

□

首页HTMLCSSJS本地书签

Django 安装 □

Django 教程



Python下有许多款不同的 Web 框架。Django是重量级选手中最有代表性的一位。许多成功的网站和APP都基于Django。

Django是一个开放源代码的Web应用框架，由Python写成。

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。

谁适合阅读本教程？

本教程适合有Python基础的开发者学习。

学习本教程前你需要了解

学习本教程前你需要了解一些基础的 Web 知识及 [Python 2.x 基础教程](#) 或 [Python 3.x 基础教程](#)。

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 □

□ 点我分享笔记

反馈/建议

□

首页HTMLCSSJS本地书签

□ Django 教程Django 创建第一个项目 □

Django 安装

在安装 Django 前，系统需要已经安装了Python的开发环境。接下来我们来具体看下不同系统下Django的安装。

Window 下安装 Django

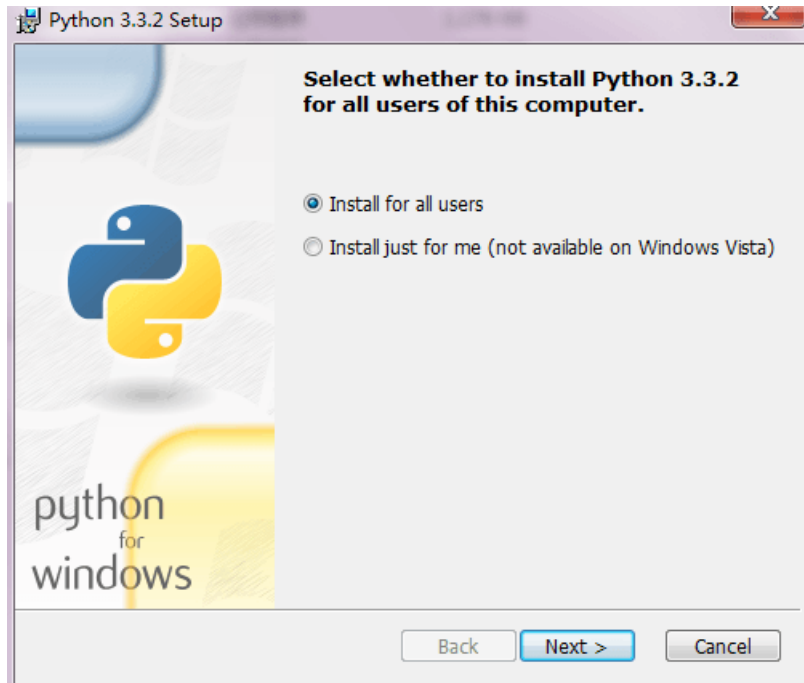
如果你还未安装Python环境需要先下载Python安装包。

- 1、Python 下载地址: <https://www.python.org/downloads/>
- 2、Django 下载地址: <https://www.djangoproject.com/download/>

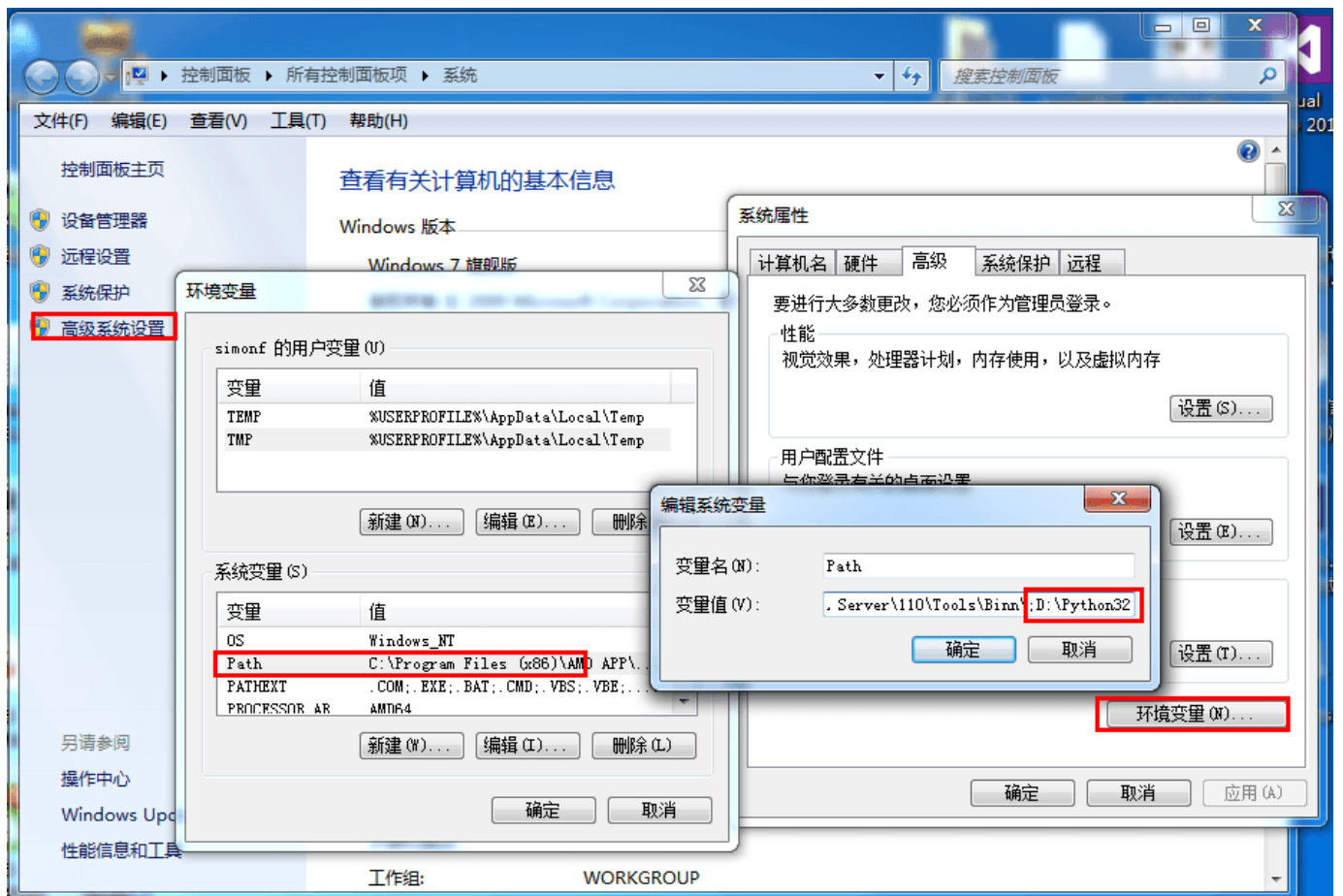
注意：目前Django 1.6.x以上版本已经完全兼容Python 3.x。

Python 安装(已安装的可跳过)

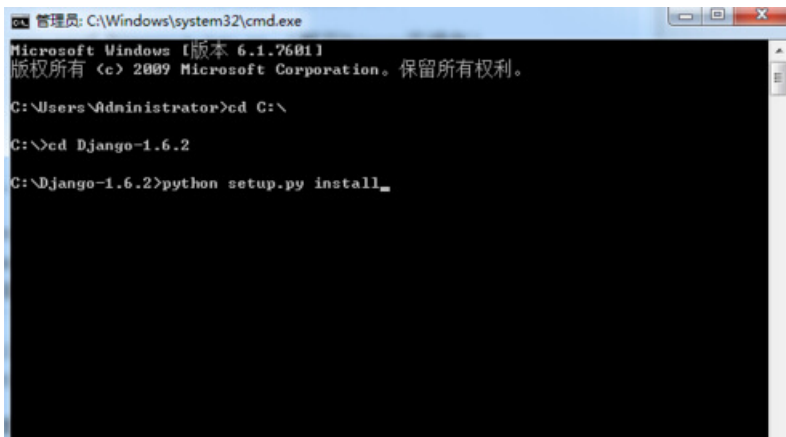
安装Python你只需要下载python-x.x.x.msi文件，然后一直点击"Next"按钮即可。



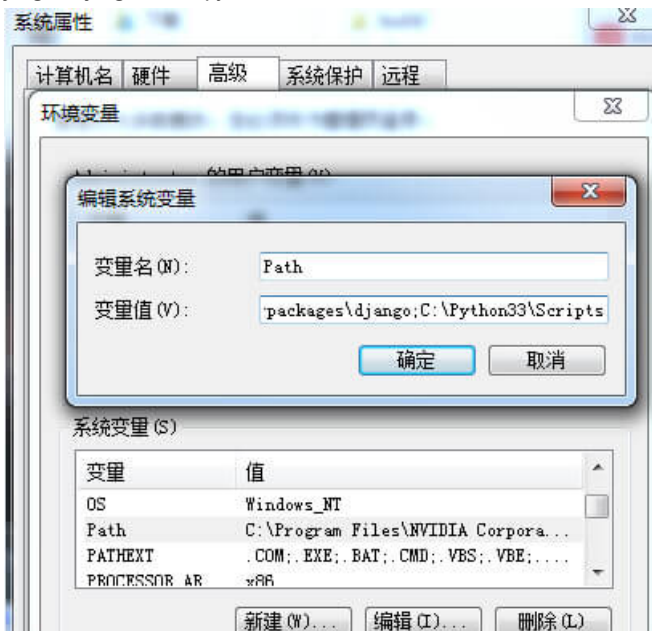
安装完成后你需要设置Python环境变量。右击计算机->属性->高级->环境变量->修改系统变量path，添加Python安装地址，本文实例使用的是C:\Python33，你需要根据你实际情况来安装。



下载 Django 压缩包，解压并和Python安装目录放在同一个根目录，进入 Django 目录，执行python setup.py install，然后开始安装，Django将要被安装到Python的Lib下site-packages。



然后是配置环境变量，将这几个目录添加到系统环境变量中： C:\Python33\Lib\site-packages\django;C:\Python33\Scripts。添加完成后就可以使用Django的django-admin.py命令新建工程了。



检查是否安装成功

输入以下命令进行检查：

```
>>> import django

>>> django.get_version()
```

```
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

C:\Users\Administrator>cd C:\

C:\>cd Django-1.6.2

C:\Django-1.6.2>python
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC v.1600 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.get_version()
'1.6.2'
>>>
```

如果输出了Django的版本号说明安装正确。

Linux 上安装 Django

yum 安装方法

以下安装位于 Centos Linux 环境下安装，如果是你的 Linux 系统是 ubuntu 请使用 apt-get 命令。

默认情况下 Linux 环境已经支持了Python。你可以在终端输入Python命令来查看是否已经安装。

```
Python 2.7.3 (default, Aug 1 2012, 05:14:39)

[GCC 4.6.3] on linux2

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

>>>
```

安装 setuptools

命令:

```
yum install python-setuptools
```

完成之后，就可以使用 easy_install 命令安装 django

```
easy_install django
```

之后我们在python解释器输入以下代码:

```
[root@solar django]# python

Python 2.7.3 (default, May 15 2014, 14:49:08)

[GCC 4.8.0] on linux2

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

>>> import django

>>> django.VERSION

(1, 6, 5, 'final', 0)

>>>
```

我们可以看到输出了Django的版本号，说明安装成功。

pip 命令安装方法

```
pip install Django
```

如果 pip < 1.4, 安装方法如下:

```
pip install https://www.djangoproject.com/download/1.11a1/tarball/
```

源码安装方法

下载源码包: <https://www.djangoproject.com/download/>

输入以下命令并安装:

```
tar xzvf Django-X.Y.tar.gz      # 解压下载包

cd Django-X.Y                   # 进入 Django 目录

python setup.py install          # 执行安装命令
```

安装成功后 Django 位于 Python 安装目录的 site-packages 目录下。

Mac 下安装

下载

从这里下载最新的稳定版本: Django-1.x.y.tar.gz, 在页面右侧列表下载, 如下图:

For the impatient:

- Latest release: [Django-1.9.tar.gz](#)
- Checksums: [Django-1.9.checksum.txt](#)
- Release notes: [Online documentation](#)

记住是最新的官方版本哦,其中xy是版本号。进入你下载该文件的文件夹目录,执行如下命令:(Mac下默认是/Users/xxx/Downloads, xxx是你的用户名)

```
$ tar xzvf Django-1.x.y.tar.gz
```

你也可以从 Github 上下载最新版,地址: <https://github.com/django/django>:

```
git clone https://github.com/django/django.git
```

安装

进入解压后的目录:

```
cd Django-1.x.y

sudo python setup.py install
```

安装成功后会输出以下信息:

```
.....
```

```
Processing dependencies for Django==1.x.y
```

```
Finished processing dependencies for Django==1.x.y
```

再进入我们的站点目录，创建 **Django** 项目：

```
$ django-admin.py startproject testdj
```

启动服务：

```
cd testdj # 切换到我们创建的项目
```

```
$ python manage.py runserver
```

```
.....
```

```
Starting development server at http://127.0.0.1:8000/
```

```
Quit the server with CONTROL-C.
```

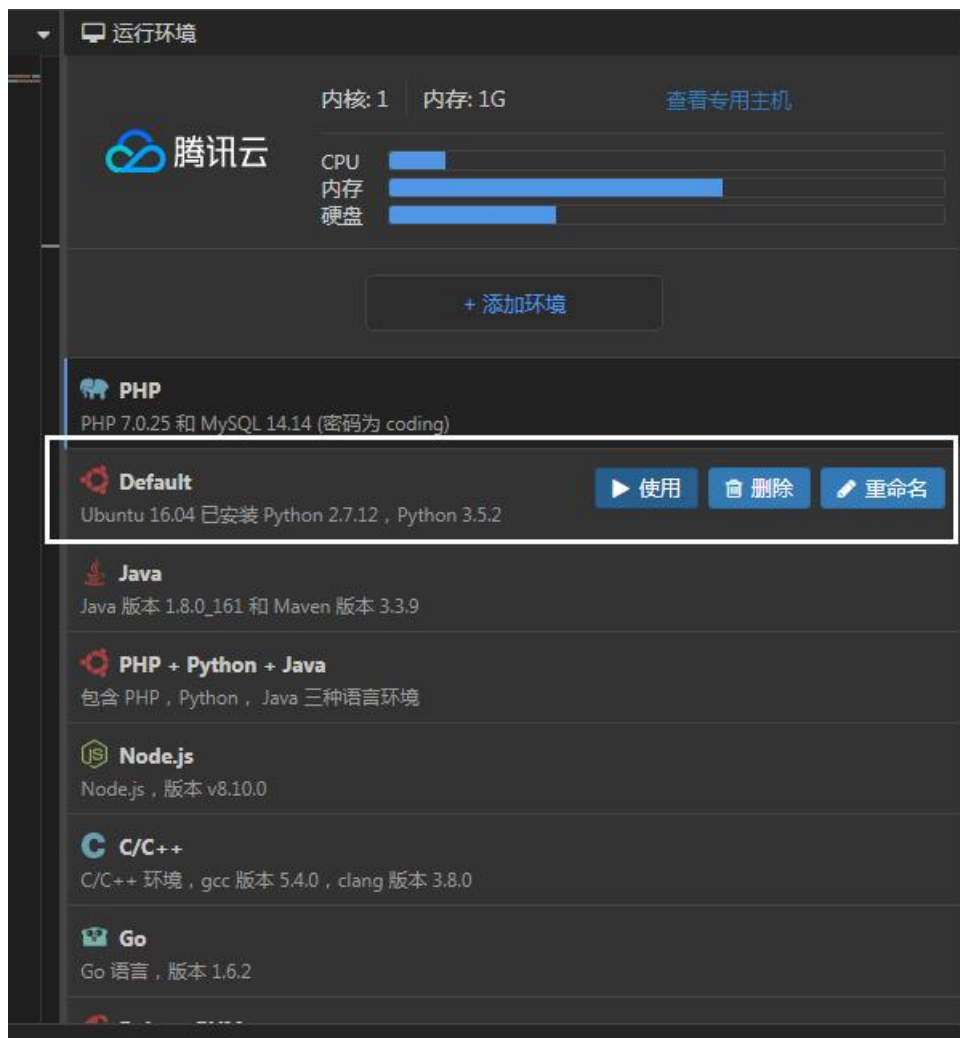
以上信息说明，项目已启动，访问地址为<http://127.0.0.1:8000/>。

在 **Cloud Studio** 中运行 **Django**

下面我们介绍如何在 **Cloud Studio** 中安装、使用 **Django**：

step1: 访问 [Cloud Studio](#)，注册/登录账户。

step2: 在右侧的运行环境菜单选择："ubuntu"



step3: 在下方的终端执行命令，使用 pip 安装：

```
sudo pip3 install django
```

step4: 查看 Django 版本：

```
django-admin --version
```

[Django 教程](#)

[Django 创建第一个项目](#)

[点我分享笔记](#)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号：闽ICP备15012807号-1

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

Django 创建第一个项目

本章我们将介绍 Django 管理工具及如何使用 Django 来创建项目，第一个项目我们以 **HelloWorld** 来命名项目。

测试版本说明：

Python 2.7.10

Django 1.10.6

Django 管理工具

安装 Django 之后，您现在应该已经有了可用的管理工具 `django-admin.py`。我们可以使用 `django-admin.py` 来创建一个项目：

我们可以来看下 `django-admin.py` 的命令介绍：

```
[root@solar ~]# django-admin.py
```

```
Usage: django-admin.py subcommand [options] [args]
```

Options:

`-v VERBOSITY, --verbosity=VERBOSITY`

Verbosity level; 0=minimal output, 1=normal output,

2=verbose output, 3=very verbose output

`--settings=SETTINGS` The Python path to a settings module, e.g.

"myproject.settings.main". If this isn't provided, the

DJANGO_SETTINGS_MODULE environment variable will be

used.

`--pythonpath=PYTHONPATH`

A directory to add to the Python path, e.g.

"/home/djangoprojects/myproject".

`--traceback` Raise on exception

`--version` show program's version number and exit

`-h, --help` show this help message and exit

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

Available subcommands:

[django]

check


```
cleanup

compilemessages

createcachetable

.....省略部分.....
```

创建第一个项目

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

```
django-admin.py startproject HelloWorld
```

最新版的 Django 请使用 `django-admin` 命令:

```
django-admin startproject HelloWorld
```

创建完成后我们可以查看下项目的目录结构:

```
$ cd HelloWorld/

$ tree

.

|-- HelloWorld

|   |-- __init__.py

|   |-- settings.py

|   |-- urls.py

|   `-- wsgi.py

`-- manage.py
```

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"。

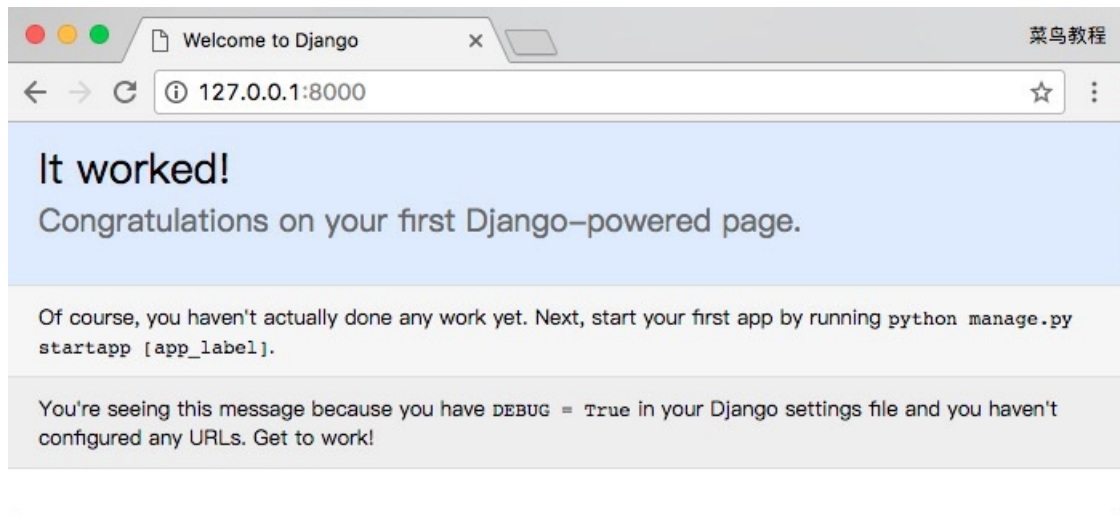
HelloWorld/wsgi.py: 一个 WSGI 兼容的 Web 服务器的入口,以便运行你的项目。

接下来我们进入 `HelloWorld` 目录输入以下命令,启动服务器:

```
python manage.py runserver 0.0.0.0:8000
```

0.0.0.0 让其它电脑可连接到开发服务器，8000 为端口号。如果不说明，那么端口号默认为 8000。

在浏览器输入你服务器的ip及端口号，如果正常启动，输出结果如下：



视图和 URL 配置

在先前创建的 HelloWorld 目录下的 HelloWorld 目录新建一个 view.py 文件，并输入代码：

HelloWorld/HelloWorld/view.py 文件代码：

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("Hello world ! ")
```

接着，绑定 URL 与视图函数。打开 urls.py 文件，删除原来代码，将以下代码复制粘贴到 urls.py 文件中：

HelloWorld/HelloWorld/urls.py 文件代码：

```
from django.conf.urls import url
from . import view
urlpatterns = [
    url(r'^$', view.hello),
]
```

整个目录结构如下：

```
$ tree

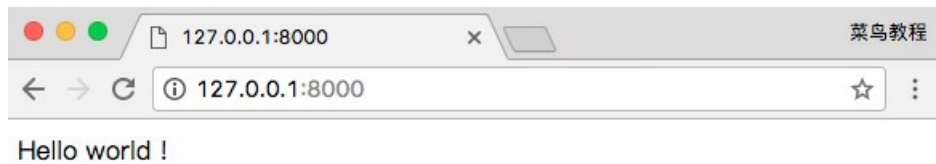
.
|-- HelloWorld
|   |-- __init__.py
|   |-- __init__.pyc
|   |-- settings.py
|   |-- settings.pyc
|   |-- urls.py          # url 配置
|   |-- urls.pyc
|   |-- view.py          # 添加的视图文件
|   |-- view.pyc         # 编译后的视图文件
```

```
| | -- wsgi.py

| `-- wsgi.pyc

`-- manage.py
```

完成后，启动 Django 开发服务器，并在浏览器访问打开浏览器并访问：



我们也可以修改以下规则：

HelloWorld/HelloWorld/urls.py 文件代码：

```
from django.conf.urls import url
from . import view
urlpatterns = [
    url(r'^hello$', view.hello),
]
```

通过浏览器打开 `http://127.0.0.1:8000/hello`，输出结果如下：



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，分别是两个必选参数：**regex**、**view** 和两个可选参数：**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。

[Django 安装](#)

[Django 模板](#)

2 篇笔记
#2

[写笔记](#)

启动django后，不能访问，报400错误。

原因：没有开启允许访问
处理：编辑HelloWorld目录下setting.py，把其中的
ALLOWED_HOSTS=[]改成ALLOWED_HOSTS=['*']##* 表示任意地址。

zwbill1年前 (2017-05-24)
#1



如果是 Django >= 2.0 的版本，urls.py 的 django.conf.urls 已经被 django.urls 取代。
django.urls 的用法参考如下：

```
from django.urls import path

from . import view


urlpatterns = [

    path('', view.hello),

    path('world/', view.world)

]
```

其中最大的几个改变如下：
import url 变成了 **import path**
如果是路径，则须在路径后加个 `/`
旧版 `django` 的用法：

```
from django.conf.urls import url

from . import view


urlpatterns = [

    url(r'^hello$', view.hello),

]
```

新版的参考写法：

```
from django.urls import path

from . import view


urlpatterns = [

    path('hello/', view.hello),

]
```

pan934122个月前 (07-25)

[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)[Django 创建第一个项目](#)[Django 模型](#)

Django 模板

在上一章节中我们使用 `django.http.HttpResponse()` 来输出 "Hello World! "。该方式将数据与视图混合在一起，不符合 Django 的 MVC 思想。本章节我们将为大家详细介绍 Django 模板的应用，模板是一个文本，用于分离文档的表现形式和内容。

模板应用实例

我们接着上一章节的项目将在 `HelloWorld` 目录底下创建 `templates` 目录并建立 `hello.html`文件，整个目录结构如下：

```
HelloWorld/

|-- HelloWorld

|   |-- __init__.py
|   |-- __init__.pyc
|   |-- settings.py
|   |-- settings.pyc
|   |-- urls.py
|   |-- urls.pyc
|   |-- view.py
|   |-- view.pyc
|   |-- wsgi.py
|   `-- wsgi.pyc

|-- manage.py

`-- templates

    `-- hello.html
```

`hello.html` 文件代码如下：

HelloWorld/templates/hello.html 文件代码：

```
<h1>{{ hello }}</h1>
```

从模板中我们知道变量使用了双括号。

接下来我们需要向 Django 说明模板文件的路径，修改 `HelloWorld/settings.py`，修改 `TEMPLATES` 中的 `DIRS` 为 `[BASE_DIR+"/templates",]`，如下所示：

HelloWorld/HelloWorld/settings.py 文件代码：

```
...TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [BASE_DIR+"/templates",], # 修改位置
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]
```

我们现在修改 **view.py**，增加一个新的对象，用于向模板提交数据：

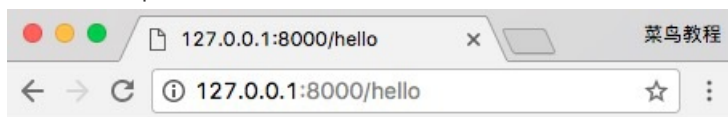
HelloWorld/HelloWorld/view.py 文件代码：

```
# -*- coding: utf-8 -*-
#from django.http import HttpResponse
from django.shortcuts import render
def hello(request):
    context = {}
    context['hello'] = 'Hello World!'
    return render(request, 'hello.html', context)
```

可以看到，我们这里使用 **render** 来替代之前使用的 **HttpResponse**。**render** 还使用了一个字典 **context** 作为参数。

context 字典中元素的键值 "hello" 对应了模板中的变量 "{ { hello } }"。

再访问访问 <http://127.0.0.1:8000/hello>，可以看到页面：



Hello World!

这样我们就完成了使用模板来输出数据，从而实现数据与视图分离。

接下来我们将具体介绍模板中常用的语法规则。

Django 模板标签

if/else 标签

基本语法格式如下：

```
{% if condition %}

    ... display

{% endif %}
```

或者：

```
{% if condition1 %}

    ... display 1
```

```
{% elif condition2 %}

    ... display 2

{% else %}

    ... display 3

{% endif %}
```

根据条件判断是否输出。**if/else** 支持嵌套。

{% if %} 标签接受 **and** , **or** 或者 **not** 关键字来对多个变量做判断 , 或者对变量取反 (**not**), 例如:

```
{% if athlete_list and coach_list %}

    athletes 和 coaches 变量都是可用的。

{% endif %}
```

for 标签

{% for %} 允许我们在一个序列上迭代。

与Python的 **for** 语句的情形类似, 循环语法是 **for X in Y** , **Y**是要迭代的序列而**X**是在每一个特定的循环中使用的变量名称。

每一次循环中, 模板系统会渲染在 **{% for %}** 和 **{% endfor %}** 之间的所有内容。

例如, 给定一个运动员列表 **athlete_list** 变量, 我们可以使用下面的代码来显示这个列表:

```
<ul>

{% for athlete in athlete_list %}

    <li>{{ athlete.name }}</li>

{% endfor %}

</ul>
```

给标签增加一个 **reversed** 使得该列表被反向迭代:

```
{% for athlete in athlete_list reversed %}

...

{% endfor %}
```

可以嵌套使用 **{% for %}** 标签:

```
{% for athlete in athlete_list %}

    <h1>{{ athlete.name }}</h1>

    <ul>
```

```
{% for sport in athlete.sports_played %}

    <li>{{ sport }}</li>

{% endfor %}

</ul>

{% endfor %}
```

ifequal/ifnotequal 标签

{% ifequal %} 标签比较两个值，当他们相等时，显示在 {% ifequal %} 和 {% endifequal %} 之中所有的值。

下面的例子比较两个模板变量 `user` 和 `currentuser`：

```
{% ifequal user currentuser %}

    <h1>Welcome!</h1>

{% endifequal %}
```

和 {% if %} 类似， {% ifequal %} 支持可选的 {% else %} 标签：8

```
{% ifequal section 'sitenews' %}

    <h1>Site News</h1>

{% else %}

    <h1>No News Here</h1>

{% endifequal %}
```

注释标签

Django 注释使用 `{# #}`。

```
{# 这是一个注释 #}
```

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。

include 标签

{% include %} 标签允许在模板中包含其它的模板的内容。

下面这个例子都包含了 **nav.html** 模板：

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

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：

HelloWorld/templates/base.html 文件代码：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>
<h1>Hello World!</h1>
<p>菜鸟教程 Django 测试。</p>
{% block mainbody %}
<p>original</p>
{% endblock %}
</body>
</html>
```

以上代码中，名为 **mainbody** 的 **block** 标签是可以被继承者们替换掉的部分。

所有的 **{% block %}** 标签告诉模板引擎，子模板可以重载这些部分。

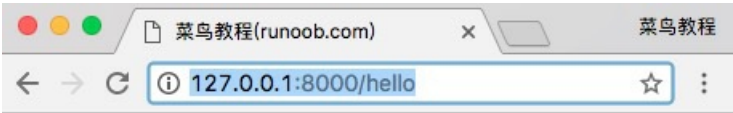
hello.html 中继承 **base.html**，并替换特定 **block**，**hello.html** 修改后的代码如下：

HelloWorld/templates/hello.html 文件代码：

```
{%extends "base.html" %}
{% block mainbody %}
<p>继承了 base.html 文件</p>
{% endblock %}
```

第一行代码说明 **hello.html** 继承了 **base.html** 文件。可以看到，这里相同名字的 **block** 标签用以替换 **base.html** 的相应 **block**。

重新访问地址 **http://127.0.0.1:8000/hello**，输出结果如下：



Hello World!

菜鸟教程 Django 测试。

继承了 base.html 文件

[Django 创建第一个项目](#)

[Django 模型](#)



1 篇笔记
#1



[写笔记](#)

关于报错: **TemplateDoesNotExist** (**Django 1.11.6 Python 3.6**)
没找到模板的问题一般都较为简单, 在print(BASE_DIR)之后发现目录还有一级, 再填上就好了。
附上配置:

```
'DIRS': [os.path.join(BASE_DIR, 'helloworld/templates')],
```

PS:注意斜杠是 '/'
一个萌新踩过的坑

xxdd12个月前 (10-16)

[反馈/建议](#)

Copyright © 2013-2018 菜鸟教程 [runoob.com](#) All Rights Reserved. 备案号: 闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

[Django 模板](#)

[Django 表单](#)

Django 模型

Django 对各种数据库提供了很好的支持, 包括: PostgreSQL、MySQL、SQLite、Oracle。
Django 为这些数据库提供了统一的调用API。我们可以根据自己业务需求选择不同的数据库。
MySQL 是 Web 应用中最常用的数据库。本章节我们将以 Mysql 作为实例进行介绍。你可以通过本站的 [MySQL 教程](#) 了解更多Mysql的基础知识。
如果你没安装 mysql 驱动, 可以执行以下命令安装:

```
sudo pip install mysqlclient
```

数据库配置

我们在项目的 settings.py 文件中找到 DATABASES 配置项, 将其信息修改为:

HelloWorld/HelloWorld/settings.py: 文件代码:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # 或者使用 mysql.connector.django
        'NAME': 'test',
        'USER': 'test',
        'PASSWORD': 'test123',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

这里添加了中文注释，所以你需要在 `HelloWorld/settings.py` 文件头部添加 `# -*- coding: UTF-8 -*-`。

上面包含数据库名称和用户的信息，它们与 MySQL 中对应数据库和用户的设置相同。Django 根据这一设置，与 MySQL 中相应的数据库和用户连接起来。

定义模型

创建 APP

Django规定，如果要使用模型，必须要创建一个app。我们使用以下命令创建一个 `TestModel` 的 app:

```
django-admin startapp TestModel
```

目录结构如下：

```
HelloWorld

|-- TestModel

|   |-- __init__.py

|   |-- admin.py

|   |-- models.py

|   |-- tests.py

|   `-- views.py
```

我们修改 `TestModel/models.py` 文件，代码如下：

HelloWorld/TestModel/models.py: 文件代码：

```
# models.py
from django.db import models
class Test(models.Model):
    name = models.CharField(max_length=20)
```

以上的类名代表了数据库表名，且继承了 `models.Model`，类里面的字段代表数据表中的字段(name)，数据类型则由 `CharField`（相当于 `varchar`）、`DateTimeField`（相当于 `datetime`），`max_length` 参数限定长度。

接下来在 `settings.py` 中找到 `INSTALLED_APPS` 这一项，如下：

```
INSTALLED_APPS = (

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',
```

```
'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'TestModel',          # 添加此项

)
```

在命令行中运行：

```
$ python manage.py migrate    # 创建表结构

$ python manage.py makemigrations TestModel  # 让 Django 知道我们在我们的模型有一些变更

$ python manage.py migrate TestModel    # 创建表结构
```

看到几行 "Creating table..." 的字样，你的数据表就创建好了。

```
Creating tables ...

.....

Creating table TestModel_test  #我们自定义的表

.....
```

表名组成结构为：应用名_类名（如：TestModel_test）。

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。

数据库操作

接下来我们在 HelloWorld 目录中添加 testdb.py 文件（下面介绍），并修改 urls.py:

HelloWorld/HelloWorld/urls.py: 文件代码：

```
from django.conf.urls import *
from . import view,testdb
urlpatterns = [
    url(r'^hello$', view.hello),
    url(r'^testdb$', testdb.testdb),
]
```

添加数据

添加数据需要先创建对象，然后再执行 save 函数，相当于SQL中的INSERT:

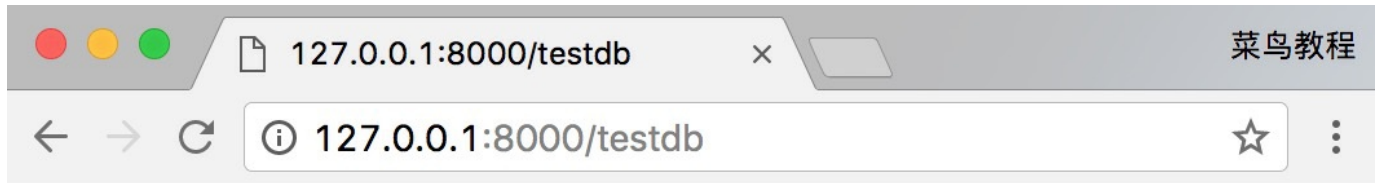
HelloWorld/HelloWorld/testdb.py: 文件代码：

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from TestModel.models import Test
# 数据库操作
def testdb(request):
    test1 = Test(name='runoob')
    test1.save()
```

```
return HttpResponse("<p>数据添加成功! </p>")
```

访问 <http://127.0.0.1:8000/testdb> 就可以看到数据添加成功的提示。

输出结果如下:



数据添加成功!

获取数据

Django提供了多种方式来获取数据库的内容, 如下代码所示:

HelloWorld/HelloWorld/testdb.py: 文件代码:

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from TestModel.models import Test
# 数据库操作
def testdb(request):
    # 初始化
    response = ""
    response1 = ""
    # 通过objects这个模型管理器的all()获得所有数据行, 相当于SQL中的SELECT * FROM
    list = Test.objects.all()
    # filter相当于SQL中的WHERE, 可设置条件过滤结果
    response2 = Test.objects.filter(id=1)
    # 获取单个对象
    response3 = Test.objects.get(id=1)
    # 限制返回的数据 相当于 SQL 中的 OFFSET 0 LIMIT 2;
    Test.objects.order_by('name')[0:2]
    #数据排序
    Test.objects.order_by("id")
    # 上面的方法可以连锁使用
    Test.objects.filter(name="runoob").order_by("id")
    # 输出所有数据
    for var in list:
        response1 += var.name + " "
    response = response1
    return HttpResponse("<p>" + response + "</p>")
```

更新数据

修改数据可以使用 `save()` 或 `update()`:

HelloWorld/HelloWorld/testdb.py: 文件代码:

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from TestModel.models import Test
# 数据库操作
def testdb(request):
    # 修改其中一个id=1的name字段, 再save, 相当于SQL中的UPDATE
    test1 = Test.objects.get(id=1)
    test1.name = 'Google'
    test1.save()
    # 另外一种方式
    #Test.objects.filter(id=1).update(name='Google')
    # 修改所有的列
```

```
# Test.objects.all().update(name='Google')
return HttpResponse("<p>修改成功</p>")
```

删除数据

删除数据库中的对象只需调用该对象的`delete()`方法即可：

HelloWorld/HelloWorld/testdb.py: 文件代码：

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from TestModel.models import Test
# 数据库操作
def testdb(request):
    # 删除id=1的数据
    test1 = Test.objects.get(id=1)
    test1.delete()
    # 另外一种方式
    # Test.objects.filter(id=1).delete()
    # 删除所有数据
    # Test.objects.all().delete()
    return HttpResponse("<p>删除成功</p>")
```

☐ Django 模板

Django 表单 ☐

☐ 点我分享笔记

反馈/建议

Copyright © 2013-2018 菜鸟教程 [runoob.com](http://www.runoob.com) All Rights Reserved. 备案号：闽ICP备15012807号-1



[首页](#) [HTML](#) [CSS](#) [JS](#) [本地书签](#)

☐ Django 模型

Django Admin 管理工具 ☐

Django 表单

HTML表单是网站交互性的经典方式。 本章将介绍如何用Django对用户提交的表单数据进行处理。

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

我们在之前的项目中创建一个 `search.py` 文件，用于接收用户的请求：

/HelloWorld/HelloWorld/search.py 文件代码：

```
# -*- coding: utf-8 -*-
from django.http import HttpResponse
from django.shortcuts import render_to_response
# 表单
def search_form(request):
    return render_to_response('search_form.html')
# 接收请求数据
```

```
def search(request):
    request.encoding='utf-8'
    if 'q' in request.GET:
        message = '你搜索的内容为: ' + request.GET['q']
    else:
        message = '你提交了空表单'
    return HttpResponse(message)
```

在模板目录 `templates` 中添加 `search_form.html` 表单:

/HelloWorld/templates/search_form.html 文件代码:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>
<form action="/search" method="get">
<input type="text" name="q">
<input type="submit" value="搜索">
</form>
</body>
</html>
```

`urls.py` 规则修改为如下形式:

/HelloWorld/HelloWorld/urls.py 文件代码:

```
from django.conf.urls import url
from . import view, testdb, search
urlpatterns = [
    url(r'^hello$', view.hello),
    url(r'^testdb$', testdb.testdb),
    url(r'^search-form$', search.search_form),
    url(r'^search$', search.search),
]
```

访问地址 `http://127.0.0.1:8000/search-form` 并搜索, 结果如下所示:



POST 方法

上面我们使用了GET方法。视图显示和请求处理分成两个函数处理。

提交数据时更常用POST方法。我们下面使用该方法, 并用一个URL和处理函数, 同时显示视图和处理请求。

我们在 `templates` 创建 `post.html`:

/HelloWorld/templates/post.html 文件代码:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>菜鸟教程(runoob.com)</title>
</head>
<body>
<form action="/search-post" method="post">
{% csrf_token %}
<input type="text" name="q">
<input type="submit" value="Submit">
</form>
```

```
</form>
<p>{{ rlt }}</p>
</body>
</html>
```

在模板的末尾，我们增加一个 `rlt` 记号，为表格处理结果预留位置。

表格后面还有一个 `{% csrf_token %}` 的标签。`csrf` 全称是 `Cross Site Request Forgery`。这是 Django 提供的防止伪装提交请求的功能。POST 方法提交的表格，必须有此标签。

在 `HelloWorld` 目录下新建 `search2.py` 文件并使用 `search_post` 函数来处理 POST 请求：

/HelloWorld/HelloWorld/search2.py 文件代码：

```
# -*- coding: utf-8 -*-
from django.shortcuts import render
from django.views.decorators import csrf
# 接收POST请求数据
def search_post(request):
    ctx = {}
    if request.POST:
        ctx['rlt'] = request.POST['q']
    return render(request, "post.html", ctx)
```

`urls.py` 规则修改为如下形式：

/HelloWorld/HelloWorld/urls.py 文件代码：

```
from django.conf.urls import url
from . import view, testdb, search, search2
urlpatterns = [
    url(r'^hello$', view.hello),
    url(r'^testdb$', testdb.testdb),
    url(r'^search-form$', search.search_form),
    url(r'^search$', search.search),
    url(r'^search-post$', search2.search_post),
]
```

访问 `http://127.0.0.1:8000/search-post` 显示结果如下：



完成以上实例后，我们的目录结构为：

```
HelloWorld

|-- HelloWorld

|   |-- __init__.py
|   |-- __init__.pyc
|   |-- search.py
|   |-- search.pyc
|   |-- search2.py
|   |-- search2.pyc
```



```
| |-- settings.py

| |-- settings.pyc

| |-- testdb.py

| |-- testdb.pyc

| |-- urls.py

| |-- urls.pyc

| |-- view.py

| |-- view.pyc

| |-- wsgi.py

| `-- wsgi.pyc

|-- TestModel

| |-- __init__.py

| |-- __init__.pyc

| |-- admin.py

| |-- admin.pyc

| |-- apps.py

| |-- migrations

| | |-- 0001_initial.py

| | |-- 0001_initial.pyc

| | |-- __init__.py

| | `-- __init__.pyc

| |-- models.py

| |-- models.pyc

| |-- tests.py

| `-- views.py

|-- db.sqlite3

|-- manage.py

`-- templates

    |-- base.html

    |-- hello.html

    |-- post.html

    `-- search_form.html
```

Request 对象

每个 `view` 函数的第一个参数是一个 `HttpRequest` 对象，就像下面这个 `hello()` 函数：

```
from django.http import HttpResponse

def hello(request):

    return HttpResponse("Hello world")
```

`HttpRequest`对象包含当前请求URL的一些信息：

属性	描述
path	请求页面的全路径,不包括域名—例如, <code>"/hello"</code> 。
method	请求中使用的HTTP方法的字符串表示。全大写表示。例如: <code>if request.method == 'GET':</code> <code>do_something()</code> <code>elif request.method == 'POST':</code> <code>do_something_else()</code>
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HTTP POST方法提交请求，但是表单中可以没有数据。因此，不能使用语句 <code>if request.POST</code> 来判断是否使用HTTP POST方法；应该使用 <code>if request.method == "POST"</code> (参见本表的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，然后再查找GET属性。借鉴PHP's <code>\$_REQUEST</code> 。 例如，如果GET = { <code>"name": "john"</code> } 和POST = { <code>"age": "34"</code> },则 <code>REQUEST["name"]</code> 的值是 <code>"john"</code> , <code>REQUEST["age"]</code> 的值是 <code>"34"</code> 。 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 >标签中name属性的值。files中的每个value="" name="" type="file" 同时也是一个标准python字典对象，包含下面三个keys:<br=""/> 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"属性时FILES才拥有数据。否则，FILES 是一个空字典。

META	<p>包含所有可用HTTP头部信息的字典。例如：</p> <p>CONTENT_LENGTH</p> <p>CONTENT_TYPE</p> <p>QUERY_STRING: 未解析的原始查询字符串</p> <p>REMOTE_ADDR: 客户端IP地址</p> <p>REMOTE_HOST: 客户端主机名</p> <p>SERVER_NAME: 服务器主机名</p> <p>SERVER_PORT: 服务器端口</p> <p>META 中这些头加上前缀HTTP_最为Key, 例如:</p> <p>HTTP_ACCEPT_ENCODING</p> <p>HTTP_ACCEPT_LANGUAGE</p> <p>HTTP_HOST: 客户发送的HTTP主机头信息</p> <p>HTTP_REFERER: referring页</p> <p>HTTP_USER_AGENT: 客户端的user-agent字符串</p> <p>HTTP_X_BENDER: XBender头信息</p>
user	<p>是一个django.contrib.auth.models.User 对象，代表当前登录的用户。</p> <p>如果访问用户当前没有登录，user将被初始化为django.contrib.auth.models.AnonymousUser的实例。</p> <p>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. else: # Do something for anonymous users.</pre> <p>只有激活Django中的AuthenticationMiddleware时该属性才可用</p>
session	<p>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支持时该属性才可用。</p>
raw_post_data	<p>原始HTTP POST数据，未解析过。高级处理时会有用处。</p>

Request对象也有一些有用的方法：

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"]的操作。
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/?print=true"
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> 返回最后一个value。
<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 QueryDict 对象上被调用(就是通过copy()产生的一个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, 该方法添加字典 items, 而不是替换它们: <div><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></div>
<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__getitem__(): <div><pre>>>> q = QueryDict('a=1&a=2&a=3') >>> q.items() [('a', '3')]</pre></div>
<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__getitem__():

此外, QueryDict也有一些方法, 如下表:

方法	描述
copy()	返回对象的拷贝, 内部实现是用Python标准库的copy.deepcopy(). 该拷贝是mutable(可更改的) — 就是说, 可以更改该拷贝的值。
getlist(key)	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, 则返回空list。 It's guaranteed to return a list of some sort..
setlist(key,list_)	设置key的值为list_ (unlike __setitem__()).
appendlist(key,item)	添加item到和key关联的内部list.
setlistdefault(key,list)	和setdefault有一点不同, 它接受list而不是单个value作为参数。
lists()	和items()有一点不同, 它会返回key的所有值, 作为一个list, 例如: <div>>>> q = QueryDict('a=1&a=2&a=3') >>> q.lists() [('a', ['1', '2', '3'])]</div>
urlencode()	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=5").

❏

2 篇笔记

#2

❏ 写笔记

遇到问题:
执行search请求 响应, 出现错误, 中文解析不了:

'ascii' codec can't decode byte 0xe4 in position 0: ordinal not in range(128)

解决办法, 添加如下代码到 search.py

```
import sys

reload(sys)

sys.setdefaultencoding('utf8')
```



楼上说的中文解析不了的问题我也遇到了，加 u 标记成 **unicode** 字符即可。

```
message = '你搜索的内容为: ' + request.GET['q']
```

改成

```
message =u '你搜索的内容为: ' + request.GET['q']
```

zifengxue4个月前 (06-11)

反馈/建议



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。你可以在项目的 settings.py 中的 INSTALLED_APPS 看到它：

/HelloWorld/HelloWorld/settings.py 文件代码：

```
INSTALLED_APPS = (  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
)
```

django.contrib是一套庞大的功能集，它是Django基本代码的组成部分。

激活管理工具

通常我们在生成项目时会在 urls.py 中自动设置好，我们只需去掉注释即可。
配置项如下所示：

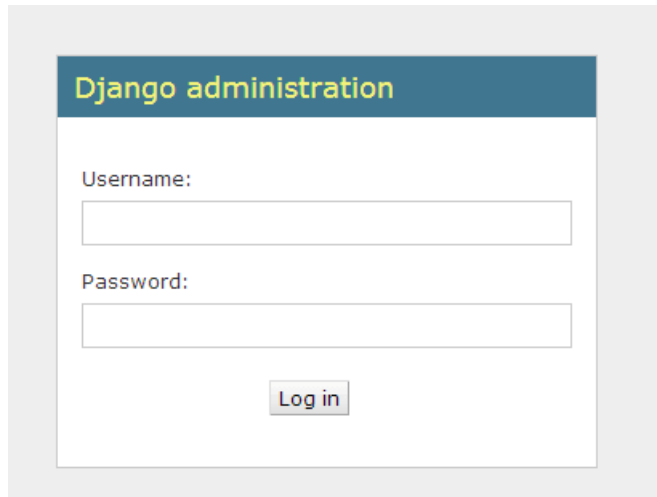
/HelloWorld/HelloWorld/urls.py 文件代码：

```
# urls.py  
from django.conf.urls import url  
from django.contrib import admin  
urlpatterns = [  
    url(r'^admin/', admin.site.urls),  
]
```

当这一切都配置好后，Django 管理工具就可以运行了。

使用管理工具

启动开发服务器，然后在浏览器中访问 `http://127.0.0.1:8000/admin/`，得到如下界面：



The image shows the Django administration login page. It has a dark blue header with the text 'Django administration' in yellow. Below the header, there are two input fields: 'Username:' and 'Password:'. At the bottom, there is a 'Log in' button.

你可以通过命令 `python manage.py createsuperuser` 来创建超级用户，如下所示：

```
# python manage.py createsuperuser

Username (leave blank to use 'root'): admin

Email address: admin@runoob.com

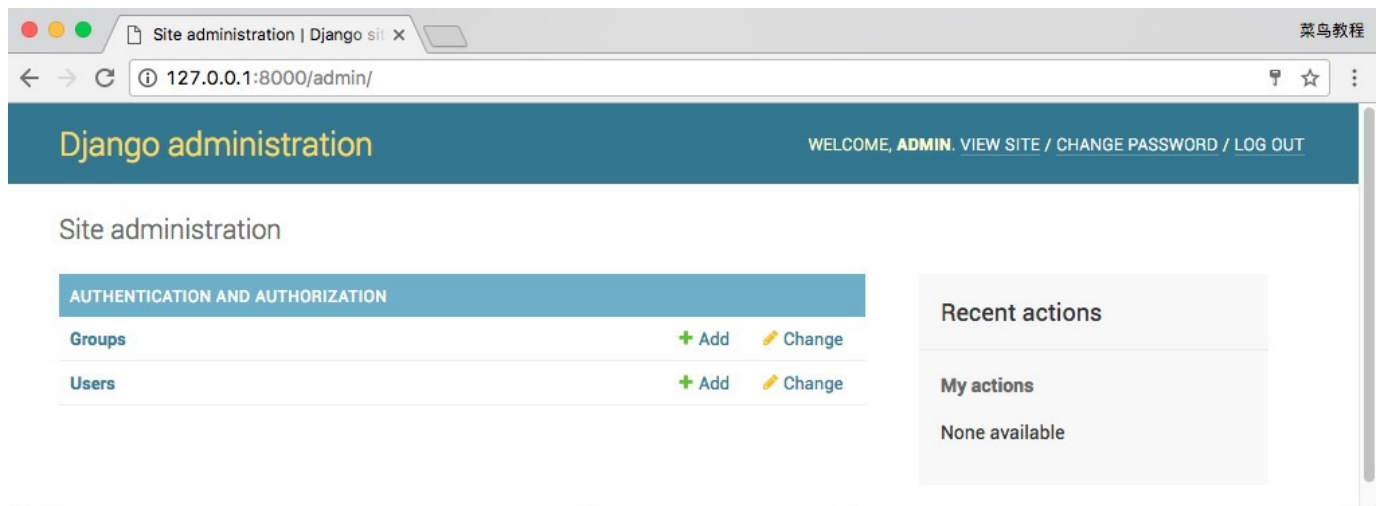
Password:

Password (again):

Superuser created successfully.

[root@solar HelloWorld]#
```

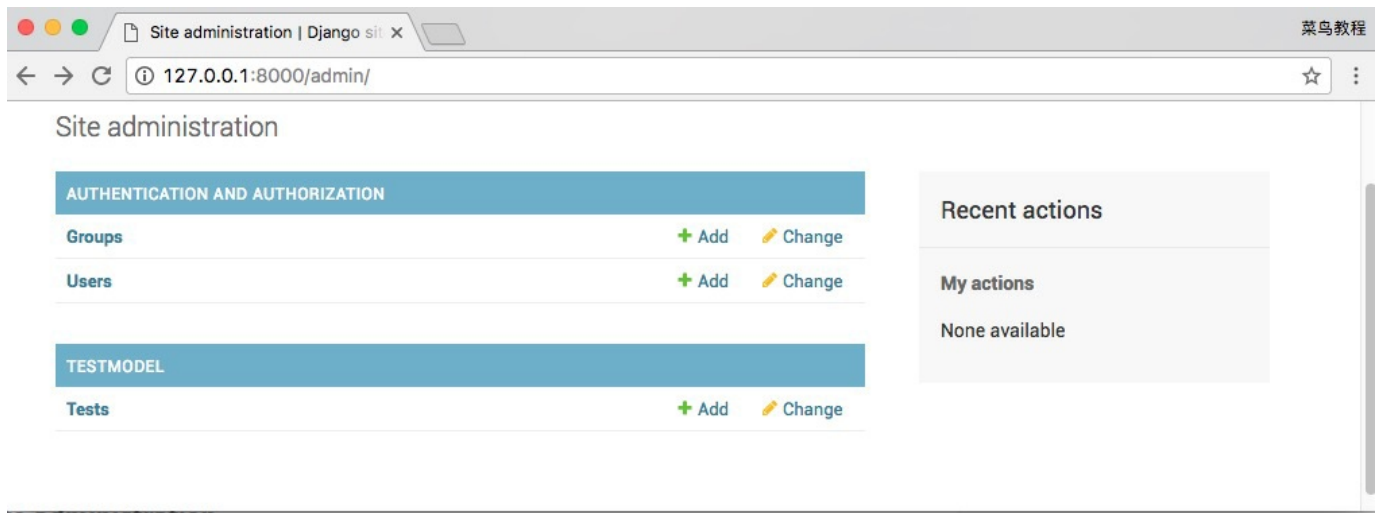
之后输入用户名密码登录，界面如下：



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`。比如，我们之前在 `TestModel` 中已经创建了模型 `Test`。修改 `TestModel/admin.py`：

```
from django.contrib import admin
from TestModel.models import Test
# Register your models here.
admin.site.register(Test)
```

刷新后即可看到 `Testmodel` 数据表：



复杂模型

管理页面的功能强大，完全有能力处理更加复杂的数据模型。

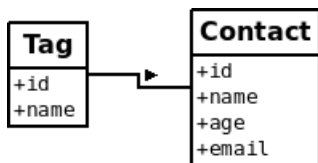
先在 `TestModel/models.py` 中增加一个更复杂的数据模型：

HelloWorld/TestModel/models.py: 文件代码：

```
from django.db import models
# Create your models here.
class Test(models.Model):
    name = models.CharField(max_length=20)
class Contact(models.Model):
    name = models.CharField(max_length=200)
    age = models.IntegerField(default=0)
    email = models.EmailField()
    def __unicode__(self):
        return self.name
class Tag(models.Model):
    contact = models.ForeignKey(Contact)
    name = models.CharField(max_length=50)
    def __unicode__(self):
        return self.name
```

这里有两个表。Tag 以 Contact 为外部键。一个 Contact 可以对应多个 Tag。

我们还可以看到许多在之前没有见过的属性类型，比如 `IntegerField` 用于存储整数。

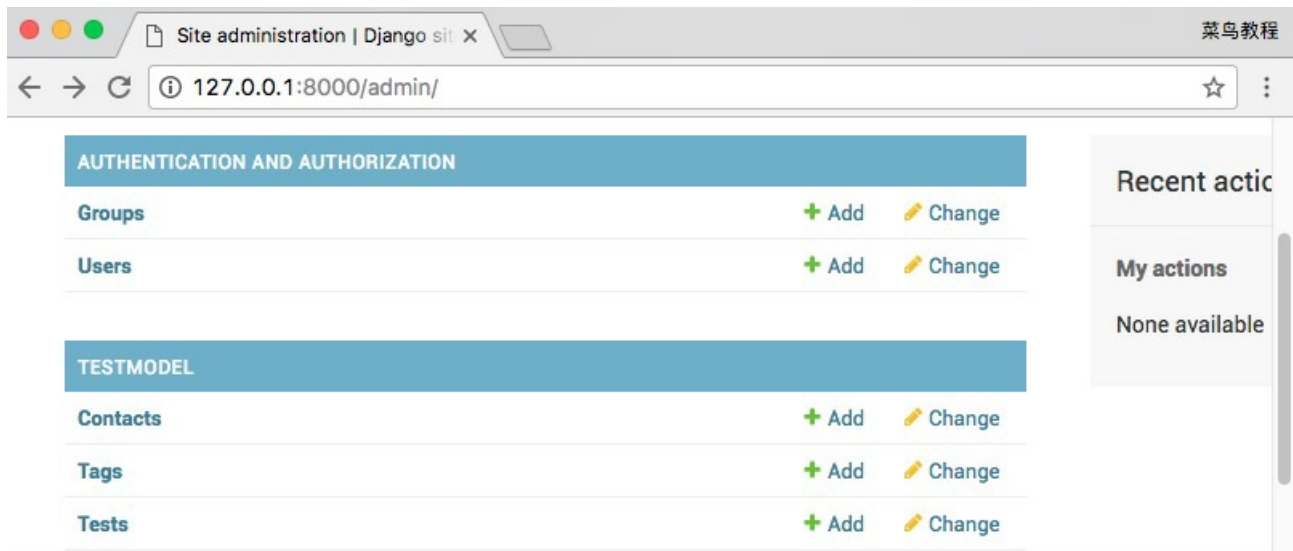


在 `TestModel/admin.py` 注册多个模型并显示：

HelloWorld/TestModel/admin.py: 文件代码：

```
from django.contrib import admin
from TestModel.models import Test, Contact, Tag
# Register your models here.
admin.site.register([Test, Contact, Tag])
```

刷新管理页面，显示结果如下：



在以上管理工具我们就能进行复杂模型操作。

如果你之前还未创建表结构，可使用以下命令创建：

```
$ python manage.py makemigrations TestModel # 让 Django 知道我们在我们的模型有一些变更

$ python manage.py migrate TestModel # 创建表结构
```

自定义表单

我们可以自定义管理页面，来取代默认的页面。比如上面的 "add" 页面。我们想只显示 **name** 和 **email** 部分。修改 `TestModel/admin.py`：

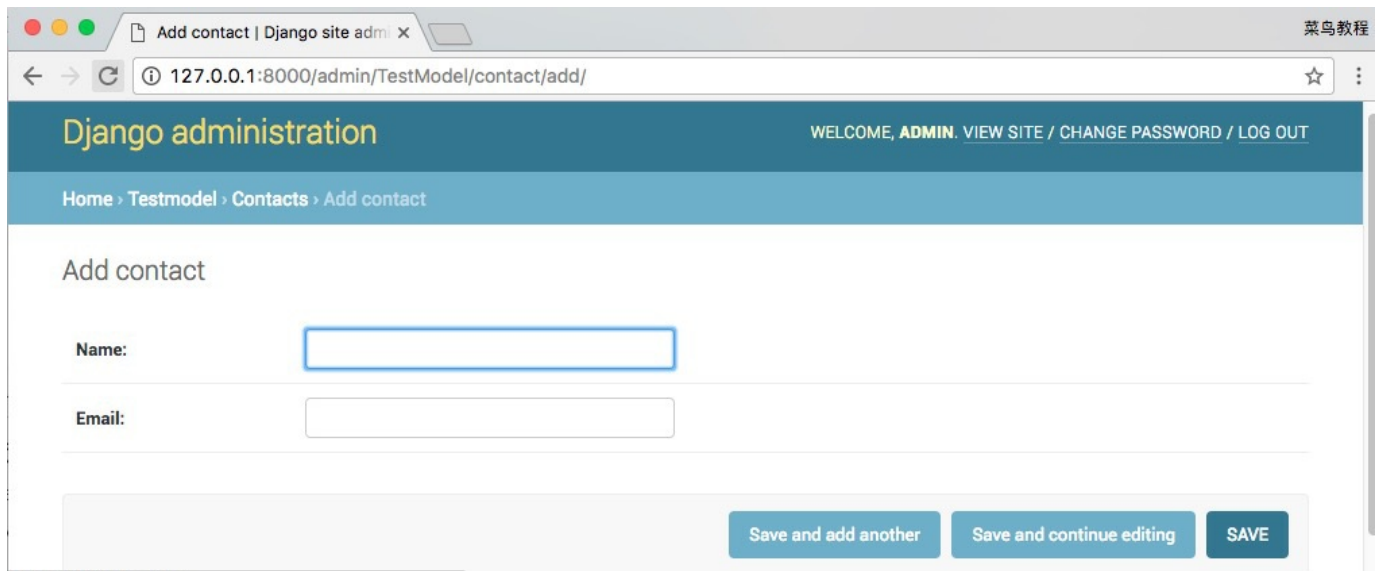
HelloWorld/TestModel/admin.py: 文件代码：

```
from django.contrib import admin
from TestModel.models import Test, Contact, Tag
# Register your models here.
class ContactAdmin(admin.ModelAdmin):
    fields = ('name', 'email')
admin.site.register(Contact, ContactAdmin)
admin.site.register([Test, Tag])
```

以上代码定义了一个 `ContactAdmin` 类，用以说明管理页面的显示格式。

里面的 `fields` 属性定义了要显示的字段。

由于该类对应的是 `Contact` 数据模型，我们在注册的时候，需要将它们一起注册。显示效果如下：



我们还可以将输入栏分块，每个栏也可以定义自己的格式。修改 TestModel/admin.py为：

HelloWorld/TestModel/admin.py: 文件代码:

```
from django.contrib import admin
from TestModel.models import Test,Contact,Tag
# Register your models here.
class ContactAdmin(admin.ModelAdmin):
    fieldsets = (
        ['Main',{
            'fields':('name','email'),
        }],
        ['Advance',{
            'classes': ('collapse',), # CSS
            'fields': ('age',),
        }]
    )
admin.site.register(Contact, ContactAdmin)
admin.site.register([Test, Tag])
```

上面的栏目分为了 Main 和 Advance 两部分。classes 说明它所在的部分的 CSS 格式。这里让 Advance 部分隐藏：

Main

Name:

Email:

Advance (Show)

Save and add another

Save and continue editing

SAVE

Advance 部分旁边有一个 Show 按钮，用于展开，展开后可点击 Hide 将其隐藏，如下图所示：

Main

Name:

Email:

Advance (Hide)

Age:

0

Save and add another

Save and continue editing

SAVE

内联(Inline)显示

上面的 Contact 是 Tag 的外部键，所以有外部参考的关系。
而在默认的页面显示中，将两者分离开来，无法体现出两者的从属关系。我们可以使用内联显示，让 Tag 附加在 Contact 的编辑页面上显示。
修改TestModel/admin.py:

HelloWorld/TestModel/admin.py: 文件代码:

```
from django.contrib import admin
from TestModel.models import Test,Contact,Tag
# Register your models here.
class TagInline(admin.TabularInline):
    model = Tag
class ContactAdmin(admin.ModelAdmin):
    inlines = [TagInline] # Inline
    fieldsets = (
        ['Main',{
            'fields':('name','email'),
```

```

    ]],
    ['Advance',{
    'classes': ('collapse',),
    'fields': ('age',),
    }]
    )
admin.site.register(Contact, ContactAdmin)
admin.site.register([Test])

```

显示效果如下:

列表页的显示

在 `Contact` 输入数条记录后, `Contact` 的列表页看起来如下:

我们也可以自定义该页面的显示, 比如在列表中显示更多的栏目, 只需要在 `ContactAdmin` 中增加 `list_display` 属性:

HelloWorld/TestModel/admin.py: 文件代码:

```

from django.contrib import admin
from TestModel.models import Test,Contact,Tag
# Register your models here.
class TagInline(admin.TabularInline):
    model = Tag
class ContactAdmin(admin.ModelAdmin):
    list_display = ('name','age', 'email') # list
    inlines = [TagInline] # Inline

```

```
fieldsets = (
    ['Main',{
        'fields':('name','email'),
    }],
    ['Advance',{
        'classes': ('collapse',),
        'fields': ('age',),
    }]
)
admin.site.register(Contact, ContactAdmin)
admin.site.register([Test])
```

刷新页面显示效果如下：

Action: 0 of 3 selected

<input type="checkbox"/>	NAME	AGE	EMAIL
<input type="checkbox"/>	alibaba	0	admin@alibaba.com
<input type="checkbox"/>	google	0	admin@google.com
<input type="checkbox"/>	runoob	3	admin@runoob.com

3 contacts

搜索功能在管理大量记录时非常有，我们可以使用 `search_fields` 为该列表页增加搜索栏：

HelloWorld/TestModel/admin.py: 文件代码：

```
from django.contrib import admin
from TestModel.models import Test,Contact,Tag
# Register your models here.
class TagInline(admin.TabularInline):
    model = Tag
class ContactAdmin(admin.ModelAdmin):
    list_display = ('name','age', 'email') # list
    search_fields = ('name',)
    inlines = [TagInline] # Inline
    fieldsets = (
        ['Main',{
            'fields':('name','email'),
        }],
        ['Advance',{
            'classes': ('collapse',),
            'fields': ('age',),
        }]
    )
admin.site.register(Contact, ContactAdmin)
admin.site.register([Test])
```

在本实例中我们搜索了 `name` 为 `runoob` 的记录，显示结果如下：

Select contact to change

1 result (3 total)

Action: 0 of 1 selected

<input type="checkbox"/>	NAME	AGE	EMAIL
<input type="checkbox"/>	runoob	3	admin@runoob.com

Django Admin 管理工具还有非常多实用的功能，感兴趣的同学可以深入研究下。

☐ Django 表单

Django Nginx+uwsgi 安装配置 ☐



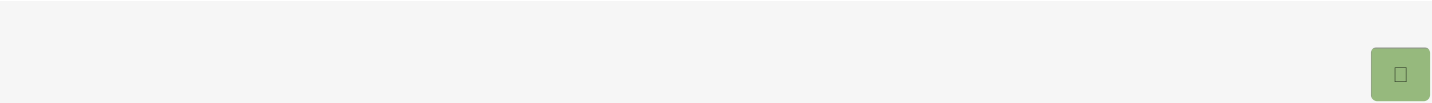


一个 `Contact` 类可以有多个 `Tag`:
关联 `contact` 外键时会报错: `TypeError: __init__() missing 1 required positional argument: 'on_delete'`
解决办法:

```
contact = models.ForeignKey(Contact, on_delete=models.CASCADE)
```

游客3个月前 [06-26]

反馈/建议



Django Admin 管理工具

Django Nginx+uwsgi 安装配置

在前面的章节中我们使用 `python manage.py runserver` 来运行服务器。这只适用测试环境中使用。
正式发布的服务，我们需要一个可以稳定而持续的服务器，比如apache, Nginx, lighttpd等，本文将以 Nginx 为例。

你也可以直接参考: [Python uwsgi 安装配置](#)

安装基础开发包

Centos 下安装步骤如下:

```
yum groupinstall "Development tools"

yum install zlib-devel bzip2-devel pcre-devel openssl-devel ncurses-devel sqlite-devel readline-devel tk-devel
```

CentOS 自带 Python 2.4.3, 但我们可以再安装Python2.7.5:

```
cd ~

wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tar.bz2

tar xvf Python-2.7.5.tar.bz2

cd Python-2.7.5

./configure --prefix=/usr/local

make && make altinstall
```

安装Python包管理

easy_install 包 <https://pypi.python.org/pypi/distribute>

安装步骤:

```
cd ~

wget https://pypi.python.org/packages/source/d/distribute/distribute-0.6.49.tar.gz

tar xf distribute-0.6.49.tar.gz

cd distribute-0.6.49

python2.7 setup.py install

easy_install --version
```

pip 包: <https://pypi.python.org/pypi/pip>

安装 pip 的好处是可以用 pip list、pip uninstall 管理 Python 包，easy_install 没有这个功能，只有 uninstall。

安装 uwsgi

uwsgi: <https://pypi.python.org/pypi/uWSGI>

uwsgi 参数详解: <http://uwsgi-docs.readthedocs.org/en/latest/Options.html>

```
pip install uwsgi

uwsgi --version    # 查看 uwsgi 版本
```

测试 uwsgi 是否正常:

新建 test.py 文件, 内容如下:

```
def application(env, start_response):

    start_response('200 OK', [('Content-Type','text/html')])

    return "Hello World"
```

然后在终端运行:

```
uwsgi --http :8001 --wsgi-file test.py
```

在浏览器内输入: <http://127.0.0.1:8001>, 查看是否有"Hello World"输出, 若没有输出, 请检查你的安装过程。

安装 Django

```
pip install django
```

测试 django 是否正常, 运行:

```
django-admin.py startproject demosite

cd demosite

python2.7 manage.py runserver 0.0.0.0:8002
```

在浏览器内输入：<http://127.0.0.1:8002>，检查django是否运行正常。

安装 Nginx

安装命令如下：

```
cd ~

wget http://nginx.org/download/nginx-1.5.6.tar.gz

tar xf nginx-1.5.6.tar.gz

cd nginx-1.5.6

./configure --prefix=/usr/local/nginx-1.5.6 \

--with-http_stub_status_module \

--with-http_gzip_static_module

make && make install
```

你可以阅读 [Nginx 安装配置](#) 了解更多内容。

uwsgi 配置

uwsgi支持ini、xml等多种配置方式，本文以 ini 为例，在/etc/目录下新建uwsgi9090.ini，添加如下配置：

```
[uwsgi]

socket = 127.0.0.1:9090

master = true           //主进程

vhost = true            //多站模式

no-site = true          //多站模式时不设置入口模块和文件

workers = 2             //子进程数

reload-mercy = 10

vacuum = true           //退出、重启时清理文件

max-requests = 1000

limit-as = 512

buffer-size = 30000

pidfile = /var/run/uwsgi9090.pid    //pid文件，用于下面的脚本启动、停止该进程
```

```
daemonize = /website/uwsgi9090.log
```

Nginx 配置

找到nginx的安装目录（如：/usr/local/nginx/），打开conf/nginx.conf文件，修改server配置：

```
server {  
  
    listen      80;  
  
    server_name localhost;  
  
    location / {  
  
        include uwsgi_params;  
  
        uwsgi_pass 127.0.0.1:9090;           //必须和uwsgi中的设置一致  
  
        uwsgi_param UWSGI_SCRIPT demosite.wsgi; //入口文件，即wsgi.py相对于项目根目录的位置，“.”相当于一层目录  
  
        uwsgi_param UWSGI_CHDIR /demosite;    //项目根目录  
  
        index index.html index.htm;  
  
        client_max_body_size 35m;  
  
    }  
  
}
```

你可以阅读 [Nginx 安装配置](#) 了解更多内容。

设置完成后，在终端运行：

```
uwsgi --ini /etc/uwsgi9090.ini &  
  
/usr/local/nginx/sbin/nginx
```

在浏览器输入：<http://127.0.0.1>，你就可以看到 django 的 "It work" 了。

☐ Django Admin 管理工具



1 篇笔记
#1



☐ 写笔记

安装 uwsgi 如果失败，有可能是缺少Python的头文件和静态库，需要安装开发版本：
For apt (Ubuntu, Debian...):

```
sudo apt-get install python-dev # for python2.x installs  
  
sudo apt-get install python3-dev # for python3.x installs
```

For yum (CentOS, RHEL...):


```
sudo yum install python-devel
```

For dnf (Fedora...):

```
sudo dnf install python2-devel # for python2.x installs
```

```
sudo dnf install python3-devel # for python3.x installs
```

For zypper (openSUSE...):

```
sudo zypper in python-devel # for python2.x installs
```

```
sudo zypper in python3-devel # for python3.x installs
```

tianqixin8个月前 (02-06)

反馈/建议