# Vue 核心技术与实战

## day07

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

# vuex概述

目标：明确 vuex 是什么，应用场景，优势


**1. 是什么：**

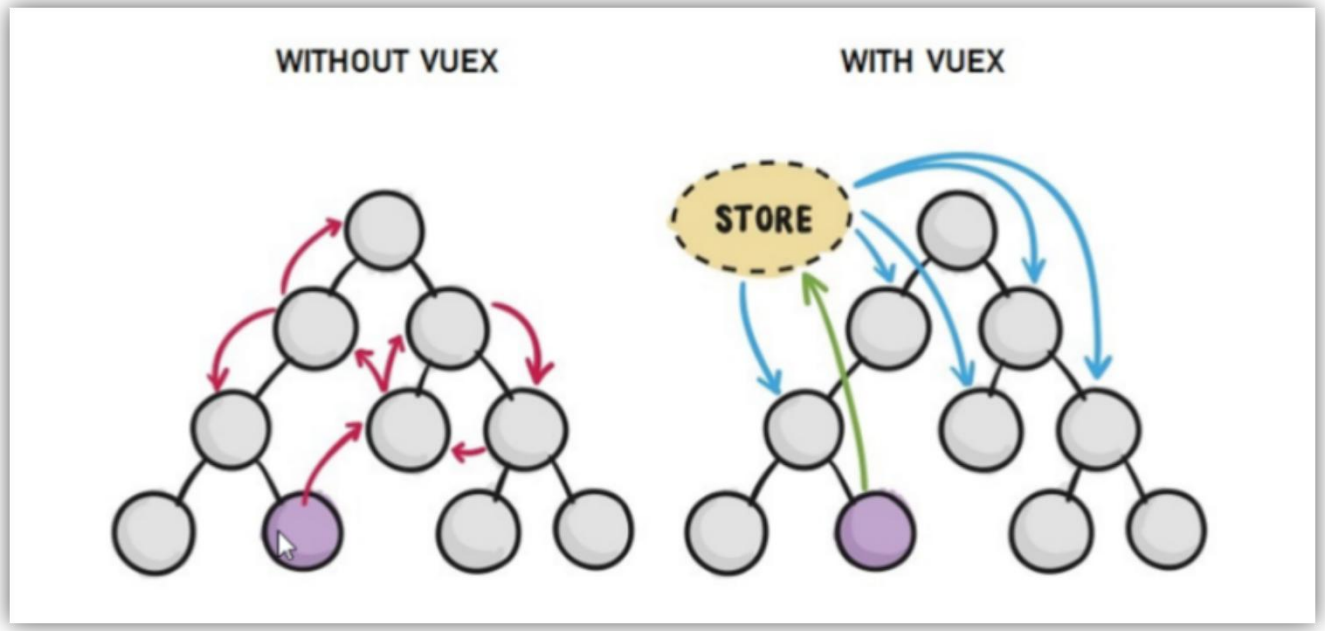
vuex 是一个 vue 的 状态管理工具，状态就是数据。

大白话：vuex 是一个插件，可以帮我们管理 vue 通用的数据 (多组件共享的数据)　例如：购物车数据　个人信息数据

**2. 场景：**

① 某个状态 在 很多个组件 来使用 (个人信息)

② 多个组件 共同维护 一份数据 (购物车)




**3. 优势：**

① 共同维护一份数据，数据集中化管理

② 响应式变化

③ 操作简洁 (vuex提供了一些辅助函数)

# 构建 vuex [多组件数据共享] 环境

目标：基于脚手架创建项目，构建 vuex 多组件数据共享环境


效果是三个组件, 共享一份数据:

- 任意一个组件都可以修改数据
- 三个组件的数据是同步的

# 创建一个空仓库

目标：安装 vuex 插件，初始化一个空仓库


1. yarn add vuex@3

3. Vue.use(Vuex)
   创建仓库 new Vuex.Store()

安装 vuex → 新建vuex模块文件 → 创建仓库 → main.js 导入挂载

2. 新建 store/index.js 专门存放 vuex


4. 在 main.js 中导入挂载到 Vue 实例上

检验：this.$store

```
App.vue?11c4:19
Store {_committing: false, _actions: {…}, _actionSubscribers: Array(1), _mutations: {…}, _wrappedGetters: {…}, …}
```

## 核心概念 - state 状态


目标：明确如何给仓库 提供 数据，如何 使用 仓库的数据

1. 提供数据：

State 提供唯一的公共数据源，所有共享的数据都要统一放到 Store 中的 State 中存储。

在 state 对象中可以添加我们要共享的数据。

```
// 创建仓库
const store = new Vuex.Store({
  // state 状态，即数据，类似于vue组件中的data
  // 区别：
  // 1. data 是组件自己的数据
  // 2. state 是所有组件共享的数据
  state: {
    count: 101
  }
})
```

## 核心概念 - state 状态


目标：明确如何给仓库 提供 数据，如何 使用 仓库的数据

2. 使用数据：

① 通过 store 直接访问

② 通过辅助函数

```
获取 store:
(1) this.$store
(2) import 导入 store


模板中：      {{ $store.state.xxx }}
组件逻辑中：   this.$store.state.xxx
JS模块中：    store.state.xxx
```

## 核心概念 - state 状态

目标：明确如何给仓库 提供 数据，如何 使用 仓库的数据

```
{{ count }}

// 把state中数据，定义在组件内的计算属性中
computed: {
  count () {
    return this.$store.state.count
  }
},
```

2. 使用数据：

① 通过 store 直接访问

② 通过辅助函数 (简化)

mapState是辅助函数，帮助我们把 store中的数据 自动 映射到 组件的计算属性中

```
import { mapState } from 'vuex'
```

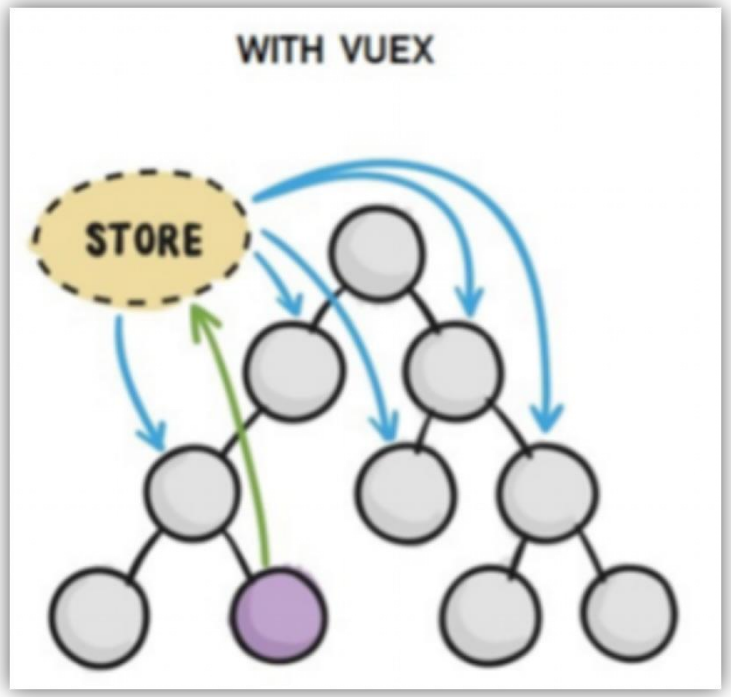

```
computed: {
  ...mapState(['count'])
 }
```

```
mapState(['count'])
```

## 核心概念 - mutations

目标：明确 vuex 同样遵循单向数据流，组件中不能直接修改仓库的数据

通过 strict: true 可以开启严格模式


```
// 创建仓库
const store = new Vuex.Store({
  // 开启严格模式
  strict: true,
  // 通过 state 可以提供数据
  state: {
    title: '仓库大标题',
    count: 100
  }
})
```

this.$store.state.count++ (错误写法)

## 核心概念 - mutations

目标：掌握 mutations 的操作流程，来修改 state 数据。（state数据的修改只能通过 mutations ）

1. 定义 mutations 对象，对象中存放修改 state 的方法

```
const store  = new Vuex.Store({
  state: {
    count: 0
  },
  // 定义mutations
  mutations: {
    // 第一个参数是当前store的state属性
    addCount (state) {
      state.count += 1
    }
  }
})
```

2. 组件中提交调用 mutations

```
this.$store.commit('addCount')
```

# 核心概念 - mutations

目标：掌握 mutations 传参语法

提交 mutation 是可以传递参数的 `this.$store.commit( 'xxx', 参数 )`


1. 提供 mutation 函数 (带参数 - 提交载荷 payload )

```
mutations: {
  ...
  addCount (state, n) {
    state.count += n
  }
},
```

2. 页面中提交调用 mutation

```
this.$store.commit('addCount', 10)
```

Tips: 提交参数只能一个，如果有多个参数，包装成一个对象传递

```
this.$store.commit('addCount', {
  count: 10,
  ...
})
```

## 核心概念 - mutations - 练习

目标：减法功能，巩固 mutations 传参语法


封装mutation 函数 → 页面中commit调用

# 核心概念 - mutations - 练习

目标：实时输入，实时更新，巩固 mutations 传参语法


输入框<br>内容渲染 → 监听输入<br>获取内容 → 封装mutation<br>处理函数 → 调用传参

:value　　@input　　mutation传参　　commit调用

# 辅助函数 - mapMutations

目标：掌握辅助函数 mapMutations，映射方法

mapMutations 和 mapState很像，它是把位于mutations中的方法提取了出来，映射到组件methods中

```
mutations: {
  subCount (state, n) {
    state.count -= n
  },
}
```

```
import { mapMutations } from 'vuex'

methods: {
  ...mapMutations(['subCount'])
}
```

```
methods: {
  subCount (n) {
    this.$store.commit('subCount', n)
  },
}
```

```
this.subCount(10) 调用
```

# 核心概念 - actions

目标：明确 actions 的基本语法，处理异步操作。

需求: 一秒钟之后, 修改 state 的 count 成 666。

说明：mutations 必须是同步的 (便于监测数据变化，记录调试)


```
mutations: {
  changeCount (state, newCount) {
    state.count = newCount
  }
}
```

1. 提供action 方法

```
actions: {
  setAsyncCount (context, num) {
    // 一秒后，给一个数，去修改 num
    setTimeout(() => {
      context.commit('changeCount', num)
    }, 1000)
  }
}
```

2. 页面中 dispatch 调用

```
this.$store.dispatch('setAsyncCount', 200)
```

# 辅助函数 - mapActions

目标：掌握辅助函数 mapActions，映射方法

mapActions 是把位于 actions中的方法提取了出来，映射到组件methods中

```
actions: {
  changeCountAction (context, num) {
    setTimeout(() => {
      context.commit('changeCount', num)
    }, 1000)
  }
}
```

```
import { mapActions } from 'vuex'

methods: {
  ...mapActions(['changeCountAction'])
}
```

```
methods: {
  changeCountAction (n) {
    this.$store.dispatch('changeCountAction', n)
  },
}
```

```
this.changeCountAction(666) 调用
```

# 核心概念 - getters

目标：掌握核心概念 getters 的基本语法 (类似于计算属性)

说明：除了state之外，有时我们还需要从state中派生出一些状态，这些状态是依赖state的，此时会用到getters

例如：state中定义了list，为 1-10 的数组，组件中，需要显示所有大于5的数据

```
state: {
    list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
}
```

1. 定义 getters

```
getters: {
  // 注意:
  // (1) getters函数的第一个参数是 state
  // (2) getters函数必须要有返回值
  filterList (state) {
    return state.list.filter(item => item > 5)
  }
}
```

2. 访问getters

① 通过 store 访问 getters

```
{{ $store.getters.filterList }}
```

② 通过辅助函数 mapGetters 映射

```
computed: {
  ...mapGetters(['filterList'])
},
```

```
{{ filterList }}
```

# 核心概念 - 模块 module (进阶语法)

目标：掌握核心概念 module 模块的创建

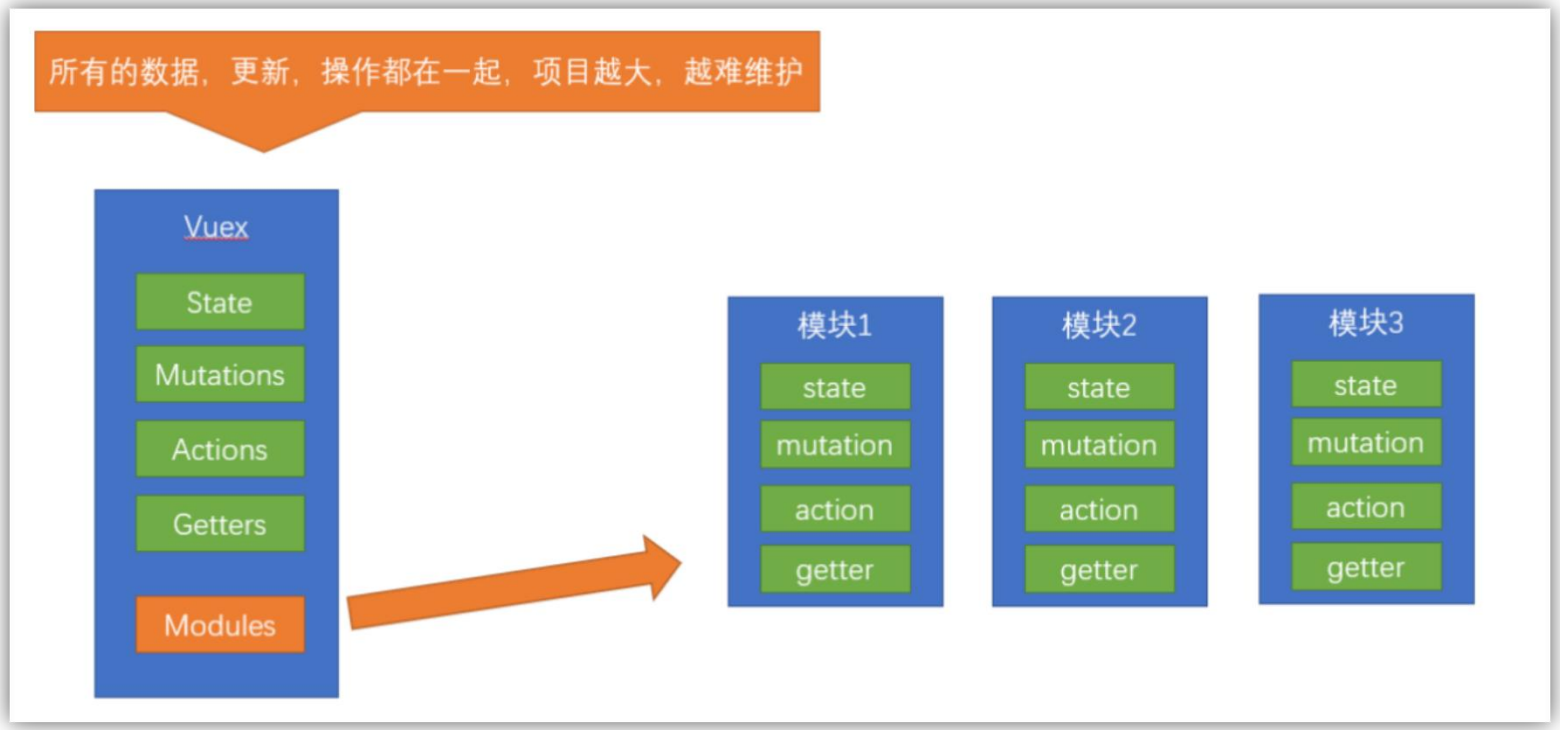
由于 vuex 使用单一状态树，应用的所有状态会集中到一个比较大的对象。当应用变得非常复杂时，store 对象就有可能变得相当臃肿。(当项目变得越来越大的时候，Vuex会变得越来越难以维护)

```
state: {
  userInfo: {
    name: 'zs',
    age: 18
  },
  theme: 'dark',
  desc: '测试项目'
},
```

用户模块 user

设置模块 setting

单一状态树



# 核心概念 - 模块 module (进阶语法)

目标：掌握核心概念 module 模块的创建

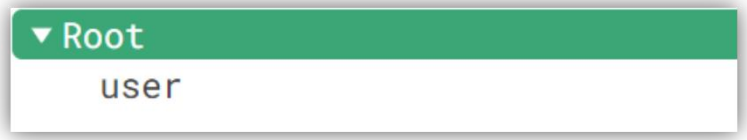
模块拆分：

user模块: store/modules/user.js


```
const state = {
  userInfo: {
    name: 'zs',
    age: 18
  }
}
const mutations = {}
const actions = {}
const getters = {}
export default {
  state,
  mutations,
  actions,
  getters
}
```

```
import user from './modules/user'

const store = new Vuex.Store({
  modules: {
    user
  }
})
```



# 核心概念 - 模块 module (进阶语法)

目标：掌握模块中 state 的访问语法

尽管已经分模块了，但其实子模块的状态，还是会挂到根级别的 state 中，属性名就是模块名

使用模块中的数据：

① 直接通过模块名访问 $store.state.模块名.xxx

② 通过 mapState 映射

默认根级别的映射 mapState([ 'xxx' ])

子模块的映射 mapState('模块名', ['xxx']) - 需要开启命名空间

```
export default {
  namespaced: true,
  state,
  mutations,
  actions,
  getters
}
```

# 核心概念 - 模块 module (进阶语法)

目标：掌握模块中 getters 的访问语法

使用模块中 getters 中的数据：

① 直接通过模块名访问 $store.getters['模块名/xxx ']

② 通过 mapGetters 映射

默认根级别的映射 mapGetters([ 'xxx' ])

子模块的映射 mapGetters('模块名', ['xxx']) - 需要开启命名空间

```
export default {
  namespaced: true,
  state,
  mutations,
  actions,
  getters
}
```

# 核心概念 - 模块 module (进阶语法)

目标：掌握模块中 mutation 的调用语法

注意：默认模块中的 mutation 和 actions 会被挂载到全局，需要开启命名空间，才会挂载到子模块。

调用子模块中 mutation：

① 直接通过 store 调用 $store.commit('模块名/xxx '，额外参数)

② 通过 mapMutations 映射

默认根级别的映射 mapMutations([ 'xxx' ])

子模块的映射 mapMutations('模块名', ['xxx']) - 需要开启命名空间


```
export default {
  namespaced: true,
  state,
  mutations,
  actions,
  getters
}
```

# 核心概念 - 模块 module (进阶语法)

目标：掌握模块中 action 的调用语法 (同理 - 直接类比 mutation 即可)

注意：默认模块中的 mutation 和 actions 会被挂载到全局，需要开启命名空间，才会挂载到子模块。

调用子模块中 action：

① 直接通过 store 调用 $store.dispatch('模块名/xxx '，额外参数)

② 通过 mapActions 映射

默认根级别的映射 mapActions([ 'xxx' ])

子模块的映射 mapActions('模块名', ['xxx']) - 需要开启命名空间

```
export default {
  namespaced: true,
  state,
  mutations,
  actions,
  getters
}
```

# 综合案例 - 购物车

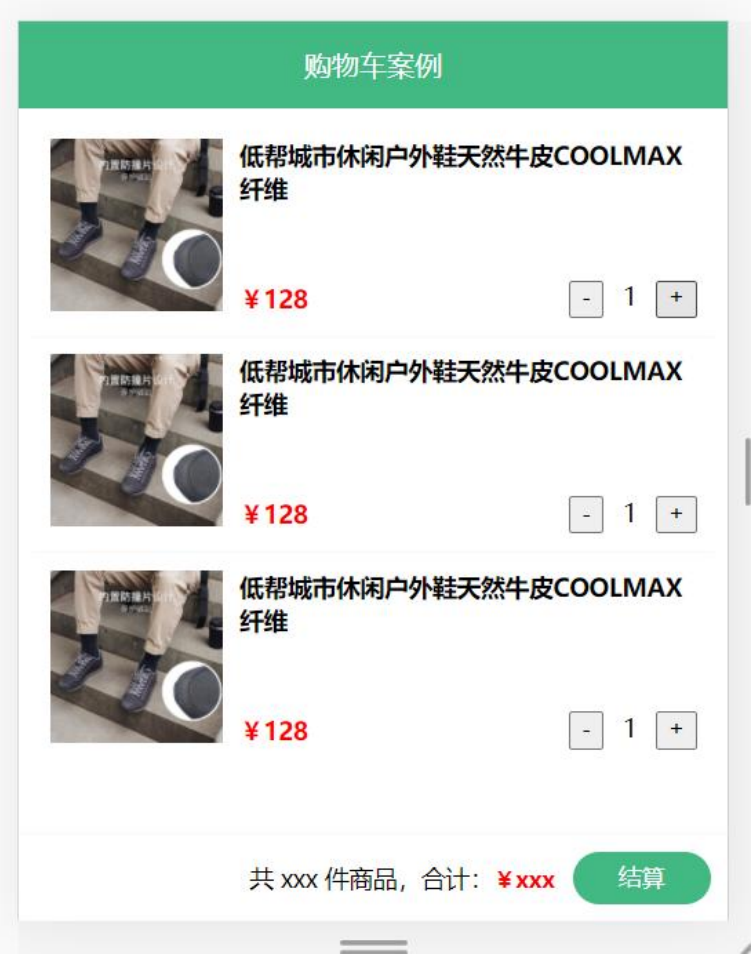
**目标：**功能分析，创建项目，构建分析基本结构

1. 功能模块分析
   ① 请求动态渲染购物车，数据存 vuex
   ② 数字框控件 修改数据
   ③ 动态计算 总价和总数量

2. 脚手架新建项目 (注意：勾选vuex)

   vue create vue-cart-demo

3. 将原本src内容清空，替换成素材的《vuex-cart-准备代码》并分析



## 综合案例 - 购物车

**目标：** 构建 cart 购物车模块

说明：既然明确数据要存 vuex，建议分模块存，购物车数据存 cart 模块，将来还会有 user 模块，article 模块...

1. 新建 `store/modules/cart.js`

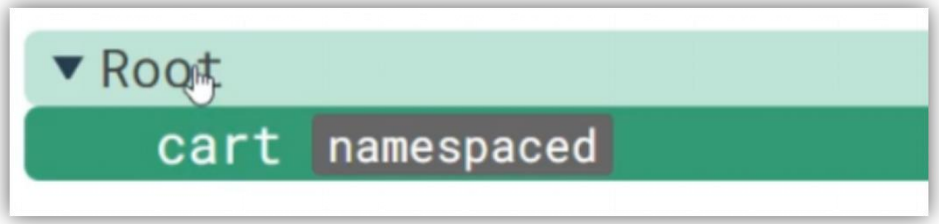
```
export default {
  namespaced: true,
  state () {
    return {
      list: []
    }
  },
}
```

2. 挂载到 vuex 仓库上 `store/index.js`

```
import cart from './modules/cart'

const store = new Vuex.Store({
  modules: {
    cart
  }
})

export default store
```



# 综合案例 - 购物车

**目标：** 基于 json-server 工具，准备后端接口服务环境

1. 安装全局工具 json-server （全局工具仅需要安装一次）【官网】

   yarn global add json-server 或 npm i json-server -g

2. 代码根目录新建一个 db 目录
3. 将资料 index.json 移入 db 目录
4. 进入 db 目录，执行命令，启动后端接口服务

   json-server index.json
5. 访问接口测试 http://localhost:3000/cart

```
PS C:\Users\Administrator\Desktop\准备代码> json-server index.json

  \{^_^}/ hi!

  Loading index.json
  Done

  Resources
  http://localhost:3000/cart
```


```
[
  {
    "id": 100001,
    "name": "低帮城市休闲户外鞋天然牛皮COOLMAX纤维",
    "price": 128,
    "count": 1,
    "thumb": "https://yanxuan-item.nosdn.127.net/3a56a913e68
  },
  {
    "id": 100002,
    "name": "网易味央黑猪猪肘330g*1袋",
    "price": 39,
    "count": 10,
    "thumb": "https://yanxuan-item.nosdn.127.net/d0a56474a84
  },
```


推荐： json-server --watch index.json (可以实时监听 json 文件的修改)

# 综合案例 - 购物车

**目标：** 请求获取数据存入 vuex, 映射渲染

数据已存入 vuex

安装 axios → 准备 actions 和 mutations → 调用action 获取数据 → 动态渲染

mapState映射

```
state: { list: [] },
mutations: {
  updateList (state, payload) {
    state.list = payload
  }
},
actions: {
  async getList (ctx) {
    const res = await axios.get('http://localhost:3000/cart')
    ctx.commit('updateList', res.data)
  }
}
```

页面调用 action

```
$store.dispatch('模块名/xxx')
```

# 综合案例 - 购物车

**目标：**修改数量功能完成　　注意：前端 vuex 数据，后端数据库数据都要更新

点击事件 → 页面中 dispatch → 提供action 函数 → 提供mutation 函数


```
mutations: {
    updateCount (state, payload) {
      const goods = state.list.find((item) => item.id === payload.id)
      goods.count = payload.count
    }
},
actions: {
  async updateCountAsync (ctx, payload) {
      await axios.patch('http://localhost:3000/cart/' + payload.id, {
        count: payload.count
      })
      ctx.commit('updateCount', payload)
    }
},
```

# 综合案例 - 购物车

**目标：** 底部 getters 统计

1. 提供 getters

```
getters: {
  total (state) {
    return state.list.reduce((sum, item) => sum + item.count, 0)
  },
  totalPrice (state) {
    return state.list.reduce((sum, item) => sum + item.count *
item.price, 0)
  }
}
```

2. 使用 getters

```
computed: {
  ...mapGetters('cart', ['total', 'totalPrice'])
}
```


传智教育旗下高端IT教育品牌