day08-前台项目08 购物车静态与交互

逻辑思维

购物车静态

1. addCartSuccess组件 查看商品详情 router-link to /detail
2. 去购物车结算 shopCar组件 放入 src pages下 注册 配置路由 找router routes
3. AddcartSuccess a标签 换 router link to /shopCart
4. shopCart 修改静态页面 ul 里面 干掉cart-list-con3 样式干掉 .cart-list-con3

   宽度修正 shopCart 宽度con1 - con 6 百分比 15 35 10 17 10 13
5. 获取购物车列表 api 接口请求函数

   reqShopCartList '/cart/cartList' 路径切忌多空格 'get'
6. store modules shopCart

   state cartList [] mutations RECEIVE_CARTLIST(state,

   async getShopCartList({commit}) await reqShopCartList

   == 会出现隐式转换 === 更为严格不会出现隐式转换 9:08 补充

   result.code === 200 commit RECEIVE_CARTLIST result.data

   else alert 获取购物车列表失败

   catch alert 请求获取购物车失败
7. ShopCart mounted 发请求 this.getShopCartList methods this.$store.dispatch()
8. Vuex RECEIVE_CARTLIST 看数据 shopcart cartInfoList

   NetWork 看请求 请求成功 [] 无数据

   添加商品 修改数量 添加购物车 去购物车结算
9. 无登录状态 用户行为 添加购物车 去购物车结算

   http请求 无状态请求 无用户标识符区分不同用户

   无状态请求：根源http协议 无状态协议

   1次 2次请求发送给后台 均被识别为请求 不识别用户

所以引入了 cookie 与 session 保持状态请求 记录用户行为

cookie 客户端 9:30补充

session 服务器 sessionkey sessionValue expiration time

sessionkey 随机字符串 sessionValue 用户id相关信息 expiration time 过期时间

session 安全性比较高 网络之间流通的只是 随机字符串

session cookie 现在没人用 比较low 落后 于是出现了 Token

10. 添加购物车的时候 添加标识：和后端配合 添加用户标识

    store modules user 添加用户 临时的唯一标识

    state userTempId getUserTempId() 专门造用户唯一标识

    点击请求 state全部初始化

11. utils userabout.js function getUserTempId NanoID UUID

    唯一 一个用户只有一个不能重复，轻易不能变 localstorage

    看看有没有 如果有 就不需要再给用户造新的

    let userTempId localStorage.getItem('userTempId_key')

    if !userTempId userTempId = uuidv4()
    localStorage.setItem('userTemp_key',userTempId)

    return userTempId

    export default getUserTempId

12. store modules user.js import getUserTempId

    每一次请求都要带上用户唯一标识 拦截器

    utils request.js 引入store

    let userTempId = store.state.userTempId/localStorage.getItem('user')

    localStorage.getItem('user')也行，但不好 效率低慢 store中快得多

    Vuex获取效率比较高 初始化取一次 快得多

    if(userTempId) config.headers.userTempId = userTempId

13. 重新 添加商品 修改数量 添加购物车 去购物车结算

Vuex RECEIVE_CARTLIST 看数据 shopcart cartInfoList

实现 未登录状态 状态保持 购物车支付 补充10:10

面试亮点 购物车支付 添加用户临时唯一标识

14. 渲染数据展示 直接展示数据 需要计算展示数据

    shopcart cartList 无购物券 有购物券

    ShopCart组件 获取数据 computed ...mapState

    两层遍历数据 cart-body vfor cart cartList :key index

    cart-list vfor cartInfo cart.cartInfoList :key cartInfo.id

15. input cart-list-con1 :checked "cartInfo.isChecked"

16. cart-list-con2 img :src "cartInfo.imgUrl" cartInfo.skuName cart.skuPrice

17. input :value cartInfo.skuNum

18. 需要计算的属性 全选 已选 总价

    已选择几件商品 打钩各商品数量合

    计算选择的商品数量 统计必然是 reduce 双层数组高阶方法 外层看作遍历 搞内层

    高阶函数：参数或者返回值是函数的函数叫高阶方法或高阶函数

    computed checkedNum return this.cartList.reduce((prev, item) =>
    {item.cartInfoList.reduce((prave1,item1)=>{if(item1.isCheed){preve1 +=
    item1.skuNum}return prev1},0) return prev}, 0)

    已选择 {{check}}

19. 总价 shopCart组件 allMoney(){}

20. 全选 11:20

    单个CheckBox v-model收集的是布尔值 这个值最终作用的就是checked

    成组的CheckBox v-model 收集的是数组

    v-model 和 checked 点击事件

    cn.vuejs.org/v2/guide/forms/html

    v-model isCheckAll:{

    get(){

一项没选 整体没选

```
return this.cartList.every(item => {

return item.cartInfoList.every(item1 => {

return item1.isChecked
```

}

})

},

简化 return this.cartList.every(item => item.cartInfoList

set(){

}

}

单个使用多选框其实有两套方法

i. 采用v-model配合计算属性(get和set)
ii. 采用check属性值绑 11:30

修改数量 修改状态 删除(单个删除，多个删除) 上市公司 资本就注入

购物车交互

修改数量 修改状态 删除(单个删除，多个删除)

修改购物车商品数量

21. shopCart组件 @click changeNum() -1 1 cartInfo

@change changeNum($event.target.value - cartInfo.skuInfo)

methods 点击修改购物车商品数量 changeNum(cartInfo, disNum)

判断最终的量是不是小于1 如果小于1就让最终的量等于1

变化的量 就应该修正为 1 - 原本有的量

if(cartInfo.skuNum + disNum < 1) disNum = 1 - cartInfo.skuNum

22. shopCart组件 async changeNum 发请求

    try this.$store.dispatch('addOrUpdateShopCart', {skuIdId:cartInfo.skuId,skuNum:disNum}) alert 修改购物车数量成功 this.getShkoCartList()

    catch alert 修改购物车数量失败

    测试：去购物车结算 点击+-或输入 数量 进行 商品数量修改操作

修改购物车单选全选状态

23. ShopCart组件 li class="cart-list-con1"处 @click updateOneIsChecked(cartInfo)

    methods 点击修改单个的选中状态 updateOneIsChecked(cartInfo)

    api 接口请求函数 修改单个的选中状态

    export const reqUpdateOneIsChecked (skuId,isChecked)

    return request url /cart/checkcart/${skuId}$

24. store modules shopcart.js 修改购物车单个的选中状态

    async updateOneIsChecked {commit},{skuId,isChecked}

    try result await reqUpdateOneIsChecked(skuId,isChecked)

    14:32 一般return错误对象 以免后面继续 点操作符

    if result.code === 200 return "ok" else return Promise.reject(new Error(('failed')

    catch return Promise.reject(error)

25. ShopCart组件 methods updateOneIsChecked 续写

    try this.$store.dispatch("updateOneIsChecked", {

    skuId: cartInfo.skuId,

    isChecked: cartInfo.isChecked ? 0 : 1,

    });

    alert 修改单车状态成功

    this.getShopCartList

    catch alert 修改单车状态失败

    测试：走一步 侧一步 单选 反选 全选为计算出来的

26. api接口文档 最后 批量选中购物车接口 {isChecked}

    api index.js 接口请求函数 修改多个购物车选中状态

    export const reqUpdateAllIsChecked (skuIdList,isChecked)

    return request ({

    url: /cart/batchCheckCart/${isChecked} methods: 'post' data: skuIdList

    })
27. vuex 三连环 store modules shopcart.js 批量修改购物车的选中状态

    async updateAllIsChecked {commit},{skuIdList,Ischecked}

    try result await reqUpdateAllIsChecked(skuIdList,isChecked)

    if result.code 200 return 'ok'

    else return Promise.reject(new Error('faild))

    catch return Promise.reject(error)
28. ShopCart组件 197H set(val) 先准备两个参数

    val拿的是你修改后的最新值 这个值一会给你上面全部要改得值

    {let isChecked = val ? 1 : 0

    let skuIdList = []

    第一种 forEach

    this.cartList.forEach(item => {

```
item.cartInfoList.forEach(item1 => {

        if(item1.isChecked !== isChecked){

                skuIdList.push(item1.skuId)
```

    }

    测试：console.log(skuIdList) F12 点击全选

    第二种 reduce 工作中常用 高阶方法常用 比forEach效率高

concat 合并数组 将里面数组合并至调用的数组返回新数组

skuIdList = this.cartList.reduce((prev,item) => {

prev = prev.concat (item.cartInfoList.reduce((prev1,item1) => {

```
        if(item1.isChecked !== isChecked){
                prev1.push(item1.skuId)
        }
        return prev1
}, []))
return prev
```

}, [])

测试：console.log(skuIdList)

此处 await必须加 不然点击全选按钮不等待 异步延迟加载可能出现问题

try await this.$store.dispatch('updateAllIsChecked',{skuIdList,isChecked})

alert 修改所有的购物车状态成功

this.getShopCartList()

catch alert 修改所有的购物车状态失败

删除(单个删除，多个删除)

29. ShopCart组件 64H javascript:; @click

    methods 点击删除单个购物车

    api index.js 删除单个购物车 接口文档 删除购物车商品 {skuId} DELETE

    export const reqDeleteOneCart (skuId)

    return request url method: detele

30. store module shopcart.js 删除单个购物车

    async deleteOneCart {commit},skuId

try reqDeleteOneCart(skuId)

if result.code === 200 return 'ok'

else

31. 续写 methods deleteOneCart(cartInfo)
32. try await this.$store.dispatch('deleteOneCart',cartInfo.skuId)

    alert 删除购物车成功

    this.getShopCartList

    catch
33. shopCart组件 82H 删除选中的商品 @click deleteCheckedCart javascript:;

    methods 点击删除选中的购物车 deleteCheckedCart 发请求 写api接口请求函数

    api index.js export const reqDeleteCheckedCart = (skuIdList)

    {return request({ url /cart/batchDeleteCart, method: 'delete' data: skuIdList})}
34. store shopcart.js Vuex 三连环 删除选中的购物车

    async deleteCheckedCart {commit}, skuIdList

    try result await reqDeleteCheckedCart(skuIdList)

    if result.code === 200 return 'ok'

    else return Promise.reject(new Error('failed')
35. 续写 methods deleteCheckedCart

    let skuIdList = []

    this.cartList.forEach((item) => {

```
itme.cartInfoList.forEach((item1) => {
        if(item1.isChecked){
                skuIdList.push(item1.skuId)
```

    try await this.$store.dispatch
36. 全选 勾选未取消 isCheckAll get() && length !== 0