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

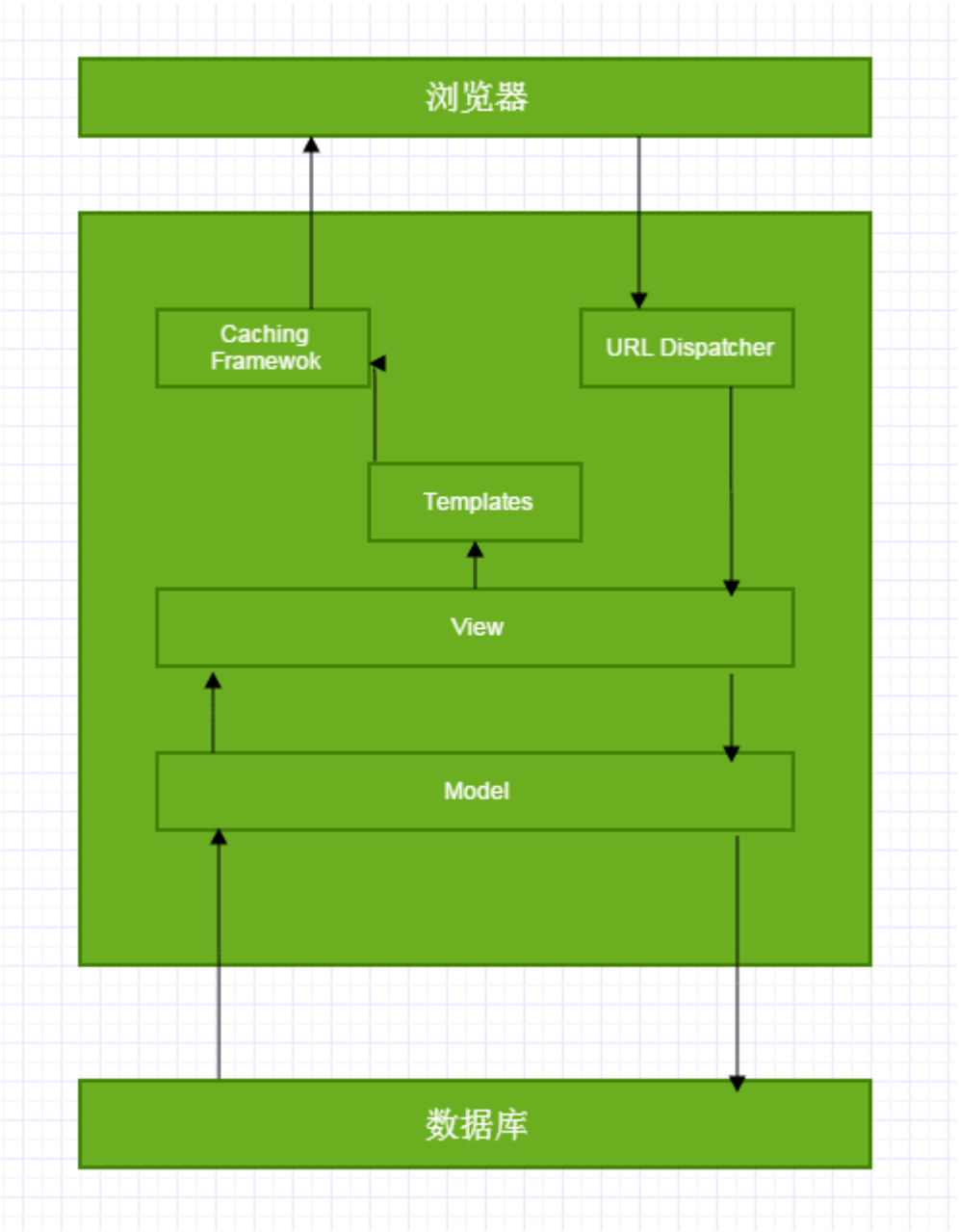
Django 是 python 中目前风靡的 Web Framework，那么什么叫做 Framework 呢，框架能够帮助你把程序的整体架构搭建好，而我们所需要做的工作就是填写逻辑，而框架能够在合适的时候调用你写的逻辑，而不需要我们自己去调用逻辑，让 Web 开发变的更敏捷.

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
```

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

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 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 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 `my_blog` 的 Django 项目

创建项目的指令如下：

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

现在来看一下整个项目的文件结构

```
$ tree my_blog #打印树形文件结构
```

```
my_blog
├── manage.py
└── my_blog
├── __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 `article` app

```
$ python manage.py startapp article
```

现在让我们重新看一下整个项目的结构

```
── article
│ ├── __init__.py
│ ├── admin.py
│ ├── migrations
│ │ └── __init__.py
│ ├── models.py
│ ├── tests.py
│ └── views.py
├── db.sqlite3
├── manage.py
```

```
├── my_blog
├── __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 my_blog/my_blog/setting.py 下添加新建 app

```
INSTALLED_APPS = (
...
'article', #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
```

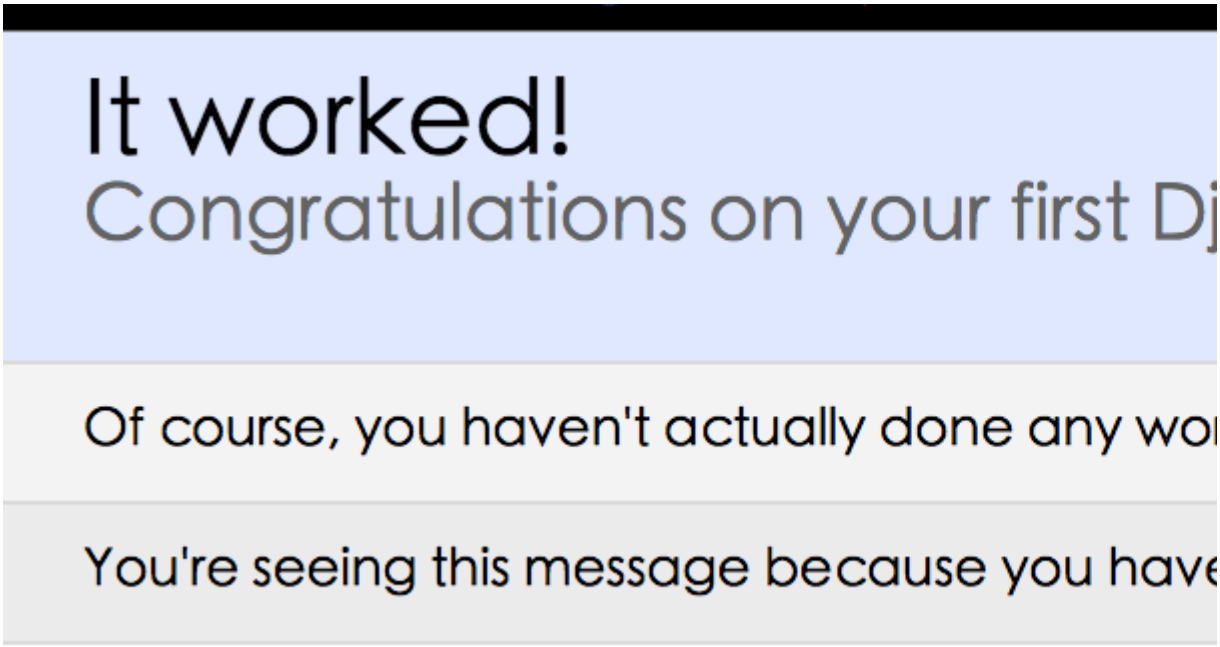
运行成功后,会显示如下信息

```
#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 'my_blog.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 my_blog #创建项目
python manage.py startapp article #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 my_blog/my_blog/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 my_blog/article/models.py 下编写如下程序:

```
from django.db import models

# Create your models here.
class Article(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 Article 对象要怎么表示自己, 一般系统默认使用

`<Article: Article 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 'article':
0001_initial.py:
- Create model Article
```

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, article, contenttypes
Running migrations:
Applying article.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 article.models import Article
>>> #create 数据库增加操作
>>> Article.objects.create(title = 'Hello World', category = 'Python', content
= '我们来做一个简单的数据库增加操作')
<Article: Article object>
>>> Article.objects.create(title = 'Django Blog 学习', category = 'Python',
content = 'Django 简单博客教程')
<Article: Article object>

>>> #all 和 get 的数据库查看操作
>>> Article.objects.all() #查看全部对象, 返回一个列表, 无对象返回空 list
[<Article: Article object>, <Article: Article object>]
>>> Article.objects.get(id = 1) #返回符合条件的对象
<Article: Article object>

>>> #update 数据库修改操作
>>> first = Article.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()
>>> Article.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
[<Article: Article 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 my_blog/my_blog/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',
```

```
'article'
)
```

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

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

urlpatterns = patterns('',
# Examples:
# url(r'^$', 'my_blog.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'^$', 'article.views.home'),
)
```

创建超级用户

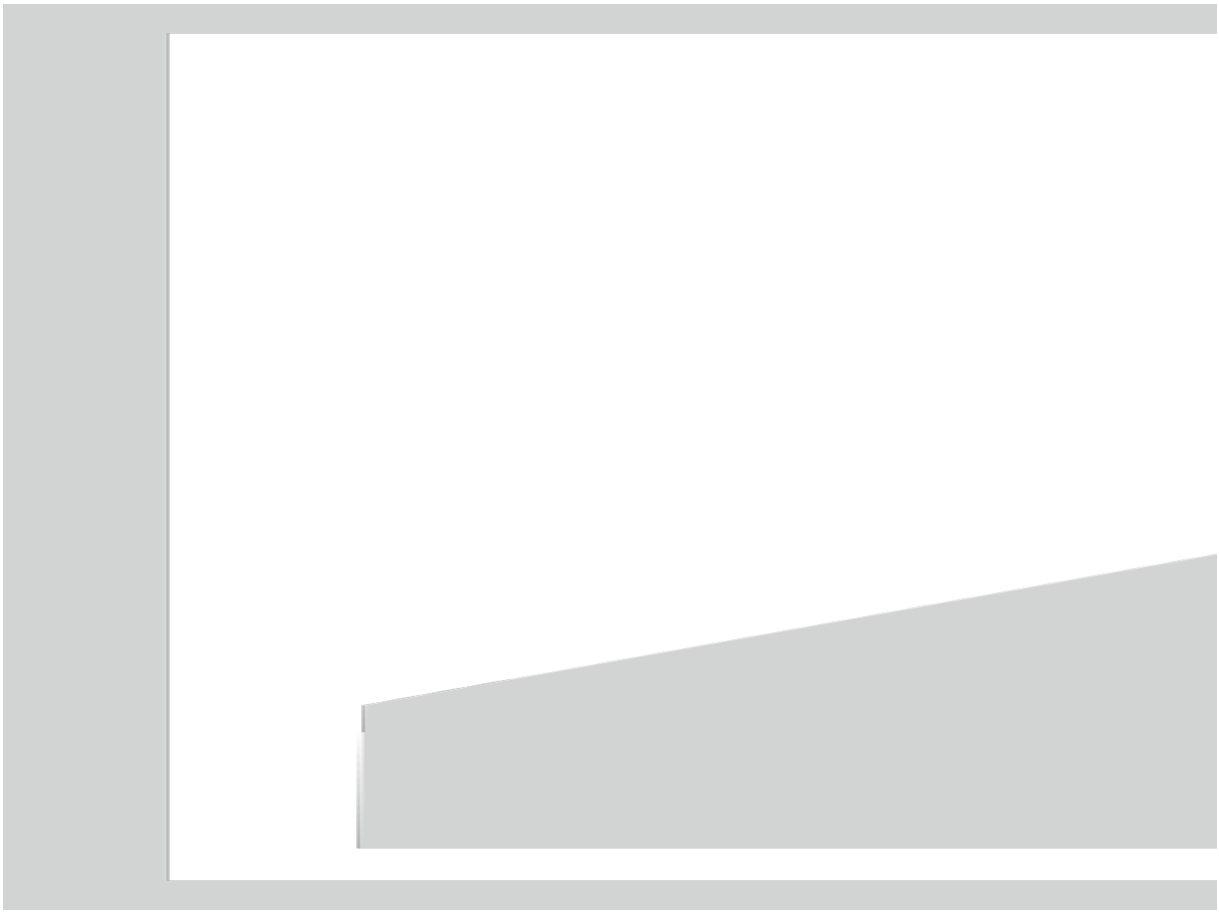
使用如下命令账号创建超级用户(如果使用了 `python manage.py syncdb` 会要求你创建一个超级用户)

```
$ 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 输入账户和密码进入后台管理, 如下:

后台



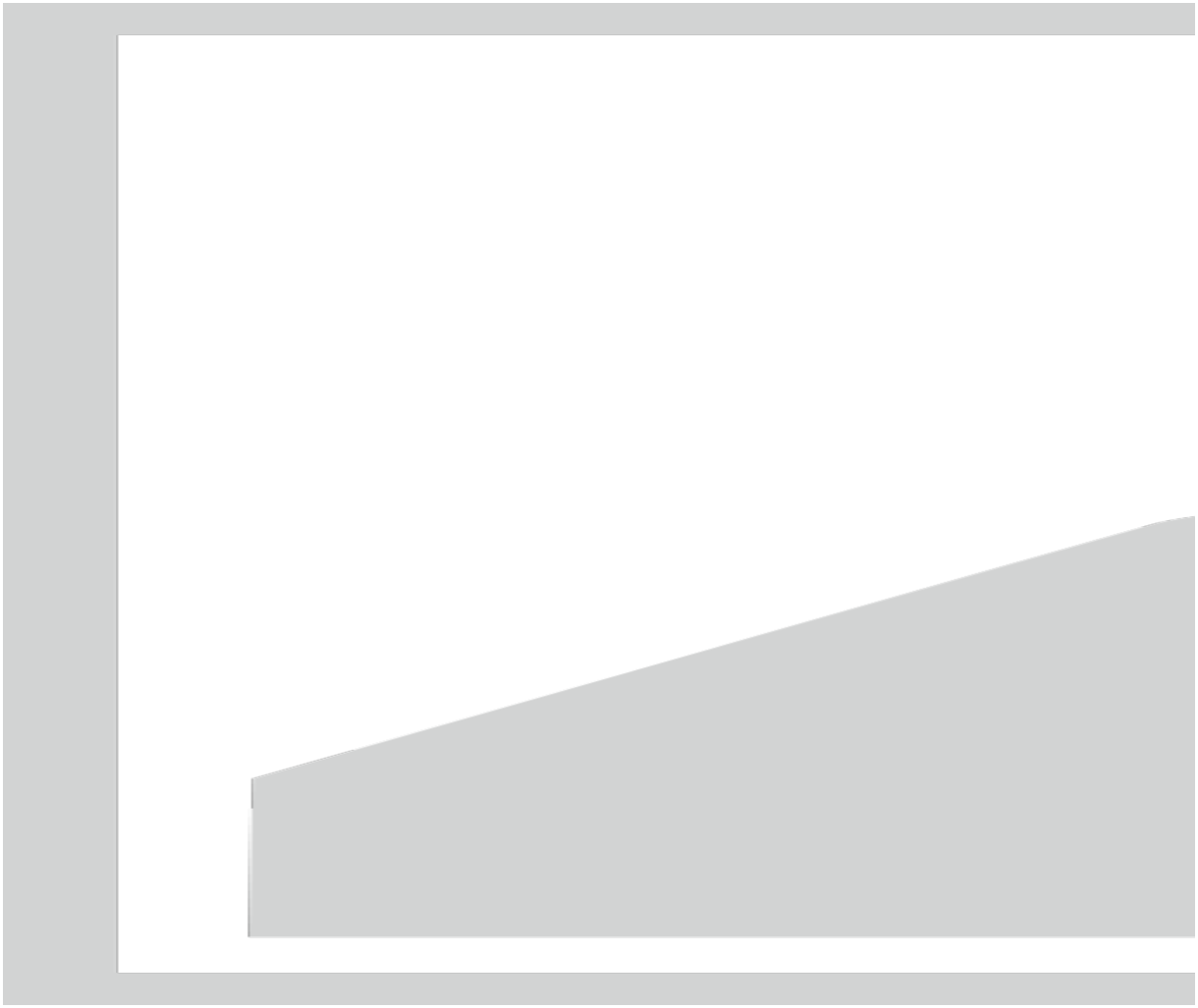
进入

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 my_blog/article/admin.py 中增加代码:

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

# Register your models here.
admin.site.register(Article)
```

保存后, 再次刷新页面, 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 中查看,

尝试使用 django-admin-bootstrap 美化后台管理界面

安装

```
$ pip install bootstrap-admin
```

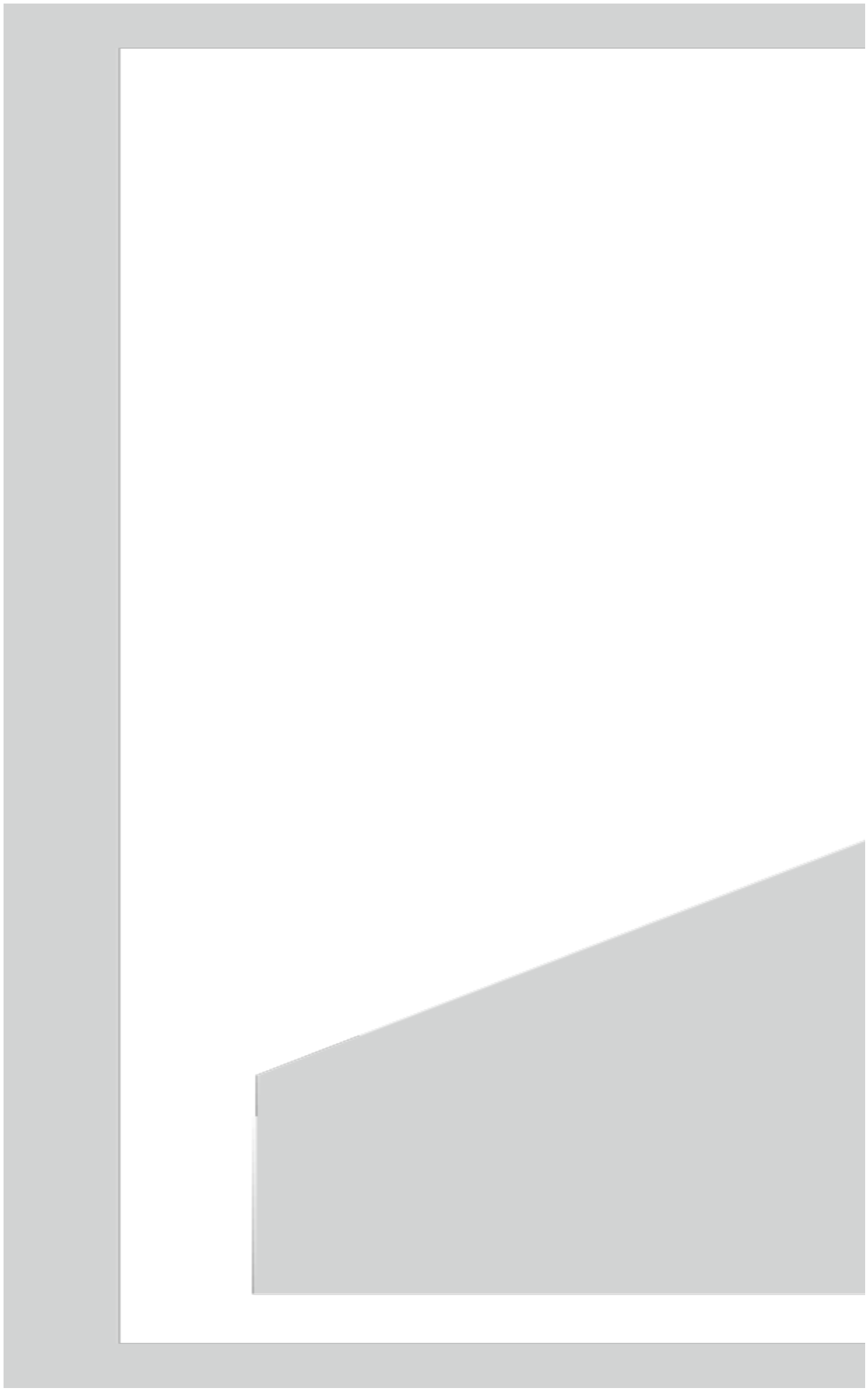
配置

然后在 my_blog/my_blog/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',
'article',
)

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



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 my_blog/article/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

`my_blog/my_blog/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'^$', 'my_blog.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

- `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 article/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 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 my_blog/article/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 my_blog/my_blog/urls.py 中设置对应的 url,

```
urlpatterns = patterns('',
# Examples:
# url(r'^$', 'my_blog.views.home', name='home'),
# url(r'^blog/', include('blog.urls')),

url(r'^admin/', include(admin.site.urls)),
url(r'^$', 'article.views.home'),
url(r'^(?P<my_args>\d+)/$', 'article.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 my_blog/article/views.py 代码:

```
from django.shortcuts import render
from django.http import HttpResponse
from article.models import Article

# Create your views here.
def home(request):
return HttpResponse("Hello World, Django")

def detail(request, my_args):
post = Article.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 Article 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 Article.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, 同样是响应式页面设置, 这也将是未来的主流吧..

在 my_blog 下添加文件名, 文件夹名为 `templates`

```
mkdir templates
#看到当前文件构成
my_blog
├── article
│   ├── __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
├── my_blog
│ ├── __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 my_blog/my_blog/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 article/view.py 中添加一个函数逻辑

```
from django.shortcuts import render
from django.http import HttpResponse
from article.models import Article
from datetime import datetime

# Create your views here.
def home(request):
return HttpResponse("Hello World, Django")

def detail(request, my_args):
post = Article.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 my_blog/urls.py 下

```
url(r'^test/$', 'article.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 my_blog/article/view.py, 并删除 test.html

```
# -*- coding: utf-8 -*-
from django.shortcuts import render
from django.http import HttpResponse
from article.models import Article
from datetime import datetime

# Create your views here.
def home(request):
post_list = Article.objects.all() #获取全部的 Article 对象
return render(request, 'home.html', {'post_list' : post_list})
```

修改 my_blog/my_blog/urls.py

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

urlpatterns = patterns('',
# Examples:
# url(r'^$', 'my_blog.views.home', name='home'),
```

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

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

```
my_blog
├── article
│   ├── __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
├── my_blog
│   ├── __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 `my_blog_tutorial`, 填写简单的描述

```
#查看当前目录位置
```

```
$ pwd
/Users/andrew_liu/Python/Django/my_blog

#在项目的根目录下初始化 git
git init
Initialized empty Git repository
in/Users/andrew_liu/Python/Django/my_blog/.git/

#添加远程 github
$ git remote add blog git@github.com:Andrew-liu/my_blog_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:Andrew-liu/my_blog_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 my_blog/article/views.py 代码:

```
# -*- coding: utf-8 -*-
from django.shortcuts import render
from django.http import HttpResponse
from article.models import Article
from datetime import datetime
from django.http import Http404

# Create your views here.
def home(request):
post_list = Article.objects.all() #获取全部的 Article 对象
return render(request, 'home.html', {'post_list' : post_list})

def detail(request, id):
try:
post = Article.objects.get(id=str(id))
except Article.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 my_blog/my_blog/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'^$', 'my_blog.views.home', name='home'),
# url(r'^blog/', include('blog.urls')),

url(r'^admin/', include(admin.site.urls)),
url(r'^$', 'article.views.home', name = 'home'),
url(r'^(?P<id>\d+)/$', 'article.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/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 article.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 article.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 任意阅读

- 阅读手册
- Github 源代码地址
- 联系作者

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 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 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 国人写的 django 使用教程
- Getting Started with - Django 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 - Django 教程的书籍, 最新版是 1.6 版本

- Getting Started with Django on - Heroku Django 部署在 Herohu 的官方文档
- Django Packages 所有 Django 的第三方库的聚集地
- -Awesome Django 个人应用集合 Django 第三方库的分类整理,强烈推荐看!!!!