0. 实现步骤

① 初始化项目基本结构

② 封装 EsHeader 组件

③ 基于 axios 请求商品列表数据

④ 封装 EsFooter 组件

⑤ 封装 EsGoods 组件

⑥ 封装 EsCounter 组件

1. 初始化项目结构

1. 运行如下的命令，初始化 `vite` 项目：

```
npm init vite-app code-cart
cd code-cart
npm install
```

2. 清理项目结构：
   - 把 bootstrap 相关的文件放入 `src/assets` 目录下
   - 在 `main.js` 中导入 `bootstrap.css`
   - 清空 `App.vue` 组件
   - 删除 `components` 目录下的 `HelloWorld.vue` 组件
3. 为组件的样式启用 less 语法：

```
npm i less -D
```

4. 初始化 `index.css` 全局样式如下：

```
:root {
  font-size: 12px;
}
```

2. 封装 es-header 组件

2.1 创建并注册 EsHeader 组件

1. 在 `src/components/es-header/` 目录下新建 `EsHeader.vue` 组件：

```
<template>
  <div>EsHeader 组件</div>
</template>

<script>
export default {
  name: 'EsHeader',
}
</script>

<style lang="less" scoped></style>
```

2. 在 `App.vue` 组件中导入并注册 `EsHeader.vue` 组件：

```
// 导入 header 组件
import EsHeader from './components/es-header/EsHeader.vue'

export default {
  name: 'MyApp',
  components: {
    // 注册 header 组件
    EsHeader,
  },
}
```

3. 在 `App.vue` 的 `template` 模板结构中使用 `EsHeader` 组件：

```
<template>
  <div>
    <h1>App 根组件</h1>

    <!-- 使用 es-header 组件 -->
    <es-header></es-header>
  </div>
</template>
```

2.2 封装 es-header 组件

0. 封装需求：

- 允许用户自定义 `title` 标题内容
- 允许用户自定义 `color` 文字颜色
- 允许用户自定义 `bgcolor` 背景颜色
- 允许用户自定义 `fsize` 字体大小
- `es-header` 组件必须**固定定位**到页面顶部的位置，**高度**为 45px，**文本居中**，z-index 为 999

1. 在 `es-header` 组件中封装以下的 props 属性：

```
export default {
  name: 'EsHeader',
  props: {
    title: { // 标题内容
      type: String,
      default: 'es-header',
    },
    bgcolor: { // 背景颜色
      type: String,
      default: '#007BFF',
    },
    color: { // 文字颜色
      type: String,
      default: '#ffffff',
    },
    fsize: { // 文字大小
      type: Number,
      default: 12,
    },
  },
}
```

2. 渲染标题内容，并动态为 DOM 元素绑定行内的 `style` 样式对象：

```
<template>
  <div :style="{ color: color, backgroundColor: bgcolor, fontSize: fsize + 'px' }">{{ title }}</div>
</template>
```

3. 为 DOM 节点添加 `header-container` 类名，进一步美化 `es-header` 组件的样式：

```
<template>
```

```
    <div class="header-container" :style="{ color: color, backgroundColor: bgcolor, fontSize: fsize + 'px' }">
      {{ title }}
    </div>
  </template>

  <style lang="less" scoped>
  .header-container {
    height: 45px;
    line-height: 45px;
    text-align: center;
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    z-index: 999;
  }
  </style>
```

4. 在 `App` 根组件中使用 `es-header` 组件时，通过 `title` 属性指定标题内容：

```
<template>
  <div class="app-container">
    <h1>App 根组件</h1>

    <!-- 为 es-header 组件指定 title 属性的值 -->
    <es-header title="购物车案例"></es-header>
  </div>
</template>
```


3. 基于 axios 请求商品列表数据

3.1 全局配置 axios

1. 运行如下的命令安装 `axios`：

```
npm i axios -S
```

2. 在 `main.js` 入口文件中导入并全局配置 axios：

```
import { createApp } from 'vue'
import App from './App.vue'
import './assets/css/bootstrap.css'
import './index.css'

// 导入 axios
import axios from 'axios'

const app = createApp(App)

// 配置请求的根路径
axios.defaults.baseURL = 'https://www.escook.cn'
// 将 axios 挂载为全局的 $http 自定义属性
app.config.globalProperties.$http = axios

app.mount('#app')
```

3.2 请求商品列表数据

1. 在 `App.vue` 根组件中声明如下的 `data` 数据：

```
data() {
  return {
    // 商品列表的数据
    goodslist: [],
  }
},
```

2. 在 `App.vue` 根组件的 `created` 生命周期函数中，**预调用** `获取商品列表数据` 的 `methods` 方法：

```
// 组件实例创建完毕之后的生命周期函数
created() {
  // 调用 methods 中的 getGoodsList 方法，请求商品列表的数据
  this.getGoodsList()
},
```

3. 在 `Ap.vue` 根组件的 `methods` 节点中，声明刚才预调用的 `getGoodsList` 方法：

```
methods: {
  // 请求商品列表的数据
  async getGoodsList() {
    // 1. 通过组件实例 this 访问到全局挂载的 $http 属性，并发起 Ajax 数据请求
    const { data: res } = await this.$http.get('/api/cart')
    // 2. 判断请求是否成功
    if (res.status !== 200) return alert('请求商品列表数据失败！')
    // 3. 将请求到的数据存储到 data 中，供页面渲染期间使用
    this.goodslist = res.list
  },
},
```

4. 封装 es-footer 组件

4.1 创建并注册 EsFooter 组件

1. 在 `src/components/es-footer/` 目录下新建 `EsFooter.vue` 组件：

```
<template>
  <div>EsFooter 组件</div>
</template>

<script>
export default {
  name: 'EsFooter',
}
</script>

<style lang="less" scoped></style>

```

2. 在 `App.vue` 组件中导入并注册 `EsFooter.vue` 组件：

```
// 导入 header 组件
import EsHeader from './components/es-header/EsHeader.vue'
// 导入 footer 组件
import EsFooter from './components/es-footer/EsFooter.vue'

```

```
export default {
  name: 'MyApp',
  components: {
    // 注册 header 组件
    EsHeader,
    // 注册 footer 组件
    EsFooter,
  },
}
```

3. 在 `App.vue` 的 `template` 模板结构中使用 `EsFooter` 组件：

```
<template>
  <div>
    <h1>App 根组件</h1>

    <!-- 使用 es-header 组件 -->
    <es-header></es-header>
    <!-- 使用 es-footer 组件 -->
    <es-footer></es-footer>
  </div>
</template>
```


4.2 封装 es-footer 组件

4.2.0 封装需求

1. `es-footer` 组件必须**固定定位**到 页面底部 的位置，**高度**为 `50px`，**内容两端贴边对齐**，`z-index` 为 `999`
2. 允许用户自定义 `amount` 总价格（单位是元），并在渲染时 保留两位小数
3. 允许用户自定义 `total` 总数量，并渲染到 结算按钮 中；如果要结算的商品数量为`0`，则 禁用结算按钮
4. 允许用户自定义 `isfull` 全选按钮的选中状态
5. 允许用户通过 自定义事件 的形式，监听全选按钮 选中状态的变化，并获取到 最新的选中状态

- 使用示例：

```
<!-- Footer 组件 -->
<my-footer :isfull="false" :total="1" :amount="98"
@fullChange="onFullStateChange"></my-footer>
```

4.2.1 渲染组件的基础布局

1. 将 `EsFooter.vue` 组件在页面底部进行固定定位：

```vue
<template>
  <div class="footer-container">EsFooter 组件</div>
</template>

<script>
export default {
  name: 'EsFooter',
}
</script>

<style lang="less" scoped>
.footer-container {
  // 设置宽度和高度
  height: 50px;
  width: 100%;
  // 设置背景颜色和顶边框颜色
  background-color: white;
  border-top: 1px solid #efefef;
  // 底部固定定位
  position: fixed;
  bottom: 0;
  left: 0;
  // 内部元素的对齐方式
  display: flex;
  justify-content: space-between;
  align-items: center;
  // 设置左右 padding
  padding: 0 10px;
}
</style>
```

2. 根据 bootstrap 提供的 Checkboxes https://v4.bootcss.com/docs/components/forms/#checkboxes 渲染左侧的 全选 按钮：

```
<template>
  <div class="footer-container">
    <!-- 全选按钮 -->
    <div class="custom-control custom-checkbox">
      <input type="checkbox" class="custom-control-input" id="fullCheck" />
      <label class="custom-control-label" for="fullCheck">全选</label>
    </div>
  </div>
</template>
```

并在全局样式表 `index.css` 中覆盖 全选 按钮的圆角样式：

```
.custom-checkbox .custom-control-label::before {
  border-radius: 10px;
}
```

3. 渲染**合计**对应的价格区域：

```
<template>
  <div class="footer-container">
    <!-- 全选按钮 -->
    <div class="custom-control custom-checkbox">
      <input type="checkbox" class="custom-control-input" id="fullCheck" />
      <label class="custom-control-label" for="fullCheck">全选</label>
    </div>

    <!-- 合计 -->
    <div>
      <span>合计：</span>
      <span class="amount">￥0.00</span>
    </div>
  </div>
</template>
```

并在当前组件的 `<style>` 节点中美化总价格的样式：

```css
.amount {
  color: red;
  font-weight: bold;
}
```

4. 根据 bootstrap 提供的 Buttons https://v4.bootcss.com/docs/components/buttons/#examples 渲染**结算按钮**：

```vue
<template>
  <div class="footer-container">
    <!-- 全选按钮 -->
    <div class="custom-control custom-checkbox">
      <input type="checkbox" class="custom-control-input" id="fullCheck" />
      <label class="custom-control-label" for="fullCheck">全选</label>
    </div>

    <!-- 合计 -->
    <div>
      <span>合计：</span>
      <span class="amount">￥0.00</span>
    </div>

    <!-- 结算按钮 -->
    <button type="button" class="btn btn-primary">结算(0)</button>
  </div>
</template>
```

并在当前组件的 `<style>` 节点中美化结算按钮的样式：

```css
.btn-primary {
  // 设置固定高度
  height: 38px;
  // 设置圆角效果
  border-radius: 19px;
  // 设置最小宽度
  min-width: 90px;
}
```

4.2.2 封装自定义属性 amount

> amount 是已勾选商品的总价格

1. 在 EsFooter.vue 组件的 props 节点中，声明如下的自定义属性：

```
export default {
  name: 'EsFooter',
  props: {
    // 已勾选商品的总价格
    amount: {
      type: Number,
      default: 0,
    },
  },
}
```

2. 在 EsFooter.vue 组件的 DOM 结构中渲染 amount 的值：

```
<!-- 合计 -->
<div>
  <span>合计：</span>
  <!-- 将 amount 的值保留两位小数 -->
  <span class="amount">￥{{ amount.toFixed(2) }}</span>
</div>
```

4.2.3 封装自定义属性 total

> total 为已勾选商品的总数量

1. 在 EsFooter.vue 组件的 props 节点中，声明如下的自定义属性：

```
export default {
  name: 'EsFooter',
  props: {
    // 已勾选商品的总价格
    amount: {
      type: Number,
      default: 0,
    },
    // 已勾选商品的总数量
    total: {
```

```
        type: Number,
        default: 0,
      },
    },
  }
```

2. 在 `EsFooter.vue` 组件的 DOM 结构中渲染 `total` 的值：

```
<!-- 结算按钮 -->
<button type="button" class="btn btn-primary">结算({{total}})
</button>
```

3. 动态控制**结算按钮**的禁用状态：

```
<!-- disabled 的值为 true，表示禁用按钮 -->
<button type="button" class="btn btn-primary" :disabled="total ===
0">结算({{ total }})</button>
```

4.2.4 封装自定义属性 isfull

> isfull 是全选按钮的选中状态，true 表示选中，false 表示未选中

1. 在 `EsFooter.vue` 组件的 props 节点中，声明如下的自定义属性：

```
export default {
  name: 'EsFooter',
  props: {
    // 已勾选商品的总价格
    amount: {
      type: Number,
      default: 0,
    },
    // 已勾选商品的总数量
    total: {
      type: Number,
      default: 0,
    },
    // 全选按钮的选中状态
    isfull: {
      type: Boolean,
      default: false,
    },
```

```
    },
  }
```

2. 为复选框动态绑定 `ckecked` 属性的值：

```
<!-- 全选按钮 -->
<div class="custom-control custom-checkbox">
  <input type="checkbox" class="custom-control-input" id="fullCheck" :checked="isfull" />
  <label class="custom-control-label" for="fullCheck">全选</label>
</div>
```

4.2.5 封装自定义事件 fullChange

> 通过自定义事件 `fullChange`，把最新的选中状态传递给组件的使用者

1. 监听复选框选中状态变化的 `change` 事件：

```
<!-- 全选按钮 -->
<div class="custom-control custom-checkbox">
  <input type="checkbox" class="custom-control-input" id="fullCheck" :checked="isfull" @change="onCheckBoxChange" />
  <label class="custom-control-label" for="fullCheck">全选</label>
</div>
```

2. 在 `methods` 中声明 `onCheckBoxChange`，并通过事件对象 `e` 获取到最新的选中状态：

```
methods: {
  // 监听复选框选中状态的变化
  onCheckBoxChange(e) {
    // e.target.checked 是复选框最新的选中状态
    console.log(e.target.checked)
  },
},
```

3. 在 `emits` 中声明自定义事件：

```
// 声明自定义事件
emits: ['fullChange'],
```

4. 在 `onCheckBoxChange` 事件处理函数中，通过 `$emit()` 触发自定义事件，把最新的选中状态传递给当前组件的使用者：

```
methods: {
  onCheckBoxChange(e) {
    // 触发自定义事件
    this.$emit('fullChange', e.target.checked)
  },
},
```

5. 在 `App.vue` 根组件中测试 `EsFooter.vue` 组件：

```
<!-- 使用 footer 组件 -->
<es-footer :total="0" :amount="0" @fullChange="onFullStateChange">
</es-footer>
```

并在 `methods` 中声明 `onFullStateChange` 处理函数，通过形参获取到**全选按钮**最新的选中状态：

```
methods: {
  // 监听全选按钮状态的变化
  onFullStateChange(isFull) {
    // 打印全选按钮最新的选中状态
    console.log(isFull)
  },
},
```


5. 封装 es-goods 组件

5.1 创建并注册 EsGoods 组件

1. 在 `src/components/es-goods/` 目录下新建 `EsGoods.vue` 组件：

```vue
<template>
  <div>EsGoods 组件</div>
</template>

<script>
export default {
  name: 'EsGoods',
}
</script>

<style lang="less" scoped></style>
```

2. 在 `App.vue` 组件中导入并注册 `EsGoods.vue` 组件：

```vue
// 导入 header 组件
import EsHeader from './components/es-header/EsHeader.vue'
// 导入 footer 组件
import EsFooter from './components/es-footer/EsFooter.vue'
// 导入 goods 组件
import EsGoods from './components/es-goods/EsGoods.vue'

export default {
  name: 'MyApp',
  components: {
    // 注册 header 组件
    EsHeader,
    // 注册 footer 组件
    EsFooter,
    // 注册 goods 组件
    EsGoods,
  },
}
```

3. 在 `App.vue` 的 `template` 模板结构中使用 `EsGoods` 组件：

```vue
<template>
  <div class="app-container">
    <!-- 使用 header 组件 -->
    <es-header title="购物车案例"></es-header>
    <!-- 使用 goods 组件 -->
    <es-goods></es-goods>
    <!-- 使用 footer 组件 -->
    <es-footer :total="0" :amount="0"
@fullChange="onFullStateChange"></es-footer>
  </div>
</template>
```

5.2 封装 es-goods 组件

5.2.0 封装需求

1. 实现 `EsGoods` 组件的基础布局
2. 封装组件的 6 个自定义属性（id, thumb, title, price, count, checked）
3. 封装组件的自定义事件 `stateChange`，允许外界监听组件选中状态的变化

- 使用示例：

```vue
<!-- 使用 goods 组件 -->
<es-goods
  v-for="item in goodslist"
  :key="item.id"

  :id="item.id"
  :thumb="item.goods_img"
  :title="item.goods_name"
  :price="item.goods_price"
  :count="item.goods_count"
  :checked="item.goods_state"

  @stateChange="onGoodsStateChange"
></es-goods>
```

5.2.1 渲染组件的基础布局

1. 渲染 `EsGoods` 组件的基础 DOM 结构：

```vue
<template>
  <div class="goods-container">
    <!-- 左侧图片区域 -->
    <div class="left">
      <!-- 商品的缩略图 -->
      <img src="" alt="商品图片" class="thumb" />
    </div>

    <!-- 右侧信息区域 -->
    <div class="right">
      <!-- 商品名称 -->
      <div class="top">xxxx</div>
      <div class="bottom">
        <!-- 商品价格 -->
        <div class="price">￥0.00</div>
        <!-- 商品数量 -->
        <div class="count">数量</div>
      </div>
    </div>
  </div>
</template>
```

2. 美化组件的布局样式：

```less
.goods-container {
  display: flex;
  padding: 10px;
  // 左侧图片的样式
  .left {
    margin-right: 10px;
    // 商品图片
    .thumb {
      display: block;
      width: 100px;
      height: 100px;
      background-color: #efefef;
    }
  }
  // 右侧商品名称、单价、数量的样式
  .right {
```

```
      display: flex;
      flex-direction: column;
      justify-content: space-between;
      flex: 1;
      .top {
        font-weight: bold;
      }
      .bottom {
        display: flex;
        justify-content: space-between;
        align-items: center;
        .price {
          color: red;
          font-weight: bold;
        }
      }
    }
  }
```

3. 在商品缩略图之外包裹**复选框**（https://v4.bootcss.com/docs/components/forms/#checkboxes）效果：

```
<!-- 左侧图片和复选框区域 -->
<div class="left">
  <!-- 复选框 -->
  <div class="custom-control custom-checkbox">
    <input type="checkbox" class="custom-control-input" id="customCheck1" />
    <!-- 将商品图片包裹于 label 之中，点击图片可以切换“复选框”的选中状态 -->
    <label class="custom-control-label" for="customCheck1">
      <img src="" alt="商品图片" class="thumb" />
    </label>
  </div>
  <!-- <img src="" alt="商品图片" class="thumb" /> -->
</div>
```

4. 覆盖**复选框**的默认样式：

```
.custom-control-label::before,
.custom-control-label::after {
  top: 3.4rem;
}
```

5. 在 `App.vue` 组件中循环渲染 `EsGoods.vue` 组件：

```vue
<!-- 使用 goods 组件 -->
<es-goods v-for="item in goodslist" :key="item.id"></es-goods>
```

6. 为 `EsGoods.vue` 添加顶边框：

```less
.goods-container {
  display: flex;
  padding: 10px;
  // 最终生成的选择器为 .goods-container + .goods-container
  // 在 css 中，(+) 是“相邻兄弟选择器”，表示：选择紧连着另一元素后的元素，二者具有相同的父元素。
  + .goods-container {
    border-top: 1px solid #efefef;
  }
  // ...省略其他样式
}
```

5.2.2 封装自定义属性 id

> id 是每件商品的唯一标识符

1. 在 `EsGoods.vue` 组件的 `props` 节点中，声明如下的自定义属性：

```vue
export default {
  name: 'EsGoods',
  props: {
    // 唯一的 key 值
    id: {
      type: [String, Number], // id 的值可以是“字符串”也可以是“数值”
      required: true,
    },
  }
}
```

2. 在渲染复选框时动态绑定 `input` 的 `id` 属性和 `label` 的 `for` 属性值：

```
<!-- 复选框 -->
<div class="custom-control custom-checkbox">
  <input type="checkbox" class="custom-control-input" :id="id" />
  <label class="custom-control-label" :for="id">
    <img src="" alt="商品图片" class="thumb" />
  </label>
</div>
```

3. 在 `App.vue` 中使用 `EsGoods.vue` 组件时，动态绑定 `id` 属性的值：

```
<!-- 使用 goods 组件 -->
<es-goods v-for="item in goodslist" :id="item.id"></es-goods>
```

5.2.3 封装其它属性

> 除了 id 属性之外，EsGoods 组件还需要封装：
>
> **缩略图**（thumb）、**商品名称**（title）、**单价**（price）、**数量**（count）、**勾选状态**（checked）这 5 个属性

1. 在 `EsGoods.vue` 组件的 props 节点中，声明如下的自定义属性：

```
export default {
  name: 'EsGoods',
  props: {
    // 唯一的 key 值
    id: {
      type: [String, Number],
      required: true,
    },
    // 1. 商品的缩略图
    thumb: {
      type: String,
      required: true,
    },
    // 2. 商品的名称
    title: {
      type: String,
      required: true,
    },
    // 3. 单价
    price: {
```

```
      type: Number,
      required: true,
    },
    // 4. 数量
    count: {
      type: Number,
      required: true,
    },
    // 5. 商品的勾选状态
    checked: {
      type: Boolean,
      required: true,
    },
  },
}
```

2. 在 `EsGoods.vue` 组件的 DOM 结构中渲染商品的信息数据：

```
<template>
  <div class="goods-container">
    <!-- 左侧图片和复选框区域 -->
    <div class="left">
      <!-- 复选框 -->
      <div class="custom-control custom-checkbox">
        <input type="checkbox" class="custom-control-input" :id="id" :checked="checked" />
        <label class="custom-control-label" :for="id">
          <img :src="thumb" alt="商品图片" class="thumb" />
        </label>
      </div>
    </div>

    <!-- 右侧信息区域 -->
    <div class="right">
      <!-- 商品名称 -->
      <div class="top">{{ title }}</div>
      <div class="bottom">
        <!-- 商品价格 -->
        <div class="price">¥{{ price.toFixed(2) }}</div>
        <!-- 商品数量 -->
        <div class="count">数量: {{ count }}</div>
      </div>
    </div>
  </div>
```

```
</template>
```

3. 在 `App.vue` 组件中使用 `EsGoods.vue` 组件时，动态绑定对应属性的值：

```
<!-- 使用 goods 组件 -->
<es-goods
  v-for="item in goodslist"
  :key="item.id"
  :id="item.id"
  :thumb="item.goods_img"
  :title="item.goods_name"
  :price="item.goods_price"
  :count="item.goods_count"
  :checked="item.goods_state"
></es-goods>
```

5.2.4 封装自定义事件 stateChange

> **点击复选框**时，可以把**最新的勾选状态**，通过**自定义事件**的方式传递给组件的使用者。

1. 在 `EsGoods.vue` 组件中，监听 `checkbox` 选中状态变化的事件：

```
<!-- 监听复选框的 change 事件 -->
<input type="checkbox" class="custom-control-input" :id="id"
:checked="checked" @change="onCheckBoxChange" />
```

2. 在 `EsGoods.vue` 组件的 `methods` 中声明对应的事件处理函数：

```
methods: {
  // 监听复选框选中状态变化的事件
  onCheckBoxChange(e) {
    // e.target.checked 是最新的勾选状态
    console.log(e.target.checked)
  },
},
```

3. 在 `EsGoods.vue` 组件中声明自定义事件：

```
emits: ['stateChange'],
```

4. 完善 `onCheckBoxChange` 函数的处理逻辑，调用 `$emit()` 函数触发自定义事件：

```
methods: {
  // 监听复选框选中状态变化的事件
  onCheckBoxChange(e) {
   // 向外发送的数据是一个对象，包含了 { id, value } 两个属性
    this.$emit('stateChange', {
      id: this.id,
      value: e.target.checked,
    })
  },
},
```

5. 在 `App.vue` 根组件中使用 `EsGoods.vue` 组件时，监听它的 `stateChange` 事件：

```
<!-- 使用 goods 组件 -->
<es-goods
  v-for="item in goodslist"
  :key="item.id"
  :id="item.id"
  :thumb="item.goods_img"
  :title="item.goods_name"
  :price="item.goods_price"
  :count="item.goods_count"
  :checked="item.goods_state"
  @stateChange="onGoodsStateChange"
></es-goods>
```

并在 `App.vue` 的 `methods` 中声明如下的事件处理函数：

```
methods: {
  // 监听商品选中状态变化的事件
  onGoodsStateChange(e) {
    // 1. 根据 id 进行查找（注意：e 是一个对象，包含了 id 和 value 两个属性）
    const findResult = this.goodslist.find(x => x.id === e.id)
    // 2. 找到了对应的商品，则更新其选中状态
    if (findResult) {
      findResult.goods_state = e.value
    }
  },
}
```

6. 实现合计、结算数量、全选功能

6.1 动态统计已勾选商品的总价格

> 需求分析：
>
> 合计的商品总价格，依赖于 goodslist 数组中每一件商品信息的变化，此场景下适合使用**计算属性**。

1. 在 App.vue 中声明如下的计算属性：

```
computed: {
  // 已勾选商品的总价
  amount() {
    // 1. 定义商品总价格
    let a = 0
    // 2. 循环累加商品总价格
    this.goodslist
      .filter(x => x.goods_state)
      .forEach(x => {
        a += x.goods_price * x.goods_count
      })
    // 3. 返回累加的结果
    return a
  },
},
```

2. 在 App.vue 中使用 EsFooter.vue 组件时，动态绑定**已勾选商品的总价格**：

```
<!-- 使用 footer 组件 -->
<es-footer :total="0" :amount="amount"
@fullChange="onFullStateChange"></es-footer>
```

6.2 动态统计已勾选商品的总数量

> 需求分析：
>
> 已勾选商品的总数量依赖项 goodslist 中商品勾选状态的变化，此场景下适合使用计算属性。

1. 在 App.vue 中声明如下的计算属性：

```
computed: {
  // 已勾选商品的总数量
  total() {
    // 1. 定义已勾选的商品总数量
    let t = 0
    // 2. 循环累加
    this.goodslist
      .filter(x => x.goods_state)
      .forEach(x => (t += x.goods_count))
    // 3. 返回计算的结果
    return t
  },
},
```

2. 在 `App.vue` 中使用 `EsFooter.vue` 组件时，动态绑定**已勾选商品的总数量**：

```
<!-- 使用 footer 组件 -->
<es-footer :total="total" :amount="amount"
@fullChange="onFullStateChange"></es-footer>
```

6.3 实现全选功能

1. 在 `App.vue` 组件中监听到 `EsFooter.vue` 组件的选中状态发生变化时，立即更新 `goodslist` 中每件商品的选中状态即可：

```
<!-- 使用 footer 组件 -->
<es-footer :total="total" :amount="amount"
@fullChange="onFullStateChange"></es-footer>
```

2. 在 `onFullStateChange` 的事件处理函数中修改每件商品的选中状态：

```
methods: {
  // 监听全选按钮状态的变化
  onFullStateChange(isFull) {
    this.goodslist.forEach(x => x.goods_state = isFull)
  },
}
```

7. 封装 es-counter 组件

7.1 创建并注册 EsCounter 组件

1. 在 `src/components/es-counter/` 目录下新建 `EsCounter.vue` 组件：

```vue
<template>
  <div>EsCounter 组件</div>
</template>

<script>
export default {
  name: 'EsCounter',
}
</script>

<style lang="less" scoped></style>
```

2. 在 `EsGoods.vue` 组件中导入并注册 `EsCounter.vue` 组件：

```vue
// 导入 counter 组件
import EsCounter from '../es-counter/EsCounter.vue'

export default {
  name: 'EsGoods',
  components: {
    // 注册 counter 组件
    EsCounter,
  }
}
```

3. 在 `EsGoods.vue` 的 `template` 模板结构中使用 `EsCounter.vue` 组件：

```
<div class="bottom">
  <!-- 商品价格 -->
  <div class="price">￥{{ price.toFixed(2) }}</div>
  <!-- 商品数量 -->
  <div class="count">
    <!-- 使用 es-counter 组件 -->
    <es-counter></es-counter>
  </div>
</div>
```

7.2 封装 es-counter 组件

7.2.0 封装需求

1. 渲染组件的 基础布局
2. 实现数量值的 加减操作
3. 处理 `min` 最小值
4. 使用 `watch` 侦听器处理文本框输入的结果
5. 封装 `numChange` 自定义事件

- 代码示例：

```
<es-counter :num="count" :min="1" @numChange="getNumber"></es-counter>
```

7.2.1 渲染组件的基础布局

1. 基于 bootstrap 提供的 Buttons https://v4.bootcss.com/docs/components/buttons/#examples 和 `form-control` 渲染组件的基础布局：

```
<template>
  <div class="counter-container">
    <!-- 数量 -1 按钮 -->
    <button type="button" class="btn btn-light btn-sm">-</button>
    <!-- 输入框 -->
    <input type="number" class="form-control form-control-sm ipt-num" />
    <!-- 数量 +1 按钮 -->
    <button type="button" class="btn btn-light btn-sm">+</button>
  </div>
</template>
```

2. 美化当前组件的样式：

```
.counter-container {
  display: flex;
  // 按钮的样式
  .btn {
    width: 25px;
  }
  // 输入框的样式
  .ipt-num {
    width: 34px;
    text-align: center;
    margin: 0 4px;
  }
}
```

7.2.2 实现数值的渲染及加减操作

> 思路分析：
>
> 1. 加减操作需要依赖于 EsCounter 组件的 data 数据
> 2. 初始数据依赖于父组件通过 props 传递进来
>
> 将父组件传递进来的 props 初始值转存到 data 中，形成 EsCounter 组件的内部状态！

1. 在 `EsCounter.vue` 组件中声明如下的 props：

```
props: {
  // 数量值
  num: {
    type: Number,
    default: 0,
  },
},
```

2. 在 `EsGoods.vue` 组件中通过属性绑定的形式，将数据传递到 `EsCounter.vue` 组件中：

```
<!-- 商品数量 -->
<div class="count">
  <es-counter :num="count"></es-counter>
</div>
```

> 注意：不要直接把 `num` 通过 `v-model` 指令双向绑定到 `input` 输入框，因为 vue 规定：props **的值只读的！**例如下面的做法是错误的：

```
<!-- Warning 警告：不要模仿下面的操作 -->
<input type="number" class="form-control form-control-sm ipt-num" v-model.number="num" />
```

3. 正确的做法：将 props 的初始值**转存**到 data 中，因为 data **中的数据是可读可写的！**示例代码如下：

```
export default {
  name: 'EsCounter',
  props: {
    // 初始数量值【只读数据】
    num: {
      type: Number,
      default: 0,
    },
  },
  data() {
    return {
      // 内部状态值【可读可写的数据】
      // 通过 this 可以访问到 props 中的初始值
      number: this.num,
    }
  },
}
```


并且把 data 中的 `number` 双向绑定到 `input` 输入框：

```
<input type="number" class="form-control form-control-sm ipt-num" v-model.number="number" />
```

4. 为 `-1` 和 `+1` 按钮绑定响应的点击事件处理函数：

```
<button type="button" class="btn btn-light btn-sm" @click="onSubClick">-</button>
<input type="number" class="form-control form-control-sm ipt-num" v-model.number="number" />
<button type="button" class="btn btn-light btn-sm" @click="onAddClick">+</button>
```

并在 `methods` 中声明对应的事件处理函数如下：

```
methods: {
  // -1 按钮的事件处理函数
  onSubClick() {
    this.number -= 1
  },
  // +1 按钮的事件处理函数
  onAddClick() {
    this.number += 1
  },
},
```

7.2.3 实现 min 最小值的处理

> 需求分析：
>
> 购买商品时，购买的数量最小值为 1

1. 在 `EsCounter.vue` 组件中封装如下的 `props`：

```
export default {
  name: 'EsCounter',
  props: {
    // 数量值
    num: {
      type: Number,
      default: 0,
    },
    // 最小值
    min: {
      type: Number,
      // min 属性的值默认为 NaN，表示不限制最小值
      default: NaN,
    },
  },
}
```

2. 在 `-1` 按钮的事件处理函数中，对 `min` 的值进行判断和处理：

```
methods: {
  // -1 按钮的事件处理函数
  onSubClick() {
    // 判断条件：min 的值存在，且 number - 1 之后小于 min
    if (!isNaN(this.min) && this.number - 1 < this.min) return
    this.number -= 1
  },
}
```

3. 在 `EsGoods.vue` 组件中使用 `EsCounter.vue` 组件时指定 `min` 最小值：

```
<!-- 商品数量 -->
<div class="count">
  <!-- 指定数量的最小值为 1 -->
  <es-counter :num="count" :min="1"></es-counter>
</div>
```

7.2.4 处理输入框的输入结果

> 思路分析：
>
> 1. 将输入的新值转化为整数
> 2. 如果转换的结果不是数字，或小于1，则强制 number 的值等于1
> 3. 如果新值为小数，则把转换的结果赋值给 number

1. 为输入框的 `v-model` 指令添加 `.lazy` 修饰符（当输入框触发 change 事件时更新 v-model 所绑定到的数据源）：

```
<input type="number" class="form-control form-control-sm ipt-num" v-model.number.lazy="number" />
```

2. 通过 `watch` 侦听器监听 `number` 数值的变化，并按照分析的步骤实现代码：

```
export default {
  name: 'EsCounter',
  watch: {
    // 监听 number 数值的变化
    number(newVal) {
      // 1. 将输入的新值转化为整数
      const parseResult = parseInt(newVal)
      // 2. 如果转换的结果不是数字，或小于1，则强制 number 的值等于1
```

```
      if (isNaN(parseResult) || parseResult < 1) {
        this.number = 1
        return
      }
      // 3. 如果新值为小数，则把转换的结果赋值给 number
      if (String(newVal).indexOf('.') !== -1) {
        this.number = parseResult
        return
      }
      console.log(this.number)
    },
  },
}
```

7.2.5 把最新的数据传递给使用者

> 需求分析：
>
> 当 EsGoods 组件使用 EsCounter 组件时，期望能够监听到**商品数量**的变化，此时需要使用**自定义事件**的方式，把最新的数据**传递给组件的使用者**。

1. 在 `EsCounter.vue` 组件中声明自定义事件如下：

```
emits: ['numChange'],
```

2. 在 `EsCounter.vue` 组件的 `watch` 侦听器中触发自定义事件：

```
watch: {
  number(newVal) {
    // 1. 将输入的新值转化为整数
    const parseResult = parseInt(newVal)
    // 2. 如果转换的结果不是数字，或小于1，则强制 number 的值等于1
    if (isNaN(parseResult) || parseResult < 1) {
      this.number = 1
      return
    }
    // 3. 如果新值为小数，则把转换的结果赋值给 number
    if (String(newVal).indexOf('.') !== -1) {
      this.number = parseResult
      return
    }
    // 触发自定义事件，把最新的 number 数值传递给组件的使用者
    this.$emit('numChange', this.number)
```

```
    },
  },
```

3. 在 `EsGoods.vue` 组件中监听 `EsCounter.vue` 组件的自定义事件：

```
<!-- 商品数量 -->
<div class="count">
  <es-counter :num="count" :min="1" @numChange="getNumber"></es-counter>
</div>
```

并声明对应的事件处理函数如下：

```
methods: {
  // 监听数量变化的事件
  getNumber(num) {
    console.log(num)
  },
}
```

7.2.6 更新购物车中商品的数量

> 思路分析：
>
> 1. 在 EsGoods 组件中**获取到最新的商品数量**
> 2. 在 EsGoods 组件中**声明**自定义事件
> 3. 在 EsGoods 组件中**触发**自定义事件，向外传递数据对象 { id, value }
> 4. 在 App 根组件中监听 EsGoods 组件的自定义事件，并根据 id 更新对应商品的数量

1. 在 `EsGoods.vue` 组件中声明自定义事件 `countChange`：

```
emits: ['stateChange', 'countChange'],
```

2. 在 `EsCounter.vue` 组件的 `numChange` 事件处理函数中，触发**步骤1**声明的自定义事件：

```
<es-counter :num="count" :min="1" @numChange="getNumber"></es-counter>

methods: {
  // 监听数量变化的事件
  getNumber(num) {
    // 触发自定义事件，向外传递数据对象 { id, value }
    this.$emit('countChange', {
```

```
        // 商品的 id
        id: this.id,
        // 最新的数量
        value: num,
      })
    },
}
```

3. 在 `App.vue` 根组件中使用 `EsGoods.vue` 组件时，监听它的自定义事件 `countChange`：

```
<!-- 使用 goods 组件 -->
<es-goods
  v-for="item in goodslist"
  :key="item.id"
  :id="item.id"
  :thumb="item.goods_img"
  :title="item.goods_name"
  :price="item.goods_price"
  :count="item.goods_count"
  :checked="item.goods_state"
  @stateChange="onGoodsStateChange"
  @countChange="onGoodsCountChange"
></es-goods>
```

并在 `methods` 中声明对应的事件处理函数：

```
methods: {
  // 监听商品数量变化的事件
  onGoodsCountChange(e) {
    // 根据 id 进行查找
    const findResult = this.goodslist.find(x => x.id === e.id)
    // 找到了对应的商品，则更新其数量
    if (findResult) {
      findResult.goods_count = e.value
    }
  }
}
```


黑马程序员
www.itheima.com