Vuex笔记

一、Vuex是用来干什么的

官方的说法：

> 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

我的理解：

Vuex就是用来管理前端数据的，类似一个前端数据库， 它里面的`store`就是一个数据库，`state`就是里面的表， 然后从表中取数据要用到各种方法吧，这个方法取这个，那个方法取这个，这些方法叫`getter`，都放在`getters`里面，而且它只用来取数据，不能对原数据进行更改， 既然`getter`不能更改数据，那肯定有一个能改数据的东西，那就是`mutation`，我们把更改数据的方法都放在叫做`mutation`，`mutation`放在`mutations`里面， 再然后既然你要改数据，那你新的数据从哪里来呢，这时候`action`就派上用场了，它就是用来接收新的数据，并对新数据进行一些处理，然后告诉`mutation`要把某个数据改成XX样子，然后`mutation`就去把数据改成了XX样子，当然，也可以不在`action`里面直接处理数据，可以直接`mutation`，`action`都放在`actions`里面当然，我们也可以不通过`action`直接操作`mutation`，这样的话会导致`devtools`不能追踪到状态变化，其它倒没什么大问题

尤大大是这么说`actions`和`mutations`的：

> **区分 actions 和 mutations 并不是为了解决竞态问题，而是为了能用 devtools 追踪状态变化。**
>
> 事实上在 vuex 里面 actions 只是一个架构性的概念，并不是必须的，说到底只是一个函数，你在里面想干嘛都可以，只要最后触发 mutation 就行。异步竞态怎么处理那是用户自己的事情。vuex 真正限制你的只有 mutation 必须是同步的这一点。

让我们来梳理一个整个流程：

> 用`getter`从`store`中取出`state`用来各种操作，然后要更新一下`state`，那我们就通过`action`来处理下怎么更新，然后`action`把更新后的数据给`mutation`，`mutation`把数据库中的数据改为新数据

二、为什么用Vuex

官方的说法：

> 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。
>
> 对于问题一，传参的方法对于多层嵌套的组件将会非常繁琐，并且对于兄弟组件间的状态传递无能为力。对于问题二，我们经常会采用父子组件直接引用或者通过事件来变更和同步状态的多份拷贝。以上的这些模式非常脆弱，通常会导致无法维护的代码。
>
> 因此，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！
>
> 另外，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。

我的理解：

假设你有A,B,C三个页面，都各自在组件里定义了同一套数据D

假如是父子组件，那么A传B，B再传C，这之间的操作就很繁琐；假如是兄弟组件，那么它们就无法直接进行传值了；但是如果我们用了Vuex，那么只需把数据D抽取出来，然后直接更改数据D就能把A,B,C三个页面的数据更改了

三、如何使用Vuex

Vuex的核心概念

store

我们先注册一个`store`，`store`至少要注入两个选项，`state`和`mutation`

```
const store = new Vuex.Store({
  state: {
    num: 0,
    control: {
        isShow: true
    }
  },
  getters: {
      getNum: state => {
          return state.control.isShow
      }
  },
  mutations: {
    update (state, payload) {
      state.num += payload.id
    },
    init (state) {
      state.num = 0
     }
```

```
  },
  actions: {
    changeNum (context, payload) {
        // 如果id为1，那就mutation('update')
        if (payload.id !== 1) {
            context.commit('update', payload)
        } else {
            // 否则，mutation('init')
            context.commit('init')
     }
    }
  }
})
--------------------------------------------------------------------------------
// actions 也可以写成下面这种，用ES6的解构赋值来简化代码
actions: {
    changeNum ({commit}, payload) {
        if (payload.id !== 1) {
            commit('update', payload)
        } else {
            commit('init')
     }
    }
  }
// {commit}相当于 {commit} = context
context 对象包含以下属性:
{
  state,      // 等同于 `store.state`，若在模块中则为局部状态
  rootState,  // 等同于 `store.state`，只存在于模块中
  commit,     // 等同于 `store.commit`
  dispatch,   // 等同于 `store.dispatch`
  getters,    // 等同于 `store.getters`
  rootGetters // 等同于 `store.getters`，只存在于模块中
}
```

state

`state` 就是你自己定义的一个数据结构，我在上面的 `store` 中定义了一个 `num`

下面为如何在组件中使用 `state`

```
// 首先在根组件中，通过 store: store，把store实例注入根组件下面的所有子组件
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: store
})

// 在组件中使用，通过 this.$store 访问到
childNum = this.$store.state.num
```

getter

`getter` 接收参数 `state`，返回你所需要的值

```
// 获取 control 的 isShow
getters: {
      getIsShow: state => {
          return state.control.isShow
      }
},
// 在组件中使用，通过 this.$store 访问到
// 在计算属性中使用
computed: {
  getIsShow () {
    return this.$store.getters.getIsShow
  }
}
```

mutation

更改 Vuex 的 `store` 中的 `state` 的唯一的办法是提交 `mutation`

接收一个 `state` (必选)，和一个 `payload` (可选)

然后有一个需要注意的地方， `mutation` 必须是一个**同步函数**，不能是异步函数

异步函数可以在 `action` 中去执行

```
// 如果我们需要调用 mutation ，需要使用 storm.commit 方法，有两种提交方法

// 方法一：用 载荷（payload）提交， payload是一个对象
store.commit('update', { id: 1 })

// 方法二：使用包含 type 属性的对象，type 也就是我们要提交的 mutation 的名
store.commit({
    type: 'update',
    id: 1
})

// 在组件中提交，通过 this.$store.commit('xxx')提交
this.$store.commit('update', { id: 1 })
```

action

`action` 和 `mutation` 类似，有两个不同

- `action` 提交的是 `mutation`， 不直接改变 `state`
- `action` 可以执行异步操作

接收一个 `context` (必选)，和一个 `payload` (可选)

```
// 在组件中使用，通过 this.$store.dispatch('xxx')操作
// 写在组件的 methods 中，当作函数来使用
methods: {
    changeNum() {
        this.$store.dispatch('changeNum', { id: 1 })
    }
}
```

类比 Vuex 和 Vue

拿 Vuex 实例和 Vue 实例进行一下类比，方便理解记忆

Vuex实例	对应Vue实例中的	备注
state	data	
getters	computed	
mutations	event handler(事件处理)	mutation 必须是同步函数
actions	methods	通过 commit mutation 来改变 state

组件绑定的辅助函数

Vuex还定义几个辅助函数来帮助你在组件中使用的时候简化操作

具体的使用方法就不详细说了，可以去看看官方文档，使用方法类似，挺简单的

有下面几个

- mapState
- mapGetters
- mapActions
- mapMutations