Vuex

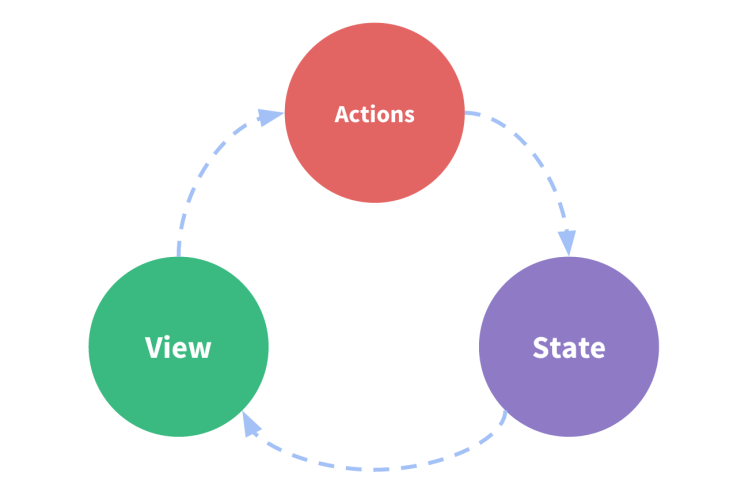
1. Vuex是什么

专为 Vue.js 应用程序开发的**状态管理模式**。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。

1. 什么是“状态管理模式”



以下是一个表示“单向数据流”理念的极简示意:



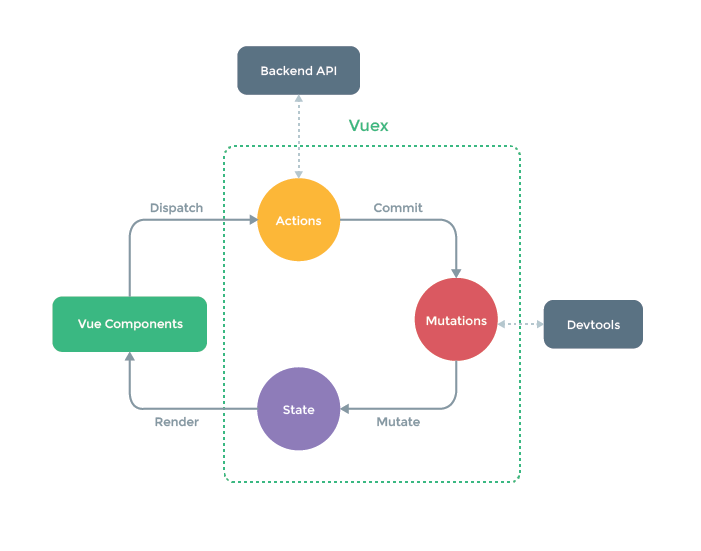
但是，当我们的应用遇到**多个组件共享状态**时，单向数据流的简洁性很容易被破坏：

* 多个视图依赖于同一状态。
* 来自不同视图的行为需要变更同一状态。

我们为什么不把组件的共享状态抽取出来，以一个全局单例模式管理呢？

在这种模式下，**我们的组件树构成了一个巨大的“视图”**，不管在树的哪个位置，任何组件都能获取状态或者触发行为！

通过**定义和隔离状态**管理中的各种概念并强制遵守一定的规则，我们的代码将会变得更结构化且易维护。



1. 什么情况系使用Vuex

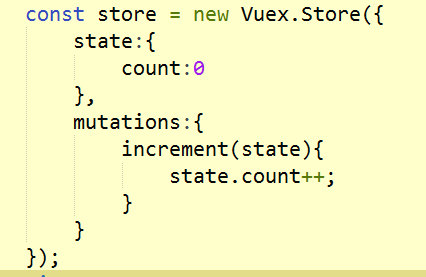
如果您需要构建是一个中大型单页应用，您很可能会考虑如何更好地在组件外部管理状态，Vuex 将会成为自然而然的选择。引用 Redux 的作者 Dan Abramov 的话说就是：

**Flux 架构就像眼镜：您自会知道什么时候需要它.**

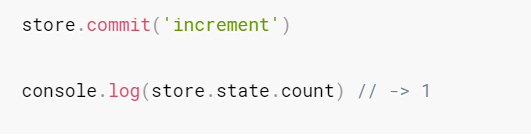
1. 开始

每一个 Vuex 应用的核心就是 store（仓库）。“store”基本上就是一个容器，它包含着你的应用中大部分的**状态 (state)**。Vuex 和单纯的全局对象有以下两点不同：

1. 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
2. 你不能直接改变 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) mutation。**这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。
3. 最简单的Store



可以通过 store.state 来获取状态对象，以及通过 store.commit 方法触发状态变更：



备注：

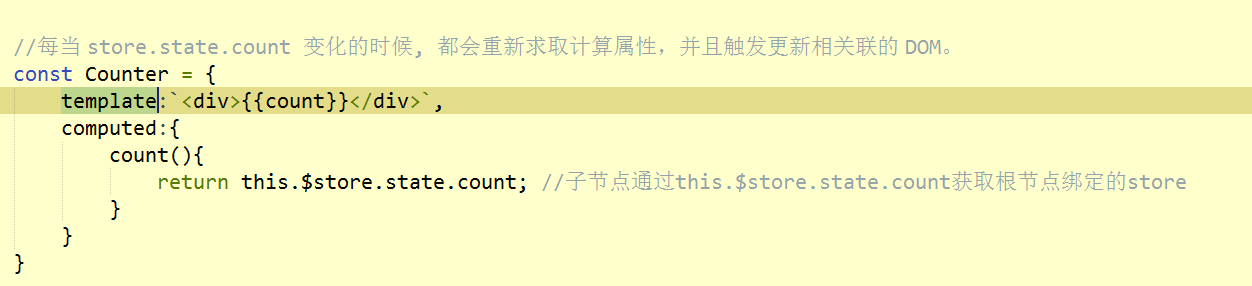
1. 通过提交 mutation 的方式，而非直接改变 store.state.count，是因为我们想要更明确地追踪到状态的变化。
2. 让我们有机会去实现一些能记录每次状态改变，保存状态快照的调试工具。有了它，我们甚至可以实现如时间穿梭般的调试体验。
3. **由于 store 中的状态是响应式的，在组件中调用 store 中的状态简单到仅需要在计算属性中返回即可。触发变化也仅仅是在组件的 methods 中提交 mutation。**
4. State

单一状态树

Vuex 使用**单一状态树，**用一个对象就包含了全部的应用层级状态。至此它便作为一个“唯一数据源 ([**SSOT**](https://en.wikipedia.org/wiki/Single_source_of_truth))”而存在。**每个应用将仅仅包含一个 store 实例。**单一状态树让我们能够直接地定位任一特定的状态片段，在调试的过程中也能轻易地取得整个当前应用状态的快照。

在Vue组件中获取Vuex状态

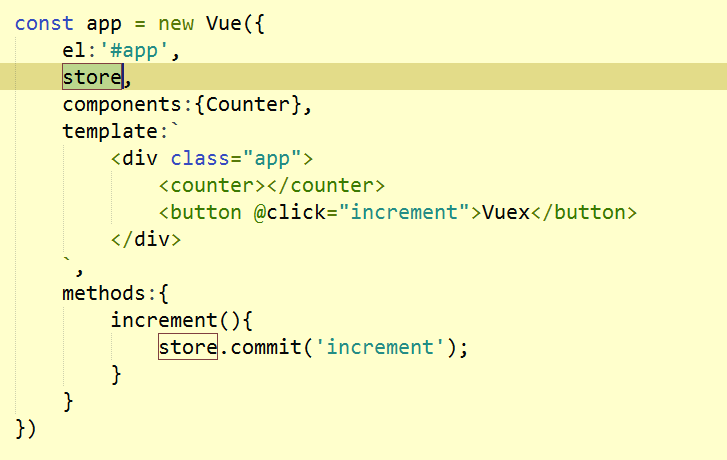
由于 Vuex 的状态存储是响应式的，从 store 实例中读取状态最简单的方法就是在[**计算属性**](https://cn.vuejs.org/guide/computed.html)中返回某个状态：



每当 store.state.count 变化的时候, 都会重新求取计算属性，并且触发更新相关联的 DOM。

**然而，这种模式导致组件依赖全局状态单例。在模块化的构建系统中，在每个需要使用 state 的组件中需要频繁地导入，并且在测试组件时需要模拟状态。**

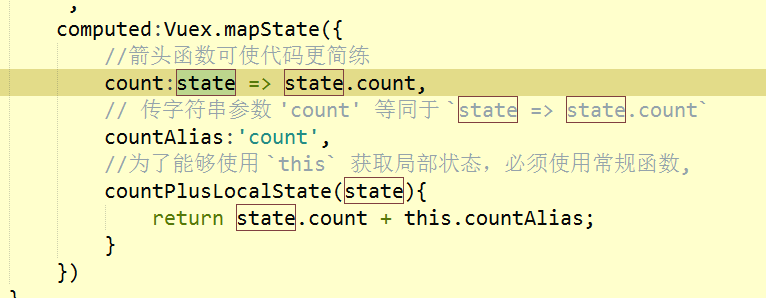
**Vuex 通过 store 选项，提供了一种机制将状态从根组件“注入”到每一个子组件中（需调用 Vue.use(Vuex)）**：



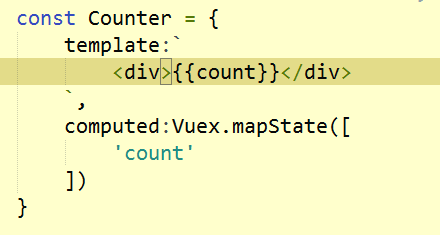
通过在根实例中注册 store 选项，该 store 实例会注入到根组件下的所有子组件中，且子组件能通过 this.$store访问到。让我们更新下 Counter 的实现：

**mapState辅助函数**

当一个组件需要获取多个状态时候，将这些状态都声明为计算属性会有些重复和冗余。为了解决这个问题，我们可以使用 mapState 辅助函数帮助我们生成计算属性，让你少按几次键：



当映射的计算属性的名称与 state 的子节点名称相同时，我们也可以给 mapState 传一个字符串数组。

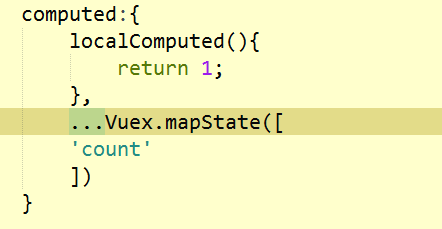


**对象展开运算符**

mapState 函数返回的是一个对象。我们如何将它与局部计算属性混合使用呢？

通常，我们需要使用一个工具函数将多个对象合并为一个，以使我们可以将最终对象传给 computed 属性。

mapState 函数返回的是一个对象。我们如何将它与局部计算属性混合使用呢？通常，我们需要使用一个工具函数将多个对象合并为一个，以使我们可以将最终对象传给 computed 属性。但是自从有了[**对象展开运算符**](https://github.com/sebmarkbage/ecmascript-rest-spread)（现处于 ECMASCript 提案 stage-3 阶段），我们可以极大地简化写法：

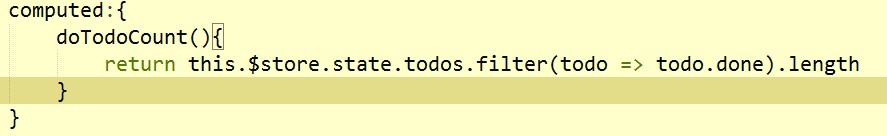


**组件仍然保有局部状态**

使用 Vuex 并不意味着你需要将**所有的**状态放入 Vuex。虽然将所有的状态放到 Vuex 会使状态变化更显式和易调试，但也会使代码变得冗长和不直观。如果有些状态严格属于单个组件，最好还是作为组件的局部状态。你应该根据你的应用开发需要进行权衡和确定。

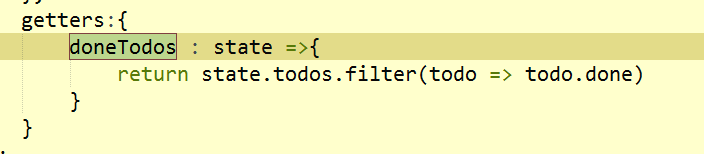
1. Getter

有时候我们需要从 store 中的 state 中派生出一些状态，例如对列表进行过滤并计数：



如果有多个组件需要用到此属性，我们要么复制这个函数，或者抽取到一个共享函数然后在多处导入它——无论哪种方式都不是很理想。

Vuex 允许我们在 store 中定义“getter”（可以认为是 store 的计算属性）。就像计算属性一样，getter 的返回值会根据它的依赖被缓存起来，且只有当它的依赖值发生了改变才会被重新计算。

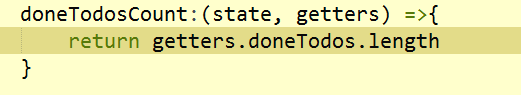


Getter 会暴露为 store.getters 对象：

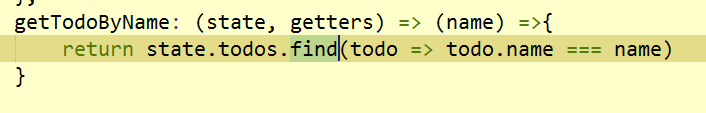


Getter 也可以接受其他 getter 作为第二个参数：

接受getters作为第二个参数



让 getter 返回一个函数，来实现给 getter 传参，在你对 store 里的数组进行查询时非常有用





mapGetters 辅助函数

mapGetters 辅助函数仅仅是将 store 中的 getter 映射到局部计算属性：

