Vue3.0学习报告

# 学习资源：「李南江」Vue3.0正式版教程2020年9月19号全网首发-Vue3 One Piece 持续更新中...

网址：

<https://www.bilibili.com/video/BV14k4y117LL?from=search&seid=5759142316795912858>

**第一次学习**

**2021.5.3 下午15:33开始 晚上20:34结束 中有间断非全为学习时间**

**1. Vue3.0中的六大亮点**

|  |  |  |
| --- | --- | --- |
| 序号 | 特性 | 解析 |
| 1 | Performance | 性能上比Vue2.0快1.3~2倍 |
| 2 | Tree shaking support | 按需编译，体积更加轻量化 |
| 3 | Composition API | 组合API，可参考React hooks理解 |
| 4 | Better TypeScript support | 对Ts提供了更好的支持 |
| 5 | Custom Renderer API | 暴露了自定义渲染API |
| 6 | Fragment,Teleport(Protal),Suspense | 更先进的组件 |

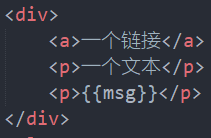
1. **Vue3.0是基于什么优化，如何做到更轻量，更快的？**

一 、diff 算法优化

Vue 2中的虚拟Dom是全量比较，Vue 3新增静态标记（PatchFlag）。在与数据变化后，与上次虚拟DOM节点比较时，只比较带有PatchFlag标记的节点。并且可以从flag信息中得知具体需要比较的内容。

静态标记就是非全量比较，只会比较那些被标记的变量，比较的数量大大减少因此提升性能

示例：





由上可知：

①在vue2.0中对于数据变化后重新渲染的DOM树，会与上次渲染的DOM树逐个比较节点

②在vue3.0的diff中，创建虚拟DOM时，会根据该DOM是否会变化而添加静态标记，数据更新需要生成新的虚拟DOM时，只会与上次渲染的且被标记的节点比较。

③不同的动态变化类型，为了便于区分，标记的数值也不同，因此在vue3.0中比较次数更少，效率更高，速度更快。

示例



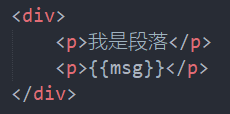
标记查询列表



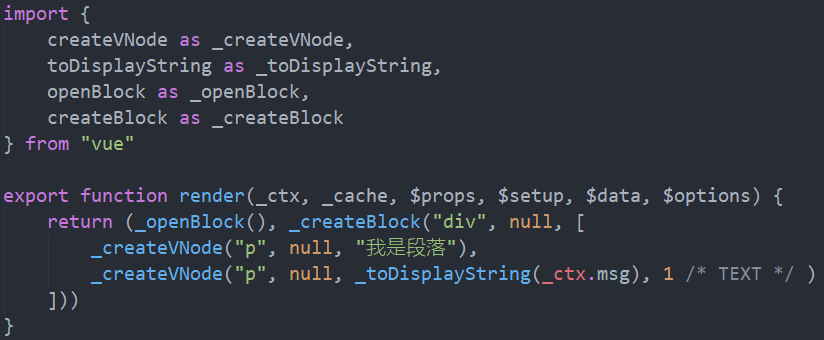
二 、hoistStatic静态提升

vue2.0中，在更新时，元素即使没有变化，也会重新创建进行渲染；vue3.0中，不参与更新的元素；会静态提升，只创建一次下次渲染直接复用。

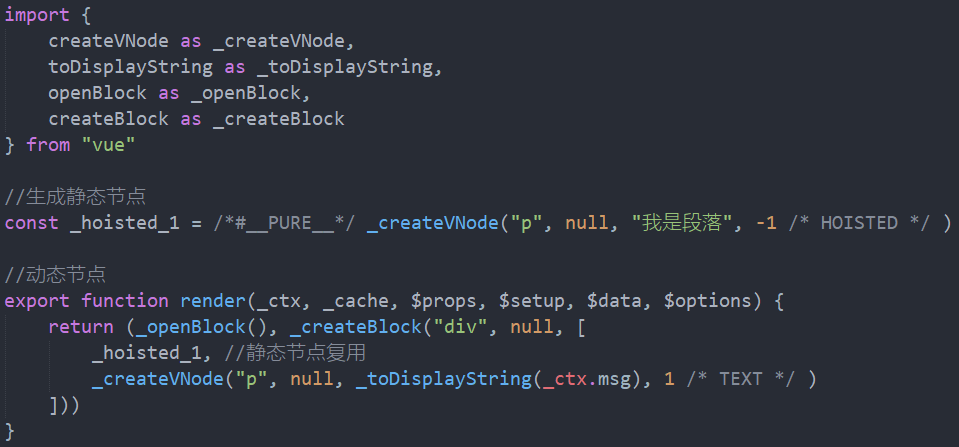
因此在vue3.0中复用更多，创建次数更少，速度更快。示例：



vue2.0静态提升前



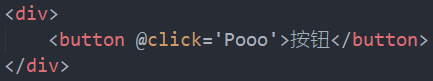
vue3.0静态提升后

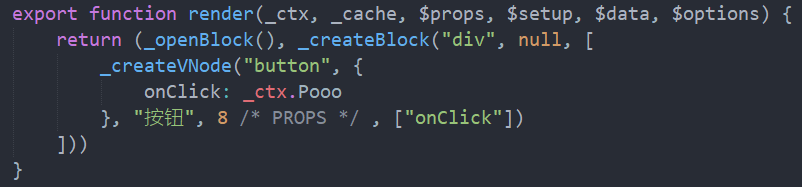


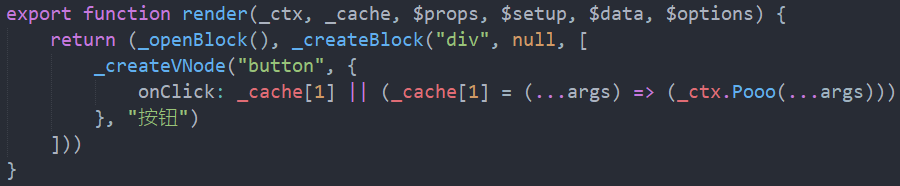
三、cachehandlers 事件侦听缓存

默认情况下 onClick 会被视为动态绑定，因此会追踪它的变化。事件绑定的函数为同一个，因此不追踪它的变化，直接缓存后进行复用。在Vue3中的diff算法中，只有存在静态标记的节点才会进行追踪，事件侦听缓存本质上是去除了不必要的diff比较

示例



然后在options中打开事件侦听缓存（cachehandlers）



1. SSR渲染

当有大量静态的内容时候，这些内容会被当做纯字符串推进一个Buffer里面，即使存在动态的绑定，会通过模板插值嵌入进去。这样会比通过虚拟dom来渲染快上很多。

当静态内容达到一定量级时候，会使用\_createStaticVNode方法在客户端dom来渲染一个static node，这些静态node，会被直接innerHtml，就不需要创建对象，然后根据对象渲染。

1. **vue3.0创建项目**
2. Vue-cli

npm install -g @vue/cli

vue create projectName

cd projectName

npm run serve

1. Webpack

git clone https://github.com/vuejs/vue-next-webpack-preview.git projectName

cd projectName

npm install

npm run dev

1. Vite

Vite是Vue作者开发的一款意图取代webpack的工具，原理是利用ES6的import发送请求加载文件的特性，进而拦截，然后做预编译，省去webpack冗长的打包

使用步骤

①安装Vite命令： npm install -g create-vite-app

②创建Vue3项目： create-vite-app PoooName

③安装依赖：cd PoooName / npm install / npm run dev

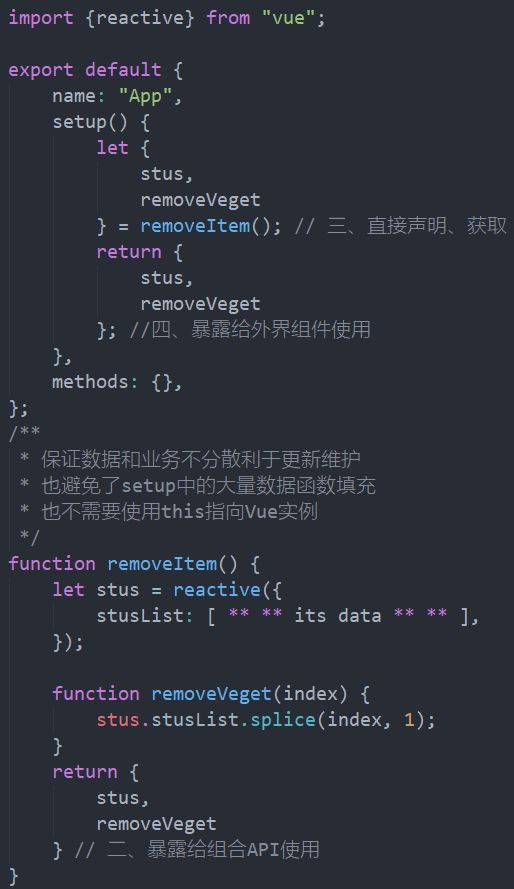
④Vue3.0中兼容2.0的写法，具体代码在此文件同级的PoooName项目文件中

1. **vue3.0中的 reactive 用法**

在2.0中对于业务实现，需要先在data中变更补充数据，然后在methods或watch中补充业务逻辑，这样数据和逻辑是分模块的，查找不便，不利于业务的管理和维护，为解决这样的问题，Vue3.0中加入了reactive。Vue3.0提供了setup 组合API的入口函数，可以把数据和业务逻辑组合在一起



另一种更加优雅的写法，也是非常非常推荐的写法是



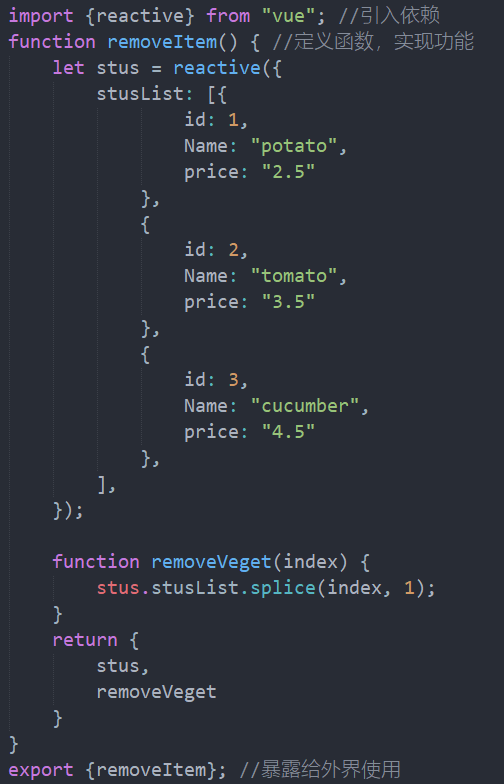
功能分离：

乍一看上方把函数整合到下方，然后在setup中引用是很简洁，若需要的业务功能多了呢，比如增加个updateItem,addItem，虽然数据和逻辑代码还是在一块，但是各种功能聚集在一块还是显得文件臃肿，那么还要继续优化，分离各个功能

①新建一个单独的JS文件，如remove.js

②在APP文件中引入这个JS文件

③这样就可以在单独的JS文件中对某个功能进行维护了





1. **vue3.0中的 Composition API本质**



Option API：即在APP中为实现业务逻辑进行的配置

在2.0中比如你要实现一个点击按钮，弹出提示语功能，你需要

①利用 Opaction API

②在 data 中配置数据

③在 methods 中配置相应函数

在3.0中通过上方 reactive 的知识点我们知道，实现这个功能，你需要

①利用 Composition API

②在 setup 中定义数据，编写函数

③通过 return{ 数据,方法}暴露出去

其实 Composition (也叫注入API)本质是在运行时

①把暴露出来的数据注入到 opaction 中的 data

②把暴露出来的函数注入到 opaction 中的 methods

小结：

Opaction API中对配置项都进行了规定，比如：

①在data中配置数据，methods中编写方法，watch中进行监听。

②保姆式的分配较为清晰，但也对代码层层分割，维护比较麻烦

Composition中更加自由，比如：

①不用担心各种this指向

②随意进行模块分割导出，维护时查找固定模块文件

1. **生命周期中的 setup**

|  |  |  |
| --- | --- | --- |
| 序列 | 选项式API | Hook inside setup |
| 1 | beforeCreate | Not needed\* |
| 2 | created | Not needed\* |
| 3 | Composition API | onBeforeMount |
| 4 | mounted | onMounted |
| 5 | ... | ... |

1. **setup的执行时机在beforeCreate和created之间，还是beforeCreate之前？**

官方文档原话：因为 setup 是围绕 beforeCreate 和 created 生命周期钩子运行的，所以不需要显式地定义它们。换句话说，在这些钩子中编写的任何代码都应该直接在 setup 函数中编写。

依据官网描述及下方2、3点推断setup肯定是在created之前，还有一种是说法是在beforeCreate之前，但我不是很能理解，根据原文beforeCreate中的任何代码可以直接在setup中编写

在Vue生命周期中我们知道：

①beforeCreate时，刚初始化一个空Vue实例对象，data和methods中数据未初始化

②created执行时，data和methods已经初始化完毕

③Composition需要把setup中的数据对应注入到data和methods中去

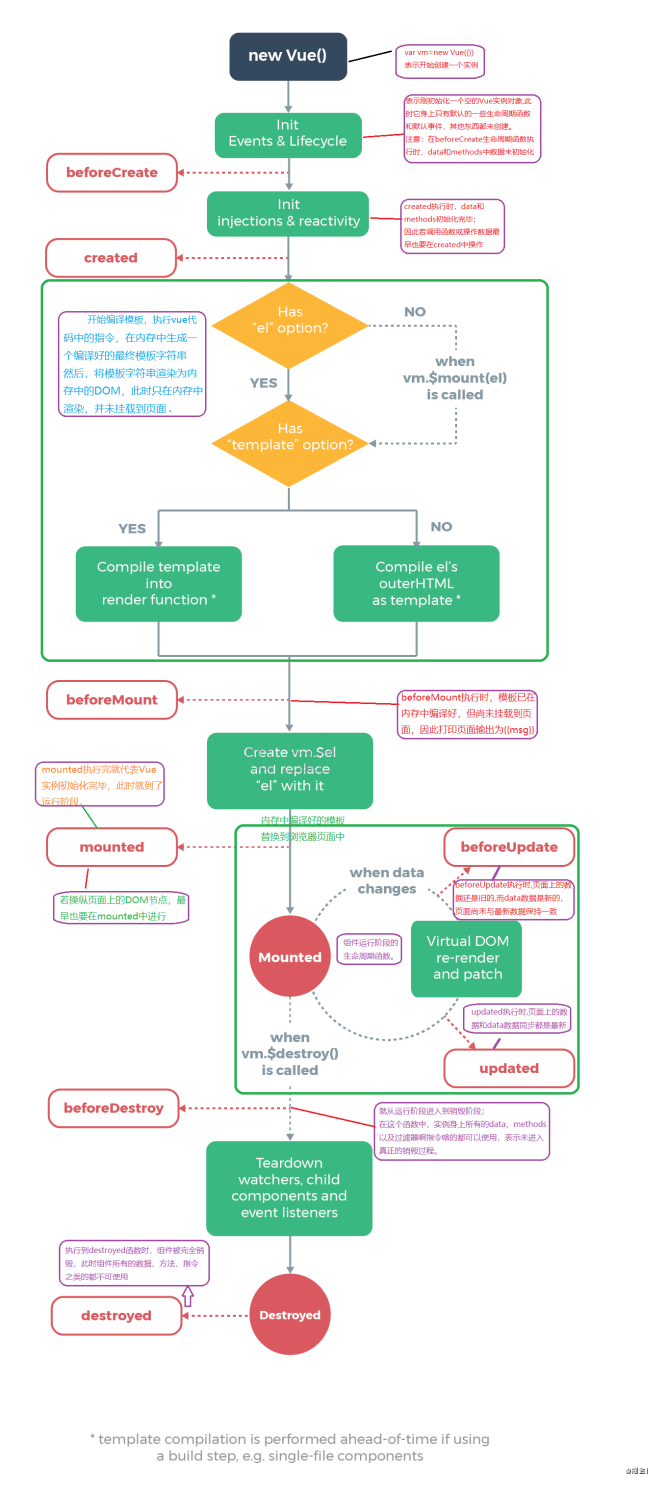
④很显然setup必须要在 created 之前执行

也因此，若你在Vue3.0中进行混合开发，不可以在 setup 中使用 data 中的数据和 methods 中的方法。在3.0中setup 里的this也被修改为了undefiend。在3.0中setup 里也不可以使用异步

①由于在执行setup函数的时候，还没有执行 created 生命周期方法，所以在 setup 函数中，是无法使用 data 和 methods

②由于不能再 setup 函数中使用 data 和 methods，所以 Vue 为了避免我们的错误使用，它直接将 setup 函数中的 this 修改成了 undefined

③setup 函数只能是同步的，不能是异步的，async setup() {} 错误使用



1. **什么是 reactive**

reactive是VUE3.0中提供的实现响应式数据的方法，在Vue2.0中使用的是defineProperty来实现的。而VUE3.0中使用的是ES6里的proxy实现的

reactive中需要注意的点：

①传递给它的类型必须是对象（JSON或者arr数组）

②并且它会自动把传递进来条件再赋值给Proxy对象

③若传递的为上述以外的对象

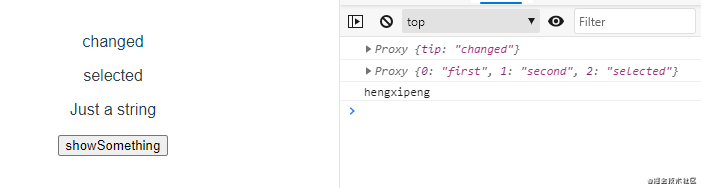
在方法中直接修改它，界面上它也不会自动更新

若想更新只能通过重新赋值的方式

示例



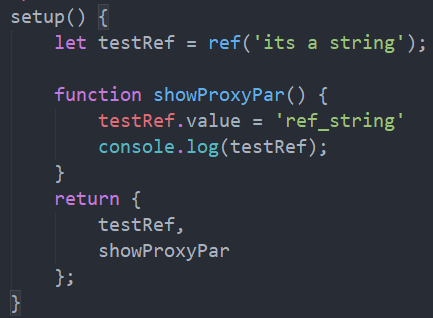
效果如下图所示，符合传递条件的参数会再赋值给Proxy，并且修改它也会直接影响视图



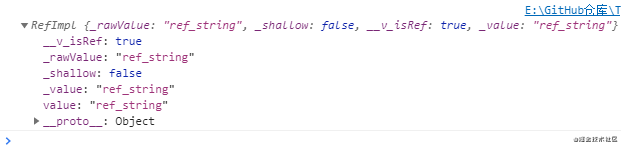
1. **什么是 ref**

它也是实现响应式数据的方法。reactivce向来都是进行传递对象，实际开发中若只想更改某简单变量则会显得大材小用。所以vue3提供了ref方法，来实现对简单值的监听

ref 底层的本质其实还是 reactive，系统会自动根据我们给 ref 传入的值将它转换成 ref(xx) -> reactive({value: xx})



如下图



注意点

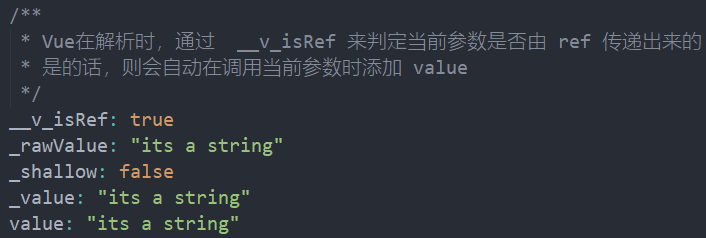
　　在 Vue 中使用 ref 的值不用通过 value 获取

　　在 JS 中使用 ref 的值必须通过 value 获取

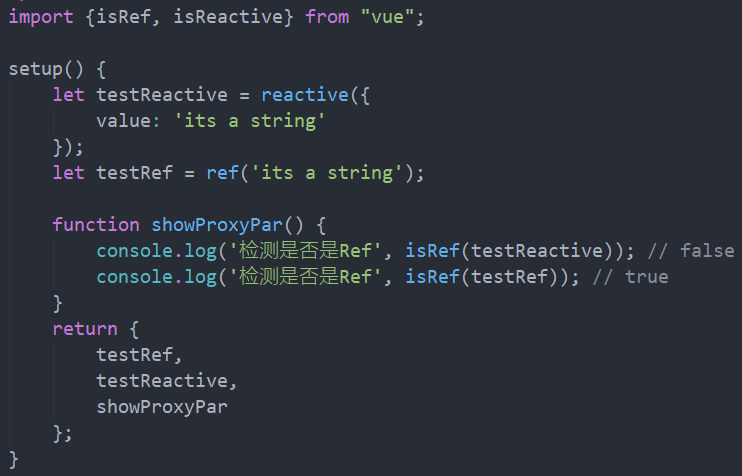
1. **ref和reactive之间的不同**

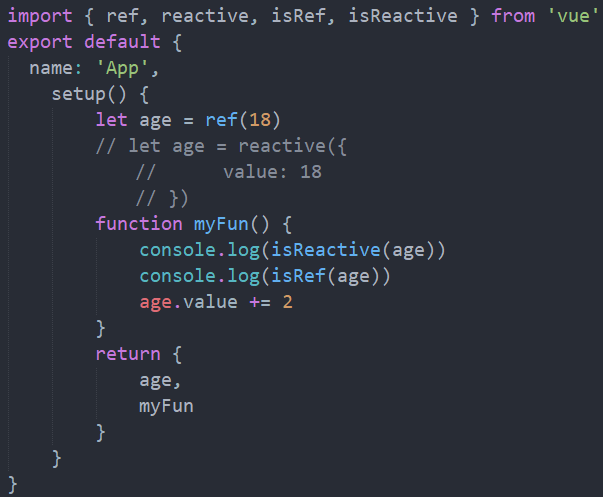
****

通过以上得知，使用ref其实相当于使用reactive，只是省略了手动创建对象的步骤。ref中底层会添加一个value的键，并且在视图中可省略调用value。经过测试，使用reactive，创建一个键值为value的Json对象，验证是否可省略value调用（不可以）得知，只有使用ref传递参数时，视图才允许省略value调用



其中Vue3.0中提供了两个方法，isReactive和isRef用来判定数据来源





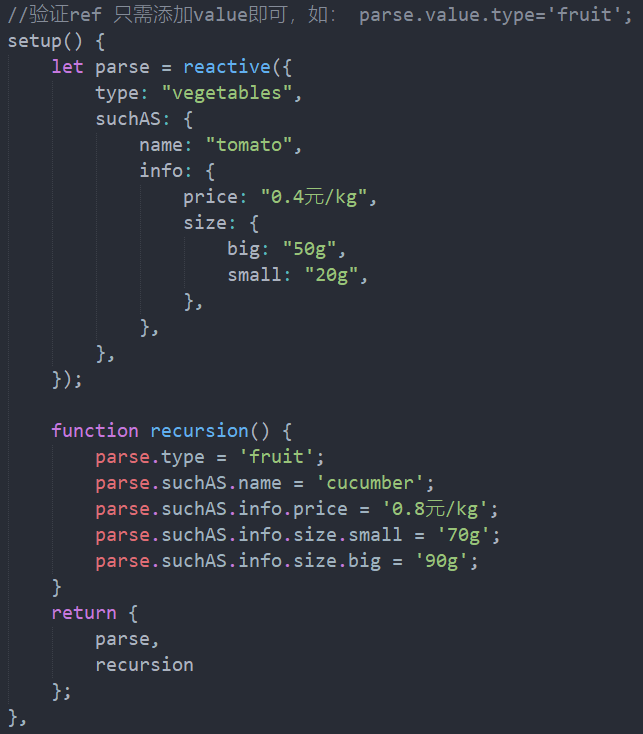
**第二次学习**

**2021.5.26 上午9:28开始 下午14:57结束 中有间断非全为学习时间**

**11. 递归监听**

通常情况下ref和reactive都会监听数据变化，每一层都包装成了一个proxy对象

验证如下，点击按钮触发recursion页面显示都会改变



数据量较大时非常消耗性能

在之前<什么是reactive>中我们知道：

reactive和ref通过递归取出参数中所有值，包装为proxy对象。递归的优与劣，涉及压栈和弹出等

**12. 非递归监听**

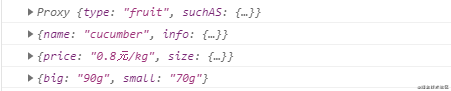
上面知道了递归监听上的种种劣势，而Vue3.0也提供了解决方案

非递归监听，即：只能监听数据的第一层。方案如下：

①引入Vue3.0中提供的shallowReactive

②改用 shallowReactive({})传递参数

观察发现，控制台只有第一层转为了proxy对象，这意味着：只有第一层的数据发生改变，才会触发UI界面的更新



而对于ref对应的shallowRef非递归监听则比较特殊

如果是通过 shallowRef 创建数据，那么 Vue监听的是 .value 的变化，并不是第一层的变化。如果想在修改其内部数据后触发界面的更新，可以调用 triggerRef 方法

①首先试引入Vue3.0中官方提供的shallowRef

②原理上与reactive相同，只是它并不会监听JSON第一层数据，而是要直接修改value的值，这样视图才会同步更新



应用场景

一般情况下 使用 ref 和 reactive 即可。只有在需要监听的数据量比较大的时候，我们才使用 shallowRef / shallowReactive

注意点：虽然他们只对第一层进行了监听，但若恰巧每次都更改了第一层数据，则也会引起下方数据和视图的同步更新，此时shallowReactive或者shallowRef就和reactive、Ref效果一模一样！

1. **数据监听补充**

通过以上这些知识点可知：

ref和reactive监听每一层数据，响应好但递归取值性能差。shallowReactive和shallowRef监听第一层（或value），性能好但更新值较麻烦。shallowRef中，为了数据和视图一致，更新值要更新整个parse.value太繁琐。

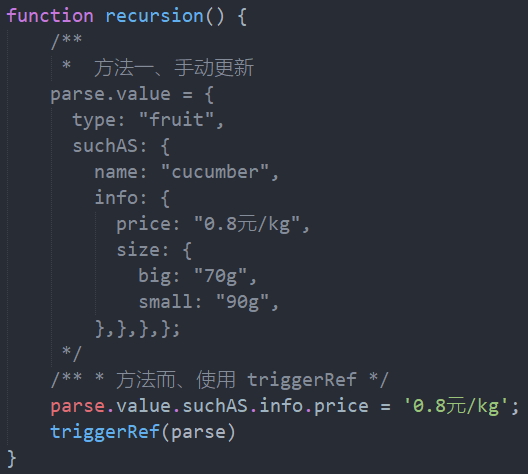
场景：若我更新数据的第三层，不整个更新value行不行？

这就用到了Vue3.0为ref准备的triggerRef

作用：根据传入的数据，主动去更新视图

老规矩，import {shallowRef, triggerRef } from "vue"，改完非首层的数据，而你使用的 是shallowRef还不想整个更新value，使用triggerRef大法，传入整个对象，就好啦

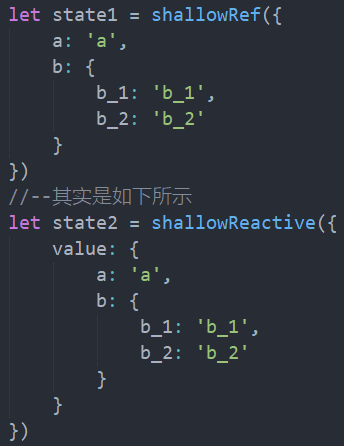
（使用reactive传入的数据，无法触发triggerRef）



1. **数据监听方式选择**

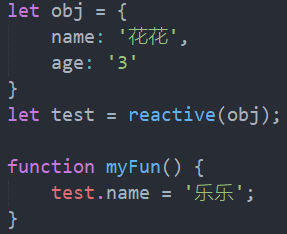
正常数据量时，通常使用ref和reactive（递归监听）即可满足业务需要。当数据量庞大且注重性能时，就需考虑shallowReactive和shallowRef了（非递归监听）

1. **shallowRef底层原理**

在看ref时，我们知道它的本质其实是reactive({value:XX})，那么shallowRef其实是 shallowReactive({value:XX})，因为通过shallowRef 创建的数据，它监听的是 .value 的变化 

1. **toRaw**

setup 中定义参数对象，在函数中直接修改页面是不会同步更新。需要利用 Ref 或者 reactive 进行包装，这样修改才生效



obj 和 test是引用关系

reactive 会把传进来的参数包装为一个 porxy 对象并返回

例子中 test 本质是一个porxy对象，而这个对象也引用了 obj

那么请注意：

直接修改obj或引用的 test 都会引起内存中数据变化

但是修改 obj因为没有 proxy监听，所以视图不会更新

再绕回来说toRaw

toRaw(state.value)，拿到响应式数据的原始数据，数据发生改变，界面也不会自动更新

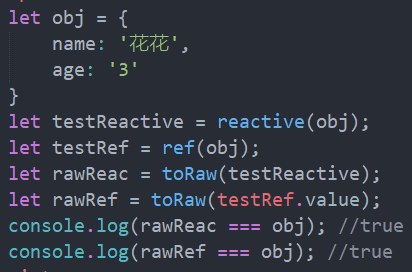
作用：返回由 reactive 或 readonly 等方法转换成响应式代理的普通对象

特点：toRaw 拿到的数据不会被监听变化，节省性能

场景：数据更改不需更新视图，为提高性能，通过 toRaw 拿到数据修改

提示：因为是原始数据，风险较大，一般不建议使用

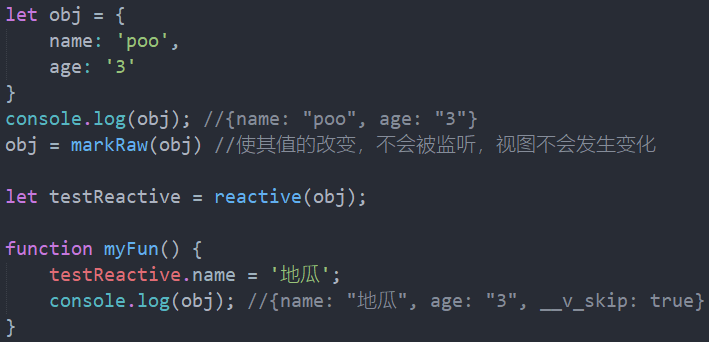
注意：若想拿到的是Ref 创建的对象，记得加 value



1. **markRaw**

作用：固定某数据，不追踪它值的变化,同时视图也不会更新。被markRaw标记的原始数据，不会被追踪变为响应式

通过控制台查看，使用markRaw的对象参数，被赋予v\_skip监听跳过标识符



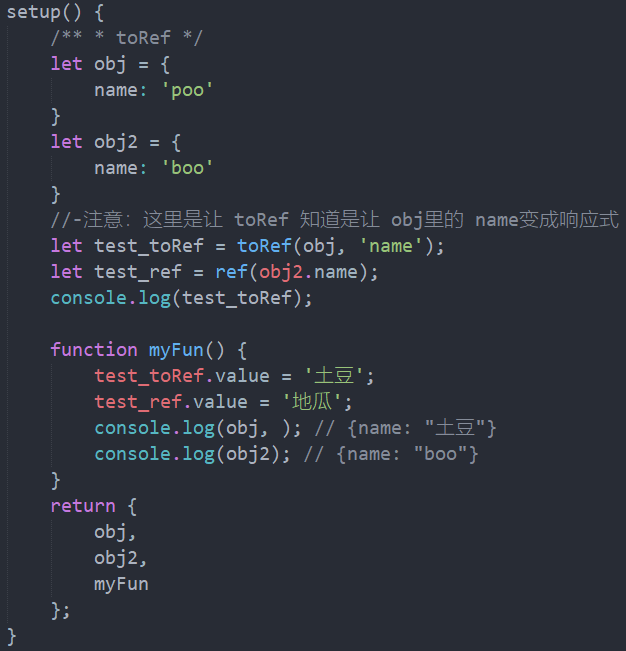
1. **toRef**

toRef和ref一样，同样也是创建响应式数据的。引用，将原始数据转化为响应式，修改响应式数据会影响以前的数据（将原始数据浅拷贝）；数据发生改变，界面也不会自动更新

ref 将对象中某属性变为响应式，修改时原数据不受影响，toRef 会改变原数据，并且 toRef 创建的数据，改变时界面不会自动更新

应用场景：性能优化

想使创建的响应式数据与元数据关联起来，更新响应式数据后，不想更新UI

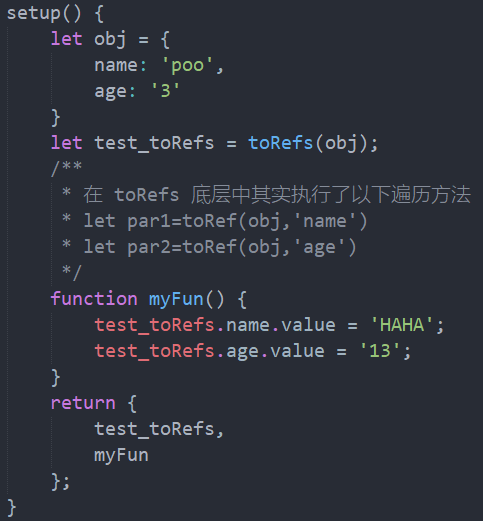


1. **toRefs**

toRef只能接受两个参数，当传递某对象多个属性值时会很麻烦

toRefs，解构，是避免 toRef 对多个属性操作繁琐

toRefs 底层原理是使用 toRef 方法遍历对象属性值



1. **在 Vue3.0 中如何通过 ref 获取元素 ？**

在Vue2.0版本内，通常使用 this.$refs.XX 获取元素

在Vue3.0中，废除了类似$的很多符号，如何获取指定元素？

根据Vue生命周期图中可知，要操作DOM，最早也要在mounted中

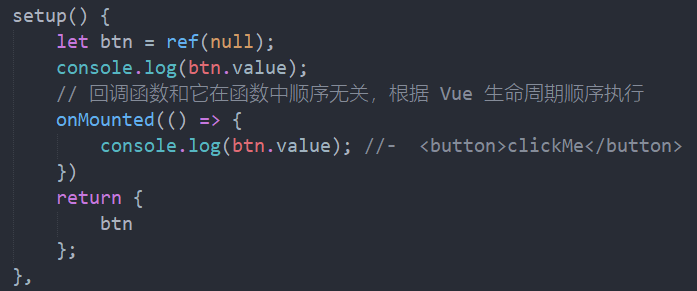
结论：

①setup 是在beforeCreate之前执行

②在生命周期中 onMounted最先准备好 DOM元素

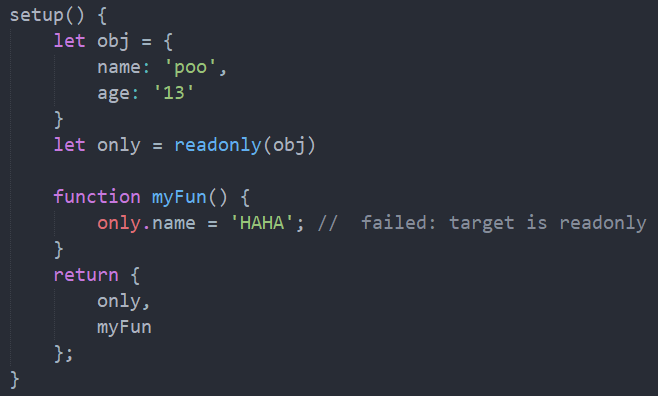
③setup中想操纵 DOM 就在函数中引用 onMounted

④Vue3.0中生命周期函数被抽离，可根据需要引入相应周期函数

**21. readonly**

Vue3.0中提供的这个API，使得数据被保护，只读不可修改。默认所有层数据都只读，若只限制第一层只读，可使用shallowReadonly。isReadonly用来检测数据创建来源是否是 readonly。若进行修改，浏览器会提示操作失败，目标只读





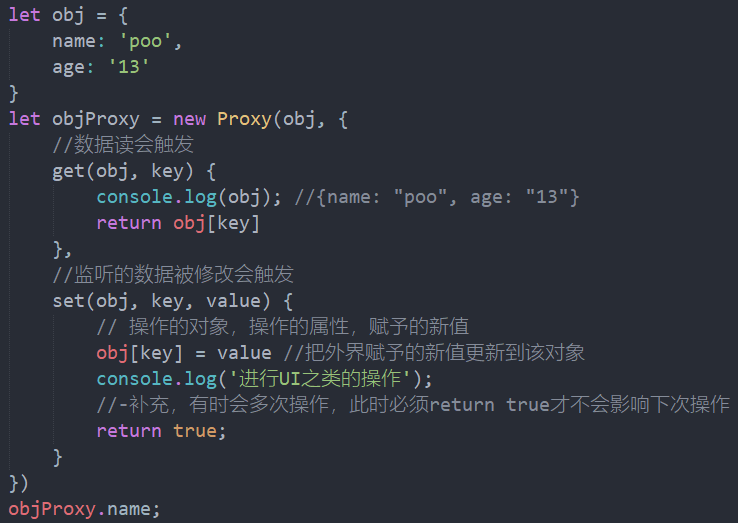
**第三次学习**

**2021.6.5 晚上18:30开始 晚上21:22结束 中无间断共计学习2小时52分钟**

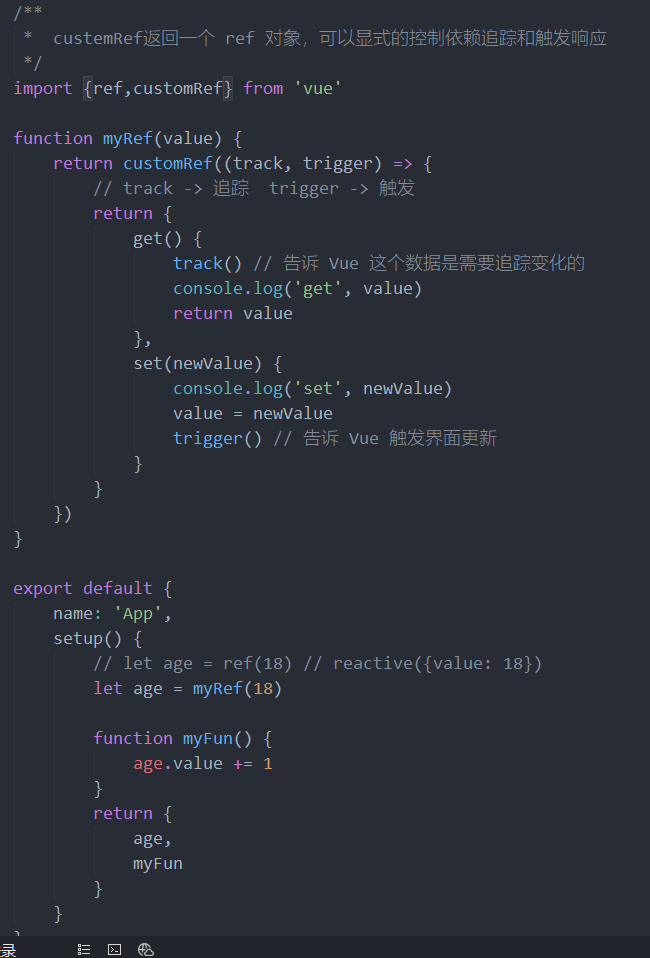
**22. Vue3.0响应式数据本质**

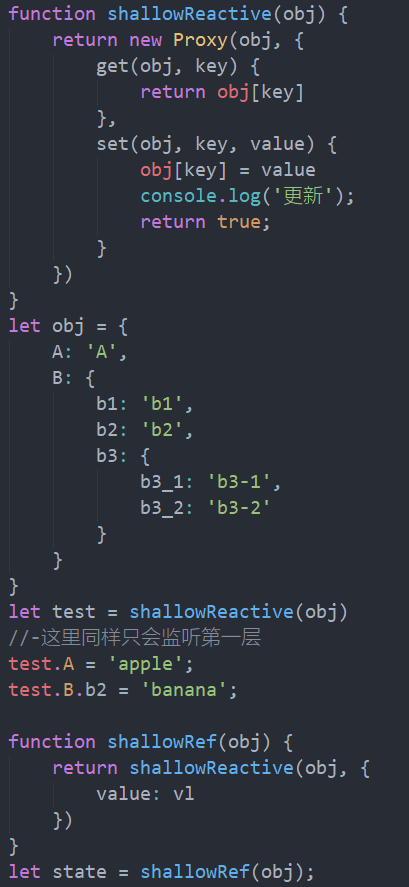
2.0中使用的 Object.defineProperty 实现响应式数据

3.0中使用的 Proxy 来实现,如下

**23. 实现shallowReactive和shallowRef**

它们二者也是通过参数传递，包装成 proxy 对象进行监听，在Proxy的set监听中，同样只监听第一层。shallowRef 只是在 shallowReactive 基础上默认添加 value 键名





1. **实现reactive和ref**

它们与上方区别在于递归监听，上方因为直接传递参数对象，所以只监听第一层。为了递归监听，那么要把数据的每一层都给包装成 Proxy对象

**25. 实现shallowReadonly和readonly**

二者区别只在于首层监听，只读拒绝修改和数据全层修改，下方实现的是 shallowReadonly。readonly实现是在 shallowReadonly 基础上移除set中的return true



学习心得：

vue3相对于Vue2来说，源码由于移除了一些不常用的API，体积更小了。其次，其新增的composition api能更灵活地组织组件的逻辑，解决Vue2的options api在大型项目中不好拆分和重用的问题。vue3 标记和提升所有静态根节点，diff的时候只比较动态节点，这也大大缩短了编译的时间。其响应式系统也大大提升，使用proxy可以拦截属性的访问、赋值、删除等操作，不需要初始化的时候遍历所有属性，另外有多层属性嵌套的话，只有访问某个属性的时候，才递归处理下一级的属性。

但我觉得虽然Vue3的性能相较于Vue2更好，但它的写法反而比Vue2的写法更麻烦了，数据每次都得申明，包括this和响应式变量写起来也更麻烦了。