**1.MVVM设计模式:**

1. 旧前端项目分为三部分:

(1). HTML 专门定义网页的内容

(2). CSS 专门定义网页的样式

(3). JS 专门为网页添加交互行为，负责修改HTML内容和CSS样式

2. 问题: 因为HTML和CSS缺少程序必须的要素(变量、分支和循环)，生活不能自理，所以一切变化都需要js来操作。导致js中有大量重复的编码。

3. 解决: MVVM设计模式（鄙视）: 对前端内容的重新划分: 3部分:

(1). 界面(View)：包括以前的HTML +CSS，且HTML还是增强版的。比如:

MVVM设计模式中的HTML支持变量:{{n}}

MVVM设计模式中的HTML中还可写分支和循环

(2). 数据模型(Model): 是专门保存页面所需的变量和函数 的对象，比如:

var data={ n:1 };

var methods={

add(){ ... },

minus(){ ... }

}

问题: 数据模型不会自己长腿跑到页面上

(3). 视图模型(ViewModel): 是专门自动保持页面与内存中数据同步的特殊对象(快递员)

**2.new Vue()做的第二件事: 创建虚拟DOM树（鄙视），并渲染页面内容**

a. 什么是虚拟DOM树: Vue扫描原HTML页面获得的仅保存可能发生变化的元素的简化版DOM树

b. 何时创建: 在new Vue()创建完对象后，自动扫描el:"#app"所指向的页面区域。在扫描过程中仅找出可能发生变化的元素，保存在一个新创建的虚拟DOM树集合中。

**c. 为什么使用虚拟DOM树: （好处）（鄙视）**

1). 内容少: 仅包含可能发生变化的元素，其余元素都不包含

2). 遍历快: 可快速找到受影响的元素

3). 渲染效率高: 只更新受影响的元素，不受影响的元素保持不变！

4). 已经封装了DOM增删改查操作，避免了大量重复的编码！

(4). 当在new Vue()中任何情况下修改了变量:

a. 修改变量其实修改的是变量的访问器属性

b. 修改访问器属性就会触发set()

c. 触发set()，就会自动执行set中的通知函数()

d. 通知发给虚拟DOM树，告知哪个变量的值发生了变化

e. 虚拟DOM树遍历自己内部的元素，只找到受本次变化影响的元素

f. 虚拟DOM树用已经封装好的DOM操作，只更新页面中受影响的元素.

(5). 总结: VUE绑定原理的实现其实就靠: 访问器属性 + 虚拟DOM树

**3.v-show vs v-if 区别： (高频笔试题)**

(1). V-show采用display:none方式隐藏元素，而不删除元素，不影响DOM树——效率高！

(2). V-if采用删除元素方式隐藏元素，会改变DOM树——效率低

1. **v-for的缺点和解决办法**

问题: (高频笔试题) 默认情况下，如果原数组内容中某个元素值改变，v-for会将所有元素重新生成一遍！——效率低

原因: v-for默认生成的元素除内容之外几乎都是一模一样的。当修改数组中某一个元素时，v-for是不知道该修改哪个对应元素副本的。

解决: 今后，凡是使用v-for，都要同时绑定一个:key="i"，且为:key属性绑定的值应该是不重复的下标：

<要反复生成的元素 v-for="(elem, i) of 数组" :key="i">

好处: 从此每个反复生成的元素副本上都有一个唯一的标识key属性。如果元素中某个位置的元素发生改变，则v-for完全可以根据改变的元素的下标，找到对应key属性值的一个元素副本，只更新一个元素即可！不必重新生成所有元素副本。

**总结:**

MVVM: 界面(View) 数据模型(Model) 视图模型(ViewModel)

Vue的绑定原理: 访问器属性+虚拟DOM树

new Vue({

el:"#app", //告诉new Vue()应该监控页面中哪个区域

data: { 页面所需的变量 },

methods: { 页面所需的函数 },

watch:{监听值的变化},

computed:{计算属性}

})

绑定与指令:

1. 如果元素内容可能发生变化: {{变量或js表达式}} 或 v-text="变量或js表达式"都行

2. 如果元素的属性值可能发生变化: :属性名="变量或js表达式"

3. 控制一个元素显示隐藏: v-show="条件" display:none

4. 控制两个元素二选一显示隐藏: v-if="条件" v-else 删除元素

5. 控制多个元素多选一显示隐藏: v-if="条件1" v-else-if="条件2" ... v-else

6. 反复生成多个相同结构的元素: v-for="(elem,i) of 数组/对象/数字" :key="i"

7. 绑定事件: @事件名="处理函数(实参值或$event)"

8. 防止短暂看到{{}}: v-cloak v-text

9. 绑定一段HTML内容: v-html

10. 只在首次渲染时绑定一次: v-once

11. 防止内容中的{{}}被编译: v-pre

12. 只要想获得表单元素的新值: v-model

**5.computed vs methods 高频笔试题**

(1). 用法: computed中的属性，使用时不要加()，而methods中的方法，调用时必须加()。

(2). 效率:

a. methods中的函数的执行结果，不会被缓存。只要重复调用一次，就会重新计算一次——效率低

b. computed中的计算属性的计算结果，会被vue缓存起来。只需要在首次加载时计算一次，然后即使反复使用，也不会反复计算——效率高！

除非计算属性所依赖的另一个变量值发生了变化，才会重新计算一次。但是计算后的结果，又会再次被vue缓存起来，反复使用。

(3). 如何选择:

a. 如果更侧重于获得一个计算结果的值时，首选computed

b. 如果更侧重于执行一项任务的步骤，而不关心结果时，首选methods

**6.懒加载**

(1)异步延迟加载

1). src文件夹/router文件夹/index.js中

const routes=[

{ path:"/", component: Home},

~~{ path:"/about", component: About}~~

{

     path: '/about',

  // this generates a separate chunk (about.[hash].js) for this route

//这段代码用于专门为当前这个路由生成一个独立的about.js文件片段。

     // which is lazy-loaded when the route is visited.

     component: () => import(

       /\* webpackChunkName: "about" \*/

       // 打包时的 分段 名   为About.vue单独打包成的独立js文件起一个名字，比如这里，About.vue就会被打包成about.js

       '../views/About.vue'

     )

   }

]

1. 彻底懒加载

a. 如果用户不请求下一个页面组件，则不会提前下载其他页面组件

b. 优点: 节约流量

c. 缺点: 可能会稍微影响下一个页面组件的加载速度

d. 如何:

1). 在vue脚手架项目的根目录下新建一个vue.config.js文件，添加以下内容

module.exports={

chainWebpack:config=>{

config.plugins.delete("prefetch")

//删除index.html开头的带有prefetch属性的link，不要异步下载暂时不需要的页面组件文件

}

}

2). 路由中要保持独立打包about.js的路由写法不变！

**7. 组件生命周期: 4个阶段 8个钩子函数**

beforeCreate()

(1). 创建create

created()

beforeMount()

(2). 挂载 mount

mounted()

beforeUpdate()

(3). 更新 update

updated()

beforeDestroy()

(4). 销毁 destroy

destroyed()

**8. 自定义指令:**

Vue.directive("指令名",{

inserted(domElem){

DOM操作

}

})

**9. 过滤器:**

Vue.filter("过滤器名",function(oldVal,自定义形参,...){

return 加工后的新值

})

{{变量 | 过滤器名(实参值,...) | ... }}

**10. 绑定样式:**

(1). 绑定内联样式:

<元素 style="固定不变的内联样式" :style="变量">

data:{

变量:{

Css属性: "属性值",

... : ...

}

}

(2). 绑定class

<元素 class="固定不变的class" :class="变量">

data:{

变量:{

Class名: true或false,

... : ...

}

}

**11. 创建一个全局组件**

Vue.component("组件名",{

template:`<唯一父元素 class="组件名">

组件的HTML内容

</唯一父元素>`,

data(){

return {

变量

}

},

... ...,

components:{ 子组件 }

})

**12. 父子组件传参: 2步**

(1). 父组件的template中:

<子组件 :自定义属性="父组件的变量">

(2). 子组件中:

props:[ "自定义属性" ]

结果: 子组件中接住的自定义属性，和data中的变量用法完全一样

**13. axios:**

axios.defaults.baseURL="http://xzserver.applinzi.com"

(1). 发送get请求:

axios.get("服务器端接口地址",{

params:{ 参数1:值1, ... : ... , ... }

}).then(result=>{

result.data

})

(2). 发送post请求:

axios.post("服务器端接口地址",{ 参数1:值1, 参数2:值2, ... })

.then(result=>{

result.data

})

**14. SPA：**

(1). 3步

a. 定义唯一完整的HTML文件

b. 为每个页面创建子组件对象

c. 创建路由器和路由字典

(2). 页面间跳转:

a. html中写死的跳转: <router-link to="/相对路径">文本</router-link>

b. js中用程序跳转: this.$router.push("/相对路径")

(3). 页面间传值:

a. 路由字典中: { path:"/相对路径/:变量", component: 组件对象, props:true }

b. 下个页面中: props:[ "变量" ]

c. 上个页面中想向下个页面传值:

<router-link to="/相对路径/变量值">文本</router-link>

this.$router.push("/相对路径/变量值")

**15.每个组件都是一个.vue文件，三部分:**

(1). <template> 当前组件的HTML内容

(2). <script> export default { ... } 当前组件的组件对象js内容

(3). <style scoped> 仅当前组件内可用的css样式

**16. ES6模块化开发:**

(1). 引入另个一文件中的内容: import 别名 from ".vue文件或.js文件的相对路径"

(2). 如果.vue文件或.js文件中包含想抛出的组件对象或js对象，则必须加export defaults修饰

**17.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。

# **18 JS中this的工作原理**

this永远指向函数运行时所在的对象，而不是函数被创建时所在的对象。匿名函数或不处于任何对象中的函数指向window 。

1.如果是call，apply,with，指定的this是谁，就是谁。

2.普通的函数调用，函数被谁调用，this就是谁。

**19 JS中的闭包**

闭包简单理解成定义在一个函数内部的函数，外部函数执行结束后的参数或变量被内部函数进行调用

# **20请说出三种减少页面加载时间的方法**

1. 优化图片
2. 优化CSS（压缩合并css，如margin-top,margin-left...)
3. 减少http请求（合并文件，合并图片）。

# **21 JavaScript 数据类型**

字符串 布尔值 数组 对象 数值

**22 单页面**

优点：请求次数少，因为首次请求就将所有页面组件一次性下载到本地，之后及时切换"页面"，只不过是更换本地不同的组件展现而已，无需向服务器重新发送请求。

缺点: 首屏加载太慢！

(1). 因为: 即使用户不一定会看后续页面，第一次也都要把所有页面组件的内容都要下载下来——首次下载内容太多！

(2). 解决:

a. 异步延迟下载: （脚手架默认选择）

1). 打包时，不要把所有组件打包在一个大的文件中，应该每个组件分开打包为单独的文件。

2). 首屏加载时，只强制加载第一个组件的内容。后续组件由底层程序异步下载。

3). 优点: 不影响首屏加载速度，又能实现单页面应用的效果

4). 缺点: 因为即使用户不看其他组件的内容，也会下载其他组件，浪费网络流量。

b. 懒加载: (可通过配置脚手架实现)

1). 打包时，不要把所有组件打包在一个大的文件中，应该每个组件分开打包为单独的文件。

2). 首屏加载时，不会下载其它任何组件。

3). 只有当用户确实切换到其他某个组件时，才临时下载用户想看的组件

4). 优点: 省流量

5). 缺点: 下载速度比加载本地组件稍慢。

6). 但还是单页面应用！因为之后下载的只是页面中部分的组件内容而已，而不是完整的HTML页面。

1. **组件中的缓存 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，或避免重新渲染。

1. **跨域请求的方式**

详情：<https://www.cnblogs.com/gitnull/p/9817405.html>

常用：

（1）jsonp跨域 （2）使用跨域资源共享（CORS）来跨域

（3）window.name+iframe （4） WebSocket

（5）代理 如nginx反向代理

**25.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。

**26.assets和static的区别**

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。  
**27.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。

1. **分别简述computed和watch的使用场景**  
   答：computed:  
   　　　　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:  
   　　　　当一条数据影响多条数据的时候就需要用watch  
   　　　　栗子：搜索数据

**29.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。

**30.vuex有哪几种属性？**  
答：有五种，分别是 State、 Getter、Mutation 、Action、 Module  
state => 基本数据(数据源存放地)  
getters => 从基本数据派生出来的数据  
mutations => 提交更改数据的方法，同步！  
actions => 像一个装饰器，包裹mutations，使之可以异步。  
modules => 模块化Vuex

**31.[Vue路由实现页面跳转的方式](https://www.cnblogs.com/Leophen/p/11265833.html)**

[1. vue中 router-link to=“path”](https://www.cnblogs.com/Leophen/p/11265833.html)

[2.JS](https://www.cnblogs.com/Leophen/p/11265833.html)中 this.$router.push

1. **vue-router的两种模式**  
   答:hash模式：即地址栏 URL 中的 # 符号；  
   history模式：window.history对象打印出来可以看到里边提供的方法和记录长度。利用了 HTML5 History Interface 中新增的 pushState() 和 replaceState() 方法。（需要特定浏览器支持）。
2. **简述每个周期具体适合哪些场景**  
   **beforeCreate：**在new一个vue实例后，只有一些默认的生命周期钩子和默认事件，其他的东西都还没创建。在beforeCreate生命周期执行的时候，data和methods中的数据都还没有初始化。不能在这个阶段使用data中的数据和methods中的方法  
   **create：**data 和 methods都已经被初始化好了，如果要调用 methods 中的方法，或者操作 data 中的数据，最早可以在这个阶段中操作  
   **beforeMount：**执行到这个钩子的时候，在内存中已经编译好了模板了，但是还没有挂载到页面中，此时，页面还是旧的  
   **mounted：**执行到这个钩子的时候，就表示Vue实例已经初始化完成了。此时组件脱离了创建阶段，进入到了运行阶段。 如果我们想要通过插件操作页面上的DOM节点，最早可以在和这个阶段中进行  
   **beforeUpdate：** 当执行这个钩子时，页面中的显示的数据还是旧的，data中的数据是更新后的， 页面还没有和最新的数据保持同步  
   **updated：**页面显示的数据和data中的数据已经保持同步了，都是最新的  
   **beforeDestory：**Vue实例从运行阶段进入到了销毁阶段，这个时候上所有的 data 和 methods ， 指令， 过滤器 ……都是处于可用状态。还没有真正被销毁  
   **destroyed：** 这个时候上所有的 data 和 methods ， 指令， 过滤器 ……都是处于不可用状态。组件已经被销毁了。  
   **34.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。

**35.JS中的异步操作：**

1、定时器都是异步操作

2、事件绑定都是异步操作

3、AJAX中一般我们都采取异步操作（也可以同步）

4、回调函数可以理解为异步（不是严谨的异步操作）

**36.v-model的使用。**  
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。

**37.vue-router有哪几种导航钩子？**  
第一种：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

**38.实现非父子之间通信**

let Hub = new Vue(); //创建事件中心  
组件1触发：  
methods: { eve() { Hub.$emit('change','hehe'); //Hub触发事件 } }  
组件2接收:  
created() { Hub.$on('change', () => { //Hub接收事件 this.msg = 'hehe'; }); }  
这样就实现了非父子组件之间的通信了.原理就是把Hub当作一个中转站！

**39.ES6面试题**

<https://www.cnblogs.com/fengxiongZz/p/8191503.html>

**40.css在组件中起作用**

在style标签中加入scoped

1. **怎么定义vue-router的动态路由，怎么获取传过来的值**

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