# 前端开发必会面试题和多种实战技巧

**一、常见面试题**

### **1、call()和apply()的区别**

实际上，apply和call的功能是一样的，只是传入的参数列表形式不同。apply传的是数组；

### **2、 什么是事件代理/事件委托？**

事件代理/事件委托是利用事件冒泡的特性，将本应该绑定在多个元素上的事件绑定在他们的祖先元素上，尤其在动态添加子元素的时候，可以非常方便的提高程序性能，减小内存空间。

### **3、什么是事件冒泡？什么是事件捕获？**

冒泡型事件：事件按照从最特定的事件目标到最不特定的事件目标(document对象)的顺序触发。

捕获型事件：事件从最不精确的对象(document 对象)开始触发，然后到最精确(也可以在窗口级别捕获事件，不过必须由开发人员特别指定)。

支持W3C标准的浏览器在添加事件时用addEventListener(event,fn,useCapture)方法，基中第3个参数useCapture是一个Boolean值，用来设置事件是在事件捕获时执行，还是事件冒泡时执行。而不兼容W3C的浏览器(IE)用attachEvent()方法，此方法没有相关设置，不过IE的事件模型默认是在事件冒泡时执行的，也就是在useCapture等于false的时候执行，所以把在处理事件时把useCapture设置为false是比较安全，也实现兼容浏览器的效果。

### **4、什么是"use strict";?使用它的好处和坏处分别是什么？**

在代码中出现表达式-"use strict"; 意味着代码按照严格模式解析，这种模式使得Javascript在更严格的条件下运行。

*****好处：*****

* 消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;
* 消除代码运行的一些不安全之处，保证代码运行的安全；
* 提高编译器效率，增加运行速度；
* 为未来新版本的Javascript做好铺垫。

*****坏处：*****

* 同样的代码，在"严格模式"中，可能会有不一样的运行结果；
* 一些在"正常模式"下可以运行的语句，在"严格模式"下将不能运行。

### **5、请解释一下JavaScript的同源策略**

同源策略，即拥有相同的协议（protocol），端口（如果指定），主机（域名）的两个页面是属于同一个源。 然而在IE中比较特殊，IE中没有将端口号加入同源的条件中，因此上图中端口不同那一项，在IE中是算同源的。 <script> <img> <iframe>中的src，href都可以任意链接网络资源，是不遵循通源策略的。

### **6、请解释JSONP的工作原理**

JSONP (JSON with Padding)是一个简单高效的跨域方式，HTML中的script标签可以加载并执行其他域的javascript，于是我们可以通过script标记来动态加载其他域的资源。例如我要从域A的页面pageA加载域B的数据，那么在域B的页面pageB中我以JavaScript的形式声明pageA需要的数据，然后在 pageA中用script标签把pageB加载进来，那么pageB中的脚本就会得以执行。JSONP在此基础上加入了回调函数，pageB加载完之后会执行pageA中定义的函数，所需要的数据会以参数的形式传递给该函数。JSONP易于实现，但是也会存在一些安全隐患，如果第三方的脚本随意地执行，那么它就可以篡改页面内容，截获敏感数据。但是在受信任的双方传递数据，JSONP是非常合适的选择。

AJAX是不跨域的，而JSONP是一个是跨域的，还有就是二者接收参数形式不一样！

### **7、什么是跨域？**

跨域需要针对浏览器的同源策略来理解，同源策略指的是请求必须是同一个端口，同一个协议，同一个域名，不同源的客户端脚本在没有明确授权的情况下，不能读写对方资源。

受浏览器同源策略的影响，不是同源的脚本不能操作其他源下面的对象。想要操作另一个源下的对象是就需要跨域。

### **8、如何防抖？**

****何为防抖**** 多次触发事件后，事件处理函数只执行一次，并且是在触发操作结束时执行，一般用于scroll事件。

****解决原理**** 对处理函数进行延时操作，若设定的延时到来之前再次触发事件，则清除上一次的延时操作定时器，重新定时。

let timer;

window.onscroll  = function () {

    if(timer){

        clearTimeout(timer)

    }

    timer = setTimeout(function () {

        //滚动条位置

        let scrollTop = document.body.scrollTop || document.documentElement.scrollTop;

        console.log('滚动条位置：' + scrollTop);

        timer = undefined;

    },200)

}

## **9、**

**var arr = [1,2,3];**

**var arr2 = arr;**

**arr.push(4);**

**arr2.push(5);**

**请问arr 和arr2的结果是？**

**答案：**

**arr = [1,2,3,4,5]**

**Arr2 = [1,2,3,4,5]**

## **10、对象属性获取的方式有二种.和[]，区别是什么，代码说明**

**答 ：var obj = {name:'amy',age:20};**

**var n = 'name';**

**obj[n] //[]可以是变量 .不行**

## **11、**

## **var arr = [{check:true,id:1},{check:false,id:2},{check:false,id:3},{check:true,id:4}]**

## **请取出check为true的值**

**答：var a = arr.filter(item=> item.check);**

## **12、**

**计算出以下商品单价和金额的总和，初始值为10**

**arr = [{count:1,price:10,name:'a'},{count:3,price:50,name:'b'},{count:5,price:23,name:'c'},{count:2,price:14,name:'d'}];**

**答：var c = arr.reduce(function(total, cur){**

**return total+cur.count\*cur.price;**

**},10)**

**console.log(c)**

## **13、**

**function fun () {**

**console.log( n );**

**n = 456;**

**console.log( n );**

**}**

**var n= 123;**

**fun( n);**

**console.log( n ); 解析的过程及结果**

**答：123 456 456**

**//解析过程**

**var n;**

**function fun(){}**

**n = 123;**

**//fun(n)执行**

**console.log(n)   //123**

**n = 456  //改变全局变量的值**

**console.log(n) //456**

**console.log( n );  //最后再执行  结果456**

## **14、**

**var length = 100;**

**function f1() {**

**console.log( this.length )**

**}**

**var obj = {**

**x: 10,**

**f2: function( f1 ){**

**f1();**

**arguments[0]();**

**}**

**}**

**obj.f2(f1,1,2,3);**

**解析的过程及结果**

**答：100 4**

**f1()没有调用者，this的指向是window 所以this.length就是100**

**arguments[0]();**

**Arguments[0]指向f1**

**arguments[0](); 表示f1(), this的指向是arguments 所以this.length就是arguments.length 值为4**

## **15、用递归来实现5的阶乘**

**答：function func(n){**

**if (n == 1){**

**return 1;**

**}**

**// func(n-1) 因为传递的参数是 n-1,那么就是求 (n-1) 的阶乘**

**return n \* func(n-1);**

**}**

**console.log(func(5))**

## **16、let和var的不同请从以下几个方面来通过代码说明**

1. 不存在变量提升

console.log(a);

var a='a';

console.log(b);

let b='b';

1. 同一个作用域内不能重复定义同一个名称

let b = 1;

let b = 2;

console.log(b);

1. 有着严格的作用域

function test2() {

let str='123';

if(true){

let str='456';

};

console.log(str);

};

test2();

1. 块级作用域的重要性

var arr = [];

for(let i=0; i<5; i++){

arr[i] = function(){

console.log(i)

}

};

arr[4]();

## **17、暂时性死区是什么，举例1-2例说明**

function f3(i){

let i;

console.log(i)

};

f3(10)

let x = y,y = 10;

function f2(){

console.log(x,y)

};

## **18、取二组数据的并集，交集，差集并去重**

var a1 = [1,2,3,4,5,6,4,5,6];

var a2 = [4,5,6,7,8,9,7,8,9];

//并集

//1、合并

var merge = [...a1,...a2];

//2、去重

var noDuplicate = new Set(merge);

//3、转类型

var a3 = [...noDuplicate];

//4、代码合并

var a3 = [...new Set([...a1,...a2])] //[1, 2, 3, 4, 5, 6, 7, 8, 9]

//交集

//1、数据过滤

//a1中包含a2

var filterD1 = a1.filter(v=>a2.includes(v)); // [4, 5, 6, 4, 5, 6]

//var filterD2 = a1.filter(v=>new Set(a2).has(v)); //[4, 5, 6, 4, 5, 6]

//a2中包含a1

//var filterD1 = a2.filter(v=>a1.includes(v)); //[4, 5, 6]

//var filterD2 = a2.filter(v=>new Set(a1).has(v)); //[4, 5, 6]

//2、去重

var noDuplicate = new Set(filterD1);

//3、转类型

var a3 = [...noDuplicate];

//4、代码合并

var a3 = [...new Set(a1.filter(v=>new Set(a2).has(v)))] ; //[4,5,6]

//差集

//a1有a2无

var a4 = [...new Set(a1.filter(v=>!new Set(a2).has(v)))]; // [1, 2, 3]

//a2有a1无

var a5 = [...new Set(a2.filter(v=>!new Set(a1).has(v)))]; // [7, 8, 9]

**19、css只在当前组件起作用**  
答：在style标签中写入**scoped**即可 例如：<style scoped></style>

**20、v-if 和 v-show 区别**  
答：简单回复：v-if按照条件是否渲染，v-show是display的block或none；

详细比较：  
v-show是css切换，v-if是完整的销毁和重新创建。

使用  
频繁切换时用v-show，运行时较少改变时用v-if

v-if=‘false’ v-if是条件渲染，当false的时候不会渲染

**21、$route和$router的区别**  
答：$route是“路由信息对象”，包括path，params，hash，query，fullPath，matched，name等路由信息参数。而$router是“路由实例”对象包括了路由的跳转方法，钩子函数等。

### **22、组件中data为什么是函数**

为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象？

因为组件是用来复用的，JS 里对象是引用关系，这样作用域没有隔离，而 new Vue 的实例，是不会被复用的，因此不存在引用对象的问题。

### **23、VUE生命周期的几个阶段**

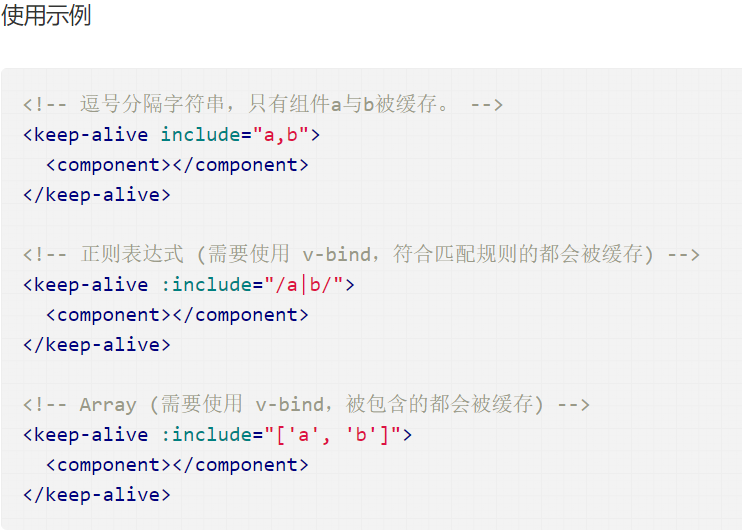
**beforeCreate**（创建前） 在数据观测和初始化事件还未开始  
**created**（创建后） 完成数据观测，属性和方法的运算，初始化事件，$el属性还没有显示出来  
**beforeMount**（载入前） 在挂载开始之前被调用，相关的render函数首次被调用。实例已完成以下的配置：编译模板，把data里面的数据和模板生成html。注意此时还没有挂载html到页面上。  
**mounted**（载入后） 在el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用。实例已完成以下的配置：用上面编译好的html内容替换el属性指向的DOM对象。完成模板中的html渲染到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。

### **24、对keep-alive的理解**

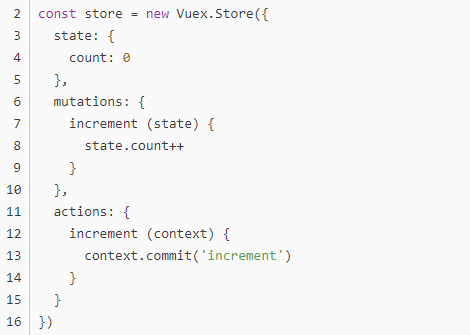
**keep-alive**是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染。  
在vue 2.1.0 版本之后，keep-alive新加入了两个属性: include(包含的组件缓存) 与 exclude(排除的组件不缓存，优先级大于include) 。

参数解释  
include - 字符串或正则表达式，只有名称匹配的组件会被缓存  
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  
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 。





### **25、vuex中mutation和action的详细区别**



1）、流程顺序

“相应视图—>修改State”拆分成两部分，视图触发Action，Action再触发Mutation。

2）、触发方式

mutation的触发通过store.commit来进行

action的触发通过store.dispatch进行

3）、角色定位

基于流程顺序，二者扮演不同的角色。

Mutation：专注于修改State，理论上是修改State的唯一途径。

Action：业务代码、异步请求。

4）、限制

角色不同，二者有不同的限制。

Mutation：必须同步执行。

Action：可以异步，但不能直接操作State。

### **26、watch和computed差异**

watch是进行数据监听，然后进行相应的操作，执行方法等conputed和methods的合体使用，比较耗性能，与vue性能优化相背而驰，尽量减少使用！computed是数据改变进行相应的数据变化，由老数据迸发新的数据（return返回），会利用缓存机制对 数据进行缓存 ，只有当 依赖数据变化的时候才会进行相应的变化

### **27、组件中data为什么是函数**

为什么组件中的 data 必须是一个函数，然后 return 一个对象，而 new Vue 实例里，data 可以直接是一个对象？

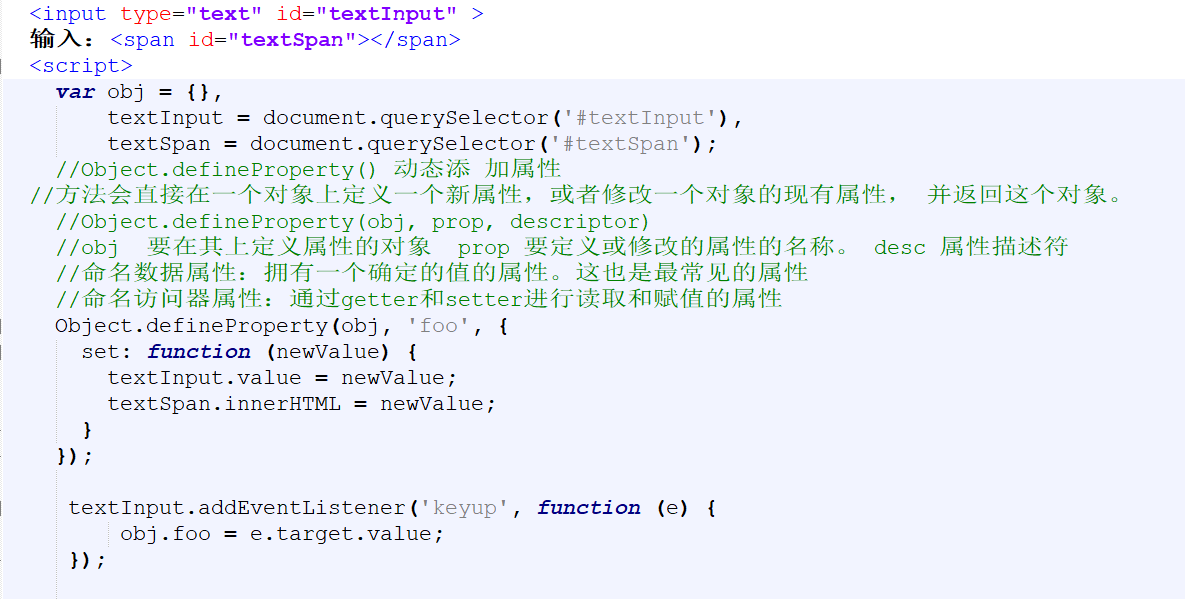
因为组件是用来复用的，JS 里对象是引用关系，这样作用域没有隔离，而 new Vue 的实例，是不会被复用的，因此不存在引用对象的问题。

**28、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 。

### **29、VUE实现数据双向绑定的原理：**

vue实现数据双向绑定主要是：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 将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的数据变化，通过Compile来解析编译模板指令（vue中是用来解析 {{}}），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变更双向绑定效果。



### **30、虚拟DOM的优缺点？**

优点:

保证性能下限: 虚拟DOM可以经过diff找出最小差异,然后批量进行patch,这种操作虽然比不上手动优化,但是比起粗暴的DOM操作性能要好很多,因此虚拟DOM可以保证性能下限

无需手动操作DOM: 虚拟DOM的diff和patch都是在一次更新中自动进行的,我们无需手动操作DOM,极大提高开发效率

跨平台: 虚拟DOM本质上是JavaScript对象,而DOM与平台强相关,相比之下虚拟DOM可以进行更方便地跨平台操作,例如服务器渲染、移动端开发等等

缺点:

无法进行极致优化: 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,比如VScode采用直接手动操作DOM的方式进行极端的性能优化

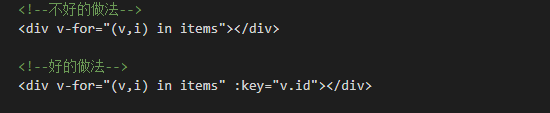
**二、企业实战中使用 Vue 的最佳做法**

## 1.始终在 v-for 中使用 :key

在需要操纵数据时，将key属性与v-for指令一起使用可以让程序保持恒定且可预测。

这是很有必要的，这样Vue就可以跟踪组件状态，并对不同的元素有一个常量引用。在使用动画或Vue转换时，key 非常有用。

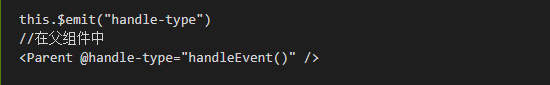
如果没有key ，Vue只会尝试使DOM尽可能高效。 这可能意味着v-for中的元素可能会出现乱序，或者它们的行为难以预测。 如果我们对每个元素都有唯一的键引用，那么我们可以更好地预测Vue应用程序将如何精确地处理DOM操作。



## 2、在事件中使用短横线命名

在发出定制事件时，最好使用短横线命名，这是因为在父组件中，我们使用相同的语法来侦听该事件。

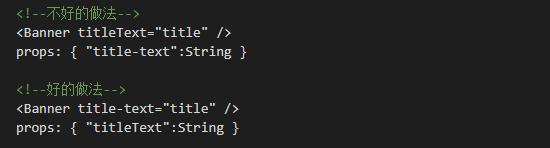
因此，为了确保我们各组件之间的一致性，并使您的代码更具可读性，请在两个地方都坚持使用短横线命名。



## 3.使用驼峰式声明 props，并在模板中使用短横线命名来访问 props

最佳做法只是遵循每种语言的约定。 在 JS 中，驼峰式声明是标准，在HTML中，是短横线命名。 因此，我们相应地使用它们。

幸运的是，Vue 已经提供了驼峰式声明和短横线命名之间转换，因此除了实际声明它们之外，我们不必担心任何事情。

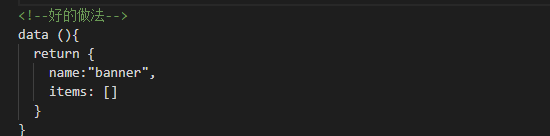


## 4.data 应始终返回一个函数

声明组件data时，data选项应始终返回一个函数。 如果返回的是一个对象，那么该data将在组件的所有实例之间共享。

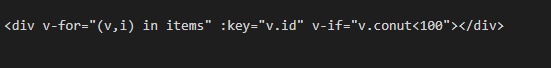


但是，大多数情况下，我们的目标是构建可重用的组件，因此我们希望每个组件返回一个惟一的对象。我们通过在函数中返回数据对象来实现这一点。

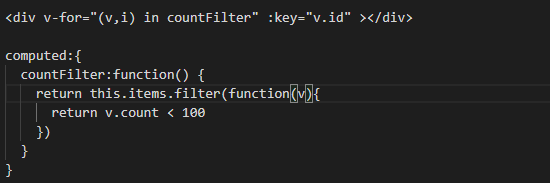


## 5. 不要在同个元素上同时使用v-if和v-for指令

为了过滤数组中的元素，我们很容易将v-if与v-for在同个元素同时使用。



问题是在 Vue 优先使用v-for指令，而不是v-if指令。它循环遍历每个元素，然后检查v-if条件。



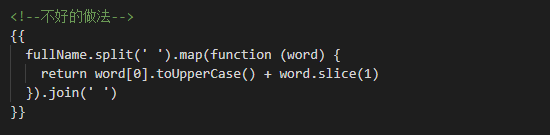
这么做有几个好处：

* 渲染效率更高，因为我们不会遍历所有元素
* 仅当依赖项更改时，才会重使用过滤后的列表
* 这写法有助于将组件逻辑从模板中分离出来，使组件更具可读性

## 6. 模板表达式应该只有基本的 JS 表达式

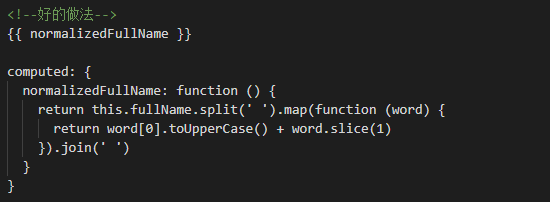
在模板中添加尽可能多的内联功能是很自然的。但是这使得我们的模板不那么具有声明性，而且更加复杂，也让模板会变得非常混乱。

为此，让我们看看Vue样式指南中另一个规范化字符串的示例，看看它有多混乱。



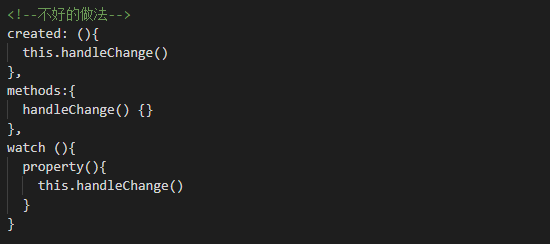
基本上，我们希望模板中的所有内容都直观明了。 为了保持这一点，我们应该将复杂的表达式重构为适当命名的组件选项。

分离复杂表达式的另一个好处是可以重用这些值。



## 7.不要在“created”和“watch”中调用方法

Vue开发人员经常犯的一个错误是他们不必要地在created和watch中调用方法。 其背后的想法是，我们希望在组件初始化后立即运行watch。

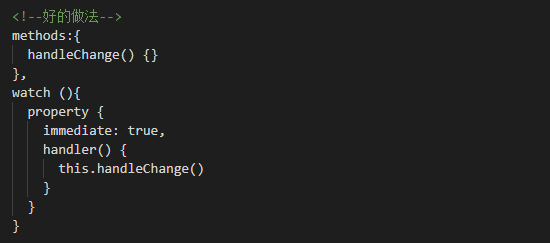


但是，Vue为此提供了内置的解决方案，这是我们经常忘记的Vue watch属性。

我们要做的就是稍微重组watch并声明两个属性：

1）handler (newVal, oldVal)-这是我们的watch方法本身。

2）immediate: true- 代表如果在 wacth 里声明了之后，就会立即先去执行里面的handler方法，如果为 false就跟我们以前的效果一样，不会在绑定的时候就执行



总结：在实际开发过程中，这些技巧对于您是有很多帮助的，可以让我们的代码更易于维护，更具有扩展性和可读性