Vuex讲解

大纲

- 介绍
- 入门使用
- 最佳实践
- 深入学习 && 使用建议

一、介绍

- 简单状态管理起步使用
- flux架构简介
- eventBus
- pub/sub模式

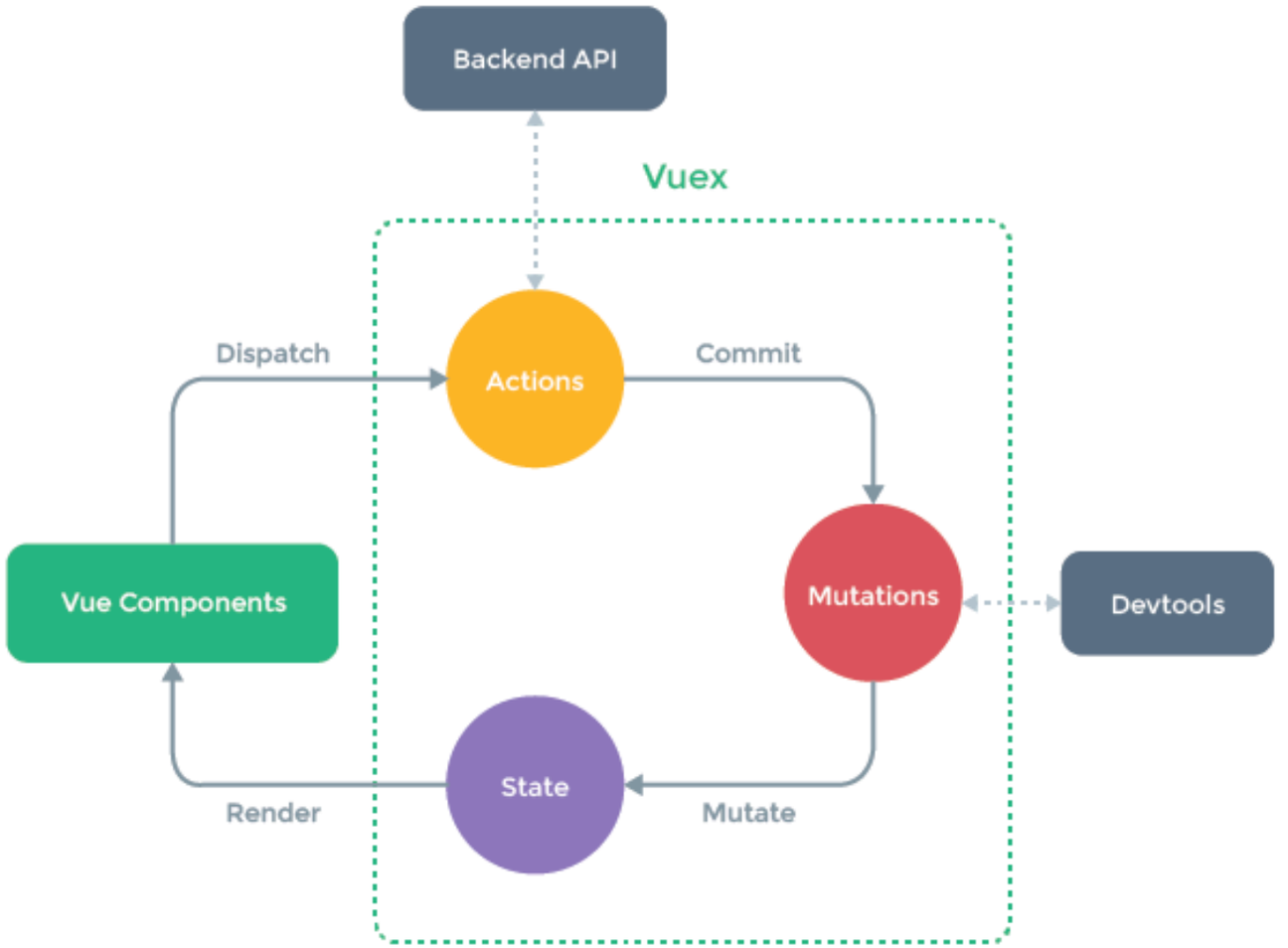
二、入门使用

- 简单的Vuex示例
- Vue组件使用Vuex
- mapState函数
- Getters对象
- mapGetters辅助函数
- Mutations
- mapMutations函数
- Aciton

为什么需要Vuex

Vuex数据流通

- Vuex 是一个专为 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。Vuex 也集成到 Vue 的官方调试工具 devtools extension，提供了诸如零配置的 time-travel 调试、状态快照导入导出等高级调试功能。
——摘自官方文档



简单的Vuex示例

- 每一个Vuex应用就是一个store，在store中包含组件中的共享状态state和改变状态的方法（暂且称作方法）mutations

- 需要注意的是只能通过mutations改变store的state的状态，不能通过store.state.count = 5;直接更改，state相当于对外的只读属性。

- 使用store.commit方法触发mutations改变state

```
JS 1-start.js > ...
1  import Vue from 'vue';
2  import Vuex form 'vuex';
3
4  Vue.use(Vuex);
5
6  const store = new Vuex.Store({
7      state: {
8          count: 0
9      },
10     mutations: {
11         increment (state) {
12             state.count++
13         }
14     }
15 })
```

```
17 store.commit('increment');
18
19 console.log(store.state.count)  // 1
```

Vue组件中使用Vuex

- Vuex状态更新 ==> Vue组件也得到更新，最简单的方法就是在Vue的computed获取state。

```
// Counter 组件
const Counter = {
    template: `<div>{{ count }}</div>`,
    computed: {
        count () {
            return store.state.count;
        }
    }
}
```

- 将store注入到根组件，可以避免在子组件里每次都引入全局store，从而直接通过this.$store来获取。

```
const Counter = {
    template: `<div>{{ count }}</div>`,
    computed: {
        count () {
            return this.$store.state.count
        }
    }
}
```

mapState函数

```
computed: {
    count () {
        return this.$store.state.count
    }
}
```

- 每次都得在computed里调用state.xx太过麻烦，这里引入mapState

- mapState函数简化这个过程

```
import { mapState } from 'vuex';

export default {
    computed: mapState ({
        count: state => state.count,
        countAlias: 'count',    // 别名 `count` 等价于 state => state.count
    })
}
```

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

Getters对象

- 需要对state对象进行做计算处理的时候，在组件里处理太过冗余繁琐。拷贝函数？抽象utils函数？

- Vuex中getters对象，可以方便我们在store中做集中的处理。

- Vue中通过store.getters对象调用。

- Getter也可以接受其他getters作为第二个参数。

```
JS 4-getters.js > ...
computed: {
    doneTodosCount () {
        return this.$store.state.todos.filter(todo => todo.done).length
    }
}
```

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

```
computed: {
    doneTodos () {
      return this.$store.getters.doneTodos
    }
}
```

```
getters: {
    doneTodos: state => {
        return state.todos.filter(todo => todo.done)
    },
    doneTodosCount: (state, getters) => {
      return getters.doneTodos.length
    }
}
```

mapGetters辅助函数

- 与mapState类似，都能达到简化代码的效果。
- mapGetters辅助函数仅仅是将store中的getters映射到局部计算属性。

JS 5-mapGetters.js

```
// 写法一:
computed: {
    // 使用对象展开运算符将 getters 混入 computed 对象中
    ...mapGetters([
      'doneTodosCount',
      'anotherGetter',
      // ...
    ])
// 写法二:
computed: mapGetters([
    'doneTodosCount',
    'anotherGetter',
    // ...
    ])
```

Mutations

- 更改Vuex的store中的状态的唯一方法就是mutations。

- 调用mutation，需要通过store.commit方法调用mutation type。

- Payload 提交载荷，建议为对象，可扩展。

- 同步的！！！

```
const store = new Vuex.Store({
  state: {
    count: 1
  },
  mutations: {
    increment (state) {
      // 变更状态
      state.count++
    }
  }
})

store.commit('increment')
```

```
// payload
mutation: {
  increment (state, payload) {
      state.totalPrice += payload.price + payload.count;
  }
}

store.commit({
  type: 'increment',
  price: 10,
  count: 8
})
```

MapMutations函数

- 同state、getters一样，也有mapMutations，帮助我们简化代码

- 使用mapMutations辅助函数将组件中的methods映射为store.commit调用。

```
import { mapMutations } from 'vuex'

export default {
  // ...
  methods: {
    ...mapMutations([
      'increment' // 映射 this.increment() 为 this.$store.commit('increment')
    ]),
    ...mapMutations({
      add: 'increment' // 映射 this.add() 为 this.$store.commit('increment')
    })
  }
}
```

actions

- Action 功能类似于 mutation，Action提交的试mutation，而不是直接操作变更状态。action可以包含异步操作。

- Action 通过 store.dispatch 方法触发

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

```
actions: {
    incrementAsync ({ commit }) {
        setTimeout(() => {
        commit('increment')
        }, 1000)
    }
}
// 以载荷形式分发
store.dispatch('incrementAsync', {
    amount: 10
})
// 以对象形式分发
store.dispatch({
    type: 'incrementAsync',
    amount: 10
})
```

总结

- state里面就是存放的我们上面所提到的状态。
- mutations就是存放如何更改状态。
- getters就是从state中派生出状态，比如将state中的某个状态进行过滤然后获取新的状态。
- actions就是mutation的加强版，它可以通过commit mutations中的方法来改变状态，最重要的是它可以进行异步操作。
- modules顾名思义，就是当用这个容器来装这些状态还是显得混乱的时候，我们就可以把容器分成几块，把状态和管理规则分类来装。这和我们创建js模块是一个目的，让代码结构更清晰。

随堂练习-小试牛刀

- 1. 我们把所有的状态从各个组件抽出来，放入了state中。
- 2. 现在给state中的list添加一个数据。
- 3. 现在，某个组件需要我们获得list中字符串长度大于10的所有数据。
- 4. 现在，某个组件需要我们需要在事件发生2秒后再向list中添加数据。
- 当代码量不断增多，这个容器的状态和Mutations，actions，getters都太多了时侯。