Django 学习

本章需要的知识点

- web 的初步认识
- 了解如何使用command-line
- 略懂Python基本语法
- 看得懂简单的HTML/CSS

目录

- Django介绍 & 安装
- Project & APP
- Model, View, Template(MVT)
- Admin
- Django ORM

Django 是什么？

> *Django 可以说是 Python 最著名的 Web Framework，一些知名的网站如 Pinterest,Instagram, Disqus 等等都使用过它来开发。*

项目地址：

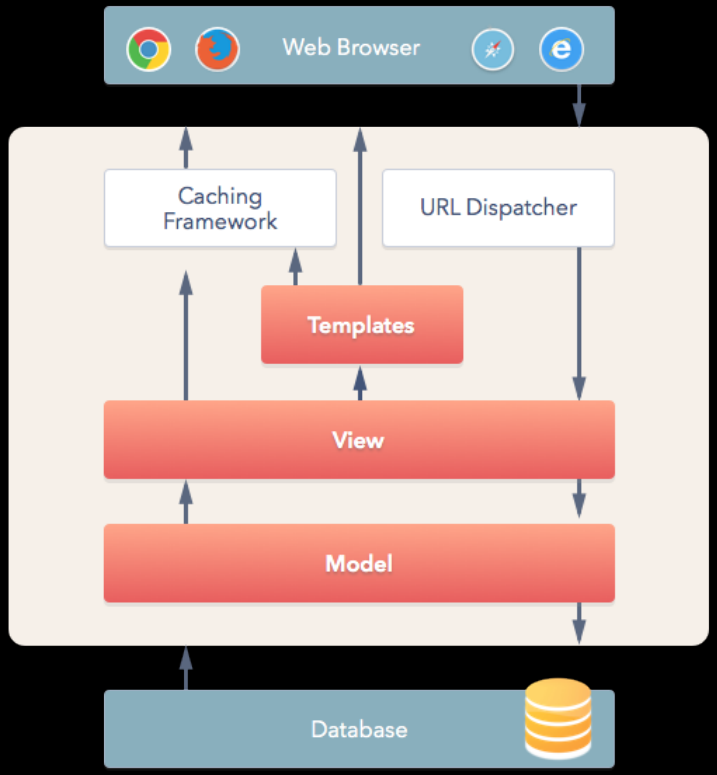
https://www.djangoproject.com/

web framework?

Web Framework 简单的说就是网站开发所使用的框架,它通常会提供给开发人员:

- 一个规范的程序框架
- 强大且丰富开发库

Django 框架架构


Web Browser
Caching Framework
URL Dispatcher
Templates
View
Model
Database

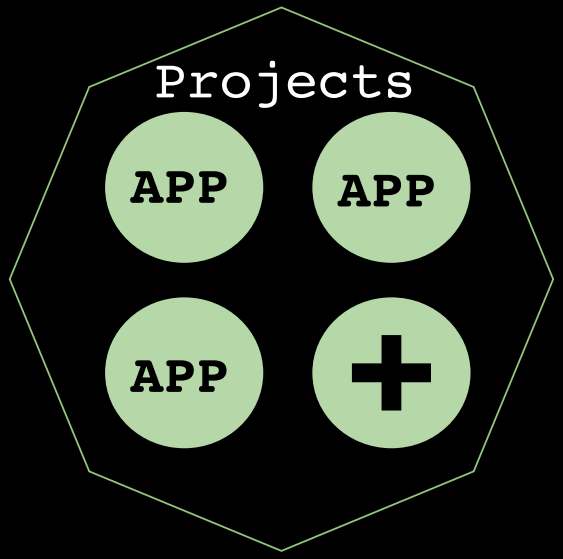
安装Django

使用pip工具安装Django， 最新版本1.9.5

```
pip install Django

# 确认安装成功
>>> import django
>>> django.VERSION
(1, 9, 5, 'final', 0)
>>>
```

Projects & APP

两者关系



一个project可以包含多个APP，比如一个微博web版，应该包含以下APP：

1. 用户配置管理(accounts)
2. 好友管理(friends)
3. 微博内容(timeline)
4. 动态消息管理(news)

创建方式

```
# 创建一个项目
django-admin.py startproject demo

# 在项目中创建APP
cd demo/
django-admin.py startapp web
```

```
$ tree
.
├── demo
│   ├── __init__.py
│   ├── settings.py # 项目配置文件
│   ├── urls.py     # 项目访问地址
│   └── wsgi.py     # wsgi app启动文件
├── manage.py       # 管理脚本
└── web
    ├── __init__.py
    ├── admin.py    # 后台管理配置模块
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py   # 模型模块
    ├── tests.py    # 测试模块
    └── views.py    # 视图模块

3 directories, 12 files
```

了解管理脚本

```
python manage.py <command> [options]
```

```
[auth]
    changepassword      #修改后台密码
    createsuperuser     #创建后台超级用户

[django]
    inspectdb           #根据数据库表结构声称model
    makemigrations      #声称migrations
    migrate             #执行migrate
    shell               #获取加载项目变量后的shell
    showmigrations      #查看migrations记录
    startapp            #创建一个APP
    startproject        #创建一个项目

[sessions]
    clearsessions       #清空sessions

[staticfiles]
    collectstatic       #集合静态文件
    findstatic          #找配置文件
```

准备: 1. 加入到项目

```
$ vim demo/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'web'  # 追加到"INSTALLED_APPS" 组里面
]
```

准备: 2. 初始化数据库

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, contenttypes, auth, sessions
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying sessions.0001_initial... OK
```

准备: 3. 创建管理用户

```
$ python manage.py createsuperuser
Username (leave blank to use 'jackeygao'): admin
Email address: gao@test.com
Password:
Password (again):
Superuser created successfully.
```

准备: 运行测试服务

```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
May 10, 2016 - 06:46:44
Django version 1.9.5, using settings 'demo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Model

什么是Model?

Django Model 用代码定义数据库数据表的结构(schema)，并通过Django指令创建数据表

Django Model的好处?

虽然SQL有响应的规范， 但是各个数据库的SQL还是有差异， Django Model正好尽可能的统一了一下， 形成统一的兼容代码级别接口。

定义后端数据库引擎

```
$ vim demo/settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

支持的后端数据库引擎

```
'django.db.backends.postgresql'
'django.db.backends.mysql'
'django.db.backends.sqlite3'
'django.db.backends.oracle'
```

Model

Field

Field就是字段， 就是数据表的字段, 字段有字段类型， Django Model内置丰富的字段类型根据场景使用. 比如字符串用 CharField, 整型用IntegerField

查阅更多字段类型：

https://docs.djangoproject.com/en/1.9/ref/models/fields/#field-types

创建Model

```
ClassRoomGrade = (
        (1, '一年级'),
        (2, '二年级'),
        (3, '三年级'),
        (4, '四年级'),
        (5, '五年级')
        )

classRoomIndex = (
        (1, '一班'),
        (2, '二班'),
        (3, '三班'),
        (4, '四班'),
        (5, '五班'),
        (6, '六班'),
        )
```

```
class ClassRoom(models.Model):
    grade = models.IntegerField(
            choices=ClassRoomGrade)
    index = models.IntegerField(
            choices=classRoomIndex)
    teacher = models.CharField(
            max_length=45, default="")

    def __str__(self):
        return "%s%s" % (self.get_grade_display(),
                self.get_index_display())
```

创建Model

```
class Student(models.Model):
    name = models.CharField(
            max_length=45,
            null=False,
            blank=False,
            )

    age = models.IntegerField()
    height = models.IntegerField()
    weight = models.IntegerField()
    classroom = models.ForeignKey(ClassRoom)

    def __str__(self):
        return "%s" % self.name
```

Django ORM

Django ORM

```
class Student(models.Model):
    name = models.CharField(
            max_length=45,
            null=False,
            blank=False,
            )

    age = models.IntegerField()
    height = models.IntegerField()
    weight = models.IntegerField()
    classroom = models.ForeignKey(ClassRoom)

    def __str__(self):
        return "%s" % self.name
```

Queryset API

```
# 根据主键select 可以使用get ， 因为是唯一的
>>> ClassRoom.objects.get(pk=2)
<ClassRoom: 一年级一班>

# 根据字段select 使用filter 返回的为list
>>> ClassRoom.objects.filter(grade=1, index=1)
[<ClassRoom: 一年级一班>, <ClassRoom: 一年级一班>]

# update where
>>> ClassRoom.objects.filter(grade=1, index=1).update(teacher="付老师")
2
>>> print ClassRoom.objects.get(pk=2).teacher
付老师

# 当条件为关系字段的时候使用"字段名" + 双下划线"__" + "sub 字段名"
>>> Student.objects.filter(classroom__grade=1, classroom__index=2)
[<Student: 张三>]
```

Queryset API

```
# 根据主键select 可以使用get ， 因为是唯一的
>>> ClassRoom.objects.get(pk=2)
<ClassRoom: 一年级一班>

# 根据字段select 使用filter 返回的为list
>>> ClassRoom.objects.filter(grade=1, index=1)
[<ClassRoom: 一年级一班>, <ClassRoom: 一年级一班>]

# update where
>>> ClassRoom.objects.filter(grade=1, index=1).update(teacher="付老师")
2
>>> print ClassRoom.objects.get(pk=2).teacher
付老师

# 当条件为关系字段的时候使用"字段名" + 双下划线"__" + "sub 字段名"
>>> Student.objects.filter(classroom__grade=1, classroom__index=2)
[<Student: 张三>]

# 删除
>>> Student.objects.filter(classroom__grade=1, classroom__index=2).delete()
(1, {u'web.Student': 1})
```

Queryset API

```
>>> from web.models import Student, ClassRoom
>>> ClassRoom(grade=1, index=1, teacher="李老师").save()
>>> ClassRoom(grade=1, index=2, teacher="王老师").save()

>>> class_room = ClassRoom.objects.filter(grade=1, index=2)[0]
>>> class_room
<ClassRoom: 一年级二班>
>>> Student(name="张三", age=6, height=138, weight=35, classroom=class_room).save()
>>> lisi = Student(name="李四", age=6, height=138, weight=35, classroom=class_room)
>>> lisi.name
'\xe6\x9d\x8e\xe5\x9b\x9b'
>>> lisi.age
6
>>> lisi.classroom.grade, lisi.classroom.index
(1, 2)
>>> lisi.classroom.get_grade_display(), lisi.classroom.get_index_display()
(u'\u4e00\u5e74\u7ea7', u'\u4e8c\u73ed')
>>> print lisi.classroom.get_grade_display(), lisi.classroom.get_index_display()
一年级 二班
>>>
```

同步数据库

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

Template

```
# 创建静态文件目录
mkdir -p web/static/
# 创建模版文件目录
mkdir -p web/templates/web/
```

编写模版

```
<table class="ui celled table">
 <thead>
   <tr>
     <th>Name</th>
     <th>Age</th>
     <th>Height</th>
     <th>Weight</th>
     <th>ClassRoom</th>
     <th>Teacher</th>
   </tr>
 </thead>
 <tbody>
   {% for student in students %}
   <tr>
     <td>{{ student.name }}</td>
     <td>{{ student.age }}岁</td>
     <td>{{ student.height }}cm</td>
     <td>{{ student.weight }}kg</td>
     <td>{{ student.classroom }}</td>
     <td><b>{{ student.classroom.teacher }}</b></td>
   </tr>
   {% endfor %}
 </tbody>
</table>
```

模版语言

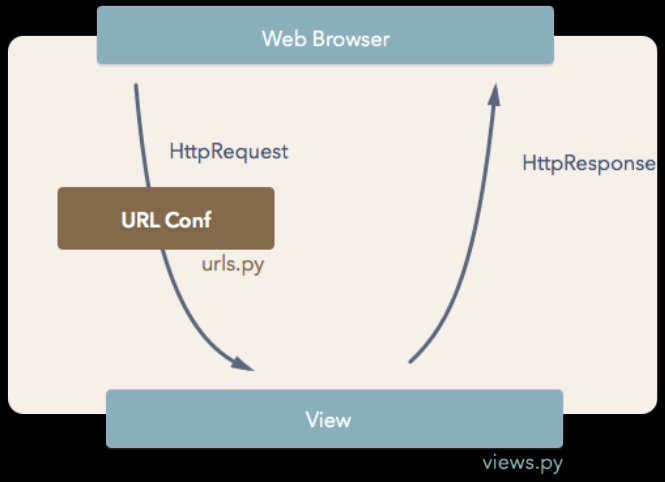
```
{% if xxxxx %}
{% elif xxxxxxx %}
{% else %}
{% endif %}


{% for i in xxx %}
{{ i.name }} # 插入变量
{% endfor %}

{{ i.name|upper }} # upper 为过滤器

{{ i.date|date:"%y %m" }} # date 格式化过滤器
```

查阅更多模版标签 和 过滤器：

https://docs.djangoproject.com/ja/1.9/ref/templates/builtins/

View

View怎么工作?



一个视图入口对应一个request对象， 返回一个response对象，通俗的讲view就是通过request来生成对应的response.

request对象

```
HttpRequest.body       # request body内容
HttpRequest.path       # request 请求路径
HttpRequest.path_info  # 请求路径信息
HttpRequest.method     # 请求方法
HttpRequest.encoding   # 请求内容编码
HttpRequest.GET        # GET方法参数
HttpRequest.POST       # POST方法参数
HttpRequest.COOKIES    # 浏览器Cookies信息
HttpRequest.FILES      # 请求携带文件
HttpRequest.META       # 请求元信息
        CONTENT_LENGTH – The length of the request body (as a string
        CONTENT_TYPE – The MIME type of the request body.
        HTTP_ACCEPT – Acceptable content types for the response.
        HTTP_ACCEPT_ENCODING – Acceptable encodings for the response
        HTTP_ACCEPT_LANGUAGE – Acceptable languages for the response
        HTTP_HOST – The HTTP Host header sent by the client.
        HTTP_REFERER – The referring page, if any.
        HTTP_USER_AGENT – The client's user-agent string.
        QUERY_STRING – The query string, as a single (unparsed) stri
        REMOTE_ADDR – The IP address of the client.
        REMOTE_HOST – The hostname of the client.
        REMOTE_USER – The user authenticated by the Web server, if a
        REQUEST_METHOD – A string such as "GET" or "POST".
        SERVER_NAME – The hostname of the server.
        SERVER_PORT – The port of the server (as a string).
```

response对象

```
HttpResponse.content             # 返回内容
HttpResponse.charset             # 返回内容编码
HttpResponse.status_code         # 返回状态 200 OK, 404 not found
HttpResponse.reason_phrase       #
HttpResponse.streaming           #
HttpResponse.closed              #


# response 子类
HttpResponseRedirect             # 转发301
HttpResponsePermanentRedirect    #
HttpResponseNotModified          # 201 没有任何改变操作
HttpResponseBadRequest           # 403 请求不规范
HttpResponseNotFound             # 404 not found
HttpResponseForbidden            # 401 没有权限
HttpResponseNotAllowed           # 方法不允许
HttpResponseGone                 #
HttpResponseServerError          # 500 内部服务器错误
```

编写view

```
$ vim web/views.py

from django.shortcuts import render_to_response
from web.models import Student


def student_list(request):
    students = Student.objects.all()
    return render_to_response('web/list.html', locals())
```

查阅view概念和更多使用方式：

https://docs.djangoproject.com/en/1.9/topics/http/views/

还差一步

URL调度 urls.py

```
$ vim demo/urls.py

from django.conf.urls import url
from django.contrib import admin
from web.views import student_list # 导入刚刚的视图

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', student_list), # 当访问首页的时候会进去到student_list 视图
]

r'^$' 代表 http://~/
r'^admin/' 代表 http://~/admin/
```

查阅url调度更多知识：

https://docs.djangoproject.com/ja/1.9/topics/http/urls/

运行测试服务器

```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
May 10, 2016 - 09:56:36
Django version 1.9.5, using settings 'demo.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

本案例源码

```
http://192.168.2.34/junqi.gao/learning-django
```

Q&A