

Django简介

什么是Django？



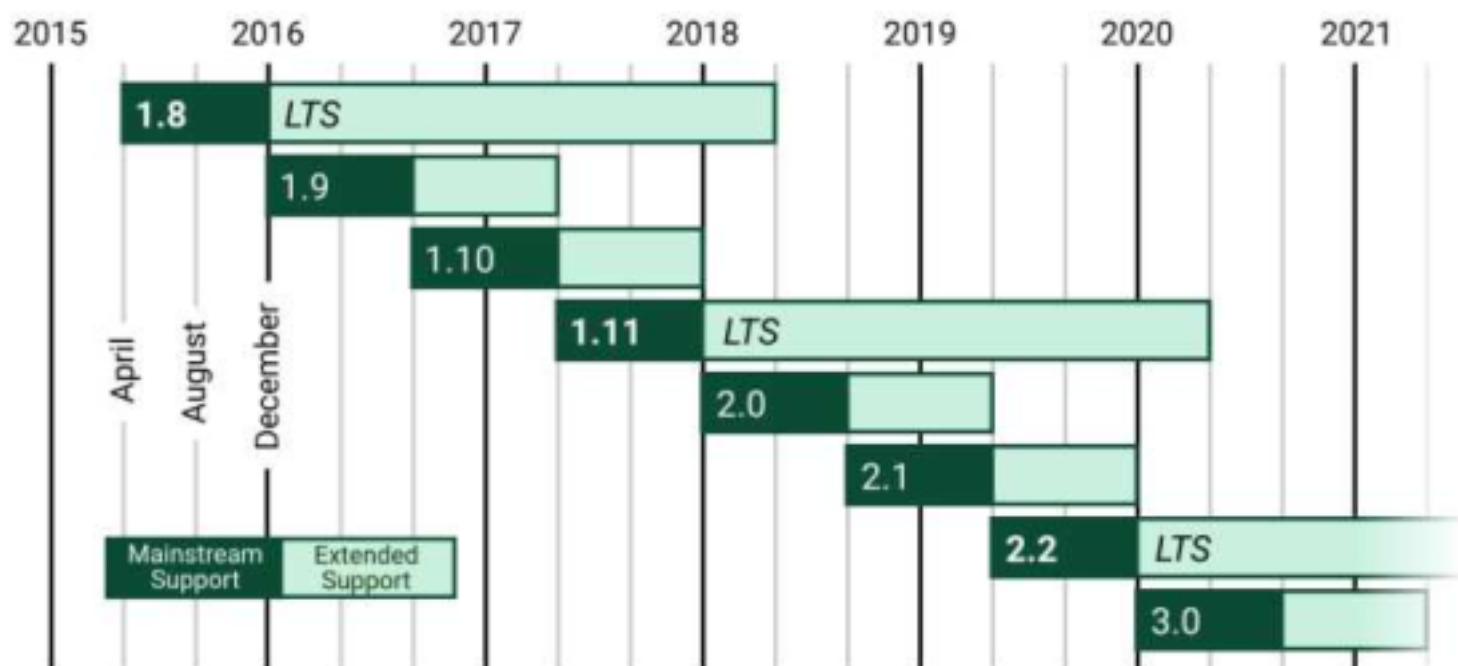
Django是居于Python编写的一个web应用框架。

Django遵守BSD版权，初次发布于2005年7月, 并于2008年9月发布了第一个正式版本1.0。

Django采用了MVC的软件设计模式，即模型M，视图V和控制器C。

Django版本对应的 Python 版本：

Django 版本	Python 版本
1.8	2.7, 3.2 , 3.3, 3.4, 3.5
1.9, 1.10	2.7, 3.4, 3.5
1.11	2.7, 3.4, 3.5, 3.6
2.0	3.5+



Django安装

Django 安装

pip 命令安装方法（ubuntu安装）

```
lzy@embsky:~$ sudo apt install python-pip
```

```
lzy@embsky:~$ sudo pip install Django
```

注意：该方法安装到python2

```
lzy@embsky:~$ sudo apt install python3-pip
```

```
lzy@embsky:~$ sudo pip3 install Django
```

注意：该方法安装到python3

pip 命令安装方法（centos7安装）

```
[root@localhost site-packages]# yum list |grep pip
```

```
[root@localhost site-packages]# yum install python2-pip.noarch
```

```
[root@localhost site-packages]# pip install django
```

注意：该方法安装到python2

```
[root@localhost site-packages]# yum install python34-pip.noarch
```

```
[root@localhost site-packages]# pip3 install django
```

注意：该方法安装到python3

pip 命令安装方法（python虚拟环境中安装）

```
[zyli@localhost ~]$ mkdir venv
```

```
[zyli@localhost ~]$ python3 -m venv ./venv
```

接下来把venv/bin目录添加到PATH换进变量

```
[zyli@localhost ~]$ vim ~/.bashrc
```

```
PATH=/home/zyli/venv/bin:$PATH
```

```
[zyli@localhost ~]$ source ~/.bashrc
```

```
[zyli@localhost ~]$ pip install --upgrade pip
```

```
[zyli@localhost ~]$ pip install django
```

注意：本文当中的案例都是采用Python虚拟环境进行

Django创建项目

创建项目

Django 创建项目

Django 管理工具

```
lzy@embsky:~$ django-admin.py
```

Type 'django-admin help <subcommand>' for help on a specific subcommand.

Available subcommands:

```
[django]
```

```
check
```

```
compilemessages
```

```
createcachetable
```

```
dbshell
```

```
diffsettings
```

```
dumpdata
```

```
flush
```

```
inspectdb
```

```
loaddata
```

```
makemessages
```

```
makemigrations
```

```
migrate
```

```
runserver
```

```
sendtestemail
```

```
shell
```

```
showmigrations
```

```
sqlflush
sqlmigrate
sqlsequencereset
squashmigrations
startapp
startproject
test
testserver
```

创建第一个项目

使用 `django-admin.py` 来创建 HelloWorld 项目：

```
lzy@embsky:~$ django-admin.py startproject HelloWorld
```

创建完成后我们可以查看下项目的目录结构：

```
lzy@embsky:~$ tree HelloWorld/
```

```
HelloWorld/
|___ HelloWorld
|   |___ __init__.py
|   |___ settings.py
|   |___ urls.py
|   |___ wsgi.py
|___ manage.py
```

1 directory, 5 files

目录说明：

-HelloWorld: 项目的容器。

-manage.py: 一个实用的命令行工具，可让你以各种方式与该 Django 项目进行交互。

-HelloWorld/__init__.py: 一个空文件，告诉 Python 该目录是一个 Python 包。

-HelloWorld/settings.py: 该 Django 项目的设置/配置。

-HelloWorld/urls.py: 该 Django 项目的 URL 声明; 一份由 Django 驱动的网站"目录"。

-HelloWorld/wsgi.py: 一个 WSGI 兼容的 Web 服务器的入口，以便运行你的项目。

WSGI: PythonWeb服务器网关接口 (Python Web Server Gateway Interface, 缩写为 WSGI)是Python应用程序或框架和Web服务器之间的一种接口，已经被广泛接受, 它已基本达成它的可移植性方面的目标。

同步数据库

```
lzy@embsky:~/HelloWorld$ python3 manage.py makemigrations
```

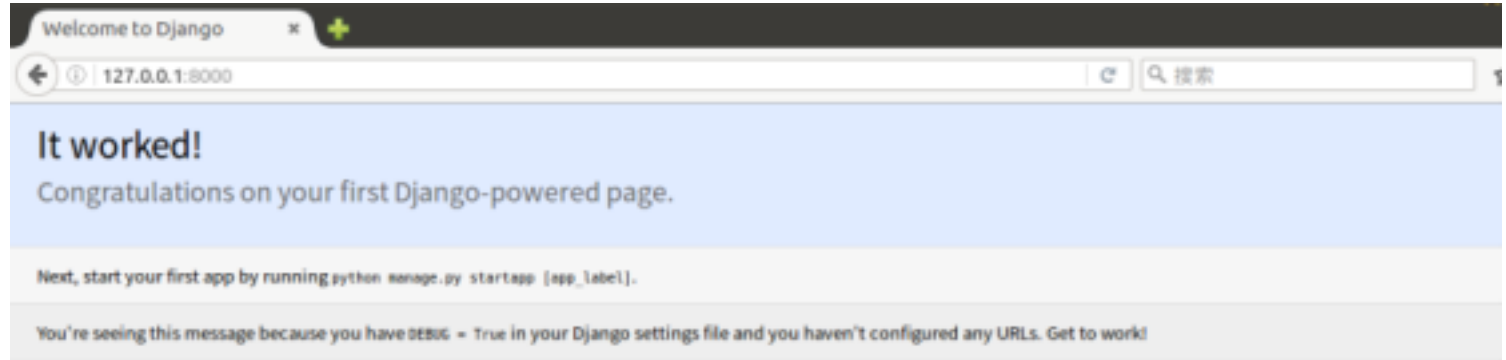
```
lzy@embsky:~/HelloWorld$ python3 manage.py migrate
```

注意：这种方法可以创建表,当你在models.py中新增了类时,运行它就可以自动在数据库中创建表了,不用手动创建

运行HelloWorld项目

```
lzy@embsky:~/HelloWorld$ python3.5 manage.py runserver 0.0.0.0:8000
```

在浏览器中输入IP和端口号测试HelloWorld项目



新添加一个视图

在HelloWorld 目录下的 HelloWorld 目录新建一个 view.py 文件

```
lzy@embsky:~/HelloWorld/HelloWorld$ touch myview.py
```

```
lzy@embsky:~/HelloWorld/HelloWorld$ vim myview.py
```

输入如下内容：

```
from django.http import HttpResponse
def helloFun(request):
    return HttpResponse("Hello Django")
```

修改URL 配置

```
lzy@embsky:~/HelloWorld/HelloWorld$ vim urls.py
```

修改如下：

```
from django.conf.urls import url
from django.contrib import admin
from . import myview
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', myview.helloFun),
    url(r'^nihao$', myview.helloFun),
]
```

在浏览器中输入IP和端口号测试HelloWorld项目



注意：项目中如果代码有改动，服务器会自动监测代码的改动并自动重新载入，所以如果你已经启动了服务器则不需手动重启。

url() 函数

Django url() 可以接收四个参数。

两个必选参数：regex、view

两个可选参数：kwargs、name，接下来详细介绍这四个参数。

-regex: 正则表达式，与之匹配的 URL 会执行对应的第二个参数 view。

-view: 用于执行与正则表达式匹配的 URL 请求。

-kwargs: 视图使用的字典类型的参数。

-name: 用来反向获取 URL。

创建项目+APP

Project+APP

一个project可以包含多个app，不同功能的模块放到不同的app

一、创建一个Project

```
lzy@embsky:/home/zyli/test/python/django$ django-admin startproject FirstProject
```

二、创建一个App

```
lzy@embsky:/home/zyli/test/python/django$ cd FirstProject/
```

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ python3 manage.py
```

startapp FirstApp

三、修改app的views.py

lzy@embsky:/home/zyli/test/python/django/FirstProject\$ vim FirstApp/views.py
修改如下：

```
from django.shortcuts import render
from django.http import HttpResponse
def appFun(request):
    return HttpResponse('开始学习Django的app啦')
```

四、修改Project的urls.py

```
from django.conf.urls import url
from django.contrib import admin
from FirstApp import views
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.appFun),
]
```

五、修改Project的setting.py

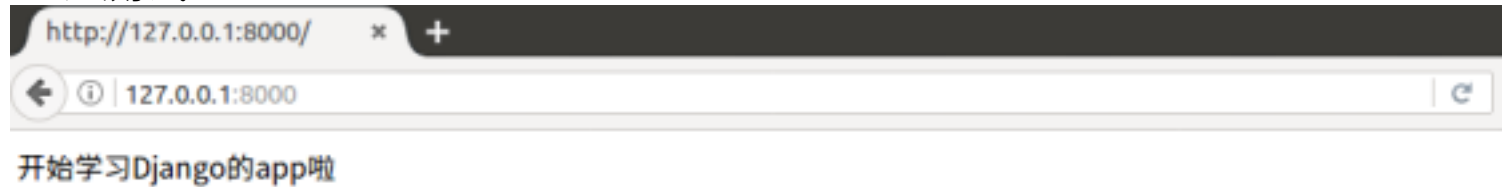
lzy@embsky:/home/zyli/test/python/django/FirstProject\$ vim FirstProject/settings.py

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

六、运行服务器

lzy@embsky:/home/zyli/test/python/django/FirstProject\$ python3 manage.py migrate
lzy@embsky:/home/zyli/test/python/django/FirstProject\$ python3 manage.py runserver

七、测试



八、扩展功能

计算加法

创建一个app calAdd

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ python3 manage.py  
startapp calAdd
```

修改app的views.py

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ vim calAdd/views.py
```

修改如下：

```
from django.shortcuts import render  
from django.http import HttpResponse  
def addFun(request):  
    a = request.GET['a']  
    b = request.GET['b']  
    return HttpResponse(str((int)a + (int)b))
```

修改项目的urls.py

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ vim FirstProject/urls.py
```

修改如下：

```
from django.conf.urls import url  
from django.contrib import admin  
from FirstApp import views  
from calAdd.views import addFun  
urlpatterns = [  
    url(r'^admin/', admin.site.urls),  
    url(r'^$', views.appFun),  
    url(r'^add$', addFun),  
]
```

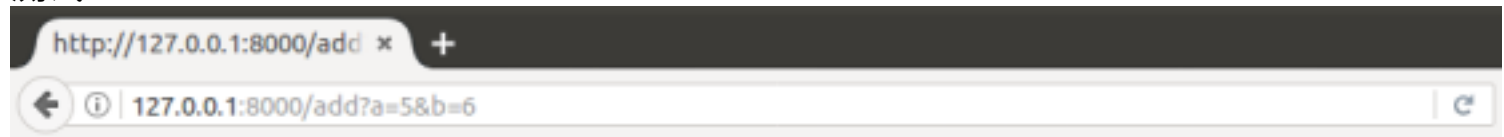
修改项目的setting.py

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ vim FirstProject/  
settings.py
```


修改如下：

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

测试



九、扩展功能

依然是计算加法

修改app的views.py

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ vim calAdd/views.py
```

```
def addFun1(request, a, b) :  
    return HttpResponse(str(int(a) + int(b)))
```

修改项目的urls.py

```
lzy@embsky:/home/zyli/test/python/django/FirstProject$ vim FirstProject/urls.py
```

```
from django.conf.urls import url  
from django.contrib import admin  
from FirstApp import views  
from calAdd.views import addFun  
from calAdd.views import addFun1  
urlpatterns = [  
    url(r'^admin/', admin.site.urls),  
    url(r'^$', views.appFun),  
    url(r'^add$', addFun),  
    url(r'^add/([0-9]+)/([0-9])+$', addFun1),  
]
```

测试



Django模板

模板基础(无APP)

Django 模板

通过模板我们实现了数据和视图的分离

模板应用实例

创建显示模板

```
lzy@embsky:~/HelloWorld$ mkdir templates
```

```
lzy@embsky:~/HelloWorld$ touch templates/hello.html
```

```
lzy@embsky:~/HelloWorld$ vim templates/hello.html
```

输入如下内容：

```
<h1>{{value1}}</h1>
```

```
<b1>{{value2}}</b1>
```

注意： {{}}中表示变量

向Django说明模板文件的路径

```
lzy@embsky:~/HelloWorld/HelloWorld$ vim settings.py
```

修改如下：

```
57     'DIRS': [BASE_DIR + "/templates"],
```

修改 myview.py

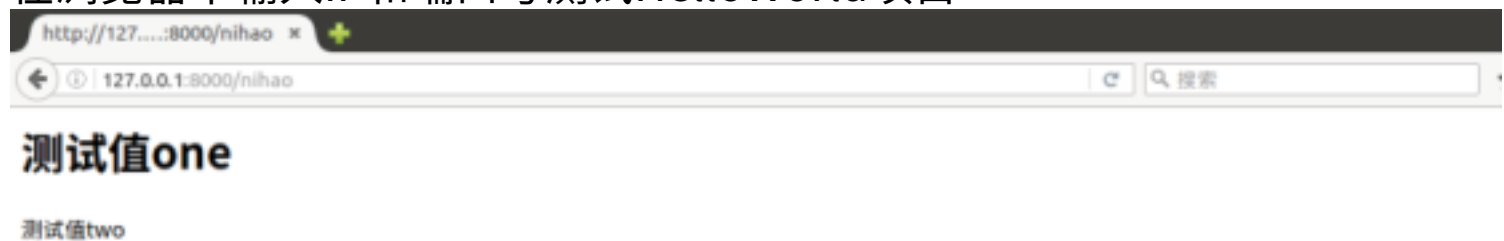
增加一个新的对象，用于向模板提交数据。

```
lzy@embsky:~/HelloWorld/HelloWorld$ vim myview.py
```

修改如下：

```
from django.shortcuts import render
def helloFun(request):
    data = {'value1': '测试值one', 'value2': '测试值two'}
    return render(request, 'hello.html', data)
```

在浏览器中输入IP和端口号测试HelloWorld项目



模板基础(有APP)

Django 模板

通过模板我们实现了数据和视图的分离

一、创建一个项目mbProject和一个app one

```
lzy@embsky:/home/zyli/test/python/django$ django-admin startproject mbProject
```

```
lzy@embsky:/home/zyli/test/python/django$ cd mbProject/
```

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ python3 manage.py
startapp one
```

二、修改项目的setting.py

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ vim mbProject/settings.py
```

修改如下：

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

三、修改app的views.py

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ vim one/views.py
```

```
from django.shortcuts import render
```

```
def index(request):
```

```
    d = {'name':'Django', 'age':10}
```

```
    return render(request, 'index.html', d) #渲染模板，并给模板传递参数
```

四、在app中创建模板

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ mkdir one/templates
```

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ touch one/templates/  
index.html
```

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ vim one/templates/  
index.html
```

输入：

```
<h1>{{name}}</h1>
```

```
<b1>{{age}}</b1>
```

五、修改项目中的urls.py

```
from django.conf.urls import url
```

```
from django.contrib import admin
```

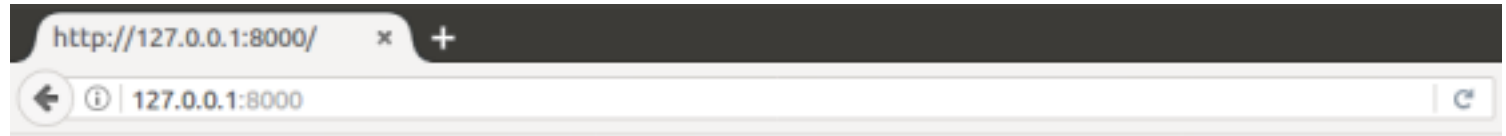
```
from one.views import index
```

```
urlpatterns = [  
    url(r'^admin/', admin.site.urls),
```

```
    url(r'^$', index),  
]
```

六、测试

```
lzy@embsky:/home/zyli/test/python/django/mbProject$ python3 manage.py migrate
lzy@embsky:/home/zyli/test/python/django/mbProject$ python3 manage.py
runserver
```



Django

10

模板标签

Django 模板标签

if/else 标签

基本语法格式如下：

```
{% if condition %}
```

```
...
```

```
{% endif %}
```

或者：

```
{% if condition1 %}
```

```
...
```

```
{% elif condition2 %}
```

```
...
```

```
{% else %}
```

```
...
```

```
{% endif %}
```

if/else 支持嵌套

if/else 标签支持and、or 或者 not 关键字来对多个变量做判断

```
{% if condition1 and condition2 %}
```

```
...
```

```
{% endif %}
```

for 标签

基本语法

```
{% for X in Y %}
```

```
...
```

```
{% endfor %}
```

遍历列表

```
{% for e in listValue %}
```

```
  {{ e }}
```

```
{% endfor %}
```

反向遍历列表

```
{% for e in listValue reversed %}
```

```
...
```

```
{% endfor %}
```

嵌套使用

```
{% for e in listValue %}
```

```
  {{ e.name }}
```

```
  {% for s in e.listValue %}
```

```
    {{ s }}
```

```
  {% endfor %}
```

```
{% endfor %}
```

ifequal/ifnotequal/else

比较user和currentuser是否相等

```
{% ifequal user currentuser %}
```

```
...
```

```
{% ifequal section 'siteneews' %}
```

```
...
```

```
{% else %}
```

```
...
```

```
{% endifequal %}
```

注释标签

Django 注释使用 `{# #}`。

```
{# 这是一个注释 #}
```

include标签

下面这个例子都包含了 `nav.html` 模板：

```
{% include "nav.html" %}
```

过滤器

模板过滤器可以在变量被显示前修改它，过滤器使用管道字符，如下所示：

`{{ name|lower }}` `{{ name }}` 变量被过滤器 `lower` 处理后，文档大写转换文本为小写。

过滤管道可以被套接，既是说，一个过滤器管道的输出又可以作为下一个管道的输入：

`{{ my_list|first|upper }}` 以上实例将第一个元素并将其转化为大写。

有些过滤器有参数。过滤器的参数跟随冒号之后并且总是以双引号包含。例如：

`{{ bio|truncatewords:"30" }}` 这个将显示变量 `bio` 的前30个词。

其他过滤器：

- `addslashes` : 添加反斜杠到任何反斜杠、单引号或者双引号前面。
- `date` : 按指定的格式字符串参数格式化 `date` 或者 `datetime` 对象，实例：`{{ pub_date|date:"F j, Y" }}`
- `length` : 返回变量的长度。

模板标签实例

一.创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ django-admin startproject Template
```

```
lzy@embsky:/home/zyli/test/python/django$ cd Template/
```

```
lzy@embsky:/home/zyli/test/python/django/Template$ python3 manage.py startapp one
```

```
lzy@embsky:/home/zyli/test/python/django/Template$ mkdir one/templates
```

```
lzy@embsky:/home/zyli/test/python/django/Template$ touch one/templates/
```

index.html

二.安装app

lzy@embsky:/home/zyli/test/python/django/Template\$ touch Template/settings.py

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

三.编写视图函数

lzy@embsky:/home/zyli/test/python/django/Template\$ vim one/views.py

```
from django.shortcuts import render  
from django.http import HttpResponse
```

```
def index(request):
```

```
    #字符串数据
```

```
    data = {'value':'This is a string'}
```

```
    #数值型数据
```

```
    data['value1'] = 123
```

```
    #列表型数据
```

```
    data['listValue'] = [1,2,3,4,5, 'test', 'haha']
```

```
    #字典型数据
```

```
    data['dicValue'] = {'a':1, 'b':2, 'c':'123'}
```

```
    return render(request, 'index.html', data)
```

```
def aIndex(request, a, b):
```

```
    return HttpResponse(str(a + b))
```

```
def bIndex(request):
```

```
    a = request.GET['a']
```

```
    b = request.GET['b']
```

```
    return HttpResponse(str(a + b))
```


四.添加路由

```
lzy@embsky:/home/zyli/test/python/django/Template$ touch Template/urls.py
from django.conf.urls import url
from one import views
urlpatterns = [
    path('admin/', admin.site.urls),
    url('^$', views.index),
    #路由的名字起作'a', 以后在模板中可以使用'a'来调用该路由
    url('^a/(\d+)/(\d+)$', views.aIndex, name='a'),
    #路由的名字起作'b', 以后在模板中可以使用'b'来调用该路由
    url('^b$', views.bIndex, name='b')
```

五.编写模板

```
lzy@embsky:/home/zyli/test/python/django/Template$ vim one/templates/
index.html
```

#普通字符串、数值型数据

```
{{value}}
```

```
{{value1}}
```

#列表遍历

```
{% for e in listValue %}
```

```
    {{e}}<br>
```

```
{% endfor %}
```

#字典

```
{{dicValue.a}}<br>
```

```
{{dicValue.b}}<br>
```

```
{{dicValue.c}}<br>
```

#遍历字典

```
{% for key,value in dicValue.items %}
```

```
    {{key}}:{{value}}
```

```
    {% empty %}
```

```
        emptyt emptyt
```

```
{% endfor %}
```

#if/else

```
{% if dicValue.a > 10 %}
```

```
    dicValue.a > 10
```

```
{% elif dicValue.a == 10 %}
```

```
dicValue.a = 10
{% else %}
    dicValue.a < 10
{% endif %}
#获取并使用'a'表示的路由
<a href="{% url 'a' 123 456 %}">link</a>
#获取并使用'b'表示的路由
<a href="{% url 'b' %}?a=10&b=20">link</a>
```

模板继承

模板继承

模板可以用继承的方式来实现复用。

接下来我们先创建之前项目的 templates 目录中添加 base.html 文件，代码如下：

创建基模板

HelloWorld/templates/base.html 文件代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>HelloWorld</title>
</head>
<body>
    This Is Base Html
    {% block page_content %}
    {% endblock %}
</body>
</html>
```

创建模板继承于base.html

HelloWorld/templates/hello.html 文件代码：

```
{% extends 'base.html' %}
{% block page_content %}
<h1>This is from Me!!!</h1>
{% endblock %}
```

静态资源

静态

css、js、font、images等都要放到app/static/下

在html中调用

```
<link href="/static/css/demo.css" rel='stylesheet' type="text/css"/>
```

Django模型

概述

Django 模型

Django 对各种数据库提供了很好的支持，包括：PostgreSQL、MySQL、SQLite、Oracle、Mongodb。

Django 为这些数据库提供了统一的调用API。

ORM&ODM

对象关系映射(Object-Relational Mapper,ORM)。

对象文档映射(Object-Document Mapper,ODM)。

在用户不知觉的情况下把高层的面向对象操作转换成低层的数据库指令。

安装Mysql驱动

```
[root@localhost ~]# pip install mysqlclient
```

在mysql中创建数据库

```
[zyli@localhost Model]$ mysql -uroot -p
```

```
MariaDB [(none)]> create database embsky default character set utf8 collate  
utf8_general_ci;
```

```
Query OK, 1 row affected (0.00 sec)
```

```
MariaDB [(none)]> exit
```

```
Bye
```

数据库访问

一.创建项目和app

```
lzy@embsky:/home/zyli/test/python/django$ django-admin.py startproject  
mxProject
```

```
lzy@embsky:/home/zyli/test/python/django$ cd mxProject/
```

```
lzy@embsky:/home/zyli/test/python/django/mxProject$ python3 manage.py  
startapp embsky
```

```
lzy@embsky:/home/zyli/test/python/django/mxProject$ vim mxProject/settings.py
```

二.设置数据库

修改如下：在这里使用mysql（所以前提是先把mysql配置好）

```
DATABASES = {  
    # 'default': {  
    #     'ENGINE': 'django.db.backends.sqlite3',
```

```
# 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
#}
'default': {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'embsky',    #数据库名字
    'USER': 'root',
    'PASSWORD': 'lzy123',
    'HOST': '127.0.0.1',
    'PORT': '3306',
}
}
```

三.安装app

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

四.创建模型

lzy@embsky:/home/zyli/test/python/django/mxProject\$ vim embsky/models.py

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

#Field官方文档<https://docs.djangoproject.com/en/dev/ref/models/fields/>

```
class Student(models.Model):
```

```
    #成员名字=类型
```

```
    name = models.CharField(max_length=32)
```

```
    age = models.IntegerField()
```

```
    salary = models.FloatField()
```

```
    password = models.CharField(max_length=256)
```

```
def __str__(self):
```

```
    return str(self.name + " " + str(self.age) + " " + str(self.salary));
```

五.更新数据库创建表

```
lzy@embsky:/home/zyli/test/python/django/mxProject$ python3 manage.py  
makemigrations
```

```
lzy@embsky:/home/zyli/test/python/django/mxProject$ python3 manage.py migrate
```

注意：此时在数据库embsky中会创建一个embsky_stduent表。

注意：尽管我们没有在models给表设置主键，但是Django会自动添加一个id作为主键。

六.通过python shell操作数据库

```
lzy@embsky:/home/zyli/test/python/django/mxProject$ python3 manage.py shell
```

```
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
```

```
[GCC 5.4.0 20160609] on linux
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
(InteractiveConsole)
```

```
>>> from embsky.models import Student
```

创建数据一：

```
>>> s = Student(name='zhangsan',age='25')
```

```
>>> s.save()
```

创建数据二：

```
>>> s1 = Student()
```

```
>>> s1.name='wangwu'
```

```
>>> s1.age=45
```

```
>>> s1.save()
```

创建数据三：

```
>>> s2 = Student.objects.create(name='lisi', age=35)
```

创建数据四：如果name和age有一个不匹配那么就创建一个新的

```
>>> l = Student.objects.get_or_create(name='123', age=13)
```

查询数据一：查询所有的数据，返回列表

```
>>> s = Student.objects.all()
```

查询数据二：

```
>>> s = Student.objects.all()[1:3]
```

查询数据三：

```
>>> s = Student.objects.get(name='zhangsan')
```

查询数据四：模糊匹配

```
>>> s = Student.objects.filter(name='zhangsan')
```

```
>>> s = Student.objects.filter(name__iexact='zhangsan') #不区分大小写
```

```
>>> s = Student.objects.filter(name__contains='12')
```

```
>>> s = Student.objects.filter(name__icontains='12') #包含 忽略大小写
```

```
>>> s = Student.objects.filter(name__regex='^[a-z]+$') #正则
```

```
>>> Student.objects.filter(id__lt=3) #小于
```

```
>>> Student.objects.filter(id__gt=3) #大于
```

查询数据五：exclude反查询

```
>>> s = Student.objects.exclude(name__regex='^[a-z]+$')
```

查询数据六：排序

```
>>> s = Student.objects.order_by("name") #升序
```

```
>>> s = Student.objects.order_by("-name") #降序
```

查询数据七：filter、order_by和exclude连用

```
>>> s = Student.objects.filter(name__regex='^[a-z]+$').exclude(name='zhangsan').exclude(name='lisi').filter(name='wangwu')
```

```
>>> s = Student.objects.filter(name__regex='^[a-z]+$').exclude(name='zhangsan').order_by('-name')
```

修改数据一：在查询结果后+update（只能是集合）

```
>>> s = Student.objects.filter(name__regex='^[a-z]+$').exclude(name='zhangsan').order_by('-name').update(name='hao',age=30)
或者：
```

```
>>> s = Student.objects.filter(name__regex='^[a-z]+$').exclude(name='zhangsan').order_by('-name')
```

```
>>> s.update(name='hao',age=30)
```

修改数据二：通过对象修改

```
>>> s = Student.objects.get(name='12')
```

```
>>> s.age=100
```

```
>>> s.save()
```

删除数据一：通过对象删除

```
>>> s = Student.objects.get(name='12')
```

```
>>> s.delete()
```

```
>>> s = Student.objects.filter(id__lt=3)
```

```
>>> s.delete()
```

或者：

```
>>> s = Student.objects.filter(id__lt=3).delete()
```

Django表单

GET

Django 表单

HTML表单是网站交互性的经典方式。

HTTP 请求

HTTP协议以"请求—回复"的方式工作。客户发送请求时，可以在请求中附加数据。服务器通过解析请求，就可以获得客户传来的数据，并根据URL来提供特定的服务。

GET 方法

创建项目gpProject和app getApp

```
lzy@embsky:/home/zyli/test/python/django$ django-admin.py startproject gpProject
```

```
lzy@embsky:/home/zyli/test/python/django$ cd gpProject/
```

```
lzy@embsky:/home/zyli/test/python/django/gpProject$ python3 manage.py startapp  
getApp
```

```
lzy@embsky:/home/zyli/test/python/django/gpProject$ vim gpProject/settings.py
```

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

修改项目的urls.py

```
from django.conf.urls import url  
from django.contrib import admin
```



```
from getApp.views import index
from getApp.views import result
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^index$', index),
    url(r'^result$', result),
]
```

修改app的views.py

lzy@embsky:/home/zyli/test/python/django/gpProject\$ vim getApp/views.py

```
from django.shortcuts import render
# Create your views here.
def index(request):
    data = {'title': 'GET Form Sumit Page'}
    return render(request, 'index.html', data)
```

```
def result(request):
    if request.method == 'GET':
        d = dict()
        d['name'] = request.GET['name']
        d['age'] = request.GET['age']
        return render(request, 'result.html', d)
```

创建模板

lzy@embsky:/home/zyli/test/python/django/gpProject\$ mkdir getApp/templates

lzy@embsky:/home/zyli/test/python/django/gpProject\$ touch getApp/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{title}}</title>
</head>
<body>
    <form action="result" method="get" name="studentInfo">
        <input type="text" name="name"/><br>
        <input type="text" name="age"/><br>
        <input type="submit" name="submit" value="register"/>
```

```
</form>
</body>
</html>
```

lzy@embsky:/home/zyli/test/python/django/gpProject\$ touch getApp/templates/result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  {{name}}<br>
  {{age}}
</body>
</html>
```

POST

POST方法

在上述例子的基础上修改

lzy@embsky:/home/zyli/test/python/django/gpProject\$ vim getApp/templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{{title}}</title>
</head>
<body>
  <form action="result" method="post" name="studentInfo">
    {% csrf_token %}
    <input type="text" name="name"/><br>
    <input type="text" name="age"/><br>
```

```
<input type="submit" name="submit" value="register"/>
</form>
</body>
</html>
```

lzy@embsky:/home/zyli/test/python/django/gpProject\$ vim getApp/views.py

```
from django.shortcuts import render
```

```
# Create your views here.
```

```
def index(request):
```

```
    data = {'title': 'Post Form Submit'}
```

```
    return render(request, 'index.html', data)
```

```
def result(request):
```

```
    if request.method == 'POST':
```

```
        d = dict()
```

```
        d['name'] = request.POST['name']
```

```
        d['age'] = request.POST['age']
```

```
        return render(request, 'result.html', d)
```

数据库

简易学生管理系统

一.创建项目gpProject和app getApp

lzy@embsky:/home/zyli/test/python/django\$ django-admin.py startproject

StudentManager

lzy@embsky:/home/zyli/test/python/django\$ cd StudentManager/

lzy@embsky:/home/zyli/test/python/django/StudentManager\$ python3 manage.py
startapp student

lzy@embsky:/home/zyli/test/python/django/StudentManager\$ vim StudentManager/
settings.py

```
#安装app
```

```
INSTALLED_APPS = [
```

```
    'django.contrib.admin',
```

```
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'student',
]
#配置数据库
DATABASES = {
    'default':{
        'ENGINE':'django.db.backends.mysql',
        'NAME':'embsky',
        'USER':'root',
        'PASSWORD':'lzy123',
        'HOST':'127.0.0.1',
        'PORT':'3306'
    }
}
```

二.更新数据库

```
lzy@embsky:/home/zyli/test/python/django/StudentManager$ python manage.py
makemigrations
lzy@embsky:/home/zyli/test/python/django/StudentManager$ python manage.py
migrate
```

三.定义设备模型

```
lzy@embsky:/home/zyli/test/python/django/StudentManager$ vim student/views.py
from django.db import models
class Student(models.Model) :
    name = models.TextField(max_length=64)
    age = models.IntegerField()
```

四.编写url

```
lzy@embsky:/home/zyli/test/python/django/StudentManager$ vim StudentManager/
urls.py
from student import views
from django.conf.urls import url
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    #首页
    url('^$', views.index),
    #注册路由
    url('^register$', views.register),
    #显示路由
    url('^show$', views.show, name='show')
]
```

五. 编写路由函数

lzy@embsky:/home/zyli/test/python/django/StudentManager\$ vim student/views.py

```
from django.shortcuts import render
from django.shortcuts import redirect
# Create your views here.
def index(request):
    data = {
        'title': 'StudentManager',
        'name': 'name',
        'age': 'age',
    }
    return render(request, 'register.html', data)

from django.http import HttpResponseRedirect
from student.models import Student
def register(request):
    if request.method == "POST":
        name = request.POST['name']
        age = request.POST['age']
        student = Student(name=name, age=age)
        student.save()
        return HttpResponseRedirect('/')

def show(request):
    data = {
        'students': [],
    }
    for student in Student.objects.all():
```

```
s = dict()
s['name'] = student.name
s['age'] = student.age
data['students'].append(s)
return render(request, 'show.html', data)
```

六.编写模板

lzy@embsky:/home/zyli/test/python/django/StudentManager\$ vim student/templates/register.html

```
<form action="register" name="register" method="post">
    {% csrf_token %}
    <input name="name" type="text"/>
    <input name="age" type="text"/>
    <input name="submit" type="submit" value="register"/>
</form>
<a href="{% url 'show' %}">Show Student</a>
```

lzy@embsky:/home/zyli/test/python/django/StudentManager\$ vim student/templates/show.html

```
{% for student in students%}
    {% for key,value in student.items %}
        {{key}}:{{value}}<br>
    {% endfor %}
{% endfor %}
```

扩展

Request 对象

Request对象属性

属性	描述
path	请求页面的全路径,不包括域名—例如, "/hello/"。
method	请求中使用的HTTP方法的字符串表示。全大写表示。例如: <pre>if request.method == 'GET': do_something() elif request.method == 'POST': do_something_else()</pre>
GET	包含所有HTTP GET参数的类字典对象。参见QueryDict 文档。
POST	包含所有HTTP POST参数的类字典对象。参见QueryDict 文档。 服务器收到空的POST请求的情况也是有可能发生的。也就是说, 表单form通过HT 数据。因此, 不能使用语句if request.POST来判断是否使用HTTP POST方法; 应该 的method属性)。 注意: POST不包括file-upload信息。参见FILES属性。
REQUEST	为了方便, 该属性是POST和GET属性的集合体, 但是有特殊性, 先查找POST属性。 \$_REQUEST。 例如, 如果GET = {"name": "john"} 和POST = {"age": '34'},则 REQUEST["name"] 强烈建议使用GET and POST,因为这两个属性更加显式化, 写出的代码也更易理解。
COOKIES	包含所有cookies的标准Python字典对象。Keys和values都是字符串。
FILES	包含所有上传文件的类字典对象。FILES中的每个Key都是<input type="file" name= value 同时也是一个标准Python字典对象, 包含下面三个Keys: · filename: 上传文件名,用Python字符串表示 · content-type: 上传文件的Content type · content: 上传文件的原始内容 注意: 只有在请求方法是POST, 并且请求页面中<form>有enctype="multipart/form-data" FILES 是一个空字典。
点击这里	点击这里

META	<p>包含所有可用HTTP头部信息的字典。 例如:</p> <ul style="list-style-type: none"> • CONTENT_LENGTH • CONTENT_TYPE • QUERY_STRING: 未解析的原始查询字符串 • REMOTE_ADDR: 客户端IP地址 • REMOTE_HOST: 客户端主机名 • SERVER_NAME: 服务器主机名 • SERVER_PORT: 服务器端口 <p>META 中这些头加上前缀HTTP_最为Key, 例如:</p> <ul style="list-style-type: none"> • HTTP_ACCEPT_ENCODING • HTTP_ACCEPT_LANGUAGE • HTTP_HOST: 客户发送的HTTP主机头信息 • HTTP_REFERER: referring页 • HTTP_USER_AGENT: 客户端的user-agent字符串 • HTTP_X_BENDER: X-Bender头信息
user	<p>是一个django.contrib.auth.models.User 对象, 代表当前登录的用户。 如果访问用户当前没有登录, user将被初始化为django.contrib.auth.models AnonymousUser。 你可以通过user的is_authenticated()方法来辨别用户是否登录:</p> <pre>if request.user.is_authenticated(): # Do something for logged-in users. ...</pre> <p>激活Django中的AuthenticationMiddleware时该属性才可用</p>
session	唯一可读写的属性, 代表当前会话的字典对象。只有激活Django中的session middleware时该属性才可用。
raw_post_data	原始HTTP POST数据, 未解析过。高级处理时会有用处。
点击这里	点击这里

Request对象方法

方法	描述
__getitem__(key)	返回GET/POST的键值,先取POST,后取GET。如果键不存在抛出 KeyError。这是我们可以使用字典语法访问HttpRequest对象。例如,request["foo"]等同于先request.POST["foo"] 然后 request.GET["foo"]
has_key()	检查request.GET or request.POST中是否包含参数指定的Key。
get_full_path()	返回包含查询字符串的请求路径。例如, "/music/bands/the_beatles/?p=1"
is_secure()	如果请求是安全的, 返回True, 就是说, 发出的是HTTPS请求。

QueryDict对象

在HttpRequest对象中, GET和POST属性是django.http.QueryDict类的实例。

QueryDict类似字典的自定义类, 用来处理单键对应多值的情况。

QueryDict实现所有标准的词典方法。还包括一些特有的方法:

方法	描述
<code>__getitem__</code>	和标准字典的处理有一点不同, 就是, 如果Key对应多个Value, <code>__getitem__()</code>
<code>__setitem__</code>	设置参数指定key的value列表(一个Python list)。注意: 它只能在一个mutable个QueryDict对象的拷贝).
<code>get()</code>	如果key对应多个value, <code>get()</code> 返回最后一个value。
<code>update()</code>	参数可以是QueryDict, 也可以是标准字典。和标准字典的update方法不同, 该 <pre>>>> q = QueryDict('a=1') >>> q = q.copy() # to make it mutable >>> q.update({'a': '2'}) >>> q.getlist('a') ['1', '2'] >>> q['a'] # returns the last ['2']</pre>
<code>items()</code>	和标准字典的items()方法有一点不同,该方法使用单值逻辑的 <code>__getitem__()</code> : <pre>>>> q = QueryDict('a=1&a=2&a=3') >>> q.items() [('a', '3')]</pre>
<code>values()</code>	和标准字典的values()方法有一点不同,该方法使用单值逻辑的 <code>__getitem__()</code> :

此外, QueryDict也有一些方法, 如下表:

方法	描述
<code>copy()</code>	返回对象的拷贝, 内部实现是用Python标准库的 <code>copy.deepcopy()</code> 。该值。
<code>getlist(key)</code>	返回和参数key对应的所有值, 作为一个Python list返回。如果key不存在, 返回空列表。some sort..
<code>setlist(key, list_)</code>	设置key的值为list_ (unlike <code>__setitem__()</code>).
<code>appendlist(key, item)</code>	添加item到和key关联的内部list.
<code>setlistdefault(key, list)</code>	和 <code>setdefault</code> 有一点不同, 它接受list而不是单个value作为参数。
<code>lists()</code>	和 <code>items()</code> 有一点不同, 它会返回key的所有值, 作为一个list, 例如: <pre>>>> q = QueryDict('a=1&a=2&a=3') >>> q.lists() [('a', ['1', '2', '3'])]</pre>
<code>urlencode()</code>	返回一个以查询字符串格式进行格式化后的字符串(e.g., "a=2&b=3&b=3")

DjangoAdmin

基本操作

Django Admin 管理工具

Django 提供了基于 web 的管理工具。

Django 自动管理工具是 django.contrib 的一部分。

注意：以下案例基于前面的StudentManager案例修改（使用的数据库是embsky）
lzy@embsky:/home/zyli/test/python/django\$ cp StudentManager/ Admin -rf

准备工作

在向数据库中添加记录的时候可能有中文不能识别的问题，原因是mysql中的某些表使用的不是uft-8的编码。

运行以下命令进行查看

mysql> show variables like 'character%';

Variable_name	Value
character_set_client	utf8
character_set_connection	utf8
character_set_database	latin1
character_set_filesystem	binary
character_set_results	utf8
character_set_server	latin1
character_set_system	utf8
character_sets_dir	/usr/share/mysql/charsets/

解决办法如下：（ubuntu+mysql5.7）

lzy@embsky:/home/zyli/test/python/django\$ sudo vim /etc/mysql/mysql.conf.d/mysqld.cnf

最后一行添加：

character-set-server=utf8

```
lzy@embsky:/home/zyli/test/python/django$ sudo vim /etc/mysql/conf.d/mysql.cnf
```

最后一行添加：

default-character-set=utf8

删除数据库（重新创建）

```
mysql> drop database embsky;
```

```
mysql> create database embsky;
```

重启数据库

```
lzy@embsky:/home/zyli/test/python/django$ sudo /etc/init.d/mysql restart
```

创建超级用户

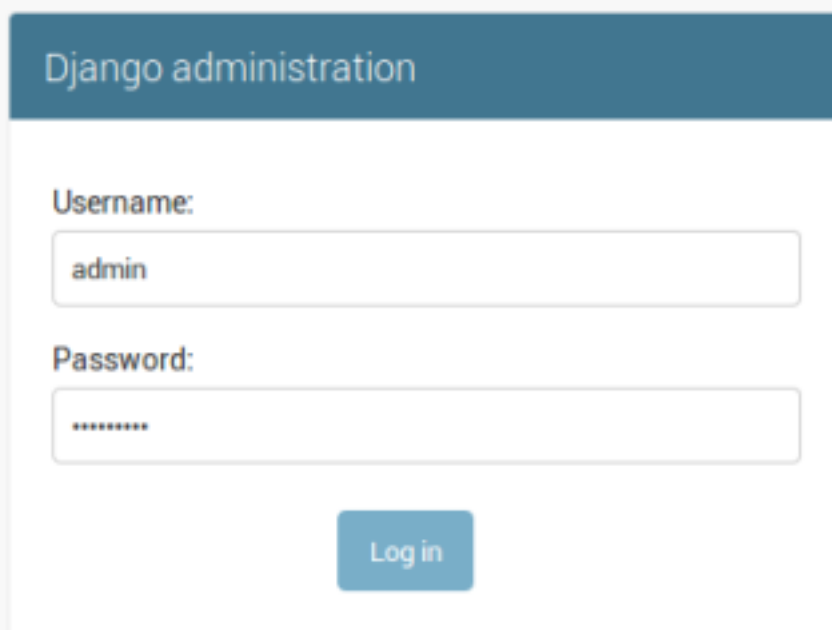
```
lzy@embsky:/home/zyli/test/python/django$ cd Admin
```

```
lzy@embsky:/home/zyli/test/python/django/Admin$ python3 manage.py
```

```
createsuperuser
```

使用管理工具

在浏览器中访问 <http://127.0.0.1:8000/admin/>



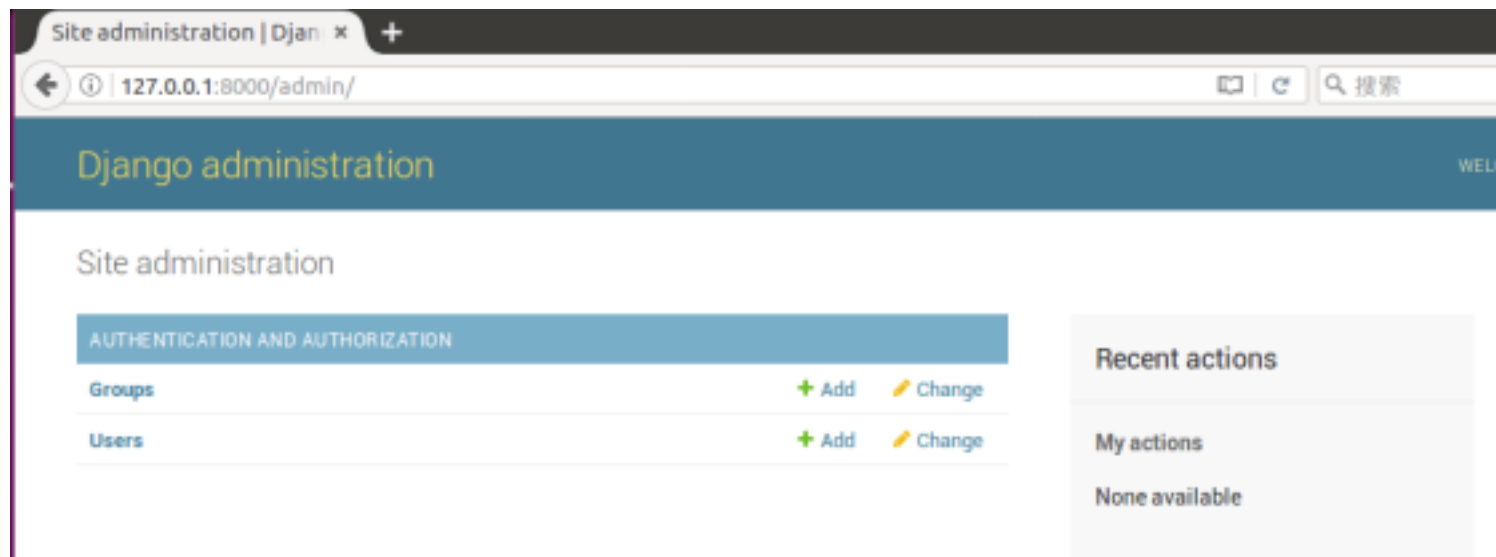
Django administration

Username:

Password:

登录

输入admin和预设的密码后如下：



使用admin管理数据模型

为了让 admin 界面管理某个数据模型，我们需要先注册该数据模型到 admin。

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim student/admin.py

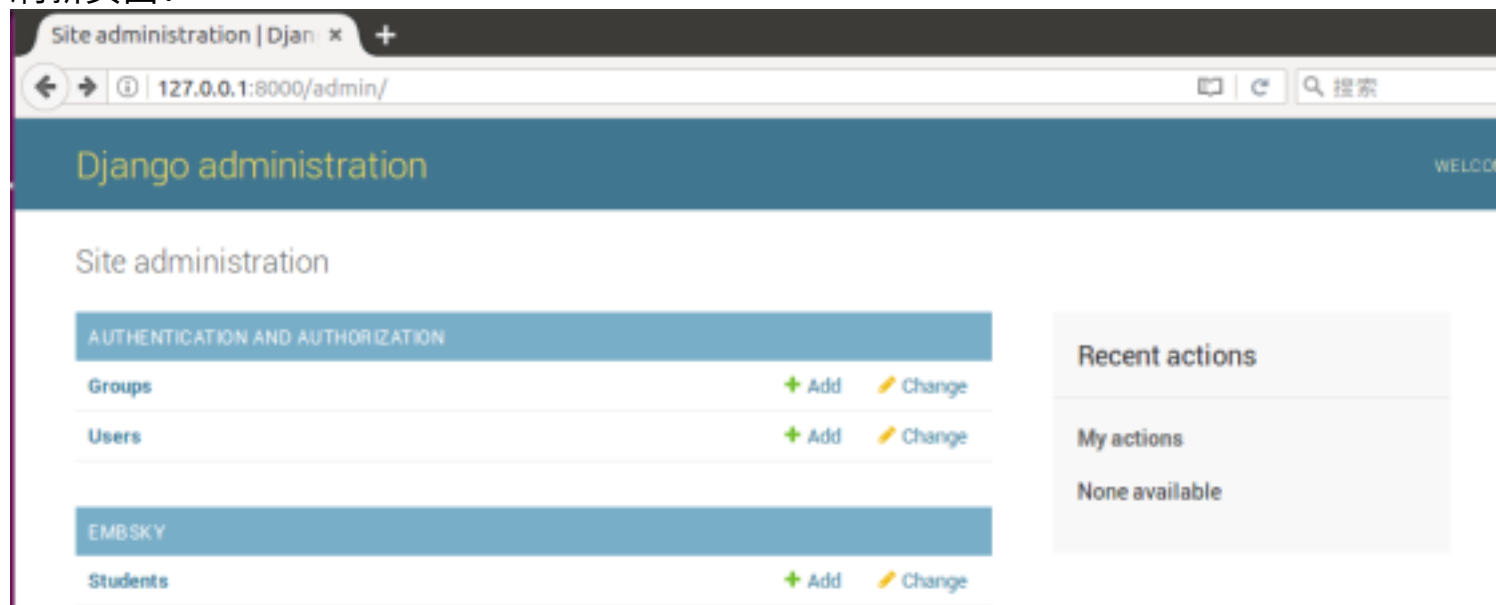
修改如下：

```
from django.contrib import admin
```

```
from embsky.models import Student
```

```
admin.site.register(Student)
```

刷新页面：



注意： 可以点击Students进行查看，点击Add可以添加

数据模型关联

数据模型关联

在很多时候数据模型之间有一些联系，比如外键。

一.在student app中再创建几个模型

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim student/models.py

添加一个Class数据模型如下，在这里我们假设一个学生必须属于一个class：

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

```
class Class(models.Model):
```

```
    name = models.TextField(max_length=64)
```

```
    def __str__(self):
```

```
        return self.name + ' ' + str(self.id)
```

```
class Student(models.Model):
```

```
    name = models.TextField(max_length=64)
```

```
    age = models.IntegerField()
```

```
    #添加外键 models.CASCADE代表删除主表记录的时候要删除相关记录
```

```
    class_id = models.ForeignKey(Class, models.CASCADE)
```

```
    def __str__(self):
```

```
        return self.name + ' ' + str(self.age) + ' ' + str(self.id) + ' at:' + str(self.class_id)
```

注意：什么是外键？ Teacher表中的主键在Salary表中就是外键。

外键的作用是什么？ 当向Student表中插入数据的时候，只有class_id在Class表中的记录存在时候才能正常插入。

当把Class表中的记录删除时，该记录对应的Student表中相关的数据也会被删除。

二.注册数据模型

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim embsky/admin.py

```
from django.contrib import admin
```

```
from .models import Student,Class
```

```
admin.site.register([Student,Class])
```

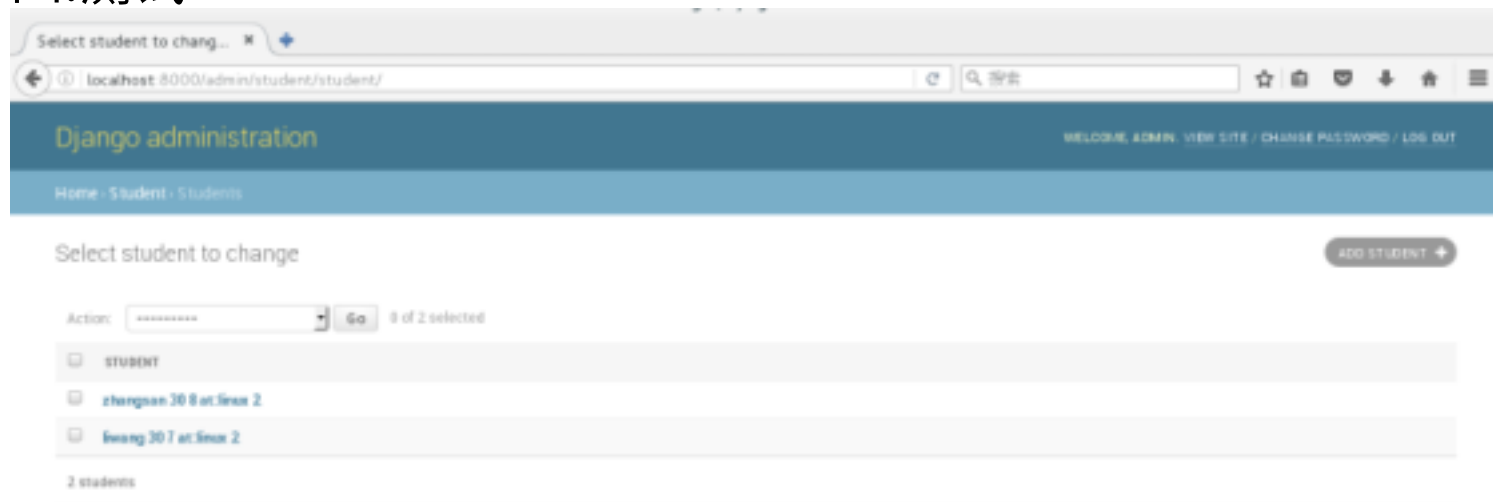
三.同步数据库

lzy@embsky:/home/zyli/test/python/django/Admin\$ python3 manage.py

makemigrations

lzy@embsky:/home/zyli/test/python/django/Admin\$ python3 manage.py migrate

四.测试



定制页面(编辑页面)

定制admin管理页面

Django提供的admin管理页面支持二次开发。

注意：以下案例基于前面的Admin案例修改（使用的数据库是embsky）

```
lzy@embsky:/home/zyli/test/python/django$ cd Admin/
```

自定义模型需要显示的字段一

```
lzy@embsky:/home/zyli/test/python/django/Admin$ vim student/admin.py
```

修改如下：

```
class StudentStyle(admin.ModelAdmin):
```

```
    fields = ('name', 'age')
```

#fields字段用来控制需要显示的字段

```
admin.site.register(Student, StudentStyle)
```

```
admin.site.register(Class)
```

测试：

Change student

HISTORY

Name:

zhangqi

Age:

20

Delete

Save and add another

Save and continue editing

SAVE

自定义模型需要显示的字段二

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim student/admin.py

修改如下：

```
class StudentStyle(admin.ModelAdmin):
```

```
    #fields = ('name','age')
```

```
    fieldsets = (
```

```
        ['must', {
```

```
            'fields':('name','age')
```

```
        }],
```

```
        ['select', {
```

```
            'fields':('class_id',)
```

```
        }]
```

```
    )
```

测试：

must

Name:

liubiao

Age:

20

select

Class id:

embedded 3



自定义模型需要显示的字段三

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim student/admin.py
修改如下：

```
class StudentStyle(admin.ModelAdmin):
```

```
    #fields = ('name','age')
```

```
    fieldsets = (
```

```
        ['must', {
```

```
            'fields':('name','age')
```

```
        }],
```

```
        ['select', {
```

```
            'classes':('collapse',),
```

```
            'fields':('class_id',)
```

```
        }]
```

```
    )
```

测试：

Add student

must

Name:

Age:

select (Show)

定制页面(内联)

定制admin管理页面

Django提供的admin管理页面支持二次开发。

一个Class数据有可能对应多个Student数据。

我们希望看到一个Class数据的时候需要看到这个Class的所有Student数据。

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim student/admin.py

修改如下：

```
from django.contrib import admin
from .models import Student,Class
```

```
class StudentStyle(admin.TabularInline):
```

```
    model = Student
```

```
    #这个表示的仅仅是Student被内联的时候的样式（经测试id不能显示）
```

```
fields = ('age',)
```

```
class ClassStyle(admin.ModelAdmin):  
    fields = ('name',)  
    inlines = [StudentStyle]
```

```
admin.site.register(Student)  
admin.site.register(Class, ClassStyle)
```

测试：

Home › Student › Class › embedded 3

Change class

Name:

STUDENTS

AGE	DELETE?
zhangqi 20 11 at: embedded 3	
<input type="text" value="20"/>	<input type="checkbox"/>

定制页面(列表)

定制admin管理页面

Django提供的admin管理页面支持二次开发。

有的时候我们需要一个显示更为友好的界面。

lzy@embsky:/home/zyli/test/python/django/Admin\$ vim student/admin.py

修改如下：

```
from django.contrib import admin
from .models import Student, Class
```

```
class StudentStyle(admin.TabularInline):
    model = Student
    fields = ('age',)
```

```
class StudentStyle1(admin.ModelAdmin):
    #显示记录时候的样式
    list_display = ('id', 'name', 'age')
```

```
class ClassStyle(admin.ModelAdmin):
    #显示记录时候的样式
    list_display = ('id', 'name')
    #显示详细信息时候的样式
    fields = ('name',)
    inlines = [StudentStyle]
```

```
admin.site.register(Student, StudentStyle1)
admin.site.register(Class, ClassStyle)
```

测试：

student页面

Django administration			WELCOME, ADMIN
Home > Student > Students			
Select student to change			
Action:	<input type="text" value=""/>	Go	0 of 9 selected
<input type="checkbox"/>	ID	NAME	AGE
<input type="checkbox"/>	16	ccg	27
<input type="checkbox"/>	15	bb	10
<input type="checkbox"/>	14	aa	20
<input type="checkbox"/>	13	zhangweo	20
<input type="checkbox"/>	12	liubiao	20
<input type="checkbox"/>	11	zhangqi	20
<input type="checkbox"/>	10	wanger	19

class页面

Select class to change

Action: Go 0 of 2 selected

<input type="checkbox"/>	ID	NAME
<input type="checkbox"/>	3	embedded
<input type="checkbox"/>	2	linux

2 classs

定制页面(搜索)

定制admin管理页面

Django提供的admin管理页面支持二次开发。

有的时候我们需要一个搜索功能。

```
lzy@embsky:/home/zyli/test/python/django/Admin$ vim student/admin.py
```

修改如下：

```
from django.contrib import admin
from .models import Student,Class
```

```
class StudentStyle(admin.TabularInline) :
    model = Student
    fields = ('age',)
```

```
class StudentStyle1(admin.ModelAdmin) :
    list_display = ('id', 'name', 'age')
    search_fields = ('name',)
```

```
class ClassStyle(admin.ModelAdmin) :
    list_display = ('id', 'name')
    fields = ('name',)
```

```
inlines = [StudentStyle]
search_fields = ('name',)
```

```
admin.site.register(Student, StudentStyle1)
```

```
admin.site.register(Class, ClassStyle)
```

测试：

Select student to change

Q	ang	Search	5 results (9 total)
Action:	-----	Go	0 of 5 selected
ID	NAME	AGE	
13	zhangweo	20	
11	zhangqi	20	
10	wanger	19	
8	zhangsan	30	
7	liwang	30	
5 students			

简单美化

安装基于bootstrapped的美化包

```
lzy@embsky:/home/zyli/test/python/django/Admin$ pip install django-admin-bootstrap
```

因为默认安装的是django1.8，所以在这里要重新安装django2.0

```
lzy@embsky:/home/zyli/test/python/django/Admin$ pip uninstall django
```

```
lzy@embsky:/home/zyli/test/python/django/Admin$ pip install django
```

安装APP

```
lzy@embsky:/home/zyli/test/python/django/Admint$ vim Test/Test/setting.py
```

```
INSTALLED_APPS = [  
    'django_admin_bootstrap',  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',
```

```
'django.contrib.sessions',  
'django.contrib.messages',  
'django.contrib.staticfiles',  
'one',  
]
```

Django表单高级

第一个小例子

Django自带的Form框架

创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ mkdir Form  
lzy@embsky:/home/zyli/test/python/django$ cd Form  
lzy@embsky:/home/zyli/test/python/django/Form$ django-admin startproject Test  
lzy@embsky:/home/zyli/test/python/django/Form$ mkdir Test  
lzy@embsky:/home/zyli/test/python/django/Form$ cd Test/  
lzy@embsky:/home/zyli/test/python/django/Form/Test$ django-admin startapp one  
lzy@embsky:/home/zyli/test/python/django/Form/Test$ touch one/forms.py  
lzy@embsky:/home/zyli/test/python/django/Form/Test$ mkdir one/templates  
lzy@embsky:/home/zyli/test/python/django/Form/Test$ touch one/templates/  
index.html
```

注册app

```
lzy@embsky:/home/zyli/test/python/django/Test$ vim Test/setting.py  
INSTALLED_APPS = [  
    'django.contrib.admin',
```

```
'django.contrib.auth',  
'django.contrib.contenttypes',  
'django.contrib.sessions',  
'django.contrib.messages',  
'django.contrib.staticfiles',  
'one'  
]
```

编写表单

lzy@embsky:/home/zyli/test/python/django/Test\$ vim one/forms.py

```
from django import forms  
class AddForm(forms.Form):  
    a = forms.FloatField()  
    b = forms.FloatField()
```

编写视图

lzy@embsky:/home/zyli/test/python/django/Test\$ vim one/views.py

```
from django.shortcuts import render  
from django.shortcuts import HttpResponseRedirect  
# Create your views here.  
from .forms import AddForm  
def index(request):  
    if request.method == 'POST':  
        form = AddForm(request.POST)  
        if form.is_valid():  
            a = form.cleaned_data['a']  
            b = form.cleaned_data['b']  
            return HttpResponseRedirect(str(a + b))  
    else:  
        form = AddForm()  
        return render(request, 'index.html', {'form':form})
```

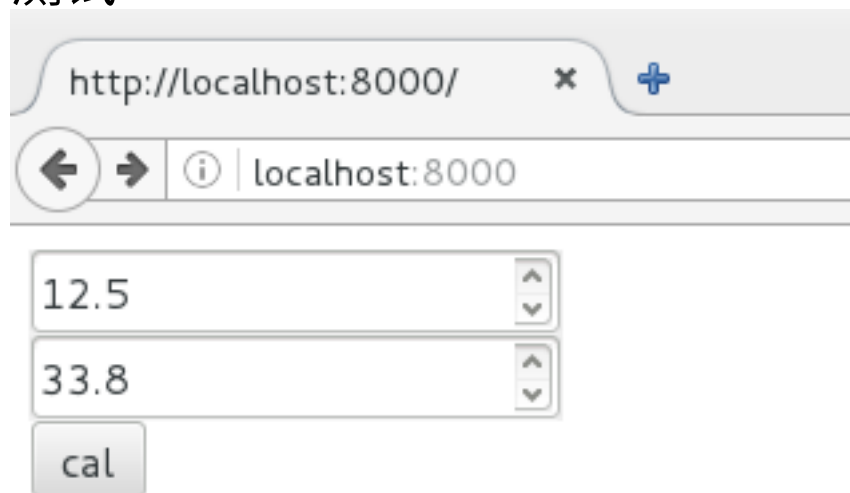
编写模板

lzy@embsky:/home/zyli/test/python/django/Test\$ vim one/templates/index.html

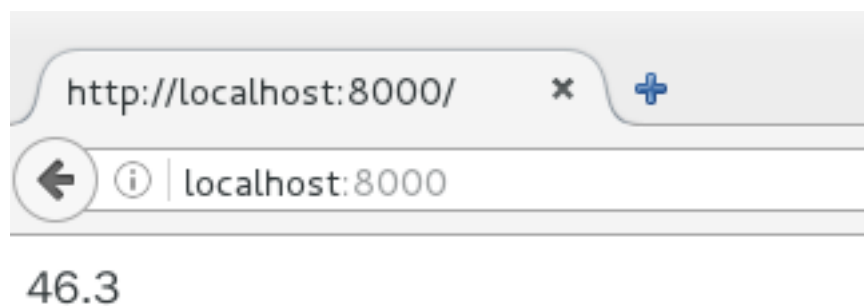
```
<form method="post">  
    {% csrf_token %}  
    {{form.a}}<br>
```

```
{{form.b}}<br>  
<input type="submit" value="cal">  
</form>
```

测试



A screenshot of a web browser window. The address bar shows 'http://localhost:8000/'. The page content includes two input fields with values '12.5' and '33.8', and a submit button labeled 'cal'.



A screenshot of a web browser window. The address bar shows 'http://localhost:8000/'. The page content displays the result '46.3'.

扩展

Django表单扩展知识

在表单的字段中可以添加一些参数，用这些参数来约束表单提交的内容

表单中的字段参数

是否为空

required=True 不可以为空, required=False 可以为空

最大长度

max_length

最小长度

min_length

自定义错误信息, invalid 是格式错误

error_messages={'required': '邮箱不能为空', 'invalid': '邮箱格式错误'}

给自动生成的input标签自定义class属性

widget=forms.TextInput(attrs={'class': 'c1'})

生成Textarea标签

widget=forms.Textarea()

自定义校验方法

自定义的检验方法可以通过validators传送给表单

```
def validate_phone(value):
```

如果不符合条件

```
    raise ValidationError('手机号码格式错误')
```

```
validators=[validate_phone,]
```

*Django*用户管理

概述

Django权限系统auth模块

auth模块是Django提供的标准权限管理系统,可以提供用户身份认证, 用户组和权限管理。

安装app

在INSTALLED_APPS中添加'django.contrib.auth'使用该APP, auth模块默认启用。

User模型

User是auth模块中维护用户信息的关系模式(继承了models.Model), 数据库中该表被命名为auth_user。

```
MariaDB [embsky]> desc auth_user;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
password	varchar(128)	NO		NULL	
last_login	datetime	YES		NULL	
is_superuser	tinyint(1)	NO		NULL	
username	varchar(150)	NO	UNI	NULL	
first_name	varchar(30)	NO		NULL	
last_name	varchar(150)	NO		NULL	
email	varchar(254)	NO		NULL	
is_staff	tinyint(1)	NO		NULL	
is_active	tinyint(1)	NO		NULL	
date_joined	datetime	NO		NULL	

User模型基本操作

注册用户（创建用户）

```
from django.contrib.auth.models import User
user = User.objects.create_user(username, email, password)
user.save()
```

用户认证

```
from django.contrib.auth import authenticate
user = authenticate(username=username, password=password)
```

修改用户密码

```
user.set_password(new_password)
```

登录用户

```
from django.contrib.auth import login  
login(request, user)
```

登出用户

```
from django.contrib.auth import logout  
logout(request)
```

登录用户权限限制

```
from django.contrib.auth.decorators import login_required  
未登录用户将被重定向到login_url指定的位置。  
若未指定login_url参数, 则重定向到settings.LOGIN_URL。  
@login_required(login_url='/accounts/login/')  
def my_view(request):  
    ...
```

模型权限

表(模型)权限限制

Django的auth系统提供了模型级的权限控制, 即可以检查用户是否对某个数据表拥有增(add), 改(change), 删(delete)权限。

检查用户权限

user.has_perm方法用于检查用户是否拥有操作某个模型的权限:

```
user.has_perm('student.add_student')  
user.has_perm('student.change_student')  
user.has_perm('student.delete_student')
```

功能: 判断user用户是否对student app中的student模型(表)有增、改、删的权限

返回值: 如果有权限则返回真, 否则返回假

装饰器

通过装饰器来限制用户对路由函数的访问

```
@permission_required('student.add_student')
def add_student(request):
    pass
```

增加权限

```
user.user_permissions.add(permission)
```

删除权限

```
user.user_permissions.delete(permission)
```

清除所有权限

```
user.user_permissions.clear()
```

自定义权限

```
class Discussion(models.Model):
    ...
    class Meta:
        permissions = (
            ("create_discussion", "Can create a discussion"),
            ("reply_discussion", "Can reply discussion"),
        )
```

判断用户是否拥有自定义权限:

```
user.has_perm('student.create_discussion')
```

当然也可以用装饰器来修饰

用户注册

Django用户注册

在Admin后台可以添加用户

创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ mkdir User
lzy@embsky:/home/zyli/test/python/django$ cd User
```

```
lzy@embsky:/home/zyli/test/python/django/User$ django-admin startproject Test
lzy@embsky:/home/zyli/test/python/django/User$ mkdir Test
lzy@embsky:/home/zyli/test/python/django/User$ cd Test/
lzy@embsky:/home/zyli/test/python/django/User/Test$ django-admin startapp one
lzy@embsky:/home/zyli/test/python/django/User/Test$ touch one/forms.py
lzy@embsky:/home/zyli/test/python/django/User/Test$ mkdir one/templates
lzy@embsky:/home/zyli/test/python/django/User/Test$ touch one/templates/
index.html
```

注册app

```
lzy@embsky:/home/zyli/test/python/django/Form/Test$ vim Test/setting.py
```

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

设置数据库

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'embsky',
        'USER': 'root',
        'PASSWORD': 'lzy123',
        'HOST': '127.0.0.1',
        'PORT': '3306',
    }
}
```

更新数据库

```
lzy@embsky:/home/zyli/test/python/django/User/Test$ python3 manage.py
makemigrations
```

```
lzy@embsky:/home/zyli/test/python/django/User/Test$ python3 manage.py migrate
```

定义表单

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/forms.py

```
from django import forms
```

```
class UserForm(forms.Form) :
```

```
    name = forms.CharField(required=True)
```

```
    email = forms.EmailField(required=True)
```

```
    password = forms.CharField(required=True, widget=forms.PasswordInput,  
min_length=6)
```

```
    password_again = forms.CharField(required=True, widget=forms.PasswordInput,  
min_length=6)
```

```
    about_me = forms.CharField()
```

定义路由函数

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/views.py

```
from django.shortcuts import render
```

```
from one.forms import UserForm
```

```
from django.contrib.auth.models import User
```

```
from django import forms
```

```
from django.shortcuts import HttpResponseRedirect
```

```
def index(request) :
```

```
    if request.method == 'POST' :
```

```
        form = UserForm(request.POST)
```

```
        if form.is_valid() :
```

```
            p = form.cleaned_data['password']
```

```
            p1 = form.cleaned_data['password_again']
```

```
            if p != p1 :
```

```
                return HttpResponseRedirect('password must be fit')
```

```
            try :
```

```
                #username必须是唯一的，如果重复则发生异常
```

```
                user = User.objects.create_user(username=form.cleaned_data['name'],\  
                                                password=form.cleaned_data['password'],\  
                                                email=form.cleaned_data['email'])
```

```
                user.save()
```

```
            except :
```

```
                return HttpResponseRedirect('username must be unique')
```

```
            return HttpResponseRedirect('Register OK')
```

else :

form = UserForm()

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

定义模板

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/templates/index.html

```
<form method="post" name="register">
    {% csrf_token %}
    {{form.email.label}}<br>{{form.email}}<br>
    {{form.name.label}}<br>{{form.name}}<br>
    {{form.password.label}}<br>{{form.password}}<br>
    {{form.password_again.label}}<br>{{form.password_again}}<br>
    {{form.about_me.label}}<br>{{form.about_me}}<br>
    <input type="submit" value="Register">
</form>
```

添加路由

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim Test/urls.py

from one.views **import** index

from django.conf.urls **import** url

urlpatterns = [

path('admin/', admin.site.urls),

url('^\$', index)

]

测试

http://localhost:8000/ x +

localhost:8000 搜索 ☆

Email
da@12.com

Name
da

Password
●●●●●●●●

Password again
●●●●●●●●

About me
1111b

Register

用户登录

定义表单

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/forms.py

添加如下：

```
class LoginForm(forms.Form) :  
    name = forms.CharField(required=True)  
    password = forms.CharField(required=True, widget=forms.PasswordInput)
```

定义路由函数

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/views.py

```
from one.forms import LoginForm  
from django.contrib.auth import authenticate  
from django.contrib import auth  
def login(request) :  
    if request.method == 'POST' :  
        form = LoginForm(request.POST)  
        if form.is_valid() :  
            name = form.cleaned_data['name']
```



```

password = form.cleaned_data['password']
user = authenticate(username=name,password=password)
if user :
    auth.login(request, user=user)
    return render(request, 'main.html')
else :
    form = LoginForm()
    return render(request, 'login.html', {'form':form})

def logout(request) :
    auth.logout(request)
    return render(request, 'main.html')

def main(request) :
    return render(request, 'main.html')

```

构建模板

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/templates/login.html

```

<form method="post" name="register">
    {% csrf_token %}
    {{form.name.label}}<br>{{form.name}}<br>
    {{form.password.label}}<br>{{form.password}}<br>
    <input type="submit" value="Login">
</form>

```

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/templates/main.html

```

{% if request.user.is_authenticated %}
Now is login status
<a href="{%url 'logout'%}" class="button">Logout</a>
{% else %}
Now is logout status
<a href="{%url 'login'%}" class="button">Login</a>
{% endif %}

```

添加路由

```
lzy@embsky:/home/zyli/test/python/django/User/Test$ vim Test/urls.py
from one.views import index,login, logout, main
from django.conf.urls import url
urlpatterns = [
    path('admin/', admin.site.urls),
    url('^register$', index),
    url('^login/$', login, name='login'),
    url('^logout/$', logout, name='logout'),
    url('^$', main)
]
```

用户限制

定义路由函数

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/views.py
使用装饰器来限定只有登录用户才能访问该路由

```
from django.contrib.auth.decorators import login_required
#如果用户没有登录则重定向到login路由
@login_required(login_url='login')
def test(request):
    return HttpResponse('test for user')
```

添加路由

```
from one.views import index,login, logout, main, test
from django.conf.urls import url
urlpatterns = [
    path('admin/', admin.site.urls),
    url('^register$', index),
    url('^login/$', login, name='login'),
    url('^logout/$', logout, name='logout'),
    url('^$', main),
    url('^test/$', test)
]
```

用户扩展

用户资源扩展

Django自带的User模型并不能满足所有需求。

定义一个描述用户信息的模型

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/models.py

```
from django.db import models
from django.contrib.auth.models import User
class MyUserInfo(models.Model):
    age = models.IntegerField()
    about_me = models.CharField(max_length=256)
    user_id = models.ForeignKey(User, models.CASCADE)
```

更新数据库

lzy@embsky:/home/zyli/test/python/django/User/Test\$ python manage.py
makemigrations

lzy@embsky:/home/zyli/test/python/django/User/Test\$ python manage.py migrate

修改注册路由函数

在注册新用户的时候增加用户信息

lzy@embsky:/home/zyli/test/python/django/User/Test\$ vim one/views.py

```
.....
try:
    user = User.objects.create_user(username=form.cleaned_data['name'],\
                                     password=form.cleaned_data['password'], \
                                     email=form.cleaned_data['email'])
    user.save()
    user_info = MyUserInfo()
    user_info.user_id_id = user.id
    user_info.age = 20
    user_info.about_me = 'I am a teacher'
```

```
user_info.save()
except :
    return HttpResponse('username must be unique')
return HttpResponse('Register OK')
.....
```

Django邮件

单封邮件发送

Django邮件

创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ mkdir Email
lzy@embsky:/home/zyli/test/python/django$ cd Email
lzy@embsky:/home/zyli/test/python/django/Email$ django-admin startproject Test
lzy@embsky:/home/zyli/test/python/django/Email$ cd Test/
lzy@embsky:/home/zyli/test/python/django/Email/Test$ django-admin startapp one
```

注册app

```
lzy@embsky:/home/zyli/test/python/django/Email/Test$ vim Test/setting.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
'one'  
]
```

增加邮件变量

```
lzy@embsky:/home/zyli/test/python/django/Email/Test$ vim Test/setting.py  
EMAIL_USE_SSL = True  
EMAIL_HOST = 'smtp.163.com' # 如果是 163 改成 smtp.163.com  
EMAIL_PORT = 465  
EMAIL_HOST_USER = 'lizhiyong_beyond@163.com' # 帐号  
EMAIL_HOST_PASSWORD = 'xxxx' # 密码  
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

编写路由函数

```
lzy@embsky:/home/zyli/test/python/django/Email/Test$ vim one/views.py  
from django.shortcuts import render  
from django.core.mail import send_mail  
from django.shortcuts import HttpResponseRedirect  
# Create your views here.  
def sendmail(request):  
    send_mail('test', 'hahahaha', 'lizhiyong_beyond@163.com',  
['bjzhangwei@uplooking.com', 'lizhiyong_beyond@163.com'], fail_silently=False)  
    return HttpResponseRedirect('send over)
```

添加url

```
lzy@embsky:/home/zyli/test/python/django/Email/Test$ vim Test/urls.py  
from django.conf.urls import url  
from one import views  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    url('^send_mail$', views.sendmail),  
]
```

一次发送多封邮件

一次发送多个邮件

send_mail函数每发送一次邮件都要建立一条tcp链接

send_mass_mail函数建立一条tcp链接发送多次邮件

编写路由函数

lzy@embsky:/home/zyli/test/python/django/Email/Test\$ vim one/views.py

```
def sendmutimail(request):  
    m1 = ('subject1', 'content:heheheh', 'lizhiyong_beyond@163.com',  
['lizhiyong@uplooking.com'])  
    m2 = ('subject2', 'content:hahahah', 'lizhiyong_beyond@163.com',  
['lizhiyong@uplooking.com'])  
    send_mass_mail((m1, m2), fail_silently=False)
```

添加url

lzy@embsky:/home/zyli/test/python/django/Email/Test\$ vim Test/urls.py

```
from django.contrib import admin  
from django.urls import path  
from django.conf.urls import url  
from one import views  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    url('^send_mail$', views.sendmail),  
    url('^send_muti_mail$', views.sendmutimail)  
]
```

发送html邮件

发送邮件的让内容更加好看丰富

发送html文档

编写路由函数

lzy@embsky:/home/zyli/test/python/django/Email/Test\$ vim one/views.py

```

from django.core.mail import EmailMultiAlternatives
def sendaltermail(request):
    content = '''
        <h1>你好！ 这里是战狼中队</h1>
        <br>
        <h2>欢迎来到这里</h2>
    '''
    msg = EmailMultiAlternatives('subject2', content, 'lizhiyong_beyond@163.com',
['lizhiyong@uplooking.com'])
    msg.content_subtype = 'html'
    msg.send(fail_silently=False)
    return HttpResponse('ok')

```

添加url

lzy@embsky:/home/zyli/test/python/django/Email/Test\$ vim Test/urls.py

```

urlpatterns = [
    path('admin/', admin.site.urls),
    url('^send_mail$', views.sendmail),
    url('^send_muti_mail$', views.sendmutimail),
    url('^send_alter$', views.sendaltermail)
]

```

文本和html邮件

邮件客户端不支持html文档怎么办？

我们可以做两手准备，如果客户能识别html则用html，否则使用存文本

编写路由函数

lzy@embsky:/home/zyli/test/python/django/Email/Test\$ vim one/views.py

```

def sendtexthtml(request):
    content = '''
        <h1>你好！ 这里是战狼中队</h1>
        <br>
        <h2>欢迎来到这里</h2>
    '''

```

'''

```
content_text = 'wuwuwuwuwuwu'
msg = EmailMultiAlternatives('subject3', content_text,
'lizhiyong_beyond@163.com', ['lizhiyong@uplooking.com'])
msg.attach_alternative(content, 'text/html')
msg.send(fail_silently=False)
return HttpResponse('send text/html ok')
```

添加url

lzy@embsky:/home/zyli/test/python/django/Email/Test\$ vim Test/urls.py

```
urlpatterns = [
    path('admin/', admin.site.urls),
    url('^send_mail$', views.sendmail),
    url('^send_muti_mail$', views.sendmutimail),
    url('^send_alter$', views.sendaltermail),
    url('^send_text_html$', views.sendtexthtml)
]
```

Django缓存

缓存分类一

Django缓存

Django提供了一个稳定的缓存系统让你缓存动态页面的结果。

这样在接下来有相同的请求就可以直接使用缓存中的数据，避免不必要的重复计算。

Django还提供了不同粒度数据的缓存，例如：你可以缓存整个页面，也可以缓存某个部分，甚至缓存整个网站。

注意：不包含GET或POST参数的页面在第一次被请求之后将被缓存指定好的一段时间。

Django缓存类型

Django提供了6中缓存类型。

开发调试缓存

这种缓存其实在内部并没有实现，只用于开发调试

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.dummy.DummyCache', # 引擎  
        'TIMEOUT': 300, # 缓存超时时间（默认300，None表示永  
        不过期，0表示立即过期）  
        'OPTIONS': {  
            'MAX_ENTRIES': 300, # 最大缓存个数（默认300）  
            'CULL_FREQUENCY': 3, # 缓存到达最大个数之后，剔除缓存个数  
            的比例，即：1/CULL_FREQUENCY（默认3）  
        },  
        'KEY_PREFIX': '', # 缓存key的前缀（默认空）  
        'VERSION': 1, # 缓存key的版本（默认1）  
        'KEY_FUNCTION': 函数名 # 生成key的函数（默认函数会生成：  
        【前缀:版本:key】）  
    }  
}
```

本地内存缓存

把缓存内容放到内存

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',  
        #给缓存放置的内存区设置一个名字  
        'LOCATION': 'unique-snowflake',  
    }  
}
```

注意：其他配置与开发调试cache配置一样

文件缓存

把缓存内容记录在文件

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',  
        #配置缓存存放的目录  
        'LOCATION': '/var/tmp/django_cache',  
    }  
}
```

注意：其他配置与开发调试cache配置一样

数据库缓存

把缓存内容放到数据库

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',  
        'LOCATION': 'my_cache_table', # 数据库表  
    }  
}
```

注意：需要python manage.py createcachetable my_cache_table创建表

注意：其他配置与开发调试cache配置一样

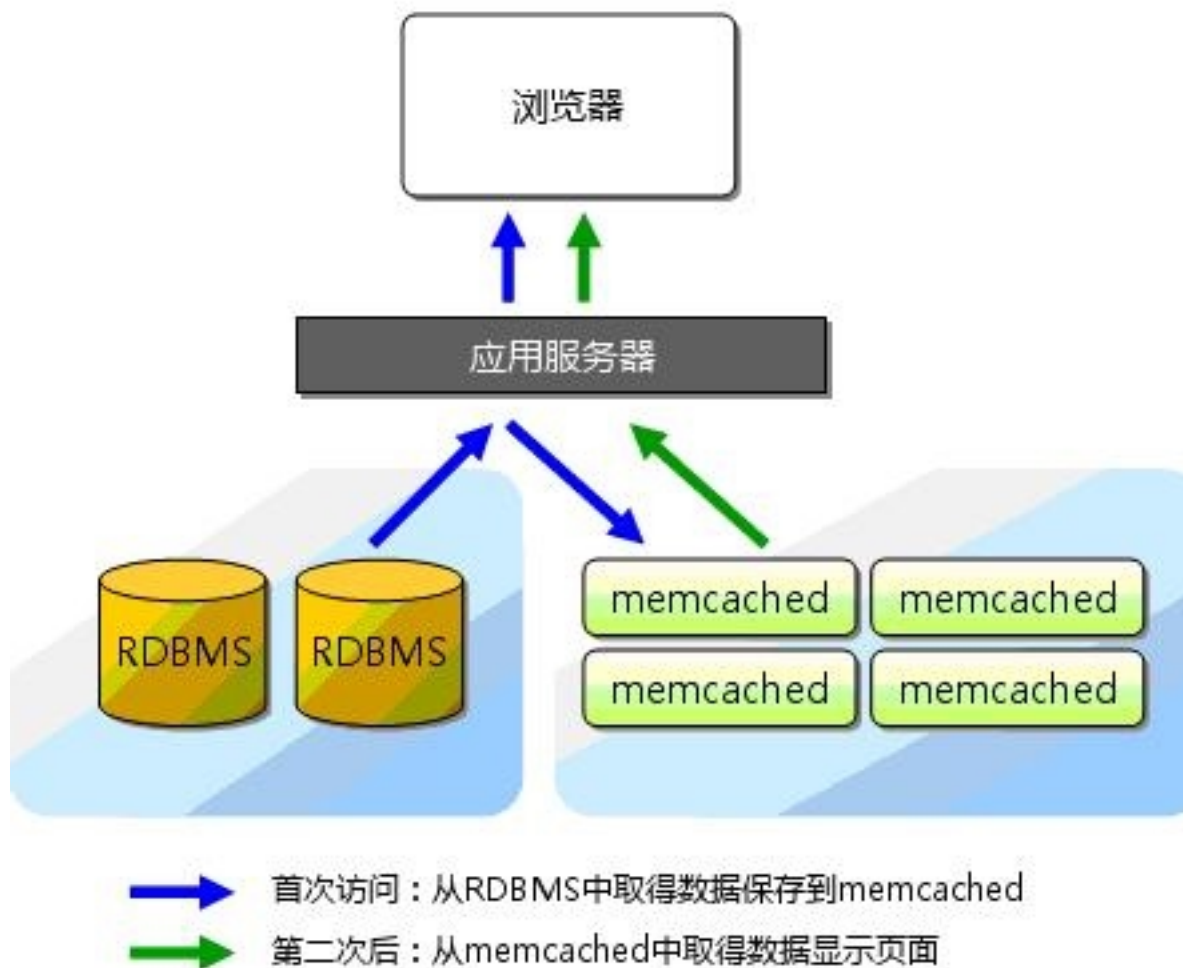
缓存分类二

Memcache缓存（python-memcached模块）

Memcache缓存是什么？

Memcached是一个自由开源的，高性能，分布式内存对象缓存系统。

Memcached是一种基于内存的key-value存储，用来存储小块的任意数据（字符串、对象）。



Linux Memcached 安装

Ubuntu/Debian: `sudo apt-get install memcached`

Redhat/Fedora/Centos: `yum install memcached`

Linux Memcached运行

```
lzy@embsky:/home/zyli/test/python/django/Cache/Test$ memcached -m 128m -p 1234 -vv
```

启动选项：

-d是启动一个守护进程

-h是查看帮助文档

-m是分配给Memcache使用的内存数量，单位是MB

-u是运行Memcache的用户

-l是监听的服务器IP地址，可以有多个地址

-p是设置Memcache监听的端口，，最好是1024以上的端口

-c是最大运行的并发连接数，默认是1024

-P是设置保存Memcache的pid文件

安装memcached模块

用于让Django使用memcached模块连接memcached

```
pip install python-memcached
```

编写cache配置文件

方法一：

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',  
        'LOCATION': '127.0.0.1:1234',  
    }  
}
```

方法二：

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',  
        'LOCATION': 'unix:/tmp/memcached.sock',  
    }  
}
```

方法三：

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',  
        'LOCATION': [  
            '172.19.26.240:1234',  
            '172.19.26.242:1234',  
        ],  
    }  
}
```

注意：其他配置与开发调试cache配置一样

安装pylibmc模块

用于让Django使用pylibmc模块连接memcached

```
yum install libmemcached-devel
```

```
pip install pylibmc
```

编写cache配置文件

方法一：

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.memcached.PyLibMCCache',  
        'LOCATION': '127.0.0.1:1234',  
    }  
}
```

方法二：（有问题）

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.memcached.PyLibMCCache',  
        'LOCATION': '/tmp/memcached.sock',  
    }  
}
```

方法三：

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.memcached.PyLibMCCache',  
        'LOCATION': [  
            '172.19.26.240:1234',  
            '172.19.26.242:1234',  
        ],  
    }  
}
```

注意：其他配置与开发调试cache配置一样

局部缓存

局部缓存

只缓存部分页面

创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ mkdir Cache
```

```
lzy@embsky:/home/zyli/test/python/django$ cd Cache
```

```
lzy@embsky:/home/zyli/test/python/django/Cache$ django-admin startproject Test
```

```
lzy@embsky:/home/zyli/test/python/django/Cache$ cd Test/
```

```
lzy@embsky:/home/zyli/test/python/django/Cache/Test$ django-admin startapp one
```

注册app

```
lzy@embsky:/home/zyli/test/python/django/Cache/Test$ vim Test/setting.py
```

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

添加缓存机制

```
lzy@embsky:/home/zyli/test/python/django/Cache/Test$ vim Test/setting.py
```

```
CACHES = {  
    'default': {  
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache', # 引擎  
        'LOCATION': 'oneCache',  
        'TIMEOUT': 3000, # 缓存超时时间（默认300，None表示永不  
过期，0表示立即过期）  
        'OPTIONS': {  
            'MAX_ENTRIES': 300, # 最大缓存个数（默认300）  
            'CULL_FREQUENCY': 3, # 缓存到达最大个数之后，剔除缓存个数的比  
例，即：1/CULL_FREQUENCY（默认3）  
        },  
    }  
}
```

添加路由函数

```
lzy@embsky:/home/zyli/test/python/django/Cache/Test$ vim one/views.py
```

```
from django.views.decorators.cache import cache_page
```

#为此路由设置缓存，在3000秒内这个路由函数不会被重复访问

```
@cache_page(3000)
```

```
def one(request):
```

```
    print('one')
```

```
    return HttpResponse('one')
```

```
def two(request):
```

```
    print('two')
```

```
    return HttpResponse('two')
```

添加路由

lzy@embsky:/home/zyli/test/python/django/Cache/Test\$ vim Test/urls.py

```
from one import views
```

```
from django.conf.urls import url
```

```
urlpatterns = [
```

```
    path('admin/', admin.site.urls),
```

```
    url('^one$', views.one),
```

```
    url('^two$', (cache_page(300))(views.two))
```

```
]
```

测试：

第一次访问one路由，one路由函数会被调用

在浏览器中继续访问one路由的时候浏览器不会发送请求

在浏览器中ctrl+F5会发送请求，但one路由函数不会被调用

说明：在浏览器和服务端都有缓存

全站缓存

全站缓存

缓存全部页面

增加中间件

lzy@embsky:/home/zyli/test/python/django/Cache/Test\$ vim Test/setting.py

```
MIDDLEWARE = [  
    'django.middleware.cache.UpdateCacheMiddleware',  
    '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',  
    'django.middleware.cache.FetchFromCacheMiddleware',  
]
```

修改路由函数

lzy@embsky:/home/zyli/test/python/django/Cache/Test\$ vim one/views.py

```
# @cache_page(3000)
```

```
def one(request):  
    print('one')  
    return HttpResponse('one')
```

```
def two(request):  
    print('two')  
    return HttpResponse('two')
```

Django会话

基本概念

会话session

浏览器和服务器通信的时候会建立一个会话。

在这个会话中我们可以存储一些数据，用来标记用户。

启用Django的会话功能

lzy@embsky:/home/zyli/test/python/django/Cache/Test\$ vim Test/setting.py

```
MIDDLEWARE = [  
    # 'django.middleware.cache.UpdateCacheMiddleware',  
    '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',  
    # 'django.middleware.cache.FetchFromCacheMiddleware',  
]
```

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

注意：会话功能默认是启动的

使用 session

session 可以在视图中任何有request的地方使用，它类似于python中的字典
添加数据：

```
request.session[key] = value
```

获取数据：

```
request.session.get(key,default=None)
```

删除数据：

```
del request.session[key]
```

注意： session 默认有效时间为两周

注意： 有效时间可以修改 (https://docs.djangoproject.com/en/dev/ref/settings/#std:setting-SESSION_COOKIE_AGE)

简单案例

创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ mkdir Session
lzy@embsky:/home/zyli/test/python/django$ cd Session
lzy@embsky:/home/zyli/test/python/django/Session$ django-admin startproject Test
lzy@embsky:/home/zyli/test/python/django/Session$ cd Test/
lzy@embsky:/home/zyli/test/python/django/Session/Test$ django-admin startapp
one
```

安装APP

```
lzy@embsky:/home/zyli/test/python/django/Session/Test$ vim Test/setting.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'one'
]
```

更新数据库

```
lzy@embsky:/home/zyli/test/python/django/Session/Test$ python manage.py
makemigrations
lzy@embsky:/home/zyli/test/python/django/Session/Test$ python manage.py
```

migrate

编写路由函数

lzy@embsky:/home/zyli/test/python/django/Session/Test\$ vim one/views.py

```
def one(request) :  
    request.session['id'] = 'lizhiyong'  
    return HttpResponse('add ok')  
  
def two(request) :  
    if request.session.get('id') :  
        del request.session['id']  
    return HttpResponse('del ok')  
  
def three(request) :  
    if request.session.get('id') :  
        return HttpResponse(request.session.get('id'))  
    else :  
        return HttpResponse('not found')
```

添加路由

lzy@embsky:/home/zyli/test/python/django/Session/Test\$ vim Test/urls.py

```
urlpatterns = [  
    path('admin/', admin.site.urls),  
    url('^one$', views.one),  
    url('^two$', views.two),  
    url('^three$', views.three)  
]
```

测试

先访问one再访问three，结果：lizhiyong

接下来访问two再访问three，结果：not found

Django二维码

Python二维码

二维码

有时候需要为某个网站链接或者资源做一个二维码提供给用户。
用户可以通过扫描二维码的方式来访问网站或者资源。

安装制作二维码需要的包

```
lzy@embsky:/home/zyli/test/python/django$ pip install Image  
lzy@embsky:/home/zyli/test/python/django$ pip install qrcode
```

使用命令制作二维码

```
lzy@embsky:/home/zyli/test/python/django$ qr 'http://www.embsky.com' >test.png
```

二维码效果



写代码制作二维码

```
lzy@embsky:/home/zyli/test/python/django$ mkdir 18QRCode/  
lzy@embsky:/home/zyli/test/python/django$ cd 18QRCode  
lzy@embsky:/home/zyli/test/python/django/18QRCode$ touch test.py  
import qrcode  
def make_qrcode(value):
```

```
img = qrcode.make(value)
return img
```

```
img = make_qrcode('http://www.embsky.com')
with open('test.png', 'wb') as f:
    img.save(f)
```

***Django*二维码**

在Django中使用二维码

创建一个工程

```
lzy@embsky:/home/zyli/test/python/django$ mkdir QRCode
lzy@embsky:/home/zyli/test/python/django$ cd QRCode
lzy@embsky:/home/zyli/test/python/django/QRCode$ django-admin startproject Test
lzy@embsky:/home/zyli/test/python/django/QRCode$ cd Test/
lzy@embsky:/home/zyli/test/python/django/QRCode/Test$ django-admin startapp
one
```

注册app

```
lzy@embsky:/home/zyli/test/python/django/QRCode/Test$ vim Test/setting.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
'one'  
]
```

添加路由函数

lzy@embsky:/home/zyli/test/python/django/QRCode/Test\$ vim one/views.py

```
from django.shortcuts import render  
import qrcode  
from django.http import HttpResponse  
from django.utils.six import BytesIO
```

```
def make_qrcode(value):  
    img = qrcode.make(value)  
    return img
```

```
def generate_qrcode(request) :  
    value = request.GET['value']  
    img = make_qrcode(value)  
    buf = BytesIO()  
    img.save(buf)  
    return HttpResponse(buf.getvalue(), content_type='image/png')
```

添加路由

lzy@embsky:/home/zyli/test/python/django/Session/Test\$ vim Test/urls.py

```
from django.conf.urls import url  
from one import views  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    url('^qrcode$', views.generate_qrcode)  
]
```

测试



Nginx部署(Centos7)

基于Centos搭建Django+Nginx+uwsgi环境

只有在测试环境中才会使用 `python manage.py runserver` 来运行服务器。
正式发布的服务，我们需要一个可以稳定、持续、支持高并发的服务器，比如apache, Nginx, lighttpd等。
接下来我们要一nginx为例子。

Nginx安装

Nginx

安装Nginx

```
[root@localhost ~]# yum install nginx
```

启动Nginx

```
[root@localhost ~]# systemctl start nginx
```

测试



UWSGI安装

UWSGI安装及测试

安装uwsgi

```
[root@localhost ~]# pip install uwsgi
```

编写测试网站代码

```
[root@localhost ~]# touch test.py
```

```
[root@localhost ~]# vim test.py
```

输入以下内容：

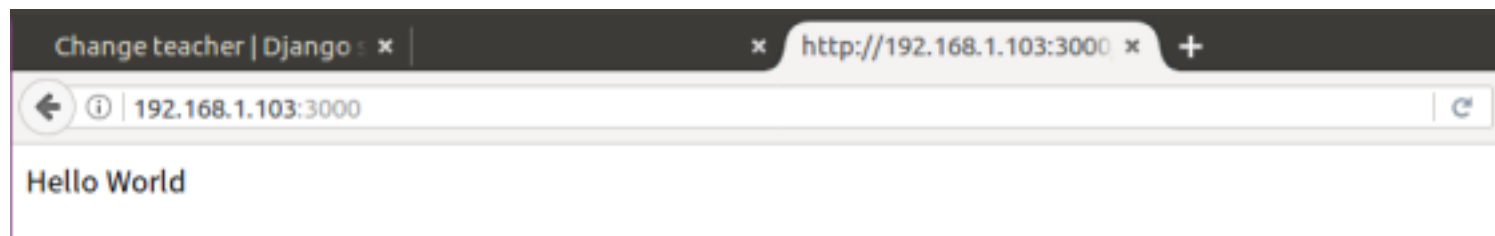
```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b"Hello World"]
```

关闭防火墙

```
[root@localhost ~]# systemctl stop firewalld.service
```

```
[root@localhost ~]# uwsgi --http :3000 --wsgi-file test.py --enable-threads
```

测试



Nginx&Uwsgi

配置UWSGI和Nginx

Nginx负责相应用户请求，Uwsgi负责Nginx和Django之间的通信。

创建一个工程

```
[root@localhost ~]# mkdir Nginx
[root@localhost ~]# cd Nginx
[root@localhost Nginx]# django-admin startproject Test
[root@localhost Nginx]# cd Test/
[root@localhost Test]# django-admin startapp one
```

安装app

```
[root@localhost Test]# vim Test/setting.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'one'
]
```

修改访问IP

```
[root@localhost Test]# vim Test/setting.py
ALLOWED_HOSTS = ['*']
```

关闭SELinux

```
[root@localhost Test]# setenforce 0
```

创建uwsgi配置文件

```
[root@localhost Test]# vim uwsgi.ini
```

```
[uwsgi]
```

#对外提供的端口号，不需要Nginx也可访问

```
http = :9000
```

#与Nginx通信的端口号

```
socket = 0.0.0.0:9090
```

#工程路径

```
chdir = /root/Nginx/Test
```

```
wsgi-file = Test/wsgi.py
```

#启动4个进程

```
processes = 4
```

#每个进程2个线程

```
threads = 2
```

#用于监控服务器负载的地址和端口

```
stats = 127.0.0.1:9191
```

#退出时候清除环境变量

```
vacuum = true
```

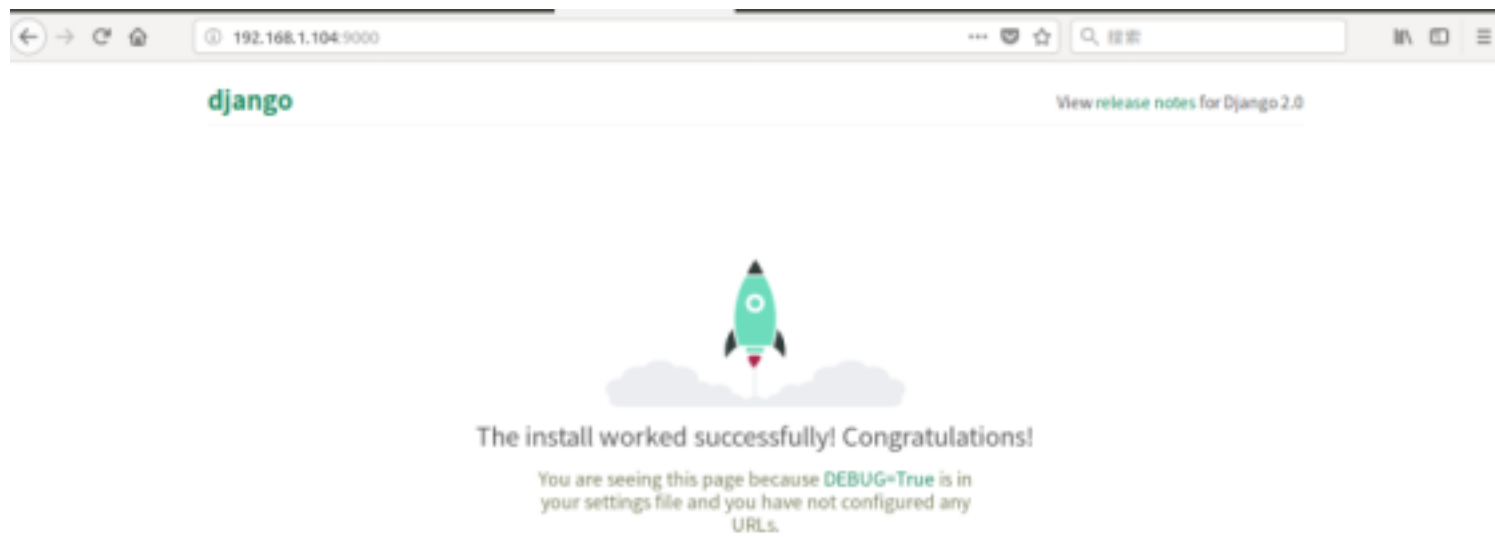
#日志文件

```
daemonize = /var/log/uwsgi.log
```

启动uwsgi

```
[root@localhost Test]# uwsgi uwsgi.ini
```

测试



修改Nginx配置文件

```
[root@localhost Test]# vim /etc/nginx/nginx.conf
```

.....

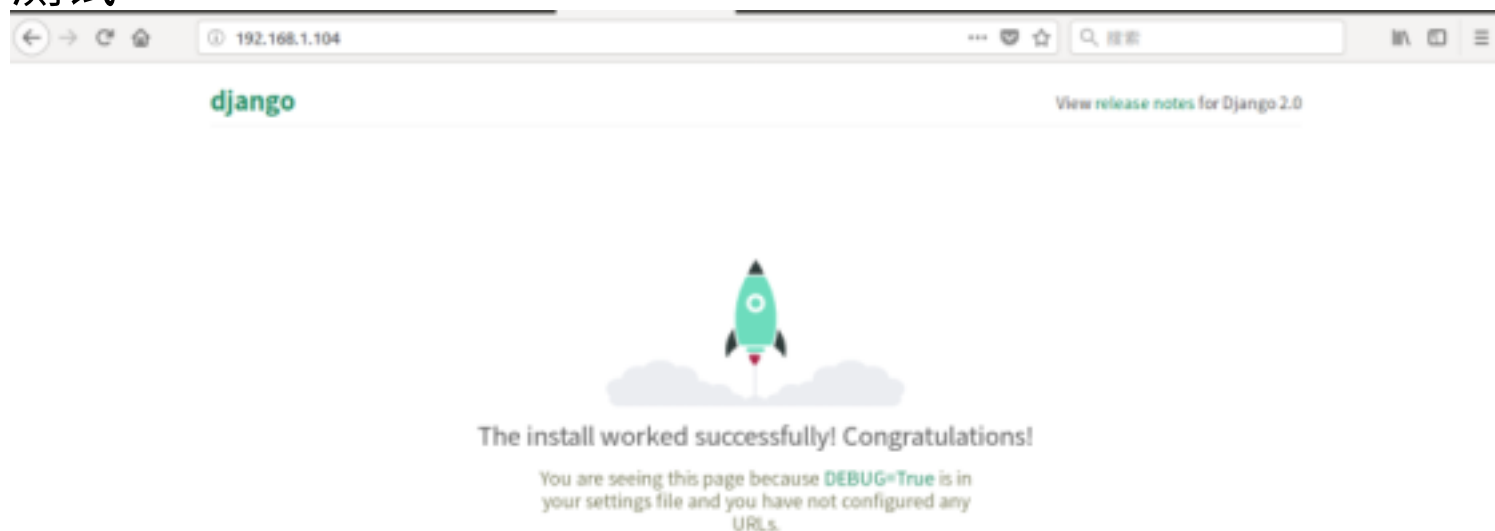
```
location / {  
    include uwsgi_params;  
    uwsgi_pass 127.0.0.1:9090;  
}
```

.....

重新启动Nginx

```
[root@localhost Test]# systemctl start nginx
```

测试



Nginx部署(Ubuntu)

基于Ubuntu搭建Django+Nginx+uwsgi环境

只有在测试环境中才会使用 `python manage.py runserver` 来运行服务器。

正式发布的服务，我们需要一个可以稳定、持续、支持高并发的服务器，比如apache, Nginx, lighttpd等。

接下来我们要以nginx为例子。

Nginx安装

安装Nginx

```
lzy@embsky:~$ sudo apt-get install nginx
```

启动Nginx

```
lzy@embsky:~$ sudo /etc/init.d/nginx start
```

测试Nginx



UWSGI安装

UWSGI安装及测试

安装uwsgi

```
lzy@embsky:~$ pip install uwsgi
```

编写测试网站代码

```
lzy@embsky:~$ touch test.py
```

```
lzy@embsky:~$ vim test.py
```

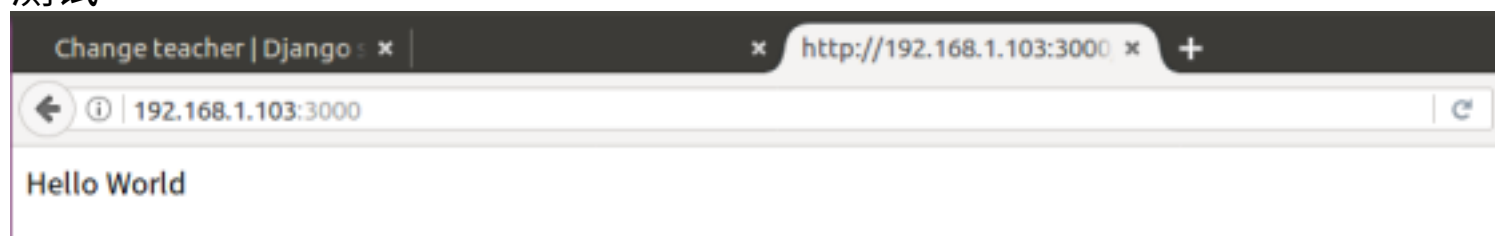
输入以下内容：

```
def application(env, start_response):  
    start_response('200 OK', [('Content-Type','text/html')])  
    return [b"Hello World"]
```

运行uwsgi

```
lzy@embsky:~$ uwsgi --http :3000 --wsgi-file test.py --enable-threads
```

测试



Nginx&Uwsgi

配置UWSGI和Nginx

Nginx负责相应用户请求，Uwsgi负责Nginx和Django之间的通信。

创建一个工程

```
lzy@embsky:~$ mkdir Nginx
```

```
lzy@embsky:~$ cd Nginx
lzy@embsky:~Nginx$ django-admin startproject Test
lzy@embsky:~Nginx$ cd Test/
lzy@embsky:~Nginx/Test$ django-admin startapp one
```

安装app

```
lzy@embsky:~Nginx/Test$ vim Test/setting.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'one'
]
```

修改访问IP

```
lzy@embsky:~Nginx/Test$ vim Test/setting.py
ALLOWED_HOSTS = ['*']
```

创建uwsgi配置文件

```
lzy@embsky:~Nginx/Test$ vim uwsgi.ini
```

```
[uwsgi]
```

#对外提供的端口号，不需要Nginx也可访问

```
http = :9000
```

#与Nginx通信的端口号

```
socket = 0.0.0.0:9090
```

#工程路径

```
chdir = /home/zyli/Nginx/Test
```

```
wsgi-file = Test/wsgi.py
```

#启动4个进程

```
processes = 4
```

#每个进程2个线程

```
threads = 2
```

#用于监控服务器负载的地址和端口

```
stats = 127.0.0.1:9191
```

#退出时候清除环境变量

vacuum = true

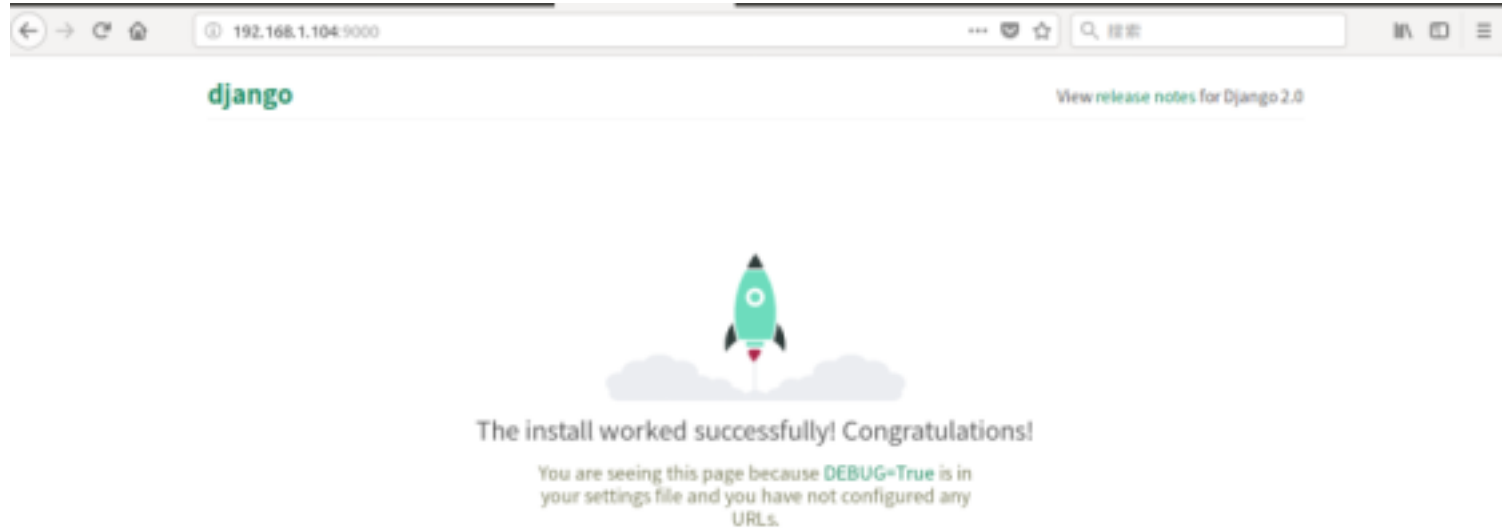
#日志文件

daemonize = /var/log/uwsgi.log

启动uwsgi

lzy@embsky:~Nginx/Test\$ uwsgi uwsgi.ini

测试



修改Nginx配置文件

lzy@embsky:~Nginx/Test\$ sudo vim /etc/nginx/sites-available/default

.....

```
location / {  
    include uwsgi_params;  
    uwsgi_pass 127.0.0.1:9090;  
}
```

.....

重新启动Nginx

lzy@embsky:~Nginx/Test\$ sudo /etc/init.d/nginx restart

测试



The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.