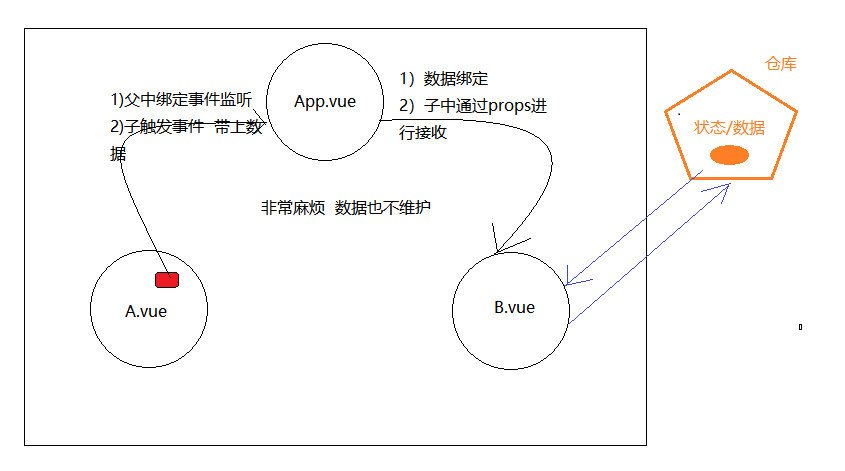
# Vuex（难一点）

Vue应用程序的**状态(**state**)管理器。**它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。



把“状态”换成“数据”。

把vue应用的数据放在一个地方管理。 每一个vue的组件都可以有data项（有自己的数据），我们可以通过父子之间的相互传递，进行数据的交换。

父传子：自定义的属性; 子组件用props

子传父：自定义的事件；子组件用emit

## 为什么要用它

如果你希望在子组件与另一个组件之间共享（它们是兄弟，表兄弟，堂兄弟关系）数据，则非常麻烦。

如果你的项目中，需要用到在各个子组件中共享数据，则你就需要用到vuex。

## 示例

**设置如下组件：**

* 一个vue实例，充当根组件
* 子组件AddNumber。
* 子组件SubNumber。

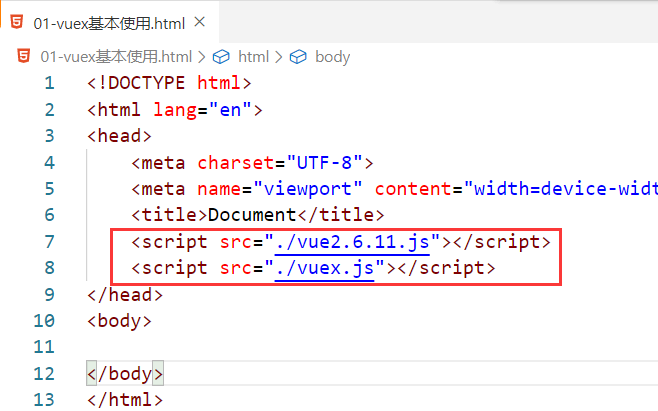
**希望在两个子组件AddNumber和subNumer之间共同维护和使用数据项:counter**

* 在vue实例中：显示 counter的值
* 在子组件AddNumber中：显示 counter的值；添加counter的值
* 在子组件subNumber中：显示 counter的值；减少counter的值

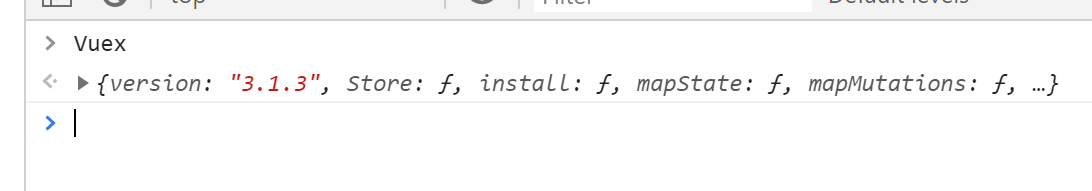
由于上面的需要，我们需要在三个地方共同使用同一数据项counter，所以我们需要用到 vuex。

## 步骤

### 先引入vue.js，再引入vuex.js



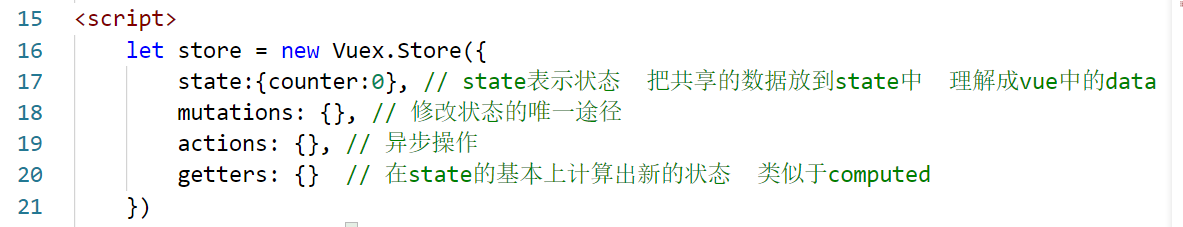
此时，你已经可以在控制台中访问vuex



### 实例化vuex中的**store**对象

实例化一个对象，就是通过new的方式去创建一个对象。

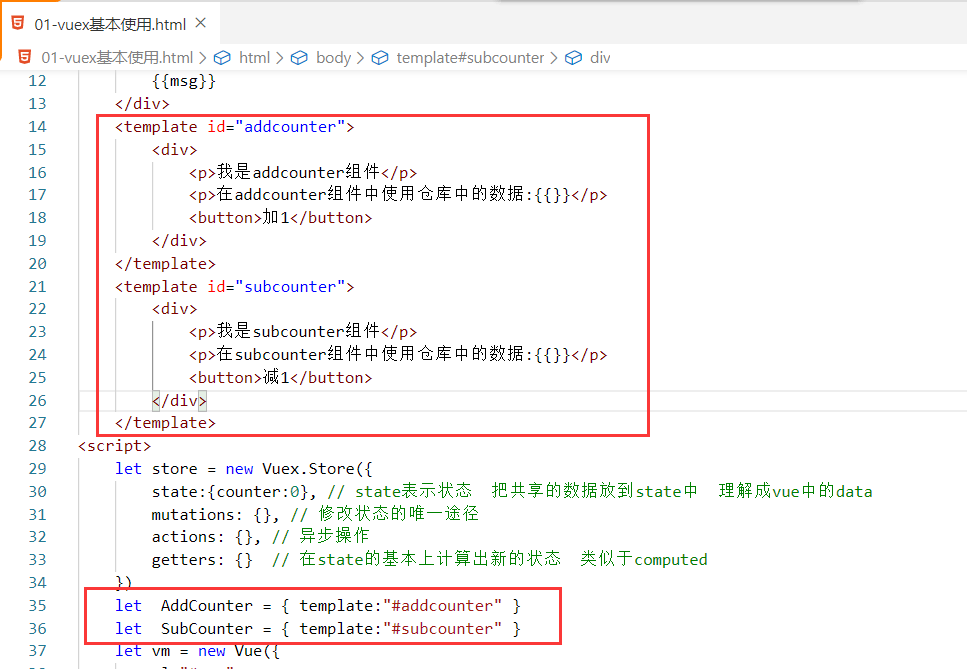
#### 格式



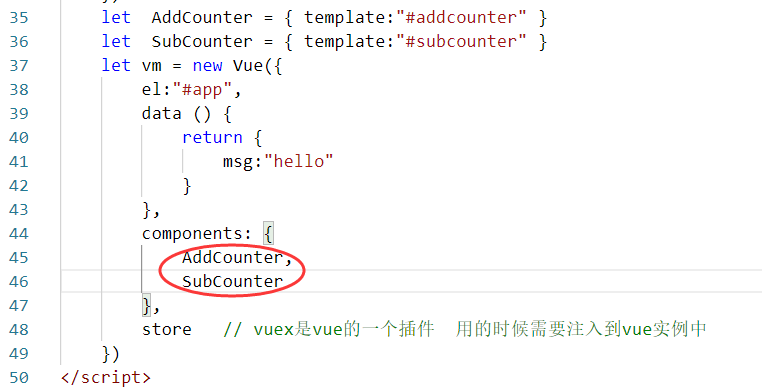
### 注入到Vue实例中



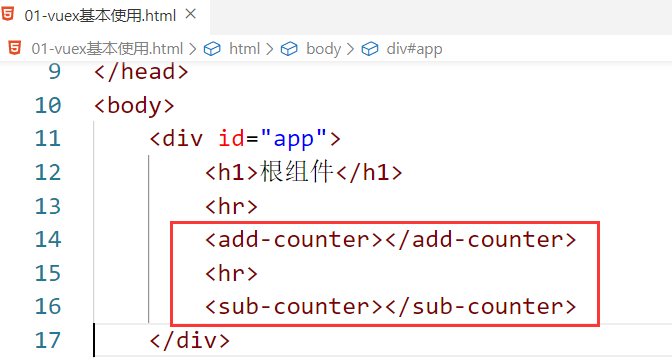
定义两个组件如下：



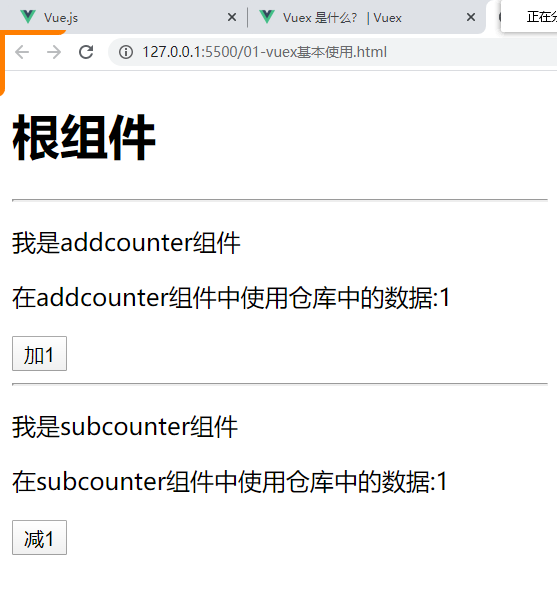
不要忘了，组件需要注册，如下：



使用组件，如下：



效果如下：



### 使用store中的数据

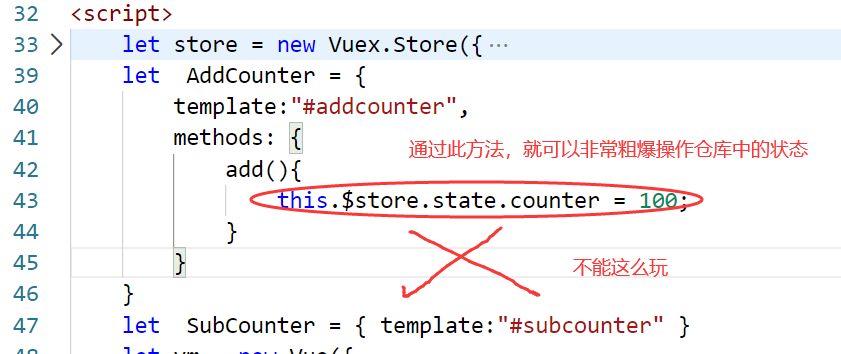
一旦你在vue的实例中注入了store，则在所有的子组件及 vue的实例中，你都可以通过：this.$store.state.数据名 去获取数据

类似于我们把router注入到vue实例中，我们就可以通过this.$router 和 this.$route操作路由。

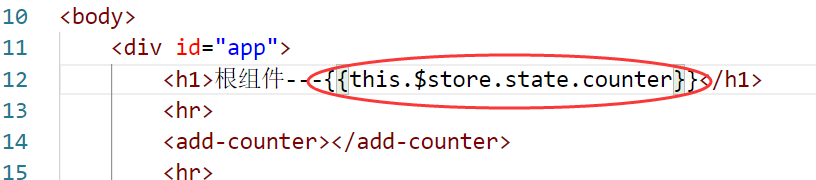
在组件中使用仓库中的数据，如下：



也可以通过：this.$store.state.数据名 = 值 去修改数据，但是，vuex反对这么做。



**你可以有两种方法去使用数据：**

1. 获取：this.$store.state.数据名  
   
2. 修改：在组件内部通过this.$commit方法触发事件，执行mutations当中的对应的方法。

## 最核心的概念- store

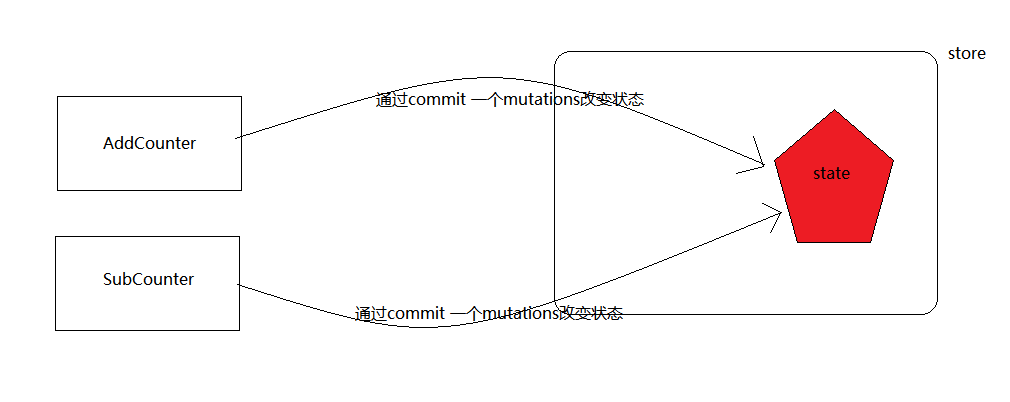
每一个 Vuex 应用的核心就是 store（仓库）。

store是一个容器，包含着vue应用的状态(state)。

有如下两大特点：

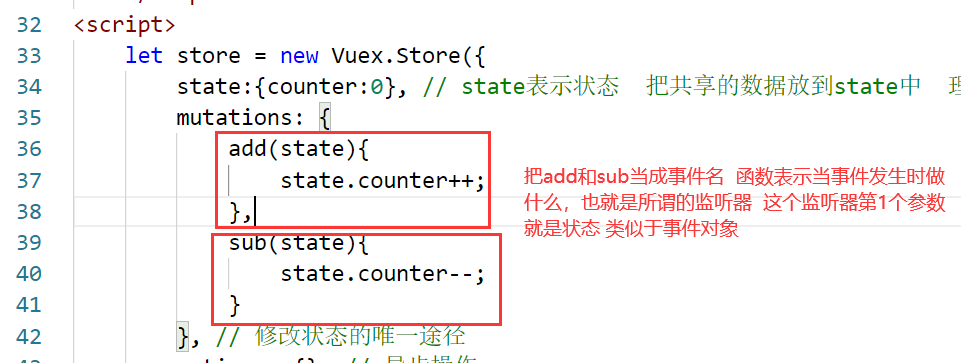
Vuex 的状态存储是响应式的。当 Vue 组件从 store 中读取状态的时候，若 store 中的状态发生变化，那么相应的组件也会相应地得到高效更新。

你不能直接更改 store 中的状态。**改变** store 中的状态的**唯一途径就是显式地提交(commit) mutations**。这样使得我们可以方便地跟踪每一个状态的变化.



## Mutations

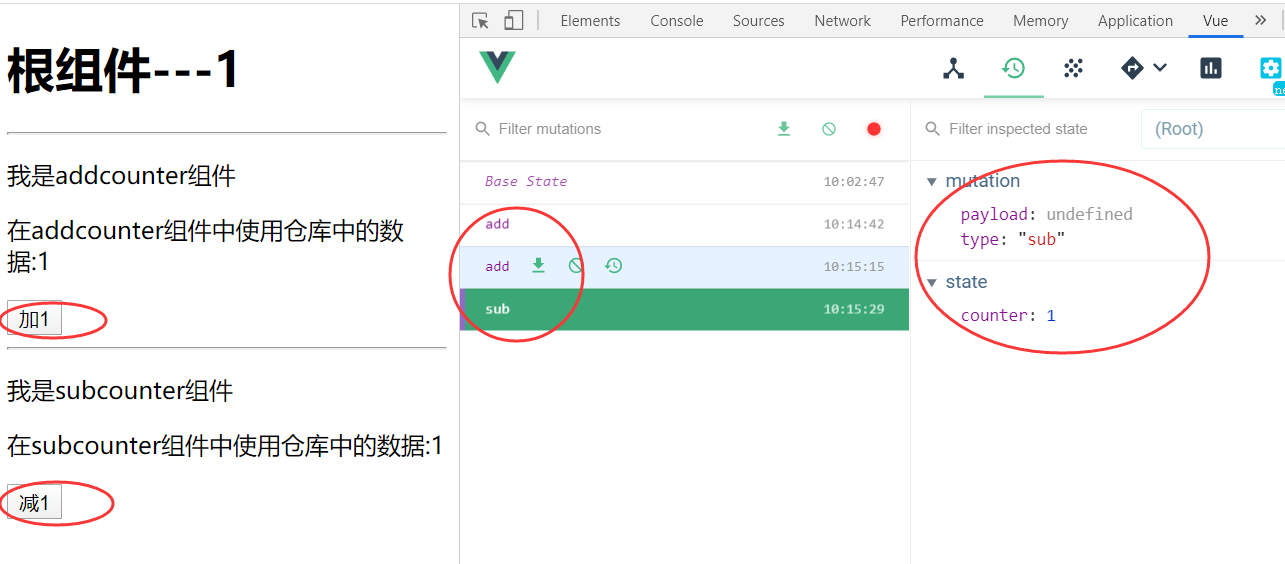
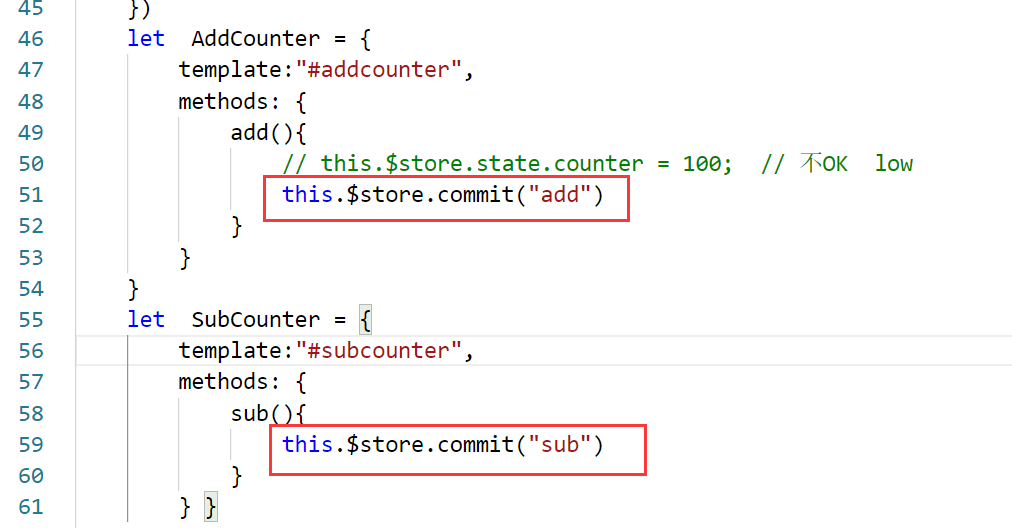
### 定义格式：



### 调用这个mutations

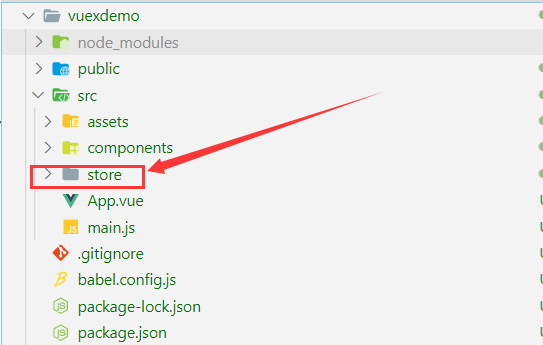
它是间接被调用的，在组件内部：

this.$store.commit(“mutations中的属性名”) . 代码如下：

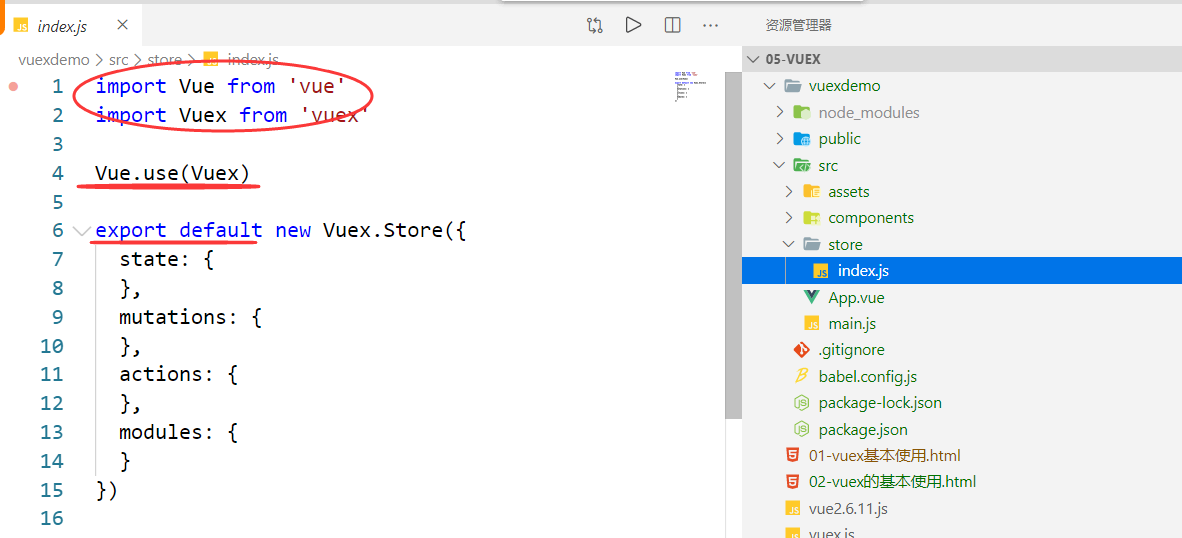


## 在vue脚手架中使用vuex

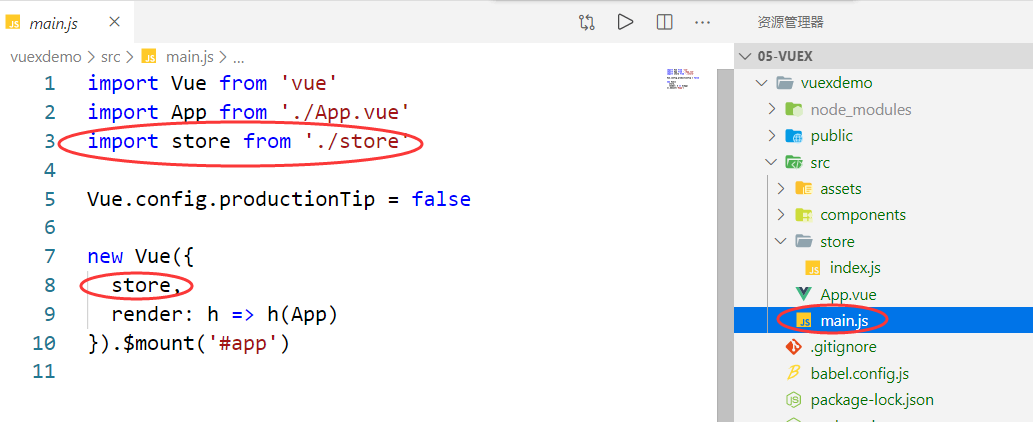
在创建项目时选好了vuex，默认它生成的代码骨架，如下：



默认就有一个叫store文件夹，里面有一个index.js，就代表vuex中的仓库 ，如下：



在项目入口中导入store仓库，需要注入到根组件中，如下：



### 示例

设置如下组件：

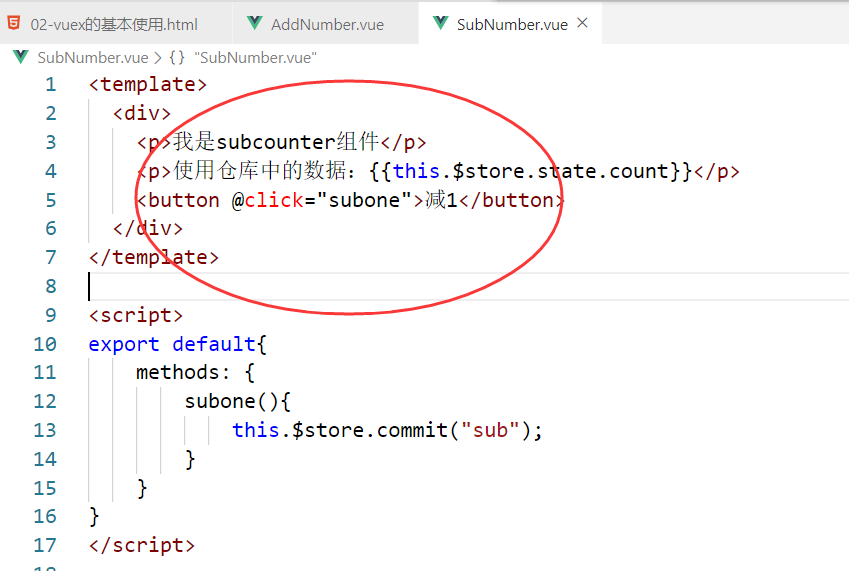
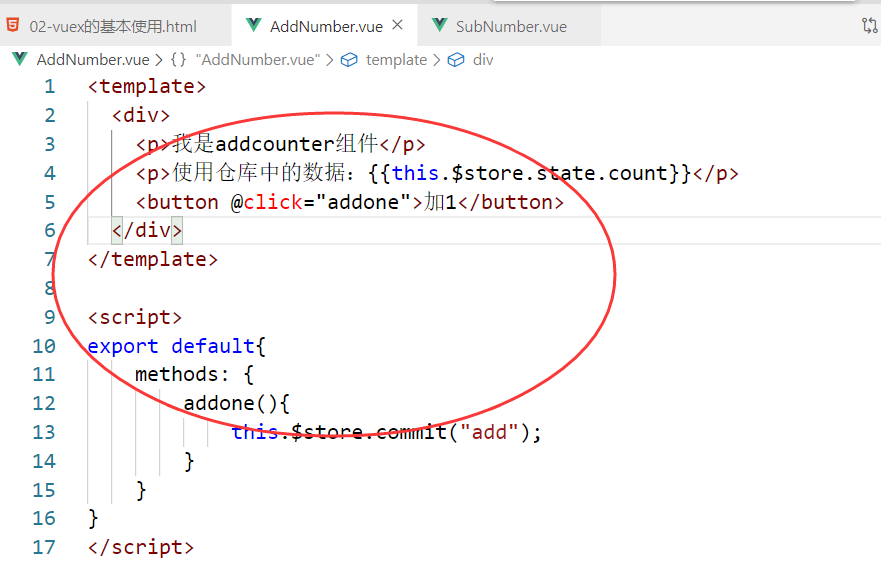
1. 一个vue实例，充当根组件
2. 子组件AddNumber。
3. 子组件subNumber。

希望在两个子组件AddNumber和subNumer之间共同维护和使用数据项:counter

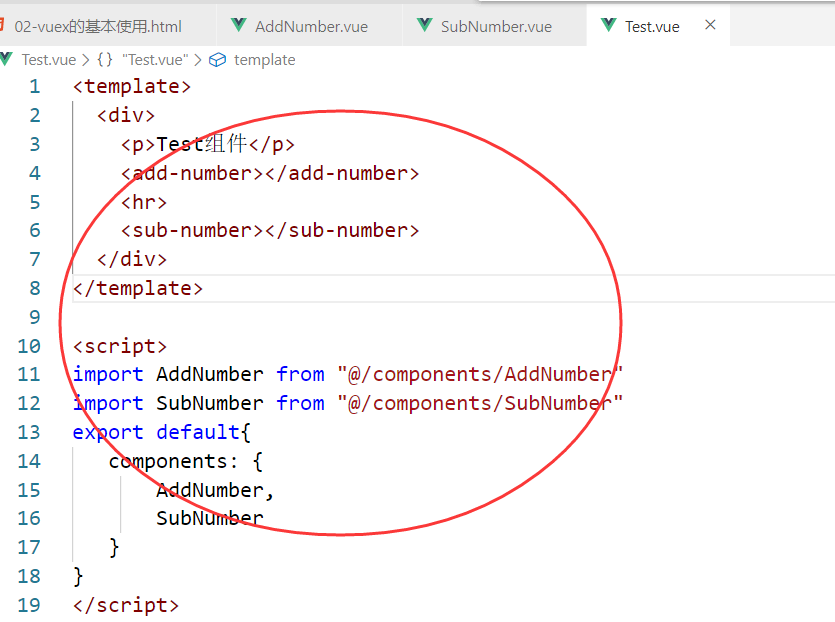
1. 在vue实例中：显示 counter的值
2. 在子组件AddNumber中：显示 counter的值；添加counter的值
3. 在子组件subNumber中：显示 counter的值；减少counter的值

### 创建两个组件，并使用

通过单文件组件的方式去创建：



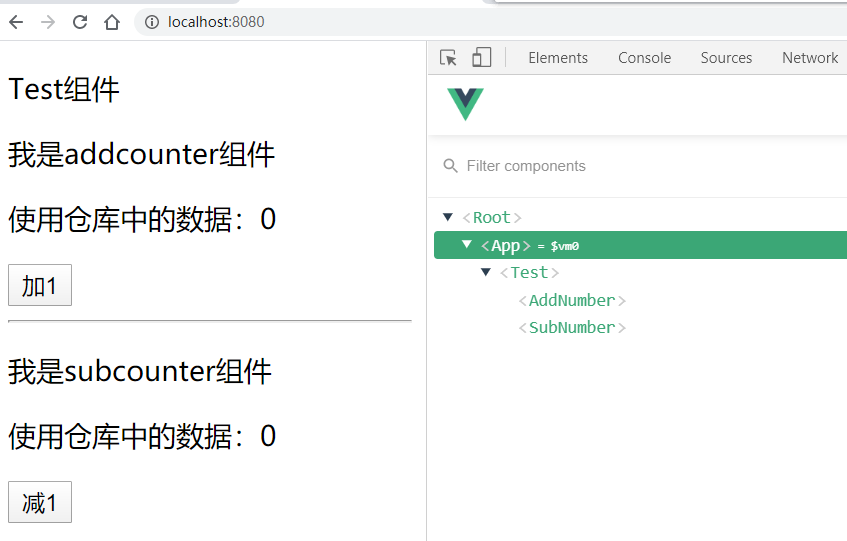
去使用这两个组件：在Test.vue组件去用



在App.vue中使用Test组件如下：



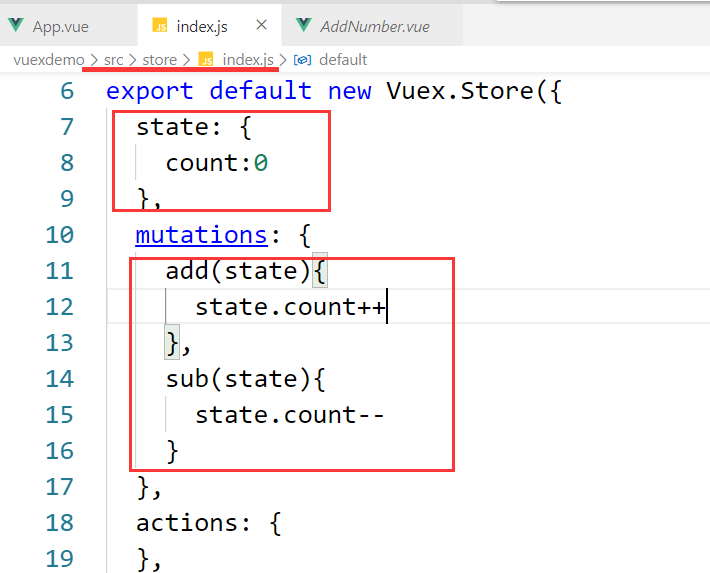
效果如下：



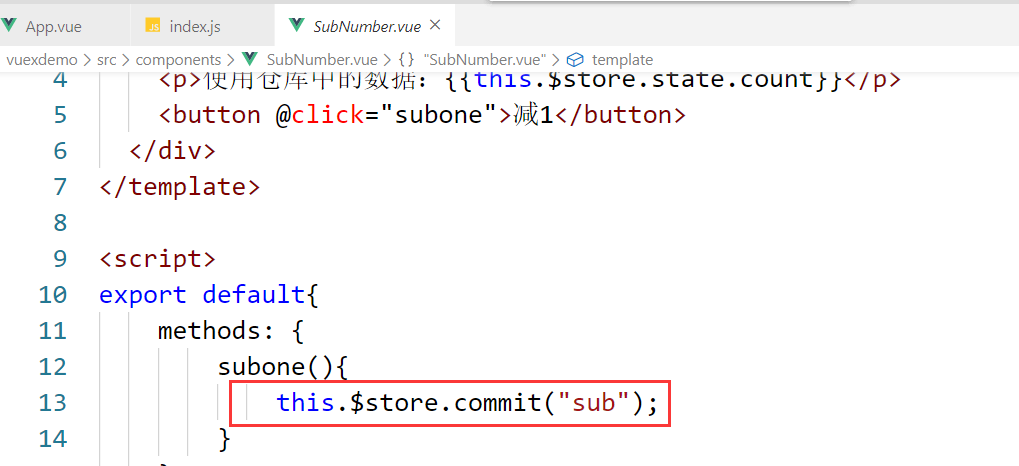
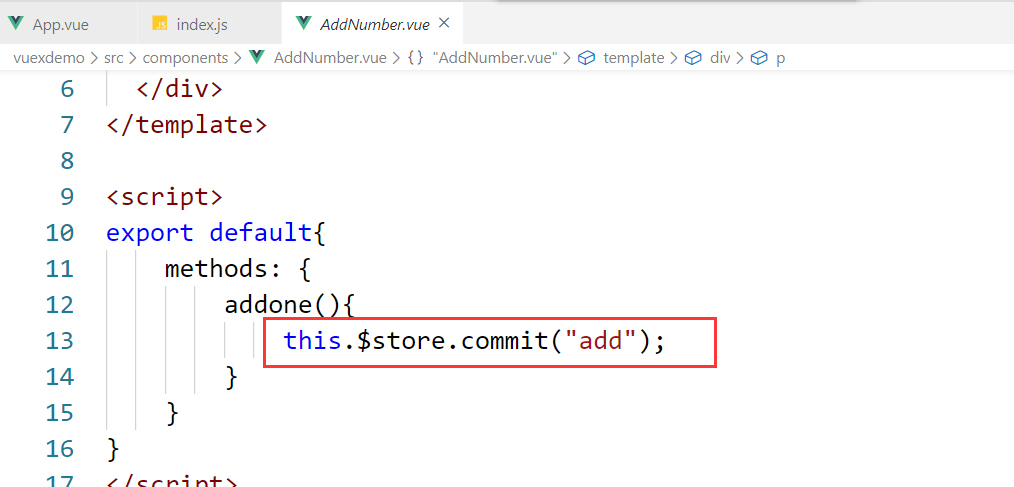
### 下面要实现加1，减1

这两个操作相当是要去修改vuex中的数据，我们必须要通过间接的方式去调用mutations中的方法，达到修改数据的目的。

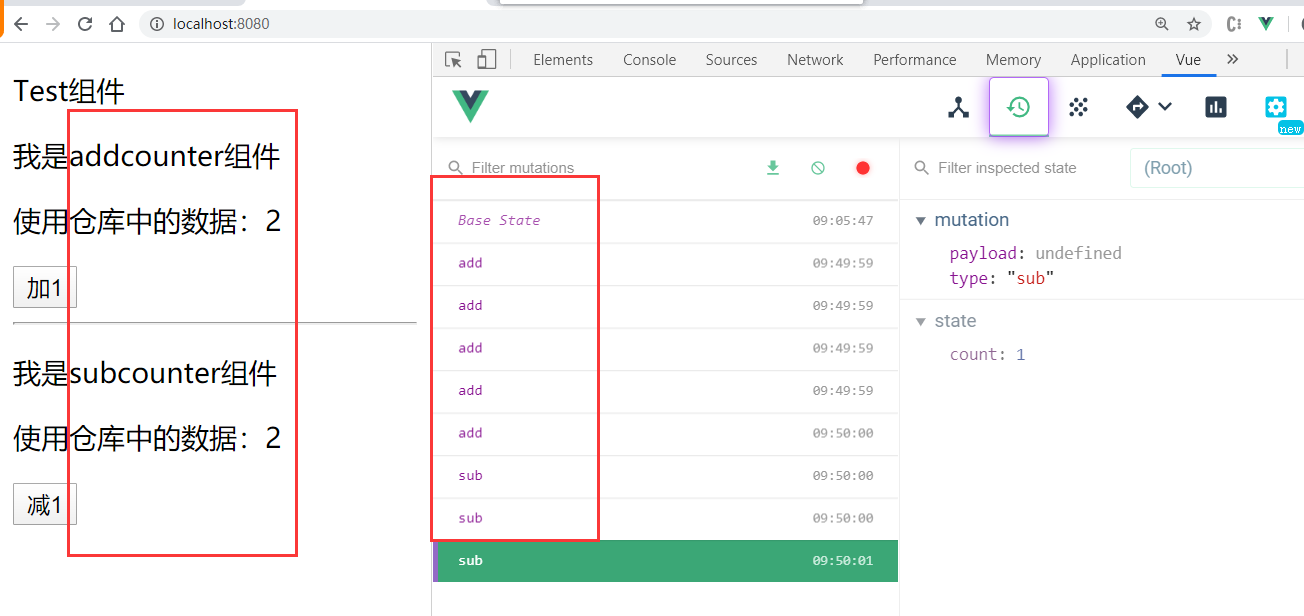
#### 首先，去vuex的实例去增加mutations。



#### 分别在两个组件中去commit事件



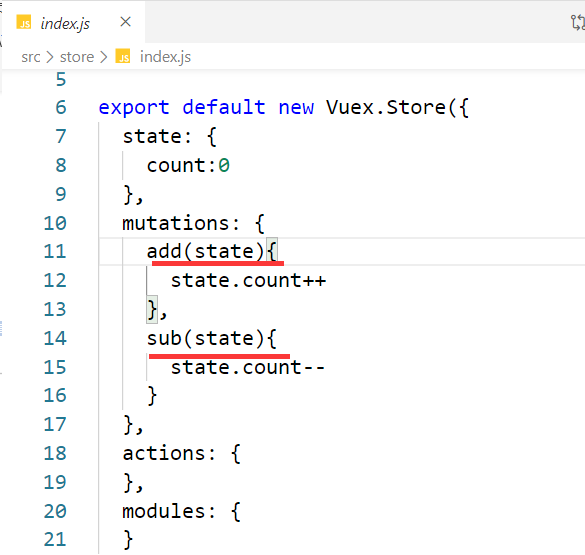
效果如下：



This.$store.commit(“add”)之后中，由于在vuex的mutations中已经注册了add这个事件，所以,commit之后，add对应的那个函数就会执行，在这个函数去修改store的state中的数据，这个修改会被调试工具记录下来。

## mutations是可以有第二个参数的

前面写mutaions我们的函数中默认第一个参数就是state。如下：



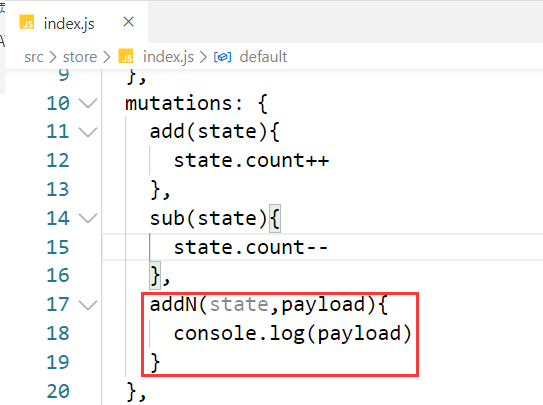
你去commit事件时，第一个参数不需要指定。

但，你也可以额外传递第二个参数 。

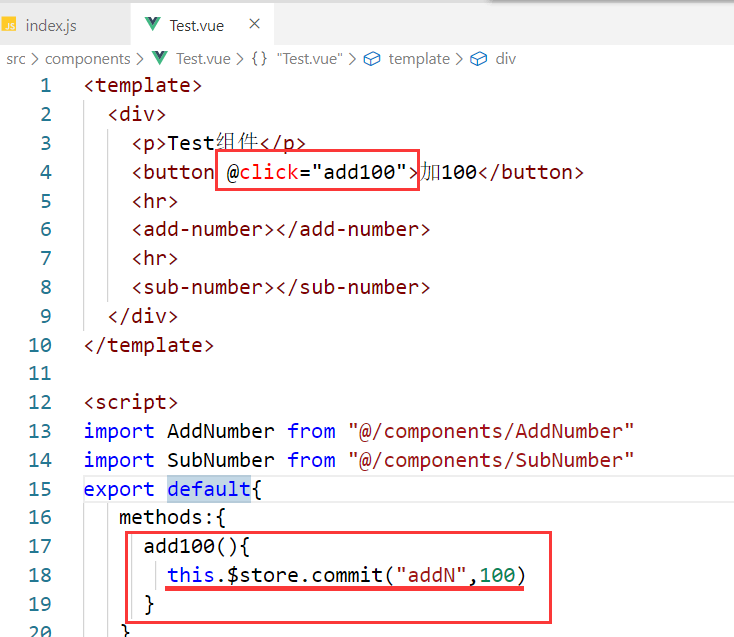
示例如下：

现在，希望在test.vue中添加一个按钮，一次给counter+100，则编写代码如下：

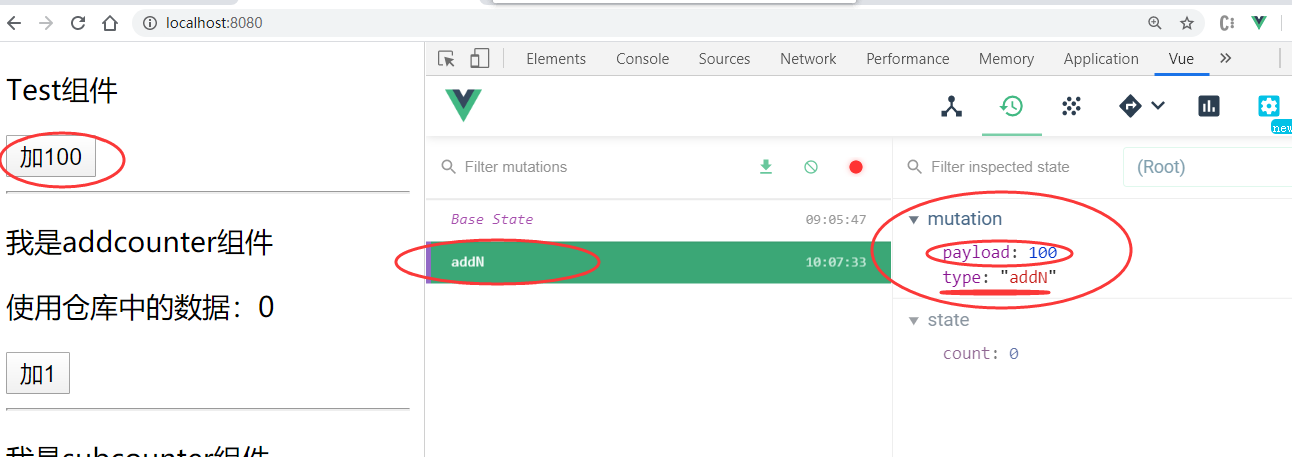
### 在mutations中加一个方法



### 在Test.vue中添加按钮的click

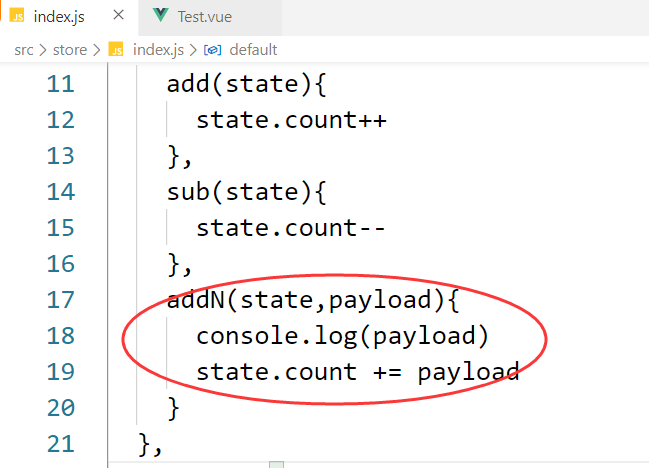


效果如下 ：

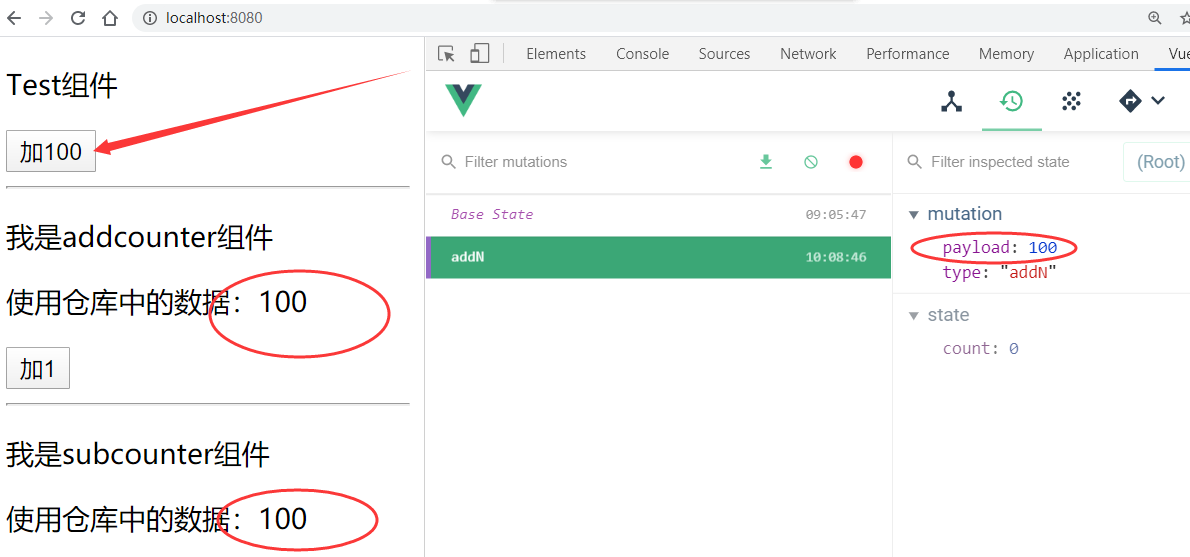


通过调试器，我们可以看到第二个参数已经传递过去了。

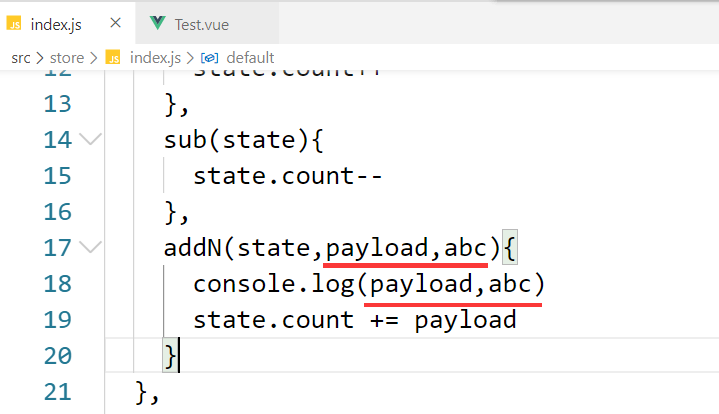
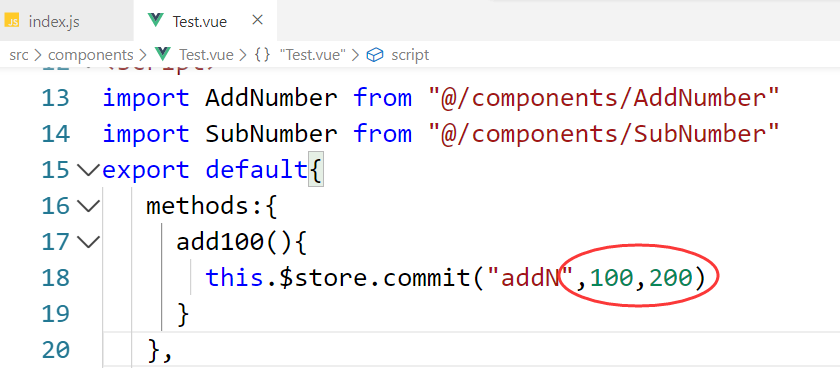
再次修改一下：

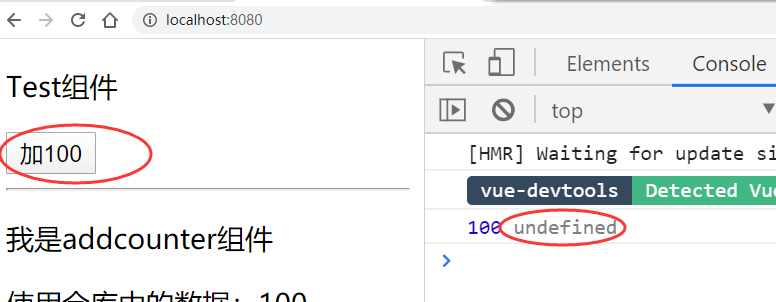


效果如下：



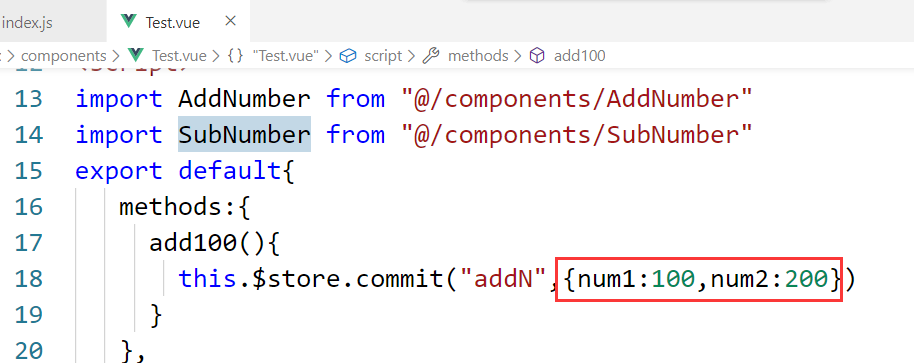
## Mutations 它不能有第三个参数

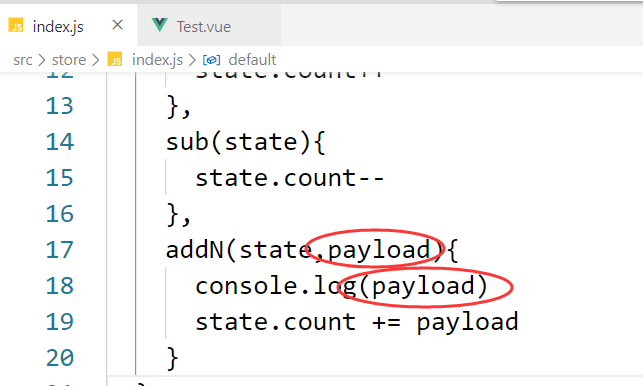


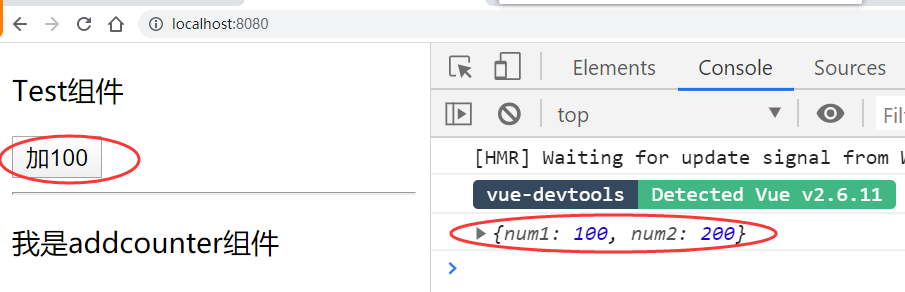


**mutations中不能有第三个参数。**

如果你非传递多个数据项，则可以用一个对象来传，如下：







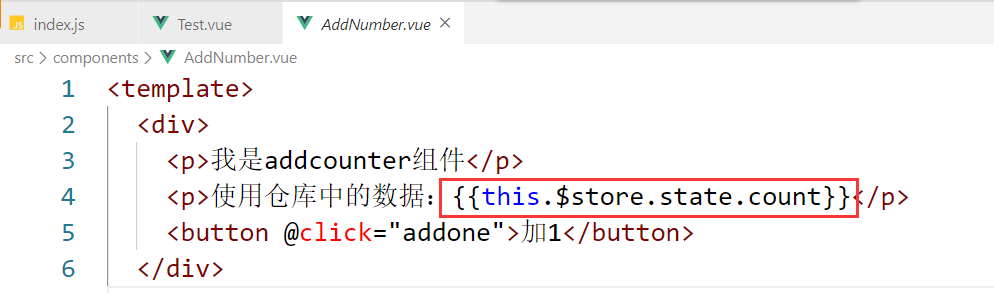
# vuex的四个概念

* state:放数据。在组件之间共享
* mutations:修改数据 修改数据的唯一途径
* getters:从state中的数据中，取出一部分来，依据数据项产生新的结果。类似于vue实例中的computed（计算属性）。
* actions:在对数据实现异步操作时，要用的

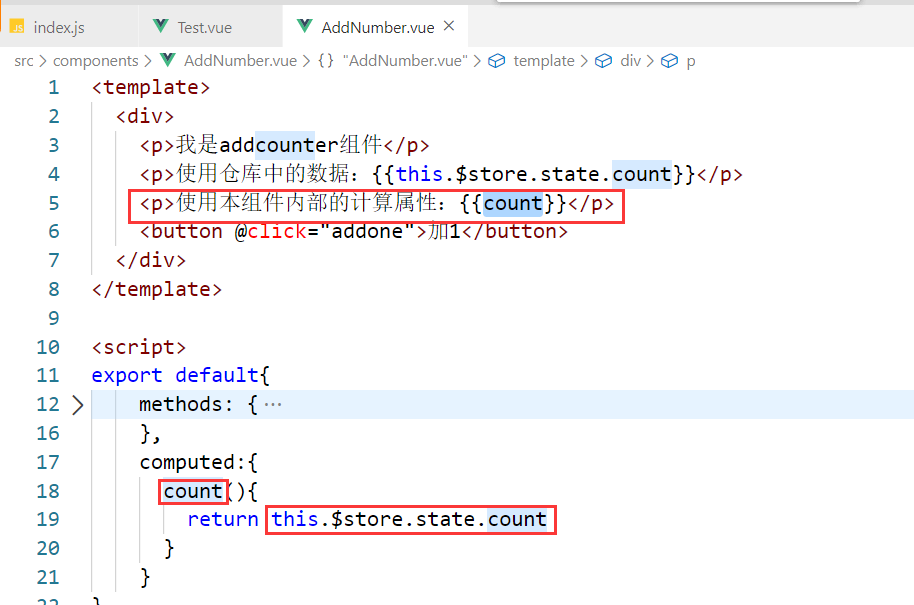
## state

### 在组件中应该如何去获取保存在vuex.中的数据

#### 方法一：this.$store.state.xxx

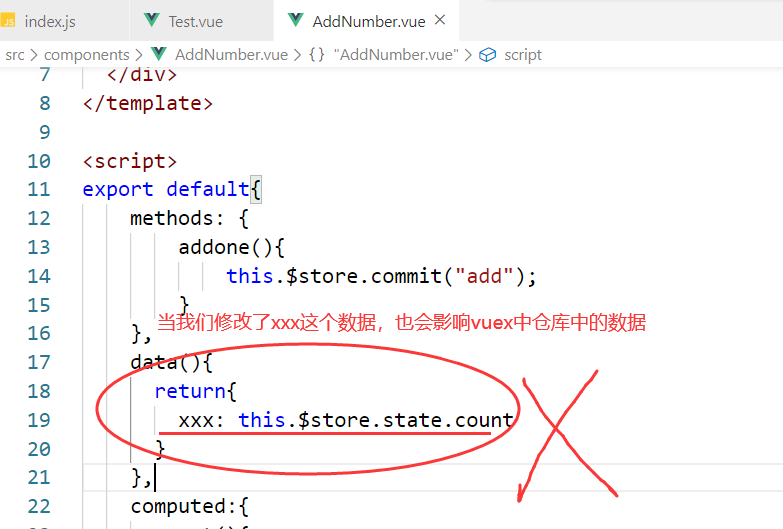


#### 方法二：变通一下，在组件内部用一个计算属性来获取数据。



**不要设置成组件的data。**为什么可以设置为计算属性，而不要设置成data？

原因是：计算属性一般就是只读的：通过它去获取数据。而data是可以直接修改的，如果你绑定到data中的数据项是引用数据类型，则对data的修改也会影响到vuex中的数据：

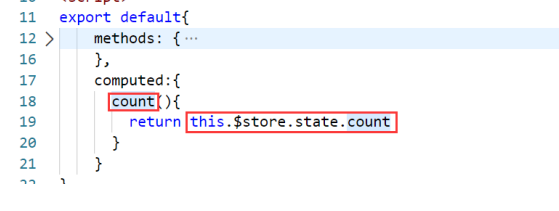


#### 方法三：mapState

this.$store.state.amount写起来比较麻烦。在vuex中提供一个**便捷的写法**：mapState。

它的作用：是把写在vuex.store.state中的数据直接映射到组件中计算属性中。

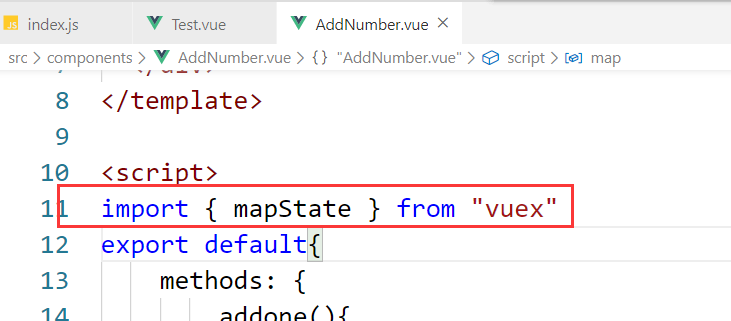
上面的映射是自己手动映射的，也不方便，如下：



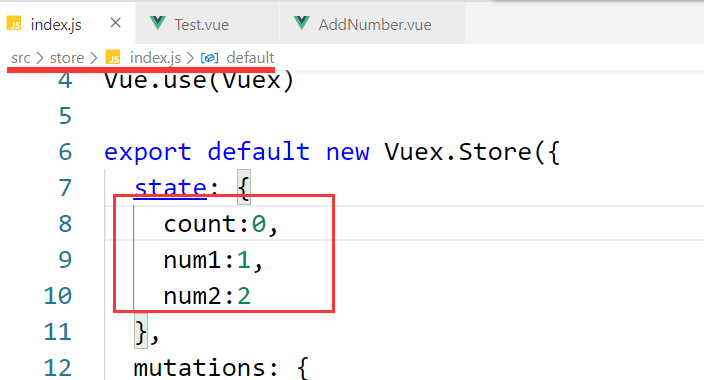
上面写法有一个特点：组件中的计算属性的名字与vuex中store中的数据项的**名字是一致**的，这样，我们特别推荐采用 “绿色通道”的方式：mapState.

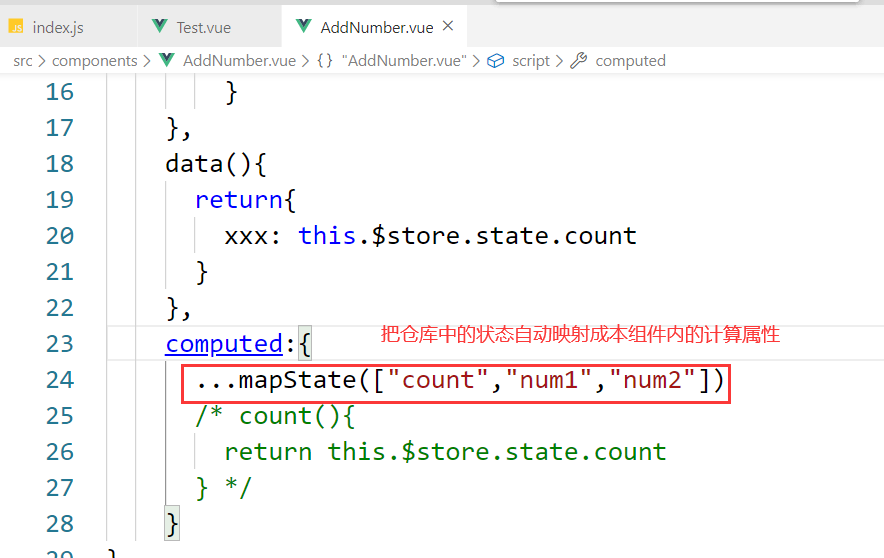
做法如下：

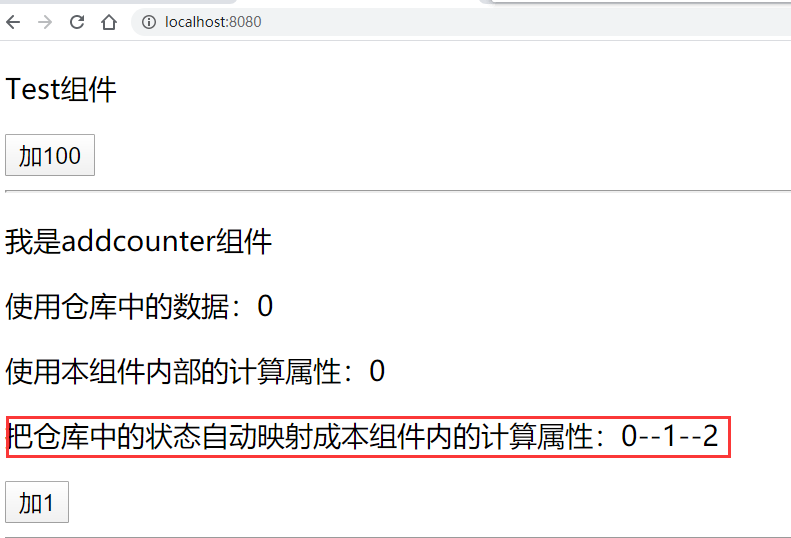
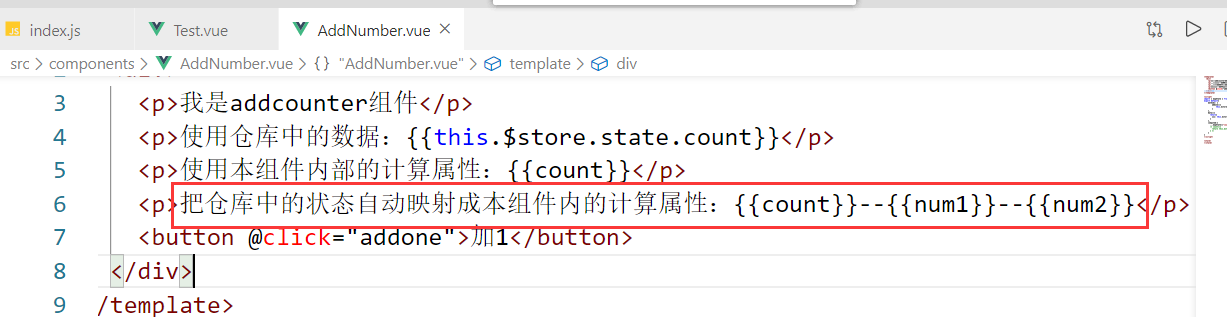
1. 在组件的script中，先导入一个特殊的函数mapState.



1. 直接在**组件计算属性**中：



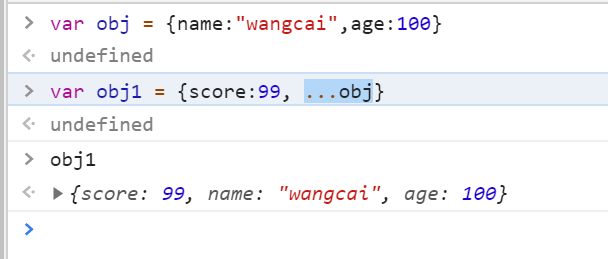




理解：

mapState([“count”,”num1”,”num2”])：mapState是一个函数，这里就是调用这个函数，实参是一个数组[“count”,”num1”,”num2”]。它的返回值是一个对象。

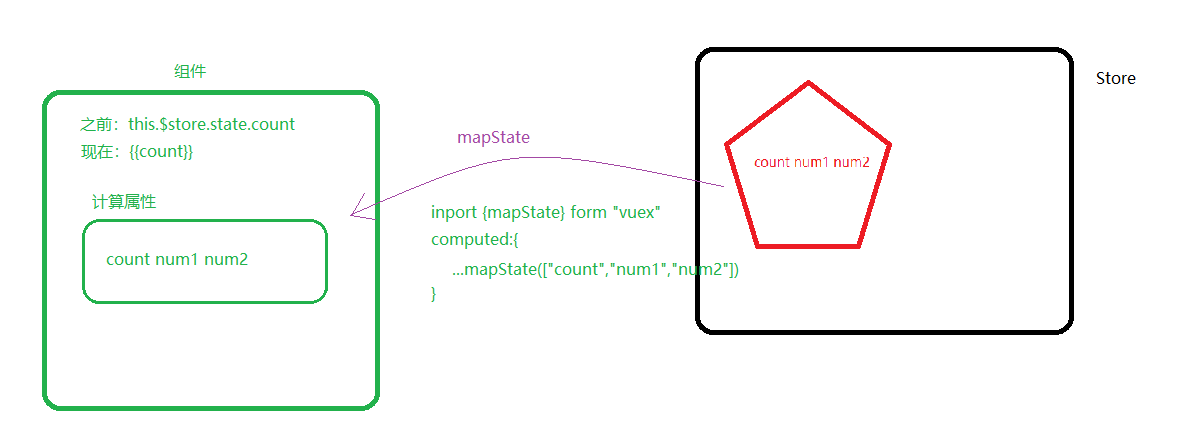
...mapState([“count”,”num1”,”num2”]):把一个对象进行“拆分”，称为拓展运算符，如下：



整个来看：computed是一个属性名，属性值是一个对象，其中的组成是对mapState()这个函数的返回值进行拓展运算。

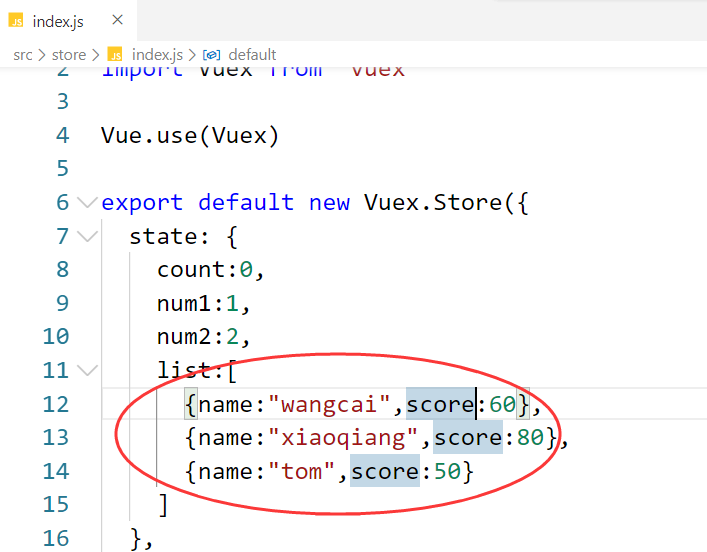
这样就可以直接在组件内部把arr和amount当作计算属性来用。

注意：如果计算属性还有其它的函数，则需要用,与...masState分隔开。



## Getters

有时希望在state中的数据基础上，派生出一些其它的数据。例如：

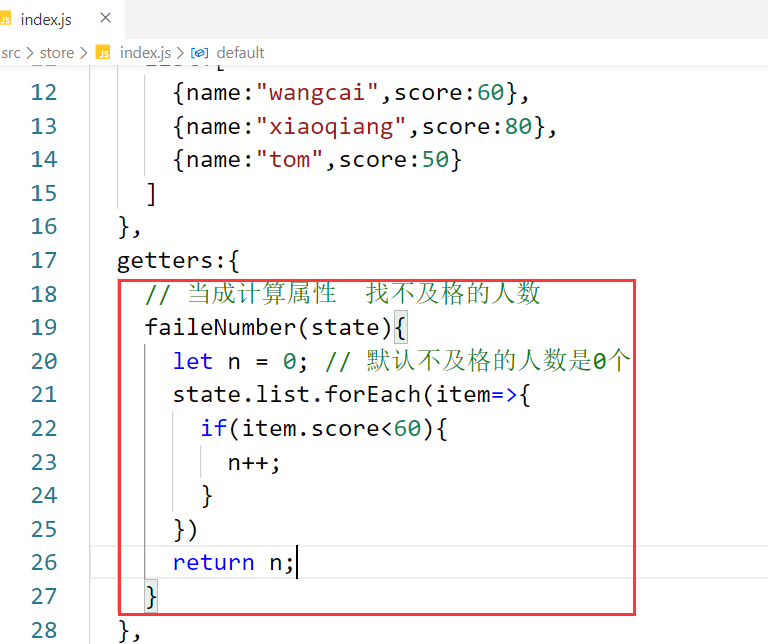


* 我们想知道有多少人的分数小于60分？
* 想知道最高的分数是多少分？

这时，就需要用到getters。

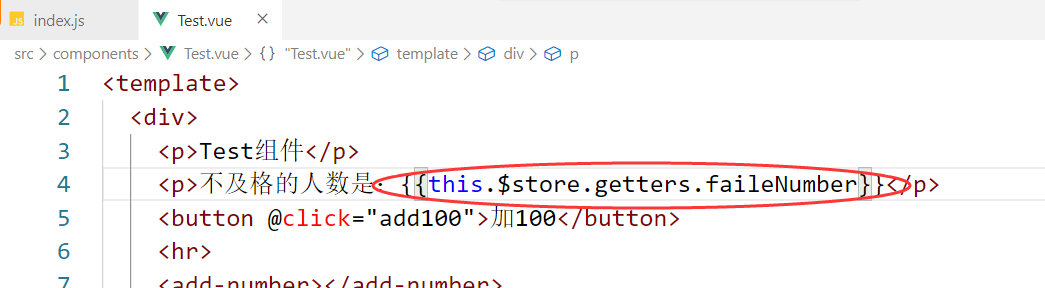
### 示例-- 在vuex定义getters

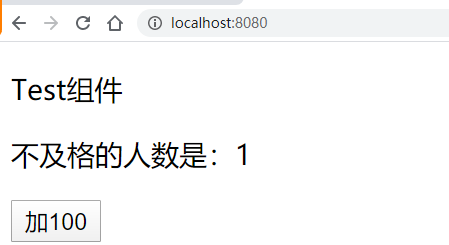
它接收state作为第一个参数。



### 示例-- 在组件中使用getters

#### 方法一：this.$store.getters.failedNumber



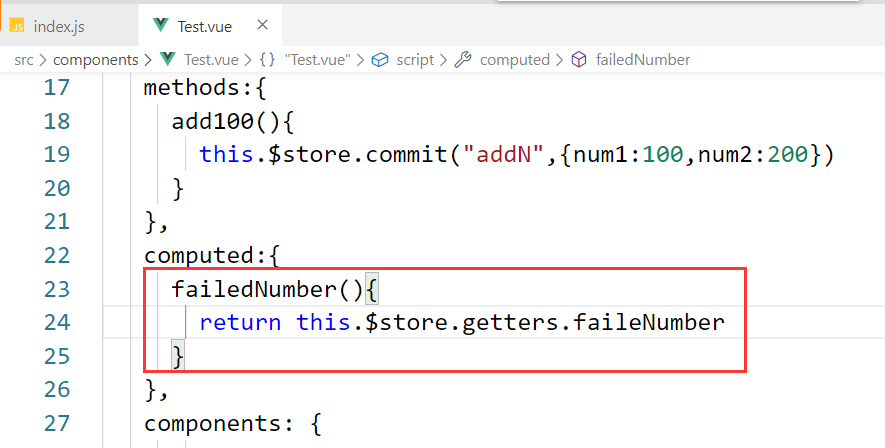


这种写法太长了，不好看。

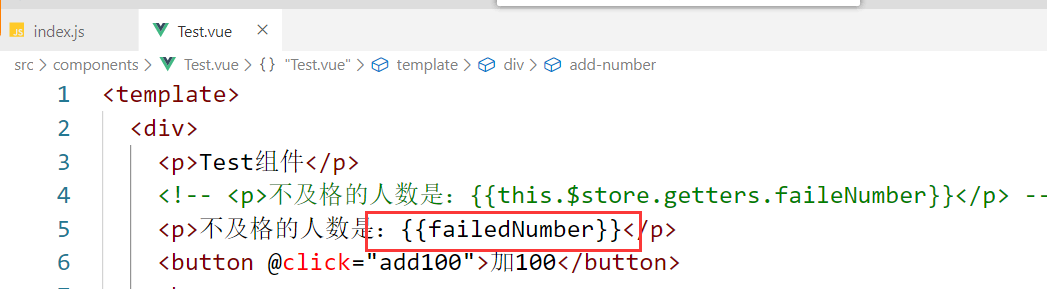
#### 方法二 ：mapGetters

与mapState类似，它的作用也是用来帮助我们去简化代码。我们如果直接定义一个计算属性也是可以的。

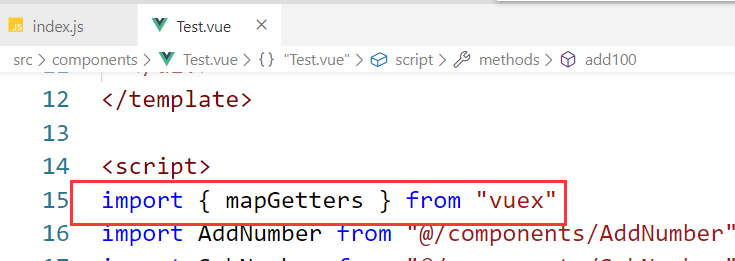
如下：



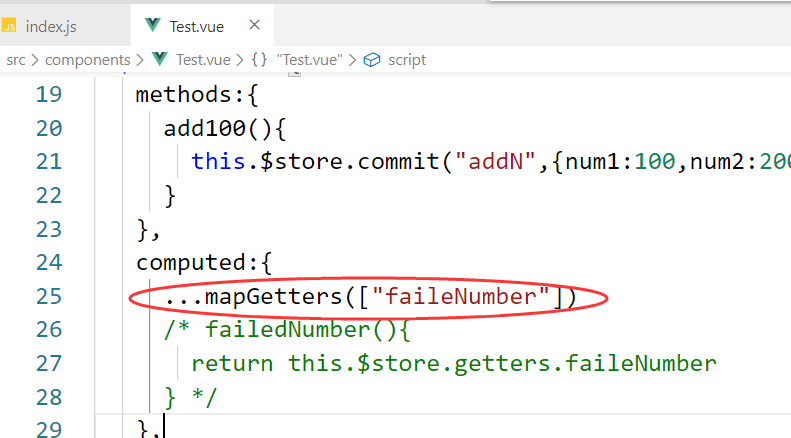
使用时就使用这个计算属性，如下：



这样，在计算属性中名字与 vuex.store.getters中的名字一致，则可以“绿色通道”:mapGetters



如果在当前的组件中，你希望计算属性用另一个名字，则可以这样：



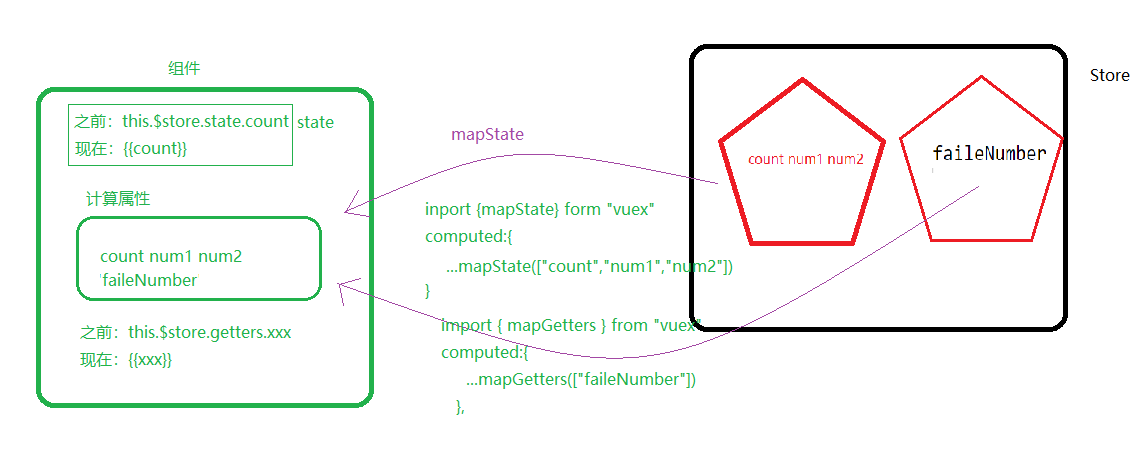
使用之，如下：



效果如下：



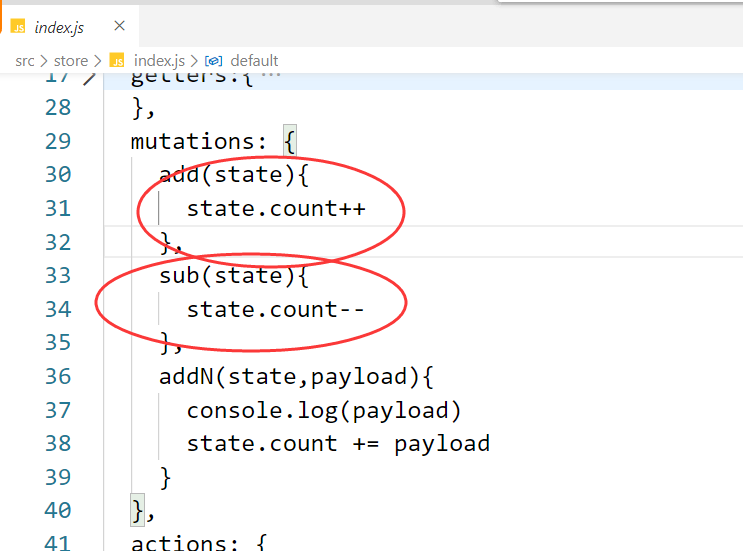
#### 小结



## Mutations

作用是：它是唯一的用来修改数据的工具。

### 定义



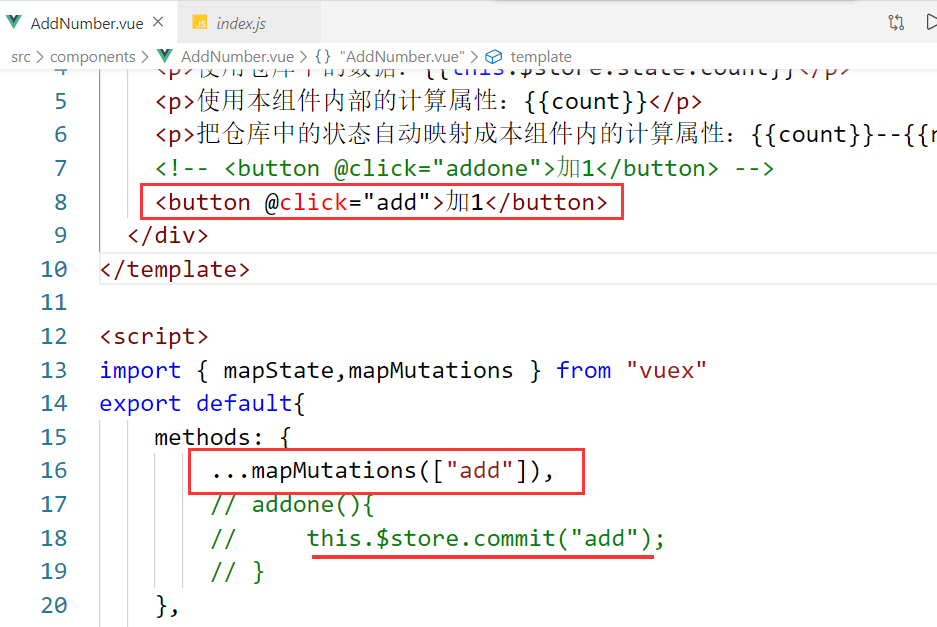
### 在组件中使用

#### 方法一:this.$store.commit

显式地通过this.$store.commit(“add”);



#### 方法二：mapMutations



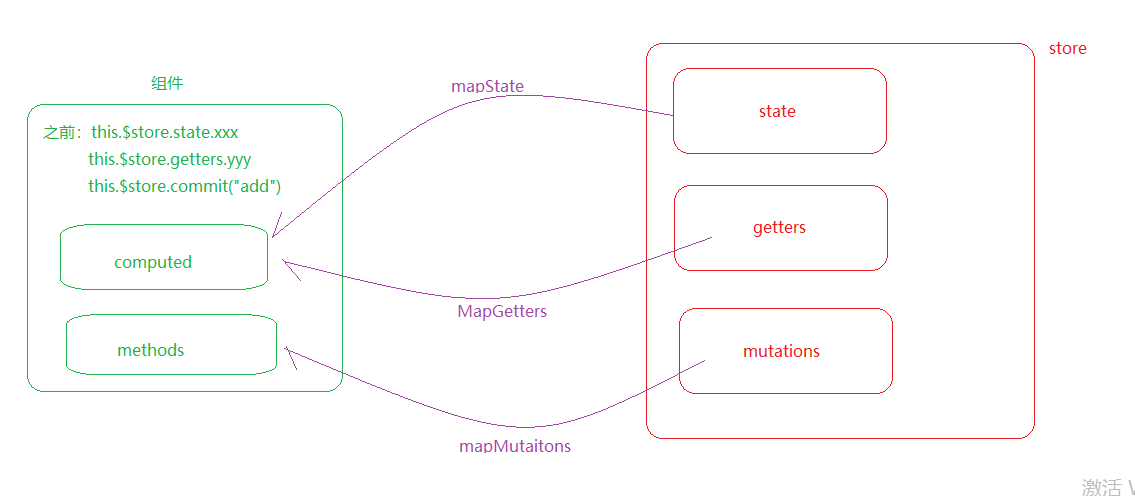
在methods中，映射store中的add。



等价于：

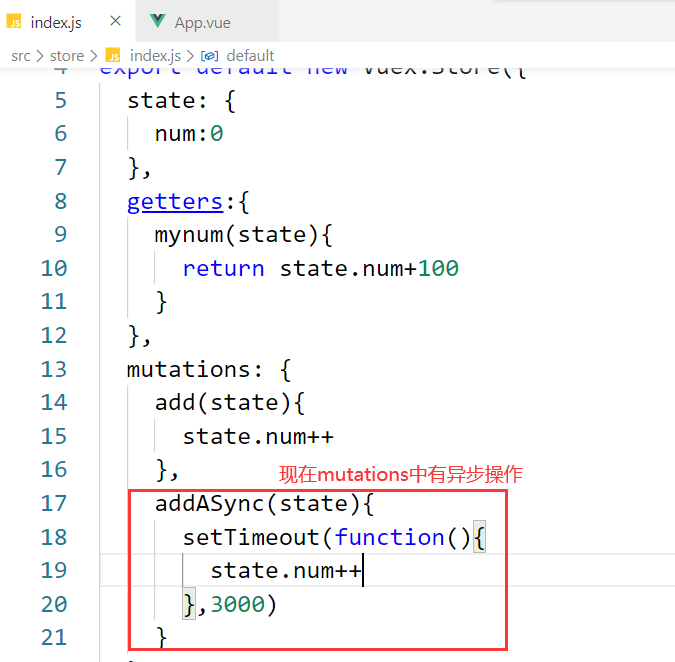


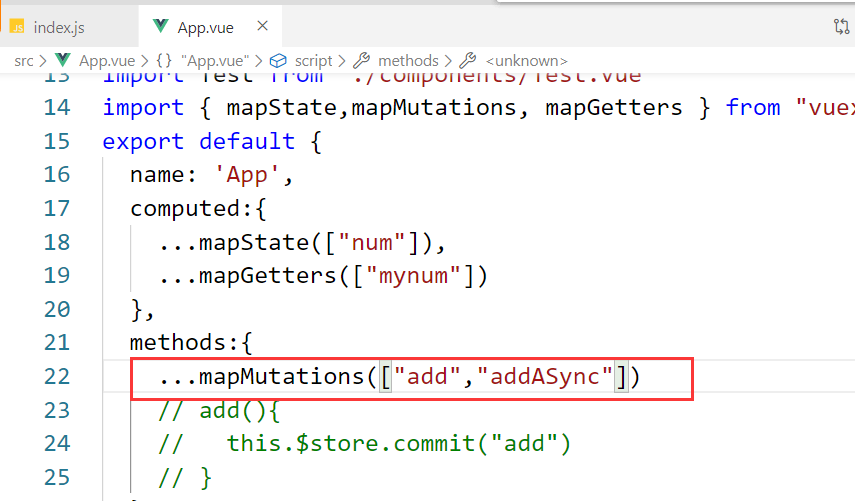
#### 小结

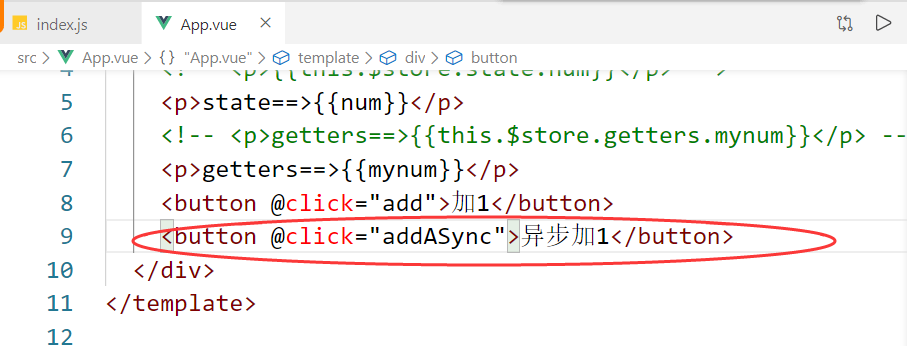


## Actions

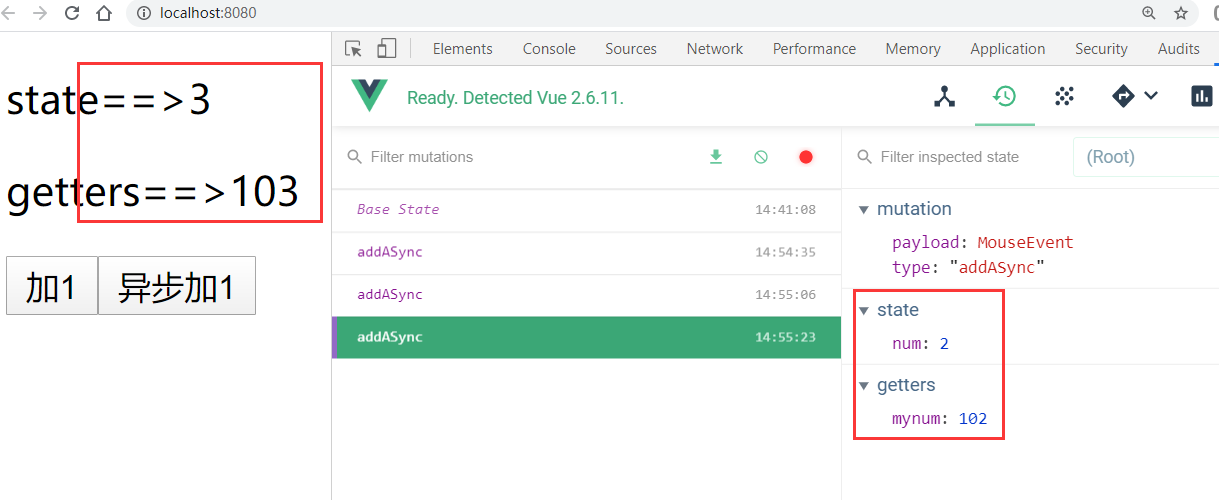
在mutations中，如果操作是异步的。如下：







addASync之后，3s才去增加数值，此时会有一个问题：



在调试器中观察到的recording与实际数值变化的两个时间节点是**不一致**的。以上为例， 记录显示：我们在在11:31:38时，点击了按钮，执行addAmount，但实际的结果是这个数值在11:31:40秒才变化（延迟2000ms）。

会给我们的调试跟踪带来影响，所以针对这种特殊的情况，我们必须要把异步操作写在actions中。

**虽然把异步操作写在actions中，但是改变状态的唯一方式是不变，还是通过mutations。**

### 定义 actions

#### 格式：

actions:{

asyncAdd:function(context,额外参数){

setTimeout(()=>{context.commit("mutaions名")},2000)

}

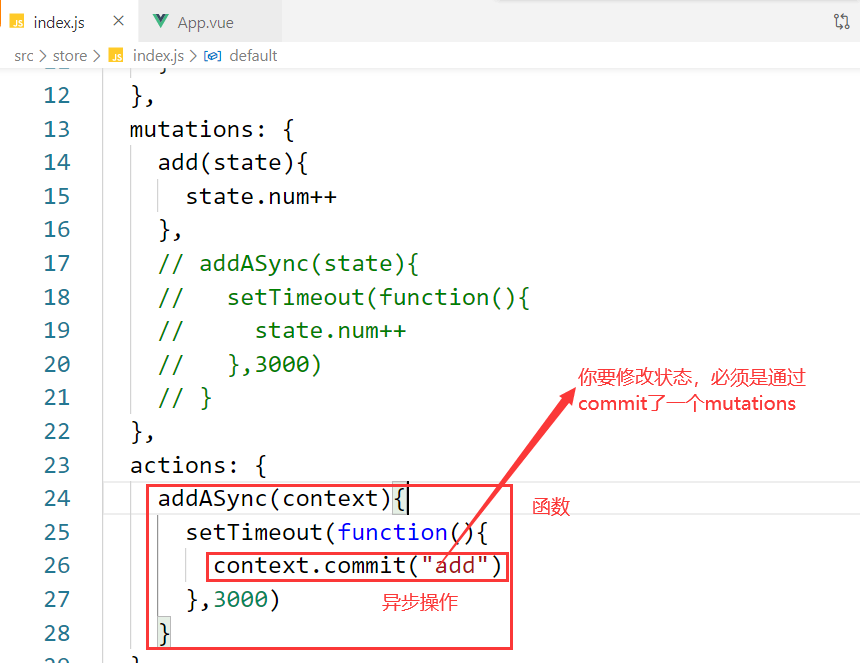
}

在 mutation 中混合异步调用会导致你的程序很难调试。

action类似于mutations，不同在于：

* actions 提交的是 mutations，而不是直接变更状态。
* actions 可以包含任意异步操作。
* action 函数接受一个与 store 实例具有相同方法和属性的 context 对象，可以通过context.commit提交，也可以通过context.state获取state。

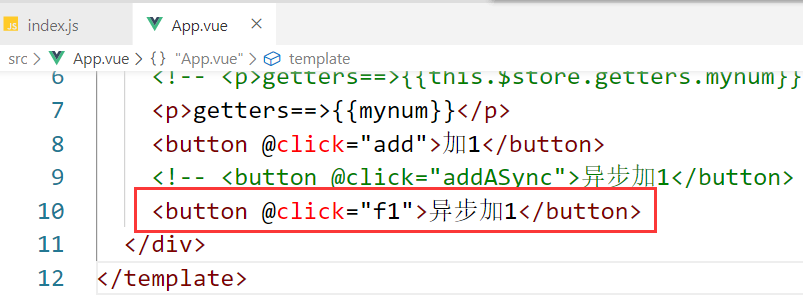
#### 示例



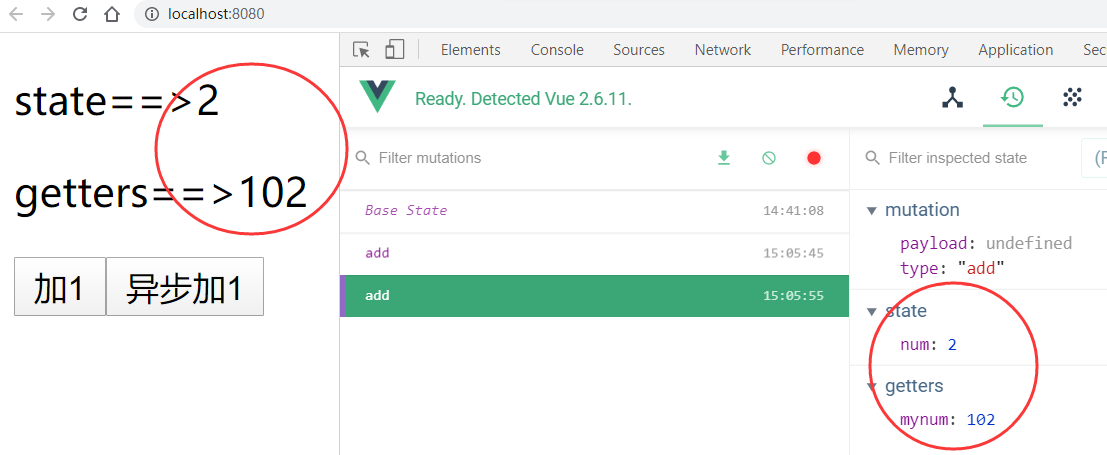
### 在组件中使用actions

在组件中使用actions有两种方法：

#### this.$store.**dispatch**(“actions名”)



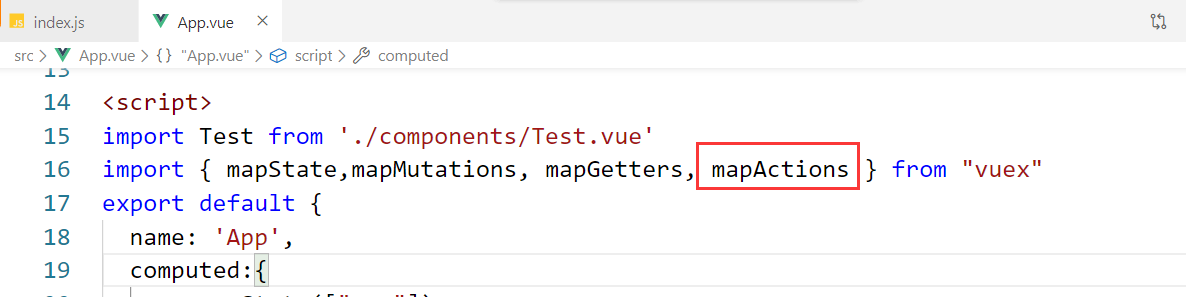
此时，我们去调试面板中观察：

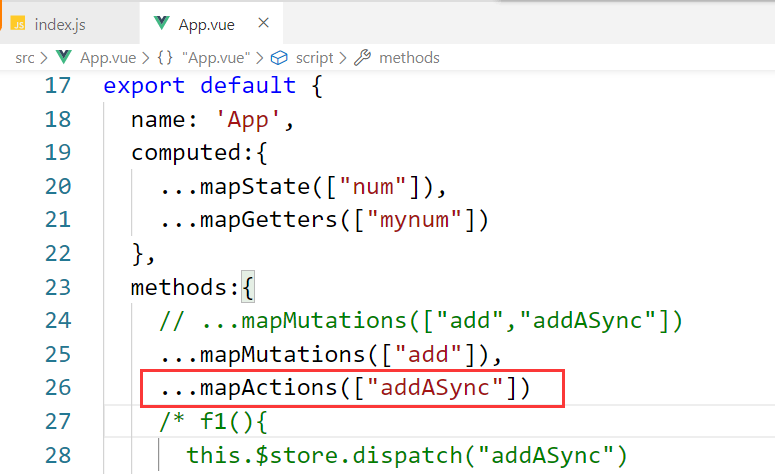


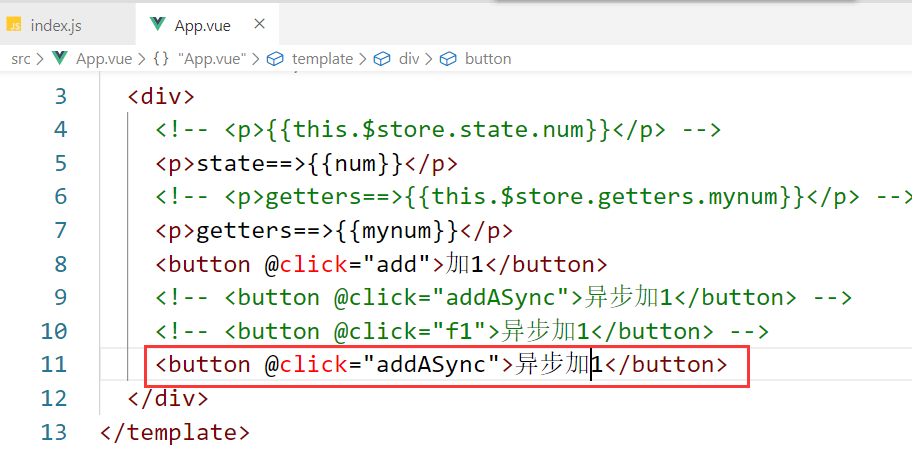
此时，数据的变化与recording中的记录就是同步的。

#### mapActions

也可以通过mapActions去走“绿色通道”，简化代码。

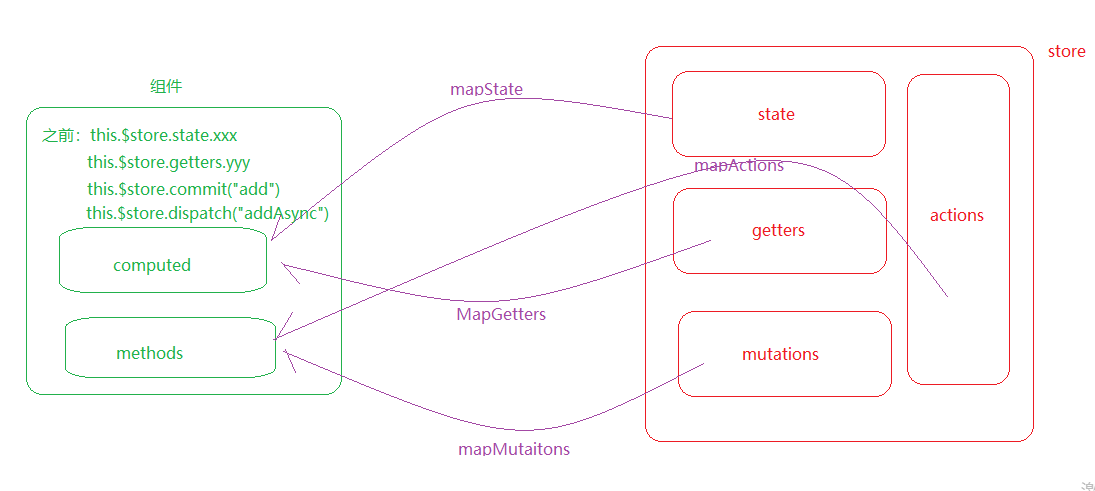






效果OK。

#### 小结



## 小结

import {mapState,mapGetters,mapMutations,mapActions} from "vuex";

map系列方法的作用(难点)。

我们之所以用 map系列的方法，是因为它可以帮助我们去简化代码。

|  |  |  |
| --- | --- | --- |
|  | 不用map | 用map |
| State | 在组件的计算属性中的使用。  computed:{  amount:function(){  return this.$store.state.amount;  }  } | Import {mapState} from “vuex”  Computed{  ...mapState([“amount”]),  } |
| Getters | computed:{  failedNumber:function(){  return this.$store.getters.failedNumber;  }  } | computed:{  ...mapGetters([“failedNumber”]),  } |
| mutations | methods:{  add:function(obj){  return this.$store.commit(“add”,obj);  }  } | methods:{  ...mapMutations([“add”])  } |
| Actions | methods:{  add:function(obj){  return this.$store.dispatch(“add”,obj);  }  } | methods:{  ...mapActions([“add”])  } |

## Store/Index.js的内容的模块化处理

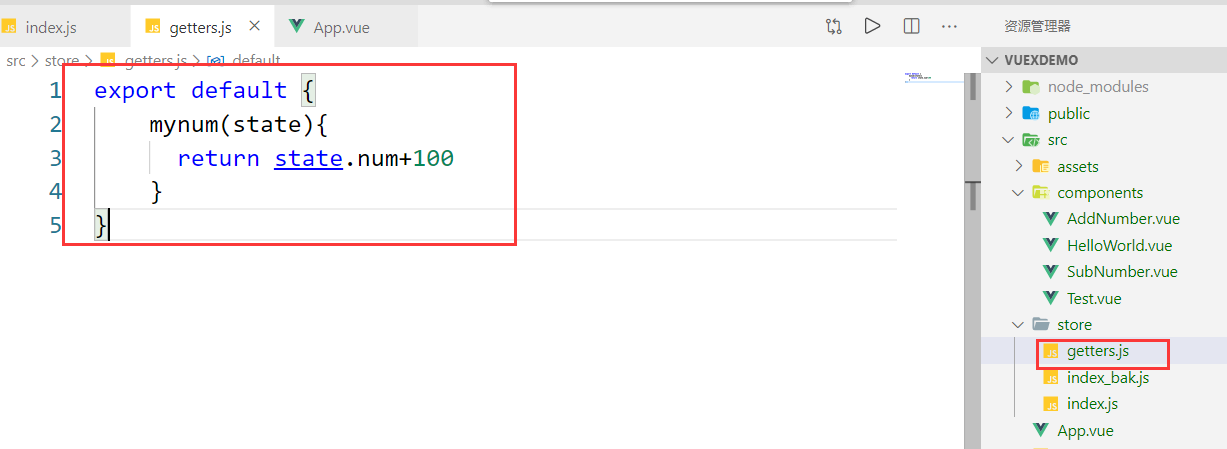
由于Vuex.store中的内容有四大项，可能这个代码全部写在一个index.js中就会很长。也不好管理。

所以，我们在有必要的情况下，可以对这个index.js这个大文件进行拆分：

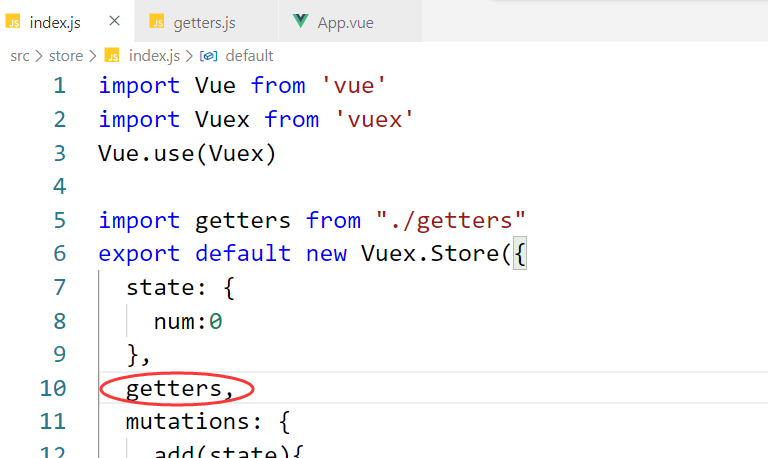
### 以getters为例

把getters拆出去。

1. 新建一个单独的js文件，叫getters.js



1. 在index.js中引入这个模块



其它的抽离是一样的。

## Module

