Vue2相关面试题

一、vue2和react的区别

相同点：

1. 维护核心库，其他功能如路由和全局状态管理交给生态来做；

2. 都有自己的构建工具

3. 都使用了虚拟DOM来提高重绘性能；

4. 都有props的概念，允许组件间的数据传递；

5. 实现组件化应用，提高复用性

不同点：

1. 数据流：Vue默认支持数据双向绑定，而React一直提倡单向数据流

2. vue默认对虚拟DOM具有优化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每当应用的状态被改变时，全部子组件都会重新渲染需要手动通过PureComponent/shouldComponentUpdate这个生命周期方法来进行控制

3. vue使用类似常规的HTML模版，react推荐你所有的模板使用JSX书写。

1. Vue2通过Object.defineProperty监听数据变化,vue3通过proxy，具备比较好的性能React监听数据是通过比较引用的方式，需要手动优化

5. react使用hock拓展，Vue2使用mixins，vue3使用hooks

二、vue2和vue3区别

答：生命周期钩子不同、数据双向绑定原理不同、定义变量和方法的方式不同、指令和插槽的使用不同(v-if和v-for不在相互冲突，移除filter、使用v-slot)、vue3支持多个根节点

1. Vue优点 - 轻量级框架大小只有几十kb、易于理解和学习、上手成本低、运行速度更快
2. 什么是MVVM？

答：Model数据模型，View视图层，ViewModel视图数据层，MVVM模式不需要用户手动渲染DOM元素，而是直接将数据绑定到viewModel上面，会自动渲染数据到页面当中，试图变化会通知viewModel层更新数据，viewModel就是我们MVVM模式当中的桥梁，向下负责转换 Model 中的数据对象，向上为View做服务，解耦了 View 层和 Model 层

五、MVVM的优点：降低代码耦合，提⾼视图或者逻辑的重⽤性、利⽤双向绑定数据更新后视图⾃动更新

六、MVVM缺点：调试Bug需要依赖一些工具，ViewModel的构建和维护的成本都会⽐较⾼会影响性能

七：库：将代码集合成一个产品，我们调用库中的方法来实现自己的功能

八、框架是为了解决一类问题而开发的产品，框架是我们在指定的位置编写好代码，框架帮我们调用

九、Vue 单页应用与多页应用的区别

1. SPA单页面应用指只有一个主页面的应用，一开始只需要加载一次js、css等相关资源。所有内容都包含在主页面对每一个功能模块组件化。单页应用跳转切换相关组件刷新局部资源。

2. MPA多页面应用指有多个独立页面的应用，每个页面必须重复加载js、css等相关资源。多页应用跳转，需要整页资源刷新。

十、请说一下vue响应式数据的理解？（Vue的响应式原理）

1. 通过Object.defineProperty/proxy对data的getter/setter方法进行拦截，利用发布订阅者模式，在getter方法中进行订阅，在setter方法中通知，让所有订阅者完成响应。data的每一个属性都新建一个订阅中心作为发布者，对于watcher、computed、template/render作为订阅者，watcher会直接订阅观察监听的属性，对于computed和template/render，如果内部执行获取了data的某一个属性值，就会执行这个属性的getter方法，然后自动完成对该属性的订阅，当属性被修改后，就会执行这个属性的setter方法，从而完成这个属性的发布通知，通知所有的订阅者进行更新。

十一、Vue 是如何实现数据双向绑定的？

数据双向绑定是数据变化更新视图视图变化更新数据，View变化更新Data可以通过事件监听的方式来实现，主要通过以下 4 个步骤来实现数据双向绑定的：

1. 实现一个监听器 Observer：对数据对象进行遍历，包括子属性对象的属性，利用 Object.defineProperty() 对属性都加上 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。

1. 2. 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。

3. 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。

4. 订阅器 Dep采用发布-订阅设计模式，用来收集订阅者 Watcher，对监听器 Observer 和 订阅者 Watcher 进行统一管理。

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数据变化，通过Compile来解析编译模板指令，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变更的双向绑定效果。

十二、vue中是如何检测数组变化？

vue2当中通过函数劫持的方式重写了数组的方法，利用原型链的特性，每次调用这些方法的时候实际上走的是我们自己重写的方法，vue3当中改用proxy，可以直接监听数组的变化

十三、使用 Object.defineProperty来进行数据劫持有什么缺点？

1. 对象默认只监控自带的属性，新增的属性响应式不生效 (层级过深，性能差)

2. 数组通过索引进行修改 或者 修改数组的长度，响应式不生效

为了解决以上问题，vm.$set -> 修改数组内部使用的是splice方法 和vm.$delete

十四、Proxy与Object.defineProperty优劣对比

1. Proxy 可以直接监听对象而非属性，可以直接监听数组的变化

2. Proxy 有多达13种拦截方法，不限于 apply、ownKeys、deleteProperty

3. Proxy 返回的是一个新对象我们可以只操作新的对象而Object.defineProperty只能遍历对象属性修改

4. Proxy 如果对象内部要全部递归代理，则Proxy可以只在调用时递归，而Object.defineProperty需要在一开始就全部递归，Proxy性能优于Object.defineProperty

5. Object.defineProperty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

十五、怎么理解vue的单向数据流？

父子组件之间的通信是单向的，是为了防止子组件意外的改变父组件的状态使得数据流难以理解，每次父级组件发生更新时，子组件中所有的props都会更新，子组件通过$emit派发一个自定义事件，父组件接收到后，由父组件修改数据

十六、vue数据的丢失 - 初始化时没有定义数据，之后更新的数据是无法触发页面渲染更新的

十七、vue 中给data中的对象属性添加一个新的属性时会发生什么？如何解决？

不会发生试图更新；需要使用$set方法相当于手动的去把新增加的对象属性处理成一个响应式的属性，此时视图也会跟着改变了。

十八、data中某一个属性的值发生改变后，视图会立即同步执行重新渲染吗？

不会立即同步执行重新渲染，Vue实现响应式并不是数据发生变化之后DOM立即变化，而是按一定的策略进行DOM的更新。在更新DOM时是异步的，监测到数据变化Vue 将开启一个队列进行缓冲，在同一事件循环中发生的所有数据变更。如果同一个watcher被多次触发，只会入到队列中一次，因为在缓冲时会去除重复数据，避免不必要的计算和DOM操作，在下一个的事件循环中会刷新队列并执行实际的工作。

十九、在data上面定义了一个$name属性，可以通过Vue.$name拿到吗？

不可以，因为vue内部将所有$开头的属性都默认为是vue本身内置的属性

二十、直接给一个数组项赋值，Vue 能检测到变化吗？

不能，数组考虑性能原因没有用defineProperty对数组的每一项进行拦截，而是选择重写数组方法，在Vue中修改数组的索引和长度是无法监控到的。需要通过上面的方法修改数组才会触发数组对应的watcher进行更新。数组中如果是对象数据类型也会进行递归劫持。可以通过Vue.$set来进行处理, 其核心内部用的是splice方法，针对修改数组的长度使用\*\*vm.items.splice(newLength)\*\*解决

二十一、 Vue中封装的数组方法有哪些，其如何实现页面更新

push,shift,pop,splice,unshift,sort,reverse；首先获取到这个数组的\_\_ob\_\_即Observer对象，如果有新的值就调用observeArray继续对新的值观察变化（也就是通过target.\_\_proto\_\_ == arrayMethods来改变了数组实例的原型）然后手动调用notify通知渲染watcher执行update

二十二、Vue怎么用vm.$set解决数组对象新增属性不能响应 - 通过调用重写的数组splice方法实现

二十三、给响应式数据添加标识为什么不可以直接使用value.\_\_ob\_\_ = this

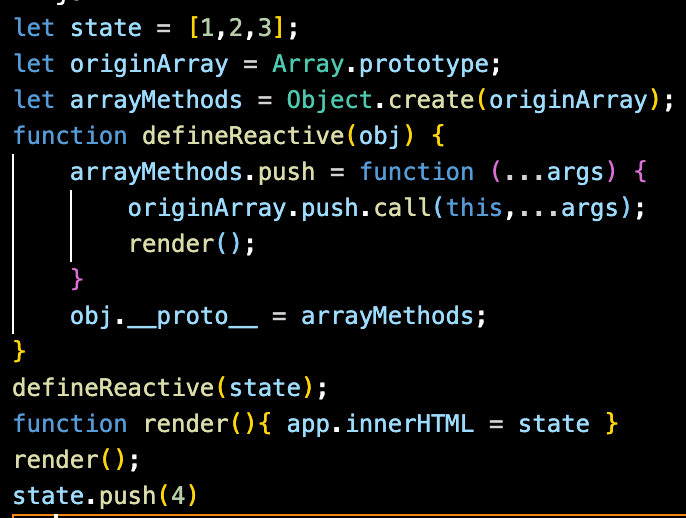
因为value是一个被观测得响应式的值如果这样赋值就会触发Object.defineProperty方法set方法，此时无限递归会陷入死循环，通过Object.defineProperty方法添加\_\_ob\_\_属性这个属性是不能被枚举不能被循环出来的，所以不会有影响

二十四、vue的构造函数为什么没有使用class语法糖

vue构造函数的方法很多，如果使用class进行维护会篇幅很大，对于重写方法的行为也不是很友好，所以还是采用了构造函数的方式，在其原型上增加方法进行拓展

二十五、定义响应式数据的时候在defineReactive的set方法内部，不直接value[key]赋值？会陷入死循环

二十六、手动实现一个proxy的响应式 二十七、Vue是如何重写数组方法的？

二十八、vue为什么采用异步渲染？

vue的数据更新是采用组件级别的更新策略会节省性能，vue会将同一个组件的watcher更新过滤在一起在合适的时机更新，主要操作是：dep.notify通知watcher进行数据更新，然后依次调用watcher的update方法，通过queueWatcher方法将多个属性依赖的相同的watcher整合成一个（每一个watcher都有一个id相同id的整合在一起主要是预防多次更新）最后通过nextTick方法异步清空watcher队列，vue会在数据更新了之后，再去异步更新视图，提高了性能

二十九、Vue中模板编译原理？

模板编译：vue中的模板template无法被浏览器解析并渲染，因为不是正确的HTML语法，所有需要将template转化成一个JavaScript函数，这样浏览器就可以执行这一个函数并渲染出对应的HTML元素，就可以让视图跑起来了。模板编译分为解析parse、优化optimize、生成generate，最终生成可执行函数render。

1. parse阶段：使用大量正则表达式对template字符串进行解析将标签指令属性等转化为抽象语法树AST。
2. optimize阶段：遍历AST找到其中的一些静态节点并进行标记方便在页面重渲染的时候进行diff比较时，直接跳过这一些静态节点，优化runtime的性能。3. generate将最终AST转化为render函数字符串。

三十、template预编译是什么？

预编译在项目构建的编译template的过程。模板编译只会在组件实例化的时候编译一次生成渲染函数之后在也不会进行编译。在项目构建时可以让实际组件在 runtime 时直接跳过模板渲染，进而提升性能

三十一、vue 是怎么解析template的? template会变成什么?

源码会执行 compileToFunctions 将template转化为render函数，调用parse方法将template转化为ast对静态节点做优化，generate将ast抽象语法树编译成 render字符串并将静态部分放到 staticRenderFns 中，最后通过 new Function(``render``) 生成render函数

三十二、模板引擎的实现原理 - 就是\*\*new Function + with\*\*来进行实现的

三十三、computed 什么时候初始化 - vue创建实例的时候

三十四、computed 的缓存是怎么做

computed本身就是一个计算属性类型的watcher，缓存是通过控制watcher实例对象的dirty属性做到的，computed将lazy赋值dirty给一个初始值，让watcher控制缓存的任务开始，如果 computed依赖的数据变化，dirty=true，调用evalute重新计算然后更新缓存值watcher.value；computed的A依赖B，就会将B收集到A的watcher种，B变了就会通知A更新，就会判断dirty = true重新计算否则走缓存

三十五、computed是怎么可以直接使用实例访问到的

在defineComputed函数当中通过`Object.defineProperty(target, key, sharedPropertyDefinition)`将computed的所有属性都定义在了vm上面，所以可以直接使用实例访问到的

三十六、computed是怎样进行计算的

1. 页面更新，读取 computed 的时候，Dep.target 会设置为 页面 watcher。

2. computed被读取，createComputedGetter包装的函数触发，第一次会进行计算

3. computed-watcher.evaluted 被调用，进而computed-watcher.get被调用，Dep.target被设置为computed-watcher，旧值页面 watcher 被缓存起来。computed 计算会读取 data，此时 data 就收集到 computed-watcher，同时 computed-watcher 也会保存到 data 的依赖收集器 dep（用于下一步）。computed 计算完毕，释放Dep.target，并且Dep.target 恢复上一个watcher（页面watcher）

4. 手动 watcher.depend， 让 data 再收集一次 Dep.target，于是 data 又收集到 恢复了的页面watcher，从而使得页面A与数据C关联上

三十七、computed与watch的区别

1. 计算属性computed和监听器watcher都可以观察属性的变化从而做出响应
2. computed本质上就是一个watcher是作为缓存功能的观察者，可以将data的属性进行复杂的计算生成一个新的值提供给渲染函数使用，依赖发生变化computed不会立即重新计算而是先标记为脏数据下次computed被获取会重新计算并返回而watch并不具备缓存性，监听器watch提供一个监听函数，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

三十八、computed与watch的使用场景

1. 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
2. 当我们需要在数据变化时执行异步或开销较大的操作时使用watch，使用watch选项允许我们执行异步操作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，计算属性无法做到的。

三十九、在使用计算属性时，函数名和 data 数据源中的数据可以同名吗？

不可以重名，props，methods 都不可以重名，因为 Vue 会把这些属性挂载在组件实例上，直接使用 this 就可以访问，如果重名就会导致冲突。

四十、watch的原理

watch用于监控用户的data变化，数据变化后会触发对应的watch的回调方法,本质调用的是vue实例的$watch方法，$watch内部创建watcher，如果用户传进来的还有一些特殊的操作如immediate会借助vue响应式原理，默认在取值时会将这个旧值保存下来，然后将watcher存放到对应属性dep中当数据发生变化时通知对应的watcher重新执行getter函数拿到返回的结果作为旧值执行回调函数将新旧值作为参数

四十一、组件中data为什么是一个函数？

因为组件是用来复用但是JS里对象是引用关系如果是对象作用域没有隔离data属性值会相互影响，函数可以保证每个实例可以维护一份被返回对象的独立的拷贝，组件实例之间的data属性值不会互相影响；

四十二、new Vue -data为啥是对象 - new Vue的实例是不会被复用的，因此不存在引用对象的问题。

四十三、 Vue 组件间通信有哪几种方式？

1. 父子组件间通信- A.props/ $eimt B. $refs/$parent C. provide/inject
2. 兄弟组件间通信 - eventBus（$emit / $on)、$parent/$ref
3. 任意组件之间 - eventBus、vuex

四十四、eventBus本质 - 通过创建一个空的 Vue 实例来作为消息传递的对象，通信的组件引入这个实例，通信的组件通过在这个实例上监听和触发事件，来实现消息的传递。

四十五、vue组件会在什么时候下被销毁？

1. 当v-if为false的时候 2. 父组件销毁的时候 3. 调用this.$destory的时候 4. 页面关闭，路由跳转（没有使用keep-alive时的路由切换）

四十六、如何保存页面的当前的状态

2种情况前组件会被卸载和前组件不会被卸载

1. 前组件会被卸载:

A. LocalStorage / SessionStorage

优点：兼容性好，不需要额外库或工具,简单快捷，基本可以满足大部分需求。

缺点：状态通过 JSON 方法储存（相当于深拷贝），如果状态中有特殊情况（比如 Date 对象、Regexp 对象等）的时候会得到字符串而不是原来的值,如果 B 组件后退或者下一页跳转并不是前组件，那么 flag 判断会失效，导致从其他页面进入 A 组件页面时 A 组件会重新读取 Storage，会造成很奇怪的现象

1. 路由传值

优点：简单快捷，不会污染 LocalStorage / SessionStorage;可以传递 Date、RegExp 等特殊对象

缺点：如果 A 组件可以跳转至多个组件，那么在每一个跳转组件内都要写相同的逻辑。

2. 组件不会被卸载

A. 单页面渲染: 要切换的组件作为子组件全屏渲染，父组件中正常储存页面状态。

优点：代码量少,不需要考虑状态传递过程中的错误

缺点：增加 A 组件维护成本,需要传入额外的 prop 到 B 组件,无法利用路由定位页面

B. keep-alive来缓存页面,当组件在keep-alive内被切换时组件的activated、deactivated这两个生命周期钩子函数会被执行 被包裹在keep-alive中的组件的状态将会被保留

四十七、vue.js中如何实现路由嵌套

首先实例化根组件，在根组件中定义组件渲染容器，然后，挂载路由，当切换路由时，将会切换整个页面。

四十八、template和jsx的有什么分别？

都是render的一种表现形式，不同的是：JSX比template具有更高的灵活性复杂的组件中具有优势，template 在代码结构上更符合视图与逻辑分离的习惯，更简单、更直观、更好维护

四十九、组件渲染和更新过程

渲染组件时，会通过 vue.extend() 方法构建子组件的构造函数，并进行实例化。最终手动调用 $mount() 进行挂载。更新组件时会进行 patchVnode 流程，核心就是 diff 算法。

五十、为什么要使用异步组件？

节省打包出的结果，异步组件分开打包，采用jsonp的方式进行加载，有效解决文件过大的问题；核心就是包组件定义变成一个函数，依赖import语法，可以实现文件的分割加载。

五十一、谈谈你对 Vue 生命周期的理解？

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、卸载等一系列过程这些函数方便用户在特定阶段有机会添加他们自己的代码

Vue生命周期总共可以分为8个阶段：创建前后, 载入前后, 更新前后, 销毁前后，以及一些特殊场景的生命周期。vue3中新增了三个用于调试和服务端渲染场景

1. beforeCreate：组件实例被创建之初，组件的属性生效之前，在这个阶段只有默认的生命周期函数以及默认的事件，在当前阶段data，methods、computed以及watch上的数据和方法都不能被访问

2. created：组件实例已经完全创建，属性也绑定，但真实dom还没有生成$el还不可用已经完成了数据监测，data和methods初始化完毕可以使用数据更改数据，但是在这里修改的数据不会触发updated函数。可以做一些初始数据的获取，在当前阶段无法与DOM进行交互可以通过vm.$nextTick来访问DOM

3. beforeMount：在挂载开始之前被调用render函数首次被调用，在这之前template模版已经导入渲染函数编译；而当前虚拟dom已经创建完成即将开始渲染，此时修改DOM不会触发updated

4. mounted：在挂载完成后发生el被新创建vm.$el替换，并挂载到实例上去之后调用该钩子此时真实的dom挂载完毕数据完成双向绑定可以访问到真实的dom节点使用$refs属性对dom进行操作

5. beforeUpdate：在更新之前，发生在虚拟DOM打补丁之前，也就是响应式数据更新，虚拟dom重新渲染之前被触发，可以在当前阶段\*\*更改数据\*\*，不会造成重新渲染

6. updated：发生在更新完成之后，当前阶段组件dom已经更新完成

7. beforeDestory：发生在实例销毁之前此时实例完全可以被使用，可以用来清除定时器和复原一些状态

8. destroyed：在实例销毁后剩下了dom空壳。组件已被拆解数据绑定被卸除监听被移出子实例被销毁

keep-alive - activated激活时调用、deactivated组件停用时调用

<keep-alive>包裹动态组件时会缓存不活动的组件实例，而不是销毁它们。<keep-alive>是一个抽象组件，它自身不会渲染一个DOM元素，也不会出现在父组件链中。

五十二、Vue 的父组件和子组件生命周期钩子函数执行顺序？

1. 加载渲染过程：父 beforeCreate -> 父 created -> 父 beforeMount -> 子 beforeCreate -> 子 created -> 子 beforeMount -> 子 mounted -> 父 mounted

2. 子组件更新过程: 父 beforeUpdate -> 子 beforeUpdate -> 子 updated -> 父 updated

3. 父组件更新过程: 父 beforeUpdate -> 父 updated

4. 销毁过程:父 beforeDestroy -> 子 beforeDestroy -> 子 destroyed -> 父 destroyed

五十三、在哪个生命周期内调用异步请求？

可以在钩子函数 created、beforeMount、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。推荐在 created 钩子函数中调用异步请求可以更快获取到服务端数据，减少页面 loading 时间

五十四、在什么阶段才能访问操作DOM？

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。

五十五、父组件可以监听到子组件的生命周期吗？

1. 子组件在mounted函数当中通过this.$emit('mounted')触发父组件的事件, 父组件监听到子组件挂载 mounted就绑定处理事件@mounted="something"

2. 在父组件引用子组件时通过 @hook 来监听 \*\*@hook:mounted="doSomething"\*\*, 其它的生命周期事件也可以通过hook的方式

五十六、生命周期钩子是如何实现的?

vue内部主要是使用callHook方法来调用对应的方法。核心是一个发布订阅模式，将钩子订阅好（内部采用数组的方式存储），在对应的阶段进行发布

生命周期两种形式，用户自定义好的生命周期和Vue的全局静态方法Vue.mixin，Vue采用了生命周期函数合并策略使用队列存储好每一类生命周期具备的函数集合，先订阅好后续在特定的时机依次执行即可

五十七、初始化vue实例上面的方法按照props -> methods -> data -> computed -> watch顺序加载

五十八、vue模版当中的优先级？ - render > template > el

五十九、vue 中怎么重置 data并且原理是什么？ - 使用`Object.assgin(this.$data, this.$option.data())`

这个方法就是一个简单的对象合并的方法，`this.$options.data()`就是在组件内部书写的 `data` 函数，执行这个函数就会返回一份初始的 data 数据

六十、你所了解的props是怎样的? props 是子组件访问父组件数据的唯一接口

六十一、 created和mounted的区别

+ 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。

六十二、Vue为什么没有类似于React中shouldComponentUpdate的生命周期？

React的Virtual Dom Diff进行差异检测,很多组件肯定不会发生变化需要手动操作来减少diff从而提高程序整体的性能、Vue在一开始就知道那个组件发生了变化并不需要手动控制diff,而组件内部采用的diff方式实际上是可以引入类似于shouldComponentUpdate相关生命周期的,但是通常合理大小的组件不会有过量的diff,手动优化的价值有限

六十三、vue 的keep-alive的作用是什么？（谈谈你对keep-alive的理解）

如果需要在组件切换的时候，保存一些组件的状态防止多次渲染，就可以使用 keep-alive 组件包裹需要保存的组件。keep-alive有以下三个属性：

include 字符串或正则表达式，只有名称匹配的组件会被匹配；

exclude 字符串或正则表达式，任何名称匹配的组件都不会被缓存；

max 数字，最多可以缓存多少组件实例。

keep-alive 包裹动态组件时，会缓存不活动的组件实例。

六十四、keep-alive的流程

1. 判断组件 name 不在 include 或者在 exclude 中直接返回 vnode，说明该组件不被缓存。
2. 获取组件实例 key 如果有获取实例的 key，否则重新生成。
3. key生成规则，cid +"∶∶"+ tag ，不能只有cid因为相同的构造函数可以注册为不同的本地组件。
4. 如果缓存对象内存在，则直接从缓存对象中获取组件实例给 vnode ，不存在则添加到缓存对象中。
5. 最大缓存数量，当缓存组件数量超过 max 值时，清除 keys 数组内第一个组件。

六十五、keep-alive的实现原理

从源码的角度分析组件是自动执行render函数，keep-alive 具体是通过 cache 数组缓存所有组件的 vnode 实例，当include和exclude发生改变时会对cache对象进行遍历取出每一项的name值与新的缓存规则进行匹配，获取组件的key值后去this.cache对象中去寻找是否有该值，如果有命中缓存，如果匹配不上，会将该组件 key 从 keys 数组中删除，然后 push 到 keys数组最后，以便清除最不常用组件。在组件销毁的时候执行pruneCacheEntry函数

附加题：缓存后如何获取数据

beforeRouteEnter、actived

六十六、keep-alive的实现步骤

1. 获取 keep-alive 下第一个子组件的实例对象，通过他去获取这个组件的组件名
2. 通过当前组件名去匹配原来 include 和 exclude，判断当前组件是否需要缓存，不需要缓存，直接返回当前组件的实例vNode
3. 需要缓存，判断他当前是否在缓存数组，存在在原来位置上的 key 给移除，同将组件key放到数组最后面，不存在，将组件 key 放入数组判断当前 key数组是否超过 max 所设置的范围超过那么削减未使用时间最长的一个组件的 key
4. 最后将这个组件的 keepAlive 设置为 true

六十七、keep-alive 本身的创建过程和 patch 过程

缓存渲染会根据 vnode.componentInstance = undefined和 keepAlive 属性判断不会执行组件的created、mounted 等钩子函数，而是对缓存的组件执行 patch 过程直接把缓存的 DOM 对象直接插入到目标元素中，完成了数据更新的情况下的渲染过程。组件的首次渲染判断组件的 abstract 属性，才往父组件里面挂载 DOM，判断当前 keepAlive 和 componentInstance 是否存在来判断是否要执行组件 prepatch 还是执行创建 componentlnstance；prepatch 操作就不会在执行组件的 mounted 和 created 生命周期函数，而是直接将 DOM 插入

六十八、 LRU （least recently used）缓存策略

LRU 缓存策略∶ 从内存中找出最久未使用的数据并置换新的数据。根据数据的历史访问记录来进行淘汰数据，其核心思想是 "如果数据最近被访问过，那么将来被访问的几率也更高"。 最常见的实现是使用一个链表保存缓存数据

1. 新数据插入到链表头部 2. 每当缓存命中（即缓存数据被访问），则将数据移到链表头部 3. 链表满的时候，将链表尾部的数据丢弃。

六十九、keep-alive的生命周期

deactivated、activated。同时，beforeDestroy 和 destroyed 就不会再被触发了，因为组件不会被真正销毁。当组件被换掉时，会被缓存到内存中、触发 deactivated 生命周期；当组件被切回来时，再去缓存里找这个组件、触发 activated钩子函数

七十、什么是 mixin

Mixin使Vue 组件编写可插拔和可重用的功能。如果希望在多个组件之间重用一组组件选项，例如生命周期 hook、方法等可以编写为 mixin，并在组件中简单的引用它。然后将 mixin 的内容合并到组件中。如果你要在 mixin 中定义生命周期 hook，那么它在执行时将优化于组件自已的 hook。

附加题：mixin的合并策略

主要是调用merOptions方法，优先递归处理 mixins，先遍历合并parent 中的key，调用mergeField方法进行合并，然后保存在变量options，再遍历 child合并补上parent中没有的key，调用mergeField方法进行合并，保存在变量options，最后通过 mergeField 函数进行了合并

附加题： Vue的几种类型的合并策略

替换型策略有props、methods、inject、computed，就是将新的同名参数替代旧的参数

合并型策略是data, 通过set方法进行合并和重新赋值

队列型策略有生命周期函数和watch，原理是将函数存入一个数组，然后正序遍历依次执行

叠加型有component、directives、filters，通过原型链进行层层的叠加

七十一、vue中mixins的属性/方法/生命周期函数的优先级

1. 对于data定义属性，组件中定义属性覆盖mixins中同名字段

2. 对于created、mounted等生命周期函数，mixins中生命周期函数优先执行（执行顺序按mixins中顺序），再执行组件中生命周期函数

3. 对于methods中的同名方法，组件内的方法覆盖mixins中的方法

注：对于相同的computed属性，组件的computed属性覆盖mixins内的computed属性，而对于相同的watch监听，mixins中的watch监听先执行。

七十二、简述 mixin、extends 的覆盖逻辑

mixin和 extends均是用于合并、拓展组件的，两者均通过mergeOptions方法实现合并

1. mixins 接收一个混入对象的数组，其中混入对象可以像正常的实例对象一样包含实例选项，这些选项会被合并到最终的选项中。Mixin 钩子按照传入顺序依次调用，并在调用组件自身的钩子之前被调用。
2. extends 主要是为了便于扩展单文件组件，接收一个对象或构造函数
3. data/provide：只会将自己有的但是组件上没有的内容混合到组件上，重复定义默认使用组件上的，如果data里面的值是对象，将递归内部对象继续按照这个策略进行合并；
4. props/methods/inject/computed/组件、过滤器、指令属性/el/propsData：只会将自己有的但是组件上没有的内容混合到组件上
5. watch：合并watch监控的回掉方法，执行顺序是将mixins/entends里面的watch定义的回掉函数然后才是组件的回掉函数；
6. Hooks生命周期钩子同一种钩子的回掉函数会被合并成数组，执行顺序是先mixins/entends里面定义的钩子函数，然后才是组件内部的

七十三、mergeOptions 的执行过程

1. 规范化选项（normalizeProps、normalizelnject、normalizeDirectives)

2. 对未合并的选项，进行判断合并处理。

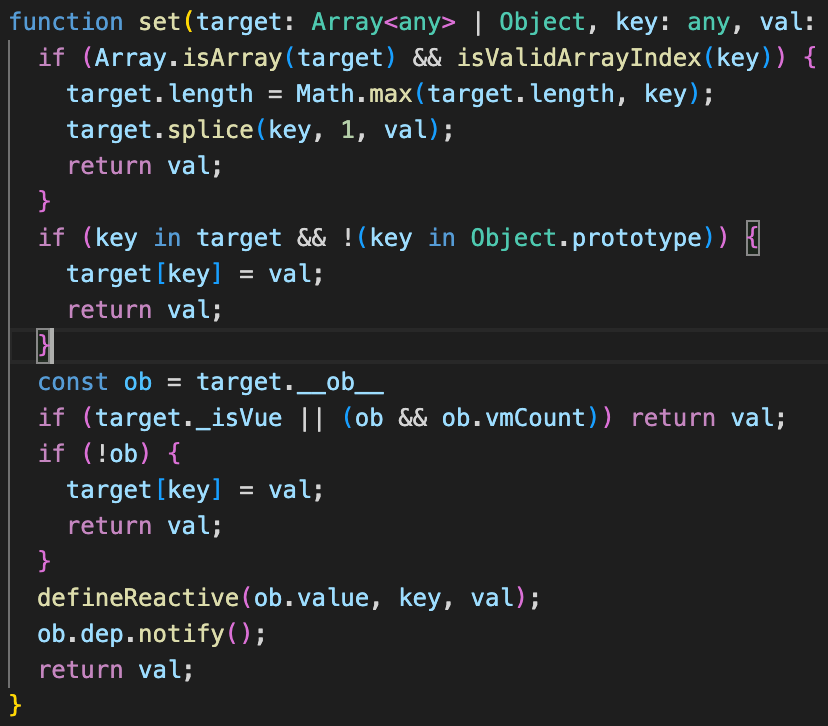
3. 根据一个通用 Vue 实例所包含的选项进行分类逐一判断合并，如 props、data、 methods、watch、computed、生命周期等，将合并结果存储在新定义的 options 对象里。

4. 返回合并结果 options。

七十四、vm.$set 的实现原理是

如果目标是数组，直接使用数组的 splice 方法触发相应式；如果目标是对象，会先判读属性是否存在、对象是否是响应式，最终如果要对属性进行响应式处理，则是通过调用 defineReactive 方法进行响应式处理defineReactive方法就是 Vue 给对象属性采用Object.defineProperty动态添加getter和setter的功能

七十五、手写vm.$set 七十六、手写EventEmitter，实现on/emit/off方法

七十七、$nextTick 原理

nextTick其本质是对 JavaScript 执行原理EventLoop的一种应用。nextTick 的核心是把回调函数放入callbacks等待执行，利用了如Promise 、MutationObserver、setImmediate、setTimeout的原生 JavaScript 方法来模拟对应的微/宏任务的实现，本质是为了利用 JavaScript 这些异步回调任务队列来实现 Vue 框架中自己的异步回调队列，事件循环到了微任务或者宏任务执行函数依次执行callbacks中的回调

nextTick允许开发者在实际项目中使用这个方法来满足实际应用中对DOM更新数据时机的后续逻辑处理

同步更新，则多次对一个或多个属性赋值，会频繁触发 UI/DOM 的渲染，可以减少一些无用渲染

七十八、$nextTick的作用

1. 在数据变化后执行的某个操作，而这个操作需要使用随数据变化而变化的DOM结构的时候，这个操作就需要方法在nextTick()的回调函数中。
2. 如果在created()钩子进行DOM操作也一定要放在nextTick()的回调函数中因为在created钩子函数中页面DOM还未渲染没办法操作DOM想要操作DOM必须将操作代码放在nextTick回调函数中

七十九、怎么给 vue 定义全局的方法？- 将方法挂载到Vue.prototype上面

八十、delete和Vue.delete删除数组的区别

delete 只是被删除的元素变成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。

八十一、v-show和v-if有什么区别？

1. v-if是动态的向DOM树内添加或者删除DOM元素；v-show是通过设置DOM的display样式控制显隐

2. 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；v-show只是简单的基于css切换；

3. v-if是惰性的，如果初始条件为假啥不做；只有在条件第一次变为真时才开始局部编译; v-show是在任何条件下无论首次条件是否为真都被编译然后被缓存而且DOM元素保留；

4.v-if有更高的切换消耗；v-show有更高的初始渲染消耗；v-if适合不怎么改变；v-show适合频繁切换

八十二、v-for和v-if哪个优先级会更高

vue2实践中不应该把v-for和v-if放一起。在vue2中，v-for的优先级是高于v-if，把它们放在一起，输出的渲染函数中可以看出会先执行循环再判断条件，哪怕我们只渲染列表中一小部分元素，也得在每次重渲染的时候遍历整个列表，这会比较浪费；另外需要注意的是在vue3中则完全相反，v-if的优先级高于v-for，所以v-if执行时，它调用的变量还不存在，就会导致异常；通常我们会在过滤列表和避免渲染本应该被隐藏的列表时候联用，此时我们可以通过computed或者包裹一层yemplate来进行避免连用的状况

八十三、v-model是如何实现的，语法糖实际是什么？（v-model的原理是什么？）

1. v-model使用在表单元素上： 动态绑定了input的value指向了message变量，并且在触发input事件的时候去动态把message设置为目标值，（$event.target指代当前触发的事件对象的dom）; text & textarea -> value属性 + input; checkbox/radio -> checked属性和change事件；select -> value作为prop并将change作为事件。

2. v-model作用在组件上：在自定义组件中，子组件通过this.$emit(‘input’,value)来对父组件传值，父组件接受到之后让e.target.value赋值给input中的value从而实现组件内部暴露出组件的值到 v-model所绑定的值中去，本质是原生控件绑定事件捕捉到原生组件的值利用$emit方法触发input方法组件监听到input事件然后把值传入到value中

八十四、v-model 可以被用在自定义组件上吗？如何使用？ - 使用相当于组件绑定value和input事件

八十五、v-if、v-show、v-html 的原理

1. v-if会调用addIfCondition方法，生成vnode的时候会忽略对应节点，render的时候就不会渲染；
2. v-show会生成vnode，render的时候也会渲染成真实节点，只是在render过程中会在节点的属性中修改show属性值，也就是常说的display；
3. v-html会先移除节点下的所有节点，调用html方法，通过addProp添加innerHTML属性，归根结底还是设置innerHTML为v-html的值。

八十六、v-html会导致什么问题？ - XSS攻击、v-html会替换标签内部的元素

八十七、vue.js中常用指令

v-if判断对象是否隐藏；v-for循环渲染；v-bind绑定一个属性（可以简写为：）v-model实现数据双向绑定（支持.trim .number修饰符）、v-once 渲染一次、v-html将字符串转换成dom插入到标签当中、v-show不满足是dom隐藏（不可以使用在template标签上面）、v-on 可以简写成@给元素绑定事件（常用修饰符.stop .prevent .self .once .passive）

八十八、在vue中说说你知道的自定义指令

自定义指令两种：

1. 一种`全局自定义指令`，自定义指令directive方法接收两个参数，一个是`指令名称`，另一个是`函数`；

2. 第二种是`局部自定义指令`，通过组件的`directives`属性定义。

指令定义对象可以提供如下几个钩子函数:

1. bind：只调用一次，指令第一次绑定到元素时调用 bind(el, bindings, vnode, oldVode)，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 inserted(el)（仅保证父节点存在，但不一定已被插入文档中）

3. update：所在组件的VNode更新时调用，但是可能发生在其子VNode更新之前调用。指令的值可能发生了改变，也可能没有。但是可以通过比较更新前后的值来忽略不必要的模板更新

4. componentUpdated：指令所在组件的 VNode及其子VNode全部更新后调用。

5. unbind：只调用一次，指令与元素解绑时调用。

八十九、自定义指令使用场景

1. 普通DOM元素进行底层操作的时候，可以使用自定义指令
2. 自定义指令是用来操作DOM的。尽管Vue推崇数据驱动视图的理念，但并非所有情况都适合数据驱动。自定义指令就是一种有效的补充和扩展，不仅可用于定义任何的DOM操作，并且是可复用的。

九十、自定义指令使用案例

初级应用：鼠标聚焦 、下拉菜单、相对时间转换、滚动动画；高级应用：自定义指令实现图片懒加载、自定义指令集成第三方插件

九十一、vue.js中标签如何绑定事件 - v-on/@语法糖

九十二、简述vue的key的原理

实际开发过程中具备两种途径：1. 当我们在使用v-for时需要给单元加上key 2. 用+new Date生成的时间戳作为key，手动强制触发重新渲染

附加题：v-for当中的key属性的作用是什么？

如果不用key，Vue会采用就地复地原则，设置了key会使 diff 操作可以更准确更快速。Vue 的 diff 过程可以概括为：oldCh 和 newCh 各有两个头尾的变量 oldStartIndex、oldEndIndex 和 newStartIndex、newEndIndex，它们会新节点和旧节点会进行两两对比，即一共有4种比较方式：newStartIndex 和oldStartIndex 、newEndIndex 和 oldEndIndex 、newStartIndex 和 oldEndIndex 、newEndIndex 和 oldStartIndex，如果以上 4 种比较都没匹配，如果设置了key，就会用 key 再进行比较，在比较的过程中，遍历会往中间靠，一旦 StartIdx > EndIdx 表明 oldCh 和 newCh 至少有一个已经遍历完了，就会结束比较。所以Vue中key的作用是：key是为Vue中vnode的唯一标记，通过这个key使我们的diff操作可以更准确、更快速

1. 更准确：因为带 key 就不是就地复用了，在 sameNode 函数 a.key === b.key 对比中可以避免就地复用的情况。所以会更加准确。 2. 更快速：利用 key 的唯一性生成 map 对象来获取对应节点，比遍历方式更快

附加题：用+new Date()生成的时间戳作为key，手动强制触发重新渲染

当拥有新值的rerender作为key时，拥有了新key的Comp出现了，那么旧key Comp会被移除，新key Comp触发渲染

附加题：就地复地原则

最小化element的移动，并且会尝试尽最大程度在同适当的地方对相同类型的element，做patch或者reuse。

九十三、vue事件绑定原理

每一个vue实例都是一个event bus，当子组件被创建父组件将事件传递给子组件，子组件初始化会有一个$on方法将事件注册到内部，在需要时使用$emit触发函数对于原生native事件，使用addEventListener绑定在真实的dom元素上面

九十四、slot是什么？有什么作用？原理是什么？

slot插槽-内容分发机制，组件内部的模板引擎使用slot元素作为承载分发内容的出口。插槽slot是子组件的一个模板标签元素，而这一个标签元素是否显示，以及怎么显示是由父组件决定的。slot又分三类，默认插槽，具名插槽和作用域插槽。

1. 默认插槽：当slot没有指定name属性值的时候一个默认显示插槽，一个组件内有且只有一个匿名插槽。
2. 具名插槽：带有name属性的slot，一个组件可以出现多个具名插槽。
3. 作用域插槽：在子组件渲染作用域插槽时，可以将子组件内部的数据传递给父组件，让父组件根据子组件的传递过来的数据决定如何渲染该插槽。

实现原理：当子组件vm实例化时，获取到父组件传入的slot标签的内容，存放在vm.$slot中，默认插槽为vm.$slot.default，具名插槽为vm.$slot.xxx，xxx 为插槽名name，当组件执行渲染函数时候，遇到slot标签，使用$slot中的内容进行替换，此时可以为插槽传递数据，若存在数据就是作用域插槽。

九十五、过滤器的作用，如何实现一个过滤器filter

filters过滤器是用来过滤数据的改变用户看到的输出，过滤器是一个函数，它会把表达式中的值始终当作函数的第一个参数。过滤器用在插值表达式 {{ }} 和 v-bind 表达式中，然后放在操作符“ | ”后面进行指示

实现原理：

1. 在编译阶段通过parseFilters将过滤器编译成函数调用（串联过滤器则是一个嵌套的函数调用，前一个过滤器执行的结果是后一个过滤器函数的参数）
2. 编译后通过调用resolveFilter函数找到对应过滤器并返回结果
3. 执行结果作为参数传递给toString函数，而toString执行后，其结果会保存在Vnode的text属性中，渲染到视图

九十六、常见的事件修饰符及其作用

.stop：等同于 JavaScript 中的 event.stopPropagation() ，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与事件冒泡方向相反事件捕获由外到内.self会触发自己范围内事件不包含子元素.once触发一次。九十七、使用过 Vue SSR 吗？说说 SSR？

SSR就是在客户端请求服务器的时候，服务器到数据库中获取到相关的数据，并且在服务器内部将Vue组件渲染成HTML，并且将数据HTML一并返回给客户端，而当客户端拿到服务器渲染的HTML和数据之后，由于数据已经有了，客户端不需要再一次请求数据，而只需要将数据同步到组件或者Vuex内部即可。除了数据意外，HTML也结构已经有了，客户端在渲染组件的时候，也只需要将HTML的DOM节点映射到Virtual DOM即可，不需要重新创建DOM节点，这个将数据和HTML同步的过程，又叫做客户端激活。

九十八、ssr的缺点

1. 更多的开发条件限制只支持beforCreate和created两个钩子函数且应用程序需要处于node运行环境
2. 需要更多的服务器负载

九十九、服务端渲染的优点

1. 更好的 SEO: SSR 是直接由服务端返回已经渲染好的页面所以搜索引擎爬取工具可以抓取渲染好页面
2. 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，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，

一百、了解过 style 上加 scoped 属性的原理吗？

scoped属性只属于当下模块局部有效使样式私有化（模块化）不对全局造成污染

scoped 的实现原理：Vue 中的 scoped 属性的效果主要通过PostCSS转译实现

PostCSS 给所有 DOM 添加了一个唯一不重复的动态属性（data-v-\*\*），、然后给 CSS 选择器额外添加一个对应的属性选择器来选择该组件中 DOM，对于第三方组件能通过穿透选择器 ::v-deep 方式来解决

一百零一、vue目录结构assets和static的区别

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

不相同点：

1. 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。
2. 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iconfont.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，不再需要处理，直接上传。

一百零二、vue实例挂载中发生了什么？

new Vue会调用\_init方法，定义 $set、$get 、$delete、$watch等方法，定义 $on、$off、$emit、$off等事件，定义 \_update、$forceUpdate、$destroy生命周期，调用$mount进行页面的挂载，挂载的时候主要是通过mountComponent方法，定义updateComponent更新函数，执行render生成虚拟DOM，\_update将虚拟DOM生成真实DOM结构，并且渲染到页面中

一百零二、组件和插件的区别？

组件 是用来构成业务模块目标是 App.vue；插件是用来增强你的技术栈的功能模块目标是 Vue 本身

一百零三、什么是虚拟DOM？如何实现一个虚拟DOM？说说你的思路

vue-router相关的面试题

一、vue-router 路由模式有几种

3种 1. hash模式使用URL hash 值来作为路由 2. history模式依赖 HTML5 History API 和服务器配置

3.abstract 支持所有js运行环境如果发现没有浏览器的 API，路由会自动强制进入这个模式

二、vue-router - hash路由模式实现原理吗？

基于 location.hash来实现的，location.hash值就是URL中#后面的内容，路由模式的实现主要是基于：

1. URL中hash 值是客户端一种状态，当向服务器端发出请求时hash部分不会被发送不能做seo优化
2. hash改变都会在浏览器的访问历史中增加一个记录，能通过浏览器的回退、前进按钮控制hash切换
3. 可以通过 a 标签并设置 href 属性，当用户点击这个标签后，URL 的 hash 值会发生改变；或者使用js来对 loaction.hash 进行赋值改变 URL hash 值
4. 使用 hashchange 事件来监听hash值变化从而对页面进行跳转，window.location.hash.substr(1);

三、history的实现原理

HTML5 提供了 History API 来实现 URL 的变化。通过history.pushState和 history.repalceState这两个 API 可以在不进行刷新的情况下，操作浏览器的历史纪录。pushState是新增一个历史记录repalceState是直接替换当前的历史记录，特性：1. 我们可以使用 popstate 事件来监听 url 的变化，从而对页面进行跳转（渲染）；2. pushState或 replaceState不会触发 popstate 事件，浏览器不会向后端发送请求，这时我们需要手动触发页面跳转（渲染）。popstate 事件的执行是在点击浏览器的前进后退按钮的时候，才会被触发，这两个方法提供了对历史记录进行修改的功能，当这两个方法执行修改时，只能改变当前地址栏的 url，但浏览器不会向后端发送请求

三、在vue项目中如何获取页面的hash变化

1. window.onhashchange监听hash事件window.addEventListener('hashChange', () => {})

2. 通过watch:{ $route:{ handler(newVal,oldVal){ }, deep:true } }

四、hash和history的区别

主要在浏览器的兼容性hash比history更出色，支持低版本浏览器和IE；通过history.pushState存在优势

1. pushState设置的新URL可以是与当前URL同源的任意URL；而 hash 只可修改#后面的部分，因此只能设置与当前 URL 同文档的 URL pushState设置的新 URL 可以与当前 URL 一模一样，这样也会把记录添加到栈中；而 hash 设置的新值必须与原来不一样才会触发动作将记录添加到栈中
2. pushState通过stateObject参数可以添加任意类型的数据到记录中而hash只可添加短字符串

四、什么是前端路由

前端路由是在保证只有一个HTML页面的情况下通过对每个视图展示形式匹配一个特殊的url来实现所谓的切换效果。不会重新向服务端发送请求也不会跳转页面无论是刷新前进还是后退都可以通过特殊url实现。

1. vue-router 路由钩子函数是什么（有哪几种导航守卫）、

钩子函数包括：全局守卫、路由守卫、组件守卫

1. 全局守卫/钩子函数：beforeEach、beforeResolve、afterEach
2. 路由独享守卫：beforeEnter写在路由配置里面只有访问路径才能触发
3. 组件级路由钩子：beforeRouteEnter、 beforeRouteUpdate、 beforeRouteLeave写在组件内部

六、vue-router 路由钩子函数执行完整导航流程

1. 导航被触发 2. 在失活的组件里面调用 beforeRouteLeave 守卫3. 调用全局的 beforeEach 守卫

4. 在重用的组件内部调用 beforeRouteUpdate 守卫（2.2+）5. 在路由配置中调用 beforeEnter

6. 解析异步组件路由7. 在被激活的组件内部调用 beforeRouteEnter

8. 调用全局的 beforeResolve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. 调用 beforeRouteEter 守卫中传给next的回调函数，创建好的组件实例会作为回调函数的参数传入

七、vue-router 动态路由是什么？有什么问题？

在vue-router的路由路径中支持使用动态路径参数的形式来使用户可以根据不同的业务场景拿到不同的组件入参，支持两种方式：params和query；两种方式的区别

1. query方式 name、path都可以使用params方式只能用name
2. query更加类似ajax当中的get穿参数、params更类似于post，query在浏览器地址栏当中显示参数，params不显示
3. query刷新不会丢失query里面的数据、params刷新会丢失params的数据

八、$route和$router的区别?

$router为VueRouter实例，想要导航到不同URL，则使用$router.push方法；$route为当前router跳转对象，里面可以获取name、path、query、params等

1. vue路由懒加载如何实现？

路由懒加载是将各个模块分开打包，在用户查看下相关模块内容时就直接引入相关模块的打包文件然后进行显示有效的解决了浏览器可能出现短暂时间空白页的情况。懒加载可以分割代码块，提高页面的初始加载效率解决白屏问题。方法1.使用es6的import 2. webpack提供的require.ensure(相同chunkName会被打包成一个js) 3. 使用异步组件resolve

import 是解构过程并且是编译时执行，require 是赋值过程并且是运行时才执行，也就是异步加载，require的性能相对于import稍低，因为require是在运行时才引入模块并且还赋值给某个变量

十、$router.push相当于点击路由链接(可以返回到当前路由界面）队列的方式（先进先出）

十一、$router.replace用新路由替换当前路由(不可以返回到当前路由界面) --> 栈的方式（先进后出）

、

、

Webpack相关面试题

一、谈谈你对webpack的理解？

答：webpack是js应用程序的静态模块打包工具，在处理应用程序的时候，内部会构建一个依赖图对应映射到项目所需要的各个模块

二、webpack的工作流程了解吗？

1. 初始化流程
2. 初始化参数：解析webpack配置参数，合并shell传入的和webpack.config.js文件配置的参数，形成最后的配置结果，各个配置项拷贝到 options 对象中
3. 初始化Compiler编译对象，为webpack声明周期不执行具体的任务，只是做调度工作
4. 加载所有配置插件plugin，监听webpack构建生命周期的事件节点做出相应的反应
5. 编译构建流程
6. 执行Compiler对象的run方法开始执行编译
7. 根据配置中的entry找出入口文件从入口文件出发开始解析文件即执行before-resolve和build module，构建AST语法树调用所有配置的Loader对模块进行编译（即执行normal-module-loader以及programe）再找出该模块依赖的模块再递归本步骤直到所有入口依赖文件都经过了处理
8. 根据入口和模块之间的依赖关系组装成一个个包含多个模块的Chunk再把每个 Chunk 转换成一个单独的文件加入到输出列表，执行seal方法并开始优化操作，在确定好输出内容后根据配置确定输出的路径和文件名把文件内容写到文件系统（即执行dist）
9. Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用Webpack 提供的 API 改变 Webpack 的运行结果
10. webpack编译构建流程即Complier -> run方法主要流程
11. compile开始编译：构建Compilation对象执行模块创建、依赖收集、分块、打包等等
12. 然后根据入口文件分析模块和依赖的模块来创建这些模块对象
13. 根据入口依赖使用工厂函数创建一下空的module对象
14. 执行buildModule，调用使用的loader完成模块的转换使用acorn解析转换后的内容输出AST，从配置的入口分析AST，当遇到require语句后将其加入到依赖的模块列表同时对新找出的这个模块再次分析
15. seal方法做了那些优化
16. 优化打包块，像提取公共文件的CommonsChunkPlugin就是在这个钩子函数触发时候进行
17. CommonsChunkPlugin提取公共代码的工作流程

CommonsChunkPlugin插件主要是为了生成webpack最后产出的asset文件