# VUE常见面试题

**VUE概念方面**

**1.vue优点？**

答：轻量级框架：只关注视图层，是一个构建数据的视图集合，大小只有几十kb；

简单易学：国人开发，中文文档，不存在语言障碍 ，易于理解和学习；

双向数据绑定：保留了angular的特点，在数据操作方面更为简单；

组件化：保留了react的优点，实现了html的封装和重用，在构建单页面应用方面有着独特的优势；

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

虚拟DOM：dom操作是非常耗费性能的， 不再使用原生的dom操作节点，极大解放dom操作，但具体操作的还是dom不过是换了另一种方式；

运行速度更快:相比较与react而言，同样是操作虚拟dom，就性能而言，vue存在很大的优势。

**2.vue的两个核心点**

答：数据驱动、组件系统

数据驱动：ViewModel，保证数据和视图的一致性。

组件系统：应用类UI可以看作全部是由组件树构成的。

**3.vue和jQuery的区别**

答：jQuery是使用选择器（$）选取DOM对象，对其进行赋值、取值、事件绑定等操作，其实和原生的HTML的区别只在于可以更方便的选取和操作DOM对象，而数据和界面是在一起的。比如需要获取label标签的内容：$("lable").val();,它还是依赖DOM元素的值。

Vue则是通过Vue对象将数据和View完全分离开来了。对数据进行操作不再需要引用相应的DOM对象，可以说数据和View是分离的，他们通过Vue对象这个vm实现相互的绑定。这就是传说中的**MVVM**。

**4.mvvm 框架是什么？**

答：vue是实现了双向数据绑定的mvvm框架，当视图改变更新模型层，当模型层改变更新视图层。在vue中，使用了双向绑定技术，就是View的变化能实时让Model发生变化，而Model的变化也能实时更新到View。

**5.你们vue项目是打包了一个js文件，一个css文件，还是有多个文件？**

答：根据vue-cli脚手架规范，一个js文件，一个CSS文件。

**6.请说下封装 vue 组件的过程？**

答：1. 建立组件的模板，先把架子搭起来，写写样式，考虑好组件的基本逻辑。(os：思考1小时，码码10分钟，程序猿的准则。)

2. 准备好组件的数据输入。即分析好逻辑，定好 props 里面的数据、类型。

3. 准备好组件的数据输出。即根据组件逻辑，做好要暴露出来的方法。

4. 封装完毕了，直接调用即可

**7. 引进组件的步骤**

答: 在template中引入组件；

在script的第一行用import引入路径；用component中写上组件名称。

**8.请说出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; app.vue是一个应用主组件；main.js是入口文件。

**9.assets和static的区别**

答：**相同点**：assets和static两个都是存放静态资源文件。项目中所需要的资源文件图片，字体图标，样式文件等都可以放在这两个文件下，这是相同点

**不相同点**：assets中存放的静态资源文件在项目打包时，也就是运行npm run build时会将assets中放置的静态资源文件进行打包上传，所谓打包简单点可以理解为压缩体积，代码格式化。而压缩后的静态资源文件最终也都会放置在static文件中跟着index.html一同上传至服务器。static中放置的静态资源文件就不会要走打包压缩格式化等流程，而是直接进入打包好的目录，直接上传至服务器。因为避免了压缩直接进行上传，在打包时会提高一定的效率，但是static中的资源文件由于没有进行压缩等操作，所以文件的体积也就相对于assets中打包后的文件提交较大点。在服务器中就会占据更大的空间。

**建议**：将项目中template需要的样式文件js文件等都可以放置在assets中，走打包这一流程。减少体积。而项目中引入的第三方的资源文件如iconfoont.css等文件可以放置在static中，因为这些引入的第三方文件已经经过处理，我们不再需要处理，直接上传。

**10.什么是前端渲染？什么是后端渲染？**

1）第一阶段：

后端渲染：jsp/java等服务器

后端路由：处理URL和页面之间的映射关系

后端路由缺点：整个页面模块都又后端人员编写和维护，前端人员如果想要开发界面，需要通过php、java等编写页面代码，此时，html代码和数据以及对应逻辑会混在一起，编写与维护非常困难

2）第二阶段：

前端渲染：浏览器中显示的网页中的大部分内容，都是由前端写的js代码在浏览器中执行，最终渲染出来界面。后端只提供API来返回数据，前端通过ajax获取数据，通过js将数据渲染到网页。

前端渲染优点：前后端责任清洗，后端专注于数据，前端专注于交互和可视化。当移动端出现后，后端不需要任何处理，依然使用之前的一套API即可

3）第三阶段：

单页面富应用（SPA）阶段：SPA最主要的特点是在前后端分离的基础上加了一层前端路由，由前端来维护一套路由规则。静态资源服务器中只有一个html界面

此时，URL和页面的映射关系为前端路由（vue-router）。前端路由的核心是指改变URL，但页面不进行整体的刷新

**11.单页面应用和多页面应用区别及优缺点**

答：**单页面应用（SPA）**，通俗一点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的 html, js, css。所有的页面内容都包含在这个所谓的主页面中。但在写的时候，还是会分开写（页面片段），然后在交互的时候由路由程序动态载入，单页面的页面跳转，**仅刷新局部资源**。多应用于pc端。

**单页面的优点**：用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点spa对服务器压力较小；前后端分离；页面效果会比较炫酷（比如切换页面内容时的专场动画）。

**单页面缺点**：不利于seo；导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）；初次加载时耗时多；页面复杂度提高很多。

**多页面（MPA）**，就是指一个应用中有多个页面，页面跳转时是整页刷新

**12.vue初始化页面闪动问题**

答：使用vue开发时，在vue初始化之前，由于div是不归vue管的，所以我们写的代码在还没有解析的情况下会容易出现花屏现象，看到类似于{{message}}的字样，虽然一般情况下这个时间很短暂，但是我们还是有必要让解决这个问题的。

首先：在css里加上[v-cloak] {display: none;}。

如果没有彻底解决问题，则在根元素加上style="display: none;" :style="{display: 'block'}"

**13.vue常用的修饰符**

答：.stop：等同于JavaScript中的event.stop- Propagation()，防止事件冒泡；

.prevent：等同于JavaScript中的event.- preventDefault()，防止执行预设的行为（如果事件可取消，则取消该事件，而不停止事件的进一步传播）；

.capture：与事件冒泡的方向相反，事件捕获由外到内；

.self：只会触发自己范围内的事件，不包含子元素；

.once：只会触发一次。

**14、写过自定义指令吗？原理是什么？**

答：指令本质上是装饰器，是 vue 对 HTML 元素的扩展，给 HTML 元素添加自定义功能。vue 编译 DOM 时，会找到指令对象，执行指令的相关方法。

自定义指令有五个生命周期（也叫钩子函数），分别是 bind、inserted、update、component-Updated、unbind

1、bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。

2、inserted：被绑定元素插入父节点时调用。

3、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。通过比较前后的绑定值。

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

5、unbind：只调用一次，指令与元素解绑时调用。

**15. Vue.set 方法原理**

答：了解 Vue 响应式原理的同学都知道在**两种情况下修改 Vue 是不会触发视图更新的**：  
 1、在实例创建之后添加新的属性到实例上（给响应式对象新增属性）  
 2、直接更改数组下标来修改数组的值。

**Vue.set 或者说是 $set 原理如下**

答：因为响应式数据 我们给对象和数组本身新增了\_\_ob\_\_属性，代表的是 Observer 实例。当给对象新增不存在的属性，首先会把新的属性进行响应式跟踪 然后会触发对象 \_\_ob\_\_ 的dep收集到的 watcher 去更新，当修改数组索引时我们调用数组本身的 splice 方法去更新数组。

**16. vue 中使用了哪些设计模式？**

答：1、工厂模式 - 传入参数即可创建实例  
虚拟 DOM 根据参数的不同返回基础标签的 Vnode 和组件 Vnode。

2、单例模式 - 整个程序有且仅有一个实例  
vuex 和 vue-router 的插件注册方法 install 判断如果系统存在实例就直接返回掉。

3、发布-订阅模式。（vue 事件机制）

4、观察者模式。（响应式数据原理）

5、装饰器模式（@装饰器的用法）

6、策略模式，策略模式指对象有某个行为，但是在不同的场景中，该行为有不同的实现方案 - 比如选项的合并策略。

**17．函数式组件使用场景和原理**

**函数式组件与普通组件的区别:**

答：1、函数式组件需要在声明组件时指定 functional:true

2、不需要实例化，所以没有this，this通过render函数的第二个参数context代替

3、没有生命周期钩子函数，不能使用计算属性，watch

4、不能通过$emit对外暴露事件，调用事件只能通过context.listeners.click的方式调用外部传入的事件

5、因为函数组件时没有实例化的，所以在外部通过ref去引用组件时，实际引用的是HTMLElement

6、函数式组件的props可以不用显示声明，所以没有在props里面声明的属性都会被自动隐式解析为prop，而普通的组件所有未声明的属性都解析到$attrs里面，并自动挂载到组件根元素上（可以通过inheritAttrs属性禁止）

优点：1.由于函数组件不需要实例化，无状态，没有生命周期，所以渲染性要好于普通组件2.函数组件结构比较简单，代码结构更清晰

**VUE指令相关**

**1.说出几种vue当中的指令和它的用法？**

答：v-once - 定义它的元素或组件只渲染一次，包括元素或组件的所有节点，首次渲染后，不再随数据的变化重新渲染，将被视为静态内容。

v-cloak - 这个指令保持在元素上直到关联实例结束编译 -- 解决初始化慢到页面闪动的最佳实践。

v-bind - 绑定属性，动态更新HTML元素上的属性。例如 v-bind:class。

v-on - 用于监听DOM事件。例如 v-on:click/v-on:keyup

v-html - 赋值就是变量的innerHTML -- 注意防止xss攻击

v-text - 更新元素的textContent

v-model - 1、在普通标签。变成value和input的语法糖，并且会处理拼音输入法的问题。2、再组件上。也是处理value和input语法糖。

v-if / v-else / v-else-if。可以配合template使用；在render函数里面就是三元表达式。

v-show - 使用指令来实现 -- 最终会通过display来进行显示隐藏

v-for - 循环指令编译出来的结果是 -L 代表渲染列表。优先级比v-if高最好不要一起使用，尽量使用计算属性去解决。注意增加唯一key值，不要使用index作为key。

v-pre - 跳过这个元素以及子元素的编译过程，以此来加快整个项目的编译速度。

**2.v-show和v-if指令的共同点和不同点？**

答: 共同点：都能控制元素的显示和隐藏；

v-if 是真正的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建；也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

v-show 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 的 “display” 属性进行切换。

所以，v-if 适用于在运行时很少改变条件，不需要频繁切换条件的场景；v-show 则适用于需要非常频繁切换条件的场景。

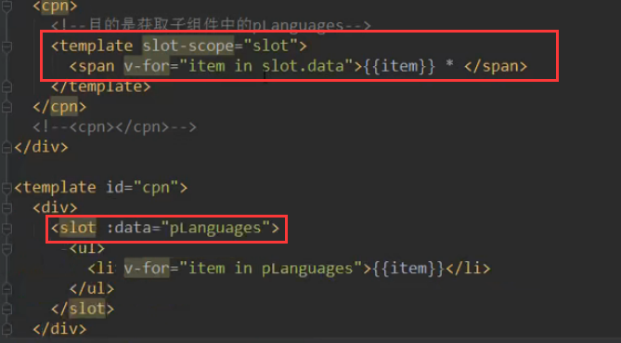
**3.如何理解slot？Slot在什么场景下用？**

Slot（插槽）目的：让封装的组件更加具有扩展性，让使用者决定组件内部的一些内容到底展示什么。封装共性，插槽中为不同内容。

简单来说，假如父组件需要在子组件内放一些DOM，那么这些DOM是显示、不显示、在哪个地方显示、如何显示，就是slot分发负责的活。（slot可以设定默认值）

**具名插槽**：替换有name属性的slot时，必须使用slot=“name”属性指定，与之对应的，无name则称为**匿名插槽**

**作用域插槽**：在插槽上绑定数据（父组件替换插槽的标签，但内容由子组件提供）

例：

Ps：父组件模板的所有东西都会在父级作用域内编译，子组件模板的所有东西都会在子级作用域内编译

**3.v-if和v-for的优先级**

答：当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。所以，不推荐v-if和v-for同时使用。

如果v-if和v-for一起用的话，vue中的的会自动提示v-if应该放到外层去。

**4. vue-loader是什么？使用它的用途有哪些？**

答：vue文件的一个加载器，将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

**5.v-modal的使用。**

答：v-model用于表单数据的双向绑定，其实它就是一个语法糖，这个背后就做了两个操作：

v-bind绑定一个value属性；

v-on指令给当前元素绑定input事件。

**6.v-on可以监听多个方法吗？**

答：可以，栗子：<input type="text" v-on="{ input:onInput,focus:onFocus,blur:onBlur, }">

**7、v-model的原理？**

答：我们在 vue 项目中主要使用 v-model 指令在表单 input、textarea、select 等元素上创建双向数据绑定，我们知道 v-model 本质上不过是语法糖，v-model 在内部为不同的输入元素使用不同的属性并抛出不同的事件：

text 和 textarea 元素使用 value 属性和 input 事件；

checkbox 和 radio 使用 checked 属性和 change 事件；

select 字段将 value 作为 prop 并将 change 作为事件。

以 input 表单元素为例：

<input v-model='something'>

如果在自定义组件中，v-model 默认会利用名为 value 的 prop 和名为 input 的事件

**8、Class 与 Style 如何动态绑定？**

答：Class 可以通过对象语法和数组语法进行动态绑定：对象语法：

<div **v-bind**:class="{ active: isActive, 'text-danger': hasError }"></div>

数组语法：

<div v-bind:class="[isActive ? activeClass : '', errorClass]"></div>

Style 也可以通过对象语法和数组语法进行动态绑定：对象语法：

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

数组语法：

<div v-bind:style="[styleColor, styleSize]"></div>

**9.delete和Vue.delete删除数组的区别**

答：delete只是被删除的元素变成了 empty/ undefined 其他的元素的键值还是不变。Vue.delete 直接删除了数组改变了数组的键值。

**10.分别简述computed和watch的使用场景**

答：computed:是计算属性，依赖其它属性值，并且 computed 的值有缓存，只有它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值（即当一个属性受多个属性影响的时候就需要用到computed）

最典型的栗子：购物车商品结算的时候

watch: 更多的是「观察」的作用，类似于某些数据的监听回调 ，每当监听的数据变化时都会执行回调进行后续操作（当一条数据影响多条数据的时候就需要用watch）

运用场景：

当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；

当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作 ( 访问一个 API )，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

**VUE生命周期相关**

**1.什么是 vue 生命周期？有什么作用？**

答：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、编译模版、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。

每个 Vue 实例在被创建时都要经过一系列的初始化过程——例如，需要设置数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时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。（ps：生命周期钩子就是生命周期函数）例如，如果要通过某些插件操作DOM节点，如想在页面渲染完后弹出广告窗， 那我们最早可在mounted 中进行。

**2.** **Vue的生命周期方法有哪些？一般在哪一步发送请求？**

| 生命周期 | 描述 |
| --- | --- |
| beforeCreate | 组件实例被创建之初，组件的属性生效之前，data和methods中的数据都没有被初始化 |
| created | 组件实例已经完全创建，属性也绑定，data和methods中都已经被初始化好了（可操作的最早的函数）  但真实 dom 还没有生成，$el 还不可用 |
| beforeMount | 在挂载开始之前被调用：相关的 render 函数首次被调用 |
| mounted | el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子 |
| beforeUpdate | 组件数据更新之前调用，发生在虚拟 DOM 打补丁之前 |
| update | 组件数据更新之后 |
| activited | keep-alive 专属，组件被激活时调用 |
| deactivated | keep-alive 专属，组件被销毁时调用 |
| beforeDestory | 组件销毁前调用 |
| destoryed | 组件销毁后调用 |

**3.** **在哪个生命周期内调用异步请求？**

答：可以在钩子函数 created、before-Mount、mounted 中进行调用，因为在这三个钩子函数中，data 已经创建，可以将服务端端返回的数据进行赋值。但是本人推荐在 created 钩子函数中调用异步请求，因为在 created 钩子函数中调用异步请求有以下优点：能更快获取到服务端数据，减少页面 loading 时间；ssr 不支持 beforeMount 、mounted 钩子函数，所以放在 created 中有助于一致性；

**4.第一次页面加载会触发哪几个钩子？**

答：beforeCreate， created， beforeMount， mounted

**5.created和mounted的区别**

答：created:在模板渲染成html前调用，即通常初始化某些属性值，然后再渲染成视图。

mounted:在模板渲染成html后调用，通常是初始化页面完成后，再对html的dom节点进行一些需要的操作。

**5.vue获取数据在哪个周期函数**

答：一般 created/beforeMount/mounted 皆可.

比如如果你要操作 DOM , 那肯定 mounted 时候才能操作.

**6.请详细说下你对vue生命周期的理解？**

答：总共分为8个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

**BeforeCreate/created**： 在beforeCreated阶段，vue实例的挂载元素$el、methods、 data都为undefined，还未初始化。在**created**阶段，vue实例的数据对象data有了，$el还没有。

**beforeMount/mounted**：在**beforeMount**阶段，模板已经编译完成，在内存中，尚未把模板渲染到页面中。而且在beforeMount执行时，页面中的元素尚未被真正替换，只是之前写的一些模板字符串（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,表示内存中的模板，已经真实的挂载到了页面中，用户已经可以看到渲染号的页面了。是实例创建期间的最后一个生命周期函数，当执行完mounted就表示实例已经被完全创建好了，组件已经脱离了创建阶段，进入了运行阶段。此时如果没有其他操作，实例就静静的存储在内存中。如果要通过某些插件操作页面上的DOM节点，最早要在mounted中进行

**beforeUpdate /Update**：当data变化时，会有选择性的触发beforeUpdate和updated方法。

BeforeUpdate表示界面还没被更新，但数据已经被更新了。此步表示先根据data中的最新数据，在内存中重新渲染一份最新的内存DOM树，并将其重新渲染到真实的页面当中去，完成了从data（Model层）-> view（视图层的更新）。即Update事件执行时，页面和数据已经保持同步了，都是最新的

beforeDestory/destoryed：在beforeDestory时，实例上所有的data和所有methods等都处于可用状态，还未真正执行销毁

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。

**7、Vue 的父组件和子组件生命周期钩子函数执行顺序？**

答：Vue 的父组件和子组件生命周期钩子函数执行顺序可以归类为以下 4 部分：

加载渲染过程

父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted

子组件更新过程

父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated

父组件更新过程

父 beforeUpdate -> 父 updated

销毁过程

父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed

**8、父组件可以监听到子组件的生命周期吗？**

答：比如有父组件 Parent 和子组件 Child，如果父组件监听到子组件挂载 mounted 就做一些逻辑处理，可以通过以下写法实现：

// Parent.vue

以上需要手动通过 $emit 触发父组件的事件，更简单的方式可以在父组件引用子组件时通过 @hook 来监听即可，如下所示：

// Parent.vue

当然 @hook 方法不仅仅是可以监听 mounted，其它的生命周期事件，例如：created，updated 等都可以监听。

**VUE路由相关**

**1.vue-router 是什么?它有哪些组件**

答：Vue用来写路由一个插件。有router-link、router-view两种组件，均为Vue在源码中设置的全局组件。

**2. 配置路由的过程**

**A：搭载框架**

1.使用Vue.use安装插件

2.并创建VueRouter对象以配置路由和组件之间的应用关系。

3.将router对象传入到Vue实例中,即在Vue实例中挂载创建的路由实例（在index.js路由配置文件中使用export default router导出，在main.js中使用import router from ‘./router/index.js’导入

**B：设置映射关系**

例：配置路由const routers =[

{

Path:‘/home’

Component:对应的home组件

Redirect：‘/TEST’重定向，默认展示

//默认为hash模式，在创建实例模式中，设置mode：’history’即可定义为history模式。

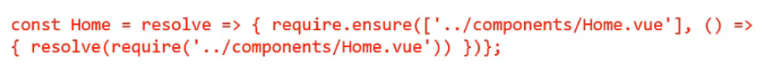
}

]

并使用<router-link to=’/home’replace（禁止返回后退）tag=”button” active-class（路由匹配成功时，自动设置router-link-active的class，可修改默认样式）>使用（router-link相当于一个a标签），

最后使用<router-view>相当于渲染组件的占位

**3. vue-router 路由模式有几种？**

答：vue-router 有 3 种路由模式：hash、history、abstract。其中，3 种路由模式的说明如下：

hash: 使用 URL hash 值来作路由。支持所有浏览器，包括不支持 HTML5 History Api 的浏览器；（实现原理自行补充），网页不存在刷新，不会重新请求资源。Location.hash = ‘XXX’

history: 依赖 HTML5 History API 和服务器配置。具体可以查看 HTML5 History 模式；（实现原理自行补充）history.pushState({},’’,’about), 主要用于栈结构且永远展示栈顶。push与pop操作满足先进后出。也可以改变URL而不发生刷新操作。 History.replaceState（{，‘’，‘about’}）替换操作。

History.go(2)：显示栈结构中指定数据

以上操作均为实现URL的前进后退操作

abstract: 支持所有 JavaScript 运行环境，如 Node.js 服务器端。如果发现没有浏览器的 API，路由会自动强制进入这个模式.

**4.怎么定义vue-router 的动态路由? 怎么获取传过来的值？**

答：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。（“：to”：写成对象形式，v-bind）

<router-link **:to**=”’/user’+ userid”></>在computed中动态获取userid进行拼接

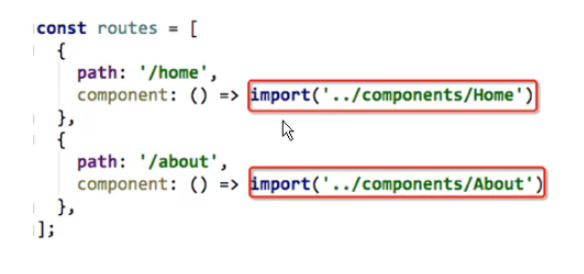
Path中设置：path：‘/user/:userid’

**5.vue-router实现路由懒加载（ 动态加载路由 ）**

答:路由懒加载：当打包构建应用的时候。Js包会变得非常大，影响页面加载。如果把不同路由对应的组件分割为不同的代码块，然后当路由被访问时才加载对应组件，这样更加高效

三种方式

第一种：Vue异步组件技术,异步加载. Vue-router配置路由, 使用Vue的异步组件技术, 可以实现按需加载 .但是,这种情况下一个组件生成一个js文件。

第二种：路由懒加载(使用import，推荐)。

第三种：webpack提供的require.ensure()，vue-router配置路由，使用webpack的require. ensure技术，也可以实现按需加载。这种情况下，多个路由指定相同的chunkName，会合并打包成一个js文件。

第四种：AMD写法

**6.$route 和 $router 的区别**

答：$router是在/router/index.js文件中定义的VueRouter的实例，在源码中为一个大类，有push、replace、to等方法（script标签中想要导航到不同的URL,使用$router.push方法。返回上一个历史history用$router.to(-1)）

$route为当前router跳转对象。里面可以获取当前路由的name,path,query,parmas等。

**7.Vue-router跳转和location.href有什么区别**

答：使用location.href='/url'来跳转，简单方便，但是刷新了页面；

使用history.pushState('/url')，无刷新页面，静态跳转；

引进router，然后使用router.push('/url')来跳转，使用了diff算法，实现了按需加载，减少了dom的消耗。

其实使用router跳转和使用history.pushState ()没什么差别的，因为vue-router就是用了history.pushState()，尤其是在history模式下。

**8.路由全局导航守卫（拦截）**

**目的：**监听路由的跳转过程，在某个监听函数中进行一些操作。

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

Next() //继续执行下一步

From：跳转源

To：跳转目标,route类型

Document.title = to.matched[0]. meta.title

(meta.title为路由映射中自己定义的标题名)

})

总结：导航守卫主要用于监听路由的进入和离开，vue-router提供了beforeEach和afterEach的钩子函数，会在路由即将改变前和改变后触发

此外导航守卫还有还有全局解析守卫、路由独享守卫，组件内的守卫等等

**9.路由的嵌套（子路由）**

在父级路由中定义children:[

{ path:’/news’,component:XXX},

{ path:’/messages’,component:XXX},

]

<router-link :to=”/home/users”></>

**10.路由的参数传递**

**方法一：params的方式**

配置路由格式：/router/:id

传递：使用to方法进行拼接

<router-link :to=”’/user’+ userid”></>在computed中动态获取userid进行拼接

传递后形成的方式：/user/news

**方法二：query类型(大量数据时适用，传入对象)**

配置路由方式：/router，普通配置

传递方式：对象中使用query的key作为传递方式

传递后形成的路径：/router?id=123

例：

父子件<router-link ：to=”{ path:’/profile’,query:{name:’why’,age:’15’}}”>

子组件获取：{{$route.query.name}}

在methods进行路由路径拼接：

This.$router.push(‘/user’+ this.userId(组件中另一个方法))

也可拼接对象

This.$router.push({

Path:’/user’,

Query:{

Name:’aaa’,

Age:17,

}

})

**11、谈谈你对 keep-alive 的了解？**

答：<keep-alive></keep-alive>是 Vue 内置的一个组件，可以使被包含的组件保留状态，避免重新渲染，是一个抽象组件，在v页面渲染完成后不会被渲染成为一个DOM元素。其有以下特性：一般结合路由和动态组件一起使用，用于缓存组件；（例：router-view为一个全局组件，如果被包含在keep-alive中，所有路径匹配到的视图组件都会被缓存）

常用的两个属性 include、exclude，允许组件有条件的进行缓存（提供 include 和 exclude 属性，两者都支持字符串或正则表达式。include表示只有名称匹配的组件会被缓存，exclude表示任何名称匹配的组件都不会被缓存 ，其中 exclude 的优先级比 include高）。

也可以使用meta来判定是否缓存：

{

path:'/',

name:'home',

components:Home,

meta:{

keepAlive:true //需要被缓存的组件

},

两个生命周期：activated/deactivated，用来得知当前组件是否处理活跃状态（对应两个钩子函数 activated 和 deactivated ，当组件被激活时，触发钩子函数 activated，当组件被移除时，触发钩子函数 deactivated。）

keep-alive 运用了 LRU 算法，选择最近最久未使用的组件予以淘汰。

**VUE数据传输相关**

**1、Vue 组件间通信有哪几种方式？**

答：Vue 组件间通信是面试常考的知识点之一，这题有点类似于开放题，你回答出越多方法当然越加分，表明你对 Vue 掌握的越熟练。Vue 组件间通信只要指以下 3 类通信：父子组件通信、隔代组件通信、兄弟组件通信，下面我们分别介绍每种通信方式且会说明此种方法可适用于哪类组件间通信。

（2）父子组件的访问方式：

**方法1： props / $emit（父子传值）**

**父向子**通过 props 传递数据，通过属性传递 可传递动态属性 也可以传递静态属性 可传递数字、布尔值、数组、对象等任何类型的值，v-bind绑定也可以定义每个传输接口的一些属性：type/default/required

**子向父**通过 $emit 向父组件派发事件，父组件调用获取子组件参数，ref 作用在组件上，指向的是组件的实例，子组件实例上的方法都可以调用（ref如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素）

**例**：Props:{

Test：{ //注意，v-bind目前不支持驼峰，使用时应c-info(cInfo)

Type：Object/String/Int….

Default:{

Return xxx

} 默认值

Required：true //布尔值，必须传输

}

}

Emit：自定义事件：this.$emit(‘itemClick’，传参)

在父组件中监听模板，@itemClick =”父组件中自定义的方法”

**方法2： $ref / $emit （父子调用方法）**

**父向子**通过 $ref 获取子组件数据 调用子组件方法，ref 作用在组件上 指向的是组件的实例 实例上的方法都可以调用

**子向父** 通过 $emit 向父组件触发一个事件 子组件就可以调用父组件的方法

**方法3：$children / $parent（父子传值、调方法）**

**父向子** 通过 $children 获取子组件数据和调用子组件方法

**子向父** 通过 $parent 获取父组件数据和调用父组件方法($root 和 $parent 都能够实现访问父组件的属性和方法，两者的区别在于，如果存在多级子组件，通过parent 访问得到的是它最近一级的父组件，通过root 访问得到的是根父组件(App.vue) 所以存在组件嵌套的情况下 不要使用 $root)

**方法4： $emit / $on（父子 兄弟 跨级）**

父子、兄弟、跨级这种方法通过一个空的 Vue 实例作为中央事件总线（事件中心），用它来触发事件和监听事件,巧妙而轻量地实现了任何组件间的通信，包括父子、兄弟、跨级。当我们的项目比较大时，可以选择更好的状态管理解决方案 vuex。

**方法5： provide / inject（跨级 依赖注入）**

1跨级： Vue2.2.0 新增 API,这对选项需要一起使用，以允许一个祖先组件向其所有子孙后代注入一个依赖，不论组件层次有多深，并在起上下游关系成立的时间里始终生效。一言而蔽之：祖先组件中通过 provider 来提供变量，然后在子孙组件中通过 inject 来注入变量。

**方法6： $attrs / $listeners（跨级）**

跨级：多级组件嵌套需要传递数据时，通常使用的方法是通过 vuex。但如果仅仅是传递数据，而不做中间处理，使用 vuex 处理，未免有点大材小用。为此 Vue2.4 版本提供了另一种方法—— $attrs / $listeners

**方法7： Vuex**

**Vuex** 适用于父子、隔代、兄弟组件通信

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。每一个 Vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着你的应用中大部分的状态 ( state )。

Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

改变 store 中的状态的唯一途径就是显式地提交 (commit) mutation。这样使得我们可以方便地跟踪每一个状态的变化。

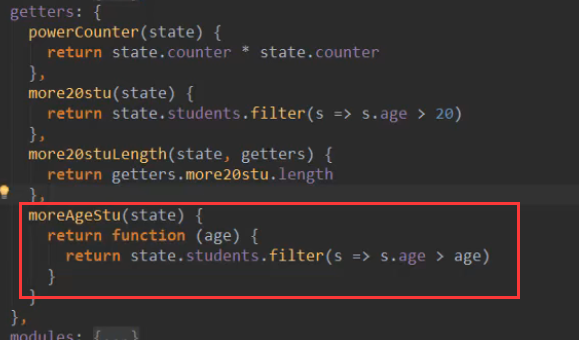
Ps：状态管理补充：主要是将多个组件共享的变量全部存储在一个对象中，将这个对象放在顶层的VUE实例中，让其他组件可以使用。但是自己封装的对象做到响应式比较麻烦

**2. vuex主要包括以下五个模块/属性：**

**1.State**：定义了应用状态的数据结构，可以在这里设置默认的初始状态。（基本数据(数据源存放地)）Vuex使用了单一状态树管理应用层级别的全部状态，让我们能最直接的找到某个状态的片段，方便管理和维护，即在一个项目的store中只有一个state

**2.Getter**：允许**组件从 Store 中获取数据**，mapGetters 辅助函数仅仅是将 store 中的 getter 映射到局部计算属性。（从基本数据派生出来的数据）

应用场景：在Store仓库里，state就是用来存放数据，若是对数据进行处理输出，比如数据要过滤，一般我们可以写到computed中。但是如果很多组件都使用这个过滤后的数据，比如饼状图组件和曲线图组件，则可以把这个数据抽提出来共享，使用store中的计算属性getters

Getters中的参数：返回一个函数（状态在变）

**3.Mutation**：是唯一更改 store 中状态的方法，且必须是同步函数。（提交更改数据的方法，同步！） //其余组件操作increment方法时：

Methods:{

Increments: function(){

This.$store.commit(‘mutatuion中的定义方法’)

}}

Mutation中的载荷/负载（payload：参数）

**4.Action**像一个装饰器，包裹mutations，可以进行**异步操作，即发送网络请求时。**vuex为了解决mutations只有同步的问题,提出了actions(异步),专门用来解决mutations只有同步无异步的问题。

在进行异步操作时，拿到mutation里的方法调用，提交。例：

aUpdateInfo(context){

setTimeOur(()=>{

context.commit(‘updateInfo’)

//mutation中的方法’

}，1000)

}

在提交时，使用dispatch，例:

updateInfo(){

this.$store.dispatch(‘aUpdateInfo’)

}

**5.Module**：允许将单一的Store 拆分为多个 store 且同时保存在单一的状态树中，模块化vuex

**3.VUEX的使用**

1. 安装插件

在store文件夹的indexjs中调用Vue.use(Vuex)

2. 创建对象

Const store = new Vuex.store({

State:{

定义状态：

Couter：1000

//别处可直接调用 $store.state.counter

},

Mutations:{

//修改state的状态

Increment(state){

State.counter ++

}

},

Actions:{},

Modules:{}

})

3. 导出store对象并在main.js 中挂载

导出：Expore default store

挂载： import store from ‘./store’

Devtools:记录每次修改的state的状态

**4.Vuex 为什么要分模块并且加命名空间？**

**模块**：由于使用单一状态树，应用的所有状态会集中到一个比较大的对象。当应用变得非常复杂时，store 对象就有可能会变得相当臃肿。为了解决以上问题，Vuex 允许我们将 store 分割成模块（module）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块。

**命名空间**： 默认情况下，模块内部的 action、mutation、getter是注册在全局命名空间的 --- 这样使得多个模块能够对同一 mutation 或 action 做出响应。如果希望你的模块具有更高的封装度和复用性，你可以通过添加 namespaced:true 的方式使其成为带命名的模块。当模块被注册后，他所有 getter、action、及mutation 都会自动根据模块注册的路径调整命名。

**6.Vuex在什么情况下用？为什么用？**

Vuex主要是为了提供多个组件共享状态的插件，并且能做到响应式。

应用场景：登录用户的头像，名称等等

商品收藏，购物车中的物品等状态信息

**7.Vuex中mutations和actions的区别**

vuex为了解决mutations只有同步的问题,提出了actions(异步),专门用来解决mutations只有同步无异步的问题

**8.Mutation中修改states的参数不会报错而直接更改states会报错？**

**2.Vue中双向数据绑定是如何实现的？**

答：vue 双向数据绑定是通过 数据劫持 结合 发布订阅模式的方式来实现的， 也就是说数据和视图同步，数据发生变化，视图跟着变化，视图变化，数据也随之发生改变；

核心：关于VUE双向数据绑定，其核心是 Object.defineProperty()方法。（自行补充）

**3. 怎样理解 Vue 的单向数据流？**

答：所有的 prop 都使得其父子 prop 之间形成了一个单向下行绑定：父级 prop 的更新会向下流动到子组件中，但是反过来则不行。

这样会防止从子组件意外改变父级组件的状态，从而导致你的应用的数据流向难以理解。

额外的，每次父级组件发生更新时，子组件中所有的 prop 都将会刷新为最新的值。这意味着你不应该在一个子组件内部改变 prop。如果你这样做了，Vue 会在浏览器的控制台中发出警告。

子组件想修改时，只能通过 $emit 派发一个自定义事件，父组件接收到后，由父组件修改。

有两种常见的试图改变一个 prop 的情形 :

这个 prop 用来传递一个初始值；这个子组件接下来希望将其作为一个本地的 prop 数据来使用。 在这种情况下，最好定义一个本地的 data 属性并将这个 prop 用作其初始值：

props: ['initialCounter'],

这个 prop 以一种原始的值传入且需要进行转换。 在这种情况下，最好使用这个 prop 的值来定义一个计算属性

props: ['size'],

**4.vue组件中data为什么必须是一个函数？**

答： 组件中的data写成一个函数，数据以函数返回值的形式定义，这样每次复用组件的时候，都会返回一份新的data，相当于每个组件实例都有自己私有的数据空间，它们只负责各自维护的数据，不会造成混乱。而单纯的写成对象形式，就是所有的组件实例共用了一个data，这样改一个全都改了。（就算是多个相同的组件实例，他们的data也不共享，因为data为函数，有自己的暂空间）

**5.** **Vue组件可以访问Vue实例数据吗？**

答：不可以，组件是一个单独功能模块的封装，这个模块有属于自己的html模板，也应该有属于自己数据保存的地方。即使可以访问，所有的数据都放在Vue实例中，Vue实例就会变得非常臃肿。

**5.params和query的区别**

答：用法：query要用path来引入，params要用name来引入，接收参数都是类似的，分别是this.$route.query.name和this.$route.params.name。

url地址显示：query更加类似于我们ajax中get传参，params则类似于post，说的再简单一点，前者在浏览器地址栏中显示参数，后者则不显示

注意点：query刷新不会丢失query里面的数据

params刷新 会 丢失 params里面的数据。

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

答：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。

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。

**7.$nextTick的使用**

答：当你修改了data的值然后马上获取这个dom元素的值，是不能获取到更新后的值，

你需要使用$nextTick这个回调，让修改后的data值渲染更新到dom元素之后在获取，才能成功。

**8、Proxy 与 Object.define Property 优劣对比**

Proxy 的优势如下:

Proxy 可以直接监听对象而非属性；

Proxy 可以直接监听数组的变化；

Proxy 有多达 13 种拦截方法,不限于 apply、ownKeys、deleteProperty、has 等等是 Object.defineProperty 不具备的；

Proxy 返回的是一个新对象,我们可以只操作新的对象达到目的,而 Object.defineProperty 只能遍历对象属性直接修改；

Proxy 作为新标准将受到浏览器厂商重点持续的性能优化，也就是传说中的新标准的性能红利；

Object.defineProperty 的优势如下:

兼容性好，支持 IE9，而 Proxy 的存在浏览器兼容性问题,而且无法用 polyfill 磨平，因此 Vue 的作者才声明需要等到下个大版本( 3.0 )才能用 Proxy 重写。

**9、Vue 框架怎么实现对象和数组的监听？**

答：如果被问到 Vue 怎么实现数据双向绑定，大家肯定都会回答 通过 Object.defineProperty() 对数据进行劫持，但是 Object.defineProperty() 只能对属性进行数据劫持，不能对整个对象进行劫持，同理无法对数组进行劫持。通过以上 Vue 源码部分查看，我们就能知道 Vue 框架是通过遍历数组 和递归遍历对象，从而达到利用 Object.define- Property() 也能对对象和数组（部分方法的操作）进行监听。

**10、使用过 Vue SSR 吗？说说 SSR？(选择性观看)**

答：Vue.js 是构建客户端应用程序的框架。默认情况下，可以在浏览器中输出 Vue 组件，进行生成 DOM 和操作 DOM。然而，也可以将同一个组件渲染为服务端的 HTML 字符串，将它们直接发送到浏览器，最后将这些静态标记"激活"为客户端上完全可交互的应用程序。

即：SSR大致的意思就是vue在客户端将标签渲染成的整个 html 片段的工作在服务端完成，服务端形成的html 片段直接返回给客户端这个过程就叫做服务端渲染。

服务端渲染 SSR 的优缺点如下：

（1）服务端渲染的优点：

更好的 SEO：因为 SPA 页面的内容是通过 Ajax 获取，而搜索引擎爬取工具并不会等待 Ajax 异步完成后再抓取页面内容，所以在 SPA 中是抓取不到页面通过 Ajax 获取到的内容；而 SSR 是直接由服务端返回已经渲染好的页面（数据已经包含在页面中），所以搜索引擎爬取工具可以抓取渲染好的页面；

更快的内容到达时间（首屏加载更快）：SPA 会等待所有 Vue 编译后的 js 文件都下载完成后，才开始进行页面的渲染，文件下载等需要一定的时间等，所以首屏渲染需要一定的时间；SSR 直接由服务端渲染好页面直接返回显示，无需等待下载 js 文件及再去渲染等，所以 SSR 有更快的内容到达时间；

（2) 服务端渲染的缺点：

更多的开发条件限制：例如服务端渲染只支持 beforCreate 和 created 两个钩子函数，这会导致一些外部扩展库需要特殊处理，才能在服务端渲染应用程序中运行；并且与可以部署在任何静态文件服务器上的完全静态单页面应用程序 SPA 不同，服务端渲染应用程序，需要处于 Node.js server 运行环境；

更多的服务器负载：在 Node.js 中渲染完整的应用程序，显然会比仅仅提供静态文件的 server 更加大量占用CPU 资源 (CPU-intensive - CPU 密集)，因此如果你预料在高流量环境 ( high traffic ) 下使用，请准备相应的服务器负载，并明智地采用缓存策略。

如果没有 SSR 开发经验的同学，可以参考本文作者的另一篇 SSR 的实践文章《Vue SSR 踩坑之旅》，里面 SSR 项目搭建以及附有项目源码。

**11、Vue 2.0 响应式数据的原理**

答：整体思路是数据劫持 + 观察者模式

对象内部通过 defineReactive 方法，使用 Object.defineProperty 将属性进行劫持（只会劫持已存在的属性），数组则是通过重写数组来实现。当页面使用对应属性时，每个属性都拥有自己的 dep 属性，存在它所依赖的 watcher （依赖收集）get，当属性变化后会通知自己对应的 watcher 去更新（派发更新）set。

1、Object.defineProperty 数据劫持  
2、使用 getter 收集依赖 ，setter 通知 watcher派发更新。  
3、watcher 发布订阅模式。

**12、axios相关内容补充**

**虚拟DOM**

**1、虚拟 DOM 的优缺点？**

答：**优点**：

保证性能下限：框架的虚拟 DOM 需要适配任何上层 API 可能产生的操作，它的一些 DOM 操作的实现必须是普适的，所以它的性能并不是最优的；但是比起粗暴的 DOM 操作性能要好很多，因此框架的虚拟 DOM 至少可以保证在你不需要手动优化的情况下，依然可以提供还不错的性能，即保证性能的下限；

无需手动操作 DOM：我们不再需要手动去操作 DOM，只需要写好 View-Model 的代码逻辑，框架会根据虚拟 DOM 和 数据双向绑定，帮我们以可预期的方式更新视图，极大提高我们的开发效率；

跨平台：虚拟 DOM 本质上是 JavaScript 对象,而 DOM 与平台强相关，相比之下虚拟 DOM 可以进行更方便地跨平台操作，例如服务器渲染、weex 开发等等。

**缺点:**

无法进行极致优化：虽然虚拟DOM 合理的优化，足以应对绝大部分应用的性能需求，但在一些性能要求极高的应用中虚拟 DOM 无法进行针对性的极致优化。

**2、虚拟 DOM 实现原理？**

答：虚拟DOM的实现原理主要包括以下3部分：

用JavaScript对象模拟真实 DOM 树，对真实 DOM 进行抽象；

diff算法 — 比较两棵虚拟DOM树的差异；

pach算法 — 将两个虚拟 DOM 对象的差异应用到真正的 DOM树。

如果对以上 3 个部分还不是很了解的同学，可以查看本文作者写的另一篇详解虚拟 DOM 的文章《深入剖析：Vue核心之虚拟DOM》

**3.为什么使用key?**

答：需要使用key来给每个节点做一个唯一标识，Diff算法就可以正确的识别此节点。

作用主要是为了高效的更新虚拟DOM。

**4、在什么阶段才能访问操作DOM？**

答：在钩子函数 mounted 被调用前，Vue 已经将编译好的模板挂载到页面上，所以在 mounted 中可以访问操作 DOM。vue 具体的生命周期示意图可以参见如下，理解了整个生命周期各个阶段的操作，关于生命周期相关的面试题就难不倒你了

**扩展题**

**1. Vue 项目进行哪些优化？**

答：这里只列举针对 Vue 的性能优化，整个项目的性能优化是一个大工程。

* 对象层级不要过深，否则性能就会差。
* 不需要响应式的数据不要放在 data 中（可以使用 Object.freeze() 冻结数据）
* v-if 和 v-show 区分使用场景
* computed 和 watch 区分场景使用
* v-for 遍历必须加 key，key最好是id值，且避免同时使用 v-if
* 大数据列表和表格性能优化 - 虚拟列表 / 虚拟表格
* 防止内部泄露，组件销毁后把全局变量和时间销毁
* 图片懒加载
* 路由懒加载
* 异步路由
* 第三方插件的按需加载
* 适当采用 keep-alive 缓存组件
* 防抖、节流的运用
* 服务端渲染 SSR or 预渲染

1. **将到来的 vue3.0 特性你有什么了解的吗？**

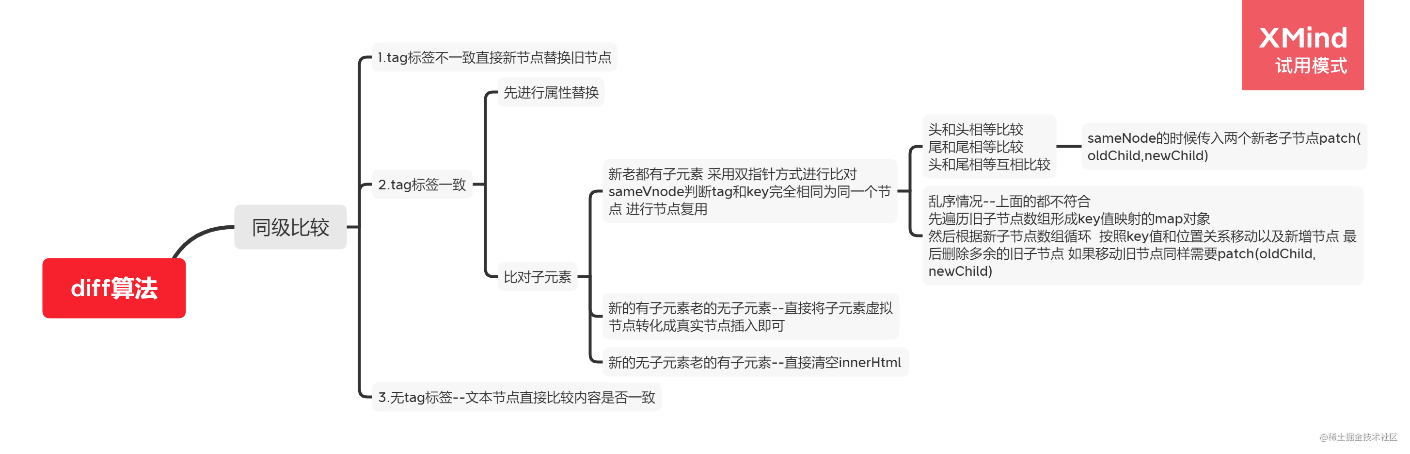
答：响应式原理的改变 Vue3.x 使用 Proxy 取代 Vue2.x 版本的 Object.define-Property。

1.组件选项声明方式 Vue3.x 使用 Composition API setup是Vue3.x新增的一个选项，他是组件内使用Composition API 的入口。

2.模板语法变化 slot 具名插槽语法，自定义指令v-model升级。

3.其他方面的更改 Suspense支持Fragment（多个根节点）和 Protal（在dom其他部分渲染组件内容）组件，针对一些特殊的场景做了处理。基于 treeShaking 优化，提供了更多的内置功能

1. **Vue 框架踩过最大的坑是什么？怎么解决的？**

**Dff算法流程**

**Vue实现双向绑定**

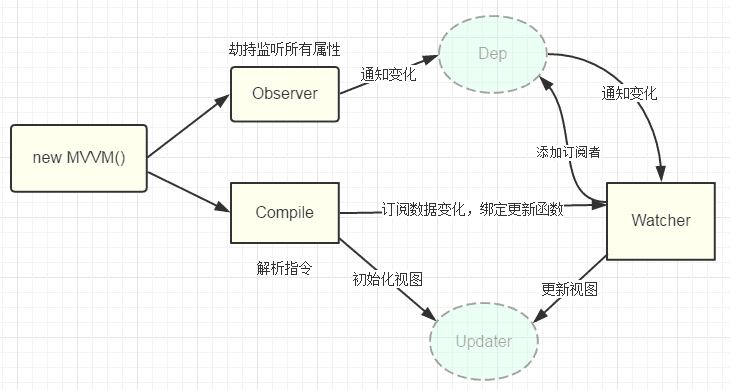
我们还是以 Vue 为例，先来看看 Vue 中的双向绑定流程是什么的

1） new Vue()首先执行初始化，对 data 执行响应化处理，这个过程发生 Observe 中；defineReactive 时为每⼀个 key 创建⼀个 Dep 实例

2） 同时对模板执行编译，找到其中动态绑定的数据，从 data 中获取并初始化视图，这个过程发生在 Compile 中；初始化视图时读取某个 key，例如 name1，创建⼀个 watcher1

3） 同时定义⼀个更新函数和 Watcher，将来对应数据变化时 Watcher 会调用更新函数

4）由于 data 的某个 key 在⼀个视图中可能出现多次，所以每个 key 都需要⼀个管家Dep来管理多个 Watcher;由于触发 name1 的 getter 方法，便将 watcher1 添加到name1对应的Dep中

5） 将来 data 中数据⼀旦发生变化,会首先找到对应的 Dep，通知所有 Watcher 执行更新函数；当 name1 更新，setter 触发时，便可通过对应 Dep 通知其管理所有 Watcher 更新

**VUE响应式原理**

理解：

1：核心点：object.defineProperty（源码中，defineReactive定义响应式）

2: 默认Vue在初始化数据时,会给data中的属性使用object.defineProperty重新定义所有属性。当页面取到对应属性时，会进行依赖收集(收集当前组件的watcher )如果属性发生变化会通知相关依赖进行更新操

**Vue如何检测数组变化**

使用函数劫持方式，重写了数组方法。（7个触及视图更新的数组操作：pop/push/shift/unshift /splice/sort/reverse）

原理：Vue将data中的数组，进行了原型链的重写（initData）。指向了自己定义的数组原型方法，这样当调用数组api时，可以通知依赖更新。如果数组中含有引用类型，会对数组中的引用类型进行再次进行监控

**Vue为何采用异步渲染**

理解：VUE是组件级更新，如果不采用异步渲染则都会对当前组件进行重新渲染。所以为了性能考虑，VUE会在本轮数据更新后，再去异步更新视图。（通知watcher更新后，依次调用watcher的update，将数据放入队列queueWatcher中，再异步情况该队列）

**NextTick实现原理？**

nextTick方法主要是使用了宏任务和微任务,定义了一个异步方法.多次调用nextTick会将方法存入队列中,通过这个异步方法清空当前队列。所以这个nextTick方法就是异步方法

**Watch 中的deep: true是如何实现的**

理解：

当用户指定了watch中的deep属性为true时,如果当前监控的值是数组类型。会对对象中的每一 项进行求值,此时会将当前watcher存入到对应属性的依赖中,这样数组中对象发生变化时也会通知数据更新

**要掌握每个生命周期内部可以做什么事**

1.created实例已经创建完成,因为它是最早触发的原因可以进行一些数据资源的请求（ajax）。2.mounted 实例已经挂载完成,可以进行一些DOM操作beforeUpdate可以在这个钩子中进一步地更改状态 ,这不会触发附加的重渲染过程。

3.updated可以执行依赖于 DOM的操作。然而在大多数情况下,你应该避免在此期间更改状态,因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。

4.destroyed可以执行一些优化操作,清空定时器,解除绑定事件

**计算属性computed和methods的对比**

计算属性中有getter与setter方法（一般没有该方法），即计算属性一般为只读属性。计算属性是基于它们的响应式依赖进行**缓存**的。只在相关响应式依赖发生改变时它们才会重新求值。（人话：Computed缓存未发生变化，直接返回原来结果，只有结果变时才调用；而methods每次都会进行计算）

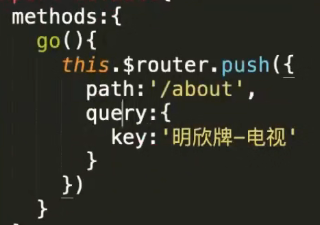
**Vue数组哪些方法是响应式的？**

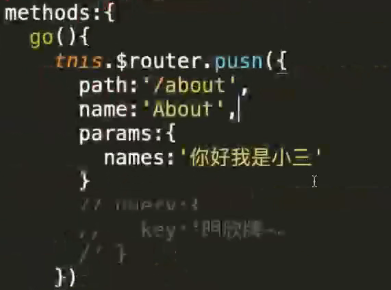
答：push方法、pop方法、shift方法、unshift方法、splice方法、Vue中的set方法Vue.set(要修改的对象，索引值，修改后的值)

不是响应式的方法：==通过索引值直接赋值（this.num[0]==5）

**生命周期都怎么用？你用哪些？**

**路由传值（显式，隐式）**

显示：query{}

隐式：name{}

**URL: 协议：//主机：端口/路径/文件名？查询**

**Nexttick**

**Axios二次封装**

**Computed/methods/watch**

**样式穿透**

**Scoped原理**

**Vue中引用组件有几种方式？分别简述步骤**

**简述在vue中使用插件的步骤**

**举例使用钩子函数。**

**异步加载，promise等**

**浏览器渲染原理、内核等相关补充知识**

**跨域问题**

**响应式布局**

Npm安装中的—save表示什么意思

**4.跨域8种解决方案**

iframe + document.domain location.hash window.name等三种

postMessage

proxyTable

nodejs 中间件

cors

websocket

jsonp

nginx反向代理

作者：进军的王小二

链接：https://juejin.cn/post/7026947170683076621

来源：稀土掘金

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

原型链？？

深拷贝浅拷贝

懒加载

手写promise，ajax

JSONP