1、介绍一下MVC MVVM

1. 什么是MVVM?

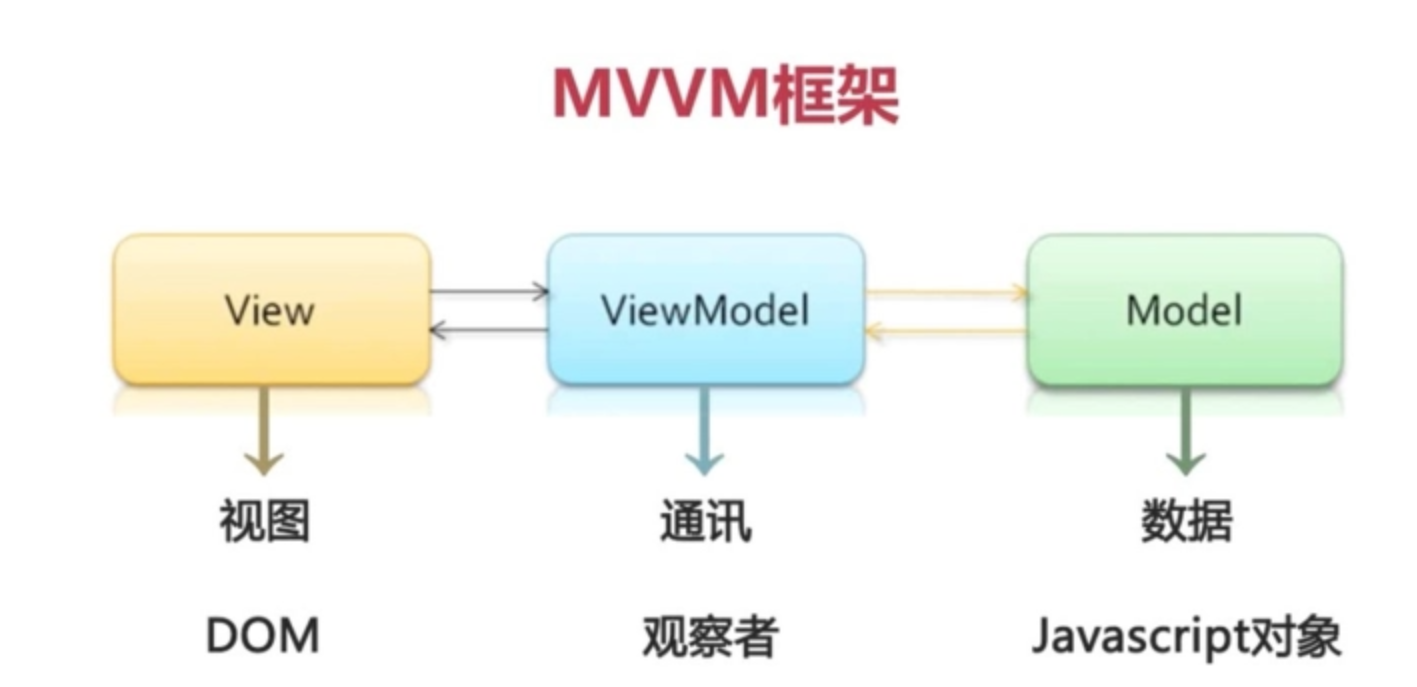
在MVVM框架下 视图和模型是不能直接通信 的，只能通过ViewModel进行交互，它能够监

听到数据的变化，然后通知视图进行自动更新，而当用户操作视图时，VM也能监听到视图

的变化，然后通知数据做相应改动，这实际上就实现了数据的 双向绑定 。并且V和VM可以

进行通信。





Model（模型）

模型是指代表真实状态内容的领域模型（面向对象），或指代表内容的数据访问层（以数据为中

心）。

View（视图）

就像在MVC和MVP模式中一样，视图是用户在屏幕上看到的结构、布局和外观（UI）。

ViewModel（视图模型）

视图模型是暴露公共属性和命令的视图的抽象。MVVM没有MVC模式的控制器，也没有MVP模式的

presenter，有的是一个绑定器。在视图模型中，绑定器在视图和数据绑定器之间进行通信。

优点:

低耦合 :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

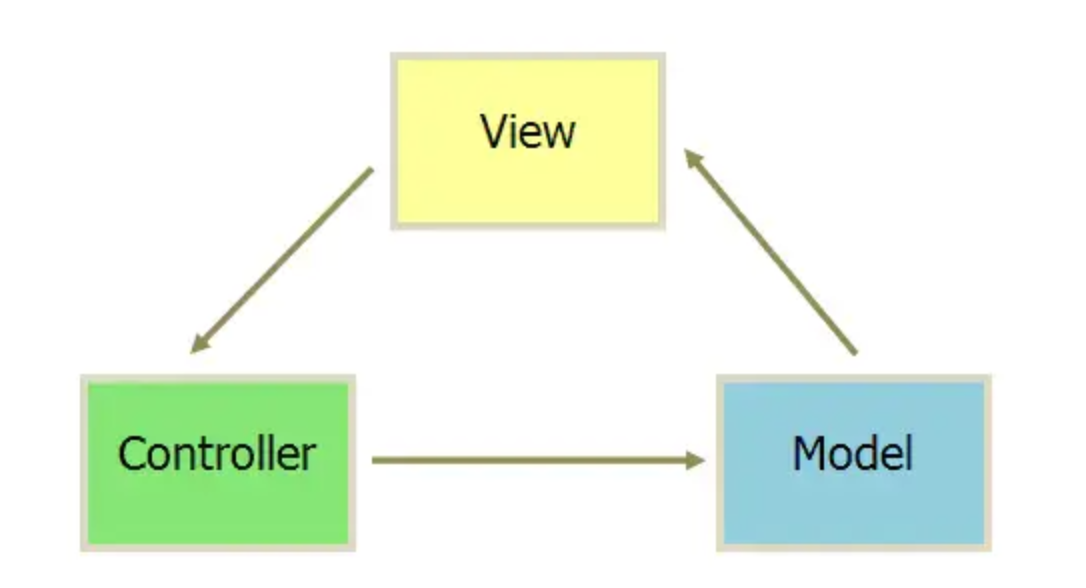
的时候Model可以不变,当Model变化的时候View也可以不变。

可重用性 : 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。

独立开发 : 开发人员可以专注于业务逻辑和数据的开发,设计人员可以专注于页面的设计。

2. 什么是MVC?

MVC是应用最广泛的软件架构之一,一般MVC分为:Model(模型),View(视图),Controller(控制器)。 这主要是基于分层的目的,让彼此的职责分开.View一般用过Controller来和Model进行联系。Controller是Model和View的协调者,View和Model不直接联系。基本都是单向联系。M和V指的意思和MVVM中的M和V意思一样。C即Controller指的是页面业务逻辑。MVC是单向通信。也就是View跟Model，必须通过Controller来承上启下。



Model（模型）表示应用程序核心（如数据库）。

View（视图）显示效果（HTML页面）。

Controller（控制器）处理输入（业务逻辑）。

MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。 　　通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。 　　通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。 　　通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

优点:

低耦合

重用性高

生命周期成本低

部署快

可维护性高

有利软件工程化管理

3. MVC与MVVM的区别:

MVC和MVVM的区别并不是VM完全取代了C，ViewModel存在目的在于抽离Controller中展示的业务逻辑，而不是替代Controller，其它视图操作业务等还是应该放在Controller中实现。也就是说MVVM实现的是业务逻辑组件的重用。

MVC中Controller演变成MVVM中的ViewModel

MVVM通过数据来显示视图层而不是节点操作

MVVM主要解决了MVC中大量的dom操作使页面渲染性能降低,加载速度变慢,影响用户体验

2、为什么data是一个函数

组件的data写成一个函数，数据以函数返回值形式定义，这样每复用一次组件，就会返回一分新的data，类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份data，就会造成一个变了全都会变的结果。

3、Vue组件通讯有哪些方式？

1、props 和 $emit。父组件向子组件传递数据是通过props传递的，子组件传递给父组件是通过$emit触发事件来做到的。

2、$parent 和 $children 获取单签组件的父组件和当前组件的子组件。

3、$attrs 和 $listeners A -> B -> C。Vue2.4开始提供了$attrs和$listeners来解决这个问题。

4、父组件中通过 provide 来提供变量，然后在子组件中通过 inject 来注入变量。（官方不推荐在实际业务中适用，但是写组件库时很常用。）

5、$refs 获取组件实例。

6、envetBus 兄弟组件数据传递，这种情况下可以使用事件总线的方式。

7、vuex 状态管理

4、Vue的生命周期方法有哪些？一般在哪一步发送请求？

beforeCreate 在实例初始化之后，数据观测（data observe）和 event/watcher 事件配置之前被调用。在当前阶段 data、methods、computed 以及 watch 上的数据和方法都不能被访问。

created 实例已经创建完成之后被调用。在这一步，实例已经完成以下的配置：数据观测（data observe ），属性和方法的运算，watch/event 事件回调。这里没有 $el，如果非要想与 DOM 进行交互，可以通过vm.$nextTick 来访问 DOM。

beforeMount 在挂载开始之前被调用：相关的 render 函数首次被调用。

mounted 在挂载完成后发生，在当前阶段，真实的 Dom 挂载完毕，数据完成双向绑定，可以访问到 Dom节点。

beforeUpdate 数据更新时调用，发生在虚拟 DOM 重新渲染和打补丁 （patch）之前。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。（数据修改页面未修改）

updated 发生在更新完成之后，当前阶段组件 Dom 已经完成更新。要注意的是避免在此期间更新数据，因为这个可能导致无限循环的更新，该钩子在服务器渲染期间不被调用。

beforeDestroy 实例销毁之前调用。在这一步，实例仍然完全可用。我们可以在这时进行 善后收尾工作，比如清除定时器。

destroyed Vue实例销毁后调用。调用后，Vue实例指示的东西都会解绑定，所有的事件监听器会被移除，左右的子实例也会被销毁，该钩子在服务器端渲染不被调用。

activated 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

异步请求在哪一步发起？

可以在钩子函数 created、beforeMount、mounted 中进行异步请求，因为在这三个钩子函数中，data已经创建，可以将服务器端返回的数据进行赋值。

如果异步请求不需要依赖 DOM 推荐加载 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；

如果依赖DOM元素：需要再mounted里面进行请求

5、v-if 和 v-show 的区别

v-if 在编译过程中会被转化成三元表达式，条件不满足时不渲染此节点。元素销毁和重建控制显示隐藏

v-show 会被编译成指令，条件不满足时控制样式将此节点隐藏（display:none） css样式控制

使用场景

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 适用于需要非常频繁切换条件的场景。

扩展补充：display:none 、 visibility:hidden 和 opacity:0 之间的区别？

三者公共点都是 隐藏。

不同点：

是否占据空间。

display:none，隐藏之后不占位置；visibility:hidden、opacity:0，隐藏后任然占据位置。

子元素是否继承。

display:none --- 不会被子元素继承，父元素都不存在了，子元素也不会显示出来。

visibility:hidden --- 会被子元素继承，通过设置子元素 visibility:visible 来显示子元素。

opacity:0 --- 会被子元素继承，但是不能设置子元素 opacity:0 来先重新显示。

事件绑定。

display:none 的元素都已经不存在了，因此无法触发他绑定的事件。

visibility:hidden 不会触发他上面绑定的事件。

opacity:0 元素上面绑定的事件时可以触发的。

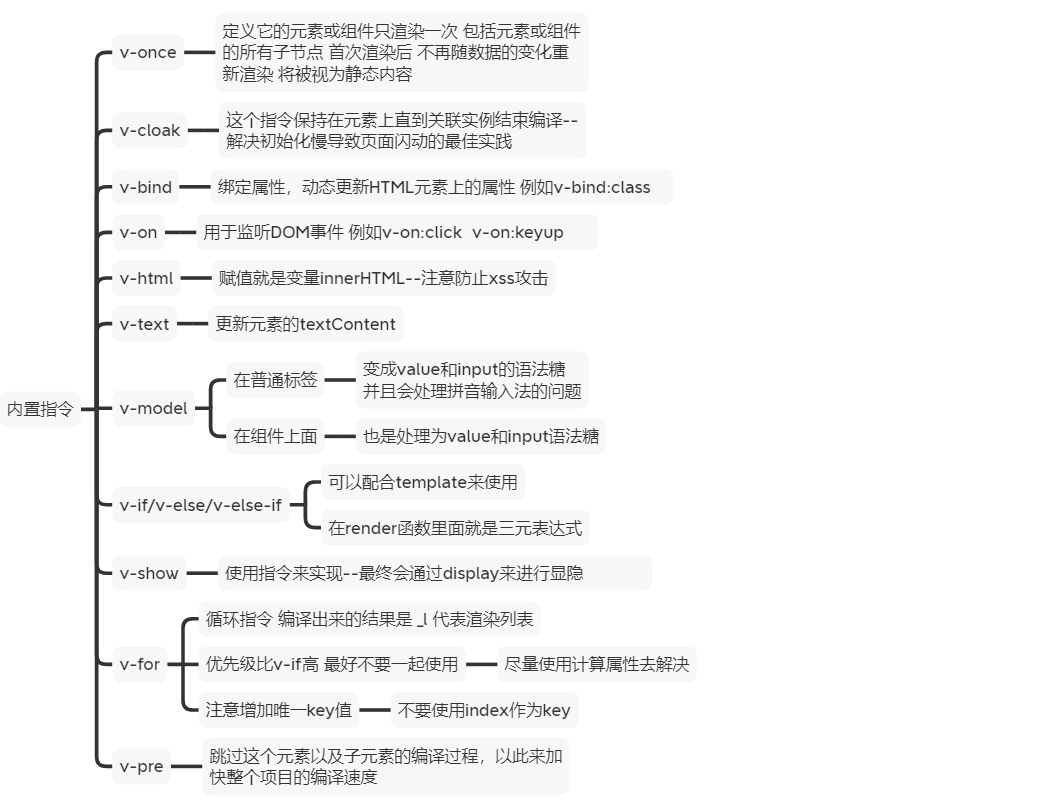
过度动画。

transition对于display是无效的。

transition对于visibility是无效的。

transition对于opacity是有效的。

6、说说 vue 内置指令



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。

7、怎样理解 Vue 的单项数据流

数据总是从父组件传到子组件，子组件没有权利修改父组件传过来的数据，只能请求父组件对原始数据进行修改。这样会防止从子组件意外改变父组件的状态，从而导致你的应用的数据流向难以理解。

注意：在子组件直接用 v-model 绑定父组件传过来的 props 这样是不规范的写法，开发环境会报警告。

如果实在要改变父组件的 props 值可以再data里面定义一个变量，并用 prop 的值初始化它，之后用$emit 通知父组件去修改。

多种方法实现：在子组件直接用 v-model 绑定父组件传过来的 props

方法1：利用get set方法

方法2：监听器

方法3：对象写法（推荐）

8、computed 和 watch 的区别和运用的场景

computed 是计算属性，依赖其它属性计算值，并且 computed 的值有缓存，只有当计算值变化才会返回内容，他可以设置getter和setter。

watch 监听到值的变化就会执行回调，在回调中可以进行一系列的操作。

计算属性一般用在模板渲染中，某个值是依赖其它响应对象甚至是计算属性而来；

而侦听属性适用于观测某个值的变化去完成一段复杂的业务逻辑。

9、v-if 和 v-for 为什么不建议一起使用

v-for和v-if不要在同一标签中使用，因为解析时先解析v-for在解析v-if。如果遇到需要同时使用时可以考虑写成计算属性的方式。

永远不要把 v-if 和 v-for 同时用在同一个元素上，带来性能方面的浪费（每次渲染都会先循环再进行条件判断）

如果避免出现这种情况，则在外层嵌套template（页面渲染不生成dom节点），在这一层进行v-if判断，然后在内部进行v-for循环

<template v-if="isShow">

<p v-for="item in items">

</template>

如果条件出现在循环内部，可通过计算属性computed提前过滤掉那些不需要显示的项

computed: {

items: function() {

return this.list.filter(function (item) {

return item.isShow

})

}

}

10. Vue 2.0 响应式数据的原理（常问）

整体思路是 数据劫持 + 观察者模式

Vue 在初始化数据时 ，会使用 Object.defineProperty 重新定义 data 中的所有属性 ，当页面 使用对 应 属性时，首先会进行 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 更新操作( 发布订阅 )

Vue2.x 采用 数据劫持结合发布订阅模式 （PubSub 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。

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 整合了 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，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（例如 input 操作）->数据

model 变更的双向绑定效果。

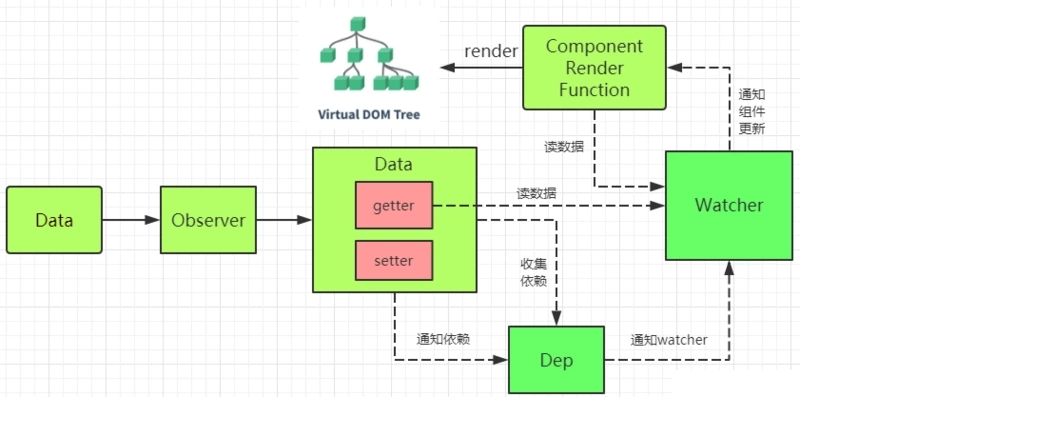
Vue3.x 放弃了 Object.defineProperty ，使用 ES6 原生的 Proxy，来解决以前使用

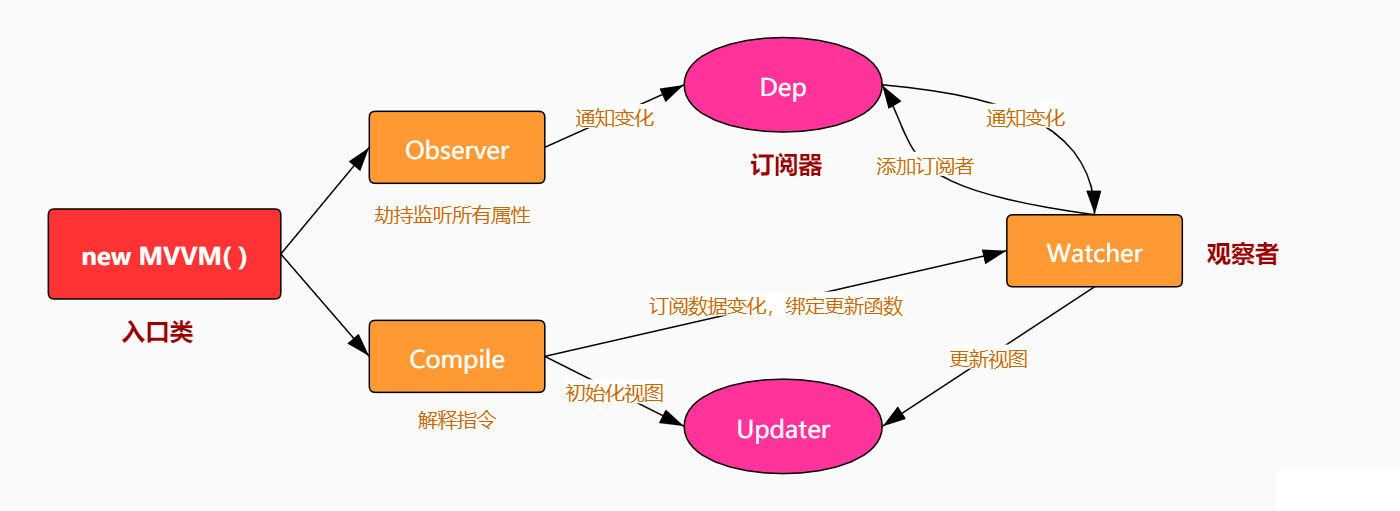
Object.defineProperty 所存在的一些问题。

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。





11、Vue 如何检测数组变化

Vue2.x 中实现检测数组变化的方法，是将数组的常用方法进行了 重写 。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 。

流程:

1. 初始化传入 data 数据执行 initData

2. 将数据进行观测 new Observer

3. 将数组原型方法指向重写的原型

4. 深度观察数组中的引用类型

有两种情况无法检测到数组的变化 。

1. 当利用索引直接设置一个数组项时，例如 vm.items[indexOfItem] = newValue

2. 当修改数组的长度时，例如 vm.items.length = newLength

不过这两种场景都有对应的解决方案。

利用索引设置数组项的替代方案

//使用该方法进行更新视图

// vm.$set，Vue.set的一个别名

vm.$set(vm.items, indexOfItem, newValue)

12、Vue的父子组件生命周期钩子函数执行顺序

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

总结： 父组件先开始 子组件先结束

13、v-model 双向绑定的原理是什么？

v-model 本质 就是 : value + input 方法的语法糖 。可以通过 model 属性的 prop 和 event 属性来进行自定义。原生的 v-model，会根据标签的不同生成不同的事件和属性。

例如：

1. text 和 textarea 元素使用 value 属性和 input 事件

2. checkbox 和 radio 使用 checked 属性和 change 事件

3. select 字段将 value 作为 prop 并将 change 作为事件

以输入框为例，当用户在输入框输入内容时，会触发 input 事件，从而更新 value。而 value 的改变同样会更新视图，这就是 vue 中的双向绑定。双向绑定的原理，其实现思路 如下：

首先要对 数据进行劫持监听 ，所以我们需要设置 一个监听器 Observe r，用来 监听 所有属 性。如果属性发上变化了，就需要告 诉订阅者 Watcher 看是否需要更新 。

因为订阅者是有很多个，所以我们需要有一个 消息订阅器 Dep 来专门收集这些订阅者 ，然 后在监听器 Observer 和订阅者 Watcher 之间 进行统一管理的。

接着，我们还需要有一个 指令解析器 Compile ，对每个节点元素进 行扫描和解析 ，将相关指令对应初始化成一个订阅者 Watcher，并替换模板数据或者绑定相应的函数，此时当订阅者 Watcher 接收到相应属性的变化，就会执行对应的更新函数，从而更新视图。

因此接下去我们执行以 下 3 个步骤，实现数据的双向绑定 ：

1. 实现一个 监听器 Observer ，用来 劫持并监听所有属 性，如果有变动的，就通知订阅 者。

2. 实现一个 订阅者 Watcher ，可以 收到属性的变化通知并执 行相应的函数，从而更新视 图。

3. 实现一个 解析器 Compile ，可以 扫描和解析每个 节点的相关指令，并根据 初始化模板数 据以及初始化相应的订阅器。

14. Vue3.x 响应式数据可以从以下方面展开回答：

Vue3.x 响应式数据原理是什么？

Proxy 只会代理对象的第一层，那么 Vue3 又是怎样处理这个问题的呢 ？

监 测数组的时候可能触发多次 get/set ，那么如何 防止触发多 次呢？

参考答案： Vue3.x 响应式数据原理是什么？ 在 Vue 2 中，响应式原理就是使用的 Object.defineProperty 来实现的。但是在 Vue 3.0 中采用了 Proxy，抛弃了 Object.defineProperty 方法。 究其原因，主要是以下几点：

Object.defineProperty 无法监控 到数组 下标的变化 ，导致通过数组下标添加元素，不

能实时响应 Object.defineProperty 只能 劫持对象 的属性，从而需要对每个对象，每

个属性进行遍历，如果，属性值是对象，还需要深度遍历。 Proxy 可以劫持整个对象，

并返回一个新的对象 。

P roxy 不仅可以代理对象，还可以代理数组。还可以代理动态

增加的属性。 Proxy 有多达 13 种拦截方法 Proxy作为新标准将受到浏览器厂商重点

持续的性能优化 Proxy 只会代理对象的第一层，那么 Vue3 又是怎样处理这个问题的

呢？ 判断当前 Reflect.get 的返回值是否为 Object，如果是则再通过 reactive 方

法做代理， 这样就实现了深度观测。 监测数组的时候可能触发多次 get/set，那么

如何防止触发多次呢？ 我们可以判断 key 是否为当前被代理对象 target 自身属性，

也可以判断旧值与新值是否相等，只有满足以上两个条件之一时，才有可能执行

trigger。

15. vue2.x 和 vuex3.x 渲染器的 diff 算法分别说一下？

参考答案：

简单来说，diff 算法有以下过程

同级比较，再比较子节点

先判断一方有子节点一方没有子节点的情况(如果新的 children 没有子节点，将旧的子

节点移除)

比较都有子节点的情况(核心 diff)

递归比较子节点

正常 Diff 两个树的时间复杂度是 O(n^3)，但实际情况下我们很少会进行跨层级的移动 DOM，所以 Vue 将 Diff 进行了优化，从O(n^3) -> O(n)，只有当新旧 children 都为多个子节点时才需要用核心的 Diff 算法进行同层级比较。

Vue2 的核心 Diff 算法采用了双端比较的算法 ，同时从新旧 children 的两端开始进行比较， 借助 key 值找到可复用的节点，再进行相关操作。相比 React 的 Diff 算法，同样情况 下可以减少移动节点次数，减少不必要的性能损耗，更加的优雅。

Vue3.x 借鉴了 ivi 算法和 inferno 算法

在 创建 VNode 时 就确定其类型，以及在 mount/patch 的过程中 采用位运算来判断 一个 VNode 类型，在这个基础之上再配合核心的 Diff 算法，使得性能上较 Vue2.x 有了提 升 。该算法中还运用了动态规划的思想求解最长递归子序列。

16. hash 模式和 history 模式的实现原理

hash 值的变化 ， 不会 导致浏览器 向服务器发出请 求，浏览器不发出请求，就不会刷新页面；通过监听 hashchange 事件可以知道 hash 发生了哪些变化，然后根据 hash 变化来实现更新页面部分内容的操作。

history 模式 的实现，主要是 HTML5 标准发布的 两个 API ， pushState 和 replaceState ，这两API 可以在改变 URL，但是不会发送请求。这样就可以监听 url 变化来实现更新页面部分内容的操作。

两种模式的区别：

首先是在 URL 的展示上，hash 模式有“#”，history 模式没有

刷新页面时，hash 模式可以正常加载到 hash 值对应的页面，而 history 没有处理的话，会返回404，一般需要后端将所有页面都配置重定向到首页路由

在兼容性上，hash 可以支持低版本浏览器和 IE

hash 模式

1 、 location.has 的值实际就是 URL 中 # 后面的东西。它的特点在于： hash 虽然出现 URL 中，但不会被包含在 HTTP 请求中，对后端完全没有影响，因此改变 hash 不会重新 加载页面。

2 、可以为 hash 的改变添加监听事件

window.addEventListener("hashchange",funcRef,false)

每一次改变 hash (window.location.hash) ，都会在浏览器的访问历史中增加一个记录， 利用hash 的以上特点，就可以实现前端路由 “ 更新视图但不重新请求页面 ” 的功能了

特点：兼容性好但是不美观

history 模式

利用 HTML5 History Interface 中新增的 pushState() 和 replaceState() 方法。 这两个方法应用于浏览器的历史记录站，在当前已有的 back 、 forward 、 go 的基础上， 他们提供了对历史记录进行修改的功能。这两个方法有个共同点：当调用他们修改浏览器 历史记录栈后，虽然当前 URL 改变了，但浏览器不会刷新页面，这就为单页面应用前端路 由“ 更新视图但不重新请求页面 ” 提供了基础

特点：虽然美观，但是刷新会出现 404 需要后端进行配置。

17、vue-router 路由钩子函数是什么？执行顺序是什么？

路由钩子的执行流程，钩子函数种类有：全局守卫、路由守卫、组件守卫。

完整的导航解析流程：

1 、导航被触发。

2 、在失活的组件里调用 beforeRouterLeave 守卫。

3 、调用全局的 beforeEach 守卫。

4 、在重用的组件调用 beforeRouterUpdate 守卫（ 2.2+ ）。

5 、在路由配置里面 beforeEnter 。

6 、解析异步路由组件。

7 、在被激活的组件里调用 beforeRouterEnter 。

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 beforeRouterEnter 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。

18、vue-router 动态路由是什么？有什么问题。

我们经常需要把某种模式匹配到的所有路由，全都映射到同个组件。例如，我们有一个

User 组件，对于所有 ID 各不相同的用户，都要使用这个组件来渲染。那么，我们可以在 vue-router 的路由路径中使用 “ 动态路径参数 ” （ dynamic segment ）来达到这个效果：

1 const User = {

2 template: "

3 User", };

4 const router = new VueRouter({

5 routes: [

6 // 动态路径参数 以冒号开头

7 { path: "/user/:id", component: User },

8 ],

9 });

问题 ： vue-router 组件复用导致路由参数失效怎么办？

解决方案 ：

1 、通 过 watch 监听 路由参数再发请求

1 watch：{

2 "router":function(){

3 this.getData(this.$router.params.xxx)

4 }

5 }

2 、用 :key 来阻止复用

router-view :key="$route.fullPath"

19、谈一下对 vuex 的个人理解

vuex 是什么

vuex 是一个专为 Vue 应用程序开发 的状态管理器， 采用集中式 存储管理 应用的所有组件的状态。每 一个 vuex 应用的核心就是 store（仓库）。“store” 基本上就是一个容器，它包含着应用中大部分 的状态 (state)。

为什么需要 vuex

由于组件只维护自身的状态(data)，组件创建时或者路由切换时，组件会被初始化，从而导致 data 也 随之销毁。

使用方法

在 main.js 引入 store，注入。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。

什么场景下会使用到 vuex 如果是 vue 的小型应用，那么没有必要使用 vuex，这个时候使用 vuex 反而会带来负担。组件之间的 状态传递使用 props、自定义事件来传递即可。 但是如果 涉及到 vue 的大型应用 ，那么就需要类似于 vuex 这样的集中管 理状态的状态机来管理所有 组件的状态。例如登录状态、加入购物车、音乐播放等，总之只要是开发 vue 的大型应用，都推荐使 用 vuex 来管理所有组件状态

主要包括以下几个模块：

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化状态。

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。

Mutation:是唯一更改 store 中状态的方法，且必须是同步函数。

Action:用于提交 mutation，而不是直接变更状态，可以包含任意异步请求。

Module:允许将单一的 Store 拆分更多个 store 且同时保存在单一的状态树中。

20、Vuex 页面刷新数据丢失怎么解决？

需要做 vuex 数据持久化 ，一般使 用本地储存的方案 来保存数据，可以自己设计存储方

案，也可以使用第三方插件。

推荐使用 vuex-persist ( 脯肉赛斯特 ) 插件，它是为 Vuex 持久化储存而生的一个插件。

不需要你手动存取 storage ，而是直接将状态保存至 cookie 或者 localStorage 中。

21、vue 中使用了哪些设计模式？

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。

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。

发布-订阅模式。（vue 事件机制）

观察者模式。（响应式数据原理）

装饰器模式（@装饰器的用法）

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。

22、你都做过哪些 Vue 的性能优化？

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

23、nextTick 的作用是什么？他的实现原理是什么

作用 ：vue 更新 DOM 是异步更新的，数据变化，DOM 的更新不会马上完成， nextTick的回调是在下次 DOM 更新循环结束之后执行的延迟回调 。

实现原理 ：nextTick 主要使用了 宏任务和微任务 。根据执行环境分别尝试采用

Promise：可以将函数延迟到当前函数调用栈最末端

MutationObserver ：是 H5 新加的一个功能，其功能是监听 DOM 节点的变动，在所有 DOM 变动完成后，执行回调函数setImmediate：用于中断长时间运行的操作，并在浏览器完成其他操作（如事件和显 示更新）后立即运行回调函数

如果以上都不行则采用 setTimeout 把函数延迟到 DOM 更新之后再使用，原因是宏任务消耗大于微任务，优先使用微任务，最后使用消耗最大的宏任务

24、keep-alive 使用场景和原理

keep-alive 组件是 vue 的内置组件 ，用于 缓存内部组件 实例。这样做的目的在于，keep

alive 内部的组件切回时， 不用重新创建 组件实例，而直接使用缓存中的实例，一方面能够

避免创建组件带来的开销，另一方面可以保留组件的状态 。

keep-alive 具有 include 和 exclude 属性，通过它们可以控制哪些组件进入缓存。另外它 还提供了 max 属性，通过它可以设置最大缓存数，当缓存的实例超过该数时，vue 会移除最久没有使用的组件缓存。

受keep-alive的影响，其内部所有嵌套的组件都具有两个生命周期钩子函数，分别是activated 和 deactivated，它们分别在组件激活和失活时触发。第一次 activated 触发是在 mounted 之后

在具体的实现上，keep-alive 在内部维护了一个 key 数组和一个缓存对象

1 // keep-alive 内部的声明周期函数

2 created () {

3 this.cache = Object.create(null)

4

5 this.keys = []

6 }

key 数组记录目前缓存的组件 key 值，如果组件没有指定 key 值，则会为其自动生成一个

唯一的 key 值 cache 对象以 key 值为键，vnode 为值，用于缓存组件对应的虚拟 DOM

在 keep-alive 的渲染函数中，其基本逻辑是判断当前渲染的 vnode 是否有对应的缓存，

如果有，从缓存中读取到对应的组件实例；如果没有则将其缓存。、

当缓存数量超过 max 数值时，keep-alive 会移除掉 key 数组的第一个元素。

25、Vue.set 方法原理

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。

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）

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

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。

26. Proxy 相比 defineProperty 的优势在哪里

Vue3.x 改用 Proxy 替代 Object.defineProperty

原因在于 Object.defineProperty 本身存在的一 些问题 ：

Object.defineProperty 只能劫持对象属性的 getter 和 setter 方法。

Object.definedProperty 不支持数组(可以监听数组,不过数组方法无法监听自己重写)，更准确的说是不支持数组的各种 API(所以 Vue 重写了数组方法。

而相比 Object.defineProperty，Proxy 的优点在于：

Proxy 是直接代理劫持整个对象。

Proxy 可以直接监听对象和数组的变化，并且有多达 13 种拦截方法。

目前，Object.definedProperty 唯一比 Proxy 好的一点就是兼容性，不过 Proxy 新标准

也受到浏览器厂商重点持续的性能优化当中