第三版：Vue 35 道

请讲述下 VUE 的 MVVM 的理解?

MVVM 是 Model-View-ViewModel 的缩写，即将数据模型与数据表现层通过数据驱动进行分离，从而只需要关系数据模型的开发，而不需要考虑页面的表现，具体说来如下：

1、 Model 代表数据模型：主要用于定义数据和操作的业务逻辑。

2、 View 代表页面展示组件（即 dom 展现形式）：负责将数据模型转化成 UI 展现出来。

3、 ViewModel 为 model 和 view 之间的桥梁：监听模型数据的改变和控制视图行为、处理用户交互。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，

4、 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。

VUE 的生命周期及理解?

1、 总共分为 8 个阶段，具体为：创建前/后，载入前/后，更新前/后，销毁前/后。

2、 创建前/后： 在 beforeCreated 阶段：ue 实例的挂载元素 el 还没有。

3、 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。

4、 更新前/后：当 data 变化时，会触发 beforeUpdate 和 updated 方法。

5、 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。

6、 具体讲解及应用

beforeCreate：在 new 一个 vue 实例后，只有一些默认的生命周期钩子和默认事件，其他的东西都还没创建，data 和 methods 中的数据都还没有初始化。不能在这个阶段使用 data 中的数据和 methods 中的方法

create：data 和 methods 都已经被初始化好了，如果要调用 methods 中的方法，或者操作 data 中的数据，最早可以在这个阶段中操作

beforeMount：执行到这个钩子的时候，在内存中已经编译好了模板了，但是还没有挂载到页面中，此时，页面还是旧的，不能直接操作页面的 dom 和获取 dom 对象

mounted：执行到这个钩子的时候，就表示 Vue 实例已经初始化完成了。此时组件脱离了创建阶段，进入到了运行阶段。如果我们想要通过插件操作页面上的 DOM 节点，最早可以在和这个阶段中进行

beforeUpdate： 当执行这个钩子时，页面中的显示的数据还是旧的，data 中的数据是更新后的，页面还没有和最新的数据保持同步

updated：页面显示的数据和 data 中的数据已经保持同步了，都是最新的

beforeDestory：Vue 实例从运行阶段进入到了销毁阶段，这个时候上所有的 data 和 methods、指令、过滤器 ……都是处于可用状态。还没有真正被销毁

destroyed： 这个时候上所有的 data 和 methods、指令、过滤器 ……都是处于不可用状态。组件已经被销毁了。

v-if 和 v-show 的区别?

1、 共同点：都能控制元素的显示和隐藏；

2、 不同点：实现本质方法不同，v-show 本质就是通过控制 css 中的 display 设置为 none，控制隐藏，只会编译一次；v-if 是动态的向 DOM 树内添加或者删除 DOM 元素，若初始值为 false，就不会编译了。而且 v-if 不停的销毁和创建比较消耗性能。

3、 如果要频繁切换某节点，使用 v-show(切换开销比较小，初始开销较大)。如果不需要频繁切换某节点使用 v-if（初始渲染开销较小，切换开销比较大）。

v-if 和 v-for 同时使用在同一个标签上的表现?

当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。
所以，不推荐 v-if 和 v-for 同时使用。如果 v-if 和 v-for 一起用的话，vue 中的的会自动提示 v-if 应该放到外层去

v-for 中的 key 的理解?

需要使用 key 来给每个节点做一个唯一标识，Diff 算法就可以正确的识别此节点。主要是为了高效的更新虚拟 DOM。

vue 中 transition 的理解?

定义 transition 时需要设置对应的 name，具体语法为：需要动画的内容或者组件或者页面

过渡动画主要包含 6 个 class，分别为：

1、 v-enter：定义元素进入过渡的初始状态，在元素插入前生效，插入后一帧删除，

2、 v-enter-active：在元素插入前生效，在动画完成后删除，

3、 v-enter-to：在元素插入后一帧生效，在动画完成后删除，

4、 v-leave：离开过渡的初始状态，在元素离开时生效，下一帧删除

5、 v-leave-active：在离开过渡时生效，在动画完成后删除

6、 v-leave-to：离开过渡结束状态，在离开过渡下一帧生效，在动画完成后删除

v 会转化为对应的 transition 的 name 值

1、 当然我们也可以自定义这六个 class 可以直接在 transition 中设置对应的属性为对应的 class 名称，属性有：enter-class，enter-active-class，enter-to-class，leave-class，leave-active-class，leave-to-class

2、 在同时使用过渡和 css 动画的时候 可以设置 type 属性来制定 vue 内部机制监听 transitioned 或者 animationed 事件来完成过渡还是动画的监听

3、 如果需要设置对应的过渡时间，可以直接设置属性 duration，可以直接接收一个数字（单位为毫秒），也可以接收一个对象{enter:1000,leave:300}

4、 也可以设置过渡的钩子函数，具体有：before-enter，enter，after-enter，enter-cancelled，before-leave，leave，after-leave，leave-cancelled

vue 的自定义指令?

自定义指令分为全局指令和组件指令，其中全局指令需要使用 directive 来进行定义，组件指令需要使用 directives 来进行定义，具体定义方法同过滤器 filter 或者其他生命周期，具体使用方法如下：

全局自定义指令 directive(name,{})，其中 name 表示定义的指令名称（定义指令的时候不需要带 v-，但是在调用的时候需要哦带 v-），第二个参数是一个对象，对象中包括五个自定义组件的钩子函数，具体包括：

bind 函数：只调用一次，指令第一次绑定在元素上调用，即初始化调用一次，

inserted 函数：并绑定元素插入父级元素（即 new vue 中 el 绑定的元素）时调用（此时父级元素不一定转化为了 dom）

update 函数：在元素发生更新时就会调用，可以通过比较新旧的值来进行逻辑处理

componentUpdated 函数：元素更新完成后触发一次

unbind 函数：在元素所在的模板删除的时候就触发一次

钩子函数对应的参数 el,binding,vnode,oldnode,具体参数讲解如下：

a、el 指令所绑定的元素 可以直接操组 dom 元素

b、binding 一个对象，具体包括以下属性：

1、 name：定义的指令名称 不包括 v-

2、 value：指令的绑定值，如果绑定的是一个计算式，value 为对应计算结果

3、 oldvalue：指令绑定元素的前一个值，只对 update 和 componentUpdated 钩子函数有值

4、 expression：指令绑定的原始值 不对值进行任何加工

5、 arg：传递给指令的参数

6、 modifiers：指令修饰符，如：v-focus.show.async 则接收的 modifiers 为 {show：true，async：true}

c、vnode：vue 编译生成的虚拟 dom

d、oldVnode：上一个 vnode，只在 update 和 componentUpdated 钩子函数中有效

如果不需要其他钩子函数，可以直接简写为：
directive（“focus”,function(el,binding){})

vue 的实现原理?

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。

具体步骤：

第一步：需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，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

第二步：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

第三步：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，则功成身退。

第四步：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。

vue 的 diff 算法理解?

diff 算法的作用：用来修改 dom 的一小段，不会引起 dom 树的重绘

diff 算法的实现原理：diff 算法将 virtual dom 的某个节点数据改变后生成的新的 vnode 与旧节点进行比较，并替换为新的节点，具体过程就是调用 patch 方法，比较新旧节点，一边比较一边给真实的 dom 打补丁进行替换

具体过程详解：

a、在采用 diff 算法进行新旧节点进行比较的时候，比较是按照在同级进行比较的，不会进行跨级比较：

b、当数据发生改变的时候，set 方法会调用 dep.notify 通知所有的订阅者 watcher，订阅者会调用 patch 函数给响应的 dom 进行打补丁，从而更新真实的视图

c、patch 函数接受两个参数，第一个是旧节点，第二个是新节点，首先判断两个节点是否值得比较，值得比较则执行 patchVnode 函数，不值得比较则直接将旧节点替换为新节点。如果两个节点一样就直接检查对应的子节点，如果子节点不一样就说明整个子节点全部改变不再往下对比直接进行新旧节点的整体替换

d、patchVnode 函数：找到真实的 dom 元素；判断新旧节点是否指向同一个对象，如果是就直接返回；如果新旧节点都有文本节点，那么直接将新的文本节点赋值给 dom 元素并且更新旧的节点为新的节点；如果旧节点有子节点而新节点没有，则直接删除 dom 元素中的子节点；如果旧节点没有子节点，新节点有子节点，那么直接将新节点中的子节点更新到 dom 中；如果两者都有子节点，那么继续调用函数 updateChildren

e、updateChildren 函数：抽离出新旧节点的所有子节点，并且设置新旧节点的开始指针和结束指针，然后进行两辆比较，从而更新 dom（调整顺序或者插入新的内容 结束后删掉多余的内容）

vue 组件的通信（父子组件和非父子组件）？

父子组件通信

传递参数可以使用 props，传递函数可以直接在调用子组件的时候传递自定义事件，并使用$emit 来调用，例如：

```
//父组件
<div classs="parent">
    <child @getinfo="myname" :userinfo="usermessage"></child>
 <div>
 export default {
      data(){
           return {
                usermessage:'我是父亲'
           }
      },
      methods:{
           myname(name){
                console.log('我的名字叫'+name)
           }
      }
 }

//子组件
<div classs="child">
    来源：{{userinfo}}
    <button @click="getname">显示我的名字</button>
 <div>
 export default {
      props:['userinfo'],
      methods:{
```

```
            getname(){
                this.$emit('getinfo','bilibili')
            }
        }
}
```

兄弟组件通信

首先建立一个 vue 实例空白页（js 文件）

```
import Vue from 'vue'
  export default new Vue()
```

组件 a（数据发送方）通过使用 $emit 自定义事件把数据带过去

```
<template>
    <div>
        <span>A 组件->{{msg}}</span>
        <input type="button" value="把 a 组件数据传给 b" @click
="send">
    </div>
</template>
<script>
import vmson from "../../../util/emptyVue"
export default {
    data(){
        return {
            msg:{
                a:'111',
                b:'222'
            }
```

```
        }
    },
    methods:{
        send:function(){
            vmson.$emit("aevent",this.msg)
        }
    }
}
</script>
```

组件 b（数据接收方）使用而通过 $on 监听自定义事件的 callback 接收数据

```
<template>
 <div>
    <span>b 组件,a 传的的数据为->{{msg}}</span>
 </div>
</template>
<script>
        import vmson from "../../../util/emptyVue"
        export default {
            data(){
                    return {
                        msg:""
                    }
                },
            mounted(){
                vmson.$on("aevent",(val)=>{//监听事件 aevent，回调函数要使用箭头函数;
                    console.log(val);//打印结果：我是 a 组件的数据
                    this.msg = val;
                })
```

```
        }
    }
</script>
```

vue 的路由模式及区别?

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；

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，否则会报 404 错误

vue 与 react、angular 的比较?

Vue

轻量级框架：只关注视图层，是一个构建数据的视图集合，大小只有几十 kb；
简单易学：国人开发，中文文档，不存在语言障碍，易于理解和学习；
双向数据绑定：保留了 angular 的特点，在数据操作方面更为简单；
组件化：保留了 react 的优点，实现了 html 的封装和重用，在构建单页面应用方面有着独特的优势；

视图，数据，结构分离：使数据的更改更为简单，不需要进行逻辑代码的修改，只需要操作数据就能完成相关操作；

虚拟 DOM：dom 操作是非常耗费性能的，不再使用原生的 dom 操作节点，极大解放 dom 操作，但具体操作的还是 dom 不过是换了另一种方式；

运行速度更快:相比较与 react 而言，同样是操作虚拟 dom，就性能而言，vue 存在很大的优势。

React

相同点：

React 采用特殊的 JSX 语法，Vue.js 在组件开发中也推崇编写.vue 特殊文件格式，对文件内容都有一些约定，两者都需要编译后使用；中心思想相同：一切都是组件，组件实例之间可以嵌套；都提供合理的钩子函数，可以让开发者定制化地去处理需求；都不内置列数 AJAX，Route 等功能到核心包，而是以插件的方式加载；在组件开发中都支持 mixins 的特性。

不同点：

React 采用的 Virtual DOM 会对渲染出来的结果做脏检查；Vue.js 在模板中提供了指令，过滤器等，可以非常方便，快捷地操作 Virtual DOM。

Angular

相同点：

都支持指令：内置指令和自定义指令；都支持过滤器：内置过滤器和自定义过滤器；都支持双向数据绑定；都不支持低端浏览器。

不同点：

AngularJS 的学习成本高，比如增加了 Dependency Injection 特性，而 Vue.js 本身提供的 API 都比较简单、直观；在性能上，AngularJS 依赖对数据做脏检查，所以 Watcher 越多越慢；Vue.js 使用基于依赖追踪的观察并且使用异步队列更新，所有的数据都是独立触发的。

vue-roter 的钩子函数?

vue 路由钩子大致可以分为三类:

全局钩子

主要包括 beforeEach 和 aftrEach,beforeEach 函数有三个参数:

1、 to:router 即将进入的路由对象

2、 from:当前导航即将离开的路由

3、 next:Function,进行管道中的一个钩子，如果执行完了，则导航的状态就是 confirmed （确认的）；否则为 false，终止导航。

4、 afterEach 函数不用传 next()函数这类钩子主要作用于全局,一般用来判断权限,以及以及页面丢失时候需要执行的操作,例如:

```
//使用钩子函数对路由进行权限跳转
router.beforeEach((to, from, next) => {
    const role = localStorage.getItem('ms_username');
    if(!role && to.path !== '/login'){
        next('/login');
    }else if(to.meta.permission){
        // 如果是管理员权限则可进入，这里只是简单的模拟管理员权限而已
```

```
        role === 'admin' ? next() : next('/403');
    }else{
        // 简单的判断 IE10 及以下不进入富文本编辑器，该组件不兼容
        if(navigator.userAgent.indexOf('MSIE') > -1 && to.path ===
'/editor'){
            Vue.prototype.$alert('vue-quill-editor 组件不兼容 IE10 及以下
浏览器，
            请使用更高版本的浏览器查看', '浏览器不兼容通知', {
                confirmButtonText: '确定'
            });
        }else{
            next();
        }
    }
})
```

单个路由里面的钩子

主要用于写某个指定路由跳转时需要执行的逻辑

组件路由

主要包括 beforeRouteEnter 和 beforeRouteUpdate,beforeRouteLeave,这几个钩子都是写在组件里面也可以传三个参数(to,from,next),作用与前面类似.

```
beforeRouteEnter(to, from, next) {
    next(vm => {
      if (
        vm.$route.meta.hasOwnProperty('auth_key') &&
        vm.$route.meta.auth_key != ''
      ) {
```

```
        if (!vm.hasPermission(vm.$route.meta.auth_key)) {
          vm.$router.replace('/admin/noPermission')
        }
      }
    })
  }
```

vuex 的使用?

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，具体包括：

1、 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。

2、 getter：state 的计算属性，类似 vue 的计算属性，主要用来过滤一些数据。

3、 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。可以异步函数调用

4、 mutation：mutations 定义的方法动态修改 Vuex 的 store 中的状态或数据

5、 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。

vue 的 filter 的理解与用法?

全局过滤器必须写在 vue 实例创建之前

```
Vue.filter('testfilter', function (value,text) { // 返回处理后的值
    return value+text
    })
```

局部写法：在组件实例对象里挂载

```
filters: {
            changemsg:(val,text)\=>{ return val + text
            }
     }
```

3）使用方式：只能使用在{{}}和： v-bind 中，定义时第一个参数固定为预处理的数，后面的数为调用时传入的参数，调用时参数第一个对应定义时第二个参数，依次往后类推

```
<h3 :title="test|changemsg(1234)">{{test|changemsg(4567)}}</h3>
//多个过滤器也可以串行使用
<h2>{{name|filter1|filter2|filter3}}</h2>
```

vue-cli 项目中注册多个全局过滤器写法：

```
//1.创建一个单独的文件定义并暴露函数对象
const filter1 = function (val) {
  return val + '--1'
}
const filter2 = function (val) {
  return val + '--2'
}
```

```
const filter3 = function (val) {
  return val + '--3'
}

export default {
  filter1,
  filter2,
  filter3
}

//2.导入 main.js(在 vue 实例之前)
import filters from './filter/filter.js'

//3.循环注册过滤器
Object.keys(filters).forEach(key=>{
  Vue.filter(key,filters[key])
})
```

vue 的 keep-alive 的理解?

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染，页面第一次进入，钩子的触发顺序:created-> mounted-> activated，退出时触发 deactivated ,当再次进入（前进或者后退）时，只触发 activated 事件挂载的方法等，只执行一次的放在 mounted 中；组件每次进去执行的方法放在 activated 中；其有几个属性如下：

1、 include - 字符串或正则表达式，只有名称匹配的组件会被缓存

2、 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

3、 include 和 exclude 的属性允许组件有条件地缓存。二者都可以用“，”分隔字符串、正则表达式、数组。当使用正则或者是数组时，要记得使用 v-bind 。

```
<!-- 逗号分隔字符串，只有组件 a 与 b 被缓存。-->
<keep-alive include="a,b">
  <component></component>
</keep-alive>

<!-- 正则表达式 (需要使用 v-bind，符合匹配规则的都会被缓存) -->
<keep-alive :include="/a|b/">
  <component></component>
</keep-alive>

<!-- Array (需要使用 v-bind，被包含的都会被缓存) -->
<keep-alive :include="['a', 'b']">
  <component></component>
</keep-alive>
```

如何封装一个 vue 组件?

根据业务需求，建立组件的模板，先把架子搭起来，写写样式，考虑好组件的基本逻辑。

1、 准备好组件的数据输入。即分析好逻辑，定好 props 里面的数据、类型。

2、 准备好组件的数据输出。即根据组件逻辑，做好要暴露出来的方法。

3、 封装完毕了，直接调用即可

vue 首屏白屏如何解决?

1、 路由懒加载

2、 vue-cli 开启打包压缩 和后台配合 gzip 访问

3、 进行 cdn 加速

4、 开启 vue 服务渲染模式

5、 用 webpack 的 externals 属性把不需要打包的库文件分离出去，减少打包后文件的大小

6、 在生产环境中删除掉不必要的 console.log

```
plugins: [
    new webpack.optimize.UglifyJsPlugin({ //添加-删除 console.log
      compress: {
        warnings: false,
        drop_debugger: true,
        drop_console: true
      },
      sourceMap: true
    }),
```

开启 nginx 的 gzip ,在 nginx.conf 配置文件中配置

```
http {   //在 http 中配置如下代码,
    gzip on;
    gzip_disable "msie6";
```

```
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 8; #压缩级别
    gzip_buffers 16 8k;
    #gzip_http_version 1.1;
    gzip_min_length 100; #不压缩临界值
    gzip_types text/plain application/javascript application/x-javascript
text/css
     application/xml text/javascript application/x-httpd-php image/jpeg
image/gif image/png;
 }
```

添加 loading 效果，给用户一种进度感受

vue 中的 v-cloak 的理解?

使用 v-cloak 指令设置样式，这些样式会在 Vue 实例编译结束时，从绑定的 HTML 元素上被移除。

一般用于解决网页闪屏的问题，在对一个的标签中使用 v-cloak，然后在样式中设置[v-cloak]样式,[v-cloak]需写在 link 引入的 css 中，或者写一个内联 css 样式，写在 import 引入的 css 中不起作用。

vue 中 template 编译的理解?

就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点），具体为：

首先,通过 compile 编译器把 template 编译成 AST 语法树(abstract syntax tree 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用以创建编译器的。

另外 compile 还负责合并 option。
然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）

v-model 的理解?

v-model 用于表单数据的双向绑定，其实它就是一个语法糖，这个背后就做了两个操作：

1、 v-bind 绑定一个 value 属性；

2、 v-on 指令给当前元素绑定 input 事件

computed 和 watch 的用法和区别?

computed

1、 变量不在 data 中定义，而是定义在 computed 中，写法跟写方法一样，有返回值。函数名直接在页面模板中渲染，不加小括号 。

2、 根据传入的变量的变化 进行结果的更新。

3、 计算属性基于响应式依赖进行缓存。如其中的任意一个值未发生变化，它调用的就是上一次计算缓存的数据，因此提高了程序的性能。而 methods 中每调用一次就会重新计算一次，为了进行不必要的资源消耗，选择用计算属性。

watch

计算属性的时候 初始化的时候就可以被监听到并且计算 但是 watch 是发生改变的时候才会触发。

当有一些数据需要随着其它数据变动而变动时，或者当需要在数据变化时执行异步或开销较大的操作时，使用 watch。

总结：

1、 计算属性变量在 computed 中定义，属性监听在 data 中定义。

2、 计算属性是声明式的描述一个值依赖了其他值，依赖的值改变后重新计算结果更新 DOM。属性监听的是定义的变量，当定义的值发生变化时，执行相对应的函数。

$nextTick 的使用?

在 vue 中理解修改数据后，对应的 dom 需要一定的时间进行更新，因此为了能够准确的后去更新后的 dom，可以采用延迟回调的方法进行更新 dom 的获取，所以出现了$nextTick 来进行延迟回调。即：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。

data 为什么是一个函数?

这是有 JavaScript 的特性所导致，在 component 中，data 必须以函数的形式存在，不可以是对象。

组建中的 data 写成一个函数，数据以函数返回值的形式定义，这样每次复用组件的时候，都会返回一份新的 data，相当于每个组件实例都有自己私有的数据空间，它们只负责各自维护的数据，不会造成混乱。而单纯的写成对象形式，就是所有的组件实例共用了一个 data，这样改一个全都改了。

vue 单页面和传统的多页面区别?

单页面应用（SPA）

通俗一点说就是指只有一个主页面的应用，浏览器一开始要加载所有必须的 html, js, css。所有的页面内容都包含在这个所谓的主页面中。但在写的时候，还是会分开写（页面片段），然后在交互的时候由路由程序动态载入，单页面的页面跳转，仅刷新局部资源。多应用于 pc 端。

多页面（MPA）

指一个应用中有多个页面，页面跳转时是整页刷新

单页面的优点：

用户体验好，快，内容的改变不需要重新加载整个页面，基于这一点 spa 对服务器压力较小；前后端分离；页面效果会比较炫酷（比如切换页面内容时的专场动画）。

单页面缺点：

不利于 seo；导航不可用，如果一定要导航需要自行实现前进、后退。（由于是单页面不能用浏览器的前进后退功能，所以需要自己建立堆栈管理）；初次加载时耗时多；页面复杂度提高很多。

vue 常用的修饰符?

1、 .stop：等同于 JavaScript 中的 event.stopPropagation()，防止事件冒泡；

2、 .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）；

3、 .capture：与事件冒泡的方向相反，事件捕获由外到内；

4、 .self：只会触发自己范围内的事件，不包含子元素；

5、 .once：只会触发一次。

vue 更新数组时触发视图更新的方法?

push()；pop()；shift()；unshift()；splice()；sort()；reverse()

route 和 router 的区别?

$router

router 是 VueRouter 的一个对象，通过 Vue.use(VueRouter)和 VueRouter 构造函数得到一个 router 的实例对象，这个对象中是一个全局的对象，他包含了所有的路由包含了许多关键的对象和属性,常见的有：

1）push：向 history 栈添加一个新的记录，当我们点击浏览器的返回按钮时可以看到之前的页面

```
// 字符串
    this.$router.push('home')
// 对象
    this.$router.push({ path: 'home' })
// 命名的路由
    this.$router.push({ name: 'user', params: { userId: 123 }})
// 带查询参数，变成 /register?plan=123
    this.$router.push({ path: 'register', query: { plan: '123' }})
```

2）go：页面路由跳转 前进或者后退

```
// 页面路由跳转 前进或者后退
this.$router.go(-1) // 后退
```

3）replace：push 方法会向 history 栈添加一个新的记录，而 replace 方法是替换当前的页面，不会向 history 栈添加一个新的记录

$route

1、 $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。

2、 $route.path：字符串，对应当前路由的路径，总是解析为绝对路径，如 "/foo/bar"。

3、 $route.params：一个 key/value 对象，包含了 动态片段 和 全匹配片段，如果没有路由参数，就是一个空对象。

4、 $route.query：一个 key/value 对象，表示 URL 查询参数。例如，对于路径 /foo?user=1，则有 $route.query.user == 1，如果没有查询参数，则是个空对象。

5、 $route.hash：当前路由的 hash 值（不带#），如果没有 hash 值，则为空

字符串。

6、 $route.fullPath：完成解析后的 URL，包含查询参数和 hash 的完整路径。

7、 dollar;route.matched：数组，包含当前匹配的路径中所包含的所有片段所对应的配置参数对象。

8、 $route.name：当前路径名字

9、 $route.meta：路由元信息

vue-router 实现懒加载的方式?

vue 异步组件

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

```
/* vue 异步组件技术 */
{
  path: '/home',
  name: 'home',
  component: resolve => require(['@/components/home'],resolve)
},{
  path: '/index',
  name: 'Index',
  component: resolve => require(['@/components/index'],resolve)
},{
  path: '/about',
  name: 'about',
  component: resolve => require(['@/components/about'],resolve)
}
```

es 提案的 import()

路由懒加载(使用 import)

```
// 下面 2 行代码，没有指定 webpackChunkName，每个组件打包成一个 js 文件。
/* const Home = () => import('@/components/home')
const Index = () => import('@/components/index')
const About = () => import('@/components/about') */
// 下面 2 行代码，指定了相同的 webpackChunkName，会合并打包成一个 js 文件。把组件按组分块
const Home =  () => import(/* webpackChunkName: 'ImportFuncDemo' */ '@/components/home')
const Index = () => import(/* webpackChunkName: 'ImportFuncDemo' */ '@/components/index')
const About = () => import(/* webpackChunkName: 'ImportFuncDemo' */ '@/components/about')
{
  path: '/about',
  component: About
}, {
  path: '/index',
  component: Index
}, {
  path: '/home',
  component: Home
}
```

webpack 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。

```
/* 组件懒加载方案三: webpack 提供的 require.ensure() */
{
  path: '/home',
  name: 'home',
  component: r => require.ensure([], () =>
r(require('@/components/home')), 'demo')
}, {
  path: '/index',
  name: 'Index',
  component: r => require.ensure([], () =>
r(require('@/components/index')), 'demo')
}, {
  path: '/about',
  name: 'about',
  component: r => require.ensure([], () =>
r(require('@/components/about')), 'demo-01')
}
```

delete 和 Vue.delete 删除数组的区别?

delete 只是被删除的元素变成了 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。

路由跳转和 location.href 的区别?

使用 location.href='/url'来跳转，简单方便，但是刷新了页面；
使用路由方式跳转，无刷新页面，静态跳转；

vue 的 solt 的用法?

在子组件内使用特殊的 < slot>元素就可以为这个子组件开启一个 slot（插槽），在父组件模板里，插入在子组件标签内的所有内容将替代子组件的 < slot> 标签及它的内容。

简单说来就是：在子组件内部用 标签占位，当在父组件中使用子组件的时候，我们可以在子组件中插入内容，而这些插入的内容则会替换 标签的位置。

当然：单个 solt 的时候可以不对 solt 进行命名，如果存在多个 则一个可以不命名，其他必须命名，在调用的时候指定名称的对应替换 slot，没有指定的则直接默认无名称的 solt

on 、 off 理解?

$emit

触发当前实例上的自定义事件（并将附加参数都传给监听器回调）

$on

监听实例上自定义事件并调用回调函数，监听 emit 触发的事件

$once

监听一个自定义事件，但是只触发一次，在第一次触发之后移除监听器。

$off

用来移除自定义事件监听器。如果没有提供参数，则移除所有的事件监听器；如果只提供了事件，则移除该事件所有的监听器；如果同时提供了事件与回调，则只移除这个回调的监听器。

这四个方法的实现原理是：通过对 vue 实例挂载，然后分别使用对象存储数组对应的函数事件，其中 emit 通过循环查找存储的数组中对应的函数进行调用，once 只匹配一次就就结束，on 是将对应的函数存储到数组中，off 是删除数组中指定的元素或者所有的元素事件。具体可以参考文章：VUEemit 实现

refs、$parent 的使用?

$root

可以用来获取 vue 的根实例，比如在简单的项目中将公共数据放再 vue 根实例上(可以理解为一个全局 store),因此可以代替 vuex 实现状态管理；

$refs

在子组件上使用 ref 特性后，this.属性可以直接访问该子组件。可以代替事件 emit 和$on 的作用。

使用方式是通过 ref 特性为这个子组件赋予一个 ID 引用,再通过 this.$refs.testId 获取指定元素。

注意：refs。

$parent

$parent 属性可以用来从一个子组件访问父组件的实例，可以替代将数据以 prop 的方式传入子组件的方式；当变更父级组件的数据的时候，容易造成调试和理解难度增加；

vue 开发遇到的问题?

样式污染

在编写样式中，如果需要防止样式的污染，可以使用两种方式，一种是在组件的根元素上增加一个唯一的 class 或者 id，然后在编写组件的样式时候在根元素对应的 class 或者 id 下进行编写；另一种方式是在对应的 style 上添加 scoped 关键字，不过该关键字对引用的框架 UI 无效

router-link 在安卓上不起作用

不起作用的原因是因为转码编译的问题，可以使用 babel 来进行处理，安装 babel polypill 插件解决

初始化页面出现闪屏乱码的问题

这是因为 vue 还没有解析的情况下会容易出现花屏现象，看到类似于{{data}}的字样，可以使用两种方式来进行处理，一种为：在设置 index.html 的根元素的元素的样式为 display:none，然后在 mounted 中的$nextTick 函数中 display:block 展示；另一种方式是使用 vue 的内置指令：v-cloak,并且在 css 中设置样式

```
[v-cloak] {
    display: none;
}
```

router-link 上事件无效解决方法

使用@click.native 来进行调用原生的 js 事件。原因：router-link 会阻止 click 事件，.native 指直接监听一个原生事件。