Django搭建简易博客

极客学院出版

前言

本书主要面向 Django 学习者和博客开发者, 其中的代码可以在作者的 Github 上获得, 主要是在 Mac 上进行开发, 并没有在其他运行环境下做测试

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
- Github 源代码地址 (https://github.com/Andrew-liu/my_blog_tutorial)
- 鸣谢作者 (http://andrewliu.tk/)
- 关注作者Weibo (http://weibo.com/dinosaurliu)

目录

1 Django简介

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:

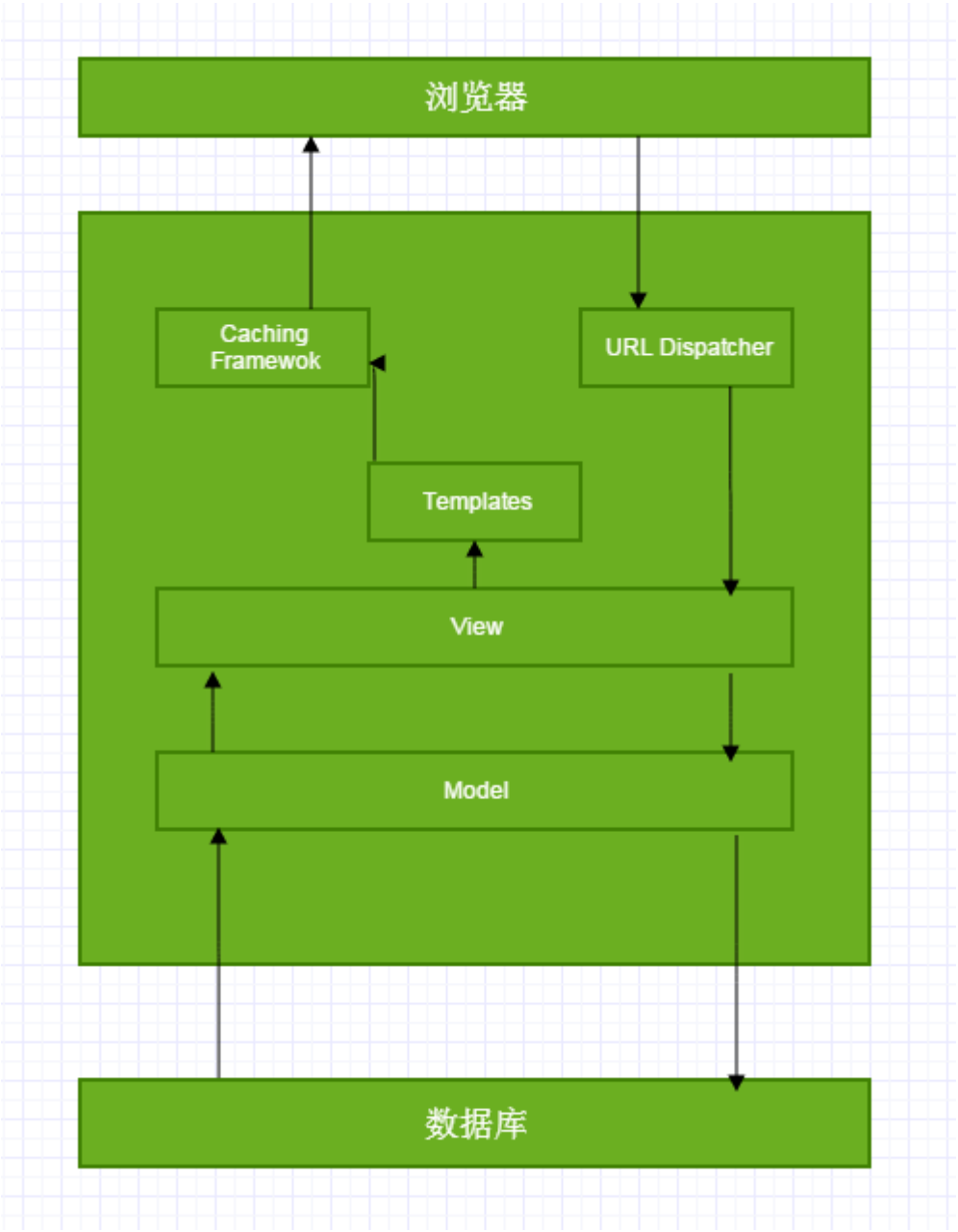
- 希望能写出一个系列文章, 我也不知道到底能写多少
- 能够让认真阅读这个系列的文章的人, 能在读完之后做出一个简单的博客
- 教会读者使用简单的git操作和github
- 希望能够加深自己对 Django 的理解

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.



图片 1.1 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

2

开发环境和Django安装

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 (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
```

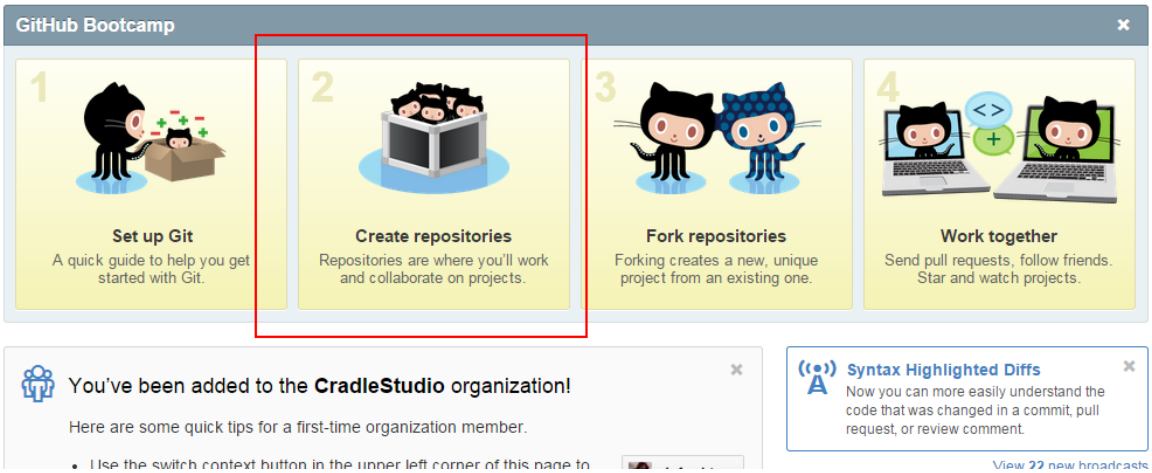
Ubuntu下git安装

```
$ 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



图片 2.1 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 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/#download) 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

3

项目与APP

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
```

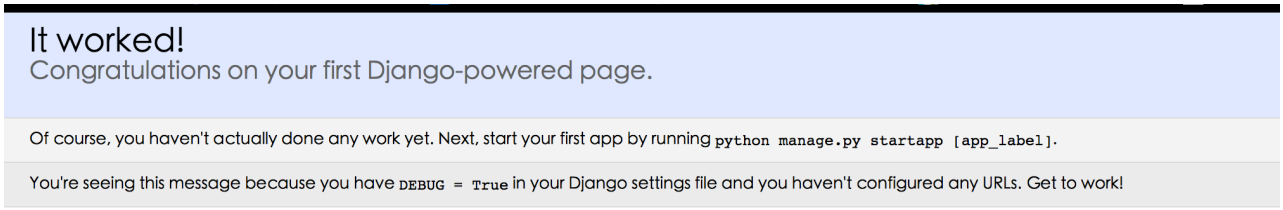
运行成功后,会显示如下信息

```
#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



图片 3.1 成功

说明你成功走出了第一步!

命令梳理:

```
python manage.py [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
```

4
Models

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`` 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

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


>>> #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.objects.get(id = 1)  #返回符合条件的对象


>>> #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=)
>>> 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
[]

Blog.objects.all()  # 选择全部对象
Blog.objects.filter(caption='blogname')  # 使用 filter() 按博客题目过滤
Blog.objects.filter(caption='blogname', id="1") # 也可以多个条件
#上面是精确匹配 也可以包含性查询
Blog.objects.filter(caption__contains='blogname')

Blog.objects.get(caption='blogname') # 获取单个对象 如果查询没有返回结果也会抛出异常
```

```
#数据排序
Blog.objects.order_by("caption")
Blog.objects.order_by("-caption") # 倒序

#如果需要以多个字段为标准进行排序（第二个字段会在第一个字段的值相同的情况下被使用到），使用多个参数就可以了
Blog.objects.order_by("caption", "id")

#连锁查询
Blog.objects.filter(caption__contains='blogname').order_by("-id")

#限制返回的数据
Blog.objects.filter(caption__contains='blogname')[0]
Blog.objects.filter(caption__contains='blogname')[0:3] # 可以进行类似于列表的操作
```

> 当然还有更多的API, 可以查看官方文档

Admin

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天管理的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

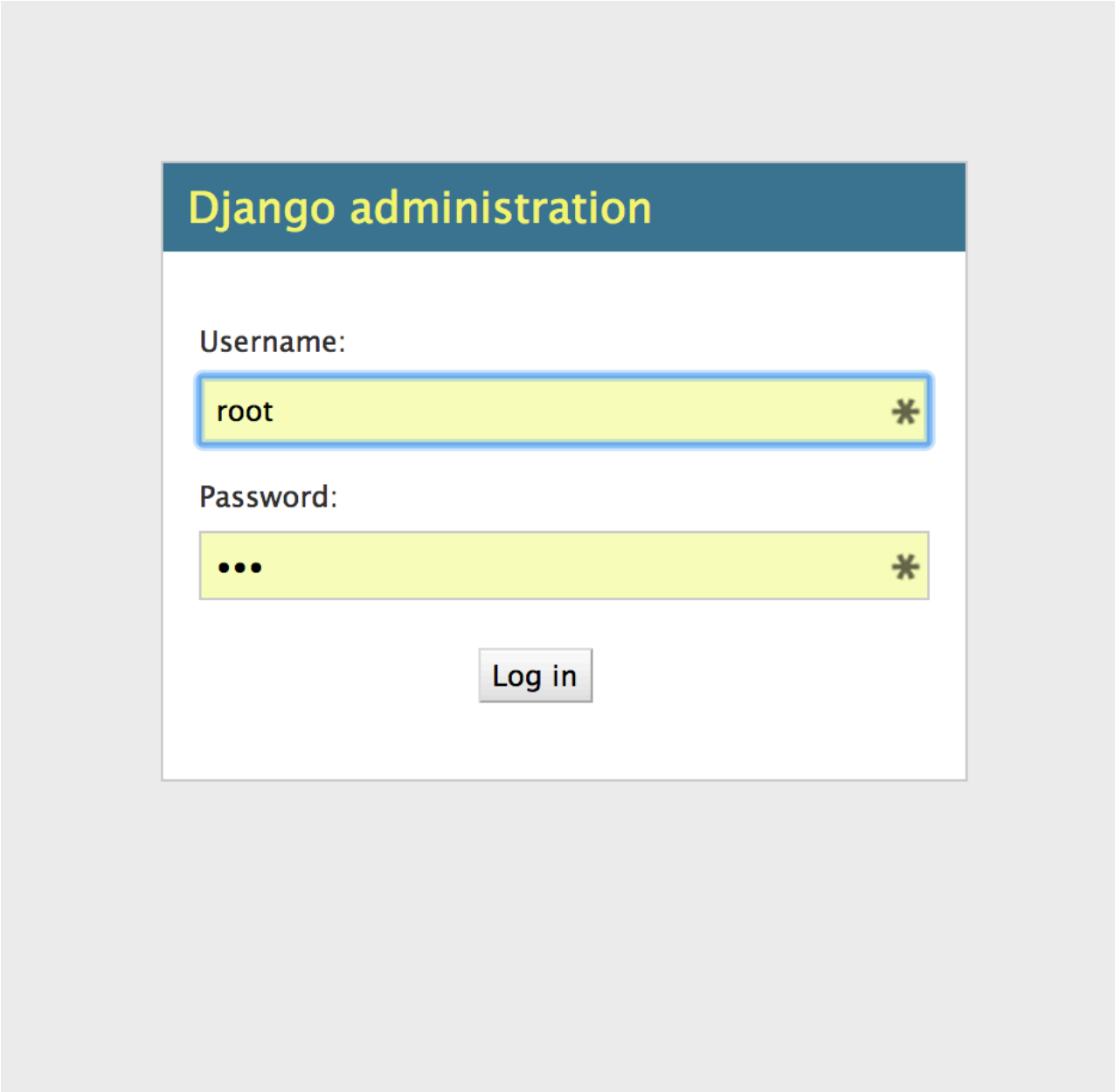
urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'my_blog.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    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'),
)
```

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][1]输入账户和密码进入后台管理, 如下:



图片 5.1 后台

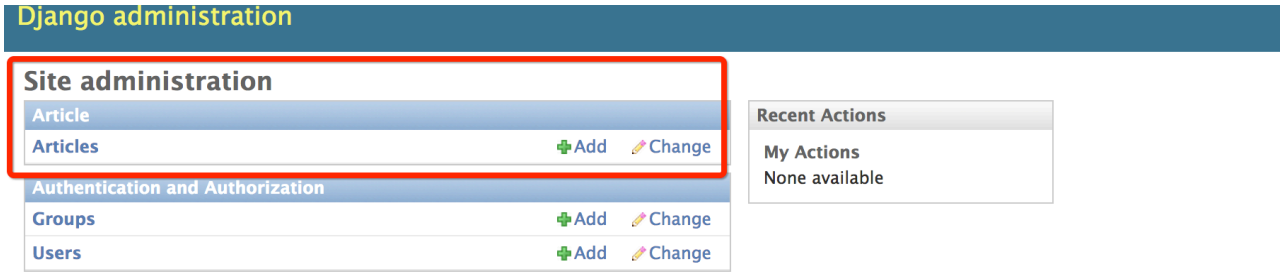
图片 5.2 进入

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`



图片 5.3 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][5]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

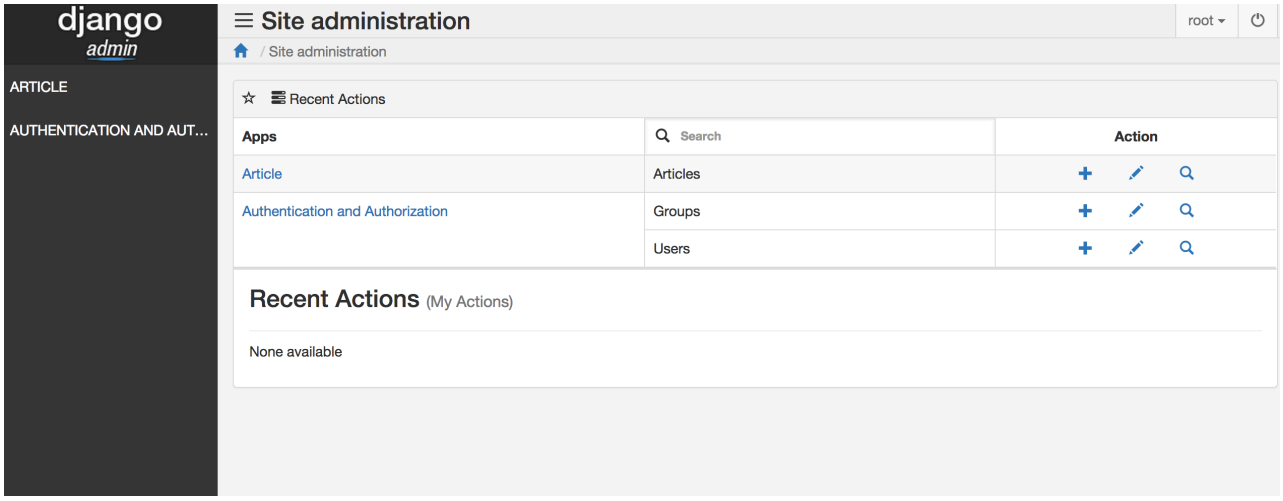
然后在my_blog/my_blog/setting.py中修改 `INSTALLED_APPS`

```
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',
)

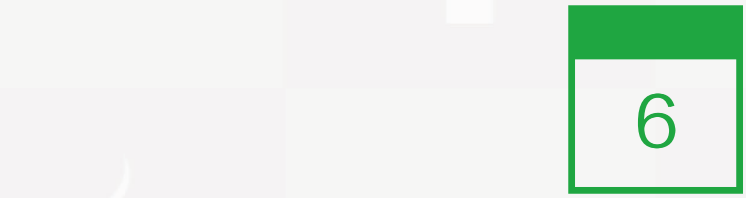
from django.conf import global_settings
TEMPLATE_CONTEXT_PROCESSORS = global_settings.TEMPLATE_CONTEXT_PROCESSORS + (
    'django.core.context_processors.request',
)
BOOTSTRAP_ADMIN_SIDEBAR_MENU = True
```

保存后, 再次刷新页面, `127.0.0.1:8000/admin`



图片 5.4 第三方

> 界面是不是美腻了许多...



6 Views和URL

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HTTP Response,

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 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'),  #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

图片 6.1 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

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

title = Hello World, category = python, date_time = 2014-12-27 02:12:04.773004+00:00, content = Hello Wordl

图片 6.2 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

7

Template

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][1], 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
```

在my_blog/my_blog/setting.py下设置templates的位置

```
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, 可以看到

图片 7.1 test

在template文件夹下增加base.html, 并在其中增加如下代码

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
          </p>
      </header>

        <div class="post-description">
          <p>
            {{ post.content }}
          </p>
        </div>
    </section>
  {% endfor %}
</div><!-- /.blog-post -->
{% endblock %}
```

其中

- `{% for in %}`与`{% endfor %}`成对存在, 这是template中提供的for循环tag
- `{% if %} {% 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})
```

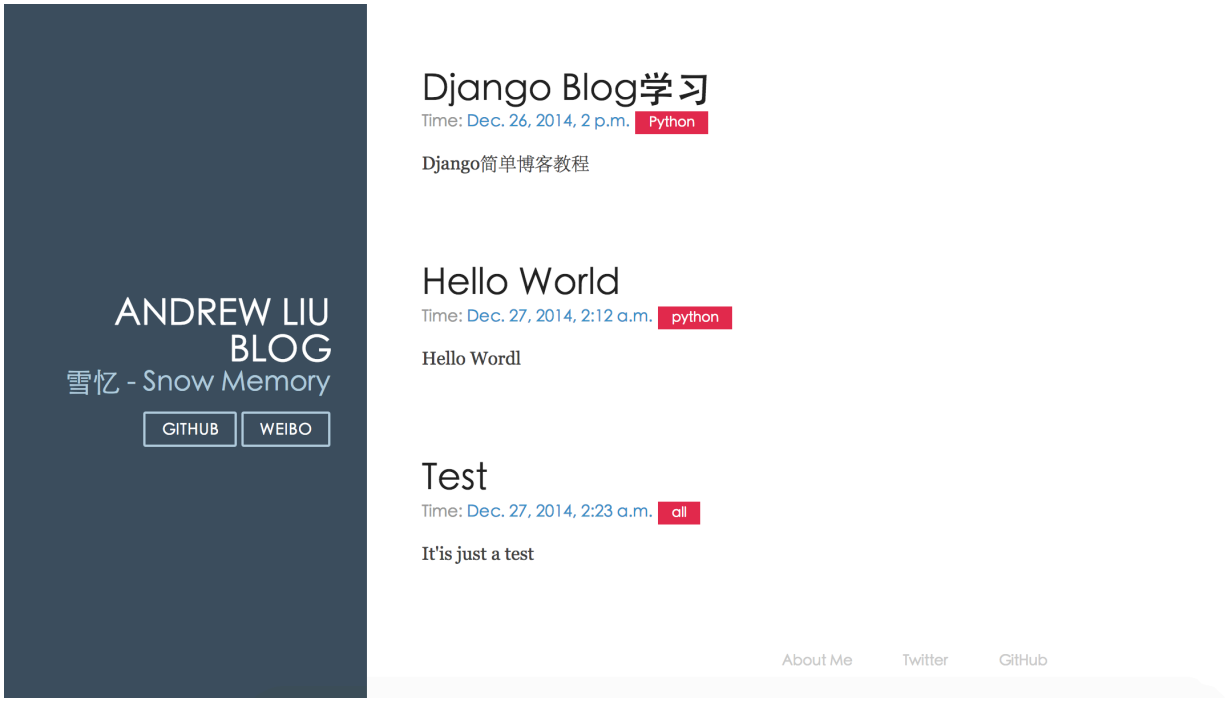
修改my_blog/my_blog/urls.py

```
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



图片 7.2 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.
```

8

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'^(?Pd+)/$', '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
          </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 pos
          </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>

</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, `'^(?Pd+)/$'` 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

图片 8.1 博客

9

多说,markdown和代码高亮

添加多说

在Django1.5版本前是有内置的评论系统的, 不过现在已经放弃使用了, 在国内比较常用的是 多说 , 在国外是 disqus , 因为文章主要面对 国内用户, 所以采用多说 (http://duoshuo.com/)

在网站上注册账号或者直接用社交账号进行登录,并会生成一个 short_name , 可以在个人界面中的工具中找到一段通用代码, 这段代码非常重要, 用于多说评论框的代码段:

```
<!-- 多说评论框 start -->
    <div class="ds-thread" data-thread-key="请将此处替换成文章在你的站点中的ID" data-title="请替换成文章的标题" data-url
<!-- 多说评论框 end -->
<!-- 多说公共JS代码 start (一个网页只需插入一次) -->
<script type="text/javascript">
var duoshuoQuery = {short_name:"请在此处替换成自己的短名"};
    (function() {
        var ds = document.createElement('script');
        ds.type = 'text/javascript';ds.async = true;
        ds.src = (document.location.protocol == 'https:' ? 'https:' : 'http:') + '//static.duoshuo.com/embed.js';
        ds.charset = 'UTF-8';
        (document.getElementsByTagName('head')[0]
         || document.getElementsByTagName('body')[0]).appendChild(ds);
    })();
    </script>
<!-- 多说公共JS代码 end -->
```

在templates中新建一个duoshuo.html并将代码放入其中, 并做一些修改

```
<!-- 多说评论框 start -->
    <div class="ds-thread" data-thread-key="{{ post.id }}" data-title="{{ post.title }}" data-url="{{ post.get_absolute_url }}">
<!-- 多说评论框 end -->
<!-- 多说公共JS代码 start (一个网页只需插入一次) -->
<script type="text/javascript">
var duoshuoQuery = {short_name:"andrewliu"};
    (function() {
        var ds = document.createElement('script');
        ds.type = 'text/javascript';ds.async = true;
        ds.src = (document.location.protocol == 'https:' ? 'https:' : 'http:') + '//static.duoshuo.com/embed.js';
        ds.charset = 'UTF-8';
        (document.getElementsByTagName('head')[0]
         || document.getElementsByTagName('body')[0]).appendChild(ds);
    })();
    </script>
<!-- 多说公共JS代码 end -->
```

然后在my_blog/article/models.py中重写get_absolute_url方法

```
from django.db import models
from django.core.urlresolvers import reverse

 # 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

 #获取URL并转换成url的表示格式
    def get_absolute_url(self):
        path = reverse('detail', kwargs={'id':self.id})
        return "http://127.0.0.1:8000%s" % path

    def __str__(self) :
        return self.title

    class Meta:
        ordering = ['-date_time']
```

然后修改 post.html

```
{% extends "base.html" %}
{% load custom_markdown %}

{% 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
                    </p>
            </header>

                <div class="post-description">
                    <p>
                        {{ post.content|custom_markdown }}
                    </p>
                </div>
        </section>
```

```
    {% include "duoshuo.html" %}
</div><!-- /.blog-post -->
{% endblock %}
```

只需要将 duoshuo.html 加入到当前页面中, {% include "duoshuo.html" %} 这句代码就是将duoshuo.html加入到当前的页面中

现在启动web服务器, 在浏览器中输入, 看看是不是每个博文页面下都有一个多说评论框了..

图片 9.1 多说

markdown你的博文

markdown 越来越流行, 越来越多的写博客的博主都喜欢上了makrdown这种标记性语言的易用性和美观性. 像 简书 (http://www.jianshu.com/) , 作业部落 (https://www.zybuluo.com/) , Mou (http://25.io/mou/) 都是比较出名的markdown在线或者离线形式

现在我们就来markdown自己的博客吗, 首先是安装 markdown 库, 使用下面命令

```
#首先是安装markdown
$ pip install markdown  #记得激活虚拟环境
```

现在说说怎么markdown你的博文, 在article下建立新文件夹 templatetags ,然后我们来定义的自己的 template filter, 然后在templatetags中建立 __init__.py , 让文件夹可以被看做一个包, 然后在文件夹中新建 custom_markdown.py 文件, 添加代码

```
import markdown

from django import template
from django.template.defaultfilters import stringfilter
from django.utils.encoding import force_text
from django.utils.safestring import mark_safe
```

```
register = template.Library()  #自定义filter时必须加上

@register.filter(is_safe=True)  #注册template filter
@stringfilter  #希望字符串作为参数
def custom_markdown(value):
    return mark_safe(markdown.markdown(value,
        extensions = ['markdown.extensions.fenced_code', 'markdown.extensions.codehilite'],
                                safe_mode=True,
                                enable_attributes=False))
```

然后只需要对需要进行markdown化的地方进行简单的修改,

```
{% extends "base.html" %}
{% load custom_markdown %}

{% 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
            </p>
        </header>

        <div class="post-description">
            <p>
                {{ post.content|custom_markdown }}
            </p>
        </div>
    </section>
    {% include "duoshuo.html" %}
</div><!-- /.blog-post -->
{% endblock %}
```

`{% load custom_markdown %}` 添加自定义的filter, 然后使用filter的方式为 `{{ post.content|custom_markdow n }}` , 只需要对需要使用markdown格式的文本添加管道然后再添加一个自定义filter就好了.

现在启动web服务器, 在浏览器中输入, 可以看到全新的的markdown效果

代码高亮

这里代码高亮使用一个CSS文件导入到网页中就可以实现了, 因为在上面写 markdown 的filter中已经添加了扩展高亮的功能, 所以现在只要下载CSS文件就好了.

在pygments (http://pygments.org/demo/440022/?style=paraiso-light_) 找到你想要的代码主题, 我比较喜欢 monokai , 然后在pygments-css (https://github.com/richleland/pygments-css) 下载你喜欢的CSS主题, 然后加入当博客目录的static目录下, 或者最简单的直接上传七牛进行CDN加速

修改 base.html 的头部

```
<!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>{% block title %} Andrew Liu Blog {% endblock %}</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">
    <link rel="stylesheet" href="http://picturebag.qiniudn.com/monokai.css">
</head>
```

<link rel="stylesheet" href="http://picturebag.qiniudn.com/monokai.css"> 添加CSS样式到base.html就可以了.">http://picturebag.qiniudn.com/monokai.css">`添加CSS样式到base.html就可以了.

现在启动web服务器, 添加一个带有markdown样式的代码的文章, 就能看到效果了, 在浏览器中输入 http://127.0.0.1:8000/


ANDREW LIU
BLOG
雪忆 - Snow Memory
主页
归档
GITHUB
WEIBO
专题
ABOUT ME
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
Read More >>>
Test
Time: 2014 Dec 27
All
It'is just a test
Read More >>>

图片 9.2 高亮

10

归档, AboutMe和标签分类

> 这一章节说的东西都是一些知识回顾,

归档

归档就是列出当前博客中所有的文章, 并且能够显示时间, 很容易的可以写出对应的view和模板来

在my_blog/my_blog/view下新建归档view

```
    def archives(request) :
        try:
            post_list = Article.objects.all()
        except Article.DoesNotExist :
            raise Http404
        return render(request, 'archives.html', {'post_list' : post_list,
                                                 'error' : False})
```

在my_blog/templates新建模板 archives.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 pos
                    </p>
            </header>
        </section>
    {% endfor %}
</div><!-- /.blog-post -->
{% endblock %}
```

并在my_blog/my_blog/usls.py中添加对应url配置

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

    urlpatterns = patterns('',

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

```
    url(r'^$', 'article.views.home', name = 'home'),
    url(r'^(?Pd+)/$', 'article.views.detail', name='detail'),
    url(r'^archives/$', 'article.views.archives', name = 'archives'),
)
```

AboutMe

这个就不多说了

在my_blog/my_blog/view.py下添加新的逻辑

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

在my_blog/template下新建模板aboutme.html, 内容如下, 大家可以自定义自己喜欢的简介

```
{% extends "base.html" %}
{% load custom_markdown %}

{% block content %}
<div class="posts">
      <p> About Me 正在建设中 </p>
</div><!-- /.blog-post -->
{% endblock %}
```

并在my_blog/my_blog/usls.py中添加对应url配置

```
 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'),
    url(r'^archives/$', 'article.views.archives', name = 'archives'),
    url(r'^aboutme/$', 'article.views.about_me', name = 'about_me'),
)
```

标签分类

实现功能: 点击对应的标签按钮, 会跳转到一个新的页面, 这个页面是所有相关标签的文章的罗列

只需要在在my_blog/my_blog/view.py下添加新的逻辑

```
    def search_tag(request, tag) :
        try:
            post_list = Article.objects.filter(category__iexact = tag) #contains
        except Article.DoesNotExist :
            raise Http404
        return render(request, 'tag.html', {'post_list' : post_list})
```

可以看成是对tag的查询操作, 通过传入对应点击的tag, 然后对tag进行查询

在对应的有tag的html网页中修改代码

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

{% load custom_markdown %}
{% 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-
                    </p>
            </header>

                <div class="post-description">
                    <p>
                        {{ post.content|custom_markdown }}
                    </p>
                </div>
                <a class="pure-button" href="{% url 'detail' id=post.id %}">Read More >>> </a>
        </section>
    {% endfor %}
</div><!-- /.blog-post -->
{% endblock %}
```

仔细看这一句 {{ post.category|title }} . 其中标签对超链接已经发生改变, 这是在对标签就行点击时, 会将标签作为参数, 传入到对应的view中执行逻辑, 然后进行网页跳转...

并在my_blog/my_blog/usls.py中添加对应url配置

```
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'),
    url(r'^archives/$', 'article.views.archives', name = 'archives'),
    url(r'^aboutme/$', 'article.views.about_me', name = 'about_me'),
    url(r'^tag(?P<tag>\w+)/$', 'article.views.search_tag', name = 'search_tag'),
)
```

现在在浏览器中输入 http://127.0.0.1:8000/ , 点击对应的归档或者ABOUT ME 或者标签按钮可以看到对应的效果

11

搜索和ReadMore

搜索功能

搜索功能的实现设计:

- 前段界面输入搜索关键字, 传送到对应view中
- 在对应的view中进行数据库关键字搜索
- 这里搜索可以只对文章名搜索或者全文搜索

首先在my_blog/templates下添加所有输入框

```
<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="{% url 'archives' %}">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>
                <form class="pure-form" action="/search/" method="get">
                <input class="pure-input-3-3" type="text" name="s" placeholder="search">
                </form>
                </li>
                <li class="nav-item">
                    <a class="button-success pure-button" href="{% url 'about_me' %}">About Me</a>
                </li>
            </ul>
        </nav>
```

```
    </div>
  </div>
```

在my_blog/article/views.py中添加查询逻辑

```
def blog_search(request):
  if 's' in request.GET:
    s = request.GET['s']
    if not s:
      return render(request,'home.html')
    else:
      post_list = Article.objects.filter(title__icontains = s)
      if len(post_list) == 0 :
        return render(request,'archives.html', {'post_list' : post_list,
                                                'error' : True})
      else :
        return render(request,'archives.html', {'post_list' : post_list,
                                                'error' : False})
  return redirect('/')
```

这里为了简单起见, 直接对 archives.html 进行修改, 使其符合查询逻辑

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

{% block content %}
<div class="posts">
  {% if error %}
    <h2 class="post-title">没有相关文章题目</a></h2>
  {% else %}
  {% 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 pos
          </p>
      </header>
    </section>
  {% endfor %}
  {% endif %}
</div><!-- /.blog-post -->
{% endblock %}
```

添加了if判断逻辑, 然后还需要修改 views中的archives

```
def archives(request) :
    try:
        post_list = Article.objects.all()
    except Article.DoesNotExist :
        raise Http404
    return render(request, 'archives.html', {'post_list' : post_list,
                                        'error' : False})
```

最后添加 `my_blog/my_blog/urls.py` 设置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'^$', 'article.views.home', name = 'home'),
    url(r'^(?P<id>\d+)/$', 'article.views.detail', name='detail'),
    url(r'^archives/$', 'article.views.archives', name = 'archives'),
    url(r'^aboutme/$', 'article.views.about_me', name = 'about_me'),
    url(r'^tag(?P<tag>\w+)/$', 'article.views.search_tag', name = 'search_tag'),
    url(r'^search/$','article.views.blog_search', name = 'search'),
)
```

ReadMore功能

对于ReadMore的前段按钮界面设置早已经添加过了, 所以这里只需要进行简单的设置就好了

通过使用Django中内建的filter就可以速度实现

```
    {{ value|truncatewords:2 }} #这里2表示要显示的单词数, 以后的会被截断, 不在显示
```

这里只需要修改my_blog/templates/home.html界面中的变量的过滤器

```
    #将正文截断设置为10
     {{ post.content|custom_markdown|truncatewords:20 }}
```

在浏览器中输入 `http://127.0.0.1:8000/` 可以看到效率(最好把博文设置的长一些)

RSS和分页

RSS功能

Django是一个全面型框架, 很多功能都可以直接找到, 对于RSS功能, 可以从其中的高层框架的 聚合Feed框架 中找到(The syndication feed framework (https://docs.djangoproject.com/en/1.7/ref/contrib/syndication/))

上层Feed生成框架可以直接应用 Feed类 , 我们可以直接继承Feed在其中定义自己的方法

在my_blog/article/views.py中定义类

```
from django.contrib.syndication.views import Feed  #注意加入import语句

class RSSFeed(Feed) :
    title = "RSS feed - article"
    link = "feeds/posts/"
    description = "RSS feed - blog posts"

    def items(self):
        return Article.objects.order_by('-date_time')

    def item_title(self, item):
        return item.title

    def item_pubdate(self, item):
        return item.add_date

    def item_description(self, item):
        return item.content
```

然后在my_blog/my_blog/urls.py中设置对应的url

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

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'),
    url(r'^archives/$', 'article.views.archives', name = 'archives'),
```

```
    url(r'^aboutme/$', 'article.views.about_me', name = 'about_me'),
    url(r'^tag(?Pw+)/$', 'article.views.search_tag', name = 'search_tag'),
    url(r'^search/$','article.views.blog_search', name = 'search'),
    url(r'^feed/$', RSSFeed(), name = "RSS"),  #新添加的urlconf, 并将name设置为RSS, 方便在模板中使用url
)
```

下面修改my_blog/templates/base.html, 在其中添加RSS按钮

```
<!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>{% block title %} Andrew Liu Blog {% endblock %}</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">
  <link rel="stylesheet" href="http://picturebag.qiniudn.com/monokai.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="{% url 'archives' %}">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>
          <form class="pure-form" action="/search/" method="get">
```

```
            <input class="pure-input-3-3" type="text" name="s" placeholder="search">
            </form>
            </li>
            <li class="nav-item">
              <a class="button-success pure-button" href="{% url 'about_me' %}">About Me</a>
            </li>
            <li class="nav-item">
              <a class="button-success pure-button" href="{% url 'RSS' %}">RSS</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>
```

保存后, 在浏览器中输入 `http://127.0.0.1:8000/` 可以看到新增的RSS按钮, 点击看以看到对应的效果

图片 12.1 RSS

Time: 2014 Dec 27 Python

动态URL

运行已经做好的博客框架, 会
立的页面.

我第一个想到的方法是给每
rail等是如果解决的, 我是第

Read More >>>

Test

Time: 2014 Dec 27 All

It'is just a test

Read More >>>

Hello World

Time: 2014 Dec 27 Python

```
<?xml version="1.0" encoding="utf-8"?>
<rss xmlns:atom="http://www.w3.org/2005/Atom" version="2.0"><channel><title>RSS fe
<description>RSS feed - blog posts</description><atom:link rel="self" href="http:/
11 Jan 2015 02:07:12 -0000</lastBuildDate><item><title>简单的Markdown测试</title><li

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

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,
法耦合性太强，而且用户不友好，缺点非常多

&gt; 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:

```python
# -*- 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
```

图片 12.2 RSS

更多功能可以查看The syndication feed framework (https://docs.djangoproject.com/en/1.7/ref/contrib/syndication/) 官方文档

分页功能

当博客文章较多的时候, 我们并不希望以此在主页上显示全部的博客文章, 而是希望能够每页显示固定的文章数目, 这样既能提高性能, 也能提高美观度, 何乐而不为呢?

> 现在这一章节来看看怎么实现分页功能

- 首先添加包
- 重写home方法
- 修改模板

修改my_blog/my_blog/views.py中的home函数

```
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

def home(request):
    posts = Article.objects.all()
    paginator = Paginator(posts, 2)
    page = request.GET.get('page')
```
```

```
    try :
        post_list = paginator.page(page)
    except PageNotAnInteger :
        post_list = paginator.page(1)
    except EmptyPage :
        post_list = paginator.paginator(paginator.num_pages)
    return render(request, 'home.html', {'post_list' : post_list})
```

修改my_blog/templates下的 home.html

```
 from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger  #添加包

def home(request):
    posts = Article.objects.all()  #获取全部的Article对象
    paginator = Paginator(posts, 2) #每页显示两个
    page = request.GET.get('page')
    try :
        post_list = paginator.page(page)
    except PageNotAnInteger :
        post_list = paginator.page(1)
    except EmptyPage :
        post_list = paginator.paginator(paginator.num_pages)
    return render(request, 'home.html', {'post_list' : post_list})
修改my_blog/templates下的home.html

{% extends "base.html" %}

{% load custom_markdown %}
{% 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-
                    </p>
            </header>

                <div class="post-description">
                    <p>
                        {{ post.content|custom_markdown|truncatewords:10 }}
                    </p>
                </div>
                <a class="pure-button" href="{% url 'detail' id=post.id %}">Read More >>> </a>
```

```
    </section>
  {% endfor %}

  {% if post_list.object_list and post_list.paginator.num_pages > 1 %}
    <div>
    <ul class="pager">
    {% if post_list.has_previous %}
      <li><a href="?page={{ post_list.previous_page_number }}">上一页</a></li>
    {% endif %}

    {% if post_list.has_next %}
      <li><a href="?page={{ post_list.next_page_number }}">下一页</a></li>
    {% endif %}
    </ul>
    </div>
  {% endif %}
</div><!-- /.blog-post -->
{% endblock %}
```

我设置的是每页显示两篇博文, 可以修改成其他数值

更多细节可以查看pagination (https://docs.djangoproject.com/en/1.7/topics/pagination/) 官方文档

保存后, 在浏览器中输入 `http://127.0.0.1:8000/` 可以看到新增的下一页按钮(博客文章要多加几篇), 点击看以看到对应的效果

> 到目前为止, 博客的基本功能都实现了, 下一篇我们将讲到怎么在云端进行部署

最后我们把整个blog更新到github仓库中

```
  $ git add . #添加全部更新到暂存区
  $ git commit -m "rss and paginator"  #提交到git
  [master b53356b] rss and paginator
   2 files changed, 24 insertions(+), 1 deletion(-)
  $ git push  #提交到github中
```

13

更上一层楼

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-started-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!!!!

极客学院

jikexueyuan.com

中国最大的IT职业在线教育平台

更多信息请访问

http://wiki.jikexueyuan.com/project/django-set-up-blog/