# #### 1、前端框架（UI/JS）框架

<https://blog.csdn.net/enweitech/article/details/89316979>

Node.js是一个基于Chrome JavaScript运行时建立的平台， 用于方便地搭建响应速度快、易于扩展的网络应用。Node.js 使用事件驱动， 非阻塞I/O 模型而得以轻量和高效，非常适合在分布式设备上运行数据密集型的实时应用。

简单的说 node.js 就是运行在服务端的 JavaScript。

NodeJS适合运用在高并发、I/O密集、少量业务逻辑的场景。

AngularJS有着诸多特性，最为核心的是：MVVM、模块化、自动化双向数据绑定、语义化标签、依赖注入等等。

用途：通过描述我们应该就能很好的明白AngularJS的真实用途了，MVVM，模块化，自动化双向数据绑定等等。除了简单的dom操作外，更能体现Js编程的强大。

React

React 可以非常轻松地创建用**户交互界面**。为你应用的每一个状态设计简洁的视图,在**数据改变时 React 也可以高效地更新渲染界面**。React 起源于 Facebook 的内部项目，因为该公司对市场上所有 JavaScript MVC 框架，都不满意，就决定自己写一套，用来架设Instagram 的网站。

# #### 1、Vue.js是

一个构建数据驱动的 web 界面的渐进式框架。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兼具angular.js和react.js的优点，并剔除了它们的缺点”。

Vue.js 是用于构建交互式的 Web  界面的库。它提供了 MVVM 数据绑定和一个可组合的组件系统，具有简单、灵活的 API。从技术上讲， Vue.js 集中在 MVVM 模式上的视图模型层，并通过双向数据绑定连接视图和模型。实际的 DOM 操作和输出格式被抽象出来成指令和过滤器。相比其它的 MVVM 框架，Vue.js 更容易上手。

目前市场上比较流行的前后端分离的开发模式，大多前端都是vueJS做的。

# #### 2、Vue与其他框架的区别？

1.与AngularJS的区别  
相同点：  
都支持指令：内置指令和自定义指令。  
都支持过滤器：内置过滤器和自定义过滤器。  
都支持双向数据绑定。  
都不支持低端浏览器。  
不同点：  
（1）.AngularJS的学习成本高，比如增加了Dependency Injection特性，而Vue.js本身提供的API都比较简单、直观。  
（2）.在性能上，AngularJS依赖对数据做脏检查，所以Watcher越多越慢。  
Vue.js使用基于依赖追踪的观察并且使用异步队列更新。所有的数据都是独立触发的。  
（3）.vue中数据放在data对象里面，angular数据绑定在$scope上面  
（4）.vue有组件化概念，angular中没有  
2.与React的区别  
相同点：  
React采用特殊的JSX语法，Vue.js在组件开发中也推崇编写.vue特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用。  
中心思想相同：一切都是组件，组件实例之间可以嵌套。  
都提供合理的钩子函数，可以让开发者定制化地去处理需求。  
都不内置列数AJAX，Route等功能的核心包，而是以插件的方式加载。  
在组件开发中都支持mixins的特性。  
不同点：  
React依赖Virtual DOM,而Vue.js使用的是DOM模板。React采用的Virtual DOM会对渲染出来的结果做脏检查。  
Vue.js在模板中提供了指令，过滤器等，可以非常方便，快捷地操作DOM。

**#### 2、Angular React 和 Vue的比较**

前端这几年的技术发展很快，细分下来，主要可以分成四个方面：

1.开发语言技术，主要是ES6&7，coffeescript，typescript等；

2.开发框架，如Angular，React,Vue.js，Angular2等；

3.开发工具的丰富和前端工程化，像Grunt，Gulp，Webpack，npm，eslint，mocha这些技术；

4.前端开发范围的扩展，如服务端的nodejs,express,koa,meteor,GraphQL;移动端和跨平台的PhoneGap,ionic,ReactNative，Weex;计算机图形学和3维建模领域的WebGL(three.js等)；可视化与数据分析领域的d3.js等；包括浏览器不断开放的更多新特性和接口比如svg,canvas,蓝牙，电池，本地存储，service worker，Houdini等新的API能力，以及像WASM这样的底层优化技术；

就开发框架这块，Angular(1&2),React,Vue目前占据着主流地位而且会相持比较长的一段时间，所以这里对比一下这三门技术，以便之后的技术选型。

一 数据流

数据绑定

Angular 使用双向绑定即：界面的操作能实时反映到数据，数据的变更能实时展现到界面。

实现原理：

$scope变量中使用脏值检查来实现。像ember.js是基于setter，getter的观测机制，

$scope.$watch函数，监视一个变量的变化。函数有三参数，”要观察什么”，”在变化时要发生什么”,以及你要监视的是一个变量还是一个对象。

使用ng-model时，你可以使用双向数据绑定。

使用$scope.$watch（视图到模型）以及$scope.$apply（模型到视图），还有$scope.$digest

调用$scope.$watch时只为它传递了一个参数，无论作用域中的什么东西发生了变化，这个函数都会被调用。在ng-model中，这个函数被用来检查模型和视图有没有同步，如果没有同步，它将会使用新值来更新模型数据。

双向绑定的三个重要方法：

$scope.$apply()

$scope.$digest()

$scope.$watch()

在angularjs双向绑定中，有2个很重要的概念叫做dirty check，digest loop，dirty check（脏检测）是用来检查绑定的scope中的对象的状态的，例如，在js里创建了一个对象，并且把这个对象绑定在scope下，这样这个对象就处于digest loop中，loop通过遍历这些对象来发现他们是否改变，如果改变就会调用相应的处理方法来实现双向绑定

Vue 也支持双向绑定，angular默认为单向绑定，数据从父组件单向传给子组件。在大型应用中使用单向绑定让数据流易于理解。

脏检测的利弊

和ember.js等技术的getter/setter观测机制相比（优）：

getter/setter当每次对DOM产生变更，它都要修改DOM树的结构，性能影响大，Angular会把批量操作延时到一次更新，性能相对较好。

和Vue相比（劣）：

Vue.js 有更好的性能，并且非常非常容易优化，因为它不使用脏检查。Angular，当 watcher 越来越多时会变得越来越慢，因为作用域内的每一次变化，所有 watcher 都要重新计算。并且，如果一些 watcher 触发另一个更新，脏检查循环（digest cycle）可能要运行多次。 Angular 用户常常要使用深奥的技术，以解决脏检查循环的问题。有时没有简单的办法来优化有大量 watcher 的作用域。Vue.js 则根本没有这个问题，因为它使用基于依赖追踪的观察系统并且异步列队更新，所有的数据变化都是独立地触发，除非它们之间有明确的依赖关系。唯一需要做的优化是在 v-for 上使用 track-by。

React-单向数据流

MVVM流的Angular和Vue，都是通过类似模板的语法，描述界面状态与数据的绑定关系，然后通过内部转换，把这个结构建立起来，当界面发生变化的时候，按照配置规则去更新相应的数据，然后，再根据配置好的规则去，从数据更新界面状态。

React推崇的是函数式编程和单向数据流：给定原始界面（或数据），施加一个变化，就能推导出另外一个状态（界面或者数据的更新）。

React和Vue都可以配合Redux来管理状态数据。

二 视图渲染

Angular1

AngularJS的工作原理是:HTML模板将会被浏览器解析到DOM中, DOM结构成为AngularJS编译器的输入。AngularJS将会遍历DOM模板, 来生成相应的NG指令,所有的指令都负责针对view(即HTML中的ng-model)来设置数据绑定。因此, NG框架是在DOM加载完成之后, 才开始起作用的。

React

React 的渲染建立在 Virtual DOM 上——一种在内存中描述 DOM 树状态的数据结构。当状态发生变化时，React 重新渲染 Virtual DOM，比较计算之后给真实 DOM 打补丁。

Virtual DOM 提供了函数式的方法描述视图，它不使用数据观察机制，每次更新都会重新渲染整个应用，因此从定义上保证了视图与数据的同步。它也开辟了 JavaScript 同构应用的可能性。

在超大量数据的首屏渲染速度上，React 有一定优势，因为 Vue 的渲染机制启动时候要做的工作比较多，而且 React 支持服务端渲染。

React 的 Virtual DOM 也需要优化。复杂的应用里可以选择 1. 手动添加 shouldComponentUpdate 来避免不需要的 vdom re-render；2. Components 尽可能都用 pureRenderMixin，然后采用 Flux 结构 + Immutable.js。其实也不是那么简单的。相比之下，Vue 由于采用依赖追踪，默认就是优化状态：动了多少数据，就触发多少更新，不多也不少。

React 和 Angular 2 都有服务端渲染和原生渲染的功能。

Vue.js 不使用 Virtual DOM 而是使用真实 DOM 作为模板，数据绑定到真实节点。Vue.js 的应用环境必须提供 DOM。Vue.js 有时性能会比 React 好\*\*，而且几乎不用手工优化。

三 性能与优化

性能方面，这几个主流框架都应该可以轻松应付大部分常见场景的性能需求，区别在于可优化性和优化对于开发体验的影响。Vue 的话需要加好 track-by 。React 需要 shouldComponentUpdate 或者全面 Immutable，Angular 2 需要手动指定 change detection strategy。从整体趋势上来说，浏览器和手机还会越变越快，框架本身的渲染性能在整个前端性能优化体系中，会渐渐淡化，更多的优化点还是在构建方式、缓存、图片加载、网络链路、HTTP/2 等方面。

四 模块化与组件化

Angular1 -> Angular2

Angular1使用依赖注入来解决模块之间的依赖问题，模块几乎都依赖于注入容器以及其他相关功能。不是异步加载的，根据依赖列出第一次加载所需的所有依赖。

可以配合类似于Require.js来实现异步加载，懒加载（按需加载）则是借助于 ocLazyLoad 方式的解决方案，但是理想情况下应该是本地框架会更易懂。

Angular2使用ES6的module来定义模块，也考虑了动态加载的需求。

Vue

Vue中指令和组件分得更清晰。指令只封装 DOM 操作，而组件代表一个自给自足的独立单元 —— 有自己的视图和数据逻辑\*\*。在 Angular1 中两者有不少相混的地方。

React

一个 React 应用就是构建在 React 组件之上的。

组件有两个核心概念：props,state。

一个组件就是通过这两个属性的值在 render 方法里面生成这个组件对应的 HTML 结构。

传统的 MVC 是将模板放在其他地方，比如 script 标签或者模板文件，再在 JS 中通过某种手段引用模板。按这种思路，想想多少次我们面对四处分散的模板片段不知所措？纠结模板引擎，纠结模板存放位置，纠结如何引用模板。

React 认为组件才是王道，而组件是和模板紧密关联的，组件模板和组件逻辑分离让问题复杂化了。所以就有了 JSX 这种语法，就是为了把 HTML 模板直接嵌入到 JS 代码里面，这样就做到了模板和组件关联，但是 JS 不支持这种包含 HTML 的语法，所以需要通过工具将 JSX 编译输出成 JS 代码才能使用（可以进行跨平台开发的依据，通过不同的解释器解释成不同平台上运行的代码，由此可以有RN和React开发桌面客户端）。

五 语法与代码风格

React,Vue,Angular2都支持ES6，Angular2官方拥抱了TypeScript这种 JavaScript 风格。

React 以 JavaScript 为中心，Angular 2 依然保留以 HTML 为中心。Angular 2 将 “JS” 嵌入 HTML。React 将 “HTML” 嵌入 JS。Angular 2 沿用了 Angular 1 试图让 HTML 更强大的方式。

React 推荐的做法是 JSX + inline style，也就是把 HTML 和 CSS 全都整进 JavaScript 了。Vue 的默认 API 是以简单易上手为目标，但是进阶之后推荐的是使用 webpack + vue-loader 的单文件组件格式（template,script,style写在一个vue文件里作为一个组件）

**#### 2、angular和vue的区别**

1.各自的优缺点

2.应用场景

3.各自的双向数据绑定原理

4.vue虚拟dom：

利用在内存中生成与真实DOM与之对应的数据结构，这个在内存中生成的结构称之为虚拟DOM 。

当数据发生变化时，能够智能地计算出重新喧嚷组件的最小代价并应用到DOM操作上。

angular：1.工程化 2.双向数据绑定 3.依赖注入

值得注意的是当数据变化十分频繁时，脏检测对浏览器性能的消耗将会很大，官方注明的最大检测脏值为2000个数据。

vue:1.上手操作简单 2.项目结构不够工程化 3.组件间数据传递很麻烦

优点：配合es6 /webpack的模块化开发，组件化开发

# #### 2、Vue 和 React 区别

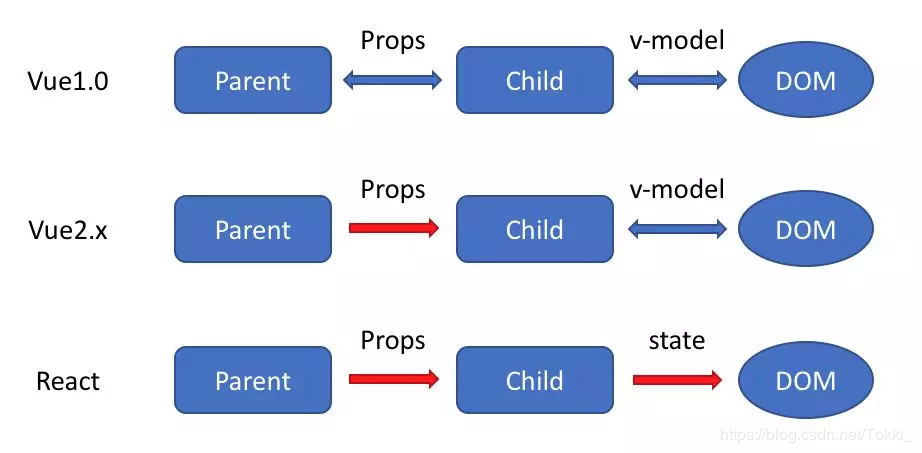
1.监听数据变化的实现原理不同

Vue 通过 getter/setter 以及一些函数的劫持，能精确快速的计算出 Virtual DOM 的差异。这是由于它在渲染过程中，会跟踪每一个组件的依赖关系，不需要重新渲染整个组件树。

React 默认是通过比较引用的方式进行的，如果不优化，每当应用的状态被改变时，全部子组件都会重新渲染，可能导致大量不必要的 VDOM 的重新渲染。

   Vue 不需要特别的优化就能达到很好的性能，而对于 React 而言，需要通过 PureComponent/shouldComponentUpdate 这个生命周期方法来进行控制。如果你的应用中，交互复杂，需要处理大量的 UI 变化，那么使用 Virtual DOM 是一个好主意。如果你更新元素并不频繁，那么 Virtual DOM 并不一定适用，性能很可能还不如直接操控 DOM。

      为什么 React 不精确监听数据变化呢？这是因为 Vue 和 React 设计理念上的区别，Vue 使用的是可变数据，而 React 更强调数据的不可变。

2.数据流的不同  


Vue 中默认支持双向绑定，组件与 DOM 之间可以通过 v-model 双向绑定。但是，父子组件之间，props 在 2.x 版本是单向数据流

React 一直提倡的是单向数据流，他称之为 onChange/setState()模式。

      不过由于我们一般都会用 Vuex 以及 Redux 等单向数据流的状态管理框架，因此很多时候我们感受不到这一点的区别了。

3.模板渲染方式的不同

在表层上，模板的语法不同

React 是通过 JSX 渲染模板

而 Vue 是通过一种拓展的 HTML 语法进行渲染

在深层上，模板的原理不同，这才是他们的本质区别：

React 是在组件 JS 代码中，通过原生 JS 实现模板中的常见语法，比如插值，条件，循环等，都是通过 JS 语法实现的

Vue 是在和组件 JS 代码分离的单独的模板中，通过指令来实现的，比如条件语句就需要 v-if 来实现

      对这一点，我个人比较喜欢 React 的做法，因为他更加纯粹更加原生，而 Vue 的做法显得有些独特，会把 HTML 弄得很乱。举个例子，说明 React 的好处：react 中 render 函数是支持闭包特性的，所以我们 import 的组件在 render 中可以直接调用。但是在 Vue 中，由于模板中使用的数据都必须挂在 this 上进行一次中转，所以我们 import 一个组件完了之后，还需要在 components 中再声明下，这样显然是很奇怪但又不得不这样的做法。

# #### 2、前端模块化开发中webpack、npm、node、nodejs之间的关系

<https://blog.csdn.net/AngelLover2017/article/details/84801673>

what is webpack?

Webpack 是一个前端资源加载/打包工具。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。

webpack是一个工具，这个工具可以帮你处理好各个包/模块之间的依赖关系（modules with dependencies），并将这些复杂依赖关系的静态文件打包成一个或很少的静态文件，提供给浏览器访问使用；除此之外，webpack因为可以提高兼容性，可以将一些浏览器尚不支持的新特性转换为可以支持格式，进而减少由新特性带来的浏览器的兼容性问题

我们有个概念，webpack是一个打包工具，可以帮你把你的项目这里的项目其实就是指通过模块化开发的项目 打包为简洁版的浏览器可识别的静态资源

what is npm?

介绍了webpack，我们可能会疑问，我的JS，CSS，HTML文件分开写，挺好的呀，为什么要使用webpack工具，进行复杂的各项配置。在传统前端开发模式下，我们确实是按照JS/CSS/HTML文件分开写的模式就可以，但是随着前端的发展，社区的壮大，各种前端的库和框架层出不穷，我们项目中可能会使用很多额外的库，如何有效管理这些引入的库文件是一个大问题，而且我们知道基于在HTML中使用<script>引入的方式，有两个弊端，一个是会重复引入，二是当库文件数量很多时管理成为一个大难题。

面对这样的局面，为了简化开发的复杂度，前端社区涌现了很多实践方法。模块化就是其中一项成功实践，而npm就是这样在社区 其实就是node社区中产生的

npm 为你和你的团队打开了连接整个 JavaScript 天才世界的一扇大门。它是世界上最大的软件注册表，每星期大约有 30 亿次的下载量，包含超过 600000 个 包（package） （即，代码模块）。来自各大洲的开源软件开发者使用 npm 互相分享和借鉴。包的结构使您能够轻松跟踪依赖项和版本。

npm 由三个独立的部分组成：

网站

注册表（registry）

命令行工具 (CLI)

网站 是开发者查找包（package）、设置参数以及管理 npm 使用体验的主要途径。

注册表 是一个巨大的数据库，保存了每个包（package）的信息。

CLI 通过命令行或终端运行。开发者通过 CLI 与 npm 打交道。

这是npm的官方网站给出的介绍，一般来说提起npm有两个含义，一个是说npm官方网站，一个就是说npm包管理工具。npm社区或官网是一个巨大的Node生态系统，社区成员可以随意发布和安装npm生态中的包，也就是不用在重复造轮子了，别人造好了，你直接安装到你的项目中就可以使用，但是因为前面说了，当包引入数量很多时管理就成为了一个问题，这个就是npm为开发者行了方便之处，npm已经为你做好了依赖和版本的控制，也就是说使用npm可以让你从繁杂的依赖安装和版本冲突中解脱出来，进而关注你的业务而不是库的管理。

而webpack就是将你从npm中安装的包打包成更小的浏览器可读的静态资源，这里需要注意的是，webpack只是一个前端的打包工具，打包的是静态资源，和后台没有关系，虽然webpack依赖于node环境

what is node or nodejs？

其实node和nodejs两个概念没有太大差别，我个人认为唯一的区别就是，人们说起node的时候语境更多的是再说node环境，而说nodejs时更多的是在说node是一门可以提供后端能力的技术。本质上来说，node就是nodejs，nodejs就是node

简单的说 Node.js 就是运行在服务端的 JavaScript。

Node.js 是一个基于Chrome JavaScript 运行时建立的一个平台。

Node.js是一个事件驱动I/O服务端JavaScript环境，基于Google的V8引擎，V8引擎执行Javascript的速度非常快，性能非常好。

node环境基于V8引擎提供了一种可以让JS代码跑在后端的能力，这就是node。其实这里的node本身和我们这篇讲的前端模块化没啥关系。但是因为npm是产生与node社区，node中也是通过npm来加载模块的，所以有必要说一下他们之间的关系。

npm 是 Node.js 官方提供的包管理工具，他已经成了 Node.js 包的标准发布平台，用于 Node.js 包的发布、传播、依赖控制

webpack npm node之间关系？

webpack是npm生态中的一个模块，我们可以通过全局安装webpack来使用webpack对项目进行打包；

webpack的运行依赖于node的环境，没有node是不能打包的，但是webpack打包后的项目本身只是前端静态资源和后台没有关系，也就是说不依赖与node，只要有后台能力的都可以部署项目

npm是于Node社区中产生的，是nodejs的官方包管理工具，当你下载安装好node的时候，npm cli就自动安装好了

正是因为npm的包管理，使得项目可以模块化的开发，而模块化的开发带来的这些改进确实大大的提高了我们的开发效率，但是利用它们开发的文件往往需要进行额外的处理才能让浏览器识别，而手动处理又是非常繁琐的，这就是webpack工具存在的意义。

# #### 3、react和vue区别

相同点

· 都支持服务器端渲染

· 都有Virtual DOM,组件化开发,通过props参数进行父子组件数据的传递,都实现webComponent规范

· 数据驱动视图

· 都有支持native的方案,React的React native,Vue的weex

不同点

· React严格上只针对MVC的view层,Vue则是MVVM模式

· virtual DOM不一样,vue会跟踪每一个组件的依赖关系,不需要重新渲染整个组件树.而对于React而言,每当应用的状态被改变时,全部组件都会重新渲染,所以react中会需要shouldComponentUpdate这个生命周期函数方法来进行控制

· 组件写法不一样, React推荐的做法是 JSX + inline style, 也就是把HTML和CSS全都写进JavaScript了,即'all in js'; Vue推荐的做法是webpack+vue-loader的单文件组件格式,即html,css,jd写在同一个文件;

· 数据绑定: vue实现了数据的双向绑定,react数据流动是单向的

· state对象在react应用中不可变的,需要使用setState方法更新状态;在vue中,state对象不是必须的,数据由data属性在vue对象中管理

# #### 4、mvvm mvc是什么区别

MVVM即Model-View-ViewModel。  
它是将“数据模型数据双向绑定”的思想作为核心，因此在View和Model之间没有联系，通过ViewModel进行交互，而且Model和ViewModel之间的交互是双向的，因此视图的数据的变化会同时修改数据源，而数据源数据的变化也会立即反应到View上。  
Vue是以数据为驱动的，Vue自身将DOM和数据进行绑定，一旦创建绑定，DOM和数据将保持同步，每当数据发生变化，DOM会跟着变化。 ViewModel是Vue的核心，它是Vue的一个实例。Vue实例时作用域某个HTML元素上的，这个HTML元素可以是body，也可以是某个id所指代的元素。 DOM Listeners和Data Bindings是实现双向绑定的关键。DOM Listeners监听页面所有View层DOM元素的变化，当发生变化，Model层的数据随之变化；Data Bindings监听Model层的数据，当数据发生变化，View层的DOM元素随之变化。

MVC即Model-View-Controller  
MVC是比较直观的架构模式，用户操作->View（负责接收用户的输入操作）->Controller（业务逻辑处理）->Model（数据持久化）->View（将结果反馈给View）。

# #### 5、:bootstrap响应式实现的原理

百分比布局+媒体查询

# #### 6、vue中的通信都有哪些？

父子通信，兄弟通信，vuex。

父子通信都怎么传值的？  
父传子用props接收参数进行传值，在子组件中定义一个props，来接受父组件传过来的值，并在子组件中使用。

7、那如果想传参严谨一些怎么实现呢？比如只接受某些类型的参数

我们在props中可以定义参数的类型，比如我就固定接收一个数字类型，我可以定一个变量为num，设置一些类型。

props:{

num:Number

}

4、嗯，那子传父呢？  
子传父，用一个回调函数，通过this.$emit()的方法。它有两个参数，第一个是父组件中监听的方法名，第二个是方法。父组件监听这个方法名即可。

5、嗯，那还有没有其他方法可以实现的呢？  
vue中新增了两个新增事件，一个是v-bind="$attrs"，一个是v-on='$listeners'。当我们在创建一些组建的时候，这两个事件可以帮助我们监听到子组件的改变，从而实现了父组件的变化。

6、那这两个方法分别是什么用途呢？  
v-bind="$attrs" 它包含了父作用域中不作为 prop 被识别的特性绑定；  
v-on='$listeners' 它包含了父作用域中的事件监听器，可以触发父组件中的事件。

# #### 8、vuex是什么，说说其中都有什么?

vuex是一个专为vue.js应用程序开发的状态管理器，它采用集中式存储管理应用的所有组件的状态，并且以相应的规则保证状态以一种可以预测的方式发生变化。

state: vuex使用单一状态树，用一个对象就包含了全部的应用层级状态

mutation: 更改vuex中state的状态的唯一方法就是提交mutation

action: action提交的是mutation，而不是直接变更状态，action可以包含任意异步操作

getter: 相当于vue中的computed计算属性

Module

vueX是处理复杂的父子通信时，需要用到的技术。  
state中可以定义一些常量，getters中写触发state改变的方法，通常用到getters比较少。  
还有两个可以存放方法的地方是actions和mutations，一个是异步，一个是同步，我们一般都用mutations存放一些方法，在页面中通过this.$store.commit()方法来触发mutations，从而改变state中的值。

# #### 9、封装组件（或者问如果页面有相同的功能块怎么办？

很简单啊，在components文件夹中创建一个 .vue 组件，我们标准命名首字母大写。写上公用的组件和样式，内部的值都用动态绑定。并且创建props接收方法即可。不过现在的UI库也都特别多，我们一般都直接使用UI库就可以了，简单还方便，有不同的地方，简单封装一下就可以了。

**#### 8、封装 vue 组件的过程？**

首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：效率低、难维护、复用性等问题。

然后，使用Vue.extend方法创建一个组件，然后使用Vue.component方法注册组件。子组件需要数据，可以在props中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用emit方法。

# #### 10、路由封装

我们都是使用公司给的框架，vue-cli在创建路由时，会直接引入路由组件的，我们只需要在 views文件夹下创建自己的页面，在路由 path 中定义页面路径；写个 name 名称；在 component 中可以用个箭头函数写上页面的路径即可。并且我们一般会定义一个重定向 redirect 到一个指定页。

# #### 11、vue环境配置（虽然说公司都是配置好的，不过也要简单的说说，[点此跳转vue框架配置](https://yq.aliyun.com/articles/710824?spm=a2c4e.11155435.0.0.28923312l5g18S" \t "https://yq.aliyun.com/articles/_blank)） 目前我们用的都是vue3.0的配置，下载完框架后，自建 vue.config.js 文件，其中可以引入node中的 path 来打包一些样式文件等，是他们压缩到一个盘下。vue3.0提供了一个 publicPath属性，之前2.0叫 baseURL。在这我们可以进行测试环境或上线环境的配置，可以通过 process.env.NODE\_ENV === 'production' ? '/自己项目的路径path/' : '/' 进行三目判断，我们还可以配置各种 loader ，以及配置 proxy 来解决跨域请求。

# #### 12、vue的生命周期

常用的有8个（生命周期不仅仅是8个），都是成对出现。分别是：  
beforeCreate阶段、created阶段；  
beforeMount阶段、mounted阶段；  
beforeUpdate阶段、updated阶段；  
beforeDestroy阶段、destroyed阶段；  
分别为：创建前/后，载入前/后，更新前/后，销毁前/后。

# #### 13、那实例是在哪个阶段创建完的？

在mounted阶段实例创建完成。

# #### 16、hash和history模式

为了构建SPA，vue引入了前端路由系统vue-router。

vue-route有两种模式：history模式和hash模式。  
直观上的区别就是差一个“#”，hash中的地址是写在“#”后边的；但history没有。  
使用hash模式，hash之前的内容会包含在请求中；

history模式URL就要和后端进行一致，所以要改为history也需要后端的配合，否则会报错。

所以hash模式在每次刷新页面时是直接更改“#”后的东西，history每次刷新会重新向后端请求整个网址，也就是重新请求服务器。如果后端没有及时响应，就会报错404！。

history的好处是可以进行修改历史记录，并且不会立刻像后端发起请求。不过如果对于项目没有硬性标准要求，我们可以直接使用hash模式开发。

# #### 14、实例初始化是在哪个阶段？

在created阶段实例初始化，但还未完成。

# #### 18、vue-router

vue-router是vuex.js官方的路由管理器，它和vue.js的核心深度集成，让构建页面应用变得易如反掌

<router-link> 组件 支持用户在具有路由功能的应用中 (点击) 导航。 通过 to 属性指定目标地址

<router-view> 组件 是一个 functional 组件，渲染路径匹配到的视图组件。

<keep-alive> 组件 是一个用来缓存的组件

router.beforeEach

router.afterEach

to: Route: 即将要进入的目标 路由对象

from: Route: 当前导航正要离开的路由

next: Function: 一定要调用该方法来 resolve 这个钩子。执行效果依赖 next 方法的调用参数。

# #### 19、Vue通信

1.props和$emit

2.中央事件总线 EventBus(基本不用)

3.vuex(官方推荐状态管理器)

4.$parent和$children

当然还有一些其他办法，但基本不常用，或者用起来太复杂来。 介绍来通信的方式，还可以扩展说一下使用场景，如何使用，注意事项之类的。

# #### 20、在工作遇到那些问题，解决方法是什么

经常遇到的问题就是Cannot read property ‘prototype’ of undefined

解决办法通过浏览器报错提示代码定位问题，解决问题

Vue项目中遇到视图不更新，方法不执行，埋点不触发等问题

一般解决方案查看浏览器报错，查看代码运行到那个阶段未之行结束，阅读源码以及相关文档等

# #### 21、Vue原理

Vue是采用数据劫持配合发布者-订阅者模式，通过Object.defineProperty来()来劫持各个属性的getter和setter。在数据发生变化的时候，发布消息给依赖收集器，去通知观察者，做出对应的回调函数去更新视图。

具体就是：

MVVM作为绑定的入口，整合Observe观察者,Compil编译和Watcher监听三者，通过Observe来监听model的变化

通过Compil来解析编译模版指令，最终利用Watcher搭起Observe和Compil之前的通信桥梁

从而达到数据变化 => 更新视图，视图交互变化(input) => 数据model变更的双向绑定效果。

# #### 22、Vue路由守卫有哪些，怎么设置，使用场景等

常用的两个路由守卫：router.beforeEach 和 router.afterEach

每个守卫方法接收三个参数：

to: Route: 即将要进入的目标 路由对象

from: Route: 当前导航正要离开的路由

next: Function: 一定要调用该方法来 resolve 这个钩子。

在项目中，一般在beforeEach这个钩子函数中进行路由跳转的一些信息判断。

判断是否登录，是否拿到对应的路由权限等等。

# #### 23、v-show与v-if的区别

条件渲染指令，与v-if不同的是，无论v-show的值为true或false，元素都会存在于HTML代码中；而只有当v-if的值为true，元素才会存在于HTML代码中。v-show指令只是设置了元素CSS的style值

· vue中显隐方法常用两种，v-show和v-if，但这两种是有区别的。

· 实现本质方法区别

* vue-show本质就是标签display设置为none，控制隐藏
* vue-if是动态的向DOM树内添加或者删除DOM元素

· 编译的区别

* v-show其实就是在控制css
* v-if切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件

· 编译的条件

* v-show都会编译，初始值为false，只是将display设为none，但它也编译了
* v-if初始值为false，就不会编译了

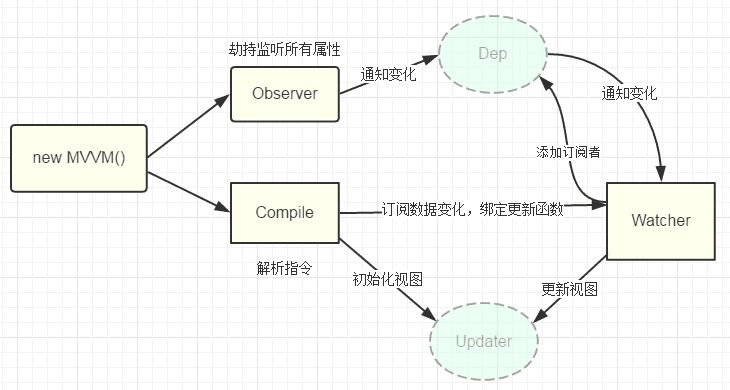
· 性能

* v-show只编译一次，后面其实就是控制css，而v-if不停的销毁和创建，故v-show性能更好一点。

# #### **24、如何让css只在当前组件中起作用**

在每一个vue组件中都可以定义各自的css，js，如果希望组件内写的css只对当前组件起作用，只需要在style中写入scoped，即：   
<style scoped></style>   
四、指令keep-alive   
在vue-router写着keep-alive，keep-alive的含义： 如果把切换出去的组件保留在内存中，可以保留它的状态或避免重新渲染。为此可以添加一个keep-alive指令   
<component :is='curremtView' keep-alive></component>。

# #### 27、Vue的双向数据绑定原理是什么？vue data是怎么实现



vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

具体步骤：

第一步：需要observe的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter和getter  
这样的话，给这个对象的某个值赋值，就会触发setter，那么就能监听到了数据变化

第二步：compile解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

第三步：Watcher订阅者是Observer和Compile之间通信的桥梁，主要做的事情是:  
1、在自身实例化时往属性订阅器(dep)里面添加自己  
2、自身必须有一个update()方法  
3、待属性变动dep.notice()通知时，能调用自身的update()方法，并触发Compile中绑定的回调，则功成身退。

# #### 28、vue实现父子组件通信，以及非父子组件通信？

1、父子:  
使用props：  
props监听父组件传递过来的信息  
传递过来后，可直接引用，就如已经传递过来数据塞到data  
使用$refs:  
this.$refs.XXX  
2、子父:  
自定义事件:  
父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件  
子组件$emit()触发，父组件$on()监听  
使用$parent：  
this.$parent.XXX  
3、非父子:  
简单场景用bus，复杂场景用vuex

[IMG_256](https://www.cnblogs.com/wu-web/p/javascript:void(0);)

<div id="app4"><display></display><increment></increment></div><script>var bus = new Vue();

Vue.component('increment', {

template: `<button @click="add">+</button>`,data: function () {return {count: 0}

},

methods: {

add: function () {

bus.$emit('inc', this.count+=1)

}

}

});

Vue.component('display', {

template: `<span>Clicked: <mark>{{c}}</mark> times</span>`,

data: function () {return {c: 0}

},

created: function () {var self=this;// bus.$on('inc', function (num) {

// self.c = num

// });bus.$on('inc', (num) =>this.c = num

);

}

});

vm = new Vue({

el: "#app4",

})</script>

# #### 29、vue响应式原理？

工作原理是当我们把一个普通的JavaScript对象传给Vue 实例的data选项的时候，Vue会遍历此对象的所有属性，并使用Object.definePropert把这些属性全部转化为getter/setter。(Object.definePropert是仅ES5支持，且没有垫片脚本的特性，因此Vue不支持IE8及更低版本浏览器。）用户看不到getter/setter，但是在内部它们让Vue追踪依赖，在属性被访问和修改时通知变化。每个组件实例都有相应的watcher实例对象，它会在组件渲染的过程中把属性记录为依赖，之后当依赖项的setter被调用时，会通知watcher重新计算，从而使它关联的组件得以更新。

# #### 30、vue-router实现原理？ 单页面应用(SPA)的核心之一是: 更新视图而不重新请求页面, 实现这一点主要是两种方式: 1.Hash: 通过改变hash值 2.History: 利用history对象新特性

而在vue-router中，它提供mode参数来决定采用哪一种方式，选择流程如下:  
默认Hash-->如果浏览器支持History新特性改用History-->如果不在浏览器环境则使用abstract

基本方法分析:  
Hash  
1.push()  
功能: 设置新的路由添加历史记录并更新视图,常用情况是直接点击切换视图  
调用流程:  
1 $router.push() //显式调用方法  
2 HashHistory.push() //根据hash模式调用,设置hash并添加到浏览器历史记录（window.location.hash= XXX）  
3 History.transitionTo() //开始更新  
4 History.updateRoute() //更新路由  
5 {app.\_route= route}   
6 vm.render() //更新视图

2.replace  
功能: 替换当前路由并更新视图，常用情况是地址栏直接输入新地址  
流程与push基本一致  
但流程2变为替换当前hash （window.location.replace= XXX）不懂此方法的可见: http://www.w3school.com.cn/jsref/met\_loc\_replace.asp

3.监听地址栏变化  
在setupListeners中监听hash变化(window.onhashchange)并调用replace

History  
1.push  
与hash模式类似，只是将window.hash改为history.pushState  
2.replace  
与hash模式类似，只是将window.replace改为history.replaceState  
3.监听地址变化  
在HTML5History的构造函数中监听popState（window.onpopstate）

两种模式对比  
History模式的优点:  
1.History模式的地址栏更美观。。。  
2.History模式的pushState、replaceState参数中的新URL可为同源的任意URL（可为不同的html文件），而hash只能是同一文档  
3.History模式的pushState、replaceState参数中的state可为js对象，能携带更多数据  
4.History模式的pushState、replaceState参数中的title能携带字符串数据（当然，部分浏览器，例如firefox不支持title，一般title设为null，不建议使用）  
缺点:  
对于单页面应用来说，理想的场景是仅仅在进入应用时加载页面（例如index.html）,后续的网络操作靠ajax完成，  
而不会重新请求页面。  
但当用户直接在用户栏输入地址时则会重新请求，当地址带有参数时两者情况不一样  
Hash 例如: xxx.com/#/id=5 HTTP请求不会包含后面的hash值，所以请求的仍然是 xxx.com,没有问题  
History 例如: xxx.com/id=5 这时请求的便是xxx.com/id=5，如后端没有配置对应id=XXX的路由处理，则会返回404错误。

# #### 31、怎么定义vue-router的动态路由？怎么获取传过来的动态参数？

在router目录下的index.js文件中，对path属性加上/:id。 使用router对象的params.id

# #### 32、vue-router有哪几种导航钩子？

三种，一种是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。第二种：组件内的钩子；第三种：单独路由独享组件

# #### 34、导航钩子有哪些？它们有哪些参数？

导航钩子有：a/全局钩子和组件内独享的钩子。b/beforeRouteEnter、afterEnter、beforeRouterUpdate、beforeRouteLeave  
参数：有to（去的那个路由）、from（离开的路由）、next（一定要用这个函数才能去到下一个路由，如果不用就拦截）最常用就这几种

# #### 35、scss是什么？安装步骤？特性？

预处理css，把css当前函数编写，定义变量,嵌套。

使用步骤：

第一步：用npm 下三个loader（sass-loader、css-loader、node-sass）

第二步：在build目录找到webpack.base.config.js，在那个extends属性中加一个拓展.scss

第三步：还是在同一个文件，配置一个module属性

第四步：然后在组件的style标签加上lang属性 ，例如：lang=”scss”

先装css-loader、node-loader、sass-loader等加载器模块，在webpack-base.config.js配置文件中加多一个拓展:extenstion，再加多一个模块：module里面test、loader  
有哪几大特性:  
1、可以用变量，例如（$变量名称=值）；  
2、可以用混合器，例如（）  
3、可以嵌套

# #### 36、vuex是？怎么用？哪种功能场景使用它？

vue框架中状态管理。有五种属性，分别是 State、 Getter、Mutation 、Action、 Module  
在main.js引入store，注入。新建了一个目录store，….. export 。  
应用级的状态集中放在store中； 改变状态的方式是提交mutations，这是个同步的事物； 异步逻辑应该封装在action中。  
场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

# #### 38、vuex的State特性是？

一、Vuex就是一个仓库，仓库里面放了很多对象。其中state就是数据源存放地，对应于与一般Vue对象里面的data  
二、state里面存放的数据是响应式的，Vue组件从store中读取数据，若是store中的数据发生改变，依赖这个数据的组件也会发生更新  
三、它通过mapState把全局的 state 和 getters 映射到当前组件的 computed 计算属性中

# #### 39、vuex的Getter特性是？

一、getters 可以对State进行计算操作，它就是Store的计算属性  
二、 虽然在组件内也可以做计算属性，但是getters 可以在多组件之间复用  
三、 如果一个状态只在一个组件内使用，是可以不用getters

**#### 40、vuex的Mutation特性是？**

一、Action 类似于 mutation，不同在于：

二、Action 提交的是 mutation，而不是直接变更状态。

三、Action 可以包含任意异步操作

**#### 41、Vue.js中ajax请求代码应该写在组件的methods中还是vuex的actions中？**

一、如果请求来的数据是不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入vuex 的state里。

二、如果被其他地方复用，这个很大几率上是需要的，如果需要，请将请求放入action里，方便复用，并包装成promise返回，在调用处用async await处理返回的数据。如果不要复用这个请求，那么直接写在vue文件里很方便。

**#### 42、不用Vuex会带来什么问题？**

一、可维护性会下降，你要想修改数据，你得维护三个地方

二、可读性会下降，因为一个组件里的数据，你根本就看不出来是从哪来的

三、增加耦合，大量的上传派发，会让耦合性大大的增加，本来Vue用Component就是为了减少耦合，现在这么用，和组件化的初衷相背

**#### 43、vue生命周期的理解？**

Vue 实例从创建到销毁的过程，就是生命周期。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。

总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后： 在beforeCreated阶段，vue实例的挂载元素$el和数据对象data都为undefined，还未初始化。在created阶段，vue实例的数据对象data有了，$el还没有。

载入前/后：在beforeMount阶段，vue实例的$el和data都初始化了，但还是挂载之前为虚拟的dom节点，data.message还未替换。在mounted阶段，vue实例挂载完成，data.message成功渲染。

更新前/后：当data变化时，会触发beforeUpdate和updated方法。

销毁前/后：在执行destroy方法后，对data的改变不会再触发周期函数，说明此时vue实例已经解除了事件监听以及和dom的绑定，但是dom结构依然存在

**#### 44、vue生命周期的作用是什么？**

它的生命周期中有多个事件钩子，让我们在控制整个Vue实例的过程时更容易形成好的逻辑。

**#### 45、第一次页面加载会触发哪几个钩子？**

第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

**#### 46、DOM 渲染在 哪个周期中就已经完成？**

DOM 渲染在 mounted 中就已经完成了。

**#### 47、每个周期具体适合哪些场景？**

答：生命周期钩子的一些使用方法： beforecreate : 可以在这加个loading事件，在加载实例时触发 created : 初始化完成时的事件写在这里，如在这结束loading事件，异步请求也适宜在这里调用 mounted : 挂载元素，获取到DOM节点 updated : 如果对数据统一处理，在这里写上相应函数 beforeDestroy : 可以做一个确认停止事件的确认框 nextTick : 更新数据后立即操作dom

**#### 49、vue-loader是？用途有哪些？**

解析.vue文件的一个加载器，跟template/js/style转换成js模块。

用途：js可以写es6、style样式可以scss或less、template可以加jade等

**#### 50、你对Vue.js的template编译的理解？**

简而言之，就是先转化成AST树，再得到的render函数返回VNode（Vue的虚拟DOM节点）

详情步骤：

首先，通过compile编译器把template编译成AST语法树（abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile是createCompiler的返回值，createCompiler是用以创建编译器的。另外compile还负责合并option。

然后，AST会经过generate（将AST语法树转化成render funtion字符串的过程）得到render函数，render的返回值是VNode，VNode是Vue的虚拟DOM节点，里面有（标签名、子节点、文本等等）

**#### 53、Vue提供了几种脚手架模板**

webpack- 一个全面的webpack+vue-loader的模板，功能包括热加载，linting,检测和CSS扩展。

webpack-simple- 一个简单webpack+vue-loader的模板，不包含其他功能，让你快速的搭建vue的开发环境。

browserify- 一个全面的Browserify+vueify 的模板，功能包括热加载，linting,单元检测。

browserify-simple- 一个简单Browserify+vueify的模板，不包含其他功能，让你快速的搭建vue的开发环境。

simple- 一个最简单的单页应用模板。

pwa- 一个集成pwa环境的webapp的模板

# #### 54、常见的几种MVVM的实现方式

发布者-订阅者模式（backbone.js）

脏值检查（angular.js）

数据劫持（vue.js）

**#### 54、Vue组件实现轮播**

<https://www.jb51.net/article/135812.htm>

**#### 55、vue实现菜单权限控制**

<https://www.cnblogs.com/ssh-007/archive/2019/01/20/10295575.html>

**#### 56、vue全家桶面试**

 vue：主要Vue

​ vue-router：关于路由方面的配置

​ vuex：数据共享和缓存用

​ vue-resource：于后台交互用（现在改为axios 但是axios不是Vue里面的东西）

​ vue-cli：快速创建项目的脚手架

<https://www.jianshu.com/p/c71fde6f2d34>

# **#### 577、**vue的npm run dev和npm run build

关于vue的npm run dev和npm run build

├─build

│ ├─build.js

│ ├─check-versions.js

│ ├─dev-client.js

│ ├─dev-server.js

│ ├─utils.js

│ ├─vue-loader.conf.js

│ ├─webpack.base.conf.js

│ ├─webpack.dev.conf.js

│ ├─webpack.prod.conf.js

│ └─webpack.test.conf.js

├─config

│ ├─dev.env.js

│ ├─index.js

│ ├─prod.env.js

│ └─test.env.js

├─...

└─package.json

以上是关于bulid与run的所有文件

指令分析

package.json里面

"dev": "node build/dev-server.js",

"build": "node build/build.js",

意思：运行”npm run dev”的时候执行的是build/dev-server.js文件，

运行”npm run build”的时候执行的是build/build.js文件。

build文件夹分析

build/dev-server.js

npm run dev 执行的文件build/dev-server.js文件，执行了：

检查node和npm的版本

引入相关插件和配置

创建express服务器和webpack编译器

配置开发中间件（webpack-dev-middleware）和热重载中间件（webpack-hot-middleware）

挂载代理服务和中间件

配置静态资源

启动服务器监听特定端口（8080）

自动打开浏览器并打开特定网址（localhost:8080）

说明： express服务器提供静态文件服务，不过它还使用了http-proxy-middleware，一个http请求代理的中间件。前端开发过程中需要使用到后台的API的话，可以通过配置proxyTable来将相应的后台请求代理到专用的API服务器。

build/webpack.base.conf.js

dev-server依赖的webpack配置是webpack.dev.conf.js文件，

测试环境下使用的是webpack.prod.conf.js

webpack.dev.conf.js中又引用了webpack.base.conf.js

webpack.base.conf.js主要完成了下面这些事情：

配置webpack编译入口

配置webpack输出路径和命名规则

配置模块resolve规则

配置不同类型模块的处理规则

这个配置里面只配置了.js、.vue、图片、字体等几类文件的处理规则，如果需要处理其他文件可以在module.rules里面配置。

build/webpack.dev.conf.js

在webpack.base.conf的基础上增加完善了开发环境下面的配置，主要包括下面几件事情：

将hot-reload相关的代码添加到entry chunks

合并基础的webpack配置

使用styleLoaders

配置Source Maps

配置webpack插件

build/check-versions.js和build/dev-client.js

最后是build文件夹下面两个比较简单的文件，

dev-client.js似乎没有使用到，代码也比较简单，这里不多讲。

check-version.js完成对node和npm的版本检测

build/utils.js和build/vue-loader.conf.js

webpack配置文件中使用到了utils.js和vue-loader.conf.js这两个文件，utils主要完成下面3件事：

配置静态资源路径

生成cssLoaders用于加载.vue文件中的样式

生成styleLoaders用于加载不在.vue文件中的单独存在的样式文件

vue-loader.conf则只配置了css加载器以及编译css之后自动添加前缀。

**build/build.js**

构建环境下的配置，

build.js主要完成下面几件事：

loading动画

删除创建目标文件夹

webpack编译

输出信息

build/webpack.prod.conf.js

构建的时候用到的webpack配置来自webpack.prod.conf.js，该配置同样是在webpack.base.conf基础上的进一步完善。主要完成下面几件事情：

合并基础的webpack配置

使用styleLoaders

配置webpack的输出

配置webpack插件

gzip模式下的webpack插件配置

webpack-bundle分析

说明： webpack插件里面多了丑化压缩代码以及抽离css文件等插件。

config文件夹分析

config/index.js

config文件夹下最主要的文件就是index.js了，

在这里面描述了开发和构建两种环境下的配置，前面的build文件夹下也有不少文件引用了index.js里面的配置。

config/dev.env.js、config/prod.env.js和config/test.env.js

这三个文件就简单设置了环境变量而已，没什么特别的。

**#### 58、web项目如何部署到服务器？**

打包，上传到服务器，验证

**#### 59、钩子函数**

指令定义函数提供了几个钩子函数（可选）：

bind: 只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。

inserted: 被绑定元素插入父节点时调用（父节点存在即可调用，不必存在于 document 中）。

update: 被绑定元素所在的模板更新时调用，而不论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新（详细的钩子函数参数见下）。

componentUpdated: 被绑定元素所在模板完成一次更新周期时调用。

unbind: 只调用一次， 指令与元素解绑时调用。

钩子函数的参数有：

el: 指令所绑定的元素，可以用来直接操作 DOM 。

binding: 一个对象，包含以下属性：

name: 指令名，不包括 v- 前缀。

value: 指令的绑定值， 例如： v-my-directive="1 + 1", value 的值是 2。

oldValue: 指令绑定的前一个值，仅在 update 和 componentUpdated 钩子中可用。无论值是否改变都可用。

expression: 绑定值的表达式或变量名。 例如 v-my-directive="1 + 1" ， expression 的值是 "1 + 1"。

arg: 传给指令的参数。例如 v-my-directive:foo， arg 的值是 "foo"。

modifiers: 一个包含修饰符的对象。 例如： v-my-directive.foo.bar, 修饰符对象 modifiers 的值是 { foo:

true, bar: true }。

vnode: Vue 编译生成的虚拟节点。

oldVnode: 上一个虚拟节点，仅在 update 和 componentUpdated 钩子中可用。

**#### 61、vue路由守卫(导航守卫)**

官方文档

导航守卫其实也是路由守卫，也可以是路由拦截，我们可以通过路由拦截，来判断用户是否登录，该页面用户是否有权限浏览，需要结合meta来实现

vue中路由守卫一共有三种，一个全局路由守卫，一个是组件内路由守卫，一个是router独享守卫。

所谓的路由守卫可以简单的理解为一座房子的门口的保安，想要进入这个房子就必须通过保安的检查，要告诉路由守卫你从哪里来？总不能随便陌生人就给放进去？要到哪里去？然后保安再告诉你下一步该怎么做？如果你的确是这个房子主人允许进入的人，那就让你进入，否则就要打电话给房子主人，跟房主商量(登录注册)，给你权限。

// 通过这个匹配判断是否有该权限或要求，这个一般作为页面权限设置，比如哪些页面需要登录才能进入，哪些不需要

to.matched.some(res => res.meta.requireAuth)

一、全局路由守卫

所谓全局路由守卫，就是小区大门，整个小区就这一个大门，你想要进入其中任何一个房子，都需要经过这个大门的检查

全局路由守卫有个两个：一个是全局前置守卫，一个是全局后置守卫

router.beforeEach((to, from, next) => {

console.log(to) => // 到哪个页面去？

console.log(from) => // 从哪个页面来？

next() => // 一个回调函数

}

router.afterEach（to，from） = {}

next()：回调函数参数配置

next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址

next('/') 或者 next({ path: '/' }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: 'home' 之类的选项以及任何用在 router-link 的 to prop 或 router.push 中的选项

二、组件路由守卫

// 跟methods: {}等同级别书写，组件路由守卫是写在每个单独的vue文件里面的路由守卫

beforeRouteEnter (to, from, next) {

// 注意，在路由进入之前，组件实例还未渲染，所以无法获取this实例，只能通过vm来访问组件实例

next(vm => {})

}

beforeRouteUpdate (to, from, next) {

// 同一页面，刷新不同数据时调用，

}

beforeRouteLeave (to, from, next) {

// 离开当前路由页面时调用

}

三、路由独享守卫

路由独享守卫是在路由配置页面单独给路由配置的一个守卫

export default new VueRouter({

routes: [

{

path: '/',

name: 'home',

component: 'Home',

beforeEnter: (to, from, next) => {

// ...

}

}

]

})

**#### 62、vue-router如何响应路由参数变化？**

问题：当使用路由参数时，例如从 /content?id=1 到 content?id=2，此时原来的组件实例会被复用。这也意味着组件的生命周期钩子不会再被调用，此时vue应该如何响应路由参数 的变化？

复用组件时，想对路由参数的变化作出响应的话， 可以watch (监测变化) $route 对象

const User = {

template: '...',

watch: {

'$route' (to, from) {

// 对路由变化作出响应...

}

}

}

1

或者使用 2.2 中引入的 beforeRouteUpdate 守卫：

const User = {

template: '...',

beforeRouteUpdate (to, from, next) {

// react to route changes...

// don't forget to call next()

}

}

**#### 63、vue-router （路由管理器）导航解析流程**

1、导航被触发。

2、在失活的组件里调用离开守卫。

3、调用全局的 beforeEach守卫。

4、在重用的组件里调用 beforeRouteUpdate守卫 (2.2+)。

5、在路由配置里调用beforeEnter。

6、解析异步路由组件。

7、在被激活的组件里调用beforeRouteEnter。

8、调用全局的 beforeResole 守卫 (2.5+)。

9、导航被确认。

10、调用全局的afterEach 钩子。

11、触发 DOM 更新。

12、用创建好的实例调用beforeRouteEnter 守卫中传给 next 的回调函数。

**#### 64、vue-router有哪几种导航钩子（ 导航守卫 ）？**

1、全局守卫： router.beforeEach

2、全局解析守卫： router.beforeResolve

3、全局后置钩子： router.afterEach

4、路由独享的守卫： beforeEnter

5、组件内的守卫： beforeRouteEnter、beforeRouteUpdate (2.2 新增)、beforeRouteLeave

导航表示路由正在发生改变，vue-router 提供的导航守卫主要用来:通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中：全局的, 单个路由独享的, 或者组件级的。

注意：参数或查询的改变并不会触发进入/离开的导航守卫。 你可以通过 观察 $route 对象 来应对这些变化，或使用 beforeRouteUpdate的组件内守卫。

1、全局守卫：

使用 router.beforeEach 注册一个全局前置守卫：

const router = new VueRouter({ ... })

router.beforeEach((to, from, next) => {

// ...

})

当一个导航触发时，全局前置守卫按照创建顺序调用。守卫是异步解析执行，此时导航在所有守卫 resolve 完之前一直处于等待中。

每个守卫方法接收三个参数：

to: Route:即将要进入的目标 路由对象

from: Route: 当前导航正要离开的路由

next: Function:一定要调用该方法来resolve这个钩子。执行效果依赖 next 方法的调用参数。

next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是confirmed (确认的)。

next(false): 中断当前的导航。如果浏览器的 URL 改变了 (可能是用户手动或者浏览器后退按钮)，那么 URL 地址会重置到 from 路由对应的地址。

next(’/’) 或者 next({ path: ‘/’ }): 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 replace: true、name: ‘home’ 之类的选项以及任何用在router-link的 to prop或 router.push中的选项。

next(error): (2.4.0+) 如果传入 next 的参数是一个 Error 实例，则导航会被终止且该错误会被传递给 router.onError()注册过的回调。

确保要调用 next方法，否则钩子就不会被 resolved

2、全局解析守卫：

2.5.0 新增

在 2.5.0+ 你可以用 router.beforeResolve 注册一个全局守卫。这和 router.beforeEach 类似，区别是：在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被调用。

3、全局后置钩子

你也可以注册全局后置钩子，然而和守卫不同的是，这些钩子不会接受 next 函数也不会改变导航本身：

router.afterEach((to, from) => {

// ...

})

4、路由独享的守卫

你可以在路由配置上直接定义 beforeEnter 守卫：

const router = new VueRouter({

routes: [

{

path: '/foo',

component: Foo,

beforeEnter: (to, from, next) => {

// ...

}

}

]

})

这些守卫与全局前置守卫的方法参数是一样的。

5、组件内的守卫

最后，你可以在路由组件内直接定义以下路由导航守卫：

beforeRouteEnter

beforeRouteUpdate (2.2 新增)

beforeRouteLeave

const Foo = {

template: `...`,

beforeRouteEnter (to, from, next) {

// 在渲染该组件的对应路由被 confirm 前调用

// 不！能！获取组件实例 `this`

// 因为当守卫执行前，组件实例还没被创建

},

//不过，你可以通过传一个回调给 next来访问组件实例。

//在导航被确认的时候执行回调，并且把组件实例作为回调方法的参数。

beforeRouteEnter (to, from, next) {

next(vm => {

// 通过 `vm` 访问组件实例

})

},

beforeRouteUpdate (to, from, next) {

// 在当前路由改变，但是该组件被复用时调用

// 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，

// 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。

// 可以访问组件实例 `this`

},

beforeRouteLeave (to, from, next) {

// 导航离开该组件的对应路由时调用

// 可以访问组件实例 `this`

}

}

注意：beforeRouteEnter 是支持给next 传递回调的唯一守卫。对于beforeRouteUpdate 和 beforeRouteLeave 来说，this 已经可用了，所以不支持传递回调，因为没有必要了:

beforeRouteUpdate (to, from, next) {

// just use `this`

this.name = to.params.name

next()

}

离开守卫beforeRouteLeave：通常用来禁止用户在还未保存修改前突然离开。该导航可以通过 next(false) 来取消：

beforeRouteLeave (to, from , next) {

const answer = window.confirm('Do you really want to leave? you have unsaved changes!')

if (answer) {

next()

} else {

next(false)

}

}

**#### 65、vue生命周期中钩子的作用**

vue 生命周期和生命周期钩子函数：

vue 的生命周期是： vue 实例从创建到销毁，也就是从开始创建、初始化函数、编译模板、挂载 Dom ->渲染、更新 -> 渲染、卸载 等一系列过程。 在这个过程中也会运行一些叫做生命周期钩子的函数，这给了用户在不同阶段添加自己的代码的机会

vue 的每个周期的钩子函数有哪些作用

生命周期的钩子函数总共有11个

beforeCreate : 在实例初始化之后，数据观测（dataobserver）和 event/watcher 事件配置之前被调用。

created ：在实例创建完成后，被立即调用。在这一步，实例已经完后曾以下配置：①.数据观测（data observer）②.属性和方法运算 ③. watch/event 事件回调。然而挂载阶段还没开始，$el 属性目前不可见。

beforeMount ：在挂载之前被调用：相关的render函数首次被调用。

mounted ：el 被新创建的 vm.$el 替换，并挂在到实例去之后 调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

beforeUpdata : 数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM ，比如手动移除已添加的时间监听器，该钩子在服务器端渲染期间不被调用，因为只有初次渲染会在服务器端进行。

updated : 由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。

actived : keep-alive 组件激活时调用。该钩子在服务器端渲染期间不被调用。

deactived : keep-alive 组件停用时调用。该钩子在服务器端渲染期间不被调用。

beforeDestroy : 实例销毁之前调用。在这一步，实例仍然完全可用。该钩子在服务器端渲染期间不被调用。

destroy : vue 实例销毁后调用。调用后，vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

errorCaptured(2.5.0+新增) ：当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数。错误对象，发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止错误继续向上传播。

注意

mounted 、 updated 不会承诺所有的子组件也都一起被挂载。如果你希望得到整个视图都渲染完毕，可以用vm.$nextTick 替换掉 mounted 、 updated ：

updated: function (){

this.$nextTick(function (){

// Code that will run only after the

// entire view has been re-rendere

})

}

http 请求建议在 created 生命周期内发出 vue 生命周期图示：

**#### 66、vue实现菜单权限控制**

<https://www.cnblogs.com/ssh-007/archive/2019/01/20/10295575.html>

# #### 67、active-class是哪个组件的属性？嵌套路由怎么定义？

答：vue-router模块的router-link组件。

只需要我们在 VueRouter 的参数中使用 children 配置，这样就可以很好的实现路由嵌套。

# #### 70、v-model是？怎么用？ vue中标签怎么绑定事件？

Vue中Vue-model原理是什么？v-model是Vue用于表单元素上创建双向数据绑定，它本质是一个语法糖，在单向数据绑定的基础上，增加了监听用户输入事件并更新数据的功能。下面的文章中将给大家说一说v-model的实现原理及其使用方法的介绍。

**v-model实现原理**

**genDirectives**

Vue初始化组件时通过genDirectives(el,state)初始化指令。（这里的el已经通过parseHTML将html结构转换成Vue的AST语法树，state是根据用户定义组件的options新建的CodegenState对象）。

|  |  |
| --- | --- |
| 1  2  3  4 | <div id="test">      请输入：<input type="text" v-model="message"><br/>      你输入的是：<input type="text" v-model="message" disabled >  </div> |

当用户在html页面上写了v-model指令进行数据双向绑定，Vue通过state找到model指令对应的方法model(el,dir,\_warn)，并执行该方法。（这里双!!表示将函数返回结果转换成Boolean类型）

|  |  |
| --- | --- |
| 1  2  3  4 | **var** gen = state.directives[dir.name];  **if** (gen) {        needRuntime = !!gen(el, dir, state.warn);      } |

**model**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | **if** (el.component) {      genComponentModel(el, value, modifiers);  **return** false    } **else** **if** (tag === 'select') {      genSelect(el, value, modifiers);    } **else** **if** (tag === 'input' && type === 'checkbox') {      genCheckboxModel(el, value, modifiers);    } **else** **if** (tag === 'input' && type === 'radio') {      genRadioModel(el, value, modifiers);    } **else** **if** (tag === 'input' || tag === 'textarea') {      genDefaultModel(el, value, modifiers);    } **else** **if** (!config.isReservedTag(tag)) {      genComponentModel(el, value, modifiers);  **return** false    } **else** {      warn$1(        "<" + (el.tag) + " v-model=\"" + value + "\">: " +        "v-model is not supported on this element type. " +        'If you are working with contenteditable, it\'s recommended to ' +        'wrap a library dedicated for that purpose inside a custom component.'      );    } |

model()根据表单元素的tag标签以及type属性的值，调用不同的方法也就验证了官网所说的“随表单控件类型不同而不同。”这里调用的就是genDefaultModel().

**genDefaultModel**

|  |  |
| --- | --- |
| 1 | **var** type = el.attrsMap.type; |

获取表单元素的类型，此处type='text'.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | {  **var** value$1 = el.attrsMap['v-bind:value'] || el.attrsMap[':value'];  **var** typeBinding = el.attrsMap['v-bind:type'] || el.attrsMap[':type'];  **if** (value$1 && !typeBinding) {  **var** binding = el.attrsMap['v-bind:value'] ? 'v-bind:value' : ':value';        warn$1(          binding + "=\"" + value$1 + "\" conflicts with v-model on the same element " +          'because the latter already expands to a value binding internally'        );      }    } |

检测该表单元素是否同时有v-model绑定和v-bind:value。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **var** ref = modifiers || {};  **var** lazy = ref.lazy;  **var** number = ref.number;  **var** trim = ref.trim;  **var** needCompositionGuard = !lazy && type !== 'range';  **var** event = lazy    ? 'change'    : type === 'range'      ? RANGE\_TOKEN      : 'input'; |

获取修饰符lazy,number及trim.

.lazy 取代input监听change事件

.number 输入字符串转为数字

.trim 输入首尾空格过滤

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **var** valueExpression = '$event.target.value';  **if** (trim) {     valueExpression = "$event.target.value.trim()";   }  **if** (number) {     valueExpression = "\_n(" + valueExpression + ")";   } |

定义变量valueExpression，本例子的情况valueExpression为'$event.target.value'。

|  |  |
| --- | --- |
| 1 | **var** code = genAssignmentCode(value, valueExpression); |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **function** genAssignmentCode (    value,    assignment  ) {  **var** res = parseModel(value);  **if** (res.key === null) {  **return** (value + "=" + assignment)    } **else** {  **return** ("$set(" + (res.exp) + ", " + (res.key) + ", " + assignment + ")")    }  } |

通过genAssignmentCode()方法生成v-model value值得代码。根据本文的例子返回的结果就是"message=$event.target.value"。

|  |  |
| --- | --- |
| 1  2  3 | **if** (needCompositionGuard) {      code = "if($event.target.composing)return;" + code;    } |

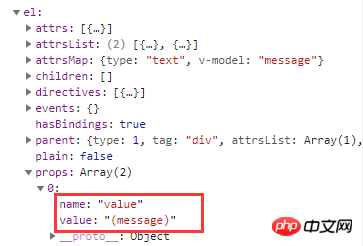
添加event.target.composing判断。event.target.composing用于判断此次input事件是否是IME构成触发的，如果是IME构成，直接return。IME 是输入法编辑器(Input Method Editor) 的英文缩写，IME构成指我们在输入文字时，处于未确认状态的文字。

带下划线的ceshi就属于IME构成，它会同样会触发input事件，但不会触发v-model更新数据。

|  |  |
| --- | --- |
| 1 | addProp(el, 'value', ("(" + value + ")")); |

|  |  |
| --- | --- |
| 1  2  3  4 | **function** addProp (el, name, value) {    (el.props || (el.props = [])).push({ name: name, value: value });    el.plain = false;  } |

给el添加prop



|  |  |
| --- | --- |
| 1 | addHandler(el, event, code, null, true); |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **function** addHandler (el,name,value,modifiers,important,warn){      /\*其他代码省略\*/  **var** newHandler = {          value: value.trim()        };  **var** handlers = events[name];  **if** (Array.isArray(handlers)) {          important ? handlers.unshift(newHandler) : handlers.push(newHandler);        } **else** **if** (handlers) {          events[name] = important ? [newHandler, handlers] : [handlers, newHandler];        } **else** {          events[name] = newHandler;        }  } |

将code作为el的对应event处理的方法handler,此处events['input']=if($event.target.composing)return;message =$event.target.value  
最后原来的html结构就由：

|  |  |
| --- | --- |
| 1 | <input type="text" v-model="message"> |

变成了：

|  |  |
| --- | --- |
| 1 | <input type="text" :value="message"  @input="if($event.target.composing)return;message =$event.target.value"> |

**添加修饰符**

如果添加了trim修饰符  
原来的html结构就由：

|  |  |
| --- | --- |
| 1 | <input type="text" v-model.trim="message"> |

变成了：

|  |  |
| --- | --- |
| 1 | <input type="text" :value="message"  @input="if($event.target.composing)return;message =$event.target.value.trim()"> |

如果添加了lazy修饰符  
原来的html结构就由：

|  |  |
| --- | --- |
| 1 | <input type="text" v-model.lazy="message"> |

变成了：

|  |  |
| --- | --- |
| 1 | <input type="text" :value="message"  @change="if($event.target.composing)return;message =$event.target.value"> |

如果添加了number修饰符  
原来的html结构就由：

|  |  |
| --- | --- |
| 1 | <input type="text" v-model.number="message"> |

变成了：

|  |  |
| --- | --- |
| 1 | <input type="text" :value="message"  @change="if($event.target.composing)return;message = \_n($event.target.value)"> |

这里的\_n是在installRenderHelpers里面定义的，指向toNumber方法。

|  |  |
| --- | --- |
| 1  2  3  4 | **function** installRenderHelpers (target) {    /\*其他代码省略\*/    target.\_n = toNumber;  } |

而toNumber就是使用parseFloat函数来转的数字，再使用isNaN判断转换结果，如果结果是NaN,那么就返回原字符串，否则返回转为数字后的结果：

|  |  |
| --- | --- |
| 1  2  3  4 | **function** toNumber (val) {  **var** n = parseFloat(val);  **return** isNaN(n) ? val : n  } |

注意:number修饰符不能限制输入的内容，就算输入的不是数字，也可能会被转换，如输入'1次测试',转换的结果就是1

答：可以实现双向绑定，指令（v-class、v-for、v-if、v-show、v-on）。vue的model层的data属性。绑定事件：<input @click=doLog() />

# #### 68、mint-ui是什么？怎么使用？说出至少三个组件使用方法？

答：基于vue的前端组件库。npm安装，然后import样式和js，vue.use（mintUi）全局引入。

在单个组件局部引入：import {Toast} from ‘mint-ui’。

组件一：Toast(‘登录成功’)；

组件二：mint-header；

组件三：mint-swiper

# #### 71、axios是什么？怎么使用？描述使用它实现登录功能的流程？

答：请求后台资源的模块。npm install axios -S装好，然后发送的是跨域，需在配置文件中config/index.js进行设置。后台如果是Tp5则定义一个资源路由。js中使用import进来，然后.get或.post。返回在.then函数中如果成功，失败则是在.catch函数中

# #### 72、axios+tp5进阶中，调用axios.post(‘api/user’)是进行的什么操作？axios.put(‘api/user/8′)呢？

答：跨域，添加用户操作，更新操作。

# #### 73、什么是RESTful API？怎么使用?

答：是一个api的标准，无状态请求。请求的路由地址是固定的，如果是tp5则先路由配置中把资源路由配置好。标准有：.post .put .delete

# #### 74、mvvm框架是什么？它和其它框架（jquery）的区别是什么？哪些场景适合？

答：一个model+view+viewModel框架，数据模型model，viewModel连接两个

区别：vue数据驱动，通过数据来显示视图层而不是节点操作。

场景：数据操作比较多的场景，更加便捷

# #### 75、自定义指令（v-check、v-focus）的方法有哪些？它有哪些钩子函数？还有哪些钩子函数参数？

答：全局定义指令：在vue对象的directive方法里面有两个参数，一个是指令名称，另外一个是函数。组件内定义指令：directives

钩子函数：bind（绑定事件触发）、inserted(节点插入的时候触发)、update（组件内相关更新）

钩子函数参数：el、binding

# #### 76、说出至少4种vue当中的指令和它的用法？

答：v-if：判断是否隐藏；v-for：数据循环出来；v-bind:class：绑定一个属性；v-model：实现双向绑定

# #### 77、vue.cli项目中src目录每个文件夹和文件的用法？

答：assets文件夹是放静态资源；

components是放组件；

router是定义路由相关的配置;

view视图；

app.vue是一个应用主组件；

main.js是入口文件

# #### 78、vue.cli中怎样使用自定义的组件？有遇到过哪些问题吗？

第一步：在components目录新建你的组件文件（smithButton.vue），script一定要export default {

第二步：在需要用的页面（组件）中导入：import smithButton from ‘../components/smithButton.vue’

第三步：注入到vue的子组件的components属性上面,components:{smithButton}

第四步：在template视图view中使用，<smith-button> </smith-button>

问题有：smithButton命名，使用的时候则smith-button。

# #### 79、描述下 vue 从初始化页面--修改数据--刷新页面 UI 的过程？

  当 Vue 进入初始化阶段时，一方面 Vue 会遍历 data 中的属性，并用 Object.defineProperty 将它转化成 getter/setter 的形式，实现数据劫持(暂不谈 Vue3.0 的 Proxy)；另一方面，Vue 的指令编译器 Compiler 对元素节点的各个指令进行解析，初始化视图，并订阅 Watcher 来更新试图，此时 Watcher 会将自己添加到消息订阅器 Dep 中，此时初始化完毕。  
  当数据发生变化时，触发 Observer 中 setter 方法，立即调用 Dep.notify(),Dep 这个数组开始遍历所有的订阅者，并调用其 update 方法，Vue 内部再通过 diff 算法，patch 相应的更新完成对订阅者视图的改变。

# #### 80、Vue 的响应式系统的?

响应式系统简述:

任何一个 Vue Component 都有一个与之对应的 Watcher 实例

Vue 的 data 上的属性会被添加 getter 和 setter 属性

当 Vue Component render 函数被执行的时候, data 上会被 触碰(touch), 即被读, getter 方法会被调用, 此时 Vue 会去记录此 Vue component 所依赖的所有 data。(这一过程被称为依赖收集)

data 被改动时（主要是用户操作）, 即被写, setter 方法会被调用, 此时 Vue 会去通知所有依赖于此 data 的组件去调用他们的 render 函数进行更新

# #### 81、虚拟 DOM 实现原理

虚拟DOM本质上是JavaScript对象,是对真实DOM的抽象

状态变更时，记录新树和旧树的差异

最后把差异更新到真正的dom中

既然 Vue 通过数据劫持可以精准探测数据变化,为什么还需要虚拟 DOM 进行 diff 检测差异?

# #### 81、Vue 的变化侦测原理

前置知识: 依赖收集、虚拟 DOM、响应式系统  
现代前端框架有两种方式侦测变化，一种是pull，一种是push

pull: 其代表为React，我们可以回忆一下React是如何侦测到变化的,我们通常会用setStateAPI显式更新，然后React会进行一层层的Virtual Dom Diff操作找出差异，然后Patch到DOM上，React从一开始就不知道到底是哪发生了变化，只是知道「有变化了」，然后再进行比较暴力的Diff操作查找「哪发生变化了」，另外一个代表就是Angular的脏检查操作。

push: Vue的响应式系统则是push的代表，当Vue程序初始化的时候就会对数据data进行依赖的收集，一但数据发生变化,响应式系统就会立刻得知。因此Vue是一开始就知道是「在哪发生变化了」，但是这又会产生一个问题，如果你熟悉Vue的响应式系统就知道，通常一个绑定一个数据就需要一个Watcher，一但我们的绑定细粒度过高就会产生大量的Watcher，这会带来内存以及依赖追踪的开销，而细粒度过低会无法精准侦测变化,因此Vue的设计是选择中等细粒度的方案,在组件级别进行push侦测的方式,也就是那套响应式系统,通常我们会第一时间侦测到发生变化的组件,然后在组件内部进行Virtual Dom Diff获取更加具体的差异，而Virtual Dom Diff则是pull操作，Vue是push+pull结合的方式进行变化侦测的。

# #### 82、Vue 中 key 值的作用？

  当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。key 的作用主要是为了高效的更新虚拟DOM。

[回到顶部](https://www.cnblogs.com/chenwenhao/p/11258895.html" \l "_labelTop)

# #### 83、Vue 组件间通信方式?

[Vue 组件间通信六种方式](https://juejin.im/post/5cde0b43f265da03867e78d3)

props/$emit

$emit/$on

vuex

$attrs/$listeners

provide/inject

$parent/$children 与 ref

# #### 84、watch、methods 和 computed 的区别?

watch 为了监听某个响应数据的变化。computed 是自动监听依赖值的变化，从而动态返回内容，主要目的是简化模板内的复杂运算。所以区别来源于用法，只是需要动态值，那就用 computed ；需要知道值的改变后执行业务逻辑，才用 watch。

methods是一个方法，它可以接受参数，而computed 不能，computed 是可以缓存的，methods 不会。computed 可以依赖其他 computed，甚至是其他组件的 data。

# #### 85、vue 中怎么重置data?

使用Object.assign()，vm.$data可以获取当前状态下的data，vm.$options.data(this)可以获取到组件初始化状态下的data。

复制Object.assign(this.$data, this.$options.data(this)) // 注意加this，不然取不到data() { a: this.methodA } 中的this.methodA。

原因参考：[Vue中的this.$options.data()和this.$data](https://blog.csdn.net/mocoe/article/details/89682022)

# #### 86、组件中name 作用？

项目使用 keep-alive 时，可搭配组件 name 进行缓存过滤

DOM 做递归组件时需要调用自身 name

vue-devtools 调试工具里显示的组见名称是由vue中组件name决定的

# #### 87、route和router 的区别是什么？

route是路由信息对象”，包括

path,params,hash,query,fullPath,matched,name等路由信息参数。  
router是“路由实例对象”，包括了路由的跳转方法(push、replace)，钩子函数等。

# #### 89、Vue 的 nextTick原理

nextTick 的原理正是 vue 通过异步队列控制 DOM 更新和 nextTick 回调函数先后执行的方式。如果大家看过这部分的源码，会发现其中做了很多 isNative()的判断，因为这里还存在兼容性优雅降级的问题。

1. 为什么需要 nextTick  
    Vue 是异步修改 DOM 的并且不鼓励开发者直接接触 DOM，但有时候业务需要必须对数据更改--刷新后的 DOM 做相应的处理，这时候就可以使用 Vue.nextTick(callback)这个 api 了。

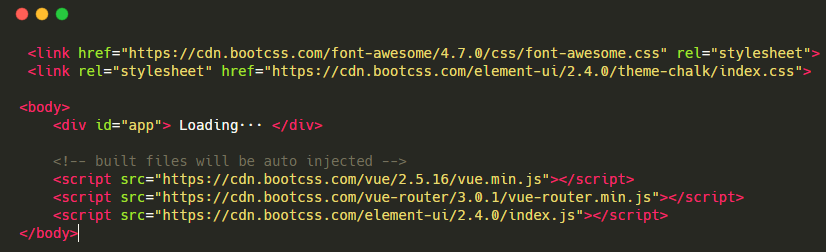
2. 理解原理前的准备  
   首先需要知道事件循环中宏任务和微任务这两个概念(这其实也是面试常考点)。请阅大佬文章--[彻底搞懂浏览器 Event-loop](https://juejin.im/post/5c947bca5188257de704121d)  
常见的宏任务有

script, setTimeout, setInterval, setImmediate, I/O, UI rendering  
常见的微任务有

process.nextTick(Nodejs),Promise.then(), MutationObserver;

# #### 90、vue 首屏加载优化

1. 把不常改变的库放到 index.html 中，通过 cdn 引入

  
然后找到 build/webpack.base.conf.js 文件，在 module.exports = { } 中添加以下代码

复制externals: {

'vue': 'Vue',

'vue-router': 'VueRouter',

'element-ui': 'ELEMENT',

},

这样 webpack 就不会把 vue.js, vue-router, element-ui 库打包了。声明一下，我把 main.js 中对 element 的引入删掉了，不然我发现打包后的 app.css 还是会把 element 的 css 打包进去，删掉后就没了。  
然后你打包就会发现 vendor 文件小了很多~

2. vue 路由的懒加载

import或者require懒加载。你打包就会发现，多了很多 1.xxxxx.js；2.xxxxx.js 等等，而 vendor.xxx.js 没了，剩下 app.js 和 manifest.js，而且 app.js 还很小，我这里是 100k 多一点。

3. 不生成 map 文件

找到 config/index.js，修改为 productionSourceMap: false

4. vue 组件尽量不要全局引入

5. 使用更轻量级的工具库

6. 开启gzip压缩

这个优化是两方面的，前端将文件打包成.gz文件，然后通过nginx的配置，让浏览器直接解析.gz文件。

7. 首页单独做服务端渲染

如果首页真的有瓶颈，可以考虑用 node 单独做服务端渲染，而下面的子页面仍用 spa 单页的方式交互。  
这里不推荐直接用 nuxt.js 服务端渲染方案，因为这样一来增加了学习成本，二来服务端的维护成本也会上升，有时在本机测试没问题，在服务端跑就有问题，为了省心，还是最大限度的使用静态页面较好。

# #### 91、Vue 3.0

   关于Vue 3.0有幸看过尤大的关于3.0版本的[RFC Vue Function-based API RFC](https://zhuanlan.zhihu.com/p/68477600)。大致说了三个点，第一个是关于提出的新API setup()函数，第二个说了对于Typescript的支持，最后说了关于替换Object.defineProperty为 Proxy 的支持。  
   详细说了下关于Proxy代替带来的性能上的提升，因为传统的原型链拦截的方法，无法检测对象及数组的一些更新操作，但使用Proxy又带来了浏览器兼容问题。

[回到顶部](https://www.cnblogs.com/chenwenhao/p/11258895.html" \l "_labelTop)

# #### 92、vue-cli 脚手架替我们做了哪些工作？

首先需要知道 vue-cli 是什么？它是基于 Vue.js 进行快速开发的完整系统，也可以理解成是很多 npm 包的集合。其次，vue-cli 完成的功能有哪些？

.vue 文件 --> .js 文件  
ES6 语法 --> ES5 语法  
Sass,Less,Stylus --> CSS  
对 jpg,png,font 等静态资源的处理  
热更新  
定义环境变量，区分 dev 和 production 模式  
...

如果开发者需要补充或修改默认设置，需要在 package.json 同级下新建一个 vue.config.js 文件

# #### 93、详述虚拟DOM中的diff算法

1，首先要先讲一下虚拟DOM是如何实现的

虚拟DOM是通过js语法来在内存中维护一个通过数据解构描述出来的一个模拟DOM树，当数据发生改变的时候，会先对虚拟DOM进行模拟修改，然后在通过新的虚拟DOM树与旧的虚拟DOM树来对比，而这个对比就是通过diff算法来进行的虚拟DOM最大的意义不在于性能的提升（JavaScript对象比DOM对象性能高），而在于抽象了DOM的具体实现（对DOM进行了一层抽象）

so：步骤一：用JS对象模拟DOM树

步骤二：比较两棵虚拟DOM树的差异

步骤三：把差异应用到真正的DOM树上

同时维护新旧两棵虚拟DOM树，当数据发生改变的时候，开始执行对比

首先对根元素进行对比，如果根元素发生改变就直接对根元素替换

#### 94、vue的动画属性

Vue在元素显示与隐藏的过渡中，提供了 6 个 class 来切换：

v-enter：定义进入过渡的开始状态。在元素被插入之前生效，在元素被插入之后的下一帧移除。

v-enter-active：定义进入过渡生效时的状态。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡/动画完成之后移除。这个类可以被用来定义进入过渡的过程时间，延迟和曲线函数。

v-enter-to: 2.1.8版及以上 定义进入过渡的结束状态。在元素被插入之后下一帧生效 (与此同时 v-enter 被移除)，在过渡/动画完成之后移除。

v-leave: 定义离开过渡的开始状态。在离开过渡被触发时立刻生效，下一帧被移除。

v-leave-active：定义离开过渡生效时的状态。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在过渡/动画完成之后移除。这个类可以被用来定义离开过渡的过程时间，延迟和曲线函数。

v-leave-to: 2.1.8版及以上 定义离开过渡的结束状态。在离开过渡被触发之后下一帧生效 (与此同时 v-leave 被删除)，在过渡/动画完成之后移除。