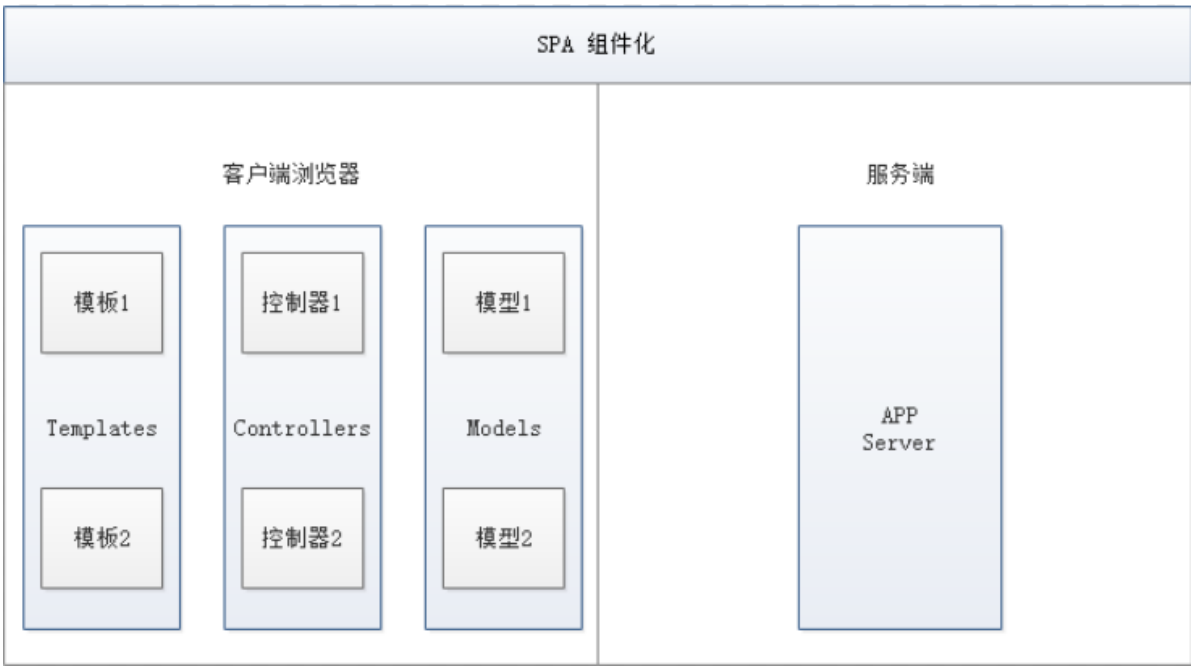
- **前后端接口的约定：** 如果后端的接口一塌糊涂，如果后端的业务模型不够稳定，那么前端开发会很痛苦；不少团队也有类似尝试，通过接口规则、接口平台等方式来做。**有了和后端一起沉淀的 接口规则，还可以用来模拟数据，使得前后端可以在约定接口后实现高效并行开发。**
- **前端开发的复杂度控制：** SPA 应用大多以功能交互型为主，JavaScript 代码过十万行很正常。大量 JS 代码的组织，与 View 层的绑定等，都不是容易的事情。

> 前端为主的 MV* 时代

此处的 MV* 模式如下：

- MVC（同步通信为主）：Model、View、Controller
- MVP（异步通信为主）：Model、View、Presenter
- MVVM（异步通信为主）：Model、View、ViewModel

为了降低前端开发复杂度，涌现了大量的前端框架，比如：`AngularJS`、`React`、`Vue.js`、`EmberJS`等，这些框架总的原则是先按类型分层，比如 Templates、Controllers、Models，然后再在层内做切分，如下图：



优点：

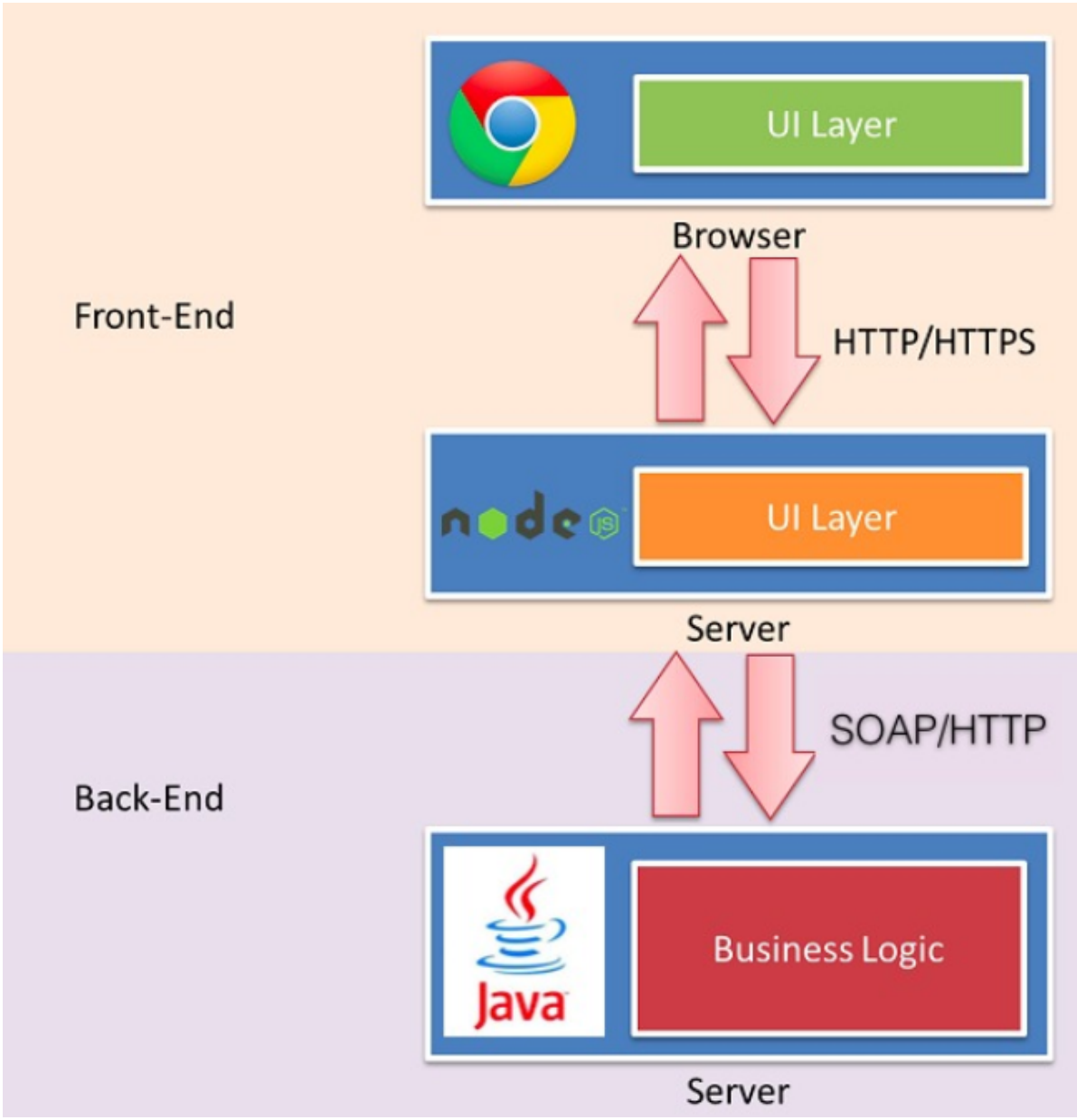
- **前后端职责很清晰：** 前端工作在浏览器端，后端工作在服务端。清晰的分工，可以让开发并行，测试数据的模拟不难，前端可以本地开发。后端则可以专注于业务逻辑的处理，输出 RESTful等接口。
- **前端开发的复杂度可控：** 前端代码很重，但合理的分层，让前端代码能各司其职。这一块蛮有意思的，简单如模板特性的选择，就有很多很多讲究。并非越强大越好，限制什么，留下哪些自由，代码应该如何组织，所有这一切设计，得花一本书的厚度去说明。
- **部署相对独立：** 可以快速改进产品体验

缺点：

- 代码不能复用。比如后端依旧需要对数据做各种校验，校验逻辑无法复用浏览器端的代码。如果可以复用，那么后端的数据校验可以相对简单化。
- 全异步，对 SEO 不利。往往还需要服务端做同步渲染的降级方案。
- 性能并非最佳，特别是移动互联网环境下。
- SPA 不能满足所有需求，依旧存在大量多页面应用。URL Design 需要后端配合，前端无法完全掌控。

> NodeJS 带来的全栈时代

前端为主的 MV* 模式解决了很多很多问题，但如上所述，依旧存在不少不足之处。随着 NodeJS 的兴起，JavaScript 开始有能力运行在服务端。这意味着可以有一种新的研发模式：



在这种研发模式下，前后端的职责很清晰。对前端来说，两个 UI 层各司其职：

- Front-end UI layer 处理浏览器层的展现逻辑。通过 CSS 渲染样式，通过 JavaScript 添加交互功能，HTML 的生成也可以放在这层，具体看应用场景。
- Back-end UI layer 处理路由、模板、数据获取、Cookie 等。通过路由，前端终于可以自主把控 URL Design，这样无论是单页面应用还是多页面应用，前端都可以自由调控。后端也终于可以摆脱对展现的强关注，转而可以专心于业务逻辑层的开发。

通过 Node，Web Server 层也是 JavaScript 代码，这意味着部分代码可前后复用，需要 SEO 的场景可以在服务端同步渲染，由于异步请求太多导致的性能问题也可以通过服务端来缓解。前一种模式的不足，通过这种模式几乎都能完美解决掉。

与 JSP 模式相比，全栈模式看起来是一种回归，也的确是一种向原始开发模式的回归，不过是一种螺旋上升式的回归。

基于 NodeJS 的全栈模式，依旧面临很多挑战：

- 需要前端对服务端编程有更进一步的认识。比如 TCP/IP 等网络知识的掌握。
- NodeJS 层与 Java 层的高效通信。NodeJS 模式下，都在服务器端，RESTful HTTP 通信未必高效，通过 SOAP 等方式通信更高效。一切需要在验证中前行。
- 对部署、运维层面的熟练了解，需要更多知识点和实操经验。
- 大量历史遗留问题如何过渡。这可能是最大最大的阻力。

看到这里，相信很多同学就可以理解，为什么我总在课堂上说：“前端想学后台很难，而我们后端程序员学任何东西都很简单”；就是因为我们后端程序员具备相对完善的知识体系。

全栈！So Easy！ヽ(￣▽￣)ﾉ!

> 小结

综上所述，模式也好，技术也罢，没有好坏优劣之分，只有适合不适合；前后分离的开发思想主要是基于 `SoC`（关注度分离原则），上面种种模式，都是让前后端的职责更清晰，分工更合理高效。

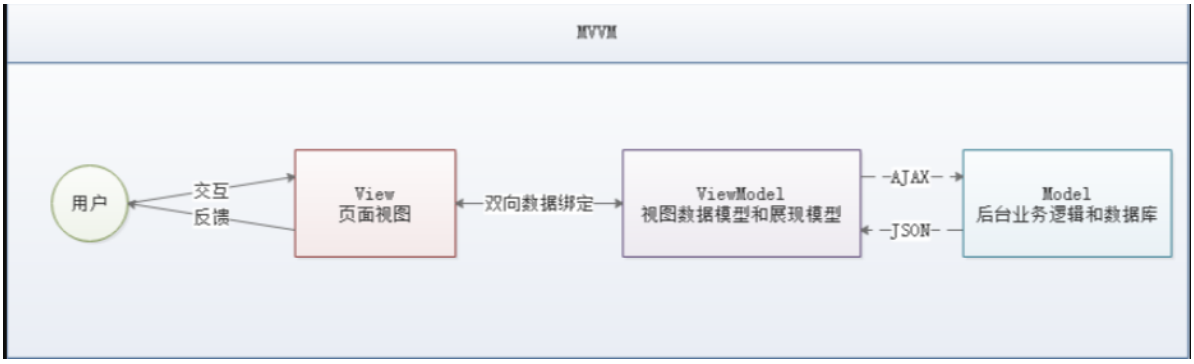
MVVM模式

> 什么是MVVM模式?

MVVM（Model-View-ViewModel）是一种软件架构设计模式，由微软 WPF（用于替代 WinForm，以前就是用这个技术开发桌面应用程序的）和 Silverlight（类似于 Java Applet，简单点说就是在浏览器上运行的 WPF） 的架构师 Ken Cooper 和 Ted Peters 开发，是一种简化用户界面的**事件驱动编程方式。**由 John Gossman（同样也是 WPF 和 Silverlight 的架构师）于 2005 年在他的博客上发表。

MVVM 源自于经典的 MVC（Model-View-Controller）模式。MVVM 的核心是 ViewModel 层，负责转换 Model 中的数据对象来让数据变得更容易管理和使用，其作用如下：

- 该层向上与视图层进行双向数据绑定
- 向下与 Model 层通过接口请求进行数据交互



MVVM 已经相当成熟了，当下流行的 MVVM 框架有 `Vue.js`，`AngularJS` 等。

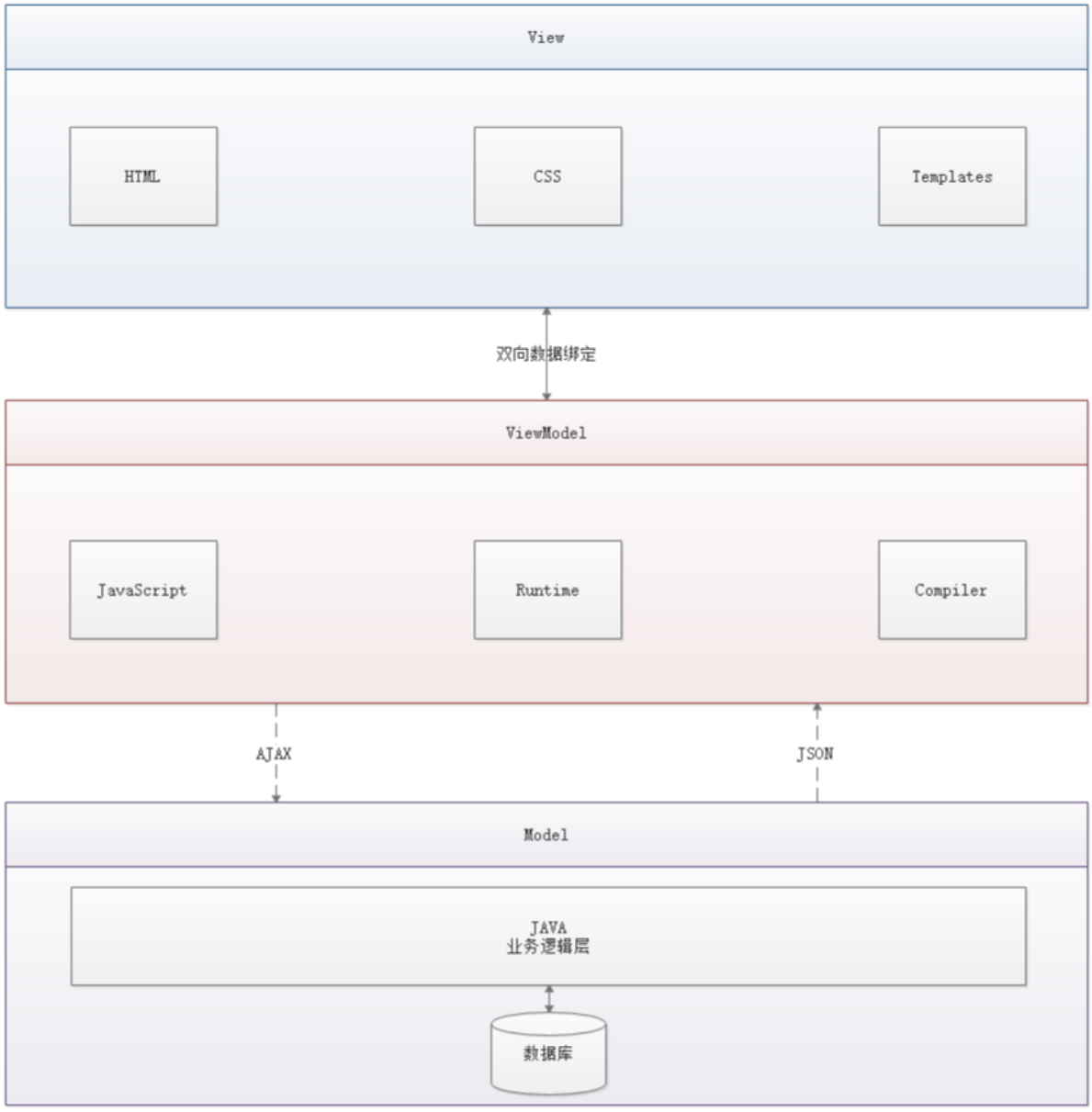
> 为什么要使用 MVVM

MVVM 模式和 MVC 模式一样，主要目的是分离视图（View）和模型（Model），有几大好处：

- **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。
- **可复用**： 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。

- **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。
- **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。

> MVVM 的组成部分



View：

View 是视图层，也就是用户界面。前端主要由 HTML 和 CSS 来构建，为了更方便地展现 ViewModel 或者 Model层的数据，已经产生了各种各样的前后端模板语言，比如 FreeMarker、Thymeleaf 等等，各大 MVVM 框架如 Vue.js，AngularJS，EJS 等也都有自己用来构建用户界面的内置模板语言。

Model：

Model 是指数据模型，泛指后端进行的各种业务逻辑处理和数据操控，主要围绕数据库系统展开。这里的难点主要在于需要和前端约定统一的 接口规则

ViewModel：

ViewModel 是由前端开发人员组织生成和维护的视图数据层。在这一层，前端开发者对从后端获取的 Model 数据进行转换处理，做二次封装，以生成符合 View 层使用预期的视图数据模型。

需要注意的是 ViewModel 所封装出来的数据模型包括视图的状态和行为两部分，而 Model 层的数据模型是只包含状态的。

- 比如页面的这一块展示什么，那一块展示什么这些都属于视图状态（展示）
- 页面加载进来时发生什么，点击这一块发生什么，这一块滚动时发生什么这些都属于视图行为（交互）

视图状态和行为都封装在了 ViewModel 里。这样的封装使得 ViewModel 可以完整地去描述 View 层。由于实现了双向绑定，ViewModel 的内容会实时展现在 View 层，这是激动人心的，因为前端开发者再也不必低效又麻烦地通过操纵 DOM 去更新视图。

MVVM 框架已经把最脏最累的一块做好了，我们开发者只需要处理和维护 ViewModel，更新数据视图就会自动得到相应更新，真正实现 `事件驱动编程`。

View 层展现的不是 `Model` 层的数据，而是 `ViewModel` 的数据，由 `ViewModel` 负责与 `Model` 层交互，这就**完全解耦了 View 层和 Model 层，这个解耦是至关重要的，它是前后端分离方案实施的重要一环。**

第一个Vue程序

> 简介

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式框架，发布于 2014 年 2 月。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库（如：vue-router，vue-resource，vuex）或既有项目整合。

> MVVM 模式的实现者

- Model：模型层，在这里表示 JavaScript 对象
- View：视图层，在这里表示 DOM（HTML 操作的元素）
- ViewModel：连接视图和数据的中间件，Vue.js 就是 MVVM 中的 ViewModel 层的实现者

在 MVVM 架构中，是不允许 数据 和 视图 直接通信的，只能通过 ViewModel 来通信，而 ViewModel 就是定义了一个 Observer 观察者。

- ViewModel 能够观察到数据的变化，并对视图对应的内容进行更新
- ViewModel 能够监听到视图的变化，并能够通知数据发生改变

至此，我们就明白了，Vue.js 就是一个 MVVM 的实现者，他的核心就是实现了 DOM 监听 与 数据绑定

> 为什么要使用 Vue.js

- 轻量级，体积小是一个重要指标。Vue.js 压缩后有只有 20多kb （Angular 压缩后 56kb+，React 压缩后 44kb+）
- 移动优先。更适合移动端，比如移动端的 Touch 事件
- 易上手，学习曲线平稳，文档齐全
- 吸取了 Angular（模块化）和 React（虚拟 DOM）的长处，并拥有自己独特的功能，如：计算属性
- 开源，社区活跃度高
- .....

> 第一个Vue程序准备

注意：Vue 不支持 IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟的 ECMAScript 5 特性。但它支持所有兼容 ECMAScript 5 的浏览器。

开发版本

- 包含完整的警告和调试模式：https://vuejs.org/js/vue.js
- 删除了警告，30.96KB min + gzip：https://vuejs.org/js/vue.min.js

CDN

- `<script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>`
- `<script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.min.js"></script>`

> Hello，Vue

Vue.js 的核心是实现了 MVVM 模式，她扮演的角色就是 ViewModel 层，那么所谓的第一个应用程序就是展示她的 数据绑定 功能，操作流程如下：

1、创建一个 HTML 文件 01-hello.html

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>狂神说</title>
</head>
<body>

</body>
</html>
```

2、引入 Vue.js

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>
```

3、创建一个 Vue 的实例

```
<script type="text/javascript">
    var vm = new Vue({
        el: '#vue',
        data: {
            message: 'Hello Vue!'
        }
    });
</script>
```

说明：

- `el:'#vue'`：绑定元素的 ID
- `data:{message:'Hello Vue!'}`：数据对象中有一个名为 message 的属性，并设置了初始值 Hello Vue!

4、将数据绑定到页面元素（视图层）

```
<div id="vue">
    {{message}}
</div>
```

说明：只需要在绑定的元素中使用 双花括号 将 Vue 创建的名为 message 属性包裹起来，即可实现数据绑定功能，也就实现了 ViewModel 层所需的效果，是不是和 EL 表达式非常像?

测试：

为了能够更直观的体验 Vue 带来的数据绑定功能，我们需要在浏览器测试一番，操作流程如下：

1、在浏览器上运行第一个 Vue 应用程序，进入 开发者工具

2、在控制台输入 `vm.message = 'Hello World'` ，然后 回车，你会发现浏览器中显示的内容会直接变成 Hello World

此时就可以在控制台直接输入 vm.message 来修改值，中间是可以省略 data 的，在这个操作中，我并没有主动操作 DOM，就让页面的内容发生了变化，这就是借助了 Vue 的 数据绑定 功能实现的；MVVM 模式中要求 ViewModel 层就是使用 观察者模式 来实现数据的监听与绑定，以做到数据与视图的快速响应。

> VsCode 技巧

文件 => 首选项 => 用户代码片段 => 新建全局代码片段：vue-html.code-snippets

注意：制作代码片段的时候，字符串中如果包含文件中复制过来的“Tab”键的空格，要换成“空格键”的空格

```
{
    "vue htm": {
        "scope": "html",
        "prefix": "vuehtml",
        "body": [
            "<!DOCTYPE html>",
            "<html lang=\"en\">",
            "",
            "<head>",
            "    <meta charset=\"UTF-8\">",
            "    <meta name=\"viewport\" content=\"width=device-width,
initial-scale=1.0\">",
            "    <meta http-equiv=\"X-UA-Compatible\" content=\"ie=edge\">",
            "    <title>Document</title>",
            "</head>",
            "",
            "<body>",
            "    <div id=\"app\">",
            "",
            "    </div>",
            "    <script
src=\"https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js\"></script>",
            "    <script>",
            "        new Vue({",
            "            el: '#app',",
            "            data: {",
```

```
            "                $1",
            "            }",
            "        })",
            "    </script>",
            "</body>",
            "",
            "</html>",
        ],
        "description": "my vue template in html"
    }
}
```

创建一个 02-vuetemplate.html , 输出 vue，发现可以自动补全!

基础语法

我们对于基础语法，说白了就是实现元素赋值，循环，判断，以及事件响应即可!

1、v-bind

我们已经成功创建了第一个 Vue 应用！看起来这跟渲染一个字符串模板非常类似，但是 Vue 在背后做了大量工作。现在数据和 DOM 已经被建立了关联，所有东西都是响应式的。我们在控制台操作对象属性，界面可以实时更新!

我们还可以使用`v-bind`来绑定元素特性!

代码：01-v-bind.html

```
<body>
    <div id="app">
        <!--
            如果要将模型数据绑定在html属性中
            则使用 v-bind 指令,此时title中显示的是模型数据
        -->
        <h1 v-bind:title="message">鼠标悬停几秒钟查看此处动态绑定的提示信息！</h1>
        <!-- v-bind 指令的简写形式： 冒号（:） -->
        <h1 :title="message">我是标题</h1>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        new Vue({
            el: '#app',
            data: {
                message: '页面加载于 ' + new Date().toLocaleString()
            }
        })
    </script>
</body>
```

你看到的 v-bind 特性被称为指令。指令带有前缀 v- ，以表示它们是 Vue 提供的特殊特性。

除了使用插值表达式{{}}进行数据渲染，也可以使用 v-bind指令，它的简写的形式就是一个冒号（:）

2、v-if 系列

什么是条件判断语句，就不需要我说明了吧（￣▽￣），以下两个属性！

- v-if
- v-else-if
- v-else

代码：02-v-if.html

```
<body>
    <div id="app">
        <!--
            === 三个等号在 JS 中表示绝对等于（就是数据与类型都要相等）
        -->
        <h1 v-if="type === 'A'">A</h1>
        <h1 v-else-if="type === 'B'">B</h1>
        <h1 v-else-if="type === 'C'">C</h1>
        <h1 v-else>who</h1>
    </div>

    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        var vm = new Vue({
            el: '#app',
            data: {
                type: 'A'
            }
        })
    </script>
</body>
```

测试：观察在控制台输入 vm.type = 'B'、'C'、'D' 的变化

3、v-for

语法格式：

```
<div id="vue">
    <li v-for="item in items">
        {{ item.message }}
    </li>
</div>
```

注：items 是数组，item是数组元素迭代的别名。和Thymeleaf模板引擎的语法和这个十分的相似！

代码：03-v-for.html

```
<body>
    <div id="app">
        <li v-for="item in items">
            {{ item.message }}
        </li>
    </div>
```

```
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        var vm = new Vue({
            el: '#app',
            data: {
                //items数组
                items: [
                    {message: '狂神说Java'},
                    {message: '狂神说前端'}
                ]
            }
        });
    </script>
</body>
```

测试：在控制台输入 `vm.items.push({message: '狂神说运维'})` ，尝试追加一条数据，你会发现浏览器中显示的内容会增加一条 `狂神说运维` .

4、v-on

`v-on` 监听事件：

事件有Vue的事件、和前端页面本身的一些事件！我们这`click`是vue的事件，可以绑定到Vue中的`methods`中的方法事件！

代码：04-v-on.html

```
<body>
    <div id="app">
        <!--
            在这里我们使用了 v-on 绑定了 click 事件
            并指定了名为 sayHi 的方法
        -->
        <button v-on:click="sayHi">点我</button>
        <!-- v-on 指令的简写形式 @ -->
        <button @click="sayHi">点我</button>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        var vm = new Vue({
            el: '#app',
            data: {
                message: 'Hello World'
            },
            // 方法必须定义在 Vue 实例的 methods 对象中
            methods: {
                sayHi: function (event) {
                    // `this` 在方法里指向当前 Vue 实例
                    alert(this.message);
                }
            }
        });
    </script>
</body>
```

点击测试即可

5、v-model

> 双向数据绑定

Vue.js 是一个 MVVM 框架，即数据双向绑定，即当数据发生变化的时候，视图也就发生变化，当视图发生变化的时候，数据也会跟着同步变化。这也算是 Vue.js 的精髓之处了。

值得注意的是，我们所说的数据双向绑定，一定是对于 UI 控件来说的，非 UI 控件不会涉及到数据双向绑定。对于我们处理表单，Vue.js 的双向数据绑定用起来就特别舒服了。

你可以用 `v-model` 指令在表单 < input>、< textarea> 及 < select> 元元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素。尽管有些神奇，但 v-model 本质上不过是语法糖。它负责监听用户的输入事件以更新数据，并对一些极端场景进行一些特殊处理。

> 测试代码

代码：05-v-bind-1.html 文本框

```
<body>
    <div id="app">
        <!-- v-bind:value只能进行单向的数据渲染 -->
        <input type="text" v-bind:value="searchMap.keyWord">
        <!-- v-model 可以进行双向的数据绑定  -->
        <input type="text" v-model="searchMap.keyWord">
        <p>您要查询的是：{{searchMap.keyWord}}</p>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        new Vue({
            el: '#app',
            data: {
                searchMap:{
                    keyWord: 'kuangshen'
                }
            }
        })
    </script>
</body>
```

填写完毕后，打开浏览器，对比两个输入框，查看效果！

代码：05-v-bind-2.html 单复选框

```
<body>
    <div id="app">
        单复选框：
        <input type="checkbox" id="checkbox" v-model="checked">
          
        <label for="checkbox">{{ checked }}</label>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
```

```
    <script>
        new Vue({
            el: '#app',
            data: {
                checked: false
            }
        })
    </script>
</body>
```

代码：05-v-bind-3.html 多复选框

```
<body>
    <div id="app">
        多复选框：
        <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
        <label for="jack">Jack</label>
        <input type="checkbox" id="john" value="John" v-model="checkedNames">
        <label for="john">John</label>
        <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
        <label for="mike">Mike</label>
        <span>选中的值：{{ checkedNames }}</span>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                checkedNames: []
            }
        })
    </script>
</body>
```

代码：05-v-bind-4.html 单选按钮

```
<body>
    <div id="app">
        单选按钮：
        <input type="radio" id="one" value="One" v-model="picked">
        <label for="one">One</label>
        <input type="radio" id="two" value="Two" v-model="picked">
        <label for="two">Two</label>
        <span>选中的值：{{ picked }}</span>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>
    <script>
        new Vue({
            el: '#app',
            data: {
                picked: ''
            }
        })
```

```
    </script>
</body>
```

代码：05-v-bind-5.html 下拉框

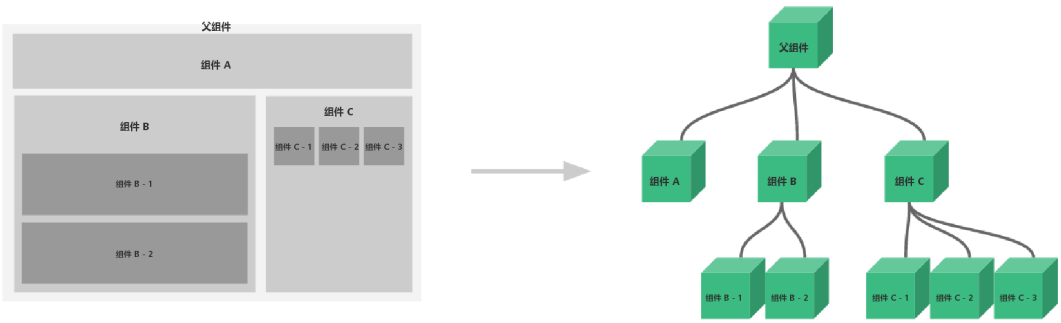
```
<body>
    <div id="app">
        下拉框：
        <select v-model="selected">
            <option disabled value="">请选择</option>
            <option>A</option>
            <option>B</option>
            <option>C</option>
        </select>
        <span>选中的值: {{ selected }}</span>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        var vm = new Vue({
            el: '#app',
            data: {
                selected: ''
            }
        });
    </script>
</body>
```

注意：如果 `v-model` 表达式的初始值未能匹配任何选项，< select> 元素将被渲染为“未选中”状态。在 iOS 中，这会使用户无法选择第一个选项。因为这样的情况下，iOS 不会触发 change 事件。因此，更推荐像上面这样提供一个值为空的禁用选项。

组件

> 什么是组件

组件是可复用的 `Vue` 实例，说白了就是一组可以重复使用的模板，跟 JSTL 的自定义标签、Thymeleaf 的 `th:fragment` 等框架有着异曲同工之妙。通常一个应用会以一棵嵌套的组件树的形式来组织：



例如，你可能会有页头、侧边栏、内容区等组件，每个组件又包含了其它的像导航链接、博文之类的组件。

> 第一个 Vue 组件

注意：在实际开发中，我们并不会用以下方式开发组件，以下方法只是为了让大家理解什么是组件。

使用 Vue.component() 方法注册组件：

06-vue-component.html

```
<body>
    <div id="app">
        <ul>
            <!-- 有点类似自定义标签 -->
            <my-component-li></my-component-li>
        </ul>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        // 先注册组件
        Vue.component('my-component-li', {
            template: '<li>Hello li</li>'
        });

        // 再实例化 Vue
        var vm = new Vue({
            el: '#app'
        });
    </script>
</body>
```

说明：

- Vue.component()：注册组件
- my-component-li：自定义组件的名字
- template：组件的模板

> 使用 props 属性传递参数

像上面那样用组件没有任何意义，所以我们是需要传递参数到组件的，此时就需要使用 `props` 属性了！

注意：默认规则下 props 属性里的值不能为大写：

06-vue-component-2.html

```
<body>
    <div id="app">
        <ul>
            <my-component-li v-for="item in items" v-bind:item="item">
            </my-component-li>
        </ul>
```

```
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        // 先注册组件
        Vue.component('my-component-li', {
            props: ['item'],
            template: '<li>Hello {{item}}</li>'
        });

        // 再实例化 Vue
        var vm = new Vue({
            el: '#app',
            data: {
                items: ["张三", "李四", "王五"]
            }
        });
    </script>
</body>
```

说明：

- `v-for="item in items"`：遍历 `Vue` 实例中定义的名为 `items` 的数组，并创建同等数量的组件；
- `v-bind:item="item"`：将遍历的 `item` 项绑定到组件中 `props` 定义的名为 `item` 属性上；
- = 号左边的 item 为 props 定义的属性名，右边的为 `item in items` 中遍历的 item 项的值；

计算属性

计算属性的重点突出在 `属性` 两个字上（属性是名词），首先它是个 `属性` 其次这个属性有 `计算` 的能力（计算是动词），这里的 `计算` 就是个函数；简单点说，它就是一个能够将计算结果缓存起来的属性（将行为转化成了静态的属性），仅此而已；可以想象为缓存！

代码测试：07-vue-computed.html

```
<body>
    <div id="app">
        <!--注意，一个是方法，一个是属性-->
        <p>调用当前时间的方法：{{currentTime1()}}</p>
        <p>当前时间的计算属性：{{currentTime2}}</p>
    </div>
    <script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js">
</script>
    <script>
        var vue = new Vue({
            el: '#app',
            data: {
                message: 'Hello Vue'
            },
            methods: {
                currentTime1: function () {
                    return Date.now();
                }
            },
```

```
            computed: {
                //currentTime2 ，这是一个属性！不是方法
                currentTime2: function () {
                    this.message;
                    return Date.now();
                }
            }
        })
    </script>
</body>
```

注意：methods 和 computed 里的东西不能重名

说明：

- methods：定义方法，调用方法使用 currentTime1()，需要带括号；
- computed：定义计算属性，调用属性使用 currentTime2，不需要带括号；this.message 是为了能够让 currentTime2 观察到数据变化而变化；
- 如果在方法中的值发生了变化，则缓存就会刷新！可以在控制台使用 `vm.message="qinjiang"` 改变下数据的值，再次测试观察效果！

结论：

调用方法时，每次都需要进行计算，既然有计算过程则必定产生系统开销，那如果这个结果是不经常变化的呢？此时就可以考虑将这个结果缓存起来，采用计算属性可以很方便的做到这一点，**计算属性的主要特性就是为了将不经常变化的计算结果进行缓存，以节约我们的系统开销；**

插槽

在 Vue 中我们使用 `<slot>` 元素，作为承载分发内容的出口，作者称其为 插槽，可以应用在组合组件的场景中；

比如准备制作一个待办事项组件（todo），该组件由待办标题（todo-title）和待办内容（todo-items）组成，但这三个组件又是相互独立的，该如何操作呢？

08-vue-todo.html

第一步: 定义一个待办事项的组件

```
<todo></todo>
<script type="text/javascript">
// 定义一个待办事项的组件
Vue.component('todo', {
template: '<div>\
             <div>待办事项</div>\
                <ul>\
                    <li>学习狂神说Java</li>\
                </ul>\
            </div>'
});
</script>
```

第二步: 我们需要让待办事项的标题和值实现动态绑定,怎么做呢? 我们可以留出一个插槽!

1、将上面的代码留出一个插槽,即 slot

```
Vue.component('todo', {
    template: '<div>\
                    <slot name="todo-title"></slot>\
                    <ul>\
                        <slot name="todo-items"></slot>\
                    </ul>\
                </div>'
});
```

2、定义一个名为 todo-title 的待办标题组件 和 todo-items 的待办内容组件

```
Vue.component('todo-title', {
    props: ['title'],
    template: '<div>{{title}}</div>'
});
```

```
//这里的index,就是数组的下标,使用for循环遍历的时候,可以循环出来!
Vue.component('todo-items', {
    props: ['item', 'index'],
    template: '<li>{{index + 1}}. {{item}}</li>'
});
```

3、实例化 Vue 并初始化数据

```
new Vue({
    el: '#app',
    data: {
        todoItems: ['狂神说Java', '狂神说运维', '狂神说前端']
    }
})
```

4、将这些值,通过插槽插入

```
<div id="app">
    <todo>
        <todo-title slot="todo-title" title="秦老师系列课程"></todo-title>
        <todo-items slot="todo-items" v-for="(item, index) in todoItems"
                    v-bind:item="item" v-bind:index="index"></todo-items>
    </todo>
</div>
```

说明:我们的 todo-title 和 todo-items 组件分别被分发到了 todo 组件的 todo-title 和 todo-items 插槽中

> 进阶：自定义事件

通过以上代码不难发现，数据项在 Vue 的实例中，但删除操作要在组件中完成，那么组件如何才能删除 Vue 实例中的数据呢？此时就涉及到参数传递与事件分发了，Vue 为我们提供了自定义事件的功能很好的帮助我们解决了这个问题；使用 this.$emit('自定义事件名', 参数)，操作过程如下:

1、在vue的实例中,增加了 methods 对象并定义了一个名为 removeTodoItems 的方法

```
new Vue({
    el: '#app',
    data: {
```

```
        todoItems: ['狂神说Java', '狂神说运维', '狂神说前端']
    },
    methods: {
        // 该方法可以被模板中自定义事件触发
        removeTodoItems: function (index) {
            console.log("删除 " + this.todoItems[index] + " 成功");
            // splice() 方法向/从数组中添加/删除项目，然后返回被删除的项目
            // 其中 index 为添加/删除项目的位置，1 表示删除的数量
            this.todoItems.splice(index, 1);
        }
    }
})
```

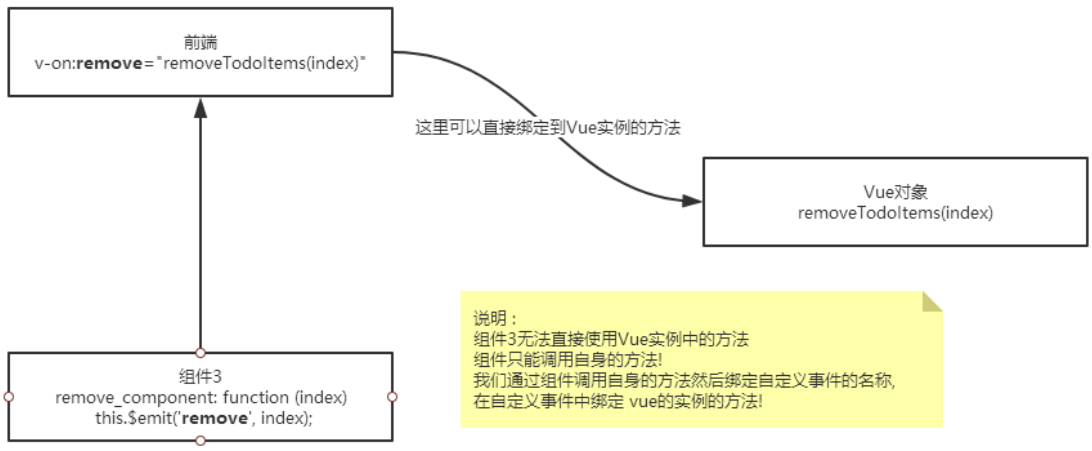
2、修改 todo-items 待办内容组件的代码,增加一个删除按钮,并且绑定事件!

```
Vue.component('todo-items', {
    props: ['item', 'index'],
    template: '<li>{{index + 1}}. {{item}}  <button @click="remove_component">删除</button></li>',
    methods: {
        remove_component: function (index) {
            // 这里的 remove 是自定义事件的名称，需要在 HTML 中使用 v-on:remove 的方式指派
            this.$emit('remove', index);
        }
    }
});
```

3、修改 todo-items 待办内容组件的 HTML 代码,增加一个自定义事件,比如叫 remove,可以和组件的方法绑定,然后绑定到vue的方法中!

```
<!--增加了 v-on:remove="removeTodoItems(index)" 自定义事件，该事件会调用 Vue 实例中定义的名为 removeTodoItems 的方法-->
<todo-items slot="todo-items" v-for="(item, index) in todoItems"
            v-bind:item="item" v-bind:index="index"
            v-on:remove="removeTodoItems(index)"></todo-items>
```

> 逻辑理解



> Vue 入门小结

核心 : 数据驱动 , 组件化

优点 : 借鉴了 AngulaJS 的模块化开发 和 React 的虚拟Dom , 虚拟Dom就是把Dom操作放到内存中执行 ;

常用的属性 :

- v-if
- v-else-if
- v-else
- v-for
- v-on 绑定事件 , 简写`@`
- v-model 数据双向绑定
- v-bind 给组件绑定参数,简写 `:`

组件化 :

- 组合组件， slot 插槽。
- 组件内部绑定事件需要使用到 `this.$emit("事件名",参数)` ;
- 计算属性的特色,缓存计算数据

遵循SoC 关注度分离原则,Vue是纯粹的视图框架,并不包含,比如Ajax之类的通信功能,为了解决通信问题,我们需要使用Axios 框架做异步通信;

Axios

> 什么是Axios

Axios 是一个开源的可以用在浏览器端和 `NodeJS` 的异步通信框架，她的主要作用就是实现 AJAX 异步通信，其功能特点如下：

- 从浏览器中创建 `XMLHttpRequests`
- 从 node.js 创建 http 请求
- 支持 Promise API [JS中链式编程]
- 拦截请求和响应
- 转换请求数据和响应数据
- 取消请求
- 自动转换 JSON 数据
- 客户端支持防御 XSRF（跨站请求伪造）

GitHub：https://github.com/axios/axios

中文文档：http://www.axios-js.com/

> 为什么要使用 Axios

由于 `Vue.js` 是一个 视图层框架 并且作者（尤雨溪）严格准守 SoC （关注度分离原则），所以 `Vue.js` 并不包含 AJAX 的通信功能，为了解决通信问题，作者单独开发了一个名为 `vue-resource` 的插件，不过在进入 2.0 版本以后停止了对该插件的维护并推荐了 `Axios` 框架。少用jQuery，因为它操作Dom太频繁！

> 第一个 Axios 应用程序

1、咱们开发的接口大部分都是采用 JSON 格式，可以先在项目里模拟一段 JSON 数据，数据内容如下：创建一个名为 data.json 的文件并填入上面的内容，放在项目的根目录下。

```
{
  "name": "狂神说Java",
  "url": "https://blog.kuangstudy.com",
  "page": 1,
  "address": {
    "street": "洪崖洞",
    "city": "重庆市",
    "country": "中国"
  }
}
```

2、测试代码

```
<!DOCTYPE html>
<html lang="en" xmlns:v-bind="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="UTF-8">
    <title>狂神说Java</title>
    <!--v-cloak 解决闪烁问题-->
    <style>
        [v-cloak] {
            display: none;
        }
    </style>
</head>
<body>

<div id="vue" v-cloak>
    <div>名称：{{info.name}}</div>
    <div>地址：{{info.address.country}}-{{info.address.city}}-{{info.address.street}}</div>
    <div>链接：<a v-bind:href="info.url" target="_blank">{{info.url}}</a></div>
</div>

<!--引入 JS 文件-->
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.21/dist/vue.js"></script>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script type="text/javascript">
    var vm = new Vue({
        el: '#vue',
        data() {
            return {
                info: {
                    name: null,
                    address: {
                        country: null,
                        city: null,
                        street: null
                    },
                    url: null
                }
```

```
            }
        },
        mounted() { //钩子函数
            axios
                .get('data.json')
                .then(response => (this.info = response.data));
        }
    });
</script>

</body>
</html>
```

说明：

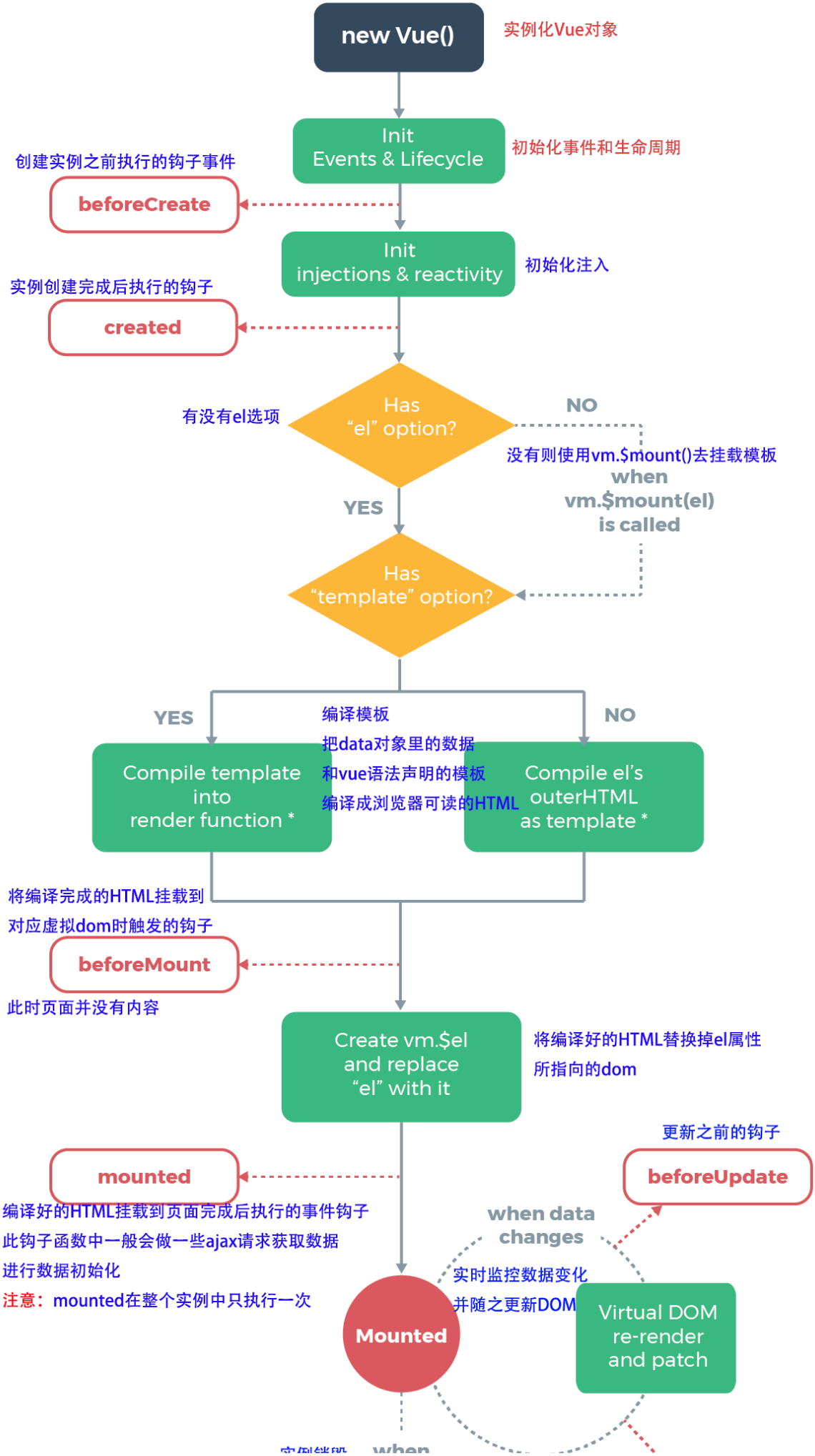
1. 在这里使用了 v-bind 将 a:href 的属性值与 Vue 实例中的数据进行绑定
2. 使用 axios 框架的 get 方法请求 AJAX 并自动将数据封装进了 Vue 实例的数据对象中
3. 我们在data中的数据结构必须要和`Ajax`响应回来的数据格式匹配!

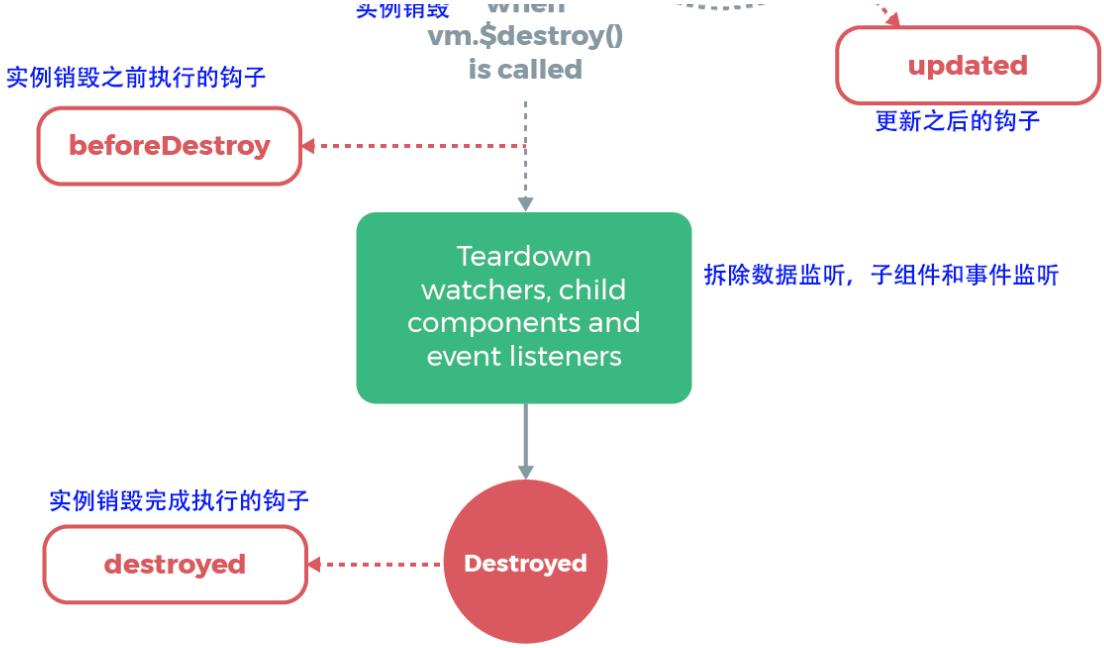
> Vue的生命周期

官方文档：https://cn.vuejs.org/v2/guide/instance.html#生命周期图示

Vue 实例有一个完整的生命周期，也就是从开始创建、初始化数据、编译模板、挂载 DOM、渲染→更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。通俗说就是 Vue 实例从创建到销毁的过程，就是生命周期。

在 Vue 的整个生命周期中，它提供了一系列的事件，可以让我们在事件触发时注册 JS 方法，可以让我们用自己注册的 JS 方法控制整个大局，在这些事件响应方法中的 this 直接指向的是 Vue 的实例。


new Vue()
实例化Vue对象
Init
Events & Lifecycle
初始化事件和生命周期
创建实例之前执行的钩子事件
beforeCreate
Init
injections & reactivity
初始化注入
实例创建完成后执行的钩子
created
Has
“el” option?
有没有el选项
NO
没有则使用vm.$mount()去挂载模板
when
vm.$mount(el)
is called
YES
Has
“template” option?
YES
编译模板
把data对象里的数据
和vue语法声明的模板
编译成浏览器可读的HTML
NO
Compile template
into
render function *
Compile el’s
outerHTML
as template *
将编译完成的HTML挂载到
对应虚拟dom时触发的钩子
beforeMount
此时页面并没有内容
Create vm.$el
and replace
“el” with it
将编译好的HTML替换掉el属性
所指向的dom
更新之前的钩子
mounted
beforeUpdate
编译好的HTML挂载到页面完成后执行的事件钩子
此钩子函数中一般会做一些ajax请求获取数据
进行数据初始化
注意：mounted在整个实例中只执行一次
when data
changes
实时监控数据变化
并随之更新DOM
Mounted
Virtual DOM
re-render
and patch



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

初识路由

> Vue.js 路由允许我们通过不同的 URL 访问不同的内容。
>
> 通过 Vue.js 可以实现多视图的单页Web应用（single page web application，SPA）

1、创建router文件夹

2、npm 下载 资源

```
npm install vue
npm install vue-router
```

3、新建一个lib目录，将资源复制出来

4、创建一个 路由.html

5、引入js

```
<!--资源导入: 一定要注意导入顺序的问题，大坑！！！-->
<script src="./lib/vue.min.js"></script>
<script src="./lib/vue-router.min.js"></script>
```

6、编写html

```
<!-- app -->
<div id="app">
    <h1>Hello App!</h1>
    <p>
```

```
        <!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
        <!-- 通过传入 `to` 属性指定链接. -->
        <router-link to="/">首页</router-link>
        <router-link to="/student">会员管理</router-link>
        <router-link to="/teacher">讲师管理</router-link>
    </p>
    <!-- 路由出口 -->
    <!-- 路由匹配到的组件将渲染在这里 -->
    <router-view></router-view>
</div>
```

7、编写js

```
<script>
    // 1. 定义（路由）组件。
    // 复杂的组件也可以从独立的vue文件中引入
    const Welcome = { template: '<div>欢迎</div>' }
    const Student = { template: '<div>student list</div>' }
    const Teacher = { template: '<div>teacher list</div>' }

    // 2. 定义路由
    // 每个路由应该映射一个组件。
    const routes = [
        { path: '/', redirect: '/welcome' }, //设置默认指向的路径
        { path: '/welcome', component: Welcome },
        { path: '/student', component: Student },
        { path: '/teacher', component: Teacher }
    ]

    // 3. 创建 router 实例，然后传 `routes` 配置
    const router = new VueRouter({
        routes // （缩写）相当于 routes: routes
    })

    // 4. 创建和挂载根实例。
    // 从而让整个应用都有路由功能
    new Vue({
        el: '#app',
        router
    })

    // 现在，应用已经启动了!
</script>
```

测试，路由跳转成功!

拓展：Vue-Cli

> 什么是vue-cli

vue-cli 官方提供的一个脚手架,用于快速生成一个 vue 的项目模板；

预先定义好的目录结构及基础代码，就好比咱们在创建 Maven 项目时可以选择创建一个骨架项目，这个骨架项目就是脚手架,我们的开发更加的快速;

主要的功能：

- 统一的目录结构
- 本地调试
- 热部署
- 单元测试
- 集成打包上线

> 需要的环境

安装 vue-cli

```
cnpm install vue-cli -g
# 测试是否安装成功
# 查看可以基于哪些模板创建 vue 应用程序，通常我们选择 webpack
vue list
```

```
C:\Users\Administrator>vue list

  Available official templates:

  ★  browserify - A full-featured Browserify + vueify setup with hot-reload, linting & unit testing.
  ★  browserify-simple - A simple Browserify + vueify setup for quick prototyping.
  ★  pwa - PWA template for vue-cli based on the webpack template
  ★  simple - The simplest possible Vue setup in a single HTML file
  ★  webpack - A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction.
  ★  webpack-simple - A simple Webpack + vue-loader setup for quick prototyping.
```

> 第一个 vue-cli 应用程序

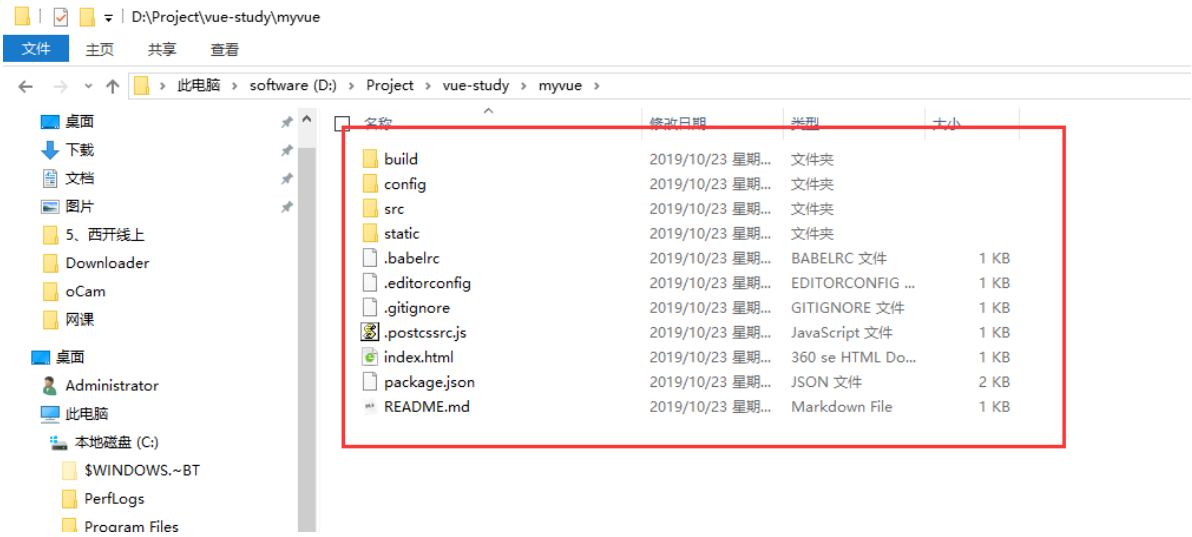
1、我们新建一个文件夹 vue-cli;

2、创建一个基于 webpack 模板的 vue 应用程序

```
# 这里的 myvue 是项目名称，可以根据自己的需求起名
vue init webpack myvue
# 一路都选择no即可;
```

说明：

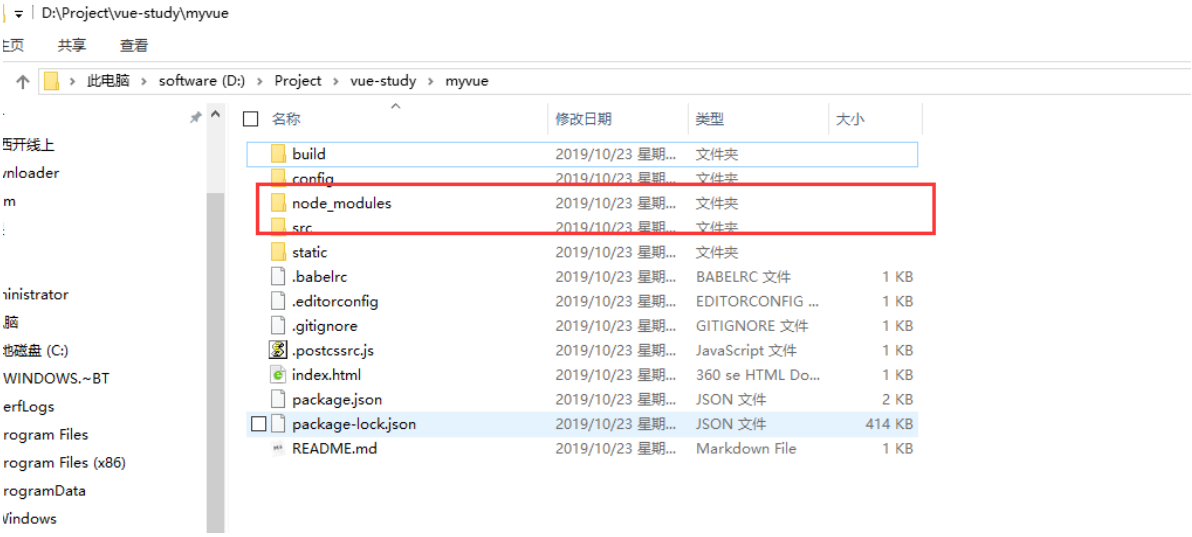
- Project name：项目名称，默认 回车 即可
- Project description：项目描述，默认 回车 即可
- Author：项目作者，默认 回车 即可
- Install vue-router：是否安装 vue-router，选择 n 不安装（后期需要再手动添加）
- Use ESLint to lint your code：是否使用 ESLint 做代码检查，选择 n 不安装（后期需要再手动添加）
- Set up unit tests：单元测试相关，选择 n 不安装（后期需要再手动添加）
- Setup e2e tests with Nightwatch：单元测试相关，选择 n 不安装（后期需要再手动添加）
- Should we run npm install for you after the project has been created：创建完成后直接初始化，选择 n，我们手动执行;运行结果!



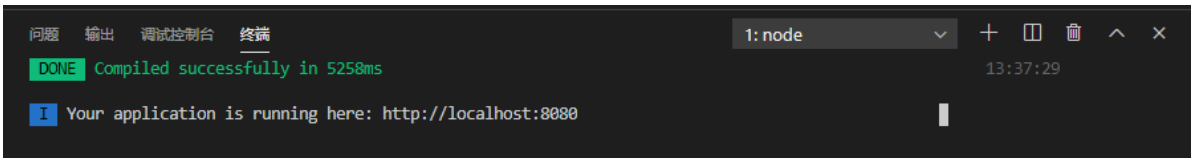
> 初始化并运行

```
cd myvue
npm install
npm run dev
```

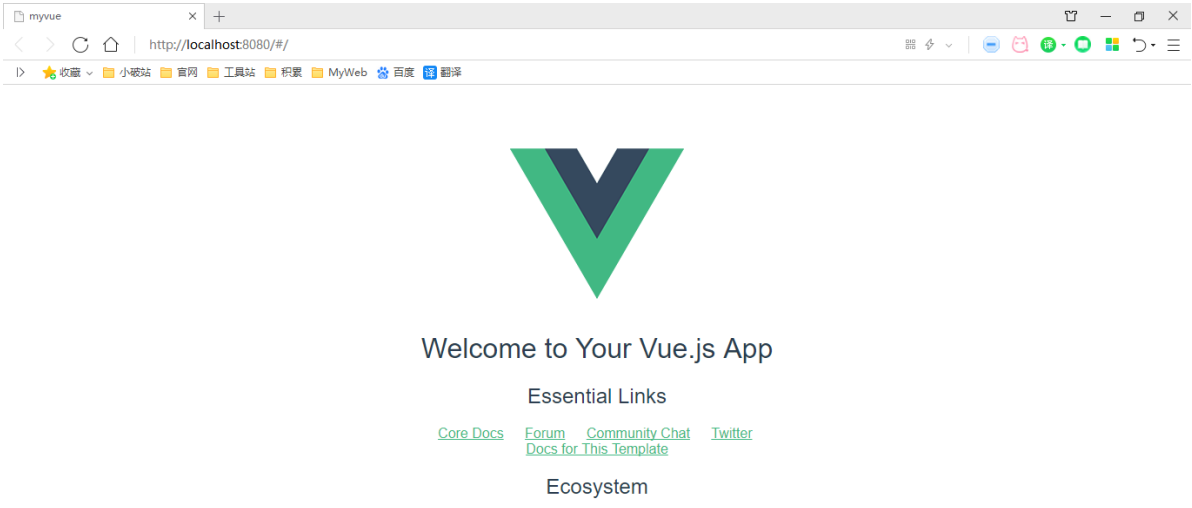
执行完成后,目录多了很多依赖



安装并运行成功后在浏览器输入：http://localhost:8080



效果:

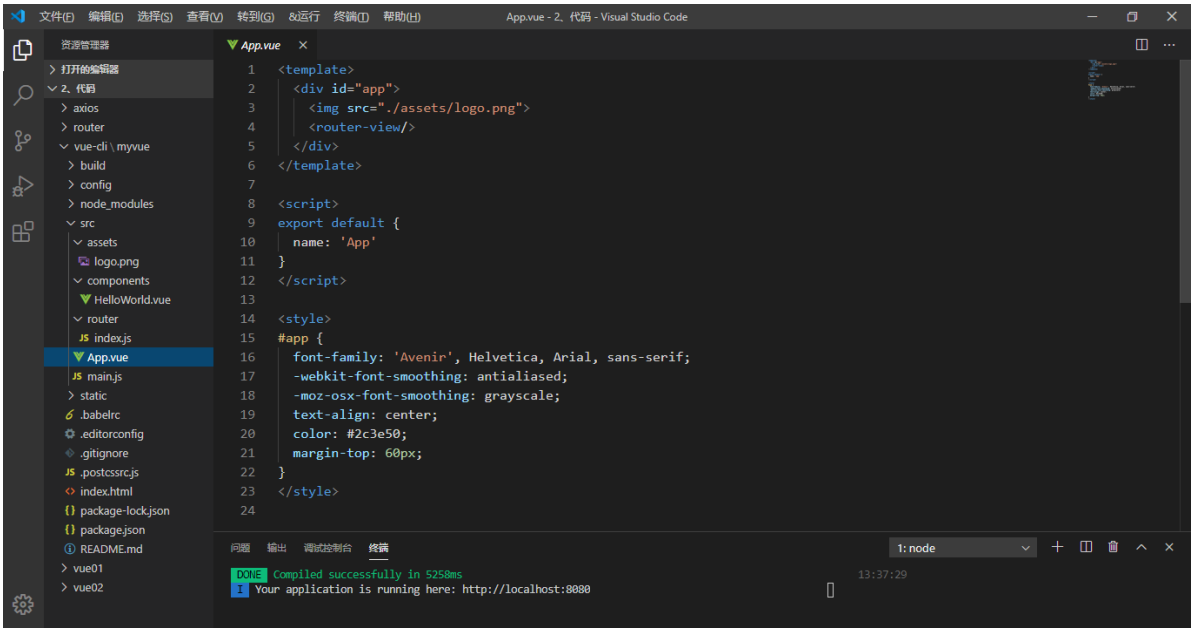


Vue-cli目录结构

- build 和 config：WebPack 配置文件
- node_modules：用于存放 npm install 安装的依赖文件
- src： 项目源码目录
- static：静态资源文件
- .babelrc：Babel 配置文件，主要作用是将 ES6 转换为 ES5
- .editorconfig：编辑器配置
- eslintignore：需要忽略的语法检查配置文件
- .gitignore：git 忽略的配置文件
- .postcssrc.js：css 相关配置文件，其中内部的 module.exports 是 NodeJS 模块化语法
- index.html：首页，仅作为模板页，实际开发时不使用
- package.json：项目的配置文件
    - name：项目名称
    - version：项目版本
    - description：项目描述
    - author：项目作者
    - scripts：封装常用命令
    - dependencies：生产环境依赖
    - devDependencies：开发环境依赖

src 目录

`src` 目录是项目的源码目录，所有代码都会写在这里！



main.js

项目的入口文件，我们知道所有的程序都会有一个入口

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'

Vue.config.productionTip = false;

/* eslint-disable no-new */
new Vue({
  el: '#app',
  components: { App },
  template: '<App/>'
});
```

- `import Vue from 'vue'`：ES6 写法，会被转换成 require("vue");（require 是 NodeJS 提供的模块加载器）
- `import App from './App'`：意思同上，但是指定了查找路径，./ 为当前目录
- `Vue.config.productionTip = false`：关闭浏览器控制台关于环境的相关提示
- `new Vue({...})`：实例化 Vue
  - `el: '#app'`：查找 index.html 中 id 为 app 的元素
  - `template: ''`：模板，会将 index.html 中 替换为
  - `components: { App }`：引入组件，使用的是 import App from './App' 定义的 App 组件;

> App.vue

```vue
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <HelloWorld/>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

- template：HTML 代码模板，会替换 < App /> 中的内容
- import HelloWorld from './components/HelloWorld'：引入 HelloWorld 组件用于替换 template 中的 < HelloWorld/>
- export default{...}：导出 NodeJS 对象，作用是可以通过 import 关键字导入
  - name: 'App'：定义组件的名称
  - components: { HelloWorld }：定义子组件
- 在hello,Vue中,关于 < style scoped> 的说明：CSS 样式仅在当前组件有效，声明了样式的作用域，是当前的界面私有的!

拓展：再探路由

> 说明

Vue Router 是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

- 嵌套的路由/视图表
- 模块化的、基于组件的路由配置
- 路由参数、查询、通配符
- 基于 Vue.js 过渡系统的视图过渡效果

- 细粒度的导航控制
- 带有自动激活的 CSS class 的链接
- HTML5 历史模式或 hash 模式，在 IE9 中自动降级
- 自定义的滚动条行为

> 安装

基于第一个 `vue-cli` 进行测试学习;先查看node_modules中是否存在 vue-router

vue-router 是一个插件包，所以我们还是需要用 npm/cnpm 来进行安装的。打开命令行工具，进入你的项目目录，输入下面命令。

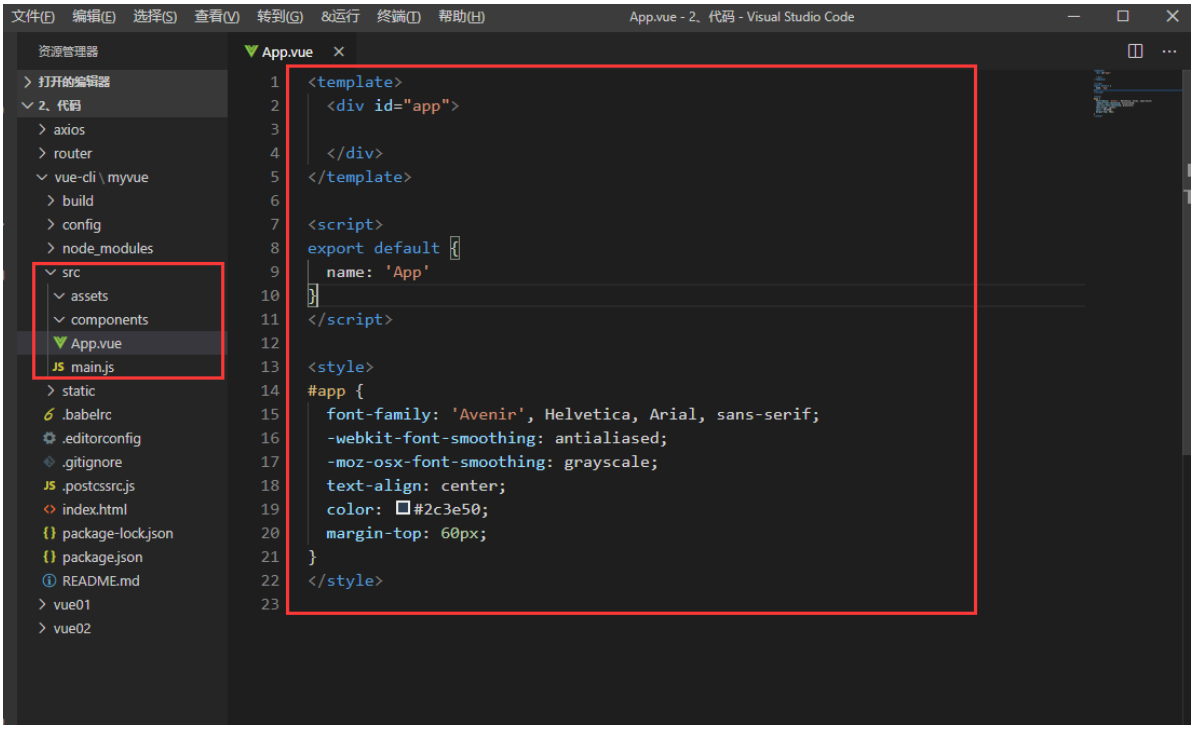
```
npm install vue-router --save-dev
```

如果在一个模块化工程中使用它，必须要通过 Vue.use() 明确地安装路由功能：

```
import Vue from 'vue'
import VueRouter from 'vue-router'
// 显示的使用
Vue.use(VueRouter);
```

> 测试

可以先清理掉多余的文件：Src 下面就很干净了



1、`components` 目录下存放我们自己编写的组件

2、定义组件 Content.vue 组件

```vue
<template>
  <div>
    <h1>内容页</h1>
  </div>
</template>

<script>
 export default {
     name: "Content"
 }
</script>
```

3、安装路由,在src目录下,新建一个文件夹 : `router` ,专门存放路由，写入 index.js

```js
import Vue from 'vue'
// 导入路由插件
import Router from 'vue-router'
// 导入上面定义的组件
import Content from '../components/Content'
// 安装路由
Vue.use(Router);
// 配置路由
export default new Router({
  routes: [
    {
      // 路由路径
      path: '/content',
      // 路由名称
      name: 'content',
      // 跳转到组件
      component: Content
    }
  ]
});
```

4、我们在新建一个 Main.vue 组件

```vue
<template>
  <div>
    <h1>首页</h1>
  </div>
</template>

<script>
 export default {
     name: "Main"
 }
</script>
```

5、将Main组件也配置到路由中!

```js
import Vue from 'vue'
// 导入路由插件
import Router from 'vue-router'
// 导入上面定义的组件
import Content from '../components/Content'
import Main from '../components/Main'
```

```
// 安装路由
Vue.use(Router);
// 配置路由
export default new Router({
  routes: [
    {
      // 路由路径
      path: '/content',
      // 路由名称
      name: 'content',
      // 跳转到组件
      component: Content
    },
    {
        // 路由路径
        path: '/main',
        // 路由名称
        name: 'main',
        // 跳转到组件
        component: Main
      }
  ]
});
```

6、在`main.js`中配置路由

```
import Vue from 'vue'
import App from './App'

// 导入上面创建的路由配置目录
import router from './router'

Vue.config.productionTip = false

new Vue({
  el: '#app',
  // 配置路由
  router,
  components: { App },
  template: '<App/>'
})
```

7、在`App.vue`中使用路由

```
<template>
  <div id="app">
    <!--
      router-link：  默认会被渲染成一个 <a> 标签，to 属性为指定链接
      router-view：  用于渲染路由匹配到的组件
    -->
    <router-link to="/main">首页</router-link>
    <router-link to="/content">内容</router-link>
    <router-view></router-view>
  </div>
</template>
```

启动测试一下：`npm run dev`

练习： 在现有的基础上，在增加一个路由组件，优化一下！

> 路由模式与404

路由模式有两种：

- hash：路径带 # 符号，如 http://localhost/#/login
- history：路径不带 # 符号，如 http://localhost/login

修改路由配置，代码如下：

```
export default new Router({
  mode: 'history',
  routes: [
  ]
});
```

处理 404 创建一个名为 `NotFound.vue` 的视图组件，代码如下：

```
<template>
  <div>
    页面不存在，请重试！
  </div>
</template>

<script>
  export default {
    name: "NotFount"
  }
</script>

<style scoped>

</style>
```

修改路由配置，代码如下：

```
import NotFound from '../components/NotFound'

{
   path: '*',
   component: NotFound
}
```

Element-ui

> 简介

element-ui 是饿了么前端出品的基于 Vue.js的 后台组件库，方便程序员进行页面快速布局和构建

官网： http://element-cn.eleme.io/#/zh-CN

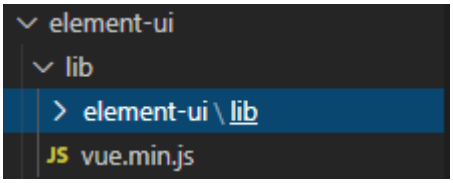
> 入门

1、创建文件夹 element-ui

2、npm 安装依赖

```
npm install vue
npm i element-ui -S
```

3、新建一个lib目录，导入库



4、创建 ui.html，引入 js 和 css 文件

```
<!-- import CSS -->
<link rel="stylesheet" href="./lib/element-ui/lib/theme-chalk/index.css">
<!-- import Vue before Element -->
<script src="./lib/vue.min.js"></script>
<!-- import JavaScript -->
<script src="./lib/element-ui/lib/index.js"></script>

```

5、编写页面

```
<div id="app">
    <el-button @click="visible = true">Button</el-button>
    <el-dialog :visible.sync="visible" title="Hello world">
        <p>Try Element</p>
    </el-dialog>
</div>

```

.sync：本例用于组件和程序间的通讯，使弹窗的关闭按钮起作用

6、编写JS

```
<script>
    new Vue({
      el: '#app',
      data: function () {//定义Vue中data的另一种方式
        return { visible: false }
      }
    })
</script>

```

7、运行测试效果！

【带领学员去element-ui官网上找组价测试，15~20分钟】

vue-element-admin

> 概述

vue-element-admin是基于element-ui 的一套后台管理系统集成方案。

功能：https://panjiachen.github.io/vue-element-admin-site/zh/guide/#功能

GitHub地址：https://github.com/PanJiaChen/vue-element-admin

项目在线预览：https://panjiachen.gitee.io/vue-element-admin

> 安装测试使用

1、在提供的素材中，找到 vue-element-admin-master ， 解压到我们的代码目录中

2、使用VsCode打开

```
# 解压压缩包
# 进入目录
cd vue-element-admin-master

# 安装依赖
npm install --registry=https://registry.npm.taobao.org

#错误Cannot find module 'node-sass'
cnpm install node-sass@latest --registry=https://registry.npm.taobao.org

# 启动。执行后，浏览器自动弹出并访问http://localhost:9527/
npm run dev
```

> vue-admin-template

1、简介

vue-admin-template是基于vue-element-admin的一套后台管理系统基础模板（最少精简版），可作为模板进行二次开发。

GitHub地址：https://github.com/PanJiaChen/vue-admin-template

建议：你可以在 `vue-admin-template` 的基础上进行二次开发，把 `vue-element-admin` 当做工具箱，想要什么功能或者组件就去 `vue-element-admin` 那里复制过来。

2、安装

在提供的素材中，找到 vue-admin-template-master ， 解压到我们的代码目录中，使用VsCode打开！

```
# 解压压缩包 # 安装python2环境
# 进入目录
cd vue-admin-template-master

# 安装依赖
npm install --registry=https://registry.npm.taobao.org

#错误Cannot find module 'node-sass'
cnpm install node-sass@latest

npm run dev
```

启动。执行后，浏览器自动弹出并访问http://localhost:9528/

观察浏览器的网络请求，发现这个登录的请求是远程的请求，但是有时候会不稳定，所以我们之后会换成本地的请求