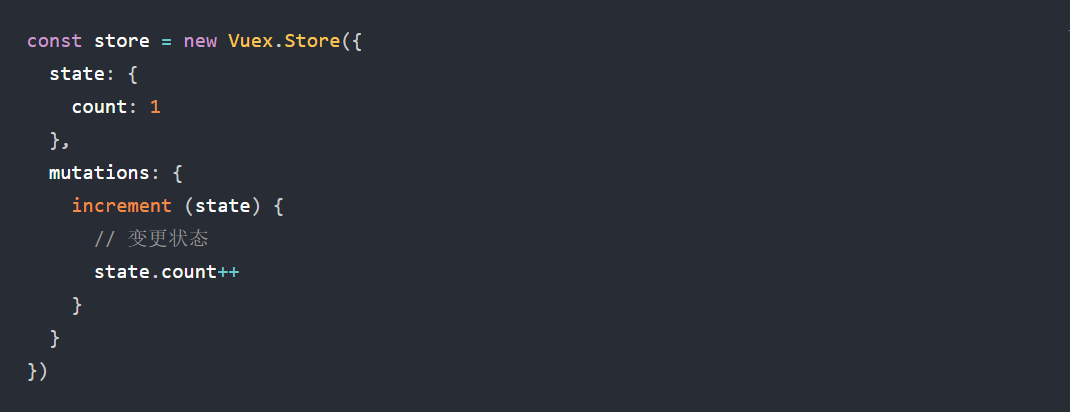
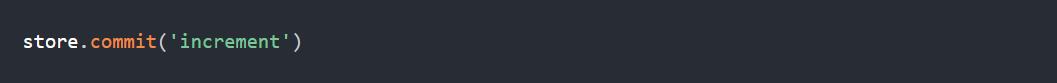
**一、mutations**

更改Vuex的store中的状态的唯一方法是提交mutation。Vuex中的mutation 非常类似于事件：每个mutation都有一个字符串的**事件类型(type)**和一个**回调函数(handler)**。这个回调函数就是我们实际进行状态更改的地方，并且它会接受state作为第一个参数：

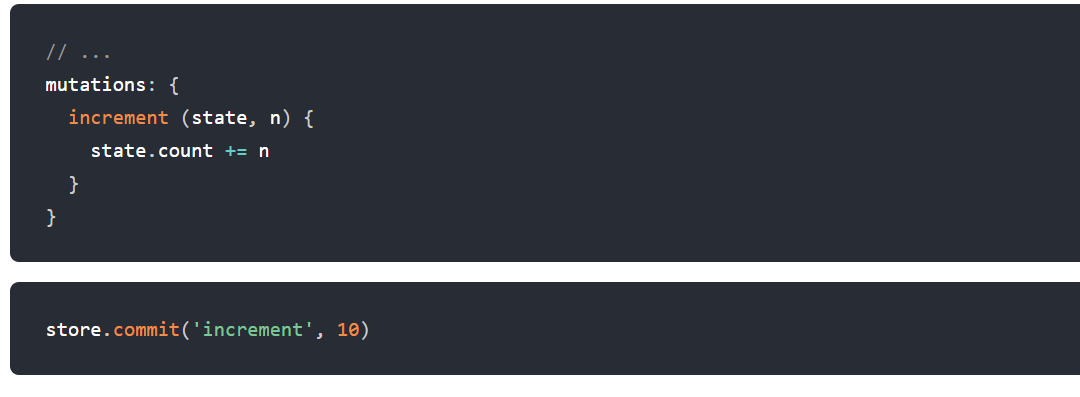


你不能直接调用一个mutation handler。这个选项更像是事件注册：“当触发一个类型为increment的mutation时，调用此函数。”要唤醒一个mutation handler，你需要以相应的type调用**store.commit**方法：

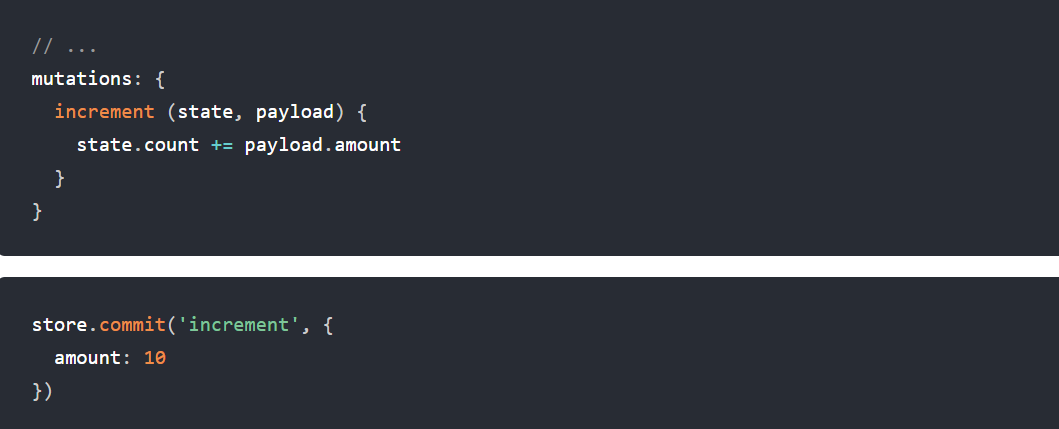


### [#](https://vuex.vuejs.org/zh/guide/mutations.html#%E6%8F%90%E4%BA%A4%E8%BD%BD%E8%8D%B7%EF%BC%88payload%EF%BC%89) 提交载荷（Payload）

你可以向store.commit传入额外的参数，即mutation的**载荷（payload）**：

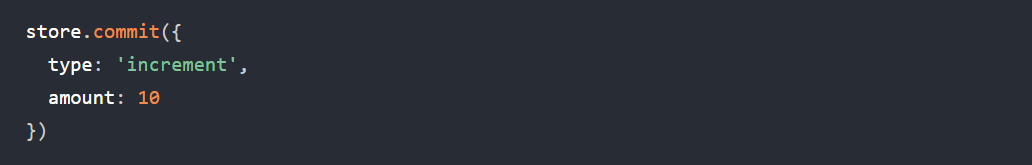


在大多数情况下，载荷应该是一个对象，这样可以包含多个字段并且记录的 mutation会更易读：

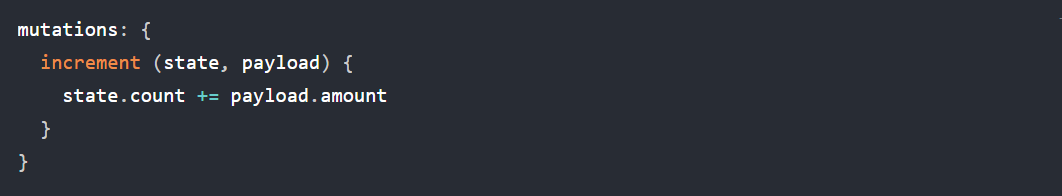


### [#](https://vuex.vuejs.org/zh/guide/mutations.html#%E5%AF%B9%E8%B1%A1%E9%A3%8E%E6%A0%BC%E7%9A%84%E6%8F%90%E4%BA%A4%E6%96%B9%E5%BC%8F) 对象风格的提交方式

提交mutation的另一种方式是直接使用包含type属性的对象：



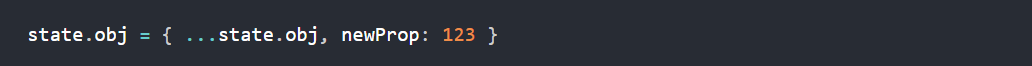
当使用对象风格的提交方式，整个对象都作为载荷传给mutation函数，因此 handler保持不变：



### [#](https://vuex.vuejs.org/zh/guide/mutations.html#mutation-%E9%9C%80%E9%81%B5%E5%AE%88-vue-%E7%9A%84%E5%93%8D%E5%BA%94%E8%A7%84%E5%88%99) Mutation 需遵守 Vue 的响应规则

既然Vuex的store中的状态是响应式的，那么当我们变更状态时，监视状态的 Vue组件也会自动更新。这也意味着Vuex中的mutation也需要与使用Vue一样遵守一些注意事项：

* 最好提前在你的store中初始化好所有所需属性。
* 当需要在对象上添加新属性时，你应该
* 使用Vue.set(obj, 'newProp', 123), 或者
* 以新对象替换老对象。例如，利用 stage-3 的对象展开运算符我们可以这样写：



### [#](https://vuex.vuejs.org/zh/guide/mutations.html#%E4%BD%BF%E7%94%A8%E5%B8%B8%E9%87%8F%E6%9B%BF%E4%BB%A3-mutation-%E4%BA%8B%E4%BB%B6%E7%B1%BB%E5%9E%8B) 使用常量替代 Mutation 事件类型

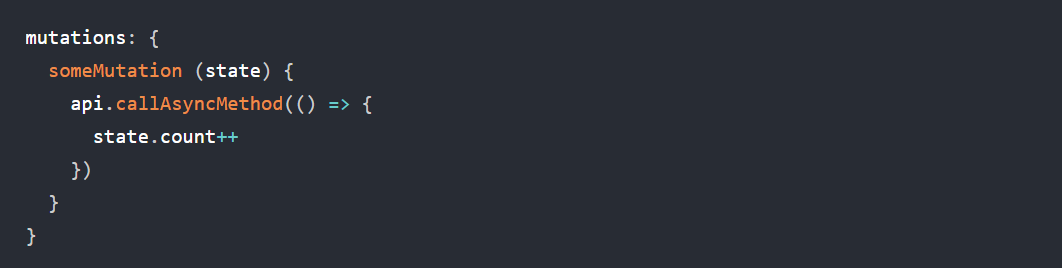
使用常量替代mutation事件类型在各种Flux实现中是很常见的模式。这样可以使linter之类的工具发挥作用，同时把这些常量放在单独的文件中可以让你的代码合作者对整个app包含的mutation一目了然：



用不用常量取决于你——在需要多人协作的大型项目中，这会很有帮助。但如果你不喜欢，你完全可以不这样做。

### [#](https://vuex.vuejs.org/zh/guide/mutations.html#mutation-%E5%BF%85%E9%A1%BB%E6%98%AF%E5%90%8C%E6%AD%A5%E5%87%BD%E6%95%B0) Mutation 必须是同步函数

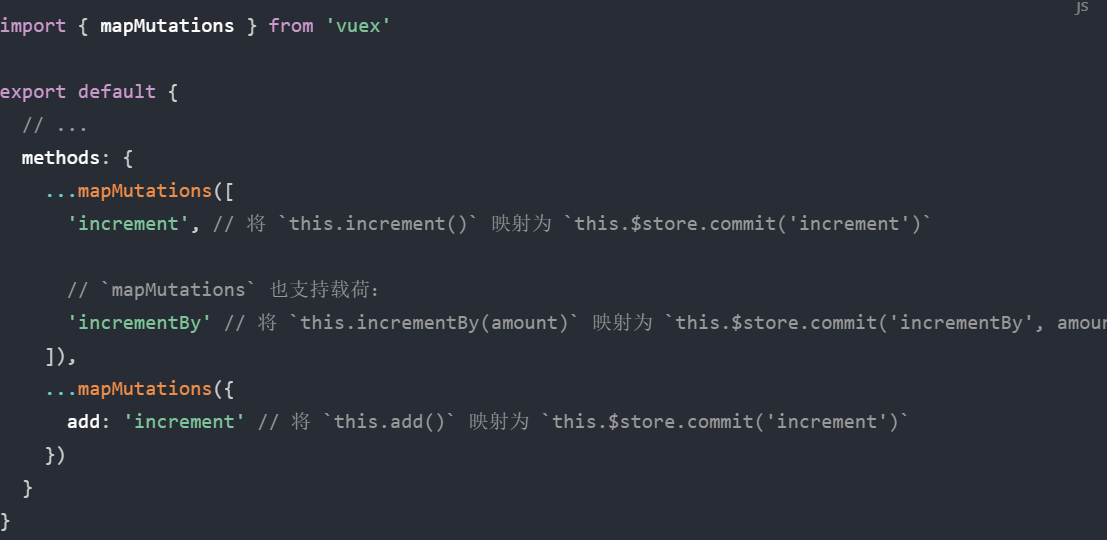
一条重要的原则就是要记住**mutation必须是同步函数**。为什么？请参考下面的例子：



现在想象，我们正在debug一app 并且观察devtool中的mutation日志。每一条 mutation 被记录，devtools都需要捕捉到前一状态和后一状态的快照。然而，在上面的例子中mutation中的异步函数中的回调让这不可能完成：因为当 mutation 触发的时候，回调函数还没有被调用，devtools不知道什么时候回调函数实际上被调用——实质上任何在回调函数中进行的状态的改变都是不可追踪的。

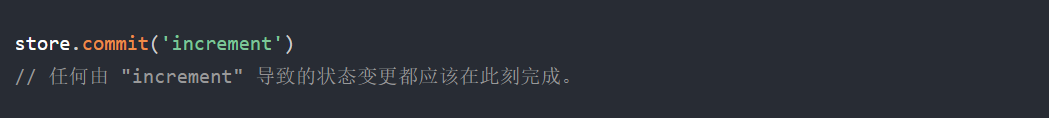
### [#](https://vuex.vuejs.org/zh/guide/mutations.html#%E5%9C%A8%E7%BB%84%E4%BB%B6%E4%B8%AD%E6%8F%90%E4%BA%A4-mutation) 在组件中提交 Mutation

你可以在组件中使用this.$store.commit('xxx')提交mutation，或者使用mapMutations辅助函数将组件中的methods映射为store.commit调用（需要在根节点注入store）。



### [#](https://vuex.vuejs.org/zh/guide/mutations.html#%E4%B8%8B%E4%B8%80%E6%AD%A5%EF%BC%9Aaction) 下一步：Action

在mutation中混合异步调用会导致你的程序很难调试。例如，当你调用了两个包含异步回调的 mutation 来改变状态，你怎么知道什么时候回调和哪个先回调呢？这就是为什么我们要区分这两个概念。在 Vuex 中，**mutation 都是同步事务**。



**二、Vue Router**

**（一）简介**

Vue Router是Vue.js官方的路由管理器。它和Vue.js的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

* 嵌套的路由/视图表
* 模块化的、基于组件的路由配置
* 路由参数、查询、通配符
* 基于 Vue.js 过渡系统的视图过渡效果
* 细粒度的导航控制
* 带有自动激活的 CSS class 的链接
* HTML5 历史模式或 hash 模式，在 IE9 中自动降级
* 自定义的滚动条行为

用Vue.js + Vue Router创建单页应用，是非常简单的。使用Vue.js，我们已经可以通过组合组件来组成应用程序，当你要把Vue Router添加进来，我们需要做的是，将组件 (components)映射到路由(routes)，然后告诉Vue Router在哪里渲染它们。下面是个基本例子：

* HTML:



* JavaScript：



通过注入路由器，我们可以在任何组件内通过 this.$router 访问路由器，也可以通过 this.$route 访问当前路由：



（二）场景

1.重定向

* 含义：“重定向”的意思是，当用户访问/a时，URL将会被替换成/b，然后匹配路由为/b。
* 课堂对应场景：点击“Go todo”会访问“welcome”的路径。

2.编程式导航

* 含义：想要导航到不同的URL，则使用router.push方法。这个方法会向history栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的URL。
* 课堂对应场景：在对话框输入“豆豆”，点击“start”即会出现“HELLO 豆豆”页面。

3.路由组件传参

* 含义：在组件中使用$route会使之与其对应路由形成高度耦合，从而使组件只能在某些特定的URL上使用，限制了其灵活性。可使用 props 将组件和路由解耦。
* 课堂对应场景：在对话框输入“豆豆”，会出现“welcome”和“counter”两个路径渲染的页面。

4.嵌套路由

* 含义：实际生活中的应用界面，通常由多层嵌套的组件组合而成。同样地，URL 中各段动态路径也按某种结构对应嵌套的各层组件。一个被渲染组件同样可以包含自己的嵌套 <router-view>，要在嵌套的出口中渲染组件，需要在 VueRouter 的参数中使用 children 配置
* 课堂对应场景：在对话框输入“豆豆”，会出现“Welcome”、“Go to Counter”、“Go todo”三个click以及 “HELLO 豆豆”的字样，此时可点击“Go to Counter”，进入其渲染的页面。

5.命名路由

* 含义：有时候，通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。你可以在创建Router实例的时候，在routes配置中给某个路由设置名称。
* 课堂对应场景：给链接起名字，即“name”，引用时便可直接饮用其name。