# Vue.js

Vue.js是一个前端框架，他是基于HTML，css和JavaScript创建的。

Vue 的两个核心功能：

****声明式渲染****：Vue 基于标准 HTML 拓展了一套模板语法，使得我们可以声明式地描述最终输出的 HTML 和 JavaScript 状态之间的关系。就不用像JS一样一写一大堆了。模板语法会有一种调用API的感觉，通过操作模板实现效果。不用操作原生的DOM了。

****响应式****：Vue 会自动跟踪 JavaScript 状态并在其发生变化时响应式地更新 DOM。简单点说就是数据变化后DOM会根据数据变化实时更新DOM。

官网的介绍是渐进式的JavaScript框架。

关于渐进式的含义。



这里解释的就很清楚了，可以逐步集成，就是可以按照你的需求进行操作。你不需要的部分可以不用。开发时按照你的要求一步步写就行了。

其实vue常用版本有vue2和vue3。其实来说vue3会慢慢超过vue2，但是vue2还是得学，因为vue3是基于vue2进行改变的。更何况很大一部分以前的项目都是基于vue2写的。光是维护这块就得很懂vue2。Java的JDK都多少个版本了现在还在用8。所以编程这种东西，肯定是上下都得兼容，都得学。下面从vue2开始讲起。

**Vue2.X**

对新手来说是不推荐直接用脚手架的，基础语法弄懂了再用脚手架。但是基础不牢地动山摇，摇摆一阵后还是决定从头开始一点点的写，所以这里我们暂时就不用脚手架写了。从基础写起，毕竟基础不牢地动山摇。

我们现在用CDN引入的方法在原生HTML里面写vue。



这里的root容器其实就是#demo这个盒子。

            初识Vue：

                1.想让Vue工作，就必须创建一个Vue实例，且要传入一个配置对象；

                2.root容器里的代码依然符合html规范，只不过混入了一些特殊的Vue语法；

                3.root容器里的代码被称为【Vue模板】；

                4.Vue实例和容器是一一对应的；

                5.真实开发中只有一个Vue实例，并且会配合着组件一起使用；

                6.{{xxx}}中的xxx要写js表达式，且xxx可以自动读取到data中的所有属性；

                7.一旦data中的数据发生改变，那么页面中用到该数据的地方也会自动更新；

上面只是写了一个基本的vue。下面介绍一下模板语法

Vue模板语法有2大类：

                    1.插值语法：

                            功能：用于解析标签体内容。

                            写法：{{xxx}}，xxx是js表达式，且可以直接读取到data中的所有属性。

                    2.指令语法：

                            功能：用于解析标签（包括：标签属性、标签体内容、绑定事件.....）。

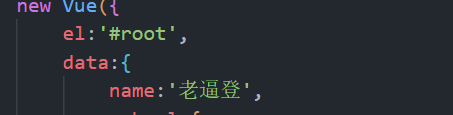
                            举例：v-bind:href="xxx" 或  简写为 :href="xxx"，xxx同样要写js表达式，

                                     且可以直接读取到data中的所有属性。

                            备注：Vue中有很多的指令，且形式都是：v-????，此处我们只是拿v-bind举个例子。

**插值语法。**

其实就是一个插值表达式{{}}通过这个东西我们可以拿到vue里面的data。通过表达式的形式放入模板中。



我们在vue中的老逼登也是成功的加入了模板中。这就是插值语法的基本使用。

**指令语法**

这个指令语法其实就是声明式渲染的一部分，他可以避免很多dom操作。这里以v-bind为例：



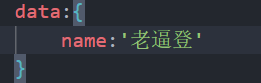


其实这两句都是一个东西:其实就是v-bind的简写。V-bind后面跟的也是表达式。我们的模板会自动给这个表达式解析出来。

**总结一下：不管是插值语法还是指令语法他都是对JS表达式的解析。模板会帮我们解析出JS表达式根据表达式内容渲染DOM。减少了直接操作DOM，这就是VUE的声明式与原生JS的命令式的优势之处。**

上面那只是体现了一个解析表达式的作用。其实v-bind还有绑定的作用。



绑定一般都是用于给表单绑定，我们这样写的话下面data里面的数据更新一遍，我们input表单里面的数据就会更新一遍。这样就实现了一个单向的数据绑定。我们借助开发者工具改一下data看看页面会不会响应。



改了他也是实现了响应。这样我们就通过v-bind把data中的数据绑给了input。



有单向绑定就必有双向绑定。上面就是一个双向绑定，他与单向绑定类似。区别在于单向绑定只能做到数据变页面触发DOM更新。V-model在实现单向绑定的基础下多了一条线。就是DOM更新后数据也会发生改变。

我们这里把中登改了开发者工具中自然也就成了中登。这就代表着我们实现了双向的绑定。其实双向绑定的本质是@input和:value。这里我们先不写，在后面vue3中再写。

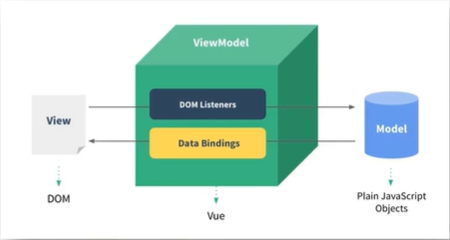
**el与data的两种写法**

上面写的el和data只是在vue2中比较好用。但是vue3甚至更早的脚手架都不会这么写了，el我们通常在vue的实例对象后用$mount挂载写。data通常都写成函数。具体为什么我们到组件那块再写。如下就是比较标准的一个data和el的写法。



**MVVM模型**

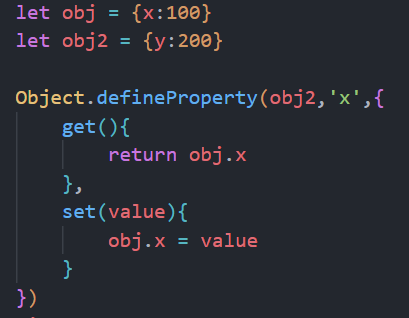
这个东西涉及了vue的响应式这个原理所以有必要写一下。如下图所示：

其实MVVM就是这三个东西，下面这个黄色的线体现的就是vue的数据驱动。数据变了模板就会更新。上面那条线就是正好相反的，模板变了数据变。ViewModel这个东西就相当于一个中介。通过他把数据和模板联系起来。

**关于底层的响应式原理**

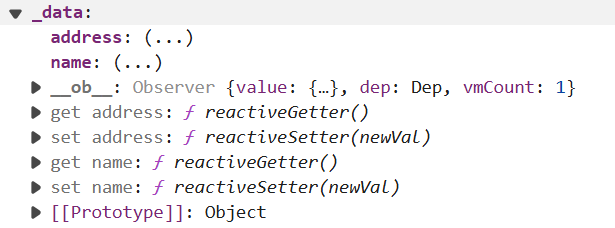
上面既然说了响应式，那就顺便把vue2的响应式给讲清楚了。其实vue2底层是用defineproperty做的数据代理。这个东西是JS里的东西之前没写过，在这里先讲一下他是干啥的。他本来是给对象添加属性的东西。

这就是他最原始的用法。因为其set和get两个东西是函数。所以就给了人操作空间。所以就出现了代理这个东西。代理说简单点就是一个人把另一个人的事做了。下面是一个代理的代码。

这段代码里obj2多了个x属性，但是x属性不管是读还是改。操作的其实都是obj1中的数据，这也就是事件代理了。

**Vue中也正是用这个办法把\_data里的数据放到实例对象也就是VM上，由实例对象代理\_data数据实现模板更新，同时通过set和get这俩玩意操纵data这个被代理的数据。从而实现响应式。**

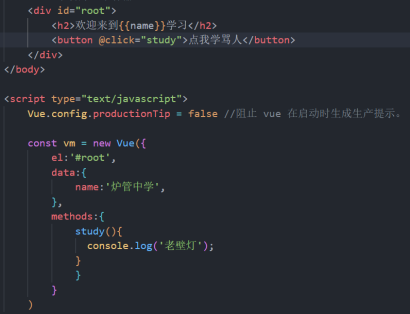
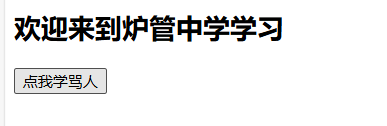
我们用这段代码对响应式做一个具化的解释。我们在控制台打印的实例化对象VM中有这两个属性。里面...其实就是我们代理的表现。Data里面这两个数据其实就是被代理者。VM则是代理者。



在底层VM正是通过get和set对\_data里面的数据进行更改实现代理。大概就是这个样子。

**关于事件**

既然说了vue用了声明式不用操作dom，那对于dom事件vue自然也有他的方法。来实现，其实就是一个指令语法v-on简写为@。下面以点击事件为例，对事件进行一个操作。

通过这段代码我们实现了点击事件。

我们每次点击控制台都会打印老逼登说明我们的事件是好用的。

细节：函数method里面我们是可以传参数的，不传参默认就是e，但是如果传了参数还要拿到e就要在传参的时候用$event这个占位符实现占位。从而达到有e的效果。

事件其实是有修饰符的，修饰符使事件实现功能更方便。

Vue中的事件修饰符：

                        1.prevent：阻止默认事件（常用）；

                        2.stop：阻止事件冒泡（常用）；

                        3.once：事件只触发一次（常用）；

                        4.capture：使用事件的捕获模式；

                        5.self：只有event.target是当前操作的元素时才触发事件；

                        6.passive：事件的默认行为立即执行，无需等待事件回调执行完毕；

我们这里就简单用once做一个示范。



一旦我们这么写。我们无论怎么点这个按钮他都只会说一句老逼登了。。

其他事件如keyup与这些都类似也就不写了。

**计算属性（computed）**

与data，method是同一级别的东西。通常他用于对data里的数据进行计算来生成一个新属性。来避免我们在{{}}这个插值表达式里写一大串的东西。下面我们将两个代码进行对比，相信一下就能看出计算属性的作用。

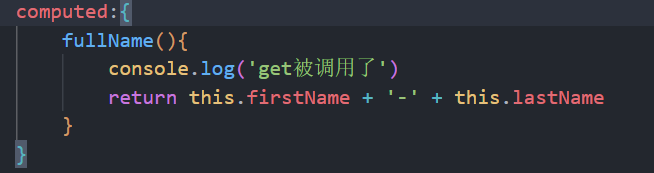


第二个图其实就是计算属性的语法。我们在计算属性里面通过getter和setter实现一个计算。

Getter与setter调用时机：get在渲染DOM时就会调用一次。之后再调用就是用来计算计算属性的属性发生变化时会被调用。Set则是计算属性被修改时调用。这个是一个重要知识点。

**计算属性的简写**

计算属性中的set很少用，当set用不到的时候我们喜欢直接简写成一个函数。



函数里面的写法其实和get的写法一模一样了。

**Watch监视**

这个东西和computed，data，method也是一级的通常用于监视数据的变化。

我们给一个数据绑定了监视属性。一旦他发生了变化那我们的watch里面的handler就会被调用。Handler有两个参数。新值和旧值，通常开发的时候我们都用的新值。同时里面有immediate：true和deep:true这种的配置项有很多，这两个比较常用。前者用于数据第一次出现不管变没变调用一次函数。后者用于对对象数据监听，监听内部属性的变化。

**其实watch也有简写**

他的简写也是函数形式的，当没有deep这种配置项时可以用简写形式。写成一个函数。



其实就是把handler写在后面了。

**v-bind绑定样式**

这就是绑定样式的一个规则。

**条件渲染（v-show,v-if）**

就是根据条件的真假来判断DOM元素是否渲染。

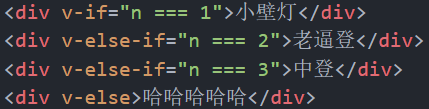


这种情况这个标签是不会显示的。

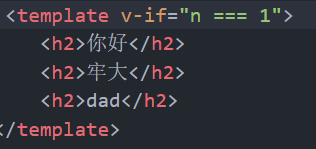


相反的这样写这个标签就是会显示的了。

V-if与v-show的语法相似，只不过多了个if else判断。唯一区别特别大的地方是 v-if直接就把标签删了。也就是没有这个dom结构了，而v-show只是用display：none隐藏了。Dom结构还是存在的。

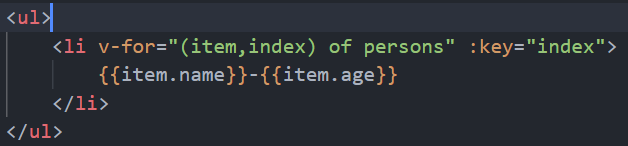


细节：上面说了v-if会直接把dom结构删了。为了保持结构稳定我们通常在v-if的标签外放一层template标签只作为稳定结构用。Template标签本身并没有任何意义。



**列表渲染（v-for）**

V-for能遍历的是数组，对象，字符串，以及指定次数（vue3移除）。



大概语法如上。只要是数据data里的可迭代对象我们v-for都能遍历。通过这个方法我们可以批量生成dom结构。

V-for中的key也是一个关键角色。

其作用主要存在于DOM更新时的diff算法中。DOM更新时diff算法会根据key的旧真实dom和虚拟dom比较进行比较。如果数据相同采用原dom。如果数据不同，则根据key将虚拟dom和原真实dom信息比较生成新的真实dom替换原来的真实dom渲染到页面上。如果虚拟dom没找到相同的真实dom就生成新dom添加到页面上。这也造成了如果有打乱顺序的操作那我们用index作为key就会出现dom里的数据对不上号的情况。尤其是输入框里的数据会出现驴唇不对马嘴的情况。所以我们的key通常用ID，电话号码这种唯一标识。尽量不去用index，但是如果不出现打乱顺序的操作，用index也是无所谓的。

**关于vue的数据监测**

其实这个数据监测，属于是vue响应式的一个遗留问题。如果追溯他依然会追溯到我们vue2底层使用defineproperty实现响应式这里。这里有一个坑就是，我们在vue实例化对象vm创建之后。如果我们通过事件来给data添加一个新的数据那么会有一个非常逆天的现象。就是我们添加的这个新数据，没有响应式。这种问题在vue3里不是问题。直接加一个ref套上就行了。但是在vue2里可能要麻烦点。想解决这个问题，首先要知道vue2是通过defineproperty里的setter属性来监视数据的。而我们的这个setter和getter是在vm实例化对象一创建就出现的。这也就导致了我们后面添加的数据不会有setter，没有setter的话我们的vue就监测不到。就是这个问题。所以在vue2他提供的是vm.$set(target,name,value)或Vue.set(target,name,value)\*\*通常我们在vue实例化对象里面都用this.set(target,name,value)。

还有一个问题就是，我们的对象数组里对象内属性变化他监测不到。其原因可以理解为我们的setter监测数组的是地址。数组内对象变化了。地址没变。所以他监测不到变化。所以vue把原生的数组操作封装了一下。名字还是原来的名字

Splice，slice，concact这种的。我们要改数组里面的数据我们就用原生数组自带的方法。数据代理其实是数据劫持的一种方法。

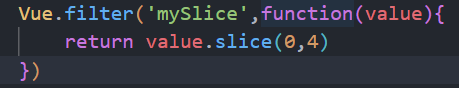
细节：vm.$set,或者vue.set,这两个方法添加响应式数据，没有办法添加到根数据（也就是data上）这点注意。面试可能会问。

**关于绑定获取表单事件补充**

之前我们用v-model对input的输入框text进行了绑定。但需知。不是所有的表单他都像text一样有输入值一样。很多表单像radio，checkbox，select ，option...这些他没有数据。但是我们需要用v-model实现数据驱动，这种时候。我们需要给表单带上value属性。具体属性与数据对应绑定关系如下。



**~~过滤器~~**

~~这个东西在vue3里已经不存在了但是还是想写一下。因为其实也没什么东西。Vue实例化对象里面有一个filters和computed，data这些平级。我们只要在filter里面写函数，他会接到一个value函数。对value进行一系列操作。然后在插值表达式里用~~这种语法给他截取就行了。如果想要全局使用就这样。

**关于内置指令**

除了之前的v-for,v-bind,v-model,v-on外还有几个比较简单的内置指令。

v-text这个指令可以将表达式解析到标签里，但是不能解析HTML标签。

V-HTML这个就可以解析HTML标签。非常的危险，容易造成XSS攻击。不要放在用户输入的数据上，不然安全性问题就会贼严重。

v-cloak指令（没有值）：

                        1.本质是一个特殊属性，Vue实例创建完毕并接管容器后，会删掉v-cloak属性。

                        2.使用css配合v-cloak可以解决网速慢时页面展示出{{xxx}}的问题。

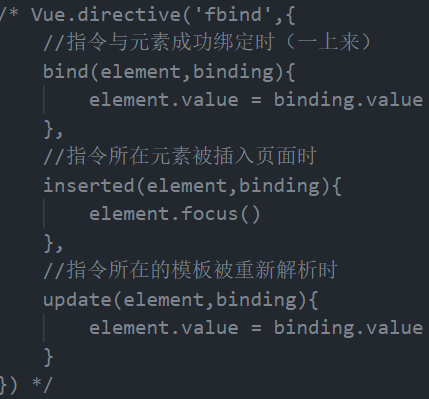
v-once:写了v-once的标签渲染一次后就死了之后数据变化响应式也影响不到他。

V-pre跳过没有指令的标签加快运行速度。

**自定义指令**

内置指令虽然多，但总有不够用的时候，这个时候自定义指令就发挥了他的用处。

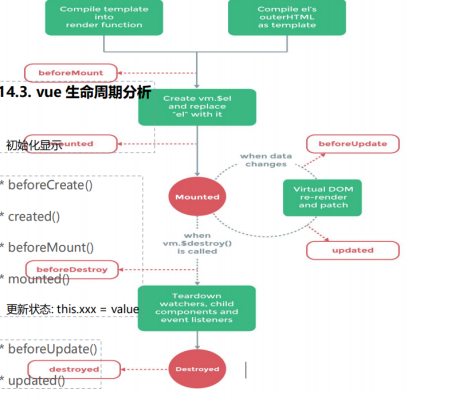
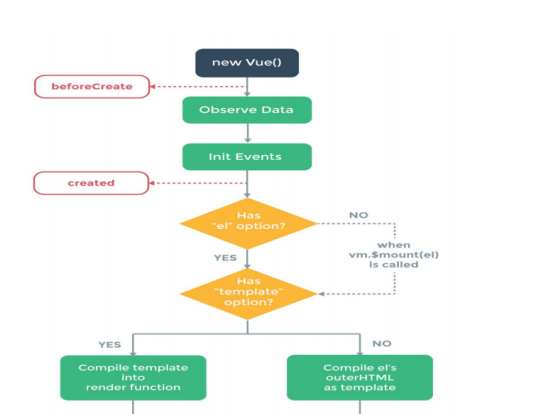
自定义指令是directive与computed,data,methods同级，里面是一个对象。如果简写就是一个函数。他可以接收到参数element，binding。Element为绑定的dom元素，binding为dom元素绑定的值。通过对这两个参数进行操作达到想要的方法。其实他内部分为三个阶段。bind,inserted,update三阶段，分别代表绑定，插入页面，更新dom三个阶段。在对应时段做对应的事更精准，更细节。



全局自定义指令可以在所有vue实例用。局部只能自己这个vue实例能用。就这么个区别。

**关于Vue 的声明周期**

人这一辈子有很多重要的事从出生到入土，Vue从出生到入土也有很多重要的事情：这也就是vue所谓的生命周期。官网上提供的生命周期流程图。



生命周期具化到代码中就是几个钩子。这里把常用的几个写一下。剩下的在特定情况下出现的钩子在下面写。其实常用的就是这四对八个。

**Beforecreate：**生命周期、事件有了，在这之前数据代理并没开始。也就没办法通过vue的实例化对象VM来访问代理对象data。

**Created：**在这时候数据监测代理都有了。我们已经可以用vm访问data里的数据了。

**Beforemount：**在这之前虚拟dom编译完了。但在这里面进行dom操作操作的是没被vue编译的虚拟dom所以操作无效。

**Mounted：**在这时候虚拟dom已经编译完了。可以进行dom操作且有效，但通常不在这个位置进行操作，而是在updated里面。这里多是对自定义事件绑定，网络请求，消息订阅等事情进行初始化。

**Beforeupdate：**这时候数据监测到了并且已经实现了更新，但是模板并没有更新，也就是响应式没完全完成。

**Updated：**这个时候diff算法已经比较完了，真实dom生成了并且放在了页面上，也就是说数据和模板都是新的。

**Beforedestory：**在组件销毁之前做的事，解绑自定义事件，消息订阅，关闭定时器等操作在这一步进行，在这一步更改数据是有效的但是不会触发模板解析与更新，也就是没有响应式，无法调用update。

**Destoryed：**这一步就是已经没了，都没了也没啥操作了，处理后事基本都是死前也就是beforedestory那里处理掉了，所以这块就没啥事了。

细节 ：常用的生命周期钩子：

 1.mounted: 发送ajax请求、启动定时器、绑定自定义事件、订阅消息等【初始化操作】。

 2.beforeDestroy: 清除定时器、解绑自定义事件、取消订阅消息等【收尾工作】。

关于销毁Vue实例

1.销毁后借助Vue开发者工具看不到任何信息。

 2.销毁后自定义事件会失效，但原生DOM事件依然有效。

 3.一般不会在beforeDestroy操作数据，因为即便操作数据，也不会再触发更新流程了。

**关于组件化**

组件是vue中非常重要的一部分，说是vue的核心一点也不过分。组件化极大的提高了代码的复用性。开发和维护的效率都变得非常高。组件分为单文件组件和非单文件组件。

**非单文件组件**

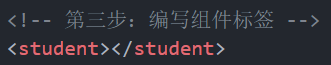
所有的组件都在一个文件里，我们创建好几个组件然后在vue实例里面注册，然后再在模板中使用。注册完后再在模板中使用。

这是我们创建的一个组件用vue.extend创建。注意这个vue.extend会给我们创建一个vuecomponent的函数。每个组件的vuecomponent都不一样。在我们用标签格式使用组件时这个vuecomponent就会发力new一个vuecomponent。

其实component和vue实例之间底层通过原型联系起来了。这样使组件能通过原型访问vue上面的$..这种方法。

进行注册后就可以使用了。这种是局部注册其实可以全局注册的。

这就是全局注册的语法。

最后我们在模板里用标签的形式写就完成了组件的使用。

这样写的话需要有一个template属性，因为都在一个文件里，组件的结构就写在template属性里面。就像这样的。template相当于组件的结构。

**之所以称为非单文件组件是因为他的组件与使用组件的vue实例对象都放在了一个文件里并不是一个组件一个文件。这种写法不是很符合组件设计的初心所以开发时很少会用这种组件。**

**另外组件也是可以嵌套的，就是给子组件再注册组件这种，很类似于分封制。**

**还有一点很重要就是组件里面的data是以函数的形式存在的，return的返回值是一个对象。如果我们不以函数的形式写组件里面的data就会导致一个很逆天的问题，就是我们所有组件里的数据都是依赖于一个data，这种情况下组件数据就是共享的。一个组件里的数据改了其他组件里的数据都得改。很不方便，所以我们用函数的形式返回对象，这样我们就可以实现每个组件的数据独立的。**

**组件名是为了在开发者工具里看的方便。**

**单文件组件**

和他的名字一样一个组件就是一个文件，我们在单文件组件里组件的写法和非单文件组件里面一样。且单文件组件的CSS，html，JS三部分，vue提供了三个特别的标签来使写代码更容易。

这个标签里写HTML

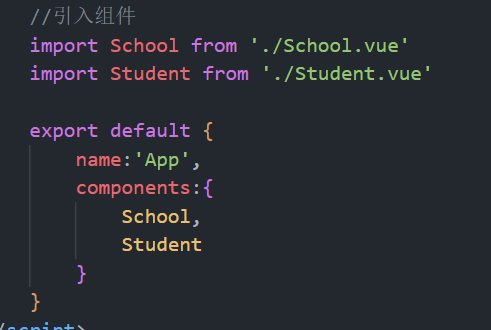
这个标签里写JS

这个标签里写CSS

但注意导出，另外有一个小技巧就是组件前面的vue.extend可以不用写。直接写成一个对象导出也可以。



导出后最后的使用都是在APP.vue里面引入注册，的。App.vue就是跟组件。最后把跟组件交给index.html用script标签引入就可以使用了。

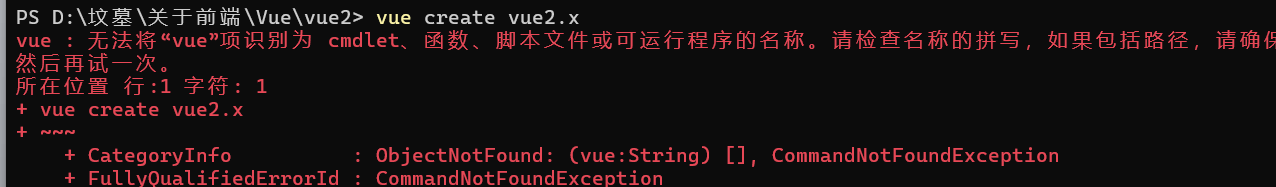


**脚手架创建工程**

打开cmd输入以下命令。



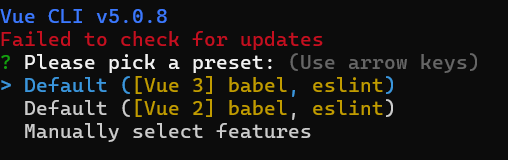
我现在在vue2这个文件夹里用vue-cli就是工程化开发脚手架创建了一个项目。当我们敲下回车他就会给我们创建一个vue2项目。

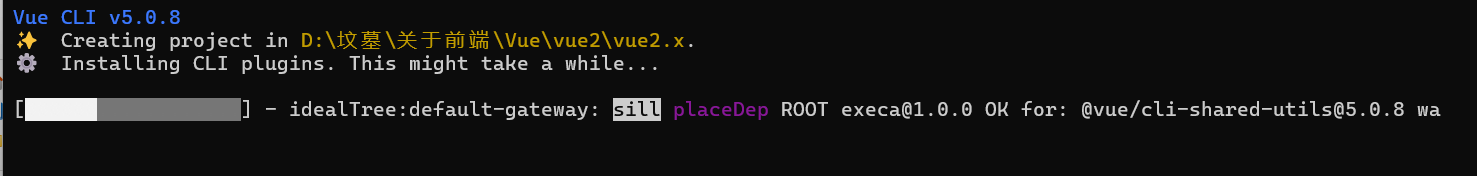


这里报错了是因为没安装vue-cli。我们直接npm安装vue-cli就行了。

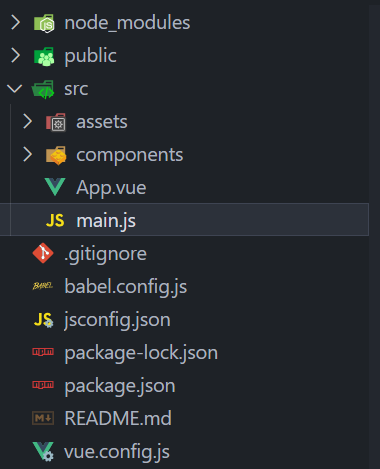


Vue安装完毕就可以执行命令了。

会出现一个交互式的创建。这里根据需要点就行了。下面是我选好了执行创建的画面。



这里可以看到我们已经有这个文件了我们在vscode里面打开就能看到我们创建的工程文件。



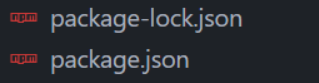
下面对脚手架创建的解构进行一些分析。

这个文件主要是对我们npm包下载的东西进行管理用的，我们npm下的东西都会储存在里面。

这个文件夹里储存的是静态资源，静态资源的意思就是不参与打包，像图片，视频，音频和index.html这种东西都放在这里面。

编写代码主要是在src里面进行编写的，包括组件，以及后面的路由都是在这里面编写的。里面的asset也是放资源的不过他会被打包。根据开发需要来选择是将资源放入public还是src。这个就是入口文件，万物开始的地方。我们的APP.vue最后就是给他的APP.vue可以理解为所有组件的整合。

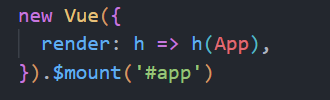
这个就是Git的支持文件。毕竟开发肯定要用Git

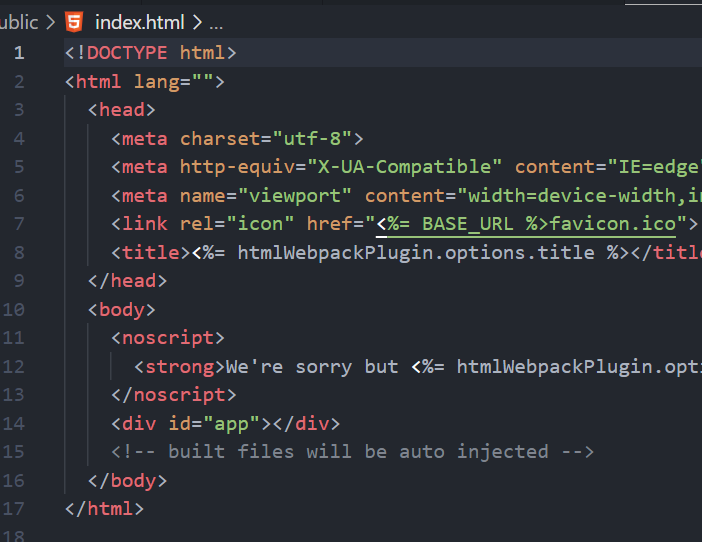
这个是ECMAscript的支持文件。在vue中开发也会用到很多ES6的语法，有了这个东西就不用担心vue不认识ES6了。这个就是JS的支持了。这俩就是关于我们node下载的包。Lock为版本锁定，防止版本更替给项目带来影响。

这个东西更是重量级，他是给vue设置默认配置的，因为vue的默认不是适合所有项目开发的，我们可以通过这个配置来设置vue使他满足我们实际开发的要求。要不然就得去改源码了。

这玩意相当于项目里的记事本，对一些变量的定义组件的定义这种东西就放在这个里面。

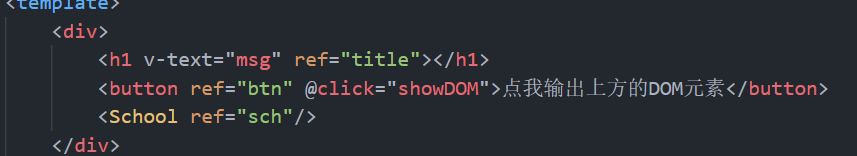
**注意：**Vue脚手架其他地方都很好懂的但是唯独main.js里面的这块代码与不用脚手架不一样。

。$mount前面在写el和data的两种写法的时候说了。这个render其实就是一个函数。其实如果把脚手架构建的这个初始环境翻遍了就能发现他public下面的index.html透露着一种诡异的感觉。

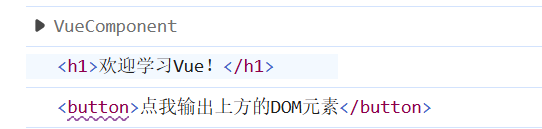
毕竟是一个主要文件并且容器app还在里面，他却没有任何引入。那如何把组件给挂载到容器里呢？这其实就是render函数的作用。把组件挂载到app这个容器上面。这个函数的h接收到的就是容器的element元素。细了就不说了，说多了更看不懂了，自己体会就行了。

**ref属性**

这个属性在vue2里是给标签打标记来拿到dom的。原生JS如果想拿到DOM中的标签需要用复杂的操作。先要打id再document.getElementById这种方法拿。在Vue2中这种方法被ref取代了。想拿到DOM元素的话，比较好用的方法其实就是给标签打上ref然后在vue实例对象中用this.$refs.ref名拿到。代码就是下面这样。





这样就可以拿到dom元素了。

**组件通信部分**

组件化开发是vue中很重要的部分，而组件之间要有数据的交换，不然会导致一个组件的数据只能在一个组件里面用的情况。无法实现数据共享必将给开发带来很多的不便。Vue肯定想到了这种情况所以组件通信就是为这个玩意准备的。组件通信方法有好几种，下面一一进行解释。

**Props**

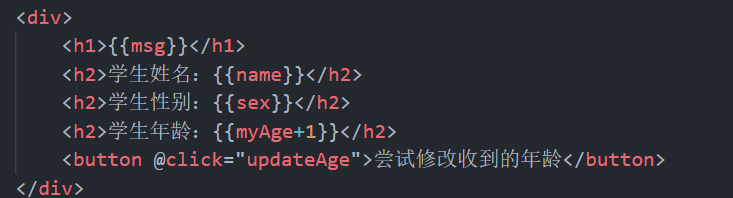
他可以用于父传子和子传父，也可以用于兄弟组件,但主流用法其实就是父传子，子传父有其他方法，在下面会进行一个介绍。现在先对props的用法讲解。代码如下：

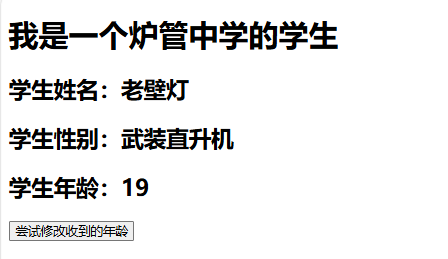


我们在父组件中只要在子组件的实例化标签上加入我们需要传的数据。

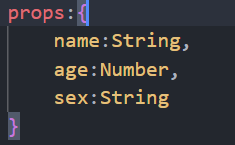
子组件就可以通过props接收到。并且能用在模板中，这样就实现了组件通信的父传子。





他可以接收到数据，数据也可以使用，也可以更改就跟自己的一样。

Props功能不止这一点，他还可以规定参数的类型。这种时候就不能用简单的数组写了，就要用对象方法写。

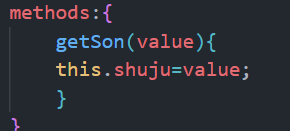
这种情况就可以**规定传参数的类型**，如果不按规矩传控制台就会报错。

**他还能限制参数是不是必传，和设置对象默认值**，用对象套对象的方法就行了。



Props的父传子大概就是这么用的。

Props的子传父。这个东西很巧，个人感觉跟JSONP一样是野路子。

我们在父组件里面定义一个方法。



把他传给子组件。

子组件用props接住了。

然后通过点击的方法触发父亲传来的方法。方法触发后就可以在父组件接到子组件的数据。



点击之后父组件就完成了更新。

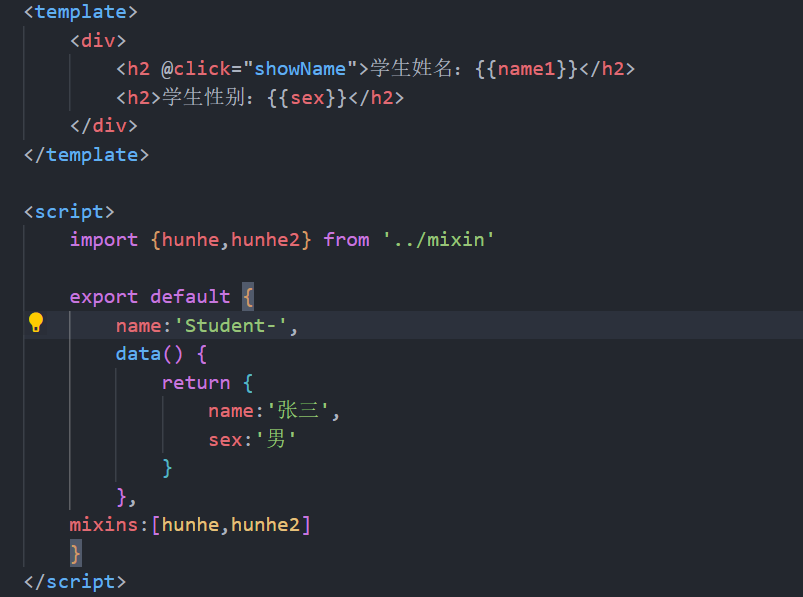
我们就是通过函数把子组件的数据传给了父组件，就是这么简单。

**mixin混入**

这个玩意就是帮助实现数据，方法的一个封装然后用到组件中的，实例代码如下：

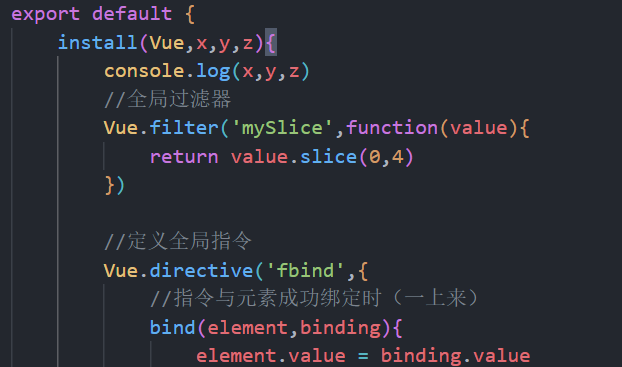
这是我们定义的Mixin里面写了一些方法和数据。

我们只要在组件里面引用后利用mixins属性注册我们就可以在组件里面使用数据与方法了。

注册完毕后就可以在组件中使用我们没有写的数据。这本质也是提高代码复用性的方法，但是有组件通信就显得这个东西鸡肋了。用的可能就会比较少。

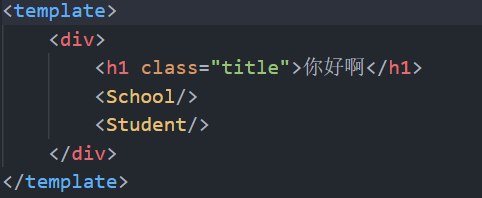
**关于插件**

其实插件就是用来定义全局注册属性的，像全局mixin，directive自定义指令，filter过滤器，或者是对vue原型进行操作的东西。都写在一个插件的JS文件里以对象里套函数的方法写，然后再进行导出。之后再在main.js这个入口文件里vue.use(插件名)就行了。

这就是关于插件的的部分代码。

**scope**

这是一个组件里的东西。他是存在于组件三个标签template，script，style中的style上。他存在的目的其实就是为了防止CSS样式冲突的问题。就是父组件子组件都写了同一个id，这时候如果我们加入样式就会产生样式冲突。可能我们的子组件会使用父组件的样式。这种情况是我们不希望看到的。

父组件

子组件。

两个组件里面都有一个叫title的class名。如果我们不写scoped，会导致父子组件通用子组件的样式。

而写了scope后就没有这个问题了。每个组件里面的style都只给自己这个组件用，别的组件想用也用不了。

**组件的自定义事件**

[之前所用的@click点击事件@keyup这种事件都是vue内置的事件。我们通过点击或者按键盘就能触发。而自定义事件的格式也是这样的，只不过触发要用到this.$emit这个东西，正是这个东西可以帮我们实现组件通信。通常我们用他来实现子传父，比如说他不但可以触发事件还能携带参数。这个参数就是我们子组件要传给父组件的内容。下面是代码示例。](mailto:之前所用的@click点击事件@keyup这种事件都是vue内置的事件。我们通过点击或者按键盘就能触发。而自定义事件的格式也是这样的，只不过触发要用到this.$emit这个东西，正是这个东西可以帮我们实现组件通信。通常我们用他来实现子传父，比如说他不但可以触发事件还能携带参数。这个参数就是我们子组件要传给父组件的内容。下面是代码示例。)

我们跟student组件绑定了一个事件，这个事件一旦触发getTudi这个函数就会调用。

我们在子组件里用this.$emit这个方法触发了事件，并且把子组件要传给父组件的内容传给了父组件。



父组件接到了刀哥名，并且把他用变量存了起来。这样我们用的时候调用变量就行了。

这样就实现了一个简单的子传父。

其实还有一种方法绑定自定义事件。



在mounted挂载这个声明周期里面用ref拿到组件的dom然后用$on绑定自定义事件，前面的是事件名，后面的是事件触发后执行的函数。而子组件里还是用那种方法触发。。

用完的时候通常在beforedetory钩子里面解绑自定义事件防止组件都没了事件还在这种浪费性能的情况出现。

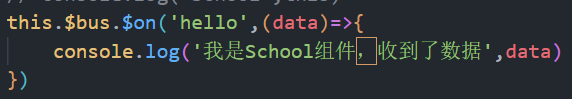
这个方法其实是为了后面的全局事件总线做铺垫。

**全局事件总线**

这个东西和上面的第二种自定义事件很像。只不过他是放在vue上的而不是放在某个组件上的。我们要在vue的原型上面添加一个属性让这个属性有vue的实例化对象的所有属性。



在main.js入口文件里进行这个操作安装全局事件总线。又因为所有的vuecomponent都能访问到vue原型上的属性，所以所有的vue组件也都能访问到$bus这个东西。



这样我们所有组件之间都可以通过总线通信了，接受数据的on，提供数据的emit激发。并且提供数据的销毁时记得$off解绑。

**消息订阅与发布**

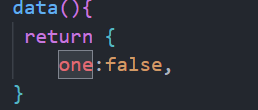
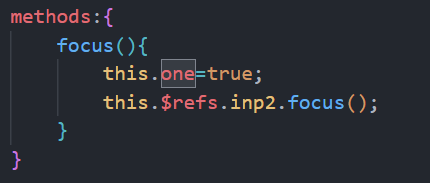




这个东西与事件总线的原理一样，只不过是借助JS的第三方库实现的。发布相当于$emit，订阅相当于$on。取消订阅相当于$off都是一样的原理，就不过多赘述了VUE2里面使用这两个里面的哪一个都是可以的。

**$nextTick**

这个东西主要解决的是vue模板更新产生的一个漏洞。Vue模板的更新默认是所有数据都改完了准备好了再更新。这时候就会出现一个顺序问题。





在这两块代码里的逻辑是这样的：开始的时候不会显示这个表单，但是如果我们的one变成true就会显示表单。那我们就想在表单显示的同时让他聚焦。但是如果按上面这么写出现后不会聚焦。正如上面说的，所有数据更新完后才更新模板，模板里根本没有这个东西怎么聚焦。Vue为了解决这个问题就提供了$nextTick这个API解决这种问题。



他会默认模板更新完毕后进行回调，这样就解决了模板没出现就修改模板导致修改无效的问题。

**关于代理的配置**

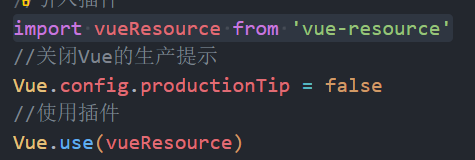
首先了解一下何为代理，代理就是由一个人替另一个做事。出现代理的原因是浏览器的同源策略在上面跨域时就说了，协议域名端口必须全部一样才能请求，不然请求不到。后端有cors前端有JSONP这种野路子，vue用配置代理服务器的方法解决这个问题，只要在vue.config.js里面配置代理服务器，之后请求只要向代理服务器发请求，代理服务器自然就会把请求发送到请求的地方。

我们需要在VUE.CLI的文档里面找配置。

官网里的这里自会告诉你怎么配置。

**Vueresource**

这是一个插件，既然是插件那使用就分为三步，安装，引入使用。

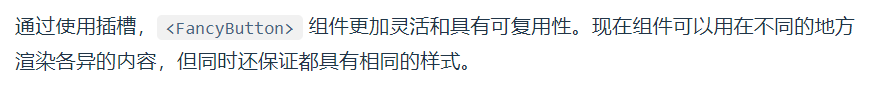
这样就实现了使用。他的作用其实是和Axios一个功能。在vue上多了一个$http属性用来发网络请求。



可以这么理解：this.$http=axios。

**关于插槽**

插槽个人感觉也算是组件通信的一种方式。他是父组件对子组件的结构进行管理的一种方式。



父组件向子组件传递带数据的标签，当一个组件有不确定的结构时, 就需要使用

slot 技术，注意：插槽内容是在父组件中编译后, 再传递给子组件的。

这是官网对插槽作用的概述。其实就是使组件更加灵活了。插槽说白了就是一个萝卜一个坑。

插槽分为默认插槽，具名插槽和作用域插槽，下面进行一一的解释。、

**默认插槽**



就是在组件标签里面加一点结构然后再在子组件里用slot组件接收插槽传来的结构。



同时也可以加一点默认值。

**具名插槽**

具名插槽这个东西主要解决的是多个插槽位置问题。Slot如果没有标识，就不知道到底哪个部分给哪个插槽。这时候需要给插槽加一个name属性，像这样：

然后在父组件处加上v-slot:name或者slot=“name”来决定放在哪。其实v-slot:和slot=“”效果是一样的，只是写法不一样罢了。

**作用域插槽**

作用域插槽这个东西其实就是子组件把数据交给父组件管理的一个方法。

子组件把数据传给父组件

父组件以对象的形式接收到了子组件传来的数据。根据数据可以更好的生成结构放在插槽里。从而传给子组件的插槽更加完美。

**VueX**

概念：专门在 Vue 中实现集中式状态（数据）管理的一个 Vue 插件，对 vue 应

用中多个组件的共享状态进行集中式的管理（读/写），也是一种组件间通信的方

式，且适用于任意组件间通信。

现在vue官方已经推荐pina了，因为pina比vuex更简单更强，vuex基本已经被pina替代，但是vue2项目中用的都是vuex所以我们现在讲一下怎么用。

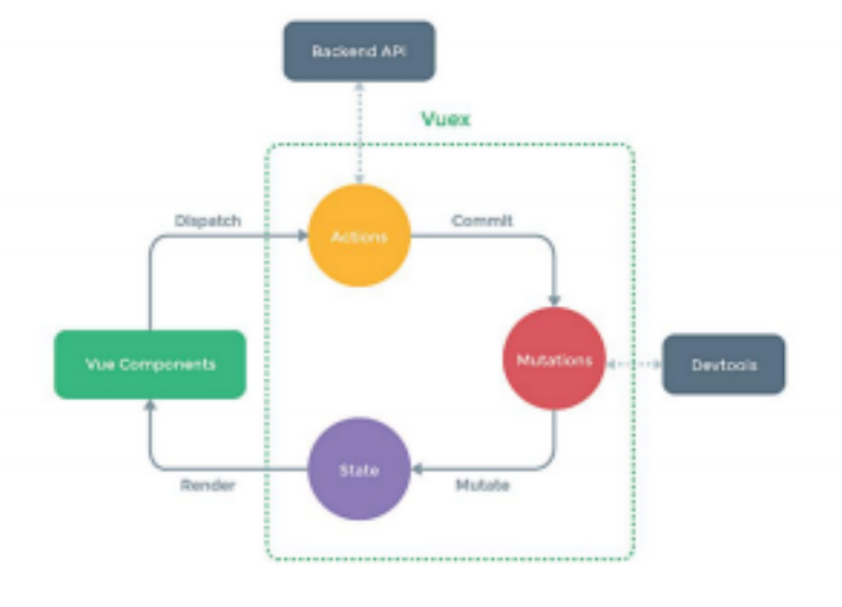
Vuex的使用场景。

1. 多个组件依赖于同一状态

比如，组件一要使用刀哥这个对象，组件二也要使用刀哥这个对象，如果只是两个组件使用刀哥可以使用组件通信的方法解决但是如果是十个二十个组件，单纯通信会变得很复杂。所以数据放在vuex里使用，会方便很多。

1. 来自不同组件的行为需要变更同一状态

比如，组件一想改刀哥，组件二也想改刀哥，那直接放在vuex里，到时候不管哪个组件，改vuex里的数据就行了。

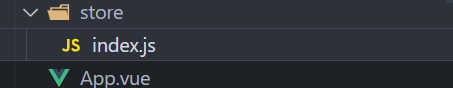
这是vuex的原理图。

**创建vuex**

我们先创建一个vuex

通常在项目里不管是vuex还是pina我们都放在src文件夹下建立的store文件夹的index.js里面。

这样写就安装了store，这里解释一下为什么不跟main.js里面vue.use()：其实是一个先后顺序的问题，我们的vuex 的使用要在vue实例对象vm之前，如果都在main.js里引入一定是在vm实例化对象之后。所以都放在store里面。

之后在用import引入main.js，最后在vm实例对象里面使用就行了。

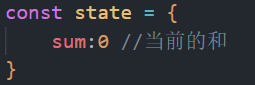
这里用了简写形式。完整的写是这样的。

这里触发简写了而已。

Vuex的核心API有五个，分别是：Actions,mutations,state,getter,modules。下面一一进行讲解。

**State**

这里以对象的方法写了一堆数据

这就是state。读取数据的时候用的是this.$store.state.sum这样的写法。

模板里写可以省一个this。

**Acions**

这个东西个人的理解就是一个中转站，他的作用就是对修改数据的请求做核对加工。加一些业务逻辑都是在这里面写，像延时修改这样的。他能收到contex上下文，和一个value,context上下文能dispatch但通常不用，通常用的都是commit传给mutation。

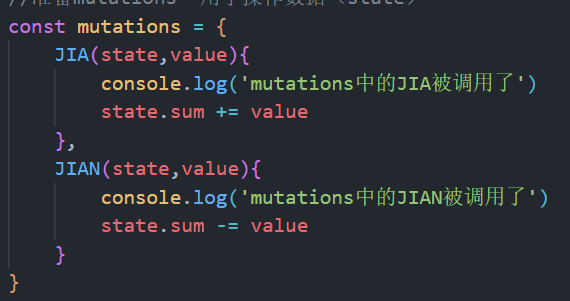
是对象套函数的方法实现的。

在组件里面使用要用this.$store.dispatch(名字,value)



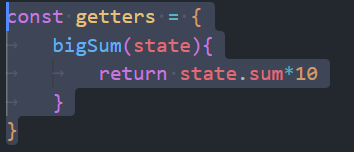
**Mutations**

修改state里面的数据，他也是对象套函数的方法实现的，他里面的函数能收到的参数是state和value。通过把value进行加工传给state就可以实现修改数据。

在组件里写用this.$store.commit(名字,value)

这样子的。在store里面被action调用是这样的。

**Getters**



对state里面的数据进行加工，到时候直接调用getters。省去了将state读到组件里之后再

进行一系列加工的麻烦过程。

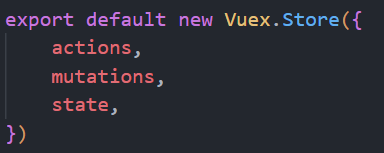
组件里面调用只要this.$Store.getters.bigsum



这样的。

**生成store导出**

之前只是准备了store的组成部分，真正导出还要是一个store，

。

**关于mapstate，mapmutation，mapaction，mapgetters。**

像上面那样在组件里面this.$store在组件里面写非常麻烦。如果想简写就用到了mapstate他们这几个。

**Mapstate**

在计算属性里面用...mapstate就可以把state数据拿到。



其实这是一个简写的方法。因为键和值一样所以只写了键。

**Mapgetters**



与mapstate的写法基本上保持一致。

**Mapactions**



其实都是这个逻辑，你给他键值对他会自动解析生成函数。下面那个也一样，不过注意传值是在调用函数的时候传。

**Mapmutations**

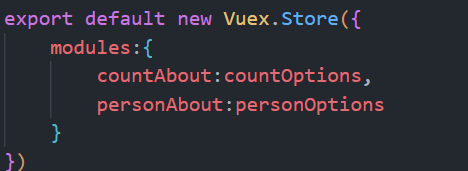


**modules**

像上面那样写，少了还行，多了就非常的麻烦，一堆一堆的。数据，函数。没办法分清哪个是哪个的，这时候modules就会发力。



把acitions,state,mutations,getters，按数据所属分类。封装成一个一个模块然后导出。



最后在vuex里面用modules配置好。之后就能在组件里面用了。但是用法稍微改了。前面要多套一层。，，

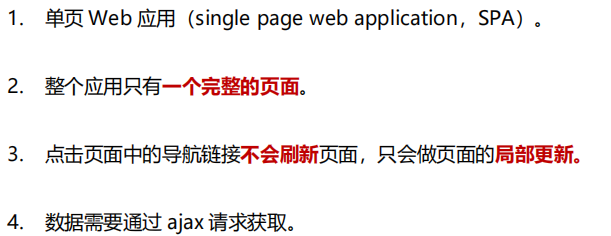
如果用map...这种方法

前面要加模块名。

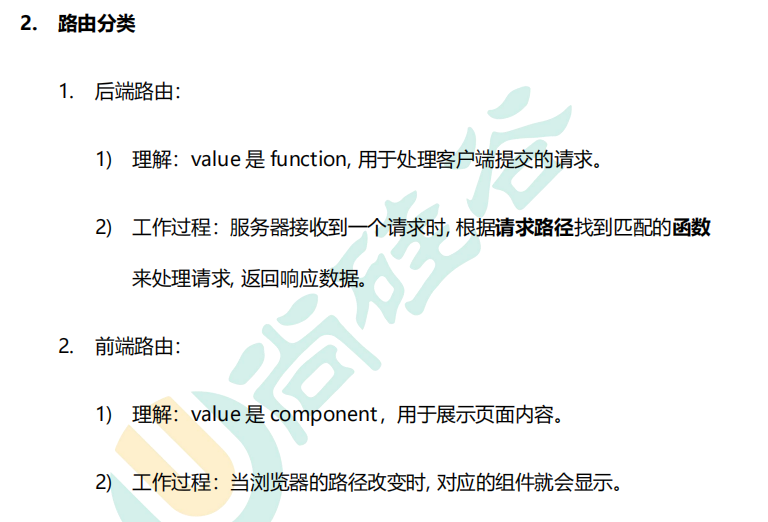
**模块化一定要加namespaced开启命名空间，不然store就废了。**

**路由**

路由是专门用来实现 SPA 应用。SPA应用的意思其实就是单页面应用。所有行为切换页面都是在一个页面里的。避免了每次都跳转到一个新的页面，极大的提高了效率和速度。

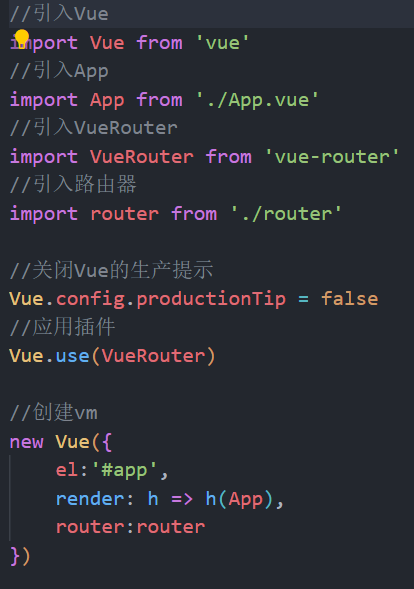


路由是以键值对的形式存在的。



**创建一个路由**

先在main.js这个入口文件里面，用引入vue-router然后使用插件。再建立一个router问件夹，里面放index.js。这个文件里引入vue-router。New router，最后导出。在main.js里面引用并且配置到vm里。

这就是路由的基本创建。

创建之后使用要注意配置路由出口



出口配置完后实现路由跳转用routerlink这种方法。

**嵌套路由：**

在需要嵌套的路由下面多一个children属性。像这样。



路由组件不会放在component里面了，会单独在一个page文件夹里面存放。

**路由传参**

有些时候要带着参数访问路由，根据请求不同带的参数一般为params和query两种。

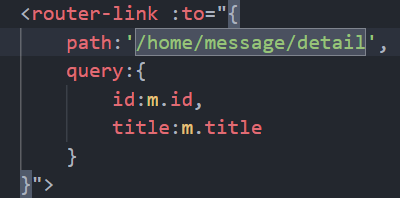
分别介绍一下。

**Query参数**

通常用于get请求。写法如下：



?后面带的都是我们要传的参数，他通常是用?和&连接的，也有对象式的写法。

使用时则是this.$router.query拿到数据。。

**Params参数。**

param参数要有占位符。



一直/的方法传参。

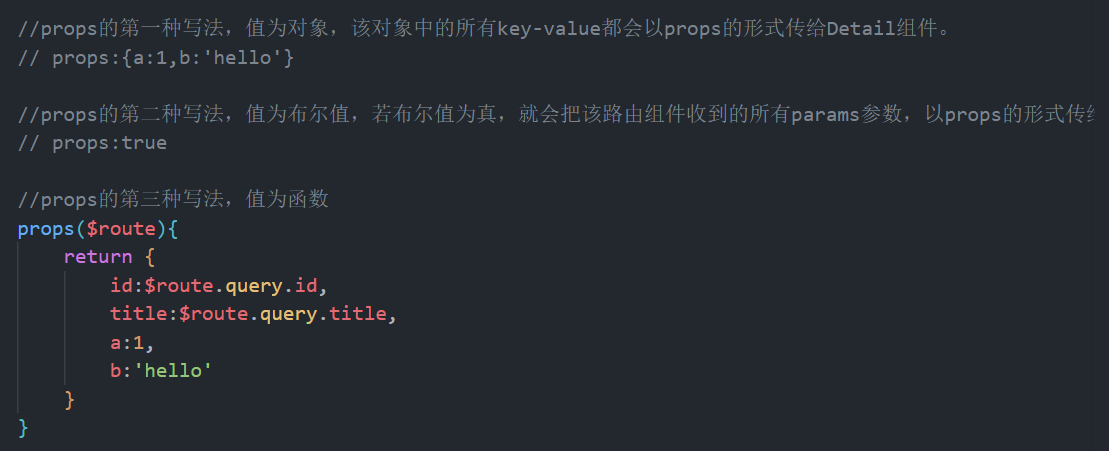
对象式的写法：

路由配置时出了path和component两个属性外还可以有一个name属性给路由命名。

这就是路由名的配置，我们params传参不接受路由路径，所以用路由名来代替一下。

**路由的props**

像上面那种写法来说，还是要用this.$route这种方法拿数据，很麻烦。所以直接在配置路由时就用props的形式把参数都拿到然后再props传给路由组件。



有三种方法：

第一种是对象类型的：

这种配置参数都是写死的很少用

第二种是布尔类型的配置：

这种方法只要开启了一个true那么所有的params参数都可以直接在子组件里面props拿到。但是query没办法用这种

第三种是函数类型的配置：

这个函数能拿到$router返回一个对象。Params和query都能拿到。这种方法是最方便的了。

**Routerlink的replace属性**

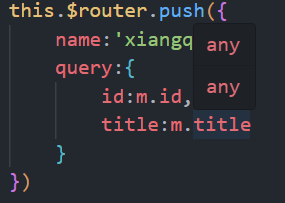
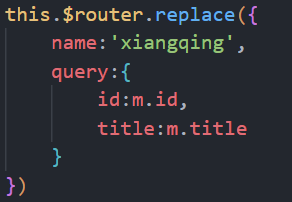


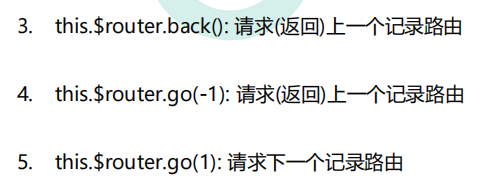
这涉及了浏览器的两种浏览方法，push方法和replace方法，默认开启的就是push方法。这两种方法其实就是对浏览器的历史记录能否回退，回退到哪进行操作的。

**编程式路由导航**

有时候路径的跳转不一定要通过点击，如果我们只使用router-link方法写路由的话，所有跳转全部被默认成了a标签。这不是我们想看到的，所以用编程式路由导航来解决这个问题。

他能实现与routerlink一样的方法，且他的使用范围更广。

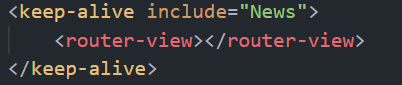




这两种方法，正好对应了push模式和replace模式。里面配置对象可以写路径形式，也可以写对象形式，但是对象形式明显的更方便，所以我们通常使用对象形式。

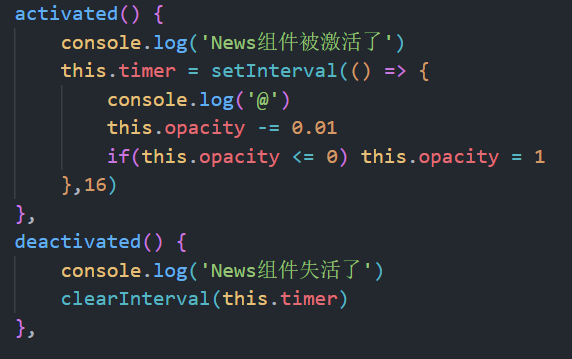
**缓存路由组件**

每次路由组件的切换伴随的都是一个组件的销毁与一个组件的创建。如果切换的频繁，可以不用销毁组件，这时候缓存路由组件就发挥了作用。在路由出口外面套一个标签就可以缓存路由组件了。



缓存单个与缓存多个的结果就是这样了。

配合上面的缓存路由组件使用的是两个新的生命周期钩子。因为组件不会被创建和销毁。所以我们创建销毁的生命周期钩子会被下面两个生命周期钩子替代。Actived替代的就是mounted，deactived替代的就是beforedestory



**路由守卫问题**

这玩意就相当于一个中间件，对路由跳转的进入和离开进行了一些校验，保证路由安全。

有三种路由守卫：

**全局路由守卫**

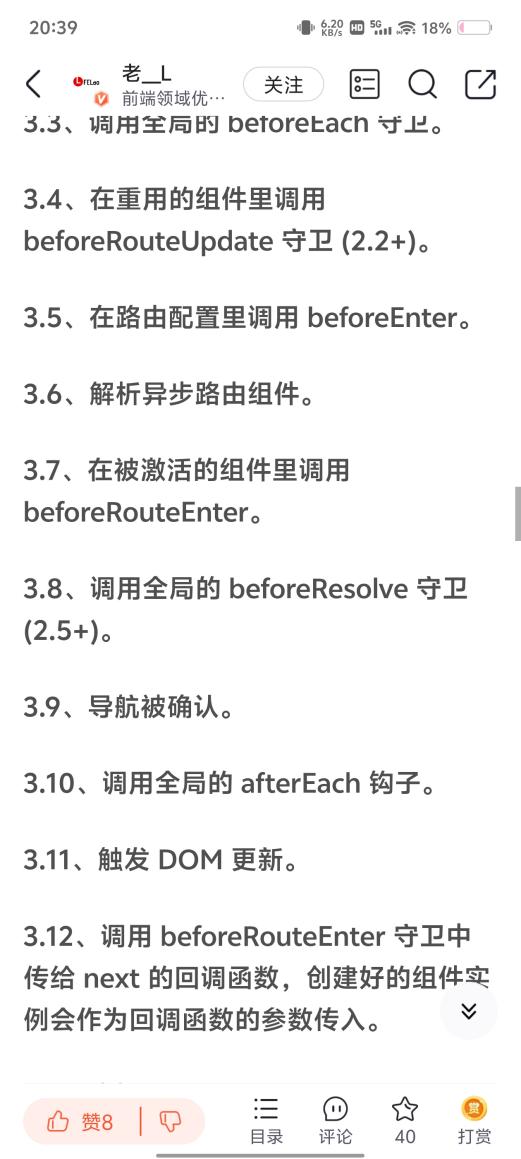
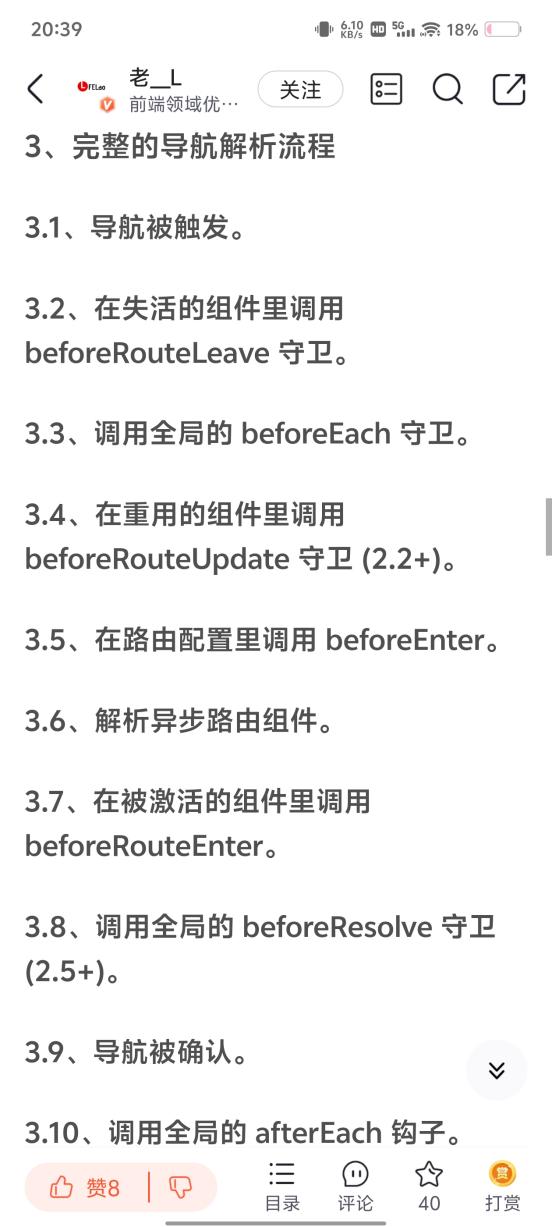
这就是他的用法。所有路由都得经过这个守卫。

**独享路由守卫**



只有这一个路由能用，所以称之为独享。值得注意的是他的逻辑与全局的没区别。

**组件内路由守卫**

在组件里面被当生命周期使用了。逻辑也差不多就那样。只有在这一个组件里面能用。

这是三个守卫的执行顺序。

**浏览器路由的两种模式hash(哈希模式）history(模式)**



路由的模式在配置路由时就可以标好，两种模式的区别就在url上，一个有#一个没有。

值得注意的是hash模式虽然路径好看但是要后端配合。



b站用的就是hash模式。

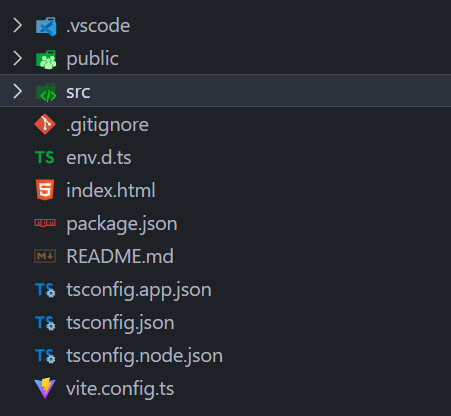
我们组件库这块用的一般都是history模式。

**vue3.x**

vue3发布已经有四年了，现在主流已经变成了vue3。他在vue2上增加了一些好东西。解决了vue2的一些问题，比如提供了组合式API，用proxy替代了defineproperty，增加了typescript的支持等。

**用脚手架创建vue3工程。**

在cmd里输入这个命令就可以用vite创建vue项目。之后就是交互式创建，根据需求创建就行了，不做过多解释。

这就是创建之后的文件夹与vue2相比多了ts支持，vue.config.js变成了vite.config.js。且入口文件变成了index.html，在index.html，他用script标签把main.ts引入了。写代码方面的一个大的变化就是把JS代码换成了TS，剩下的语法改变都是在vue2的基础上实现的。

有一点变化就是createapp这个东西替代了原来newVue配合render函数实现挂载的那一块。

**编写APP组件**

，一样是老朋友，template，script,style，与vue2如出一辙，所以就不进行过多的讲解。注意一下script的标签多了一个lang=“ts”，这个就是我们前面创建脚手架的时候设置的。

**组合式API与选项式API**

选项式API就是vue2那种的，data，method，computed列在一起。这种写法写多了看起来会很麻烦，虽然可以用mixin混入但文件分多了还是很烦。Vue3为了解决这个问题提供了组合式API。使用组合式API可以使编码非常的舒服。一个功能的代码放在一块，易于维护。



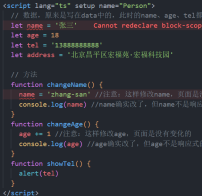
注意的一点是setup返回的也是一个函数。

这就是组合式的基本写法，。而且setup是可以和原来选项式api一起写的，只不过有点乱。不建议这么写。如果遇到了别人写的就一错到底，屎山上面叠屎山吧，别无他法了。

虽然setup函数回到了JS的写法但是有一点很难过就是要注意return，返回一个两个还行，变量和函数一多了，一下返回的就是一大串，看起来就头大非常的不方便。

Vue3也是很贴心的解决了这个问题。提供了一个setup的语法糖，直接帮我们省去了return，setup这个函数也不用写了。直接在script标签里写代码。无脑干就行了。

写法如下：只要在script标签里面写个setup就行了。

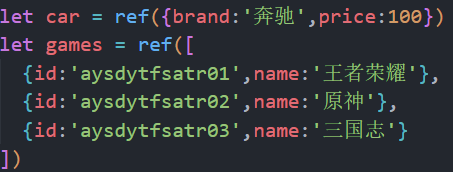
再也不用return了哈哈哈哈哈哈哈！

这样写是方便了。但是没有办法得到响应式。也就是说我们调用setup里面的方法更改数据模板是不会更新的。这时候就要引出ref和reactive这两个东西解决响应式的。

值得一提的是vue3底层用的是proxy和reflect替代了defineproperty解决响应式问题。在这题一嘴，像这种问题一般都是面试会问的。

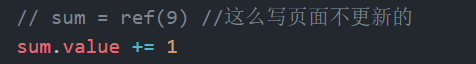
回到主线，我们从vue中引入的ref属性可以帮我们的数据拿到响应式，数据一旦用**ref**套了就算是响应式的了。用ref套住的数据在模板中可以直接写{{name}}。但是如果在JS中写就要加.value。因为ref套住之后他就变成了对象。

与ref类似**reactive**这个玩意也是给数据提供响应式的，只不过他是用于给对象提供响应式 的。就是这样。

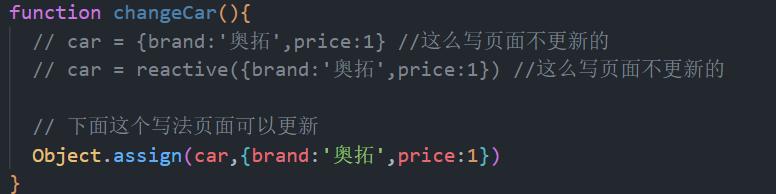
Ref也可以给对象创建响应式数据。ref本来是只管一层的，但是vue底层进行了一个骚操作。如果把对象用ref套了，ref会发现，然后去找ref进行包装。所以ref包装对象很方便，但是感觉也没必要。

**细节：**

Ref包裹的数据要改必须.value拿到数据直接用ref(数据)赋值是没用的。



Reactive包裹的对象，改单个数据可以，但改整个对象的话直接赋值或者是reactive包裹赋值都是没用的。要是想硬改就用object的assign方法实现。



**toref和torefs**

将响应式对象中的每个属性赋予响应式，二者的功能是一样的，只不过torefs可以批量。



**计算属性**

和vue2里的计算属性差不多，只不过变成了函数。且需要引入。

函数里面的配置对象也和vue2一样。如果不需要修改直接就是一个回调函数，vue3不维护this所以直接写箭头函数就行了。如果需要改就不能写简写了，就得乖乖的写get和set函数。另外回顾一下调用时机：get函数的调用时机是初始化调用一次之后再就是计算属性所依赖的数据发生变化的时候调用。

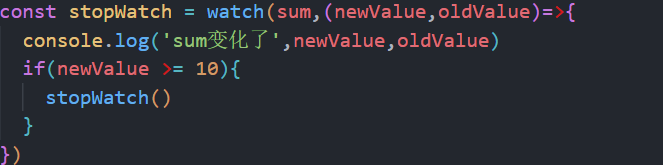
Set的调用是计算属性计算的值被修改之后调用，他会收到被修改成为的值。

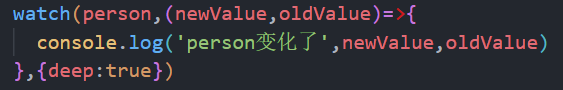
**Watch监视**

与vue2的监视保持一致，但是他也变成了一个函数，也需要手动引入。

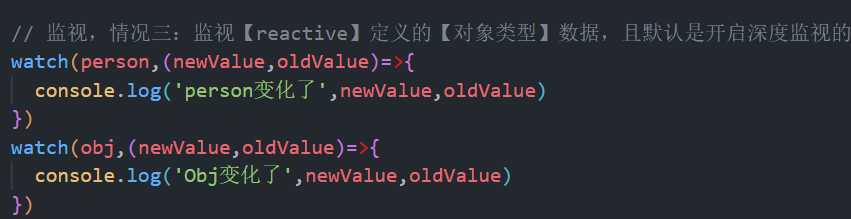
但是vue3里面的监视只能监视四种值：

Ref定义的对象，基本数据类型

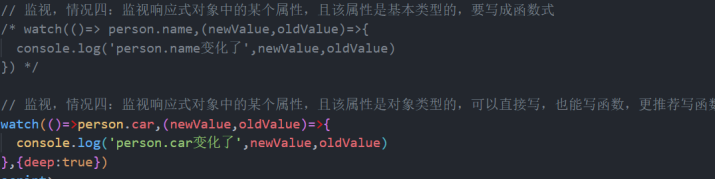
这段代码监视的是基本属性，直接写就行了。

这段代码监视的是对象，需要加配置项，开启深度监视。因为他默认监视的是地址值。

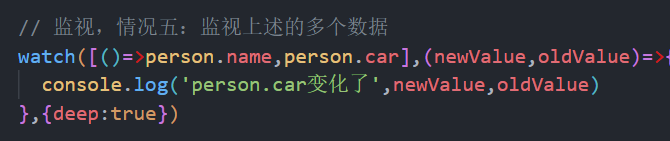
reactive定义的对象



Ref或reactive定义的对象的某个属性

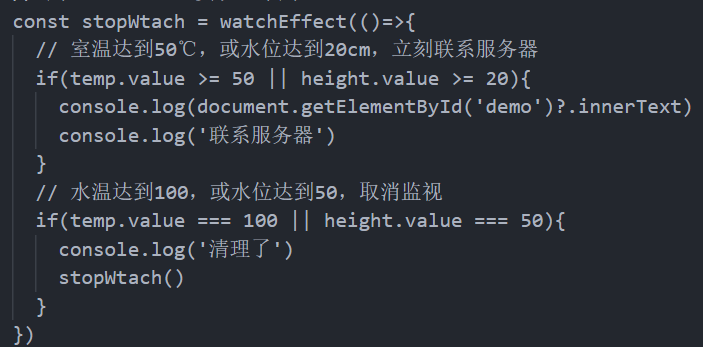
属性是基本类型写成函数，对象类型的怎么写都行。

包含上面几种对象的数组。



**WatchEffect**

这个东西是watch的一个方便化写法，他会自动监视其中所用的数据变化，并且调用内部的逻辑。本质上是一个自调用的立即执行函数。



**关于标签上的ref**

在vue2中就用ref给标签打记号来拿dom元素了。Vue3里面ref依然能够用来获取dom元素，只不过语法发生了一点点的改变。



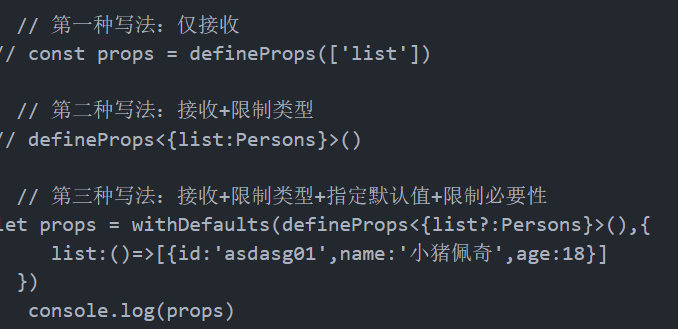


同理如果给组件标签用这个玩意标记也一样有效。能够拿到组件对象。同时可以通过.value操作子组件的数据，但是vue保护了数据的安全，所以我们子组件要用defineExpose暴露数据之后父组件才能正常拿到。



**Props**

这玩意也和vue2没啥大的变化。父组件只要在子组件的组件标签上写数据。子组件接收就行了。只不过接收的props从对象变成了一个函数，引入之后方可使用。



**生命周期**

与vue2的生命周期相比并没有做出什么大的改变。

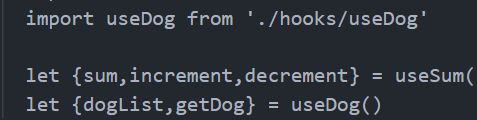
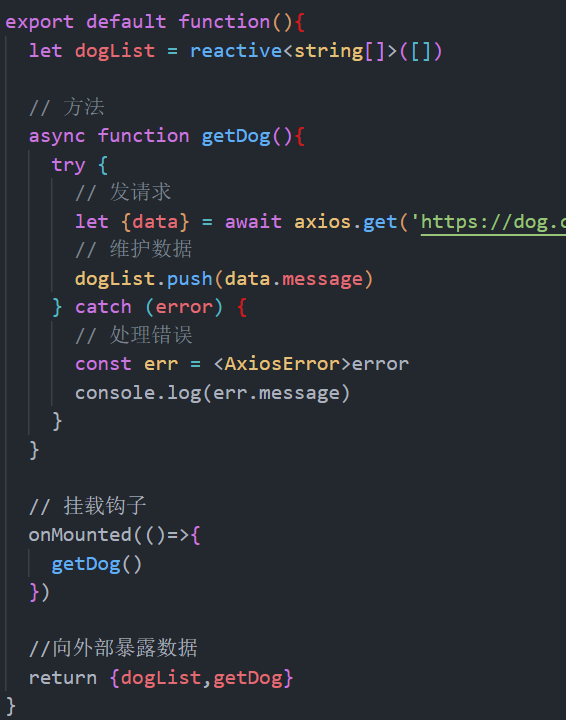
只是将beforeCreated和created用setup替代了。

将beforedestroy和destroy用onBeforeUnmounte和onUnmounted替代了。

同时在其他的生命周期前都加了On。

**自定义hook**

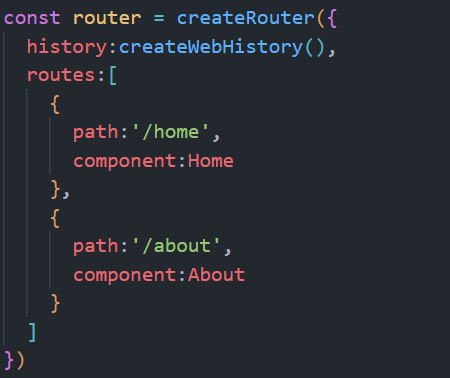
其实这玩意就是模块化，组合式API如果都在setup这里面写的话，写多了也容易找不着北，所以把相关性强的代码放在一个文件里，用一个函数配合返回值的方法导出。用的时候引入解构就能用，这也是组合式API很核心的一部分。



**路由**

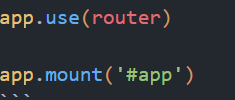
和vue2路由一个意思，实现SPA（单页面应用）。只不过语法有一点点的变化。之前vue2里面的路由我们用的是new VueRouter()这里我们的路由改用了createRouter



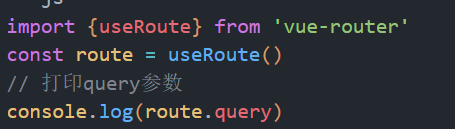
hash和history也改成了函数的形式。其他配置项都一样。



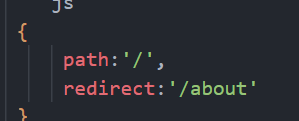
路由出口和RouterLink换成了大驼峰

挂载方式发生了一点点 的变化。

路由传参：传参的时候没有变化与vue2保持一致，接收参数的时候发生了一些变化。



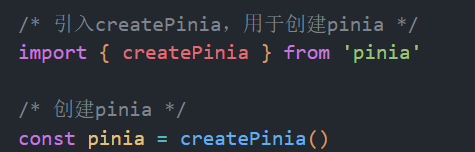
路由重定向：将特定的路径，重新定向到已有的路由。



命名路由，嵌套路由，路由跳转，replace属性，路由的props模式与vue2完全保持一致。

**Pinia**

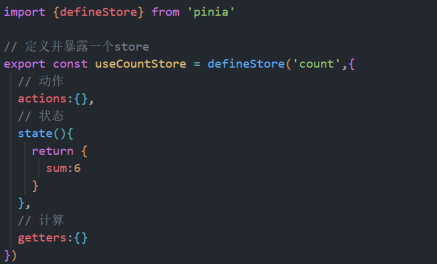
Vuex的一个平替，这个玩意用起来比vuex简单：正如pina官网所说。他是一个符合直觉的集中式状态管理工具。现在官方不推vuex而推pinia

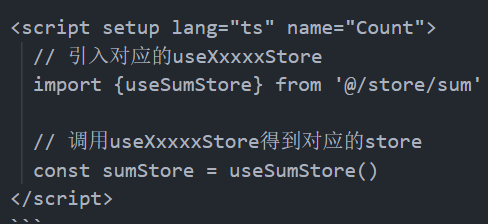


毕竟都是插件，所以用法与vuerouter的用法一样。引入，创建，使用三步走。

与vuex一样也放在store文件夹里，创建一个index.js



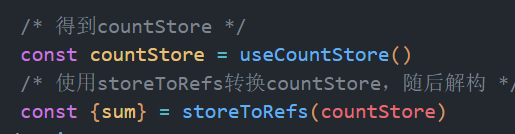
依然是引入创建导出三步走。



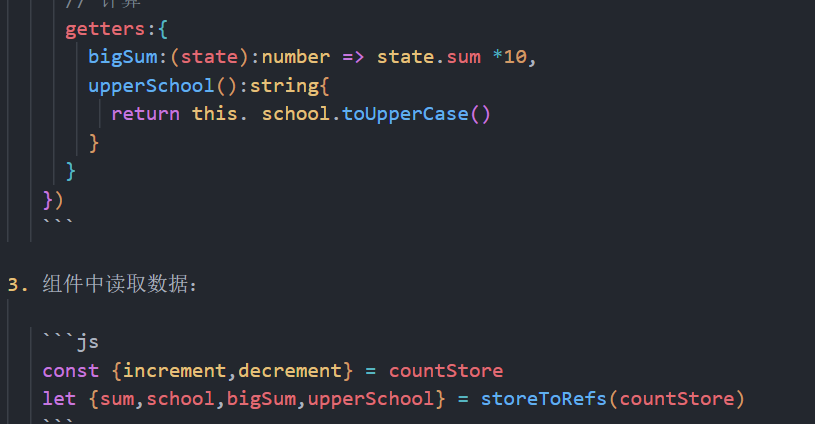
就是这样了。读取时直接引用然后赋给变量就行。

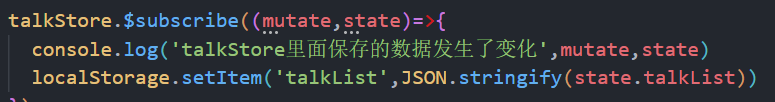


修改时调用action里的函数就能修改。



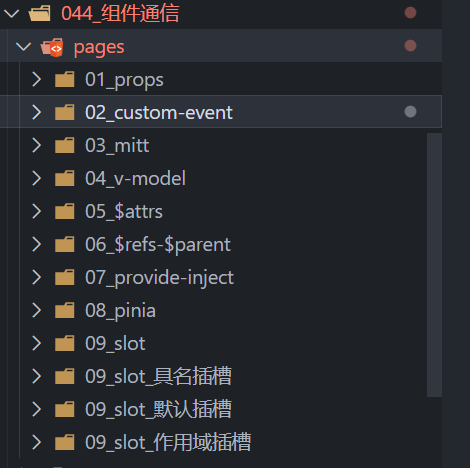
Pina提供了storetorefs实现数据的响应式。他只会给state响应式，如果用ref会把所有的都给响应式。

getter的读写与state基本一致。

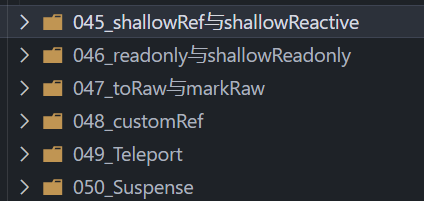


这个玩意能监听组件里面数据的变化。

这是组合式写法，直接抛弃了action，state，getter，该怎么写怎么写了。最后注意return就行。



组件通信，大致上与vue2一样。看看得了。mitt类似全局事件总线。Provide，reject为新增的祖父与子通信方法。



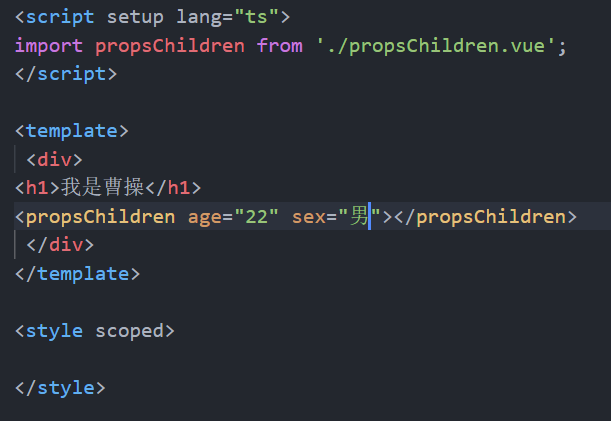
ShallowRef那些就是给浅层加响应式的，readonly套上之后改不了，shallow是一层改不了。

toRaw让数据没响应式，markRaw让数据永远没有响应式。CustomRef是自定义ref实现方法。核心在get里面的track打标记，set里面的trigger做追踪。Teleport是一个解决定位问题的相对性的。Suspence是为了异步等待时响应一些内容用的。

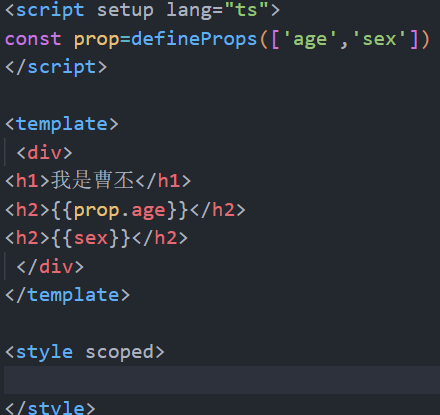
## Props（父传子）

路由都已经配置好了,下面是父子组件

父组件



子组件

原理很简单就是父组件和子组件之间用props的方法实现了父传子的一个效果。**注意:props拿到的数据是只读的,想改是不可以的。**

## 自定义事件（子传父）

子组件与父组件如下:

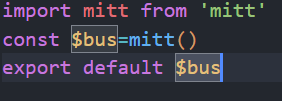
这是子组件。

这是父组件。

当我们父组件要获得子组件的数据时就在组件标签上自定义一个事件。子组件通过defineEmmit的方法接受事件,返回一个处理函数。再通过事件来调用这个处理函数即可实现子传父。

## 3.全局事件总线（兄弟组件）

Vue2里面有个全局事件总线叫$bus公交车,在vue3里有我们可以通过mitt来实现一个类似的效果,就是类似于pubsub一样,一个订阅一个发布。

mitt是个方法,我们先用实例化一下这个方法。

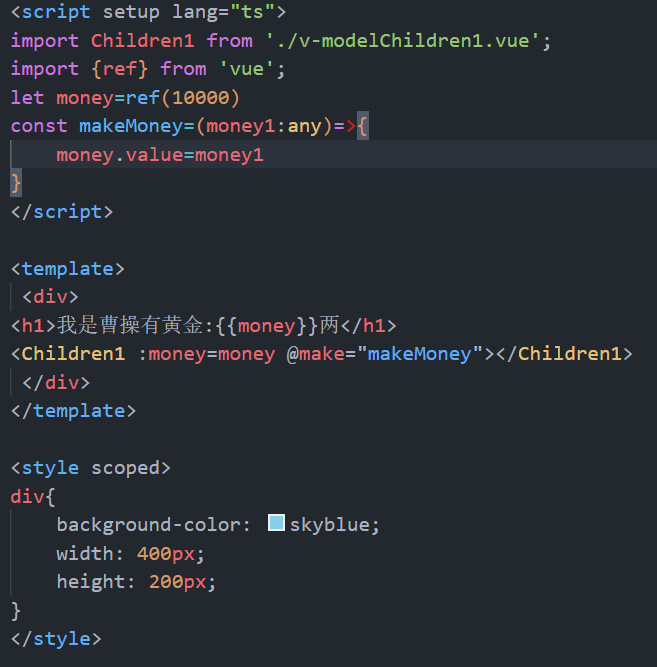
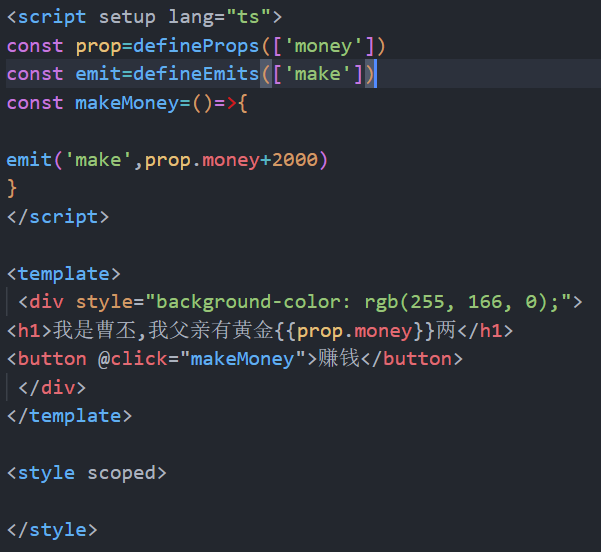
之后就可以利用这方法实现兄弟组件之间的通信。

这是兄弟1,这是兄弟2。

**我们兄弟二要给兄弟一传数据就要emit一个数据,在兄弟二里通过on就可以接收数据。**

## 4.V-model双向绑定实现组件数据同步

实现数据同步可用用props和emitt实现。



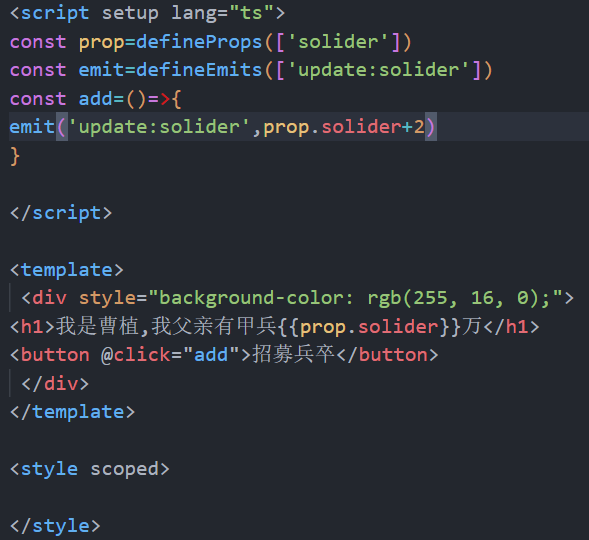
父组件的数据通过props传给子组件,子组件改了数据就用emitt传给父组件实现一个数据同步的效果,但仔细看就会发现这个写法很像一个东西。



没错就是v-model。v-model拆开了不就是一个:value,一个@input或者@update吗？



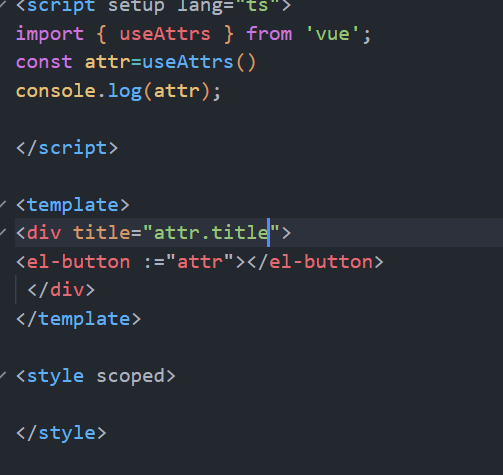
那我们完全可以按照上面那么写。这个就是组件封装时经常使用的方法,Element-plus里面是喜欢用这种方法的。**只不过即使用这种方法子组件里依然要用defineProps和defineEmmit。**



## 5.useAttr方法

这个方法和props很像,他能够接收父组件传的所有属性。

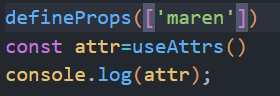


我们可以用它对elementPlus这种组件库进行一个二次封装。

**但要注意的是props接收了的useAttr是无法得到的。**

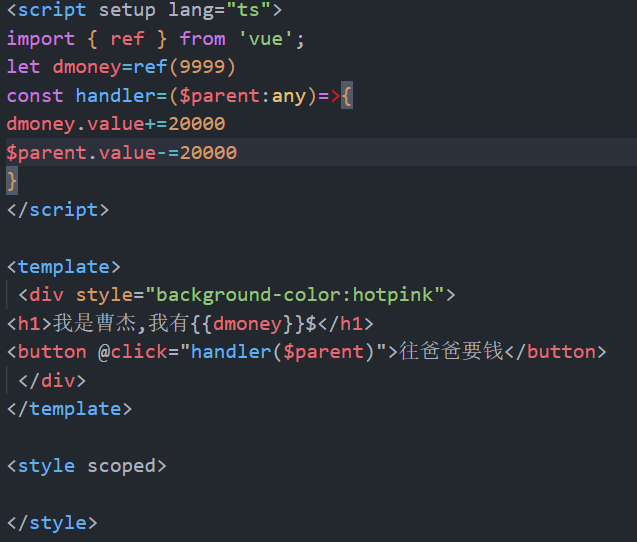
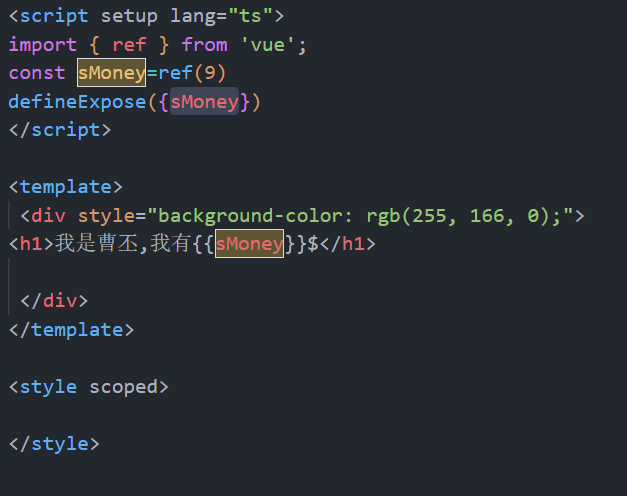


这里多加一个属性。

我们用props接住。Attr就不会再拿到了。



## 6.ref和$parent



上面三个依次是儿子组件,父亲组件,女儿组件。

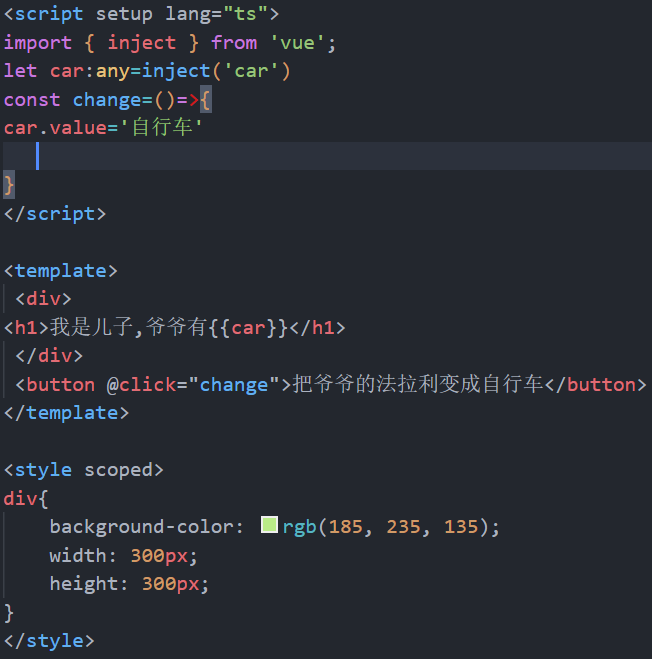
**父组件通过ref的方法拿到了子组件的数据,实现了往儿子要钱的操作。**

**女儿组件通过$parent方法拿到了父组件的数据,实现了往爸爸要钱的操作。**

**但不管是ref还是parent都要有difineExpose这个操作。**

## provide与inject

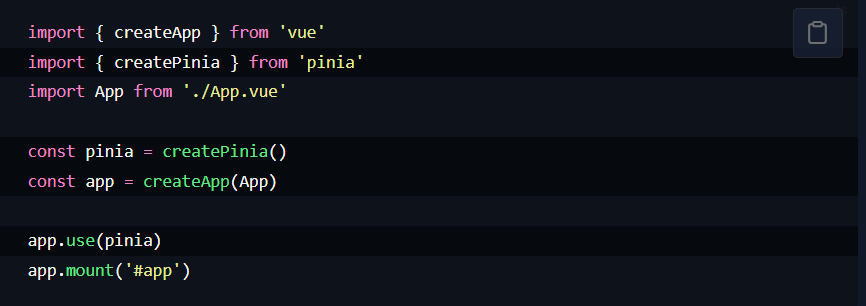
这俩东西主要是用于隔辈传数据的。通过这个可以跨过父组件,直接从爷爷组件传数据到孙子组件,同时,传给孙子组件的数据还可以更改。



## Pina

Pina就是现在Vue官方推荐的状态管理工具。通俗点说就是把数据放到一个独立于任何一个组件外的仓库里,我们可以在任何组件里面使用,更改Pina里面的数据。





这就是官方给的配置pina的方法。

**之后就可以定义商店了。商店在src的store文件夹下,这是一种规范。**

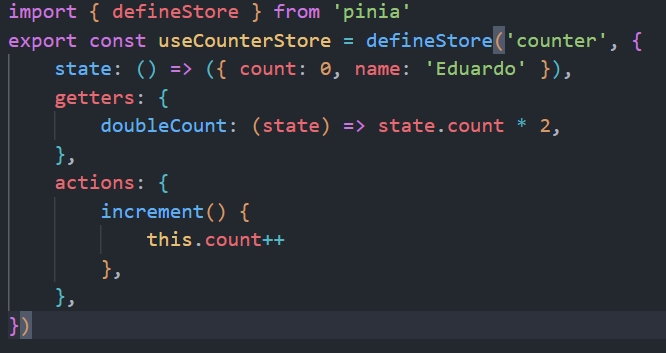


这是官方给的创建商店的方法。

Pina的写法也分两种:组合式和选项式。下面分别进行一个示范。

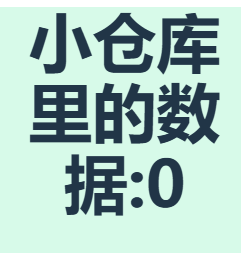
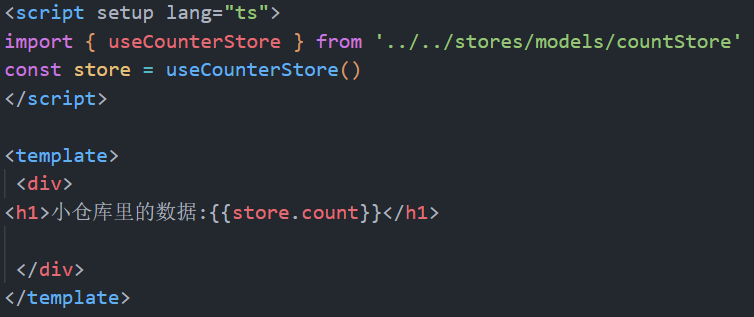
### 选项式API

选项式API更接近于vuex,只不过少了个mutation提交的过程而已。主要还是state数据,getter计算属性，action方法三种。



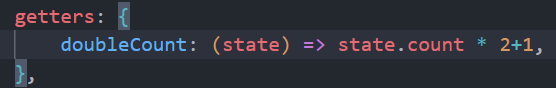
#### State

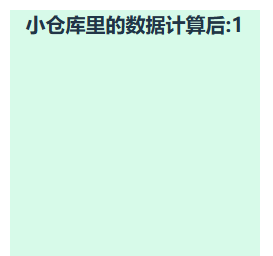
这是我们商店的原始形态。我们可以在任何组件里拿到state。



#### Getter

我们也可以用getter来计算值,可以把他当成vue里的computed计算属性。



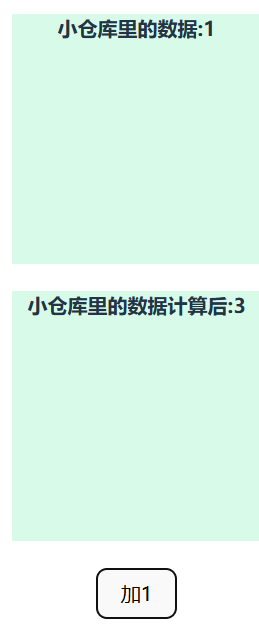
就是一个计算属性的作用。

#### Action

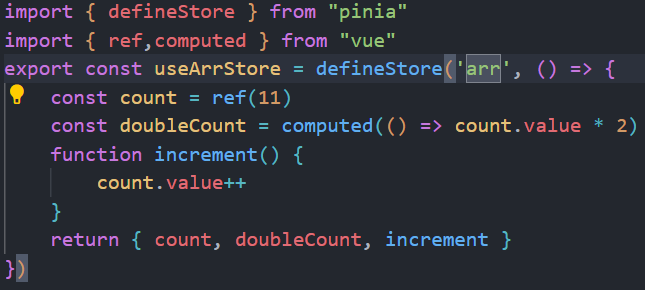
这就是一个方法了,我们在组件里可以通过Action这种方法来改动仓库里的数据。



我们一旦点了这个按钮,那仓库里的count数据就会发生改变。连带着getter也会重新计算一遍。

**仓库里的数据一旦改变那所有组件里的数据都会改变。**

### 组合式



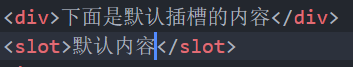
逻辑依然和选项式是差不多的,这里就比较贴合Vue3了,这里用的state是ref的数据,这里用的getter是computed计算方法,这里用的action直接用的是function方法。

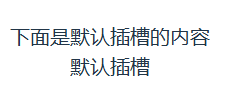
## 插槽slot

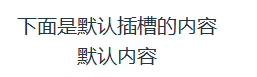
插槽这玩意其实就是用在组件标签上面的。他可以更方便的把父组件的内容放在子组件里面,一步到位。常用的插槽有：默认插槽,具名插槽,作用域插槽三种。

#### 默认插槽

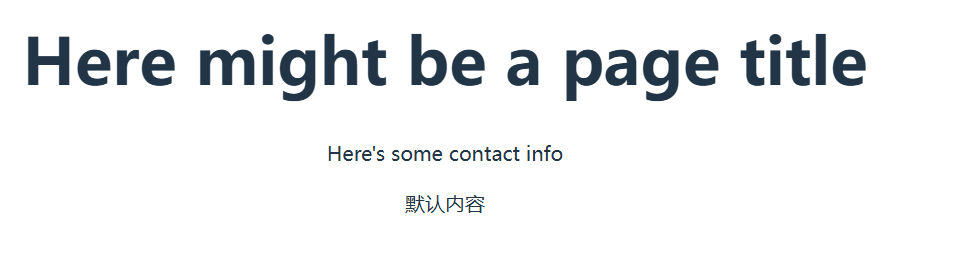
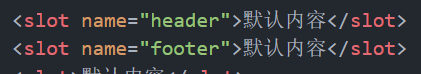
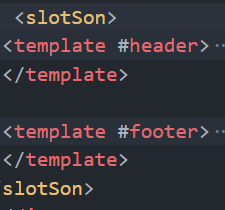




我们子组件里填的默认内容会被组件标签中间的内容给覆盖掉。但如果我们插槽中间啥都不写那就会显示默认内容了。



#### 具名插槽



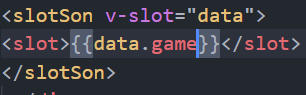
内容很多的时候就要标明name一一对应上去才能插进去。

#### 作用域插槽

作用域插槽这个东西其实就是子组件把数据交给父组件管理的一个方法。



子组件把数据传给父组件

父组件以对象的形式接收到了子组件传来的数据。根据数据可以更好的生成结构放在插槽里。从而传给子组件的插槽更加完美。

