自强学堂
Django教程

Sun, 20 Sep 2015

自强学堂 Django教程

[Sun, 20 Sep 2015]

- 自强学堂 Django教程

自强学堂 Django教程

- Django 基础教程 [Sun, 20 Sep 12:27]
- Django 简介 [Sun, 20 Sep 12:27]
- Django 环境搭建 [Sun, 20 Sep 12:27]
- Django 基本命令 [Sun, 20 Sep 12:27]
- Django 视图与网址 [Sun, 20 Sep 12:27]
- Django 视图与网址进阶 [Sun, 20 Sep 12:27]
- Django 模板（templates) [Sun, 20 Sep 12:27]
- Django 模板进阶 [Sun, 20 Sep 12:27]
- Django渲染json到模板 [Sun, 20 Sep 12:27]
- Django 模型(数据库) [Sun, 20 Sep 12:27]
- Django 自定义 Field [Sun, 20 Sep 12:27]
- Django QuerySet API [Sun, 20 Sep 12:27]
- Django 后台 [Sun, 20 Sep 12:27]
- Django 表单 [Sun, 20 Sep 12:27]
- Django 配置 [Sun, 20 Sep 12:27]
- Django 静态文件 [Sun, 20 Sep 12:27]
- Django 部署 (Apache) [Sun, 20 Sep 12:27]
- Django 部署 (nginx) [Sun, 20 Sep 12:27]
- Django 数据导入 [Sun, 20 Sep 12:27]
- Django 数据迁移 [Sun, 20 Sep 12:27]
- Django 多数据库联用 [Sun, 20 Sep 12:27]
- Django 用户注册系统 [Sun, 20 Sep 12:27]
- Django 缓存系统 [Sun, 20 Sep 12:27]

- Django 生成静态网页 [Sun, 20 Sep 12:27]
- Django 安全 [Sun, 20 Sep 12:27]
- Django 国际化 [Sun, 20 Sep 12:27]
- Django session [Sun, 20 Sep 12:27]
- Django Ajax [Sun, 20 Sep 12:27]
- Django Ajax CSRF 认证 [Sun, 20 Sep 12:27]
- Django Sitemap 站点地图 [Sun, 20 Sep 12:27]
- 只用 Django 数据库 [Sun, 20 Sep 12:27]
- Django 通用视图 [Sun, 20 Sep 12:27]
- Django 中间件 [Sun, 20 Sep 12:27]
- Django 微信接口 [Sun, 20 Sep 12:27]
- Django 单元测试 [Sun, 20 Sep 12:27]
- 开发内容管理系统 [Sun, 20 Sep 12:27]
- 其它（书写中） [Sun, 20 Sep 12:27]
- Django CMS [Sun, 20 Sep 12:27]
- Python/Django 二维码 [Sun, 20 Sep 12:27]

Django 基础教程

Django 是由 Python 开发的一个免费的开源网站框架，可以用于快速搭建高性能，优雅的网站！

你一定可以学会，Django 很简单！本教程一直在更新，从开始写到现在大概写了一年多了，现在也一直在坚持写，每一篇教程都可能随时更新，可以在网站首页看到最近更新的情况。

我阅读学习了全部的 Django英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。本教程开始写的时候是 Django 的版本是 1.6，Django的更新很快，自强学堂的教程也随着更新了，兼顾了后来的新版本，从 Django 1.5 到最新的 Django 1.8 中应该都没有问题。

自强学堂 就是用 Django 搭建的站点！

本教程作者： **涂伟忠(未经同意,禁止转载!)**

除了本教程外的其它教程

自强学堂 学习分享 的一篇文章：**Django 学习资源**，如果有更好的教程也可以在文章下推荐哦！

开发过程中遇到的各种问题可以在这里提问（或者直接在相应的教程下回复该教程相关的问题）： Django 开发常见问题及答案汇总

学Django需要什么基础

1. Django是 python 语言写的一个网络框架的包，所以你得知道一些 Python 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 HTML, CSS, JavaScript 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

Django 特点

强大的数据库功能

用python的类继承，几行代码就可以拥有一个丰富，动态的数据库操作接口（API），如果需要你也能执行SQL语句

自带的强大的后台功能

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理你的内容！

优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

缓存系统

与memcached或其它的缓存系统联用，更出色的表现，更快的加载速度。

国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

Django 简介

自强学堂的django教程将节省你大量的时间，并且使你的web开发充满乐趣。通过Django，你可以建立一个高性能的web应用而只花费最少的时间和精力。

Django 中提供了开发网站经常用到的模块，常见的代码都为你写好了，通过减少重复的代码，Django 使你能够专注于 web 应用上有趣的关键性的东西。为了达到这个目标，Django 提供了通用Web开发模式的高度抽象，提供了频繁进行的编程作业的快速解决方法，以及为"如何解决问题"提供了清晰明了的约定。Django的理念是DRY(Don't Repeat Yourself)来鼓励快速开发！

让我们一览 Django 全貌

urls.py

网址入口，关联到对应的views.py中的一个函数（或者generic类），访问网址就对应一个函数。

views.py

处理用户发出的请求，从urls.py中对应过来，通过渲染templates中的网页可以将显示内容，比如登陆后的用户名，用户请求的数据，输出到网页。

models.py

与数据库操作相关，存入或读取数据时用到这个，当然用不到数据库的时候 你可以不使用。

forms.py

表单，用户在浏览器上输入数据提交，对数据的验证工作以及输入框的生成等工作，当然你也可以不使用。

templates 文件夹

views.py 中的函数渲染templates中的Html模板，得到动态内容的网页，当然可以用缓存来提高速度。

admin.py

后台，可以用很少量的代码就拥有一个强大的后台。

settings.py

Django 的设置，配置文件，比如 DEBUG 的开关，静态文件的位置等。

Django 环境搭建

以下方法中任何一种方法安装都可，不用每个都试一次，建议自行安装 bpython，这样在用起来会爽很多。

最后面讲了如何 使用virtualenv实现多个互不干扰的开发环境。

一，Linux用自带源进行安装

1.1 ubuntu 下安装 django

```
sudo apt-get install python-django -y
```

1.2 Fedora 下安装用 yum

```
yum install python-django
```

二. 用 pip 来安装

2.1 需要先安装pip

(1). ubuntu:

```
sudo apt-get install python-pip
```

(2). Fedora:

```
yum install python-pip
```

(3). Linux, Mac OSX, Windows 下都可用 get-pip.py 来安装 pip：https://pip.pypa.io/en/latest/installing.html

或者直接下载：get-pip.py 然后运行在终端运行 `python get-pip.py` **就可以安装** `pip`。

Note: 也可以下载 pip 源码包，运行 **python setup.py install** 进行安装

2.2 利用 pip 安装 Django

```
(sudo) pip install Django
```

或者 `(sudo) pip install Django==1.6.10` 或者 `pip install Django==`

Windows 用户不要加 sudo

如果提示 **'python'不是内部或外部命令，也不是可运行的程序或批处理文件。**

那说明你的 Python 没有安装好，或者环境变量没有配置正确，最简单的办法是安装新版本的 Python 2.7.9, 里面集成了 pip，安装时要勾选上环境变量这一个

还可以参见：**Python 环境搭建**

三. 下载源码安装

https://www.djangoproject.com/download/

如果是源码包, 比如 django-1.7.6.tar.gz

3.1 Linux 或 Mac 下

```
tar -xvzf django-1.7.6.tar.gz
cd django-1.7.6
(sudo) python setup.py install
```

3.2 Windows 下

直接用解压软件解压，然后到命令行（XP/Win7点击**开始**，在下面的那个输入框中输入 cmd, Win8在开始那里点右键，选择命令行)

比如在 **D:\django-1.7.6** 这个文件夹下

```
cd D:
cd django-1.7.6
python setup.py install
```

什么？提示 **'python'不是内部或外部命令，也不是可运行的程序或批处理文件。**

那说明你的 Python 没有安装好，或者路径没有配置正确，参见：Python 环境搭建

检查是否安装成功

终端上输入 python ,点击 Enter，进行 python 环境

```
>>> import django
>>> django.VERSION
(1, 7, 6, 'final', 0)
>>>
>>> django.get_version()
'1.7.6'
```

如果运行后看到版本号，就证明安装成功了，有问题请评论!

扩展：搭建多个互不干扰的开发环境

我们会遇到这样的情况，有的项目需要 Django 1.5, 有的项目需要 Django 1.8，每个项目依赖不同，但是还要运行在同一个电脑或服

务器上，应该怎么办呢?

用 virtualenv 可以做到！参见 Python 三大神器第二部分：http://www.ziqiangxuetang.com/python/pip-virtualenv-fabric.html

---

Django 基本命令

本节主要是为了让您了解一些django最基本的命令，**请尝试着记住它们，并且多多练习下**

1. 新建一个 django-project

```
django-admin.py startproject project-name
```

一个 project 一般为一个项目

2. 新建 app

```
python manage.py startapp app-name
或 django-admin.py startapp app-name
```

一般一个项目有多个app, 当然通用的app也可以在多个项目中使用。

3. 同步数据库

```
python manage.py syncdb
```

注意：Django 1.7.1及以上的版本需要用以下命令

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

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。

备注：对已有的 models 进行修改，Django 1.7之前的版本的

Django都是无法自动更改表结构的，不过有第三方工具 south,详见 Django 数据库迁移 一节。

4. 使用开发服务器

```
python manage.py runserver

# 当提示端口被占用的时候，可以用其它端口：
python manage.py runserver 8001
python manage.py runserver 9999

# 监听所有可用 ip
python manage.py runserver 0.0.0.0:8000
# 如果是外网或者局域网电脑上可以用其它电脑查看开发服务器
# 访问对应的 ip加端口，比如 http://172.16.20.2:8000
```

5. 清空数据库

```
python manage.py flush
```

此命令会询问是 yes 还是 no, 选择 yes 会把数据全部清空掉，只留下空表。

6.创建超级管理员

```
python manage.py createsuperuser
```

7. 导出数据 导入数据

```
python manage.py dumpdata appname > appname.json
python manage.py loaddata appname.json
```

关于数据操作 详见：数据导入数据迁移，现在了解有这个用法就可以了。

8. django 项目环境终端

```
python manage.py shell
```

如果你安装了 bpython 或 ipython 会自动用它们的界面，强烈推荐用 bpython

9. 数据库命令行

```
python manage.py dbshell
```

Django 会自动进入在settings.py中设置的数据库，如果是 MySQL 或 postgreSQL,会要求输入数据库用户密码。

在这个终端可以执行数据库的SQL语句。如果您对SQL比较熟悉，可能喜欢这种方式。

10. 更多命令

终端上输入 `python manage.py` 可以看到详细的列表，在忘记了名称的时候特别有

更详细的介绍，点击对应版本去官网查看： 1.6 1.7 dev

Django 视图与网址

Django中网址是写在 urls.py 文件中，用正则表达式对应 views.py 中的一个函数(或者generic类),我们用一个项目来演示。

下载本节所有源代码：

学习编程最好的办法就是动手敲代码，请按照教程做，本节很简单，不提供源代码了，动手开始吧！

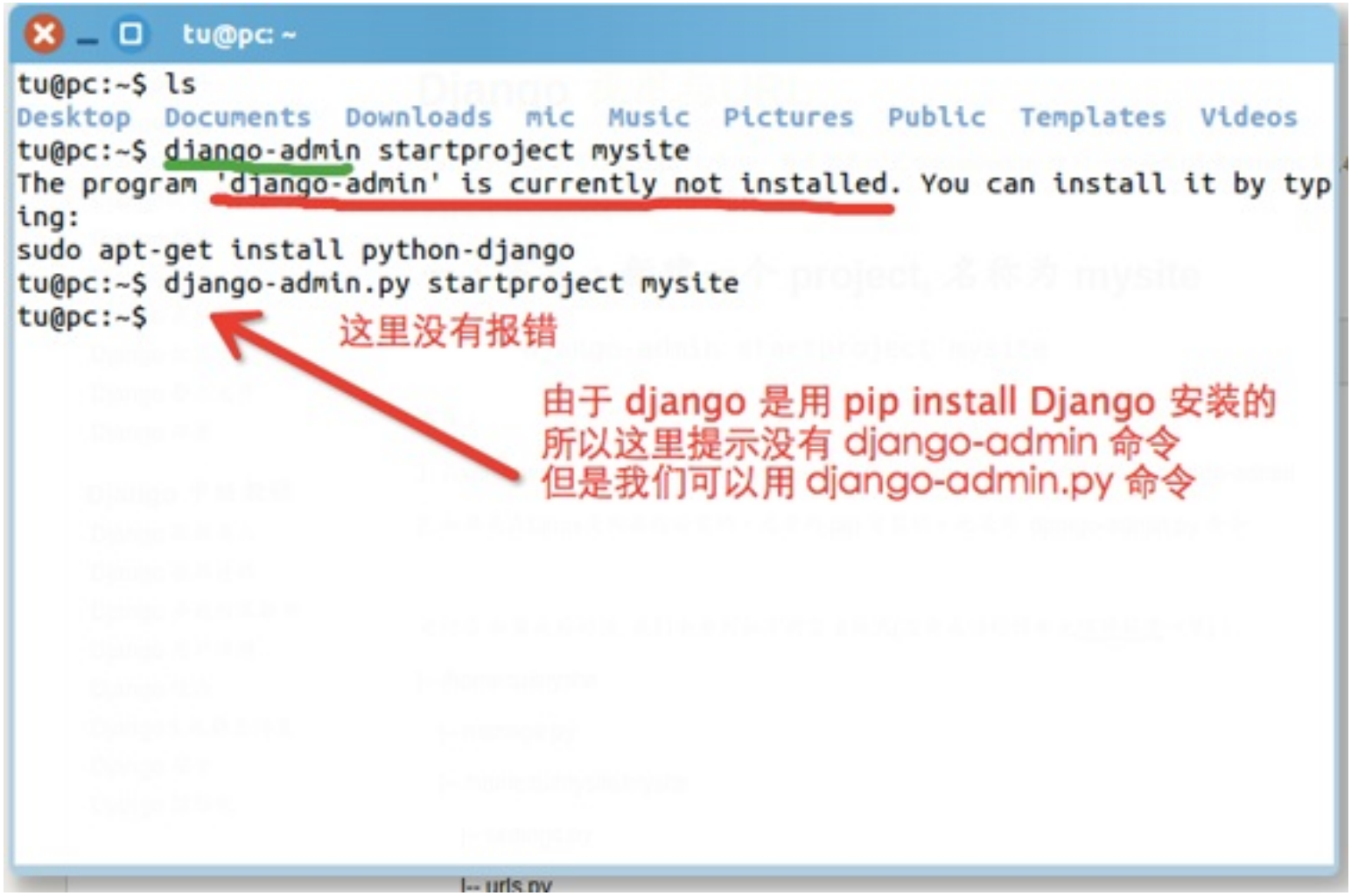
一，首先，新建一个项目(project), 名称为 mysite

```
django-admin startproject mysite
```

备注：

1. 如果 django-admin 不行，请用 django-admin.py

2. 如果是在Linux是用源码安装的，或者用 pip 安装的，也是用 django-admin.py 命令



运行后,如果成功的话, 我们会看到如下的目录样式 (没有成功的请参见环境搭建一节):

```
mysite
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

我们会发现执行命令后，新建了一个 **mysite** 目录，其中还有一个 **mysite** 目录，这个子目录 mysite 中是一些项目的设置 **settings.py** 文件，总的urls配置文件 **urls.py** 以及部署服务器时用到的 **wsgi.py** 文件， **__init__.py** 是python包的目录结构必

须的，与调用有关。

我们到外层那个 mysite 目录下(不是mysite中的mysite目录)

二, 新建一个应用(app), 名称叫 learn

```
python manage.py startapp learn # learn 是一个app的名称
```

我们可以看到mysite中多个一个 learn 文件夹，其中有以下文件

```
learn/
├── __init__.py
├── admin.py
├── models.py
├── tests.py
└── views.py
```

把我们新定义的app加到settings.py中的INSTALL_APPS中

修改 mysite/mysite/settings.py

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

    'learn',
)
```

备注,这一步是干什么呢? 新建的 app 如果不加到 INSTALL_APPS 中的话, django 就不能自动找到app中的**模板文件**(app-name/templates/下的文件)和**静态文件**(app-name/static/中的

文件)，后面你会学习到它们分别用来干什么.

定义视图函数

我们在learn这个目录中,把views.py打开,修改其中的源代码,改成下面的

```
#coding:utf-8
from django.http import HttpResponse

def index(request):
    return HttpResponse(u"欢迎光临 自强学堂!")
```

第一行是声明编码为utf-8, 因为我们在代码中用到了中文,如果不声明就报错.

第二行引入HttpResponse，它是用来向网页返回内容的，就像Python中的 print 一样，只不过 HttpResponse 是把内容显示到网页上。

我们定义了一个**index()**函数，第一个参数必须是**request**,，与网页发来的请求有关，可以包含**get**或**post**的内容,函数返回一行字到网页。

那我们访问什么网址才能看到刚才写的这个函数呢？怎么让网址和函数关联起来呢?

定义视图函数相关的URL(网址)

我们打开 mysite/mysite/urls.py 这个文件, 修改其中的代码:

```
from django.conf.urls import patterns, include, url
```

```
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^$', 'learn.views.index', name='home'),# Notice this
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
)
```

以上都修改并保存后,我们来看一下效果!

在终端上运行 python manage.py runserver 我们会看到类似下面的信息:

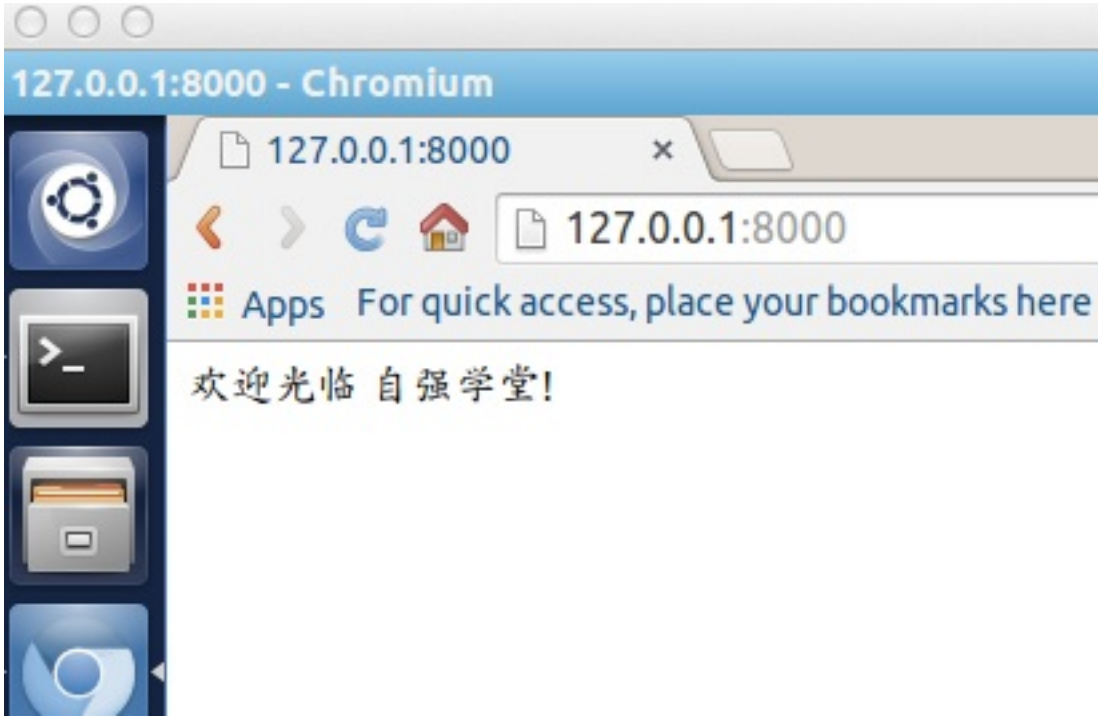
```
$ python manage.py runserver

Validating models...

0 errors found
May 24, 2014 - 10:22:14
Django version 1.6.5, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

注意：如果是在另一台电脑上访问要用 python manage.py ip:port 的形式，比如监听所有ip:

```
python manage.py runserver 0.0.0.0:8000
```
监听机器上所有ip 8000端口，访问时用电脑的ip代替 127.0.0.1

我们打开浏览器,访问 http://127.0.0.1:8000/

不出意料的话会看到:



Django中的urls.py用的是正则进行匹配的，如果不熟悉，您可以学习正则表达式以及Python正则表达式。

Django视图与网址进阶

一、在网页上做加减法

源代码下载：zqxt_views.zip

1. 采用 /add/?a=4&b=5 这样GET方法进行

```
django-admin.py startproject zqxt_views
cd zqxt_views
python manage.py startapp calc
```

自动生成目录如下所示：

```
zqxt_views/
├── calc
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt_views
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

我们修改一下 calc/views.py文件

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

```
def add(request):
    a = request.GET['a']
    b = request.GET['b']
    c = int(a)+int(b)
    return HttpResponse(str(c))
```

接着修改 zqxt_views/urls.py 文件，添加一个网址来对应我们刚才新建的视图函数

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^add/$', 'calc.views.add', name='add'), # 注意修改了这-
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
)
```

我们打开开发服务器并访问

```
python manage.py runserver
```

如果提示 Error: That port is already in use.在后面加上端口号8001,88

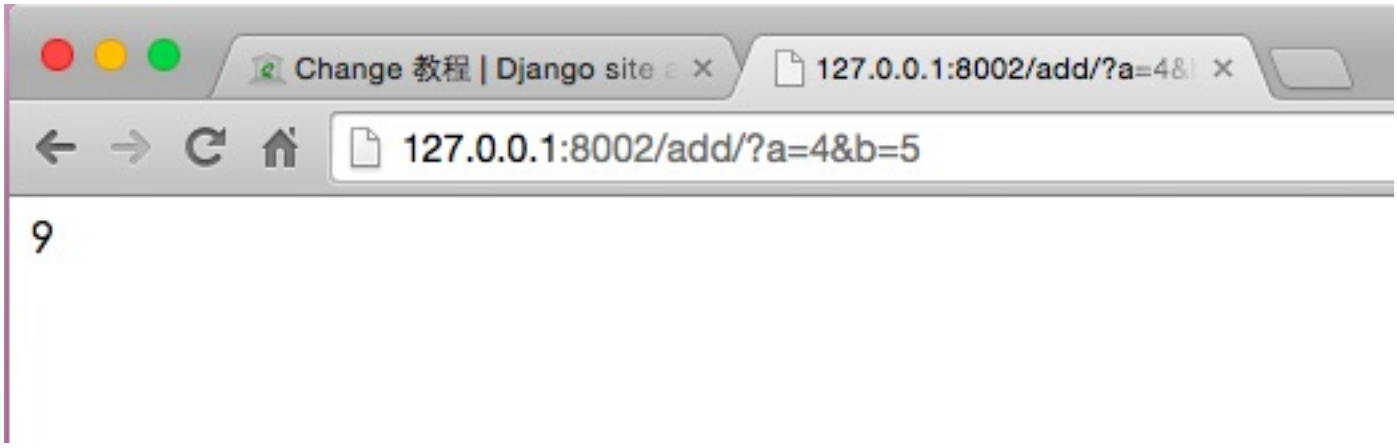
```
python manage.py runserver 8001
```

打开网址：http://127.0.0.1:8000**/add/** 就可以看到

MultiValueDictKeyError at /add/

这是因为我们并没有传值进去，我们在后面加上 **?a=4&b=5**，即访问 **http://127.0.0.1:8000/add/?a=4&b=5**

就可以看到网页上显示一个 9，试着改变一下a和b对应的值试试看?



2. 采用 /add/3/4/ 这样的网址的方式

前面介绍的时候就说过 Django 支持优雅的网址

我们接着修改 calc/views.py文件，再新定义一个add2 函数，原有部分不再贴出

```
def add2(request,a,b):
    c = int(a) + int(b)
    return HttpResponse(str(c))
```

接着修改 zqxt_views/urls.py 文件，再添加一个新的 url

```
    url(r'^add/(\d+)/(\d+)/$', 'calc.views.add2', name='add2'),
```

我们可以看到网址中多了 (\d+)，正则表达式中 \d 代表一个数字，+ 代表一个或多个前面的字符，写在一起 \d+ 就是一个或多个数字，用括号括起来的意思是保存为一个**子组**（更多知识请参见 Python 正则表达式），**每一个子组都作为一个参数被** views.py **中的函数接收**。

我们再访问 http://127.0.0.1:8000/add/4/5/ 就可以看到和刚才同样的效果，但是这回网址更优雅了

二、url 中的 name

我们还有刚才的代码，再来看一下 urls.py 中的代码

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Examples:
    url(r'^add/$', 'calc.views.add', name='add'),
    url(r'^add/(\d+)/(\d+)/$', 'calc.views.add2', name='add2'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
)
```

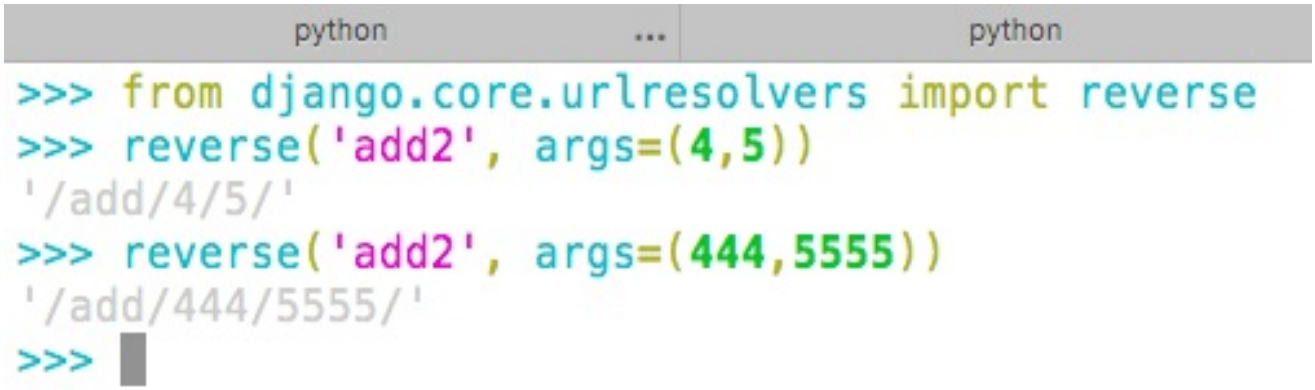
url(r'^add/$', 'calc.views.add', **name='add'**), 这里的 **name='add'** 是用来干什么的呢?

我们在开发的时候，刚开始想用的是 /add/4/5/ ，**后来需求发生变化**，比如我们又想改成 /4_add_5/这样的格式，但是我们在网页中，代码中很多地方都写的是 /add/4/5/，**这样就导致每个地方都要改，修改的代价很大，一不小心，有的地方没改过来，就不能用了。**

那么有没有更优雅的方式来解决这个问题呢? 当然答案是肯定的:

我们在终端上输入(推荐安装 bpython, 这样Django会用 bpython 的 shell)

```
python manage.py shell
```



reverse 接收 url 中的 name 作为第一个参数，我们在代码中就可以通过 reverse() 来获取对应的网址（这个网址可以用来跳转，也可以用来计算相关页面的地址），只要对应的 url 的name不改，就不用改代码中的网址。

在网页模板中也是一样，可以很方便的使用。

不带参数的：

```
{% url 'name' %}
```

带参数的：参数可以是变量名

```
{% url 'name' 参数 %}
```

```
<a href="{% url 'add2' 4 5 %}">link</a>
```

上面的代码渲染成最终的页面是

```
<a href="/add/4/5/">link</a>
```

这样就可以通过 `{% url 'add2' 4 5 %}` 获取到对应的网址 `/add/4/5/`

当 urls.py 进行更改，前提是不改 name（这个参数设定好后不要轻易改），获取的网址也会动态地跟着变，比如改成：

```
url(r'^new_add/(\d+)/(\d+)/$', 'calc.views.add2', name='add2'),
```

这时 {% url 'add2' 4 5 %} 就会渲染对应的网址成 /new_add/4/5/, reverse 函数也是一样会获取新的网址，这样改网址时只需要改 urls.py 中的正则表达式（url 参数第一部分），其它地方都“自动”跟着变了，是不是这样更好呢?

开始可能觉得直接写网址更简单，但是用多了你一定会发现，用“死网址”的方法很糟糕。

Django 模板

在前面的几节中我们都是用简单的 django.http.HttpResponse 来把内容显示到网页上，本节将讲解如何使用渲染模板的方法来显示内容。

本节代码是基于 Django 1.8，但Django 1.6，1.7也是通用的，操作流程也是一样的。

1. 创建一个 zqxt_tmpl 项目，和一个 名称为 learn 的应用，并且

```
django-admin.py startproject zqxt_tmpl
cd zqxt_tmpl
python manage.py startapp learn
```

2. 把 learn 加入到 settings.INSTALLED_APPS中

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

    'learn',
)
```

3. 打开 learn/views.py 写一个首页的视图

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')
```

4. 在 learn目录下新建一个 templates 文件夹，里面新建一个 home.html

默认配置下，Django 的模板系统会自动找到app下面的templates文件夹中的模板文件。

目录的结构是这样的：

```
zqxt_tmpl
├── learn
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── templates
│   │   └── home.html
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt_tmpl
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

4 directories, 12 files
```

5. 在 home.html 中写一些内容

```
<!DOCTYPE html>
<html>
<head>
    <title>欢迎光临</title>
</head>
<body>
欢迎光临自强学堂
</body>
</html>
```

6. 将视图函数对应到网址，更改 zqxt_tmpl/urls.py

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

urlpatterns = [
    url(r'^$', 'learn.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
]
```

7. 同步一下数据库

```
python manage.py syncdb
```

8. 运行开发服务器，看看效果

```
python manage.py runserver
```

图和前面有一节的一样，不在放上来了，把代码放上来 
zqxt_tmpl.zip

模板中的一些循环，条件判断，标签，过滤器等使用请看下一节的内容。

补充：

网站模板的设计，一般的，我们做网站有一些通用的部分，比如 **导航，底部，访问统计代码等等**

nav.html, bottom.html, tongji.html

可以写一个 base.html 来包含这些通用文件（include)

```
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}默认标题{% endblock %} - 自强学堂</tit
</head>
<body>

{% include 'nav.html' %}

{% block content %}
<div>这里是默认内容，所有继承自这个模板的，如果不覆盖就显示这里的默认内容。
{% endblock %}

{% include 'bottom.html' %}

{% include 'tongji.html' %}

</body>
</html>
```

如果需要，写足够多的 **block** 以便继承的模板可以重写该部分，**include** 是包含其它文件的内容，就是把一些网页共用的部分拿出来，重复利用，改动的时候也方便一些，还可以把广告代码放在一个单独的html中，改动也方便一些，在用到的地方include进去。其它的页面继承自 **base.html** 就好了，继承后的模板也可以在 block 块中 include 其它的模板文件。

比如我们的首页 home.html，继承或者说扩展(extends)原来的 **base.html**，可以简单这样写，重写部分代码（默认值的那一部分不用改）

```
{% extends 'base.html' %}

{% block title %}欢迎光临首页{% endblock %}

{% block content %}
{% include 'ad.html' %}
这里是首页，欢迎光临
```

```
{% endblock %}
```

注意：模板一般放在app下的templates中，Django会自动找到。假如我们每个app都有一个 index.html，当我们在views.py中使用的时候，如何判断是当前 app 的 home.html 呢?

这就需要把每个app中的 templates 文件夹中再建一个 app 的名称，仅和该app相关的模板放在 app/templates/app/ 目录下面，

例如：项目 zqxt 有两个 app，分别为 tutorial 和 tryit

```
zqxt
├── tutorial
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── templates
│   │   └── tutorial
│   │       ├── index.html
│   │       └── search.html
│   ├── tests.py
│   └── views.py
├── tryit
│   ├── __init__.py
│   ├── admin.py
│   ├── models.py
│   ├── templates
│   │   └── tryit
│   │       ├── index.html
│   │       └── poll.html
│   ├── tests.py
│   └── views.py
├── manage.py
└── zqxt
    ├── __init__.py
    ├── settings.py
```

```
    ├── urls.py
    └── wsgi.py
```

这样，使用的时候就是 "tutorial/index.html" 和 "tryit/index.html" 这样有app作为名称的一部分，就不会混淆。

模板中的一些循环，条件判断，标签，过滤器等使用请看Django 模板进阶。

Django 模板进阶

本节主要讲 **Django模板中的循环，条件判断，常用的标签，过滤器的使用。**

1. 列表，字典，类的实例的使用
2. 循环：迭代显示列表，字典等中的内容
3. 条件判断：判断是否显示该内容，比如判断是手机访问，还是电脑访问，给出不一样的代码。
4. **标签**：for，if 这样的功能都是标签。
5. **过滤器**：管道符号后面的功能，比如{{ var|length }}，求变量长度的 length 就是一个过滤器。

实例一，显示一个基本的字符串在网页上

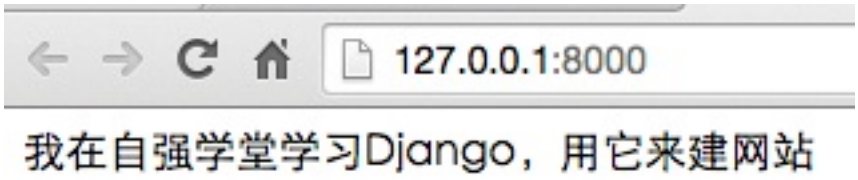
views.py

```
# -*- coding: utf-8 -*-
from django.shortcuts import render


def home(request):
    string = u"我在自强学堂学习Django，用它来建网站"
    return render(request, 'home.html', {'string': string})
```

在视图中我们传递了一个字符串名称是 string 到模板 home.html，在模板中这样使用它：

home.html

```
{{ string }}
```



实例二，讲解了基本的 for 循环 和 List内容的显示

views.py

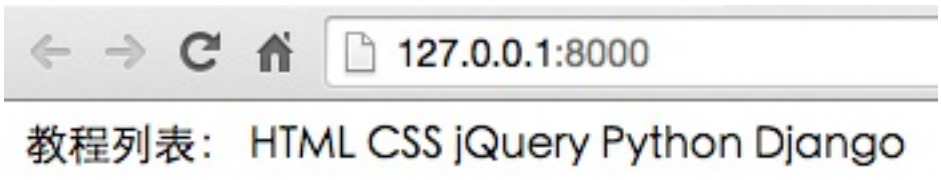
```
def home(request):
    TutorialList = ["HTML", "CSS", "jQuery", "Python", "Django"
    return render(request, 'home.html', {'TutorialList': Tutori
```

在视图中我们传递了一个List到模板 home.html，在模板中这样使用它：

home.html

```
教程列表：
{% for i in TutorialList %}
{{ i }}
{% endfor %}
```

for 循环要有一个结束标记，上面的代码假如我们对应的是首页的网址（自己修改urls.py），显示在网页上就是：



简单总结一下：一般的变量之类的用{{ }}（变量），功能类的，比如循环，条件判断是用{% %}（标签）

实例三，显示字典中内容：

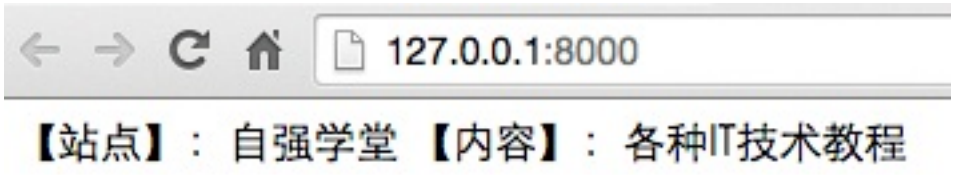
views.py

```
def home(request):
    info_dict = {'site': u'自强学堂', 'content': u'各种IT技术教程'
    return render(request, 'home.html', {'info_dict': info_dict
```

home.html

```
站点：{{ info_dict.site }} 内容：{{ info_dict.content }}
```

在模板中取字典的键是用点info_dict.site，而不是Python中的info_dict['site']，效果如下：



还可以这样遍历字典：

```
{% for key, value in info_dict.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

其实就是遍历这样一个 List：`[('content', u'自强学堂'), ('site', u'各种IT技术教程')]`

实例四，在模板进行 条件判断和 for 循环的详细操作：

views.py

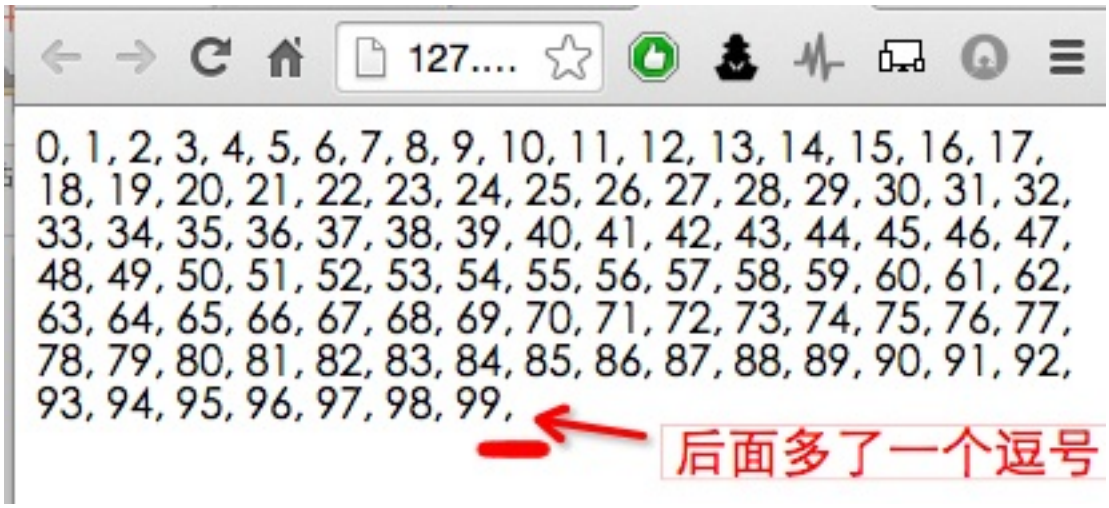
```
def home(request):
    List = map(str, range(100))# 一个长度为100的 List
    return render(request, 'home.html', {'List': List})
```

假如我们想用逗号将这些元素连接起来：

home.html

```
{% for item in List %}
    {{ item }},
{% endfor %}
```

效果如下：



我们会发现最后一个元素后面也有一个逗号，这样肯定不爽，如果判断是不是遍历到了最后一个元素了呢?

用变量 forloop.last 这个变量，如果是最后一项其为真，否则为假，更改如下：

```
{% for item in List %}
```

```
    {{ item }}{% if not forloop.last%},{% endif %}
{% endfor %}
```

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99

forloop.last 判断
不是最后一项就加逗号（，）
是最后一项就不加

在for循环中还有很多有用的东西，如下：

变量	描述
forloop.counter	索引从 1 开始算
forloop.counter0	索引从 0 开始算
forloop.revcounter	索引从最大长度到 1
forloop.revcounter0	索引从最大长度到 0
forloop.first	当遍历的元素为第一项时为真
forloop.last	当遍历的元素为最后一项时为真
forloop.parentloop	用在嵌套的 for 循环中，获取上一层 for 循环的 forloop

当列表中可能为空值时用 for empty

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
```

```
    <li>抱歉，列表为空</li>
{% endfor %}
</ul>
```

实例五，模板上得到视图对应的网址：{% url "view-name" arg1 arg2 %}

```
# urls.py
urlpatterns = patterns('',
    url(r'^add/(\d+)/(\d+)/$', 'calc.views.add', name='add'),
)

# template html
{% url 'add' 4 5 %}
```

这样网址上就会显示出：/add/4/5/ 这个网址，假如我们以后修改 urls.py 中的

```
r'^add/(\d+)/(\d+)/$'
```

这一部分，改成另的，比如：

```
r'^jiafa/(\d+)/(\d+)/$'
```

这样，我们不需要再次修改模板，当再次访问的时候，网址会自动变成 /jiafa/4/5/

还可以使用 as 语句将内容取别名（相当于定义一个变量），多次使用（但视图名称到网址转换只进行了一次）

```
{% url 'some-url-name' arg arg2 as the_url %}

<a href="{{ the_url }}">链接到：{{ the_url }}</a>
```

实例六，模板中的逻辑操作：

6.1、 ==, !=, >=, <=, >, < 这些比较都可以在模板中使用，

比如：

```
{% if var >= 90 %}
成绩优秀，自强学堂你没少去吧！学得不错
{% elif var >= 80 %}
成绩良好
{% elif var >= 70 %}
成绩一般
{% elif var >= 60 %}
需要努力
{% else %}
不及格啊，大哥！多去自强学堂学习啊！
{% endif %}
```

and, or, not, in, not in 也可以在模板中使用

假如我们判断 num 是不是在 0 到 100 之间：

```
{% if num <= 100 and num >= 0 %}
num在0到100之间
{% else %}
数值不在范围之内！
{% endif %}
```

假如我们判断 'ziqiangxuetang' 在不在一个列表变量 List 中：

```
{% if 'ziqiangxuetang' in List %}
自强学堂在名单中
{% endif %}
```

完整的内容参官方文档：https://docs.djangoproject.com/en/1.8/ref/templates/b

---

Django渲染json到模板

有时候我们想把一个 list 或者 dict 传递给 javascript，处理后显示到网页上。

这里讲述两种方法：

一，页面加载完成后，在页面上操作，在页面上通过 ajax 方法得到新的数据并显示在网页上，这种情况适用于动态加载一些内容，但是我们不知道用户需要哪个，比如用户输入一个值或者点击某个地方，动态地把内容显示在网页上，详见：Django Ajax

二，直接在视图函数（views.py中的函数）中渲染一个 list 或 dict 和其它的网页部分一起显示到网页上

需要注意的是，我们如果直接这么做，传递到 js 的时候，网页的内容会被转义，得到的格式会报错。

views.py

```
from __future__ import unicode_literals
from django.shortcuts import render

def home(request):
    List = ['自强学堂', '渲染Json到模板']
    return render(request, 'home.html', {'List': List})
```

home.html 中的一部分

```
<script type="text/javascript">
    var List = {{ List }};
    alert(List);
</script>
```

访问时会得到 Uncaught SyntaxError: Unexpected token ILLEGAL

需要注意两点：1. views.py中返回的函数中的值要用 `json.dumps()`处理，2. 在网页上要加一个 safe 过滤器。

views.py

```
# -*- coding: utf-8 -*-

from __future__ import unicode_literals

import json
from django.shortcuts import render

def home(request):
    List = ['自强学堂', '渲染Json到模板']
    Dict = {'site': '自强学堂', 'author': '涂伟忠'}
    return render(request, 'home.html', {
            'List': json.dumps(List),
            'Dict': json.dumps(Dict)
        })
```

home.html 只给出了 js 核心部分：

```
//列表
var List = {{ List|safe }};
//字典
var Dict = {{ Dict|safe }};
```

如果你对 js 比较熟悉，至此为止，下面的不用于看了，如果不太熟悉，可以参考下面的更详细的代码。

html 完全代码及完整代码下载（最后面）：

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>欢迎光临 自强学堂! </title>
<script src="http://apps.bdimg.com/libs/jquery/1.10.2/jquery.mi
</head>
<body>
<div id="list"> 学习 </div>
<div id='dict'></div>
<script type="text/javascript">
    //列表
    var List = {{ List|safe }};

    //下面的代码把List的每一部分放到头部和尾部
    $('#list').prepend(List[0]);
    $('#list').append(List[1]);

    console.log('--- 遍历 List 方法 1 ---')
    for(i in List){
        console.log(i);// i为索引
    }

    console.log('--- 遍历 List 方法 2 ---')
    for (var i = List.length - 1; i >= 0; i--) {
        // 鼠标右键，审核元素，选择 console可以看到输入的值
        console.log(List[i]);
    };

    console.log('--- 同时遍历索引和内容，使用 jQuery.each() 方法 ---
    $.each(List, function(index, item){
        console.log(index);
        console.log(item);
    });


    // 字典
    var Dict = {{ Dict|safe }};
    console.log("--- 两种字典的取值方式  ---")
    console.log(Dict['site']);
    console.log(Dict.author);

    console.log("---  遍历字典  ---");
    for(i in Dict) {
        console.log(i + Dict[i]);//注意，此处 i 为键值
```

```
    }
</script>
</body>
</html>
```

完整代码下载：zqxt_json.zip （基于Django 1.8，注意settings.py文件和低版本可能不兼容，其它部分相同）

Django 模型（数据库）

Django 模型是与数据库相关的，与数据库相关的代码一般写在 models.py 中，Django 支持 sqlite3, MySQL, PostgreSQL等数据库，只需要在settings.py中配置即可，不用更改models.py中的代码，丰富的API极大的方便了使用。

本节的代码：（Django 1.6, Python 2.7 测试环境）

learn_models.zip

大家可以按照我的步骤来开始做：

```
django-admin.py startproject learn_models # 新建一个项目
cd learn_models # 进入到该项目的文件夹
django-admin.py startapp people # 新建一个 people 应用（app）
```

补充：新建app也可以用 python manage.py startapp people，需要指出的是，django-admin.py 是安装Django后多出的一个命令，并不是指一个 django-admin.py 脚本在当前目录下。

那么project和app什么关系呢，一个项目一般包含多个应用，一个应用也可以用在多个项目中。

将我们新建的应用（people）添加到 settings.py 中的 INSTALLED_APPS中，也就是告诉Django有这么一个应用。

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
```

```
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'people',
)
```

我们打开 **people/models.py** 文件，修改其中的代码如下：

```
from django.db import models

class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()
```

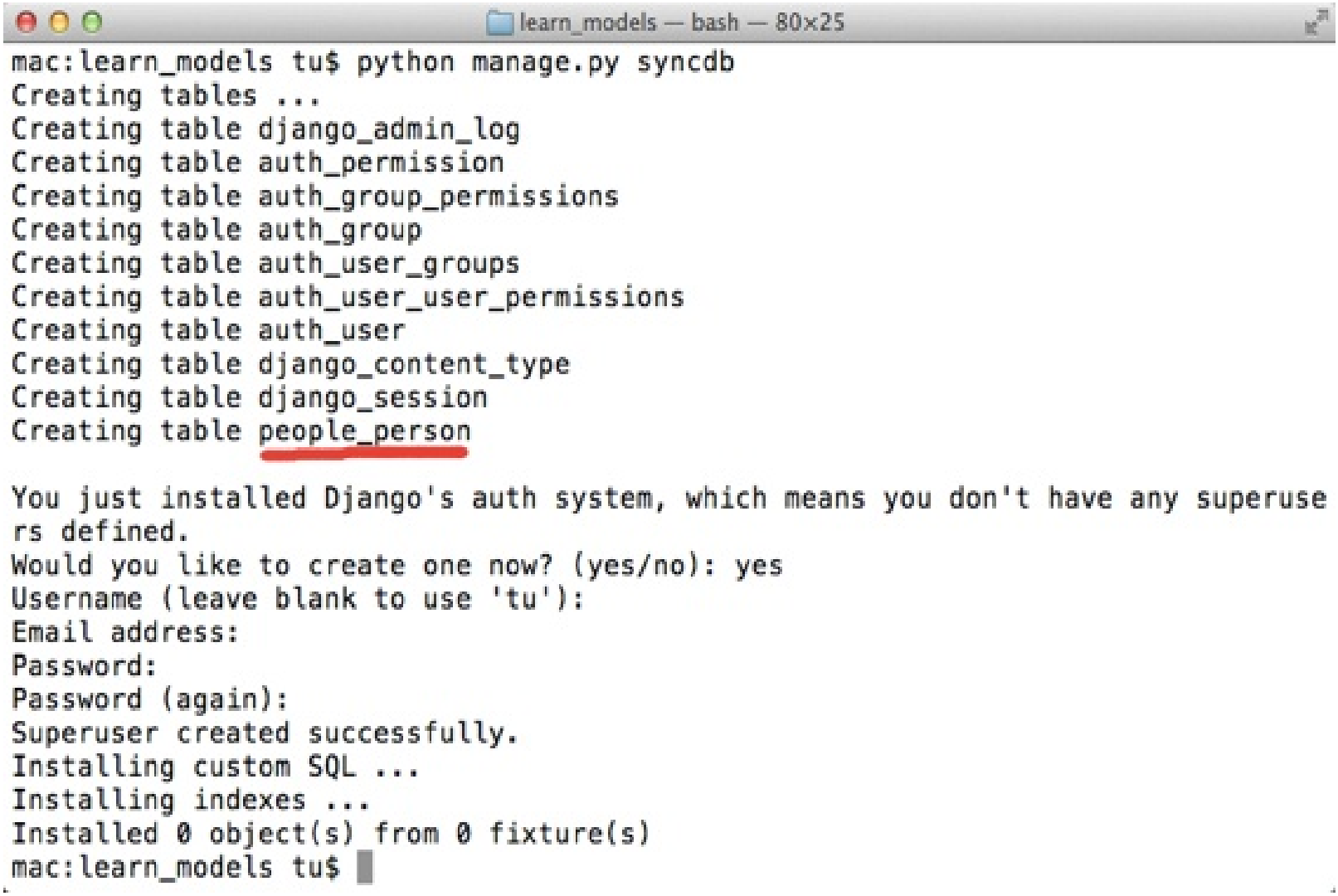
我们新建了一个Person类，继承自models.Model, 一个人有姓名和年龄。这里用到了两种**Field**，更多Field类型可以参考教程最后的链接。

我们来同步一下数据库

```
python manage.py syncdb # 进入 manage.py 所在的那个文件夹下输入这个命
```

注意：Django 1.7 及以上的版本需要用以下命令

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



我们会看到，Django生成了一系列的表，也生成了我们新建的people_person这个表，那么如何使用这个表呢?

Django提供了丰富的API，下面演示如何使用它。

```
$ python manage.py shell

>>> from people.models import Person
>>> Person.objects.create(name="WeizhongTu", age=24)
<Person: Person object>
>>>
```

我们新建了一个用户WeizhongTu 那么如何从数据库是查询到它呢?

```
>>> Person.objects.get(name="WeizhongTu")
<Person: Person object>
```

>>>

我们用了一个 .objects.get() 方法查询出来符合条件的对象，但是大家注意到了没有，查询结果中显示<Person: **Person object**>，这里并没有显示出与WeizhongTu的相关信息，如果用户多了就无法知道查询出来的到底是谁，查询结果是否正确，我们重新修改一下 people/models.py

name 和 age 等字段中不能有 __（双下划线，因为在Django QuerySet API中有特殊含义（用于关系，包含，不区分大小写，以什么开头或结尾，日期的大于小于，正则等）

也不能有Python中的关键字，name 是合法的，student_name 也合法，但是student__name不合法，try, class, continue 也不合法，因为它是Python的关键字(import keyword; print(keyword.kwlist) 可以打出所有的关键字)

```
from django.db import models


class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()

    def __unicode__(self):
    # 在Python3中使用 def __str__(self)
        return self.name
```

按CTRL + C退出当前的Python shell, 重复上面的操作，我们就可以看到:

新建一个对象的方法有以下几种：

1. Person.objects.create(name=name,age=age)

2. p = Person(name="WZ", age=23)

   p.save()

3. p = Person(name="TWZ")

   p.age = 23

   p.save()

4. Person.objects.get_or_create(name="WZT", age=23)

   这种方法是防止重复很好的方法，但是速度要相对慢些，返回一个元组，第一个为Person对象，第二个为True或False, 新建时返回的是True, 已经存在时返回False.

获取对象有以下方法：

1. Person.objects.all()

2. Person.objects.all()[:10] 切片操作，获取10个人，不支持负索引，切片可以节约内存

3. Person.objects.get(name=name)

   get是用来获取一个对象的，如果需要获取满足条件的一些人，就要用到filter

4. Person.objects.filter(name="abc") # 等于 Person.objects.filter(name__exact="abc") 名称严格等于 "abc" 的人

5. Person.objects.filter(name__iexact="abc") # 名称为 abc 但是不区分大小写，可以找到 ABC, Abc, aBC，这些都符合条件

6. Person.objects.filter(name__contains="abc") # 名称中包含 "abc"的人

7. Person.objects.filter(name__icontains="abc") #名称中包含 "abc"，且abc不区分大小写

8. Person.objects.filter(name__regex="^abc") # 正则表达式查询

9. Person.objects.filter(name__iregex="^abc")# 正则表达式不区分大小写

   filter是找出满足条件的，当然也有排除符合某条件的

10. Person.objects.exclude(name__contains="WZ") # 排除包含 WZ 的Person对象

11. Person.objects.filter(name__contains="abc").exclude(ag # 找出名称含有abc, 但是排除年龄是23岁的

参考文档:

Django models 官方教程: https://docs.djangoproject.com/en/dev/topics/db/mode

Fields相关官方文档: https://docs.djangoproject.com/en/dev/ref/models/fielc

---

Django 自定义Field

Django 的官方提供了很多的 Field，但是有时候还是不能满足我们的需求，不过Django提供了自定义 Field 的方法：

提示：如果现在用不到可以跳过这一节，不影响后面的学习，等用到的时候再来学习不迟。

来一个简单的例子吧。

1. 减少文本的长度，保存数据的时候压缩，读取的时候解压缩，如果发现压缩后更长，就用原文本直接存储：

```
class CompressedTextField(models.TextField):
    """    model Fields for storing text in a compressed format
    __metaclass__ = models.SubfieldBase

    def to_python(self, value):
        if not value:
            return value

        try:
            return value.decode('base64').decode('bz2').decode(
        except Exception:
            return value

    def get_prep_value(self, value):
        if not value:
            return value

        try:
            value.decode('base64')
            return value
        except Exception:
```

```
            try:
                tmp = value.encode('utf-8').encode('bz2').encod
            except Exception:
                return value
            else:
                if len(tmp) > len(value):
                    return value

                return tmp
```

to_python 函数用于转化数据库中的字符到 Python的变量，get_prep_value 用于将Python变量处理后(此处为压缩）保存到数据库，使用和Django自带的 Field 一样。

2. 比如我们想保存一个 列表到数据库中，在读取用的时候要是Python的列表的形式，我们来自己写一个 ListField：

这个ListField继承自 TextField，代码如下：

```
from django.db import models
import ast

class ListField(models.TextField):
    __metaclass__ = models.SubfieldBase
    description = "Stores a python list"

    def __init__(self, *args, **kwargs):
        super(ListField, self).__init__(*args, **kwargs)

    def to_python(self, value):
        if not value:
            value = []

        if isinstance(value, list):
            return value

        return ast.literal_eval(value)
```

```
    def get_prep_value(self, value):
        if value is None:
            return value

        return unicode(value) # use str(value) in Python 3

    def value_to_string(self, obj):
        value = self._get_val_from_obj(obj)
        return self.get_db_prep_value(value)
```

使用它很简单，首先导入 ListField，像自带的 Field 一样使用：

```
class Article(models.Model):
    labels = ListField()
```

在终端上尝试：

```
>>> from app.models import Article
>>> d = Article()
>>> d.labels
[]
>>> d.labels = ["Python", "Django"]
>>> d.labels
["Python", "Django"]
```

示例代码：

zqxt_custom_fields_project.zip

下载上面的代码，解压，进入项目目录，输入 python manage.py shell 搞起

```
>>> from blog.models import Article

>>> a = Article()
```

```
>>> a.labels.append('Django')
>>> a.labels.append('custom fields')

>>> a.labels
['Django', 'custom fields']

>>> type(a.labels)
<type 'list'>

>>> a.content = u'我正在写一篇关于自定义Django Fields的教程'
>>> a.save()
```

参考网址：

https://djangosnippets.org/snippets/2014/

https://docs.djangoproject.com/en/dev/howto/custom-model-fields/

Django QuerySet API

上一篇我们学习了Django 模型，也学习了一些基本的创建与查询。这里专门来讲一下**数据库接口**相关的接口（QuerySet API），当然您也可以选择暂时跳过此节，如果以后用到数据库相关的时候再看也是可以的。

从数据库中查询出来的结果一般是一个集合，这个集合叫做QuerySet。

文中的例子大部分是基于这个 blog/models.py

```
from django.db import models


class Blog(models.Model):
    name = models.CharField(max_length=100)
    tagline = models.TextField()

    def __unicode__(self):  # __str__ on Python 3
        return self.name

class Author(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()

    def __unicode__(self):  # __str__ on Python 3
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog)
    headline = models.CharField(max_length=255)
    body_text = models.TextField()
    pub_date = models.DateField()
```

```
    mod_date = models.DateField()
    authors = models.ManyToManyField(Author)
    n_comments = models.IntegerField()
    n_pingbacks = models.IntegerField()
    rating = models.IntegerField()

    def __unicode__(self):  # __str__ on Python 3
        return self.headline
```

1. QuerySet 创建对象的方法

```
>>> from blog.models import Blog
>>> b = Blog(name='Beatles Blog', tagline='All the latest Beatl
>>> b.save()
```

总之，一共有四种方法

```
# 方法 1
Author.objects.create(name="WeizhongTu", email="tuweizhong@163.

# 方法 2
twz = Author(name="WeizhongTu", email="tuweizhong@163.com")
twz.save()

# 方法 3
twz = Author()
twz.name="WeizhongTu"
twz.email="tuweizhong@163.com"

# 方法 4，首先尝试获取，不存在就创建，可以防止重复
Author.objects.get_or_create(name="WeizhongTu", email="tuweizhc
# 返回值(object, True/False)
```

备注：前三种方法返回的都是对应的 object，最后一种方法返回的是一个元组，(object, True/False)，创建时返回 True，已经存在时返回 False

当有一对多，多对一，或者多对多的关系的时候，先把相关的对象查

询出来

```
>>> from blog.models import Entry
>>> entry = Entry.objects.get(pk=1)
>>> cheese_blog = Blog.objects.get(name="Cheddar Talk")
>>> entry.blog = cheese_blog
>>> entry.save()
```

2. 获取对象的方法（上一篇的部分代码）

```
Person.objects.all() # 查询所有
Person.objects.all()[:10] 切片操作，获取10个人，不支持负索引，切片可以
Person.objects.get(name="WeizhongTu") # 名称为 WeizhongTu 的一条，

get是用来获取一个对象的，如果需要获取满足条件的一些人，就要用到filter
Person.objects.filter(name="abc") # 等于Person.objects.filter(na
Person.objects.filter(name__iexact="abc") # 名称为 abc 但是不区分

Person.objects.filter(name__contains="abc") # 名称中包含 "abc"的
Person.objects.filter(name__icontains="abc") #名称中包含 "abc"，

Person.objects.filter(name__regex="^abc") # 正则表达式查询
Person.objects.filter(name__iregex="^abc")# 正则表达式不区分大小写

# filter是找出满足条件的，当然也有排除符合某条件的
Person.objects.exclude(name__contains="WZ") # 排除包含 WZ 的Perso
Person.objects.filter(name__contains="abc").exclude(age=23) # 
```

3. QuerySet 是可迭代的，比如：

```
es = Entry.objects.all()
for e in es:
    print(e.headline)
```

Entry.objects.all() 或者 es 就是 QuerySet 是查询所有的 Entry 条目。

注意事项：

(1). 如果只是检查 Entry 中是否有对象，应该用 Entry.objects.all().**exists()**

(2). QuerySet 支持切片 Entry.objects.all()[:10] 取出10条，可以节省内存

(3). 用 len(es) 可以得到Entry的数量，但是推荐用 Entry.objects.count()来查询数量，后者用的是SQL：SELECT COUNT(*)

(4). list(es) 可以强行将 QuerySet 变成 列表

4. QuerySet 是可以用pickle序列化到硬盘再读取出来的

```
>>> import pickle
>>> query = pickle.loads(s)      # Assuming 's' is the pickled s
>>> qs = MyModel.objects.all()
>>> qs.query = query             # Restore the original 'query'.
```

5. QuerySet 查询结果排序

作者按照名称排序

```
Author.objects.all().order_by('name')
Author.objects.all().order_by('-name') # 在 column name 前加一个1
```

6. QuerySet 支持链式查询

```
Author.objects.filter(name__contains="WeizhongTu").filter(email
Author.objects.filter(name__contains="Wei").exclude(email="tuwe

# 找出名称含有abc，但是排除年龄是23岁的
Person.objects.filter(name__contains="abc").exclude(age=23)
```

7. QuerySet 不支持负索引

```
Person.objects.all()[:10] 切片操作，前10条
Person.objects.all().reverse()[:2] # 最后两条
Person.objects.all().reverse()[0] # 最后一条

# 使用 order_by，在栏目名（column name）前加一个负号
Author.objects.order_by('-id')[:20] # id最大的20条
```

8. QuerySet 重复的问题，使用 .distinct() 去重

一般的情况下，QuerySet 中不会出来重复的，重复是很罕见的，但是当跨越多张表进行检索后，结果并到一起，可以会出来重复的值（我最近就遇到过这样的问题）

```
qs1 = Pathway.objects.filter(label__name='x')
qs2 = Pathway.objects.filter(reaction__name='A + B >> C')
qs3 = Pathway.objects.filter(inputer__name='WeizhongTu')

# 合并到一起
qs = qs1 | qs2 | qs3
```

这个时候就有可能出现重复的

```
# 去重方法
qs = qs.distinct()
```

最后更新 2015.04.09 未完待续。。

参考：

Django数据库操作官方文档： QuerySet

API: https://docs.djangoproject.com/en/dev/ref/models/qu

Django 后台

django的后台我们只要加少些代码，就可以实现强大的功能。

建议有总是查阅官方文档 (本文是基于django 1.6版本测试的)

与后台相关文件：每个app中的 admin.py 文件与后台相关。

本节代码下载：后台帐号 tu 密码 zqxt

 zqxt_admin.zip

下面示例是做一个后台添加博客的例子（基于django 1.6以上版本）：

一，新建一个 名称为 zqxt_admin 的项目

```
django-admin.py startproject zqxt_admin
```

二，新建一个 叫做 blog 的app

```
# 进入 zqxt_admin 文件夹
django-admin.py startapp blog
```

二，修改 blog 文件夹中的 models.py

```
# coding:utf-8
from django.db import models
```

```
class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')

    pub_date = models.DateTimeField(u'发表时间', auto_now_add=Tr
    update_time = models.DateTimeField(u'更新时间',auto_now=True
```

四，把 blog 加入到settings.py中的INSTALLED_APPS中

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

    'blog',
)
```

提示：INSTALLED_APPS 是一个元组，每次加入新的app的时候，在后面都加一个逗号，这是一个好习惯。

五，同步所有的数据表

```
# 进入包含有 manage.py 的文件夹
python manage.py syncdb
```

注意：Django 1.7及以上的版本需要用以下命令

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

可以看到：

Creating tables ...

Creating table django_admin_log

Creating table auth_permission

Creating table auth_group_permissions

Creating table auth_group

Creating table auth_user_groups

Creating table auth_user_user_permissions

Creating table auth_user

Creating table django_content_type

Creating table django_session

Creating table blog_article

You just installed Django's auth system, which means you don't have any superusers defined.

Would you like to create one now? (yes/no): yes

Username (leave blank to use 'tu'): tu

Email address:

Password:

Password (again):

Superuser created successfully.

Installing custom SQL ...

Installing indexes ...

Installed 0 object(s) from 0 fixture(s)

六，修改 admin.py

进入 blog 文件夹，修改 admin.py 文件（如果没有新建一个），内容如下

```
from django.contrib import admin
from .models import Article


admin.site.register(Article)
```

只需要这三行代码，我们就可以拥有一个强大的后台！

提示：urls.py中关于 admin的已经默认开启，如果没有，请参见：https://docs.djangoproject.com/en/dev/ref/contrib/adm

七，打开 开发服务器

```
python manage.py runserver
# 如果提示 8000 端口已经被占用，可以用 python manage.py runserver 80
```

访问 http://localhost:8000/admin/ 输入设定的帐号和密码，就

可以看到：

点击 Articles，动手输入 添加几篇文章，就可以看到：

我们会发现所有的文章都是叫 Article object，这样肯定不好，比如我们要修改，如何知道要修改哪个呢?

我们修改一下 blog 中的models.py

```
# coding:utf-8
from django.db import models

class Article(models.Model):
    title = models.CharField(u'标题', max_length=256)
    content = models.TextField(u'内容')

    pub_date = models.DateTimeField(u'发表时间', auto_now_add=Tr
    update_time = models.DateTimeField(u'更新时间',auto_now=True

    def __unicode__(self):# 在Python3中用 __str__ 代替 __unicode_
        return self.title
```

我们加了一个 __unicode__ 函数，刷新后台网页，会看到：

所以**推荐定义 Model 的时候 写一个 __unicode__ 函数(或 __str__函数)**

技能提升：如何兼容python2.x和python3.x呢?

示例如下：

```
# coding:utf-8
from __future__ import unicode_literals

from django.db import models
from django.utils.encoding import python_2_unicode_compatible

@python_2_unicode_compatible
class Article(models.Model):
    title = models.CharField('标题', max_length=256)
    content = models.TextField('内容')

    pub_date = models.DateTimeField('发表时间', auto_now_add=Tru
    update_time = models.DateTimeField('更新时间',auto_now=True,

    def __str__(self):
        return self.title
```

python_2_unicode_compatible 会自动做一些处理去适应python不同的版本，本例中的 unicode_literals 可以让python2.x 也像 python3 那个处理 unicode 字符，以便有更好地兼容性。

八，在列表显示与字段相关的其它内容

后台已经基本上做出来了，可是如果我们还需要显示一些其它的fields，如何做呢?

```
from django.contrib import admin
from .models import Article

class ArticleAdmin(admin.ModelAdmin):
    list_display = ('title','pub_date','update_time',)

admin.site.register(Article,ArticleAdmin)
```

list_display 就是来配置要显示的字段的，当然也可以显示非字段内容，或者字段相关的内容，比如：

```
class Person(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)

    def my_property(self):
        return self.first_name + ' ' + self.last_name
    my_property.short_description = "Full name of the person"

    full_name = property(my_property)
```

在admin.py中

```
class PersonAdmin(admin.ModelAdmin):
    list_display = ('full_name',)
```

到这里我们发现我们又有新的需求，比如要改 models.py 中的字段，添加一个文章的状态（草稿，正式发布），这时候我们就需要更改表，django 1.7以前的都不会自动更改表，我们需要用第三方插件 South，参见 Django 迁移数据。

Django 1.7 及以上用以下命令来同步数据库表的更改

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

其它一些常用的功能：

搜索功能：search_fields = ('title', 'content',) 这样就可以按照标题或内容搜索了

https://docs.djangoproject.com/en/dev/ref/contrib/admin/#

筛选功能：list_filter = ('status',) 这样就可以根据文章的状态去筛选，比如找出是草稿的文章

https://docs.djangoproject.com/en/dev/ref/contrib/admin/#

新增或修改时的**布局顺序**：https://docs.djangoproject.com/en/dev/ref/contrib/adm

有时候我们需要对django admin site进行修改以满足自己的需求，那么我们可以从哪些地方入手呢?

更多的docs大家参见官方的文档：https://docs.djangoproject.com/en/dev/ref/contrib/adm

以下举例说明：

1.定制加载的列表，根据不同的人显示不同的内容列表，比如输入员只能看见自己输入的，审核员能看到所有的草稿，这时候就需要重写get_queryset方法

```
class MyModelAdmin(admin.ModelAdmin):
    def get_queryset(self, request):
        qs = super(MyModelAdmin, self).get_queryset(request)
        if request.user.is_superuser:
            return qs
        else:
            return qs.filter(author=request.user)
```

该类实现的功能是如果是超级管理员就列出所有的，如果不是，就仅列出访问者自己相关的

2.定制搜索功能（django 1.6及以上才有)

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

    def get_search_results(self, request, queryset, search_term
        queryset, use_distinct = super(PersonAdmin, self).get_s
        try:
            search_term_as_int = int(search_term)
            queryset |= self.model.objects.filter(age=search_te
        except:
            pass
        return queryset, use_distinct
```

queryset 是默认的结果，search_term 是在后台搜索的关键词

3.修改保存时的一些操作，可以检查用户，保存的内容等，比如保存时加上添加人

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj.user = request.user
        obj.save()
```

其中obj是修改后的对象，form是返回的表单（修改后的），**当新建一个对象时 change = False, 当修改一个对象时 change = True**
如果需要获取修改前的对象的内容可以用

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

```
class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        obj_original = self.model.objects.get(pk=obj.pk)
        obj.user = request.user
        obj.save()
```

那么又有问题了，这里如果原来的obj不存在，也就是如果我们是新建的一个怎么办呢，这时候可以用try,except的方法尝试获取,当然更好的方法是判断一下这个对象是新建还是修改，是新建就没有obj_original，是修改就有

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        if change:# 更改的时候
            obj_original = self.model.objects.get(pk=obj.pk)
        else:# 新增的时候
            obj_original = None

        obj.user = request.user
        obj.save()
```

4, 删除时做一些处理,

```
from django.contrib import admin

class ArticleAdmin(admin.ModelAdmin):
    def delete_model(self, request, obj):
        """
        Given a model instance delete it from the database.
        """
        # handle something here
        obj.delete()
```

Django的后台非常强大，这个只是帮助大家入门，更完整的还需要

查看官方文档，如果你有更好的方法或不懂的问题，欢迎评论!

Django 表单

有时候我们需要在前台用 get 或 post 方法提交一些数据，所以自己写一个网页，用到 html 表单的知识。

第一节：源码下载 zqxt_form_learn1.zip

比如写一个计算 a和 b 之和的简单应用，网页上这么写

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>


<form action="/add/" method="get">
    a: <input type="text" name="a"> <br>
    b: <input type="text" name="b"> <br>

    <input type="submit" value="提交">
</form>


</body>
</html>
```

把这些代码保存成一个index.html，放在 templates 文件夹中。

网页的值传到服务器是通过 <input> 或 <textarea>标签中的 **name** 属性来传递的，在服务器端这么接收：

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

```
def index(request):
    return render(request, 'index.html')

def add(request):
    a = request.GET['a']
    b = request.GET['b']
    a = int(a)
    b = int(b)
    return HttpResponse(str(a+b))
```

request.GET 可以看成一个字典，用GET方法传递的值都会保存到其中，可以用 request.GET.get('key', None)来取值，没有时不报错。

再将函数和网址对应上，就可以访问了，详情参见源码。

这样就完成了基本的功能，基本上可以用了。

但是，比如用户输入的不是数字，而是字母，就出错了，还有就是提交后再回来已经输入的数据也会没了。

当然如果我们手动将输入之后的数据在 views 中都获取到再传递到网页，这样是可行的，但是很不方便，所以 Django 提供了更简单易用的 forms 来解决验证等这一系列的问题。

第二节，使用 Django 的 表单 (forms)

例子足够简单，但是能说明问题

源码下载：zqxt_forms2.zip

新建一个 zqxt_form2 项目

```
django-admin.py startproject zqxt_form2
# 进入到 zqxt_form2 文件夹，新建一个 tools APP
```

```
python manage.py startapp tools
```

在tools文件夹中新建一个 forms.py 文件

```
from django import forms

class AddForm(forms.Form):
    a = forms.IntegerField()
    b = forms.IntegerField()
```

我们的视图函数 views.py 中

```
# coding:utf-8
from django.shortcuts import render
from django.http import HttpResponse

# 引入我们创建的表单类
from .forms import AddForm

def index(request):
    if request.method == 'POST':# 当提交表单时

        form = AddForm(request.POST) # form 包含提交的数据

        if form.is_valid():# 如果提交的数据合法
            a = form.cleaned_data['a']
            b = form.cleaned_data['b']
            return HttpResponse(str(int(a) + int(b)))

    else:# 当正常访问时
        form = AddForm()
    return render(request, 'index.html', {'form': form})
```

对应的模板文件 index.html

```
<form method='post'>
{% csrf_token %}
{{ form }}
<input type="submit" value="提交">
</form>
```

再在 urls.py 中对应写上这个函数

```
from django.conf.urls import patterns, include, url

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # 注意下面这一行
    url(r'^$', 'tools.views.index', name='home'),
    url(r'^admin/', include(admin.site.urls)),
)
```

新手可能觉得这样变得更麻烦了，有些情况是这样的，但是 Django 的 forms 提供了：

1. 模板中表单的渲染
2. 数据的验证工作，某一些输入不合法也不会丢失已经输入的数据。
3. 还可以定制更复杂的验证工作，如果提供了10个输入框，必须必须要输入其中两个以上，在 forms.py 中都很容易实现

也有一些将 Django forms 渲染成 Bootstrap 的插件，也很好用，很方便。

扩展：如果提交后在同一个页面显示结果呢？请看 Django Ajax

Django 配置

运行 django-admin.py startproject [project-name] 命令会生成一系列文件，在Django 1.6版本以后的 settings.py 文件中有以下语句：

```
# Build paths inside the project like this: os.path.join(BASE_D
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
```

这里用到了python中一个神奇的变量 __file__ 这个变量可以获取到当前文件（包含这个代码的文件）的路径。
os.path.dirname(__file__) 得到文件所在目录，再来一个 os.path.dirname()就是目录的上一级，BASE_DIR 即为 项目 所在目录。我们在后面的与目录有关的变量都用它，这样使得移植性更强。

```
# SECURITY WARNING: don't run with debug turned on in productic
DEBUG = True
TEMPLATE_DEBUG = True
```

DEBUG＝True 时，如果出现 bug 便于我们看见问题所在，但是部署时最好不要让用户看见bug的详情，可能一些不怀好心的人攻击网站，造成不必要的麻烦。

```
ALLOWED_HOSTS = ['*.besttome.com','www.ziqiangxuetang.com']
```

ALLOWED_HOSTS 允许你设置哪些域名可以访问，即使在

Apache中绑定了，这里不允许的话，也是不能访问的。

当 DEBUG=False 时，这个为必填项，如果不想输入，可以用 **ALLOW_HOSTS = ['*']** 来允许所有的。

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR,'static')
```

static 是静态文件所有目录，比如 jquery.js, bootstrap.min.css 等文件。

一般来说我们只要把静态文件放在 APP 中的 static 目录下，部署时用 python manage.py collectstatic 就可以把静态文件收集到 STATIC_ROOT 目录，但是有时我们有一些共用的静态文件，这时候可以设置 **STATICFILES_DIRS** 另外弄一个文件夹，如下：

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
    '/var/www/static/',
)
```

这样我们就可以把静态文件放在 common_static 和 /var/www/static/中了，Django也能找到它们。

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR,'media')
```

media文件夹用来存放用户上传的文件，与权限有关，详情见 Django 静态文件 和 Django 部署

有时候有一些模板不是属于app的，比如 baidutongji.html, share.html等，

Django 1.5 - Django 1.7

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
    os.path.join(BASE_DIR, 'templates2').replace('\\', '/'),
    # ...
)
```

Django 1.8 及以上版本

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTempl
        'DIRS': [
            os.path.join(BASE_DIR,'templates').replace('\\', '/
            os.path.join(BASE_DIR,'templates2').replace('\\', '
        ],
        'APP_DIRS': True,
]
```

这样 就可以把模板文件放在 templates 和 templates2 文件夹中了。

Django 静态文件

静态文件是指 网站中的 js, css, 图片，视频等文件

开发阶段

推荐用新版本的Django进行开发，可以肯定的是 Django 1.4 以后的版本应该都支持下面的设置

注意：Django 1.4 版本需要在 project/urls.py 底部加上：

```
from django.contrib.staticfiles.urls import staticfiles_urlpatt

urlpatterns += staticfiles_urlpatterns()
```

Django 1.4 静态文件相关文档：https://docs.djangoproject.com/en/1.4/howto/static-files/

Django 1.5 - Django 1.8 不需要添加上面的代码。

settings.py 静态文件相关示例代码：

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.8/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'collected_static')
```

```
# 其它 存放静态文件的文件夹，里面不能包含 STATIC_ROOT
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
    '/path/to/others/static/',
)

# 这个是默认设置，默认会找 STATICFILES_DIRS 中所有文件夹和各app下的 st
STATICFILES_FINDERS = (
    "django.contrib.staticfiles.finders.FileSystemFinder",
    "django.contrib.staticfiles.finders.AppDirectoriesFinder"
)
```

静态文件放在对应的 app 下的 static 文件夹中，当 DEBUG = True 时，Django 就能自动找到放在里面的静态文件。

示例项目 dj18static, 应用 app 下面有一个 static 里面有一个 zqxt.png 图片：

```
dj18static
├── blog
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── static # 应用 blog 下的 static，默认会找这个文件夹
│   │   └── 【zqxt.png】
│   ├── tests.py
│   │
│   └── views.py
├── common_static # 已经添加到了 STATICFILES_DIRS 的文件夹
│   └── js
│       └── 【jquery.js】
│
├── dj18static
│   ├── __init__.py
```

```
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
└── manage.py
```

当 settings.py 中的 DEBUG = True 时，打开开发服务器 python manage.py runserver 直接访问 /static/zqxt.png 就可以找到这个静态文件。

也可以在 settings.py 中指定所有 app 共用的静态文件，比如 jquery.js 等

```
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "common_static"),
)
```

把 jquery.js 放在 common_static/js/ 下，这样就可以 在 /static/js/jquery.js 中访问到它！

示例下载：dj18static.zip

其它参考办法（当你想为静态文件分配多个不同的网址时，可能会用上这个）：

当然也可以自己指定静态文件夹, 在urls.py的最后边这样写

```
# static files
import os
from django.conf.urls.static import static
from django.conf import settings
if settings.DEBUG:
    media_root = os.path.join(settings.BASE_DIR,'media2')
    urlpatterns += static('/media2/', document_root=media_root)
```

也可以这样

```
from django.conf.urls.static import static

urlpatterns = patterns('',
    url(r'^$', 'app.views.index', name='index'),
    url(r'^admin/', include(admin.site.urls)),
) + static('/media2/', document_root=media_root)
```

部署时

1. 收集静态文件

```
python manage.py collectstatic
```

这一句话就会把以前放在app下static中的静态文件全部拷贝到 settings.py 中设置的 **STATIC_ROOT** 文件夹中

2. 用 apache2 或 nginx 示例代码

apache2配置文件

```
Alias /static/ /path/to/static/

<Directory /path/to/static>
    Require all granted
</Directory>
```

nginx 示例代码：

```
location /media  {
    alias /path/to/project/media;
}

location /static {
```

```
    alias /path/to/project/static;
}
```

Apache 完整的示例代码：

```
<VirtualHost *:80>
        ServerName www.ziqiangxuetang.com
        ServerAlias ziqiangxuetang.com
        ServerAdmin tuweizhong@163.com

        Alias /favicon.ico /path/to/static/favicon.ico
        Alias /media/ /path/to/media/
        Alias /static/ /path/to/static/

        <Directory /path/to/media>
                Require all granted
        </Directory>

        <Directory /path/to/static>
                Require all granted
        </Directory>

        WSGIScriptAlias / /path/to/prj/prj/wsgi.py
        <Directory /path/to/prj/prj>
        <Files wsgi.py>
                Require all granted
        </Files>
        </Directory>
</VirtualHost>
```

如果你用的是apache 2.2 版本 用下面的代替 Require all granted 赋予权限

```
Order allow,deny
Allow from all
```

备注：（用 apachectl -v 命令查看 apache2版本号）

补充：有没有不把静态文件放在static和media文件夹中，放在任何地方都能访问到的方法呢?

刚开始搞Django的时候其实我也一直在找，因为毕竟asp,php可以直接访问路径中对应的静态文件。

下面是一个例子

```
<VirtualHost *:80>
    ServerName www.ziqiangxuetang.com
    ServerAdmin tuweizhong@163.com

    AliasMatch "(?i)^/(.+)\.ico$" "/path/to/project/$1.ico"
    AliasMatch "(?i)^/(.+)\.js$" "/path/to/project/$1.js"
    AliasMatch "(?i)^/(.+)\.css$" "/path/to/project/$1.css"
    AliasMatch "(?i)^/(.+)\.jpg$" "/path/to/project/$1.jpg"
    AliasMatch "(?i)^/(.+)\.jpeg$" "/path/to/project/$1.jpeg"
    AliasMatch "(?i)^/(.+)\.png$" "/path/to/project/$1.png"
    AliasMatch "(?i)^/(.+)\.gif$" "/path/to/project/$1.gif"
    AliasMatch "(?i)^/(.+)\.xml$" "/path/to/project/$1.xml"
    AliasMatch "(?i)^/(.+)\.xsl$" "/path/to/project/$1.xsl"
    AliasMatch "(?i)^/(.+)\.txt$" "/path/to/project/$1.txt"
    AliasMatch "(?i)^/(.+)\.zip$" "/path/to/project/$1.zip"
    AliasMatch "(?i)^/(.+)\.rar$" "/path/to/project/$1.txt"

    <Directory /path/to/project>
        Require all granted
    </Directory>

    WSGIScriptAlias / /path/to/project/project/wsgi.py

    ErrorLog /path/to/project/error.log
    CustomLog /path/to/project/access.log common
</VirtualHost>
```

更详细的部署讲解请查看 Django 部署（apache2)

nginx 详细完整的代码看 Django 部署 （nginx)

Django 部署(Apache)

本文讲述部署的方法和常见的问题，并给出了在 BAE, JAE, SAE 等上面部署的实例。

如果不是在自己的服务器上部署，当开发完成后可以部署到 BAE, SAE, JAE, 也可以使用阿里云的服务器。

部署到JAE的例子：https://code.jd.com/twz915/django_demo（实测，JAE好像目前不可以用了）

部署到BAE的例子：https://github.com/twz915/BAE_Django（实测，推荐）

部署到SAE的例子：https://github.com/twz915/django-sae（Fork smallcode同学的，没有测试过，SAE 有一定的免费份额）

Django + nginx + Gunicorn/uwsgi 部署方式，参见另一篇：Django 部署（nginx）

自己的服务器（比如用的阿里云服务器）请看下文：

如果是新手，个人推荐用ubuntu,除非你对linux非常熟悉，ubuntu

服务器的优点：

一，开机apache2等都自动启动，不需要额外设置

二，安装软件非常方便 apt-get 搞定

三，安装ssh，git等也非常容易，几乎是傻瓜化

可以先用用 Linux Mint, 它用起来更简单，和ubuntu兼容。

下面是ubuntu上的部署详细步骤：

1. 安装 apache2 和 mod_wsgi

```
sudo apt-get install apache2 libapache2-mod-wsgi
```

2. 确认安装的apache2版本号

```
apachectl -v
```

Server version: Apache/**2.4**.6 (ubuntu)

Server built: Dec 5 2013 18:32:22

3. 准备一个新网站

ubuntu的apache2配置文件在 /etc/apache2/ 下

新建一个网站配置文件

```
sudo vi /etc/apache2/sites-available/sitename.conf
```

示例内容如下：

```
<VirtualHost *:80>
    ServerName www.yourdomain.com
    ServerAlias otherdomain.com
    ServerAdmin tuweizhong@163.com

    Alias /media/ /home/tu/blog/media/
    Alias /static/ /home/tu/blog/static/

    <Directory /home/tu/blog/media>
        Require all granted
    </Directory>

    <Directory /home/tu/blog/static>
        Require all granted
    </Directory>

    WSGIScriptAlias / /home/tu/blog/blog/wsgi.py

    <Directory /home/tu/blog/blog>
    <Files wsgi.py>
        Require all granted
    </Files>
    </Directory>
</VirtualHost>
```

如果你的apache版本号是 2.2.x

用下面的代替 Require all granted

```
Order deny,allow
Allow from all
```

4. 修改wsgi.py文件

上面的配置中写的
WSGIScriptAlias / /home/tu/blog/blog/wsgi.py

就是把apache2和你的网站project联系起来了

```
import os
from os.path import join,dirname,abspath

PROJECT_DIR = dirname(dirname(abspath(__file__)))#3
import sys # 4
sys.path.insert(0,PROJECT_DIR) # 5

os.environ["DJANGO_SETTINGS_MODULE"] = "blog.settings" # 7

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

第 3，4，5 行为新加的内容，作用是让脚本找到django项目的位置，也可以在sitename.conf中做，用WSGIPythonPath,想了解的自行搜索，第 7 行如果一台服务器有多个django project时一定要修改成上面那样，否则访问的时候会发生网站互相串的情况，即访问A网站到了B网站，一会儿正常，一会儿又不正常（当然也可以使用mod_wsgi daemon 模式,点击这里查看）

5. 设置目录和文件权限

一般目录权限设置为 755，文件权限设置为 644

假如项目位置在 /home/tu/zqxt （在zqxt 下面有一个manage.py，zqxt 是项目名称）

```
cd /home/tu/
sudo chmod -R 644 zqxt
sudo find zqxt -type d -exec chmod 755 \{\} \;
```

apache 服务器运行用户可以在 /etc/apache2/envvars 文件里面改，这里使用的是默认值，当然也可以更改成自己的当前用户，这样的话权限问题就简单很多。以下是默认设置：

```
# Since there is no sane way to get the parsed apache2 config i
# settings are defined via environment variables and then used
# /etc/init.d/apache2, /etc/logrotate.d/apache2, etc.

export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
```

上传文件夹权限

media 文件夹一般用来存放用户上传文件，static 一般用来放自己网站的js，css，图片等，在settings.py中的相关设置

STATIC_URL 为静态文件的网址 STATIC_ROOT 为静态文件的根目录，

MEDIA_URL 为用户上传文件夹的根目录，MEDIA_URL为对应的访问网址

在settings.py中设置：

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/dev/howto/static-files/
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR,'static')

# upload folder
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR,'media')
```

在 Linux 服务器上，用户上传目录还要设置给 www-data 用户的写权限，下面的方法比较好，不影响原来的用户的编辑。

假如上传目录为 zqxt/media/uploads 文件夹,进入media文件夹，将 uploads 用户组改为www-data，并且赋予该组**写权限**:

```
cd media/ # 进入media文件夹
sudo chgrp -R www-data uploads
sudo chmod -R g+w uploads
```

备注：这两条命令，比直接用sudo chown -R www-data:www-data uploads 好，因为下面的命令不影响文件原来所属用户编辑文件，fedora系统应该不用设置上面的权限，但是个人强烈推荐用ubuntu,除非你对linux非常熟悉，你自己选择。

如果你使用的是**sqlite3数据库**，还会提示 Attempt to write a readonly database,同样要给www-data写数据库的权限

进入项目目录的上一级，比如project目录为 /home/tu/blog 那就进入 /home/tu 执行下面的命令（和修改上传文件夹类似）

```
sudo chgrp www-data blog
sudo chmod g+w blog
sudo chgrp www-data blog/db.sqlite3  # 更改为你的数据库名称
sudo chmod g+w blog/db.sqlite3
```

备注：上面的不要加 -R ,-R是更改包括所有的子文件夹和文件，这样不安全。个人建议可以专门弄一个文件夹,用它来放sqlite3数据库，给该文件夹www-data写权限，而不是整个项目给写权限，有些文件只要读的权限就够了，给写权限会造成不安全。

6. 激活新网站

```
sudo a2ensite sitename 或 sudo a2ensite sitename.conf
```

如果顺利，这样网站就搭建成功，访问你的网址试试看，如果出现问

题就接着看下面的。

7. 错误排查

一，没有静态文件，网站打开很乱，没有布局，多半是静态文件没有生效。

1. 确保你的配置文件中的路径是正确的
2. 确保你的settings.py中的文件设置正确
3. 收集静态文件(详细静态文件部署教程)

```
python manage.py collectstatic
```

二，网站打开后报错

这时你可以把settings.py更改

```
DEBUG = True
```

重启服务器

```
sudo service apache2 restart
```

再访问网站 来查看具体的出错信息。

如果这样做还看不到出错信息，只是显示一个服务器错误，你可以查看apache2的错误日志

```
cat /var/log/apache2/error.log
```

根据错误日志里面的内容进行修正!

总结:

部署时文件对应关系:

sitename.conf ---> wsgi.py---> settings.py----> urls.py ----> views.py

扩展

明白了上面的关系, 一个 Django project 使用多个域名或让app使用子域名很简单,只要新建一个 wsgi.py 文件,更改里面对应的settings文件,新的settings文件可以对应新的urls.py,从而做到访问与原来不同的地址!

Django 部署(nginx)

1. 运行开发服务器测试

```
cd zqxt # 进入项目 zqxt 目录
python manage.py runserver
```

运行开发服务器测试，确保开发服务器下能正常打开网站。

2. 安装 nginx 和 需要的包

2.1 安装 nginx 等软件

ubuntu / Linux Mint 等，下面简写为 (ubuntu):

```
sudo apt-get install python-dev nginx
```

centos / Fedora/ redhat 等，下面简写为 (centos)

```
sudo yum install epel-release
sudo yum install python-devel nginx
```

2.2 安装 supervisor, 一个专门用来管理进程的工具，我们用它来管理 gunicorn/uwsgi

```
sudo pip install supervisor
```

Ubuntu用户 请直接看 3，以下是CentOS 注意事项:

CentOS下，如果不是非常懂 SELinux 和 iptables 的话，为了方便调试，可以先临时关闭它们，如果发现部署了之后出不来结果，可

以临时关闭测试一下，这样就知道是不是 SELinux 和 iptables 的问题

CentOS 7 iptables如何使用：http://stackoverflow.com/questions/24756240/how-can-i-use-iptables-on-centos-7

将 SELinux 设置为宽容模式，方便调试：

```
sudo setenforce 0
```

防火墙相关的设置：

可以选择临时关闭防火墙

```
sudo service iptables stop
```

或者开放一些需要的端口，比如 80

```
sudo iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
```

上面的两条命令，如果是 CentOS 7 用

临时关闭防火墙

```
sudo systemctl stop firewalld
```

或者 开放需要的端口

```
sudo firewall-cmd --zone=public --add-port=80/tcp --permanent
sudo firewall-cmd --reload
```

备注：由于我还没有用 最新版本的 Fedora ， 需要用 dnf 来安装包，有需求的同学自测，可以参考这里。

3. 使用 gunicorn / uwsgi 来部署 (二选一)

注意：以下为二选一，不需要两个都用

3.1 使用 gunicorn（纯Python实现的包）：

```
sudo pip install gunicorn
```

在项目目录下运行下面的命令进行测试：

```
gunicorn -w4 -b0.0.0.0:8001 zqxt.wsgi
```

-w 表示开启多少个worker，-b 表示要使用的ip和port，我们这里用的是 8001，0.0.0.0代表监控电脑的所有 ip。

如果使用了 virtualenv 可以这样

```
/path/to/env/bin/gunicorn --chdir /path/to/project --pythonpath
```

用 --pythonpath 指定依赖包路径，多个的时候用逗号,隔开，如：'/path/to/lib,/home/tu/lib'

3.2 使用 uwsgi（纯C语言实现的包）：

安装 uwsgi

```
sudo pip install uwsgi
```

使用 uwsgi 运行项目

```
uwsgi --http :8001 --chdir /path/to/project --home=/path/to/env
```

这样就可以跑了，--home 指定virtualenv 路径，如果没有可以去掉。project.wsgi 指 project/wsgi.py 文件

如果提示端口已经被占用:

```
probably another instance of uWSGI is running on the same addre
bind(): Address already in use [core/socket.c line 764]
```

这时可以把相关的进程 kill 掉:

按照端口进行查询:

```
lsof -i :8002
```

可以查出:

```
COMMAND  PID USER   FD   TYPE             DEVICE SIZE/OFF NODE
uwsgi   2208   tu    4u  IPv4 0x53492abadb5c9659      0t0  TCP
uwsgi   2209   tu    4u  IPv4 0x53492abadb5c9659      0t0  TCP
```

这时根据 PID 可以用下面的命令 kill 掉相关程序:

```
sudo kill -9 2208 2209
```

按照程序名称查询:

```
ps aux | grep uwsgi
```

4. 使用supervisor来管理进程

生成 supervisor 默认配置文件，比如我们放在 /etc/supervisord.conf 路径中:

```
sudo echo_supervisord_conf > /etc/supervisord.conf
```

打开 supervisor.conf 在最底部添加:

```
[program:zqxt]
```

```
command=/path/to/uwsgi --http :8003 --chdir /path/to/zqxt --mc
directory=/path/to/zqxt
startsecs=0
stopwaitsecs=0
autostart=true
autorestart=true
```

command 中写上对应的命令，这样，就可以用 supervisor 来管理了

```
supervisord -c /etc/supervisord.conf
```

重启 zqxt 程序（项目）：

```
supervisorctl -c /etc/supervisord.conf restart zqxt
```

启动，停止，或重启 supervisor 管理的某个程序 或 所有程序：

```
supervisorctl -c /etc/supervisord.conf [start|stop|restart] [pr
```

以 uwsgi 为例，上面这样使用一行命令太长了，我们使用 ini 配置文件来搞定，比如项目在 /home/tu/zqxt 这个位置，

在其中新建一个 uwsgi.ini 全路径为 /path/to/project/uwsgi.ini

```
[uwsgi]
chdir=/path/to/project
module=zqxt.wsgi:application
socket = /tmp/zqxt.sock
master=True
pidfile=/tmp/zqxt-master.pid
vacuum=True
max-requests=5000
daemonize=/path/to/project/zqxt.log
```

注意上面的 /tmp/zqxt.sock ，一会儿我们把它和 nginx 关联起来。

修改 supervisor 配置文件中的 command 一行：

```
[program:zqxt]
 command=/path/to/uwsgi --ini /path/to/project/uwsgi.ini
 directory=/path/to/zqxt
 startsecs=0
```

然后重启一下 supervisor：

```
supervisorctl -c /etc/supervisord.conf restart zqxt
```
或者 `supervisorctl -c /etc/supervisord.conf restart all`

5. 配置 nginx

新建一个网站 zqxt

```
sudo vim /etc/nginx/sites-available/zqxt.conf
```

写入以下内容：

```
upstream zqxt_uwsgi {
    server unix:///tmp/zqxt.sock;
}

server {
    listen      80;
    server_name www.ziqiangxuetang.com;
    charset     utf-8;

    client_max_body_size 75M;

    location /media  {
        alias /path/to/project/media;
    }

    location /static {
        alias /path/to/project/static;
    }
```

```
    location / {
        uwsgi_pass  zqxt_uwsgi;
        include     /etc/nginx/uwsgi_params;
    }
}
```

激活网站：

```
sudo ln -s /etc/nginx/sites-available/zqxt.conf /etc/nginx/site
```

重启 nginx 服务器：

```
sudo service nginx reload
```
或者 `sudo service nginx restart`

一些有用的参考教程：

Django 官网部署教程：

https://docs.djangoproject.com/en/1.8/howto/deployment/

https://docs.djangoproject.com/en/1.8/howto/deployment/

一些博客相关教程：

http://www.ituring.com.cn/article/201045

http://www.jianshu.com/p/be9dd421fb8d

http://blog.csdn.net/tengzhaorong/article/details/1283315

nginx 与 socket

http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html#setting-up-nginx-for-your-site

防火墙:

iptables: https://www.digitalocean.com/community/tutorials/how-to-setup-a-basic-ip-tables-configuration-on-centos-6

centos 7
FireWalld: http://stackoverflow.com/questions/24756240/how-can-i-use-iptables-on-centos-7

ubuntu ufw 防火墙：http://wiki.ubuntu.org.cn/Ufw%E4%BD%BF%E7%94%

uwsgi ini 配置文件：http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html#configuring-uwsgi-to-run-with-a-ini-file

---

Django 数据导入

从网上下载的一些数据，excel表格，xml文件，txt文件等有时候我们想把它导入数据库，应该如何操作呢?

以下操作符合 Django版本为 1.6 ，兼顾 Django 1.7, Django 1.8 版本，理论上Django 1.4, 1.5 也没有问题，没有提到的都是默认值

备注：你可能会问数据从哪儿来的，比如你用python从以前的blog上获取过来的，想导入现在的博客，或者别人整理好的数据，或者你自己整理的excel表，一个个地在网站后台复制粘贴你觉得好么？这就是批量导入的必要性。

下载本教程源代码： mysite.zip

建议先不要看源码，按教程一步步做下去，遇到问题再试试源代码，直接复制粘贴，很快就会忘掉，自己动手打一遍

我们新建一个项目 mysite, 再新建一个 app，名称为blog

```
django-admin.py startproject mysite
cd mysite
python manage.py startapp blog
```

把 blog 中的 models.py 更改为以下内容

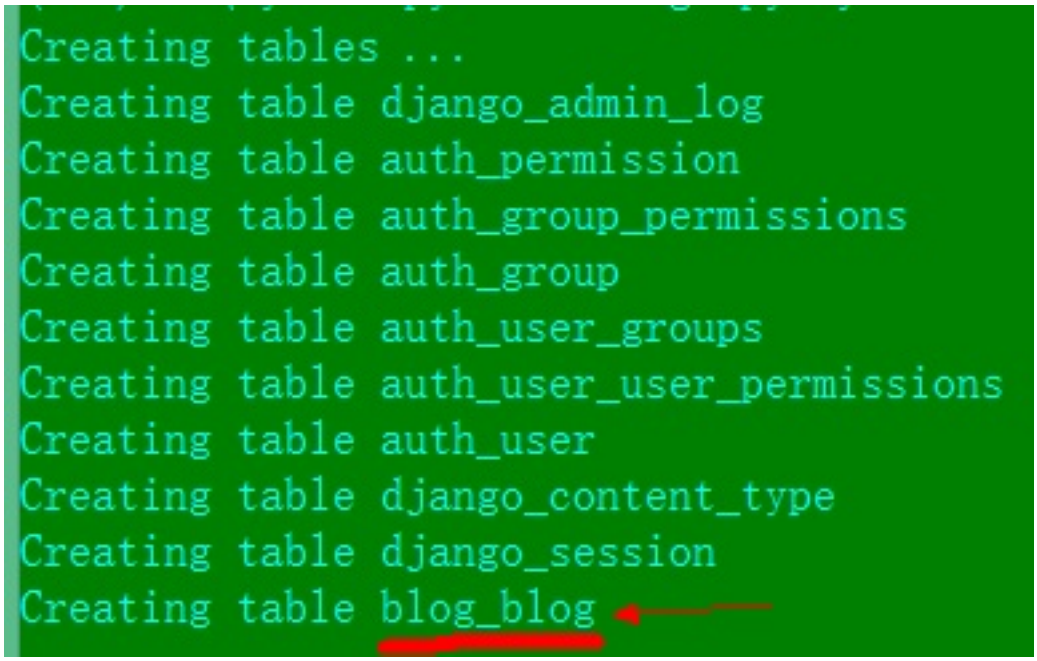
```
#!/usr/bin/python
#coding:utf-8
```

```
from django.db import models

class Blog(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

    def __unicode__(self):
        return self.title
```

一，同步数据库，创建相应的表

```
python manage.py syncdb
```

Django 1.6以下版本会看到：



Django 创建了一些默认的表，注意后面那个红色标记的blog_blog是appname_classname的样式，这个表是我们自己写的Blog类创建的

Django 1.7.6及以上的版本会看到：（第六行即为创建了对应的blog_blog表）

```
Operations to perform:
  Synchronize unmigrated apps: blog
  Apply all migrations: admin, contenttypes, auth, sessions
Synchronizing apps without migrations:
  Creating tables...
    Creating table blog_blog
  Installing custom SQL...
  Installing indexes...
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK

You have installed Django's auth system, and don't have any sup
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'tu'): tu
Email address:
Password:
Password (again):
Superuser created successfully.
```

二，输入 python manage.py shell

进入该项目的django环境的终端（windows如何进入对应目录？看 Django环境搭建 的 3.2 部分）

先说如何用命令新增一篇文章：

```
$ python manage.py shell
>>> from blog.models import Blog
>>> Blog.objects.create(title="The first blog of my site",
                        content="I am writing my blog on Termin
```

这样就新增了一篇博文，我们查看一下

```
>>> Blog.objects.all() # 获取所有blog
[<Blog: The first blog of my site>]
```

还有两种方法(这两种差不多)：

```
>>> blog2 = Blog()
>>> blog2.title = "title 2"
>>> blog2.content = "content 2"
>>> blog2.save()
```

或者

```
>>> blog2 = Blog(title="title 2",content="content 2")
>>> blog2.save()
```

后面两种方法也很重要，尤其是用在修改数据的时候，要记得最后要保存一下 blog.save(),第一种Blog.objects.create()是自动保存的。

三，批量导入

比如我们要导入一个文本，里面是标题和内容，中间用四个*隔开的，示例(oldblog.txt)：

```
title 1****content 1
title 2****content 2
title 3****content 3
title 4****content 4
title 5****content 5
title 6****content 6
title 7****content 7
title 8****content 8
title 9****content 9
```

在终端导入有时候有些不方便，我们在mysite目录下写一个脚本，叫 txt2db.py，把oldblog.txt也放在mysite下

```
#!/usr/bin/env python
#coding:utf-8

import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.setting

'''
Django 版本大于等于1.7的时候，需要加上下面两句
import django
django.setup()
否则会抛出错误 django.core.exceptions.AppRegistryNotReady: Models
'''

import django
if django.VERSION >= (1, 7):#自动判断版本
    django.setup()


def main():
    from blog.models import Blog
    f = open('oldblog.txt')
    for line in f:
        title,content = line.split('****')
        Blog.objects.create(title=title,content=content)
    f.close()

if __name__ == "__main__":
    main()
    print('Done!')
```

好了，我们在终端运行它

```
python txt2db.py
# 运行完后显示 一个 Done! 导入完成!
```

运行完毕后会打出一个 "Done!", 数据已经全部导入!

四，导入数据重复 解决办法

如果你导入数据过多，导入时出错了，或者你手动停止了，导入了一部分，还有一部分没有导入。或者你再次运行上面的命令，你会发现数据重复了，怎么办呢?

django.db.models 中还有一个函数叫 get_or_create() 有就获取过来，没有就创建，用它可以避免重复，但是速度可以会慢些，因为要先尝试获取，看看有没有

只要把上面的

```
Blog.objects.create(title=title,content=content)
```

换成下面的就不会重复导入数据了

```
Blog.objects.get_or_create(title=title,content=content)
```

返回值是（BlogObject, True/False) 新建时返回 True, 已经存在时返回 False。

更多数据库API的知识请参见官网文档：QuerySet API

五, 用fixture导入（更快）

最常见的fixture文件就是用python manage.py dumpdata 导出的文件,示例如下:

```
[
  {
    "model": "myapp.person",
    "pk": 1,
```

```
    "fields": {
      "first_name": "John",
      "last_name": "Lennon"
    }
  },
  {
    "model": "myapp.person",
    "pk": 2,
    "fields": {
      "first_name": "Paul",
      "last_name": "McCartney"
    }
  }]
```

你也可以根据自己的models,创建这样的json文件,然后用 python manage.py loaddata fixture.json 导入

详见:https://docs.djangoproject.com/en/dev/howto/initial-data/

可以写一个脚本,把要导入的数据转化成 json 文件,这样导入也会更快些!

六，Model.objects.bulk_create() 更快更方便

```
#!/usr/bin/env python
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.setting

def main():
    from blog.models import Blog
    f = open('oldblog.txt')
    BlogList = []
    for line in f:
        title,content = line.split('****')
        blog = Blog(title=title,content=content)
        BlogList.append(blog)
    f.close()
```

```
    Blog.objects.bulk_create(BlogList)

if __name__ == "__main__":
    main()
    print('Done!')
```

由于Blog.objects.create()每保存一条就执行一次SQL，而bulk_create()是执行一条SQL存入多条数据，做会快很多！当然用列表解析代替 for 循环会更快！！

```
#!/usr/bin/env python
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mysite.setting

def main():
    from blog.models import Blog
    f = open('oldblog.txt')

    BlogList = []
    for line in f:
        parts = line.split('****')
        BlogList.append(Blog(title=parts[0], content=parts[1]))

    # 以上四行 也可以用 列表解析 写成下面这样
    # BlogList = [Blog(title=line.split('****')[0], content=lin

    Blog.objects.bulk_create(BlogList)

if __name__ == "__main__":
    main()
    print('Done!')
```

当然也可以利用数据中的导出，再导入的方法，见下一节。

Django 数据迁移

写过Django项目的同学，必然会遇到这个问题：在django 1.6以及以前的版本中，我们测试，当发现model要改，怎么办？我们修改了models.py 之后 python manage.py syncdb 只会将新类创建表，删除的类询问是否要删除对应的表，但是在原来的类上增加字段或者删除字段只能参考 python manage.py sql appname 给出的SQL语句，然后自己手动到数据库去用 SQL 语言操作。但是这样非常容易出错！不过我们可以使用 south 这个 app 来实现表的更改。

Django 1.7 及以后的版本集成了 South 的功能。

本文主要讲数据库的转移和对已经创建的表的修改方法

一，简单的数据导出与导入（简单的迁移）

1. django 项目提供了一个导出的方法 python manage.py dumpdata, 不指定 appname 时默认为导出所有的app

```
python manage.py dumpdata [appname] > appname_data.json
```

比如我们有一个项目叫 mysite, 里面有一个 app 叫 blog ,我们想导出 blog 的所有数据

```
python manage.py dumpdata blog > blog_dump.json
```

2. 数据导入,不需要指定 appname

```
python manage.py loaddata blog_dump.json
```

备注：一些常用的

```
python manage.py dumpdata auth > auth.json # 导出用户数据
```

二，数据库的迁移

2.1. 用 Django 自带的命令

比如早期我们为了开发方便，用的sqlite3数据库，后来发现网站数据太多，sqlite3性能有点跟不上了，想换成postgreSQL,或者MySQL的时候。

如果还我还使用上面的命令，如果你运气好的话，也许会导入成功，流程如下：

2.1.1. 从原来的数据库导出所有数据

```
python manage.py dumpdata > mysite_all_data.json
```

2.1.2. 将mysite_all_data.json传送到另一个服务器或电脑上导入

```
python manage.py loaddata mysite_all_data.json
```

如果你运气好的话可能会导入完成，但是往往不那么顺利，原因如下：

a) 我们在写models的时候如果用到CharField,就一定要写max_length,在sqlite3中是不检查这个最大长度的，你写最大允许长度为100，你往数据库放10000个，sqlite3都不报错，而且不截断数据的长度，这似乎是slite3的优点，但是也给从sqlite3导入其它数据库带来了困难,因为MySQL和PostgreSQL数据库都会检查最大长度，超出时就报错！

b) Django 自带的contentType会导致出现一些问题

用上面的方法只迁移一个app应该问题不大，但是如果有用户，用户组挂钩，事情往往变得糟糕！如果导入后没有对数据进行修改，你可以考虑重新导入，可能还要快一些，如果是手动在后台输入或者修改过，这种方法就不适用了

2.2, 用数据库自带的导出导入命令

假定 Django 用的数据库名称为 zqxt

2.2.1 在 PostgreSQL 中：

```
# 导出数据库 zqxt 到 zqxt.sql 文件中
pg_dump zqxt > zqxt.sql

# 导入数据库到 新的服务器
psql zqxt -f zqxt.sql

#注意：数据导入导出可能需要数据库超级权限,用 sudo su postgres 切换到数据
```

2.2.2 在MySQL 中：

使用网页工具，比如phpMyAdmin 导入导出很简单，这里就不说了，主要说一下命令行如何操作：

```
# 导出数据库 zqxt 到 zqxt.sql 文件中
mysqldump -u username -p --database zqxt > zqxt.sql

# 导入数据库到 新的服务器
mysql -u username -p
输入密码进入 MySQL 命令行
> source /path/to/zqxt.sql
```

总结：其它的数据库，请自行搜索如何导入导出，整个数据库导出的好处就是对数据之间的关系处理比较省事，比如自强学堂里面的很多教程，上一篇和下一篇是一个一对一的关系，这样的话用 python

manage.py dumpdata 无法导出教程与教程的关系，但是数据库整个导出就没有任何问题，当然也可以写一个脚本去导出关系再导入。Django 自带的 python manage.py dumpdata 和 python manage.py loaddata 最大的好处就是可以跨数据库进行导入导出。

三，使用South等工具进行迁移数据（数据表的更改）

Django 1.7及以上的版本：

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

Django 1.6及以下版本：

常用的工具有：south, dmigrations, django-evolution 推荐使用 south， Django 1.7 中集成了 south

Django 的第三方 app South 就是专门做数据库表结构自动迁移工作，Jacob Kaplan-Moss 曾做过一次调查，South 名列最受欢迎的第三方 app。事实上，它现在已经俨然成为 Django 事实上的数据库表迁移标准，很多第三方 app 都会带 South migrations 脚本。

1, 安装South

```
(sudo) apt-get install python-django-south
# 或者利用pip安装 (sudo) pip install South
```

或者用 easy_install South 来安装，也可以下载源码包，或者运行 python setup.py install 安装，这里下载

2. 使用方法

一个好的程序使用起来必定是简单的，South和它的宗旨一样，使用简单。只需要简单几步，针对已经建好model和创建完表的应用。

把south加入到settings.py中的INSTALL_APPS中

```
# Application definition
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'blog',
    'south',
)
```

修改好后运行一次python manage.py syncdb, Django会新建一个south_migrationhistory表，用来纪录Migration的历史纪录。

```
> python manage.py syncdb
Syncing...
Creating tables ...
Creating table south_migrationhistory
Installing custom SQL ...
Installing indexes ...
No fixtures found.

Synced:
 > django.contrib.admin
 > django.contrib.auth
 > django.contrib.contenttypes
 > django.contrib.sessions
 > django.contrib.messages
```

```
 > django.contrib.staticfiles
 > blog
 > south

Not synced (use migrations):
```

如果要把之前建好的blog这个app加入到 Migration：

```
>>> python manage.py convert_to_south blog
```

你会发现blog文件夹中多了一个 migrations 目录，里面有一个 0001_initial.py 文件。

接着，当你对 Blog.models 做任何修改后，只要执行：

```
>>> python manage.py schemamigration blog --auto
```

South就会帮助我们找出哪些地方做了修改，如果你新增的数据表没有给default值，并且没有设置null=True, south会问你一些问题，因为新增的column对于原来的旧的数据不能为Null的话就得有一个值。顺利的话，在migrations文件夹下会产生一个0002_add_mobile_column.py，但是这一步并没有真正修改数据库的表，我们需要执行 python manage.py migrate ：

```
> python manage.py migrate
Running migrations for blog:
 - Migrating forwards to 0002_add_mobile_column.
 > blog:0002_add_mobile_column
 - Loading initial data for blog.
No fixtures found.
```

这样所做的更改就写入到了数据库中了。

恢复到以前

South好处就是可以随时恢复到之前的一个版本，比如我们想要回到最开始的那个版本：

```
> python manage.py migrate blog 0001
 - Soft matched migration 0001 to 0001_initial.
Running migrations for blog:
 - Migrating backwards to just after 0001_initial.
 < blog:0002_add_mobile_column
```

这样就搞定了，数据库就恢复到以前了，比你手动更改要方便太多了。

备注：但是注意models.py还是要你手动去掉新增的column的

Django 多数据库联用

本文讲述在一个 django project 中使用多个数据库的方法, 多个数据库的联用 以及多数据库时数据导入导出的方法。

直接给出一种简单的方法吧，想了解更多的到官方教程，点击此处

代码文件下载:

project_name.zip

1. 每个app都可以单独设置一个数据库

settings.py中有数据库的相关设置，有一个默认的数据库 default, 我们可以再加一些其它的，比如:

```
# Database
# https://docs.djangoproject.com/en/1.6/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    },
    'db1': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'dbname1',
        'USER': 'your_db_user_name',
        'PASSWORD': 'yourpassword',
        "HOST": "localhost",
    },
    'db2': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'dbname2',
```

```
        'USER': 'your_db_user_name',
        'PASSWORD': 'yourpassword',
        "HOST": "localhost",
    },
}

# use multi-database in django
# add by WeizhongTu
DATABASE_ROUTERS = ['project_name.database_router.DatabaseAppsR
DATABASE_APPS_MAPPING = {
    # example:
    #'app_name':'database_name',
    'app1': 'db1',
    'app2': 'db2',
}
```

在project_name文件夹中存放 database_router.py 文件，内容如下：

```
#!/usr/bin/python

from django.conf import settings

class DatabaseAppsRouter(object):
    """
    A router to control all database operations on models for d
    databases.

    In case an app is not set in settings.DATABASE_APPS_MAPPING
    will fallback to the `default` database.

    Settings example:

    DATABASE_APPS_MAPPING = {'app1': 'db1', 'app2': 'db2'}
    """
    def db_for_read(self, model, **hints):
        """Point all read operations to the specific database.
        if settings.DATABASE_APPS_MAPPING.has_key(model._meta.a
```

```
            return settings.DATABASE_APPS_MAPPING[model._meta.a
        return None

    def db_for_write(self, model, **hints):
        """Point all write operations to the specific database.
        if settings.DATABASE_APPS_MAPPING.has_key(model._meta.a
            return settings.DATABASE_APPS_MAPPING[model._meta.a
        return None

    def allow_relation(self, obj1, obj2, **hints):
        """Allow any relation between apps that use the same da
        db_obj1 = settings.DATABASE_APPS_MAPPING.get(obj1._meta
        db_obj2 = settings.DATABASE_APPS_MAPPING.get(obj2._meta
        if db_obj1 and db_obj2:
            if db_obj1 == db_obj2:
                return True
            else:
                return False
        return None

    def allow_syncdb(self, db, model):
        """Make sure that apps only appear in the related datab
        if db in settings.DATABASE_APPS_MAPPING.values():
            return settings.DATABASE_APPS_MAPPING.get(model._me
        elif settings.DATABASE_APPS_MAPPING.has_key(model._meta
            return False
        return None
```

这样就实现了指定的 app 使用指定的数据库了,当然你也可以多个sqlite3一起使用，相当于可以给每个app都可以单独设置一个数据库！如果不设置或者没有设置的app就会自动使用默认的数据库。

2.多个数据库联用时数据导入导出

使用的时候和一个数据库的区别是:

如果不是defalut(默认数据库）要在命令后边加 --database=数据库对应的settings.py中的名称 如： --database=db1 或 --database=db2

数据库同步（创建表）

```
python manage.py syncdb #同步默认的数据库，和原来的没有区别

#同步数据库 db1 （注意：不是数据库名是db1,是settings.py中的那个db1，不还
python manage.py syncdb --database=db1
```

数据导出

```
python manage.py dumpdata app1 --database=db1 > app1_fixture.js
python manage.py dumpdata app2 --database=db2 > app2_fixture.js
python manage.py dumpdata auth > auth_fixture.json
```

数据库导入

```
python manage.py loaddata app1_fixture.json --database=db1
python manage.py loaddata app2_fixture.json --database=db2
```

Django 用户注册系统

Django 1.6 - Django 1.8 请直接看这个例子：点击下载

以下适用于 Django 1.5 及以下版本:

注册系统可以使用现有的包 django-registration 官方带模板的 pypi

如果想要用户用邮件注册还要用 django-registration-email

下面是具体的方法:

1. 安装所需的包，推荐用跨平台的 pip 来安装 python 相关的包

```
pip install django-registration
pip install django-registration-email
```

2. 在settings.py中的 INSTALL_APPS 中加上

```
INSTALLED_APPS = [
    # all your other apps
    'registration',
    'registration_email',
]
```

3. 更新 urls.py:

```
url(r'^accounts/', include('registration_email.backends.default
```

4. 增加一些其它的设置到 settings.py:

```
ACCOUNT_ACTIVATION_DAYS = 7 # 激活期限

AUTHENTICATION_BACKENDS = (
    'registration_email.auth.EmailBackend',
)
# NOTICE:not work `lambda request, user: '/'`
LOGIN_REDIRECT_URL = '/'#登陆成功后跳转的网址

# NOTICE:not work only '/accounts/activate/complete/'
REGISTRATION_EMAIL_ACTIVATE_SUCCESS_URL = lambda request, user:
REGISTRATION_EMAIL_REGISTER_SUCCESS_URL = lambda request, user:
```

5. 运行同步数据库命令

```
python manage.py syncdb
```

6. 如果用到找回密码的功能还要配置邮箱，个人域名可以使用阿里云的企业邮箱

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'

EMAIL_USE_TLS = False
EMAIL_HOST = 'smtp.tuweizhong.com'
EMAIL_PORT = 25
EMAIL_HOST_USER = 'ziqiangxuetang@tuweizhong.com'
EMAIL_HOST_PASSWORD = 'PASSWORD'
DEFAULT_FROM_EMAIL = 'ziqiangxuetang@tuweizhong.com'
```

Django 缓存系统

Django 官方关于cache的介绍：https://docs.djangoproject.com/en/1.6/topics/cache/

Django 是动态网站，一般来说需要实时地生成访问的网页，展示给访问者，这样，内容可以随时变化，但是从数据库读多次把所需要的数据取出来，要比从内存或者硬盘等一次读出来 付出的成本大很多。

缓存系统工作原理：

对于给定的网址，尝试从缓存中找到网址，如果页面在缓存中，直接返回缓存的页面，如果缓存中没有，一系列操作（比如查数据库）后，保存生成的页面内容到缓存系统以供下一次使用，然后返回生成的页面内容。

Django settings 中 cache 默认为

```
{
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCac
    }
}
```

也就是默认利用**本地的内存**来当缓存，速度很快。当然可能出来内存不够用的情况，其它的一些内建可用的 Backends 有

```
'django.core.cache.backends.db.DatabaseCache'
'django.core.cache.backends.dummy.DummyCache'
'django.core.cache.backends.filebased.FileBasedCache'
'django.core.cache.backends.locmem.LocMemCache'
```

```
'django.core.cache.backends.memcached.MemcachedCache'
'django.core.cache.backends.memcached.PyLibMCCache'
```

在 github 上也有用 redis 做 Django的缓存系统的开源项目：https://github.com/niwibe/django-redis

利用文件系统来缓存：

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBa
        'LOCATION': '/var/tmp/django_cache',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 1000
        }
    }
}
```

利用数据库来缓存，利用命令创建相应的表：python manage.py createcachetable cache_table_name

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache
        'LOCATION': 'cache_table_name',
        'TIMEOUT': 600,
        'OPTIONS': {
            'MAX_ENTRIES': 2000
        }
    }
}
```

下面用一些实例来说明如何使用 Django 缓存系统

一般来说我们用 Django 来搭建一个网站，要用到数据库等。

```
from django.shortcuts import render
def index(request):
    # 读取数据库等 并渲染到网页
    # 数据库获取的结果保存到 queryset 中
    return render(request, 'index.html', {'queryset':queryset})
```

像这样每次访问都要读取数据库，一般的小网站没什么问题，当访问量非常大的时候，就会有很多次的数据库查询，肯定会造成访问速度变慢，服务器资源占用较多等问题。

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

@cache_page(60 * 15) # 秒数，这里指缓存 15 分钟，不直接写900是为了提高
def index(request):
    # 读取数据库等 并渲染到网页
    return render(request, 'index.html', {'queryset':queryset})
```

当使用了cache后，访问情况变成了如下：

访问一个网址时，尝试从 cache 中找有没有缓存内容
如果网页在缓存中显示缓存内容，否则生成访问的页面，保存在缓存中以便下次使用，

```
given a URL, try finding that page in the cache
if the page is in the cache:
    return the cached page
else:
    generate the page
    save the generated page in the cache (for next time)
    return the generated page
```

Memcached 是目前 Django 可用的最快的缓存

另外，Django 还可以共享缓存。

Django 生成静态页面

如果网站的流量过大，每次访问时都动态生成，执行SQL语句，消耗大量服务器资源，这时候可以考虑生成静态页面。

生成静态很简单，下面是一个例子：

只要在views.py中这样写就行了

```
from django.shortcuts import render
from django.template.loader import render_to_string
import os


def my_view(request):
    context = {'some_key': 'some_value'}

    static_html = '/path/to/static.html'

    if not os.path.exists(static_html):
        content = render_to_string('template.html', context)
        with open(static_html, 'w') as static_file:
            static_file.write(content)

    return render(request, static_html)
```

上面的例子中，当用户访问时，如果判断没有静态页面就自动生成静态页面，然后返回静态文件，当文件存在的时候就不再次生成。

也可以用一个文件夹，比如在project下建一个 static_html 文件夹，把生成的静态文件都放里面，让用户像访问静态文件那样访问页面。

更佳办法

但是一般情况下都不需要生成静态页面，因为Django 有缓存功能，使用 **Django Cache(缓存)**就相当于把生成生成静态页面，而且还有自动更新的功能，比如30分钟刷新一下页面内容。

用Django管理静态网站内容

如果服务器上不支持Django环境，你可以在本地上搭建一个Django环境，然后生成静态页面，把这些页面放到不支持 Django 的服务器上去，在本地更新，然后上传到服务器，用Django来管理和更新网站的内容，也是一个不错的做法，还可以更安全，听说有很多黑客都是这么做的。

Django 安全

本文介绍Django关于安全的一些特征，包括如何使基于Django的网站的一些建议。

关于安全的官方文档：https://docs.djangoproject.com/en/dev/#security

官方文档包括以下几个方面：

- Security overview
- Disclosed security issues in Django
- Clickjacking protection
- Cross Site Request Forgery protection
- Cryptographic signing
- Security Middleware

Django表单用在模板中的时候我们会加一句 `{% csrf_token %}`

Django 国际化

Django 官方教程：https://docs.djangoproject.com/en/1.7/#internationalization-and-localization

Django 支持国际化，多语言。Django的国际化是默认开启的，如果您不需要国际化支持，那么您可以在您的设置文件中设置 USE_I18N = False，那么Django会进行一些优化，不加载国际化支持机制。

NOTE: 18表示Internationlization这个单词首字母I和结尾字母N之间的字母有18个。I18N就是Internationlization的意思。

Django 完全支持文本翻译，日期时间数字格式和时区。

本质上讲，Django做了两件事：

1. 它允许开发者指定要翻译的字符串
2. Django根据特定的访问者的偏好设置 进行调用相应的翻译文本。

开启国际化的支持，需要在settings.py文件中设置

```
MIDDLEWARE_CLASSES = (
    ...
    'django.middleware.locale.LocaleMiddleware',
)
```

```
LANGUAGE_CODE = 'en'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True

LANGUAGES = (
    ('en', ('English')),
    ('zh-cn', ('中文简体')),
    ('zh-tw', ('中文繁體')),
)

#翻译文件所在目录，需要手工创建
LOCALE_PATHS = (
    os.path.join(BASE_DIR, 'locale'),
)

TEMPLATE_CONTEXT_PROCESSORS = (
    ...
    "django.core.context_processors.i18n",
)
```

生成需要翻译的文件：

```
django-admin.py makemessages -l zh-cn
django-admin.py makemessages -l zh-tw
```

手工翻译 locale 中的文本后，我们需要编译一下，这样翻译才会生效

```
django-admin.py compilemessages
```

Django session

Django完全支持也匿名会话，简单说就是使用跨网页之间可以进行通讯，比如显示用户名，用户是否已经发表评论。session框架让你存储和获取访问者的数据信息，这些信息保存在服务器上（默认是数据库中），以 cookies 的方式发送和获取一个包含 session ID的值，并不是用cookies传递数据本身。

启用session

编辑settings.py中的一些配置

MIDDLEWARE_CLASSES 确保其中包含以下内容

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

INSTALLED_APPS 是包含

```
'django.contrib.sessions',
```

这些是默认启用的。如果你不用的话，也可以关掉这个以节省一点服务器的开销。

提示：您也可以配置使用比如 cache 来存储 session

在视图中使用 session

request.session 可以在视图中任何地方使用，它类似于python中的字典

session 默认有效时间为两周，可以在 settings.py 中修改默认值：参见这里

```
# 创建或修改 session：
request.session[key] = value
# 获取 session：
request.session.get(key,default=None)
# 删除 session
del request.session[key] # 不存在时报错
```

session 例子

比如写一个不让用户评论两次的应用：

```
from django.http import HttpResponse

def post_comment(request, new_comment):
    if request.session.get('has_commented', False):
        return HttpResponse("You've already commented.")
    c = comments.Comment(comment=new_comment)
    c.save()
    request.session['has_commented'] = True
    return HttpResponse('Thanks for your comment!')
```

一个简化的登陆认证：

```
def login(request):
    m = Member.objects.get(username=request.POST['username'])
    if m.password == request.POST['password']:
        request.session['member_id'] = m.id
        return HttpResponse("You're logged in.")
    else:
        return HttpResponse("Your username and password didn't


def logout(request):
```

```
    try:
        del request.session['member_id']
    except KeyError:
        pass
    return HttpResponse("You're logged out.")
```

当登陆时验证用户名和密码，并保存用户id在 session 中，这样就可以在视图中用 request.session['member_id']来检查用户是否登陆，当退出的时候，删除掉它。

Django Ajax

有时候我们需要在不刷新的情况下载入一些内容，在网页的基本知识中我们介绍了 ajax 技术。

在本文中讲解如何用 Django 来实现 不刷新网页的情况下加载一些内容。

由于用 jQuery 实现 ajax 比较简单，所以我们用 jQuery库来实现，想用原生的 javascript 的同学可以参考：ajax 教程，下面也有例子提供下载。

本节有多个实例提供下载，通过看代码可以更快的学习。

第一节，源代码下载：zqxt_ajax_1.zip

这里用 Django 表单 第一节 中的一个例子，我们要实现的是在不刷新的情况下显示计算结果到页面上。

修改 index.html 文件

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>
<form action="/add/" method="get">
    a: <input type="text" id="a" name="a"> <br>
    b: <input type="text" id="b" name="b"> <br>
    <p>result: <span id='result'></span></p>
    <button type="button" id='sum'>提交</button>
</form>

<script src="http://apps.bdimg.com/libs/jquery/1.11.1/jquery.mi
```

```
<script>
    $(document).ready(function(){
      $("#sum").click(function(){
        var a = $("#a").val();
        var b = $("#b").val();

        $.get("/add/",{'a':a,'b':b}, function(ret){
            $('#result').html(ret)
        })
      });
    });
</script>
</body>
</html>
```

在原来的基础上，在一些元素上加了 id, 以便于获取值和绑定数据，然后我们用了jQuery.get() 方法，并用 $(selector).html() 方法将结果显示在页面上，如下图：

备注：关于请求头和 request.is_ajax() 方法使用

views.py 中可以用 `request.is_ajax()` 方法判断是否是 ajax 请求，需要添加一个 HTTP 请求头：

原生javascript：

```
xmlhttp.setRequestHeader("X-Requested-With", "XMLHttpRequest");
```

用 jQuery：

用 $.ajax 方法代替 $.get，因为 $.get 在 IE 中不会发送 ajax header

服务器端会将请求头的值全部大写，中划线改成下划线，并在非标准的头前面加上 HTTP_，这个过程可以认为相当于以下Python代码：

```
STANDARD_HEADERS = ['REFER', 'HOST', ...] # just for example

def handle_header(value):
    value = value.replace('-', '_').upper()

    if value in STANDARD_HEADERS:
        return value

    return 'HTTP_' + value
```

判断ajax方法，以及原生的 javascript实现ajax的示例下载：
zqxt_views_ajax.zip

第二节，源代码下载：zqxt_ajax_list_dict.zip

更复杂的例子，传递一个数组或字典到网页，由JS处理，再显示出来。

views.py

```
from django.http import HttpResponse
import json

def ajax_list(request):
    a = range(100)
```

```
    return HttpResponse(json.dumps(a), content_type='applicatic

def ajax_dict(request):
    name_dict = {'twz': 'Love python and Django', 'zqxt': 'I am
    return HttpResponse(json.dumps(name_dict), content_type='ap
```

Django 1.7 及以后的版本有更简单的方法（使用 JsonResponse(官方文档))：

```
from django.http import JsonResponse

def ajax_list(request):
    a = range(100)
    return JsonResponse(a, safe=False)

def ajax_dict(request):
    name_dict = {'twz': 'Love python and Django', 'zqxt': 'I am
    return JsonResponse(name_dict)
```

在 django 1.6 及以前的旧版本中可以自己写一个 JsonResponse 方法，如下：

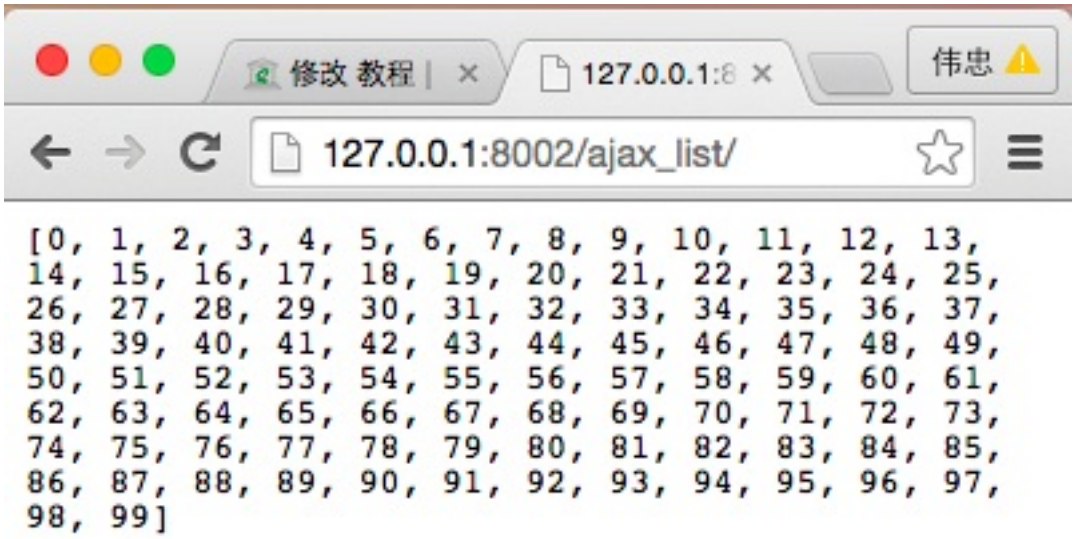
```
from django.http import HttpResponse

import json

class JsonResponse(HttpResponse):
    def __init__(self,
            content={},
            mimetype=None,
            status=None,
            content_type='application/json'):

        super(JsonResponse, self).__init__(
            json.dumps(content),
            mimetype=mimetype,
            status=status,
            content_type=content_type)
```

写好后，我们在 urls.py 中添加以下两行：

```
    url(r'^ajax_list/$', 'tools.views.ajax_list', name='ajax-li
    url(r'^ajax_dict/$', 'tools.views.ajax_dict', name='ajax-di
```

打开开发服务器 python manage.py runserver

我们访问对应的网址会看到输出值：



下一步就是在无刷新的情况下把内容加载到网页了，我们修改一下首页的模板 index.html

```
<!DOCTYPE html>
<html>
<body>
<p>请输入两个数字</p>
<form action="/add/" method="get">
    a: <input type="text" id="a" name="a"> <br>
```

```
    b: <input type="text" id="b" name="b"> <br>
    <p>result: <span id='result'></span></p>
    <button type="button" id='sum'>提交</button>
</form>


<div id="dict">Ajax 加载字典</div>
<p id="dict_result"></p>

<div id="list">Ajax 加载列表</div>
<p id="list_result"></p>


<script src="http://apps.bdimg.com/libs/jquery/1.11.1/jquery.mi
<script>
    $(document).ready(function(){
      // 求和 a + b
      $("#sum").click(function(){
        var a = $("#a").val();
        var b = $("#b").val();

        $.get("/add/",{'a':a,'b':b}, function(ret){
            $('#result').html(ret);
        })
      });

      // 列表 list
      $('#list').click(function(){
          $.getJSON('/ajax_list/',function(ret){
            //返回值 ret 在这里是一个列表
            for (var i = ret.length - 1; i >= 0; i--) {
              // 把 ret 的每一项显示在网页上
              $('#list_result').append(' ' + ret[i])
            };
          })
      })

      // 字典 dict
      $('#dict').click(function(){
          $.getJSON('/ajax_dict/',function(ret){
              //返回值 ret 在这里是一个字典
```

```
                $('#dict_result').append(ret.twz + '<br>');
                // 也可以用 ret['twz']
            })
        })
    });
</script>
</body>
</html>
```

技能提升：getJSON中的写的对应网址，用 urls.py 中的 name 来获取是一个更好的方法!

标签：{% url 'name' %}

```
<script>
    $(document).ready(function(){
      // 求和 a + b
      $("#sum").click(function(){
        var a = $("#a").val();
        var b = $("#b").val();

        $.get("{% url 'add' %}",{'a':a,'b':b}, function(ret){
            $('#result').html(ret);
        })
      });

      // 列表 list
      $('#list').click(function(){
          $.getJSON("{% url 'ajax-list' %}",function(ret){
            //返回值 ret 在这里是一个列表
            for (var i = ret.length - 1; i >= 0; i--) {
              // 把 ret 的每一项显示在网页上
              $('#list_result').append(' ' + ret[i])
            };
          })
      })

      // 字典 dict
      $('#dict').click(function(){
          $.getJSON("{% url 'ajax-dict' %}",function(ret){
```

```
                //返回值 ret 在这里是一个字典
                $('#dict_result').append(ret.twz + '<br>');
                // 也可以用 ret['twz']
            })
        })
    });
</script>
```

这样做最大的好处就是在修改 urls.py 中的网址后，不用改模板中对应的网址。

补充：如果是一个复杂的 列表 或 字典，因为比如如下信息：

```
person_info_dict = [
    {"name":"xiaoming", "age":20},
    {"name":"tuweizhong", "age":24},
    {"name":"xiaoli", "age":33},
]
```

这样我们遍历列表的时候，每次遍历得到一个字典，再用字典的方法去处理，当然有更简单的遍历方法：

用 $.each() 方法代替 for 循环，html 代码（jQuery）

```
$.getJSON('ajax-url-to-json', function(ret) {
    $.each(ret, function(i,item){
        // i 为索引，item为遍历值
    });
});
```

最后，附上一个**返回图片并显示的ajax实例**：

代码下载：zqxt_ajax_pic.zip

Django Ajax CSRF 认证

CSRF（Cross-site request forgery跨站请求伪造，也被称为“one click attack”或者session riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），但它与XSS非常不同，并且攻击方式几乎相左。XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

Django 中自带了 防止CSRF攻击的功能，但是一些新手不知道如何使用，给自己编程带来了麻烦。常常会出现下面django csrf token missing or incorrect的错误。

GET 请求不需要 CSRF 认证，POST 请求需要正确认证才能得到正常的请求。在POST表单中加入 {% csrf_token %}

```
<form method="POST" action="/post-url/">
    {% csrf_token %}

    <input name='zqxt' value="自强学堂学习Django技术">
</form>
```

如果使用Ajax调用的时候，就要麻烦一些。需要注意以下几点：

1. 在视图中使用 **render** （而不要使用 render_to_response）
2. 使用 jQuery 的 ajax 或者 post 之前 加入这个 js 代码：http://www.ziqiangxuetang.com/media/django/csrf

```
jQuery(document).ajaxSend(function(event, xhr, settings) {
    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie && document.cookie != '') {
            var cookies = document.cookie.split(';');
            for (var i = 0; i < cookies.length; i++) {
                var cookie = jQuery.trim(cookies[i]);
                // Does this cookie string begin with the name
                if (cookie.substring(0, name.length + 1) == (na
                    cookieValue = decodeURIComponent(cookie.sub
                    break;
                }
            }
        }
        return cookieValue;
    }
    function sameOrigin(url) {
        // url could be relative or scheme relative or absolute
        var host = document.location.host; // host + port
        var protocol = document.location.protocol;
        var sr_origin = '//' + host;
        var origin = protocol + sr_origin;
        // Allow absolute or scheme relative URLs to same origi
        return (url == origin || url.slice(0, origin.length + 1
            (url == sr_origin || url.slice(0, sr_origin.length
            // or any other URL that isn't scheme relative or a
            !(/^(\/\/|http:|https:).*/.test(url));
    }
    function safeMethod(method) {
        return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
    }

    if (!safeMethod(settings.type) && sameOrigin(settings.url))
        xhr.setRequestHeader("X-CSRFToken", getCookie('csrftoke
    }
});
```

或者 更为优雅简洁的代码（不能写在 .js 中，要直接写在模板文件中）：

```
$.ajaxSetup({
    data: {csrfmiddlewaretoken: '{{ csrf_token }}' },
});
```

这样之后，就可以像原来一样的使用 jQuery.ajax() 和 jQuery.post()了

最后，附上一个 Django Ajax CSRF 实例：  exam.zip

---

Django Sitemap 站点地图

Django 中自带了 sitemap框架，用来生成 xml 文件

Django sitemap 演示：http://www.ziqiangxuetang.com/sitemap.xml

sitemap 很重要，可以用来通知搜索引擎页面的地址，页面的重要性，帮助站点得到比较好的收录。

开启sitemap功能的步骤

settings.py 文件中 django.contrib.sitemaps 和 django.contrib.sites 要在 INSTALL_APPS 中

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites',
    'django.contrib.sitemaps',
    'django.contrib.redirects',

    #####
    #othther apps
    #####
)
```

Django 1.7 及以前版本：

TEMPLATE_LOADERS 中要加

入 'django.template.loaders.app_directories.Loader'，像这样：

```
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
 )
```

Django 1.8 及以上版本新加入了 TEMPLATES 设置，其中 APP_DIRS 要为 True，比如：

```
# NOTICE: code for Django 1.8, not work on Django 1.7 and below
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTempl
        'DIRS': [
            os.path.join(BASE_DIR,'templates').replace('\\', '/
        ],
        'APP_DIRS': True,
    },
]
```

然后在 urls.py 中如下配置：

```
from django.conf.urls import url
from django.contrib.sitemaps import GenericSitemap
from django.contrib.sitemaps.views import sitemap

from blog.models import Entry


sitemaps = {
    'blog': GenericSitemap({'queryset': Entry.objects.all(), 'd
    # 如果还要加其它的可以模仿上面的
}

urlpatterns = [
    # some generic view using info_dict
    # ...
```

```
    # the sitemap
    url(r'^sitemap\.xml$', sitemap, {'sitemaps': sitemaps},
        name='django.contrib.sitemaps.views.sitemap'),
]
```

但是这样生成的 sitemap，如果网站内容太多就很慢，很耗费资源，可以采用分页的功能：

```
from django.conf.urls import url
from django.contrib.sitemaps import GenericSitemap
from django.contrib.sitemaps.views import sitemap

from blog.models import Entry

from django.contrib.sitemaps import views as sitemaps_views
from django.views.decorators.cache import cache_page


sitemaps = {
    'blog': GenericSitemap({'queryset': Entry.objects.all(), 'd
    # 如果还要加其它的可以模仿上面的
}

urlpatterns = [
    url(r'^sitemap\.xml$',
        cache_page(86400)(sitemaps_views.index),
        {'sitemaps': sitemaps, 'sitemap_url_name': 'sitemaps'})
    url(r'^sitemap-(?P<section>.+)\.xml$',
        cache_page(86400)(sitemaps_views.sitemap),
        {'sitemaps': sitemaps}, name='sitemaps'),
]
```

这样就可以看到类似如下的 sitemap，如果本地测试访问 http://localhost:8000/sitemap.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
```

```
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
<sitemap><loc>http://www.ziqiangxuetang.com/sitemap-tutorials.x
</sitemapindex>
```

查看了下分页是实现了，但是全部显示成了 ?p=页面数，而且在百度站长平台上测试，发现这样的sitemap百度报错，于是看了下Django的源代码：

在这
里 https://github.com/django/django/blob/1.7.7/dj

于是对源代码作了修改，变成了本站的sitemap的样子，比 ?p=2 这样更优雅

引入 下面这个 比如是 sitemap_views.py

```
import warnings
from functools import wraps

from django.contrib.sites.models import get_current_site
from django.core import urlresolvers
from django.core.paginator import EmptyPage, PageNotAnInteger
from django.http import Http404
from django.template.response import TemplateResponse
from django.utils import six

def x_robots_tag(func):
    @wraps(func)
    def inner(request, *args, **kwargs):
        response = func(request, *args, **kwargs)
        response['X-Robots-Tag'] = 'noindex, noodp, noarchive'
```

```
        return response
    return inner

@x_robots_tag
def index(request, sitemaps,
          template_name='sitemap_index.xml', content_type='appl
          sitemap_url_name='django.contrib.sitemaps.views.sitem
          mimetype=None):

    if mimetype:
        warnings.warn("The mimetype keyword argument is deprecа
            "content_type instead", DeprecationWarning, stackle
        content_type = mimetype

    req_protocol = 'https' if request.is_secure() else 'http'
    req_site = get_current_site(request)

    sites = []
    for section, site in sitemaps.items():
        if callable(site):
            site = site()
        protocol = req_protocol if site.protocol is None else s
        for page in range(1, site.paginator.num_pages + 1):
            sitemap_url = urlresolvers.reverse(
                    sitemap_url_name, kwargs={'section': sectic
            absolute_url = '%s://%s%s' % (protocol, req_site.dc
            sites.append(absolute_url)

    return TemplateResponse(request, template_name, {'sitemaps'
                            content_type=content_type)

@x_robots_tag
def sitemap(request, sitemaps, section=None, page=1,
            template_name='sitemap.xml', content_type='applicat
            mimetype=None):

    if mimetype:
        warnings.warn("The mimetype keyword argument is deprecа
            "content_type instead", DeprecationWarning, stackle
        content_type = mimetype
```

```
    req_protocol = 'https' if request.is_secure() else 'http'
    req_site = get_current_site(request)

    if section is not None:
        if section not in sitemaps:
            raise Http404("No sitemap available for section: %r
        maps = [sitemaps[section]]
    else:
        maps = list(six.itervalues(sitemaps))

    urls = []
    for site in maps:
        try:
            if callable(site):
                site = site()
            urls.extend(site.get_urls(page=page, site=req_site,
                                      protocol=req_protocol))
        except EmptyPage:
            raise Http404("Page %s empty" % page)
        except PageNotAnInteger:
            raise Http404("No page '%s'" % page)
    return TemplateResponse(request, template_name, {'urlset':
                            content_type=content_type)
```

如果还是不懂，可以下载附件查看：**zqxt_sitemap.zip**

更多参考：

官方文档：https://docs.djangoproject.com/en/dev/ref/contrib/site

Django ORM

本文介绍如何只使用Django的数据库。

Django 的数据库接口非常好用，我们甚至不需要知道SQL语句如何书写，就可以轻松地查询，创建一些内容，所以有时候想，在其它的地方使用Django的 ORM呢？它有这么丰富的 QuerySet API.

示例代码：Django_DB.zip

settings.py

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
SECRET_KEY = 'at8j8i9%=+m@topzgjzvhs#64^0&qlr6m5yc(_&me%!@jp-7y

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

在这个文件中写上 SQLite, MySQL或PostgreSQL的信息，这样就可以运用这个数据库了。

新建确保每个app下有一个 models.py 和 __init__.py 文件，就可以享受 Django 的 ORM 带来的便利！

可以用 Django QuerySet API 来创建，查询，删除，修改，不用写SQL语句。

更详细的请查看本文提供的源代码。

Django 通用视图

我们用Django开发，比如做一个博客，我们需要做一个博文列表，需要分页，这样我们需要计算出一共有多少篇文章，根据每页显示的博文数量，我们从数据库是显示出相应的页对应的文章，这样使用数据库的查询可以实现，但是这种需求是比较普遍的，所以Django中有更简单的方法来实现，最简单的方法就是使用 generic 类来做。

有时候我们想将一个模板直接显示出来，还不得不写一个视图函数，其实可以用 TemplateView 可以直接写在 urls.py 中，这样的例子还有很多，下面一一介绍：

在urls.py中使用类视图的时候都是调用它的 `.as_view()` 函数

一，Base Views

1. django.views.generic.base.View

这个类是通用类的基类，其它类都是继承自这个类，一般不会用到这个类，个人感觉用函数更简单些。

```
# views.py
from django.http import HttpResponse
from django.views.generic import View

class MyView(View):

    def get(self, request, *args, **kwargs):
        return HttpResponse('Hello, World!')

# urls.py
from django.conf.urls import patterns, url
```

```
from myapp.views import MyView

urlpatterns = patterns('',
    url(r'^mine/$', MyView.as_view(), name='my-view'),
)
```

2. django.views.generic.base.TemplateView

在 get_context_data() 函数中可以用于传递一些额外的内容到网页

```
# views.py

from django.views.generic.base import TemplateView

from articles.models import Article

class HomePageView(TemplateView):

    template_name = "home.html"

    def get_context_data(self, **kwargs):
        context = super(HomePageView, self).get_context_data(**
        context['latest_articles'] = Article.objects.all()[:5]
        return context


# urls.py

from django.conf.urls import patterns, url

from myapp.views import HomePageView

urlpatterns = patterns('',
    url(r'^$', HomePageView.as_view(), name='home'),
)
```

3. django.views.generic.base.RedirectView

用来进行跳转，默认是永久重定向（301），可以直接在urls.py中进行，非常方便：

```
from django.conf.urls import patterns, url
from django.views.generic.base import RedirectView

urlpatterns = patterns('',
    url(r'^go-to-django/$', RedirectView.as_view(url='http://dj
    url(r'^go-to-ziqiangxuetang/$', RedirectView.as_view(url='h
)
```

其它的使用方式：(new in Django1.6)

```
# views.py
from django.shortcuts import get_object_or_404
from django.views.generic.base import RedirectView

from articles.models import Article

class ArticleCounterRedirectView(RedirectView):

    url = ' # 要跳转的网址，
    # url 可以不给，用 pattern_name 和 get_redirect_url() 函数 来解

    permanent = False #是否为永久重定向，默认为 True
    query_string = True # 是否传递GET的参数到跳转网址，True时会传递，
    pattern_name = 'article-detail' # 用来跳转的 URL，看下面的 get


    # 如果url没有设定，此函数就会尝试用pattern_name和从网址中捕捉的参数
    # 即 reverse(pattern_name, args) 得到相应的网址，
    # 在这个例子中是一个文章的点击数链接，点击后文章浏览次数加1，再跳转到真
    def get_redirect_url(self, *args, **kwargs):
         If url is not set, get_redirect_url() tries to reverse
        article = get_object_or_404(Article, pk=kwargs['pk'])
        article.update_counter() # 更新文章点击数，在models.py中实现
        return super(ArticleCounterRedirectView, self).get_redi



# urls.py
```

```
from django.conf.urls import patterns, url
from django.views.generic.base import RedirectView

from article.views import ArticleCounterRedirectView, ArticleDe

urlpatterns = patterns('',

    url(r'^counter/(?P<pk>\d+)/$', ArticleCounterRedirectView.a
    url(r'^details/(?P<pk>\d+)/$', ArticleDetail.as_view(), nam
)
```

二，Generic Display View （通用显示视图）

1. django.views.generic.detail.DetailView

DetailView 有以下方法：

1. dispatch()
2. http_method_not_allowed()
3. get_template_names()
4. get_slug_field()
5. get_queryset()
6. get_object()
7. get_context_object_name()
8. get_context_data()
9. get()
10. render_to_response()

```
# views.py
from django.views.generic.detail import DetailView
from django.utils import timezone

from articles.models import Article

class ArticleDetailView(DetailView):

    model = Article # 要显示详情内容的类

    template_name = 'article_detail.html'
    # 模板名称，默认为 应用名/类名_detail.html（即 app/modelname_det

    # 在 get_context_data() 函数中可以用于传递一些额外的内容到网页
    def get_context_data(self, **kwargs):
        context = super(ArticleDetailView, self).get_context_da
        context['now'] = timezone.now()
        return context


# urls.py
from django.conf.urls import url

from article.views import ArticleDetailView

urlpatterns = [
    url(r'^(?P<slug>[-_\w]+)/$', ArticleDetailView.as_view(), n
]
```

article_detail.html

```
<h1>标题：{{ object.title }}</h1>
<p>内容：{{ object.content }}</p>
<p>发表人：{{ object.reporter }}</p>
<p>发表于：{{ object.pub_date|date }}</p>


<p>日期：{{ now|date }}</p>
```

2. django.views.generic.list.ListView

ListView 有以下方法:

1. dispatch()
2. http_method_not_allowed()
3. get_template_names()
4. get_queryset()
5. get_context_object_name()
6. get_context_data()
7. get()
8. render_to_response()

```
# views.py
from django.views.generic.list import ListView
from django.utils import timezone

from articles.models import Article

class ArticleListView(ListView):

    model = Article

    def get_context_data(self, **kwargs):
        context = super(ArticleListView, self).get_context_data
        context['now'] = timezone.now()
        return context



# urls.py:

from django.conf.urls import url
```

```
from article.views import ArticleListView

urlpatterns = [
    url(r'^$', ArticleListView.as_view(), name='article-list'),
]
```

article_list.html

```
<h1>文章列表</h1>
<ul>
{% for article in object_list %}
    <li>{{ article.pub_date|date }} - {{ article.headline }}</l
{% empty %}
    <li>抱歉，目前还没有文章。</li>
{% endfor %}
</ul>
```

未完待续

Class-based views 官方文档:

https://docs.djangoproject.com/en/dev/ref/class-based-views/#built-in-class-based-views-api

---

Django 中间件

我们从浏览器发出一个请求 Request，得到一个响应后的内容 HttpResponse ， 这个请求传递到 Django的过程如下：

也就是说，每一个请求都是先通过中间件中的 process_request 函数，这个函数返回 None 或者 HttpResponse 对象，如果返回前者，继续处理其它中间件，如果返回一个 HttpResponse，就处理中止，返回到网页上。

中间件不用继承自任何类（可以继承 object ），下面一个中间件大概的样子：

```
class CommonMiddleware(object):
    def process_request(self, request):
        return None

    def process_response(self, request, response):
        return response
```

还有 process_view, process_exception 和 process_template_response 函数。

一，比如我们要做一个 拦截器，发生有恶意访问网站的人，就拦截他！

假如我们通过一种技术，**比如统计一分钟访问页面数，太多就把他的 IP 加入到黑名单 BLOCKED_IPS（这部分没有提供代码，主要讲中间件部分）**

```
#项目 zqxt 文件名 zqxt/middleware.py

class BlockedIpMiddleware(object):
    def process_request(self, request):
        if request.META['REMOTE_ADDR'] in getattr(settings, "BI
            return http.HttpResponseForbidden('<h1>Forbidden</h
```

这里的代码的功能就是 获取当前访问者的 IP (request.META['REMOTE_ADDR'])，如果这个 IP 在黑名单中就拦截，如果不在就返回 None (函数中没有返回值其实就是默认为 None)，把这个中间件的 Python 路径写到settings.py中

```
MIDDLEWARE_CLASSES = (
    'zqxt.middleware.BlockedIpMiddleware',
    ...其它的中间件
```

)

Django 会从 MIDDLEWARE_CLASSES 中按照从上到下的顺序一个个执行中间件中的 process_request 函数，而其中 process_response 函数则是最前面的最后执行。

二，再比如，我们在网站放到服务器上正式运行后，DEBUG改为了False，这样更安全，但是有时候发生错误不能显示错误详情页面，有没有办法处理好这两个事情呢?

1. 普通访问者看到的是友好的报错信息
2. 管理员看到的是错误详情，以便于修复 BUG

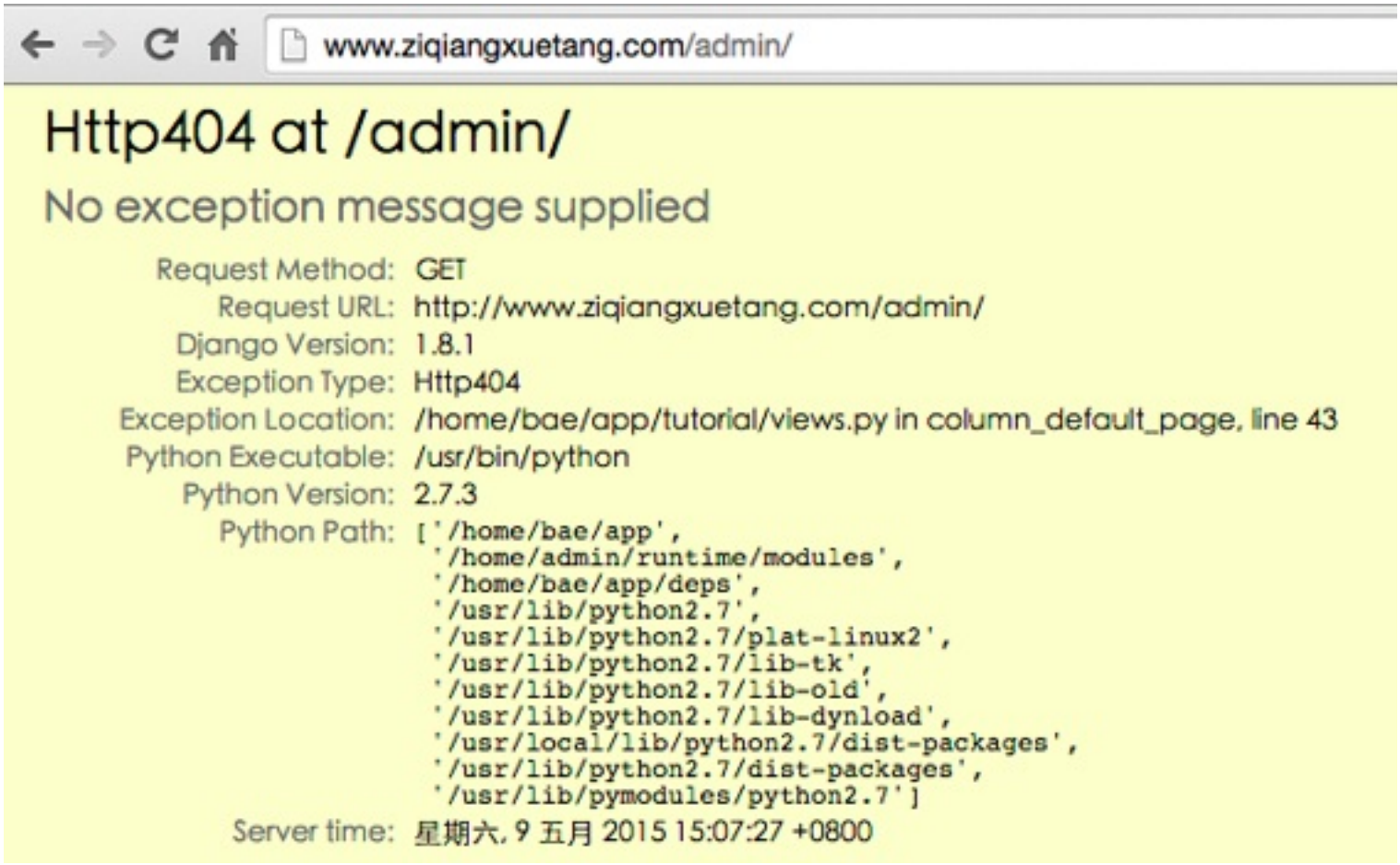
当然可以有，利用中间件就可以做到！代码如下：

```
import sys
from django.views.debug import technical_500_response
from django.conf import settings

class UserBasedExceptionMiddleware(object):
    def process_exception(self, request, exception):
        if request.user.is_superuser or request.META.get('REMOT
            return technical_500_response(request, *sys.exc_inf
```

把这个中间件像上面一样，加到你的 settings.py 中的 MIDDLEWARE_CLASSES 中，可以放到最后，这样可以看到其它中间件的 process_request的错误。

当访问者为管理员时，就给出错误详情，比如访问本站的不存在的页面：http://www.ziqiangxuetang.com/admin/

普通人看到的是普通的 404（自己点开看看），而我可以看到：



三，分享一个简单的识别手机的中间件，更详细的可以参考这个：django-mobi 或 django-mobile

```
MOBILE_USERAGENTS = ("2.0 MMP","240x320","400X240","AvantGo","B
    "Blazer","Cellphone","Danger","DoCoMo","Elaine/3.0","Eudora
    "Googlebot-Mobile","hiptop","IEMobile","KYOCERA/WX310K","LG
    "MIDP-2.","MMEF20","MOT-V","NetFront","Newt","Nintendo Wii"
    "Nokia","Opera Mini","Palm","PlayStation Portable","portalm
    "ProxiNet","SHARP-TQ-GX10","SHG-i900","Small","SonyEricsson
    "SymbianOS","TS21i-10","UP.Browser","UP.Link","webOS","Wind
    "WinWAP","YahooSeeker/M1A1-R2D2","iPhone","iPod","Android",
    "BlackBerry9530","LG-TU915 Obigo","LGE VX","webOS","Nokia58

class MobileTemplate(object):
    """
    If a mobile user agent is detected, inspect the default arg
    func, and if a template name is found assume it is the temp
    attempt to load a mobile template based on the original tem
    """

    def process_view(self, request, view_func, view_args, view_
```

```
        if any(ua for ua in MOBILE_USERAGENTS if ua in
            request.META["HTTP_USER_AGENT"]):
            template = view_kwargs.get("template")
            if template is None:
                for default in view_func.func_defaults:
                    if str(default).endswith(".html"):
                        template = default
            if template is not None:
                template = template.rsplit(".html", 1)[0] + ".m
                try:
                    get_template(template)
                except TemplateDoesNotExist:
                    pass
                else:
                    view_kwargs["template"] = template
                    return view_func(request, *view_args, **vie
        return None
```

参考文档：https://docs.djangoproject.com/en/1.8/topics/http/mid

Python/Django 微信接口

注册或登陆 微信公众平台 点击左侧的 开发者中心

填写相应的网址，Token(令牌) 是随便写的，你自己想写什么就写什么，微信验证时检验是否写的和服务器上的TOKEN一样，一样则通过。

关注一下自强学堂的微信号吧，可以随时随地查阅教程哦，体验一下自强学堂的微信的各种功能再阅读效果更佳!

自己动手写微信的验证： views.py

```
#coding=utf-8
import hashlib
import json
from lxml import etree
from django.utils.encoding import smart_str
from django.views.decorators.csrf import csrf_exempt
from django.http import HttpResponse
from auto_reply.views import auto_reply_main # 修改这里

WEIXIN_TOKEN = 'write-a-value'

@csrf_exempt
def weixin_main(request):
```

```
    """
    所有的消息都会先进入这个函数进行处理，函数包含两个功能，
    微信接入验证是GET方法，
    微信正常的收发消息是用POST方法。
    """
    if request.method == "GET":
        signature = request.GET.get("signature", None)
        timestamp = request.GET.get("timestamp", None)
        nonce = request.GET.get("nonce", None)
        echostr = request.GET.get("echostr", None)
        token = WEIXIN_TOKEN
        tmp_list = [token, timestamp, nonce]
        tmp_list.sort()
        tmp_str = "%s%s%s" % tuple(tmp_list)
        tmp_str = hashlib.sha1(tmp_str).hexdigest()
        if tmp_str == signature:
            return HttpResponse(echostr)
        else:
            return HttpResponse("weixin  index")
    else:
        xml_str = smart_str(request.body)
        request_xml = etree.fromstring(xml_str)
        response_xml = auto_reply_main(request_xml)# 修改这里
        return HttpResponse(response_xml)
```

auto_reply_main 是用来处理消息，回复消息的，需要自己进一步完善。

使用第三方包实现：

关于Django开发微信，有已经做好的现在的包可以使用 wechat_sdk 这个包，使用文档 也比较完善，但是在处理加密一部分没有做，在微信公众平台上，需要用**明文验证**，如果要加密，自己参照微信官网的加密算法。

使用 wechat_sdk 的例子（自强学堂微信号简化后的例子）：

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.http.response import HttpResponse, HttpResponseBadR
from django.views.decorators.csrf import csrf_exempt

from wechat_sdk import WechatBasic
from wechat_sdk.exceptions import ParseError
from wechat_sdk.messages import TextMessage


WECHAT_TOKEN = 'zqxt'
AppID = ''
AppSecret = ''

# 实例化 WechatBasic
wechat_instance = WechatBasic(
    token=WECHAT_TOKEN,
    appid=AppID,
    appsecret=AppSecret
)

@csrf_exempt
def index(request):
    if request.method == 'GET':
        # 检验合法性
        # 从 request 中提取基本信息 (signature, timestamp, nonce,
        signature = request.GET.get('signature')
        timestamp = request.GET.get('timestamp')
        nonce = request.GET.get('nonce')

        if not wechat_instance.check_signature(
                signature=signature, timestamp=timestamp, nonce
            return HttpResponseBadRequest('Verify Failed')

        return HttpResponse(
            request.GET.get('echostr', ''), content_type="text/


    # 解析本次请求的 XML 数据
    try:
```

```
        wechat_instance.parse_data(data=request.body)
    except ParseError:
        return HttpResponseBadRequest('Invalid XML Data')

    # 获取解析好的微信请求信息
    message = wechat_instance.get_message()

    # 关注事件以及不匹配时的默认回复
    response = wechat_instance.response_text(
        content = (
            '感谢您的关注！\n回复【功能】两个字查看支持的功能，还可以回复
            '\n【<a href="http://www.ziqiangxuetang.com">自强学堂
            ))
    if isinstance(message, TextMessage):
        # 当前会话内容
        content = message.content.strip()
        if content == '功能':
            reply_text = (
                    '目前支持的功能：\n1. 关键词后面加上【教程】两个字
                    '比如回复 "Django 后台教程"\n'
                    '2. 回复任意词语，查天气，陪聊天，讲故事，无所不能!
                    '还有更多功能正在开发中哦 ^_^\n'
                    '【<a href="http://www.ziqiangxuetang.com">
                )
        elif content.endswith('教程'):
            reply_text = '您要找的教程如下：'

        response = wechat_instance.response_text(content=reply_

    return HttpResponse(response, content_type="application/xml
```

下面是一个更详细复杂的使用例子：

models.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals
```

```
from django.db import models


class KeyWord(models.Model):
    keyword = models.CharField(
        '关键词', max_length=256, primary_key=True, help_text='月
    content = models.TextField(
        '内容', null=True, blank=True, help_text='回复给用户的内容

    pub_date = models.DateTimeField('发表时间', auto_now_add=Tru
    update_time = models.DateTimeField('更新时间', auto_now=True
    published = models.BooleanField('发布状态', default=True)

    def __unicode__(self):
        return self.keyword

    class Meta:
        verbose_name='关键词'
        verbose_name_plural=verbose_name
```

views.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.http.response import HttpResponse, HttpResponseBadR
from django.views.decorators.csrf import csrf_exempt

from wechat_sdk import WechatBasic
from wechat_sdk.exceptions import ParseError
from wechat_sdk.messages import (TextMessage, VoiceMessage, Ima
    VideoMessage, LinkMessage, LocationMessage, EventMessage
)

from wechat_sdk.context.framework.django import DatabaseContext
from .models import KeyWord as KeyWordModel
```

```
# 实例化 WechatBasic
wechat_instance = WechatBasic(
    token='zqxt',
    appid='xx',
    appsecret='xx'
)


@csrf_exempt
def index(request):
    if request.method == 'GET':
        # 检验合法性
        # 从 request 中提取基本信息 (signature, timestamp, nonce,
        signature = request.GET.get('signature')
        timestamp = request.GET.get('timestamp')
        nonce = request.GET.get('nonce')

        if not wechat_instance.check_signature(
                signature=signature, timestamp=timestamp, nonce
            return HttpResponseBadRequest('Verify Failed')

        return HttpResponse(
            request.GET.get('echostr', ''), content_type="text/

    # POST
    # 解析本次请求的 XML 数据
    try:
        wechat_instance.parse_data(data=request.body)
    except ParseError:
        return HttpResponseBadRequest('Invalid XML Data')

    # 获取解析好的微信请求信息
    message = wechat_instance.get_message()
    # 利用本次请求中的用户OpenID来初始化上下文对话
    context = DatabaseContextStore(openid=message.source)

    response = None

    if isinstance(message, TextMessage):
        step = context.get('step', 1)  # 当前对话次数，如果没有则返
        # last_text = context.get('last_text')  # 上次对话内容
```

```
content = message.content.strip()  # 当前会话内容

if message.content == '新闻':
    response = wechat_instance.response_news([
        {
            'title': '自强学堂',
            'picurl': 'http://www.ziqiangxuetang.com/st
            'description': '自强学堂致力于提供优质的IT技术教
            'url': 'http://www.ziqiangxuetang.com',
        }, {
            'title': '百度',
            'picurl': 'http://doraemonext.oss-cn-hangzh
            'url': 'http://www.baidu.com',
        }, {
            'title': 'Django 教程',
            'picurl': 'http://www.ziqiangxuetang.com/me
            'url': 'http://www.ziqiangxuetang.com/djang
        }
    ])
    return HttpResponse(response, content_type="applica

else:
    try:
        keyword_object = KeyWordModel.objects.get(keywo
        reply_text = keyword_object.content
    except KeyWordModel.DoesNotExist:
        try:
            reply_text = KeyWordModel.objects.get(keywo
        except KeyWordModel.DoesNotExist:
            reply_text = ('/:P-(好委屈，数据库翻个遍也没找到
                '试试下面这些关键词吧：\nKEYWORD_LIST\n'
                '<a href="http://www.rxnfinder.org">Rxn
                '感谢您的支持！/:rose'
                )

# 将新的数据存入上下文对话中
context['step'] = step + 1
context['last_text'] = content
context.save()  # 非常重要！请勿忘记！临时数据保存入数据库！

if 'KEYWORD_LIST' in reply_text:
```

```
            keyword_objects = KeyWordModel.objects.exclude(keyw
                '关注事件', '测试', 'test', '提示']).filter(publis
            keywords = ('{}. {}'.format(str(i), k.keyword)
                        for i, k in enumerate(keyword_objects,
            reply_text = reply_text.replace(
                'KEYWORD_LIST', '\n'.join(keywords))

        # 文本消息结束

    elif isinstance(message, VoiceMessage):
        reply_text = '语音信息我听不懂/:P-(/:P-(/:P-('
    elif isinstance(message, ImageMessage):
        reply_text = '图片信息我也看不懂/:P-(/:P-(/:P-('
    elif isinstance(message, VideoMessage):
        reply_text = '视频我不会看/:P-('
    elif isinstance(message, LinkMessage):
        reply_text = '链接信息'
    elif isinstance(message, LocationMessage):
        reply_text = '地理位置信息'
    elif isinstance(message, EventMessage):  # 事件信息
        if message.type == 'subscribe':  # 关注事件(包括普通关注事
            follow_event = KeyWordModel.objects.get(keyword='关
            reply_text = follow_event.content

            # 如果 key 和 ticket 均不为空，则是扫描二维码造成的关注事
            if message.key and message.ticket:
                reply_text += '\n来源：扫描二维码关注'
            else:
                reply_text += '\n来源：搜索名称关注'
        elif message.type == 'unsubscribe':
            reply_text = '取消关注事件'
        elif message.type == 'scan':
            reply_text = '已关注用户扫描二维码！'
        elif message.type == 'location':
            reply_text = '上报地理位置'
        elif message.type == 'click':
            reply_text = '自定义菜单点击'
        elif message.type == 'view':
            reply_text = '自定义菜单跳转链接'
        elif message.type == 'templatesendjobfinish':
            reply_text = '模板消息'
```

```
    response = wechat_instance.response_text(content=reply_text
    return HttpResponse(response, content_type="application/xml
```

Django 单元测试

自强学堂Django一系列教程，前面的例子都是我们写好代码后，运行开发服务器，在浏览器上自己点击测试，看写的代码是否正常，但是这样做很麻烦，**因为以后如果有改动，可能会影响以前本来正常的功能，这样以前的功能又得测试一遍**，非常不方便，Django中有完善的单元测试，**我们可以对开发的每一个功能进行单元测试，这样只要运行一个命令 `python manage.py test`，就可以测试功能是否正常。**

一言以蔽之，测试就是检查代码是否按照自己的预期那样运行。

测试驱动开发：有时候，我们知道自己需要的功能（结果），并不知道代码如何书写，这时候就可以利用测试驱动开发（Test Driven Development），先写出我们期待得到的结果（把测试代码先写出来），再去完善代码，直到不报错，我们就完成了。

《改善Python的91个建议》一书中说：单元测试绝不是浪费时间的无用功，它是高质量代码的保障之一，在软件开发的一节中值得投入精力和时间去把好这一关。

1. Python 中 单元测试简介：

下面是一个 Python的单元测试简单的例子：

假如我们开发一个除法的功能，有的同学可能觉得很简单，代码是这样的：

```
def division_funtion(x, y):
    return x / y
```

但是这样写究竟对还是不对呢，有些同学可以在代码下面这样测试：

```
def division_funtion(x, y):
    return x / y


if __name__ == '__main__':
    print division_funtion(2, 1)
    print division_funtion(2, 4)
    print division_funtion(8, 3)
```

但是这样运行后得到的结果，自己每次都得算一下去核对一遍，很不方便，Python中有 unittest 模块，可以很方便地进行测试，详情可以看官网文档。

Python 2 (https://docs.python.org/2/library/unittest.html)

Python 3 (https://docs.python.org/3/library/unittest.html)

下面是一个简单的示例：

```
import unittest


def division_funtion(x, y):
    return x / y


class TestDivision(unittest.TestCase):
    def test_int(self):
        self.assertEqual(division_funtion(9, 3), 3)

    def test_int2(self):
        self.assertEqual(division_funtion(9, 4), 2.25)
```

```
    def test_float(self):
        self.assertEqual(division_funtion(4.2, 3), 1.4)


if __name__ == '__main__':
    unittest.main()
```

我简单地写了三个测试示例（不一定全面，只是示范，比如没有考虑除数是0的情况），运行后发现：

```
F.F
======================================================================
FAIL: test_float (__main__.TestDivision)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/tu/YunPan/mydivision.py", line 16, in test_float
    self.assertEqual(division_funtion(4.2, 3), 1.4)
AssertionError: 1.4000000000000001 != 1.4

======================================================================
FAIL: test_int2 (__main__.TestDivision)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/tu/YunPan/1.py", line 13, in test_int2
    self.assertEqual(division_funtion(9, 4), 2.25)
AssertionError: 2 != 2.25

----------------------------------------------------------------------
Ran 3 tests in 0.001s

FAILED (failures=2)
```

汗！发现了没，竟然两个都失败了，测试发现：

4.2除以3 等于 1.4000000000000001 不等于期望值 1.4

9除以4等于2，不等于期望的 2.25

下面我们就是要修复这些问题，再次运行测试，直到运行不报错为

止。

譬如根据实际情况，假设我们只需要保留到小数点后6位，可以这样改：

```
def division_funtion(x, y):
    return round(float(x) / y, 6)
```

再次运行就不报错了：

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

2. Django 中 单元测试：(不断完善中，后期会增加对前面讲解的内容的测试)

2.1 简单测试例子：

```
from django.test import TestCase
from myapp.models import Animal


class AnimalTestCase(TestCase):
    def setUp(self):
        Animal.objects.create(name="lion", sound="roar")
        Animal.objects.create(name="cat", sound="meow")

    def test_animals_can_speak(self):
        """Animals that can speak are correctly identified"""
        lion = Animal.objects.get(name="lion")
        cat = Animal.objects.get(name="cat")
        self.assertEqual(lion.speak(), 'The lion says "roar"')
```

```
        self.assertEqual(cat.speak(), 'The cat says "meow"')
```

这个例子是测试myapp.models 中的 Animal 类相关的方法功能。

2.2 用代码访问网址的方法：

```
>>> from django.test import Client
>>> c = Client()
>>> response = c.post('/login/', {'username': 'john', 'password
>>> response.status_code
200
>>> response = c.get('/customer/details/')
>>> response.content
'<!DOCTYPE html...'
```

我们可以用 django.test.Client 的实例来实现 get 或 post 内容，检查一个网址返回的网页源代码。

默认情况下CSRF检查是被禁用的，如果测试需要，可以用下面的方法：

```
>>> from django.test import Client
>>> csrf_client = Client(enforce_csrf_checks=True)
```

使用 csrf_client 这个实例进行请求即可。

指定浏览USER-AGENT:

```
>>> c = Client(HTTP_USER_AGENT='Mozilla/5.0')
```

模拟post上传附件：

```
from django.test import Client
c = Client()

with open('wishlist.doc') as fp:
    c.post('/customers/wishes/', {'name': 'fred', 'attachment':
```

测试网页返回状态：

```
from django.test import TestCase

class SimpleTest(TestCase):
    def test_details(self):
        response = self.client.get('/customer/details/')
        self.assertEqual(response.status_code, 200)

    def test_index(self):
        response = self.client.get('/customer/index/')
        self.assertEqual(response.status_code, 200)
```

我们用 self.client 即可，不用 client = Client() 这样实例化，更方便，我们还可以继承 Client，添加一些其它方法：

```
from django.test import TestCase, Client

class MyTestClient(Client):
    # Specialized methods for your environment
    ...

class MyTest(TestCase):
    client_class = MyTestClient

    def test_my_stuff(self):
        # Here self.client is an instance of MyTestClient...
        call_some_test_code()
```

定制 self.client 的方法：

```
from django.test import Client, TestCase


class MyAppTests(TestCase):
    def setUp(self):
        super(MyAppTests, self).setUp()
        self.client = Client(enforce_csrf_checks=True)
```

```
    def test_home(self):
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
```

Django 开发内容管理系统

用Django开发一个简易的内容管理系统，比如显示新闻的列表，点击进去可以看内容详情等，新闻发布网站。

一，搭建互不干扰的 Python 包开发环境

我们有的时候会发现，一个电脑上有多个项目，一个依赖 Django 1.8，另一个比较旧的项目又要用 Django 1.5，这时候怎么办呢?

我们需要一个依赖包管理的工具来处理不同的环境。**如果不想搭建这个环境，可以直接去看 2.2**

1.1 环境搭建

开发会用 virtualenv 来管理多个开发环境，virtualenvwrapper 使得virtualenv变得更好用

```
# 安装:
(sudo) pip install virtualenv virtualenvwrapper
```

Linux/Mac OSX 下:

修改~/.bash_profile或其它环境变量相关文件，添加以下语句

```
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/workspace
source /usr/local/bin/virtualenvwrapper.sh
```

Windows 下:

```
pip install virtualenvwrapper-win
```

【可选】Windows下默认虚拟环境是放在用户名下面的Envs中的，与桌面，我的文档，下载等文件夹在一块的。更改方法：计算机，属性，高级系统设置，环境变量，添加WORKON_HOME，如图（windows 10 环境变量设置截图）：

1.2 使用方法：

mkvirtualenv zqxt：创建运行环境**zqxt**

workon zqxt: 工作在 **zqxt** 环境

其它的：

rmvirtualenv ENV：删除运行环境ENV

mkproject mic：创建mic项目和运行环境mic

mktmpenv：创建临时运行环境

lsvirtualenv: 列出可用的运行环境

lssitepackages: 列出当前环境安装了的包

创建的环境是独立的，互不干扰，无需sudo权限即可使用 pip 来进行包的管理。

二，安装软件，开发 minicms 项目

2.1 创建一个开发环境 minicms

```
# Linux 或 Mac OSX
```

```
mkproject minicms

# windows
mkvirtualenv minicms
```

Mac OSX 下的输出示例：

```
mkproject minicms

New python executable in minicms/bin/python
Installing setuptools, pip...done.
Creating /Users/tu/YunPan/workspace//minicms
Setting project for minicms to /Users/tu/YunPan/workspace/minic
(minicms)tu@mac ~/YunPan/workspace/minicms $
```

2.2 安装 Django

```
pip install Django==1.8.3
```

2.3 创建项目 minicms 和 应用 news

```
django-admin.py startproject minicms
cd minicms
python manage.py startapp news
```

添加 news 到 settings.py 中的 INSTALLED_APPS 中。

2.4 规划 news 中的栏目和每篇文章相关的字段

栏目：名称，网址，简介等

文章：标题，作者，网址，内容等

我们假设一篇文章只有一个作者（文章和作者是多对一的关系），一篇文章可以属于多个栏目（栏目和文章是**多对多**的关系）

为了用到更多的情况，我们假设作者可以为空，栏目不能为空。

开写 models.py

```
# -*- coding: utf-8 -*-
from __future__ import unicode_literals

from django.db import models
from django.utils.encoding import python_2_unicode_compatible


@python_2_unicode_compatible
class Column(models.Model):
    name = models.CharField('栏目名称', max_length=256)
    slug = models.CharField('栏目网址', max_length=256, db_index
    intro = models.TextField('栏目简介', default='')

    def __str__(self):
        return self.name

    class Meta:
        verbose_name = '栏目'
        verbose_name_plural = '栏目'
        ordering = ['name']  # 按照哪个栏目排序


@python_2_unicode_compatible
class Article(models.Model):
    column = models.ManyToManyField(Column, verbose_name='归属栏
```

```
    title = models.CharField('标题', max_length=256)
    slug = models.CharField('网址', max_length=256, db_index=Tru

    author = models.ForeignKey('auth.User', blank=True, null=Tr
    content = models.TextField('内容', default='', blank=True)

    published = models.BooleanField('正式发布', default=True)

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = '教程'
        verbose_name_plural = '教程'
```

2.5 创建数据库

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

2.6 创建完数据库后，用了一段时间，我们发现以前的文章的字段不合理

比如我们想记录文章添加的日期，修改的日期，我们更改 models.py （不变动的大部分省去了，添加两个字段）

```
...省略
class Article(models.Model):
    ...原来的字段省去

    pub_date = models.DateTimeField('发表时间', auto_now_add=Tru
    update_time = models.DateTimeField('更新时间', auto_now=True

    ...省略
```

这时候，我们对 models.py 进行了更改，这些字段数据库中还没有，我们要同步更改到数据库中去：

```
python manage.py makemigrations news
```

You are trying to add a non-nullable field 'pub_date' to article without a default; we can't do that (the database needs something to populate existing rows).

Please select a fix:

 1) Provide a one-off default now (will be set on all existing rows)

 2) Quit, and let me add a default in models.py

这段话的意思是 pub_date 字段没有默认值，而且非Null 那么

1) 指定一个一次性的值供更改数据库时使用。

2) 停止当前操作，在 models.py 中给定默认值，然后再来 migrate。

我们选择第一个，输入 **1**

Select an option: 1

Please enter the default value now, as valid Python

The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now()

>>> **timezone.now()**

Migrations for 'news':

  0002_auto_20150728_1232.py:

- Add field pub_date to article

- Add field update_time to article

这样是生成了一个对表进行更改的 py 文件在 news/migrations 文件夹中，我们要执行更改

```
python manage.py migrate 或 python manage.py migrate news
```

2.7 创建一个脚本，导入一些数据到数据库中

我们导入一些演示数据：

栏目： [<Column: 体育新闻>, <Column: 社会新闻>, <Column: 科技新闻>]

文章：[<Article: 体育新闻_1>, <Article: 体育新闻_2>, <Article: 体育新闻_3>, <Article: 体育新闻_4>, <Article: 体育新闻_5>, <Article: 体育新闻会>, <Article: 体育新闻_7>, <Article: 体育新闻_8>, <Article: 体育新闻_9>, <Article: 体育新闻_10>, <Article: 社会新闻_1>, <Article: 社新闻_2>,'... (remaining elements truncated)...']

create_demo_records.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# @Date    : 2015-07-28 20:38:38
# @Author  : Weizhong Tu (mail@tuweizhong.com)
# @Link    : http://www.tuweizhong.com

'''
create some records for demo database
'''
```

```
from minicms.wsgi import *
from news.models import Column, Article


def main():
    columns_urls = [
      ('体育新闻', 'sports'),
      ('社会新闻', 'society'),
      ('科技新闻', 'tech'),
    ]

    for column_name, url in columns_urls:
        c = Column.objects.get_or_create(name=column_name, slug

        # 创建 10 篇新闻
        for i in range(1, 11):
            article = Article.objects.get_or_create(
                title='{}_{}'.format(column_name, i),
                slug='article_{}'.format(i),
                content='新闻详细内容： {} {}'.format(column_name
            )[0]
            article.column.add(c)


if __name__ == '__main__':
    main()
    print("Done!")
```

假设这个文件被保存为 create_demo_records.py （和 manage.py 放在一块，同一个文件夹下）

运行脚本 导入数据：

```
python create_demo_records.py
```

Done!

终端上显示一个 Done! 就这样 Duang 的一下，数据就导进去

了！

上面所有操作的代码：  minicms1.zip

第一次提交：github: https://github.com/twz915/django-minicms/tree/dff3175817385 2344af5d8d5b4fad858a0b169(

内容管理系统继续开发，点此查看第二部分

Django 基础教程

Django 是由 Python 开发的一个免费的开源网站框架，可以用于快速搭建高性能，优雅的网站！

你一定可以学会，Django 很简单！本教程一直在更新，从开始写到现在大概写了一年多了，现在也一直在坚持写，每一篇教程都可能随时更新，可以在网站首页看到最近更新的情况。

我阅读学习了全部的 Django英文的官方文档，觉得国内比较好的Django学习资源不多，所以决定写自己的教程。本教程开始写的时候是 Django 的版本是 1.6，Django的更新很快，自强学堂的教程也随着更新了，兼顾了后来的新版本，从 Django 1.5 到最新的 Django 1.8 中应该都没有问题。

自强学堂 就是用 Django 搭建的站点！

本教程作者： **涂伟忠(未经同意,禁止转载!)**

除了本教程外的其它教程

自强学堂 学习分享 的一篇文章：**Django 学习资源**，如果有更好的教程也可以在文章下推荐哦！

开发过程中遇到的各种问题可以在这里提问（或者直接在相应的教程下回复该教程相关的问题）： Django 开发常见问题及答案汇总

学Django需要什么基础

1. Django是 python 语言写的一个网络框架的包，所以你得知道一些 Python 基础知识。

2. 其次你最好有一些做网站的经验，懂一些网页 HTML, CSS, JavaScript 的知识。

没有经验也没有关系，慢慢来就好了，你一定可以学会，Django 很简单！

Django 特点

强大的数据库功能

用python的类继承，几行代码就可以拥有一个丰富，动态的数据库操作接口（API），如果需要你也能执行SQL语句

自带的强大的后台功能

几行简单的代码就让你的网站拥有一个强大的后台，轻松管理你的内容！

优雅的网址

用正则匹配网址，传递到对应函数，随意定义，如你所想！

模板系统

强大，易扩展的模板系统，设计简易，代码，样式分开设计，更容易管理。

缓存系统

与memcached或其它的缓存系统联用，更出色的表现，更快的加载速度。

国际化

完全支持多语言应用，允许你定义翻译的字符，轻松翻译成不同国家的语言。

Django CMS

CMS 的意思是 Content Management System 内容管理系统，一般拿就可以使用，不会编程也能做出网站来，还可以在原来的基础上再次开发，减少工作量，这里列举了一些出名的 CMS:

- Opps - A content management platform built for large portals.
- django-cms - The easy-to-use and developer-friendly CMS.

  这个不是为初学者开发的，想在基础上开发需要对Django比较了解才行，是一个基本的框架。
- mezzanine - A content management platform built using the Django framework.

  简单易用，自带Blog和用户注册系统，拿来就可以用
- wagtail - A new Django content management system.
- django-fiber - Django Fiber, a simple, user-friendly CMS for all your Django projects

Python/Django 生成二维码

一，包的安装和简单使用

1.1 用Python来生成二维码很简单，可以看 qrcode 这个包：

```
pip install qrcode
```

qrcode 依赖 Image 这个包：

```
pip install Image
```

如果这个包安装有困难，可选纯Python的包来实现此功能，见下文。

1.2 安装后就可以使用了，这个程序带了一个 qr 命令：

```
qr 'http://www.ziqiangxuetang.com' > test.png
```

1.3 下面我们看一下如何在 代码 中使用

```
import qrcode

img = qrcode.make('http://www.tuweizhong.com')
# img <qrcode.image.pil.PilImage object at 0x1044ed9d0>

with open('test.png', 'wb') as f:
    img.save(f)
```

这样就可以生成一个带有网址的二维码，但是这样得把文件保存到硬盘中。

【备注】：纯Python的包的使用：

安装：

```
pip install git+git://github.com/ojii/pymaging.git#egg=pymaging
pip install git+git://github.com/ojii/pymaging-png.git#egg=pyma
```

使用方法大致相同，命令行上：

```
qr --factory=pymaging "Some text" > test.png
```

Python中调用：

```
import qrcode
from qrcode.image.pure import PymagingImage
img = qrcode.make('Some data here', image_factory=PymagingImage
```

二，Django 中使用

我们可以用 Django 直接把生成的内容返回到网页，以下是操作过程：

2.1 新建一个 zqxtqrcode 项目，tools 应用：

```
django-admin.py startproject zqxtqrcode
python manage.py startapp tools
```

2.2 将 tools 应用 添加到 项目 settings.py 中

```
INSTALLED_APPS = (

    ...

    'tools',
)
```

2.3 我们修改 tools/views.py

```
from django.http import HttpResponse
import qrcode
from cStringIO import StringIO


def generate_qrcode(request, data):
    img = qrcode.make(data)

    buf = StringIO()
    img.save(buf)
    image_stream = buf.getvalue()

    response = HttpResponse(image_stream, content_type="image/p
    response['Last-Modified'] = 'Mon, 27 Apr 2015 02:05:03 GMT'
    response['Cache-Control'] = 'max-age=31536000'
    return response
```

上面对返回结果进行了处理，浏览器会缓存图片，提高再次加载的速度。

2.4 添加视图函数到 zqxtqrcode/urls.py

```
    url(r'^qrcode/(.+)$', 'tools.views.generate_qrcode', name='
```

2.5 同步数据库，打开开发服务器：

```
python manage.py syncdb
python manage.py runserver
```

打开：http://127.0.0.1:8000/qrcode/http://www.tuweizhong.c 就可以看到如下效果：

这样生成 二维码的接口就写好了 ^_^，实例采用的是返回图片流的方式，这样不用写文件到硬盘，接口调用更方便，如果要加速，可以

用Django缓存来实现。

源代码下载：

基于 Django 1.8，tools app 可以在 Django 1.4-Django1.8之间使用，更低版本的自测，应该也没什么问题，建议按教程步骤来一遍，这样学的更好

zqxtqrcode.zip

参考：https://pypi.python.org/pypi/qrcode/
