

Vue 笔记

曹思远

2021 年 1 月 18 日

目录

1 起手式	3
1.1 Vue 自学路线图	3
1.2 项目搭建	3
1.2.1 两种方法	3
1.2.2 @vue/cli 用法	3
1.2.3 安装 Vue 的选项	4
1.3 Vue 实例	4
1.3.1 如何使用 Vue 实例	4
1.3.2 Vue 实例的作用	6
1.3.3 SEO 基本原理	6
1.4 理解两种 vue 的区别	7
1.4.1 深入理解	7
1.4.2 引用错了会怎样	7
2 构造选项	8
2.1 new Vue 里有什么	8
2.2 options 里有什么	8
2.2.1 options 里的五类属性	8
2.2.2 属性分段	9
2.2.3 入门属性	9
3 数据响应式	10
3.1 Vue 对 data 做了什么	10
3.1.1 小实验	10
3.1.2 小结	10
3.1.3 示意图	11
3.2 深入理解数据响应式	11
3.3 Vue 的 bug	11
3.3.1 Vue.set 和 this.\$set	12
3.3.2 data 中有数组怎么办?	12
3.3.3 ES6 写法	13
3.3.4 ES5 写法-原型	13
3.4 总结	13
3.5 刁钻的题	14

4	computed 和 watch	14
4.1	进阶属性	14
4.2	复习一下响应式原理	15
4.3	computed - 计算属性	16
4.4	Watch - 监听	16
5	模板, 指令与修饰符	17
5.1	模板 template 三种写法	17
5.2	模板里有哪些语法	17
5.2.1	展示内容	18
5.2.2	绑定属性	18
5.2.3	绑定事件	18
5.2.4	条件判断	19
5.2.5	循环	19
5.2.6	显示隐藏	19
5.3	总结	19
5.4	指令 Directive	20
5.4.1	指令	20
5.4.2	修饰符	20
5.4.3	.sync 修饰符	21
5.4.4	搞清楚这 4 个修饰符就行了	21
6	进阶构造属性	21
6.1	Directives	21
6.1.1	自定义指令	21
6.1.2	两种写法	21
6.1.3	directiveOptions	22
6.1.4	指令的作用	22
6.2	mixins	22
6.2.1	减少重复	22
6.3	extends	23
6.3.1	减少重复	23
6.4	provide 和 inject	23
6.4.1	使用举例	23
6.5	总结	24
7	表单与 v-model	24
8	Vue Router-前端路由实现思路	24
8.1	路由基本概念	24
8.2	hash 模式	25
8.3	history 模式	25
8.4	memory 模式	25
8.5	三种模式 router 总结	25
9	深入理解 Vue 动画原理	25

起手式

1.1 Vue 自学路线图

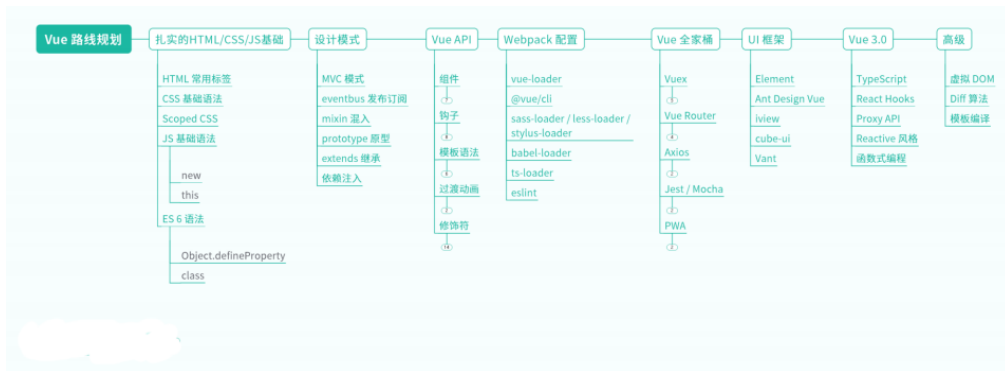


图 1: Vue 自学路线图

1.2 项目搭建

搞出一个使用 **Vue** 的项目

1.2.1 两种方法

1. 方法一: 使用 @vue/cli

- 搜索 @vue/cli, 进入官网
- 打开文档, 打开创建一个项目章节

2. 方法二: 自己从零搭建 Vue 项目

- 使用 webpack 或者 rollup 从零开始
- 适合老手

1.2.2 @vue/cli 用法

1. 全局安装: `yarn global add @vue/cli`
2. 创建目录: `vue create 路径` (路径可以用. 点)
3. 选择使用哪些配置
4. 进入目录, 运行 `yarn serve` 开启 webpack-dev-server
5. 用 WebStorm 或 VSCode 打开项目开始干
6. 进入 @vue/cli 官网看目录

1.2.3 安装 Vue 的选项

1. 选错了请 Ctrl+C 中断然后重来
2. 没有截图的都使用默认选项
3. 真实项目自行斟酌

```
? Please pick a preset:  
  default (babel, eslint)  
> Manually select features
```

```
? Check the features needed for your project:  
(*) Babel  
( ) TypeScript  
( ) Progressive Web App (PWA) Support  
( ) Router  
( ) Vuex  
(*) CSS Pre-processors  
(*) Linter / Formatter  
(*) Unit Testing  
>( ) E2E Testing
```

```
? Pick additional lint features:  
( ) Lint on save  
>(*) Lint and fix on commit
```

```
? Pick a unit testing solution:  
  Mocha + Chai  
> Jest
```

```
? Save this as a preset for future projects? (y/N) n
```

```
$ cd hello-world  
$ yarn serve
```

4. [vue demo 地址](#)

1.3 Vue 实例

很重要，做项目必须用到

1.3.1 如何使用 Vue 实例

1. 方法一：从 HTML 得到视图
 - 也就是文档里说的完整版 Vue
 - 从 CDN 引入 vue.js 或 vue.min.js

- 也可以通过 import 引入 vue.js 或者 vue.min.js
- 完整版视图支持从 html 引入
- 也可以用 template 写在 js 里面
- 完整版不好的地方是给用户的体积变大了

2. 方法二: 用 JS 构建视图

- 使用 vue.runtime.js, 也就是非完整版
- 不支持从 html 里获得视图, template 也不行
- 这种方法不方便, 但实际是对的
- 必须要用 render(createElement) 的方式把所有元素构造出来

```
new Vue({
  el: "#app",
  render(createElement) {
    const h = createElement;
    return h('div', [this.n, h('button', {
      on: {click: this.add}, '+1'
    })])
  },
  data: {
    n: 0
  },
  methods: {
    add() {
      this.n += 1
    }
  }
})
```

- 好处是更加的独立, 不需要编译器, 减少百分之 30 的体积

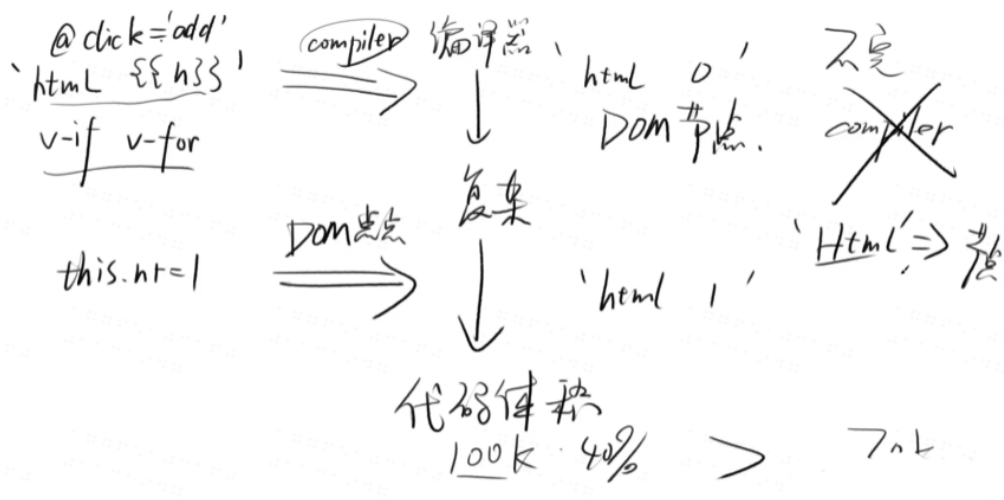


图 2: 完整版与不完整版的区别

3. 方法三: 使用 vue-loader

- 可以把.vue 文件以及其 template 里的东西翻译成 h 构建方法
- 但这样做 HTML 就只有一个 div#app, SEO 不友好
- 通过 webpack 让用户写的时候是完整版, 但实际下载的时候是非完整版

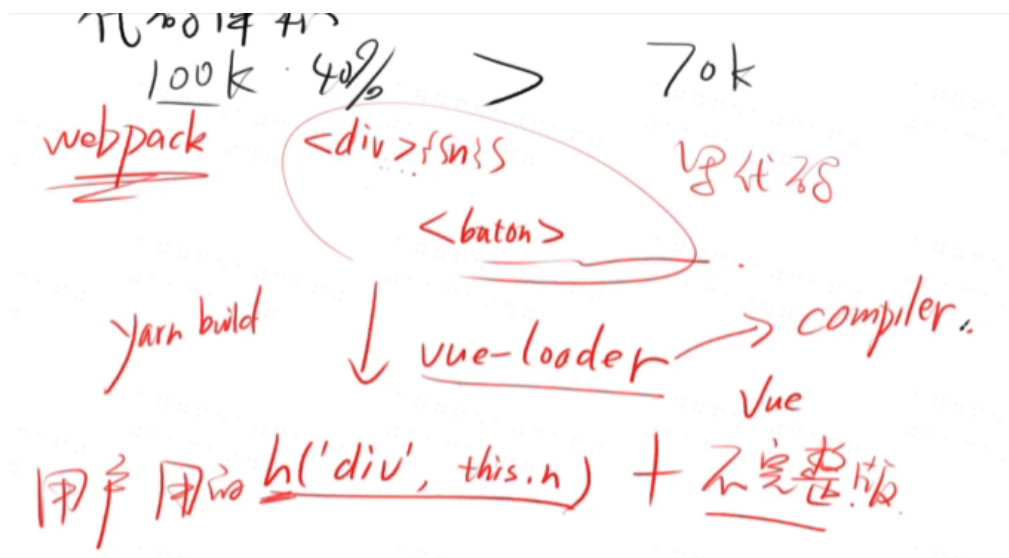


图 3: 用 webpack 通过 vue-loader 转化减少代码体积

1.3.2 Vue 实例的作用

1. Vue 实例就如同 jQuery 实例

- 封装了对 DOM 的所有操作
- 封装了对 data 的所有操作

2. 操作 DOM

- 无非就是监听事件，改变 DOM

3. 操作 data

- 无非就是增删改查
- Vue2 还有一个 bug(面试考，在后面响应式原理里)

4. 没有封装 ajax

- 用 axios 的 ajax 功能

1.3.3 SEO 基本原理

1. seo 友好

- 搜索引擎优化
- 你可以认为搜索引擎就是不停地 curl
- 搜索引擎根据 curl 结果猜测页面内容
- 如果你的页面都是用 JS 创建的 div，那么 curl 就很瞎

2. 那怎么办

- 给 curl 一点内容
- 把 title, description, keyword, h1, a 写好即可
- 原则：让 curl 能得到页面的信息，seo 就能正常工作

1.4 理解两种 vue 的区别

1.4.1 深入理解

深入理解两种区别			
	Vue 完整版	Vue 非完整版	评价
特点	有 compiler	没有 compiler	compiler 占 40% 体积
视图	写在 HTML 里 或者写在 template 选项	写在 render 函数里 用 h 来创建标签	h 是尤雨溪写好传给 render 的
cdn 引入	vue.js	vue.runtime.js	文件名不同，生成环境后缀 为 .min.js
webpack 引入	需要配置 alias	默认使用此版	尤雨溪配置的
@vue/cli 引入	需要额外配置	默认使用此版	尤雨溪、蒋豪群配置的

最佳实践：总是使用非完整版，然后配合 vue-loader 和 vue 文件思路：

1. 保证用户体验，用户下载的 JS 文件体积更小，但只支持 h 函数
2. 保证开发体验，开发者可直接在 vue 文件里写 HTML 标签，而不写 h 函数
3. 脏话让 loader 做，vue-loader 把 vue 文件里的 HTML 转为 h 函数

真 TM 聪明，这就是工程师干的事

图 4: 两种 vue 版本的区别

1.4.2 引用错了会怎样

1. vue.js 错用成了 vue.runtime.js

- 无法将 HTML 编译成视图

2. vue.runtime.js 错用成 vue.js

- 代码体积变大，因为 vue.js 有编译 HTML 的功能

2 构造选项

2.1 new Vue 里有什么

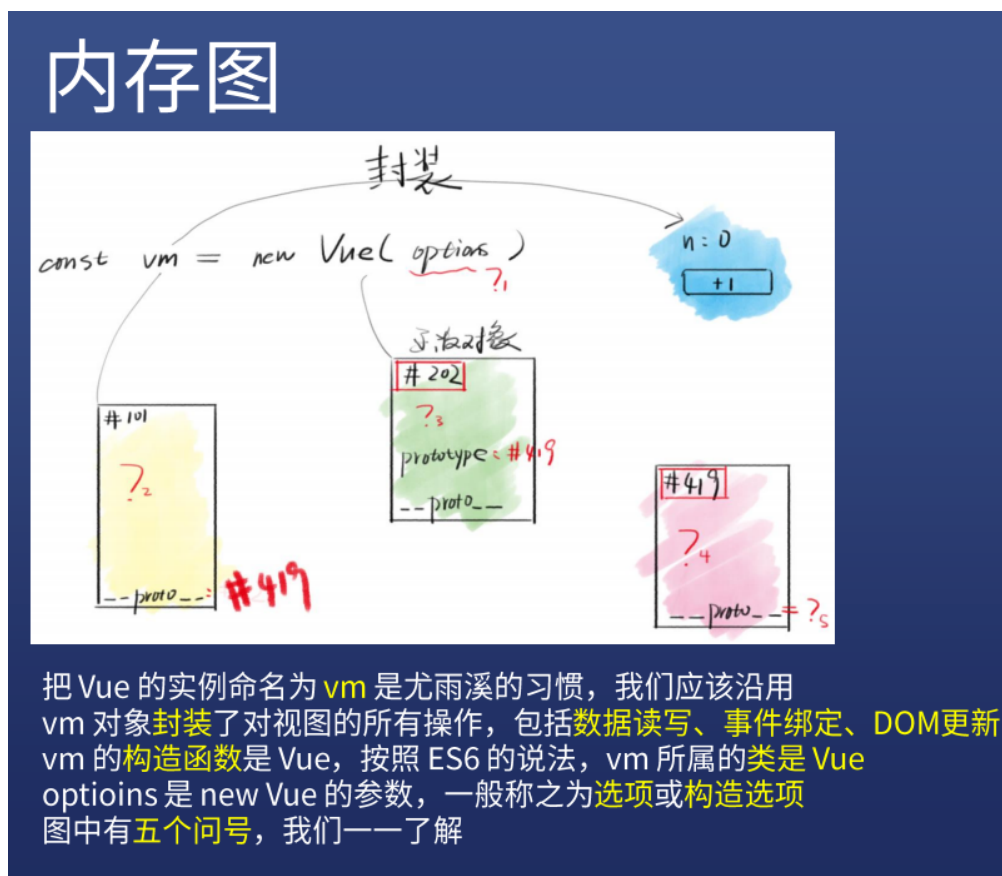


图 5: 实例的内存图

2.2 options 里有什么

2.2.1 options 里的五类属性

1. 数据

- **data**, **props**, **propsData**, **computed**, **methods**, **watch**

2. DOM

- **el**, **template**, **render**, **renderError**

3. 生命周期钩子

- **beforeCreate**, **created**, **beforeMount**, **mounted**, **beforeUpdate**, **updated**
- **activated**, **deactivated**, **beforeDestroy**, **destroyed**, **errorCaptured**

4. 资源

- **directives**, **filters**, **components**

5. 组合

- **parent**, **mixins**, **extends**, **provide**, **inject**

2.2.2 属性分段

1. 红色属性 (9)

- 好学，必学，几句话就能明白

2. 黄色属性 (9)

- 高级属性，稍微费点力

3. 绿色属性 (5)

- 简单

4. 蓝色属性 (2)

- 不常用，可学可不学

5. 紫色属性 (2)

- 比较特殊，重点

6. 黑色属性 (3)

- 很不常用，用的时候看一下文档

2.2.3 入门属性

重点！[演示代码地址](#)

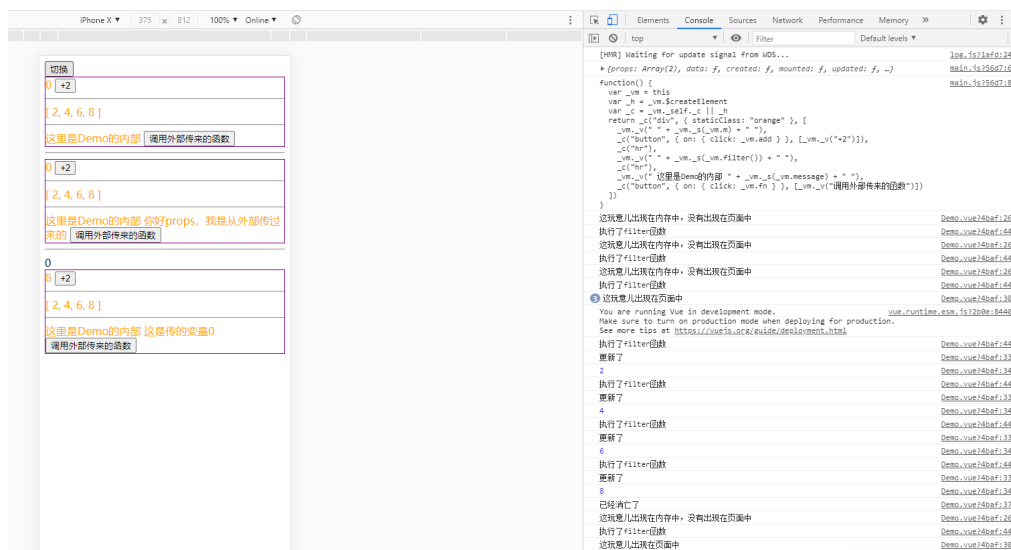


图 6: 代码演示结果

1. el - 挂载点

- 可以用 \$mount 代替

2. data - 内部数据

- 支持对象和函数，优先用函数，如果不这样，组件间就会共用 data

3. methods - 方法

- 事件处理函数或者是普通函数

- 每次执行都会调用，可能是毫无意义的，这个时候就需要 computed

4. components

- Vue 组件，注意大小写
- 三种引入方式，优先使用模块化

5. 四个钩子

- created - 实例出现在内存中
- mounted - 实例出现在页面中
- updated - 实例更新了
- destroyed - 实例从页面和内存中消亡了 (具体看代码演示)

6. props - 外部数据

- 也叫属性
- message='n' 传入字符串
- :message='n' 传入 this.n 数据
- :fn='add' 传入 this.add 函数

3 数据响应式

3.1 Vue 对 data 做了什么

3.1.1 小实验

1. data 变了

- [示例代码](#)
- myData 居然变了
- 一开始是 {n:0}, 传给 new Vue 之后立马变成 {n:(...)}
- {n:(...)} 是个什么玩意，为什么表现和 {n:0} 一致？
- 需要先学习一下 es6 的 [getter 和 setter](#)
- [示例代码](#)
- 理解 n:(...) 还需要学习一下 [Object.defineProperty](#)
- [示例代码](#)
- 代理模式继续理解
- 注意代码思路，研究方法比知识本身更重要，不读源码也能了解真相

3.1.2 小结

1. Object.defineProperty

- 可以给对象添加属性 value
- 可以给对象添加 getter/setter
- getter/setter 用于对属性的读写进行监控

2. 啥是代理 (设计模式)

- 对 myData 对象的属性读写，全权由另一个对象 vm 负责
- 那么 vm 就是 myData 的代理 (类比房东租房)
- 比如 myData.n 不用，便要用 vm.n 来操作 myData.n

3. vm=new Vue({data:myData})

- 一，会让 vm 成为 myData 的代理 (proxy)
- 二，会对 myData 的所有属性进行监控
- 为什么要监控，为了防止 myData 的属性变了，vm 不知道
- vm 知道了又如何？知道属性变了就可以调用 render(data) 呀！
- UI=render(data)

3.1.3 示意图

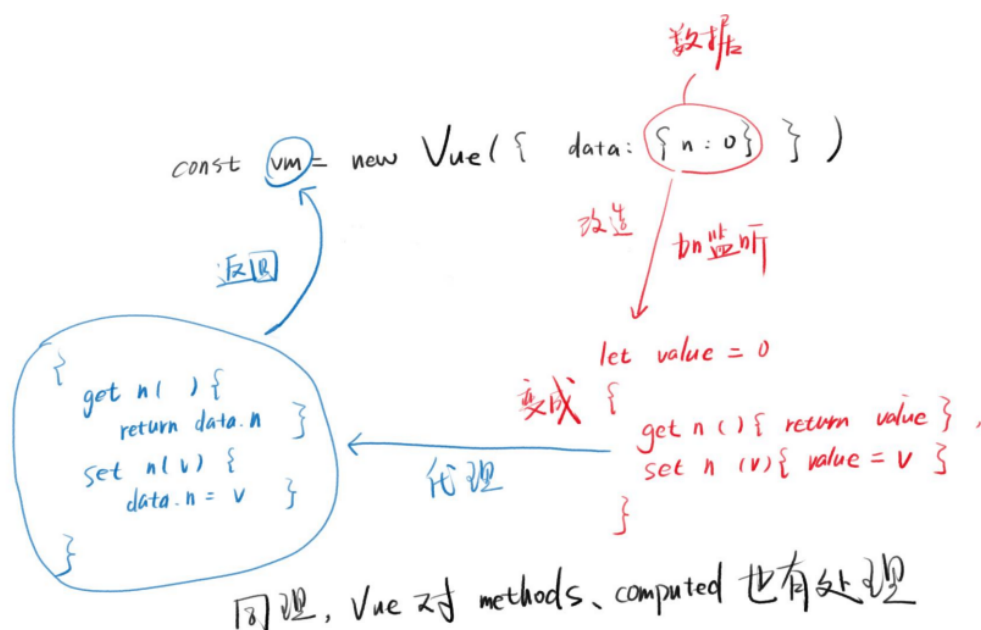


图 7: 如果 data 有多个属性 n,m,k, 那么就会有 get n/get m/get k 等

3.2 深入理解数据响应式

1. 什么是响应式

- 我打你一拳，你会喊疼，那你就是响应式的
- 若一个物体能对外界的刺激做出反应，它就是响应式的

2. Vue 的 data 是响应式

- const vm = new Vue(data:n:0)
- 我如果修改 vm.n, 那么 UI 中的 n 就会响应我
- Vue2 通过 Object.defineProperty 来实现数据响应式

3.3 Vue 的 bug

1. Object.defineProperty 的问题

- Object.defineProperty(obj, 'n', {...})

- 必须要有一个'n'，才能监听 & 代理 obj.n
- 如果前端开发者比较水，没有给 n 怎么办
- [示例一](#)
- 如果 data 为空，不会显示 undefined 或 null，只会给出一个警告
- [示例二](#)
- Vue 只会检查第一层属性
- 此时如果我点击 set b，请问视图中会显示 1 吗？
- 答案是：不会
- 为什么：因为 Vue 没办法监听一开始不存在的 obj.b

2. 解决办法

- 一，那我把 key 都声明好，后面不再加属性不就行了
- 使用 Vue.set 或者 this.\$set

3.3.1 Vue.set 和 this.\$set

1. 作用

- 新增 key
- 自动创建代理和监听 (如果没有创建过)
- 触发 UI 更新 (但并不会立即更新)

2. 举例

- this.\$set(this.object, 'm', 100)

3.3.2 data 中有数组怎么办？

1. 你没法提前声明所有 key

- [示例代码](#)
- 数组的长度可以一直增加，下标就是 key
- 你看，你没有办法提前把数组的 key 都声明出来
- Vue 也不能检测对你新增了下标
- 难道每次改数组都要用 Vue.set 或者 this.\$set

2. 尤雨溪的做法

- 篡改数组的 API，见 vue 文档中变异方法章节
- 这 7 个 API 都会被 Vue 篡改，调用后会更新 UI
- 如何篡改见下面代码

3.3.3 ES6 写法

注：这不代表 Vue 的真实实现，但基本思路是这样，程序员不需要看源码也能实现出来的

```
class VueArray extends Array {
  push(...args) {
    const oldLength = this.length //this 就是当前数组
    super.push(...args)
    console.log('你push了')
    for(let i = oldLength; i < this.length; i++) {
      Vue.set(this, i, this[i])
      //将每个新增的key都告诉Vue
    }
  }
}
```

3.3.4 ES5 写法-原型

这个需要前端基础笔记中原型链的知识

```
const vueArrayPrototype = {
  push: function() {
    console.log('你push了')
    return Array.prototype.push.apply(this, arguments)
  }
}
vueArrayPrototype.__proto__ = Array.prototype
//上面这句话用的不是标准属性，仅学习使用

const array = Object.create(vueArrayPrototype)
array.push(1)
```

3.4 总结

1. 对象中新增的 key

- Vue 没有办法事先监听和代理
- 要使用 set 来新增 key，创建监听和代理，更新 UI
- 最好提前把属性都写出来，不要新增 key
- 但数组做不到不新增 key

2. 数组中新增的 key

- 也可用 set 来新增 key，更新 UI
- 不过尤雨溪篡改了 7 个 API 方便你对数组进行增删
- 这 7 个 API 会自动处理监听和代理，并更新 UI
- 结论：数组新增 key 最好通过 7 个 API

3.5 刁钻的题

有如下代码

html

```
<div id="app">
  <span class=span-a>
    {{obj.a}}
  </span>
  <span class=span-b>
    {{obj.b}}
  </span>
</div>
```

js

```
var app = new Vue({
  el: '#app',
  data: {
    obj: {
      a: 'a',
    }
  },
})
app.obj.a = 'a2'
app.obj.b = 'b'
```

请问最终 span-a 和 span-b 中分别展示什么字符串?

- ☐ span-a 中显示a2, span-b 中不显示
- ☐ span-a 中显示a2, span-b 中显示b

图 8: 刁钻的题

[刁钻的题目代码地址](#)

1. 分析

- span-b 之所以会显示 b，是因为视图在显示 span-a 的 a2 时，顺便更新了 span-b

2. 要理解为什么 span-b 会更新，要点是理解视图更新其实是异步的。

- 当我们让 a 从 'a1' 变成 'a2' 时，Vue 会监听到这个变化，
- 但是 Vue 并不能马上更新视图，因为 Vue 是使用 Object.defineProperty 这样的方式来监听变化的，
- 监听到变化后会创建一个视图更新任务到任务队列里。（文档有写）
- 所以在视图更新之前，要先把余下的代码运行完才行，也就是会运行 b = 'b'
- 等到视图更新的时候，由于 Vue 会去做 diff（文档有写）
- 于是 Vue 就会发现 a 和 b 都变了，自然会去更新 span-a 和 span-b

4 computed 和 watch

4.1 进阶属性

1. computed - 计算属性

- 不需要加括号
- 它会根据依赖是否变化来缓存

2. watch - 监听

- 一旦 data 变化，就执行的函数

- options.watch 用法
- this.\$watch 用法
- deep, immediate 含义

3. directives - 指令

- 内置指令 v-if/v-for/v-bind/v-on
- 自定义指令, 如 v-focus
- 指令是为了减少重复的 DOM 操作

4. mixin - 混入

- 重复三次之后的出路
- 混入 v.s. 全局混入
- 选项自动合并
- 混入就是为了减少重复的构造选项

5. extends - 继承

- 先了解一下 Vue.extend
- 你觉得用了 mixin 还是重复
- 于是你自己写了一个 View, 它继承 Vue
- 你还可以预先定义其他构造选项
- 那为什么不用 ES6 的 extends 呢?

6. provide/inject

- 爷爷想和孙子讲话怎么办
- 祖宗想跟它的所有后代讲话怎么办
- 答案是全局变量, 但是全局变量太 low
- 所以我们需要局部的全局变量

4.2 复习一下响应式原理

1. options.data

- 会被 Vue 监听
- 会被 Vue 实例代理
- 每次对 data 的读写都会被 Vue 监控
- Vue 会在 data 变化时更新 UI

2. 本节内容

- data 变化时除了更新 UI, 还能做点啥?

4.3 computed - 计算属性

1. 用途

- 被计算出来的属性就是[计算属性](#)
- 例 1: [用户名展示](#)
- 例 2: [列表展示不用 computed](#)
- 例 3: [列表展示用 computed](#)

2. 缓存

- 如果依赖的属性没有变化，就不会重新计算
- getter/setter 默认不会做缓存，Vue 做了特殊处理
- 如何缓存？看[缓存示例](#)(仅供参考学习，不代表 Vue 真实实现)

4.4 Watch - 监听

1. 用途

- 当数据变化时，执行一个函数
- 例 1: [异步撤销](#)
- 例 2: [模拟 computed](#) 说实话，这样做很傻

2. 何谓变化？

- 看[示例](#)
- obj 原本是 {a:'a'}, 现在 obj={a:'a'}, 请问
- obj 变了没有？变了。obj.a 变了没有？，没变。
- 简单类型看值，复杂类型 (对象) 看地址
- 这其实就是 === 的规则

3. 语法

- 语法 1

```
watch: {
  //o1: ()=>{} //别用这种，这里的this是全局对象
  o2: function(value, oldValue){},
  o3(){},
  o4: [f1, f2],
  o5: 'methodName',
  o6: {handler:fn, deep:true, immediate:true},
  'object.a': function() {}
}
```

- 语法 2

```
vm.$watch('xxx', fn, {deep: ..., immediate: ...})
//其中'xxx'可以改为一个返回字符串的函数
```

4. deep:true 是干什么的

- 如果 object.a 变了，请问 object 算不算也变了
- 如果你选哟答案式也变了，那么就用 deep:true
- 如果你需要答案是没有变，那么就用 deep:false
- deep 的意思是，监听 object 的时候是否往深了看

5 模板，指令与修饰符

5.1 模板 template 三种写法

1. 一，Vue 完整版吗，写在 HTML 里

```
<div id=xxx>
  {{n}}
  <button @click="add">+1</button>
</div>

new Vue({
  el: '#xxx',
  data:{n:0}, //data可以改成函数
  method:{add(){} }
})
```

2. 二，Vue 完整版，写在选项里

```
<div id=app>
</div>

new Vue({
  template: `
    <div>
      {{n}}
      <button @click="add">+1</button>
    </div>`,
  data:{n:0}, //data可以改成函数
  methods:{add(){ this.n += 1 }}
}).$mount('#app')
//注意一个细节: div#app会被替代
```

3. 三，非 Vue 完整版，配合 xxx.vue 文件

```
<template>
  <div>
    {{n}}
    <button @click="add">
      +1
    </button>
  </div>
</template>
<script>
  export default {
    data() { return {n:0} },
    //data必须为函数
    methods:{add(){ this.n += 1 }}
  }
</script>
<style>这里写CSS</style>
//然后在另一个地方写如下代码
import xxx from './xxx.vue'
//xxx 是一个options对象
new Vue({
  render: h => h(xxx)
}).$mount('#app')
```

5.2 模板里有哪些语法

我们把 HTML 模板叫做 template

5.2.1 展示内容

1. 表达式

- `{{ object.a }}` 表达式
- `{{ n + 1 }}` 可以写任何运算
- `{{ fn(n) }}` 可以调用函数
- 如果值为 `undefined` 或 `null` 就不显示
- 另一种写法为 `<div v-text=“表达式”></div>` (这种写法没有意义，大家都用上面的)

2. HTML 内容

- 假设 `data.x` 值为 `hi`
- `<div v-html=“x”></div>` 即可显示粗体 hi

3. 我就想展示 n

- `<div v-pre>{{ n }}</div>`
- `v-pre` 不会对模板进行编译

5.2.2 绑定属性

1. 绑定 src

- ``

2. v-bind: 简写为:

- ``

3. 绑定对象

- `<div :style=“border: ‘1px solid red’, height:100”` 注意这里可以把 ‘100px’ 写成 100`</div>`

5.2.3 绑定事件

1. v-on: 事件名

- `<button v-on:click=“add”>+1</button>` 点击之后，Vue 会运行 `add()`
- `<button v-on:click=“xxx(1)”>xxx</button>` 点击之后，Vue 会运行 `xxx(1)`
- `<button v-on:click=“n+=1”>+1</button>` 点击之后，Vue 会运行 `n+=1`
- 点击之后，Vue 会运行 `n+=1`，即发现函数就加括号调用之，否则直接运行代码
- 这导致一个问题，如果 `xxx(1)` 返回一个函数咋办
- 答：用 Vue 的人怎么可能想出这么复杂的用法

2. 缩写

- `<button @click=“n+=1”>+1</button>`
- 正常人都用缩写

5.2.4 条件判断

if...else

```
<div v-if="x > 0">
  x 大于 0
</div>
<div v-else-if="x===0">
  x 为 0
</div>
<div v-else>
  x 小于 0
</div>
```

5.2.5 循环

for(value, key) in 对象或数组

```
<ul>
  <li v-for="(u, index) in users" :key="index">
    索引: {{index}} 值: {{u.name}}
  </li>
</ul>
<ul>
  <li v-for="(value, name) in obj" :key="name">
    属性名: {{name}} 属性值: {{value}}
  </li>
</ul>
```

坑预警: :key="index" 有 bug, 后面讲

5.2.6 显示隐藏

1. v-show

```
<div v-show="n%2===0"> n 是偶数 </div>
```

高手一般不用, 用 css 切换

2. 近似等于

```
<div :style="{display:n%2===0?'block':'none'}"> n 是偶数 </div>
```

- 但你要注意, 看得见的元素 display 不只有 block
- 如 table 的 display 为 table
- 如 li 的 display 为 list-item

5.3 总结

```
1 <input name="username"> - HTML
2 <input name="username" /> - HTML
3 <div></div> - HTML
4
5 <input name="username" /> - XML
6 <div />
7
8 div.style.border = '1px solid red'
9 div.style.background = 'red'
10 div.style.height = 100 // '100px' '100em'
11
12
13 $div.on('click', fn)
14
15 命令式编程: div.innerHTML = x
16 声明式编程
```

图 9: 总结

Vue 模板主要特点有

- 使用 XML 语法 (不是 HTML)
- 使用 `{{}}` 插入表达式
- 使用 `v-html` `v-on` `v-bind` 等指令操作 DOM
- 使用 `v-if` `v-for` 等指令实现条件判断和循环
- `v-if` 是什么时候出现在 dom 树里, `v-show` 是什么时候展示出来 (css 隐藏的)

5.4 指令 Directive

5.4.1 指令

1. 什么是指令

```
<div v-text="x"></div>
<div v-html="x"></div>
```

以 `v-` 开头的东西就是指令

2. 语法

- `v-指令名:参数=值`, 如 `v-on:click=add`
- 如果 `值` 里没有特殊字符, 则可以不加引号
- 有些指令没有参数和值, 如 `v-pre`
- 有些指令没有值, 如 `v-on:click.prevent`

5.4.2 修饰符

1. 有些指令支持修饰符

- `@click.stop="add"` 表示阻止事件传播/冒泡
- `@click.prevent="add"` 表示阻止默认动作 [代码示例](#)
- `@click.stop.prevent="add"` 同时表示两种意思

2. 一共有多少修饰符呢?

- `v-on` 支持的有 `.keycode|keyAlias` [代码示例](#)
- `.stop.prevent.capture.self.once.passive.native`
- 快捷键相关 `.ctrl.alt.shift.meta.exact`
- 鼠标相关 `.left.right.middle`
- `v-bind` 支持的有 `.prop.camel.sync`
- `v-model` 支持的有 `.lazy.number.trim`

5.4.3 .sync 修饰符

1. 场景描述

- 爸爸给儿子钱，儿子要花钱怎么办 [代码示例](#)
- 答：儿子打电话（触发事件）向爸爸要钱
- Vue 规则：组件不能修改 props 外部数据
- Vue 规则：this.\$emit 可以触发事件，并传参
- Vue 规则：\$event 可以获取 \$emit 的参数

2. 由于这种场景很场景

- 所以尤雨溪发明了.sync，示例
- :money.sync="total" 等价于
- :money="total" v-on:update:money="total=\$event"

5.4.4 搞清楚这 4 个修饰符就行了

1. @click.stop="xxx"
2. @click.prevent="xxx"
3. @keypress.enter="xxx"
4. :money.sync="total"

6 进阶构造属性

directive, mixins, extends, provide, inject

6.1 Directives

指令

6.1.1 自定义指令

1. 我们已经学了一些内置指令
 - v-if v-for v-show v-html 等
2. 今天学习如何自己造一个指令
 - 目标：造出 v-x，点击即出现一个 x

6.1.2 两种写法

1. 一，声明一个全局指令
 - Vue.directive('x', directiveOptions) 你就可以在任何组件里用 v-x 了
 - [代码示例](#)
2. 二，声明一个局部指令

```

new Vue({
  ...,
  directives: {
    "x": directiveOptions
  }
})
//v-x只能用于该实例中

```

6.1.3 directiveOptions

1. 五个函数属性

- bind(el, info, vnode, oldVnode) - 类似 created
- inserted(参数同上) - 类似 mounted
- update(参数同上) - 类似 updated
- componentUpdated(参数同上) - 用的不多
- unbind(参数同上) - 类似 destroyed

2. 自制 v-on2 指令，模仿 v-on

- [代码地址](#)

6.1.4 指令的作用

1. 主要用于 DOM 操作

- Vue 实例/组件用于数据绑定，事件监听，DOM 更新
- Vue 指令主要目的就是原生 DOM 操作

2. 减少重复

- 如果某个 DOM 操作你经常使用，就可以封装为指令
- 如果某个 DOM 操作比较复杂，也可以封装为指令

6.2 mixins

混入，混入就是 TM 复制，前端就喜欢把简单概念搞复杂

6.2.1 减少重复

1. 类比

- directives 的作用是减少 DOM 操作的重复
- mixins 的作用是减少 data, methods, 钩子的重复

2. 场景描述

- 假设我们需要在每个组件上添加 name 和 time
- 在 created, beforeDestroy 时，打出提示，并报出存活时间
- 一共由五个组件，请问你怎么做？
- 一，给每个组件添加 data 和钩子，共 5 次
- 二，或者使用 mixins 减少重复

- [已完成的示例地址](#)

3. mixins 技巧

- 选项智能合并
- Vue.mixin 全局使用 (不推荐)

6.3 extends

继承, extends 是比 mixins 更抽象一点的封装 如果你嫌写五次 mixins 麻烦, 可以考虑 extends 一次, 不过实际工作中用得很少

6.3.1 减少重复

1. 与 mixins 同样的需求

- 不想要每次都写一个 mixins, 咋办?
- 你可以使用 Vue.extend 或 options.extends

2. 代码演示

```
const MyVue = Vue.extend({
  data() { return {name: '', time: undefined} },
  created() {
    if (!this.name) { console.error('no name!') }
    this.time = new Date()
  },
  beforeDestroy() {
    const duration = (new Date()) - this.time
    console.log(`${this.name} ${duration}`)
  }
})
// 然后我们就可以使用 new MyVue(options) 了
```

3. [完整代码地址](#)

6.4 provide 和 inject

提供和注入

6.4.1 使用举例

1. 需求

- 一键换肤功能: 默认蓝色, 可以切换为红色
- 文字大小: 默认正常, 可以改成大或小
- [已完成的示例](#)
- 祖先栽树 (provide), 后人乘凉 (inject)

2. 总结

- 作用: 大范围的 data 和 method 等共用
- 注意: 不能只传 themeName 不传 changeTheme, 因为 themeName 的值是被复制给 provide 的
- 思考: 传引用可以吗? 可以, 但是不推荐, 因为容易失控

6.5 总结

1. directives 指令

- 全局用 `Vue.directive('x',...)`
- 局部用 `options.directives`
- 作用是减少 DOM 操作相关重复代码

2. mixins 混入

- 全局用 `Vue.mixin(...)`
- 局部用 `options.mixins:[mixin1, mixin2]`
- 作用是减少 options 里的重复

3. extends 继承/扩展

- 全局用 `Vue.extend(...)`
- 局部用 `options.extends:...`
- 作用跟 mixins 差不多，只是形式不同

4. provide/inject 提供和注入

- 祖先提供东西，后代注入东西
- 作用是大范围，隔 N 代共享信息

7 表单与 v-model

1. 基本用法

- `input/textarea/checkbox/radio/select/form`

2. 修饰符

- `.lazy/.number/.trim`

3. v-model

- 面试官问你 Vue 的双向绑定，其实就是在问 v-model
- 但是实际上 v-model 是 `v-bind:value` 和 `v-on:input` 的语法糖
- `v-on:input="???"` 这里需要自己背下来

4. [v-model 代码演示地址](#)

5. [自制 input 代码地址](#)

6. v-model 是从 Angular1 的 ngModel 借鉴来的

8 Vue Router-前端路由实现思路

8.1 路由基本概念

路由，分发，路由表，默认路由，404 路由/保底路由，嵌套路由

8.2 hash 模式

[hash 模式代码地址](#)

8.3 history 模式

[history 模式代码地址](#)

8.4 memory 模式

[memory 模式代码地址](#)

8.5 三种模式 router 总结

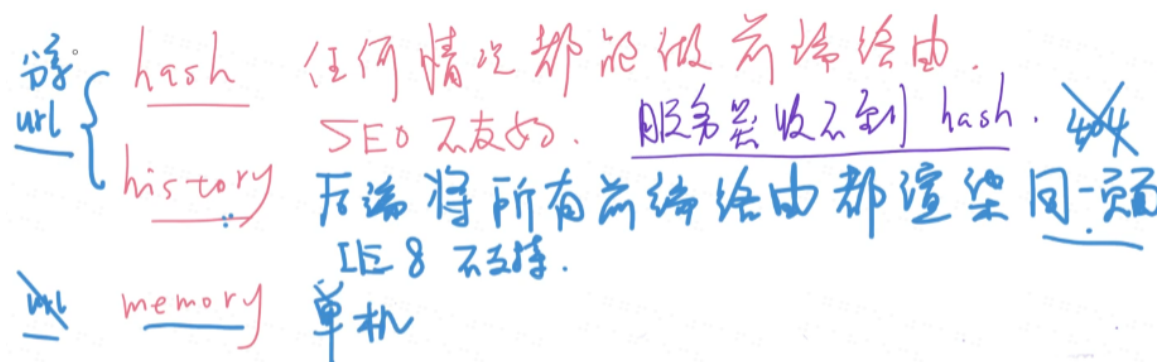


图 10: 三种模式 router 总结

9 深入理解 Vue 动画原理