这里是Docode代码训练营第三讲专题

第一讲：环境配置+项目认知

第二讲：AGI介绍+AI搭建

第三讲：Web工程介绍+AI对话网页搭建

值得一提的是，第三讲为本次训练营的提高内容，不要求强制掌握，即仅需看过代码成功运行整体项目（推荐流程1）也算通过。

第三讲完成推荐流程有两个：

1. 阅读本文件中的《第三讲课程文档》了解相关知识——阅读、理解hw中的`example-finished-hw3-code`项目并成功运行

   > 验收标准，与之前相同：需提交pr，并commit时标注hw3-名称。pr内容仅需在hw文件夹里的个人名字下文件夹中放入运行成功的截图即可

2. 阅读本文件中的《第三讲课程文档》了解相关知识——跟随本文的《第三讲代码任务设计》完成`example-unfinished-hw3-code`代码补充

   > 验收标准，与之前相同：需提交pr，并commit时标注hw3-名称。pr内容需在hw文件夹的个人名字下文件夹单独开一个code3文件夹进行项目实践与运行。(你需要把example-unfinished-hw3-code的代码粘贴到你的路径下)

第三讲课程文档：从终端到网页 - 构建完整的AI对话界面

一、课程目标

通过本讲学习，你将能够：

1. 深入理解Web开发的基本原理和前端三大技术
2. 掌握Flask模板系统和静态文件服务
3. 学会前后端数据交互的完整流程
4. 完成一个功能完善的网页版AI对话系统

二、Web开发基础知识详解

2.1 什么是Web开发?

Web开发是创建网站和Web应用程序的过程。它分为两个主要部分：

前端（Frontend）：用户看到和交互的部分

- 负责用户界面（UI）和用户体验（UX）
- 运行在用户的浏览器中
- 主要技术：HTML、CSS、JavaScript

后端（Backend）：服务器端的逻辑处理

- 负责数据处理、业务逻辑、数据库操作
- 运行在服务器上
- 主要技术：Python（Flask）、数据库等

2.2 前端三剑客深入解析

HTML（超文本标记语言）详解

HTML是网页的骨架，使用标签来定义网页的结构和内容。

HTML文档结构：

```
<!DOCTYPE html>            <!-- 声明文档类型 -->
<html lang="zh">           <!-- 根元素，设置语言 -->
<head>                     <!-- 文档头部，包含元数据 -->
    <meta charset="UTF-8">     <!-- 字符编码 -->
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- 响应式设计 --
>
    <title>页面标题</title>    <!-- 浏览器标签页标题 -->
    <link rel="stylesheet" href="style.css"> <!-- 引入CSS文件 -->
</head>
<body>                      <!-- 文档主体，包含可见内容 -->
    <h1>这是标题</h1>
    <p>这是段落</p>
    <div>这是容器</div>
    <script src="script.js"></script> <!-- 引入JavaScript文件 -->
</body>
</html>
```

常用HTML标签详解：

- `<div>`：块级容器，用于布局分组
- `<span>`：行内容器，用于文本片段
- `<h1>-<h6>`：标题标签，h1最大，h6最小
- `<p>`：段落标签
- `<button>`：按钮，可以触发JavaScript事件
- `<input>`：输入框，有多种类型（text、password、email等）
- `<textarea>`：多行文本输入框
- `<ul>`、`<ol>`、`<li>`：无序列表、有序列表、列表项
- `<a>`：超链接标签

HTML属性：

```
<div id="chat-container" class="container main">
    <!-- id: 唯一标识符，用于CSS和JavaScript选择 -->
    <!-- class: 类名，用于CSS样式 -->
</div>

<input type="text" placeholder="请输入消息...">
    <!-- type: 输入类型 -->
    <!-- placeholder: 占位符文本 -->
```

CSS（层叠样式表）详解

CSS负责网页的外观和布局，让网页变得美观和易用。

CSS工作原理：

1. **选择器**：确定要设置样式的HTML元素
2. **属性**：要改变的样式特征
3. **值**：属性的具体设置

CSS选择器类型：

```
/* 元素选择器 - 选择所有div元素 */
```

```css
div {
    background-color: #f0f0f0;
}

/* 类选择器 - 选择class="message"的元素 */
.message {
    padding: 10px;
    margin: 5px;
}

/* ID选择器 - 选择id="chat-container"的元素 */
#chat-container {
    height: 500px;
    overflow-y: auto;
}

/* 后代选择器 - 选择.message内的.content元素 */
.message .content {
    font-size: 16px;
}

/* 伪类选择器 - 鼠标悬停时的样式 */
button:hover {
    background-color: #0056b3;
}
```

CSS盒模型：

```css
.box {
    width: 200px;           /* 内容宽度 */
    height: 100px;          /* 内容高度 */
    padding: 10px;          /* 内边距：内容与边框的距离 */
    border: 2px solid #333; /* 边框：围绕内容的线条 */
    margin: 15px;           /* 外边距：元素与其他元素的距离 */
}
```

CSS布局方式：

```css
/* Flexbox布局 - 现代网页布局的首选 */
.flex-container {
    display: flex;
    flex-direction: column;    /* 垂直排列 */
    justify-content: center;   /* 主轴居中 */
    align-items: center;       /* 交叉轴居中 */
}

/* Grid布局 - 二维网格布局 */
.grid-container {
    display: grid;
    grid-template-columns: 1fr 3fr; /* 两列，1:3比例 */
    gap: 20px;
}
```

响应式设计：

```
/* 媒体查询 - 根据屏幕大小应用不同样式 */
@media (max-width: 768px) {
    .container {
        padding: 10px;
        font-size: 14px;
    }
}
```

JavaScript详解

JavaScript是网页的"大脑"，负责处理用户交互和动态功能。

JavaScript基础语法：

```
// 变量声明
const userName = "小明";          // 常量，不可改变
let messageCount = 0;            // 变量，可以改变
var oldStyle = "不推荐";         // 旧式声明（不推荐）

// 函数定义
function sendMessage(content) {
    console.log("发送消息:", content);
    return "消息已发送";
}

// 箭头函数（现代写法）
const sendMessage2 = (content) => {
    console.log("发送消息:", content);
    return "消息已发送";
};
```

DOM操作详解：

```
// 获取页面元素
const button = document.getElementById('send-btn');            // 通过ID获取
const messages = document.getElementsByClassName('message');  // 通过类名获取
const inputs = document.querySelectorAll('input');           // 通过CSS选择器获取

// 修改元素内容
button.textContent = "发送";                    // 修改纯文本
button.innerHTML = "<strong>发送</strong>";     // 修改HTML内容

// 修改元素样式
button.style.backgroundColor = "blue";
button.classList.add('active');                // 添加CSS类
button.classList.remove('disabled');           // 移除CSS类

// 创建新元素
const newDiv = document.createElement('div');
newDiv.className = 'message user';
```

```
newDiv.textContent = '用户消息';
document.body.appendChild(newDiv);
```

事件处理：

```
// 添加事件监听器
button.addEventListener('click', function() {
    console.log('按钮被点击');
});

// 键盘事件
input.addEventListener('keypress', function(event) {
    if (event.key === 'Enter') {
        console.log('用户按下回车键');
    }
});

// 表单事件
form.addEventListener('submit', function(event) {
    event.preventDefault(); // 阻止默认提交行为
    console.log('表单提交被拦截');
});
```

异步编程与Fetch API：

```
// 使用Fetch发送HTTP请求
fetch('/chat', {
    method: 'POST',                                  // 请求方法
    headers: {
        'Content-Type': 'application/json'           // 请求头
    },
    body: JSON.stringify({message: '你好'})          // 请求体
})
.then(response => {
    if (!response.ok) {
        throw new Error('网络请求失败');
    }
    return response.json();                          // 解析JSON响应
})
.then(data => {
    console.log('收到响应:', data);
    // 处理成功响应
})
.catch(error => {
    console.error('请求出错:', error);
    // 处理错误
});

// 现代async/await语法
async function sendMessageAsync() {
    try {
        const response = await fetch('/chat', {
```

```
            method: 'POST',
            headers: {'Content-Type': 'application/json'},
            body: JSON.stringify({message: '你好'})
        });

        const data = await response.json();
        console.log('收到响应:', data);
    } catch (error) {
        console.error('请求出错:', error);
    }
}
```

2.3 Flask Web框架深入

Flask基础概念

Flask是一个轻量级的Python Web框架，用于构建Web应用程序。

Flask应用结构：

```
from flask import Flask, render_template, request, jsonify

# 创建Flask应用实例
app = Flask(__name__)

# 路由装饰器 - 定义URL和处理函数的对应关系
@app.route('/')                    # 根路径
def index():
    return "Hello World!"

@app.route('/user/<name>')         # 动态路由
def user_profile(name):
    return f"用户：{name}"

@app.route('/api/data', methods=['POST'])  # 指定HTTP方法
def handle_data():
    data = request.json            # 获取JSON数据
    return jsonify({"status": "success"})  # 返回JSON响应
```

Flask模板系统（Jinja2）

模板继承：

```
<!-- base.html - 基础模板 -->
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}默认标题{% endblock %}</title>
</head>
<body>
    {% block content %}{% endblock %}
</body>
```

```
</html>

<!-- index.html - 继承基础模板 -->
{% extends "base.html" %}

{% block title %}AI助手{% endblock %}

{% block content %}
<h1>欢迎使用AI助手</h1>
{% endblock %}
```

模板语法：

```
<!-- 变量输出 -->
<h1>{{ page_title }}</h1>

<!-- 条件判断 -->
{% if user_name %}
    <p>欢迎，{{ user_name }}! </p>
{% else %}
    <p>请先登录</p>
{% endif %}

<!-- 循环遍历 -->
{% for message in messages %}
    <div class="message">{{ message.content }}</div>
{% endfor %}

<!-- 静态文件链接 -->
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">
<script src="{{ url_for('static', filename='js/main.js') }}"></script>
```

通用Flask静态文件服务

```
项目目录/
├── app.py
├── static/                 # 静态文件目录
|   ├── css/
|   |   └── style.css
|   ├── js/
|   |   └── main.js
|   └── images/
|       └── logo.png
└── templates/              # 模板目录
    └── index.html
```

在模板中引用静态文件：

```
<!-- CSS文件 -->
<link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}">

<!-- JavaScript文件 -->
<script src="{{ url_for('static', filename='js/main.js') }}"></script>

<!-- 图片文件 -->
<img src="{{ url_for('static', filename='images/logo.png') }}" alt="Logo">
```

2.4 前后端交互原理深入

HTTP协议基础

HTTP请求方法：

- `GET`：获取数据（查询）
- `POST`：发送数据（创建）
- `PUT`：更新数据（完整更新）
- `PATCH`：部分更新数据
- `DELETE`：删除数据

HTTP状态码：

- `200`：成功
- `201`：创建成功
- `400`：客户端错误（请求格式错误）
- `401`：未授权
- `404`：资源未找到
- `500`：服务器内部错误

JSON数据格式

JSON（JavaScript Object Notation）是前后端数据交换的标准格式：

```
// JavaScript对象
const messageData = {
    "message": "你好，AI助手",
    "timestamp": "2025-05-30T10:30:00Z",
    "user_id": 12345
};

// 转换为JSON字符串（发送给服务器）
const jsonString = JSON.stringify(messageData);

// 从JSON字符串解析为对象（接收服务器响应）
const parsedData = JSON.parse(jsonString);
```

Python中的JSON处理：

```python
import json

# Python字典转JSON
data = {"response": "你好！有什么可以帮助您的吗？"}
json_string = json.dumps(data, ensure_ascii=False)

# JSON转Python字典
json_data = '{"message": "你好"}'
python_dict = json.loads(json_data)

# Flask中的便捷方法
from flask import jsonify, request

@app.route('/api/chat', methods=['POST'])
def chat_api():
    user_data = request.json          # 自动解析JSON请求
    response_data = {"reply": "收到消息"}
    return jsonify(response_data)     # 自动转换为JSON响应
```

完整的前后端交互流程

1. 用户操作触发事件

```javascript
// 用户点击发送按钮
document.getElementById('send-btn').addEventListener('click', function() {
    const message = document.getElementById('input').value;
    sendMessageToServer(message);
});
```

2. JavaScript发送HTTP请求

```javascript
async function sendMessageToServer(message) {
    try {
        const response = await fetch('/chat', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'Accept': 'application/json'
            },
            body: JSON.stringify({
                message: message,
                timestamp: new Date().toISOString()
            })
        });

        if (!response.ok) {
            throw new Error(`HTTP error! status: ${response.status}`);
        }

        const data = await response.json();
        displayResponse(data.response);
    } catch (error) {
```

```
        console.error('请求失败:', error);
        displayError('发送消息失败，请重试');
    }
}
```

3. Flask后端处理请求

```
@app.route('/chat', methods=['POST'])
def handle_chat():
    try:
        # 获取请求数据
        request_data = request.json
        user_message = request_data.get('message')

        if not user_message:
            return jsonify({'error': '消息不能为空'}), 400

        # 调用AI模型处理
        ai_response = toyagi.chat(user_message)

        # 返回响应
        return jsonify({
            'response': ai_response,
            'status': 'success',
            'timestamp': datetime.now().isoformat()
        })

    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

4. JavaScript处理响应并更新界面

```
function displayResponse(response) {
    // 创建新的消息元素
    const messageDiv = document.createElement('div');
    messageDiv.className = 'message ai';
    messageDiv.textContent = response;

    // 添加到消息容器
    const messagesContainer = document.getElementById('messages');
    messagesContainer.appendChild(messageDiv);

    // 滚动到最新消息
    messagesContainer.scrollTop = messagesContainer.scrollHeight;
}
```

2.5 现代Web开发实践示例

用户体验（UX）优化

1. 加载状态指示

```
function showLoading() {
    const loadingDiv = document.createElement('div');
    loadingDiv.id = 'loading';
    loadingDiv.innerHTML = '🤔 AI正在思考...';
    document.getElementById('messages').appendChild(loadingDiv);
}

function hideLoading() {
    const loadingDiv = document.getElementById('loading');
    if (loadingDiv) {
        loadingDiv.remove();
    }
}
```

2. 错误处理

```
function handleError(error) {
    console.error('错误详情:', error);

    // 显示用户友好的错误信息
    const errorMessage = error.message.includes('fetch')
        ? '网络连接出现问题，请检查网络后重试'
        : '处理请求时出现错误，请稍后重试';

    displayErrorMessage(errorMessage);
}
```

3. 输入验证

```
function validateInput(message) {
    if (!message || message.trim().length === 0) {
        alert('请输入消息内容');
        return false;
    }

    if (message.length > 1000) {
        alert('消息长度不能超过1000个字符');
        return false;
    }

    return true;
}
```

第三讲代码任务设计

任务概述

基于第二讲的终端对话系统，完成网页版AI对话界面的搭建。你只需要填写少量的关键代码，就能拥有一个功能完整的Web应用！

代码TODO任务

> 直接点左侧的放大镜 搜索输入TODO，依次完成就好！很简单的朋友们

1. `app.py`

```
# TODO1：完成主页路由
# 提示：使用@app.route装饰器定义根路径'/'，函数名为index，返回render_template('index.html')
@app.route('/')
def index():
    return ______('index.html')

# TODO2：调用toyagi处理用户消息
# 提示：使用toyagi.chat()方法处理user_message
response = toyagi.______(user_message)
return jsonify({"response": response})

# TODO3：完成清空记忆的路由
# 提示：路径为'/memory/clear'，方法为['POST']
@app.route('______', methods=['______'])
```

2.`templates/index.html`

```
<!-- TODO4：设置页面标题 -->
<!-- 提示：在title标签中写入"笃小实 - 您的AI助手" -->
<title>______</title>

<!-- TODO5：引入CSS文件 -->
<!-- 提示：使用url_for函数，路径为static/css/style.css -->
<link rel="stylesheet" href="{{ url_for('static', filename='______') }}">

<!-- TODO6：引入JavaScript文件 -->
<!-- 提示：使用url_for函数，路径为static/js/main.js -->
<script src="{{ url_for('static', filename='______') }}"></script>
```

3. CSS样式 `static/css/style.css`

```
/* TODO7：设置body的基础样式 */
/* 提示（也可自由搭配）：字体为'Segoe UI', 字体大小16px, 行高1.5, 文字颜色#333, 背景颜色#f5f5f5,  */
body {
    font-family: '_____', Tahoma, Geneva, Verdana, sans-serif;
    font-size: _____px;
    line-height: _____;
    color: _____;
    background-color: _____;
    min-height: _____;
}
```

4. JavaScript交互 `static/js/main.js`

```
// TODO8：定义清空聊天函数
function clearChat() {
    // TODO9：显示确认对话框
    // 提示：使用confirm函数询问'确定要清空聊天记录吗？'
    if (confirm('_____')) {
        // TODO10：清空聊天消息显示区域
        // 提示：设置innerHTML为空字符串
        chatMessages.innerHTML = '_____';

        // TODO11：调用后端清空记忆
        // 提示：fetch到'/memory/clear'，方法为'POST'
        fetch('_____', {
            method: '_____'
        })
        .then(response => response.json())
        .then(data => {
            console.log('记忆已清空');
            showToast('聊天记录已清空', 'success');
        })
        .catch(error => {
            console.error('清空记忆时出错:', error);
            showToast('清空记忆时出错', 'error');
        });
    }
}
```

任务完成指引

第一步：理解代码结构

1. **仔细阅读每个文件的注释**，了解每部分的作用
2. **查看TODO提示**，理解需要填写的内容
3. **参考课程文档**中的技术知识点

第二步：按顺序完成TODO

1. **从TODO1开始**，按数字顺序逐个完成
2. **每完成几个TODO就测试一下**，确保功能正常
3. **遇到问题时回顾课程文档**的相关章节

第三步：测试应用

1. **运行Flask应用**：`python app.py`
2. **打开浏览器**访问 `http://localhost:5000`
3. **测试基本功能**：发送消息、接收回复、清空聊天

第四步：调试和完善

1. **使用浏览器开发者工具**查看控制台错误
2. **检查网络请求**是否正常发送和接收
3. **优化界面细节**，提升用户体验

完成这些任务后，恭喜你！你已经成功创建了一个功能完整的Web版AI助手！🎉