django+mysql的web项目

工具准备：

python，pycharm，django，mysql，git全部安装并能正常使用

git使用：

上传本地代码：

1. 在包含git文件的目录里右键git bash here
2. git add .
3. git commit -m “描述”
4. git push

同步远程代码到本地：

1. 在包含git文件的目录里右键git bash here
2. git pull

搭建环境：

1. 创建一个文件夹，cmdcd到文件夹创建虚拟环境：python -m venv wlenv（名字）
2. 激活虚拟环境：wlenv\Scripts\activate
3. 安装django：pip install django
4. 创建django项目：django-admin startporject worklog（名字）
5. 同步数据库：python manage.py migrate
6. 测试django环境：python manage.py runserver（可选ip+端口号）
7. 切库，在mysql创建一个数据库eg：worklog\_db设置字符集为utf-8
8. 切库，安装mysqlclient：pip install mysqlclient
9. 切库，配置settings文件：在DATABASES里将engine数据库改为mysql，填上数据库名字，用户名和密码。
10. 切库，同步数据库：python manage.py migrate
11. 切库，查看数据库中是否创建了表格，runserver服务查看是否切库成功
12. 创建项目app：python manage.py startapp worklogweb（django前端）
13. 配置settings文件，把新创建的APP加入INSTALLED\_APP中
14. 修改语言，settings文件里最后LANGUAGE\_CODE:’zh-Hans’
15. 修改时区，settings文件最后TIME\_ZONE:中国范围内有Asia/Shanghai, Asia/Harbin, Asia/Chongqing, Asia/Urumqi, Asia/Kashgar

web类项目概述

web标准框架：MVC（model，view，controller）

model 模型，控制数据库表结构

view 视图，负责具体前端输出的内容，eg颜色，文字，表单

controller 控制器，具体负责实现的功能，eg增删改查，加减乘除

Router路由

用户访问地址的地址，通过router映射到controller的功能

Template渲染模板

用户将要看的页面，html

django框架：MTV（model，template，view）对应MVC，只是名字不同



用户发出访问请求—>django router（URL）将请求匹配path路径到views里的具体函数—>view根据被匹配的函数执行代码—>可能执行model调取数据库内的数据—>将所有数据渲染成模板发送到用户的浏览器展示

项目文件

urls.py

HTTP请求进入django时有限调用该文件，文件里包含所有路由，第一个参数是访问的地址，第二个参数是要调用的views.py里的映射文件。

settings.py

项目配置文件

wsgi.py

正式启用django项目时配置该网关文件

views.py

文件包含调用函数，让程序调用template里的html页面渲染并发送

setting.py

公有配置

BASE\_DIR=os.path(...)项目绝对路径，注册使用别的文件夹或文件时会用到

debug

启动模式，True为调试模式

1. 检测代码改动后立刻重启服务
2. 报错页面

False为正式模式

ALLOWED\_HOSTS=[ ]

请求头HOST头，只接受请求头包含其中的请求，有过滤作用

局域网内其他主机访问django项目服务

1. 启动方式为

python manage.py runserver 0.0.0.0:8000 （四个0代表服务器所有网卡IP都能被访问）

1. 添加服务器内网IP至ALLOWED\_HOSTS=[内网IP]

INSTALLED\_APPS

配置django应用

MIDDLEWARE

中间件，高级应用

ROOT\_URLCONF

配置主路由文件

自定义配置

1. 大写
2. 个性化，不要与公有配置重名

URL

定义：统一资源定位符

作用：用来表示互联网上某个资源的地址

一般语法格式为：protocol://hostname[:port]/path[?query][#fragment] [ ]内可省略，共6大部分。

1. protocol（协议）指使用的什么协议
2. http通过HTTP访问该资源
3. https通过安全的HTTPS访问该资源
4. file资源是本地计算机上的文件。格式file:///
5. bostname（主机名）是指存放资源的服务器的域名系统（DNS）主机名、域名或IP
6. port（端口号）

1）整数、可选（范围为1024—65535，用户可用的端口号）

1. path（路由地址）

由0或多个“/”符号隔开的字符串，一般用来表示主机上的一个目录或文件地址。路由地址决定了服务器端如何处理这个请求

1. query（查询）以问号开头，问号后都是查询字符串

可选，用于给动态网页传递参数，可有多个参数，用“&”符号隔开，每个参数名和值用“=”隔开

1. fragment（信息片段，锚点）以#号开始，类似书签，前端技术，输入带锚点的地址浏览器将直接定位到锚点所在地方

处理URL请求

eg：浏览器地址栏输入http://127.0.0.1:8000/page/2003/

1. django从settings.py文件中根据ROOT\_URLCONF找到主路由，默认情况下该文件在项目同名目录下的urls.py
2. django加载主路由文件中的urlpatterns变量：一个包含很多path的数组
3. 依次匹配urlpatterns中的path，匹配到第一个合适的path后中断后续匹配
4. 匹配成功—调用对应视图函数处理请求，返回响应
5. 匹配失败—返回404响应

视图函数

用于接收浏览器请求（HttpResquest对象）并通过HttpResquest对象返回响应的函数。此函数可以接收浏览器请求并根据业务逻辑返回响应的内容给浏览器

语法

def XXX\_view（request[,其他参数...]）:

return HttpResponse对象

路由配置

path（）函数

导入：from django.urls import path

语法：path（route, views,name=None）

参数：

1. route：字符串类型，匹配请求路径
2. views：指定路径所对应的试图处理函数的名称
3. name：为地址起别名，在模板中地址反向解析时使用

path转换器

语法：<转换器类型：自定义名>

作用：若转换器类型匹配到对应类型的数据，则将数据按照关键字传参的方式传递给视图函数

eg：path(‘page’/<int:page>’, views.xxx)

转换器类型：

1. str匹配除了/之外的非空字符串
2. int匹配0或任何正整数，返回一个int
3. slug匹配任意由ASCII字母或数字以及连字符和下划线组成的短标签
4. path匹配非空字段，包括路径分隔符

views.py里的函数要指定自定义的转换器的名字，相当于传参进来

eg：def pagex(request, page):

re\_path（）函数（精准匹配）

在url的匹配过程中可以施工正则表达式

语法：re\_path(reg, view, name=xxx) 正则表达式为命名分组模式(?P<name>pattern);匹配提取参数后用关键字传参的方式传递给视图函数

请求和响应

请求是指浏览器通过HTTP协议发送给服务器端的数据

响应是指服务器端接收到请求后做相应的处理后再回复给浏览器端的数据



HTTP1.0定义了三种请求方法：GET、POST、HEAD（开发最常用网民用不到）

HTTP1.1新增了五种请求方法：OPTIONS、PUT、DELETE、TRACE、CONNECT





put：更新

delete：删除

connect：翻墙用

django处理请求的流程

1. 请求在django中实则就是视图函数的第一个参数，即HttpRequest对象
2. django接收到http协议的请求后，会根据请求数据报文创建HttpRequest对象
3. HttpRequest对象通过属性描述了请求的所有相关信息





django响应



HTTP状态码的英文为HTTP Status Code

200：请求成功

301：永久重定向—资源（网页等）被永久转移到其他URL

302：临时重定向（登录即临时重定向）

404：请求的资源（网页等）不存在

500：内部服务器错误（一定是代码有错）

HTTP状态码共分为5类

1\*\*：信息，服务器收到请求，需要请求者继续执行操作

2\*\*：成功，操作被成功接收并处理

3\*\*：重定向，许哟进一步的操作以完成请求

4\*\*：客户端错误，请求包含语法错误或无法完成请求

5\*\*：服务器错误，服务器在处理请求的过程中发生了错误

django中的响应对象

构造函数格式：

HttpResponse(content=响应体, content\_type=响应体数据类型, status=状态码)

作用：

向客户端浏览器返回响应，同时携带响应体内容

常用的content\_type：

1. text/html（默认的，html文件）
2. text/plain（纯文本）
3. text/css（css文件）
4. text/javascript（js文件）
5. multipart/form-data（文件提交）
6. application/json（json传输）
7. application/xml（xml文件）

HttpResponse子类



GET和POST请求

统一由视图函数接收请求，通过判断request.method区分具体的请求动作

样例：

if request.method == ‘GET’:

处理GET请求的代码块

elif request.method == ‘POST’:

处理POST请求的代码块（处理用户提交的数据）

elif XXXXX == ‘XXXX’:

......

GET，一般用于向服务器获取数据

能够产生GET请求的场景：

1. 浏览器地址栏中输入URL，回车后
2. <a href=”地址?参数=值&参数=值”>
3. form表单中的method为GET

GET请求中，如果有数据需要传递给服务器，通常用查询字符串Query String 注：不要传递敏感数据

url格式：xxx?参数1=值1&参数2=值2...

服务器端接收参数，获取客户端请求GET提交的数据：

request.GET[‘参数名’]

request.GET.get(‘参数名’, ‘默认值’)

request.GET.getlist(‘参数名’)

单个key可以对应多个value，用数组盛放，用get取只取数组中的最后一个值，用getlist拿能拿到全部值（直接拿一个数组）。

典型应用场景：问卷调查的多选

浏览器传递非敏感数据给服务器方式

1. 通过url传递
2. 通过GET请求的查询字符串Query String传递

POST，一般向服务器提交大量/隐私数据

客户端通过表单等POST请求将数据传递给服务器

<form method = ‘POST’ action = “/login”>

姓名：<input type = “text” name = ”username”>

<input type = ‘submit’ value = ’登录’>

</form>

action：指明form表单将数据发给谁，浏览器会将ip和action里的值组合起来作为地址发送数据

name：传递数据时作为key名

服务器端接收参数，通过request.method来判断是否为post请求

if request.method == ‘POST’:

处理POST请求的代码块

else:

...

request.POST[‘参数名’]

request.POST.get(‘参数名’, ‘’)

request.PSOT.getlist(‘参数名’)

取消csrf验证，否则django将会拒绝客户端发来的POST请求，报403

注释掉settings.py中MIDDLEWARE中的CsrfViewsMiddleWare的中间件

MTV模式

T：模板

模板层（Templates）（html页面）

1. 模板是可以根据字典数据动态变化的html网页
2. 模板可以根据视图中传递的字典数据动态生成相应的html网页

模板配置

1. 创建模板文件夹<项目名>/templates
2. 在settings.py中TEMPATES配置项
3. BACKEND：指定模板引擎
4. DIRS：模板的搜索目录（可以是一个或多个）
5. APP\_DIRS：是否要在应用中的templates文件夹中搜索模板文件
6. OPTIONS：有关模板的选项
7. 配置中需要修改的部分

设置DIRS - ‘DIRS’: [os. path. join(BASE\_DIR, ‘templates’)],

模板加载方式

1. 通过loader获取模板，通过HttpResponse进行响应

在视图函数中

from django.template import loader

1. 通过loader加载模板

t=loader.get\_template(“模板文件夹”)

1. 将t转换为html字符串

html=t.render(字典数据)

1. 用响应对象将转换的字符串内容返回给浏览器

return HttpResponse(html)

1. 使用render()直接加载并响应模板

在views.py函数中

from django.shortcuts import render

return render(request, ’模板文件名’, 字典数据)

视图层与模板层之间的交互

1. 视图函数中可以将python变量封装到字典（dic）中传递到模板

样例：

def xxx\_view(request):

dic = {

“变量1”: “值1”,

“变量2”: “值2”,

}

return render(request, ‘xxx.html’, dic)

1. 模板中，可以用{{ 变量名 }}的语法调用视图传进来的变量

模板的变量

能传递到模板中的数据类型

str-字符串 int-整型

list-数组 tuple-元组

dict-字典 func-方法

obj-类实例化的对象

在模板中使用变量语法

{{ 变量名 }}

{{ 变量名.index }} 调用list容器中的某一个值

{{ 变量名.key }} 调用字典中某个键的值

{{ 对象.方法 }} 调用对象的方法

{{ 函数名 }} 调用函数

模板（html）标签

将一些服务器端的功能嵌入到模板（html）中，例如流程控制（for循环，if控制）等

标签语法

{% 标签 %}

...

{% 结束标签 %}

if标签

{% if 条件表达式1 %}

...

{% elif 条件表达式2 %}

...

{% else %}

...

{% endif %}

if标签配和selected可以实现记录用户上次选择的功能，视频p10 33min

for标签

{% for 变量 in 可迭代对象 %}

... 循环语句

{% empty %}

... 可迭代对象无数据时填充的语句

{% endfor %}

内置变量-forloop

变量 描述

forloop.counter 循环的当前迭代（从1开始索引）

forloop.counter0 循环的当前迭代（从0开始索引）

forloop.revcounter counter值得倒序

forloop.revcounter0 revcounter值的倒序

forloop.first 如果这是第一次通过循环，则为真

forloop.last 如果这是最后一次循环，则为真

forloop.parentloop 当前嵌套循环，parentloop表示外层循环

模板过滤器

定义：在变量输出时对变量的值进行处理

作用：可以通过使用过滤器来改变变量的输出显示

语法：{{ 变量| 过滤器1: ’参数值1’ | 过滤器2: ‘参数值2’... }}

常用过滤器：

lower 将字符串全部转换为小写

upper 将字符串全部转换为大写

safe 默认不对变量内的字符串进行html转义

add: “n” 将value的值增加n

truncatechars: ‘n’ 如果字符串字符多于指定的字符数量，那么会被截断。截 断的字符串将以可翻译的省略号序列(“...”)结尾

模板层继承

模板继承可以使父模板的内容重用，子模板直接继承父模板的全部内容并可以覆盖父模板中相应的块

语法—父模板中：

1. 定义父模板中的块block标签
2. 标识出哪些在子模块中时允许被修改的
3. block标签：在父模板中定义，可以在子模版中覆盖

语法—子模板

1. 继承父模板extends标签（写在模板文件第一行）

例如{% extends ‘base.html’ %}

1. 子模板重写父模板中的内容块

{% block block\_name %}

子模板块用来覆盖父模板中 block\_name 块的内容

{% endblock block\_name %}

重写覆盖规则

1. 不重写，将按照父模板的效果显示
2. 重写，则按照重写的效果显示

注：子模板继承不了父模板的变量

url地址

绝对地址：

<http://127.0.0.1:8000/page/1>

相对地址：

1. ’/page/1’ / 开头的相对地址，浏览器会把当前地址栏里的协议，ip和端口加上这个地址，作为最终访问地址。

eg：当前页面地址栏为http://127.0.0.1:8000/page/3，当前相对地址结果为http://127.0.0.1:8000/page/1

1. ‘page/1’ 没有 / 开头相对地址，浏览器会根据当前url的最后一个/之前的内容加上该相对地址作为最终访问地址。

eg：当前页面地址栏为http://127.0.0.1:8000/topic/detail，当前相对地址结果为http://127.0.0.1:8000/topic/page/1

url反向解析

指在视图或模板中，用path定义的名称来动态查找或计算出相应的路由

path函数的语法

path(route, views, name=”别名”)

eg：path(‘page’,views.page\_view, name=”page\_url” )

根据path中的name关键字传参给url定义一个唯一的名字，在模板或视图中，可以通过这个名字反向推断出此url信息

模板中—通过url标签实现地址的反向解析

{% url ‘别名’ %}

{% url ‘别名’ ‘参数值1’ ‘参数值2’ %}

eg：{% url ‘pagen’ ‘400’ %} 位置传参

{% url ‘person’ age=’18’ name=’xxx’ %} 关键字传参

在视图函数中可调用django中的reverse方法进行反向解析

from django.urls import reverse

reverse(‘别名’, args=[ ], kwargs={ }) args为位置传参 kwargs为关键字传参

eg：reverse(‘pagen’, args=[300])

reverse(‘person’, kwargs={‘name’ : ‘xxx’, ‘age’ : 18 })

静态文件

1. 配置settings.py中静态文件访问路径

SRTATIC\_URL=’/static/’

1. 配置settings.py中存储路径STATICFILES\_DIRS

STATICFILES\_DIRS保存的时静态文件在服务器端的存储位置

STATICFILES\_DIRS=(os.path.join(BASE\_DIR, “static”),)

模板中访问静态文件—img标签为例

1. 直接拼接访问路径

<img src=”/static/images/lena.jpg”>

<img src=”http://127.0.0.1:8000/static/images/lena.jpg”>

1. 通过{% static %}标签访问静态文件（推荐）
2. 加载static：{% load static %}
3. 使用静态资源：{% static ‘静态资源路径’ %}
4. eg：<img src=”{% static ‘images/lena.jpg’ %}”>

标签访问静态文件更动态

1212