**1 对于MVVM的理解**

MVVM 是 Model-View-ViewModel 的缩写

* Model 代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑。
* View 代表UI 组件，它负责将数据模型转化成UI 展现出来。
* ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步View 和 Model的对象，连接Model和View
* 在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。
* ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM，不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理

**2 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

* 创建前/后： 在beforeCreate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

**什么是vue生命周期？**

* 答： Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

**vue生命周期的作用是什么？**

* 答：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。

**vue生命周期总共有几个阶段？**

* 答：它可以总共分为8个阶段：创建前/后、载入前/后、更新前/后、销毁前/销毁后。

**第一次页面加载会触发哪几个钩子？**

* 答：会触发下面这几个beforeCreate、created、beforeMount、mounted 。

**DOM 渲染在哪个周期中就已经完成？**

* 答：DOM 渲染在 mounted 中就已经完成了

**3 Vue实现数据双向绑定的原理：Object.defineProperty()**// Object.defineProperty() 方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。

用法:Object.defineProperty(obj, prop, descriptor) **obj**要在其上定义属性的对象。prop要定义或修改的属性的名称。descriptor将被定义或修改的属性描述符。

* vue实现数据双向绑定主要是：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，触发相应监听回调。当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty() 将它们转为 getter/setter。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。
* vue的数据双向绑定 将MVVM作为数据绑定的入口，整合Observer，Compile和Watcher三者，通过Observer来监听自己的model的数据变化，通过Compile来解析编译模板指令（vue中是用来解析 {{}}），最终利用watcher搭起observer和Compile之间的通信桥梁，达到数据变化 —>视图更新；视图交互变化（input）—>数据model变更双向绑定效果。

**4 Vue组件间的参数传递**

**父组件与子组件传值**

父组件传给子组件：子组件通过props方法接受数据；

* 子组件传给父组件： $emit 方法传递参数

**非父子组件间的数据传递，兄弟组件传值**

eventBus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。项目比较小时，用这个比较合适（虽然也有不少人推荐直接用VUEX，具体来说看需求）

**5 Vue的路由实现：hash模式 和 history模式**

* hash模式：在浏览器中符号“#”，#以及#后面的字符称之为hash，用 window.location.hash 读取。特点：hash虽然在URL中，但不被包括在HTTP请求中；用来指导浏览器动作，对服务端安全无用，hash不会重加载页面。
* history模式：history采用HTML5的新特性；且提供了两个新方法： pushState()， replaceState()可以对浏览器历史记录栈进行修改，以及popState事件的监听到状态变更

**5 vue路由的钩子函数**

首页可以控制导航跳转，beforeEach，afterEach等，一般用于页面title的修改。一些需要登录才能调整页面的重定向功能。

* beforeEach主要有3个参数to，from，next。
* to：route即将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resolve这个钩子。执行效果依赖next方法的调用参数。可以控制网页的跳转

**6 vuex是什么？怎么使用？哪种功能场景使用它？**

* 只用来读取的状态集中放在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。
* 在main.js引入store，注入。新建了一个目录store，… export
* **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
* state：Vuex 使用单一状态树,即每个应用将仅仅包含一个store 实例，但单一状态树和模块化并不冲突。存放的数据状态，不可以直接修改里面的数据。
* mutations：mutations定义的方法动态修改Vuex 的 store 中的状态或数据
* getters：类似vue的计算属性，主要用来过滤一些数据。
* action：actions可以理解为通过将mutations里面处里数据的方法变成可异步的处理数据的方法，简单的说就是异步操作数据。view 层通过 store.dispath 来分发 action

modules：项目特别复杂的时候，可以让每一个模块拥有自己的state、mutation、action、getters，使得结构非常清晰，方便管理

**7 v-if 和 v-show 区别**

* 答：v-if按照条件是否渲染，v-show是display的block或none；

**8 $route和$router的区别**

* $route是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。
* 而$router是“路由实例”对象包括了路由的跳转方法，钩子函数等

**9 如何让CSS只在当前组件中起作用?**

将当前组件的<style>修改为<style scoped>

**10 <keep-alive></keep-alive>的作用是什么?**

* <keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例,主要用于保留组件状态或避免重新渲染

比如有一个列表和一个详情，那么用户就会经常执行打开详情=>返回列表=>打开详情…这样的话列表和详情都是一个频率很高的页面，那么就可以对列表组件使用<keep-alive></keep-alive>进行缓存，这样用户每次返回列表的时候，都能从缓存中快速渲染，而不是重新渲染

**11 指令v-el的作用是什么?**

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标.可以是 CSS 选择器，也可以是一个 HTMLElement 实例,

**12 在Vue中使用插件的步骤**

* 采用ES6的import ... from ...语法或CommonJS的require()方法引入插件
* 使用全局方法Vue.use( plugin )使用插件,可以传入一个选项对象Vue.use(MyPlugin, { someOption: true })

**13 请列举出3个Vue中常用的生命周期钩子函数?**

* created: 实例已经创建完成之后调用,在这一步,实例已经完成数据观测, 属性和方法的运算, watch/event事件回调. 然而, 挂载阶段还没有开始, $el属性目前还不可见
* mounted: el被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。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。
* activated: keep-alive组件激活时调用

**14 �vue-cli 工程技术集合介绍**

**问题一：构建的 vue-cli 工程都到了哪些技术，它们的作用分别是什么？**

* vue.js：vue-cli工程的核心，主要特点是 双向数据绑定 和 组件系统。
* vue-router：vue官方推荐使用的路由框架。
* vuex：专为 Vue.js 应用项目开发的状态管理器，主要用于维护vue组件间共用的一些 变量 和 方法。
* axios（ 或者 fetch 、ajax ）：用于发起 GET 、或 POST 等 http请求，基于 Promise 设计。
* vuex等：一个专为vue设计的移动端UI组件库。
* 创建一个emit.js文件，用于vue事件机制的管理。
* webpack：模块加载和vue-cli工程打包器。

**问题二：vue-cli 工程常用的 npm 命令有哪些？**

* 下载 node\_modules 资源包的命令：

npm install

* 启动 vue-cli 开发环境的 npm命令：

npm run dev

* vue-cli 生成 生产环境部署资源 的 npm命令：

npm run build

* 用于查看 vue-cli 生产环境部署资源文件大小的 npm命令：

npm run build --report

在浏览器上自动弹出一个 展示 vue-cli 工程打包后 app.js、manifest.js、vendor.js 文件里面所包含代码的页面。可以具此优化 vue-cli 生产环境部署的静态资源，提升 页面 的加载速度

**15 NextTick**

nextTick 可以让我们在下次 DOM 更新循环结束之后执行延迟回调，用于获得更新后的 DOM

**16 vue的优点是什么？**

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

**17 路由之间跳转？**

**声明式（标签跳转）**

<router-link :to="index">

**编程式（ js跳转）**

router.push('index')

**18 实现 Vue SSR**

**其基本实现原理**

* app.js 作为客户端与服务端的公用入口，导出 Vue 根实例，供客户端 entry 与服务端 entry 使用。客户端 entry主要作用挂载到 DOM 上，服务端 entry 除了创建和返回实例，还进行路由匹配与数据预获取。
* webpack 为客服端打包一个 Client Bundle ，为服务端打包一个 Server Bundle 。
* 服务器接收请求时，会根据 url，加载相应组件，获取和解析异步数据，创建一个读取 Server Bundle 的 BundleRenderer，然后生成 html 发送给客户端。
* 客户端混合，客户端收到从服务端传来的 DOM 与自己的生成的 DOM 进行对比，把不相同的 DOM 激活，使其可以能够响应后续变化，这个过程称为客户端激活 。为确保混合成功，客户端与服务器端需要共享同一套数据。在服务端，可以在渲染之前获取数据，填充到 stroe 里，这样，在客户端挂载到 DOM 之前，可以直接从 store 里取数据。首屏的动态数据通过 window.\_\_INITIAL\_STATE\_\_ 发送到客户端

Vue SSR 的实现，主要就是把 Vue 的组件输出成一个完整 HTML, vue-server-renderer 就是干这事的 ( 使用JavaScript开发应用一般来说会有两种方向可以选择，一是spa，单页应用。另一个是ssr(server side render)

* Vue SSR 需要做的事多点（输出完整 HTML），除了 complier -> vnode，还需如数据获取填充至 HTML、客户端混合（hydration）、缓存等等。 相比于其他模板引擎（ejs, jade 等），最终要实现的目的是一样的，性能上可能要差点

**19 Vue 组件 data 为什么必须是函数**

* 每个组件都是 Vue 的实例。
* 组件共享 data 属性，当 data 的值是同一个引用类型的值时，改变其中一个会影响其他

**20 Vue computed 实现**

* 建立与其他属性（如：data、 Store）的联系；
* 属性改变后，通知计算属性重新计算

实现时，主要如下

* 初始化 data， 使用 Object.defineProperty 把这些属性全部转为 getter/setter。
* 初始化 computed, 遍历 computed 里的每个属性，每个 computed 属性都是一个 watch 实例。每个属性提供的函数作为属性的 getter，使用 Object.defineProperty 转化。
* Object.defineProperty getter 依赖收集。用于依赖发生变化时，触发属性重新计算。
* 若出现当前 computed 计算属性嵌套其他 computed 计算属性时，先进行其他的依赖收集

**21 Vue complier 实现**

* 模板解析这种事，本质是将数据转化为一段 html ，最开始出现在后端，经过各种处理吐给前端。随着各种 mv\* 的兴起，模板解析交由前端处理。
* 总的来说，Vue complier 是将 template 转化成一个 render 字符串。

可以简单理解成以下步骤：

* parse 过程，将 template 利用正则转化成 AST 抽象语法树。
* optimize 过程，标记静态节点，后 diff 过程跳过静态节点，提升性能。
* generate 过程，生成 render 字符串

**22 怎么快速定位哪个组件出现性能问题**

用 timeline 工具。 大意是通过 timeline 来查看每个函数的调用时常，定位出哪个函数的问题，从而能判断哪个组件出了问题

嵌套路由怎么定义？

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

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

简述一下Sass、Less，且说明区别？

他们是动态的样式语言，是CSS预处理器,CSS上的一种抽象层。他们是一种特殊的语法/语言而编译成CSS。

变量符不一样，less是@，而Sass是$;

Sass支持条件语句，可以使用if{}else{},for{}循环等等。而Less不支持;

Sass是基于Ruby的，是在服务端处理的，而Less是需要引入less.js来处理Less代码输出Css到浏览器

axios是什么？怎么使用？描述使用它实现登录功能的流程？

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

vuex是什么？怎么使用？哪种功能场景使用它？

vuex是vue生态系统中的状态管理。在main.js引入store，注入。新建了一个目录store，….. export 。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

Vue的双向数据绑定原理是什么？

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。

请详细说下你对vue生命周期的理解？

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

简单描述每个周期具体适合哪些场景？

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

Object.defineProperty() 方法会直接在一个对象上定义一个新属性，或者修改一个对象的现有属性， 并返回这个对象。 //该方法允许精确添加或修改对象的属性。

canvas和svg比较

1.canvas

依赖分辨率

不支持事件处理器

弱的文本渲染能力

能够以 .png 或 .jpg 格式保存结果图像

最适合图像密集型的游戏，其中的许多对象会被频繁重绘复制代码

2.SVG

不依赖分辨率

支持事件处理器

最适合带有大型渲染区域的应用程序（比如谷歌地图）

复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）

不适合游戏应用

对于MVVM的理解

MVVM 是 Model-View-ViewModel 的缩写。 Model代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑。 View 代表UI 组件，它负责将数据模型转化成UI 展现出来。 ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步View 和 Model的对象，连接Model和View。 在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。 ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

Vue组件间的参数传递

1.父组件与子组件传值 父组件传给子组件：子组件通过props方法接受数据; 子组件传给父组件：$emit方法传递参数 2.非父子组件间的数据传递，兄弟组件传值 eventBus，就是创建一个事件中心，相当于中转站，可以用它来传递事件和接收事件。项目比较小时，用这个比较合适。（虽然也有不少人推荐直接用VUEX，具体来说看需求咯。技术只是手段，目的达到才是王道。）

a链接跳转的地址

"＃"包含了一个位置信息

默认的锚点是＃top 也就是网页的上端

而javascript:void(0)  仅仅表示一个死链接

Vue的路由实现：hash模式 和 history模式

\*\*hash模式：\*\*在浏览器中符号“#”，#以及#后面的字符称之为hash，用window.location.hash读取； 特点：hash虽然在URL中，但不被包括在HTTP请求中；用来指导浏览器动作，对服务端安全无用，hash不会重加载页面。 hash 模式下，仅 hash 符号之前的内容会被包含在请求中，如 www.xxx.com，因此对于后端来说，即使没有做到对路由的全覆盖，也不会返回 404 错误。

\*\*history模式：\*\*history采用HTML5的新特性；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及popState事件的监听到状态变更。 history 模式下，前端的 URL 必须和实际向后端发起请求的 URL 一致

vue路由的钩子函数

首页可以控制导航跳转，beforeEach，afterEach等，一般用于页面title的修改。一些需要登录才能调整页面的重定向功能。

beforeEach主要有3个参数to，from，next：

to：route即将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resolve这个钩子。执行效果依赖next方法的调用参数。可以控制网页的跳转。

比较简单的一句话就可以解释

1.css只在当前组件起作用 答：在style标签中写入scoped

2.v-if 和 v-show 区别 答：v-if按照条件是否渲染，v-show是display的block或none；

3.router的区别 答：router是“路由实例”对象包括了路由的跳转方法，钩子函数等

4.vue.js的两个核心是什么？ 答：数据驱动、组件系统

5.vue几种常用的指令 答：v-for 、 v-if 、v-bind、v-on、v-show、v-else

6.vue常用的修饰符？ 答：.prevent: 提交事件不再重载页面；.stop: 阻止单击事件冒泡；.self: 当事件发生在该元素本身而不是子元素的时候会触发；.capture: 事件侦听，事件发生的时候会调用