组件的设计

10.1 应用组件设计主页功能模块

10.1.1 应用插槽设计 TabBar 组件

本节介绍将首页底部导航封装为一个单独的 TabBar 组件。

1. 将本书提供的资源文件夹中的 TabBar 组件需要的图片文件夹复制到 myproject 项目的路径 src→assets→img 下，如图所示。

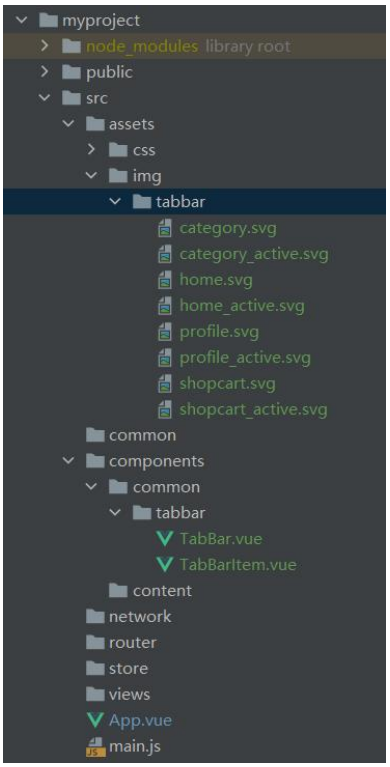


图 11-1 复制 tabbar 组件到 myproject 项目

2. 由于 src→components→common 文件夹一般用于存放可以同时用于本项目以及其他项目的公共组件，为了提高复用性创建组件 tabbar，在 common 路径下新建文件夹 tabbar，在 src→components→common→tabbar 中新建两个组件文件，分别为 TabBar.vue 和 TabBarItem.vue 文件。由于项目中一般只有一个 tabbar，因此在<template></template>中设置"tab-bar"为 id。

TabBar.vue 组件中的代码如下：

```
<template>//结构设计
  <div id="tab-bar">
    <slot></slot>
  </div>
</template>
```

```
<script>//行为设计
  export default {
    name: "TabBar"
  }
</script>
<style scoped>//样式设计
  #tab-bar {
    display: flex;
    background-color: #f6f6f6;
    position: fixed;
    left: 0;
    right: 0;
    bottom: 0;
    box-shadow: 0 -1px 1px rgba(100,100,100,.2);
  }
</style>
```

知识点：插槽的应用

<slot></slot>插槽，插槽实质是对子组件的扩展，插槽的作用是为了让组件更具有扩展性，通过<slot>插槽向组件内部指定位置传递内容。slot 是为了父组件可以在子组件中加入内容，直接在组件模板需要的位置放入<slot></slot>标签对。一般，封装的原则主要是抽取共性，保留个性，对个性的部分预留插槽 slot。

其中，display: flex;Flex 是 Flexible Box 的缩写，意为“弹性布局”，用来为盒状模型提供最大的灵活性。任何一个容器都可以指定为 Flex 布局。采用 Flex 布局的元素，称为 Flex 容器（flex container），简称“容器”。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称“项目”。

插槽显不显示、怎样显示是由父组件控制，而插槽在哪里显示由子组件控制。插槽显示的位置由子组件自身决定，插槽写在组件模板的什么位置，父组件传过来的模板将来就显示在什么位置。

例如，在一个 Father 父组件内部使用一个名为 Child 的子组件，向子组件的内部的“指定位置”传递内容：

```
<!-- 这是子组件-→
<div class="child">
  <h2>Child 的标题</h2>
  <slot>我是一个插槽</slot>
</div>
```

```
<!-- 这是父组件-→
<div class="father">
  <h1>Father 的标题</h1>
  <Child>
    我是需要放置到插槽中的内容
  </Child>
</div>
```

上述代码运行结果如下：

```
Father 的标题
Child 的标题
我是需要放置到插槽中的内容
```

常用的插槽主要有匿名插槽和具名插槽。

匿名插槽，也称为单个插槽或默认插槽；具名插槽，意思就是具有名字的插槽，名字通过属性 name 来定义。一个组件中可以有很多具名插槽，出现在不同的位置。

匿名插槽就是没有设置 name 属性的插槽，可以放置在组件的任意位置，一个组件中只能有一个该类插槽，作为找不到匹配的内容片段时的备用插槽，匿名插槽只能作为没有 slot 属性的元素的插槽。例如：

```
<div class="child">
      <h1>子组件</h1>
       <slot name="head">头部默认值</slot>
       <slot name="body">主体默认值</slot>
       <slot>这是个匿名插槽(没有 name 属性)，这串字符是匿名插槽的默认值。</slot>
 </div>
```

```
<div class="parent">
 <h1>父组件</h1>
 <child>
       <p slot="body">我是主体</p>
       <p>我是其他内容</p>
       <p slot="footer">我是底部</p>
 </child>
 </div>
```

上述代码运行结果如下：

```
父组件
子组件
头部默认值
我是主体
我是其他内容
```

注意：

我是底部 部分被丢弃了，是因为子组件中没有 name="footer"的插槽与之匹配。如果子组件中的匿名插槽不存在，则 我是其他内容 也会被丢弃。

插槽的默认值：

在模板的 slot 标签对之间插入的标签将成为插槽的默认值，如果在 html 中没有在自定义组件标签对之间插入相应标签，则会使用插槽内容的默认标签；

如果有多个值，同时放入到组件进行替换时，一起作为替换元素。

10.1.2 设计 TabBarItem 组件

设计组件 TabBarItem，在其中定义两个插槽，插槽的 name 分别为“item-icon”和“item-text”，分别可以传入图片、文字。在两个插槽外层添加 div，用于设置样式。由于项目中 tabbaritem 为多项，因此在<template></template>中设置"tab-bar-item"为 class。TabBarItem.vue 组件中的代码如下：

```
<template>
```

```
    <div class="tab-bar-item">
      <slot name="item-icon"></slot>
      <slot name="item-text"></slot>
    </div>
</template>

<script>
    export default {
      name: "TabBarItem"
    }
</script>

<style scoped>
    .tab-bar-item {
      flex: 1;
      text-align: center;
      height: 49px;
      font-size: 14px;
    }
    .tab-bar-item img {
      width: 24px;
      height: 24px;
      margin-top: 3px;
      vertical-align: middle;
      margin-bottom: 2px;
    }
</style>
```

在 App.vue 中添加以下代码：

```
<template>
    <div id="app">
      <tab-bar>
        <tab-bar-item>
          <img slot="item-icon" src="~assets/img/tabbar/home.svg" alt="">
          <div slot="item-text">首页</div>
        </tab-bar-item>
        <tab-bar-item>
          <img slot="item-icon" src="~assets/img/tabbar/category.svg" alt="">
          <div slot="item-text">分类</div>
        </tab-bar-item>
        <tab-bar-item>
          <img slot="item-icon" src="~assets/img/tabbar/shopcart.svg" alt="">
          <div slot="item-text">购物车</div>
        </tab-bar-item>
        <tab-bar-item>
```

```
            <img slot="item-icon" src="~assets/img/tabbar/profile.svg" alt="">
            <div slot="item-text">我的</div>
        </tab-bar-item>
    </tab-bar>
  </div>
</template>

<script>
  //以下两条代码用于引入组件 TabBar 和 TabBarItem
  import TabBar from 'components/common/tabbar/TabBar'
  import TabBarItem from 'components/common/tabbar/TabBarItem'
  export default {
    name: 'App',
    components: {   //用于注册组件
      TabBar,
      TabBarItem
    }
  }
</script>

<style>
  @import "assets/css/base.css";
</style>
```

在终端输入以下命令运行项目，项目在浏览器的运行效果如图所示，按下 F12 按键可以打开开发者调试工具，可以看到页面低端的四个功能模块。

```
npm run serve
```

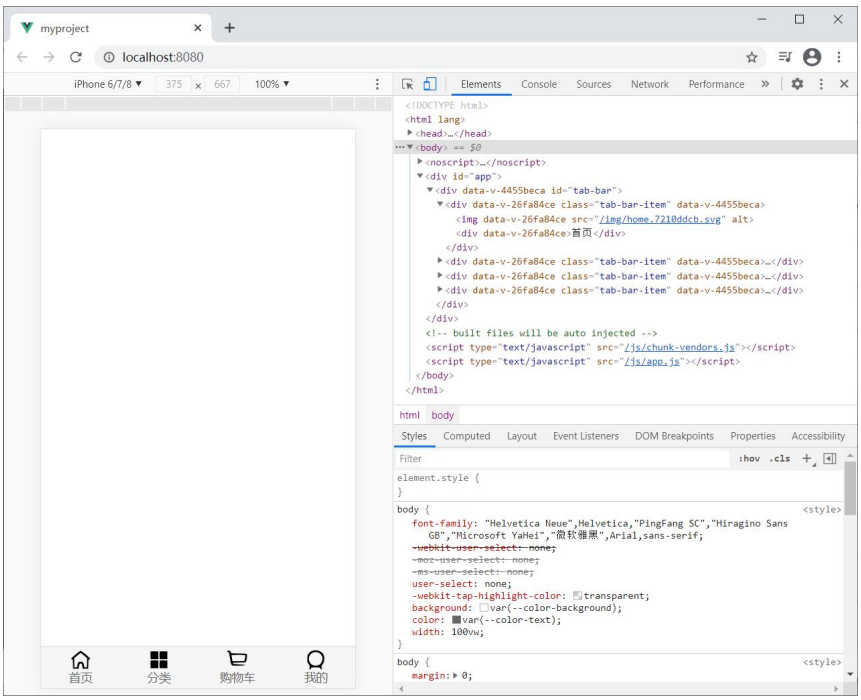


图 11-2 项目运行在浏览器效果

10.1.3 改进 TabBar 组件

当某一功能模块处于 active 状态时，一般需要设置图标和文字为不同的颜色。因此需要在上图的基础上添加不同 active 状态对图标和文字颜色的动态设置。这里定义另外一个插槽，插入 active-icon 的数据，同时定义一个变量 isActive，通过 v-show 来决定是否显示对应的 icon。

1. 分别修改 TabBarItem 和 App.vue 中的代码，增加插槽显示 active 状态下切换不同颜色图片的显示。TabBarItem.vue 中的代码如下，相比 3.1.3 节中有修改的代码改为斜体加粗显示，以便区分。

```
<template>
  <div class="tab-bar-item">
    <slot name="item-icon"></slot>
    <slot name="item-icon-active"></slot>
    <slot name="item-text"></slot>
  </div>
</template>
```

App.vue 文件修改后如下：

```
<template>
  <div id="app">
    <tab-bar>
      <tab-bar-item>
        <img slot="item-icon" src="~assets/img/tabbar/home.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/home_active.svg" alt="">
        <div slot="item-text">首页</div>
      </tab-bar-item>
      <tab-bar-item>
        <img slot="item-icon" src="~assets/img/tabbar/category.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/category_active.svg" alt="">
        <div slot="item-text">分类</div>
      </tab-bar-item>
      <tab-bar-item>
        <img slot="item-icon" src="~assets/img/tabbar/shopcart.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/shopcart_active.svg" alt="">
        <div slot="item-text">购物车</div>
      </tab-bar-item>
      <tab-bar-item>
        <img slot="item-icon" src="~assets/img/tabbar/profile.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/profile_active.svg" alt="">
        <div slot="item-text">我的</div>
      </tab-bar-item>
    </tab-bar>
  </div>
</template>
```

在终端执行 npm run serve，浏览器显示如图所示。

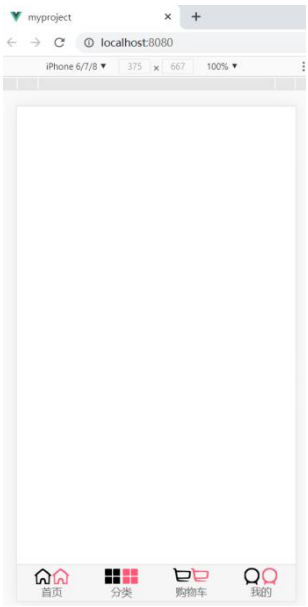
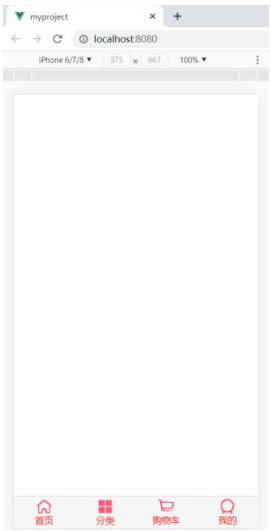


图 11-3 项目运行效果

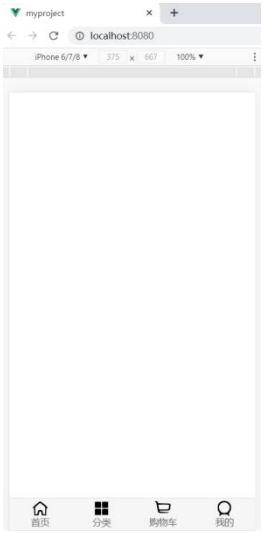
2. 目前在页面底端显示两张图标，我们需要使用变量 isActive 的 True 或 False 控制两种图片的显示，修改 TabBarItem.vue 文件，代码如下，项目运行效果如图所示。注意：为了避免插槽中的属性被父组件替换，一般将插槽外部加上 标记对。有修改的代码改为斜体加粗显示。

```
<template>
  <div class="tab-bar-item">
    <div v-if="!isActive"><slot name="item-icon"></slot></div>
    <div v-else><slot name="item-icon-active"></slot></div>
    <div :class="{active: isActive}"><slot name="item-text"></slot></div>
  </div>
</template>

<script>
  export default {
    name: "TabBarItem",
    data(){
      return{
        isActive:true
      }
    }
  }
</script>

<style scoped>
  .tab-bar-item {
    flex: 1;
    text-align: center;
    height: 49px;
```

```
        font-size: 14px;
    }
    .tab-bar-item img {
        width: 24px;
        height: 24px;
        margin-top: 3px;
        vertical-align: middle;
        margin-bottom: 2px;
    }
    .active{
        color: red;
    }
</style>
```



isActive 值为 true 的情况



isActive 值为 false 的情况

图 11-4 项目运行效果

10.1.4 TabBarItem 绑定路由数据

为 TabBar 组件的不同模块绑定路由，即单击不同的功能模块在页面展示功能相对应的不同页面。

vue-router 是 Vue.js 官方的路由插件，与 vue.js 深度集成，适合用于构建单页面应用。vue-router 基于路由和组件，路由用于设定访问路径，将路径和组件映射起来。在 vue-router 的单页面应用中，页面路径的改变就是组件的切换。

1. 在项目中安装路由

在终端执行 npm install vue-router --save 安装路由，路由属于运行时依赖。

在 Github 上 git clone 一个项目并打开项目下的 package.json 文件，一般都有 dependencies 和 devDependencies 两个配置项，dependencies 中存放的是项目依赖，devDependencies 存放的是环境依赖，也就是项目开发时所需要的依赖。

node package 有两种常用的依赖 dependencies 和 devDependencies。

（1）dependencies 是生产环境需要的依赖，正常运行该包时所需要的依赖项，即运行时依赖。写入模块的代码如下，如果不指定版本号 version，则默认安装最新版本。

```
npm install/i xxx@version -S/--save
```

（2）devDependencies 是开发环境需要的依赖，不用于线上生产环境，例如一些进行单元测试相关的包，写入模块的代码如下。

```
npm install/i xxx -D/--save-dev
```

在 vue 项目中，若文件使用 import 引入 devDependencies 中的插件则会把当前引入的插件打包到文件中，若不引入则不打包。而 dependencies 中的插件不管是否引入都会被打包到文件中。

在终端成功安装路由后，终端显示如图所示。

图 11-5 终端成功安装路由

2. 配置 TabBar 组件的路由

在 2.3.2 节的优化目录结构中，已经在项目的 src 目录下新建了路由 router 目录，用于管理和配置路由。在 src→router 路径下新建 index.js 文件，设置路由。

在项目中使用 vue-router 路由的步骤如下：

第一步：由于 vue-rooter 是一个插件，可以通过 Vue.use()安装路由功能，导入路由对象，并调用 Vue.use（VueRouter）；

第二步：创建路由实例，并且传入路由映射配置；

第三步：在 Vue 实例中挂载创建的路由实例。

具体配置路由方法如下：

（1）导入路由对象

在 src→router→index.js 文件中添加以下代码，用于导入路由对象。

```
import Vue from 'vue'
import VueRouter from 'vue-router'
```

（2）安装路由插件，即安装路由功能

```
//1.注入路由插件
Vue.use(VueRouter)
```

（3）创建路由对象，传入路由映射配置

```
//2.定义路由
const routes = [
//后面在这里配置映射关系
]
//3.创建路由 router 实例
const router = new VueRouter({
  routes
})
```

（4）导出路由 router

```
//4.导出路由 router 实例
export default router
```

（5）main.js 中注册 router

在 vue-cli 工程的入口文件 main.js 中导入 src→router→index.js 文件中导出的路由，main.js 中的代码如下，有路由导入的代码改为斜体加粗显示，以便区分（后续加粗并斜体显示的为相较之前新增的代码）。

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'

Vue.config.productionTip = false
//在 Vue 实例中挂载创建的路由实例
new Vue({
  el:'#app',
  router,
  render: h => h(App),
}).$mount('#app')
```

（6）创建路由对应的组件

在 src→views 路径中分别创建路由组件及所在的文件夹如下，这样不同的功能模块可以由团队分工协作完成。

src→views→cart→Cart.vue
src→views→cattgory→Category.vue
src→views→home→Home.vue
src→views→profile→Profile.vue

其中，src→views→home→Home.vue 代码如下，Cart.vue、Category.vue 以及 Profile.vue 中代码相似。

```
<template>
  <h2>首页</h2>
</template>

<script>
  export default {
    name: "Home"
  }
</script>

<style scoped>
</style>
```

（7）配置路由映射

在 src→router→index.js 文件中配置组件和路径映射关系，新增的代码如下。

```
import Vue from 'vue'
import VueRouter from 'vue-router'
//以下代码导入组件
const Home = () => import('views/home/Home')
const Category = () => import('views/category/Category')
const Cart = () => import('views/cart/Cart')
const Profile = () => import('views/profile/Profile')
```

```
// 1.安装插件
Vue.use(VueRouter)
// 2.创建路由对象，配置路由映射
const routes = [
  {  //设置默认路由
    path: '',
    redirect: '/home'
  },
  {
    path: '/home',
    component: Home
  },
  {
    path: '/category',
    component: Category
  },
  {
    path: '/cart',
    component: Cart
  },
  {
    path: '/profile',
    component: Profile
  }
]
const router = new VueRouter({
  routes,
  mode: 'history'   //设置路由的模式
})
// 3.导出 router
export default router
```

前端路由的核心：改变 url，但是页面不进行整体的刷新。vue-router 路由（即改变路径）有两种模式：hash（默认模式）和 history。这两种模式都会更新视图，但不会重新请求页面，在地址发生改变的时候，会在浏览器中新增一条记录，通过这个记录实现更新视图，但是不请求后台服务器。

hash 模式：

- url 的 hash 即是锚点#，例如 http://www.abc.com/#/hello，hash 值为 hello；
- 特点：hash 不会包括在 http 请求中，改变 hash 不会重新加载页面；
- hash 模式依靠 onhashchange()事件监听 location.hash 的改变。

history 模式：

- 利用 HTML5 History Interface 中新增的 pushState()和 replaceState()方法；
- pushState()改变 url 地址且不会发送请求；
- replaceState()可以读取历史记录栈，并且可对浏览器记录修改；
- 应用于浏览器的历史记录栈，提供对历史记录进行修改，改变当前的 url，不会向后端

发送请求。

两种模式的比较：

- history 设置的新 URL 可以是同源的任意 URL，而 hash 模式只能够修改#后面的部分，故只可设置与当前同文档的 URL；
- history 可以添加任意类型的数据到记录当中，hash 模式只能够添加短字符串；
- history 模式可以额外添加 title 属性，提供后续使用；
- history 模式则会将 URL 修改得就和正常请求后端的 URL 一样，如后端没有配置对应/user/id 的路由处理，则会返回 404 错误。

3. 监听 TabBarItem 组件中的点击事件

本文第 3.1.3 节 App.vue 中<template></template>内的代码用于向 TabBarItem 组件内传入文字和图片，若在 App.vue 中添加事件监听则需要在每一个<tab-bar-item>内添加事件监听，为了避免代码冗余，在组件 TabBarItem.vue 内部添加鼠标单击 Item 事件的监听，通过 this.$router.replace()替换路由路径，添加的代码如下。

```
<template>
  <div class="tab-bar-item" @click="itemClick">
    <div v-if="!isActive"><slot name="item-icon"></slot></div>
    <div v-else><slot name="item-icon-active"></slot></div>
    <div :class="{active: isActive}"><slot name="item-text"></slot></div>
  </div>
</template>

<script>
  export default {
    name: "TabBarItem",
    //在 TabeBarItem.vue 中设置 props 用来向子组件传值
    props: {
      path: String      //指定一个字符串型的 link 属性
    },
    data(){
      return{
        isActive:true
      }
    },
    methods: {
      itemClick() {
        this.$router.replace(this.path)              //路由跳转
      }
    }
  }
</script>
```

4. 使用路由

路由配置完成后，使用 router-view 进行渲染。在 App.vue 中使用设置过的路由，通过<router-view>组件渲染视图，router-view 主要是在构建单页应用时，方便渲染指定路由对应的组件。可以将 router-view 当做是一个容器，它渲染的组件是使用 vue-router 指定的。

```
<template>
  <div id="app">
    <router-view></router-view>
    <tab-bar>
      <tab-bar-item path="/home">
        <img slot="item-icon" src="~assets/img/tabbar/home.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/home_active.svg" alt="">
        <div slot="item-text">首页</div>
      </tab-bar-item>
      <tab-bar-item path="/category">
        <img slot="item-icon" src="~assets/img/tabbar/category.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/category_active.svg" alt="">
        <div slot="item-text">分类</div>
      </tab-bar-item>
      <tab-bar-item path="/cart">
        <img slot="item-icon" src="~assets/img/tabbar/shopcart.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/shopcart_active.svg" alt="">
        <div slot="item-text">购物车</div>
      </tab-bar-item>
      <tab-bar-item path="/profile">
        <img slot="item-icon" src="~assets/img/tabbar/profile.svg" alt="">
        <img slot="item-icon-active" src="~assets/img/tabbar/profile_active.svg" alt="">
        <div slot="item-text">我的</div>
      </tab-bar-item>
    </tab-bar>
  </div>
</template>
```

在终端输入以下命令 npm run serve 运行项目，单击页面下方不同的 Item，即可在页面上方显示 Item 对应的页面内容，项目在浏览器的运行效果如图所示。

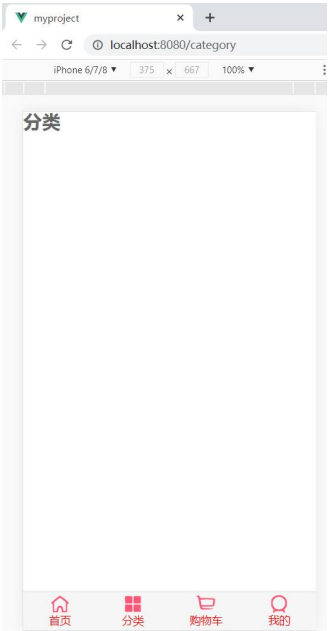


图 11-6 项目运行效果

5. 判断 Item 是否为 active 状态

由于在组件 TabBarItem.vue 中 Item 的活跃状态被设置为 True（isActive:true），因此上图页面底部的 Item 项均为亮色，这里需要动态获取 Item 的 active 状态。通过 this.$route.path.indexOf(this.link) !== -1 判断页面底端的 Item 是否为 active 状态，同时注释代码 isActive:true。TabBarItem.vue 中修改的代码如下，项目运行后单击页面底端不同的 Item 效果如图所示，被选中的 Item 为亮色显示。

```
<script>
  export default {
    name: "TabBarItem",
    props: {
      path: String
    },
    data(){
      return{
        //isActive:true
      }
    },
    computed: {
      isActive() {
        //如果当前 path 在路由映射表内，则返回 true（隐式返回），不在数组里，indexOf 会返回-1，这是 indexOf 函数
        return this.$route.path.indexOf(this.path) !== -1
      }
    },
    methods: {
      itemClick() {
        this.$router.replace(this.path)
      }
    }
  }
</script>
```

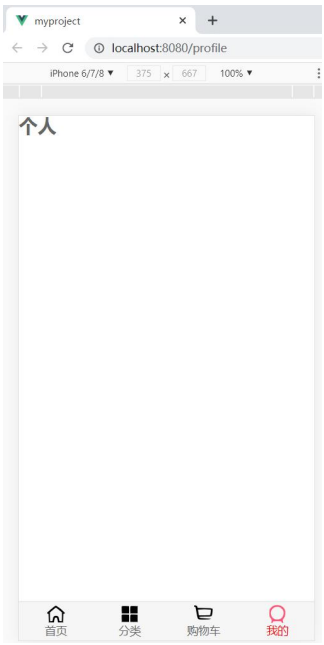


图 11-7 项目运行效果

6. 动态设置处于 active 状态的 Item 的样式

在 TabBarItem.vue 文件中设置 Item 颜色显示为红色。

```
.active{
      color: red;
  }
```

为了能够动态灵活的改变处于 active 状态的 Item 显示颜色，修改 TabBarItem.vue 文件封装新的计算属性，代码如下。

```
<template>
  <div class="tab-bar-item" @click="itemClick">
    <div v-if="!isActive"><slot name="item-icon"></slot></div>
    <div v-else><slot name="item-icon-active"></slot></div>
    <div :style="activeStyle"><slot name="item-text"></slot></div>
  </div>
</template>

<script>
  export default {
    name: "TabBarItem",
    props: {
      path: String,
      activeColor: {
        type: String,
        default: 'red'
      }
    },
    data(){
      return{
```

```
            //isActive:true
        }
      },
      computed: {
        isActive() {
          return this.$route.path.indexOf(this.path) !== -1
        },
        activeStyle() {
          //根据 isActive 计算属性的值，返回不同的 color 值
          return this.isActive ? {color: this.activeColor} : {}
        }
      },
      methods: {
        itemClick() {
          this.$router.replace(this.path)
        }
      }
    }
  </script>
```

同时在 App.vue 文件中的<tab-bar-item></tab-bar-item>中向组件 TabBarItem.vue 传入颜色参数，这样不需要修改组件 TabBarItem.vue 代码，可以在组件外设置传入组件的颜色参数。

```
  <template>
    <div id="app">
      <router-view></router-view>
      <tab-bar>
        <tab-bar-item path="/home" activeColor="deeppink">
          <img slot="item-icon" src="~assets/img/tabbar/home.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/home_active.svg" alt="">
          <div slot="item-text">首页</div>
        </tab-bar-item>
        <tab-bar-item path="/category" activeColor="deeppink">
          <img slot="item-icon" src="~assets/img/tabbar/category.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/category_active.svg" alt="">
          <div slot="item-text">分类</div>
        </tab-bar-item>
        <tab-bar-item path="/cart" activeColor="deeppink">
          <img slot="item-icon" src="~assets/img/tabbar/shopcart.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/shopcart_active.svg" alt="">
          <div slot="item-text">购物车</div>
        </tab-bar-item>
        <tab-bar-item path="/profile" activeColor="deeppink">
          <img slot="item-icon" src="~assets/img/tabbar/profile.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/profile_active.svg" alt="">
          <div slot="item-text">我的</div>
```

```
            </tab-bar-item>
          </tab-bar>
        </div>
    </template>
```

10.1.5 设计 MainTabBar 组件

App.vue 是项目的主组件，是页面的入口文件，因此不适合存放过多代码，本节将 App.vue 中用于向 TabBarItem 组件传入图片文字的语句再次抽取为一个组件。

项目 src→components 目录下两个目录 common 和 content，common 文件夹一般用于存放可以同时用于本项目以及其他项目的公共组件；content 文件夹一般用于存放与本项目业务相关的组件。因此将原 App.vue 中的代码进一步抽取为与本项目业务相关的组件，在 src→components→content→tabbar 目录下新建 MainTabBar.vue 组件。

将 App.vue 中部分内容抽取到 MainTabBar.vue 组件中，MainTabBar.vue 组件代码如下：

```
    <template>
      <tab-bar>
        <tab-bar-item path="/home" activeColor="blue">
          <img slot="item-icon" src="~assets/img/tabbar/home.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/home_active.svg" alt="">
          <div slot="item-text">首页</div>
        </tab-bar-item>
        <tab-bar-item path="/category" activeColor="blue">
          <img slot="item-icon" src="~assets/img/tabbar/category.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/category_active.svg"
alt="">
          <div slot="item-text">分类</div>
        </tab-bar-item>
        <tab-bar-item path="/cart" activeColor="blue">
          <img slot="item-icon" src="~assets/img/tabbar/shopcart.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/shopcart_active.svg"
alt="">
          <div slot="item-text">购物车</div>
        </tab-bar-item>
        <tab-bar-item path="/profile" activeColor="blue">
          <img slot="item-icon" src="~assets/img/tabbar/profile.svg" alt="">
          <img slot="item-icon-active" src="~assets/img/tabbar/profile_active.svg" alt="">
          <div slot="item-text">我的</div>
        </tab-bar-item>
      </tab-bar>
    </template>

    <script>
      //以下两条代码用于引入组件 TabBar 和 TabBarItem
      import TabBar from 'components/common/tabbar/TabBar'
      import TabBarItem from 'components/common/tabbar/TabBarItem'
```

```
    export default {
    name: "MainTabBar",
      components: {
        TabBar,
        TabBarItem
      }
    }
</script>
```

则 App.vue 中代码如下，项目运行效果与 3.1.4 节所示效果相同。

```
<template>
  <div id="app">
    <router-view></router-view>
    <main-tab-bar/>
  </div>
</template>

<script>
  import MainTabBar from 'components/content/tabbar/MainTabBar'
  export default {
    name: 'App',
    components: {
      MainTabBar
    }
  }
</script>
```

10.1.6 TabBar 组件设计小结

TabBar 组件设计方法如下，当然，也可以使用 ElementUI、mintUI 开源组件库。

1. 将首页底部导航封装为一个单独的 TabBar 组件。
2. 应用插槽显示 TabBar 中的内容，flex 布局平分 TabBar。
3. 自定义 TabBarItem，定义插槽传入图片、文字。
4. 传入高亮图片。

定义另外一个插槽，插入 active-icon 的数据

定义一个变量 isActive，通过 v-show 来决定是否显示对应的 icon

5. TabBarItem 绑定路由数据。

安装路由：npm install vue-router --save

设计 router/index.js 的内容，并创建对应的组件

在 main.js 中注册 router

在 APP.vue 中加入<router-view>组件

6. 点击 Item 跳转到对应路由，并且动态决定 isActive。

监听 Item 的点击，通过 this.$router.replace()替换路由路径

通过 this.$route.path.indexOf(this.link) !== -1 判断 Item 是否为 active

7. 动态计算 active 样式。

封装新的计算属性：this.isActive ? {'color': 'red'} : {}

10.2 封装首页导航栏

10.2.1 封装首页导航栏组件

1. 设计 NavBar.vue 组件

由于导航栏可用于多个项目，因此本文将导航栏封装为一个组件。在 src→components→common 路径下新建 navbar→NavBar.vue 组件，由于项目中有多个导航，因此在 NavBar.vue 中设置为类 class，navbar→NavBar.vue 组件内代码如下。

```
<template>
  <div class="nav-bar ignore">
    <div class="left"><slot name="left"></slot></div>
    <div class="center"><slot name="center"></slot></div>
    <div class="right"><slot name="right"></slot></div>
  </div>
</template>

<script>
  export default {
    name: "NavBar"
  }
</script>

<style scoped>
  .nav-bar {
    display: flex;
    height: 44px;
    line-height: 44px;
    text-align: center;
    box-shadow: 0 1px 1px rgba(100,100,100,.1);//设置 box 导航栏底部效果
  }
  .left {
    width: 60px;
  }
  .right {
    width: 60px;
  }
  .center {
    flex: 1;
  }
</style>
```

2. 设计 Home.Vue 组件

将上一步设计好的导航栏 NavBar.vue 组件导入到项目首页 src→views→home→Home.vue 组件，首先输入以下代码。

```
<template>
```

```
    <div id="home">
      <nav-bar><div slot="center">购物街</div></nav-bar>
    </div>
  </template>
```

以下加粗斜体代码为平台自动添加组件的导入，由于 2.3.3 节已经配置过路径别名，因此 import NavBar from "../../components/common/navbar/NavBar";可以改为

import NavBar from "components/common/navbar/NavBar";

```
  <script>
    import NavBar from "../../components/common/navbar/NavBar";
    export default {
      name: "Home",
      components: {NavBar}
    }
  </script>
  <style scoped>
  </style>
```

以上项目运行效果如图所示。

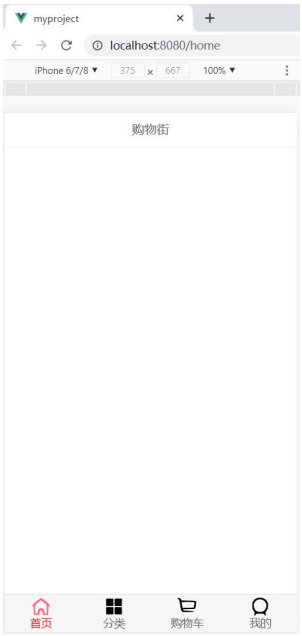


图 11-8 项目首页运行效果

3. 设置导航栏背景色

由于导航栏颜色在不同的应用位置下背景颜色可能不同，因此不适合在导航栏组件内部设置背景颜色，即不适合将背景颜色封装到导航栏组件中，而是在 src→views→home→Home.vue 组件中设置导航栏背景色，Home.vue 中新增代码如下，项目运行后则导航栏显示背景色。

```
<template>
  <div id="home">
    <nav-bar class="home-nav"><div slot="center">购物街</div></nav-bar>
  </div>
```

```
</template>
<style scoped>
  .home-nav {
    background-color: var(--color-tint);//color-tint 是 src→assets→css→base.css 中定义的变量
    color: #fff;
  }
</style>
```

10.2.2 首页轮播图展示

vue 有许多开源组件可供使用，首页轮播图的展示可以直接使用 ElementUI、mintUI（基于 Vue.js 的移动端组件库）、VantUI、vue-awesome-swiper 等开源轮播图组件，下面以 vue-awesome-swiper 为例，使用轮播图组件在项目首页显示轮播图。

1. 安装 vue-awesome-swiper 组件

vue-awesome-swiper 是一个开源的轮播图组件，其 Github 地址及官网如下：

https://github.com/surmon-china/vue-awesome-swiper

https://www.swiper.com.cn/api/index.html

选用 npm 方式安装 swiper 和 awesome-swiper 组件，注意：安装 awesome-swiper 前需要安装 swiper 插件，代码如下。安装完成后，可以在项目的 node_modules 目录下找到已安装的两个组件。

```
npm install swiper --save
npm install vue-awesome-swiper@2.6.7 --save
```

在 package.json 文件中，增加了以下两个依赖。

```
"dependencies": {
    "axios": "^0.21.1",
    "core-js": "^3.6.5",
    "swiper": "^6.4.9",
    "vue": "^2.6.11",
    "vue-awesome-swiper": "^2.6.7",
    "vue-router": "^3.4.9"
  },
```

2. 引入 awesome-swiper 组件

可以使用全局引入和在组件中引入两种方式，如果项目中只有一个轮播图可以在组件中引入，若项目中有多个轮播则需在全局中引入。若使用全局引入方式，则在 src→main.js 中添加以下代码引入 awesome-swiper 组件。

```
import VueAwesomeSwiper from 'vue-awesome-swiper'
import 'swiper/swiper-bundle.css'   //注意：本语句适用于 swiper6.x 以上版本
Vue.use(VueAwesomeSwiper)
```

3. 应用 awesome-swiper 组件

新建 src→components→common→swiper-awesome→Swiper.vue 组件，同时在 src→assets→img→swiper 路径下置入三个测试轮播图片 swiper-01.jpg、swiper-02.jpg 和 swiper-03.jpg，Swiper.vue 文件代码如下。

```
<template>
  <div>
    <swiper :options="swiperOptions">
```

```
            <swiper-slide class="swiper-item" :key="item.id" v-for="item in swiperList">
              <img class="swiper-img" :src="item.imgUrl">
            </swiper-slide>
            <!-- Optional controls, 显示圆点-→
            <div class="swiper-pagination" slot="pagination"></div>
            //<div class="swiper-button-prev" slot="button-prev"></div>
            //<div class="swiper-button-next" slot="button-next"></div>
            <div class="swiper-scrollbar" slot="scrollbar"></div>
        </swiper>
    </div>
</template>
<script>
    export default {
        name: "HomeSwiper",
        data(){
            return{
                swiperOptions:{
                    pagination:'.swiper-pagination', //分页器的类名
                    loop:true,          //  设置图片循环
                    autoplay:2000,   //设置可自动播放,每隔 2 秒滑动一次
                    //effect:'cube'     //设置不同图片切换样式
                },
                swiperList:[
                    {
                        id:'01',
                        imgUrl:require('assets/img/swiper/swiper-01.jpg')
                    },
                    {
                        id:'02',
                        imgUrl:require('assets/img/swiper/swiper-02.jpg')
                    },
                    {
                        id:'03',
                        imgUrl:require('assets/img/swiper/swiper-03.jpg')
                    }
                ]
            }
        }
}
</script>
<style scoped>
    .wrapper{
        overflow: hidden;
        width: 100%;
```

```
        padding-bottom: 26.7%;
    }
    .swiper-item{
        width: 100%;
        height: 170px;
    }
    .swiper-img{
        width: 100%;
    }
</style>
```

4. 展示轮播图

在首页 src→views→home→Home.vue 组件中显示轮播图，导入 awesome-swiper 组件，代码如下，项目运行效果如图所示。

```
<template>
    <div id="home">
        <nav-bar class="home-nav"><div slot="center">购物街</div></nav-bar>
        <home-swiper></home-swiper>
    </div>
</template>
<script>
    import NavBar from "components/common/navbar/NavBar"
    import HomeSwiper from 'components/common/swiper-awesome/Swiper'
    export default {
        name: "Home",
        components: {
            NavBar,
            HomeSwiper
        }
    }
</script>
```

图 11-9 首页轮播图运行效果

10.2.3 网络模块 axios 应用

在 3.2.2 节中轮播图中的图片数据来自于项目文件中的静态文件，但是实际开发中，数据应该来自于项目后台。在项目前端开发过程中还可以使用另外两种方法获取后台数据，一种是使用网络模块 axios 从免费提供网络请求模拟网站中获取数据，另外一种方法是自行搭建后台项目，利用 Postman 接口测试工具进行测试，本节以第一种方法为例。

axios：ajax i/o system，是用于在 Vue.js 中发送网络请求的第三方框架。虽然可通过许多方式发送网络请求，例如传统的 Ajax、jQuery-Ajax 等，但是由于配置和调用方式等较复杂，因此一般使用 Vue.js 框架作者推荐的 axios，使用方便。

axios 是一个基于 Promise 的发送 HTTP 请求的工具库，可以发送 get、post 请求。axios 框架的特点主要有：在浏览器中发送 XMLHttpRequests 请求；在 node.js 中发送 http 请求；支持 Promise API；拦截请求和响应；转换请求和响应数据；自动转换 JSON 数据；客户端支持保护安全免受 XSRF 攻击。axios 支持的请求方式主要有以下几种：

```
axios(config)
axios.request(config)
axios.get(url[, config])
axios.delete(url[, config])
axios.head(url[, config])
axios.post(url[, data[, config]])
axios.put(url[, data[, config]])
axios.patch(url[, data[, config]])
```

1. 安装网络请求模块 axios

由于 Vue 运行时依赖 axios，所以采用‘--save’的安装方式，安装 axios 语句如下。

```
npm install axios --save
```

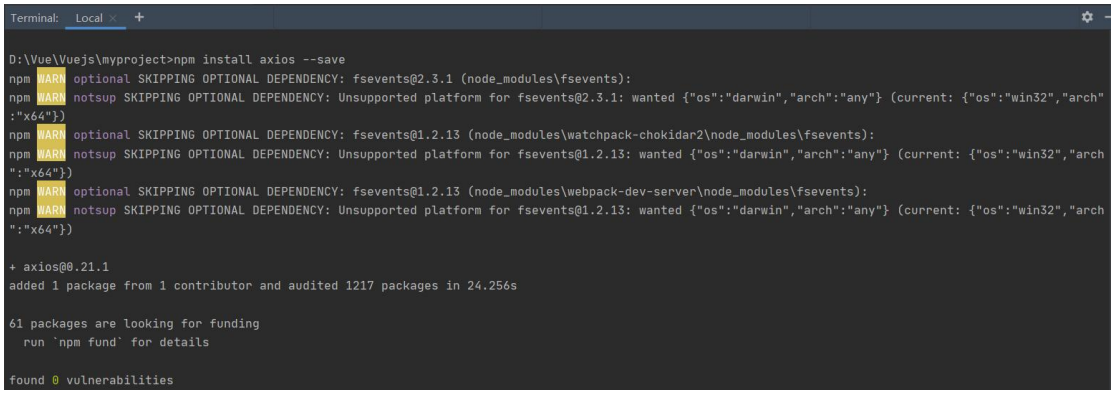


图 11-10 成功安装网络请求 axios 提示

2. axios 的基本使用

由于企业级软件一般是多人开发，由于接口之间互相依赖，若依赖的服务进度缓慢或没有在环境中运行，则无法进行功能测试，进而不能及时交付项目。同样，对于 WEB 前端开发者，当后端 API 开发缓慢时，若需要测试应用展示效果，使用一个稳定的测试接口，就不必等后端进度模拟后端 API。可以使用提供网络请求模拟网站，例如：

https://github.com/Runscope/httpbin

https://github.com/dreamhead/moco

http://jsonplaceholder.typicode.com/

http://123.207.32.32:8000/home/multidata

https://github.com/eolinker 等，测试 HTTP 请求和响应的各种信息，比如 cookie、ip、headers 和登录验证等，且支持 GET、POST 等多种方法。也可以本地搭建 MySQL 数据库，使用 Postman 测试后台项目接口，例如本地服务器地址 http://127.0.0.1/mydata/multidata。

当使用多个服务器时，如果只抽取一个地址（地址值可能对于一台服务器），可能在其他请求中请求的是另外一个服务器的地址（对应另一台服务器），那么这个抽取的地址就不适应对另外一台服务器的请求。在这种情况下，使用 axios 时就不会使用全局设置，而是单独地创建一个 axios 实例（即在局部配置 axios）。

3. axios 的封装

由于项目的多个页面需要网络请求并导入框架，如果网络请求框架发生变化，修改多个页面则项目耦合度过高，因此将网络请求封装到一个文件。一般在 src→network 路径下创建 request.js 文件用于存放封装的网络配置文件，代码如下。

```
import axios from 'axios'
export function request(config) {
  // 1.创建 axios 的实例
  const instance = axios.create({
    //baseURL 会在发送请求的时候拼接在 url 参数的前面
    baseURL: 'http://123.207.32.32:8000',   //以此地址为例
    timeout: 5000
  })
  // 2.axios 的拦截器：
  //请求拦截，所有的网络请求会先访问这个方法，在其中为请求添加自定义的内容
  instance.interceptors.request.use(config => {
    return config
  }, err => {
    // console.log(err);
  })
  // 3.响应拦截
  instance.interceptors.response.use(res => {
    return res.data
  }, err => {
    console.log(err);
  })
  // 4.发送真正的网络请求
  return instance(config)
}
```

4. 封装针对首页的网络请求

将获取的轮播图数据展示到首页 Home.vue。新建 src→network→home.js 文件，封装针对首页所有的网络请求，即可以对首页的网络请求进行统一的管理，代码如下，即 Home.vue 针对 home.js 开发。

```
import {request} from "./request";
export function getHomeMultidata() {
  return request({
    url: '/home/multidata'
```

```
    })
  }
```

5. 设计 Home.vue 组件进行网络请求

将封装好的轮播图组件 Swiper.vue 和 SwiperItem.vue 置于新建路径下 src→components→common→swiper，同时在该路径下新建 index.js 文件管理两个轮播图组件，index.js 文件代码如下。

```
import Swiper from './Swiper'
import SwiperItem from './SwiperItem'
export {
  Swiper, SwiperItem
}
```

在 src→views→home→Home.vue 中导入 home.js，并使用 created()发送网络数据请求，Home.vue 中新增代码如下，首页经过网络请求展示轮播图。

```
<template>
  <div id="home">
    <nav-bar class="home-nav"><div slot="center">购物街</div></nav-bar>
    <swiper>
      <swiper-item v-for="item in banners">
        <a :href="item.link">
          <img :src="item.image" alt="">
        </a>
      </swiper-item>
    </swiper>
  </div>
</template>
<script>
  import NavBar from "components/common/navbar/NavBar"
  import {Swiper,SwiperItem} from 'components/common/swiper'
  import {getHomeMultidata} from "network/home"
  export default {
    name: "Home",
    components: {
      NavBar,
      Swiper,
      SwiperItem
    },
    data(){   //保存 created 返回的数据
      return{
        banners:[],
        recommends:[]
      }
    },
    created() {//发送网络请求
      getHomeMultidata().then(res => {    //请求多个数据，获得返回值
```

```
                this.banners = res.data.banner.list;
                this.recommends = res.data.recommend.list;
            })
        }
    }
</script>
```

10.3 项目登录功能实现

10.3.1 配置后台项目环境

在 3.2.3 节中，项目使用使用网络模块 axios 从免费提供网络请求模拟网站中获取数据，但是该方法依赖于提供服务网站的稳定性，同时不够灵活，本节使用本书提供模拟的后台项目 vue-server、MySQL 数据库以及 Postman 接口测试工具。

1. 安装 MySQL 数据库

自选一种安装 MySQL 数据库的方法，例如，打开 phpStudy 官网 https://www.xp.cn/download.html，下载并安装 phpStudy v8.1 应用程序，仅启动其中的 MySQL 服务，并导入素材中的数据库文件 hyitdb.sql，如图所示。导入数据库后，在 phpStudy 的安装路径下 D:\phpstudy_pro\Extensions\MySQL5.7.26\data\hyitdb 为导入的数据库内容。

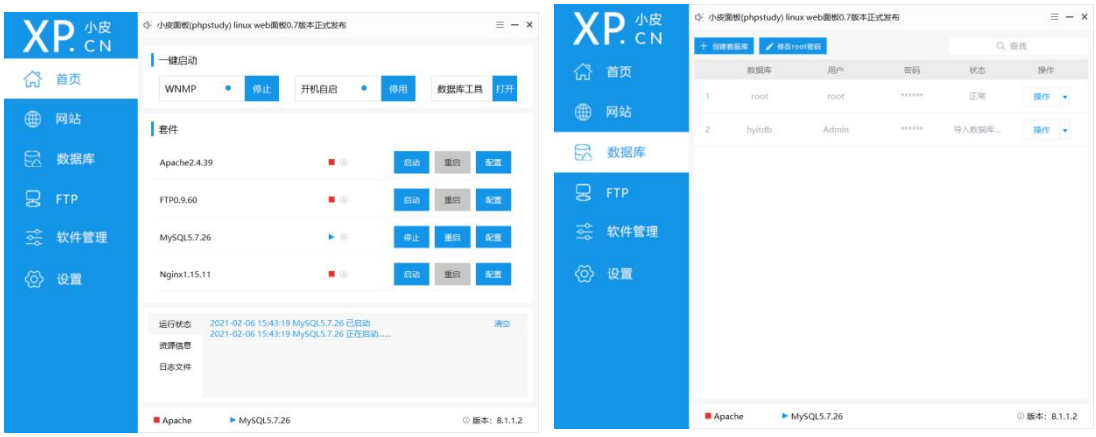


图 11-11 导入后台数据库 hyitdb

2. 运行后台项目

Node.js 环境安装成功之后，首先打开 Windows PowerShell 窗口，使用 Windows PowerShell 窗口切换到后台项目 vue-server 目录并打开该项目，在终端中输入命令安装项目运行所需的依赖包：`npm install`，同时在 PowerShell 窗口中输入命令 `node .\app.js` 启动后台项目。

3. 测试后台项目接口

接口测试就是通过测试不同情况下的入参与之相应的出参信息，判断接口是否符合或满足相应的功能性、安全性要求。接口测试工具很多，例如 postman、RESTClient、jmeter、loadrunner、SoapUI 等。Postman 是谷歌的一款接口测试插件，使用简单，支持用例管理，支持 get、post、文件上传、响应验证、变量管理、环境参数管理等功能，可以批量运行，并支持用例导出、导入。jmeter 是 Java 编写的免费开源的工具，它主要用来做性能测试，相比于 loadrunner，内存占用小，免费开源，轻巧方便、无需安装。

本文以 Postman 测试工具为例，使用 Postman 测试后台项目接口是否正常。使用素材中提供的文件安装 Postman 工具，在下图中输入 POST 请求地址

“http://127.0.0.1:8888/api/private/v1/login”，同时在“KEY”和“VALUE”中分别输入如图所示的字段和值，用于测试项目接口是否正常，若接口测试如下图所示，则表示项目接口运行正常。

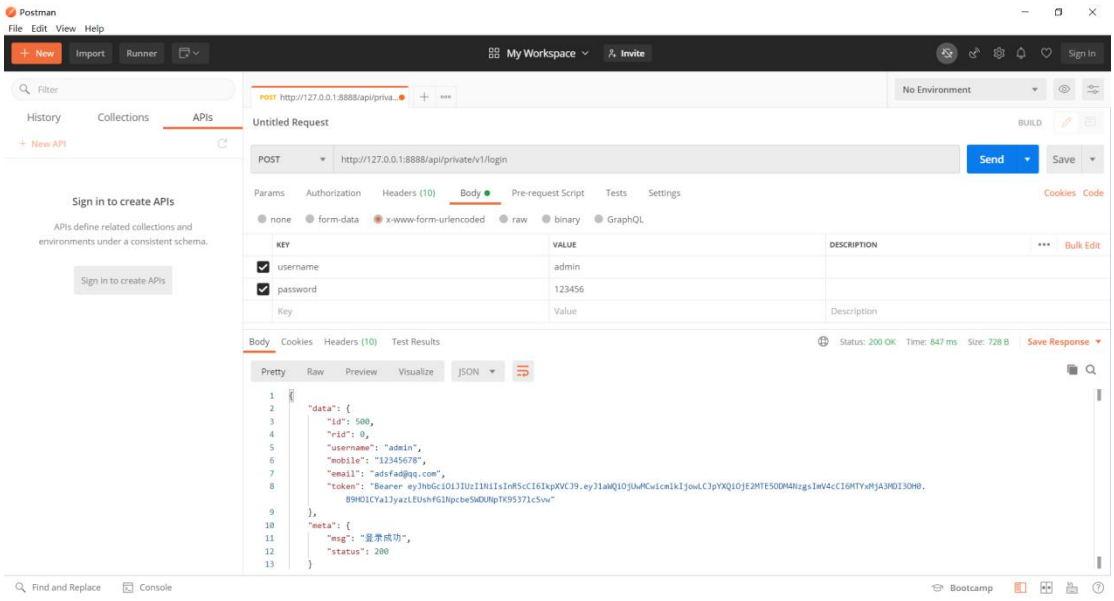


图 11-12 运行 Postman 测试项目接口

10.3.2 用户登录

实现用户登录的业务流程如下：在登录页面输入用户名和密码；调用后台接口进行验证；通过验证之后，根据后台的响应状态跳转到项目主页。以前端设计框架 Element-ui 为例设计项目登录功能。

1. 设计登录表单

在 Form 组件中，每一个表单域由一个 Form-Item 组件构成，表单域中可以放置各种类型的表单控件，包括 Input、Select、Checkbox、Radio、Switch、DatePicker、TimePicker 等。

项目运行后，单击页面底端最右侧的“我的”，若项目为未登录状态则显示登录界面，使用 ElementUI 设计登录界面，参考 ElementUI 官网中组件文档设计登录界面，src→views→profile→Profile.vue 组件中代码如下，项目运行如图所示。使用 type、plain、round 和 circle 属性定义 Button 的样式。

```
<template>
  <div class="login_box">
    <h2>用户登录</h2>
    <el-form label-width="0px" class="login_form">
      <!--用户名-->
      <el-form-item>
        <el-input></el-input>
      </el-form-item>
      <!--密码-->
      <el-form-item>
        <el-input></el-input>
      </el-form-item>
      <!--按钮区域-->
```

```
            <el-form-item class="btns">
                <el-button type="primary">登录</el-button>
                <el-button type="info">重置</el-button>
            </el-form-item>
        </el-form>
    </div>
</template>
<style scoped>
    .login_form{
        position: absolute;
        bottom: 50%;
        width: 100%;
        padding: 0 20px;
        box-sizing: border-box;
    }
    .btns{
        display: flex;
        justify-content: center;
    }
</style>
```

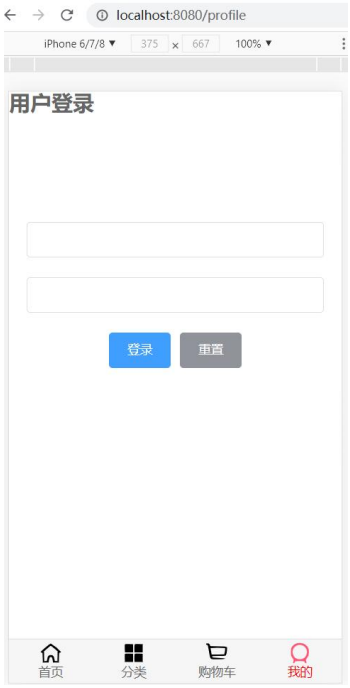


图 11-13 项目登录界面

2. 设计输入框图标

有两种方法在输入框中设置图标，可以通过 prefix-icon 和 suffix-icon 属性在 input 组件首部和尾部增加显示图标，也可以通过 slot 插槽放置图标。参考 ElementUI 文档使用属性方法设置输入框图标，src→views→profile→Profile.vue 组件中新增代码如下，项目运行如图所示。

```
    <!--用户名-->
```

```
<el-form-item>
  <el-input prefix-icon="el-icon-user-solid"></el-input>
</el-form-item>
<!--密码-->
<el-form-item>
  <el-input prefix-icon="el-icon-lock" type="password"></el-input>
</el-form-item>
```

图 11-14 输入框图标效果

3. 表单数据绑定

在表单中 el-form 标记中的:model="form"表示数据绑定，用户在表单中填入的数据都会自动同步到:model 对象中，数据对象在组件的行为区 script 中定义，为每一个表单项通过 v-model 绑定到数据对象对应的属性中，Profile.Vue 中新增代码如下，运行效果如图所示。

```
<el-form :model="loginForm" label-width="0px" class="login_form">
  <!--用户名-->
  <el-form-item>
    <el-input v-model="loginForm.username" prefix-icon="el-icon-user-solid">
    </el-input>
  </el-form-item>
  <!--密码-->
  <el-form-item>
   <el-input v-model="loginForm.password" prefix-icon="el-icon-lock" type="password">
    </el-input>
  </el-form-item>
  <!--按钮区域-->
  <el-form-item class="btns">
    <el-button type="primary">登录</el-button>
    <el-button type="info">重置</el-button>
  </el-form-item>
</el-form>

<script>
  export default {
    name: "Profile",
    data() {
      return{
        //这是登录表单的数据绑定对象
        loginForm:{
          username: 'Tom',
```

```
                password: '123456'
            }
        }
    }
</script>
```

图 11-15 表单数据绑定效果

4. 表单验证设计

表单验证是在防止用户犯错的前提下，尽可能让用户更早地发现并纠正错误。Form 组件提供了表单验证的功能，只需要通过 rules 属性传入约定的验证规则，并将 Form-Item 的 prop 属性设置为需校验的字段名即可。

设计表单验证的方法主要是首先为 el-form 通过属性绑定指定一个 rules 校验对象，在 data()中定义校验对象，其中每一个属性都是一个验证规则，最后为不同的表单 Item 项通过 prop 指定不同的验证规则进行表单的验证，Profile.Vue 中新增代码如下。

```
<el-form :model="loginForm" :rules="loginFormRules" label-width="0px"
class="login_form">
  <!--用户名-->
  <el-form-item prop="username">
    <el-input v-model="loginForm.username" prefix-icon="el-icon-user-solid"></el-input>
  </el-form-item>
  <!--密码-->
  <el-form-item prop="password">
    <el-input v-model="loginForm.password" prefix-icon="el-icon-lock" type="password">
    </el-input>
  </el-form-item>
</el-form>

data() {
      return{
        //这是登录表单的数据绑定对象
        loginForm:{
          username: 'Tom',
          password: '123456'
        },
        //这是表单的验证规则对象
        loginFormRules:{
          //验证用户名是否合法
          username:[
            { required: true, message: '请输入登录用户名', trigger: 'blur' },
```

```
            { min: 3, max: 10, message: '长度在 3 到 10 个字符', trigger: 'blur' }
          ],
          //验证密码是否合法
          password:[
            { required: true, message: '请输入登录密码', trigger: 'blur' },
            { min: 6, max: 15, message: '长度在 6 到 15 个字符', trigger: 'blur' }
          ]
        }
      }
    }
```

5. 重置表单

ElementUI 的 Form 组件提供了一些方法，其中 resetFields 方法可以对整个表单进行重置，将所有字段值重置为初始值并移除校验结果，重置表单验证结果，为重置按钮绑定事件"resetLoginForm"，Profile.Vue 中新增代码如下。

```
<el-form ref="loginFormRef" :model="loginForm" :rules="loginFormRules" label-width="0px" class="login_form">
  <!--用户名-->
  <el-form-item prop="username">
    <el-input v-model="loginForm.username" prefix-icon="el-icon-user-solid"></el-input>
  </el-form-item>
  <!--密码-->
  <el-form-item prop="password">
    <el-input v-model="loginForm.password" prefix-icon="el-icon-lock" type="password"></el-input>
  </el-form-item>
  <!--按钮区域-->
  <el-form-item class="btns">
    <el-button type="primary">登录</el-button>
    <el-button type="info" @click="resetLoginForm">重置</el-button>
  </el-form-item>
 </el-form>
```

```
    methods:{
        //重置按钮，重置登录表单
        resetLoginForm() {
          this.$refs.loginFormRef.resetFields();
        }
      }
```

6. 预验证登录组件

ElementUI 的 Form 组件提供的 validate 方法对整个表单进行校验的方法，参数为一个回调函数。该回调函数会在校验结束后被调用，并传入两个参数：是否校验成功和未通过校验的字段。若不传入回调函数，则会返回一个 promise，参数如下：

```
Function(callback: Function(boolean, object))
```

为登录按钮绑定事件"login"，Profile.Vue 中新增代码如下。

```
<el-form-item class="btns">
  <el-button type="primary" @click="login">登录</el-button>
  <el-button type="info" @click="resetLoginForm">重置</el-button>
</el-form-item>
```

使用 validate 方法对整个表单进行校验，弹出提示消息的方法为 ElementUI 组件提供的 Message 消息提示，常用于主动操作后的反馈提示，Element 注册了一个$message 方法用于调用，Message 可以接收一个字符串或一个 VNode 作为参数，它会被显示为正文内容。

若用户名和密码与数据库中一致，登录成功后跳转到主页/home 页面，将 token 保存在 sessionStorage（回话存储对象）中，token 只在当前网站打开期间生效。通过 token 可以验证用户身份，判断用户是否可以访问除登录之外的其他 API 接口，因此将 token 保存到客户端中，一般过程如下：使用用户名和密码请求服务器→服务器验证用户信息→服务器通过验证发送一个 token 给用户→客户端存储 token，并在每次请求时发送这个 token 值→服务端验证 token 值，并返回数据，登录的 token 值可以在浏览器的 Application 下查看，如图所示。

Profile.Vue 中新增代码如下。

```
methods:{
        //重置按钮，重置登录表单
        resetLoginForm() {
          this.$refs.loginFormRef.resetFields();
        },
        login() {
          this.$refs.loginFormRef.validate(async valid => {
            if (!valid) return;
            const {data: res}=await this.$http.post("login", this.loginForm);
            if (res.meta.status!==200) return this.$message.error("登录失败");
            this.$message.success("登录成功");
            window.sessionStorage.setItem("token",res.data.token);
            this.$router.push("/home");
          });
        }
      }
```

同时在 src→main.js 中配置网络请求 axios，使用 baseURL 配置网络请求的根路径，项目运行之前确定已经启动 MySQL 数据库以及后台项目 vue-server，src→main.js 中新增网络请求配置的代码如下。项目运行后，若用户名和密码与数据库中一致，则弹出“登录成功”消息提示，否则显示“登录失败”，如图所示。

```
import axios from 'axios'
//配置请求的根路径
axios.defaults.baseURL='http://127.0.0.1:8888/api/private/v1/'
Vue.prototype.$http=axios
```

图 11-16 登录成功或失败消息提示

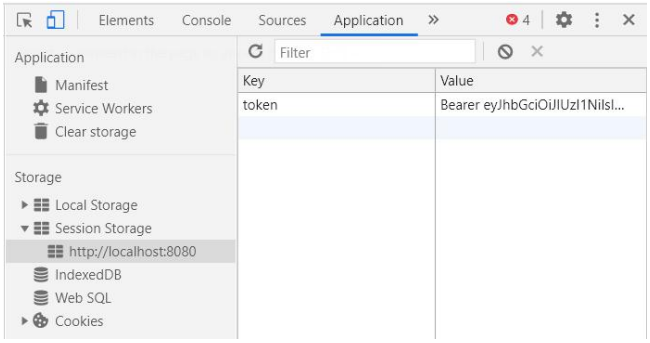


图 11-17 登录的 token 值

7. 退出登录

使用 clear 方法清空本地的 token 即可实现基于 token 的方式的退出，若需要访问页面必须重新登录生成一个新的 token，并重新跳转到登录页面。主页退出按钮代码如下：

```
methods: {
        logout(){
          window.sessionStorage.clear()
          this.$router.push('./profile')
        }
      }
```