### 1. Vue的基本原理

**Javascript对象-->作为data-->vue遍历所有属性--->使用Object.definePrototype()把属性转化为getter/setter--->vue可以追踪依赖（当属性被更改，通知变更）**

**Watcher-->渲染过程中接触过的属性记录为当前页面的依赖--->setter触发-->通知watcher-->重新渲染**

### 2. 双向数据绑定的原理

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，**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。**

消息订阅器Dep-->统一管理（监听器observer和订阅者watcher）

监听器observer:监听所有属性，有变化则通知订阅者

订阅者watcher：收到变化，执行相应的函数，更新视图

解析器compile: 解析模板指令templa，替换模板数据，初始化视图；把模板指令对应的**节点**绑定对应的**更新函数，**初始化订阅器

Watsher:

当收到dep.notice() 调用自身的updata()对视图进行更新，并触发compile的回调函数

### 3. 使用 Object.defineProperty() 来进行数据劫持有什么缺点？

不能拦截:通过下标方式修改数组数据或者给对象新增属性

对于数组：重写数组的方法，解决了数组监听的问题

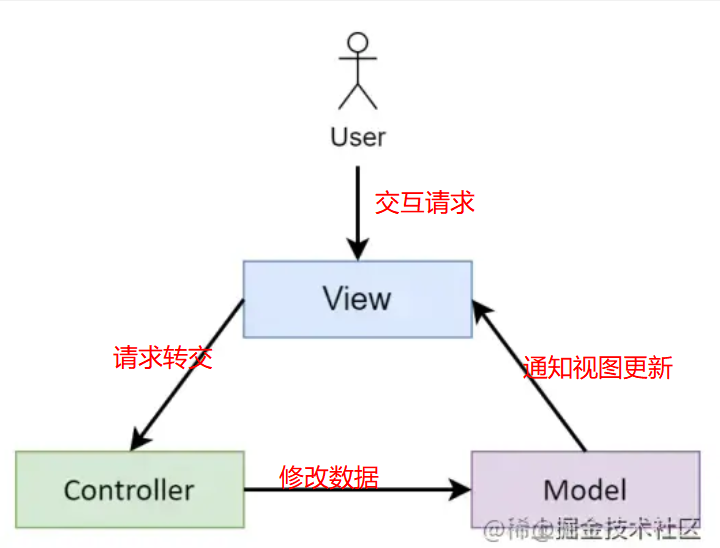
提供vue.set()给对象新增属性

在 Vue3.0 中通过使用 Proxy 对对象进行代理，从而实现数据劫持。使用Proxy 的好处是它可以完美的监听到任何方式的数据改变，唯一的缺点是兼容性的问题

### 4. MVVM、MVC、MVP的区别

**（1）MVC：单向数据流 通过分离 Model、View 和 Controller**

* View接受用户的交互请求。
* **View将请求转交给Controller。**
* Controller操作Model进行数据更新。
* 数据更新之后，Model通知View数据变化。
* View显示更新之后的数据。



1. **MVP ：双向数据流 model.view,presenter**

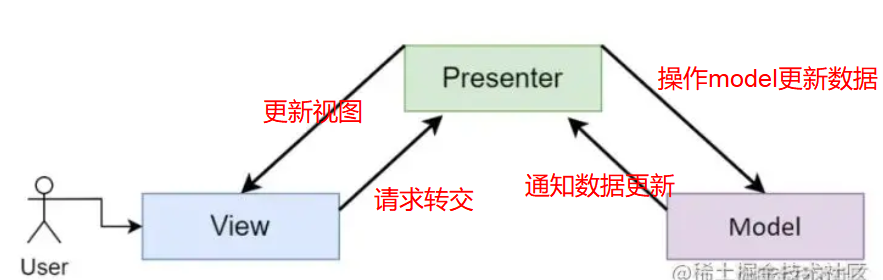
· View 接收用户交互请求

· View 将请求转交给 Presenter(V调用P接口)

· Presenter 操作Model进行数据更新(P调用M接口)

· Model 通知Presenter数据发生变化(M调用P接口)

· Presenter 更新View数据(P执行接口,V相应回调)



**Model层与View层完全分离，**修改V层不会影响M层，降低了耦合性。View->Model，Model->View的手动同步逻辑，造成Presenter比较笨重，维护起来会比较困难。

（3）MVVM：**Model、View、ViewModel：**

**双向数据绑定，实现了 Model和View的数据自动同步**

### **分离视图（View）和模型（Model）**，**提高复用性**

**对于⼤型的图形应⽤程序，视图状态较多，ViewModel的构建和维护的成本都会⽐较⾼。**

### [5. Computed](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "5-computed-和-watch-的区别) 和 Watch 的区别

computed 计算属性 : **依赖其它属性值**，并且 computed 的值**有缓存**，**不支持异步监听**，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 侦听器 : 更多的是**观察**的作用，**无缓存性**，**支持异步监听**，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。**

* **在computed中，属性有一个get方法和一个set方法，当数据发生变化时，会调用set方法。**
* **Immediate(初始化立即执行)，deep(深度监听)，handler(new,old){}**

### 6. Computed 和 Methods 的区别

* 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**；
* method 调用总会执行该函数。

### slot是什么？有什么作用？原理是什么？

### slot又名插槽，**slot元素作为承载分发内容的出口**。

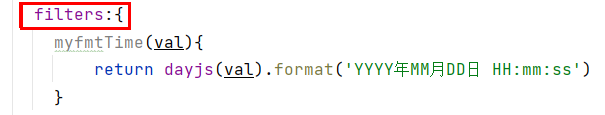
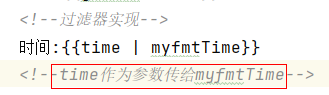
### slot又分三类，默认插槽，具名插槽和作用域插槽。

* 默认插槽：又名匿名插槽，**没有设置name属性**的插槽。一个组件内只有有一个匿名插槽。
* 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。
* 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。

# slot 与 props 的区别：通过props属性，父组件可以向子组件传递属性、方法，可是父组件不能通过属性传递带标签的内容、甚至是组件，而插槽可以。

### [8.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "8-过滤器的作用如何实现一个过滤器) 过滤器的作用，如何实现一个过滤器

**filters不会修改数据，而是过滤数据**



### 9. 如何保存页面的当前的状态

##### 1,把状态信息state保存在本地存储,页面刷新成功后再次从本地存储中读取state数据

(1)**vuex中的数据发生改变时**触发localStorage的存储操作

localstorage.setItem('state', JSON.stringify(**this.$store.state**))

(2)页面加载的时候在**created**中获取本地存储中的数据

localStorage.getItem('state') && **this.$store.replaceState**(JSON.parse(localStorage.getItem('state')));

3,在Vue中，还可以是用**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：**防止重复渲染DOM。**

* 初次进入时：**created > mounted > activated**；退出后触发 **deactivated**
* 再次进入：会触发 activated；

**Keep-alive的属性：**

**include**定义缓存白名单，keep-alive会缓存命中的组件；

**exclude**定义缓存黑名单，被命中的组件将不会被缓存；

**max**定义缓存组件上限，超出上限使用LRU的策略置换缓存数据。

**获取keep-alive包裹的组件名，对比白名单，如果匹配则查看是否缓存，没缓存就缓存，如果超出最大个数，则删掉最久未使用的实例，将组件实例的keepAlive属性值设置为true。**

### [11. v-if](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "11-v-ifv-showv-html-的原理)、v-show、v-html 的原理

**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元素的display样式属性控制显隐；**

**v-if有更高的切换消耗；v-show有更高的初始渲染消耗**

v-if适合运营条件不大可能改变；v-show适合频繁切换。

### 14. v-model 是如何实现的，语法糖实际是什么？

**（1）作用在表单元素上 :实现数据的双向绑定**

**（2）作用在组件上：实现父子组件间的通信**

**v-model在组件上的使用：在子组件标签上绑定数据，在子组件中props接收参数，this.$emit(‘input’,修改值)**

Props:父组件通过在子组件标签上绑定数据，实现父向子传递数据

自定义事件：父组件通过在子组件标签上绑定事件，并定义回调函数，在子组件中触发，实现子向父传递数据

### 16. data为什么是一个函数而不是对象

每次函数执行完毕后都会**返回一个新的对象，这样的话每个组件都会维护一份独立的对象（私有数据）**

### $nextTick 原理及作用

Vue 在修改数据后，视图不会立刻更新，而是等**同一事件循环**中的所有数据变化完成之后，再统一进行视图更新。（**数据异步更新**）

在Vue生命周期的created()钩子函数进行的DOM操作一定要放Vue.nextTick()的回调函数中

**nextTick触发的时机：**

**同一事件循环中的代码执行完毕 -> DOM 更新 -> nextTick callback触发**

### [23. Vue data](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "23-vue-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) 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 在更新 DOM 时是异步执行的。**只要侦听到数据变化， Vue 将开启一个队列，并缓冲在同一事件循环中发生的所有数据变更。

### [21. Vue](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "21-vue-单页应用与多页应用的区别) 单页应用与多页应用的区别

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。**

**用户体验好、快，内容的改变不需要重新加载整个页面，初次加载耗时多**

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**。

### [22. Vue template](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "22-vue-template-到-render-的过程) 到 render 的过程

vue的模版编译过程主要如下：**template -> ast抽象语法树 -> render函数**

1. **调用parse方法将template转化为ast（抽象语法树）**

**（2）优化ast并标记静态节点**和静态根节点，

（3）ast抽象语法树编译**render函数**。

在**beforeMount和mounted之间执行render函数生成VNode**，然后通过**patch(VNode)生成dom树并挂载**，调用mounted。

**Template->ast->标记静态节点->render函数->Vnode->path(Vnode)-->DOM**

**mixin混入**

**多个组件共享一个配置，所有可以公用的配置项都可以写入【混合】中**

**除了生命周期（冲突，组件内的+混合中的）外的配置如果名字冲突，都以组件中的为主**

### 子组件可以直接改变父组件的数据吗？

### **子组件不可以直接改变父组件的数据,**只能通过**** $emit ****派发一个自定义事件，父组件接收到后，由父组件修改。****

1. Vue是如何收集依赖的？

在**初始化 Vue** 的每个组件时，**会对组件的 data 进行初始化**，就会将由普通对象变成响应式对象，**在这个过程中便会进行依赖收集的相关逻辑**，

const dep = new Dep()**实例化一个 Dep 的实例**，然后在 **get 函数**中通过 **dep.depend() 进行依赖收集**。

### 28. 对 React 和 Vue 的理解，它们的异同

**相似之处：**

* 都使用了Virtual DOM（**虚拟DOM**）提高重绘性能；
* 都有**props的概念，允许组件间的数据传递；**
* 都鼓励**组件化**应用，将应用分拆成一个个功能明确的模块，提高复用性。

**不同之处 :**

**1）数据流（单双向）**

**Vue默认支持数据双向绑定**，而React一直提倡**单向数据流**

**2）组件化**

React中render函数是**支持闭包特性的，所以import的组件在render中可以直接调用**。

但是在Vue中，由于模板中使用的数据都必须**挂在 this 上进行一次中转**，所以 **import** 一个组件完了之后，还需要在 **components 中再声明下**。

**4）监听数据变化的实现原理不同**

* Vue 通过 **getter/setter 以及一些函数的劫持**
* React 默认是通过**比较引用的方式进行的**

### 30. assets和static的区别

Assets:中的静态资源打包时会被压缩，代码格式化，最后放在static文件中上传到服务器

vm.$set 的实现原理是：

* 通过调用 **defineReactive 方法进行响应式处理**（ defineReactive 方法就是 Vue 在初始化对象时，**给对象属性采用 Object.defineProperty 动态添加 getter 和 setter 的功能所调用的方法**）

### [35.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "35-对ssr的理解) 对SSR的理解

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**

首屏加载速度更快，服务端负载加大，**服务器端渲染只支持beforeCreate和created两个钩子**

### [39. vue](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "39-vue初始化页面闪动问题)初始化页面闪动问题

网页还在加载 Vue.js ，而导致 Vue 来不及渲染，这时页面就会显示出 Vue 源代码。

**v-cloak 指令是解决屏幕闪动的好方法。**[v-cloak] { display: none;}

### [40. extend](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "40-extend-有什么作用) 有什么作用

Vue.extend生成vueComponent构造函数（组件就是该构造函数）

### 1. 说一下Vue的生命周期

初始化->beforecreate->初始化data(),methods等-->created-->开始编译（template-->ast-->render()）-->beforemount--->渲染

（render()-->vnode-->path(vnode)-->dom）-->mounted

数据更新-->berforeupdate(数据已经更新)-->重新渲染-->updated

berforDestroy:销毁前，属性和方法还可以使用

Destroyed:销毁

Activated:进入组件 deativated:退出组件

一，渲染过程

    1，解析模板为render函数（或在开发环境已完成，vue-loader）

    2，**触发响应式，**监听data属性getter setter

    3，执行render函数，生成vnode，patch（elem，vnode）

二，更新过程

    1，修改data，触发setter（此前在getter中已被监听）

    2，重新执行render函数，生成newVnode

    3，patch（vnode，newVnode）

### [2. Vue](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "2-vue-子组件和父组件执行顺序) 子组件和父组件执行顺序

父组件beformounte之后 创建和挂载子组件 父组件mounted

父组件beforupdate之后 更新子组件 父组件update（）

父组件BeforeDestroy 子组件销毁 父组件destoryed

### 3. created和mounted的区别

Created初始化结束，还未开始编译渲染，无法获得dom节点

Mounted 挂载结束，可以获得dom节点

### 4. 一般在哪个生命周期请求异步数据

我们可以在钩子函数 **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。

建议在created，能更早获取数据库数据，减少页面加载时间，对于SSR，不支持另外两个钩子

## 1，组件通信

**（1）父子组件间通信**

**props/$emit:**父组件通过props向子组件传递数据，子组件通过$emit和父组件通信

**父组件给子组件绑定事件（v-on:test=”fn”），并定义回调函数，子组件通过this.$emit(test)触发父组件中的回调函数**

**project/inject :(非响应式)**

* **project 钩子用来发送数据或方法，写法和data()一致**
* **inject钩子用来接收数据或方法**

**在父组件中发送数据：project(){return { nums:this.num}}**

**在子组件中接收数据：inject:[‘num’]**

**ref/$parent:**

**$refs:获得子组件的实例**

**$parent:获得上一级父组件的实例**

**父组件：在子组件的实例化标签中打标识ref=’xxx’,就可以通过this.$refs.xxx操作子组件上的数据和方法**

1. **兄弟组件间通信**

**EventBust:**

$parent:

**兄click设置点击事件，用$emit设置通道传参给中转站，弟通过$on接收来自中转站的参数**

**this.$patent.$on('confirm', handleConfirm)**

**this.$patent.$emit('confirm', list)**

**（3）任意组件之间**

**eventBust :**

1. 在vue实例的berforecreate钩子中安装全局事件总线：

**vue.prototype.$bus=this**

1. 接收方：往事件总线上绑定事件，定义回调函数：

**this.$bus.$on(事件名，回调函数)**

1. 发送方：触发事件，传递数据

**This.$bus.$emit(事件名，数据)**

1. 在销毁之前解绑事件

**This.$bus.$off(事件名)**

1. **隔代组件通信**

**$attrs/$listeners:**

**爷-->sun:**

**没有声明props接收的参数，都保存在$attrs中，可以在父组件中把没有接收的数据传递给子组件**

**<sun v-bind="$attrs" ><sun/>**

**Sun-->爷：爷爷给父组件绑定自定义事件，父组件再把监听函数传递给子组件**

**<sun v-on="$listeners"><sun/>**

### [1. Vue-Router](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "1-vue-router-的懒加载如何实现) 的懒加载如何实现

(1)import按需加载

Const dd=()=>import(XXXX)

(2)Vue异步组件

Component:resolve=>require([XXXX],resolve)

1. Webpack

Const list=r=>require.ensure([],()=>r(require(XXX)),’list’)

### [2.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "2-路由的hash和history模式的区别) 路由的hash和history模式的区别

#### [1. hash](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "1-hash模式)模式（不会发送请求）

**特点**：**hash值会出现在URL里面，但是不会出现在HTTP请求中**，对后端完全没有影响。所以**改变hash值（url的锚点值），不会重新加载页面。**hash路由被称为是前端路由

**原理：** hash模式的主要原理就是**onhashchange()事件**：

**Hash改变-->onhanshchange事件触发-->按规则加载相应的代码（没有发送请求）-->hash变化对应的url也会被浏览器记录下来（实现页面的前进和后退）**

### 3. 如何获取页面的hash变化

#### 1，监听$route的变化 （采用深度监听）

2，读取window.location.hash值是否变化

#### 2. history模式（会发送请求）

**简介：** history模式的URL中没有#，它使用的是传统的路由分发模式，即用户在输入一个URL时，**服务器会接收这个请求，并解析这个URL**，然后做出相应的逻辑处理。

**如果后台没有设置相应的路由配置，会返回404**

**API：** (浏览器不会立即向后端发送请求)

window.history属性指向 History 对象，它表示当前窗口的浏览历史。

* **修改历史状态**：

**pushState() :往历史记录中添加一条记录**

添加新记录后，浏览器地址栏立刻显示新增加的历史记录，但并不会跳转到2.html，甚至也不会检查2.html是否存在，它只是成为浏览历史中的最新记录。

**replaceState()** :**修改当前的历史记录**

* 1，以上两个api虽然修改了url，但浏览器不会立即向后端发送请求。**改变url但又不刷新页面的效果**

**2，history.pushState()或history.replaceState()不会触发popstate和onhashchange事件。**

* **切换历史状态：** 包括**forward()、back()、go()**三个方法，对应浏览器的前进，后退，跳转操作。

**如果需要切换到history模式，只需要在路由实例中配置mode属性，值为history**

#### [3.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "3-两种模式对比) 两种模式对比

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，**

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**；

history模式下，**前端的url必须和实际向后端发起请求的url一致，如果没有对用的路由处理，将返回404错误。**

**history模式解决404问题**

**中间件：connect-history-api-fallback**

**下载之后引入 history=require(XXX) app.use(history())**

增加中间件的配置(如nginx)，所有未匹配到的路由都返回index.html。

**或者采用全局路由守卫：**

**$route.matched**

1 一个数组，包含当前路由的所有嵌套路径片段的路由记录

**判断to.matched中是否含有孩路由片段,则跳转到指定页面**

### [4.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "4-route-和router-的区别) *route*和router 的区别

   $route对象表示**当前的路由信息，包含了当前 URL 解析得到的信息**。包含当前的路径，参数，query对象等。

  $router对象是**全局路由的实例**，是router构造方法的实例。

### [5.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "5-如何定义动态路由如何获取传过来的动态参数) 如何定义动态路由？如何获取传过来的动态参数？

1. **params参数**

**1，在路由表中配置,使用占位符声明接收params参数**

**path:’/home/:id’**

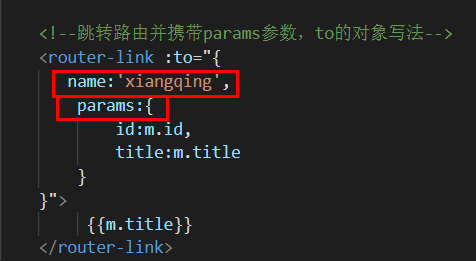
**2，传递的方式：直接在路径后面拼接参数，对象写法必须配置name属性**

**<router-link :to=”/home/${this.id}”>**

**$route.params.id //客户端的参数提取**

**This.$http.get(‘/home’,this.id)**

**req.params.id //服务端的提取参数**



1. **Query参数**

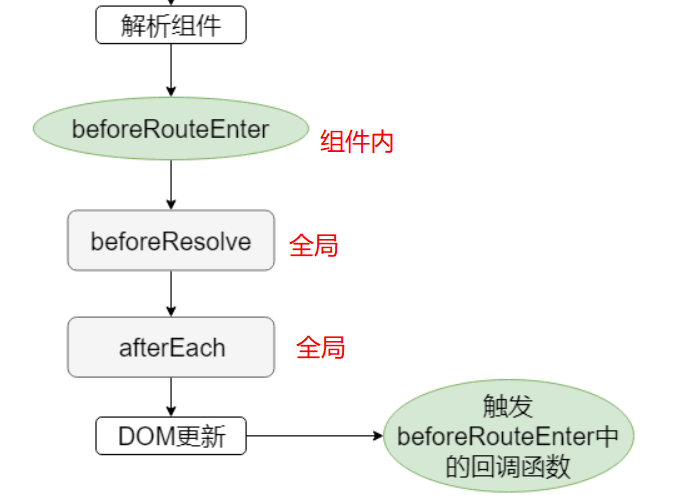
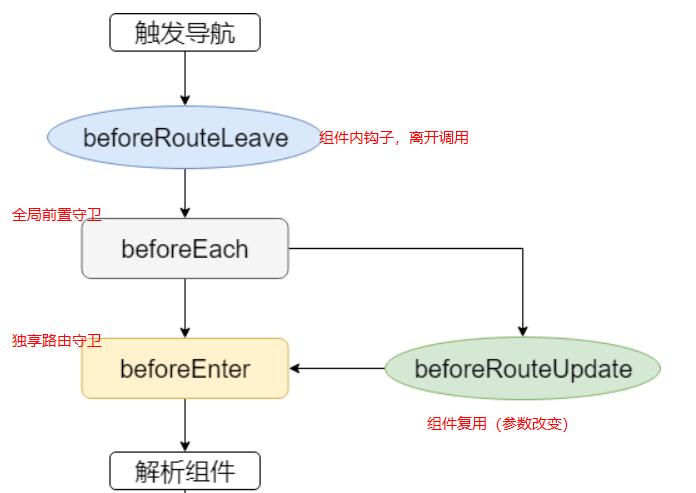
* 配置路由格式：/router，也就是普通配置
* 传递的方式：对象中使用query的key作为传递方式
* 传递后形成的路径：**/route?id=123**

**对象模式可以使用path或name**

**<router-link :to=’/home?id=${this.id}’>**

**<router-link :to=”{path:”xxxx”,query:{id:this.id}}”**

### Vue-router 路由钩子在生命周期的体现



**1,全局路由导航：**

* router.beforeEach 全局前置守卫 进入路由之前
* router.beforeResolve 全局解析守卫（2.5.0+）**在 beforeRouteEnter 调用之后调用**
* router.afterEach 全局后置钩子 进入路由之后（没有next）

**2，独享路由守卫（只有一个）beforeEnter**

**3，组件内钩子**

这三个钩子都有三个参数∶to、from、next

* **beforeRouteEnter**∶ 进入组件前触发（**访问不到this**）
* **beforeRouteUpdate**∶ 当前地址改变并且改组件被复用时触发，举例来说，带有动态参数的路径foo/∶id，在 **/foo/1 和 /foo/2 之间跳转的时候，**由于会渲染同样的foo组件，这个钩子在这种情况下就会被调用
* **beforeRouteLeave**∶ 离开组件被调用

### [7. Vue-router](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "7-vue-router跳转和locationhref有什么区别)跳转和location.href有什么区别

* 使用 location.href= /url ：**刷新了页面**；
* 使用 history.pushState( /url ) ：**无刷新页面，静态跳转**；
* 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。**

### 8. params和query的区别（name,是否显示参数，刷新是否丢失数据）

**用法**：对象写法中，query要用path来引入，**params要用name来引入**

**url地址显示**：**query在浏览器地址栏中显示参数**，params不显示

**注意**：query刷新不会丢失query里面的数据 **params刷新会丢失 params里面的数据。**

### 10. 对前端路由的理解

**单页面应用中，允许页面在不刷新的情况下更新页面内容，前端路由需要实现一个url，一套资源，映射不同的试图内容**

* **拦截用户的刷新操作，把刷新这个动作完全放到前端逻辑里消化掉。**
* **感知 URL 的变化。根据这些变化、用 JS 去给它生成不同的内容。**

**类似于修改hash值：hash值会出现在URL里面，但是不会出现在HTTP请求中**

### [1. Vuex](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "1-vuex-的原理) 的原理

Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式。**

**state => 基本数据（只用来读取的状态集中放在 store 中）**

**getters => 从基本数据派生的数据**

**mutations => 修改数据，同步（改变状态的方式是在 mutations 中）**

**actions => 修改数据，异步（异步逻辑应该封装在 action 中）**

**modules => 模块化Vuex**

流程：

**this.$store.dispatch（‘A’,this.val）-->**用户交互触发

**A（context，val）{context.commit（‘B’，val）}--->**actions中执行A

**B（state,val）{state.XXX=val}--->**mutations中执行B

**如果无更多逻辑操作，也可以跳过actions阶段，直接在用户交互触发commit函数（代替dispatch函数）**

### [2. Vuex](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "2-vuex中action和mutation的区别)中action和mutation的区别

* 在视图更新时，**先触发actions，actions再触发mutation**

**Mutations:必须同步执行，修改state的唯一路径，参数是state**

**Actions:可以异步请求，不能操作state,参数是context(调用commit方法)**

1. Vuex 和 localStorage 的区别

**（1）最重要的区别 存储位置**

###### store里的数据是保存在运行内存中的,当页面刷新时，页面会重新加载vue实例，store里面的数据就会被重新赋值初始化，**刷新页面时vuex存储的值会丢失**

localstorage 则以文件的方式**存储在本地**，只能存储字符串类型的数据，**存储对象需要 JSON的stringify和parse方法进行处理。**

**（2）应用场景**

* Vuex 是一个专为 Vue.js 应用程序开发的**状态管理模式**。它采用集中式**存储管理应用的所有组件的状态**，并以相应的规则保证状态以一种可预测的方式发生变化。vuex**用于组件之间的传值。**
* **localstorage是本地存储，是将数据存储到浏览器的方法，一般是在跨页面传递数据时使用 。**
* Vuex能做到**数据的响应式**，localstorage不能

**vuex页面刷新数据丢失的解决办法**

在vue项目中用vuex来做全局的状态管理， 发现当刷新网页后，保存在vuex实例store里的数据会丢失。

**原因：**

因为store里的数据是**保存在运行内存中的**,当页面刷新时，页面**会重新加载vue实例，store里面的数据就会被重新赋值初始化**

**解决思路：**

**将state的数据保存在localstorage、sessionstorage或cookie中**（[三者的区别](https://stackoverflow.com/questions/19867599/what-is-the-difference-between-localstorage-sessionstorage-session-and-cookies)），这样即可保证页面刷新数据不丢失且易于读取。

由于vue是单页面应用，操作都是在一个页面跳转路由，因此**sessionStorage**较为合适：

1，在app.vue的**created**钩子中替换保存的数据：

如果有缓存，则把其取出与当前**this.$store.state**合并：

**if(sessionStorage.getItem('store')){**

**this.$store.state.replaceState(Object.assign({},**

**this.$store.state,JSON.parse(sessionStorage.getItem('store'))))**

**}**

2，在页面监听berforeunload事件用于保存数据：

**window.addEventListener('beforeunload',()=>{**

**sessionStorage.setItem('store',JSON.stringify(this.$store.state))**

**})**

### [5.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "5-为什么要用-vuex-或者-redux) 为什么要用 Vuex 或者 Redux

由于传参的方法对于多层嵌套的组件将会非常繁琐，把组件的共享状态抽取出来，以一个全局单例模式管理。在这种模式下，组件树构成了一个巨大的"视图"，不管在树的哪个位置，任何组件都能获取状态或者触发行为。

### 7. Vuex和单纯的全局对象有什么区别？

* **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。
* **不能直接改变 store 中的状态**。改变 **store 中的状态的唯一途径就是显式地提交 (commit) mutation。**这样可以方便地跟踪每一个状态的变化，从而能够实现一些工具帮助更好地了解我们的应用。

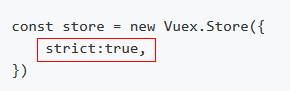
### [8.](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "8-为什么-vuex-的-mutation-中不能做异步操作) 为什么 Vuex 的 mutation 中不能做异步操作？

* **如果mutation支持异步操作，就没有办法知道状态是何时更新的，无法很好的进行状态的追踪，给调试带来困难**。

### 9. Vuex的严格模式是什么,有什么作用，如何开启？

在严格模式下，**不是由mutation函数引起的状态变更，将会抛出错误。**这能保证所有的状态变更都能被调试工具跟踪到。

在Vuex.Store 构造器选项中开启,如下



**11,一些vuex的辅助函数**

**mapState**借助mapState生成**计算属性**，从state中读取数据：

**mapGetters**借助mapGetters生成**计算属性**，从getters中读取数据

**mapMutations**

借助mapMutations生成对应的**方法**，**调用commit去联系mutations**

**mapActions**

借助mapActions生成对应的**方法**，**调用dispatch去联系actions**

### [1. Vue3.0](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "1-vue30有什么更新)有什么更新

**（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。**可以监测属性的增加和删除，检索数组的索引和长度的变化**
* 消除了 Vue 2 当中基于 Object.defineProperty 的实现所存在的很多限制：

### [3. Vue3.0](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "3-vue30-为什么要用-proxy) 为什么要用 proxy？

* 不需用使用 **Vue.$set** 或 **Vue.$delete** 触发响应式。
* 全方位的数组变化检测，消除了Vue2 无效的边界情况。
* **支持 Map，Set，WeakMap 和 WeakSet。**

### [4. Vue 3.0](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "4--vue-30-中的-vue-composition-api) 中的 Vue Composition API？

在 Vue2 中，代码是 **Options API** 风格的，也就是通过填充 (option) data、methods、computed 等属性来完成一个 Vue 组件。

Composition API本质上是**将 Options API 背后的机制暴露给用户直接使用，这样用户就拥有了更多的灵活性**，也使得 Vue3 更适合于 TypeScript 结合。

### 1. 对虚拟DOM的理解？

**将多次DOM修改的结果一次性的更新到页面上，**从而有效的减少页面渲染的次数，减少修改DOM的重绘重排次数，提高渲染性能。

在挂载前，把DOM树结构用**js对象**的形式表示出来，**保存js对象树**

**当页面状态发生改变，根据变更的状态，重构js对象树，新旧对象树对比，记录差异，把差异的地方应用到真实DOM中**

### 4. 虚拟DOM真的比真实DOM性能好吗

* **首次渲染**大量DOM时，由于**多了一层虚拟DOM的计算**，会比innerHTML**插入慢。**
* 正如它能保证性能下限，**在真实DOM操作的时候进行针对性的优化时，**还是更快的。

### [5. DIFF](https://gitee.com/AiShiYuShiJiePingXing/lovebetterworld/blob/master/1.1 前端相关/1.9 前端面试题/6 前端面试题之Vue篇.md" \l "5-diff算法的原理)算法的原理

基本流程：数据改变-->虚拟DOM（计算变更）-->操作真实DOM-->视图更新

Diff算法：

当数据发生改变时，set方法会调用Dep.notify通知所有订阅者Watcher，**订阅者就会调用patch给真实的DOM打补丁**，更新相应的视图

先遍历老节点，再遍历新节点，比较只会在同层级进行, 不会跨层级比较

### 1，了解节点对象：

{**sel（选择器,div）**，data(样式)，childre（）||text(),elm(对应的真实DOM元素)，**key**}

### 2，path函数：

**1）使用场景：**

1页面首次渲染的时候执行，把vnode渲染到空的容器中  path(container,vnode)

2数据更新时，新节点替换老节点  path(vnode,newnode)

**2）函数具体内容：**

1第一个参数不是vnode类型，初始化状态，全部新建DOM元素

2，是vnode类型，调用**samvnode**函数（比较传入的两个节点的**key和sel**）

        相同：调用**pathVnode**函数

        不相同：通过**createElm**创建DOM元素,插入新的DOM元素，删除老的DOM元素

### 3，pathVnode函数（当key和sel相同的时候调用）

1）函数主要内容：

     对比两个节点的children或者（text）,**如果相同，直接返回，如果不同，**一切以新的vnode为准****

1,新vnode有children，旧的需要删除原来的text，添加新的children，或者更新children

2,新vnode有text，旧的需要删除原来的children，添加新的text，或者更新text

3**,新老vnode 都有children ，更新children的时候需要用updateChildren**

### 4,UpdateChildren函数：

 1,新节点的start与旧节点的start比较，,新节点的start与旧节点的end比较,新节点的end与旧节点的start比较，,新节点的end与旧节点的end比较，（start++,end--）

2，**如果相同，调用pathVnode函数（递归）**

3，如果都不相同：拿到新节点中**第一个孩子的key**,与老节点的**所有孩子的key**比较，找不到，直接创建DOM元素，找到了（key相等）再比较sel,如果sel不相等，创建DOM元素，**如果sel相等，递归调用pathVnode函数**

### 6. Vue中key的作用

**1,V-if:标识一个独立的元素**

如果切换前后含有**相同类型的元素**，那么这个元素就会被复用。比如：**相同的 input 元素，那么切换前后用户的输入不会被清除掉，这样是不符合需求的，用key来标识，就不会被复用**

**2,V-for:实现高效的复用**

用 v-for 更新已渲染过的元素列表时，它默认使用“**就地复用**”的策略。如果数据项的**顺序发生了改变**，Vue 不会移动 DOM 元素来匹配数据项的顺序，而是简单复用此处的每个元素。**用key标识，就可以正确的实现复用**

**3,Diff:**

key 是为 Vue 中 vnode 的唯一标记，通过这个 key，diff 操作可以更准确、更快速

* 更准确：**避免就地复用的原则**，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。
* 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**

### 7. 为什么不建议用index作为key?

如果数组的顺序怎么颠倒，index 都是 0, 1, 2...这样排列，**导致 Vue 会复用错误的旧子节点**，做很多额外的工作。

****Vue3 diff的优化：****

**diff算法优化、静态提升、事件侦听器缓存**

在创建虚拟DOM树的时候，**会根据DOM中的内容会不会发生变化，添加一个静态标记。**那么之后在与上次虚拟节点进行对比的时候，就**只会对比这些带有静态标记的节点。**

Vue3中使用了静态提升后，对于不参与更新的元素，只会被创建一次，在渲染时直接复用即可