**写作目的**

喜欢一个学习观点以教促学, 一直以来, 学习的时候经常会发现, 某个方法某个问题自己已经明白了, 但是在教给别人的时候确说不清楚, 所以慢慢的学会了以教促学这种方法, 在教给别人知识的同时也能够提升自己对语言, 对框架的理解.

希望达到的目标:

* 希望能写出一个系列文章, 我也不知道到底能写多少
* 能够让认真阅读这个系列的文章的人, 能在读完之后做出一个简单的博客
* 教会读者使用简单的git操作和github
* 希望能够加深自己对Django的理解

**Django简介**

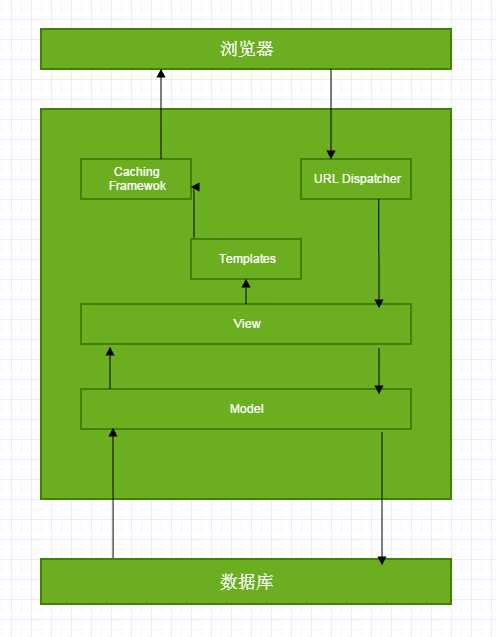
[Django](http://www.jianshu.com/p/d15188a74104)是python中目前风靡的Web Framework, 那么什么叫做Framework呢, 框架能够帮助你把程序的整体架构搭建好, 而我们所需要做的工作就是填写逻辑, 而框架能够在合适的时候调用你写的逻辑, 而不需要我们自己去调用逻辑, 让Web开发变的更敏捷.

Django是一个高级Python Web框架, 鼓励快速,简洁, 以程序设计的思想进行开发. 通过使用这个框架, 可以减少很多开发麻烦, 使你更专注于编写自己的app, 而不需要重复造轮子. Django免费并且开源.

**Django特点**

* 完全免费并开源源代码
* 快速高效开发
* 使用MTV架构(熟悉Web开发的应该会说是MVC架构)
* 强大的可扩展性.

**Django工作方式**



工作方式

用户在浏览器中输入URL后的回车, 浏览器会对URL进行检查, 首先判断协议,如果是http就按照 Web 来处理, 然互调用DNS查询, 将域名转换为IP地址, 然后经过网络传输到达对应Web服务器, 服务器对url进行解析后, 调用View中的逻辑(MTV中的V), 其中又涉及到Model(MTV中的M), 与数据库的进行交互, 将数据发到Template(MTV中的T)进行渲染, 然后发送到浏览器中, 浏览器以合适的方式呈现给用户

通过文字和图的结合希望读者能够初步理解Django的工作方式、

**开发环境**

下面仅仅是我的项目开发环境, 没有必要追求完全一致...

Mac OS X 10.10.1 #非必要

Python3.4.1

Django1.7.1

Bootstrap3.3.0 or Pure(临时决定使用的, @游逸 推荐) #非必要

Sublime Text 3 #非必要

virtualenv 1.11.6黄晶

# 虚拟环境配置

使用virtualenv创建虚拟环境, Ubuntun和Mac安装程序基本一致

#安装virtualenv

$ pip install virtualenv

#创建虚拟环境

$ virtualenv -p /usr/local/bin/python3.4 ENV3.4

Running virtualenv with interpreter /usr/local/bin/python3.4

Using base prefix '/Library/Frameworks/Python.framework/Versions/3.4'

New python executable in ENV3.4/bin/python3.4

Also creating executable in ENV3.4/bin/python

Installing setuptools, pip...done.

#激活虚拟环境

$ source /ENV3.4/bin/activate

#查看当前环境下的安装包

$ pip list

pip (1.5.6)

setuptools (3.6)

更多virtualenv使用可以参考[Virtualenv简明教程](http://andrewliu.tk/2014/12/08/Virtualenv%E7%AE%80%E6%98%8E%E6%95%99%E7%A8%8B/)

# Git安装

Git是目前世界上最先进的分布式版本控制系统

**Mac下git安装**

$ brew install git

**Ubuntu下git安装**

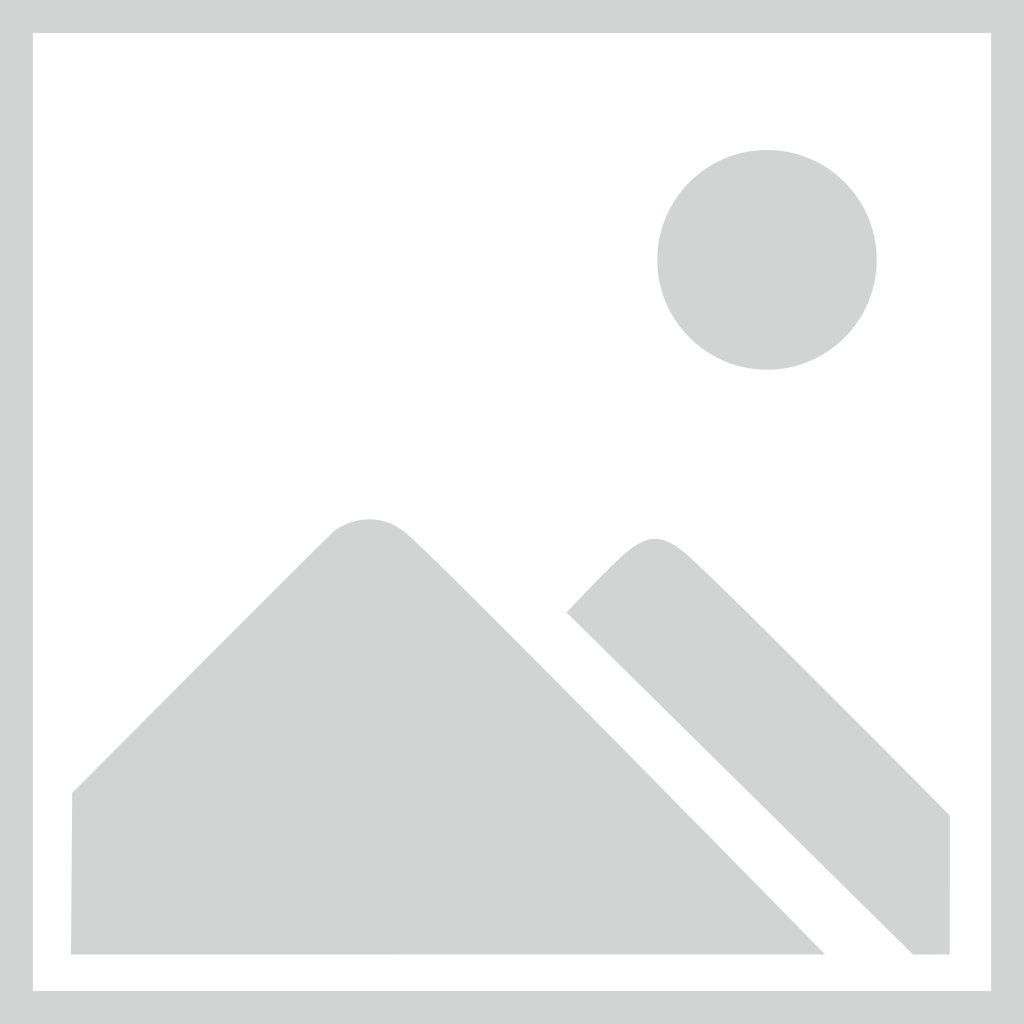
$ sudo apt-get install git

**Windows**就不说了, 没怎么用过Windows做开发, 坑太多了

## Github创建

在[Github](https://github.com/)中创建一个属于自己的帐号

新建帐号后, 请点击New repository或者下图地方



Github仓库创建

并通过[Install-SSH-Use-Github](http://andrewliu.tk/2014/09/09/2014-09-09-Install-SSH-Use-Github/)学习简单的Github与git的协作以及SSH的创建

Github和git的协作我们会在使用的时候重复提示, 但最好先进行SSH的安装和配置

# Django安装

安装最新版的Django版本

#安装最新版本的Django

$ pip install django

#或者指定安装版本

pip install -v django==1.7.1

# Bootstrap安装

Bootstrap 简洁、直观、强悍的前端开发框架，让web开发更迅速、简单

bootstrap已经有较为完善的中文文档, 可以在[bootstrap中文网](http://v3.bootcss.com/getting-started/%23download)查看

推荐下载其中的Bootstrap源码

到目前为止, 基本环境已经搭建好了

# 项目创建

现在正式开始吧, 我们创建一个名为WebStore的Django项目

**创建项目的指令如下:**

$ django-admin.py startproject WebStore

现在来看一下整个项目的文件结构

$ tree WebStore #打印树形文件结构

WebStore

├── manage.py

└── WebStore

├── \_\_init\_\_.py

├── settings.py

├── urls.py

└── wsgi.py

1 directory, 5 files

# 建立Django app

　在Django中的app我认为就是一个功能模块, 与其他的web框架可能有很大的区别, 将不能功能放在不同的app中, 方便代码的复用

建立一个commodity app

$ python manage.py startapp commodity

现在让我们重新看一下整个项目的结构

── commodity

│ ├── \_\_init\_\_.py

│ ├── admin.py

│ ├── migrations

│ │ └── \_\_init\_\_.py

│ ├── models.py

│ ├── tests.py

│ └── views.py

├── db.sqlite3

├── manage.py

├── WebStore

├── \_\_init\_\_.py

├── \_\_pycache\_\_

│ ├── \_\_init\_\_.cpython-34.pyc

│ ├── settings.cpython-34.pyc

│ ├── urls.cpython-34.pyc

│ └── wsgi.cpython-34.pyc

├── settings.py

├── urls.py

└── wsgi.py

并在WebStore/WebStore/setting.py下添加新建app

INSTALLED\_APPS = (

...

'commodity', #这里填写的是app的名称

)

# 运行程序

$ python manage.py runserver #启动Django中的开发服务器

#如果运行上面命令出现以下提示

You have unapplied migrations; your app may not work properly until they are applied.

Run 'python manage.py migrate' to apply them.

#请先使用下面命令

python manage.py migrate

#输出如下信息

Operations to perform:

Apply all migrations: contenttypes, sessions, admin, auth

Running migrations:

Applying contenttypes.0001\_initial... OK

Applying auth.0001\_initial... OK

Applying admin.0001\_initial... OK

Applying sessions.0001\_initial... OK

运行成功后,会显示如下信息

#重新运行启动Django中的开发服务器

$ python manage.py runserver

#运行成功显示如下信息

System check identified no issues (0 silenced).

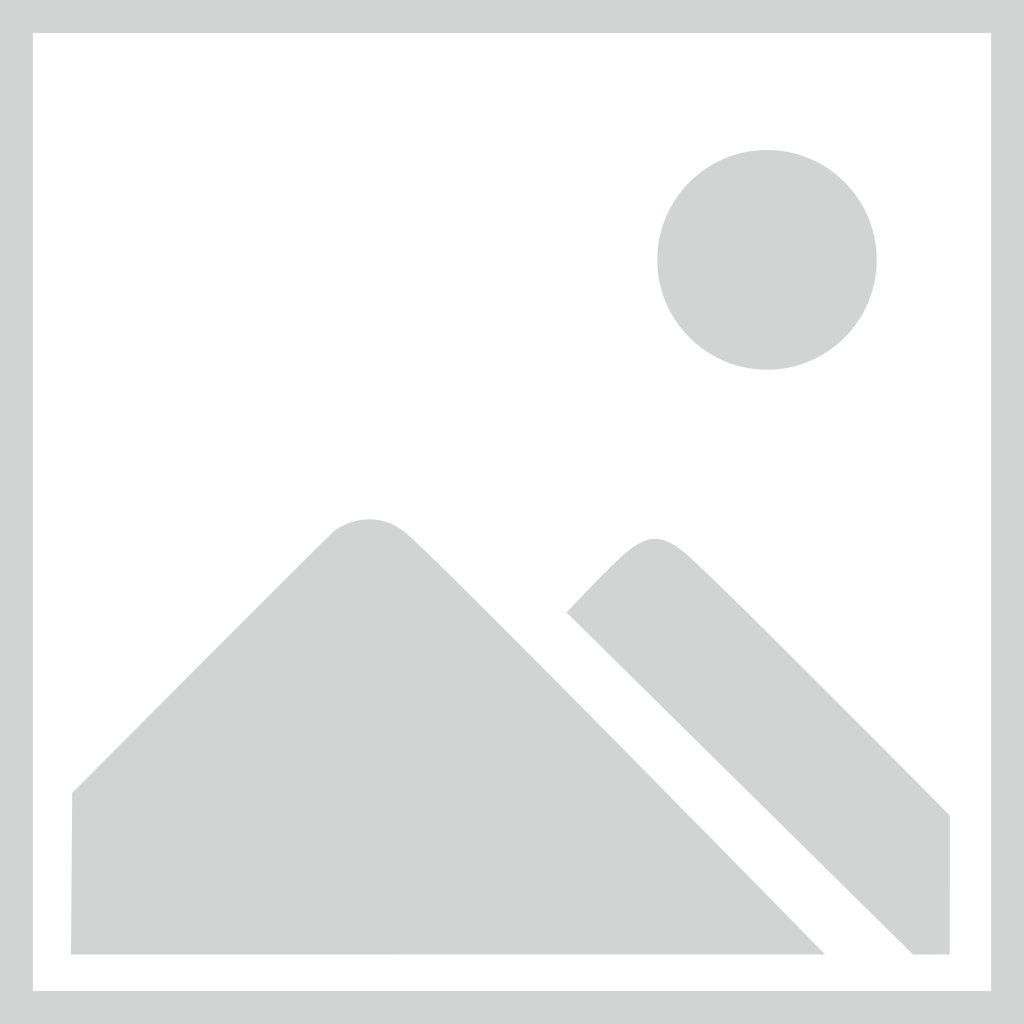
December 21, 2014 - 08:56:00

Django version 1.7.1, using settings 'WebStore.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

现在可以启动浏览器, 输入<http://127.0.0.1:8000/>, 当出现



成功

说明你成功走出了第一步!

命令梳理:

python manage.py <command> [options] #Django Command python manange.py -h帮助文档

django-admin.py startproject WebStore #创建项目

python manage.py startapp commodity #创建app

# Django Model

* 每一个Django Model都继承自django.db.models.Model
* 在Model当中每一个属性attribute都代表一个database field
* 通过Django Model API可以执行数据库的增删改查, 而不需要写一些数据库的查询语句

# 设置数据库

Django项目建成后, 默认设置了使用SQLite数据库, 在WebStore/WebStore/setting.py中可以查看和修改数据库设置:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db.sqlite3'),

}

}

还可以设置其他数据库, 如MySQL, PostgreSQL, 现在为了简单, 使用默认数据库设置

# 创建models

在WebStore/commodity/models.py下编写如下程序:

from django.db import models

# Create your models here.

class Commodity(models.Model) :

title = models.CharField(max\_length = 100) #博客题目

category = models.CharField(max\_length = 50, blank = True) #博客标签

date\_time = models.DateTimeField(auto\_now\_add = True) #博客日期

content = models.TextField(blank = True, null = True) #博客文章正文

def \_\_unicode\_\_(self) :

return self.title

class Meta: #按时间下降排序

ordering = ['-date\_time']

其中\_\_unicode\_\_(self) 函数Commodity对象要怎么表示自己, 一般系统默认使用<Commodity: Commodity object> 来表示对象, 通过这个函数可以告诉系统使用title字段来表示这个对象

* CharField 用于存储字符串, max\_length设置最大长度
* TextField 用于存储大量文本
* DateTimeField 用于存储时间, auto\_now\_add设置True表示自动设置对象增加时间

# 同步数据库

$ python manage.py migrate #命令行运行该命令

因为我们已经执行过该命令会出现如下提示

Operations to perform:

Apply all migrations: admin, contenttypes, sessions, auth

Running migrations:

No migrations to apply.

Your models have changes that are not yet reflected in a migration, and so won't be applied.

Run 'manage.py makemigrations' to make new migrations, and then re-run 'manage.py migrate' to apply them.

那么现在需要执行下面的命令

$ python manage.py makemigrations

#得到如下提示

Migrations for 'commodity':

0001\_initial.py:

- Create model Commodity

现在重新运行以下命令

$ python manage.py migrate

#出现如下提示表示操作成功

Operations to perform:

Apply all migrations: auth, sessions, admin, commodity, contenttypes

Running migrations:

Applying commodity.0001\_initial... OK

migrate命令按照app顺序建立或者更新数据库, 将models.py与数据库同步

# Django Shell

现在我们进入Django中的交互式shell来进行数据库的增删改查等操作

$ python manage.py shell

Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 5 2014, 20:42:22)

[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin

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

(InteractiveConsole)

>>>

这里进入Django的shell和python内置的shell是非常类似的

>>> from commodity.models import Commodity

>>> #create数据库增加操作

>>> Commodity.objects.create(title = 'Hello World', category = 'Python', content = '我们来做一个简单的数据库增加操作')

<Commodity: Commodity object>

>>> Commodity.objects.create(title = 'Django Blog学习', category = 'Python', content = 'Django简单博客教程')

<Commodity: Commodity object>

>>> #all和get的数据库查看操作

>>> Commodity.objects.all() #查看全部对象, 返回一个列表, 无对象返回空list

[<Commodity: Commodity object>, <Commodity: Commodity object>]

>>> Commodity.objects.get(id = 1) #返回符合条件的对象

<Commodity: Commodity object>

>>> #update数据库修改操作

>>> first = Commodity.objects.get(id = 1) #获取id = 1的对象

>>> first.title

'Hello World'

>>> first.date\_time

datetime.datetime(2014, 12, 26, 13, 56, 48, 727425, tzinfo=<UTC>)

>>> first.content

'我们来做一个简单的数据库增加操作'

>>> first.category

'Python'

>>> first.content = 'Hello World, How are you'

>>> first.content #再次查看是否修改成功, 修改操作就是点语法

'Hello World, How are you'

>>> #delete数据库删除操作

>>> first.delete()

>>> Commodity.objects.all() #此时可以看到只有一个对象了, 另一个对象已经被成功删除

[<Commodity: Commodity object>]

当然还有更多的API, 可以查看官方文档

# Admin简介

Django有一个优秀的特性, 内置了Django admin后台管理界面, 方便管理者进行添加和删除网站的内容.

# 设置Admin

新建的项目系统已经为我们设置好了后台管理功能

可以在WebStore/WebStore/setting.py中查看

INSTALLED\_APPS = (

'django.contrib.admin', #默认添加后台管理功能

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'commodity'

)

同时也已经添加了进入后台管理的url, 可以在WebStore/WebStore/urls.py中查看

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

from django.contrib import admin

urlpatterns = patterns('',

# Examples:

# url(r'^$', 'WebStore.views.home', name='home'),

# url(r'^blog/', include('blog.urls')),

url(r'^admin/', include(admin.site.urls)), #可以使用设置好的url进入网站后台

url(r'^$', 'commodity.views.home'),

)

# 创建超级用户

使用如下命令账号创建超级用户(如果使用了python manage.py syncdb会要求你创建一个超级用户)

$ python manage.py createsuperuser

Username (leave blank to use 'andrew\_liu'): root

Email address:

Password:

Password (again):

Superuser created successfully.

输入用户名, 邮箱, 密码就能够创建一个超级用户

现在可以在浏览器中输入[127.0.0.1:8000/admin](http://127.0.0.1:8000/admin)输入账户和密码进入后台管理, 如下:

后台

进入

但是你会发现并没有数据库信息的增加和删除, 现在我们在WebStore/commodity/admin.py中增加代码:

from django.contrib import admin

from commodity.models import Commodity

# Register your models here.

admin.site.register(Commodity)

保存后, 再次刷新页面, [127.0.0.1:8000/admin](http://127.0.0.1:8000/admin)

成功

对于管理界面的外观的定制还有展示顺序的修改就不详细叙述了, 感兴趣的可以查看官方文档...

# 使用第三方插件

Django现在已经相对成熟, 已经有许多不错的可以使用的第三方插件可以使用, 这些插件各种各样, 现在我们使用一个第三方插件使后台管理界面更加美观, 目前大部分第三方插件可以在[Django Packages](https://www.djangopackages.com/) 中查看,

尝试使用[django-admin-bootstrap](https://github.com/douglasmiranda/django-admin-bootstrap)美化后台管理界面

## 安装

$ pip install bootstrap-admin

## 配置

然后在WebStore/WebStore/setting.py中修改INSTALLED\_APPS

INSTALLED\_APPS = (

'bootstrap\_admin', #一定要放在`django.contrib.admin`前面

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'commodity',

)

from django.conf import global\_settings

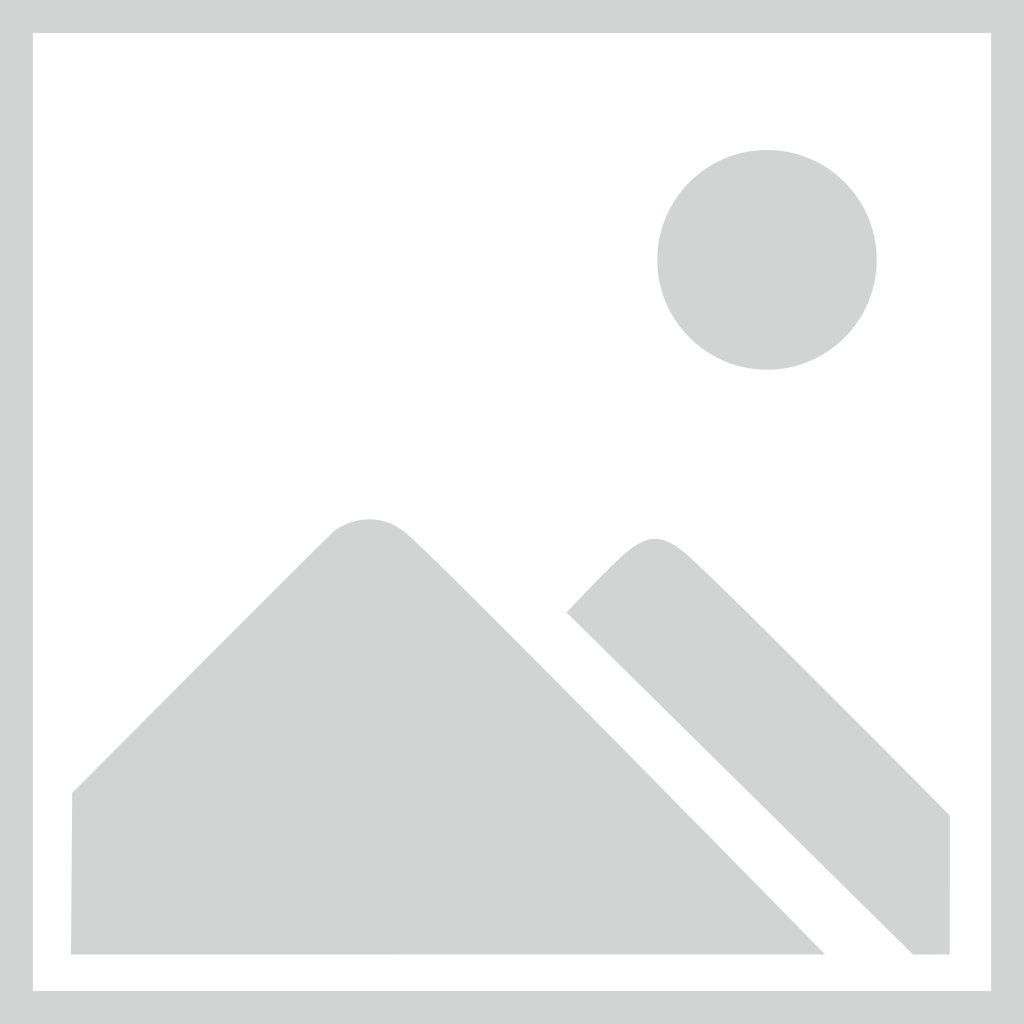
TEMPLATE\_CONTEXT\_PROCESSORS = global\_settings.TEMPLATE\_CONTEXT\_PROCESSORS + (

'django.core.context\_processors.request',

)

BOOTSTRAP\_ADMIN\_SIDEBAR\_MENU = True

保存后, 再次刷新页面, [127.0.0.1:8000/admin](http://127.0.0.1:8000/admin)



第三方

界面是不是美腻了许多...

**网页程序的逻辑**

request进来->从服务器获取数据->处理数据->把网页呈现出来

* url设置相当于客户端向服务器发出request请求的入口, 并用来指明要调用的程序逻辑
* views用来处理程序逻辑, 然后呈现到template(一般为GET方法, POST方法略有不同)
* template一般为html+CSS的形式, 主要是呈现给用户的表现形式

# 简单Django Views和URL

Django中views里面的代码就是一个一个函数逻辑, 处理客户端(浏览器)发送的HTTPRequest, 然后返回HTTPResponse,

那么那么开始在WebStore/commodity/views.py中编写简单的逻辑

#现在你的views.py应该是这样

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def home(request):

return HttpResponse("Hello World, Django")

那么如何使这个逻辑在http请求进入时, 被调用呢, 这里需要在WebStore/WebStore/urls.py中进行url设置

from django.conf.urls import include, url #alex:已经没有了patterns

from django.contrib import admin

from alticle import views #alex：1.8以上新的写法

urlpatterns = patterns('',

# Examples:

# url(r'^$', 'WebStore.views.home', name='home'),

# url(r'^blog/', include('blog.urls')),

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

url(r'^$', 'views.home'), #由于目前只有一个app, 方便起见, 就不设置include了

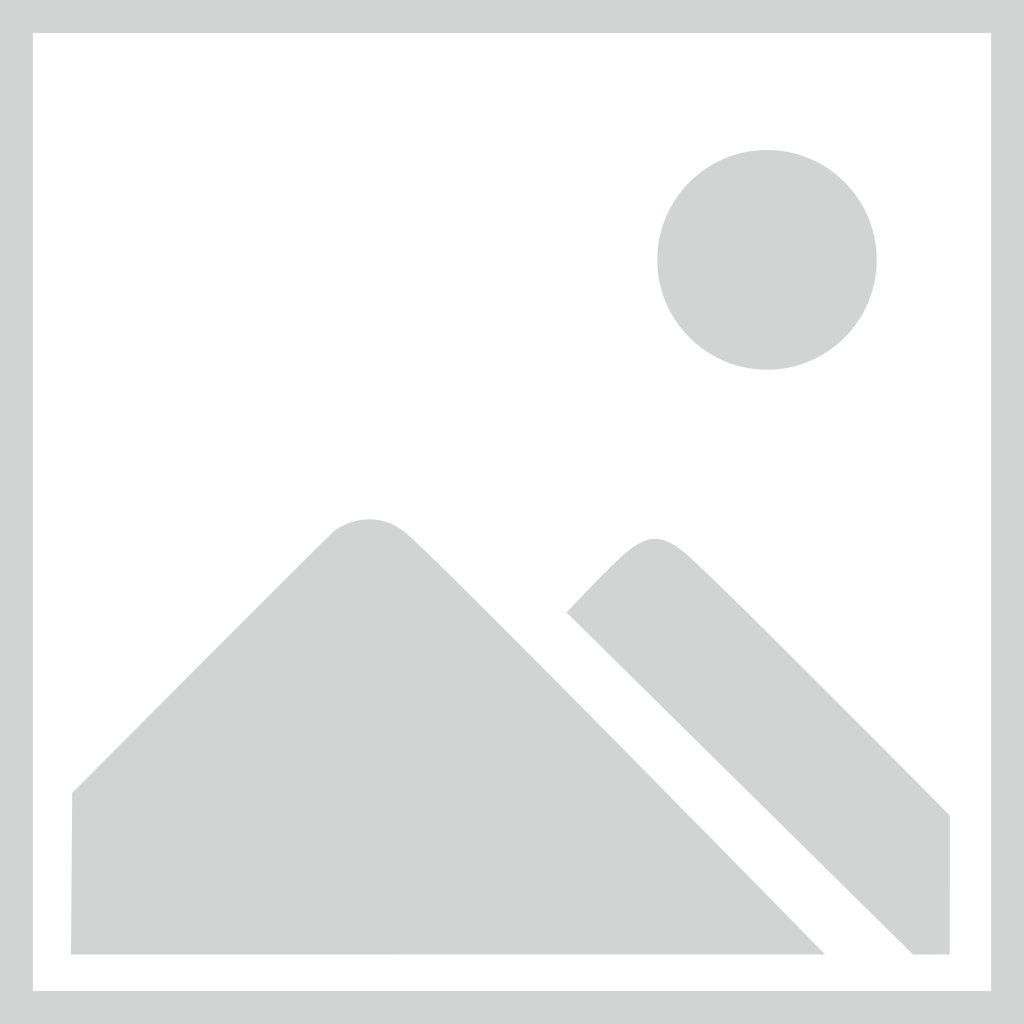
url()函数有四个参数, 两个是必须的:regex和view, 两个可选的:kwargs和name

* regex是regular expression的简写,这是字符串中的模式匹配的一种语法, Django 将请求的URL从上至下依次匹配列表中的正则表达式，直到匹配到一个为止。

更多正则表达式的使用可以查看[Python正则表达式](http://andrewliu.tk/2014/10/26/2014-10-26-Python%E6%AD%A3%E5%88%99%E8%A1%A8%E8%BE%BE%E5%BC%8F/)

* view当 Django匹配了一个正则表达式就会调用指定的view逻辑, 上面代码中会调用commodity/views.py中的home函数
* kwargs任意关键字参数可传一个字典至目标view
* name命名你的 URL, 使url在 Django 的其他地方使用, 特别是在模板中

现在在浏览器中输入[127.0.0.1:8000](http://127.0.0.1:8000/)应该可以看到下面的界面



成功

# Django Views和URL更近一步

很多时候我们希望给view中的函数逻辑传入参数, 从而呈现我们想要的结果

现在我们这样做, 在WebStore/commodity/views.py加入如下代码:

def detail(request, my\_args):

return HttpResponse("You're looking at my\_args %s." % my\_args)

在WebStore/WebStore/urls.py中设置对应的url,

urlpatterns = patterns('',

# Examples:

# url(r'^$', 'WebStore.views.home', name='home'),

# url(r'^blog/', include('blog.urls')),

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

url(r'^$', 'commodity.views.home'),

url(r'^(?P<my\_args>\d+)/$', 'commodity.views.detail', name='detail'),

)

^(?P<my\_args>\d+)/$这个正则表达式的意思是将传入的一位或者多位数字作为参数传递到views中的detail作为参数, 其中?P<my\_args>定义名称用于标识匹配的内容

一下url都能成功匹配这个正则表达数

* <http://127.0.0.1:8000/1000/>
* <http://127.0.0.1:8000/9/>

**尝试传参访问数据库**

修改在WebStore/commodity/views.py代码:

from django.shortcuts import render

from django.http import HttpResponse

from commodity.models import Commodity

# Create your views here.

def home(request):

return HttpResponse("Hello World, Django")

def detail(request, my\_args):

post = Commodity.objects.all()[int(my\_args)]

str = ("title = %s, category = %s, date\_time = %s, content = %s"

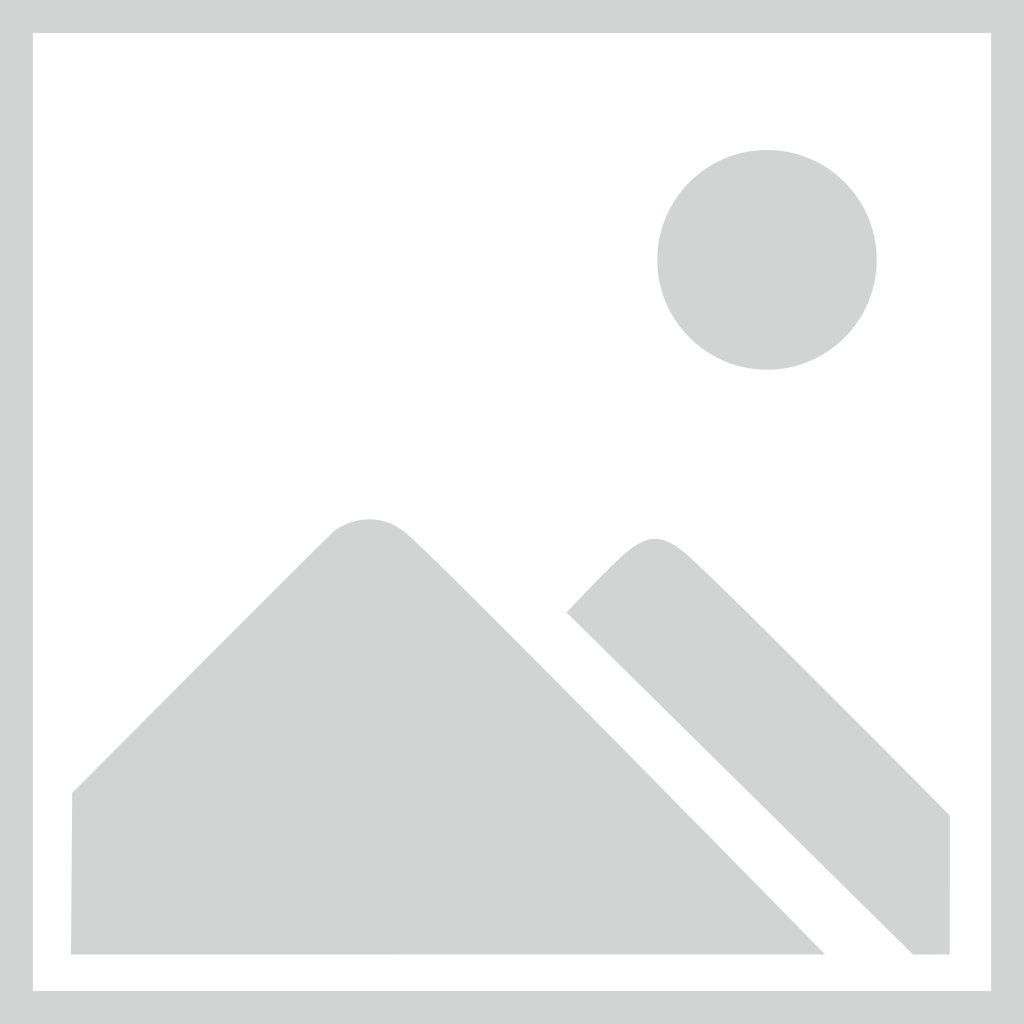
% (post.title, post.category, post.date\_time, post.content))

return HttpResponse(str)

这里最好在admin后台管理界面增加几个Commodity对象, 防止查询对象为空, 出现异常，而且输入的数字必须比数据库中已经有的博客数小。

现在可以访问<http://127.0.0.1:8000/1/>

显示如下数据表示数据库访问正确(这些数据都是自己添加的), 并且注意Commodity.objects.all()返回的是一个列表



数据

小结:

* 如何编写views和设置url
* 如何通过url向views传参
* 如何通过参数来访问数据库资源

**Template初探**

到目前为止我们只是简单的将后端数据显示到页面上, 没有涉及到HTML代码, 而优雅的网站总算通过CSS+HTML, 甚至还有强大的JS的支持.

在这个教程中要打造一个Blog, 所以我们设置一个Blog界面, 原本打算使用Bootstrap作为前段的工具, 不过经过@游逸的建议, 使用了更加轻量级的[Pure](http://purecss.io/), 同样是响应式页面设置, 这也将是未来的主流吧..

在WebStore下添加文件名, 文件夹名为templates

mkdir templates

#看到当前文件构成

WebStore

├── commodity

│ ├── \_\_init\_\_.py

│ ├── \_\_pycache\_\_

│ │ ├── \_\_init\_\_.cpython-34.pyc

│ │ ├── admin.cpython-34.pyc

│ │ ├── models.cpython-34.pyc

│ │ └── views.cpython-34.pyc

│ ├── admin.py

│ ├── migrations

│ │ ├── 0001\_initial.py

│ │ ├── \_\_init\_\_.py

│ │ └── \_\_pycache\_\_

│ │ ├── 0001\_initial.cpython-34.pyc

│ │ └── \_\_init\_\_.cpython-34.pyc

│ ├── models.py

│ ├── tests.py

│ └── views.py

├── db.sqlite3

├── manage.py

├── WebStore

│ ├── \_\_init\_\_.py

│ ├── \_\_pycache\_\_

│ │ ├── \_\_init\_\_.cpython-34.pyc

│ │ ├── settings.cpython-34.pyc

│ │ ├── urls.cpython-34.pyc

│ │ └── wsgi.cpython-34.pyc

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── templates#alex，放在app下即可

~~在WebStore/WebStore/setting.py下设置templates的位置alex：1、11下不需要设置，永默认的即可~~

~~TEMPLATE\_DIRS = (~~

~~os.path.join(BASE\_DIR, 'templates').replace('\\', '/'),~~

~~)~~

~~意思是告知项目templates文件夹在项目根目录下~~

# 第一个template

templates/test.html简单第一个 template html文件

<!--在test.html文件夹下添加-->

<!DOCTYPE html>

<html>

<head>

<title>Just test template</title>

<style>

body {

background-color: red;

}

em {

color: LightSeaGreen;

}

</style>

</head>

<body>

<h1>Hello World!</h1>

<strong>

{{ current\_time }}

</strong>

</body>

</html>

其中{{ current\_time }}是Django Template中变量的表示方式

在commodity/view.py中添加一个函数逻辑

from django.shortcuts import render

from django.http import HttpResponse

from commodity.models import Commodity

from datetime import datetime

# Create your views here.

def home(request):

return HttpResponse("Hello World, Django")

def detail(request, my\_args):

post = Commodity.objects.all()[int(my\_args)]

str = ("title = %s, category = %s, date\_time = %s, content = %s"

% (post.title, post.category, post.date\_time, post.content))

return HttpResponse(str)

def test(request) :

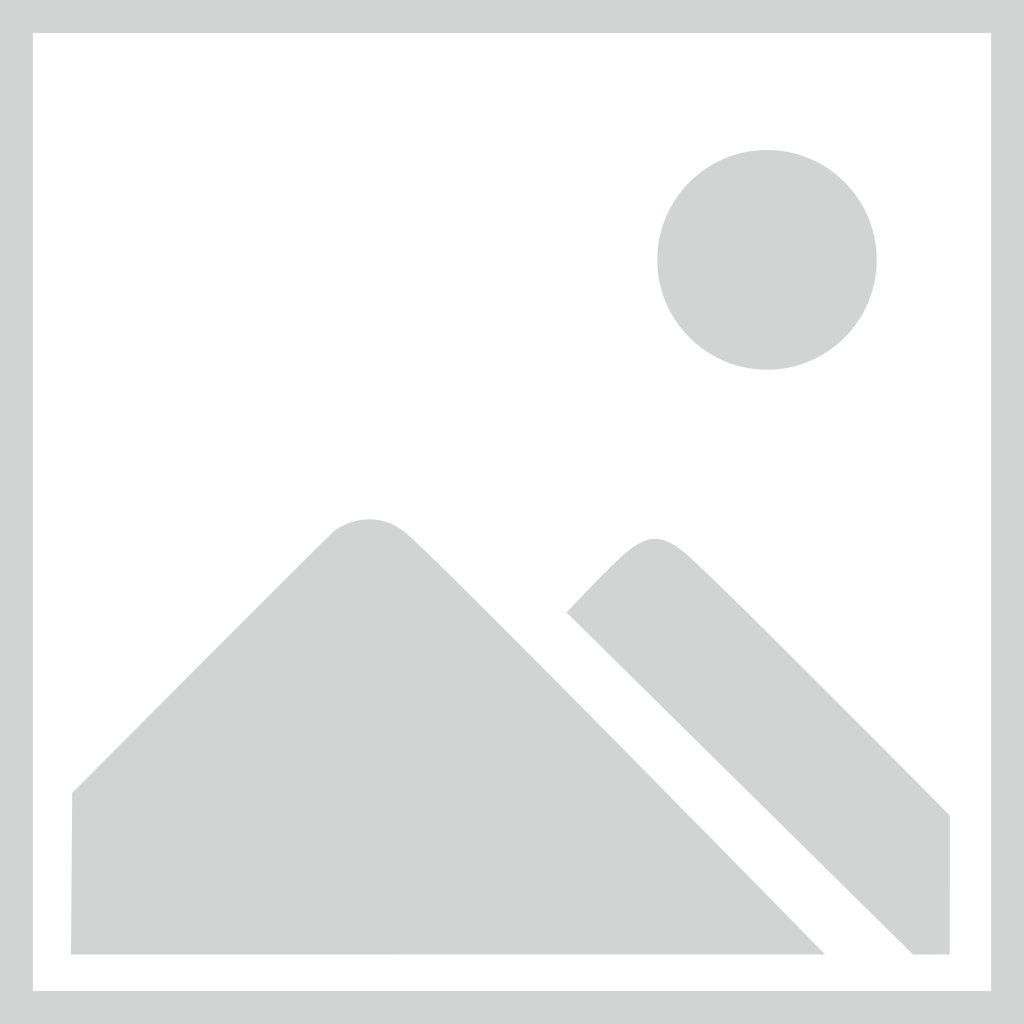
return render(request, 'test.html', {'current\_time': datetime.now()})

render()函数中第一个参数是request 对象, 第二个参数是一个模板名称，第三个是一个字典类型的可选参数. 它将返回一个包含有给定模板根据给定的上下文渲染结果的 HttpResponse对象。

然后设置对应的url在WebStore/urls.py下

url(r'^test/$', 'commodity.views.test'),

重新启动服务器python manage.py runserver, 然后在浏览器中输入<http://127.0.0.1:8000/test/>, 可以看到



test

# 正式编写template

在template文件夹下增加base.html, 并在其中增加如下代码

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta name="description" content="A layout example that shows off a blog page with a list of posts.">

<title>Andrew Liu Blog</title>

<link rel="stylesheet" href="<http://yui.yahooapis.com/pure/0.5.0/pure-min.css>">

<link rel="stylesheet" href="<http://yui.yahooapis.com/pure/0.5.0/grids-responsive-min.css>">

<link rel="stylesheet" href="<http://picturebag.qiniudn.com/blog.css>">

</head>

<body>

<div id="layout" class="pure-g">

<div class="sidebar pure-u-1 pure-u-md-1-4">

<div class="header">

<h1 class="brand-title">Andrew Liu Blog</h1>

<h2 class="brand-tagline">雪忆 - Snow Memory</h2>

<nav class="nav">

<ul class="nav-list">

<li class="nav-item">

<a class="pure-button" href="<https://github.com/Andrew-liu>">Github</a>

</li>

<li class="nav-item">

<a class="pure-button" href="<http://weibo.com/dinosaurliu>">Weibo</a>

</li>

</ul>

</nav>

</div>

</div>

<div class="content pure-u-1 pure-u-md-3-4">

<div>

{% block content %}

{% endblock %}

<div class="footer">

<div class="pure-menu pure-menu-horizontal pure-menu-open">

<ul>

<li><a href="<http://andrewliu.tk/about>/">About Me</a></li>

<li><a href="<http://twitter.com/yuilibrary>/">Twitter</a></li>

<li><a href="<http://github.com/yahoo/pure>/">GitHub</a></li>

</ul>

</div>

</div>

</div>

</div>

</div>

</body>

</html>

上面这段html编写的页面是一个模板, 其中{% block content %} {% endblock %}字段用来被其他继承这个基类模板进行重写

我们继续在templates文件夹下添加home.html文件

{% extends "base.html" %}

{% block content %}

<div class="posts">

{% for post in post\_list %}

<section class="post">

<header class="post-header">

<h2 class="post-title">

{{ post.title }}

</h2>

<p class="post-meta">

Time: <a class="post-author" href="#">

{{ post.date\_time }}</a> <a class="post-category post-category-js" href="#">{{ post.category }}

</a>

</p>

</header>

<div class="post-description">

<p>

{{ post.content }}

</p>

</div>

</section>

{% endfor %}

</div><!-- /.blog-post -->

{% endblock %}

其中

* {% for <element> in <list> %}与{% endfor %}成对存在, 这是template中提供的for循环tag
* {% if <elemtnt> %} {% else %} {% endif %}是template中提供的if语句tag
* template中还提供了一些过滤器

然后修改WebStore/commodity/view.py, 并删除test.html

# -\*- coding: utf-8 -\*-

from django.shortcuts import render

from django.http import HttpResponse

from commodity.models import Commodity

from datetime import datetime

# Create your views here.

def home(request):

post\_list = Commodity.objects.all() #获取全部的Commodity对象

return render(request, 'home.html', {'post\_list' : post\_list})

修改WebStore/WebStore/urls.py

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

from django.contrib import admin

urlpatterns = patterns('',

# Examples:

# url(r'^$', 'WebStore.views.home', name='home'),

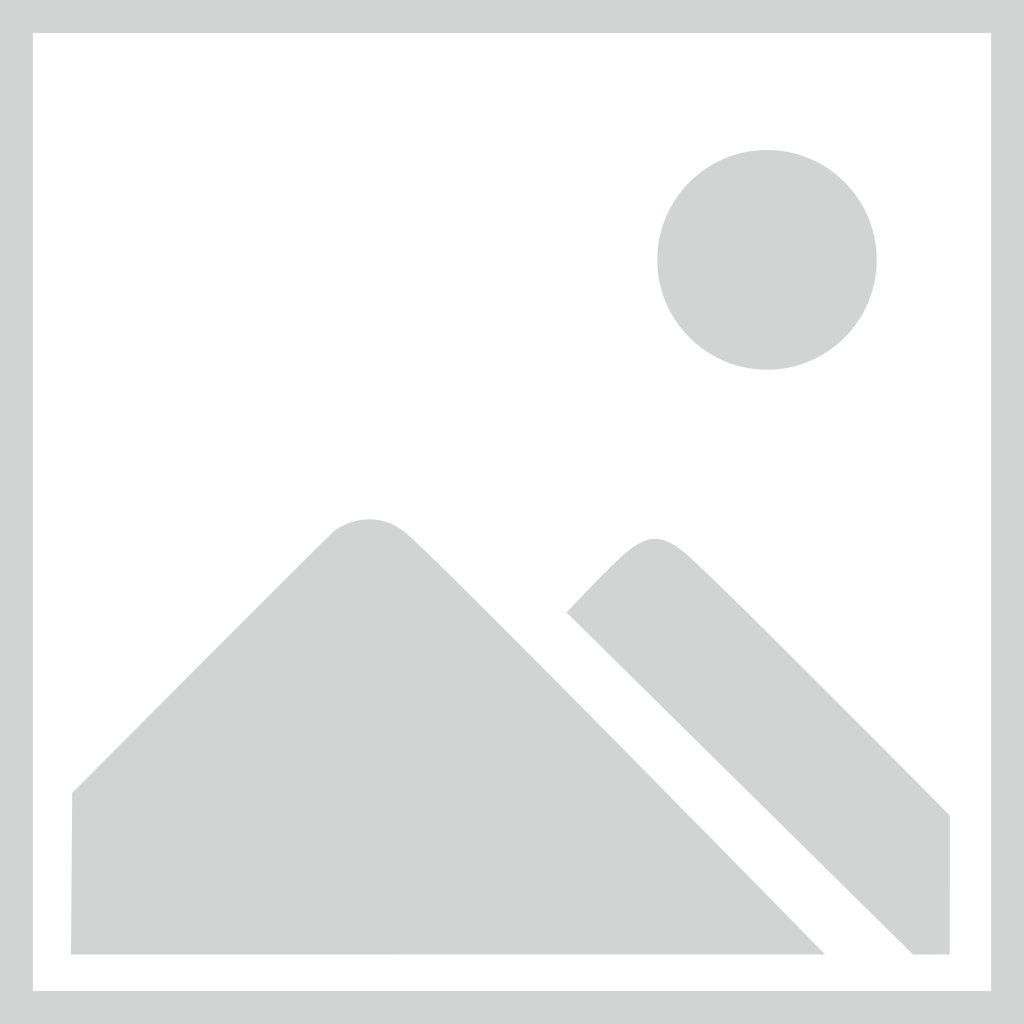
# url(r'^blog/', include('blog.urls')),

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

url(r'^$', 'commodity.views.home'),

）

现在重新打开<http://127.0.0.1:8000/>, 发现Blog的整理框架已经基本完成, 到现在我们已经了解了一些Django的基本知识, 搭建了简单地Blog框架, 剩下的就是给Blog添加功能



基本框架

查看当前整个程序的目录结构

WebStore

├── commodity

│ ├── \_\_init\_\_.py

│ ├── \_\_pycache\_\_

│ │ ├── \_\_init\_\_.cpython-34.pyc

│ │ ├── admin.cpython-34.pyc

│ │ ├── models.cpython-34.pyc

│ │ └── views.cpython-34.pyc

│ ├── admin.py

│ ├── migrations

│ │ ├── 0001\_initial.py

│ │ ├── \_\_init\_\_.py

│ │ └── \_\_pycache\_\_

│ │ ├── 0001\_initial.cpython-34.pyc

│ │ └── \_\_init\_\_.cpython-34.pyc

│ ├── models.py

│ ├── tests.py

│ └── views.py

├── db.sqlite3

├── manage.py

├── WebStore

│ ├── \_\_init\_\_.py

│ ├── \_\_pycache\_\_

│ │ ├── \_\_init\_\_.cpython-34.pyc

│ │ ├── settings.cpython-34.pyc

│ │ ├── urls.cpython-34.pyc

│ │ └── wsgi.cpython-34.pyc

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── templates

├── base.html

└── home.html

# 将代码上传到Github

在github中新建仓库WebStore\_tutorial, 填写简单的描述

#查看当前目录位置

$ pwd

/Users/andrew\_liu/Python/Django/WebStore

#在项目的根目录下初始化git

git init

Initialized empty Git repository in/Users/andrew\_liu/Python/Django/WebStore/.git/

#添加远程github

$ git remote add blog [git@github.com](mailto:git@github.com):Andrew-liu/WebStore\_tutorial.git

在根目录下增加`.gitignore'和'LICENSE'和'README.md'文件

#添加所有文件

$ git add .

#查看当前状态

$ git status

#commit操作

$ git commit -m "django tutorial init"

#上传github

$ git push -u blog master

Counting objects: 23, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (22/22), done.

Writing objects: 100% (23/23), 19.56 KiB | 0 bytes/s, done.

Total 23 (delta 1), reused 0 (delta 0)

To [git@github.com](mailto:git@github.com):Andrew-liu/WebStore\_tutorial.git

\* [new branch] master -> master

Branch master set up to track remote branch master from blog.

**动态URL**

运行已经做好的博客框架, 会发现一个问题, 只有一个主页的空盒子, 而大部分时候我们希望能够让每篇博客文章都有一个独立的页面.

我第一个想到的方法是给每篇博客文章加一个view函数逻辑, 然后设置一个独立的url(我不知道语言比如PHP, 或者web框架rail等是如果解决的, 我是第一次仔细的学习web框架, 也没有前端开发经验), 但是这种方法耦合性太强, 而且用户不友好, 缺点非常多

Django给我们提供了一个方便的解决方法, 就是动态URL

现在修改WebStore/commodity/views.py代码:

# -\*- coding: utf-8 -\*-

from django.shortcuts import render

from django.http import HttpResponse

from commodity.models import Commodity

from datetime import datetime

from django.http import Http404

# Create your views here.

def home(request):

post\_list = Commodity.objects.all() #获取全部的Commodity对象

return render(request, 'home.html', {'post\_list' : post\_list})

def detail(request, id):

try:

post = Commodity.objects.get(id=str(id))

except Commodity.DoesNotExist:

raise Http404

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

因为id是每个博文的唯一标识, 所以这里使用id对数据库中的博文进行查找

在WebStore/WebStore/urls.py中修改url设置:

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

from django.contrib import admin

urlpatterns = patterns('',

# Examples:

# url(r'^$', 'WebStore.views.home', name='home'),

# url(r'^blog/', include('blog.urls')),

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

url(r'^$', 'commodity.views.home', name = 'home'),

url(r'^(?P<id>\d+)/$', 'commodity.views.detail', name='detail'),

)

然后在templates下建立一个用于显示单页博文的界面:

#post.html{% extends "base.html" %}

{% block content %}

<div class="posts">

<section class="post">

<header class="post-header">

<h2 class="post-title">

{{ post.title }}

</h2>

<p class="post-meta">

Time: <a class="post-author" href="#">

{{ post.date\_time|date:'Y /m /d'}}</a> <a class="post-category post-category-js" href="#">{{ post.category }}

</a>

</p>

</header>

<div class="post-description">

<p>

{{ post.content }}

</p>

</div>

</section>

</div><!-- /.blog-post -->

{% endblock %}

可以发现只需要对home.html进行简单的修改, 去掉循环就可以了.

修改home.html和base.html, 加入动态链接和主页, 归档, 专题和About Me按钮

<!--home.html-->{% extends "base.html" %}

{% block content %}

<div class="posts">

{% for post in post\_list %}

<section class="post">

<header class="post-header">

<h2 class="post-title"><a href="

{% url 'detail' id=post.id %}">{{ post.title }}

</a></h2>

<p class="post-meta">

Time: <a class="post-author" href="#">

{{ post.date\_time |date:'Y /m /d'}}</a> <a class="post-category post-category-js" href="#">{{ post.category }}

</a>

</p>

</header>

<div class="post-description">

<p>

{{ post.content }}

</p>

</div>

<a class="pure-button" href="

{% url 'detail' id=post.id %}

">Read More >>> </a>

</section>

{% endfor %}

</div><!-- /.blog-post -->

{% endblock %}

<!--base.html-->

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta name="description" content="A layout example that shows off a blog page with a list of posts.">

<title>Andrew Liu Blog</title>

<link rel="stylesheet" href="<http://yui.yahooapis.com/pure/0.5.0/pure-min.css>">

<link rel="stylesheet" href="<http://yui.yahooapis.com/pure/0.5.0/grids-responsive-min.css>">

<link rel="stylesheet" href="<http://picturebag.qiniudn.com/blog.css>">

</head>

<body>

<div id="layout" class="pure-g">

<div class="sidebar pure-u-1 pure-u-md-1-4">

<div class="header">

<h1 class="brand-title"><a href="

{% url 'home' %}

">Andrew Liu Blog</a></h1>

<h2 class="brand-tagline">雪忆 - Snow Memory</h2>

<nav class="nav">

<ul class="nav-list">

<li class="nav-item">

<a class="button-success pure-button" href="/">主页</a>

</li>

<li class="nav-item">

<a class="button-success pure-button" href="/">归档</a>

</li>

<li class="nav-item">

<a class="pure-button" href="[https://github.com/Andrew-liu/WebStore\_tutorial](https://github.com/Andrew-liu/my_blog_tutorial)">Github</a>

</li>

<li class="nav-item">

<a class="button-error pure-button" href="<http://weibo.com/dinosaurliu>">Weibo</a>

</li>

<li class="nav-item">

<a class="button-success pure-button" href="/">专题</a>

</li>

<li class="nav-item">

<a class="button-success pure-button" href="/">About Me</a>

</li>

</ul>

</nav>

</div>

</div>

<div class="content pure-u-1 pure-u-md-3-4">

<div>

{% block content %}

{% endblock %}

<div class="footer">

<div class="pure-menu pure-menu-horizontal pure-menu-open">

<ul>

<li><a href="<http://andrewliu.tk/about>/">About Me</a></li>

<li><a href="<http://twitter.com/yuilibrary>/">Twitter</a></li>

<li><a href="<http://github.com/yahoo/pure>/">GitHub</a></li>

</ul>

</div>

</div>

</div>

</div>

</div>

</body>

</html>

其中主要改动

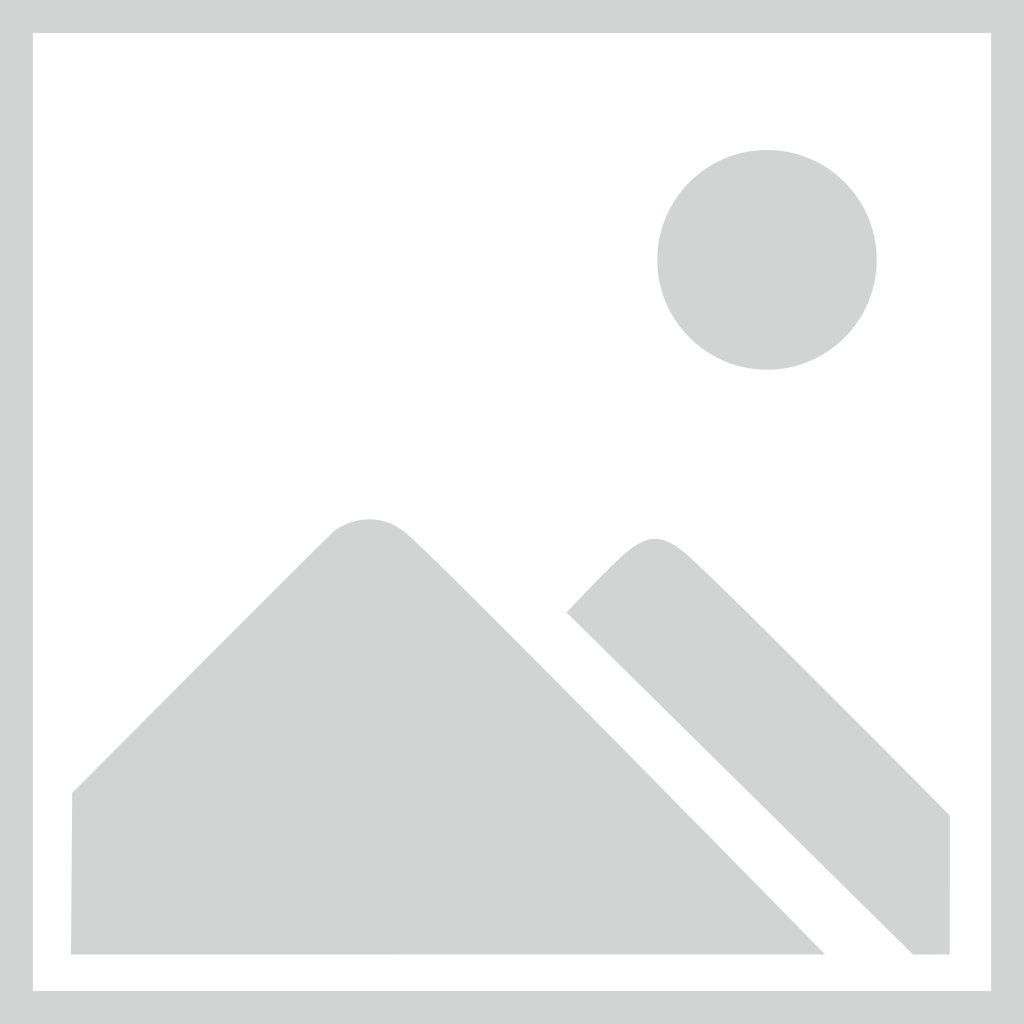
* 添加了几个导航按钮, 方便以后添加功能(暂时不添加登陆功能)
* 添加read more按钮
* 在博客文章的增加一个链接, 链接的href属性为{% url 'detail' id=post.id %}, 当点击这个文章题目时, 会将对应的数据库对象的id传入的url中, 类似于url传参, 不记得的同学可以重新回到前几页翻一下. 这里将数据库对象唯一的id传送给url设置, url取出这个id给对应的view中的函数逻辑当做参数. 这样这个id就传入对应的参数中被使用

比如: 点击到的博客文章标题的对象对应的id=2, 这个id被传送到name=detail的url中, '^(?P<id>\d+)/

比如: 点击到的博客文章标题的对象对应的id=2, 这个id被传送到name=detail的url中, '^(?P<id>\d+)/'^(?P<id>\d+)/$'#39;正则表达式匹配后取出id, 然后将id传送到commodity.views.detail作为函数参数, 然后通过get方法获取对应的数据库对象, 然后对对应的模板进行渲染, 发送到浏览器中..

#39;正则表达式匹配后取出id, 然后将id传送到commodity.views.detail作为函数参数, 然后通过get方法获取对应的数据库对象, 然后对对应的模板进行渲染, 发送到浏览器中..

此时重新运行服务器, 然后在浏览器中输入<http://127.0.0.1:8000/>点击对应的博客文章题目, 可以成功的跳转到一个独立的页面中



博客

**更上一层楼**

由于hexo中和简书中的代码解析问题, 我将所有的文章整理, 放在[gitbook](http://andrew-liu.gitbooks.io/django-blog/content/)任意阅读

* [阅读手册](http://andrew-liu.gitbooks.io/django-blog/content/)
* [Github源代码地址](https://github.com/Andrew-liu/my_blog_tutorial)
* [联系作者](http://andrewliu.tk/)

博客基本建好了, 未来我们还有更多的工作要做:

* 将view封装到类中
* 重写增删改查代码
* 重写前段模板
* 重新设计数据库关系
* 自定义留言板(连接数据库)
* 添加注册/登陆功能(Form)
* 做成社区(更多设计思考)
* ...

更多学习资源:

* [Django](https://www.djangoproject.com/) Django官网, 最全面的Django知识还是要看官方文档, 我认为写的非常棒
* [Django Book](http://www.djangobook.com/en/2.0/index.html) 这个是很早的一本django教程, 虽然版本比较早, 但是其中还有一些知识值得借鉴学习
* [Django 基础教程](http://www.ziqiangxuetang.com/django/django-tutorial.html) 国人写的django使用教程
* [Getting Started with - Django](http://gettingstartedwithdjango.com/) Django学习视频, 感觉看文章比较难的可以看这个视频
* [Two Scoops of Django -](http://www.amazon.com/Two-Scoops-Django-Best-Practices/dp/098146730X) Django教程的书籍, 最新版是1.6版本
* [Getting Started with Django on - Heroku](https://devcenter.heroku.com/articles/getting-st-%20arted-with-django) Django部署在Herohu的官方文档
* [Django Packages](https://www.djangopackages.com/) 所有Django的第三方库的聚集地
* -[Awesome Django个人应用集合](https://github.com/rosarior/awesome-django) Django第三方库的分类整理,强烈推荐看!!!!