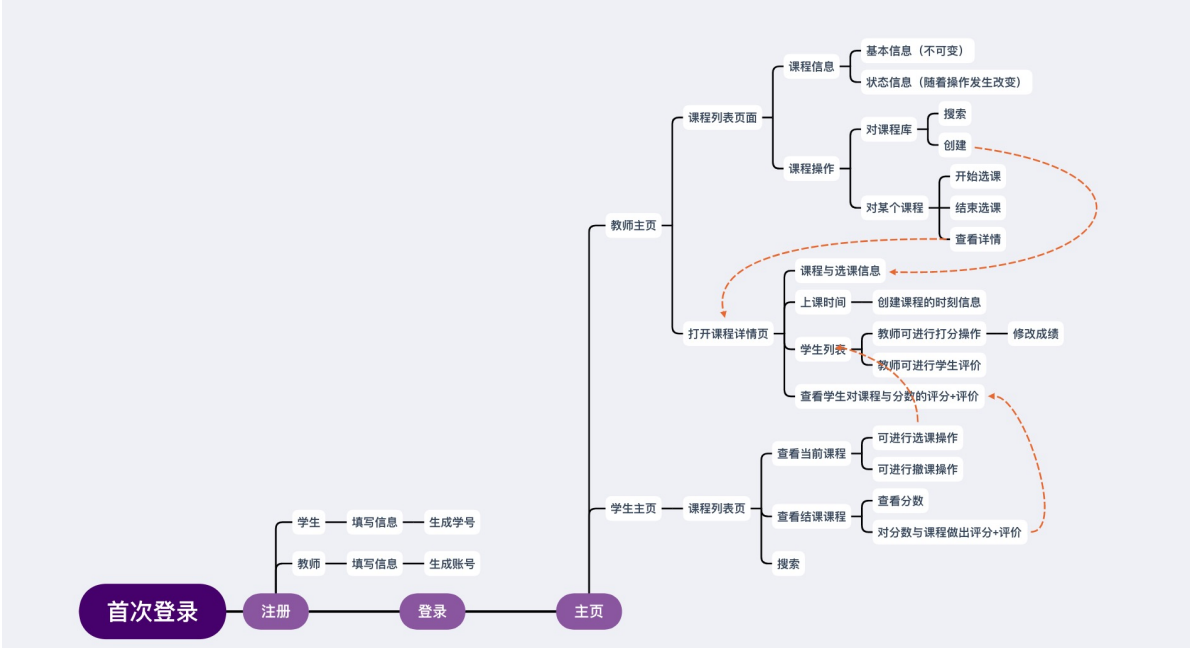
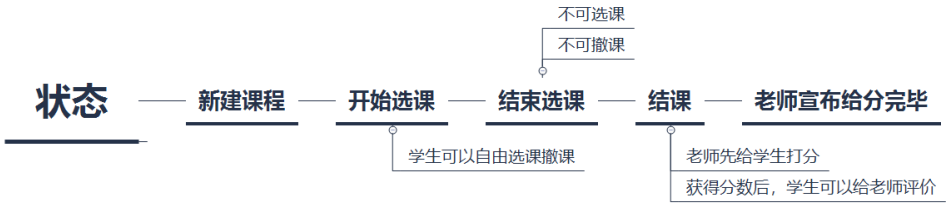
基于Python-Django框架实现的学生选课管理系统

思路概述图：



状态转换示意图：



一、新建项目并进行设置

1.新建项目

本项目名字为SSCMS

`django-admin startproject SSCMS`

运行后文件夹结构如下

```
SSCMS
|--manage.py
|--SSCMS
   |--settings.py
   |--urls.py
   |--wsgi.py
   |--__init__.py
```

2.检查设置文件settings.py

- INSTALLED_APPS

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

- MIDDLEWARE

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

- TEMPLATES

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

- DATABASES

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

3.新建应用（app）

- 何为应用?

  一个项目中承担了不同的功能的子模块，这些子模块可认为是项目的应用。

新建用户模块

新建一个模块名字为user的用户模块（应用）

`py manage.py startapp user`

在setting.py的INSTALLED_APPS中添加名为user的应用

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'user',
]
```

新建一个模块名字为course的课程模块（应用）

`py manage.py startapp course`

在setting.py的INSTALLED_APPS中添加名为course的应用

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'user',
    'course',
]
```

4.建立相关文件夹

为项目添加template和static文件夹

- template文件夹用于存放模板
- static文件夹用于存放静态文件

此时项目的文件夹架构如下

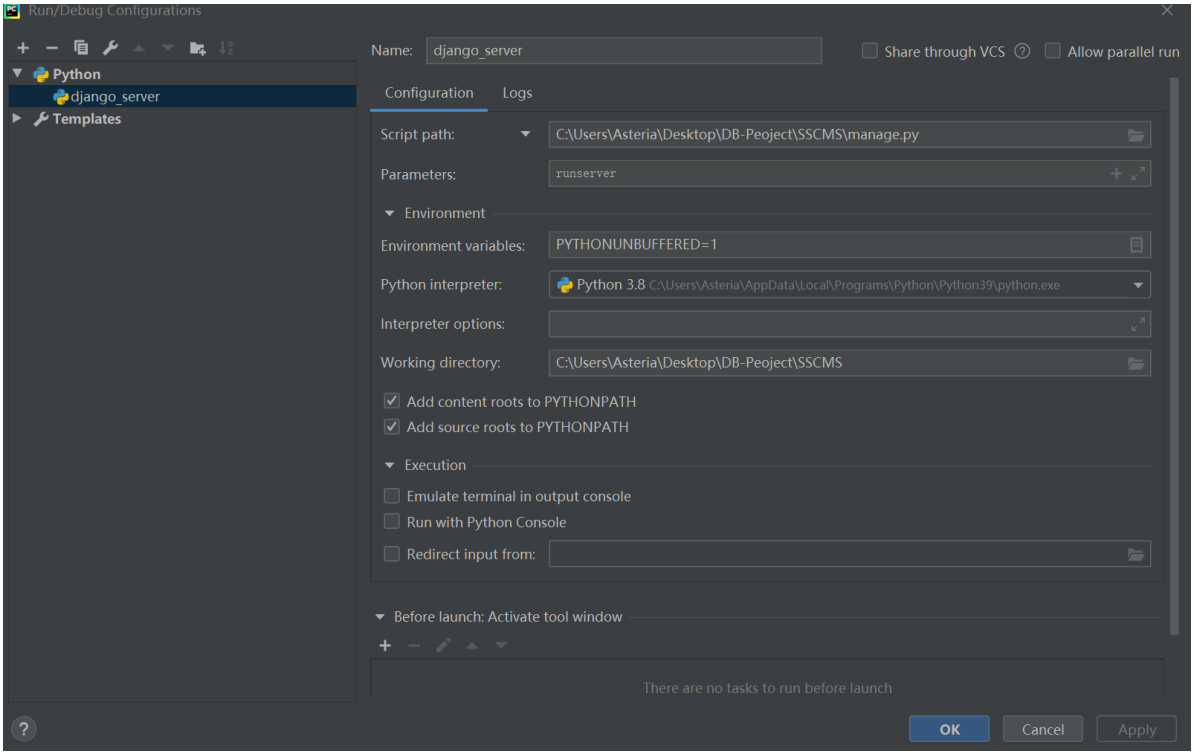
```
SSCMS
|--manage.py
|--SSCMS
   |--settings.py
   |--urls.py
   |--wsgi.py
   |--__init__.py
```

```
|--static
|--templates
|--user
   |--admin.py
   |--apps.py
   |--migrations
      |--__init__.py
   |--models.py
   |--tests.py
   |--views.py
   |--__init__.py
|--course
   |--admin.py
   |--apps.py
   |--migrations
      |--__init__.py
   |--models.py
   |--tests.py
   |--views.py
   |--__init__.py
```

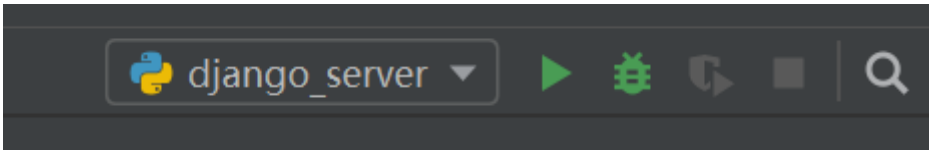
5.运行项目

（1）添加可运行的服务器配置

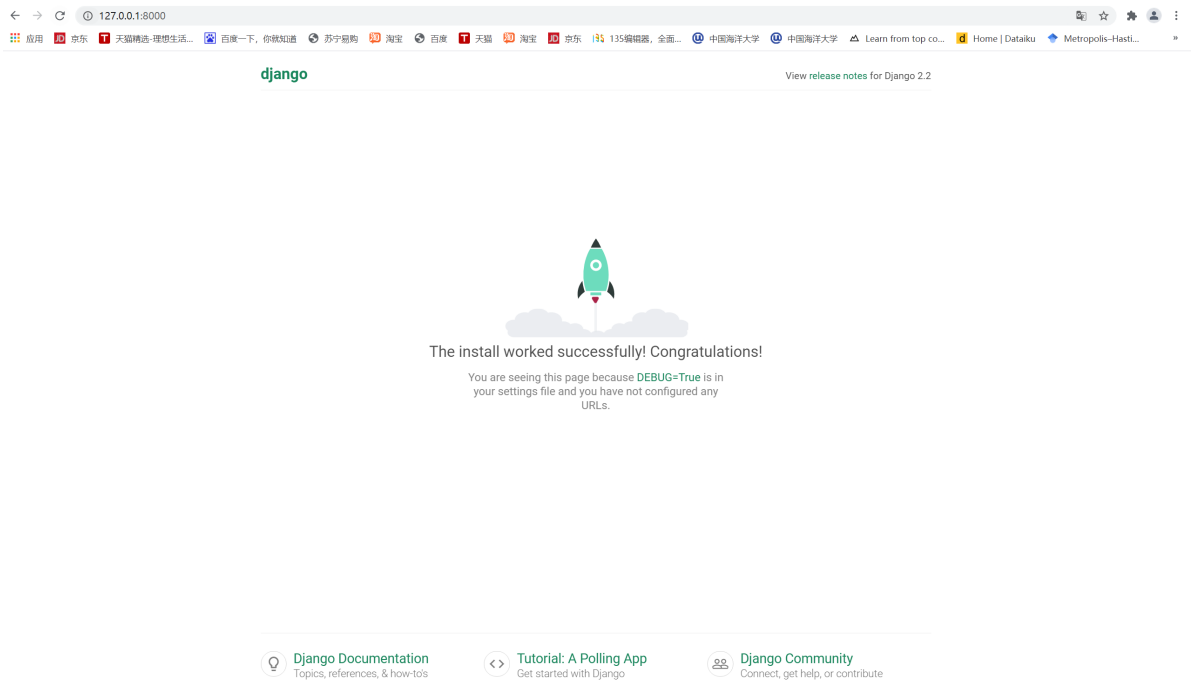
Edit Configurations中配置服务器，名字设置为django_server，脚本路径为manage.py的路径，运行变量名设置为runserver，设置完成后点击Apply按钮



点击绿色箭头运行



```
Run 'python manage.py migrate' to apply them.
May 04, 2021 - 22:43:04
Django version 2.2.11, using settings 'SSCMS.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```



二、创建用户模型

1.Django中的模型（Models)介绍与导入

> 模型是关于数据的唯一、确定的信息源。它包含存储数据的基本字段和行为。通常，每个模型映射到一个数据库表。——官方文档

建立一个模型（Model）相当于建立了一个数据库表（table）。

模型规定属性，就是数据库表规定字段（field）。

- 每个数据库表（table），都是用来记录一种事物的数据信息的，比如学生表，是用来记录学生这种对象的多个维度的信息。每个信息维度（比如姓名、性别、生日、邮箱、学号）对应数据库表的一个字段（field）。
- 对于学生这种对象，我们可以建立一个模型类（Model），模型的每个属性对应学生一个信息维度（比如姓名、性别、生日、邮箱、学号）。
- 数据库表的每一行，都是一个具体的学生的信息，对应也就是模型类（Model）的一个实例。

首先，打开项目下的`./user/models.py`文件，其中的初始内容为

```
from django.db import models
```

也就是导入了models类

2.学生对象的创建

属性：

- 姓名（name）
- 性别（gender）
- 生日（birthday）
- 邮箱（email）
- 个人简介（info）
- 年级（grade）
- 年级子学号（number）
- 密码（password）

一个学生能由年级+年级子学号唯一确定，则**年级+年级子学号**为其主键，定义为学号(student_id)。

在`models.py`中添加学生模型

- 学生年级号为4位数字组成的字符串，年级下子学号为6位数字组成的字符串。
  这两个连接起来组成学生的唯一学号，该学号也为其登录使用的账号。

```
class Student(models.Model):
    genders = [("m","男"),("f","女")]
    name = models.CharField(max_length=50, verbose_name="姓名")
    gender = models.CharField(max_length=10, choices=genders, default='m',
verbose_name="性别")
    birthday = models.DateField(verbose_name="生日")
    email = models.EmailField(verbose_name="邮箱")
    info = models.CharField(max_length=255, verbose_name="个人简介", help_text="一
句话介绍自己，不要超过250个字")

    grade = models.CharField(max_length=4, verbose_name="年级")
    number = models.CharField(max_length=6, verbose_name="年级子学号")
    password = models.CharField(max_length=30, verbose_name="密码")

    class Meta:
        contraints = [
            #复合主键：保证grade和number组合的stude_id唯一
            models.UniqueConstraint(fields=['grade','number'],name='student_id')
        ]

    def get_id(self):
        return "%s%s" % (self.grade, self.number)

    def __str__(self):
        return "%s (%s)" % (self.get_id(), self.name)
```

3.老师模型的创建

属性：

- 姓名（name）
- 性别（gender）
- 生日（birthday）
- 邮箱（email）

- 教师简介（info）
- 院系号（department_no）
- 院内编码（number）
- 密码（password）

一个老师能由院系号+院内编码唯一确定，则**院系号+院内编码**为其主键，定义为教师号(teacher_id)。

在`models.py`中添加教师模型

```
class Teacher(models.Model)
    genders = [("m","男"),("f","女")]
    name  = models.CharField(max_length=50, verbose_name="姓名")
    gender = models.CharField(max_length=10, choices=genders, default="m",
verbose_name="性别")
    birthday = models.DateField(verbose_name="生日")
    email = models.EmailField(verbose_name="邮箱")
    info = models.CharField(max_length=255, verbose_name="教师简介", help_text="不
要超过250个字")

    department_no = models.CharField(max_length=3, verbose_name="院系号")
    number = models.CharField(max_length=7, verbose_name="院内编号")
    password = models.CharField(max_length=30, verbose_name="密码")

    class Meta:
        contraints = [
            #复合主键：保证department_no和number组合的teacher_id唯一
            models.UniqueConstraint(fields=['department_no','number'],
name="teacher_id")
        ]

    def get_id(self):
        return "%s%s" % (self.department_no, self.number)

    def __str__(self):
        return "%s (%s)" % (self.get_id(),self.name)
```

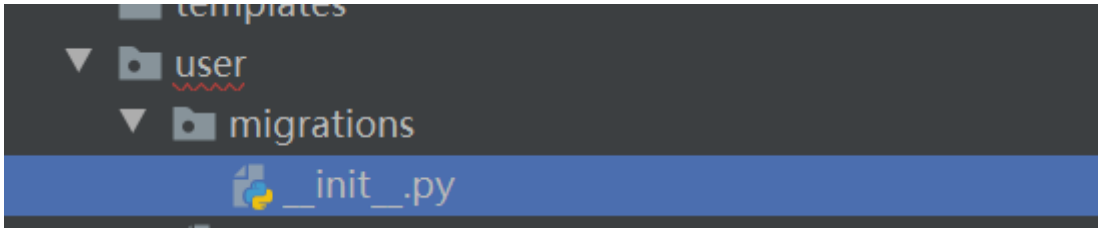
3.建立（更新）数据库

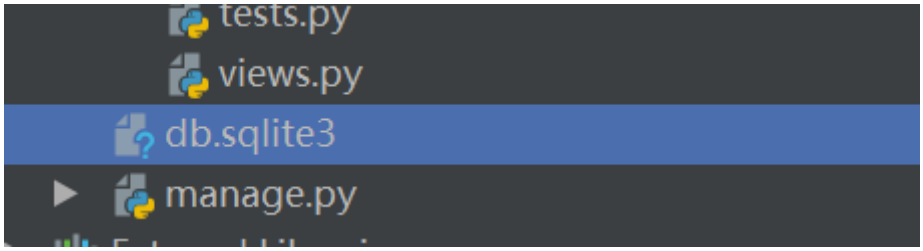
添加好模型后，我们还需要手动执行脚本，才能根据模型生成对应的数据库表。
在项目文件夹下，打开命令行，按行依次执行：

```
python manage.py makemigrations
python manage.py migrate
```

第1句会生成对应的迁移（migrations）命令。

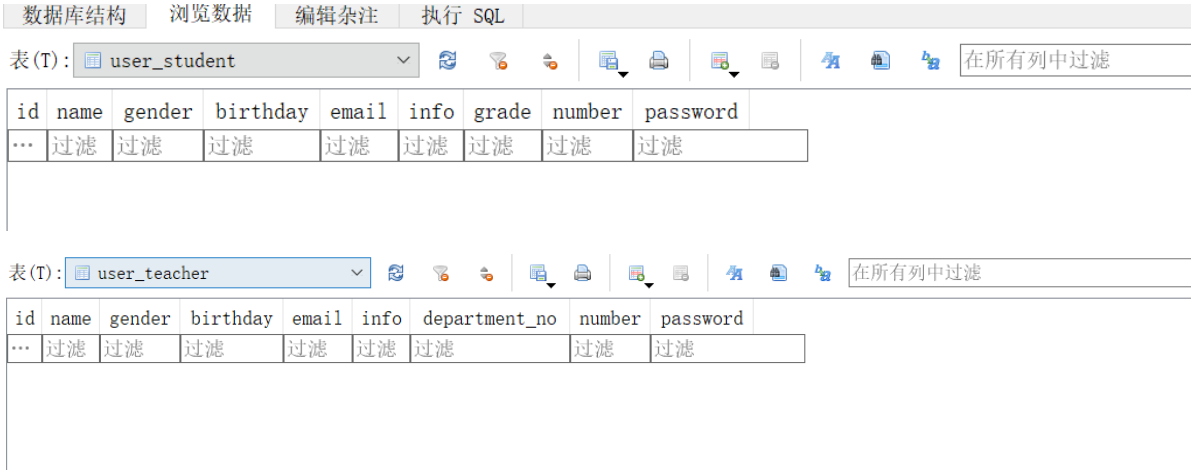
具体到本项目，由于是第一次运行，那么会在`./user/migrations`文件夹下， 生成`0001_initial.py`用于记录迁移（migrations）命令
第一次运行也会在项目文件夹下，生成一个空的 database：`db.sqlite3`





第2句会执行第一句中生成的迁移（migrations）命令。
执行完第二句，改动才真正更新到数据库文件了。
对应到本项目，就是数据库中添加了两个模型（学生、老师）对应的数据库表。

以后如果修改了模型的属性之类，也要执行上面两句脚本去更新对应的数据库表。



三、实现登录界面

1.Django的MVT软件设计模式

- Model（模型）--负责数据（第二部分实现的内容）
- View（视图）--负责逻辑
- Template（模板）--负责前端页面

实现页面的基本步骤：

完成template→实现view→设置url

2.Django处理请求（request）过程:url - > urlpattern - > view

(1)根据ROOT_URLCONF(位于setting.py)选择对应的url解析模块（默认是urls.py)

(2)加载上一步找到的url解析模块(默认是urls.py)查找变量urlpatterns

(3)按照顺序遍历urlpatterns里面的urlpatten，返回第一个匹配requested URL的 urlpattern

- urlpattern:变量urlpatterns里面的元素，它可以通过django.urls.path()函数或django.urls.re_path()函数获得

```
path(route, view, kwargs=None, name=None)
re_path(route, view, kwargs=None, name=None)
```

- route:要匹配的url
- view:匹配后要调用的view函数名
- kwargs:向view函数传递额外的参数（route也可以传递参数给view）

- name:urlpattern的名字，主要用于后面在模板中指定urlpattern

(4)上一步匹配成功，Django就会导入并调用urlpattern里面的对应的视图view(一般在views.py中)

- view可以看做是一个简单的Python函数，这个方法会根据一个HTTPrequest实例(对应输入参数request)，返回一个HTTPresponse实例
    - request是视图view中一定要有的参数，根据需要，urlpattern也有可能会给视图view添加其他输入参数

(5)第三步没有找到匹配的，或者寻找过程中没有异常抛出，Django会调用适当的错误处理视图

例子

- urls.py（项目文件夹下）

```
from django.urls import path
from my_app import views

urlpatterns = [
    path('hello/', views.hello),
]
```

- views.py（应用my_app文件夹下）

```
from django.http.response import HttpResponse

def hello(request):
    return HttpResponse("Hello")
```

Hello

访问过程的步骤：

地址的构成：

- **127.0.0.1:8000/**：浏览器通过127.0.0.1这个域名找到对应的服务器（127.0.0.1表示的是本地服务器），
  给这个服务器上8000这个端口号发送访问请求
- **hello/**： 路径，用于给Django程序匹配urlpattern

Django后台受到这个访问请求后：

(1)根据ROOT_URLCONF(位于setting.py)选择对应的url解析模块（默认是urls.py)

(2)加载上一步找到的url解析模块(默认是urls.py)查找变量urlpatterns

(3)按照顺序遍历urlpatterns里面的urlpattern，path('hello/', views.hello)可以匹配到equested URL的hello/

- urlpattern:变量urlpatterns里面的元素，它可以通过django.urls.path()函数或django.urls.re_path()函数获得

(4)上一步匹配成功，Django就会导入并调用urlpattern里面的对应的视图view中的hello方法，返回给浏览器一个HttpResponse("Hello")

3.访问的时候的参数传递

将上个例子中的函数传递方法变得更加复杂一点：比如要将`http://127.0.0.1:8000/hello/Bob`中的`Bob`作为参数的话，那么可以这么写

```
path('hello/<slug:name>', views.hello)
```

- 如果需要从url捕获字符串作为参数传递给view函数，需要使用尖括号，尖括号内写捕获后的变量名。
  - 比如`'hello/<name>'`就会将`name`对应位置的字符串捕获并以`name`作为变量名传入view函数。
- 捕获的值可以选择性地包括转换器类型，设置后会将捕获后的字符串转换为对应类型。
  - 例如，使用`<int:value>`来捕获整型参数， 其中`:`前为转换器类型，`:`后为捕获后的变量名。（如果不包含转换器，则匹配除`/`字符外的任何字符串）
  - Django中的默认转换器
    - `str`：转换器默认值，匹配除路径分隔符`'/'`外的任何非空字符串
    - `int`：匹配零或任何正整数，返回一个`int`
    - `slug`：匹配由ASCII字母或数字以及连字符和下划线组成的任何`slug`字符串
    - `uuid`：匹配格式化的UUID，返回一个`UUID`实例
    - `path`：匹配任何非空字符串，包括路径分隔符'/'

views.py中的hello方法也要添加一个参数用于去接受这个传入的变量

```
def hello(request, name):
    return HttpResponse("Hello, %s" % name)
```

4.不同app的url的处理方式

对于一个网站系统而言，可能会有很多功能模块app，每个功能模块app又有各自的urlpatterns。如果这些全部都放在项目文件夹project_name下的项目名文件夹project_name/project_name中的urls.py中的话，会很混乱也不方便管理，所以最好的办法是，每个功能模块app对应的urlpatterns放在app文件夹下的，然后在project_name/project_name中的urls.py使用include方法进行导入。
使用include方法的好处是可以将一组url植根到其他的url下。

5.view函数

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello")
```

解释：

(1)从`django.http`模块中导入`HttpResponse`类

(2)接下来，我们定义一个名为`hello`的函数。 这是视图功能。 每个视图函数都将`HttpRequest`对象作为其第一个参数，通常将其命名为`request`

- *视图函数的名称并不重要;它不需要遵循特定的方式命名才能让Django识别它。这里我们将它称为hello，因为这个名称清楚地表明了它的作用*

(3)该视图返回一个`HttpResponse`对象，其中包含生成的响应，这里是一个简单的`"Hello"`文本。每个视图函数都需要返回一个HttpResponse对象。(也有例外情况)

- *返回一个`HttpResponse`对象，不代表视图函数一定要写成`return HttpResponse("...")`，也可以返回一个看起来像其他东西，但实际是`HttpResponse`对象的或者能生成`HttpResponse`对象的函数*

HttpRequest介绍

当页面被请求时，Django会自动创建一个包含请求元数据的`HttpRequest`对象。然后Django加载适当的视图，将`HttpRequest`作为第一个参数传递给视图函数。视图函数里，通常将其命名为`request`。

常用属性:

- method
  - 请求中使用的HTTP方法的字符串（这个字符串是大写的）。这个属性常用于判断这是什么请求

```
if request.method == 'GET':
    do_something()
elif request.method == 'POST':
    do_something_else()
```

- GET
  - 一个类似字典的对象QueryDict，包含所有给定的HTTP GET参数。HTTP GET参数即get请求通过url传递的参数。以之前的视图函数和url为例，访问http://127.0.0.1:8000/hello?a=1&a=2&c=3， 其中?后面是get请求传递的参数，所以该request的GET为<QueryDict: {'a': ['1', '2'], 'c': ['3']}>，其中的键值对获取方法和字典的语法等同
- POST
  - 一个类似字典的对象`QueryDict`，包含所有给定的HTTP POST参数，前提是请求包含表单数据

HttpResponse介绍

与Django自动创建的`HttpRequest`对象不同，`HttpResponse`对象由开发者负责编写对应的代码去生成。具体来说，开发者要在每个视图中，都负责实例化、填充和返回`HttpResponse`

`HttpResponse`对象的内容可以是一个纯文本，比如上面的`HttpResponse("Hello")`

`HttpResponse`对象的内容也可以是html文本（通常都是），比如

```
def hello(request, name):
    html = "<html><body>Hello, %s</body></html>" % name
    return HttpResponse(html)
```

为了保证可读性，html文本一般都会放在专门的html文件中。借助模板(template)，能够由视图函数中的name变量动态生成HTML中的对应文本。

6.模板(template)的介绍

功能：让Django能够自动生成HTML代码

模板（Template）里面多了一些Tags（标签）语法，这在html中是没有的

- 其写法是`{%`符号开头，以`%}`符号结尾，这两个符号和内部的文本用空格相隔。

```
{% block welcome_message %}
{% endblock %}

{% block login_container %}
{% endblock %}

{% extends "user/background.html" %}
{% block login_container %}
{% endblock %}
```

7.初级表单的使用

表单的作用：让用户输入并提交数据

HTML中的表单：

Django中的表单：`django.forms.Form`

例子

(1)编写表单类

form.py：

```
#usr/bin/env python
#-*- coding:utf-8- -*-
from django import forms

class LoginForm(forms.Form):
    uid = forms.CharField(label='ID', max_length=10)
    password = forms.CharField(label='password', max_length=30,
widget=forms.PasswordInput)
```

- 从Django库中导入了form库。在创建表单时，需要用到form库中的方法：
  - `forms.Form`：必用，继承该表单类去创建自己需要的表单
  - 各种`Field`：几乎必用，定义自己表单类中需要的字段
  - 各种`widget`：常用，定义表单字段的一些规则。
- class的作用：继承`forms.Form`编写自己的表单类`LoginForm`
  - 新表单类的命名一般以`Form`结尾，按常规的命名类的规则，表达清楚该表单的功能和意义即可。
  - class中设置表单需要的字段（Field的用法与models中的用法很类似）
    - `forms`中的`Field`用`label`指定，`models`中的`Field`用`verbose_name`指定
    - `orms`中的`Field`比`models`中的`Field`多了`widget`属性
  - widget属性`forms.PasswordInput`代表这是个密码字段，填写时会隐藏其输入的内容

view.py：

```python
#usr/bin/env python
#-*- coding:utf-8- -*-
from django.shortcuts import render
from django.http.response import HttpResponse
from my_app.forms import LoginForm


def page(request):
    if request.method == 'POST':
        form = LoginForm(data=request.POST)
        if form.is_valid():
            uid = form.cleaned_data["uid"]
            return HttpResponse(uid)
    else:
        form = LoginForm()
    return render(request, 'login.html', {'form': form})
```

- `form = LoginForm()` 新建表单对象，即实例化表单
    - 这是一个没有数据的空表单，用户提交表单信息的页面，一般使用的是这样的空表单
- 当用户填写表单信息之后，提交表单时，必须使用POST方法提交， 此时视图中，可以直接使用表单数据来新建一个表单对象，代码为 `form = LoginForm(data=request.POST)`
    - 不过用户提交的表单信息不一定就是符合规则的，这个时候需要使用表单类的 `is_valid` 方法对表单进行一个数据检查 `form.is_valid()`，返回值为 `True` 则代表其数据是通过验证的
    - 通过验证后，我们将可以使用表单类的 `cleaned_data` 属性，找到所有已验证的表单数据
        - 而表单类的 `cleaned_data` 属性，是一个字典。
        - 获取其中的属性： `uid = form.cleaned_data["uid"]`
- `render` 方法用于传递给模板进行渲染

  `return render(request, 'login.html', {'form': form})`

login.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        Login
    </title>
</head>
<body>
    <div>Log in</div>
    <div>
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <p><input type="submit" value="Log in"/></p>
        </form>
    </div>
</body>
</html>
```

视图方法给Django Template(模板)中传入一个 `form` 对象后，想要在模板中渲染出来，
可以直接使用 `{{ form }}`，该语法将适当地渲染 `form` 对象的和元素。

除了`<label>/<input>`对，还有其他的输出选项:

- `{{ form.as_table }}`将把它渲染成包含在`<tr>`标签中的表格单元格
- `{{ form.as_p }}`将把它们包装在`<p>`标签中
- `{{ form.as_ul }}`将把它们包装在`<li>`标签中

注意:

- 表单的渲染输出不包括周围的`<form>`、`<table>`或者`<ul>`标记，也不包括表单的`submit`控件，这些需要我们去写。
- 其实直接使用`{{ form }}`和使用`{{ form.as_table }}`是一样的，前者是通过默认调用后者实现的。

添加主页的HTML文件

在项目的template文件下，新建user文件夹，添加`background.html`内容

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>
        学生选课管理系统
    </title>
</head>
<body>
<div class="main-container">
    <div class="main-header">
        <div class="main-title">学生选课管理系统</div>
        <div class="sub-title">Student Course Management System</div>
        {% block welcome_message %}
        {% endblock %}
    </div>
    <div class="login-container">
        {% block login_container %}
        {% endblock %}
    </div>
</div>
</body>

</html>
```

然后再添加`login_home.html`文件，内容如下

```
{% extends "user/background.html" %}
{% block login_container %}
<div class="login-kind student-login-button">
    <a href="">学生登录</a>
</div>
<div class="login-kind teacher-login-button">
    <a href="">教师登录</a>
</div>
{% endblock %}
```

8.使用模型创建表单

Django提供了一个辅助类`ModelForm`，帮助我们从Django中的models（模型）创建表单类

在`models.py`有这样一个学生类

```
from django.db import models

class Student(models.Model):
    gender = [
        ("m", "男"),
        ("f", "女")
    ]

    name = models.CharField(max_length=50, verbose_name="姓名")
    gender = models.CharField(max_length=10, choices=gender, default='m',
verbose_name="性别")
    birthday = models.DateField(verbose_name="生日")
    email = models.EmailField(verbose_name="邮箱")
    info = models.CharField(max_length=255, verbose_name="个人简介", help_text="一
句话介绍自己，不要超过250字")

    grade = models.CharField(max_length=4, verbose_name="年级")
    number = models.CharField(max_length=6, verbose_name="年级子学号")
    password = models.CharField(max_length=30, verbose_name="密码")
```

我们要给这个学生类建立一个表单，用于在注册页面提交注册信息。
那么`forms.py`代码如下

```
from django import forms
from .models import Student

class StuRegisterForm(forms.ModelForm):

    class Meta:
        model = Student
        fields = ('grade',
                  'name',
                  'password',
                  'confirm_password',
                  'gender',
                  'birthday',
                  'email',
                  'info')
```

- 其中`Meta`是元数据类，用于去编辑设置一些更深层次的设置。 要使用一个模型来创建表单，则在`Meta`元数据类中指定对应的`model`属性
- 对于和模型紧密映射的表单，有时我们不希望有些字段能够被用户编辑（比如一些需要后台按照逻辑去生成的字段）。这个时候我们可以使用：
    - `fields`属性，设置哪些字段是可以让用户进行编辑的
    - `exclude`属性，设置哪些字段是不让用户进行编辑的（用户在表单中也看不到这个字段）

上面两个属性一般使用列表（元组也可以），包含其需要指定的字段名。

`fields`属性可设置为特殊值'**all**'，以指示应该使用模型中的所有字段

补充：如果你在模型字段上设置了`edititable=False`，那么通过ModelForm从模型创建的任何表单都不会包含该字段。

重写clean方法

- 实现确认密码

一般来讲，注册账号的页面，都需要用户填写两次密码进行确认。那么我们这里需要对上面的代码进行一个拓展。
首先是要新增一个确认密码的字段，这个很简单，在`StuRegisterForm`中添加这样一行即可：

`confirm_password = forms.CharField(widget=forms.PasswordInput(), label="确认密码")`

- 验证第二次输入的密码与之前的密码字段内容是一致的

我们需要重写下这个表单类的验证方法。

这里我们只介绍下需要用到的：
表单子类的`clean()`方法，该方法可以执行需要访问多个表单字段的验证。
重写后的`StuRegisterForm`如下

```
class StuRegisterForm(forms.ModelForm):
    confirm_password = forms.CharField(widget=forms.PasswordInput(), label="确认密码")

    class Meta:
        model = Student
        fields = ('grade',
                  'name',
                  'password',
                  'confirm_password',
                  'gender',
                  'birthday',
                  'email',
                  'info')

    def clean(self):
        cleaned_data = super(StuRegisterForm, self).clean()
        password = cleaned_data.get('password')
        confirm_password = cleaned_data.get('confirm_password')
        if confirm_password != password:
            self.add_error('confirm_password', 'Password does not match.')

        return cleaned_data
```

重写后的clean方法：添加一个验证的规则

- 方法第一行先调用父类的`clean`方法进行原有的基础验证
- 从`cleaned_data`中分别取`password`和`confirm_password`这两个字段的值，进行比较。如果比较发现这两个值不一致，此时需要报错。这里又两种报错方式（推荐第二种，使用该方式可以展示多个报错，而不是只能展示第一个报错）：

- 抛出`ValidationError`异常：`raise forms.ValidationError("Password does not match.")`
- 使用`add_error`方法添加异常：`self.add_error('confirm_password', 'Password does not match.')`

9.学生信息管理系统——主页面部分

添加对应的视图(view)方法

在`./user/urls.py`中，先在开头导入视图方法，即

```
from user import views
```

设置主页的url

在`./user/urls.py`中，先在开头导入视图方法，即

```
from user import views
```

然后给`urlpatterns`列表添加元素，并引入path方法

```
from django.urls import path

urlpatterns = [
    path('login/', views.home, name="login"),
]
```

将user的url导入到项目中：

在`urls.py`中，加入urloatterns

```
from django.urls import path, include

urlpatterns = [
    path('user/', include('user.urls')),
]
```

Django项目默认设置只会读取app文件夹下的模板文件夹，在项目文件夹下添加模板文件夹后，需要在setting.py中进行对TEMPLATES的设置，添加`'DIRS': [os.path.join(BASE_DIR, 'templates')],`

10学生信息管理系统——登录界面

设计表单

登录需要用到表单来提交登录信息（账号+密码）

在`user`文件夹中新建`forms.py`， 添加以下代码来实现老师和学生的登录信息表单

```
from django import forms
from user.models import Student, Teacher


class StuLoginForm(forms.Form):
    uid = forms.CharField(label='学号', max_length=10)
    password = forms.CharField(label='密码', max_length=30,
widget=forms.PasswordInput)


class TeaLoginForm(forms.Form):
    uid = forms.CharField(label='教职工号', max_length=10)
    password = forms.CharField(label='密码', max_length=30,
widget=forms.PasswordInput)
```

添加模板

在项目的`template/user`文件夹下，添加`login_detail.html`：

```
{% extends "user/background.html" %}
{% block welcome_message %}
    <div class="welcome-message">欢迎</div>
{% endblock %}
{% block login_container %}
    {% if kind == "student" %}
        <div class="login-kind-title">我是学生</div>
    {% else %}
        <div class="login-kind-title">我是老师</div>
    {% endif %}
    <div class = "form">
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <div class="submit-button">
                <input type="submit" value="登录"/>
                <a href="">注册</a>
            </div>
        </form>
        <div class="return-button"><a href="{% url 'login' %}">返回上一页</a>
</div>
    </div>
{% endblock %}
```

添加对应的视图方法

为了区分学生和老师，这里给该视图添加了`kind`参数

`kind`必须为`"teacher"`或`"student"`，如果不是的话，就会返回一个文本标明这不是一个合适的kind。

- 不是支持的kind的情况
    - 这里建议把文本放在一个专门的py文件中，不仅方便修改和检查，在以后如果要支持多语言也方便
    - 这里在项目下建立一个`constants.py`文件，存放相关的文本

```
INVALID_KIND = "Invalid kind.kind should be student or teacher."
```

在`./user/views.py`中，添加方法

```
from django.http.response import HttpResponse

from constants import INVALID_KIND
from user.forms import StuLoginForm, TeaLoginForm

def login(request, *args, **kwargs):
    if not kwargs or kwargs.get("kind", "") not in ["student", "teacher"]:
        return HttpResponse(INVALID_KIND)

    kind = kwargs["kind"]
    context = {'kind': kind}

    if request.method == 'POST':
        if kind == "teacher":
            form = TeaLoginForm(data=request.POST)
        else:
            form = StuLoginForm(data=request.POST)

        if form.is_valid():
            uid = form.cleaned_data["uid"]

            temp_res = "hello, %s" % uid
            return HttpResponse(temp_res)
        else:
            context['form'] = form
    else:
        if kind == "teacher":
            form = TeaLoginForm()
        else:
            form = StuLoginForm()

        context['form'] = form

    return render(request, 'user/login_detail.html', context)
```

设置主页的url

在`./user/urls.py`中，给`urlpatterns`列表添加元素

```
path('login/<slug:kind>', views.login, name="login")
```

然后这个时候我们需要去更新下前面主页的href，指向对应的学生老师登录页
更新后的`login_home.html`文件如下

```
{% extends "user/background.html" %}
{% block login_container %}
<div class="login-kind student-login-button">
    <a href="{% url 'login' 'student' %}">学生登录</a>
</div>
<div class="login-kind teacher-login-button">
    <a href="{% url 'login' 'teacher' %}">教师登录</a>
</div>
{% endblock %}
```

四、实现注册功能

1.Django框架中基于类的视图（Class-based views）介绍

class-based views：基于模型自动生成的视图views

- CBVs的实质是一个视图类
- 目标：简化为了模型models制作的各种各样的视图，快速完成对model的增删改查等操作
- 实现：使用视图类的方法生成视图

他们都使用了类方法as_view来获得视图函数，但是参数不同。

常见的编辑视图

首先常用到的是一些通用的编辑视图`Generic editing views`，这些视图都在`django.views.generic.edit`中，包括：

- `FormView`
- `CreateView`
- `UpdateView`
- `DeleteView`

编辑视图的属性

- `model`： 该视图所对应的模型
- `fields`：哪些字段可展示编辑
- `form_class`：使用哪种表单来展示这个模型
- `template_name`：渲染该视图使用的模板
- `success_url`：指定编辑成功后跳转的页面

注意：

1. `fields`和`form_class`这两个参数互斥，必须且只能设置其中一个。
2. `DeleteView`无`fields`、`form_class`属性

视图使用的过程

1. 在引入了CreateView类后，就可以直接使用CBVs

```
from django.views.generic import CreateView
```

2. view.py中编写自己需要的视图类

更加推荐的写法是在view.py中继承CBVs，从而设置编写自己需要的视图类，再在urls.py中不传参调用CreateViews库中的as_view方法。在views.py中编写一个新的类CreateStudentView

```
class CreateStudentView(CreateView):
    model = Student
    fields="__all__"
    template_name = "user/register.html"
    success_url = "login"
```

3. 将生成的视图添加到url中

然后`urls.py`中的`register`的`path`就可以写为：

```
path('register', CreateStudentView.as_view(), name="register"),
```

2.学生信息管理系统——用CBVs实现注册功能

添加注册页面模板(template)

在`templates/user`下新建`register.html`如下

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Register</title>
</head>
<body>
    <div class="register-container">
        <div class="register-title">注册</div>
        <!-- form 表单中的method属性中的post方法说明浏览器向服务器发送数据
             建立连接之后，浏览器就会按照分段传输的方法将数据发送给服务器-->
        <form method="post" class="form">
            {% csrf_token %}
            <!--django 会自动创建显示表单所需要的全部字段
                修饰符as_p让Django以字段格式渲染所有的表单元素-->
            {{form.as_p}}
            <p><input type="submit" value="注册" class="submit-button"></p>
        </form>
        <div class="return-button">
            <a href="{% url 'login' kind%}">返回上一页</a>
        </div>
    </div>
</body>
</html>
```

添加对应的视图方法

注册应该实现的功能：

- 产生一个新的学生账号→在student数据库表中添加一个新的记录→新建一个学生model的实例→为model类进行新添实例的CreateView
    - 这里使用CBVs的优点在于：方便我们直接根据指定的字段生成前端表单，该生成的表单**自带检查字段格式的功能**+在后端接受表单请求后**按照表单数据**生成对应的**实例**

- 新增功能：在填写密码之后还需要确认密码，提交的时候要检查这两个密码是否一致→实现定制化的表单

（1）实现定制化表单

在`user/forms.py`文件中，添加代码

```
class StuRegisterForm(forms.ModelForm):
    confirm_password = forms.CharField(widget=forms.PasswordInput(), label="确认密码")

    class Meta:
        model = Student
        fields = ('grade',
                  'name',
                  'password',
                  'confirm_password',
                  'gender',
                  'birthday',
                  'email',
                  'info')

    def clean(self):
        cleaned_data = super(StuRegisterForm, self).clean()
        password = cleaned_data.get('password')
        confirm_password = cleaned_data.get('confirm_password')
        if confirm_password != password:
            self.add_error('confirm_password', 'Password does not match.')
        return cleaned_data


class TeaRegisterForm(forms.ModelForm):
    confirm_password = forms.CharField(widget=forms.PasswordInput(), label="确认密码")

    class Meta:
        model = Teacher
        fields = ('name',
                  'password',
                  'confirm_password',
                  'gender',
                  'birthday',
                  'email',
                  'info')

    def clean(self):
        cleaned_data = super(TeaRegisterForm, self).clean()
        password = cleaned_data.get('password')
        confirm_password = cleaned_data.get('confirm_password')
        if confirm_password != password:
            self.add_error('confirm_password', 'Password does not match.')
        return cleaned_data
```

（2）实现CBV

关键点：

- 学生学号=4位年级号+6位子学号

    - 年级号：注册的时候自己选择
    - 子学号：按照年级内的注册的先后顺序生成
- 老师编号=3位院系号+7位院内编号
    - 院系号：随机生成
    - 院内编号：按照院内的注册顺序先后生成
- 注册后的学生学号和老师编号是后台生成的，注册者并不知道，因此需要重定向后将账号展示给注册者看
    - 使用url技术来传参，传到注册详情页展示给注册者看

新建`user/cbvs.py`如下

```
from django.shortcuts import reverse,redirect
# 引入原有的CreateView方法
from django.views.generic import  CreateView
# 引入已经创建的注册表单项
from user.forms import StuRegisterForm,TeaRegisterForm
# 引入已经创建的model模型
from user.models import Student,Teacher

import random

#创建继承自CreateView的视图类
class CreateStudentView(CreateView):
    model = Student
    form_class = StuRegisterForm
    template_name = "user/register.html"
    #成功后跳转的页面
    success_url = "login"

    # form_valid的功能：生成学号=年级号+子学号
    def form_valid(self, form):
        #学生注册时选定年级自动生成学号
        #通过表单的cleaned_data属性获取学号
        grade = form.cleaned_data["grade"]
        #order_by默认升序排列，number前的负号表示降序排列
        #对现有的学生数据进行排序
        student_set = Student.objects.filter(grade=grade).order_by("-number")
        # 生成学生的编号
        if student_set.count()>0:
            #如果不是第一个学生，则先排序
            #最后一个学生位于第一个位置
            last_student = student_set[0]
            new_number = str(int(last_student.number)+1)

            for i in range(6 - len(new_number)):
                new_number = "0" + new_number
        else:
            #如果是第一个学生
            new_number = "000001"
        #新建学生实例(不保存)
        new_student = form.save(commit=False)
        #添加学生信息（将新生成的序号加到学生数据的对应位置上
        new_student.number = new_number
        #保存学生实例
        new_student.save()
        #保存多项学生信息
```

```
        #many-to-many data
        form.save_m2m()

        self.object = new_student
        #生成学生的学号
        uid = grade + new_number
        from_url = "register"
        base_url = reverse(self.get_success_url(),kwargs={'kind':'studet'})
        #完成后会返回一个HttpResponseRedirect对象
        #注册成功后，会返回到注册详情页
        return redirect(base_url + '?uid=%s&from_url=%s' % (uid, from_url))

#继承CreateView创建教师的view类
class CreateTeacherView(CreateView):
    model = Teacher
    form_class = TeaRegisterForm
    template_name = "user/register.html"
    success_url = "login"

    def post(self, request, *args,**kwargs):
        form = self.get_form()

        #判断表单的内容是否合法
        if form.is_valid():
            return self.form_valid(form)
        else:
            return self.form_invalid(form)

    def form_valid(self, form):
        #老师注册时随机生成院系号，院系号的范围为[0,300)
        department_no = random.randint(0,300)
        #把非三位数的院系号转为以0填充的字符串(比如1转为001)
        department_no = "{:0>3}".format(department_no)
        #对现有的老师进行排序
        teacher_set = Teacher.objects.filter(department_no =
department_no).order_by("-number")
        if teacher_set.count()>0:
            last_teacher = teacher_set[0]
            new_number = int(last_teacher.number) + 1
            new_number = '{:0>7}'.format(new_number)
        else:
            new_number = "0000001"

        #实例化老师对象，但是不保存
        new_teacher = form.save(commit=False)
        #添加老师的信息
        new_teacher.department_no = department_no
        new_teacher.number = new_number
        #保存老师实例
        new_teacher.save()
        # 保存多项学生信息
        # many-to-many data
        form.save_m2m()

        self.object = new_teacher

        uid = department_no + new_number
        from_url = "register"
```

```
        base_url = reverse(self.get_success_url(),kwargs={'kind':'teacher'})
        return redirect(base_url + '?uid=%s&from_url=%s' % (uid,from_url))
```

(3)实现注册视图的方法

一般来说，实现CBV后，使用CBV自带的as_view()就可以生成需要的view方法了。但是我们这里有些不同，由于有老师和学生两种注册，需要用同一个视图方法来处理这两种请求。

视图方法：

- 接收一个参数，该参数需要指明是老师注册和学生注册中的哪一种
- 视图方法内部用条件判断语句，针对不同的种类，返回不同的视图构造结果

在`user/views.py`中，继续添加代码如下

```
# 在开头导入视图类
from user.cbvs import CreateStudentView, CreateTeacherView


def register(request, kind):
    func = None
    if kind == "student":
        func = CreateStudentView.as_view()
    elif kind == "teacher":
        func = CreateTeacherView.as_view()

    if func:
        return func(request)
    else:
        return HttpResponse(INVALID_KIND)
```

更新url

在`user/urls.py`文件中，
给urlpatterns列表添加一行元素：

```
path('register/<slug:kind>', views.register, name="register")
```

再修改下`templates/user/login_detail.html`，为注册功能添加对应的进入链接

将

```
                <a href="">注册</a>
```

修改为

```
                <a href="{% url 'register' kind%}">注册</a>
```

展示注册后的账号信息

最后，修改下登录详情页部分代码，使其能够展示注册得到的账号信息，该信息是通过url来进行传参的。

更新`user/views.py`中的`login`方法(增加GET的方法)如下

```
def login(request, *args, **kwargs):
```

```
    if not kwargs or kwargs.get("kind", "") not in ["student", "teacher"]:
        return HttpResponse(INVALID_KIND)

    kind = kwargs["kind"]
    context = {'kind': kind}

    if request.method == 'POST':
        if kind == "teacher":
            form = TeaLoginForm(data=request.POST)
        else:
            form = StuLoginForm(data=request.POST)

        if form.is_valid():
            uid = form.cleaned_data["uid"]

            temp_res = "hello, %s" % uid
            return HttpResponse(temp_res)
        else:
            context['form'] = form
    elif request.method == 'GET':
        if request.GET.get('uid'):
            uid = request.GET.get('uid')
            context['uid'] = uid
            data = {"uid":uid, 'password': '12345678'}
            if kind == "teacher":
                form = TeaLoginForm(data)
            else:
                form = StuLoginForm(data)
        else:
            if kind == "teacher":
                form = TeaLoginForm()
            else:
                form = StuLoginForm()

        context['form'] = form
        if request.GET.get('from_url'):
            context['from_url'] = request.GET.get('from_url')

    return render(request, 'user/login_detail.html', context)
```

再更新`templates/user/login_detail.html`如下

```
{% extends "user/background.html" %}
{% block welcome_message %}
    {% if from_url == "register" %}
        <div class="welcome-message">注册成功，你的{% if kind == "student" %}学号{%
else %}账号{% endif %}是 {{ uid }}</div>
    {% else %}
        <div class="welcome-message">欢迎</div>
    {% endif %}
{% endblock %}
{% block login_container %}
    {% if kind == "student" %}
        <div class="login-kind-title">我是学生</div>
    {% else %}
        <div class="login-kind-title">我是老师</div>
    {% endif %}
```

```
    <div class = "form">
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <div class="submit-button">
                <input type="submit" value="登录"/>
                <a href="{% url 'register' kind%}">注册</a>
            </div>
        </form>
        <div class="return-button"><a href="{% url 'login' %}">返回上一页</a>
</div>
    </div>
{% endblock %}
```

添加返回按钮

在`templates/user/register.html`的表单下，即`</form>`之后添加

```
        <div class="return-button">
            <a href="{% url 'login' kind %}">返回上一页</a>
        </div>
```

在模板中加入kind这个变量：

在`user/cbvs.py`中，分别给`CreateStudentView`类、`CreateTeacherView`类重写一下`get_context_data`方法，如下

五、实现登录逻辑

登录的时候需要实现的功能有：

- 检查是否已经注册
  - 如果没有注册
    - 提示账号不存在
  - 如果已经注册
    - 密码不匹配---提示密码不正确
  - 密码匹配--成功登录，跳转到个人主页，通过cookie保存登录信息

1.修改登录视图(views)

修改`user/views.py`中的`login`方法如下：

```
def login(request, kind):
    if kind not in ["teacher", "student"]:
        return HttpResponse(INVALID_KIND)

    if request.method == 'POST':
        if kind == "teacher":
            form = TeaLoginForm(data=request.POST)
        else:
            form = StuLoginForm(data=request.POST)

        if form.is_valid():
            uid = form.cleaned_data["uid"]
```

```
            if len(uid) != 10:
                form.add_error("uid", "账号长度必须为10")
            else:
                if kind == "teacher":
                    department_no = uid[:3]
                    number = uid[3:]
                    object_set =
Teacher.objects.filter(department_no=department_no, number=number)
                else:
                    grade = uid[:4]
                    number = uid[4:]
                    object_set = Student.objects.filter(grade=grade,
number=number)
                if object_set.count() == 0:
                    form.add_error("uid", "该账号不存在.")
                else:
                    user = object_set[0]
                    if form.cleaned_data["password"] != user.password:
                        form.add_error("password", "密码不正确.")
                    else:
                        request.session['kind'] = kind
                        request.session['user'] = uid
                        request.session['id'] = user.id

                        return redirect("course", kind=kind)

            return render(request, 'user/login_detail.html', {'form': form,
'kind': kind})
    else:
        context = {'kind': kind}
        if request.GET.get('uid'):
            uid = request.GET.get('uid')
            context['uid'] = uid
            if kind == "teacher":
                form = TeaLoginForm({"uid": uid, 'password': '12345678'})
            else:
                form = StuLoginForm({"uid": uid, 'password': '12345678'})
        else:
            if kind == "teacher":
                form = TeaLoginForm()
            else:
                form = StuLoginForm()
        context['form'] = form
        if request.GET.get('from_url'):
            context['from_url'] = request.GET.get('from_url')

        return render(request, 'user/login_detail.html', context)
```

登录后会在cookie中存储以下信息：

- `kind`：用户类型，学生或老师
- `user`：用户学号或教师编号

2.添加个人主页(学生)和课程主页(教师)的简单形式

成功登录后，学生将跳转到个人主页；教师将跳转到课程主页。这里首先编写一个简单的主页，为后续的跳转做准备。

对于主页的处理思路是：首先创建home方法（这里可以看做是一个选择器），然后根据kind的种类跳转到对应的不同的主页进行处理。

注意，跳转到一个带参数的url，有两种写法

- `return redirect(reverse("login", kwargs={"kind": kind}))`
- `return redirect('login', kind = kind)`

`course/views.py`代码如下：

```
from django.http.response import HttpResponse
from django.shortcuts import render, reverse, redirect

from user.models import Student, Teacher
from constants import INVALID_KIND


def get_user(request, kind):
    """

    :param request:
    :param kind: teacher or student
    :return: return Teacher instance or Student instance
    """
    if request.session.get('kind', '') != kind or kind not in ["student",
"teacher"]:
        return None

    if len(request.session.get('user', '')) != 10:
        return None

    uid = request.session.get('user')
    if kind == "student":
        # 找到对应学生
        grade = uid[:4]
        number = uid[4:]
        student_set = Student.objects.filter(grade=grade, number=number)
        if student_set.count() == 0:
            return None
        return student_set[0]
    else:
        # 找到对应老师
        department_no = uid[:3]
        number = uid[3:]
        teacher_set = Teacher.objects.filter(department_no=department_no,
number=number)
        if teacher_set.count() == 0:
            return None
        return teacher_set[0]


# Create your views here.
def home(request, kind):
```

```
    if kind == "teacher":
        return teacher_home(request)
    elif kind == "student":
        return student_home(request)
    return HttpResponse(INVALID_KIND)


def teacher_home(request):
    kind = "teacher"
    user = get_user(request, kind)

    if not user:
        return redirect('login', kind=kind)

    info = {
        "name": user.name,
        "kind": kind
    }

    context = {
        "info": info
    }

    return render(request, 'course/nav.html', context)

def student_home(request):
    kind = "student"
    user = get_user(request, kind)

    if not user:
        return redirect('login', kind = kind)

    info = {
        "name": user.name,
        "kind": kind
    }

    context = {
        "info": info
    }

    return render(request, 'course/nav.html', context)
```

添加模板文件`templates/course/nav.html`

```
<!DOCTYPE html>
<html lang="en">
{% load static %}
<head>
    <meta charset="UTF-8">
    <title>
        {% block title %}{% endblock %}
    </title>
</head>
<body>
<div class="nav">
    <div class="nav-title">
```

```
        <a href="{% url 'course' kind=info.kind %}">
        <p class="main-title">学生选课管理系统</p>
        <p class="sub-title">
            {% if info.kind == "teacher" %}
                教师端
            {% elif info.kind == "student" %}
                学生端
            {% endif %}</p>
        </a>
    </div>

    <div class="name-logo">
        <div class="user-name">
            {{ info.name }}
        </div>
    </div>
</div>

<div class="main-content">
    {% block content %}{% endblock %}
</div>
</body>
</html>
```

然后添加`course/urls.py`如下

```
from django.urls import path
from course.views import *

urlpatterns = [
    path('<slug:kind>/', home, name="course"),
]
```

同时还要去改下主url，即在`SSCMS/urls.py`的`urlpatterns`中添加

```
path('course/', include("course.urls")),
```

3.实现退出登录

在`user/views.py`中添加退出登录的视图(views)

```
def logout(request):
    if request.session.get("kind", ""):
        del request.session["kind"]
    if request.session.get("user", ""):
        del request.session["user"]
    if request.session.get("id", ""):
        del request.session["id"]
    return redirect(reverse("login"))
```

在`user/urls.py`中的`urlpatterns`的添加对应的路由

```
path('logout/', views.logout, name="logout")
```

在修改下登录后的主页视图（即`templates/course/nav.html`）

即`<div class="name-logo">`的之前添加

```
    <div class="log-out">
        <a href="{% url "logout" %}">退出</a>
    </div>
```

4.简化登录主页url

目前，如果要打开登录主页，需要的地址为：`http://127.0.0.1:8000/user/login`

希望简化登录主页的地址，实现通过`http://127.0.0.1:8000`访问登录主页。可以通过修改urlpattern实现

修改后的`SSCMS/urls.py`如下

```
from django.urls import path, include
from user.views import home

urlpatterns = [
    path('', home, name="login"),
    path('user/', include("user.urls")),
    path('course/', include("course.urls")),
]
```

六、修改个人信息

目标：在用户模块添加修改个人信息的功能。

业务逻辑：学生注册信息可以填写年级，但是在修改个人信息的时候不能对其进行修改。

1.表单选择

修改学生信息的表单：在`user/forms.py`中添加如下代码

```
class StuUpdateForm(StuRegisterForm):
    class Meta:
        model = Student
        fields = ('name',
                  'password',
                  'confirm_password',
                  'gender',
                  'birthday',
                  'email',
                  'info')
```

老师信息修改的表单可以使用原来注册表单，即`TeaRegisterForm`

2.模板文件

明确了表单后，则可以添加对应模板文件`templates/user/update.html`如下

```
<!DOCTYPE html>
<html lang="en">
{% load static %}
<head>
```

```
    <meta charset="UTF-8">
    <title>
        Register
    </title>
    <link href="{% static 'css/register.css' %}" rel="stylesheet">
</head>
<body>

    <div class="register-container">
        <div class="register-title">修改个人信息</div>
        <form method="post" class="form">
            {% csrf_token %}
            {{form.as_p}}
            <p><input type="submit" value="修改" class="submit-button"/></p>
        </form>
        <div class="return-button"><a href="{% url 'course' kind %}">返回上一页
</a></div>

    </div>
</body>
```

3.添加视图(views)

首先，在视图`user/cbvs.py`开头，添加导入需要的库和类

```
from django.views.generic import UpdateView
from user.forms import StuUpdateForm
```

再最后添加如下代码

```
class UpdateStudentView(UpdateView):
    model = Student
    form_class = StuUpdateForm
    template_name = "user/update.html"

    def get_context_data(self, **kwargs):
        context = super(UpdateStudentView, self).get_context_data(**kwargs)
        context.update(kwargs)
        context["kind"] = "student"
        return context

    def get_success_url(self):
        return reverse("course", kwargs={"kind": "student"})


class UpdateTeacherView(UpdateView):
    model = Teacher
    form_class = TeaRegisterForm
    template_name = "user/update.html"

    def get_context_data(self, **kwargs):
        context = super(UpdateTeacherView, self).get_context_data(**kwargs)
        context.update(kwargs)
        context["kind"] = "teacher"
        return context

    def get_success_url(self):
```

```
        return reverse("course", kwargs={"kind": "teacher"})
```

在视图`user/views.py`开头，添加导入上面两个视图类`UpdateStudentView, UpdateTeacherView`

并添加如下代码

```
def update(request, kind):
    func = None
    if kind == "student":
        func = UpdateStudentView.as_view()
    elif kind == "teacher":
        func = UpdateTeacherView.as_view()
    else:
        return HttpResponse(INVALID_KIND)

    pk = request.session.get("id")
    if pk:
        context = {
            "name": request.session.get("name", ""),
            "kind": request.session.get("kind", "")
        }
        return func(request, pk=pk, context=context)

    return redirect("login")
```

4.添加路由

在`user/urls.py`中的`urlpatterns`添加对应的路由

```
path('update/<slug:kind>', views.update, name="update"),
```

同时再去个人主页中去添加对应的链接，使得用户可以在个人主页点击它进入信息修改页面。

这里把这个链接添加在个人主页的用户名这里， 同时出于简介美观的目的，用户名只展示一个姓（后面会给这个姓添加一个圆背景）。

在`templates/course/nav.html`中

将

```
            {{ info.name }}
```

修改为

```
            <a href="{% url 'update' info.kind %}">
                {{ info.name.0 }}
            </a>
```

七、CSS样式优化

首先，需要在项目的`static`文件夹下，新建文件夹`css`用于存放css文件。

同时需要修改下设置，把这个css文件夹放到`STATICFILES_DIRS`中，使得 Django也会在那里查找静态文件。

即在`SSCMS/settings.py`末尾添加如下代码

```
STATICFILES_DIRS = [
   os.path.join(BASE_DIR, "static"),
]
```

1.优化登录页样式

先为登录页面添加样式，在`css`文件夹下新建`login.css`如下

```
body {
    margin: 0;
}

.main-container {
    position: absolute;
    width: 100%;
    height:100%;
    background: #4a2c964d;
    background: linear-gradient(rgba(230, 100, 101, 0.2), rgba(145, 152, 229,
0.3)),
        linear-gradient(#9198e560, #4a2c9880);
}

.main-header {
    height: 45%;
    text-align: center;
    font-size: 40px;
    color: #4a2c98;
}

.main-header .main-title {
    font-size: 50px;
    margin-top: 5%;
}

.main-header .welcome-message {
    font-size: 26px;
    margin-top: 60px;
    color: #ff5722;
}

.login-container {
    height: 40%;
    width: 400px;
    margin: 0 auto;
    background: #eee;
    border-radius: 10px;
    box-shadow: 0 0 15px 2px rgba(0, 0, 0, .33);
    overflow: hidden;
}

.login-container .login-kind {
    padding-top: 10%;
    font-size: 30px
}

.login-container .login-kind a {
    text-decoration: none;
```

```
    background: #4a2c98;
    color: #eeeeee;
    padding: 10px;
    text-align: center;
    display: block;
    width:50%;
    margin: 0 auto;
    border-radius: 10px;
}


/* for login detail page  */
.login-kind-title {
    height: auto;
    padding: 2%;
    text-align: center;
    color: #4d2f99;
    width: 96%;
    font-size: 22px;
    display: block;
    background: #ccc;
    overflow: hidden
}


.login-container .form p,
.login-container .form .submit-button {
    widht: 90%;
    padding-top: 4%;
    margin: 0 auto;
    display: flex;
    align-items: center;
    justify-content: center;
    font-family: "Roboto", "Lucida Grande", "DejaVu Sans", "Bitstream Vera
Sans",
        Verdana, Arial, sans-serif;

}


.login-container .form p label {
    padding-right: 10px;
    width: 80px;
}

.login-container .form p input {
    clear: both;
    padding: 8px;
    width: 60%;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.login-container .form .submit-button,
.login-container .return-button {
```

```
    margin: 5px auto 0;
    padding-top: 20px;
}

.submit-button input,
.submit-button a {
    border: none;
    text-decoration: none;
    font-size: 18px;
    background: #4a2c98;
    color: #eeeeee;
    padding: 5px 0;
    text-align: center;
    display: block;
    width: 30%;
    margin: 5px 10px;
    border-radius: 10px
}

.return-button a{
    border: none;
    text-decoration: none;
    font-size: 18px;
    background: #cccccc;
    color: #4a2c98;
    padding: 5px 0;
    text-align: center;
    display: block;
    width: 30%;
    margin: 0 auto;
    border-radius: 10px;
}
```

并在`templates/user/background.html`引入该样式

```
    <link href="{% static 'css/login.css' %}" rel="stylesheet">
```

注意：要使用`{% static 'css/login.css' %}`语法，必须在模板文件中先使用语句`{% load static %}`

2.注册页面的样式

新建`static/css/register.css`如下

```
.register-container {
    height: 40%;
    width: 500px;
    margin: 100px auto;
    background: #eee;
    border-radius: 10px;
    box-shadow: 0 0 15px 2px rgba(0, 0, 0, 0.33);
    overflow: hidden;
}


.register-container .register-title {
```

```
    height: auto;
    padding: 2%;
    justify-content: center;
    text-align: center;
    color: #ccc;
    width: 96%;
    font-size: 22px;
    display: block;
    background: #4d2f99;
    overflow: hidden;
}

.register-container .form p {
    width: 90%;
    padding-top: 15px;
    margin: 0 auto;
    display: flex;
    align-items: center;
    justify-content: center;
    font-family: "Roboto", "Lucida Grande", "DejaVu Sans", "Bitstream Vera
Sans",
        Verdana, Arial, sans-serif;
    word-break: break-all;
    flex-flow: wrap;
}

.register-container .form p label {
    padding-right: 10px;
    width: 80px;
}


.register-container .form p input,
.register-container .form p select {
    clear: both;
    padding: 8px;
    width: 60%;
    -webkit-box-sizing: border-box;
    -moz-box-sizing: border-box;
    box-sizing: border-box;
    border: 1px solid #ccc;
    border-radius: 4px;
}

.register-container .form p span.helptext {
    color: slategrey;
}

.register-container .form p .submit-button {
    border: none;
    text-decoration: none;
    font-size: 18px;
    background: #4a2c98;
    color: #eeeeee;
    padding: 5px 0;
    text-align: center;
    display: block;
    width: 30%;
```

```
    margin: 0 10px 30px;
    border-radius: 10px;
}

.register-container .return-button {
    padding-left: 20px;
    padding-bottom: 10px;
}
```

在`templates/user/register.html`和`templates/user/update.html`中，`head`标签之前（即`<head>`之前），添加下面一行代码导入`static`。

```
{% load static %}
```

引入css样式：

```
    <link href="{% static 'css/register.css' %}" rel="stylesheet">
```

3.设置主页样式

添加`static/css/nav.css`如下

```
body,p {
    margin: 0;
    padding: 0;
}

html, body {
    height: 100%;
    width: 100%;
}

body {
    background: #ccc;
}

.nav {
    background: #4a2c98;
    width: 100%;
}

.nav a {
    color: #ccc;
    text-decoration: unset;
}

.nav .nav-title,
.nav .name-logo,
.nav .log-out  {
    display: inline-block;
    margin: 5px;
}

.nav p {
    display: inline-block;
    float: left;
```

```
    padding-left: 10px;
}

.nav .nav-title {
    font-size: 24px;
    line-height: 26px;
    height: 26px;
    vertical-align: top;
}


.nav p.main-title {
    margin-right: 10px;
}

.nav p.sub-title {
    border-left: 3px solid #cccccc;
}

.nav .name-logo,
.nav .log-out  {
    float: right;
    margin: 8px 5px 0;
    vertical-align: top;
}

.nav .name-logo .user-name {
    background: #ccc;
    border-radius: 50%;
    width: 24px;
    height: 24px;

    text-align: center;
    line-height: 24px;
    font-size: 16px;
    font-weight: bold;
}

.nav .name-logo .user-name a {
    color: #4a2c98;
}

.nav .log-out a {
    margin: 5px;
    background: #ccc;
    color: #4a2c98;
    border-radius: 5px;
    text-decoration: none;
    padding: 0 5px;
}
```

在`templates/course/nav.html`中，引入设置的样式

```
<link href="{% static 'css/nav.css' %}" rel="stylesheet">
```

八、创建课程模型

明确功能需求，对于课程模块，需要实现：

- 课程表(表明哪些课程是可选课程)
- 学生课程表
    - 学生和课程的关系表
    - 在学生选课结束，系统设置结束选课后生成
- 时刻表
    - 对课程上课的时间进行设置

1.设置课程状态

这一步是后面所有步骤的基础，由于状态的切换会导致表的录入的时间有所不同

在`constants.py`中去记录好（即添加代码如下）

```
COURSE_STATUS = {
    1: "未开始选课",
    2: "开始选课",
    3: "结束选课",
    4: "结课",
    5: "打分完成",
}

COURSE_OPERATION = {
    1: "开始选课",
    2: "结束选课",
    3: "结课",
    4: "给分",
    5: "查看详情"
}
```

然后在`course/models.py`中导入下面会需要的所有库

```
from django.db import models
import datetime
from user.models import Student, Teacher
from constants import COURSE_STATUS, COURSE_OPERATION
```

2.添加课程模型

在`course/models.py`中添加代码如下

```
def current_year():
    # refer: https://stackoverflow.com/questions/49051017/year-field-in-
django/49051348
    return datetime.date.today().year

class Course(models.Model):
    credits = [
        (1, 1),
        (2, 2),
        (3, 3),
        (4, 4),
```

```
        (5, 5),
    ]
    semesters = [
        ("Autumn", "上"),
        ("Spring", "下")
    ]
    name = models.CharField(max_length=50, verbose_name="课程名")
    introduction = models.CharField(max_length=250, verbose_name="介绍")
    credit = models.IntegerField(verbose_name="学分")
    max_number = models.IntegerField(verbose_name="课程最大人数")

    year = models.IntegerField(verbose_name="年份", default=current_year)
    semester = models.CharField(max_length=20, verbose_name="学期",
choices=semesters)

    # 未开始选课， 1
    # 开始选课，未结束选课 2
    # 结束选课， 3
    # 结课 4
    # 已打完分 5
    status = models.IntegerField(verbose_name="课程状态", default=1)

    teacher = models.ForeignKey(Teacher, verbose_name="课程教师",
on_delete=models.CASCADE)

    def get_status_text(self):
        return COURSE_STATUS[self.status]

    def get_op_text(self):
        return COURSE_OPERATION[self.status]

    def get_current_count(self):
        courses = StudentCourse.objects.filter(course=self, with_draw=False)
        return len(courses)

    def get_schedules(self):
        schedules = Schedule.objects.filter(course=self)
        return schedules

    def __str__(self):
        return "%s (%s)" % (self.name, self.teacher.name)
```

3.添加课程时刻表模型

在`course/models.py`中添加代码如下

```
def weekday_choices():
    weekday_str = ['周一', '周二', '周三', '周四', '周五', '周六', '周日']
    return [(i+1, weekday_str[i]) for i in range(7)]


class Schedule(models.Model):
    weekday = models.IntegerField(choices=weekday_choices(), verbose_name="日期")
    start_time = models.TimeField(verbose_name="上课时间")
    end_time = models.TimeField(verbose_name="下课时间")
```

```
    location = models.CharField(max_length=100, verbose_name="上课地点")
    remarks = models.CharField(max_length=100, verbose_name="备注", null=True,
blank = True)

    start_week = models.IntegerField(verbose_name="第几周开始")
    end_week = models.IntegerField(verbose_name="第几周结束")

    intervals = [
        (1, "无间隔"),
        (2, "每隔一周上一次")
    ]
    week_interval = models.IntegerField(verbose_name="周间隔", choices=intervals,
default=1)

    course = models.ForeignKey(Course, verbose_name="课程名",
on_delete=models.CASCADE)

    def __str__(self):
        s = "第%s周-第%s周 " % (self.start_week, self.end_week)
        if self.week_interval == 2:
            s += "隔一周 "
        s += "%s %s-%s " % (self.get_weekday_display(),
self.start_time.strftime("%H:%M"),
                            self.end_time.strftime("%H:%M"))
        s += "在%s" % self.location
        if self.remarks:
            s += " %s" % self.remarks
        return s
```

4.添加课程关系表模型

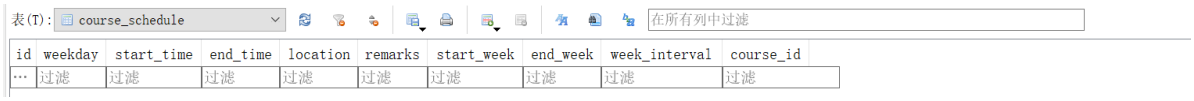
在`course/models.py`中添加代码如下

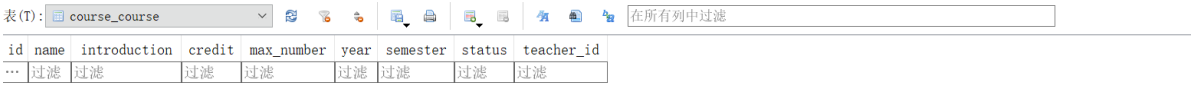
5.建立(更新)数据库

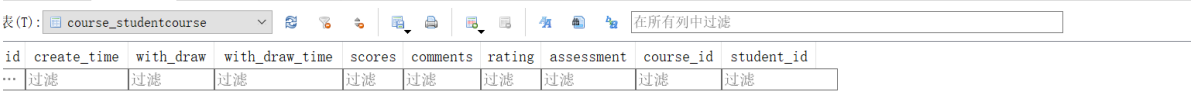
即在项目文件夹下打开命令行窗口，执行:

```
python manage.py makemigrations
python manage.py migrate
```

更新数据库后，在SQLite中查看结果







九、老师课程业务实现

对于教师端而言，需要实现的目标功能有：

- 创建课程
- 添加、删除课程时刻表
- 查看课程列表
- 操作课程：修改状态、给学生打分

在实现上述功能之前，首先，在`course/views.py`中将课程的模型类全部导入，以便后面使用

```
from .models import Course, Schedule, StudentCourse
```

1.创建课程

创建课程的表单，新建`course/forms.py`如下

```
from django import forms
from .models import Course, Schedule, StudentCourse


class CourseForm(forms.ModelForm):

    class Meta:
        model = Course
        exclude = ['status', 'teacher']
```

同时对于新建课程的请求，在`constants.py`中添加一个非法请求的响应如下

```
INVALID_REQUEST_METHOD = "Invalid request method."
```

新建对应模板`templates/course/teacher/create_course.html`如下

```
{% extends "course/nav.html" %}
{% block title %}创建课程{% endblock %}
{% block content %}
    <h3>创建课程</h3>
    <div class="form create-update-from">
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <div class="submit-button">
                <input type="submit" value="创建"/>
                <input type="button" value="返回"  onclick='window.open("{% url
'course' "teacher"%}")' />
            </div>
        </form>
    </div>
{% endblock %}
```

再在`course/views.py`中导入`CourseForm`类和`INVALID_REQUEST_METHOD`常量，然后添加代码如下

```
def create_course(request):
    user = get_user(request, "teacher")
    if not user:
```

```
        return redirect(reverse("login", kwargs={"kind": "teacher"}))

    info = {
        "name": user.name,
        "kind": "teacher",
    }

    if request.method == 'POST':
        form = CourseForm(request.POST)
        if form.is_valid():
            obj = form.save(commit=False)
            obj.status = 1
            obj.teacher = user

            obj.save()
            return redirect(reverse("course", kwargs={"kind": "teacher"}))
    elif request.method == 'GET':
        form = CourseForm()
    else:
        return HttpResponse(INVALID_REQUEST_METHOD)

    return render(request, 'course/teacher/create_course.html', {'info': info,
'form': form})
```

2.添加、删除课程时刻表

创建课程时刻表的表单，在`course/forms.py`中添加代码如下

```
class ScheduleForm(forms.ModelForm):
    class Meta:
        model = Schedule
        exclude = ["course"]
```

新建对应模板`templates/course/teacher/create_schedule.html`如下

```
{% extends "course/nav.html" %}
{% block title %}创建时刻表{% endblock %}
{% block content %}
    <h3>创建时刻表:    [{{ course.id }}] {{ course.name }}
</h3>
    <div class="form create-update-from">
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <div class="submit-button">
                <input type="submit" value="创建"/>
                <input type="button" value="返回"  onclick='window.open("{% url
'view_detail' course.id%}")' />
            </div>
        </form>
    </div>
{% endblock %}
```

在`course/views.py`中导入这个表单，然后添加代码如下

```
def create_schedule(request, course_id):
```

```
    user = get_user(request, "teacher")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "teacher"}))

    info = {
        "name": user.name,
        "kind": "teacher",
    }

    course = Course.objects.get(pk=course_id)

    if request.method == 'POST':
        form = ScheduleForm(request.POST)
        if form.is_valid():
            obj = form.save(commit=False)
            obj.course = course
            obj.save()

            return redirect(reverse("view_detail", kwargs={"course_id":
course_id}))
    elif request.method == 'GET':
        form = ScheduleForm()
    else:
        return HttpResponse(INVALID_REQUEST_METHOD)

    return render(request, 'course/teacher/create_schedule.html', {'info': info,
'form': form, "course": course})


def delete_schedule(request, schedule_id):
    user = get_user(request, "teacher")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "teacher"}))

    schedule = Schedule.objects.get(pk=schedule_id)

    course_id = request.GET.get("course_id") or schedule.course.id

    schedule.delete()

    return redirect(reverse("view_detail", kwargs={"course_id": course_id}))
```

3.查看课程列表

在本项目中，老师和学生的个人主页就是其课程主页，将展示其所有课程列表。

为老师的课程主页添加模板`templates/course/teacher/home.html`如下

```
{% extends "course/nav.html" %}
{% block title %}HomePage{% endblock %}
{% block content %}
    <div class="main-container">
        <div class="main-bar">
            <form class="search-form" method="post">
                {% csrf_token %}
                <input class="input" id="search-key" type="text" name="search"
{% if search_key != None %}value="{{ search_key }}" {% endif %}/>
```

```
                <input class="button" type="submit" value="搜索课程" />
            </form>

            <input class="button right-button" type="button" value="创建课程"
 onclick='window.open("{% url 'create_course' %}")' />
        </div>
        <table class="item-list course-list">
            <thead>
                <tr>
                    <th class="course-no">课程编号</th>
                    <th class="course-name">名称</th>
                    <th class="course-credit">学分</th>
                    <th class="course-number">当前人数<br>/总人数</th>
                    <th class="course-year">年份</th>
                    <th class="course-semester">学期</th>
                    <th class="course-status">状态</th>
                    <th class="course-operation">操作</th>
                </tr>
            </thead>
            <tbody>
                {% for course in course_list %}
                    <tr id="course-id-{{ course.id }}">
                    <td class="course-no">{{ course.id }}</td>
                    <td class="course-name">{{ course.name }}</td>
                    <td class="course-credit">{{ course.credit }}</td>
                    <td class="course-number">{{ course.get_current_count }}/{{
course.max_number }}</td>
                    <td class="course-year">{{ course.year }}</td>
                    <td class="course-semester">{{ course.get_semester_display
}}</td>
                    <td class="course-status">{{ course.get_status_text }}</td>
                    <td class="course-operation">
                        {% if course.status < 4 %}
                            <input class="button right-button" type="button"
value="{{ course.get_op_text }}"
                                onclick='location.href="{% url 'handle_course'
course.id course.status %}"' />
                        {% endif %}
                        {% if course.status == 4 %}
                        {# 结课后给分 #}
                            <input class="button right-button" type="button"
value="{{ course.get_op_text }}"
                                    onclick='location.href="{% url 'view_detail'
course.id %}"' />
                        {% else %}
                            <input class="button right-button" type="button"
value="查看详情"
                                    onclick='location.href="{% url 'view_detail'
course.id %}"' />
                        {% endif %}
                    </td>
                    </tr>
                {% endfor %}
            </tbody>

        </table>
    </div>
{% endblock %}
```

查看课程列表的功能要写在老师的主页视图中，即修改`course/views.py`中的`teacher_home`如下

里面使用了`django.db.models.Q`类，所以要在开头添加代码`from django.db.models import Q`导入这个类

```
def teacher_home(request):
    user = get_user(request, "teacher")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "teacher"}))

    info = {
        "name": user.name,
        "kind": "teacher",
    }

    is_search = False
    search_key = ""
    if request.method == "POST":
        search_key = request.POST.get("search")
        if search_key:
            is_search = True

    context = {"info": info}
    q = Q(teacher=user)
    if is_search:
        q = q & Q(name__icontains=search_key)
        context["search_key"] = search_key

    context["course_list"] = Course.objects.filter(q).order_by('status')

    return render(request, 'course/teacher/home.html', context)
```

补充说明：这里面还实现了一个搜索框，能够根据关键词去搜索课程。为了不使用js，搜索框的信息是通过post表单信息来提交的。

老师在课程主页，可以进行常规的课程状态修改：

- 开始选课
- 结束选课
- 结课

对于打分功能则需要在课程详情页实现。

所以这里一方面要实现一个课程主页的操作视图（用于实现常规课程状态修改，第4小节实现），也要实现一个课程详情页视图（用于实现打分，第5小节实现）。

4.常规课程状态修改（课程主页）

先添加一个课程详情页的模板文件`templates/course/teacher/course.html`如下：

```
{% extends "course/nav.html" %}
{% block title %}课程详情{% endblock %}
{% block content %}
```

```
    <h3>课程详情<input class="button right-button" type="button" value="返回主页"
                    onclick='window.open("{% url 'course' 'teacher'%}")'/></h3>
    <table class="item-list detail-list">
        <thead>
            <tr>
                <th>课程编号</th>
                <th>名称</th>
                <th>学分</th>
                <th>当前人数/总人数</th>
                <th>年份</th>
                <th>学期</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>{{ course.id }}</td>
                <td>{{ course.name }}</td>
                <td>{{ course.credit }}</td>
                <td>{{ course.get_current_count }}/{{ course.max_number }}</td>
                <td>{{ course.year }}</td>
                <td>{{ course.get_semester_display }}</td>
            </tr>
        </tbody>
    </table>

    <h3>上课时间<input class="button right-button" type="button" value="添加时间表"
onclick='window.open("{% url 'create_schedule' course.id%}")'/></h3>
    <table class="item-list schedule-list">
        <thead>
            <tr>
                <th class="schedule-no">编号</th>
                <th class="schedule-no">详情</th>
                <th class="schedule-no">操作</th>
            </tr>
        </thead>
        <tbody>
            {% for schedule in schedules %}
            <tr>
                <td>{{ schedule.id }}</td>
                <td>{{ schedule }}</td>
                <td>
                    <input class="button" type="button" value="删除"
                           onclick='window.open("{% url 'delete_schedule'
schedule.id%}?course_id={{ course.id }}")'/>
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

    <h3>学生列表
        {% if course.status == 4 %}
            <input class="button right-button" type="button" value="给分完成"
onclick='location.href="{% url 'handle_course' course.id 4%}"' />
        {% endif %}
    </h3>
    <table class="item-list student-list">
        <thead>
```

```
            <tr>
                <th class="student-no">学生学号</th>
                <th class="student-name">学生姓名</th>
                <th class="student-email">学生邮箱</th>
                <th class="student-score">得分</th>
                <th class="student-comments">评价</th>
                <th class="operation">操作</th>
            </tr>
        </thead>
        <tbody>
            {% for cs in course_students %}
            <tr>
                <td>{{ cs.student.get_id}}</td>
                <td>{{ cs.student.name }}</td>
                <td>{{ cs.student.email }}</td>
                <td>
                    {% if cs.scores == None %}-{% endif %}
                    {% if cs.scores != None %}{{ cs.scores }}{% endif %}
                </td>
                <td>
                    {% if cs.scores == None %}-{% endif %}
                    {% if cs.scores != None %}{{ cs.comments }}{% endif %}
                </td>
                <td class="operation">
                    {% if course.status == 4 %}
                        {% if cs.scores == None %}
                            <input class="button right-button" type="button"
value="给分"
                                onclick='location.href="{% url 'score' cs.id%}"'
/>
                        {% else %}
                            <input class="button right-button" type="button"
value="修改成绩"
                                onclick='location.href="{% url 'score' cs.id%}?
update=1"' />
                        {% endif %}
                    {% else %}
                        -
                    {% endif %}
                </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>

    {% if course.status == 5 %}
    <h3>学生评价</h3>
    <table class="item-list student-list">
        <thead>
            <tr>
                <th class="student-score">学生评分</th>
                <th class="student-comments">学生评价</th>
            </tr>
        </thead>
        <tbody>
            {% for cs in sorted_course_students %}
            {% if cs.rating != None %}
            <tr>
```

```
                <td>{{ cs.rating }}</td>
                <td>{{ cs.assessment }}</td>
            </tr>
            {% endif %}
            {% endfor %}
        </tbody>
    </table>
    {% endif %}
{% endblock %}
```

在`course/views.py`中添加代码如下

```
def handle_course(request, course_id, handle_kind):
    """
    :param request:
    :param course_id:
    :param handle_kind:
            1: "开始选课",
            2: "结束选课",
            3: "结课",
            4: "给分完成"
    :return:
    """
    user = get_user(request, "teacher")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "teacher"}))

    info = {
        "name": user.name,
        "kind": "teacher",
    }

    course = Course.objects.get(pk=course_id)
    if course.status == handle_kind and course.status < 5:
        if course.status == 4:
            scs = StudentCourse.objects.filter(course=course)
            all_given = True
            res = ""
            for sc in scs:
                if sc.scores is None:
                    all_given = False
                    res += "<div>%s 未打分</div>" % sc.student

            if all_given:
                course.status += 1
                course.save()
                return redirect(reverse("view_detail", kwargs={"course_id":
course.id}))
            else:
                return HttpResponse(res)
        else:
            course.status += 1
            course.save()

    course_list = Course.objects.filter(teacher=user)
    return render(request, 'course/teacher/home.html', {'info': info,
'course_list': course_list})
```

```python
def view_detail(request, course_id):
    user = get_user(request, "teacher")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "teacher"}))

    info = {
        "name": user.name,
        "kind": "teacher",
    }

    course = Course.objects.get(pk=course_id)
    c_stu_list = StudentCourse.objects.filter(course=course)
    sche_list = Schedule.objects.filter(course=course)

    context = {
        "info": info,
        "course": course,
        "course_students": c_stu_list,
        "schedules": sche_list
    }

    if course.status == 5:
        sorted_cs_list = sorted(c_stu_list, key=lambda cs: cs.scores)
        context["sorted_course_students"] = sorted_cs_list

    return render(request, "course/teacher/course.html", context)
```

5.打分操作（课程详情页）

学生的分数是记录在学生课程关系表中的，在学生选课成功后会新建一条对应的数据。教师打分，则是修改其中的分数字段，即对学生课程表模型进行更新。

这里我们首选CBVs中的`UpdateView`,不过要先给这个视图建立一个表单，在`course/forms.py`中添加代码如下

```python
class ScoreForm(forms.ModelForm):
    class Meta:
        model = StudentCourse
        fields = ["student", "course", "scores", "comments"]

    student = forms.CharField(label="学生", disabled=True)
    # course = forms.CharField(widget=forms.TextInput(attrs={'readonly':
'readonly'}))
    course = forms.CharField(label="课程", disabled=True)

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.initial['student'] = self.instance.student
        self.initial['course'] = self.instance.course

    def clean_student(self):
        return self.initial['student']
```

```
    def clean_course(self):
        return self.initial['course']
```

新建`course/cbvs.py`如下：

```
from django.views.generic.edit import DeleteView, CreateView, UpdateView
from django.views.generic.detail import DetailView
from django.shortcuts import render, reverse, redirect

# Relative import of GeeksModel
from .models import Schedule, StudentCourse
from .forms import ScoreForm


class ScoreUpdateView(UpdateView):
    model = StudentCourse
    form_class = ScoreForm
    template_name = 'course/teacher/score.html'

    def get(self, request, *args, **kwargs):
        self.object = self.get_object()

        title = "给分"
        if request.GET.get("update"):
            title = "修改成绩"

        info = {}
        return_url = reverse("course", kwargs={"kind": "teacher"})
        if self.object:
            teacher = self.object.course.teacher
            info = {
                "name": teacher.name,
                "kind": "teacher",
            }
            return_url = reverse("view_detail", kwargs={"course_id":
self.object.course.id})

        return self.render_to_response(self.get_context_data(info=info,
title=title, return_url=return_url))

    def get_success_url(self):
        if self.object:
            return reverse("view_detail", kwargs={"course_id":
self.object.course.id})
        else:
            return reverse("course", kwargs={"kind": "teacher"})
```

同时补上其对应的模板文件`templates/course/teacher/score.html`如下

```
{% extends "course/nav.html" %}
{% block title %}{{ title }}{% endblock %}
{% block content %}
    <h3>{{ title }}</h3>
    <div class="form create-update-from">
        <form method="post">
            {% csrf_token %}
```

```
            {{form.as_p}}
            <div class="submit-button">
                <input type="submit" value="确定"/>
                <input type="button" value="返回"  onclick='location.href="{{
return_url }}"' />
            </div>
        </form>
    </div>
{% endblock %}
```

5.添加url

上面已经把老师需要的视图方法全部实现完毕了，接下来就是添加到路由里面。

修改后的`course/urls.py`如下

```
from django.urls import path
from course.views import *
from course.cbvs import ScoreUpdateView


urlpatterns = [
    path('<slug:kind>/', home, name="course"),
    path('<slug:kind>/', home, name="course"),
    path('teacher/create_course', create_course, name="create_course"),
    path('teacher/view_detail/<int:course_id>', view_detail,
name="view_detail"),
    path('teacher/create_schedule/<int:course_id>', create_schedule,
name="create_schedule"),
    path('teacher/delete_schedule/<int:schedule_id>', delete_schedule,
name="delete_schedule"),
    path('teacher/score/<int:pk>', ScoreUpdateView.as_view(), name="score"),
    path('teacher/handle_course/<int:course_id>/<int:handle_kind>',
handle_course, name="handle_course"),
]
```

十、学生课程业务实现

学生对于课程的操作应该包括：

- 查看课程列表
- 选课撤课
- 结课后对教师进行评价

1.查看课程列表

`view_kind`分为

- `current`：查看当前课程
- `is_end`：查看结课课程
- `select`：可选课的
- `withdraw`：可撤课的

学生可以根据类别查看不同的课程

新建学生查看课程的模板`templates/course/student/home.html`如下

然后在`course/views.py`中添加代码如下

```
def view_course(request, view_kind):
    """
    :param view_kind:
        current: 查看当前课程
        is_end: 查看结课课程
        select: 选课
        withdraw: 撤课
    """
    user = get_user(request, "student")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "student"}))

    is_search = False
    search_key = ""
    if request.method == "POST":
        search_key = request.POST.get("search")
        if search_key:
            is_search = True

    info = {
        "name": user.name,
        "kind": "student",
    }

    course_list = []

    if view_kind in ["select", "current", "withdraw", "is_end"]:
        if view_kind == "select":
            q = Q(status=2)
            if is_search:
                q = q & (Q(name__icontains=search_key) |
Q(teacher__name__icontains=search_key))

            course_list = Course.objects.filter(q)

            my_course = StudentCourse.objects.filter(Q(student=user) &
Q(with_draw=False))
            my_cids = [c.course.id for c in my_course]
            course_list = [c for c in course_list if c.id not in my_cids]
        else:
            q = Q(student=user) & Q(with_draw=False)
            if is_search:
                q = q & (Q(name__icontains=search_key) |
Q(teacher__name__icontains=search_key))
            my_course = StudentCourse.objects.filter(q)
            if view_kind == "current":
                course_list = [c.course for c in my_course if c.course.status <
4]
            elif view_kind == "withdraw":
                course_list = [c.course for c in my_course if c.course.status ==
2]
            elif view_kind == "is_end":
                course_list = [c for c in my_course if c.course.status >= 4]

    else:
        return HttpResponse(INVALID_REQUEST_METHOD)
```

```
    context = {
        'info': info,
        'view_kind': view_kind,
        'course_list': course_list
    }
    if is_search:
        context["search_key"] = search_key

    return render(request, 'course/student/home.html', context)
```

课程主页即学生的个人主页（这一点与教师进行课程操作的部分有所不同），故修改`course/views.py`中的原视图方法`student_home`为

```
def student_home(request):
    return redirect(reverse("view_course", kwargs={"view_kind": "current"}))
```

2.选课撤课

选课是新建一个学生课程关系记录，撤课则是修改对应的学生课程关系记录。即学生有两种操作课程方法，`operate_kind`如下：

- `select`：选课
- `withdraw`：撤课

如果网页请求发送的方法不在这两种里面，则不符合规范，同时需要将这一信息通过响应返回告知浏览器。

因此在`constants.py`中添加`ILLEGAL_KIND = "Illegal kind for you."`

在`course/views.py`中，导入`ILLEGAL_KIND`，然后添加代码如下

```
# 在开头导入timezone
from django.utils import timezone

def operate_course(request, operate_kind, course_id):
    """
    :param operate_kind:
        select: 选课
        withdraw: 撤课
    """
    user = get_user(request, "student")
    if not user:
        return redirect(reverse("login", kwargs={"kind": "student"}))

    if operate_kind not in ["select", "withdraw"]:
        return HttpResponse(ILLEGAL_KIND)
    elif operate_kind == "select":
        course = Course.objects.filter(pk=course_id).get()
        new_course = StudentCourse(student=user, course=course)
        new_course.save()
    elif operate_kind == "withdraw":
        q = Q(course__id=course_id) & Q(student=user) & Q(with_draw=False)
        course = StudentCourse.objects.filter(q).get()
        course.with_draw = True
        course.with_draw_time = timezone.now()
```

```
        course.save()

    return redirect(reverse("view_course", kwargs={"view_kind": operate_kind}))
```

3.结课后对老师给出评价

学生给老师评教和老师给学生评分的后端逻辑是一样的，都是修改学生课程关系表内的数据。

先在`course/forms.py`中添加

```
class RateForm(forms.ModelForm):
    class Meta:
        model = StudentCourse
        fields = ["course", "scores", "comments", "rating", "assessment"]

    course = forms.CharField(label="课程", disabled=True)
    scores = forms.IntegerField(label="成绩", disabled=True)
    comments = forms.CharField(label="老师评价", disabled=True)

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.initial['course'] = self.instance.course
        self.initial['scores'] = self.instance.scores
        self.initial['comments'] = self.instance.comments

    def clean_course(self):
        return self.initial['course']

    def clean_scores(self):
        return self.initial['scores']

    def clean_comments(self):
        return self.initial['comments']
```

然后添加模板文件`templates/course/student/rating.html`：

```
{% extends "course/nav.html" %}
{% block title %}评教{% endblock %}
{% block content %}
    <h3>评教</h3>
    <div class="form create-update-from">
        <form method="post">
            {% csrf_token %}
            {{form.as_p}}
            <div class="submit-button">
                <input type="submit" value="确定"/>
                <input type="button" value="返回" onclick='location.href="{{
return_url }}"' />
            </div>
        </form>
    </div>
{% endblock %}
```

再在`course/cbvs.py`中导入`RateForm`类，然后添加代码如下

```
class RateUpdateView(UpdateView):
```

```
    model = StudentCourse
    form_class = RateForm
    template_name = 'course/student/rating.html'

    def get(self, request, *args, **kwargs):
        self.object = self.get_object()

        info = {}
        return_url = reverse("view_course", kwargs={"view_kind": "is_end"})
        if self.object:
            student = self.object.student
            info = {
                "name": student.name,
                "kind": "student",
            }

        return self.render_to_response(self.get_context_data(info=info,
return_url=return_url))

    def get_success_url(self):
        return reverse("view_course", kwargs={"view_kind": "is_end"})
```

4.学生课程详情页

采用CBVs的思路来实现

先添加模板`templates/course/student/course.html`如下

```
{% extends "course/nav.html" %}
{% block title %}课程详情{% endblock %}
{% block content %}
    <h3>课程详情</h3>
    <ul class="course-details">
        <li class="course-detail"><span class="detail-name">课程编号</span> {{
object.course.id }}</li>
        <li class="course-detail"><span class="detail-name">课程名</span> {{
object.course.name }}</li>
        <li class="course-detail"><span class="detail-name">学分</span> {{
object.course.credit }}</li>
        <li class="course-detail"><span class="detail-name">课程人数/最大人数
</span> {{ object.course.get_current_count }}/{{ object.course.max_number }}
</li>
        <li class="course-detail"><span class="detail-name">年份</span> {{
object.course.year }}</li>
        <li class="course-detail"><span class="detail-name">学期</span> {{
object.course.get_semester_display }}</li>

        <li class="course-detail"><span class="detail-name">教师</span> {{
object.course.teacher.name }}</li>
        <li class="course-detail"><span class="detail-name">上课时间</span>
            <span class="course-schedules">
            {% for schedule in object.course.get_schedules %}
                <div class="course-schedule">{{ schedule }}</div>
                <div class="course-schedule">{{ schedule }}</div>
            {% endfor %}
            </span>
        </li>
```

```
        <li class="course-detail"><span class="detail-name">得分</span>
            {% if object.scores != None %}{{ object.scores }}{% else %} - {%
endif %}
        </li>
        <li class="course-detail"><span class="detail-name">评语</span>
            {% if object.comments != None %}{{ object.comments }}{% else %} - {%
endif %}
        </li>
        <li class="course-detail"><span class="detail-name">学生评分</span>
            {% if object.rating != None %}{{ object.rating }}{% else %} - {%
endif %}
        </li>
        <li class="course-detail"><span class="detail-name">学生评价</span>
            {% if object.assessment != None %}{{ object.assessment }}{% else %} -
{% endif %}
        </li>

    </ul>
    <input type="button" value="返回" onclick='location.href="{% url
'view_course' 'is_end'%}"' />
{% endblock %}
```

再在`course/cbvs.py`中添加代码如下

5.添加url

上面已经把学生需要的视图方法全部实现完毕了，接下来就是添加到路由里面。
修改后的`course/urls.py`如下

```
from django.urls import path
from course.views import *
from course.cbvs import ScoreUpdateView, RateUpdateView, StudentCourseDetailView


urlpatterns = [
    path('<slug:kind>/', home, name="course"),
    path('teacher/create_course', create_course, name="create_course"),
    path('teacher/view_detail/<int:course_id>', view_detail,
name="view_detail"),
    path('teacher/create_schedule/<int:course_id>', create_schedule,
name="create_schedule"),
    path('teacher/delete_schedule/<int:schedule_id>', delete_schedule,
name="delete_schedule"),
    path('teacher/score/<int:pk>', ScoreUpdateView.as_view(), name="score"),
    path('teacher/handle_course/<int:course_id>/<int:handle_kind>',
handle_course, name="handle_course"),

    path('student/view/<slug:view_kind>', view_course, name="view_course"),
    path('student/operate/<int:course_id>/<slug:operate_kind>', operate_course,
name="operate_course"),

    path('student/evaluate/<int:pk>', RateUpdateView.as_view(),
name="evaluate"),
    path('student/view_detail/<int:pk>', StudentCourseDetailView.as_view(),
name="sview_detail"),
]
```

十一、完善CSS页面样式

1.优化课程主页的布局

新建`static/css/main.css`如下

```
.main-content {
    width: 900px;
    margin: 0 auto;
    background: #e6e6f0;
}

.main-container {
    width: 850px;
    margin: 0 auto;
}

.main-content h3{
    width: 850px;
}

.main-content .right-button{
    float: right;
    margin: 0 5px;
}

.main-bar {
    width: 850px;
    height: 30px;
}

.main-bar .search-form {
    display: inline-block;
}

.main-bar .button {
    height: 30px;
    vertical-align:top;
    border:none;
    color:#eee;
    background:#4a2c98;
    font-size: 16px;
    border-radius: 5px;
}

.main-bar .input{
    width: 150px;
    height: 24px;
    margin: auto 10px;
    vertical-align:top
}

.main-bar .right-button {
    float: right;
    margin: 0 5px;
}
```

然后再在`templates/course/nav.html`中导入这个css文件， 即在`</head>`之前，添加如下一行代码：

```
<link href="{% static 'css/main.css' %}" rel="stylesheet">
```

由于课程模块所有模板都是继承的`templates/course/nav.html`，所以这个样式是对所有模板生效的。

考虑到样式呈现的问题，这里对于前面已经写好的`.nav`属性和`.main-content`属性进行修改

```
.nav {
    background: #4a2c98;
    position: fixed;
    width: 100%;
    color: #ccc;
    z-index: 1;
}
```

```
.main-content {
    width: 900px;
    margin: 0 auto;
    background: #e6e6f0;
    min-height: 100%;
    position: fixed;
    left: 0;
    right: 0;
    padding: 60px 20px;
    top: 0;
}
```

2.优化课程列表样式

课程模块中，有一些页面有表格(table)样式的列表，这里优化下列表样式。

新建`static/css/list.css`如下

```
table.item-list {
    margin: 10px 0;
    width: 850px;
}

.item-list th,
.item-list td {
    box-sizing: content-box;
    width: fit-content;
    padding: 3px;
    text-align: left;
    border-bottom: 1px solid #C0C0C0;
}


.item-list tr:nth-child(even) {
    background-color: #dfdfdf;
}
```

```css
.item-list th {
    background-color: #9481c5;
}


/* for course table col width*/
.item-list th.course-no,
.item-list td.course-no {
    width: 70px;
}

.item-list th.course-name,
.item-list td.course-name {
    width: 150px;
}

.item-list th.course-credit,
.item-list td.course-credit {
    width: 40px;
}

.item-list th.course-number,
.item-list td.course-number {
    width: 70px;
}

.item-list th.course-year,
.item-list td.course-year {
    width:  50px;
}

.item-list th.course-semester,
.item-list td.course-semester {
    width:  30px;
}

.item-list th.course-status,
.item-list td.course-status {
    width:  100px;
}

.item-list th.course-teacher,
.item-list td.course-teacher {
    width: 70px;
}

.item-list th.course-operation,
.item-list td.course-operation {
    width:  150px;
}

.item-list th.course-schedule,
.item-list td.course-schedule {
    width: 200px
}

.item-list td.course-schedule {
```

```
    font-size: 10px;
}

.item-list th.course-operation.student-course,
.item-list td.course-operation.student-course {
    width:  80px;
}

.item-list th.course-year-semester,
.item-list td.course-year-semester {
    width:  70px;
}
```

老师和学生的主页有课程列表，所以需要导入这个css文件。而老师的课程详情页里有选课的学生列表，所以也需要导入这个css文件。

因此，需要导入这个css文件的模板有：

- `templates/course/student/home.html`
- `templates/course/teacher/home.html`
- `templates/course/teacher/course.html`

导入方法为，在`block`块中（比如`{% block content %}`这行后面），添加下面一行代码：

```
<link href="{% static 'css/list.css' %}" rel="stylesheet">
```

3.优化表单样式

新建`static/css/form.css`如下

```
.create-update-from {
    margin: 10px;
}

.create-update-from p{
    padding: 5px;
}


.create-update-from p:nth-child(even) {
    background-color: #dfdfdf;
}

.create-update-from p:nth-child(odd) {
    background-color: #c8c8d2;
}

.create-update-from p label{
    display:inline-block;
    width: 200px;
}

.create-update-from .submit-button {
    margin-top: 20px;
}
```

```
.create-update-from .submit-button input {
    width: 80px;
    margin-right: 20px;
}
```

课程模块有这几个使用了form表单的页面需要优化：

- `templates/course/teacher/create_course.html`
- `templates/course/teacher/create_schedule.html`
- `templates/course/teacher/score.html`
- `templates/course/student/rating.html`

将该css文件导入上面说的需要的四个模板中，导入方法
即在`block`块中（比如`{% block content %}`这行后面），添加下面一行代码：

```
<link href="{% static 'css/form.css' %}" rel="stylesheet">
```

4.特殊处理学生课程详情页

学生课程详情页这里打算不像上面那样简单的展示，所以做了一个特殊的样式来展示学生课程详情信息。
`static/css/details.css`如下

```
ul.course-details {
    margin: 20px;
    list-style: none;
    padding: 0 20px;
}

li.course-detail {
    min-height: 24px;
    padding: 2px;
}

li.course-detail .detail-name {
    display: inline-block;
    vertical-align: top;
    width: 150px;
    font-weight: bolder;
}

li.course-detail span.course-schedules {
    display: inline-block;
}

ul.course-details li:nth-child(odd) {
    background-color: #ccc;
}

ul.course-details li:nth-child(even) {
    background-color: #dfdfdf;
}
```

该css文件导入`templates/course/student/course.html`模板中，导入方法同上，即在`block`块中（比如`{% block content %}`这行后面），添加下面一行代码：

```
<link href="{% static 'css/details.css' %}" rel="stylesheet">
```

5.课程详情

模板中导入css的link标签里，使用了模板语法中的`static`标签(tag)，所以所有使用了`static`标签的模板都要在开头导入这个标签。
特别的，对于存在继承关系的模板之间。虽然`templates/course/nav.html`开头有`{% load static %}`，但是继承它的子模板中如果用到了`static`标签，仍然需要再导入一遍。

导入方法为在模板文件的`{% extends "course/nav.html" %}`这一句后面，添加这样一行代码

```
{% load static %}
```

补充说明：模板文件中如果出现了继承语句`{% extends "..." %}`，则该继承语句必须在模板的第一行。所以新增只能在这后面增添。

不过这样子一个一个增添`{% load static %}`，还是太过麻烦，尤其是需要改动多个模板文件时。
除了一个一个模板里面添加这个，Django还给我们实现了一种方便快捷的手段——在设置文件中修改。在`SSCMS/settings.py`的`TEMPLATES`中，给其Django模板（一般是第一个）配置字典中的`OPTIONS`属性，添加这样一个配置关系：

```
'builtins': ['django.templatetags.static']
```

添加这个后，模板开头有没有`{% load static %}`都可以用`static`标签了。（不过最好去除掉无用代码，删掉所有之前模板中添加的`{% load static %}`）