Contents

[Vue2.x的响应式原理 2](#_Toc37147008)

[Proxy 4](#_Toc37147009)

[Vue和React的对比 5](#_Toc37147010)

[组件化的共同思想 5](#_Toc37147011)

[监听数据变化的实现原理不同 5](#_Toc37147012)

[DOM渲染优化 5](#_Toc37147013)

[数据流不同 5](#_Toc37147014)

[模板渲染方式不同 6](#_Toc37147015)

[Vuex和Redux的区别 6](#_Toc37147016)

[侦听属性 7](#_Toc37147017)

[Methods与computed的差别： 7](#_Toc37147018)

[v-if和v-show的区别 7](#_Toc37147019)

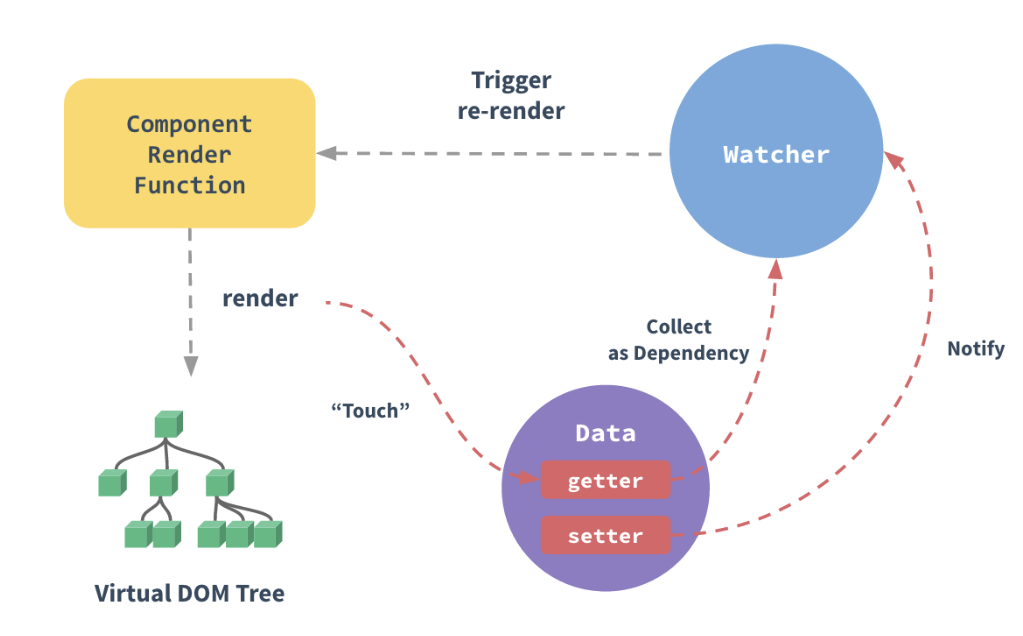
[组件中的data为什么是一个函数 7](#_Toc37147020)

[Vue的生命周期 8](#_Toc37147021)

[NextTick的作用 9](#_Toc37147022)

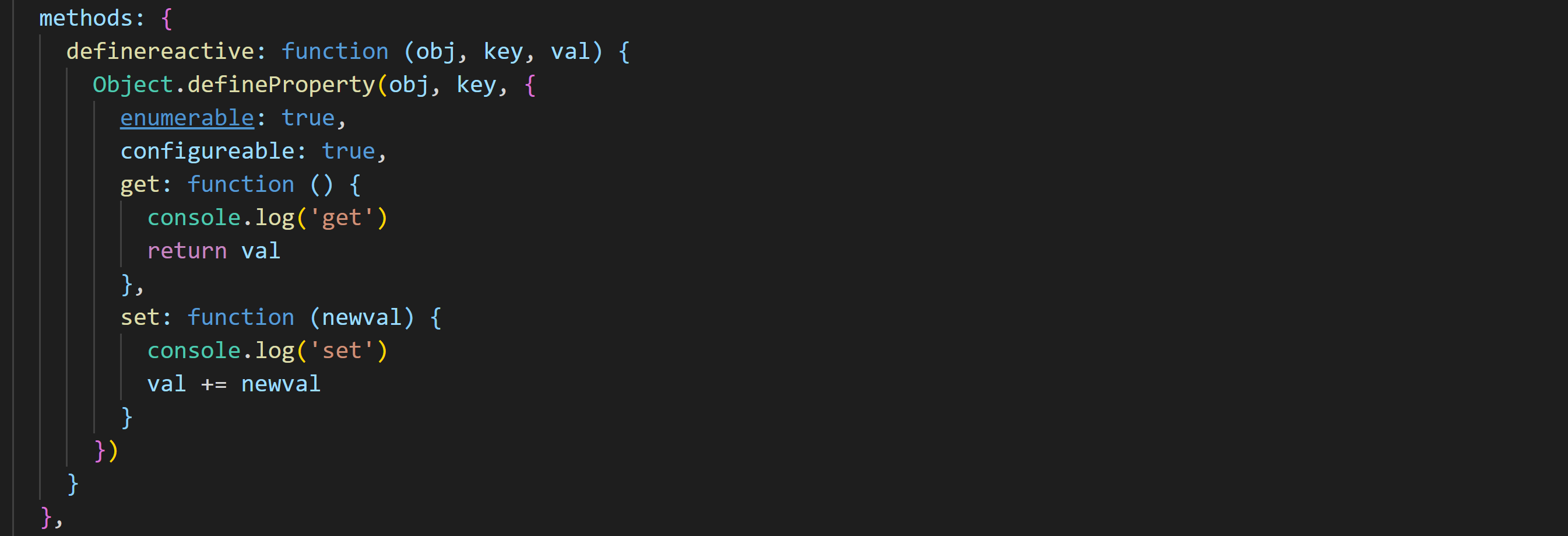
# Vue2.x的响应式原理

参考资料：<https://cn.vuejs.org/v2/guide/reactivity.html>



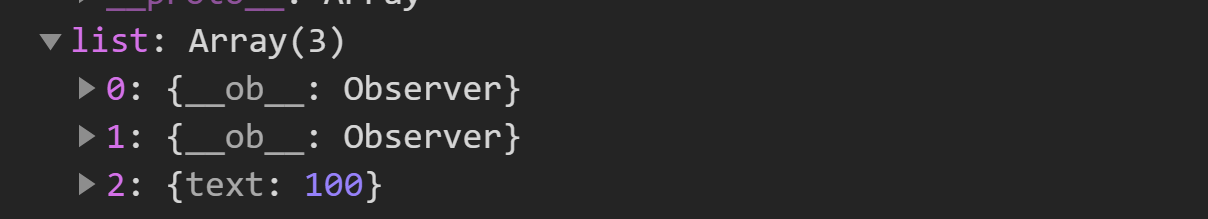
1. 把js对传入到vue实例作为data时，vue会遍历对象的所有属性并通过Object.defineProperty（IE8及以下不支持）把属性转化为getter/setter。
2. 每个组件实例对应一个watcher，在组件渲染时会把接触过的数据属性记录为依赖，当依赖项的setter被触发的时候会通知watcher，从而重新渲染组件。

**数据劫持是响应式的根本依赖：手写劫持**



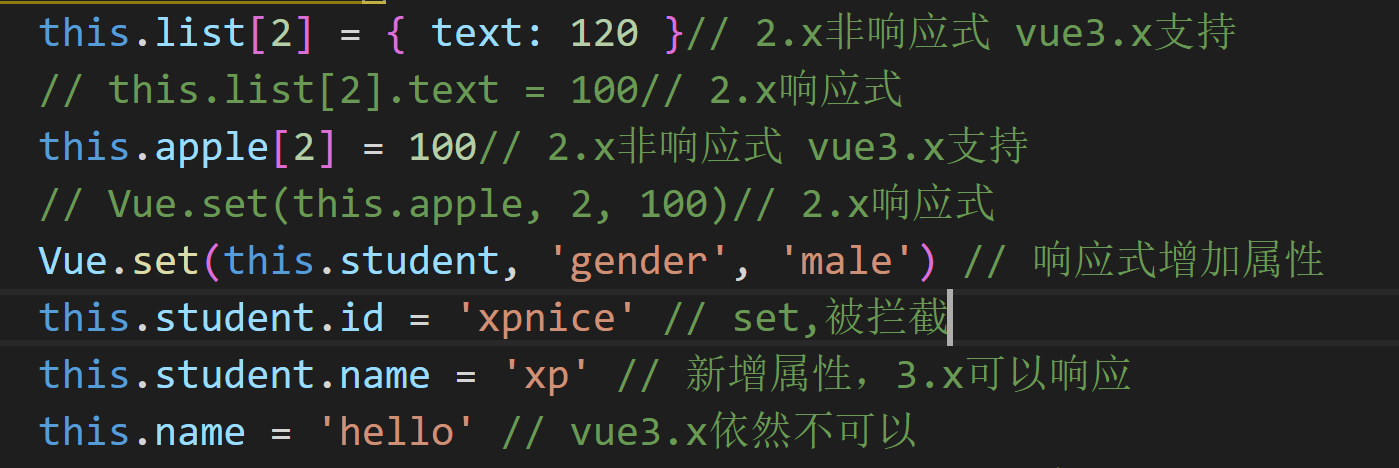
**不允许动态添加根级响应式属性(2.x和3.x都不支持)**

**Vue2.x不支持通过索引设置数组成员，3.x可以**



不会为索引的修改添加\_\_ob\_\_

通过非变易给数组整体重新赋值可以，特殊注意数组元素是类的时候



**Vue2.x不能检测对象属性的添加或删除，3.x可以**

可以使用

Vue.set(object, propertyName, value)

this.$set(this.someObject,'b',2)

this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })

**Vue2.x数据劫持为对象（包含数组中的对象及子对象）添加\_\_ob\_\_属性**

\_\_ob\_\_上有一个dep属性作为依赖收集的储存器但是数组的pop、push、shift、unshift、splice、sort和reverse方法可以响应

**流程**

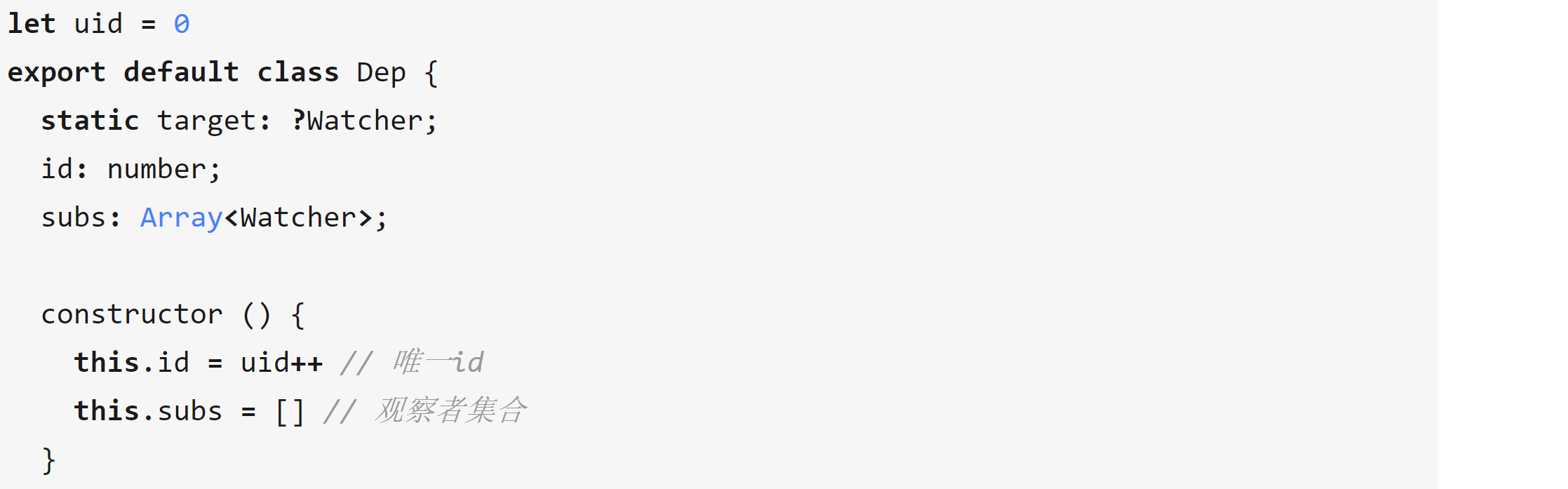
observe函数预处理创建observer实例，为被劫持的对象添加\_\_ob\_\_属性指向自身的Observer实例作为是否Observer的唯一标识，然后劫持对象成员(walk)和数组成员(observeArray)，walk中包含了defineReactive函数来实现拦截。在需要订阅的地方如编译模板添加观察者(watcher)并立刻通过取值操作触发指定属性的getter方法，从而将观察者添加进Dep(利用闭包进行依赖收集)，在setter触发的时候进行notify通知所有观察者进行update

**如何拦截一个对象？**

defineReactive会实例一个Dep，get中的dep.apend()收集依赖，get中的dep.notify来通知观察者。

**Dep是什么**

Dep是发布者，可以被多个观察者订阅



**Watcher**

可以理解为observer是对数据的，wathcer是对组件的。通过get触发取值操作，进而出发属性的getter，从而在闭包中添加watcher

# Proxy

Vue3.x用proxy替换object.defineProperty

解决了不能相应对象属性的添加和删除以及不能直接操作数组下标的问题

# Vue和React的对比

参考资料：<https://juejin.im/post/5b8b56e3f265da434c1f5f76>

## 组件化的共同思想

组件化，减少不必要的重新渲染

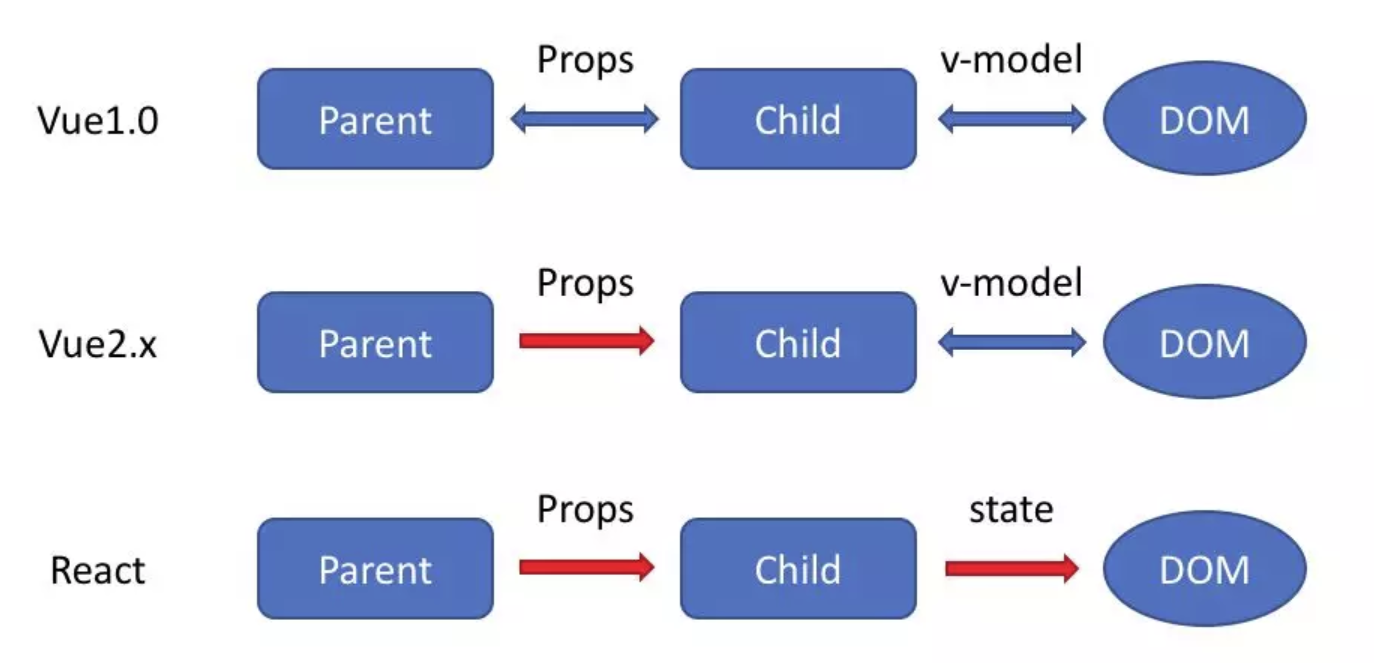
## 监听数据变化的实现原理不同

* Vue的响应式原理，
* React通过比较引用（鲁棒）

## DOM渲染优化

* Vue将模板编译成virtual DOM的渲染函数，结合相应系统，VUE可以计算出最少需要重新渲染多少组件并把DOM操作次数减到最少
* 而react需要优化，通过shouldComponentUpdate钩子函数。shouldComponentUpdate函数是重渲染时render()函数调用前被调用的函数，它接受两个参数：nextProps和nextState，分别表示下一个props和下一个state的值。并且，当函数返回false时候，阻止接下来的render()函数的调用，阻止组件重渲染，而返回true时，组件照常重渲染。

## 数据流不同



* Vue2.x可以通过v-model实现dom和组件的双向绑定
* 而react需要根据callback来setstate

## 模板渲染方式不同

表层：

React通过JSX渲染模板而Vue通过拓展的HTML语法（也可以直接写render函数）

深层次：

React在组件JS代码中通过原生JS的常见语法实现

Vue通过从js分离出来的template块的指令来实现

例子：

React import可以直接render（render函数的闭包性）

而 Vue还要在component里声明

## Vuex和Redux的区别

Vuex：

$store直接注入到了组件实例中，通过this.$store读取，dispatch来更新

Redux：

往往需用借用react-redux显式用connect将props和store中的state连接。

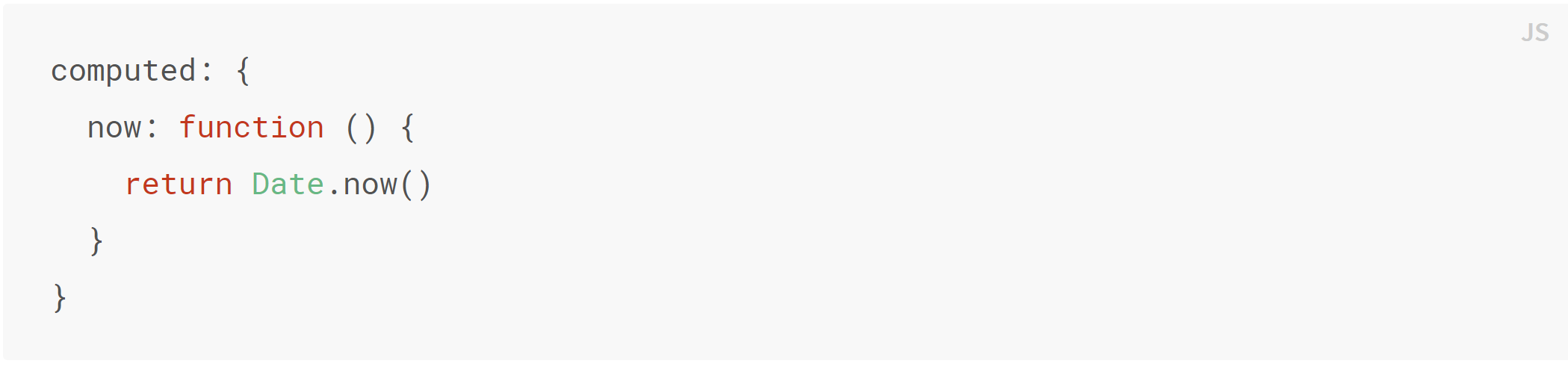
# 侦听属性

更通用的方法来观察和响应Vue实例上的数据变通。注意不要滥用，推荐使用计算属性而非命令式的watch回调，但watch可以实现异步操作（如debounce防抖）这是通过计算属性不能实现的。

# Methods与computed的差别：

二者都是函数，但是computed计算属性是基于他们的响应式依赖进行缓存的，也就是说当响应式依赖不变的时候，调用computed可以立即返回。

要特殊注意在计算属性中调用的非响应式依赖，如下图则不会更新



而methods每次都会重新执行方法

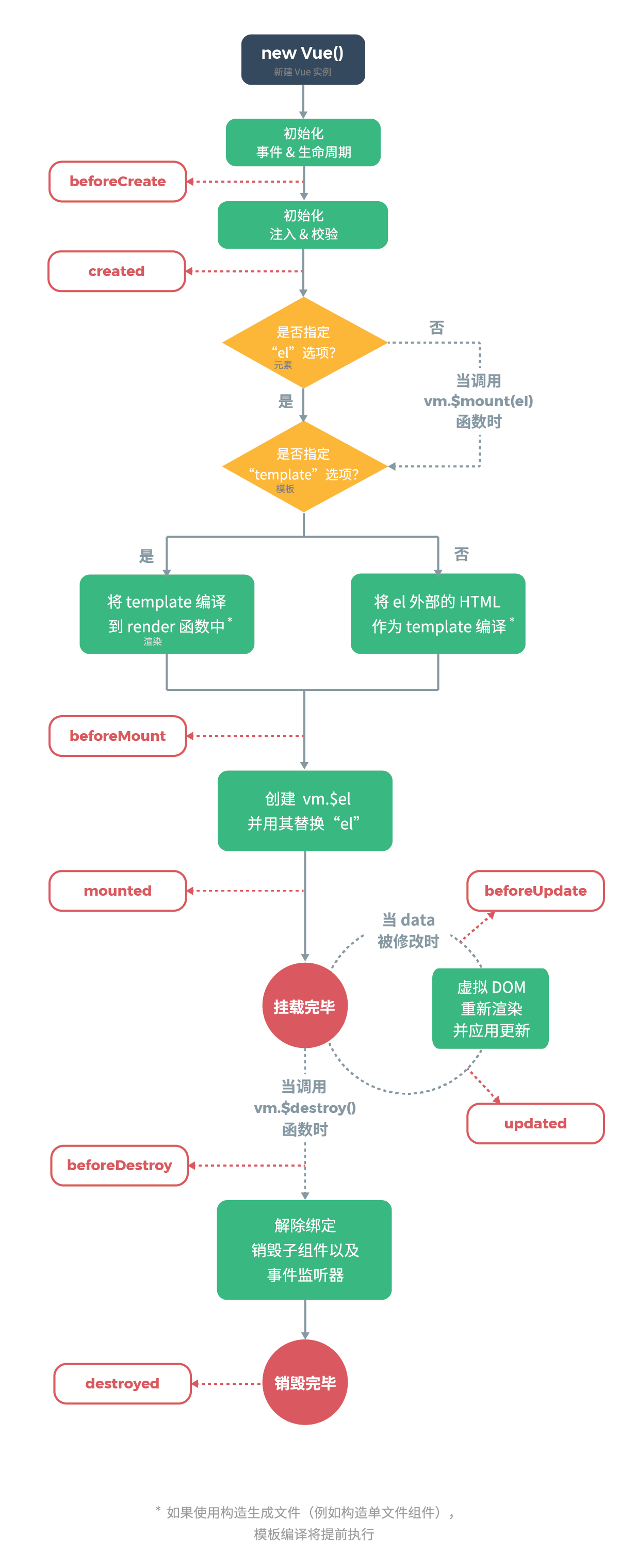
# v-if和v-show的区别

当条件不成立时，v-if不会渲染DOM元素，v-show操作的是样式（display），切换当前DOM的显示和隐藏

# 组件中的data为什么是一个函数

因为一个组件被复用多次就会创建多个实例，本质上这些实例使用的是一个构造函数，如果data是对象的话属于引用类型，会影响到所有的实例，所以为了保证组件不同的实例之间data不冲突必须是一个函数

# Vue的生命周期



# NextTick的作用

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



注意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
* mutationobserver
* setimmediate
* 以上均不行则菜用setTimeout

来定义一个异步方法，多次调用则会存入队列，通过选择的异步方法清空队列