# 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。  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 |

# 解决问题

|  |
| --- |
| 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 |

# Vuex 流程

|  |
| --- |
| IMG_256  Vuex把组件中共享的State抽离出来,任何组件都可以访问和操作其中的数据中心  用户在组件中的输入操作触发 action 调用；  Actions 通过分发 mutations 来修改 store 实例的状态；  Store 实例的状态变化反过来又通过 getters 被组件获知  改变state中的数据有且只有通过mutations中的方法,且mutations中的方法必须是同步的  如果要写异步的方法,需要些在actions中, 并通过commit到mutations中进行state中数据的更改 |

# 父子传递数据

|  |
| --- |
| 1、父子组件直接引用  2、事件来变更和同步状态的多份拷贝 |

# vuex有哪几种属性

|  |
| --- |
| 有五种，分别是 State、 Getter、Mutation 、Action、 Module |

# vuex的State特性是

|  |
| --- |
| 1、Vuex就是一个仓库，仓库里面放了很多对象。其中state就是数据源存放地，对应于与一般Vue对象里面的data  2、state里面存放的数据是响应式的，Vue组件从store中读取数据，若是store中的数据发生改变，依赖这个数据的组件也会发生更新  3、它通过mapState把全局的 state 和 getters 映射到当前组件的 computed 计算属性中 |

# vuex的Getter特性是

|  |
| --- |
| 1、getters 可以对State进行计算操作，它就是Store的计算属性  2、 虽然在组件内也可以做计算属性，但是getters 可以在多组件之间复用  3、 如果一个状态只在一个组件内使用，是可以不用getters |

# vuex的Mutation特性是

|  |
| --- |
| 1、getters 可以对State进行计算操作，它就是Store的计算属性  2、 虽然在组件内也可以做计算属性，但是getters 可以在多组件之间复用  3、 如果一个状态只在一个组件内使用，是可以不用getters |

# vuex的Mutation特性是

|  |
| --- |
| 1、Action 类似于 mutation，不同在于：  2、Action 提交的是 mutation，而不是直接变更状态。  3、Action 可以包含任意异步操作 |

# Vue.js中ajax请求代码应该写在组件的methods中还是vuex的actions中

|  |
| --- |
| 1、如果请求来的数据是不是要被其他组件公用，仅仅在请求的组件内使用，就不需要放入vuex 的state里。  2、如果被其他地方复用，这个很大几率上是需要的，如果需要，请将请求放入action里，方便复用，并包装成promise返回，在调用处用async await处理返回的数据。如果不要复用这个请求，那么直接写在vue文件里很方便。 |

# 不用Vuex会带来什么问题

|  |
| --- |
| 1、可维护性会下降，你要想修改数据，你得维护三个地方  2、可读性会下降，因为一个组件里的数据，你根本就看不出来是从哪来的  3、增加耦合，大量的上传派发，会让耦合性大大的增加，本来Vue用Component就是为了减少耦合，现在这么用，和组件化的初衷相背。  但兄弟组件有大量通信的，建议一定要用，不管大项目和小项目，因为这样会省很多事 |

# state

|  |
| --- |
| state: {  count: 0  },  没有在computed声明count，html中 如下访问  <h3>{{$store.state.count}}</h3>  在computed声明count，就 {{count}}  import store from '@/vuex/store'  computed:{  count(){  return this.$store.state.count; // 不需要 import store from '@/vuex/store'  或  return store.state.count  }  }  或  computed:mapState({  count:*state=>*state.count  })  或  computed:mapState(["count"])  或  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

|  |
| --- |
| state: {  todos: [  { id: 1, text: '...', done: true },  { id: 2, text: '...', done: false }  ]  },  getters: {  doneTodosCount: (*state*, *getters*) *=>* {  return getters.doneTodos.length  }  }  访问 store.getters.doneTodos  getter 在通过属性访问时是作为 Vue 的响应式系统的一部分缓存其中的  computed: {  doneTodosCount () {  return this.$store.getters.doneTodosCount  }  }  computed: {  // 使用对象展开运算符将 getter 混入 computed 对象中  ...mapGetters([  'doneTodosCount',  'anotherGetter',  // ...  ])  ...mapGetters(["count"])  ...mapState(["count"]),  } |

# mutation

|  |
| --- |
| 更改 Vuex 的 store 中的状态的唯一方法是提交 mutation  必须是同步函数  *const* store = new Vuex.Store({  state: {  count: 1  },  mutations: {  increment (*state*) {  // 变更状态  state.count++  }  }  })  store.commit('increment')  mutations: {  increment (state, payload) {  state.count += payload.amount  }  }  store.commit('increment', {  amount: 10  })  在组件中提交 Mutation  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)`  ]),  ...mapMutations({  add: 'increment' // 将 `this.add()` 映射为 `this.$store.commit('increment')`  })  } |

# action

|  |
| --- |
| 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。 |