**WEB框架开发**

# 一 Django框架开发

## 1.WEB 应用

Web应用程序是一种可以通过种可以通过Web访问的应用程序，程序的最大好处是用户很容易访问应用程序，用户只需要有浏览器即可，不需要再安装其他软件。应用程序有两种模式C/S、B/S。C/S是客户端/服务器端程序，也就是说这类程序一般独立运行。而B/S就是浏览器端/服务器端应用程序，这类应用程序一般借助谷歌，火狐等浏览器来运行。WEB应用程序一般是B/S模式。Web应用程序首先是“应用程序”，和用标准的程序语言，如java，python等编写出来的程序没有什么本质上的不同。在网络编程的意义下，浏览器是一个socket客户端，服务器是一个socket服务端。

## 2.http协议

### 1.协议简介

HTTP协议是Hyper Text Transfer Protocol（超文本传输协议）的缩写,是用于万维网服务器与本地浏览器之间传输超文本的传送协议。

HTTP是一个属于应用层的面向对象协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。HTTP协议工作于客户端-服务端架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。

### 2.http协议特性

1）基于TCP/IP ：HTTP协议是基于TCP/IP协议上的应用层协议。

2）基于请求-响应模式 ：HTTP协议规定，请求从客户端发出，最后服务器端响应该请求并返回。换句话说，肯定是先从客户端开始建立通信的，服务端在没有接收到请求之前不会响应。

3）无状态保存 ： HTTP是一种不保存状态,即无状态(stateless)协议。HTTP协议 自身不对请求和响应之间的通信状态进行保存。也就是说在HTTP这个 级别,协议对于发送过的请求或响应都不做持久化处理。

使用HTTP协议,每当有新的请求发送时,就会有对应的新响应产生。协议本身并不保留之前一切的请求或响应报文的信息。这是为了更快地处理大量事务,确保协议的可伸缩性,而特意把HTTP协议设计成 如此简单的。可是,随着Web的不断发展,因无状态而导致业务处理变得棘手 的情况增多了。比如,用户登录到一家购物网站,即使他跳转到该站的 其他页面后,也需要能继续保持登录状态。针对这个实例,网站为了能 够掌握是谁送出的请求,需要保存用户的状态。HTTP/1.1虽然是无状态协议,但为了实现期望的保持状态功能, 于是引入了Cookie技术。有了Cookie再用HTTP协议通信,就可以管 理状态了。有关Cookie的详细内容稍后讲解。

4）无连接 ：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

### 3.http请求协议与响应协议

http协议包含由浏览器发送数据到服务器需要遵循的请求协议与服务器发送数据到浏览器需要遵循的请求协议。用于HTTP协议交互的信被为HTTP报文。请求端(客户端)的HTTP报文做请求报文,响应端(服务器端)的做响应报文。HTTP报文本身是由多行数据构成的字文本。



1）请求协议

请求格式：



请求方式：get与post请求

* GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditBook?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的请求体中.
* GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制.
* GET与POST请求在服务端获取请求数据方式不同。

2）响应协议

响应格式：



响应状态码：状态码的值是当客户端向服务端发送请求时，返回的请求结果。借助状态码，用户可以知道服务器端是否正常处理了请求。



实例：

# -\*- coding: utf-8 -\*-  
*import* socket  
sock = socket.socket()  
sock.bind(("127.0.0.1",8801))  
sock.listen(5)  
*while True*:  
 # conn 客户端套接字  
 conn,addr = sock.accept()  
 data = conn.recv(1024)  
 # HTTP/1.1 200 OK 相应首行  
 # 读取html文件  
 # POST 请求才有请求头数据  
 # 按照http请求协议解析数据  
 # 专注于web业务开发  
 # 按照http响应协议封装数据  
 *with* open('login.html','r') *as* f:  
 data = f.read()  
 conn.send(("HTTP/1.1 200 OK\r\n\r\n%s"%data).encode('utf8'))  
 conn.close()

login.html

<!DOCTYPE html>  
<**html** lang=**"en"**>  
<**head**>  
 <**meta** charset=**"UTF-8"**>  
 <**title**>**login**</**title**>  
</**head**>  
<**body**>  
<**form** action=**"http://127.0.0.1:8801"** method=**"post"**>  
 **用户名：**<**input** type=**"text"** name=**"user"**>  
 **密码：**<**input** type=**"password"** name=**"pwd"**>  
 <**input** type=**"submit"** value=**"提交"**>  
</**form**>  
</**body**>  
</**html**>

## 3.wsgiref模块

WSGI（Web Server Common Interface）是专门为Python语言制定的web服务器与应用程序之间的网关接口规范，通俗的来说，只要一个服务器拥有一个实现了WSGI标准规范的模块（例如apache的mod\_wsgi模块），那么任意的实现了WSGI规范的应用程序都能与它进行交互。因此，WSGI也主要分为两个程序部分：服务器部分和应用程序部分。

其实 wsgiref 最基本的功能就是封装了socket，不需要自己再去写socket。因为我们不希望接触到TCP连接、HTTP原始请求和响应格式，所以，需要一个统一的接口协议来实现这样的服务器软件，让我们专心用Python编写Web业务。这个接口就是WSGI：Web Server Gateway Interface。而wsgiref模块就是python基于wsgi协议开发的服务模块。

*from* wsgiref.simple\_server *import* make\_server  
  
*def* application(*environ*, *start\_response*):  
 *start\_response*('200 OK', [('Content-Type', 'text/html')])  
 *return* [b'<h1>Hello, web!</h1>']

httpd = make\_server('', 8080, application)  
print('Serving HTTP on port 8000...')  
# 开始监听HTTP请求:  
httpd.serve\_forever()

## 4.DIY一个WEB框架

利用wsgiref模块实现一个简单的WEB框架

目录结构如下：



DIY 的web框架 功能简介：  
main.py 启动文件 ，封装socket   
1. urls.py ：路径与视图函数映射关系 ---- url 控制器  
2. views.py ： 视图函数，固定有一个形参：environ ---- 视图函数  
3. templates文件夹：存放html文件 ---- 模板  
4.modules : 在项目启动前，在数据库中创建数据库表结构的 ----与数据相关

main.py：

*from* wsgiref.simple\_server *import* make\_server  
*from* urls *import* url\_patterns  
*def* application(*environ*, *start\_response*):  
 path = *environ*.get("PATH\_INFO")  
 start\_response('200 OK', [('Content-Type', 'text/html')])  
 *print*(path)  
 func = None  
 *for* item *in* url\_patterns:  
 *if* path == item[0]:  
 func = item[1]  
 *break  
 if* func:  
 *return* [func(*environ*)]  
 *else*:  
 *return* [b'404']  
httped=make\_server("",8066,application) # 封装了socket ,application 是回调函数  
# 等待用户连接：conn,addr = sock.accept()  
httped.serve\_forever()

urls.py:

# -\*- coding: utf-8 -\*-  
# \_\_title\_\_ = 'urls.py'  
# \_\_author\_\_ = 'YangYang'  
# \_\_mtime\_\_ = '2018.06.19'  
*from* views *import* \*  
url\_patterns = [  
 ("/login", login),  
 ("/index", index),  
 ("/favicon.ico",fav),  
 ("/timer",timer),  
 ("/auth",auth),  
]

views.py

# -\*- coding: utf-8 -\*-  
# \_\_title\_\_ = 'views.py'  
# \_\_author\_\_ = 'YangYang'  
# \_\_mtime\_\_ = '2018.06.19'  
*import* pymysql  
*from* urllib.parse *import* parse\_qs  
  
*def* login(environ):  
 *with* open('templates/login.html','rb') *as* f:  
 data = f.read()  
 *return* data  
  
*def* index(environ):  
 *with* open('templates/index.html','rb') *as* f:  
 data = f.read()  
 *return* data  
  
*def* fav(environ):  
 *with* open('templates/favicon.ico','rb') *as* f:  
 data = f.read()  
 *return* data  
  
*def* timer(environ):  
 *import* datetime  
 now = datetime.datetime.now().strftime("%y-%m-%d %X")  
 *return* now.encode('utf8')  
  
*def* auth(*request*):  
  
 *try*:  
 request\_body\_size = int(*request*.get('CONTENT\_LENGTH', 0))  
 *except* (ValueError):  
 request\_body\_size = 0  
  
 request\_body = *request*['wsgi.input'].read(request\_body\_size)  
 data = parse\_qs(request\_body)  
  
 user = data.get(b"user")[0].decode("utf8")  
 pwd = data.get(b"pwd")[0].decode("utf8")  
 conn = pymysql.connect(host='101.132.161.180', port=3306, user='root', passwd='123456', db='new\_web') # db：库名  
 # 创建游标  
 cur = conn.cursor()  
 SQL = "select \* from userinfo WHERE NAME ='%s' AND PASSWORD ='%s'" % (user, pwd)  
 cur.execute(SQL)  
  
 *if* cur.fetchone():  
 f = open("templates/index.html", "rb")  
 data = f.read()  
 data = data.decode("utf8")  
 *return* data.encode("utf8")  
 *else*:  
 *return* b"user or pwd is wrong"

modules.py

# -\*- coding: utf-8 -\*-  
# \_\_title\_\_ = 'modules.py'  
# \_\_author\_\_ = 'YangYang'  
# \_\_mtime\_\_ = '2018.06.19'  
# 生成用户表  
*import* pymysql  
#连接数据库  
conn = pymysql.connect(host='101.132.161.180',port= 3306,user = 'root',passwd='123456',db='new\_web') #db：库名  
#创建游标  
cur = conn.cursor()  
sql='''  
create table userinfo(  
 id INT PRIMARY KEY ,  
 name VARCHAR(32) ,  
 password VARCHAR(32)  
)  
'''  
cur.execute(sql)  
#提交  
conn.commit()  
#关闭指针对象  
cur.close()  
#关闭连接对象

## 5.MVC与MTV模型

### 1.MVC

Web服务器开发领域注明的MVC模式，所谓MVC就是把Web应用分为模型（M）、控制器（C）和视图（V）三层，他们之间以一种插件式的、松耦合的方式连接在一起，模型负责业务对象与数据库的映射（ORM），视图负责与用户的交互（页面），控制器接受用户的输入调用模型和视图完成用户的请求，其示意图如下所示：



### 2.MTV

Django的MTV模式本质上和MVC是一样的，也是为了各组件间保持松耦合关系，只是定义上有些许不同，Django的MTV分别是：

* M代表模型（Model）：负责业务对象和数据库的关系映射（ORM）。
* T 代表模板（Template）：负责如何把页面展示给用户（html）。
* V 代表视图（View）。负责业务逻辑，并在适当时候调用Model和Template

除了以上三层之外，还需要一个URL分发器，它的作用是将一个个URL的页面请求分发给不同的View处理，View再调用相应的Model和Template，MTV的响应模式如下所示：



一般用户通过浏览器向我们的服务器发起一个请求（request），这个请求回去访问视图函数，（如果不涉及到数据调用，那么这个时候视图函数返回一个模板也就是一个网页给用户），视图函数调用模型，模型去数据库查找数据，然后逐级返回，视图函数把返回的数据填充到模板中，最后返回网页给用户。

## 6.Django的下载与基本命令

### 1.安装Django

windows : python3 -m pip install django===2.0.1

django安装路径地址：C:\Python\Python36\Scripts

linux : python3 install django===2.0.1

django安装路径地址：/usr/local/python36/bin

### 2.创建一个Django project

创建一个文件夹用于存放django项目：mkdir my\_pro，然后执行

windows：django-admin startproject mysite

linux： django-admin.py startproject mysite

### 3.创建mysite后生成的文件的作用

[root@dev mysite]# tree

├── manage.py ----- Django项目里面的工具，通过它可以调用django shell和数据库等。

└── mysite

├── \_\_init\_\_.py

├── settings.py ----包含了项目的默认设置,包括数据库信息,调试标志以及其他一些工作的变量。

├── urls.py ----- 负责把URL模式映射到应用程序。路由相关

└── wsgi.py ----- 封装的socket，实际上，django走的第一环就是这个文件

### 4.创建一个应用属于mysite项目

cd mysite  
python3 manage.py startapp blog  
  
[root@dev mysite]# tree blog/  
blog/  
├── admin.py  
├── apps.py  
├── \_\_init\_\_.py  
├── migrations  
│   └── \_\_init\_\_.py  
├── models.py ---- 数据库相关  
├── tests.py  
└── views.py ---视图函数

### 5.创建一个templates目录，用于存放html文件



### 6.启动django项目

python3 manage.py runserver 8080 (默认8000端口)

## 7.Django简单示例及静态文件配置

### 1.创建一个 static\_file\_configuration 项目和一个app01的应用



### 2.需求说明

利用django实现一个页面展示当前时间，并且点击以后不断变色的方案

### 3.django访问顺序

首先，当用户通过URL访问服务器django项目的时候，首先通过urls控制器去匹配一个视图函数，然后，如果匹配到视图函数，那么会跳转到views.py里面找到对应的函数名称，然后，views.py会把要展示的数据给到templates，templates里面的页面在经过处理后，会把html页面发送给用户。因此，接下来的的步骤就是：urls.py ---> views.py ---->template --return-->用户

### 4.urls.py控制器配置

*from* django.contrib *import* admin  
*from* django.urls *import* path  
  
*from* app01 *import* views #导入对应应用的视图  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
]

### 5.views.py 创建视图（逻辑业务处理）

*def* timer(*request*):  
 *import* time  
 ctime = time.time()  
 *return* render(*request*,'timer.html',{"ctime":ctime})

### 6.创建页面

注意：django 传送变量给html的时候，在html里面用 {{变量}} 去使用变量。

<!DOCTYPE html>  
<**html** lang=**"en"**>  
<**head**>  
 <**meta** charset=**"UTF-8"**>  
 <**title**>**timer**</**title**>  
 <**link** rel=**"stylesheet"** href=**"/static/app01/timer.css"**>  
</**head**>  
<**body**>  
<**h4**>**hi** {{ ctime }}</**h4**>  
</**body**>  
 <**script** src=**"/static/jquery-3.3.1.js"**></**script**>  
 <**script** type=**"text/javascript"** src=**"/static/app01/timer.js"**></**script**>  
</**html**>

### 7.创建css js

css文件：

h4{  
 *color*: **red**;  
}

js文件：

$(function () {  
 var a = 0;  
 $('h4').click(function () {  
 a++;  
 if(a%4===1){  
 $(this).css('color','green')  
 }else if(a%4===2) {  
 $(this).css('color','yellow')  
 }else if (a%4===3){  
 $(this).css('color','blue')  
 }else{  
 $(this).css('color','red')  
 }  
 })  
});

### 8.静态文件说明（重要）

静态文件我们一般会在该项目下面创建一个static的目录，然后用应用名称来解耦合，存放各自应用的js和css文件。如下图所示：



但是，此刻会发现，在该项目下，创建的是一个 statics的目录去存放，但是，为啥在html文件里面引入的是 static呢。这里就需要配置项目的setting.py文件

# statics的别名  
STATIC\_URL = '/static/'  
  
STATICFILES\_DIRS=[  
 os.path.join(BASE\_DIR,"statics")  
]

STATIC\_URL = '/static/' 这个是是静态文件目录的别名。这里可以更改为别的，但是一般情况下都是使用static。

STATICFILES\_DIRS ：这个是静态文件在该项目下的路径。

BASE\_DIR 可以在setting.py 的文件上面找到。

### 9.启动django项目

第一种方法：

点击箭头启动

第二种方法：python3 manage.py runserver 8000

### 10.页面展示



## 8.路由配置

### 1.路由控制之简单配置（基于正则表达式）

*from* django.contrib *import* admin  
*from* django.urls *import* path,re\_path  
# path django===2.0.\*  
# re\_path django===1.0.\*  
*from* app01 *import* views #导入对应应用的视图  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
 # 路由配置： 路径----> 视图函数  
 re\_path(r'^articles/2003/$', views.special\_case\_2003), #special\_case\_2003(request) request是请求对象  
 # 只要url里面有个正则分组，它就会作为一个位置参数，就要多传一个参数到对应的视图函数里面  
 re\_path(r'^articles/([0-9]{4})/$', views.year\_archive),  
 re\_path(r'^articles/([0-9]{4})/([1-9]{1}|1[0-2]{1})/$', views.month\_archive),  
 # re\_path(r'^articles/([0-9]{4})/([1-9]{1}|1[0-2]{1})/([0-9]+)/$', views.article\_detail),  
]

1.首先导入一个 re\_path 。

path是django 2.\* 版本以上的，但是，django兼容1.\* 版本的 re\_path

2.上面的re\_path 中的配置项是基于正则表达式的匹配规则。

一个路径（或者叫做匹配项）对应一个视图函数，要精确匹配 那就要让正则前后闭合“^” “$” 。并且，匹配规则为从上到下匹配，当匹配到一个re\_path的路径以后，就不在往下匹配。

3.要从URL中捕获一个值，只需要在它周围放置一个圆括号。

4.不需要添加一个前导的反斜杠，因为每个URL都有。例如，应该是^articles 而不是 ^/articles。

5. 每个正则表达式前面的'r' 是可选的但是建议加上。它告诉Python 这个字符串是“原始的” —— 字符串中任何字符都不应该转义

### 2.路由控制之有名分组

urls.py

*from* django.contrib *import* admin  
*from* django.urls *import* path,re\_path  
  
*from* app01 *import* views   
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
  
 re\_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[1-9]{1}|1[0-2]{1})/(?P<day>[1-9]{1}|1[0-9]{1}|2[0-9]{1}|3[0-1]{1})/$', views.article\_detail),  
]

views.py

*from* django.shortcuts *import* render,HttpResponse

*def* article\_detail(*request*,*month*,*day*,*year*):  
 *return* HttpResponse(*year*+'-'+*month*+'-'+*day*)

在Python 正则表达式中，命名正则表达式组的语法是(?P<name>pattern)，其中name 是组的名称，pattern 是要匹配的模式。

### 3.路由控制之分发

如果一个项目下有多个应用，那把所有的应用的url都放在主项目下的urls.py 里面，那就非常混乱，代码耦合太高。因此，把各自的应用的URL放在各自的应用目录下面。

1.首先在应用下面创建一个urls.py(名字随便取) .



2.在主项目下导入一个include方法，并添加一条路由规则

*from* django.contrib *import* admin  
*from* django.urls *import* path,re\_path,**include**  
*from* app01 *import* views #导入对应应用的视图  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
 # 分发  
 # re\_path(r"app01/",include("app01.urls"))  
 re\_path(r"^",include("app01.urls"))  
]

3.把应用的URL写在自己应用的app01 下的 urls.py 里面

*from* django.contrib *import* admin  
*from* django.urls *import* path,re\_path  
*from* app01 *import* views #导入对应应用的视图  
urlpatterns = [  
 re\_path(r'^articles/([0-9]{4})/$', views.year\_archive),  
 re\_path(r'^articles/([0-9]{4})/([1-9]{1}|1[0-2]{1})/$', views.month\_archive),  
 re\_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[1-9]{1}|1[0-2]{1})/(?P<day>[1-9]{1}|1[0-9]{1}|2[0-9]{1}|3[0-1]{1})/$', views.article\_detail),  
]

4.在浏览器里面访问就可以得到自己的页面

views.py的视图：

*def* article\_detail(*request*,*month*,*day*,*year*):  
 *return* HttpResponse(*year*+'-'+*month*+'-'+*day*)

URL访问页面在控制器里面的特别说明：

re\_path(r"app01/",include("app01.urls")),

如果在主项目下配置的项目URL，那么就要在url路径当中加app01（项目名称）：

例如：<http://127.0.0.1:8000/app01/articles/2013/4/12/>

re\_path(r"^",include("app01.urls")),

如果在主项目下的urls.py里面配置成这样的，那在url中就直接使用规则去访问即可：

例如：http://127.0.0.1:8000/articles/2013/4/12/

### 4.路由控制之登陆验证示例

1.在主项目的urls.py中加入一个login的路由：

urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
 path('login/',views.login)  
]

意思就是：当URL 后面是 login 的时候跳转到指定的视图 login 里面去进行逻辑处理以及获取页面

2.创建一个login.html的页面

<!DOCTYPE html>  
<**html** lang=**"en"**>  
<**head**>  
 <**meta** charset=**"UTF-8"**>  
 <**title**>**login.html**</**title**>  
</**head**>  
<**body**>  
<**form** action=**"http://127.0.0.1:8000/login/"** method=**"post"**>  
 **用户名：** <**input** type=**"text"** name=**"user"**>  
 **密码：** <**input** type=**"password"** name=**"pwd"**>  
 <**input** type=**"submit"**>  
</**form**>  
</**body**>  
</**html**>

3.对用户的请求做逻辑处理

特别提示：用户请求包含GET请求和POST请求。其中GET请求是用户访问login 的URL以后，服务器返回一个login页面，此时就完成了GET请求。然后，用户输入了用户名密码以后，点击提交，此时就是一个POST请求。因此：

GET请求：用户拿到login.html 页面

POST请求：用户发送密码信息给服务器端，然后服务器端进行逻辑处理（校验用户名密码）

4.views.py里面的逻辑处理等

*def* login(*request*):  
 # get请求直接拿到 login.html  
 # post 请求直接验证 用户信息  
 print("methon:",*request*.method)  
 *if request*.method == "GET":  
 print("GET:", *request*.GET)  
 *return* render(*request*,"login.html")  
 *elif request*.method == "POST":  
 print("POST:", *request*.POST) # POST: <QueryDict: {'user': ['ryan'], 'pwd': ['1234']}>  
 user = *request*.POST.get("user")  
 pwd = *request*.POST.get("pwd")  
 *if* user == 'ryan' *and* pwd == '123':  
 *return* HttpResponse('登陆成功')  
 *else*:  
 *return* HttpResponse('登陆失败')

## 9.流程控制之反向解析

在使用Django项目时，一个常见的需求是获取URL的最终形式，以用于嵌入到生成的内容中（视图中和显示给用户的URL等）或者用于处理服务器端的导航（重定向等）。人们强力希望不要硬编码这些URL（费力、不可扩展且容易产生错误）或者设计一种与URLconf毫不相关的专门的URL生成机制，因为这样容易导致一定程度上产生过期的URL。

在需要URL的地方，对于不同层级，Django提供不同的工具用于URL反查：

* 在模板中：使用url模块标签。
* 在python中：使用from django.urls import reverse 的reverse() 函数

### 1.html中使用反向解析

在主项目的urls.py中配置一个login的路由：

urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
 path('login.html/',views.login,name="Login"),  
]

其中，name=“Login”是 /login.html 的别名；

在对应项目中的 views.py 中的逻辑处理：

*def* login(*request*):  
 *if request*.method == "GET":  
 *return* render(*request*,"login.html")  
 *elif request*.method == "POST":  
 user = *request*.POST.get("user")  
 pwd = *request*.POST.get("pwd")  
 *if* user == 'ryan' *and* pwd == '123':  
 *return* HttpResponse('登陆成功')  
 *else*:  
 *return* HttpResponse('登陆失败')

在html页面 用url模块标签去导入对应的别名：

<!DOCTYPE html>  
<**html** lang=**"en"**>  
<**head**>  
 <**meta** charset=**"UTF-8"**>  
 <**title**>**login.html**</**title**>  
</**head**>  
<**body**>  
<**form** action={% **url** 'Login' %} **method="post"**>  
 **用户名：** <**input** type=**"text"** name=**"user"**>  
 **密码：** <**input** type=**"password"** name=**"pwd"**>  
 <**input** type=**"submit"**>  
</**form**>  
</**body**>  
</**html**>

注意：其中的 {% url 'Login' %} 这个是url模块去调用一个在Django 项目组中 urls.py 下面的别名。



然后现在来了需求，说 不在使用 login.html 去显示登陆页，如果不用 url模块去获取地址，那就会去修改html标签中 form的action的地址，这样就导致代码修改困难，地址不正确等。所以，现在要使用/login/去访问登陆也，只需要修改urls.py中的路由即可；

urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
 path('login/',views.login,name="Login"),  
]



### 2.在python代码中使用反向解析

在主项目中有一个导入的 app01 的web应用：

urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('timer/',views.timer), #添加路由转发  
 path('login/',views.login,name="Login"),  
 path('app01/',include('app01.urls'))  
]

include 就是导入app01应用的路由转发，主要是为了让结构代码看上去清晰。

APP01 的urls.py

*from* django.urls *import* path,re\_path  
*from* app01 *import* views #导入对应应用的视图  
urlpatterns = [  
 re\_path(r'^articles/2003/$', views.special\_case\_2003,name='s\_c\_2003'),  
 re\_path(r'^articles/([0-9]{4})/$',views.year\_archive,name='y\_a'),  
 re\_path(r'^articles/([0-9]{4})/([1-9]{1}|1[0-2]{1})/$', views.month\_archive),  
 re\_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[1-9]{1}|1[0-2]{1})/(?P<day>[1-9]{1}|1[0-9]{1}|2[0-9]{1}|3[0-1]{1})/$', views.article\_detail),  
]

在app01 的views.py中导入 反向解析的函数模块：

*from* django.urls *import* reverse  
*def* special\_case\_2003(*request*):  
 # HttpResponse 响应对象  
 url = reverse('s\_c\_2003')  
 print(url) # /app01/articles/2003/  
 *return* HttpResponse('special\_case\_2003')  
  
*def* year\_archive(*request*,*year*):  
 url = reverse('y\_a',args=(*year*,))  
 print(url) # /app01/articles/2012/  
 *return* HttpResponse('<p style="color: blue">%s year</p>'%*year*)  
  
*def* month\_archive(*request*,*month*,*year*):  
 url = reverse('y\_a',args=(1007,))  
 print(url) #/app01/articles/1007/  
 *return* HttpResponse('<p style="color: blue">%s year %s month</p>'%(*year*,*month*))

在python中反向解析的用法总结：

1、如果在路由转发中的URL不带不确定项（也就是精确匹配），那么在reverse进行反向解析的时候就不需要使用参数，直接使用别名即可。

2、如果在路由转发中的URL带了不确定项,就像上列中的 r'^articles/([0-9]{4})/$.那么在反向解析的时候，就要加入一个参数，不然会报错。

3、可以在views.py 中的任何地方使用别名获取到对应的地址。