第四章：TypeScript在前端框架中的应用

学习目标

- 了解 TypeScript 在 Vue.js 中的应用
- 了解 TypeScript 在 React 中的应用
- 了解 TypeScript 在 Angular 中的应用

一、在 Vue 中实现商品列表功能

1.数据模型与业务逻辑

实例演示

- 数据模型定义：商品属性字段
- 业务逻辑定义：商品筛选条件与排序

2.Vue 简介

- Vue 是一套用于构建用户界面的渐进式框架。
- 与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。
- Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。
- 与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

3.页面布局与组件选择

实例演示

- UI 框架： https://www.iviewui.com
- 页面布局设计
- 页面组件确定

创建项目并安装依赖

```
vue create vue_pro
cd vue_pro
npm install axios view-design --save
```

vue页面模板代码片段（ vue_pro/src/views/Item.vue ）

```
<template>
  <div class="itemlist">
    <h1 style="padding-top:50px;">商品信息列表</h1>
    <Row type="flex" justify="start" style="padding-top:20px;">
        <Col span="8"></Col>
        <Col span="8">
          <Input search v-model="keyword" @on-search = "search" placeholder="请输
入商品关键字匹配..." />
        </Col>
        <Col span="4">
          <Select v-model="orderVal" style="width:150px"
```

```
            clearable
            @on-change="search"
            @on-clear="search"
          >
              <Option v-for="item in optionList" :value="item.value"
:key="item.value" style="padding-left:10px;">{{ item.label }}</Option>
          </Select>
        </Col>
        <Col span="4"></Col>
    </Row>
    <div class="mainArea">
      <ul>
        <li v-for="(item,i) in itemList" :key="i"
style="width:350px;padding:5px;">
          <Card style="width:345px;">
              <div style="text-align:center">
                  <img :src="host + item.pic">
                  <div>
                    <span style="font-size: 110%;"> {{ item.name }} </span>
                    <div>
                      <span style="color:red;">
{{(item.unit_price/100).toFixed(2) + '元/' + item.unit}}</span>
                      <Button type="text" style="color:#2d8cf0;"
@click="addToCard(i)"> 加入购物车 </Button>
                    </div>
                  </div>
              </div>
          </Card>
        </li>
      </ul>
    </div>
  </div>
</template>
```

重点：

- Row/Col组件布局
- Select 与 Card 组件的应用与事件处理

4.事件处理

实例演示

- 事件定义与触发
- 事件处理

vue页面事件处理代码片段（ vue_pro/src/views/Item.vue ）

```
<script lang="ts">
import Vue from 'vue';
import Component from 'vue-class-component';
import API from '../lib/api';
@Component
export default class App extends Vue{
  itemList = [];
  host = API.HOST;
  keyword = "";
  orderVal = "";
```

```
  optionList = [
    {
      label: "价格由低到高",
      value: 1
    },
    {
      label: "价格由高到低",
      value: -1
    }
  ];
  async created(){
    const result = await API.getItems(this.keyword, this.orderVal);
    console.log(result);
    this.itemList = result.data.data;
  }
  async search(){
    console.log(this.keyword);
    console.log(this.orderVal);
    const result = await API.getItems(this.keyword, this.orderVal);
    console.log(result);
    this.itemList = result.data.data;
  }
  addToCard(){
    this.$Message.success({
      content: '已加入您的购物车'
    });
  }
}
</script>
```

重点：

- Component 装饰器的使用方法
- 自定义类App变量与事件处理函数的定义与触发
- API的调用

5.连接 API

实例演示

- API 请求实现
- 应用 API 请求数据

API访问封装代码（ vue_pro/src/lib/api.ts ）

```
// 调用 API 服务器接口
import axios from 'axios';
axios.defaults.headers = {
  'X-Requested-With': 'XMLHttpRequest',
  'Content-Type': 'application/json'
};

axios.defaults.baseURL = "http://localhost:3000/api/";
// 请求超时的时间限制
axios.defaults.timeout = 2000;
```

```
export default class {
  static HOST = "http://localhost:3000/";
  static async getItems(name?: string, order = '-1') {
    const q = [];
    if (name) {
      q.push("name=" + name);
    }
    q.push("order=" + order);
    return axios.get('/items/list?' + q.join('&'));
  }
  static async register(username: string, password: string) {
    return axios.post('/users/register', { username: username, password:
password });
  }
  static async login(username: string, password: string) {
    return axios.post('/users/login', { username: username, password: password
});
  }
  static async createOrder(order: any) {
    return axios.post('/orders/create', order);
  }
  static async getOrders(userId?: string) {
    return axios.get('/orders/list?user_id=' + (userId || ''));
  }
  static async getOrderById(id: string) {
    return axios.get('/orders/' + id);
  }
  static async changeOrderStatus(id: string, status: string) {
    return axios.post(`/orders/${id}/change_status`, { status: status });
  }
}
```

二、在 React 中实现登录功能

1.数据模型与业务逻辑

实例演示

- 数据模型定义：用户属性字段
- 业务逻辑定义：登录

2.React 简介

React 起源于 Facebook 的内部项目，是一个用于构建用户界面的 JAVASCRIPT 库，主要用于构建 UI。

其特点如下：

- 声明式设计 −React 采用声明范式，可以轻松描述应用。
- 高效 −React 通过对 DOM 的模拟，最大限度地减少与 DOM 的交互。
- 灵活 −React可以与已知的库或框架很好地配合。
- JSX − JSX 是 JavaScript 语法的扩展。React 开发不一定使用 JSX ，但我们建议使用它。
- 组件 − 通过 React 构建组件，使得代码更加容易得到复用，能够很好的应用在大项目的开发中。
- 单向响应的数据流 − React 实现了单向响应的数据流，从而减少了重复代码。

3.页面布局与组件选择

实例演示

- UI 框架： https://material-ui.com/zh/
- 页面布局设计
- 页面组件确定

创建项目并启动

```
npm init next-app  #进入交互模型
cd react_pro
npm run dev
```

JSX render 方法代码片段（ react_pro/pages/index.tsx ）

```
return (<Container maxWidth="sm">
      <CssBaseLine />
      <div style={{ marginTop: '100px', display: 'flex', flexDirection:
'column', alignItems: 'center' }}>
        <Avatar style={{ margin: '8px', backgroundColor: 'rgb(220, 0, 78)',
width: '50px', height: '50px' }}>
          <LockOutlinedIcon />
        </Avatar>
        <Typography variant="h5" style={{ margin: '12px', fontWeight: 500 }}>
          登 录
        </Typography>
        <form style={{ width: '100%', marginTop: '10px' }}>
          <TextField
            variant="outlined"
            margin="normal"
            required
            fullWidth
            id="username"
            label="登录用户名"
            name="username"
            onChange={this.usernameChange}
          />
          <TextField
            variant="outlined"
            margin="normal"
            required
            fullWidth
            type="password"
            id="password"
            label="登录密码"
            name="password"
            onChange={this.passwordChange}
          />
          <FormControlLabel
            control={<Checkbox value="remember" color="primary" />}
            style={{ padding: '10px' }}
            label="记住我"
          />
          <Button
            type="button"
            fullWidth
            color="primary"
```

```
            variant="contained"
            style={{ margin: '0px', height: "50px", fontSize: '20px' }}
            onClick={this.submit}
          >
            登 录
          </Button>
          <Grid container style={{ paddingTop: '20px' }}>
            <Grid item xs>
              <Link href="#" > {'忘记密码？'} </Link>
            </Grid>
            <Grid item xs style={{ textAlign: 'right' }}>
              <Link href="#" > {'还没有账号？注册'} </Link>
            </Grid>
          </Grid>
        </form>
      </div>
      <Box style={{ marginTop: '100px' }}>
        <Copyright />
      </Box>

      <SnackBar
        anchorOrigin={{
          vertical: 'top',
          horizontal: 'center'
        }}
        open={open}
        autoHideDuration={3000}
        onClose={this.handleClose}
      >
        <Alert onClose={this.handleClose} severity={msgType} variant="filled">
          {msg}
        </Alert>
      </SnackBar>
    </Container>)
```

`**重点：**`

- 模板中属性与事件的绑定方法
- 自定义组件的嵌套使用

4.事件处理

实例演示

- 事件定义与触发
- 事件处理

JSX 事件处理代码片段（ react_pro/pages/index.tsx ）

```
// 类组件
class SignIn extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      username: '',
      password: '',
      open: false,
```

```
      msg: '',
      msgType: 'success'
    }
    // 将事件处理函数的上下文重新绑定到父组件的this
    this.usernameChange = this.usernameChange.bind(this);
    this.passwordChange = this.passwordChange.bind(this);
    this.submit = this.submit.bind(this);
    this.handleClose = this.handleClose.bind(this);
  }
  usernameChange(e) {
    this.setState({ username: e.target.value });
  }
  passwordChange(e) {
    this.setState({ password: e.target.value });
  }
  handleClose() {
    this.setState({ open: false });
  }
  async submit() {
    console.log(this.state['username'] + ',' + this.state['password']);
    let result = await API.login(this.state['username'],
this.state['password']);
    console.log(result);
    if (result.data.code === 0) {
      console.log("登录成功");
      this.setState({ open: true, msg: '登录成功', msgType: 'success' });
    } else {
      console.log("登录失败");
      this.setState({ open: true, msg: '登录失败', msgType: 'error' });
    }
  }
 ...
}
```

`**重点：**`

- 事件处理函数重新绑定上下文this
- state中变量的初始化与修改方法

5.连接 API

实例演示

- API 请求实现
- 应用 API 请求数据

三、在 Angular 中实现购物车功能

1.Angular 简介

Angular.js是 google 开发者设计和开发的一套前端开发框架，帮助你简化前端开发的负担。
AngularJS 通过新的属性和表达式扩展了 HTML。

其特点如下：

（1）良好的应用程序结构，适合于大规模的应用程序。

（2）双向数据绑定，最大限度地减少与 DOM 的交互。

（3）指令，自定义Directive，比jQuery插件还灵活。

（4）AngularJS 使用 HTML 模板，这使事情变得简单，并允许设计人员和开发人员同时工作。

（5）依赖注入，ng模块化比较大胆的引入了Java的一些东西（依赖注入），能够很容易的写出可复用的代码，对于敏捷开发的团队来说非常有帮助。

2.数据模型与业务逻辑

实例演示

- 数据模型定义：购物车属性字段
- 业务逻辑定义：添加、显示、清除操作

3.页面布局与组件选择

实例演示

- UI 框架： https://material.angular.io/
- 页面布局设计
- 页面组件确定

创建项目、安装依赖的UI组件并启动

```
npm install -g @angular/cli
ng new ng-pro
cd ng-pro
npm install --save @angular/material @angular/cdk @angular/animations
ng serve --port 4201
```

app组件模板文件代码片段（ng-pro/src/app/app.component.html）

```
<!-- Toolbar -->
<div class="toolbar" role="banner">
  <img width="40" alt="Angular Logo"

src="
mciIHZpZXdCb3g9IjAgMCAyNTAgMjUwIj4KICAgIDxwYXRoIGZpbGw9IiNERDAwMzEiIGQ9Ik0xMjUgM
zBMMzEuOSA2My4ybDE0LjIgMTIzLjFMMTI1IDIzMGw3OC45LTQzLjcgMTQuMi0xMjMuMXoiIC8+CiAgI
CA8cGF0aCBmaWxsPSIjQzMwMDJGIiBkPSJNMTI1IDMwdjIyLjItLjFWMjMwbDc4LjktNDMuNyAxNC4yL
TEyMy4xTDEyNSAzMHoiIC8+CiAgICA8cGF0aCAgZmlsbD0iI0ZGRkZGRiIgZD0iTTEyNSA1Mi4xTDY2L
jggMTgyLjZoMjEuN2wxMS43LTI5LjJoNDkuNGwxMS43IDI5LjJIMTgzTDEyNSA1Mi4xem0xNyA4My4za
C0zNGwxNy00MC45IDE3IDQwLjl6IiAvPgogIDwvc3ZnPg==" />
  <span>商品列表</span>
  <div class="spacer"></div>
  <button mat-button style="color:#FFF; font-weight: 700; margin-right: 100px;"
(click)="showCart()">购物车</button>
</div>

<div class="content" role="main">
  <!-- 商品列表 -->
  <ul style="width: 1200px;" *ngIf="showList">
    <li *ngFor="let item of itemList">
      <mat-card class="example-card">
        <mat-card-header>
          <mat-card-title style="font-size:12px;">{{item.name}}</mat-card-title>
```

```
        </mat-card-header>
        <mat-divider></mat-divider>
        <img mat-card-image [src]="host + item.pic" style="width:300px;padding:
0 20px;margin:0;">
        <mat-divider></mat-divider>
        <mat-card-actions style="display: flex;">
          <span style="width:200px; padding-left: 20px; padding-top: 8px;">
            <span style="color:red; font-weight: 600;">
{{(item.unit_price/100).toFixed(2)}}</span> 元 / {{item.unit}}
          </span>
          <button mat-button color="primary" (click)="addToCart(item)">加入购物车
</button>
        </mat-card-actions>
      </mat-card>
    </li>
  </ul>

  <!-- 购物车表格 -->
  <table mat-table [dataSource]="cartItems" class="mat-elevation-z8"
*ngIf="showTable">
    <!-- Name Column -->
    <ng-container matColumnDef="name">
      <th mat-header-cell *matHeaderCellDef> 商品名 </th>
      <td mat-cell *matCellDef="let item"> {{item.name}} </td>
      <td mat-footer-cell *matFooterCellDef> 总计 </td>
    </ng-container>

    <!-- UnitPrice Column -->
    <ng-container matColumnDef="unit_price">
      <th mat-header-cell *matHeaderCellDef> 单价 </th>
      <td mat-cell *matCellDef="let item"> {{(item.unit_price/100).toFixed(2)}}
/ {{item.unit}}
      </td>
      <td mat-footer-cell *matFooterCellDef> </td>
    </ng-container>

    <!-- Quantity Column -->
    <ng-container matColumnDef="quantity">
      <th mat-header-cell *matHeaderCellDef> 数量 </th>
      <td mat-cell *matCellDef="let item"> {{item.quantity}} </td>
      <td mat-footer-cell *matFooterCellDef> {{getTotalQuantity()}} </td>
    </ng-container>

    <!-- Count Column -->
    <ng-container matColumnDef="count">
      <th mat-header-cell *matHeaderCellDef> 小计 </th>
      <td mat-cell *matCellDef="let item"> {{(item.quantity *
item.unit_price/100).toFixed(2)}} 元
      </td>
      <td mat-footer-cell *matFooterCellDef>
{{(getTotalAmount()/100).toFixed(2)}} 元</td>
    </ng-container>

    <!-- Action Column -->
    <ng-container matColumnDef="action">
      <th mat-header-cell *matHeaderCellDef> 操作 </th>
      <td mat-cell *matCellDef="let item">
```

```
        <button mat-button color="primary" (click)="removeCartItem(item._id)">
移除 </button>
      </td>
      <td mat-footer-cell *matFooterCellDef>
        <button mat-button color="primary" (click)="clearCart()"> 清空全部
</button>
      </td>
    </ng-container>

    <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
    <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
    <tr mat-footer-row *matFooterRowDef="displayedColumns"></tr>
  </table>
</div>
```

`**重点：**`

- 注意模板中指令的使用方法
- 属性与事件的绑定的方法

4.事件处理

实例演示

- 事件定义与触发
- 事件处理

app组件类文件代码片段（ng-pro/src/app/app.component.ts)

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'ng-pro';
  host = '';
  itemList = [];
  displayedColumns: string[] = ['name', 'unit_price', 'quantity', 'count',
'action'];
  cartItems = [];
  // 商品列表是否可见
  showList = true;
  // 购物车表格是否可见
  showTable = false;
  constructor(private cart: CartService, private api: ApiService, private
snackBar: MatSnackBar) {
    this.host = api.host;
  }
  // 初始化从API获取商品数据
  async ngOnInit() {
    this.api.getItems().subscribe(result => {
      console.log(result);
      if (0 === result.code) {
        this.itemList = result.data;
      }
    })
  }
```

```
  // 加入购物车
  addToCart(item) {
    this.cart.addToCart(item);
    this.openSnackBar("已加入您的购物车", "购物车");
  }
  // 显示snackBar
  openSnackBar(message: string, action: string) {
    this.snackBar.open(message, action, {
      duration: 2000,
      verticalPosition: 'top'
    });
  }
  // 显示购物车内容
  showCart() {
    this.showList = !this.showList;
    this.showTable = !this.showTable;
    if (this.showTable) {
      this.cartItems = this.cart.getItems();
    }
  }
  // 计算购物车内商品总数
  getTotalQuantity() {
    let count = 0;
    this.cartItems.forEach(i => count += i.quantity);
    return count;
  }
  // 计算购物车内商品总金额
  getTotalAmount() {
    let count = 0;
    this.cartItems.forEach(i => count += i.quantity * i.unit_price);
    return count;
  }

  // 从购物车中删除一个商品
  removeCartItem(id) {
    console.log('删除一个商品...');
    this.cart.removeCartItem(id);
    this.cartItems = this.cart.getItems();
  }

  // 清空购物车
  clearCart() {
    console.log('删除全部商品...');
    this.cart.clearCart();
    this.cartItems = this.cart.getItems();
  }
}
```

`**重点：**`

- constructor 构造函数中注入所依赖的其它组件类
- ngOnInit 组件初始化函数

5.客户端存储

实例演示

- 数据在客户端的存储

CartService 服务类文件代码（ng-pro/src/app/services/cart.service.ts）

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: "root"
})

export class CartService {
  constructor() {

  }
  items = new Map<string, any>([]);
  // 添加商品到购物车
  addToCart(item) {
    if (!this.items.has(item._id)) {
      item.quantity = 1;
      this.items.set(item._id, item);
    } else {
      this.items.get(item._id).quantity += 1;
    }
  }
  // 获取商品列表
  getItems() {
    return [...this.items.values()];
  }
  // 删除指定商品
  removeCartItem(id) {
    this.items.delete(id);
  }
  // 清空全部商品
  clearCart() {
    this.items.clear();
  }
}
```

`**重点：**`

- Injectable 装饰器的用法和作用
- Map 类型对数据操作几个方法

章总结

重难点

- 在 Vue 中实现商品列表功能【重点】【难点】
- 在 React 中实现登录功能【重点】【难点】
- 在 Angular 中实现购物车功能【重点】【难点】

拓展延伸

思考：如何运用本章的知识去从零开始设计实现一个新项目开发的各个环节和流程？

延伸资料：

- Angular 文档 https://docs.angularjs.org/
- Recat 文档 https://reactjs.org/docs/

- VUE 文档 https://cn.vuejs.org/v2/guide/