# VueX

## 1. 概念

- 每一个 Vuex 应用的核心就是 store（仓库）
- 1.1 Vuex 和单纯的全局对象的不同
  - Vuex 的状态存储（store）是响应式的
  - 约定：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**
- Vuex 使用**单一状态树**——是的，用一个对象就包含了全部的应用层级状态
- 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)`）：

```
const app = new Vue({
  el: '#app',
  // 把 store 对象提供给 “store” 选项，这可以把 store 的实例注入所有的子组件
  store,
  components: { Counter },
  template: `
    <div class="app">
      <counter></counter>
    </div>
  `
})
```

  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

- `mapState` 辅助函数：

```
// 在单独构建的版本中辅助函数为 Vuex.mapState
import { mapState } from 'vuex'

export default {
  // ...
  computed: mapState({
    // 箭头函数可使代码更简练
```

```
    count: state => state.count,

    // 传字符串参数 'count' 等同于 `state => state.count`
    countAlias: 'count',

    // 为了能够使用 `this` 获取局部状态，必须使用常规函数
    countPlusLocalState (state) {
      return state.count + this.localCount
    }
  })
}
```

或者：

```
computed: mapState([
  // 映射 this.count 为 store.state.count
  'count'
])
```

- 对象展开运算符：

```
computed: {
  localComputed () { /* ... */ },
  // 使用对象展开运算符将此对象混入到外部对象中
  ...mapState({
    // ...
  })
}
```

- Getter 会暴露为 `store.getters` 对象
  - Getter 接受 state 作为其第一个参数
  - Getter 也可以接受其他 getter 作为第二个参数
- getter 返回一个函数：

```
getters: {
  // ...
  getTodoById: (state) => (id) => {
    return state.todos.find(todo => todo.id === id)
  }
}
```

注意，getter 在通过方法访问时，每次都会去进行调用，而不会缓存结果

- `mapGetters` 辅助函数 - 类似 `mapState`

```
computed: {
  // 使用对象展开运算符将 getter 混入 computed 对象中
    ...mapGetters([
      'doneTodosCount',
      'anotherGetter',
      // ...
    ])
  }
```

- 更改 Vuex 的 store 中的状态的唯一方法是提交 mutation - 类似于事件
- mutation 的 **载荷（payload）** - 提交载荷

```
// 定义
mutations: {
  increment (state, n) {
    state.count += n
  }
}

// 使用
store.commit('increment', 10)
```

- 突变提交方式 - 对象

```
store.commit({
  type: 'increment',
  amount: 10
})
```

```
mutations: {
  increment (state, payload) {
    state.count += payload.amount
  }
}
```

- 新对象替换老对象

```
state.obj = { ...state.obj, newProp: 123 }
```

- 使用常量替代 Mutation 事件类型
- Mutation 必须是**同步函数** - #
- `mapMutations` 辅助函数 - 可将组件中的 methods 映射为 `store.commit` 调用 - #
- Action 类似于 mutation，不同在于：
  - Action 提交的是 mutation，而不是直接变更状态。
  - Action 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（但不是store）的 `context` 对象
- Action 通过 `store.dispatch` 方法触发：

```
store.dispatch('increment')
```

- **mapActions**

```
methods: {
    ...mapActions([
      'increment', // 将 `this.increment()` 映射为
`this.$store.dispatch('increment')`

      // `mapActions` 也支持载荷：
      'incrementBy' // 将 `this.incrementBy(amount)` 映射为
`this.$store.dispatch('incrementBy', amount)`
    ]),
    ...mapActions({
      add: 'increment' // 将 `this.add()` 映射为
`this.$store.dispatch('increment')`
    })
  }
```

- **action组合**

```
actions: {
  async actionA ({ commit }) {
    commit('gotData', await getData())
  },
  async actionB ({ dispatch, commit }) {
```

```
    await dispatch('actionA') // 等待 actionA 完成
    commit('gotOtherData', await getOtherData())
  }
}
```

- **严格模式：**

  创建 store 的时候传入 `strict: true`

  > 在严格模式下，无论何时发生了状态变更且不是由 mutation 函数引起的，将会抛出错误。这能保证所有的状态变更都能被调试工具跟踪到。

- 关于"严格模式"下， `v-model` 问题 - #
-

## 2. module

- 默认情况下，模块内部的 action、mutation 和 getter 是注册在**全局命名空间**的——这样使得多个模块能够对同一 mutation 或 action 作出响应
- 可以通过添加 `namespaced: true` 的方式使其成为带命名空间的模块

```
const store = new Vuex.Store({
  modules: {
    account: {
      namespaced: true,

      // 模块内容（module assets）
      state: { ... }, // 模块内的状态已经是嵌套的了，使用 `namespaced` 属性不会对其产生影响
      getters: {
        isAdmin () { ... } // -> getters['account/isAdmin']
      },
      actions: {
        login () { ... } // -> dispatch('account/login')
      },
      mutations: {
        login () { ... } // -> commit('account/login')
      },

      // 嵌套模块
```

```
      modules: {
        // 继承父模块的命名空间
        myPage: {
          state: { ... },
          getters: {
            profile () { ... } // -> getters['account/profile']
          }
        },

        // 进一步嵌套命名空间
        posts: {
          namespaced: true,

          state: { ... },
          getters: {
            popular () { ... } // ->
getters['account/posts/popular']
          }
        }
      }
    }
  }
})
```

- 如果你希望使用全局 state 和 getter， `rootState` 和 `rootGetter` 会作为第三和第四参数传入 getter，也会通过 `context` 对象的属性传入 action
- 若需要在全局命名空间内分发 action 或提交 mutation，将 `{ root: true }` 作为第三参数传给 `dispatch` 或 `commit` 即可
- 可以通过使用 `createNamespacedHelpers` 创建基于某个命名空间辅助函数。它返回一个对象，对象里有新的绑定在给定命名空间值上的组件绑定辅助函数：

```
import { createNamespacedHelpers } from 'vuex'

const { mapState, mapActions } =
createNamespacedHelpers('some/nested/module')

export default {
  computed: {
    // 在 `some/nested/module` 中查找
    ...mapState({
      a: state => state.a,
```

```
      b: state => state.b
    })
  },
  methods: {
    // 在 `some/nested/module` 中查找
    ...mapActions([
      'foo',
      'bar'
    ])
  }
}
```

- **动态注册模块**

```
// 注册模块 `myModule`
store.registerModule('myModule', {
  // ...
})
// 注册嵌套模块 `nested/myModule`
store.registerModule(['nested', 'myModule'], {
  // ...
})
```

- 也可以使用 `store.unregisterModule(moduleName)` 来动态卸载模块
  - 但：不能使用此方法卸载静态模块（即创建 store 时声明的模块）
- 在注册一个新 module 时，你很有可能想保留过去的 state，例如从一个服务端渲染的应用保留 state。你可以通过 `preserveState` 选项将其归档：`store.registerModule('a', module, { preserveState: true })`。
- **多例**

  使用一个函数来声明模块状态

```
const MyReusableModule = {
  state () {
    return {
      foo: 'bar'
    }
  },
  // mutation, action 和 getter 等等...
}
```

## 3. 使用

- 如果在模块化构建系统中，请确保在开头调用了 `Vue.use(Vuex)`
- **store.commit**

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
```

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

```
store.commit('increment')
```

## 4. 踩坑

-

## 5. 扩展

- 异步：

```
()=>import('login/index')
```