1.vuex状态管理

1.1 功能简介

1、当写vue项目时，当涉及到频繁的组件之间的数据通讯、一个组件需要多次派发事件时，我们的代码就会变得复杂、冗余、难以维护

需求场景：

1.在商城首页加入购物车按钮点击添加商品数量

2.在商品分类页加减按钮点击修改商品数量

3.在商品详情页可以修改商品数量

4.在购物车页面加减按钮可以修改商品数量

5.在订单结算页面展示商品数量信息

6.在我的订单页面展示商品数量信息

7.在订单详情页展示商品数量信息

2、Vuex是一个专为Vue.js应用程序开发的状态管理模式。我们可以把一些共享的数据保存至vuex中，方便各个组件修改或获取公共状态。

使用场景：

- 1. 可复用性强的数据和方法
- 2. 需要跨多级组件传递的数据
- 3. 需要发起http请求的数据

优点：

- 1. 统一数据管理
- 2. 代码分层
- 3. 减少代码量
- 4. 减少http请求次数
- 5. 便于后期维护

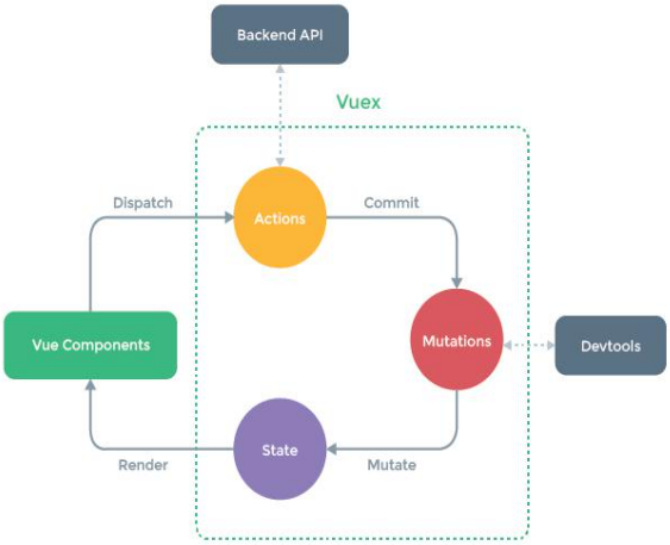
3、vue的单向数据流的简洁性在以下情况下很容易被破坏

- 多个视图依赖于同一状态
- 来自不同视图的行为需要改变同一状态

  解决思想：

  我们可以把组件的共享状态提取出来，作为全局单例模式管理。这样，组件树构成了一个巨大的视图，不论在树的哪个位置，都能获取状态或触发行为。

4、vuex的示意图



通过定义和隔离状态管理中的各种概念并通过强制规则维持视图和状态间的独立性，使我们的代码变得更结构化且易维护

1.2 下载以及安装

1、vuex的安装方式

- 步骤1：在package.json文件中的dependencies属性中增加vue的依赖包声明

```
"dependencies": {
        "vuex": "^4.1.0",
        "vuex-persistedstate": "^4.1.0"
    }
```

- 步骤2：使用cmd指令进入package.json所在目录执行安装指令,安装后在node_modules目录下生成vuex、vuex-persistedstate包

```
npm install
```

- 步骤3：在独立的store配置文件中引入vuex和vuex-persistedstate组件（如：store.js）

```
import { createStore } from 'vuex'
import createPersistedState from "vuex-persistedstate"
```

- 步骤4：创建vuex全局实例对象，并设定持久化插件 写在store.js中

```
export default createStore({
  //1、存储所有全局数据
  state: {
  },
  //2、需要通过计算获取state里的内容获取数据
  //只能读取不可修改
  getters: {
  },
  //3、定义对state各种操作，不能执行异步操作，简单直接赋值操作
  mutations: {
  },
  //4、定义对state的各种操作，action无法直接修改state，需要在mutation里更新，
  // mutation不支持异步，所以需要在action里写api的url
  actions: {
```

```
  },
  //5、加载多个state文件，防止一个state文件过长
  modules: {
  },
  //6、插件
  plugins:[]
})
```

- 步骤5：在main.js中引入vuex实例，并以插件的方式添加到组件实例对象中

```
import store from './store'

const app = createApp(App);
app.use(store)
```

1.3 state核心组件

1、vuex使用单一状态树state对象来作为“唯一数据源”，专门用于存放数据。

- 在组件中访问state下的数据，通过useStore().state来访问
- 不能通过useStore().state进行赋值，会造成显示与存储不统一
- 一般建议当需要访问state的属性时，通过计算属性访问，避免对数据赋值

2、getters属性：当需要访问的数据需要通过对state中的数据需要进行运算或逻辑处理才能返回结果时，可以使用getters。

- getters可以简单的把它理解为Store中的计算属性，当store里的值发生改变时，getters会返回重新计算的值。
- getters下的方法第一个参数为state，意味着在getters方法中可以访问state。

MyStore.vue

```
<template>
    <div class="div1">
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'

    const store = useStore()
    //计算属性
    const count = computed(() => {
        return store.state.count
    })

</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
```

```
</style>
```

store/index.js

```
import {createStore} from 'vuex'

export default createStore({
    //1、存储所有全局数据
    state: {
        count: 0
    },
    //2、需要通过计算获取state里的内容获取数据
    //只能读取不可修改
    getters: {
        //store里的值发生改变时，getter会返回重新计算的值
        getCount(state){
            return state.count
        }
    },
    //3、定义对state各种操作，不能执行异步操作，简单直接赋值操作
    mutations: {},
    //4、定义对state的各种操作，action无法直接修改state，需要在mutation里更新，
    // mutation不支持异步，所以需要在action里写api的url
    actions: {},
    //5、加载多个state文件，防止一个state文件过长
    modules: {},
    //6、插件
    plugins: []
})
```

1.4 Mutations组件

1、mutations组件：通过mutation更新state中数据，不可以直接操作store中的数据，只有正确提交mutations，才能保证state中的数据在每个组件中渲染与存储一致。

2、mutations组件的提交：

```
store.commit("login",playLoad)
```

- login为事件类型，与mutations中声明保持一致
- playLoad：为对象或基本类型的数据，提交荷载

注意事项：

Vuex的store中的状态是响应式的，当我们变更状态时，监视状态的Vue组件也会自动更新。这也意味着Vuex中的mutation也需要与使用Vue一样遵守一些注意事项：

- 在store中初始化好所有所需属性
- 当需要在对象上添加新属性时，应该使用Vue.set(obj,'newProp',123),或者以新对象替换老对象

  state.obj={...state.obj,newProp:123}

store/index.js

```
import {createStore} from 'vuex'
```

```
export default createStore({
    //1、存储所有全局数据
    state: {
        count: 0
    },
    //2、需要通过计算获取state里的内容获取数据
    //只能读取不可修改
    getters: {
        //store里的值发生改变时，getter会返回重新计算的值
        getCount(state) {
            return state.count
        }
    },
    //3、定义对state各种操作，不能执行异步操作，简单直接赋值操作
    mutations: {
        //state存储数据对象,num为相加或者相减的数据
        addCount(state, num) {
            state.count = state.count + num;
        },
        subCount(state, num) {
            //count为0不能再做减法
            if (state.count > 0) {
                state.count = state.count - num;
            } else {
                state.count = 0;
            }
        }
    },
    //4、定义对state的各种操作，action无法直接修改state，需要在mutation里更新，
    // mutation不支持异步，所以需要在action里写api的url
    actions: {},
    //5、加载多个state文件，防止一个state文件过长
    modules: {},
    //6、插件
    plugins: []
})
```

MyStore.vue

```
<template>
    <div class="div1">
        <input type="button" value="-" @click="sub"/>
        <input type="button" value="+" @click="add"/>
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
        <br />
        <router-link to="/myStore1">myStore1</router-link>
        <router-view></router-view>
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'
```

```
    const store = useStore()
    //计算属性,防止赋值
    const count = computed(() => {
        return store.state.count
    })

    const add = () => {
        //store.commit() 提交修改，
        //第一个参数为store.js的mutations定义的事件类型，第二个参数为数量
        store.commit("addCount", 1);
    }
    const sub = () => {
        store.commit("subCount", 1);
    }
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

MyStore1.vue

```
<template>
    <div class="div1">
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'

    const store = useStore()
    //计算属性
    const count = computed(() => {
        return store.state.count
    })
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

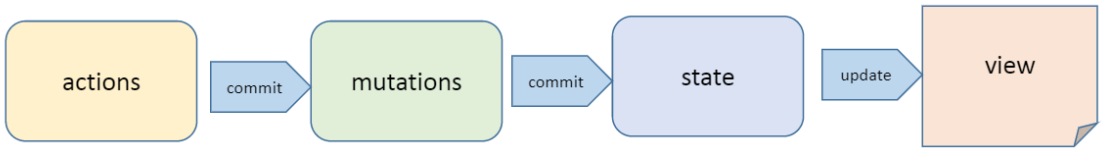
路由配置router/index.js

```
    {
        path: '/myStore',
        name: 'myStore',
        //可以通过import导入已经存在的组件文件
        component: () => import('@/views/MyStore'),
        children: [
            {path: '/myStore1', name: 'MyStore1', component: () =>
import('@/views/MyStore1')}
        ]
    },
```

1.5 Actions组件

1、Actions组件：Mutations组件的异步数据更新

- actions组件与Mutations组件的作用都是对state数据属性赋值，并同步数据显示渲染。
- Mutations组件仅适用于同步数据赋值，而actions适用于异步数据赋值
- Actions组件最终通过提交Mutations中的事件类型对state属性赋值



2、Actions组件的声明

- context为当前vuex实例对象，与this.$store一致，可以调用commit方法
- playload为荷载
- 通过调用dispath触发action组件事件类型

```
store.dispatch('addCountAsync',playLoad)
```

store/index.js

```
import {createStore} from 'vuex'

export default createStore({
    //1、存储所有全局数据
    state: {
        count: 0
    },
    //2、需要通过计算获取state里的内容获取数据
    //只能读取不可修改
    getters: {
        //store里的值发生改变时，getter会返回重新计算的值
        getCount(state) {
            return state.count
        }
    },
    //3、定义对state各种操作，不能执行异步操作，简单直接赋值操作
    mutations: {
```

```
        //state存储数据对象,num为相加或者相减的数据
        addCount(state, num) {
            state.count = state.count + num;
        },
        subCount(state, num) {
            //count为0不能再做减法
            if (state.count > 0) {
                state.count = state.count - num;
            } else {
                state.count = 0;
            }
        }
    },
    //4、定义对state的各种操作，action无法直接修改state，需要在mutation里更新，
    // mutation不支持异步，所以需要在action里写api的url
    actions: {
        myAddCount(context, num) {
            //context为当前vuex实例对象，与store一致，可以调用commit方法
            //第一个参数addCount为mutations定义的方法
            //context.commit("addCount", num);
            //模拟异步操作
            setTimeout(() => {
                context.commit("addCount", num);
            }, 2000);
        },
        mySubCount(context, num) {
            //context为当前vuex实例对象，与this.store一致，可以调用commit方法
            //第一个参数subCount为mutations定义的方法
            context.commit("subCount", num);
        }
    },
    //5、加载多个state文件，防止一个state文件过长
    modules: {},
    //6、插件
    plugins: []
})
```

MyStore.vue

```
<template>
    <div class="div1">
        <input type="button" value="-" @click="sub"/>
        <input type="button" value="+" @click="add"/>
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
        <br />
        <router-link to="/myStore1">myStore1</router-link>
        <router-view></router-view>
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'
```

```
    const store = useStore()
    //计算属性
    const count = computed(() => {
        return store.state.count
    })

    const add = () => {
        //store.commit() 提交修改，
        //第一个参数为store/index.js的mutations定义的事件类型，第二个参数为数量
        //store.commit("addCount", 1);
        //通过store.dispatch 触发 actions定义的事件类型
        store.dispatch("myAddCount",1);
    }
    const sub = () => {
        //第一个参数为store/index.js的mutations定义的事件类型，第二个参数为数量
        //store.commit("subCount", 1);
        //通过store.dispatch 触发 actions定义的事件类型
        store.dispatch("mySubCount",1);
    }
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

MyStore1.vue

```
<template>
    <div class="div1">
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'

    const store = useStore()
    //计算属性
    const count = computed(() => {
        return store.state.count
    })
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

1.6 模块化管理

由于vuex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

- 解决方案：将**store分模块管理**，每个模块都有自己内部的state、mutations、actions、getters

  store/user/index.js

```
const user = {
    state: {
        id: 1,
    },
    getters: {
        getId(state) {
            return state.id
        }
    },
    mutations: {
        saveId(state, id) {
            state.id = id
        }
    },
    actions: {}
}
export default user
```

  store/index.js

```
import {createStore} from 'vuex'
import user from '@/store/user'

export default createStore({
    ........
    //5、加载多个state文件，防止一个state文件过长
    modules: {user},
    ........
}
```

  MyStore.vue

```
<template>
    <div class="div1">
        .......
        <br/>
        <input type="button" value="SaveId" @click="addSaveId"/>
        <div>userid: {{store.getters.getId}}</div>
        <br/>
        .......
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'

    const store = useStore()
```

```
    const addSaveId = () => {
        store.commit('saveId',10)
    }
   .......
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

1.7 持久存储

问题：在登录成功后，把用户信息保存在vuex中作为全局共享数据，在页面刷新之后数据会丢失

- 分析：由于vuex中的数据保存在运行内存中，当页面刷新时，页面会重载vue实例，vuex中的数据会被重新赋值
- 解决思路：把vuex中的state数据保存一份到客户端存储中
  - localStorage:永久本地存储。本案例采用vuex-persistedstate中间件处理，简化api调用
  - sessionStorage:存储至页面关闭
  - cookie:手动设置有效时间，但存储量过小

store/index.js

```
import {createStore} from 'vuex'
import user from '@/store/user'
import createPersistedState from 'vuex-persistedstate'

export default createStore({
    //1、存储所有全局数据
    state: {
        count: 0
    },
    .......
    //5、加载多个state文件，防止一个state文件过长
    modules: {user},
    //6、插件
    //paths:['state属性名','模块属性名.属性名']
    plugins: [
        createPersistedstate({
            paths:["count","user.id"]
        })
    ]
})
```

1.8 完整的案例

MyStore.vue

```vue
<template>
    <div class="div1">
        <input type="button" value="-" @click="sub"/>
        <input type="button" value="+" @click="add"/>
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
        <br/>
        <input type="button" value="+UserId" @click="addUserId"/>
        <div>userid：{{store.getters.getId}}</div>
        <br/>
        <router-link to="/myStore1">myStore1</router-link>
        <router-view></router-view>
    </div>
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'

    const store = useStore()
    //计算属性
    const count = computed(() => {
        return store.state.count
    })
    const addUserId = () => {
        store.commit('saveId',10)
    }
    const add = () => {
        //store.commit() 提交修改，
        //第一个参数为store/index.js的mutations定义的事件类型，第二个参数为数量
        //store.commit("addCount", 1);
        //通过store.dispatch 触发 actions定义的事件类型
        store.dispatch("myAddCount", 1);
    }
    const sub = () => {
        //第一个参数为store/index.js的mutations定义的事件类型，第二个参数为数量
        //store.commit("subCount", 1);
        //通过store.dispatch 触发 actions定义的事件类型
        store.dispatch("mySubCount", 1);
    }
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

MyStore1.vue

```vue
<template>
    <div class="div1">
        <div>从state上获取的count：{{count}}</div>
        <div>从getters上获取的getCount：{{store.getters.getCount}}</div>
    </div>
```

```
</template>

<script setup>
    import {computed} from 'vue'
    import {useStore} from 'vuex'

    const store = useStore()
    //计算属性
    const count = computed(() => {
        return store.state.count
    })
</script>

<style scoped>
    .div1 {
        border: 1px #0000FF solid;
    }
</style>
```

路由配置

```
    {
        path: '/myStore',
        name: 'myStore',
        //可以通过import导入已经存在的组件文件
        component: () => import('@/views/MyStore'),
        children: [
            {path: '/myStore1', name: 'MyStore1', component: () =>
import('@/views/MyStore1')}
        ]
    },
```

store/index.js

```
import {createStore} from 'vuex'
import user from '@/store/user'
import createPersistedState from 'vuex-persistedstate'

export default createStore({
    //1、存储所有全局数据
    state: {
        count: 0
    },
    //2、需要通过计算获取state里的内容获取数据
    //只能读取不可修改
    getters: {
        //store里的值发生改变时，getter会返回重新计算的值
        getCount(state) {
            return state.count
        }
    },
    //3、定义对state各种操作，不能执行异步操作，简单直接赋值操作
    mutations: {
        //state存储数据对象,num为相加或者相减的数据
```

```
        addCount(state, num) {
            state.count = state.count + num;
        },
        subCount(state, num) {
            //count为0不能再做减法
            if (state.count > 0) {
                state.count = state.count - num;
            } else {
                state.count = 0;
            }
        }
    },
    //4、定义对state的各种操作，action无法直接修改state，需要在mutation里更新，
    // mutation不支持异步，所以需要在action里写api的url
    actions: {
        myAddCount(context, num) {
            //context为当前vuex实例对象，与store一致，可以调用commit方法
            //第一个参数addCount为mutations定义的方法
            //context.commit("addCount", num);
            //模拟异步操作
            setTimeout(() => {
                context.commit("addCount", num);
            }, 2000);
        },
        mySubCount(context, num) {
            //context为当前vuex实例对象，与this.store一致，可以调用commit方法
            //第一个参数subCount为mutations定义的方法
            context.commit("subCount", num);
        }
    },
    //5、加载多个state文件，防止一个state文件过长
    modules: {user},
     //6、插件
    //paths:['state属性名','模块属性名.属性名']
    plugins: [
        createPersistedstate({
            paths:["count","user.id"]
        })
    ]
})
```

store/user/index.js

```
const user = {
    state: {
        id: 1,
        username: 'tom'
    },
    getters: {
        getId(state) {
            console.log(state)
            return state.id
        },
        getUsername(state) {
            return state.username
```

```
        }
    },
    mutations: {
        saveId(state, id) {
            state.id = id
        },
        saveUsername(state, username) {
            state.username = username
        }
    },
    actions: {}
}
export default user
```