

第 3 章



视频讲解

项目框架设计

本章作为实践开发的开端,其核心任务是进行顶层项目架构设计。在编写具体的业务功能代码之前,确立清晰、可扩展的架构蓝图至关重要。本章将详细阐述项目的整体设计思路、功能模块划分以及技术选型依据,并搭建起一个结构规范、能够支撑后续所有功能模块开发的项目骨架。

1. 本章主要实现功能

- (1) 构建标准化的 FastAPI 项目文件结构,包括静态资源目录 static、模板目录 templates 及路由包 routers。
- (2) 配置 Jinja2 模板引擎与静态文件服务,实现前后端基础资源的加载与渲染。
- (3) 通过 APIRouter 实现模块化路由管理与基于 Swagger UI 的离线 API 文档配置。

2. 预期运行效果

项目主程序能够成功启动,通过浏览器可访问测试根路由返回的 JSON 数据;API 文档页面(/docs)在无网络环境下能正常加载,且路由结构已按模块清晰分类。

本书已将所开发的企业门户网站部署在公网(网址详见前言二维码),可直接访问并在线体验。实现效果如图 3-1 所示。



图 3-1 “派森科技”门户网站效果(仅作为教学用途,所有信息均虚构)

3.1 项目概述

本项目旨在开发一个名为“派森科技”的企业门户网站,网站内所有信息均为虚构,仅用于教学演示。在动手编写代码之前,明确项目的设计理念与功能模块是首要任务。

3.1.1 设计思路

为了确保项目具备良好的可维护性与扩展性,本实践项目遵循以下现代 Web 开发的核心设计思路:

(1) 前后端职责分离。后端(FastAPI 框架)专注于处理业务逻辑、数据库交互与 API 数据接口的提供;前端(Bootstrap 5 框架结合 Jinja2 模板)专注于页面布局结构与用户交互效果的呈现。

(2) 模块化开发。随着业务逻辑的增长,将所有代码集中于单一文件会导致极高的维护成本。本项目将网站按业务功能拆分为独立的模块,各模块拥有专属的路由处理文件和模板文件夹,从而提升代码的隔离性与可读性。

(3) 数据驱动。摒弃传统静态网站将内容硬编码在 HTML 中的方式。网站的动态内容(如新闻、产品等)将统一存储在数据库中,并借助可视化的后台管理系统进行增、删、改、

查操作,最终通过前端模板动态渲染至页面。

3.1.2 功能模块划分

结合真实企业门户网站的常规需求,本项目共划分为 8 个核心功能模块: 首页、公司简介、产品中心、新闻动态、人才招聘、认证系统、服务支持以及后台管理。具体的模块结构划分如图 3-2 所示。

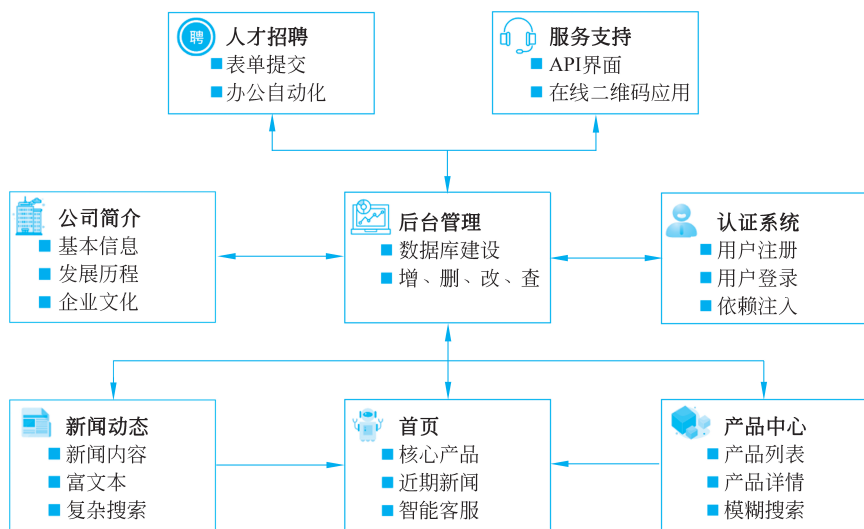


图 3-2 “派森科技”门户网站功能模块

3.2 项目框架搭建

规范的文件目录结构是大型项目成功的基础。合理的目录规划不仅能让代码组织井然有序,还能帮助开发者快速定位和修改目标模块。

3.2.1 目录结构规划

首先,在项目根目录下创建基础的文件与目录树。同时,将前期准备好的离线 swagger-ui 资源和 bootstrap 前端静态资源文件,分别复制到对应的静态资源子目录中。

完整的初始目录结构规划如下。

```
PythonWeb/
├── static/                                # 存放 CSS、JS、图片等前端静态资源
│   ├── swagger-ui/                      # 离线 API 文档静态资源
│   ├── bootstrap/                       # Bootstrap 5 框架资源
│   └── css/                             # 自定义样式
```

```
|   |-- js/           # 自定义脚本
|   |-- image/        # 图片资源
|-- templates/       # 存放 Jinja2 所需的 HTML 模板文件
|-- routers/         # 存放各个业务模块的路由文件
|   |-- __init__.py   # 公司简介模块路由 (示例)
|   |-- company.py    # FastAPI 应用主入口文件
|-- main.py
```

目录解析：

(1) static/。用于存放所有直接发送给用户浏览器的静态文件，这类文件不需要经过 Python 后端的逻辑处理。

(2) templates/。用于集中管理所有的 HTML 模板文件。

(3) routers/。用于存放各个业务模块的路由控制器文件。其中 __init__.py 文件将其标识为一个标准的 Python 包。

(4) main.py。作为整个 FastAPI 应用的唯一入口，负责应用的初始化、中间件挂载及各模块路由的集中注册。

3.2.2 主入口与模块化路由配置

在 FastAPI 中，如果将所有 URL 请求的处理逻辑都写在 main.py 中，文件会显得臃肿、难以维护。为了贯彻模块化开发思想，需要引入 APIRouter 工具包进行路由分发。

首先，在 routers 包中编写首个业务子路由。打开 routers/company.py，输入以下代码。

```
from fastapi import APIRouter

# 创建路由组，指定统一前缀与文档标签
router = APIRouter(prefix="/company", tags=["公司简介"])

@router.get("/about")
def get_company_about():
    return {"message": "这里是派森科技的公司简介页面"}
```

代码剖析：

(1) APIRouter 实例化。APIRouter 可以被视作一个小型的 FastAPI 实例。在实例化时，通过 prefix="/company" 参数为该模块下的所有路由统一添加了 URL 前缀。这意味着后续定义的 /about 路由，其最终的完整访问路径将是 /company/about。

(2) API 文档归类。tags=["公司简介"] 参数用于在自动生成的 API 文档中，将该路由归类到“公司简介”标签下，使得接口文档结构更加清晰。

子路由定义完成后，需要回到主入口文件中进行全局注册。打开 main.py，完成 FastAPI 的基础初始化与路由引入，如下所示。

```
from fastapi import FastAPI
from fastapi.staticfiles import StaticFiles
from fastapi.openapi.docs import get_swagger_ui_html
from routers import company

# 初始化应用并禁用默认在线文档 URL
```

```
app = FastAPI(
    title="派森科技",
    description="Python Web 开发从入门到实践",
    version="1.0.0",
    docs_url=None,
    redoc_url=None
)

# 挂载静态文件目录
app.mount("/static", StaticFiles(directory="static"), name="static")

# 配置离线 API 文档路由
@app.get("/docs", include_in_schema=False)
async def custom_swagger_ui_html():
    return get_swagger_ui_html(
        openapi_url=app.openapi_url,
        title=app.title + " - API Docs",
        swagger_js_url="/static/swagger-ui/swagger-ui-bundle.js",
        swagger_css_url="/static/swagger-ui/swagger-ui.css",
    )

# 注册业务模块路由
app.include_router(company.router)

# 根路由临时测试
@app.get("/")
def read_root():
    return {"message": "欢迎访问派森科技门户网站项目"}
```

代码剖析：

(1) 应用初始化。在 FastAPI() 实例化过程中,通过传入 title、description 等元数据,为项目确立了基本信息。显式设置 docs_url=None 和 redoc_url=None 是为了关闭框架默认依赖外部网络 CDN 的 API 文档,防止在离线或内网开发时出现页面白屏。

(2) 静态目录挂载。app.mount 方法将 URL 路径/static 映射到项目物理目录下的 static 文件夹。这是至关重要的一步,只有挂载后,浏览器才能合法地请求到前端的 CSS、JS 和图片等静态资源。

(3) 离线文档路由。通过自定义/docs 路由并返回内置的 get_swagger_ui_html,将文档渲染所需的 JS 和 CSS 强制指向之前准备好的本地静态资源。

(4) 路由注册引擎。app.include_router(company.router)将之前在 company.py 中定义的子路由正式接入主程序的生命周期中,使得模块化路由真正生效。

配置完毕后,在终端执行 uvicorn main:app --reload 启动服务。使用浏览器访问 127.0.0.1:8000/company/about 即可看到成功返回的 JSON 数据。同时,访问 127.0.0.1:8000/docs 可查阅已按模块分类清晰的 API 文档,如图 3-3 所示。

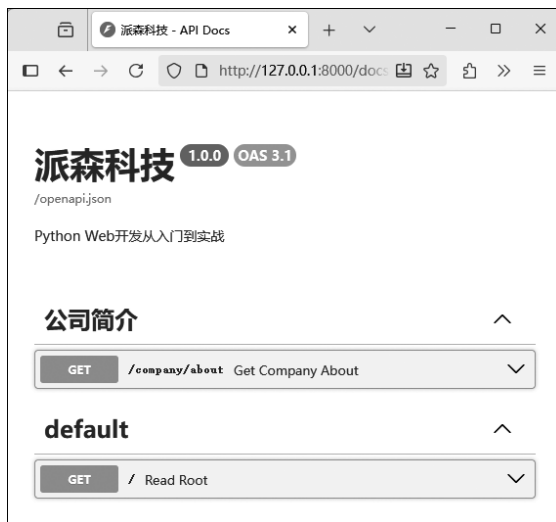


图 3-3 API 文档中的模块化路由

3.3 模板与静态文件

FastAPI 本身是一个专注于构建高性能 API 接口的底层框架,默认情况下响应的数据格式为 JSON。然而,构建面向普通用户的门户网站需要向浏览器呈现排版美观的 HTML 网页。因此,必须引入模板引擎来动态生成 HTML,并配合静态文件服务来加载样式与交互脚本。

3.3.1 安装与配置 Jinja2

Jinja2 是 Python 生态中最流行、功能最强大的模板引擎之一。它允许在 HTML 文件中嵌入 Python 风格的控制语句(如循环、条件判断)和变量,从而实现页面的动态渲染。

首先,在虚拟环境中执行以下命令安装 Jinja2 库。

```
pip install jinja2
```

安装完成后,需要修改 main.py 文件,配置模板引擎,并将原先只返回 JSON 数据的根路由改造为返回 HTML 页面,如下所示。

```
from fastapi import Request
from fastapi.responses import HTMLResponse
from fastapi.templating import Jinja2Templates

# ...保留已有代码...

# 指定模板文件存放目录
```

```
templates = Jinja2Templates(directory="templates")

# 修改根路由, 利用模板引擎渲染 HTML
@app.get("/", response_class=HTMLResponse)
async def read_root(request: Request):
    # 必须向模板传递 request 对象
    return templates.TemplateResponse(request, "home.html")
```

代码剖析:

(1) Jinja2Templates 实例化。通过指定 `directory="templates"`, 告知模板引擎应当去项目根目录下的 `templates` 文件夹中寻找所有的 HTML 模板文件。

(2) 强制响应类型。将 `@app.get` 装饰器中的 `response_class` 显式指定为 `HTMLResponse`, 这是为了确保底层 HTTP 响应头中的 `Content-Type` 被正确设置为 `text/html`, 以便浏览器能将其当作网页进行渲染, 而非当作纯文本处理。

(3) 模板上下文传递: 在 FastAPI 的规范中, `TemplateResponse` 方法的第一个参数是 `request` 对象。这样做是为了让模板引擎能够在底层直接访问到当前 HTTP 请求的网络上下文(如 URL 路径、查询参数等)。

接着, 在项目的 `templates` 目录下创建对应的 `home.html` 模板文件, 并输入以下基础 HTML 骨架。

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>派森科技 - 首页</title>
</head>
<body>
    <h1>欢迎来到派森科技</h1>
    <p>这是一个通过 Jinja2 模板引擎渲染的页面。</p>
</body>
</html>
```

确保 Uvicorn 服务仍在运行, 在浏览器中刷新访问 `127.0.0.1:8000`。此时页面显示的将不再是单调的 JSON 字符串, 而是经过浏览器解析渲染后的 HTML 网页标题与段落, 如图 3-4 所示。



图 3-4 使用模板引擎渲染的首页页面

3.3.2 静态资源引用

纯净的 HTML 结构无法满足现代网页的美观需求,必须通过外部的 CSS 文件进行样式美化。接下来将阐述如何将存放在 static 目录下的 CSS 资源正确引入 Jinja2 模板中。

首先,在 static/css 目录下新建一个名为 style.css 的样式文件,定义一些基础的演示排版逻辑,如下所示。

```
body {
    background-color: #f8f9fa;
    color: #212529;
    text-align: center;
    padding-top: 50px;
}
h1 {
    color: #0d6efd;
}
```

随后,重新编辑 templates/home.html 文件,在页面头部加载上述自定义样式。在 Jinja2 模板中引入静态资源时,建议使用内置的 url_for 函数,而不是直接硬编码物理路径。

修改后的 home.html 文件代码如下。

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>派森科技 - 首页</title>
    <!-- 动态生成并引入 CSS 文件 -->
    <link rel="stylesheet" href="{{ url_for('static', path='css/style.css') }}">
</head>
<body>
    <h1>欢迎来到派森科技</h1>
    <p>这是一个通过 Jinja2 模板引擎渲染的页面。</p>
    <p>此页面已加载自定义 CSS 样式。</p>
</body>
</html>
```

代码剖析:

(1) 动态路径解析。{{url_for('static',path='css/style.css')}} 是 Jinja2 模板引擎特有的语法标记。其中的'static'对应于 main.py 中 app.mount 方法里配置的 name="static"。

(2) 路径解耦优势。通过 url_for 函数,框架会根据当前应用的实际运行环境与路由挂载情况,自动推导并生成正确的资源 URL 路径。这种机制有效避免了在项目重构、目录层级发生改变或部署至不同服务器路径时,因硬编码导致的资源链接失效问题。

保存所有修改后的文件并刷新浏览器页面。此时,网页将呈现出带有浅灰背景色、文本整体居中以及蓝色标题的外观表现,如图 3-5 所示。这标志着“后端路由分发→模板引擎渲

染→静态资源加载”的前后端工作流已在项目骨架中成功实现。



图 3-5 模板与静态文件集成效果

小结

本章完成了项目的宏观架构设计与基础骨架搭建。通过规范标准化的目录结构,确立了模块化开发思路;借助 APIRouter 实现了路由分发,并成功集成 Jinja2 模板引擎与静态文件服务。这一系列操作构建了高可维护性的项目基础,为后续具体业务模块(如公司简介、产品中心等)的开发做好了充分的技术准备。



习题 3