电商商品列表页开发文档

> document：抖音作业 by markdown，作者：蔡陈烨阳，文档主要介绍了项目的实现功能和流程架构。

项目说明

这是一个基于 Vue 3 + Element Plus + Pinia 开发的电商商品列表页面，实现了商品展示、多维筛选、排序、分页和购物车等核心功能。

技术栈：Vue 3.3、Pinia 2.1、Element Plus 2.4、Vite 5.0

核心功能

1. 商品展示

- 24个模拟商品数据，涵盖手机、电脑、耳机等电子产品
- 响应式网格布局，自适应不同屏幕尺寸
- 商品卡片包含：图片、名称、价格、品牌、销量、评分

2. 筛选功能

支持四种筛选方式，可组合使用：

- **分类筛选**：手机通讯、电脑办公、摄影摄像、智能设备
- **品牌筛选**：Apple、华为、小米、联想等
- **价格区间**：拖动滑块选择价格范围（0-20000元）
- **关键词搜索**：搜索商品名称或描述，300ms防抖优化

3. 排序功能

提供6种排序方式：

- 综合排序（默认）
- 价格升序/降序
- 销量排序
- 评分排序
- 最新上架

4. 分页

- 每页显示数量可选：12/20/40/60
- 页码跳转和上下翻页
- 筛选或排序变化后自动回到第一页

5. 购物车

- 添加商品到购物车，自动合并相同商品
- 修改商品数量，受库存限制
- 实时计算总价和商品数量
- 数据持久化到 LocalStorage

技术实现

数据处理流程

采用三级处理模式：

```
原始数据 → 筛选 → 排序 → 分页 → 渲染
```

通过 computed 实现自动计算，任一条件变化都会触发重新计算。

状态管理

使用 Pinia 管理应用状态，分为两个 store：

productStore（商品状态）

- products：商品列表
- filters：筛选条件（分类、品牌、价格、关键词）
- sortBy：排序方式
- pagination：分页信息
- 三个计算属性：filteredProducts、sortedProducts、paginatedProducts

cartStore（购物车状态）

- items：购物车商品数组
- 计算总数量和总价
- 增删改操作自动保存到 LocalStorage

关键代码逻辑

筛选实现

```
const filteredProducts = computed(() => {
  let result = [...products.value]

  if (filters.category)
    result = result.filter(p => p.category === filters.category)

  if (filters.brand)
    result = result.filter(p => p.brand === filters.brand)

  if (filters.priceRange.length === 2)
    result = result.filter(p =>
```

```
      p.price >= filters.priceRange[0] &&
      p.price <= filters.priceRange[1]
    )

  if (filters.keyword)
    result = result.filter(p =>
      p.name.includes(filters.keyword) ||
      p.description.includes(filters.keyword)
    )

  return result
})
```

排序实现

```
const sortedProducts = computed(() => {
  const result = [...filteredProducts.value]

  switch (sortBy.value) {
    case 'price-asc': return result.sort((a, b) => a.price - b.price)
    case 'price-desc': return result.sort((a, b) => b.price - a.price)
    case 'sales': return result.sort((a, b) => b.sales - a.sales)
    case 'rating': return result.sort((a, b) => b.rating - a.rating)
    default: return result
  }
})
```

购物车持久化

```
const addToCart = (product, quantity = 1) => {
  const existing = items.value.find(item => item.id === product.id)

  if (existing) {
    existing.quantity += quantity
  } else {
    items.value.push({ ...product, quantity })
  }

  localStorage.setItem('shopping-cart', JSON.stringify(items.value))
}
```

UI 设计

视觉风格

- 主色调：紫色渐变（#667eea → #764ba2）
- 卡片圆角：16px
- 悬停效果：向上位移 + 阴影加深
- 过渡动画：300ms 缓动

布局方案

- 顶部：Logo + 搜索框 + 购物车
- 左侧：筛选面板（分类、品牌、价格）
- 右侧：排序选项 + 商品网格 + 分页器

响应式适配

- 桌面端（>1200px）：4列网格
- 平板端（768-1200px）：2-3列网格
- 手机端（<768px）：2列网格，隐藏左侧筛选

性能优化

1. **图片处理**
   - 使用 Unsplash 图片服务，URL 带参数优化（w=600&h=600&q=80）
   - loading="lazy" 原生懒加载
   - 加载失败显示占位图
2. **防抖优化**
   - 搜索输入防抖 300ms
   - 筛选条件变化防抖处理
3. **数据缓存**
   - productStore 缓存商品列表
   - cartStore 持久化到 LocalStorage
4. **代码分割**
   - 路由懒加载
   - Vite 自动代码分割

项目结构

```
src/
├── api/
|   ├── product.js          # 商品接口
|   └── mockData.js         # 模拟数据
├── components/
|   ├── ProductCard.vue     # 商品卡片
|   └── CartDrawer.vue      # 购物车抽屉
├── stores/
|   ├── productStore.js     # 商品状态
|   └── cartStore.js        # 购物车状态
├── views/
|   ├── ProductList.vue     # 商品列表页
|   └── ProductDetail.vue   # 商品详情页
├── router/
|   └── index.js            # 路由配置
```

```
├── styles/
|   └── index.scss          # 全局样式
├── App.vue
└── main.js
```

使用说明

启动项目

```bash
# 安装依赖
npm install

# 开发模式
npm run dev

# 生产构建
npm run build
```

替换真实接口

修改 `src/api/product.js`：

```javascript
const USE_MOCK = false  // 改为 false

const request = axios.create({
  baseURL: 'https://your-api.com/api',
  timeout: 10000
})
```

自定义主题

修改 `src/styles/index.scss`：

```scss
// 修改主色调
$primary-color: #your-color;
$gradient-primary: linear-gradient(135deg, #color1 0%, #color2 100%);
```

已知问题和改进方向

当前限制

- 使用模拟数据，未对接真实后端
- 购物车只有前端逻辑，未与后端同步
- 图片使用第三方服务，建议替换为项目自有资源

可扩展功能

- 商品详情页
- 用户登录和账户管理
- 订单结算流程
- 商品收藏功能
- 浏览历史记录
- 商品对比功能

技术要点总结

1. **状态管理**：Pinia 实现集中式状态管理，逻辑清晰，易于维护
2. **计算链**：filteredProducts → sortedProducts → paginatedProducts 三级计算，自动响应数据变化
3. **组件解耦**：ProductCard、CartDrawer 独立封装，可复用
4. **用户体验**：防抖、懒加载、骨架屏、友好的错误提示
5. **代码规范**：使用 Composition API，代码组织清晰

开发结束时间：2024-11-27
版本：V3.0 Final