# Vue面试题

SSR也就是服务端渲染，也就是将Vue在客户端把标签渲染成HTML的工作放在服务端完成，然后再把html直接返回给客户端。

### 谈谈你对MVVM开发模式的理解

MVVM分为Model、View、ViewModel三者。  
Model 代表数据模型，数据和业务逻辑都在Model层中定义；  
View 代表UI视图，负责数据的展示；  
ViewModel 负责监听 Model 中数据的改变并且控制视图的更新，处理用户交互操作；  
Model 和 View 并无直接关联，而是通过 ViewModel 来进行联系的，Model 和 ViewModel 之间有着双向数据绑定的联系。因此当 Model 中的数据改变时会触发 View 层的刷新，View 中由于用户交互操作而改变的数据也会在 Model 中同步。  
这种模式实现了 Model 和 View 的数据自动同步，因此开发者只需要专注对数据的维护操作即可，而不需要自己操作 dom。

优点：

1.MVVM使并行开发更加容易，使前端开发和后端开发人员互不影响。

2.抽象化View层，减少了代码中的业务逻辑

3.ViewModel比事件驱动更容易测试

4.ViewModel的测试不用关心uI的自动化和交互

缺点：

1.对于简单的ui，使用MVVM有点太重

2.声明式的数据绑定不利于调试,因为命令式的代码可以和容易的设置断点，这种模式就不利于设置这样的断点

3.在不挑剔(non-trivial)的应用里数据绑定可以创建大量的簿记(book-keeping)。你也不想结束于绑定比被绑定的对象更复杂的情况。

4.在大的应用中，在获取大量的概要(generalization)前很难设计视图-模型层

### 1 说说你对 SPA 单页面的理解，它的优缺点分别是什么

SPA（ single-page application ）仅在 Web 页面初始化时加载相应的 HTML、JavaScript 和 CSS。一旦页面加载完成，SPA 不会因为用户的操作而进行页面的重新加载或跳转；取而代之的是利用路由机制实现 HTML 内容的变换，UI 与用户的交互，避免页面的重新加载。

**优点：**

* 1 用户体验好、快，内容的改变不需要重新加载整个页面，避免了不必要的跳转和重复渲染；
* 2 基于上面一点，SPA 相对对服务器压力小；
* 3 前后端职责分离，架构清晰，前端进行交互逻辑，后端负责数据处理；

**缺点：**

* 初次加载耗时多：为实现单页 Web 应用功能及显示效果，需要在加载页面的时候将 JavaScript、CSS 统一加载，部分页面按需加载；
* 前进后退路由管理：由于单页应用在一个页面中显示所有的内容，所以不能使用浏览器的前进后退功能，所有的页面切换需要自己建立堆栈管理；
* SEO 难度较大：由于所有的内容都在一个页面中动态替换显示，所以在 SEO 上其有着天然的弱势。

### 2、v-show 与 v-if 有什么区别？

* 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 css只在当前组件起作用

在style标签中写入scoped即可 例如：<style scoped></style>

### 4 route和router的区别

Route是路由信息对象，包括path,params,hash,query,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。

### 5 Vue的两个核心是什么

数据驱动、组件系统

### 6 Vue的几种常用指令

v-for 、 v-if 、v-bind、v-on、v-show、v-else

### 7 vue常用的修饰符

v-on常用的：.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

v-bind 指令常用修饰符：

.prop - 被用于绑定 DOM 属性 (property)。(差别在哪里？)

.camel - (2.1.0+) 将 kebab-case 特性名转换为 camelCase. (从 2.1.0 开始支持)

.sync (2.3.0+) 语法糖，会扩展成一个更新父组件绑定值的 v-on 侦听器。

v-model 指令常用修饰符：

.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

### 8 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。****「key的作用是尽可能的复用 DOM 元素。」****

新旧 children 中的节点只有顺序是不同的时候，最佳的操作应该是通过移动元素的位置来达到更新的目的。

需要在新旧 children 的节点中保存映射关系，以便能够在旧 children 的节点中找到可复用的节点。key也就是children中节点的唯一标识。

### 9 什么是vue的计算属性

在模板中放入太多的逻辑会让模板过重且难以维护，在需要对数据进行复杂处理，且可能多次使用的情况下，尽量采取计算属性的方式。好处：①使得数据处理结构清晰；②依赖于数据，数据更新，处理结果自动更新；③计算属性内部this指向vm实例；④在template调用时，直接写计算属性名即可；⑤常用的是getter方法，获取数据，也可以使用set方法改变数据；⑥相较于methods，不管依赖的数据变不变，methods都会重新计算，但是依赖数据不变的时候computed从缓存中获取，不会重新计算。

### 10 对比 jQuery ，Vue 有什么不同

jQuery 专注视图层，通过操作 DOM 去实现页面的一些逻辑渲染； Vue 专注于数据层，通过数据的双向绑定，最终表现在 DOM 层面，减少了 DOM 操作

Vue 使用了组件化思想，使得项目子集职责清晰，提高了开发效率，方便重复利用，便于协同开发

### 11 计算属性computed 和事件 methods 有什么区别

computed: 计算属性是基于它们的依赖进行缓存的,只有在它的相关依赖发生改变时才会重新求值

对于 method ，只要发生重新渲染，method 调用总会执行该函数

### 12 vue事件中如何使用event对象？

注意在事件中要使用 $ 符号

### 13 什么是$nextTick？

定义：在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。所以就衍生出了这个获取更新后的DOM的Vue方法。所以放在Vue.nextTick()回调函数中的执行的应该是会对DOM进行操作的 js代码；理解：nextTick()，是将回调函数延迟在下一次dom更新数据后调用，简单的理解是：当数据更新了，在dom中渲染后，自动执行该函数

你在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中。原因是什么呢，原因是在created()钩子函数执行的时候DOM 其实并未进行任何渲染，而此时进行DOM操作无异于徒劳，所以此处一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。与之对应的就是mounted钩子函数，因为该钩子函数执行时所有的DOM挂载已完成。使用时机：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函数，此时页面并未全部渲染。

在某个动作有可能改变DOM元素结构的时候，对DOM一系列的js操作都要放进Vue.nextTick()的回调函数中

### 14 Vue 组件中 data 为什么必须是函数？

因为一个组件是可以共享的，但他们的data是私有的，所以每个组件都要return一个新的data对象，返回一个唯一的对象，不要和其他组件共用一个对象。

### 15 v-for 与 v-if 的优先级

因为一个组件是可以共享的，但他们的data是私有的，所以每个组件都要return一个新的data对象，返回一个唯一的对象，不要和其他组件共用一个对象。

### 16 vue中子组件调用父组件的方法

1 直接在子组件中通过this.$parent.event来调用父组件的方法

2 在子组件里用$emit()向父组件触发一个事件，父组件监听这个事件就行了

3 Props，父组件把方法传入子组件中，在子组件里直接调用这个方法

### 17 vue中 keep-alive 组件的作用

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 ，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；

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；

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。

### 18 vue 的生命周期是：

 vue 实例从创建到销毁，也就是从**开始创建、初始化数据、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。

19 .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。

20 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

### 21 Vue生命周期函数

****beforeCreate****（创建前） 在数据观测和初始化事件还未开始  
****created****（创建后） 完成数据观测，属性和方法的运算，初始化事件，$el属性还没有显示出来  
****beforeMount****（载入前） 在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，把data里面的数据和模板生成html。注意此时还没有挂载html到页面上。  
****mounted****（载入后） 在el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译好的html内容替换el属性指向的DOM对象。完成模板中的html渲染到html页面中。此过程中进行ajax交互。  
****beforeUpdate****（更新前） 在数据更新之前调用，发生在虚拟DOM重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。  
****updated****（更新后） 在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。调用时，组件DOM已经更新，所以可以执行依赖于DOM的操作。然而在大多数情况下，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。  
****beforeDestroy****（销毁前） 在实例销毁之前调用。实例仍然完全可用。  
****destroyed****（销毁后） 在实例销毁之后调用。调用后，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。

（ 1、创建**vue实例**，Vue();

　　2、在创建Vue实例的时候，执行了**init()**，在init过程中首先调用了**beforeCreate钩子函数**；

　　3、同时**监听data数据**，初始化**vue内部事件**，进行**属性和方法的计算**；

　　4、以上都干完了，调用**Created钩子函数**；

　　5、**模板编译**，把data对象里面的数据和vue语法写的模板编译成HTML。编译过程分三种情况：1）实例内部有template属性，直接调用，然后调用render函数去渲染；2）没有该属性调用外部html；3）都没有抛出错误；

　　6、编译模板完成，调用**beforeMount钩子函数**；

　　7、**render函数**执行之后，将渲染出来的内容**挂载到DOM节点**上；

　　8、挂在结束，调用**Mounted钩子函数**；

　　9、数据发生变化，调用**beforeUpdate钩子函数**，经历**virtual Dom**；

　　10、更新完成，调用**Updated钩子函数**；

　　11、**beforeDestory**销毁所有观察者、组件及事件监听；

　　12、**Destoryed**实例销毁；

）

**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。

**vue获取数据在哪个周期函数：**

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.

**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。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；

#### Watch和updated的区别：

watch:

1.仅仅是数据发生改变的时候会侦听到

2.只是会检测到你写在watch里的那些属性,没写的就不会触发

updated:

1.执行到它的时候时候是数据发生变化且界面更新完毕

2.不能监听到路由数据(例如网址中的参数)

3.所有的数据发生变化都会调用(消耗性能)

4.每次触发的代码都是同一个

### 22 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。）

### 23 如何解决非工程化项目，网速慢时初始化页面闪动问题？

使用v-cloak指令，v-cloak不需要表达式，它会在Vue实例结束编译时从绑定的HTML元素上移除，经常和CSS的display:none配合使用

### 24 如何在组件中使用全局常量

第一步，在 src 下新建 const 文件夹下 新建 const.js

第二步，如何在 const.js 文件下，设置常量

第三步，在 main.js 下全局引入

第四步，即可在 .vue 组件中使用

### 25 算属性的缓存和方法调用的有什么区别？

1 计算属性必须返回结果

2 计算属性是基于它的依赖缓存的。一个计算属性所依赖的数据发生变化时，它才会重新取值。

3 使用计算属性还是methods取决于是否需要缓存，当遍历大数组和做大量计算时，应当使用计算属性，除非你不希望得到缓存。

4 计算属性是根据依赖自动执行的，methods需要事件调用。

### 26 Vue中双向数据绑定是如何实现的？

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。

### 27 vue和jQuery的区别

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。

### 28 SPA首屏加载慢如何解决

安装动态懒加载所需插件；使用CDN资源。

将公用的JS库通过script标签外部引入，减小app.bundel的大小，让浏览器并行下载资源文件，提高下载速度；

在配置 路由时，页面和组件使用懒加载的方式引入，进一步缩小 app.bundel 的体积，在调用某个组件时再加载对应的js文件；

加一个首屏 loading 图，提升用户体验；

### 29 Vue-router跳转和location.href有什么区别

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。

### 30 mvvm 框架是什么？

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。

### 31 vue-router 有哪几种导航钩子?

三种，  
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

### 32 vue-router的两种模式

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）。

### 33 vue-router实现路由懒加载

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)。  
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。

### 34 vuex是什么？怎么使用？哪种功能场景使用它？

vue框架中状态管理。在main.js引入store，注入。  
新建了一个目录store.js，….. 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

### 35 vuex有哪几种属性？

有五种，分别是 State、 Getter、Mutation：commit，mapmutation 、Action:dispatch,mapaction、 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

### 36 Vue中如何在组件内部实现一个双向数据绑定？

父组件通过 props 传值给子组件，子组件通过 $emit 来通知父组件修改相应的props值，在父组件中做了两件事，一是给子组件传入props，二是监听input事件并同步自己的value属性

### 37 vue中的watch和computed有何区别

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；

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；

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；

Watch没有缓存性，更多的是观察的作用，可以监听某些数据执行回调。当我们需要深度监听对象中的属性时，可以打开deep：true选项，这样便会对对象中的每一项进行监听。这样会带来性能问题，优化的话可以使用字符串形式监听，如果没有写到组件中，不要忘记使用unWatch手动注销哦。

38Vue组件间通信方式有哪些

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。

（1）props / $emit  适用 父子组件通信

这种方法是 Vue 组件的基础，相信大部分同学耳闻能详，所以此处就不举例展开介绍。

1. ref 与 $parent / $children 适用 父子组件通信

ref：如果在普通的 DOM 元素上使用，引用指向的就是 DOM 元素；如果用在子组件上，引用就指向组件实例

$parent / $children：访问父 / 子实例

（3）EventBus （$emit / $on）  适用于 父子、隔代、兄弟组件通信

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，从而实现任何组件间的通信，包括父子、隔代、兄弟组件。

（4）$attrs/$listeners 适用于 隔代组件通信

$attrs：包含了父作用域中不被 prop 所识别 (且获取) 的特性绑定 ( class 和 style 除外 )。当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 ( class 和 style 除外 )，并且可以通过 v-bind="$attrs" 传入内部组件。通常配合 inheritAttrs 选项一起使用。

$listeners：包含了父作用域中的 (不含 .native 修饰器的)  v-on 事件监听器。它可以通过 v-on="$listeners" 传入内部组件

（5）provide / inject 适用于 隔代组件通信

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。provide / inject API 主要解决了跨级组件间的通信问题，不过它的使用场景，主要是子组件获取上级组件的状态，跨级组件间建立了一种主动提供与依赖注入的关系。

（6）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。

### 39 说一下hash和history的原理吗？

1）hash 模式的实现原理

早期的前端路由的实现就是基于 location.hash 来实现的。其实现原理很简单，location.hash 的值就是 URL 中 # 后面的内容。比如下面这个网站，它的 location.hash 的值为 '#search'：

https://www.word.com#search 复制代码

hash  路由模式的实现主要是基于下面几个特性：

URL 中 hash 值只是客户端的一种状态，也就是说当向服务器端发出请求时，hash 部分不会被发送；

hash 值的改变，都会在浏览器的访问历史中增加一个记录。因此我们能通过浏览器的回退、前进按钮控制hash 的切换；

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  JavaScript 来对 loaction.hash 进行赋值，改变 URL 的 hash 值；

我们可以使用 hashchange 事件来监听 hash 值的变化，从而对页面进行跳转（渲染）。

（2）history 模式的实现原理

HTML5 提供了 History API 来实现 URL 的变化。其中做最主要的 API 有以下两个：history.pushState() 和 history.repalceState()。这两个 API 可以在不进行刷新的情况下，操作浏览器的历史记录。唯一不同的是，前者是新增一个历史记录，后者是直接替换当前的历史记录，如下所示：

window.history.pushState(null, null, path); window.history.replaceState(null, null, path); 复制代码

history 路由模式的实现主要基于存在下面几个特性：

pushState 和 repalceState 两个 API 来操作实现 URL 的变化 ；

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）；

history.pushState() 或 history.replaceState() 不会触发 popstate 事件，这时我们需要手动触发页面跳转（渲染）。

### 40 Vue如何实现数据的双向绑定？

Vue 数据双向绑定主要是指：数据变化更新视图，视图变化更新数据

其中，View 变化更新 Data ，可以通过事件监听的方式来实现，所以 Vue 的数据双向绑定的工作主要是如何根据 Data 变化更新 View。

Vue 主要通过以下 4 个步骤来实现数据双向绑定的：

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。

实现一个解析器 Compile：解析 Vue 模板指令，将模板中的变量都替换成数据，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，调用更新函数进行数据更新。

实现一个订阅者 Watcher：Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁 ，主要的任务是订阅 Observer 中的属性值变化的消息，当收到属性值变化的消息时，触发解析器 Compile 中对应的更新函数。

实现一个订阅器 Dep：订阅器采用 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。

#### Object.defineProperty()有哪些属性，有什么局限，如何改进

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,数据描述符和存取描述符:set,get

value：属性值

writable：是否可写

enumerable：是否可枚举

Configurable:表示能否通过delete删除属性从而重新定义属性

局限：1 无法检测到对象属性的新增或删除  
　　由于js的动态性，可以为对象追加新的属性或者删除其中某个属性，这点对经过Object.defineProperty方法建立的响应式对象来说，只能追踪对象已有数据是否被修改，无法追踪新增属性和删除属性，这就需要另外处理。

2 不能监听数组的变化

　　vue在实现数组的响应式时，它使用了一些hack，把无法监听数组的情况通过重写数组的部分方法来实现响应式，这也只限制在数组的push/pop/shift/unshift/splice/sort/reverse七个方法，其他数组方法及数组的使用则无法检测到，例如如下两种使用方式：

　　vm.items[index] = newValue

　　vm.items.length--

### 41 Vue如何实现对象和数组的监听？

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.defineProperty() 也能对对象和数组（部分方法的操作）进行监听。

### 42 Proxy和Object.defineProperty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。

### 43 你对Vue项目做过哪些优化？

（1）代码层面的优化

v-if 和 v-show 区分使用场景

computed 和 watch  区分使用场景

v-for 遍历必须为 item 添加 key，且避免同时使用 v-if

长列表性能优化

事件的销毁

图片资源懒加载

路由懒加载

第三方插件的按需引入

优化无限列表性能

服务端渲染 SSR or 预渲染

（2）Webpack 层面的优化

Webpack 对图片进行压缩

减少 ES6 转为 ES5 的冗余代码

提取公共代码

模板预编译

提取组件的 CSS

优化 SourceMap

构建结果输出分析

Vue 项目的编译优化

（3）基础的 Web 技术的优化

开启 gzip 压缩

浏览器缓存

CDN 的使用

使用 Chrome Performance 查找性能瓶颈

### 44 对Vue3.0的特效有哪些了解？

（1）监测机制的改变

3.0 将带来基于代理 Proxy 的 observer 实现，提供全语言覆盖的反应性跟踪。这消除了 Vue 2 当中基于 Object.defineProperty 的实现所存在的很多限制：

只能监测属性，不能监测对象

检测属性的添加和删除；

检测数组索引和长度的变更；

支持 Map、Set、WeakMap 和 WeakSet。

新的 observer 还提供了以下特性：

用于创建 observable 的公开 API。这为中小规模场景提供了简单轻量级的跨组件状态管理解决方案。

默认采用惰性观察。在 2.x 中，不管反应式数据有多大，都会在启动时被观察到。如果你的数据集很大，这可能会在应用启动时带来明显的开销。在 3.x 中，只观察用于渲染应用程序最初可见部分的数据。

更精确的变更通知。在 2.x 中，通过 Vue.set 强制添加新属性将导致依赖于该对象的 watcher 收到变更通知。在 3.x 中，只有依赖于特定属性的 watcher 才会收到通知。

不可变的 observable：我们可以创建值的“不可变”版本（即使是嵌套属性），除非系统在内部暂时将其“解禁”。这个机制可用于冻结 prop 传递或 Vuex 状态树以外的变化。

更好的调试功能：我们可以使用新的 renderTracked 和 renderTriggered 钩子精确地跟踪组件在什么时候以及为什么重新渲染。

（2）模板

模板方面没有大的变更，只改了作用域插槽，2.x 的机制导致作用域插槽变了，父组件会重新渲染，而 3.0 把作用域插槽改成了函数的方式，这样只会影响子组件的重新渲染，提升了渲染的性能。

同时，对于 render 函数的方面，vue3.0 也会进行一系列更改来方便习惯直接使用 api 来生成 vdom 。

（3）对象式的组件声明方式

vue2.x 中的组件是通过声明的方式传入一系列 option，和 TypeScript 的结合需要通过一些装饰器的方式来做，虽然能实现功能，但是比较麻烦。3.0 修改了组件的声明方式，改成了类式的写法，这样使得和 TypeScript 的结合变得很容易。

此外，vue 的源码也改用了 TypeScript 来写。其实当代码的功能复杂之后，必须有一个静态类型系统来做一些辅助管理。现在 vue3.0 也全面改用 TypeScript 来重写了，更是使得对外暴露的 api 更容易结合 TypeScript。静态类型系统对于复杂代码的维护确实很有必要。

（4）其它方面的更改

vue3.0 的改变是全面的，上面只涉及到主要的 3 个方面，还有一些其他的更改：

支持自定义渲染器，从而使得 weex 可以通过自定义渲染器的方式来扩展，而不是直接 fork 源码来改的方式。

支持 Fragment（多个根节点）和 Protal（在 dom 其他部分渲染组建内容）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。

### 45 vue2.x中如何监测数组变化

使用了函数劫持的方式，重写了数组的方法，Vue将data中的数组进行了原型链重写，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。如果数组中包含着引用类型，会对数组中的引用类型再次递归遍历进行监控。这样就实现了监测数组变化。

### 46 什么是虚拟dom

1. 用 JavaScript 对象结构表示 DOM 树的结构；然后用这个树构建一个真正的 DOM 树，插到文档当中
2. 当状态变更的时候，重新构造一棵新的对象树。然后用新的树和旧的树进行比较，记录两棵树差异
3. 把2所记录的差异应用到步骤1所构建的真正的DOM树上，视图就更新了

### 47 虚拟dom的缺点

****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。

48 diff算法

****比较只会在同层级进行, 不会跨层级比较。****  
比较后会出现四种情况：  
1、此节点是否被移除 -> 添加新的节点  
2、属性是否被改变 -> 旧属性改为新属性  
3、文本内容被改变-> 旧内容改为新内容  
4、节点要被整个替换 -> 结构完全不相同 移除整个替换

### 49 发布订阅模式和观察者模式的区别

观察者模式定义了对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都将得到通知，并自动更新。观察者模式属于行为型模式，行为型模式关注的是对象之间的通讯，观察者模式就是观察者和被观察者之间的通讯。

发布订阅模式中，称为发布者的消息发送者不会将消息直接发送给订阅者，这意味着发布者和订阅者不知道彼此的存在。在发布者和订阅者之间存在第三个组件，称为消息代理或调度中心或中间件，它维持着发布者和订阅者之间的联系，过滤所有发布者传入的消息并相应地分发它们给订阅者。