公园管理系统 - 前端开发指南

注：目前前端框架中的的详细页面命名或者页面的分布根据开发需要自行调整

目录

---

1. 技术栈概览

1.1 核心技术栈

技术	版本	用途	说明
Vue 3	3.x	前端框架	使用 Composition API，更好的逻辑复用
Vite	4.x	构建工具	快速的开发服务器和构建工具
Element Plus	2.x	UI 组件库	基于 Vue 3 的企业级 UI 组件库
Pinia	2.x	状态管理	轻量级状态管理，替代 Vuex
Vue Router	4.x	路由管理	官方路由管理器
Axios	1.x	HTTP 请求	Promise 基础的 HTTP 客户端

1.2 开发工具

工具	用途
ESLint	代码质量检查
Prettier	代码格式化
Vitest	单元测试框架

---

2. 项目结构

2.1 目录结构

```
src/
├── components/           # 可复用组件
│   ├── common/          # 通用组件（表格、表单、对话框等）
```

```
│   └── business/       # 业务组件（用户卡片、设施状态等）
├── views/              # 页面组件
│   ├── dashboard/      # 仪表板
│   ├── visitors/       # 游客管理
│   ├── tickets/        # 票务管理
│   ├── facilities/     # 设施管理
│   ├── finance/        # 财务管理
│   ├── hr/             # 人力资源
│   └── system/         # 系统管理
├── stores/             # Pinia 状态管理
├── router/             # 路由配置
├── utils/              # 工具函数
├── api/                # API 接口封装
├── assets/             # 静态资源
└── styles/             # 全局样式
```

2.2 文件命名规范

类型	命名规范	示例
组件文件	PascalCase	`UserList.vue`
页面文件	PascalCase	`DashboardView.vue`
工具文件	kebab-case	`date-utils.js`
样式文件	kebab-case	`common-styles.scss`
常量文件	kebab-case	`api-constants.js`

---

3. 核心架构设计

3.1 权限管理设计

角色定义

```
const roles = {
  super_admin: '超级管理员',
  finance_manager: '财务管理员',
  hr_manager: '人事管理员',
  operations_manager: '运营管理员',
  ticket_manager: '票务管理员',
  customer_service: '客服人员',
  employee: '普通员工'
}
```

权限控制层级

- **路由级别**: 控制页面访问权限

- **菜单级别**: 动态显示可访问菜单
- **按钮级别**: 控制操作按钮显示
- **数据级别**: 控制数据访问范围

权限验证示例

```
// 路由守卫
router.beforeEach((to, from, next) => {
  const userStore = useUserStore()

  if (to.meta.requiresAuth && !userStore.isLoggedIn) {
    next('/login')
  } else if (to.meta.roles && !userStore.hasRole(to.meta.roles)) {
    next('/403')
  } else {
    next()
  }
})
```

3.2 状态管理设计

用户状态 (stores/user.js)

```
export const useUserStore = defineStore('user', {
  state: () => ({
    token: null,
    userInfo: null,
    permissions: []
  }),

  actions: {
    async login(credentials) {
      // 登录逻辑
    },

    async logout() {
      // 登出逻辑
    },

    hasRole(roles) {
      // 权限检查
    }
  }
})
```

应用状态 (stores/app.js)

```
export const useAppStore = defineStore('app', {
  state: () => ({
    theme: 'light',
    sidebarCollapsed: false,
    globalLoading: false,
    language: 'zh-CN'
  }),

  actions: {
    toggleSidebar() {
      this.sidebarCollapsed = !this.sidebarCollapsed
    },

    setGlobalLoading(loading) {
      this.globalLoading = loading
    }
  }
})
```

3.3 组件架构

布局组件

- **DefaultLayout**: 主布局容器，包含侧边栏和主内容区
- **Sidebar**: 侧边栏导航，支持折叠和权限控制
- **Navbar**: 顶部导航栏，包含用户信息和操作
- **Breadcrumb**: 面包屑导航，显示当前页面路径

业务组件

- **PageTemplate**: 通用页面模板，统一页面结构
- **DataTable**: 数据表格组件，支持分页、排序、筛选
- **SearchForm**: 搜索表单组件，统一搜索交互
- **各业务模块专用组件**

基础组件

基于 Element Plus 的二次封装：

- **FormDialog**: 表单对话框，统一表单交互
- **ConfirmDialog**: 确认对话框，统一确认操作
- **ImageUpload**: 图片上传组件，支持预览和删除
- **DateRangePicker**: 日期范围选择器

3.4 路由设计

路由结构

```
/login                    # 登录页
/                         # 主布局
├── /dashboard            # 仪表板
├── /visitors/*           # 游客管理
├── /tickets/*            # 票务管理
├── /facilities/*         # 设施管理
├── /finance/*            # 财务管理
├── /hr/*                 # 人力资源
└── /system/*             # 系统管理
```

路由配置示例

```
{
  path: '/visitors',
  name: 'Visitors',
  component: DefaultLayout,
  meta: {
    title: '游客管理',
    requiresAuth: true,
    roles: ['admin', 'customer_service']
  },
  children: [
    {
      path: '',
      name: 'VisitorList',
      component: () => import('@/views/visitors/VisitorList.vue'),
      meta: { title: '游客列表' }
    }
  ]
}
```

2.6 API 设计

请求封装

- **统一拦截器**: 处理认证、错误、加载状态
- **环境配置**: 支持多环境 API 地址
- **错误处理**: 统一的错误提示和处理
- **请求重试**: 自动重试机制
- **请求缓存**: 合理的缓存策略

接口规范

```
// 请求格式
{
  method: 'GET|POST|PUT|DELETE',
  url: '/api/endpoint',
```

```
  data: {},
  params: {}
}

// 响应格式
{
  code: 200,
  message: 'success',
  data: {},
  timestamp: '2024-01-01T00:00:00Z'
}
```

2.7 开发规范

命名规范

- **组件**: PascalCase (UserList.vue)
- **文件**: kebab-case (user-list.vue)
- **变量**: camelCase (userName)
- **常量**: UPPER_SNAKE_CASE (API_BASE_URL)
- **CSS类**: kebab-case (.user-card)

代码组织

```
<template>
  <!-- 模板内容 -->
</template>

<script setup>
// 1. 导入依赖
// 2. 定义 props 和 emits
// 3. 响应式数据
// 4. 计算属性
// 5. 方法定义
// 6. 生命周期钩子
</script>

<style scoped>
/* 组件样式 */
</style>
```

---

4. 开发实践指南

4.1 API 请求封装

请求拦截器

```
// 请求拦截器
service.interceptors.request.use(
  config => {
    const userStore = useUserStore()
    const appStore = useAppStore()

    // 显示全局加载状态
    appStore.setGlobalLoading(true)

    // 添加认证 token
    if (userStore.token) {
      config.headers.Authorization = `Bearer ${userStore.token}`
    }

    return config
  }
)
```

响应拦截器

```
// 响应拦截器
service.interceptors.response.use(
  response => {
    const appStore = useAppStore()
    appStore.setGlobalLoading(false)

    return response.data
  },
  error => {
    const appStore = useAppStore()
    appStore.setGlobalLoading(false)

    // 统一错误处理
    handleApiError(error)
    return Promise.reject(error)
  }
)
```

4.2 环境配置

开发环境配置

```
VITE_APP_TITLE=主题公园管理系统 - 开发环境
VITE_API_BASE_URL=http://localhost:8080/api
VITE_APP_ENV=development
```

代理配置

```
server: {
  port: 3000,
  open: true,
  proxy: {
    '/api': {
      target: 'http://localhost:8080',
      changeOrigin: true,
      secure: false
    }
  }
}
```

---

5. API 交互说明

5.1 请求响应格式

请求格式

```
{
  method: 'GET|POST|PUT|DELETE',
  url: '/api/endpoint',
  data: {},
  params: {}
}
```

响应格式

```
{
  code: 200,
  message: 'success',
  data: {},
  timestamp: '2024-01-01T00:00:00Z'
}
```

5.2 API 接口示例

用户管理接口

```
// 获取用户列表
GET /api/users?page=1&size=10
```

```
// 获取用户详情
GET /api/users/{id}

// 创建用户
POST /api/users
{
  "name": "张三",
  "email": "zhangsan@example.com",
  "role": "employee"
}

// 更新用户
PUT /api/users/{id}
{
  "name": "李四",
  "email": "lisi@example.com"
}

// 删除用户
DELETE /api/users/{id}
```

---

6. 开发实践指南

6.1 添加新功能模块的步骤

步骤1: 创建页面组件

```
<!-- src/views/example/ExampleList.vue -->
<template>
  <div class="example-list">
    <el-card>
      <template #header>
        <div class="card-header">
          <span>示例管理</span>
          <el-button type="primary" @click="handleAdd">新增</el-button>
        </div>
      </template>

      <el-table :data="tableData" :loading="loading">
        <el-table-column prop="id" label="ID" />
        <el-table-column prop="name" label="名称" />
        <el-table-column label="操作">
          <template #default="{ row }">
            <el-button size="small" @click="handleEdit(row)">编辑</el-button>
            <el-button size="small" type="danger" @click="handleDelete(row)">删除
</el-button>
          </template>
        </el-table-column>
      </el-table>
    </el-card>
```

```
  </div>
</template>

<script setup>
import { ref, onMounted } from 'vue'
import { useExampleStore } from '@/stores/example'

const exampleStore = useExampleStore()
const tableData = ref([])
const loading = ref(false)

const fetchData = async () => {
  loading.value = true
  try {
    tableData.value = await exampleStore.getList()
  } finally {
    loading.value = false
  }
}

onMounted(() => {
  fetchData()
})
</script>
```

步骤2: 创建状态管理

```
// src/stores/example.js
import { defineStore } from 'pinia'
import { request } from '@/utils/request'

export const useExampleStore = defineStore('example', {
  state: () => ({
    list: [],
    currentItem: null
  }),

  actions: {
    async getList(params = {}) {
      const data = await request.get('/examples', params)
      this.list = data.items || []
      return this.list
    },

    async getById(id) {
      const data = await request.get(`/examples/${id}`)
      this.currentItem = data
      return data
    },

    async create(item) {
```

```
    const data = await request.post('/examples', item)
    this.list.unshift(data)
    return data
  },

  async update(id, item) {
    const data = await request.put(`/examples/${id}`, item)
    const index = this.list.findIndex(x => x.id === id)
    if (index !== -1) {
      this.list[index] = data
    }
    return data
  },

  async delete(id) {
    await request.delete(`/examples/${id}`)
    this.list = this.list.filter(x => x.id !== id)
  }
 }
})
```

步骤3: 配置路由

```
// src/router/index.js
{
  path: '/example',
  name: 'Example',
  component: () => import('@/layouts/DefaultLayout.vue'),
  meta: {
    title: '示例管理',
    requiresAuth: true,
    roles: ['admin', 'employee']
  },
  children: [
    {
      path: '',
      name: 'ExampleList',
      component: () => import('@/views/example/ExampleList.vue'),
      meta: { title: '示例列表' }
    },
    {
      path: 'create',
      name: 'ExampleCreate',
      component: () => import('@/views/example/ExampleForm.vue'),
      meta: { title: '新增示例' }
    }
  ]
}
```

6.2 修改现有页面的标准流程

1. **定位文件**: 在 `src/views/` 中找到对应模块
2. **修改组件**: 更新模板、逻辑或样式
3. **更新状态**: 如需要，修改对应的 store
4. **测试验证**: 确保功能正常和数据流正确

6.3 代码质量保证

代码检查

- **ESLint**: 代码风格检查
- **Prettier**: 代码格式化
- **Vue官方风格指南**: 遵循Vue.js官方推荐

测试策略

- **单元测试**: 使用 Vitest 进行组件测试
- **集成测试**: API 接口测试

---

7. 部署配置说明

7.1 前端部署

构建配置

```
# 开发环境
npm run dev

# 生产构建
npm run build

# 预览构建结果
npm run preview
```

环境变量

```
# .env.production
VITE_APP_TITLE=主题公园管理系统
VITE_API_BASE_URL=https://api.example.com/api
VITE_APP_ENV=production
```

部署到静态服务器

```
# 构建项目
npm run build
```

```
# 将 dist 目录部署到 Web 服务器
# 例如：Nginx、Apache、或云服务商的静态托管服务
```

Nginx 配置示例

```
server {
    listen 80;
    server_name your-domain.com;
    root /path/to/dist;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://backend-server:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

相关文档

- Vue 3 官方文档
- Element Plus 组件库
- Pinia 状态管理
- Vite 构建工具
- Vue Router 路由管理

更新日志

版本	日期	更新内容
1.0.0	2024-01-01	初始版本
1.1.0	2024-01-15	添加权限管理设计
1.2.0	2024-02-01	完善开发实践指南