vuex详解

安装

```
npm install vuex —save
yarn add vuex
```

Vuex是什么？

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式

创建一个 store

```
import Vue from 'vue';
import Vuex from  'vuex';

Vue.use(Vuex);

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

})

export default store;
```

通过 store.state 来获取状态对象

通过 store.commit 方法触发状态变更

核心概念

State

Vuex 使用 单一状态树　　每个应用将仅仅包含一个 store 实例

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)

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'
import ElementUI from 'element-ui'

Vue.use(ElementUI);

new Vue({
  el: '#app',
  router,
  store,
  template: '<App/>',
  render : (h)=>h(App)
}).$mount('#app')
```

```
<template>
    <div>
        <h2>我是count</h2>
        <div>{{count}}</div>
    </div>
</template>

<script>

  export default {
    computed: {
      count(){
        return this.$store.state.count;
      }
    }
  }
</script>
```

辅助函数mapState

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

```
<template>
    <div>
        <h2>我是count</h2>
        <h4>{{count}}</h4>

        <h2>我是countAlias</h2>
        <h4>{{countAlias}}</h4>

        <h2>我是countPlusLocalState</h2>
        <h4>{{countPlusLocalState}}</h4>
    </div>
</template>

<script>
  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
      }
    })
  }
</script>
```

Getters

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：

```
import Vue from 'vue';
import Vuex from  'vuex';

Vue.use(Vuex);

const store = new Vuex.Store({
  state:{
    todos: [
      { id: 1, text: '我是你大大爷', done: true },
      { id: 2, text: '我是你二大爷', done: false }
    ]
  },

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

})

export default store;
```

```
<template>
    <div>
        <h2>我是doneTodosCount</h2>
        <h4>{{doneTodosCount}}</h4>
    </div>
</template>

<script>
  export default {
    computed: {
      doneTodosCount () {
        return this.$store.getters.doneTodos
      }
    }
  }
</script>
```

辅助函数mapGetters

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

Mutations

更改 Vuex 的 store 中的状态的唯一方法是提交 mutation。Vuex 中的 mutations 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

```
<template>
    <div>
        <h2>我是count</h2>
        <div>{{count}}</div>
        <button  type="primary" @click="add">递增按钮</button>
    </div>
</template>
<script>
    import {mapMutations} from 'vuex';
  export default {
    computed: {
      count(){
        return this.$store.state.count;
      }
    },
    methods: mapMutations({
      add:'increment'
    })
  }
</script>
```

在 mutation 中混合异步调用会导致你的程序很难调试。例如，当你能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**：

```
store.commit('increment')
// 任何由 "increment" 导致的状态变更都应该在此刻完成。
```

为了处理异步操作，让我们来看一看 **Actions**。

Actions

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。

注册一个简单action

```
/**
 * Created with wyh.
 * Date: 2017/7/21
 * Time: 下午3:32
 */

import Vue from 'vue';
import Vuex from  'vuex';

Vue.use(Vuex);

const store = new Vuex.Store({
  state:{
    count:11,
    countAlias:'count',


    todos: [
      { id: 1, text: '我是你大大爷', done: true },
      { id: 2, text: '我是你二大爷', done: false }
    ]


  },
  mutations:{
    increment(state){
      state.count++
    }
  },

  actions: {
    increment ({ commit }) {
      commit('increment')
    }
  },


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

    notDoneTodos: state => {
      return state.todos.filter(todo => !todo.done)
    }
  }


})


export default store;
```

在组件中分发Action

```
<template>
    <div>
        <h2>我是count===>Action</h2>
        <div>{{count}}</div>
        <button  type="primary" @click="add">递增按钮</button>
    </div>
</template>
<script>
    import {mapActions} from 'vuex';
  export default {
    computed: {
      count(){
        return this.$store.state.count;
      }
    },
    methods: mapActions({
      add:'increment'
    })
  }
</script>
```

Modules

由于使用单一状态树，应用的所有状态会集中到一个比较大的对象。当应用变得非常复杂时，store 对象就有可能变得相当臃肿。

为了解决以上问题，Vuex 允许我们将 store 分割成模块（module）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割：