

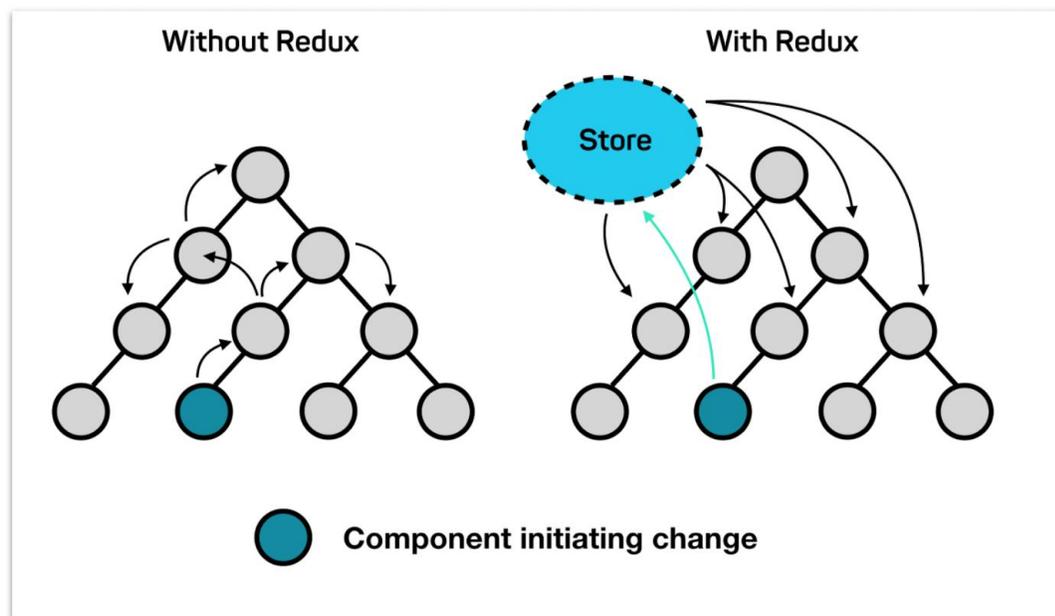


Redux快速上手

什么是Redux?

Redux 是React最常用的**集中状态管理工具**，类似于Vue中的Pinia（Vuex），**可以独立于框架运行**

作用：通过集中管理的方式管理应用的状态



Redux快速体验

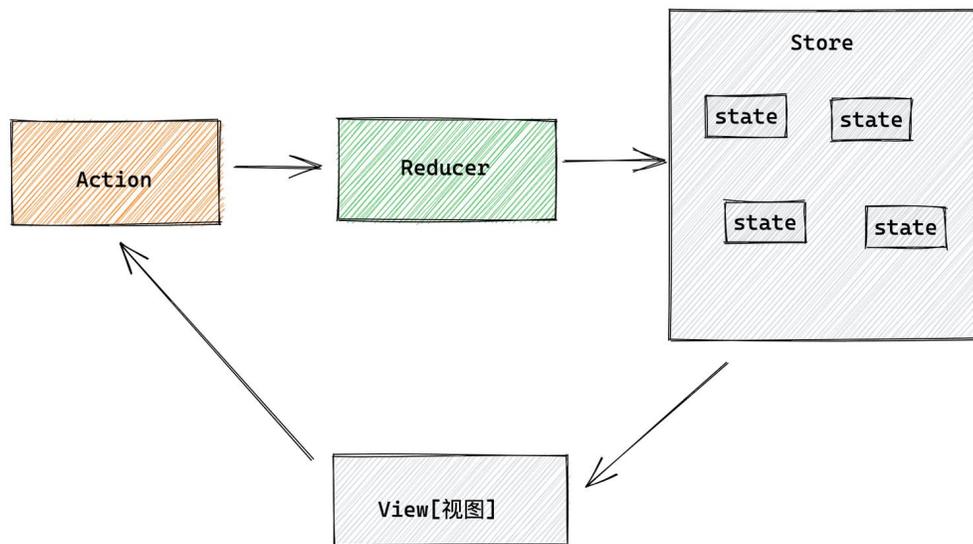
不和任何框架绑定，不使用任何构建工具，使用纯Redux实现计数器



使用步骤：

1. 定义一个 **reducer 函数**（根据当前想要做的修改返回一个新的状态）
2. 使用createStore方法传入 reducer函数 生成一个**store实例对象**
3. 使用store实例的 **subscribe方法** 订阅数据的变化（数据一旦变化，可以得到通知）
4. 使用store实例的 **dispatch方法提交action对象** 触发数据变化（告诉reducer你想怎么改数据）
5. 使用store实例的 **getState方法** 获取最新的状态数据更新到视图中

Redux管理数据流程梳理



为了职责清晰，数据流向明确，Redux把整个数据修改的流程分成了三个核心概念，分别是：`state`、`action`和`reducer`

1. `state` - 一个对象 存放着我们管理的数据状态
2. `action` - 一个对象 用来描述你想怎么改数据
3. `reducer` - 一个函数 更具`action`的描述生成一个新的`state`

02

Redux与React – 环境准备

配套工具

在React中使用redux，官方要求安装两个其他插件 - **Redux Toolkit** 和 **react-redux**

1. Redux Toolkit (RTK) - 官方推荐编写Redux逻辑的方式，是一套工具的集合集，**简化书写方式**

简化store的配置方式

内置immer支持可变式状态修改

内置thunk更好的异步创建

2. react-redux - 用来 **链接 Redux 和 React组件** 的中间件



配置基础环境

1. 使用 CRA 快速创建 React 项目

```
npx create-react-app react-redux
```

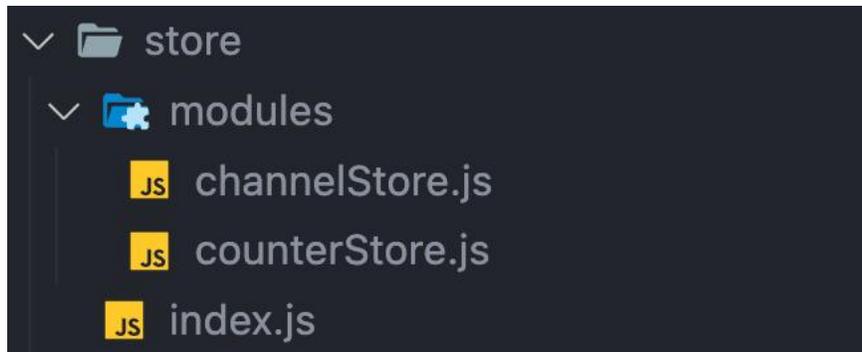
2. 安装配套工具

```
npm i @reduxjs/toolkit react-redux
```

3. 启动项目

```
npm run start
```

store目录结构设计

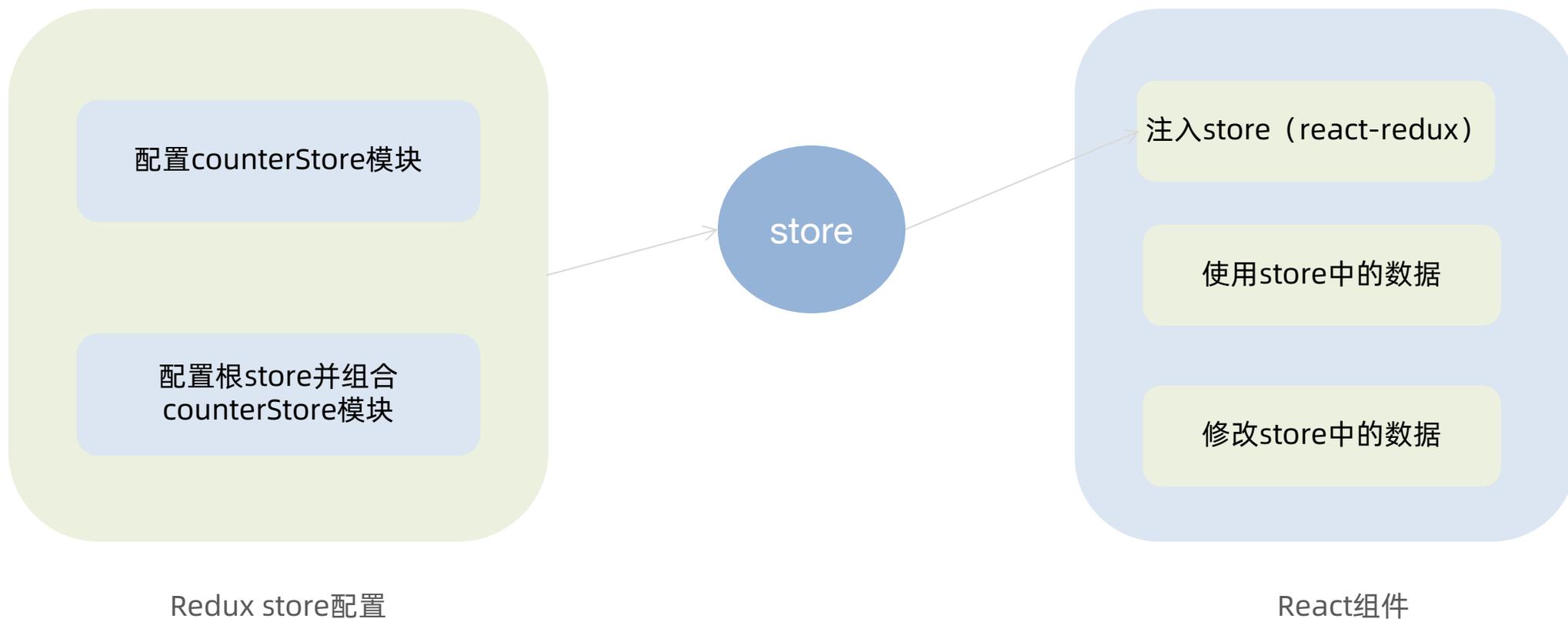


1. 通常集中状态管理的部分都会单独创建一个单独的 `store` 目录
2. 应用通常会有很多个子store模块，所以创建一个 `modules` 目录，在内部编写业务分类的子store
3. store中的入口文件 index.js 的作用是组合modules中所有的子模块，并导出store

03

Redux与React – 实现counter

整体路径熟悉



使用React Toolkit 创建 counterStore

```
1 import { createSlice } from "@reduxjs/toolkit"
2
3 const counterStore = createSlice({
4   name: 'counter',
5   // 初始状态数据
6   initialState: {
7     count: 0
8   },
9   // 修改数据的同步方法
10  reducers: {
11    increment (state) {
12      state.count++
13    },
14    decrement (state) {
15      state.count--
16    },
17  }
18 })
19
20 // 解构出创建action对象的函数 (actionCreator)
21 const { increment, decrement } = counterStore.actions
22 // 获取reducer函数
23 const counterReducer = counterStore.reducer
24 // 导出创建action对象的函数和reducer函数
25 export { increment, decrement }
26 export default counterReducer
```

counterStore.js



```
1 import { configureStore } from "@reduxjs/toolkit"
2
3 import counterReducer from "../modules/counterStore"
4
5 // 创建根store组合子模块
6 const store = configureStore({
7   reducer: {
8     counter: counterReducer
9   }
10 })
11
12 export default store
```

store/index.js

为React注入store

react-redux负责把Redux和React 链接 起来， 内置 Provider组件 通过 store 参数把创建好的store实例注入到应用中， 链接正式建立

```
1 import store from './store'
2 import { Provider } from 'react-redux'
3
4 const root = ReactDOM.createRoot(document.getElementById('root'))
5 root.render(
6   <Provider store={store}>
7     <App />
8   </Provider>
9 )
10
```

React组件使用store中的数据

在React组件中使用store中的数据，需要用到一个钩子函数 - `useSelector`，它的作用是把store中的数据映射到组件中，使用样例如下：

```
1 const { count } = useSelector(state => state.counter)
```

```
1 import { configureStore } from "@reduxjs/toolkit"
2
3 import counterReducer from "../modules/counterStore"
4
5 // 创建根store组合子模块
6 const store = configureStore({
7   reducer: {
8     counter: counterReducer
9   }
10 })
11
12 export default store
```

React组件修改store中的数据

React组件中修改store中的数据需要借助另外一个hook函数 - `useDispatch`，它的作用是生成提交action对象的dispatch函数，使用样例如下：

```
1 import { useDispatch, useSelector } from 'react-redux'
2 // 导入创建action对象的方法
3 import { decrement, increment } from './store/modules/counterStore'
4 function App () {
5   const { count } = useSelector(state => state.counter)
6   // 得到dispatch函数
7   const dispatch = useDispatch()
8   return (
9     <div className="App">
10      /* 调用dispatch提交action对象 */
11      <button onClick={() => dispatch(decrement())}>-</button>
12      <span>{count}</span>
13      <button onClick={() => dispatch(increment())}>+</button>
14    </div>
15  )
16 }
```



总结

1. 组件中使用哪个hook函数获取store中的数据?

`useSelector`

2. 组件中使用哪个hook函数获取dispatch方法?

`useDispatch`

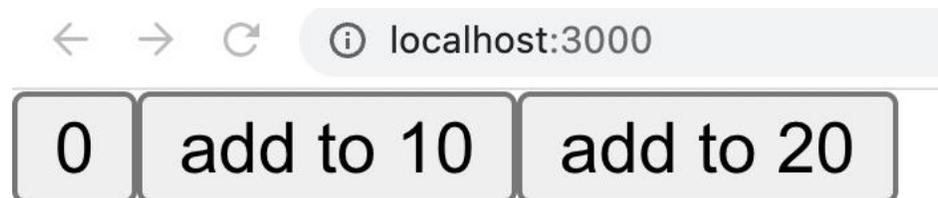
3. 如何得到要提交action对象?

执行store模块中导出的actionCreator方法

04

Redux与React – 提交action传参

需求说明



组件中有两个按钮`add to 10`和`add to 20`可以直接把count值修改到对应的数字，目标count值是在组件中传递过去的，需要在提交action的时候传递参数

提交action传参实现需求

在reducers的同步修改方法中添加action对象参数，在调用actionCreator的时候传递参数，参数会被传递到action对象payload属性上

```
counterStore.js M x
src > store > modules > js counterStore.js > ...
10 reducers: {
11   increment (state) {
12     state.count++
13   },
14   decrement (state) {
15     state.count--
16   },
17   addToNum (state, action) {
18     state.count = action.payload
19   }
20 }
21 }
22
23 // 解构出创建action对象的函数 (actionCreator)
24 const { increment, decrement, addToNum } = counterStore.actions
25 // 获取reducer函数
26 const counterReducer = counterStore.reducer
27 // 导出创建action对象的函数和reducer函数
28 export { increment, decrement, addToNum }
29 export default counterReducer
30

App.js M x
src > js App.js > ...
1 import { useDispatch, useSelector } from 'react-redux'
2 // 导入创建action对象的方法
3 import { decrement, increment, addToNum } from './store/modules/
4 function App () {
5   const { count } = useSelector(state => state.counter)
6   // 得到dispatch函数
7   const dispatch = useDispatch()
8   return (
9     <div className="App">
10       /* 调用dispatch提交action对象 */
11       <button onClick={() => dispatch(decrement())}>-</button>
12       <span>{count}</span>
13       <button onClick={() => dispatch(increment())}>+</button>
14       <button onClick={() => dispatch(addToNum(10))}>add to 10</
15       <button onClick={() => dispatch(addToNum(20))}>add to 20</
16     </div>
17   )
18 }
19
20 export default App
21
```

05

Redux与React – 异步状态操作

需求理解

- 推荐
- html
- 开发者资讯
- C++
- CSS
- 数据库
- 区块链
- go
- 产品
- 后端
- linux
- 人工智能
- php
- javascript
- 架构
-



Redux Store



React组件

异步操作样板代码

```
1 const channelStore = createSlice({
2   name: 'channel',
3   initialState: {
4     channelList: []
5   },
6   reducers: {
7     setChannels (state, action) {
8       state.channelList = action.payload
9     }
10  }
11 })
```

```
1 const { setChannels } = channelStore.actions
2 const url = 'http://geek.itheima.net/v1_0/channels'
3 const fetchChannelList = () => {
4   return async (dispatch) => {
5     const res = await axios.get(url)
6     dispatch(setChannels(res.data.data.channels))
7   }
8 }
9
10 export { fetchChannelList }
```

```
1 const dispatch = useDispatch()
2 useEffect(() => {
3   dispatch(fetchChannelList())
4 }, [dispatch])
```

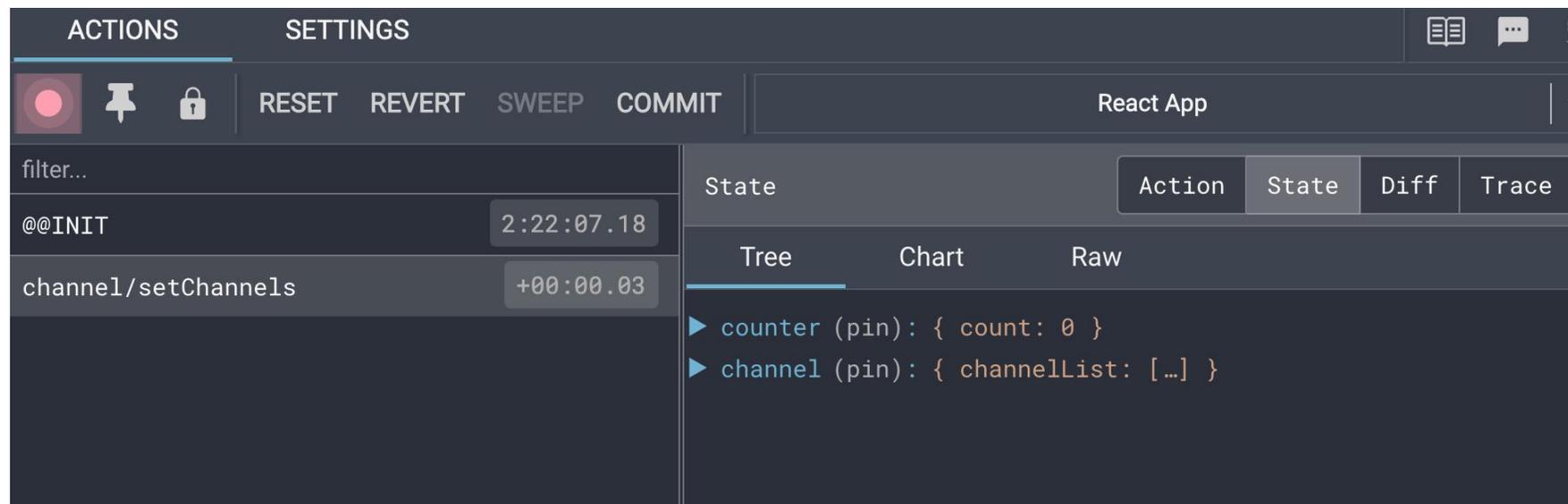
1. 创建store的写法保持不变，配置好同步修改状态的方法
2. 单独封装一个函数，在函数内部return一个新函数，在新函数中
 - 2.1 封装异步请求获取数据
 - 2.2 调用同步actionCreator传入异步数据生成一个action对象，并使用dispatch提交
3. 组件中dispatch的写法保持不变

06

Redux调试 – devtools

安装chrome调试工具

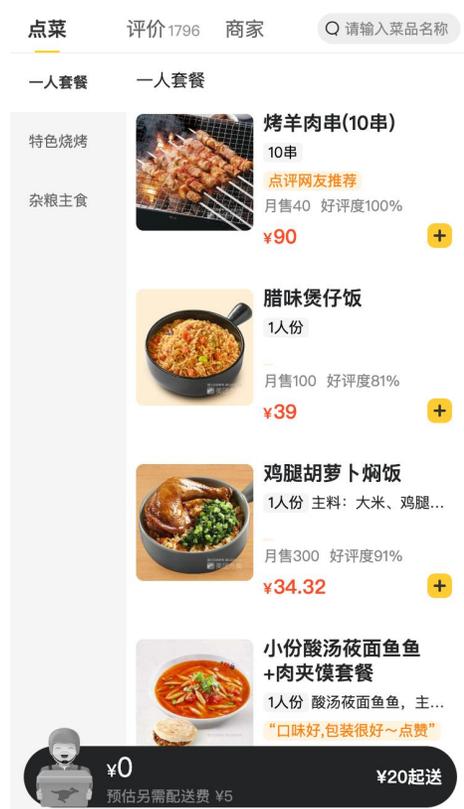
Redux官方提供了针对于Redux的调试工具，支持实时state信息展示，action提交信息查看等



07

美团外卖-案例演示和环境准备

案例演示



功能列表:

商品列表和分类渲染

添加商品

购物车操作

订单数量统计和高亮实现

基本开发思路：使用 RTK (Redux Toolkit) 来管理应用状态, 组件负责 数据渲染 和 dispatch action

准备并熟悉环境

1. 克隆项目到本地（内置了基础静态组件和模版）

```
git clone http://git.itcast.cn/heimaqianduan/redux-meituan.git
```

2. 安装所有依赖

```
npm i
```

3. 启动mock服务（内置了json-server）

```
npm run serve
```

4. 启动前端服务

```
npm run start
```

08

美团外卖-分类和商品列表渲染

需求理解



实现步骤

启动项目（mock服务 + 前端服务）



使用RTK编写store（异步action）

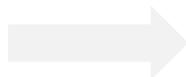
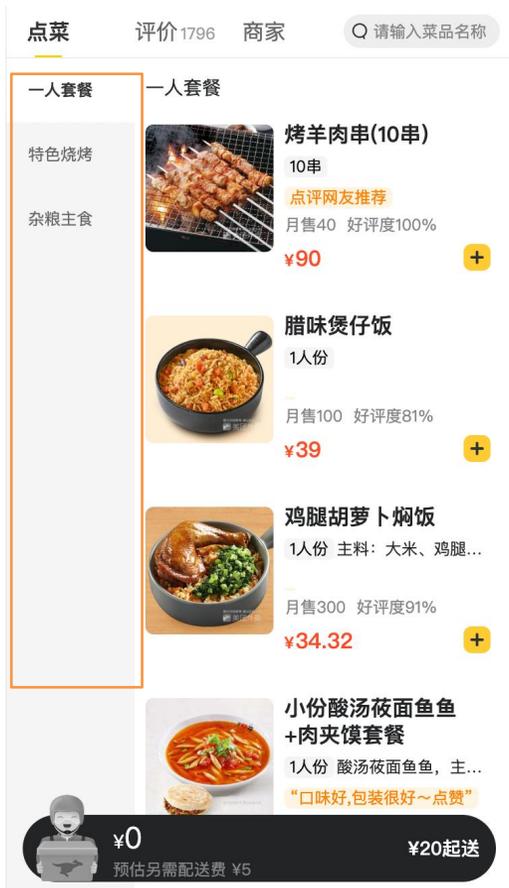


组件触发action并且渲染数据

09

美团外卖-点击分类激活交互实现

理解需求



Tab切换类交互

记录当前点击项
(activeIndex)

动态控制激活类名
(activeIndex === index)

步骤分析

使用RTK编写管理activeIndex



组件中点击时触发action更改activeIndex

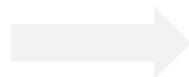
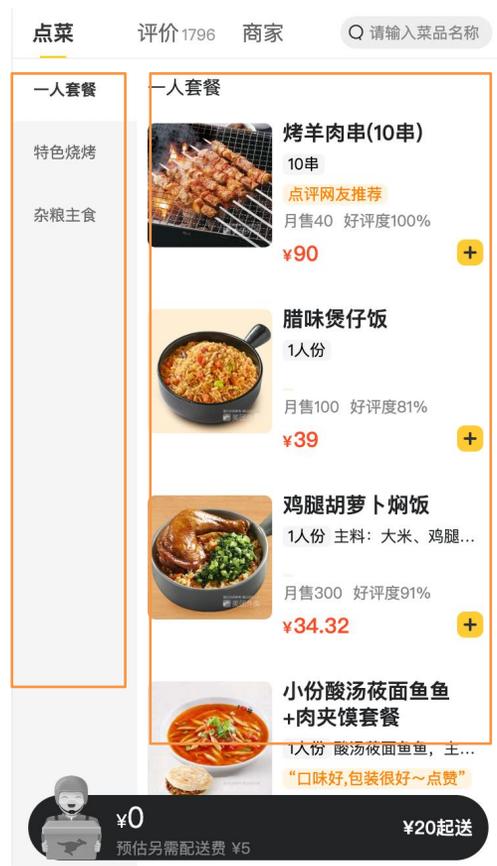


动态控制激活类名显示

10

美团外卖-商品列表切换显示

需求理解



条件渲染：控制对应项显示

`activeIndex === index && 视图`

11

美团外卖-添加购物车实现

需求理解

点菜 评价 1796 商家

一人套餐 一人套餐

特色烧烤

杂粮主食

-  **烤羊肉串(10串)**
10串
点评网友推荐
月售40 好评度100%
¥90
-  **腊味煲仔饭**
1人份
月售100 好评度81%
¥39
-  **鸡腿胡萝卜焖饭**
1人份 主料: 大米、鸡腿...
月售300 好评度91%
¥34.32
-  **小份酸汤筱面鱼鱼+肉夹馍套餐**
1人份 酸汤筱面鱼鱼, 主...



点击 + 号添加当前商品到购物车列表

实现步骤

使用RTK管理新状态cartList



思路：如果添加过，只更新数量count,没有添加过，直接push进去



组件中点击时收集数据提交action添加购物车

11

美团外卖-统计区域功能实现

需求理解



1. 购物车数量和总价统计

2. 高亮功能实现

实现步骤

基于store中的cartList的length渲染数量



基于store中的cartList累加price * count



购物车cartList的length不为零则高亮

12

美团外卖-购物车列表功能实现

需求理解



1. 控制列表渲染

2. 购物车增减逻辑实现

3. 清空购物车实现

实现步骤

使用cartList遍历渲染列表



RTK中增加增减reducer, 组件中提交
action

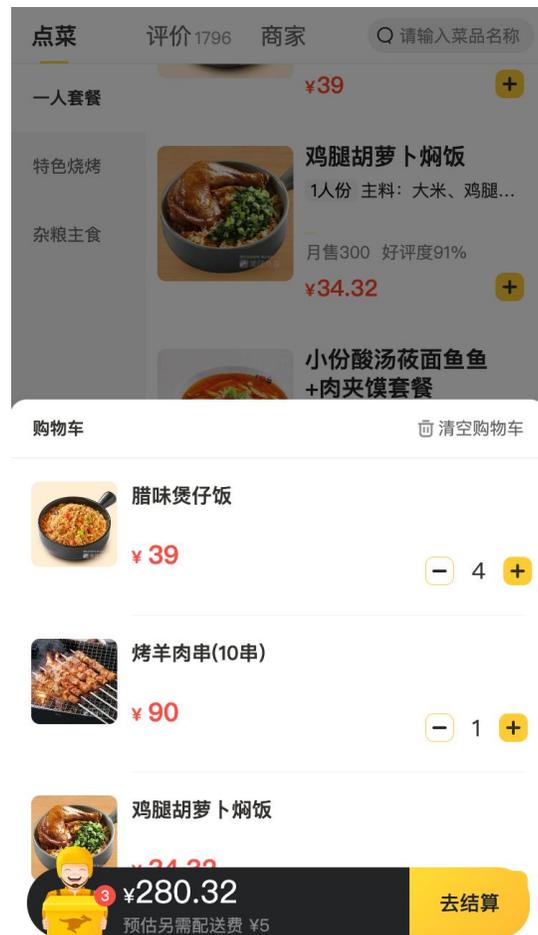


RTK中增加清除购物车reducer,组件中提交
action

13

美团外卖-控制购物车显示和隐藏

需求劣迹



1. 点击统计区域时，购物车列表显示

2. 点击蒙层区域时，购物车和蒙层隐藏

实现步骤

使用useState声明控制显隐的状态



点击统计区域设置状态为true



点击蒙层区域设置状态为false



传智教育旗下高端IT教育品牌