状态管理VueX

教学内容

- 第一节 Vuex介绍
- 第二节 Vuex安装与使用

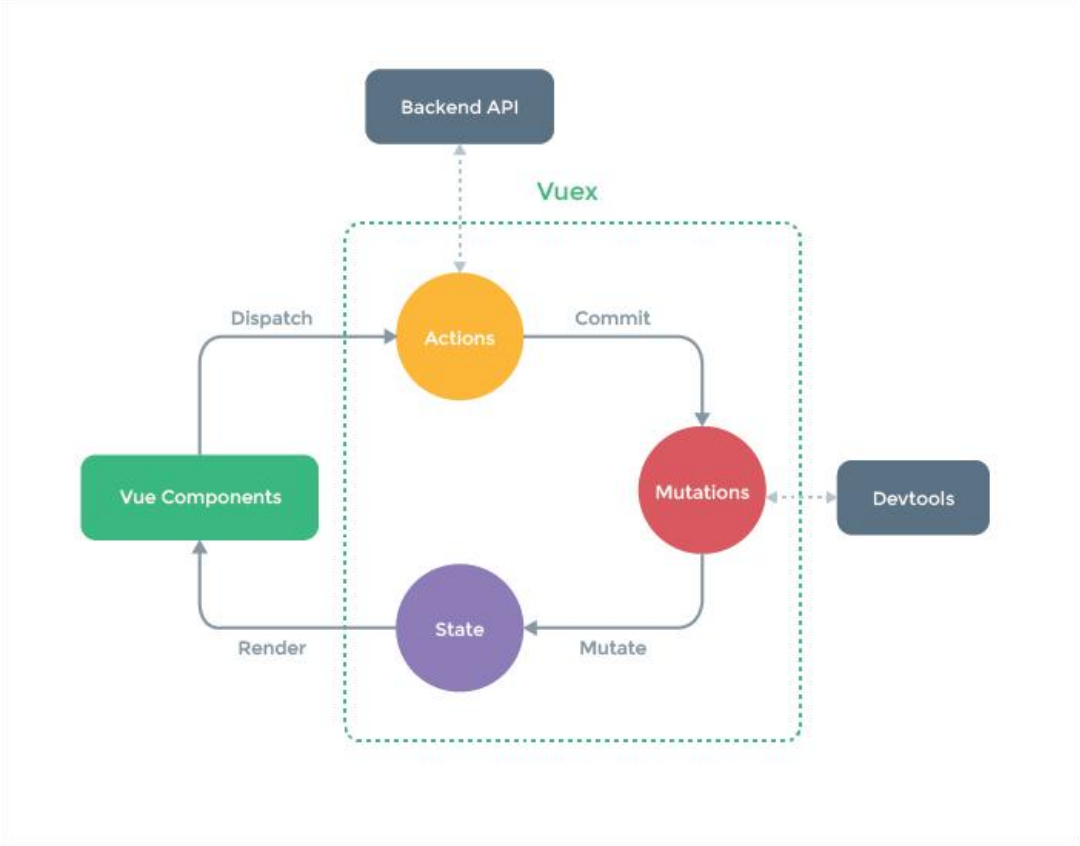
Vuex介绍

- 对于组件化开发来说，大型应用的状态往往跨越多个组件。在多层嵌套的父子组件之间传递状态已经十分麻烦，而Vue更是没有为兄弟组件提供直接共享数据的办法。
- 基于这个问题，许多框架提供了解决方案——使用全局的状态管理器，将所有分散的共享数据交由状态管理器保管，Vue也不例外。
- Vuex 是一个专为 Vue.js 应用程序开发的状态管理库，采用集中式存储管理应用的所有组件的状态。
- 简单的说，Vuex用于管理分散在Vue各个组件中的数据。
- 安装：npm install vuex@next

状态管理

- 每一个Vuex应用的核心都是一个store，与普通的全局对象不同的是，基于Vue数据与视图绑定的特点，当store中的状态发生变化时，与之绑定的视图也会被重新渲染。
- store中的状态不允许被直接修改，改变store中的状态的唯一途径就是显式地提交（commit）mutation，这可以让我们方便地跟踪每一个状态的变化。
- 在大型复杂应用中，如果无法有效地跟踪到状态的变化，将会对理解和维护代码带来极大的困扰。
- Vuex中有5个重要的概念：State、Getter、Mutation、Action、Module。

状态管理


Backend API
Vuex
Dispatch
Actions
Commit
Vue Components
Mutations
Devtools
State
Render
Mutate

State

- State用于维护所有应用层的状态，并确保应用只有唯一的数据源

```
// 创建一个新的 store 实例
const store = createStore({
  state () {
    return {
      count: 0
    }
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
```

State

- 在组件中，可以直接使用this.$store.state.count访问数据，也可以先用mapState辅助函数将其映射下来

```
// 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
    }
  })
}
```

Getter

- Getter维护由State派生的一些状态，这些状态随着State状态的变化而变化

```
const store = createStore({
  state: {
    todos: [
      { id: 1, text: '...', done: true },
      { id: 2, text: '...', done: false }
    ]
  },
  getters: {
    doneTodos: (state) => {
      return state.todos.filter(todo => todo.done)
    }
  }
})
```

Getter

- 在组件中，可以直接使用this.$store.getters.doneTodos，也可以先用mapGetters辅助函数将其映射下来，代码如下：

```
import { mapGetters } from 'vuex'

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

Mutation

- Mutation提供修改State状态的方法。

```
// 创建一个新的 store 实例
const store = createStore({
  state () {
    return {
      count: 0
    }
  },
  mutations: {
    increment (state) {
      state.count++
    }
  }
})
```

Mutation

- 在组件中，可以直接使用store.commit来提交mutation

```
methods: {
  increment() {
    this.$store.commit('increment')
    console.log(this.$store.state.count)
  }
}
```

- 也可以先用mapMutation辅助函数将其映射下来

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

    // `mapMutations` 也支持载荷：
    'incrementBy' // 将 `this.incrementBy(amount)` 映射为 `this.$store.commit('incrementBy', amount)`
  ]),
```

Action

- Action类似Mutation，不同在于:
- Action不能直接修改状态，只能通过提交mutation来修改，Action可以包含异步操作

```
const store = createStore({
  state: {
    count: 0
  },
  mutations: {
    increment (state) {
      state.count++
    }
  },
  actions: {
    increment (context) {
      context.commit('increment')
    }
  }
})
```

Action

- 在组件中，可以直接使用this.$store.dispatch('xxx')分发 action，或者使用mapActions辅助函数先将其映射下来

```
// ...
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)`
  ]),
```

Module

- 由于使用单一状态树，当项目的状态非常多时，store对象就会变得十分臃肿。因此，Vuex允许我们将store分割成模块（Module）
- 每个模块拥有独立的State、Getter、Mutation和Action，模块之中还可以嵌套模块，每一级都有着相同的结构。

Module

```
const moduleA = {
  state: () => ({ ... }),
  mutations: { ... },
  actions: { ... },
  getters: { ... }
}

const moduleB = {
  state: () => ({ ... }),
  mutations: { ... },
  actions: { ... }
}

const store = createStore({
  modules: {
    a: moduleA,
    b: moduleB
  }
})

store.state.a // -> moduleA 的状态
store.state.b // -> moduleB 的状态
```

总结

- 作为一个状态管理器，首先要有保管状态的容器——State；
- 为了满足衍生数据和数据链的需求，从而有了Getter；
- 为了可以“显式地”修改状态，所以需要Mutation；
- 为了可以“异步地”修改状态（满足AJAX等异步数据交互），所以需要Action；
- 最后，如果应用有成百上千个状态，放在一起会显得十分庞杂，所以分模块管理（Module）也是必不可少的；
- Vuex并不是Vue应用开发的必选项，在使用时，应先考虑项目的规模和特点，有选择地进行取舍，对于小型应用来说，完全没有必要引入状态管理，因为这会带来更多的开发成本；