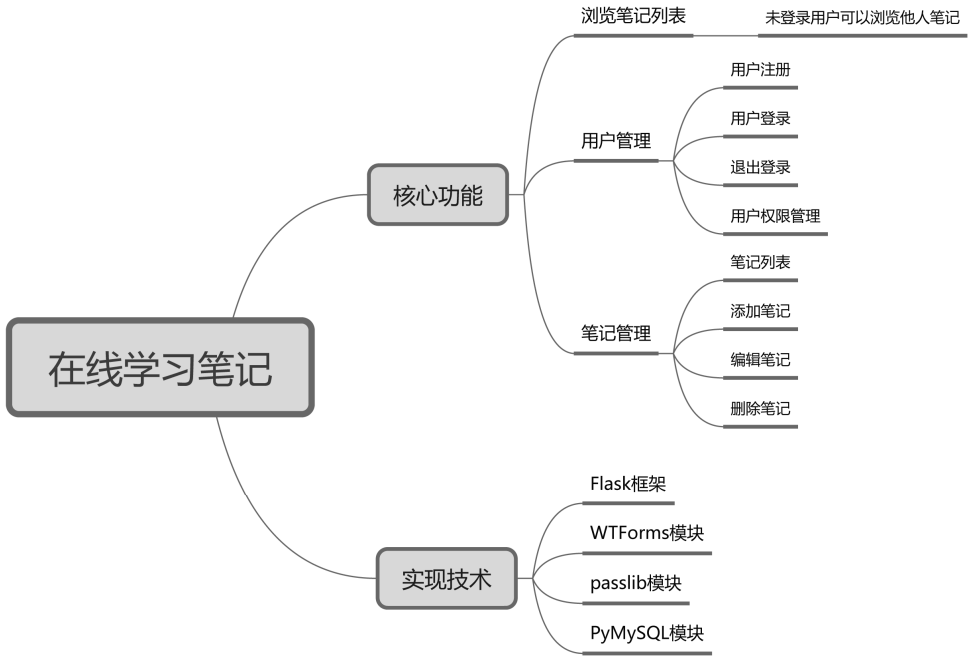
# 第 3 章 在线学习笔记

——Flask + WTForms + passlib + PyMySQL


项目微视频

在当今数字化时代，阅读习惯正逐渐从传统的纸质书籍阅读方式转向电子化、网络化的阅读方式。我们也将面临一个新的挑战，那就是如何有效地筛选、整理和记忆每天接触到的大量信息。在此背景下，开发一个在线学习笔记项目就显得尤为迫切和重要。本章将使用 Python Web 框架 Flask 开发一个在线学习笔记项目。

本项目的核心功能及实现技术如下：




## 3.1 开 发 背 景

随着互联网技术的飞速发展和智能移动设备的普及，人们越来越倾向于在碎片化的时间里通过手机、平板或计算机阅读书籍、文章与学习资料。然而，这一转变也带来了新的需求与挑战：如何高效地管理和整合个人的阅读笔记？这就需要有一个优秀的在线学习笔记项目。

Flask 框架是一种轻量级的 Python Web 框架，它具有灵活性强、扩展库丰富等特点。WTForms 模块提供了一套完整的机制来验证用户提交的数据，确保数据的合法性。将 Flask 框架与 WTForms 模块搭配使用，不仅可以提高开发效率，还能增强应用的安全性和用户体验。

本项目的实现目标如下：

☑ 笔记的创建与编辑：用户应能够轻松创建和编辑文本笔记。

- ☑ 用户体验：界面友好，操作直观简单，确保新用户能够快速上手。
- ☑ 响应式布局，用户在 PC 端和移动端都能达到较好的阅读体验。
- ☑ 性能：响应速度快，处理大量数据时仍能保持良好的性能。
- ☑ 兼容性：系统应能兼容多种浏览器和设备。

# 3.2 系 统 设 计

## 3.2.1 开发环境

本项目的开发及运行环境如下：

- ☑ 操作系统：推荐 Windows 10、Windows 11 或更高版本。
- ☑ 开发工具：PyCharm 2024（向下兼容）。
- ☑ 开发语言：Python 3.12。
- ☑ 数据库：MySQL 8.0+PyMySQL 驱动。
- ☑ Python Web 框架：Flask 3.0。

## 3.2.2 业务流程

用户访问在线学习笔记时，可以使用游客的身份浏览笔记首页，以及笔记内容。但是如果需要管理笔记（如笔记列表、添加笔记、编辑笔记、删除笔记等），就必须先注册为网站会员，登录网站后才能执行相应的操作。系统业务流程如图 3.1 所示。

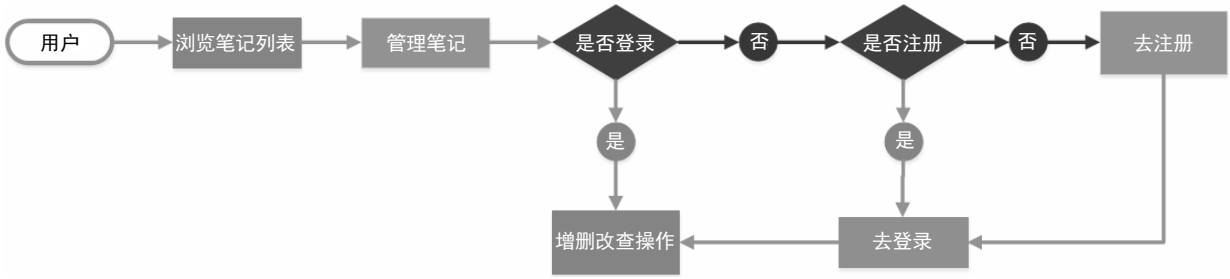



图 3.1 在线学习笔记的业务流程

**说明**

在图 3.1 中，对笔记进行管理时进行的增、删、改、查操作是指添加笔记、删除笔记、编辑笔记和查看笔记列表。

## 3.2.3 功能结构

本项目的功能结构已经在章首页中给出，其实现的具体功能如下：

- ☑ 浏览笔记列表模块：未登录用户可以浏览他人写的笔记。
- ☑ 用户管理模块：包括用户注册、用户登录、退出登录和用户权限管理等功能。
- ☑ 笔记管理模块：包括笔记列表、添加笔记、编辑笔记、删除笔记等。

**说明**

由于浏览笔记列表模块与笔记管理模块中的笔记列表实现方法基本相同，这里不对其进行单独介绍。

# 3.3 技 术 准 备

## 3.3.1 技术概览

在开发在线学习笔记时，应用的 Web 开发框架是 Flask 框架，操作数据库的模块是 PyMySQL。

### 1．Flask 框架

Flask 是一个轻量级 Python Web 框架，它把 Werkzeug 和 Jinja 黏合在一起，能够很容易地被扩展。例如，本项目中，在项目主文件 mange.py 中，首先创建 Flask 实例对象，然后创建并配置首页路由函数，在首页路由函数中，渲染首页文件，代码如下：

```
from flask import Flask,render_template

app = Flask(__name__)                          # 创建 Flask 实例对象

# 首页
@app.route('/')
def index():
    return render_template('home.html')        # 渲染模板
```

然后使用 run()方法运行程序，代码如下：

```
if __name__ == '__main__':
    app.secret_key='secret123'
    app.run(debug=True)
```

有关 Flask 框架的使用方法，在《Python 从入门到精通（第 3 版）》中有详细的讲解，对该知识不太熟悉的读者可以参考该书对应的内容。

### 2．使用 PyMySQL 模块操作数据库

由于 MySQL 服务器以独立的进程运行，并通过网络对外服务，所以，需要支持 Python 的 MySQL 驱动来连接到 MySQL 服务器。在 Python 中支持 MySQL 的数据库模块有很多，本项目选择使用简单方便的 PyMySQL 模块。使用 PyMySQL 模块时，大致可以分为 3 个步骤，下面分别进行介绍。

（1）安装 PyMySQL。

使用 pip 工具安装 PyMySQL 模块非常简单，只需要在 venv 虚拟环境下使用如下命令即可。

```
pip install PyMySQL
```

（2）连接 MySQL。

使用 PyMySQL 模块连接数据库。首先需要导入 PyMySQL 模块，然后使用 PyMSQL 的 connect()方法来连接数据库。关键代码如下：

```
import pymysql
```

```
# 打开数据库连接，参数 1：主机名或 IP；参数 2：用户名；参数 3：密码；参数 4：数据库名称
db = pymysql.connect(host="localhost", user="root", password="root", db="flask")
……省略部分代码
# 关闭数据库连接
db.close()
```

在上述代码中，重点关注 connect()函数的参数：

```
db = pymysql.connect(
                        host='localhost',       # 主机名
                        user='root',            # 用户名
                        password='root',        # 密码
                        db='flask'              # 数据库名称
)
```

此外，connect()函数还有两个常用参数设置：

☑ charset:utf8：用于设置 MySQL 字符集为 UTF-8。

☑ cursorclass: pymysql.cursors.DictCursor：用于设置游标类型为字典类型，默认为元组类型。

（3）根据下面的操作流程使用 PyMySQL 模块操作数据库。操作 MySQL 的基本流程：连接 MySQL→创建游标→执行 SQL 语句→关闭连接。

有关 PyMySQL 模块的使用方法，在《Python 从入门到精通（第 3 版）》中有详细的讲解，对该知识不太熟悉的读者可以参考该书对应的内容。

下面对实现本项目时用到的其他主要技术点进行必要介绍，如使用 WTForms 模块、使用 passlib 模块进行加密等，以确保读者可以顺利完成本项目。

## 3.3.2 使用 WTForms 模块

由于 Flask 框架内部并没有提供全面的表单验证，所以通常搭配第三方模块来实现，例如，可以使用 WTForms 模块。下面将介绍在 Flask 项目中使用 WTForms 模块的基本方法。

### 1．安装 WTForms 模块

由于 WTForms 模块不是 Flask 框架自带的，在使用 WTForms 之前，需要确保已经安装了该模块。如果没有安装，可以通过 pip 工具来安装。例如，可以在 venv 虚拟环境下使用如下命令进行安装。

```
pip install wtforms
```

### 2．创建表单类

每个表单都应该被定义为一个 Python 类，这个类继承自 wtforms.Form，用于定义表单字段及其验证规则。在这个类里，可以定义多个字段，每个字段对应 HTML 表单中的一个输入域。例如，创建一个用户登录表单，包含用户名和密码两个字段，每个字段都有相应的验证器来确保数据的有效性。代码如下：

```
from wtforms import Form, StringField, PasswordField
from wtforms.validators import DataRequired

class LoginForm(Form):
    username = StringField(
        '用户名',
        validators=[
            DataRequired(message='请输入用户名')
        ]
    )
    password = PasswordField(
        '密码',
        validators = [
```

```
            DataRequired(message='密码不能为空')
        ]
    )
```

### 3．在模板中渲染表单

在 HTML 模板中，可以直接使用表单实例的属性来渲染表单字段。例如，在模板文件中渲染上面创建的用户登录表单，代码如下：

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="UTF-8">
    <title>渲染表单</title>
</head>
<body>
  <form action="" method="POST">
    <div class="form-group">
      <label>用户名</label>
      <input type="text" name="username" class="form-control" value={{request.form.username}}>
    </div>
    <div class="form-group">
      <label>密码</label>
      <input type="password" name="password" class="form-control" value="">
    </div>
    <button type="submit" class="btn btn-primary">登录</button>
  </form>
</body>
</html>
```

### 4．处理表单提交

在 Flask 路由对应的视图函数中，首先实例化定义好的表单类，并将请求的数据传递给它。这通常是通过 request.form 来实现的。例如，处理用户登录表单，可以使用下面的代码。

```
# 用户登录
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm(request.form)      # 实例化表单类
    if request.method == 'POST':        # 如果提交表单
        # 从表单中获取字段
        username = request.form['username']
        password_candidate = request.form['password']
        print('用户名：',username,' 密码：',password_candidate)
    return render_template('demo.html')
```

上面的代码实现了访问 http://127.0.0.1:5000/login 时显示用户登录页面，如图 3.2 所示。输入用户名和密码并单击“登录”按钮后，将提交表单，同时将获取到的用户名和密码输出到控制台，如图 3.3 所示。


图 3.2　用户登录页面

```
用户名： mr  密码： mrosft
```

图 3.3　获取提交的用户名和密码

除了上面代码中使用的 StringField 和 PasswordField HTML 标准字段，WTForms 还支持很多其他的 HTML 标准字段，如表 3.1 所示。

表 3.1　WTForms 支持的 HTML 标准字段

| 字 段 类 型 | 说　　明 | 字 段 类 型 | 说　　明 |
|---|---|---|---|
| StringField | 文本字段 | BooleanField | 复选框，值为 True 和 False |
| TextAreaField | 多行文本字段 | RadioField | 一组单选框 |
| PasswordField | 密码文本字段 | SelectField | 下拉列表 |
| HiddenField | 隐藏文本字段 | SelectMultipleField | 下拉列表，可选择多个值 |
| DateField | 文本字段，值为 datetime.date 格式 | FileField | 文件上传字段 |
| DateTimeField | 文本字段，值为 datetime.datetime 格式 | SubmitField | 表单提交按钮 |
| IntegerField | 文本字段，值为整数 | FormField | 把表单作为字段嵌入另一个表单 |
| DecimalField | 文本字段，值为 decimal.Decimal | FieldList | 一组指定类型的字段 |
| FloatField | 文本字段，值为浮点数 | | |

WTForms 内置的验证函数如表 3.2 所示。

表 3.2　WTForms 支持的内置验证函数

| 字 段 类 型 | 说　　明 | 字 段 类 型 | 说　　明 |
|---|---|---|---|
| Email | 验证电子邮件地址 | NumberRange | 验证输入的值在数字范围内 |
| EqualTo | 比较两个字段的值；常用于要求输入两次密码进行确认的情况 | Optional | 无输入值时跳过其他验证函数 |
| IPAddress | 验证 IPv4 网络地址 | Required | 确保字段中有数据 |
| Length | 验证输入字符串的长度 | Regexp | 使用正则表达式验证输入值 |
| NumberRange | 验证输入的值在数字范围内 | URL | 验证 URL |

## 3.3.3　使用 passlib 模块进行加密

passlib 模块是一个强大的 Python 密码哈希库，用于安全地处理用户密码。passlib 支持 30 多种密码散列算法，每种算法都有其特定用途和强度。例如，sha256_crypt 提供了一种基于 SHA-256 的密钥派生函数，该函数使用 SHA-256 算法，并且通常包含加盐（salt）和可配置的迭代次数（rounds）以增加安全性。加盐是为了防止彩虹表攻击，而迭代次数可以增加哈希计算的复杂度，从而抵御暴力破解。下面将介绍如何使用 passlib 模块的 sha256_crypt 进行密码处理。

### 1．安装 passlib 模块

由于 passlib 模块是第三方模块，所以在使用该模块前，需要确保已经安装了该模块。如果没有安装，可以通过 pip 工具来安装。例如，可以在 venv 虚拟环境下使用如下命令进行安装：

```
pip install passlib
```

### 2．对密码进行加密

通常情况下，程序中存储的用户密码不应该采用明文存储，而是存储其哈希值。这可以通过调用特定的散列算法（如 sha256_crypt）提供的方法来实现。例如，在 passlib 模块中，采用 sha256_crypt 算法时，对应的加密方法为 encrypt()，该方法会返回一个字符串，这个字符串是原始密码经过哈希处理后的结果。例如，通过下面的代码可以为给定的密码生成一个包含盐值和哈希值的字符串：

```
hashed_password = passlib.hash.sha256_crypt.hash('mrsoft')
```

加密码后的字符串类似下面的内容：

```
$5$rounds=535000$iWOTB4gpfZXV66xk$O0oYG7Q/knj.XfvVWFmSfZfNF6lnSaiXnApyCyKsP85
```

3．验证密码

由于散列函数是一种单向函数，所以一旦数据被转换成哈希值，就几乎不可能从哈希值还原出原始数据。此时，想要验证密码，就需要将用户输入的密码与存储的哈希值进行比较，如果一样，就表示输入的密码正确。例如，通过下面的代码，可以验证用户提供的明文密码是否与之前存储的哈希值一致。

```
passlib.hash.sha256_crypt.verify('mrsoft', hashed_password)
```

# 3.4 数据库设计

## 3.4.1 数据库概要说明

本项目采用 MySQL 数据库，数据库名称为 notebook。读者可以使用 MySQL 命令行方式或 MySQL 可视化管理工具（如 Navicat）创建数据库。使用命令行方式时输入如下命令：

```
create database notebook default character set utf8;
```

## 3.4.2 创建数据表

因为本项目主要涉及用户和笔记两部分，所以在 notebook 数据库中创建两个数据表，数据表名称及作用如下。

☑ users：用户表，用于存储用户信息。

☑ articles：笔记表，用于存储笔记信息。

在 MySQL 命令行下或 MySQL 可视化管理工具（如 Navicat）下执行以下 SQL 语句创建数据表：

```
DROP TABLE IF EXISTS 'users';
CREATE TABLE 'users' (
  'id' int(8) NOT NULL AUTO_INCREMENT,
  'username' varchar(255) DEFAULT NULL,
  'email' varchar(255) DEFAULT NULL,
  'password' varchar(255) DEFAULT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS 'articles';
CREATE TABLE 'articles' (
  'id' int(8) NOT NULL AUTO_INCREMENT,
  'title' varchar(255) DEFAULT NULL,
  'content' text,
  'author' varchar(255) DEFAULT NULL,
  'create_date' datetime DEFAULT NULL,
  PRIMARY KEY ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

## 3.4.3 数据表结构

users 用户表的表结构如表 3.3 所示。

表 3.3　users 用户表的表结构

| 字　　段 | 类　　型 | 长　　度 | 是否允许为空 | 含　　义 |
|---|---|---|---|---|
| id | int | 默认 | 否 | 主键，编号 |
| username | varchar | 255 | 是 | 用户名 |
| email | varchar | 255 | 是 | 邮箱 |
| password | varchar | 255 | 是 | 密码 |

articles 笔记表的表结构如表 3.4 所示。

表 3.4　articles 笔记表的表结构

| 字　　段 | 类　　型 | 长　　度 | 是否允许为空 | 含　　义 |
|---|---|---|---|---|
| id | int | 默认 | 否 | 主键，编号 |
| title | varchar | 255 | 是 | 标题 |
| content | text | 默认 | 是 | 内容 |
| author | varchar | 255 | 是 | 作者 |
| create_date | datetime | 默认 | 是 | 记录时间 |

# 3.5　数据库操作类设计

本项目使用 PyMySQL 来驱动数据库，并实现对笔记的增、删、改、查功能。每次执行数据表操作都需要遵循如下流程：连接数据库→执行 SQL 语句→关闭数据库。

为了复用代码，单独创建一个 mysql_uitl.py 文件，该文件中包含一个 MysqlUtil 类，用于实现基本的增删改查方法。代码如下：

```
import pymysql                                                          # 引入 pymysql 模块
import traceback                                                        # 引入 python 中的 traceback 模块，跟踪错误
import sys                                                              # 引入 Python 内置的 sys 模块

class MysqlUtil():
    def __init__(self):
        '''
            初始化方法，连接数据库
        '''
        host = '127.0.0.1'                                              # 主机名
        user = 'root'                                                   # 数据库用户名
        password = 'root'                                               # 数据库密码
        database = 'notebook'                                           # 数据库名称
        self.db = pymysql.connect(host=host,user=user,password=password,db=database)   # 建立连接
        self.cursor = self.db.cursor(cursor=pymysql.cursors.DictCursor)     # 设置游标，并将游标设置为字典类型

    def insert(self, sql):
        '''
            插入数据库
            sql:插入数据库的 SQL 语句
        '''
        try:
            # 执行 SQL 语句
            self.cursor.execute(sql)
            # 提交到数据库执行
            self.db.commit()
```

```
        except Exception:                    # 方法一：捕获所有异常
            # 如果发生异常，则回滚
            print("发生异常", Exception)
            self.db.rollback()
        finally:
            # 最终关闭数据库连接
            self.db.close()

    def fetchone(self, sql):
        '''
            查询数据库：单个结果集
            fetchone(): 该方法获取下一个查询结果集。结果集是一个对象
        '''
        try:
            # 执行 SQL 语句
            self.cursor.execute(sql)
            result = self.cursor.fetchone()
        except:                              # 方法二：采用 traceback 模块查看异常
            # 输出异常信息
            traceback.print_exc()
            # 如果发生异常，则回滚
            self.db.rollback()
        finally:
            # 最终关闭数据库连接
            self.db.close()
        return result

    def fetchall(self, sql):
        '''
            查询数据库：多个结果集
            fetchall(): 接收全部的返回结果行
        '''
        try:
            # 执行 SQL 语句
            self.cursor.execute(sql)
            results = self.cursor.fetchall()
        except:                              # 方法三：采用 sys 模块回溯最后的异常
            # 输出异常信息
            info = sys.exc_info()
            print(info[0], ":", info[1])
            # 如果发生异常，则回滚
            self.db.rollback()
        finally:
            # 最终关闭数据库连接
            self.db.close()
        return results

    def delete(self, sql):
        '''
            删除结果集
        '''
        try:
            # 执行 SQL 语句
            self.cursor.execute(sql)
            self.db.commit()
        except:                              # 把这些异常保存到一个日志文件中，用来分析这些异常
            # 将错误日志输入目录文件中
            f = open("\log.txt", 'a')
            traceback.print_exc(file=f)
            f.flush()
```

```
            f.close()
            # 如果发生异常，则回滚
            self.db.rollback()
        finally:
            # 最终关闭数据库连接
            self.db.close()

    def update(self, sql):
        '''
            更新结果集
        '''
        try:
            # 执行 SQL 语句
            self.cursor.execute(sql)
            self.db.commit()
        except:
            # 如果发生异常，则回滚
            self.db.rollback()
        finally:
            # 最终关闭数据库连接
            self.db.close()
```

在使用 MysqlUtil 类时，只需要引入 MysqlUtil 类，实例化该类，并调用相应方法即可。

# 3.6 用户管理模块设计

用户管理模块主要包括 4 部分功能：用户注册、用户登录、退出登录和用户权限管理。这里的用户权限管理是指，只有登录后，用户才能访问某些页面（如控制台）。下面分别介绍每个功能的实现。

## 3.6.1 实现用户注册功能

用户注册功能是指在线学习笔记的注册新用户功能。在其页面中，需要填写用户名、邮箱、密码和确认密码。如果没有输入用户名、邮箱、密码或确认密码，系统将提示错误。此外，如果填写的格式错误也将提示错误。

### 1．创建注册路由

实现用户注册时，需要先创建用户注册的路由。在 manage.py 入口文件中，创建一个名为 app 的 Flask 实例，然后调用 app.route()函数创建路由，关键代码如下：

```
app = Flask(__name__)                                    # 创建应用
# 用户注册
@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegisterForm(request.form)                    # 实例化表单类

    # 省略部分代码
    return render_template('register.html', form=form)   # 渲染模板
```

在上述代码中，@app.route()函数的第一个参数/register 是对应的 URL 的 path 部分；第二个参数 methods 是请求方式，这里使用列表指定接受 GET 和 POST 两种方式的请求。接下来，在 register()函数中实例化 RegisterForm 类，并使用 render_template()函数渲染模板。

### 2．创建模板文件

因为 render_template()函数默认查找的模板文件路径为/templates/，所以需要在该路径下创建 register.html 模板文件。代码如下：

```
{% extends 'layout.html' %}

{% block body %}
<div class="content">
  <h1 class="title-center">用户注册</h1>
  {% from "includes/_formhelpers.html" import render_field %}
  <form method="POST" action="">
    <div class="form-group">
      {{render_field(form.email, class_="form-control")}}
    </div>
    <div class="form-group">
      {{render_field(form.username, class_="form-control")}}
    </div>
    <div class="form-group">
      {{render_field(form.password, class_="form-control")}}
    </div>
    <div class="form-group">
      {{render_field(form.confirm, class_="form-control")}}
    </div>
    <p><input type="submit" class="btn btn-primary" value="注册"></p>
  </form>
</div>
{% endblock %}
```

在上述代码中，使用了 extends 标签来引入公共文件 layout.html ，该文件包含了网站模板的基础框架，也称为父模板。由于网站页面包含很多通用的部分，如导航栏和底部信息等。将这些通用信息写入父模板，然后，使每个页面继承通用信息，并使用 block 标签来覆盖特有的信息。这样就简化了代码，达到了代码复用的目的。

此外，使用 WTForm 模块的 render_filed()函数来渲染表单中的字段。render_filed()函数的第一个参数是 Form 类的属性，该 Form 类是使用 render_tempalte()函数传递过来的，也就是 RegisterForm 类；第二个参数 _class 是模板中的 class 名称。

### 3．实现注册功能

在 register.html 注册页面中，Form 表单的 action 属性值为空，即表示当用户单击“注册”按钮时，表单提交到当前页面。因此，需要在 manage.py 文件的 register()函数中继续编写提交表单的代码。

register()函数的完整代码如下：

```
# 用户注册
@app.route('/register', methods=['GET', 'POST'])
def register():
    form = RegisterForm(request.form)                                    # 实例化表单类
    if request.method == 'POST' and form.validate():                     # 如果提交表单，并且字段验证通过
        # 获取字段内容
        email = form.email.data
        username = form.username.data
        password = sha256_crypt.encrypt(str(form.password.data))         # 对密码进行加密

        db = MysqlUtil()                                                 # 实例化数据库操作类
        sql = "INSERT INTO users(email,username,password) \
                VALUES ('%s', '%s', '%s')" % (email,username,password)   # user 表中插入记录
        db.insert(sql)
```

```
        flash('您已注册成功，请先登录', 'success')                    # 闪存信息
        return redirect(url_for('login'))                          # 跳转到登录页面

    return render_template('register.html', form=form)             # 渲染模板
```

在上述代码的 if 语句中，先通过 request.method == 'POST'来判断用户是否提交了表单。如果用户已经提交表单，就使用 form.validate()判断是否通过 RegisterForm 类的全部验证规则。如果两个条件同时满足，则获取用户提交的注册信息，并对密码进行加密。接下来，实例化 MysqlUtil 类，将用户信息写入 users 表。最后，跳转到登录页面，并使用 flash()闪存注册成功信息。如果用户没有提交表单或是字段验证失败，则执行 render_template()函数显示注册页面。

用户注册失败的页面效果如图 3.4 所示，注册成功的页面效果如图 3.5 所示。


图 3.4　用户注册失败


图 3.5　用户注册成功

## 3.6.2　实现用户登录功能

在用户登录功能中，用户需要填写正确的用户名和密码，单击“登录”按钮，即可实现用户登录。如果没有输入用户名或者密码，将提示错误。另外，输入的账号和密码长度错误也将提示错误。

### 1．创建模板文件

在/templates/路径下创建 login.html 模板文件。由于登录页面表单比较简单，只有两个字段，所以这里没有使用 WTForms 表单框架验证字段，而是直接通过 jQuery 来实现，具体代码如下：

```
{% extends 'layout.html' %}
```

```
{% block body %}
<div class="content">
  <h1 class="title-center">用户登录</h1>
  <form action="" method="POST" onsubmit="return checkLogin()">
    <div class="form-group">
      <label>用户名</label>
      <input type="text" name="username" class="form-control" value={{request.form.username}}>
    </div>
    <div class="form-group">
      <label>密码</label>
      <input type="password" name="password" class="form-control" value="">
    </div>
    <button type="submit" class="btn btn-primary">登录</button>
  </form>
</div>

<script>
  function checkLogin(){
      var username = $("input[name='username']").val()
      var password = $("input[name='password']").val()
      // 检测用户名长度
      if ( username.length < 2   || username.length > 25){
        alert('用户名长度在 2～25 个字符')
        return false;
      }
      // 检测密码长度
      if ( username.length < 2   || username.length > 25){
        alert('密码长度在 6～20 个字符之间')
        return false;
      }
  }
</script>

{% endblock %}
```

在上述代码中，由于需要验证账号长度、密码长度，所以在 Form 表单中，设置 onsubmit 属性验证表单。当单击“登录”按钮时，调用 checkLogin()函数。如果 checkLogin()函数返回 False，则表示验证没有通过，不提交表单。否则，正常提交表单。

### 2．实现登录功能

当用户填写登录信息后，还需要验证用户名是否存在，以及用户名和密码是否匹配等内容。如果验证全部通过，需要将登录标识和 username 写入 session 中，为后面判断用户是否登录做准备。此外，还需要在用户访问/login 路由时，判断用户是否已经登录，如果用户之前已经登录，那么不需要再次登录，而是直接跳转到控制台。具体代码如下：

```
# 用户登录
@app.route('/login', methods=['GET', 'POST'])
def login():
    if "logged_in" in session:                                          # 如果已经登录，则直接跳转到控制台
        return redirect(url_for("dashboard"))

    if request.method == 'POST':                                        # 如果提交表单
        # 从表单中获取字段
        username = request.form['username']
        password_candidate = request.form['password']
        sql = "SELECT * FROM users   WHERE username = '%s'" % (username)# 根据用户名查找 user 表中记录
```

```
        db = MysqlUtil()                                          # 实例化数据库操作类
        result = db.fetchone(sql)                                 # 获取一条记录
        if result :                                               # 如果查到记录
            password = result['password']                         # 用户填写的密码
            # 对比用户填写的密码和数据库中的记录密码是否一致
            if sha256_crypt.verify(password_candidate, password):  # 调用 verify 方法验证，如果为真，则验证通过
                # 写入 session
                session['logged_in'] = True
                session['username'] = username
                flash('登录成功！', 'success')                      # 闪存信息
                return redirect(url_for('dashboard'))             # 跳转到控制台
            else:                                                 # 如果密码错误
                error = '用户名和密码不匹配'
                return render_template('login.html', error=error)  # 跳转到登录页，并提示错误信息
        else:
            error = '用户名不存在'
            return render_template('login.html', error=error)
    return render_template('login.html')
```

在上述代码中，先判断 logged_in（登录标识）是否存在于 session 中。如果存在，则说明用户已经登录，直接跳转到控制台。如果不存在，则后续判断用户名和密码都正确时，通过 session['logged_in']=True 语句将 logged_in 标识存入 session，方便下次使用。

此外，还需要注意的是，在判断用户提交的密码和数据库中的密码是否匹配时，需要使用 sha256_crypt.verify()进行判断。verify()方法的第一个参数是用户输入的密码，第二个参数是数据库中保存的加密后的密码，如果返回 True，则表示密码相同，否则密码不同。

登录时，用户名不存在的页面效果如图 3.6 所示，用户名和密码不匹配的页面效果如图 3.7 所示，登录成功的页面效果如图 3.8 所示。


图 3.6　用户名不存在


图 3.7　用户名和密码不匹配

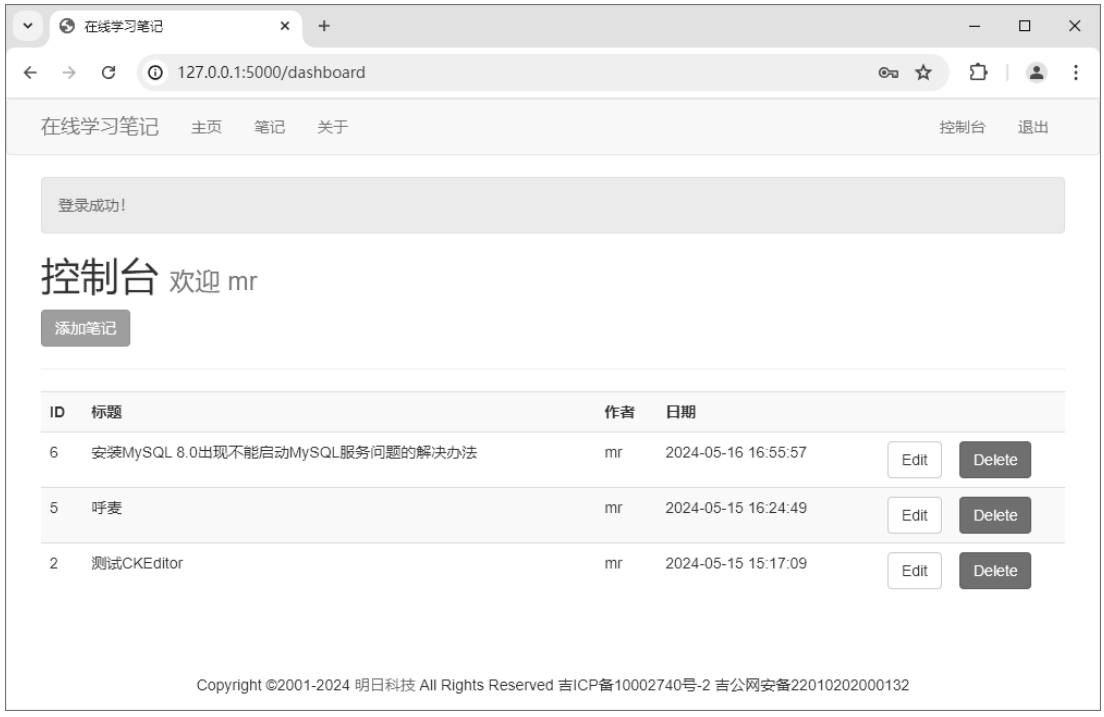



图 3.8　用户登录成功

## 3.6.3　实现退出登录功能

退出功能的实现比较简单，只是清空登录时保存在 session 中的值即可。使用 session.clear()函数来实现该功能。具体代码如下：

```
# 退出
@app.route('/logout')
@is_logged_in
def logout():
    session.clear()
    flash('您已成功退出', 'success')        # 闪存信息
    return redirect(url_for('login'))       # 跳转到登录页面
```

退出成功后，页面跳转到登录页。运行效果如图 3.9 所示。


图 3.9　退出登录页面效果

## 3.6.4　实现用户权限管理功能

在线学习笔记项目中，需要用户登录后才能访问的路由及说明如下。

- ☑ /dashboard：控制台。
- ☑ /add_article：添加笔记。
- ☑ /edit_article：编辑笔记。
- ☑ /delete_article：删除笔记。
- ☑ /logout：退出登录。

对于这些路由，可以在每一个方法中都添加如下

代码。

```
if 'logged_in' not in session:                  # 如果用户没有登录
    return redirect(url_for('login'))           # 跳转到登录页面
```

如果需要用户登录才能访问的页面很多，显然这种方式不够优雅。在此，可以使用装饰器的方式来简化代码。在 manage.py 文件中实现一个 is_logged_in 装饰器。代码如下：

```
# 如果用户已经登录
def is_logged_in(f):
    @wraps(f)
    def wrap(*args, **kwargs):
        if 'logged_in' in session:              # 判断用户是否登录
            return f(*args, **kwargs)           # 如果登录，则继续执行被装饰的函数
        else:                                   # 如果没有登录，则提示无权访问
            flash('无权访问，请先登录', 'danger')
            return redirect(url_for('login'))
    return wrap
```

定义完装饰器以后，就可以为需要用户登录的函数添加装饰器。例如，可以为 dashborad()函数添加装饰器，关键代码如下：

```
@app.route('/dashboard')
@is_logged_in
def dashboard():
    Pass
```

如果使用装饰器的方式，当执行 dashboard()函数时，优先执行 is_logged_in()函数判断用户是否登录。如果用户没有登录，则在浏览器中直接访问/dashboard，运行结果如图 3.10 所示。


图 3.10　未登录提示无权访问

# 3.7　笔记管理模块设计

笔记管理模块主要包括 4 部分功能：笔记列表、添加笔记、编辑笔记和删除笔记。因为用户必须登录后才能执行相应的操作，所以在每一个方法前添加@is_logged_in 装饰器来判断用户是否登录，如果没有登录，则跳转到登录页面。下面分别介绍每个功能的实现。

## 3.7.1　实现笔记列表功能

在控制台的笔记列表页面中，需要展示该用户的所有笔记信息。实现该功能的代码如下：

```
# 控制台
@app.route('/dashboard')
@is_logged_in
def dashboard():
    db = MysqlUtil()                            # 实例化数据库操作类
    sql = "SELECT * FROM articles WHERE author = '%s' ORDER BY create_date DESC" % (session
    ['username'])                               # 根据用户名查找用户笔记信息，并根据时间降序排序
    result = db.fetchall(sql)                   # 查找所有笔记
```

```
    if result:                                  # 如果笔记存在，则赋值给 articles 变量
        return render_template('dashboard.html', articles=result)
    else:                                       # 如果笔记不存在，则提示暂无笔记
        msg = '暂无笔记信息'
        return render_template('dashboard.html', msg=msg)
```

在上述代码中，需要注意使用 session()函数来获取用户名。如果用户登录成功，则使用 session['username'] = username 将 username 存入 session。所以，此时可以使用 session('username')来获取用户姓名。

接下来，使用 render_template()函数渲染模板文件。关键代码如下：

```
{% for article in articles %}
  <tr>
    <td>{{article.id}}</td>
    <td>{{article.title}}</td>
    <td>{{article.author}}</td>
    <td>{{article.create_date}}</td>
    <td><a href="edit_article/{{article.id}}" class="btn btn-default pull-right"> Edit</a></td>
    <td>
      <form action="{{url_for('delete_article', id=article.id)}}" method="post">
        <input type="hidden" name="_method" value="DELETE">
        <input type="submit" value="Delete" class="btn btn-danger">
      </form>
    </td>
  </tr>
{% endfor %}
```

在上述代码中，articles 变量表示所有笔记对象，使用 for 标签来遍历每一个笔记对象。

运行效果如图 3.11 所示。

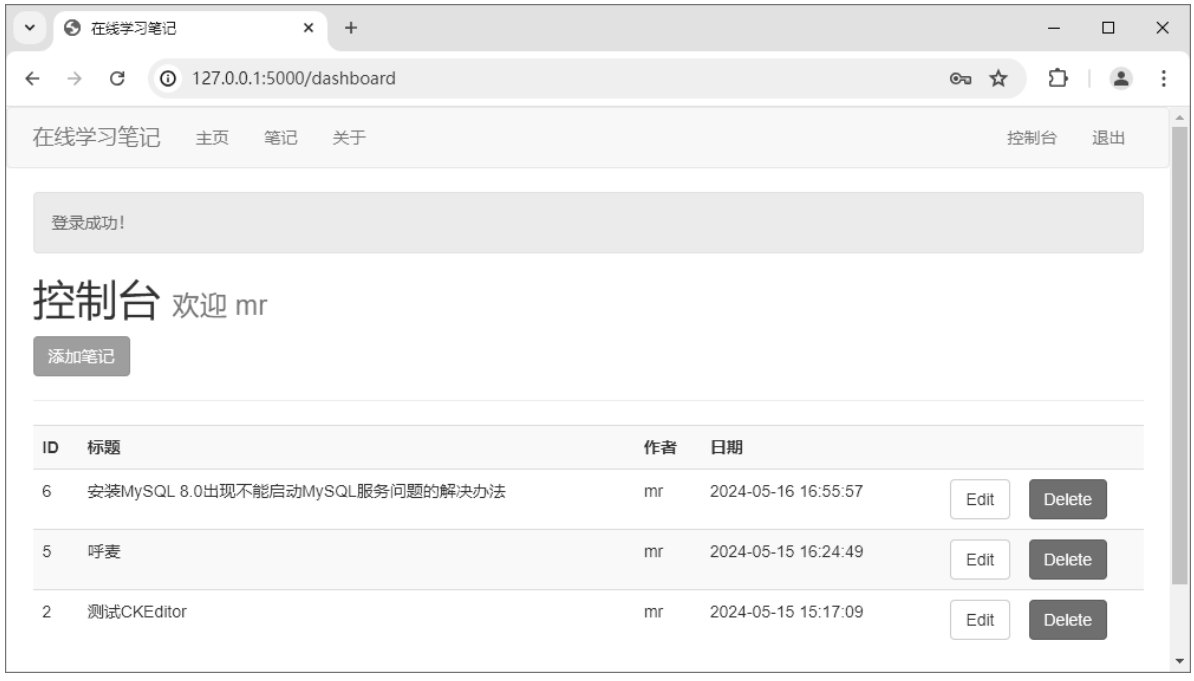



图 3.11　笔记列表页面

## 3.7.2　实现添加笔记功能

在控制台列表页面中单击“添加笔记”按钮，即可进入添加笔记页面。在该页面中，用户需要填写笔

记标题和笔记内容。实现该功能的关键代码如下：

```
# 添加笔记
@app.route('/add_article', methods=['GET', 'POST'])
@is_logged_in
def add_article():
    form = ArticleForm(request.form)                    # 实例化 ArticleForm 表单类
    if request.method == 'POST' and form.validate():    # 如果用户提交表单，并且表单验证通过
        # 获取表单字段内容
        title = form.title.data
        content = form.content.data
        author = session['username']
        create_date = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        db = MysqlUtil()                                # 实例化数据库操作类
        sql = "INSERT INTO articles(title,content,author,create_date) \
              VALUES ('%s', '%s', '%s','%s')" % (title,content,author,create_date)
        # 插入数据的 SQL 语句
        db.insert(sql)
        flash('创建成功', 'success')                     # 闪存信息
        return redirect(url_for('dashboard'))           # 跳转到控制台
    return render_template('add_article.html', form=form)   # 渲染模板
```

在上述代码中，接收表单的字段只包含标题和内容，此外，还需要使用 session()函数来获取用户名，使用 time 模块来获取当前时间。

在填写笔记内容时，使用了 CKEditor 编辑器替换普通的 textarea 文本框。CKEditor 编辑器和普通的 textarea 文本框的对比效果如图 3.12 所示。


图 3.12　CKEditor 和 textarea 效果对比

在 add_article.html 模板中使用 CKEditor 的关键代码如下：

```
{% extends 'layout.html' %}

{% block body %}
  <h1>添加笔记</h1>

  {% from "includes/_formhelpers.html" import render_field %}
  <form method="POST" action="">
    <div class="form-group">
      {{ render_field(form.title, class_="form-control") }}
    </div>
    <div class="form-group">
        <textarea id="editor" name="content"></textarea>
    </div>
    <p><input class="btn btn-primary" type="submit" value="提交">
  </form>
<script src="{{ url_for('static', filename='js/ckeditor.js') }}"></script>
<script>
    // 初始化 CKEditor
    ClassicEditor.create(document.querySelector('#editor'))
        .then(editor => {
            console.log('CKEditor 5 初始化成功！');
```

```
        })
        .catch(error => {
            console.error('CKEditor 5 初始化失败:', error);
        });
</script>
{% endblock %}
```

在上述代码中，首先在 Form 表单的文本域中设置 id="editor"，然后引入 ckeditor.js ，最后在 JavaScript 中通过 ClassicEditor.create()方法初始化 CKEditor。create()函数的参数就是表单中文本域字段，这里是通过 document.querySelector()方法根据文本域的 ID 值指定的。

添加笔记的运行效果如图 3.13 所示。


图 3.13　添加笔记

## 3.7.3　实现编辑笔记功能

在控制台列表中，单击笔记标题右侧的 Edit 按钮，即可根据笔记的 ID 进入该笔记的编辑页面。编辑页面和添加页面类似，只是编辑页面需要展示被编辑笔记的标题和内容。实现该功能的关键代码如下：

```
# 编辑笔记
@app.route('/edit_article/<string:id>', methods=['GET', 'POST'])
@is_logged_in
def edit_article(id):
    db = MysqlUtil()                                    # 实例化数据库操作类
    fetch_sql = "SELECT * FROM articles WHERE id = '%s' and author = '%s'" % (id,session
     ['username'])                                      # 根据笔记 ID 查找笔记信息
    article = db.fetchone(fetch_sql)                    # 查找一条记录
    # 检测笔记不存在的情况
    if not article:
        flash('ID 错误', 'danger')                      # 闪存信息
        return redirect(url_for('dashboard'))
    # 获取表单
    form = ArticleForm(request.form)
    if request.method == 'POST' and form.validate():    # 如果用户提交表单，并且表单验证通过
```

```
        # 获取表单字段内容
        title = request.form['title']
        content = request.form['content']
        update_sql = "UPDATE articles SET title='%s', content='%s' WHERE id='%s' and author
        = '%s'" % (title, content, id,session['username'])
        db = MysqlUtil()                                    # 实例化数据库操作类
        db.update(update_sql)                               # 更新数据的 SQL 语句
        flash('更改成功', 'success')                          # 闪存信息
        return redirect(url_for('dashboard'))               # 跳转到控制台

    # 从数据库中获取表单字段的值
    form.title.data = article['title']
    form.content.data = article['content']
    # 渲染模板
    return render_template('edit_article.html', form=form,mycontent=form.content.data)
```

在上述代码中，首先根据笔记的 ID 查找 articles 表中的笔记信息。如果 articles 表中没有此 ID，则提示错误信息。接下来，判断用户是否提交表单，并且表单验证通过。如果同时满足以上两个条件，则修改该 ID 的笔记信息，并跳转到控制台，否则，获取笔记信息后渲染模板。

编辑笔记的运行效果如图 3.14 所示。


图 3.14　编辑笔记

## 3.7.4　实现删除笔记功能

在控制台列表中，单击笔记标题右侧的 Delete 按钮，即可根据笔记的 ID 删除该笔记。删除成功后，页面跳转到控制台。实现该功能的关键代码如下：

```
# 删除笔记
@app.route('/delete_article/<string:id>', methods=['POST'])
@is_logged_in
def delete_article(id):
    db = MysqlUtil()                                    # 实例化数据库操作类
    sql = "DELETE FROM articles WHERE id = '%s' and author = '%s'" % (id,session['username'])
```

```
    # 执行删除笔记的 SQL 语句
    db.delete(sql)
    flash('删除成功', 'success')            # 闪存信息
    return redirect(url_for('dashboard'))  # 跳转到控制台
```

在上述代码中，执行删除的 SQL 语句一定要添加 WHERE id 限定条件，否则，将删除所有笔记。

# 3.8 项 目 运 行

通过前述步骤，设计并完成了“在线学习笔记”项目的开发。下面运行该项目，检验一下我们的开发成果。运行“在线学习笔记”项目的步骤如下。

（1）打开 Notebook\mysql_util.py 文件，根据自己的数据库账号和密码修改如下代码：

```
def __init__(self):
    '''
        初始化方法，连接数据库
    '''
    host = '127.0.0.1'                                                    # 主机名
    user = 'root'                                                         # 数据库用户名
    password = 'root'                                                     # 数据库密码
    database = 'notebook'                                                 # 数据库名称
    self.db = pymysql.connect(host=host,user=user,password=password,db=database) # 建立连接
    self.cursor = self.db.cursor(cursor=pymysql.cursors.DictCursor)    # 设置游标，并将游标设置为字典类型
```

（2）打开命令提示符对话框，进入 Notebook 项目文件夹所在目录，在命令提示符对话框中输入如下命令来创建 venv 虚拟环境：

```
virtualenv venv
```

（3）在命令提示符对话框中输入如下命令来启动 venv 虚拟环境：

```
venv\Scripts\activate
```

（4）在命令提示符对话框中使用如下命令来安装 Flask 等依赖包：

```
pip install -r   requirements.txt
```

（5）创建数据库。可以使用 MySQL 命令行方式或 MySQL 可视化管理工具（如 Navicat）创建数据库。例如，在 MySQL 命令行窗口（MySQL 8.0 Command line Client）中输入登录密码后，使用下面的命令来实现。

```
create database notebook default character set utf8;
```

（6）执行 SQL 脚本文件 Code\Notebook\notebook.sql，创建数据表并插入数据。例如，在 MySQL 命令行窗口（MySQL 8.0 Command line Client）中输入登录密码后，使用下面的命令来实现。

```
use notebook
SOURCE D:\Code\Notebook\notebook.sql
```

（7）在 PyCharm 的左侧项目结构中展开“在线学习笔记”的项目文件夹 Notebook，在其中选中 manage.py 文件，单击鼠标右键，在弹出的快捷菜单中选择 Run 'manage'命令，如图 3.15 所示。

（8）如果在 PyCharm 底部出现如图 3.16 所示的提示，说明程序运行成功。

（9）在浏览器中输入网址 http://127.0.0.1:5000/即可进入在线学习笔记的首页，效果如图 3.17 所示。在该页面中，不登录时，可以浏览他人的笔记，注册并登录后，可以添加、编辑或删除自己的笔记。

本章主要使用 Flask 框架开发一个在线学习笔记的网站。在项目使用了很多开发中常用的模块和方法，例如，使用 WTFomrs 模块验证表单，使用 passlib 模块对密码加密，以及使用装饰器判断用户是否登录等。

通过本章的学习，希望读者能够了解 Flask 开发流程并掌握 Web 开发中常用的模块。

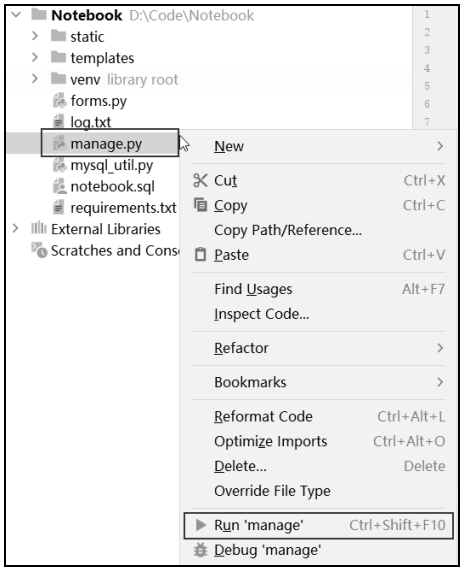



图 3.15　选择 Run 'manage'

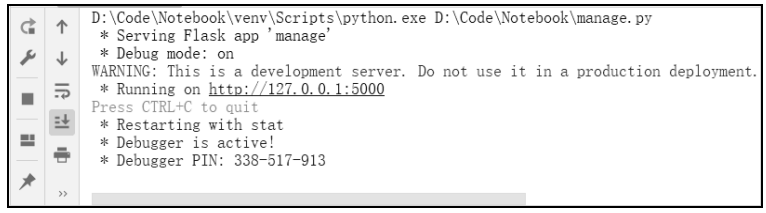



图 3.16　程序运行成功提示


图 3.17　在线学习笔记的首页

## 3.9　源 码 下 载

本章虽然详细地讲解了如何编码实现“在线学习笔记”的各个功能，但给出的代码都是代码片段，而非完整的源代码。为了方便读者学习，本书提供了该项目的完整源代码，读者可以通过扫描右侧的二维码进行下载。

源码下载