

第 1 章

组件化开发

参与过网站开发的人都知道，网站需要不断进行迭代和维护。随着网站功能的增加，项目的维护成本也随之增加。某些功能或UI的展示会在网站不同页面或不同位置重复出现，开发人员为了减少开发成本就要实现功能复用。在网站的开发中，每一个功能都是一个模块，一个模块由一个或几个组件组成。将每个模块拆分为组件是现代网站开发的必然趋势。

本章主要涉及的知识点有：

- 组件化开发的基本概念
- 使用组件化开发的原因和背景

1.1 什么是组件化开发

随着前端框架的出现，单页面应用的概念应运而生。前端的开发模式已经发生了翻天覆地的变化，不再是传统的由多个网页组成一个网站的架构模式。单页面应用中一个很重要的思想就是组件化开发，了解组件化开发是用好前端框架的前提。

1.1.1 多页应用

传统的网站开发是多页面的架构模式，就是一个网站由多个HTML页面组成。例如一个电商网站，一般由首页、产品列表页面、产品详情页面、购物车页面、结算页面等组成。每个页面对应一个HTML文件、若干个CSS文件以及若干个JavaScript文件。

项目的目录结构一般是把HTML文件放在最外层，把CSS文件、JavaScript文件放在单独的目录中。

数据的填充有两种方式，第一种是后端工程师将HTML页面进行改造，借助某种模板技术（如JSP、ASP、PHP）在服务器端动态生成HTML页面。这种开发模式下，前端代码和后端代码混在一起，网页的每次改动都需要前后端人员共同参与，增加了项目的沟通成本和协调成本。示例代码如下：

```
<table width="50%" class="border">
  <tr>
    <td height="25" align="right" class="tabletd1">name</td>
    <td height="25" class="tabletd1">
      <select name="students">
        <option value="">--please select--</option>
        <%
          ArrayList students = array.getXuesheng();
          for(int i = 0;i < students.size();i++) {
            ArrayList alRow = (ArrayList) students.get(i);
            if (client == null || client.size() == 0) {
              <%
                <option value="<%=alRow.get(6) %>"><%=alRow.get(6) %></option>
              <% } else { <%
                <option value="<%=alRow.get(5) %>"><%=alRow.get(5) %></option>
              <% } } <%
            }
          </select>
        </td>
      </tr>
    </table>
```

在上面这段代码中，Java工程师不得不面对前端的HTML代码，前端工程师也不得不面对页面上的`<%@ %>`、`<jsp>`等JSP语法，前后端耦合度太高，开发和维护起来都非常麻烦。

为了使前后端代码彻底分开，出现了另一种开发模式。前端工程师把其中的HTML文件改成`ejs`或`jade`等模板文件，通过`ajax`请求接口，并将接口返回的数据填充到页面中。在这种模式中，后端只提供数据，前端负责整个页面的模板渲染、数据填充以及交互逻辑，其本质是将模板文件和数据通过模板引擎生成最终的HTML。示例代码如下：

```
<body>
  <script>
    let users = ['geddy', 'neil', 'alex']
  </script>
  <ul>
    <% for(var i = 0; i < users.length; i++) { %>
      <% var user = users[i]; %>
      <li><%= user %></li>、
    <% } %>
  </ul>
</body>
```

以上两种开发模式都是早期多页面开发的数据填充方式。多页面模式的缺点是当用户单击一个链接并切换页面时，要刷新整个网页，也就是说浏览器需要重新从网络服务器请求HTML文档，并且下载页面相关的CSS和JavaScript静态资源。如果网站的静态资源下载比较慢，则用户体验会大打折扣。

1.1.2 单页应用

目前大部分的React项目都是单页应用，即整个网站或系统只有一个HTML文件作为容器，向用户展示的内容实际通过JavaScript进行填充，页面之间的跳转使用`history`实现。

单页应用可以带来更快的用户体验，因为浏览器不需要请求一个全新的HTML文档，也不需要为下一页重新下载CSS和JavaScript静态资源。它还可以通过动画等方式实现更动态的用户体验。

单页应用项目的目录结构一般如下所示。

```
|--public
  |--index.html
|--src
  |--page
    |--welcome.js
    |--goods.js
  |--component
    |--nav
      |--index.js
      |--index.css
  |--App.js
  |-- ...
|--node_modules
```

每个UI界面都是一个容器组件，例如src/page/welcome.js。每个容器组件由多个子组件组成，例如src/component/nav。每个组件都包含一个JS文件和一个CSS文件，JS文件负责UI渲染和逻辑交互，CSS文件负责组件的样式。然后通过打包工具将所有组件的JS文件和CSS文件打包为一个总的JS和CSS文件，例如上面结构中的App.js。最后通过webpack插件自动将总的JS和CSS文件放置于HTML文件中。在单页应用中，为每个页面分配一个路由，当路由切换时，渲染路由对应的组件。

1.1.3 组件化的概念

在单页应用中组件的概念至关重要。组件化是一种软件开发方法，首先，将软件分解为可识别和可重用的部分，使得应用程序开发人员可以独立构建和部署这些部分；然后，通过某些标准将这些组件拼接在一起。

对于什么是组件，什么不是组件，React没有任何硬性规定。一般来说，如果应用中的某部分是一个明显的“区块”，或者某个功能经常被重用，那么它可能是一个组件。

在单页应用中，经常把一些常见的UI元素制作成组件，以便可以在多个位置重复使用。如果该组件的样式或交互需要修改，那么只需修改一次，在使用该组件的任何地方就都可以看到更改。

1.2 为什么要用组件化开发

组件化的目的的一方面是让应用中的各个部分可以被复用，以减少重复的代码，提高项目可维护性；另一方面是可以更好地使团队分工协作，让不同的人负责编写不同的组件，提高工作效率。

1.2.1 前后端分离思想

前后端分离就是一个系统的前端代码和后端代码分开编写。

它是当软件技术和业务发展到一定程度时，在项目管理工作上必须进行的一种升级，是公司部门架构的一种调整。它是一种必然而不是偶然。

初期的网站应用其实是侧重于后端的，因为互联网初期的页面功能比较简单，只需要进行数据的展示、提供基本的操作就可以了，整个项目的重点放在后台的业务逻辑处理上。但是随着业务和技术的发展，前端功能越来越复杂，也变得越来越重要，同时前端的技术栈越来越丰富。因此，在开发中遇到的问题就越来越多，解决这些问题的难度就越来越大。这时我们发现前端开发不能像以前那样零散地分布在整个系统架构当中了，前端也应该像后端那样实现工程化、模块化、系统化。

如何做到这一点呢？

将前端开发从之前的前后端混合在一起的组织架构当中分离出来，专门去研究开发工程化的前端技术，迭代升级新的技术体系，以解决项目中的问题，适应技术的发展。

1.2.2 组件复用

当一个系统复杂到一定程度时，会产生很多页面，这些页面中有一部分模块可能具有相同的样式或者相同的交互。因此，需要将这些相同的部分提取出来形成一个一个的组件，从而可以在不同位置重复引用，达到复用的目的。组件复用可以提高开发效率，并且易于后期维护。

1.3 计算机选购配置系统

本节主要介绍计算机选购配置系统，它是一个企业级项目，前端使用的是React技术栈，后端使用的是基于Node.js的Express框架。本书讲解的所有知识点都围绕这个系统展开。

1.3.1 系统介绍

计算机选购配置系统是一个企业级的软件系统，是一个在线选购计算机或服务器的工具系统。这个系统的主要目的是根据用户的选择，从数据库中筛选出符合条件的性价比和配置最优的个人计算机或服务器供用户选购。

用户的选择包括所在市场/区域、国家、语言、计算机上要安装的软件 and 应用程序。

整个项目由前端React工程和后端Node工程两部分组成。

- 前端React工程中，组件的编写使用了React、Redux、React Redux、Hooks，路由使用了react-router，项目构建使用了create-react-app和webpack。
- 后端Node工程中，接口的编写使用了Express，项目搭建使用了express-generator。

1.3.2 系统 UI 界面

计算机选购配置系统的UI界面包含登录页面、注册页面和内容页面。登录页面如图1.1所示。



图 1.1 计算机选购配置系统的登录页面

注册页面如图1.2所示。



图 1.2 计算机选购配置系统的注册页面

内容页面中有两个界面UI：初始界面和产品列表界面。内容页面的初始界面如图1.3所示。

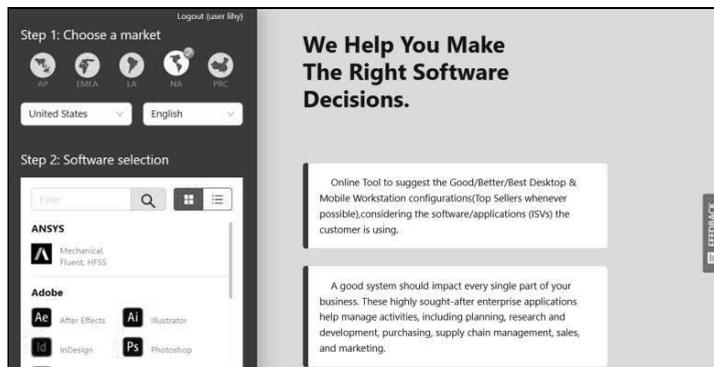


图 1.3 计算机选购配置系统的内容页面初始界面

内容页面的产品列表界面如图1.4所示。

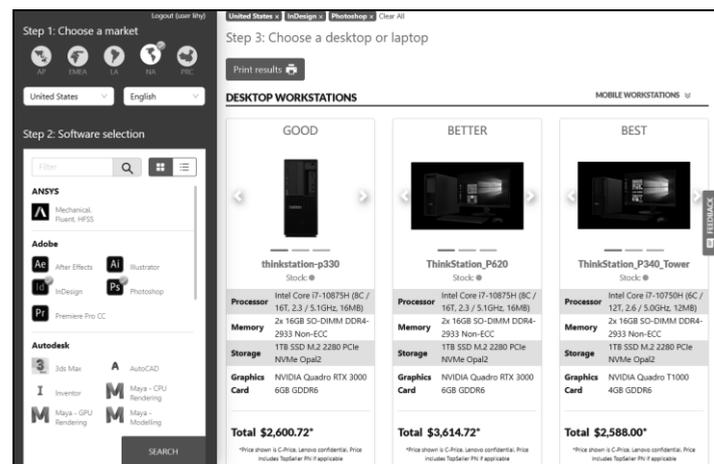


图 1.4 计算机选购配置系统中，搜索产品后的内容页面

1.3.3 登录页面和模块介绍

登录页面包括欢迎提示、登录表单和注册按钮，如图1.5所示。

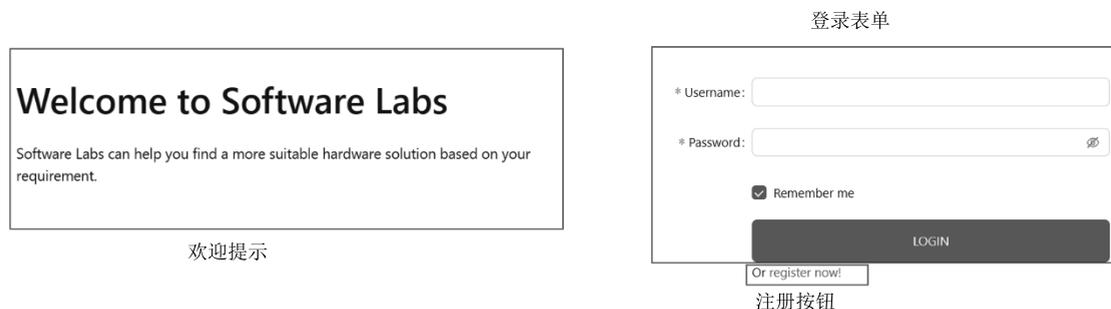


图1.5 登录页面模块划分

欢迎提示中的是固定文案，登录表单包含用户名、密码、记住我和登录按钮。当用户输入表单内容后，单击登录按钮，前端向后端发送ajax请求，调用后端接口，调用成功则表示登录成功，跳转到内容页面。

在登录页面中单击注册按钮，将跳转到注册页面。

1.3.4 注册页面和模块介绍

注册页面包括欢迎提示和注册表单，如图1.6所示。

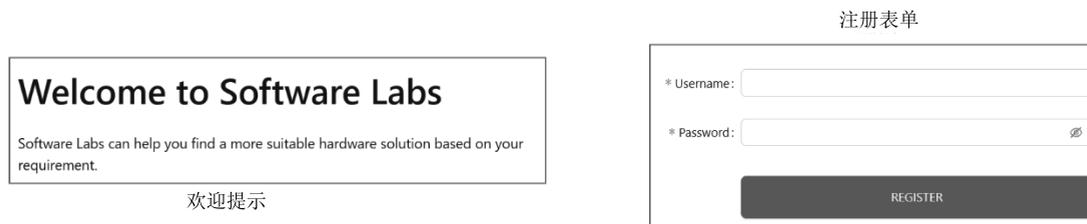


图 1.6 注册页面模块划分

注册页面和登录页面很相似，因此和登录页面的代码写在一个组件里，根据路由做判断，呈现不同的界面细节。

在注册页面，用户输入表单内容并单击注册按钮后，前端调用后端接口发送ajax请求，如果请求成功，则表示注册成功，并跳转到内容页面。

1.3.5 内容页面初始界面和模块介绍

内容页面在初始状态下，包括功能区、系统介绍banner和反馈，如图1.7所示。

左侧功能区中包括3个模块，分别是选择市场、选择软件和搜索。

在选择市场模块下，首先需要选择区域，有欧洲、亚洲、北美等，每个区域对应的国家列表不同。然后在国家列表下选择国家，每个国家对应的语言不同。最后在语言列表下选择语言。至此，选择市场模块的操作完成。

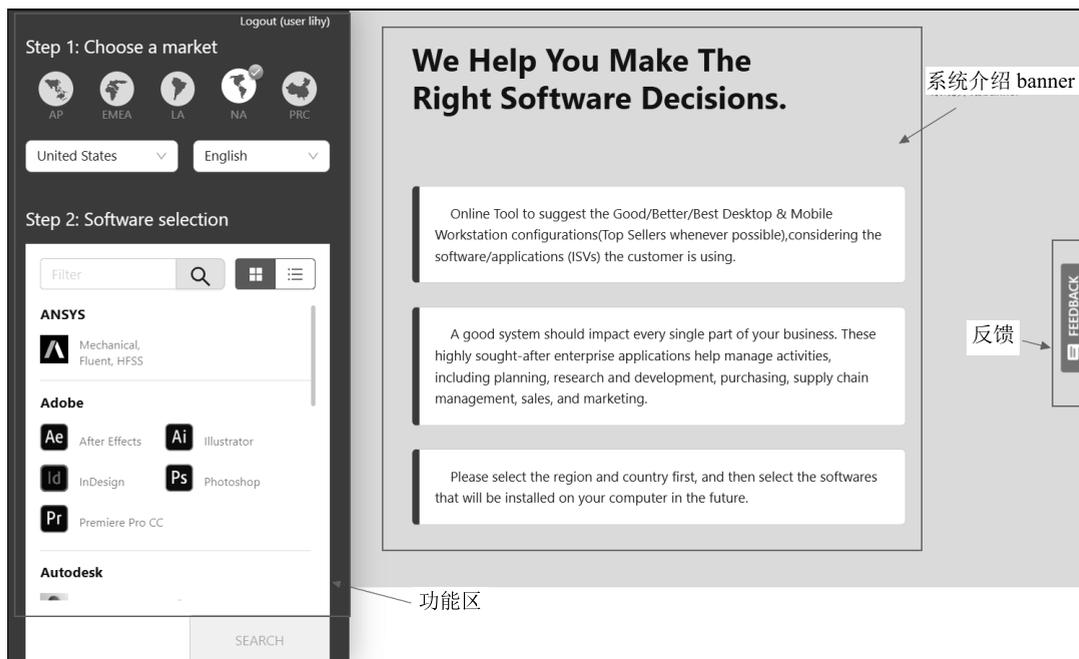


图 1.7 内容页面初始界面

由于笔者Chrome浏览器默认语言是美国英语，因此默认选择NA市场，美国，英文，如图1.8所示。

当选择市场模块操作完成后，选择软件模块会根据选择的语言展示相应语言的软件列表。用户根据需求选择1个或多个软件，当选择软件后，搜索按钮由灰色变为蓝色，表示可以搜索了，如图1.9所示。

另外，可以通过单击软件搜索框右侧的按钮来切换软件列表的展示形式，如图1.10所示。

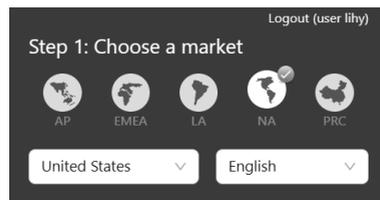


图 1.8 选择市场模块

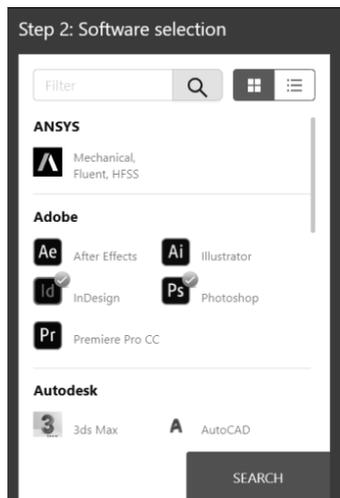


图 1.9 选择软件模块和搜索按钮

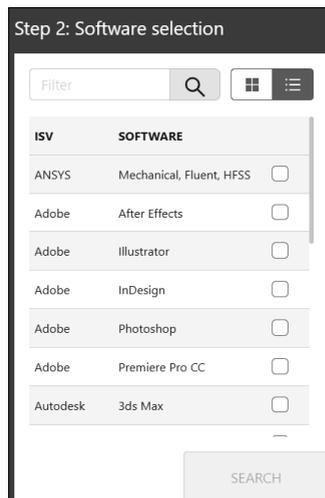


图 1.10 选择软件模块的另一种展示形式

图1.8~图1.10展示的区域、国家和语言列表数据，都是由前端向后端发送ajax，再通过接口返回的。

单击搜索按钮后，内容页面将呈现搜索到的产品列表。

1.3.6 产品列表界面和模块介绍

产品列表界面包括功能区、用户筛选的条件、产品列表、联系我们和反馈，如图1.11所示。其中，产品列表、联系我们和反馈每个功能块划分为一个模块，也就是一个组件，每个组件又可以拆分为功能更单一的子组件。产品列表模块固定展示6个产品，分为上下两块，上面是DESKTOP WORKSTATIONS，下面是MOBILE WORKSTATIONS。每一个产品都由轮播图、产品名称、产品信息、产品总价和产品提示组成。产品列表模块如图1.12所示。

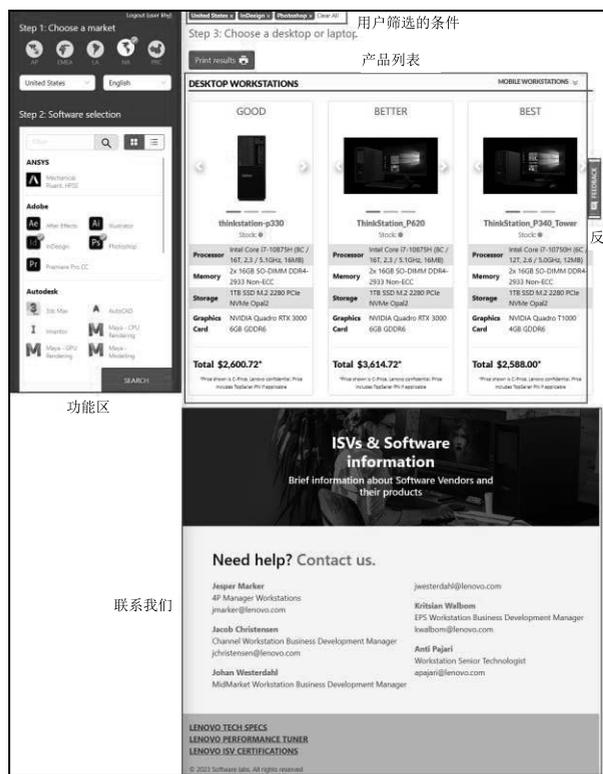


图 1.11 产品列表界面



图 1.12 计算机选购配置系统的产品列表模块

在内容页面上，单击FEEDBACK按钮后将出现弹窗界面，如图1.13所示。

产品列表界面的顶部还有条件tag和打印按钮，如图1.14所示。

联系我们模块如图1.15所示。

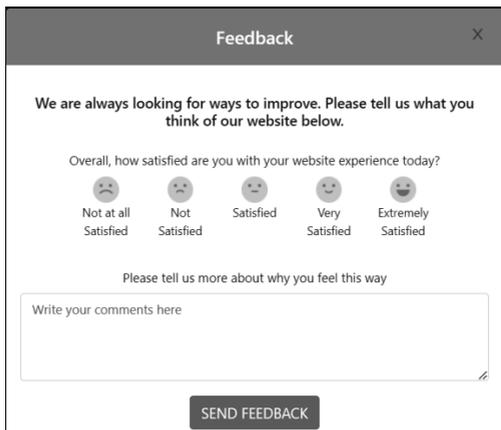


图 1.13 单击 FEEDBACK 按钮后出现弹窗界面



图 1.14 条件 tag 和打印按钮界面



图 1.15 联系我们模块

1.4 小结

本章主要介绍了多页面应用和单页面应用的概念和特点，以及组件化开发的概念和优势。第1.3节详细介绍了本书案例——计算机选购配置系统的UI界面和组件划分。通过对本章内容的学习，读者能了解网站系统的结构和模块，并能进行合适的划分，便于以后项目的开发和维护。

第 2 章

三大主流前端框架介绍

在前端项目中，可以借助某些框架（如React、Vue、Angular等）来实现组件化开发，使代码更容易复用。此时，一个网页不再是由一个个独立的HTML、CSS和JavaScript文件组成，而是按照组件的思想将网页划分成一个个组件，如轮播图组件、列表组件、导航栏组件等。将这些组件拼装在一起，就形成一个完整的网页。

本章主要涉及的知识点有：

- React框架介绍
- Vue框架介绍
- Angular框架介绍
- 如何选型

2.1 React

React框架是目前流行的前端框架之一。许多公司的项目都由React框架进行构建和编写，尤其是外企或涉及全球团队合作的项目。本节先简单介绍React框架的基础知识和必须了解的一些知识点，使读者对React有一个基本的概念和认知。

React是由Facebook团队开发的一个开源框架，官方网站如图2.1所示。

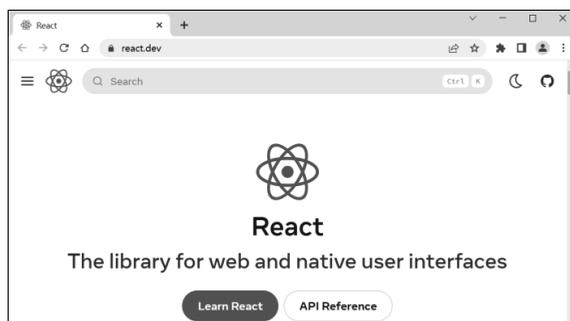


图 2.1 React 官方网站

React是一个用于构建用户界面的JavaScript库。使用React框架创建一系列的React组件（如缩略图、点赞按钮和视频等），然后将它们组合成一个页面、系统或应用程序。

React框架在开发项目时有一套流程和规范，无论你是自己工作还是与成千上万的其他开发人员合作，使用React都是一样的。它旨在让工程师可以无缝地组合由独立人员、团队或组织编写的组件。

React组件的本质是JavaScript函数。例如，下面是VideoList.js组件代码实例：

```
function VideoList({ videos, emptyHeading }) {
  const count = videos.length;
  let heading = emptyHeading;
  if (count > 0) {
    const noun = count > 1 ? 'Videos' : 'Video';
    heading = count + ' ' + noun;
  }
  return (
    <section>
      <h2>{heading}</h2>
      {videos.map(video =>
        <Video key={video.id} video={video} />
      )}
    </section>
  );
}
```

上面代码中return()中的这种标记语法称为JSX，它是React推广的JavaScript语法扩展。JSX看起来与HTML相似，对于写过HTML代码的前端工程师来说，写JSX组件非常容易，不需要记住很多特定标记，并且使用JSX标记写出的组件呈现逻辑清晰，这使得React组件易于创建、维护和删除。

React组件会接收数据并将这些数据和JSX模板编译后形成一段一段的JavaScript代码，这些JavaScript代码会将数据呈现到屏幕上。React框架可以向组件传递新的数据以响应交互，例如当用户通过表单输入内容时，React随后将更新屏幕以匹配新数据。

React是单向响应的数据流，采用单向数据绑定，即Model（数据）的更新会触发View（页面）的更新，而View的更新不会触发Model的更新，它们的作用是单向的。在React中，当用户操作View层的按钮或表单输入等需要更新Modal时，必须通过相应的Actions来进行操作。

2.2 Vue

Vue在中国公司的项目开发中非常流行，因为它具有上手快、轻量化的特点，并且文档对国人更友好。一些小型的、逻辑简单的项目大多使用Vue框架构建。

Vue是尤雨溪开发的一款开源的、构建用户界面的渐进式框架。Vue的官方网站如图2.2所示。

Vue的模板语法基于HTML的模板语法，并有特定的一套规则，例如插值语法，包括文本插值、Attribute插值等；指令语法，包括绑定事件的内部指令v-bind、v-on、v-model等，以及自定义指令；修饰符，v-on:submit.prevent等。



图 2.2 Vue 官方网站

与React类似，在底层机制中，Vue会将模板编译成JavaScript代码。结合响应式系统，当应用状态变更时，Vue能够智能地推导出需要重新渲染的组件的最少数量，并应用最少的DOM操作。

Vue支持单向数据绑定和双向数据绑定。

- 单向数据绑定时，使用v-bind属性绑定、v-on事件绑定或插值形式{{data}}。
- 双向数据绑定时，使用v-model指令，用户对View的更改会直接同步到Model。v-model本质是v-bind和v-on相组合的语法糖，是框架自动帮我们实现了更新事件。换句话说，我们完全可以采取单向绑定，自己实现类似的双向数据绑定。

2.3 Angular

Angular诞生于2009年，其出现的时间要早于React和Vue，它是一款来自谷歌的开源的Web前端框架，Angular的官方网站如图2.3所示。



图 2.3 Angular 官方网站

Angular的模板功能强大、丰富，并且还引入了Java的一些概念，是一款大而全的框架，更侧重于大型前端工程的构建，为开发人员屏蔽项目构建底层的细节提出了自己的一套解决方案。

使用Angular的难点是学习曲线比较陡峭，优点是由于使用了标准化的开发方式，后期能极大地提高开发生产力，提高开发效率。

AngularJS支持单向数据绑定和双向数据绑定。

- 单向数据绑定时，使用ng-bind指令或插值形式`{{data}}`。
- 双向数据绑定时，使用ng-model指令，用户对View的更改会直接同步到Model。

2.4 如何选型

框架选型由多个因素决定，例如项目的类型、项目的复杂程度以及项目组成员的技能掌握情况。

React适合多组件的应用程序，另外对于具有扩展和增长可能的项目，由于React组件具有声明性，因此它可以轻松处理此类复杂结构。

Vue由于具有可接受且快速的学习曲线，因此最适合解决短期的小型项目，例如，业务逻辑简单、不需要处理复杂数据结构的项目。

Angular 最适合大型和高级项目，用于创建有着复杂基础架构的大型企业应用程序。

2.5 小结

本章主要介绍了目前流行的三大前端框架，包括它们各自的特点、基础语法和数据绑定类型。最后介绍了开发项目时应该如何进行选型。

第 3 章

前端环境的搭建

开发项目首先要搭建代码编写环境。这个环境承载着HTML、CSS、JavaScript代码的运行、打包和编译。在搭建好的项目环境中，前端可以使用本地服务器进行开发，下载项目中所需的第三方工具包，配置接口调用时的服务器端代理地址，并在不同项目下切换不同版本Node.js。

本章主要涉及的知识点有：

- Node.js
- npm
- nvm
- CLI

3.1 Node.js 的安装与使用

Node.js（也称nodejs或者Node）是JavaScript代码运行时，前端项目中必不可少的一个环境。所有第三方工具包都依赖Node.js。

1. Node.js的下载和安装

通常是到Node.js官方网站下载稳定版本（LTS），首先，根据计算机操作系统选择Windows或Linux版本，根据系统处理器选择32位或64位；然后单击安装包进行安装即可，如图3.1所示。



图3.1 根据计算机系统下载Node.js

2. Node.js在前端项目中的作用

Node.js在纯前端开发中使用的功能比较少，一般有3个。

第一个是项目中使用了很多第三方工具包，它们的使用要依赖Node.js。如图3.2所示是计算机选购配置系统对应的客户端项目software-labs-client中依赖的部分第三方工具包。

第二个是前端项目的工程化，例如打包、构建、部署等。

第三个是搭建本地服务器，用于与后端通信进行接口调用。

不过，现在前端不必用Node.js底层代码一步一步地写本地服务器，通常有已经封装好的第三方插件，直接使用即可。

Node.js中启动服务的内置对象是http对象，这个对象有一些方法，例如创建一个服务器、监听端口等。流行的Express就是用Node.js的http对象写的。而前端打包工具webpack中自带的devServer就是一个轻量级的Node.js Express服务器，相当于是一个封装好的“Express的http服务器+调用webpack-dev-middleware”。

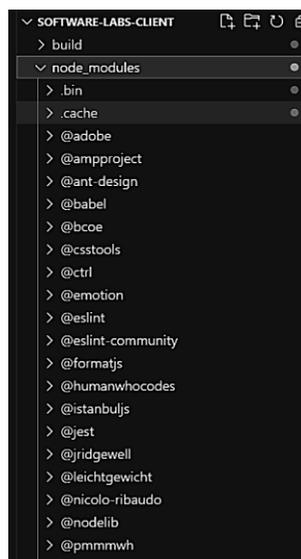


图3.2 software-labs-client中依赖的部分第三方工具包

3.2 npm 的安装与使用

npm是JavaScript的包管理工具，也是目前Node.js默认的包管理工具。npm能解决Node.js代码部署上的很多问题，例如从npm服务器下载别人编写的第三方包到本地使用、从npm服务器下载并安装别人编写的命令程序到本地使用，或者将自己编写的包或命令程序上传到npm服务器供别人使用。

由于新版的Node.js已经集成了npm，因此npm也一并安装好了。可以通过输入“npm -v”来测试是否成功安装npm，命令如下：

```
npm -v
```

出现版本提示表示安装成功。例如，在Windows系统中，在VS Code终端输入npm -v后，显示npm版本，如图3.3所示。

```
PS D:\project\book\softwareSystem\software-labs-client> npm -v
8.1.0
```

图3.3 显示npm的版本

npm常见的使用场景有以下3种：

- 允许用户从npm服务器下载别人编写的第三方包到本地使用。
- 允许用户从npm服务器下载并安装别人编写的命令程序到本地使用。
- 允许用户将自己编写的包或命令程序上传到npm服务器供别人使用。

开发项目时，不是所有交互和功能都需要自己写，例如axios、antd等，可以使用别人编写好的第三方工具包，这些包可以从npm服务器下载。

使用npm install安装第三方工具包的命令如下：

```
npm install <Module Name>
```

npm install可以缩写为npm i。

npm安装包时有3个常见参数：-d、-s、-g。

- -s即--save，包名会被注册在package.json的dependencies里面，在生产环境下这个包的依赖依然存在。
- -d即--dev，包名会被注册在package.json的devDependencies里面，仅在开发环境下存在的包用-d，如Babel、sass-loader这些解析器。
- -g表示全局安装，安装过一次后，在当前计算机的其他项目里也可以使用。

devDependencies里面的插件只用于开发环境，不用于生产环境，而dependencies是需要发布到生产环境的。

- 如果一个包需要在生产环境使用时就用npm install模块名-s，如axios、antd、react。
- 如果一个包只在开发时使用就用npm install模块名-d。

3.3 nvm 的安装与使用

nvm是一个Node.js版本管理器，也就是说，一个nvm可以管理多个Node.js版本。通常一个前端工程师可能同时参与多个项目的开发和维护，而不同项目由于创建时间不同，Node.js版本可能不同。有了nvm，可以方便快捷地安装、切换不同版本的Node.js。

1. nvm的安装

打开nvm的GitHub地址<https://github.com/coreybutler/nvm-windows/releases/tag/1.1.7>，下载Assets的第三个nvm-setup.zip安装包，下载之后双击安装包进行安装。

可以通过输入“nvm -v”来测试是否成功安装nvm。命令如下：

```
nvm -v
```

出现版本提示表示安装成功。例如，在Windows系统中，在VS Code终端输入nvm -v后，显示nvm版本，如图3.4所示。

2. nvm的使用

图3.4所示是官方提供的命令行使用方法，实际项目中常用的使用方法有4个：

(1) 安装Node.js某个版本：（xxx为Node.js版本号）

```
nvm install xxx
```

(2) 卸载Node.js某个版本：（xxx为Node.js版本号）

```
nvm uninstall xxx
```

```

PS D:\project\book\softwareSystem\software-labs-client> nvm -v
Running version 1.1.7.

Usage:

nvm arch                : Show if node is running in 32 or 64 bit mode.
nvm install <version> [arch] : The version can be a node.js version or "latest" for the latest stable version.
                                Optionally specify whether to install the 32 or 64 bit version (defaults to system arch).
                                Set [arch] to "all" to install 32 AND 64 bit versions.
                                Add --insecure to the end of this command to bypass SSL validation of the remote download server.
nvm list [available]    : List the node.js installations. Type "available" at the end to see what can be installed. Alias
sed as ls.
nvm on                  : Enable node.js version management.
nvm off                 : Disable node.js version management.
nvm proxy [url]        : Set a proxy to use for downloads. Leave [url] blank to see the current proxy.
                                Set [url] to "none" to remove the proxy.
nvm node_mirror [url]  : Set the node mirror. Defaults to https://nodejs.org/dist/. Leave [url] blank to use default url.
nvm npm_mirror [url]   : Set the npm mirror. Defaults to https://github.com/npm/cli/archive/. Leave [url] blank to default url.
nvm uninstall <version> : The version must be a specific version.
nvm use [version] [arch] : Switch to use the specified version. Optionally specify 32/64bit architecture.
                                nvm use <arch> will continue using the selected version, but switch to 32/64 bit mode.
nvm root [path]        : Set the directory where nvm should store different versions of node.js.
                                If <path> is not set, the current root will be displayed.
nvm version             : Displays the current running version of nvm for Windows. Aliased as v.

```

图3.4 显示nvm的版本

(3) 查看所有已安装的Node.js版本:

```
nvm list
```

(4) 切换到某个Node.js版本: (xxx为Node.js版本号)

```
nvm use xxx
```

3.4 CLI 与 create-react-app

一般企业根据业务需要往往会部署运行多套软件系统。相应地，前端工程师需要搭建多个项目。如果这些项目创建的目录结构或者组织架构有很多共同之处，那么搭建一个新项目不必每次都从0开始，可以使用前端脚手架实现前端架构的重用，以减少重复工作。

1. 什么是CLI

广义的CLI只是Command Line Interface（命令行界面）的缩写，是指在用户提示符下输入可执行指令的界面。它通常不支持鼠标，用户通过键盘输入指令，计算机接收到指令后，予以执行。

在前端项目中，不同的技术栈会有自己的目录结构、工作流程，所以很多前端框架（例如React、Vue、Angular、Ember）会有自己的脚手架工具（一般就叫xxx-cli）。因此，在前端项目中，CLI通常指的就是脚手架。

使用一个脚手架命令，项目的目录结构、webpack配置、Babel配置、空的测试文件都会自动生成。工程师可以直接写核心业务代码，不必做重复性工作。

2. create-react-app

create-react-app是Facebook专门出的一个快速构建React项目的脚手架，官方网站地址是<https://create-react-app.bootcss.com/>，create-react-app是基于webpack+ES 6创建的。

create-react-app有两种使用方法。

方法1:

```
npm install create-react-app -g
create-react-app software-labs-client --template redux
```

第一行代码是全局安装create-react-app脚手架工具;第二行代码是使用create-react-app创建项目,项目名是software-labs-client,并且自动集成Redux、Redux Toolkit和React-Redux。

--template后面也可以加其他第三方工具,例如create-react-app software-labs-client --template redux-typescript,表示集成Redux并且使用TypeScript语法。

software-labs-client是计算机选购配置系统的客户端部分,主要完成页面的展示、交互、通过接口从后端获取数据等功能。在后面的服务器端章节还有software-labs-server,它是后端项目,负责编写接口以及实现MongoDB数据库的增、删、改、查。

方法2:

```
npx create-react-app software-labs-client --template redux
```

此行代码表示安装create-react-app脚手架并创建项目software-labs-client,是第一种方法的合并。如图3.5所示是使用命令npx create-react-app software-labs-client --template redux创建计算机选购配置系统的客户端项目software-labs-client。

```
PS D:\project> npx create-react-app software-labs-client --template redux

Creating a new React app in D:\project\software-labs-client.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template-redux...

[###.....] | idealTree:eslint: timing idealTree:node_modules/eslint Completed in 961ms
```

图3.5 使用create-react-app创建计算机选购配置系统的客户端项目software-labs-client

项目创建好后,在VS Code的终端中输入npm run start启动项目,在浏览器中输入http://localhost:3000/展示页面,如图3.6所示。

在VS Code中打开项目software-labs-client,此时的目录结构如图3.7所示。



图3.6 项目启动后的默认页面

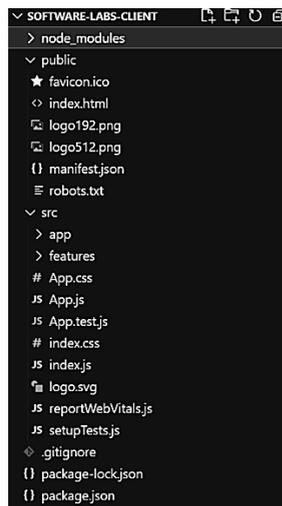


图3.7 项目software-labs-client的目录结构

结构说明:

- (1) `node_modules`是项目依赖的所有第三方包。
- (2) `public`文件夹保存所有静态资源,并且这些静态资源不会被`webpack`编译。其中 `index.html`是页面模板。一般情况下, `public`文件夹中的内容不会被修改,所有项目基本相同。
- (3) `src`文件夹是最重要的目录,它里面存放的是项目的所有源代码,其中`index.js`是项目的JavaScript入口文件,即所有JavaScript逻辑从这里开始;`features/`下保存所有项目的自定义组件。这里不同项目根据需要可以自行修改。
- (4) `src/app/store.js`是Redux的store文件,保存所有Redux数据。
- (5) `src/App.js`是容器组件,即初始化启动后看到的首页。
- (6) `package-lock.json`文件主要用来锁定包的版本。使用`npm install`安装包后会自动生成`package-lock.json`,但它并不是每次都生成,只有在不存在的情况下才会自动生成。当`package-lock.json`存在并且包发生变化时,会自动同步更新。
- (7) `README.md`是项目信息,里面包含项目的启动、介绍及文件说明等。
- (8) `package.json`文件定义项目的基本信息,如项目名称、版本号、作者、在该项目下可执行的命令以及项目的依赖模块等。例如`scripts`属性下的命令可以启动项目、测试项目、打包项目等。

当安装第三方模块时,模块的相关信息会被自动添加到`package.json`文件中,示例代码如下:

```
{
  "name": "software-labs-client",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@reduxjs/toolkit": "^1.9.5",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^14.4.3",
    "axios": "^1.4.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-redux": "^8.0.5",
    "react-scripts": "5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
```

```

    "not dead",
    "not op_mini all"
  ],
  "development": [
    "last 1 chrome version",
    "last 1 firefox version",
    "last 1 safari version"
  ]
}
}
}

```

代码解析:

- name属性保存项目名称。
- dependencies保存项目部署生产环境时依赖的所有第三方包。还有一个和dependencies类似的属性是devDependencies, 用于保存项目在开发环境时依赖的第三方包。这些包只在开发时使用, 因为生产环境的代码是已经经过打包后的文件, 有些包已经不需要了。
- scripts是可执行命令。例如在项目根目录下打开终端, 输入命令npm start可以启动项目进行开发, 输入命令npm run build可以打包项目文件。注意, 除了start可以直接用npm start外, 其他都要用npm run xxx。
- eslintConfig是代码检测, 可以配合ESLint工具进行代码检测。
- browserslist是浏览器列表配置, 表示当执行npm run build进行生产环境代码构建时, 所有代码兼容全球使用率>0.2%的各种浏览器; 当执行npm run start进行开发时, 网站的所有代码兼容Chrome、Firefox和Safari最新版本的现代浏览器。这里提供了良好的开发体验, 尤其是在使用诸如async/await之类的语言功能时, 能在生产中提供与许多浏览器的高度兼容性。

以上就是使用脚手架时默认创建的项目的结构和目录。

上面的目录和结构可以按照实际项目进行适当修改, 主要是src/下的文件需要根据UI界面划分的模块来创建文件夹和文件, 修改后的目录结构如图3.8所示。

修改后的项目结构中, src/下增加了assets/, 用于保存需要webpack编译的第三方JS、CSS或者图片等。如图3.9所示, 我们放置了项目中的公用CSS文件: font.css是所有字体的样式, icon.css是所有图标的样式, normalize.css是所有通用样式; fonts/文件夹保存所有字体, images/保存所有图片。

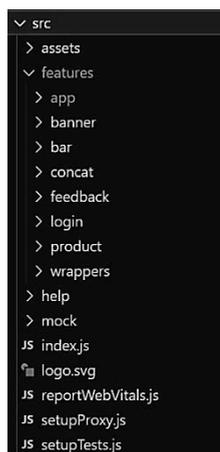


图 3.8 修改后项目 software-labs-client 的结构

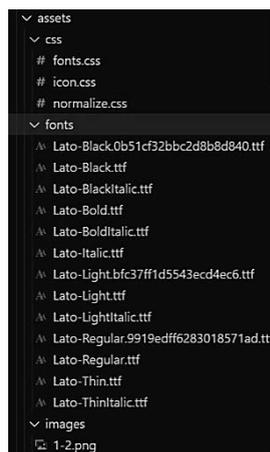


图 3.9 software-labs-client 下的 assets/文件夹

3.5 小结

本章主要介绍了构建前端项目需要准备的工作，包括Node.js的安装，npm、nvm以及React.js项目的脚手架create-react-app的使用。至此，我们通过`npx create-react-app software-labs-client --template redux`这个命令创建了一个空的React项目，并且根据software-labs-client的界面UI对src/文件夹下的目录进行了调整。