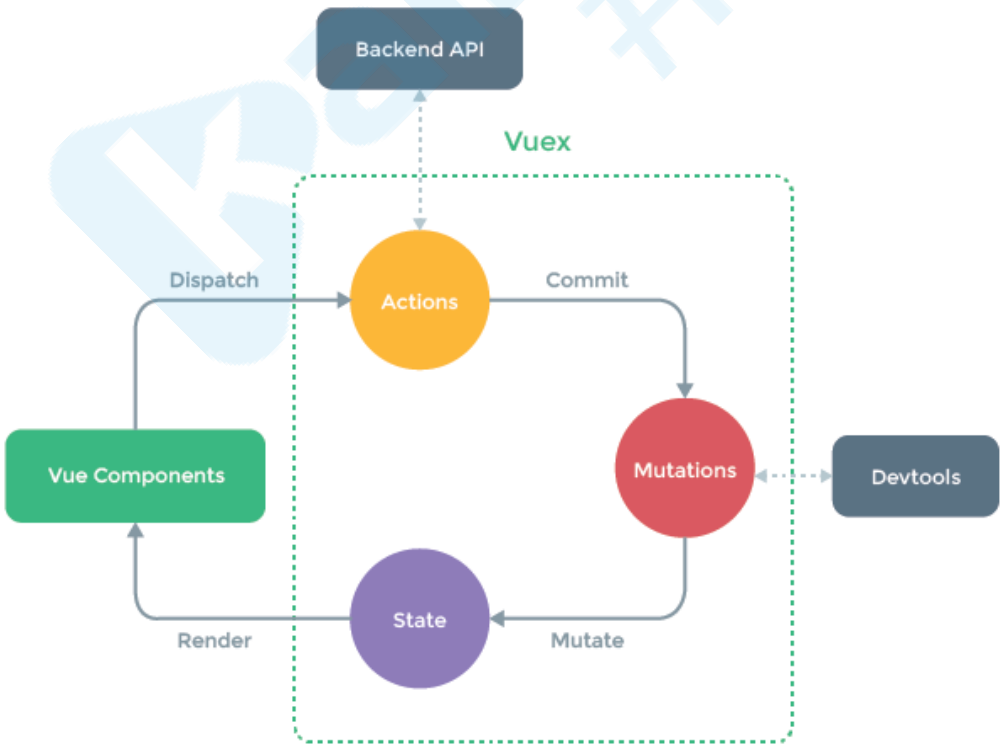
Vue预习课：统一状态管理 - Vuex

Vuex

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**可预测的方式发生变化。**



安装

```
vue add vuex
```

起始

State

将应用全局状态定义在state中

```
state: {
    isLogin: false
}
```

Mutation

修改state只能通过mutation

```
mutations: {
    login(state) {
        state.isLogin = true
    },
    logout(state) {
        state.isLogin = false
    }
},
```

获取和修改状态

使用store.state获取状态

```
<button @click="login" v-if="!$store.state.isLogin">登录</button>
<button @click="logout" v-else>登出</button>
```

修改状态只能通过store.dispatch(mutation)

```
this.$store.commit('login')
this.$store.commit('logout')
```

Action

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。

```
login({commit}, username) {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            if (username === 'admin') {
                commit('login')
                resolve()
            } else {
                reject()
            }
        }, 1000);
    })
}
```

派发动作

```
this.$store.dispatch('login', 'admin').then(() => {
    this.$router.push(this.$route.query.redirect)
}).catch(() => {
    alert('用户名或密码错误')
})
```

最佳实践

模块化

使用modules定义多个子模块利于组件复杂状态

```
import user from './user'

export default new Vuex.Store({
  modules: {
    user,
  }
})
```

移动先前登录状态相关代码到user.js

```
export default {
   namespaced: true, // 避免命名冲突
   // ...
}
```

访问方式响应变化

```
// Login.vue
<button @click="login" v-if="!$store.state.user.isLogin">登录</button>

this.$store.dispatch('user/login', 'admin').then(() => {
    const redirect = this.$route.query.redirect || '/'
    this.$router.push(redirect)
}).catch(() => {
    alert('用户名或密码错误')
})
```

```
// router/index.js
store.state.user.isLogin
```

mapState()/mapMutation()/mapAction()

通过这些映射方法可以让大家少敲几个字，避免对$store直接访问。

state相关修改，Login.vue

```
import { mapState } from 'vuex'

computed: {
    ...mapState('user', ['isLogin'])
}
```

```
<button @click="login" v-if="!isLogin">登录</button>
```

action相关修改

```
import { mapActions } from 'vuex'

methods: {
    login() {
        this['user/login']('admin').then(...)
    },
    ...mapActions(['user/login', 'user/logout'])
},
```

Getter

可以使用getters从store的state中派生出一些状态

```
export default {
  namespaced: true,
  state: {
    isLogin: false,
    username: '' // 用户名
  },
  mutations: {
```

```
    setUsername(state, username) {
      state.username = username
    }
  },
  getters: { // 派生出欢迎信息
    welcome: state => {
      return state.username + ',欢迎回来';
    }
  },
  actions: {
    login({ commit }, username) {
      return new Promise((resolve, reject) => {
        setTimeout(() => {
          if (username === 'admin') {
            // 登录成功，设置用户名
            commit('setUsername', username)
            resolve()
          } else {
            reject()
          }
        }, 1000);
      })
    }
  },
}
```

严格模式

严格模式下，无论何时发生了状态变更且不是由 mutation 函数引起的，将会抛出错误。这能保证所有的状态变更都能被调试工具跟踪到。开启严格模式 `strict: true`

```
const store = new Vuex.Store({
  // ...
  strict: true
})
```

插件

Vuex 的 store 接受 `plugins` 选项，这个选项暴露出每次 mutation 的钩子。Vuex 插件就是一个函数，它接收 store 作为唯一参数：

```
const myPlugin = store => {
  // 当 store 初始化后调用
}
```

注册插件：

```
const store = new Vuex.Store({
  // ...
  plugins: [myPlugin]
})
```

范例：实现登录状态持久化，store/plugins/persist.js

```
export default store => {
  // 初始化时从localStorage获取数据
  if(localStorage) {
    const user = JSON.parse(localStorage.getItem('user'))
    if (user) {
      store.commit('user/login')
      store.commit('user/setUsername', user.username)
    }
  }
  // 用户状态发生变化时缓存之
  store.subscribe((mutation, state) => {
    if (mutation.type.startsWith('user/')) {
      localStorage.setItem('user', JSON.stringify(state.user))
    } else if (mutation.type === 'user/logout') {
      localStorage.removeItem('user')
    }
  })
}
```