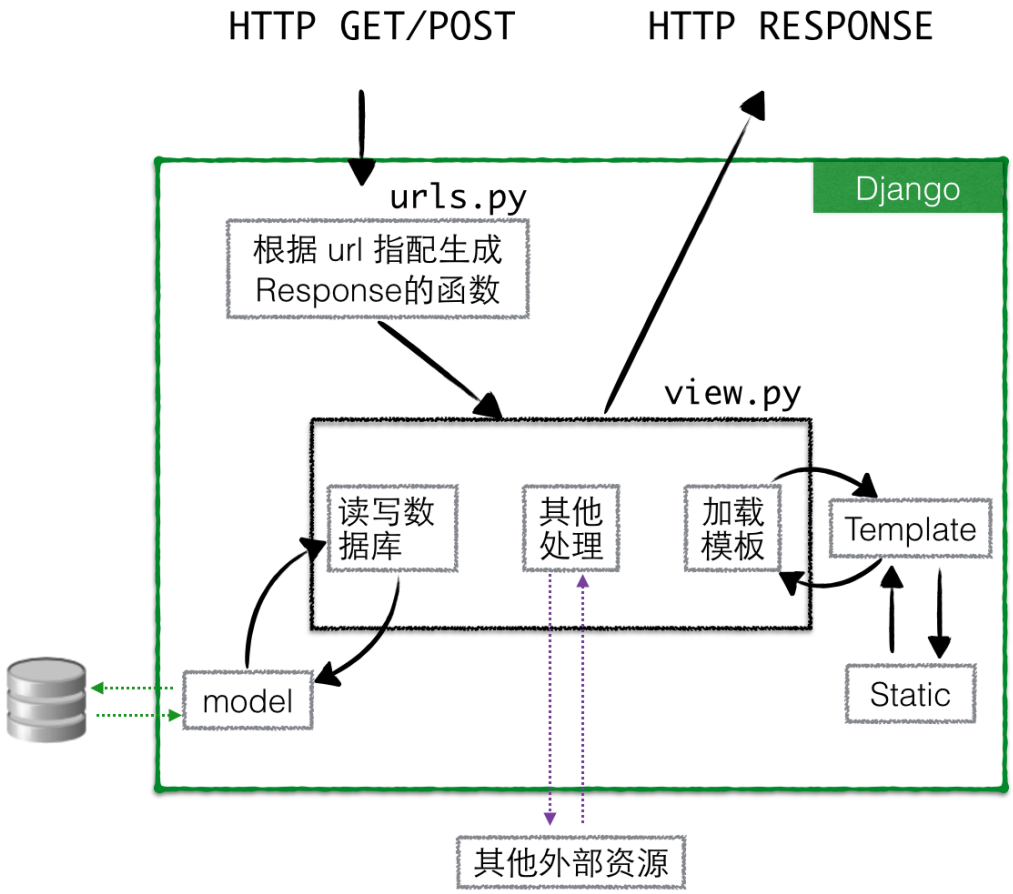
入门Django(上)

1. Django
    1. 一个web框架
    2. 支持动态网站、网络应用程序、网络服务的开发。
2. 用web框架写web应用，而不直接用python写web应用呢？
    1. 为了偷懒。
    2. 不用框架, 需要连接数据库、查询数据库、关闭数据库，python代码文件掺杂html标签、css样式。每始一个web应用，都要从头写起，重复许多枯燥无味的代码。
    3. web框架提供了通用web开发模式.
    4. Django web框架，有着数以万计的用户和贡献者，丰富的文档，活跃的社区，web开发很好的选择。
3. 本文结合 Django 官方文档 `First steps` 中的6个小教程
    1. 上篇主要来分析Django处理Http Request的机制
    2. 下篇介绍下Django的后台管理，以及单元测试等强大的功能。

Django 工作流程

1. Django 处理Http Resquest 的流程

1. 

    Django工作流程
2. Django如何根据url决定处理函数.
    1. Django要求程序员提供urls.py文件，该类文件指定请求链接与处理函数的一一对应关系。
        1. 正则表达式的方式指定请求链接, 并且不指定域名.
            1. 指定正则表达式 `^polls/\d+/$` 匹配http://127.0.0.1:8000/polls/12/ 即可。
        2. Django中在urls.py添加下语句即可。

```
urlpatterns = patterns(
    '',
    url(r'^polls/$', views.index),
)
```

2. 当请求链接为http://127.0.0.1:8000/polls/时，自动调用`views.py`中的函数`index()`处理请求。

3. views.index做哪些工作呢？
    1. 加载返回内容的模板， 如`index.html`。
        ```
        #views.py中 定义index函数
        def index(request):
            template = loader.get_template('polls/index.html')
        ```

4. 模板文件 是返回页面的一个骨架
    1. 模板中使用 静态文件, 如 css、javascript等
        1. 用`{% raw %}{% load staticfiles %}{% endraw %}`声明，然后直接引用即可. 如:
            ```
            <link rel="stylesheet" type="text/css" href="{% raw %}{% static
            'polls/style.css' %}{% endraw %}" />
            ```
    2. 模板中使用 参数和简单的逻辑语句
        ```
        {% raw %}{% for question in latest_question_list %}{% endraw %}
                <li>{{ question.question_text }}</a></li>
        {% raw %}{% endfor %}{% endraw %}
        ```
        2. for循环遍历latest_question_list，逐个输出question_text。
            1. 参数latest_question_list的值是 views.py 中计算出来给模板文件的
        3. 这里假设 从数据库中取出最新的5个question，如下：
            1. `latest_question_list = Question.objects.order_by('-pub_date')[:5]`

5. Django 封装了数据库的读写操作
    1. Django 封装了数据库
        1. 不用SQL语句去查询、更新数据库等
        2. 用python的方式定义数据库结构 (model.py 定义数据库)
        3. 用python的方式读写内容。
        4. 连接数据库、关闭数据库 交给Django 完成。
    2. 例子: Question数据库结构的定义：
        ```
        class Question(models.Model):
            question_text = models.CharField(max_length=200)
            pub_date = models.DateTimeField('date published')

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

6. 有模板文件骨架，又有参数、逻辑语句、静态文件等血肉，一个丰满的页面就诞生了

7. 完整的index函数

```
def index(request):
    latest_question_list = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = RequestContext(request, {
        'latest_question_list': latest_question_list,
    })
    return HttpResponse(template.render(context))
```

第一个Django项目

1. Django中projects和apps的区别。
    1. App : 做某件事的web应用， 如 一个用户认证系统，或 公开投票系统；
    2. project : 是一个web站点， 包括许多app和一些配置。
    3. 一个project 包含许多app，一个app可以用于许多project中。
2. 用Django时, 先创建一个project， 如: mysite，如下：
    1. mysite
        1.
```
$ django-admin.py startproject mysite
$ tree -L 2 mysite
mysite
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

1 directory, 5 files
```
        2. `mysite/settings.py` 项目配置
            1. 配置时区，数据库连接, 或 应用的添加、删除等
            2. 数据库设置，Django支持sqlite、mysql、oracle等
                1. 先安装、启动数据库，建立相应的账户，后 使用。
            3. 这里 用python内置的sqlite，settings里面的数据库配置不更改。
    2. 生成相应的数据库表
        1. `$ python manage.py migrate`
            1. migrate是Django 1.7引入的命令，较早的版本 用其他的命令代替
        2. 为什么新建的空项目里有数据库表呢？
            1. 默认情况，项目配置文件`settings.py`里有Django自带的应用，如下：
                1.
```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
)
```
            2. 这些应用和数据库交互。(通常默认的应用很有用，可以根据需求删减)

3. python manage.py makemigrations和migrate的区别

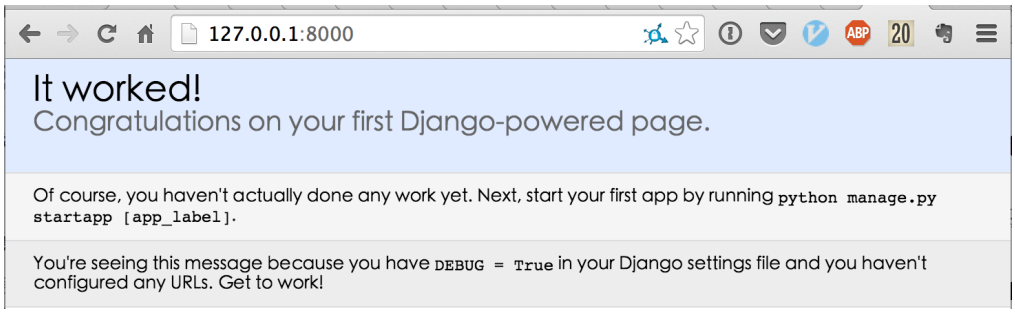
    1. python manage.py makemigrations 生成迁移
       定义完models.py，django会根据属性生成迁移，在项目中的migrtations文件夹中会生成一个0001_initial.py文件
    2. python manage.py migrate 执行迁移
       django会根据0001_initial.py文件迁移表，执行迁移，将定义的各个属性写入mysql中自动生成一张表，表 名是应用名_models的类名

3. 到此, Django项目可以运行. 开启Django自带的`开发环境: web服务`：

    1. 开启自带的开发环境

        1. `$ python manage.py runserver`

4. 调试: 在浏览器打开 http://127.0.0.1:8000/ ，看看Django的 `It worked!` 页面。

    1. 

    Django 欢迎页面

3. 增加新功能

    1. 任务：

        1. 在一个问答系统中添加问题；
        2. 显示所有已经添加的问题。
        3. 这个任务 涉及到向后台写数据，从后台读取数据

    2. 新建 名为`questions`的app(完成这项任务)

        1. `$ python manage.py startapp questions`

    3. 设计数据库, 存储问题

        1. 建名为Question的表格，有context字段。
        2. Django提供 models 设计数据库. `questions/models.py`

            1.
```
from django.db import models

class Question(models.Model):
    context = models.CharField(max_length=200)
```

    4. 将`questions`应用添加到项目的配置文件`mysite/settings`：

        1. `mysite/settings`的中添加app应用

            1.
```
INSTALLED_APPS = (
    'django.contrib.admin',
    ...,
    'questions',
)
```

    5. 命令生成Question数据库表格

        1. `$ python manage.py makemigrations questions`
        2. `$ python manage.py migrate`

    6. 处理函数与URL一一匹配

1. 设计三个URL地址. 地址 不包含域名
    1. `add/, add_done/, index/` 分别 展示填写问题页面，添加成功后页面，显示所有问题页面。
2. 在`mysite/urls.py`中指定相应的处理函数，如下：
    1. ```
       from django.conf.urls import patterns, include, url
       from questions import views

       urlpatterns = patterns(
           '',
           url(r'^add/$', views.add),    #路由与处理函数一一配对
           url(r'^index/$', views.index),
           url(r'^add_done/$', views.add_done),
       )
       ```
3. 在`questions/views.py`中实现 index, add, add_done处理函数：
    1. 三个处理函数
        1. index: 获取问题，传给模板文件，返回Response；
        2. add: 返回添加问题表单页面；
        3. add_done: 获取POST得到的问题，将其添加到数据库，返回Response；
    2. ```
       #代码
       def index(request):
           question_list = Question.objects.all()

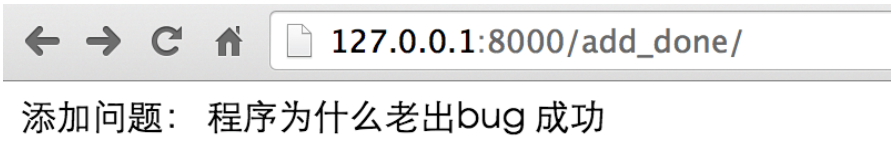
           return render(
               request,
               "questions/index.html",
               {'question_list': question_list},
           )

       def add_done(request):
           add_question = Question()
           content = request.POST['content']
           add_question.context = content
           add_question.save()
           return render(
               request,
               "questions/add_done.html",
               {'question': content},
           )

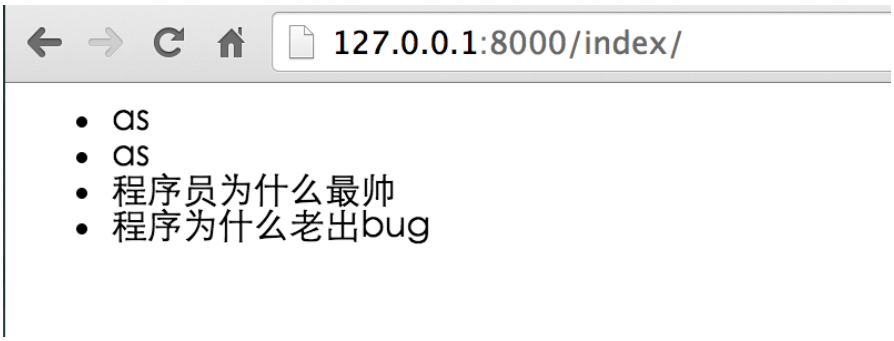
       def add(request):
           return render(request, "questions/add.html")
       ```
    3. render函数加载模板, 以字典的形式传递参数，返回Response页面。
        1. 模板文件内容???，
4. 运行截图如下：

1. 

添加问题

2. 

添加成功

3. 

列出问题

以上即 Django处理Http Request的过程

入门Django(下)

为项目添加一个简单的后台， 管理 Question 数据库

写测试单元查看程序有没有什么Bug。

后台管理

1. 首先 添加后台管理员账号:
    1. `createsuperuser` 命令 添加后台管理员账号
        1. ```
           $ python manage.py createsuperuser
           Username (leave blank to use 'feizhao'): happy
           Email address:
           Password:
           Password (again):
           Superuser created successfully.
           ```
    2. 通过 http://127.0.0.1:8000/admin/ 进入管理员登录页面。
        1. 用创建的管理员账号登录
        2. 看到`Groups`和`Users`，它们是Django内置的认证模块`django.contrib.auth`提供的数据库
           ```

3. 进入Users 看到 创建的管理员用户happy 。

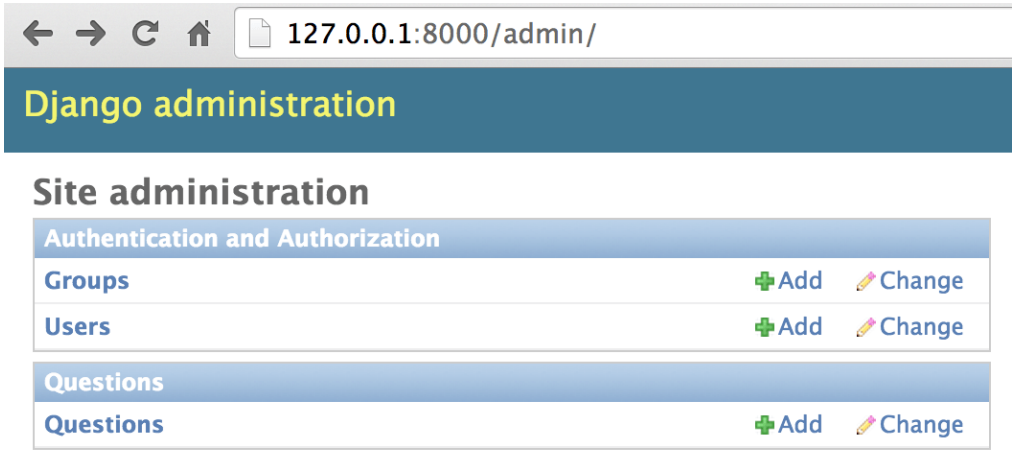
2. 注册数据库

    1. 目前后台还看不到 Question 数据库，因为告诉后台它的存在。
    2. 在questions应用的 `admin.py` 文件里 注册该数据库. 注册语句:

        1. ```
           #admin.py
           from django.contrib import admin
           from questions.models import Question

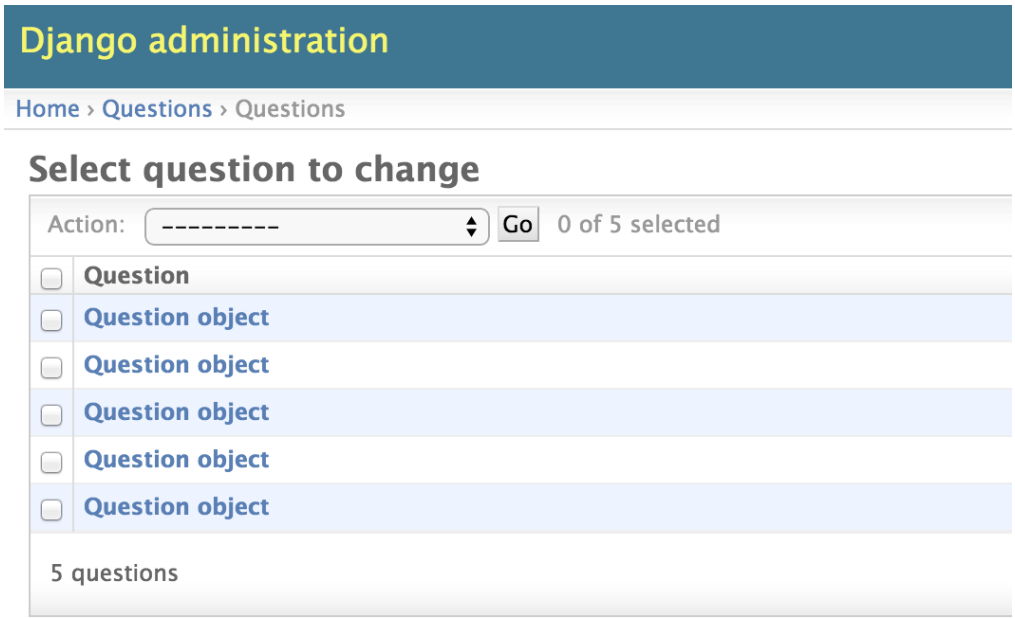
           admin.site.register(Question)
           ```

    3. 刷新后台，看到Question数据库. 如图:

        1. 

           Question数据库

    4. 进入Question数据库，看到每一条记录

        1. 

           数据库记录

        2. 默认情况，每条记录是 `str()` 返回的内容.

            1. 而我们没有在 `class Question(models.Model)` 中覆盖该方法
            2. 可以指定数据库记录显示某个字段. 方法，修改admin.py：
           ```

1. ```
   #admin.py
   class QuestionAdmin(admin.ModelAdmin):
       list_display = ('context',)

   admin.site.register(Question, QuestionAdmin)
   ```

2. 这样每条记录显示的是context内容了. 进去某条记录, 会看到所有的字段，并且可以进行更新、删除、添加等操作

3. Django后台的可定制性 非常高，可以按照自己爱好打造属于自己的后台。

自动化测试

1. Django 提供了完整的自动化测试机制. 以 questions 应用为例
    1. 发现: 问题描述框没有输入任何内容时点击提交，仍然会跳转到添加成功页面. 也就是说 添加了一个空的问题. 这当然不是我们想要的
    2. 写一个程序 测试我们的添加问题的功能。
2. Django 实现测试:
    1. 在questions应用 新建`tests.py`文件， 里面写好测试逻辑，然后用 django 的测试系统完成测试。
        1. 测试程序questions/tests.py：
            1. ```
               #tests.py
               from django.test import TestCase
               from django.test import Client

               class QuestionMethodTests(TestCase):
                   def test_add(self):
                       client = Client()
                       response = client.post('/add_done/', {'content': ""})
                       self.assertNotEqual(response.status_code, 200)
               ```
    2. 模拟一个客户端client, 将空字符串传给content字段，然后发起一个post请求到`/add_done/`页面(默认情况下测试时并不检查CSRF字段)，然后断言post请求不成功(也就是返回包的状态码不为200)。
        1. 运行测试程序：
            1. ```
               $ python manage.py test questions
               Creating test database for alias 'default'...
               F
               ======================================================================
               FAIL: test_add (questions.tests.QuestionMethodTests)
               ----------------------------------------------------------------------
               Traceback (most recent call last):
                 File
               "/Users/feizhao/Documents/python_demo/mysite/questions/tests.py
               ", line 10, in test_add
                   self.assertNotEqual(response.status_code, 200)
               AssertionError: 200 == 200

               ----------------------------------------------------------------------
               Ran 1 test in 0.009s
               ```
               ```

```
FAILED (failures=1)
Destroying test database for alias 'default'...
```

2. 测试没通过，说明确实插入了空白问题。
    1. 测试 不需要运行web服务. 这样节省HTTP服务的开销，提高测试的速度。
    2. 对views中的add_done改动，如下：

```
#views.py
def add_done(request):
    content = request.POST['content']
    if content != "":
        add_question = Question()
        add_question.context = content
        add_question.save()
        return render(
            request,
            "questions/add_done.html",
            {'question': content},
        )
    else:
        return redirect("/add/")
```

        1. 首先检查字符串是否为空.
            1. 若空重定向页面到`/add/`，
            2. 若不为空则添加问题成功。
    3. 再次运行测试程序，则通过测试，结果如下：

```
$ python manage.py test questions
Creating test database for alias 'default'...
.
----------------------------------------------------------------------
-------
Ran 1 test in 0.007s

OK
Destroying test database for alias 'default'...
```

3. 这个应用还有bug 是一个问题可能重复提交多次，这里不详细阐述。

命令行交互

1. 想验证 某条语句, 或 输出某个变量.
    1. 直接在项目实现非常麻烦。
    2. 用python解释器的交互模式，可避免手动导入django的配置环境
    3. 运行`python manage.py shell`，然后用django的API. 在当前项目目录进行交互，如下：

        1.
```
$ python manage.py shell
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
(InteractiveConsole)
```

```
>>> from questions.models import Question
>>> null_question = Question()
>>> null_question.save()
>>> for question in Question.objects.all():
...     print question.context
...
as
as
程序员为什么最帅
程序为什么老出bug

>>>
```

2. 交互模式用起来事半功倍

深入学习

1. Django中国社区是国内的Django开发社区，人气不是很旺
    1. @evilbinary在这里一个博客，兼容wp，代码高亮功能支持 提供了一个用Django搭建的博客，并给出了源码， 可以学习
2. 一些不错的Django开源项目
    1. 如 BBS论坛fairybbs，
    2. 这个登录的应用django-siteuser。
3. 中文的教程 djangobook 2.0，
    1. 书中Django版本太低，不推荐使用。
    2. 英文的资料还是挺丰富，不过还是推荐读文档，虽然文档有时候特别坑人(被坑了好多次)。
4. **Stackoverflow**
    1. 谁用谁知道，不用担心英语太烂，放代码和错误提示，实在不行用Google翻译加一点描述就行
    2. stackoverflow作为全球最大的技术问答网站
5. Segmentfault 思否
    1. 这些问答网站，很多Django用户在邮件列表(邮件列表是`groups.google.com`，所以你懂的)里提问题、回答问题，这里的氛围非常不错，各种问题都有人来帮你。
6. 决定好好玩Django了， 先看一下Django FAQ，可能会解决关于Django的一些疑问。