第三章：TypeScript 实战（订单系统）

学习目标

- 掌握 TypeScript 开发简单的 API Server
- 掌握 TypeScript 在前端调用后端接口交换数据
- 体验 TypeScript 全栈开发

一、需求分析与设计

1.需求分析

- 订单系统用户为了两类，一类是管理员，另一类是消费者
- 消费者可注册，登录，选择商品创建订单，并可查看已创建的所有订单信息
- 管理员可按条件搜索所有订单信息，并可以修改订单状态

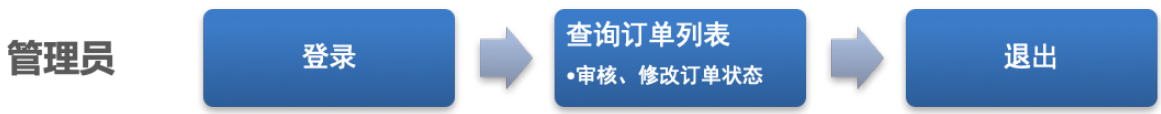
2.实体关系设计

属性名	属性描述 [用户表]
username	用户名
password	登录密码
role	角色：管理员/ 消费者

属性名	属性描述 [商品表]
item_name	商品名称
unit	计量单位：个/箱/盒
unit_price	单价
pic	图片

属性名	属性描述 [订单表]
order_no	订单编号
items	商品名称与数量、单价
user	消费者
amount	总金额
status	订单状态
create_time	创建时间

3.业务流程设计



二、前后端技术介绍

1.Node.js 与 Express 框架介绍

Node.js是什么？

- 简单的说 Node.js 就是运行在服务端的 JavaScript。
- Node.js 是一个基于Chrome JavaScript 运行时建立的一个平台。

- Node.js是一个事件驱动I/O服务端JavaScript环境，基于Google的V8引擎，
- V8引擎执行Javascript的速度非常快，性能非常好。

Express 框架是什么？

- Express 是一个保持最小规模的灵活的 Node.js Web 应用程序开发框架，为 Web 和移动应用程序提供一组强大的功能。
- 使用您所选择的各种 HTTP 实用工具和中间件，快速方便地创建强大的 API。
- Express 提供精简的基本 Web 应用程序功能，而不会隐藏您了解和青睐的 Node.js 功能。
- 许多流行的开发框架都基于 Express 构建。

2.MongoDB 介绍

- MongoDB 是一个基于分布式文件存储的数据库。由 C++ 语言编写。
- 旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。
- MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。
- MongoDB是高性能，开源，无模式的文档型数据库，数据格式是bson。
- MongoDB大数据处理性能优异，提供全索引支持，包括文档内的对象，查询优化器会分析表达式，极大提高查询效率。

3.Vue 介绍

- 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。

三、构建后端API Server

1.创建 HTTP Server

实例演示

- 创建 Node.js 项目
- 添加配置文件
- 编写简单 Server 程序，并调试

app实例代码（ api_server/src/index.ts ）

```
// API Server
import * as express from 'express';
import * as cors from 'cors';
import config = require('./config.json');
import * as db from './models2/index';
db.init();
const app = express();

app.use(async (req, res, next) => {
  console.log(`received ${req.method} : ${req.path}`);
  await next();
});
// 处理跨域
app.use(cors());
// 处理json数据
app.use(express.json());
```

```
app.use(express.urlencoded({ extended: true }));
// 静态资源文件处理
app.use(express.static('./static'));
// 关联api路由
//app.use('/api/', require('./routers/api'));
app.use('/api/', require('./routers/api2'));
app.listen(config.port, () => {
  console.log(`API Server is on ${config.port}...`);
})
```

`**重点：**`

- 中间件的应用
- api路由的关联

2.添加 API 路由

实例演示

- RESTful API 设计
- 实现 API 代码

RESTful API规范与设计举例

```
/*
RESTful API 举例：
1. 所有的事物都被抽象为资源；
2. 每个资源都有一个唯一的资源标识符；
3. 对资源的各种操作不会改变资源标识符；
4. 所有的操作都是无状态的。

GET: 读取（Read）
POST: 新建（Create）
PUT: 更新（Update）
PATCH: 更新（Update）,部分更新
DELETE: 删除（Delete）

POST  /users/              : 创建一个新用户
PUT   /users/:id           : 根据 id 更新一个用户数据（全部字段）
PATCH /users/:id           : 根据 id 更新一个用户数据（部分字段）
GET   /users/              : 查询所有用户记录
GET   /users/:id           : 根据 id 查询一个用户数据
DELETE /users/:id          : 根据 id 删除一个用户
*/
```

api实现代码片段（ api_server/src/routers/api.ts ）

```
const router = Router();

// 用户注册
router.post('/users/register', (req, res, next) => {
  //res.send("users.regsiter");
  console.log(req.body);
  let user = new User(req.body.username, req.body.password);
  if (user.save()) {
    res.send({ code: 0 });
  } else {
```

```
    res.send({ code: 1 });
  }
});

// 用户登录
router.post('/users/login', (req, res, next) => {
  //res.send("users.login");
  console.log(req.body);
  let result = User.login(req.body.username, req.body.password);
  if (result) {
    res.send({
      code: 0,
      data: result
    });
  } else {
    res.send({
      code: -1,
      msg: "wrong username or password"
    });
  }
});

// 商品列表
router.get('/items/list', (req, res, next) => {
  //res.send("items.list");
  res.send({
    code: 0,
    data: Item.getList()
  })
});
```

重点：

- 请求方式：Post 与 Get
- URL Path部分命名
- 返回JSON格式数据

3.调试 API

实例演示

- 创建测试数据
- 使用 curl 命令请求 API
- 验证接口正确性

curl 命令实例: POST方式请求用户登录接口

```
curl -X POST -H 'content-type: application/json' -d
'{"username":"jack","password":"123456"}' http://localhost:3000/api/users/login
```

查看curl使用方法

```
curl -h
```

四、数据库设计与实现

1.数据模型定义

实例演示

- Mongoose 简介
- 数据模型定义格式

2.数据模型实现

实例演示

- 数据表详细设计，创建模型文件
- 实现 API 逻辑（如：数据校检、查询条件等）

数据模型实例代码：订单数据模型（api_server/src/models2/order.ts）

```
// 订单数据模型
import { Document, Schema, Model, model } from 'mongoose';
import { IOrder } from '../interfaces/order';

export interface IOrderModel extends IOrder, Document {

}

export const OrderSchema: Schema = new Schema({
  order_no: {
    type: String,
    required: true
  },
  user: {
    type: Schema.Types.ObjectId,
    ref: 'User',
    required: [true, '需要用户ID']
  },
  items: {
    type: [{
      item: {
        type: Schema.Types.ObjectId,
        ref: 'Item',
      },
      unit_price: {
        type: Number,
        required: true
      },
      quantity: {
        type: Number,
        required: true
      }
    }],
    required: [true, '需要商品信息列表']
  },
  amount: {
    type: Number,
    required: true
  },
  status: {
    type: String,
    enum: ['handling', 'completed'],
```

```
    default: 'handling',
    required: true
  }
},
  {
    collection: "orders",
    versionKey: false,
    validateBeforeSave: true,
    timestamps: {
      createdAt: 'created_at',
      updatedAt: 'updated_at'
    }
  });

export const Order: Model<IOrderModel> = model<IOrderModel>('Order',
OrderSchema);
```

重点：

- 内套子字段（复杂类型）的定义
- ObjectId 类型字段关联其它数据模型

3.调试 API

实例演示

- 测试 API
- 修复 Bug

五、前端展示与操作

1.创建 Vue 项目

实例演示

- 创建项目并初始化
- 安装依赖组件并配置

创建Vue项目并启动

```
npm install -g @vue/cli
vue create front_end
cd front_end
npm run serve
```

2.创建 UI 页面

实例演示

- 创建页面，并选择合适的页面组件
- 定义组件的行为与事件处理

Vue页面文件实例（front_end/src/views/Item.vue）

```
<template>
  <div class="about">
    <h1>商品信息列表</h1>
```

```
    <ul>
      <li v-for="(item,i) in itemList" :key="i"
style="width:320px;padding:5px;">
        <el-card :body-style="{ padding: '0px', height: '350px'}">
          <img :src="host + item.pic" class="image" style="padding-top:30px;" />
          <div style="padding: 14px;">
            <span>{{item.name}}</span>
            <div class="bottom clearfix">
              <time class="time">{{ (item.unit_price/100).toFixed(2) + '元/' +
item.unit }}</time>
              <el-button type="text" class="button" @click="addToCart(i)">加入购
物车</el-button>
            </div>
          </div>
        </el-card>
      </li>
    </ul>
  </div>
</template>

<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;
  async created() {
    let result = await API.getItems();
    console.log(result);
    this.itemList = result.data.data;
  }
  addToCart(index: number) {
    console.log(index);
    this.$store.commit("addToCart", this.itemList[index]);
    this.$message({
      message: "已加入您的购物车",
      type: "success"
    });
  }
}
</script>
<style scoped>
li {
  list-style: none;
  padding: 0;
  margin: 0;
  float: left;
  display: block;
}
</style>
```

重点：

- 模板代码中属性与事件处理函数的绑定
- vue中store的用法

router 配置代码（ front_end/src/router/index.ts)

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/Home.vue'
import Login from '../views/Login.vue'
import Register from '../views/Register.vue'
import Item from '../views/Item.vue'
import Cart from '../views/Cart.vue'
import Order from '../views/Order.vue'
import store from '../store'
Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    name: 'home',
    component: Home,
    redirect: { name: 'item' },
    children: [
      {
        path: '/item',
        name: 'item',
        component: Item
      },
      {
        path: '/cart',
        name: 'cart',
        component: Cart
      },
      {
        path: '/order',
        name: 'order',
        component: Order
      }
    ]
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  },
  {
    path: '/register',
    name: '/register',
    component: Register
  },
]

const router = new VueRouter({
  mode: 'history',
  base: process.env.BASE_URL,
  routes
})
router.beforeEach((to, from, next) => {
  console.log(store.state);
  console.log(to);
```

```
  if (to.name == 'login' || to.name == 'register' || store.state.user) {
    return next();
  } else {
    return next('/login');
  }
});
export default router
```

重点：

- 父子页面的路由配置
- 路由判断与页面跳转

store 本地缓存代码 (front_end/src/store/index.ts)

```
import Vue from 'vue'
import Vuex from 'vuex'
import VuePersistence from 'vuex-persist'
const vuexLocal = new VuePersistence({
  //storage: window.localStorage
  saveState: (key, state, storage) => {
    let all = {
      user: (<any>state).user,
      cart: Array.from((<any>state).cart || [])
    };
    window.localStorage.setItem(key, JSON.stringify(all));
  },
  restoreState: (key, storage) => {
    let all = JSON.parse(window.localStorage.getItem(key) || "{}");
    return {
      user: all.user || null,
      cart: new Map(all.cart)
    }
  }
})
Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    cart: new Map<string, any>([]),
    user: null
  },
  mutations: {
    saveUser(state: any, user) {
      state.user = user;
    },
    clearUser(state) {
      state.user = null;
    },
    addToCart(state, item) {
      if (!state.cart.has(item._id)) {
        item.quantity = 1;
        state.cart.set(item._id, item);
      } else {
        state.cart.get(item._id).quantity += 1;
      }
    },
```

```
    removeFromCart(state, id) {
      state.cart.delete(id);
    },
    clearCart(state) {
      state.cart.clear();
    }
  },
  actions: {
  },
  modules: {
  },
  plugins: [vuexLocal.plugin]
})
```

重点：

- vue store 实现数据的缓存的方式
- 购物车商品数据的添加、删除与清空操作业务逻辑

3.联调 API

实例演示

- 封装 API 访问组件
- 请求与提交数据

API访问封装代码（ front_end/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() {
    return axios.get('/items/list');
  }
  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 });
  }
}
```

`重点：`

- 配置 axios header 参数
- Post 与 Get 参数提交方式

章总结

重难点

- 需求分析与设计【重点】【难点】
- 构建 API Server【重点】【难点】
- 前端展示与操作【重点】
- 数据库设计【易错点】
- 前端 API 调用【易错点】

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：

- TypeScript 官网文档 http://www.typescriptlang.org/docs/home.html
- TypeScript 中文文档 https://www.tslang.cn/docs/home.html
- Node.js 文档 https://nodejs.org/en/docs/
- MongoDB 文档 https://docs.mongodb.com/manual/
- Express 文档 http://expressjs.com/
- VUE 文档 https://cn.vuejs.org/v2/guide/