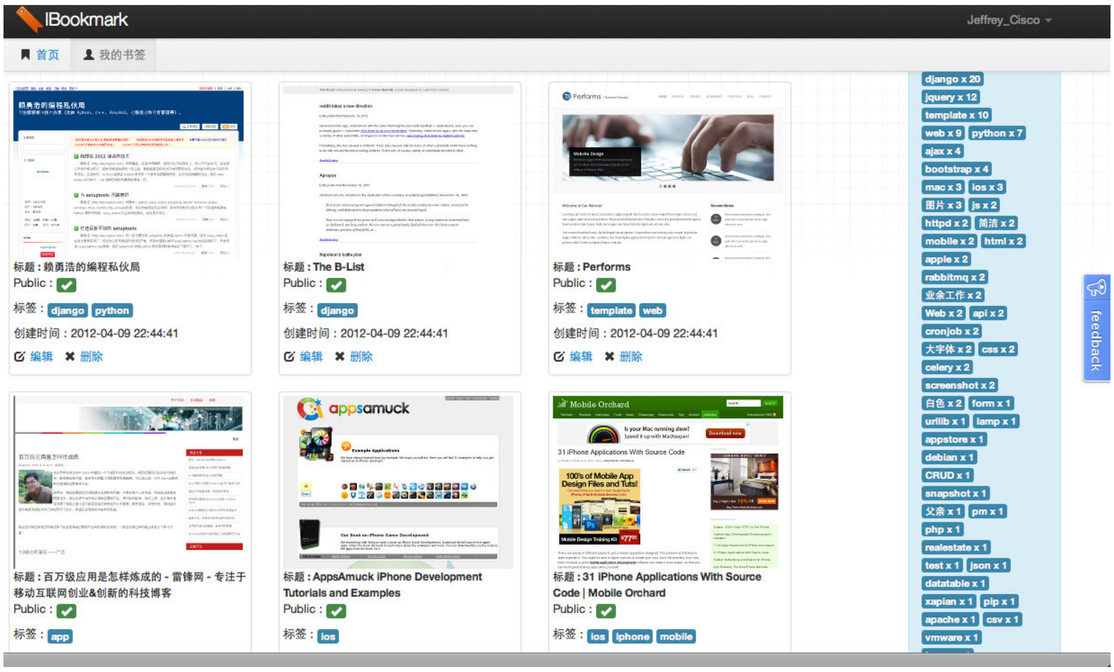
Django web application development tutorial

The goal of this tutorial

Recently, We developed a Django web app named "iBookmark (http://www.ibookmark.me)", which can help users manage their bookmark online, just like Google and delicious bookmark. Users can import and export their bookmarks from browsers and Google, delicious.



标题：赖勇浩的编程私伙局
Public : ✓
标签：django python
创建时间：2012-04-09 22:44:41
编辑 删除

标题：The B-List
Public : ✓
标签：django
创建时间：2012-04-09 22:44:41
编辑 删除

标题：Performs
Public : ✓
标签：template web
创建时间：2012-04-09 22:44:41
编辑 删除

标题：百万级应用是怎样炼成的 - 雷锋网 - 专注于移动互联网创业&创新的科技博客
Public : ✓
标签：app

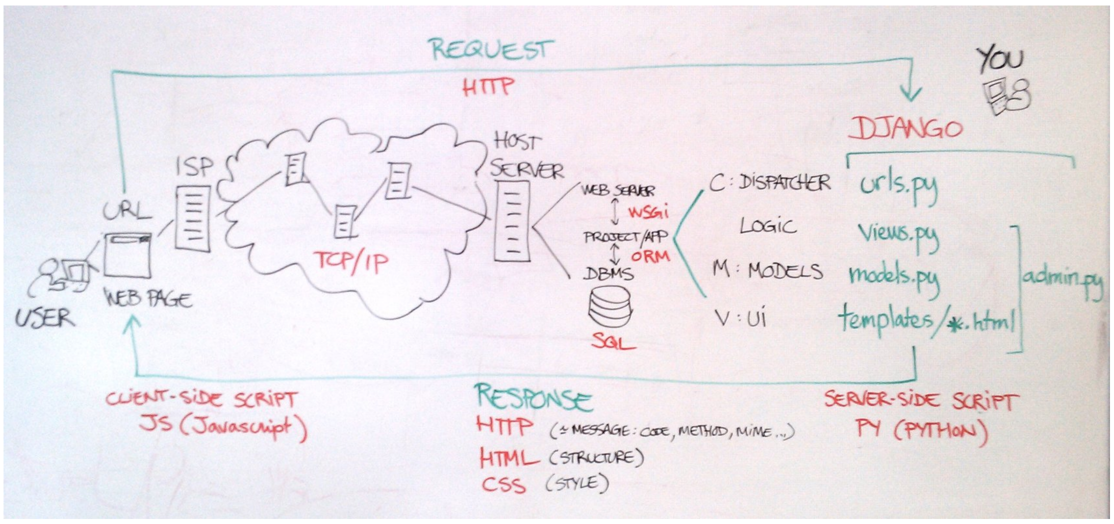
标题：AppsAmuck iPhone Development Tutorials and Examples
Public : ✓
标签：ios

标题：31 iPhone Applications With Source Code | Mobile Orchard
Public : ✓
标签：ios iphone mobile

The snapshot of the http://ibookmark.me

The tutorial goal is to show how to use Python/Django to build a small popular web app. I'll use the "iBookmark" as the example for this tutorial.

Introduction to Django



The Django architecture

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

What is Django

Django is a high-level Python Web framework...

A Web framework is software that abstracts common problems of Web development and provides shortcuts for frequent programming tasks.

A good web framework finds the “pain points” of web developers and smoothes them over — but never gets in the way! It should let you work at a much higher level of abstraction, so you don’t need to worry about details of HTTP, SQL, or whatever. Again, it shouldn’t get in your way if you need to “step down” a level.

Python itself is another key “feature” of Django. It’s a beautiful, concise, powerful, high-level language with an amazing community; many things are easier in Django simply because of the tools you can build on top of.
Django also makes a point of not changing anything about how Python works; if you learn Django, you’re also learning Python. This helps down the road.

...Django encourages rapid development...

Regardless of how many powerful features it has, a Web framework is worthless if it doesn't save you time (Do you mean Java?). Django's philosophy is to do all it can to facilitate hyper-fast development. With Django, you build Web sites in a matter of hours, not days; weeks, not years.

This comes directly out of real-world problems. We're programmers, yes, but we work at a news organization. When a big story breaks, we don't have the luxury of a long development cycle. Every convenience in Django is there because it makes you more productive.

...and clean, pragmatic design.

Django strictly maintains a clean design throughout its own code and makes it easy to follow best Web-development practices in the applications you create.
The philosophy here is to make it easy to do things the "right" way.

Django philosophy

https://docs.djangoproject.com/en/1.4/misc/design-philosophies/

Loose coupling

A fundamental goal of Django's stack is loose coupling and tight cohesion. The various layers of the framework shouldn't "know" about each other unless absolutely necessary.

For example, the template system knows nothing about Web requests, the database layer knows nothing about data display and the view system doesn't care which template system a programmer uses.

Although Django comes with a full stack for convenience, the pieces of the stack are independent of another wherever possible.

Less code

Django apps should use as little code as possible; they should lack boilerplate. Django should take full advantage of Python's dynamic capabilities, such as introspection.

Quick development

The point of a Web framework in the 21st century is to make the tedious aspects of Web development fast. Django should allow for incredibly quick Web development.

Don’t repeat yourself (DRY)

Every distinct concept and/or piece of data should live in one, and only one, place. Redundancy is bad. Normalization is good.
The framework, within reason, should deduce as much as possible from as little as possible.
See also the discussion of DRY on the Portland Pattern Repository (http://c2.com/cgi/wiki?DontRepeatYourself)

Explicit is better than implicit

This is a core Python principle, and it means Django shouldn’t do too much “magic.” Magic shouldn’t happen unless there’s a really good reason for it. Magic is worth using only if it creates a huge convenience unattainable in other ways, and it isn’t implemented in a way that confuses developers who are trying to learn how to use the feature.

Consistency

The framework should be consistent at all levels. Consistency applies to everything from low-level (the Python coding style used) to high-level (the “experience” of using Django).

Other frameworks

Compared to other frameworks, Django is quick, not dirty, full-stack and popular in job markets.

How to learn Django

1: Learn Python
2: Learn Django concepts and conventions
3: Google and Document

Preparing the development environment

From this chapter, we'll begin our trip to develop a real but tiny web app using Django. First, we should prepare the development environment.

Step 1: Install the python and PIP

Since Django only supports the python 2.x, so we should install the python 2.x. Download the latest python 2.x to install. After install please check the result as following:

```
$ python
Python 2.7.1 (r271:86832, Jul 31 2011, 19:30:53)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "hello world"
hello world
>>> exit()
```

PIP is a tool for installing and managing Python packages, such as those found in the Python Package Index. It's a replacement for easy_install. To install PIP, we can check install document at:
http://www.pip-installer.org/en/latest/installing.html

We can put all dependent third python libraries into a txt file, such as requirement.txt, such as:

```
#All dependences here#
django
# If you need a special version #
# django==1.4#
```

Then we can install all required packages using:

```
pip install -r webapp/requirement.txt
```

Step 2 Setup Django development environment

```
$ pip install django
Downloading/unpacking django
  Downloading django-1.4.tar.gz (7.6Mb): 7.6Mb downloaded
  Running setup.py egg_info for package django

Installing collected packages: django
```

```
    Running setup.py install for django
      changing mode of build/scripts-2.7/django-admin.py from 644 to 755

      changing mode of /Users/jeffrey/Dev/django_tutorial/env/bin/django-admin.py to 755
Successfully installed django
Cleaning up...
```

To check the version of django:

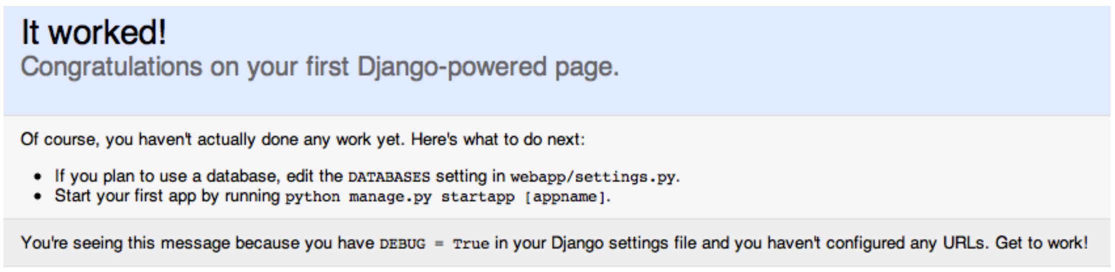
```
>>> import django
>>> django.get_version()
'1.4'
```

Step 3: Using Django-admin to build the a project

Django provides a tool to do Django development tasks, such as build a project, add an app, sync database, etc. We'll introduce the tool later, now we need use it to build a project.

```
$ django-admin.py startproject webapp
$ cd webapp/
$ python manage.py runserver
Validating models...

0 errors found
django version 1.3.1, using settings 'webapp.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

If you open browser and visit the server, you can find the server should work.



It worked!
Congratulations on your first Django-powered page.

Of course, you haven't actually done any work yet. Here's what to do next:

- If you plan to use a database, edit the `DATABASES` setting in `webapp/settings.py`.
- Start your first app by running `python manage.py startapp [appname]`.

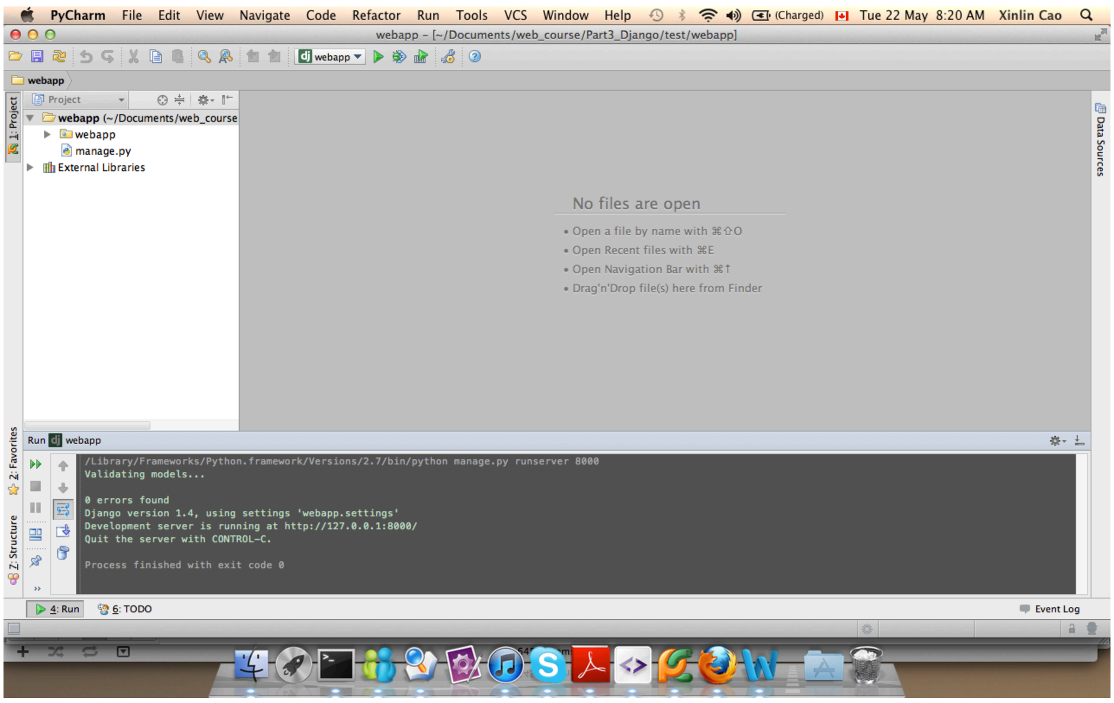
You're seeing this message because you have `DEBUG = True` in your Django settings file and you haven't configured any URLs. Get to work!

Choose an IDE?

Here are some IDEs which you can choose:

- Text editor
- Eclipse + pyDev
- PyCharm

We suggest user can choice PyCharm, We like this IDE (http://www.jetbrains.com/pycharm/).

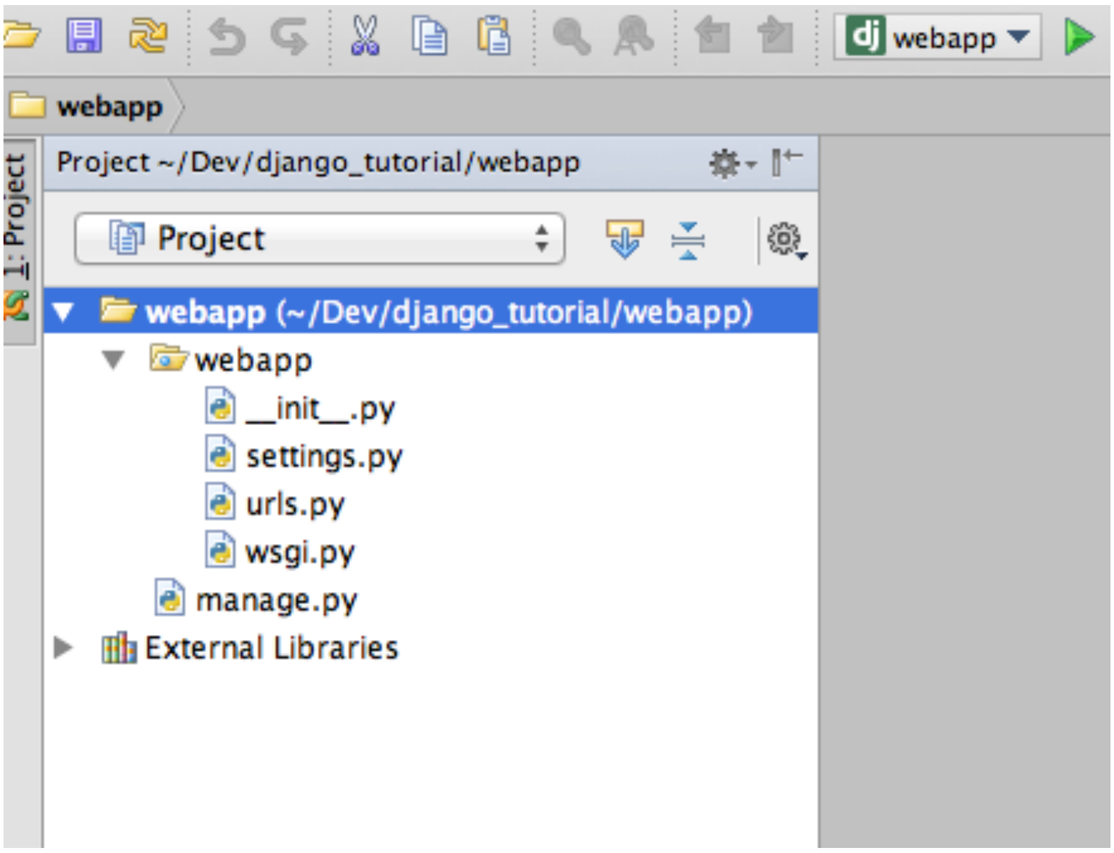


```
/Library/Frameworks/Python.framework/Versions/2.7/bin/python manage.py runserver 8000
Validating models...

0 errors found
Django version 1.4, using settings 'webapp.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

Process finished with exit code 0
```

PyCharm File Edit View Navigate Code Refactor Run Tools VCS Window Help
webapp - [~/Documents/web_course/Part3_Django/test/webapp]
webapp (~/Documents/web_course
webapp
manage.py
External Libraries
No files are open
Open a file by name with ⌘⇧O
Open Recent files with ⌘E
Open Navigation Bar with ⌘↑
Drag'n'Drop file(s) here from Finder
Run webapp
4: Run
6: TODO
Event Log

Inspect the Django project structure

Remember the latest web app?



The following is a list of file generated in Django project folder:

1. The outer webapp/ directory is just a container for your project. Its name doesn't matter to Django; you can rename it to anything you like.
2. manage.py: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in django-admin.py and manage.py.
3. The inner webapp/ directory is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. import webapp.settings).
4. webapp/__init__.py: An empty file that tells Python that this directory should be considered a Python package. (Read more about packages in the official Python docs if you're a Python beginner.)
5. webapp/settings.py: Settings/configuration for this Django project. Django settings will tell you all about how settings work.
6. webapp/urls.py: The URL declarations for this Django project; a "table of contents" of your Django-powered site. You can read more about URLs in URL dispatcher.
7. webapp/wsgi.py: An entry-point for WSGI-compatible webservers to serve your project. See How to deploy with WSGI for more details.Prepare development

What is WSGI?

WSGI is the Web Server Gateway Interface. It is a specification for web servers

and application servers to communicate with web applications (though it can also be used for more than that). It is a Python standard, described in detail in PEP 333.
For more, see Learn about WSGI
(http://wsgi.readthedocs.org/en/latest/learn.html).

Basic request & response

In the last chapter, we have built a web app, and run it worked! But it's only for demo and no practical purpose. Now we need go on to build our real application base on it.

Setup the database

In settings.py, we should define the database：

```
import os.path
PROJECT_ROOT = os.path.abspath(os.path.dirname(__file__))
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.sqlite3',
'NAME': os.path.join(PROJECT_ROOT, 'dev.db'),
'USER': '', # Not used with sqlite3.
'PASSWORD': '', # Not used with sqlite3.
'HOST': '', # Set to empty string for localhost. Not used with sqlite3.
'PORT': '', # Set to empty string for default. Not used with sqlite3.
}
}
```

And need to run "python manage.py syncdb" in webapp directory.

```
$python manage.py syncdb
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
```

```
You just installed django's auth system, which means you don't have any
superusers defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'jeffrey'): admin
E-mail address: admin@admin.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
No fixtures found.
```

Note: If you encounter any errors about encoding, you can type “export LC_ALL="en_US.UTF-8"”(https://code.djangoproject.com/ticket/16017) in the terminal to solve this issue.

After we have generated database, then we can use some sqlite took, such as sqlitemanager, to debug and manage the database.

Add "apps" package directory in "webapp"

We used the "apps" directory as the application root path. Please note that “apps” is a Python package directory, it need to include a file named "__init__.py", please do not forget it. In "apps" package, we create the first app:

```
$ cd webapp/webapp
$ mkdir apps
$ cd apps
$ touch __init__.py

$ python ../../manage.py startapp bookmark
$ cd bookmark
total 24
drwxr-xr-x    6 jeffrey    staff    204    4 16 20:26 .
drwxr-xr-x    4 jeffrey    staff    136    4 16 20:26 ..
-rw-r--r--    1 jeffrey    staff      0    4 16 20:26 __init__.py
-rw-r--r--    1 jeffrey    staff     57    4 16 20:26 models.py
-rw-r--r--    1 jeffrey    staff    383    4 16 20:26 tests.py
-rw-r--r--    1 jeffrey    staff     26    4 16 20:26 views.py
```

You can see in webapp/webapp/apps/bookmarks/ package, there are 4 python files.

1. models.py: defined the models. A model is the single, definitive source of data about your data. It contains the essential fields and behaviors of the data you're storing. django follows the DRY Principle. The goal is to define your data model in one place and automatically derive things from it.

2. views.py: defined the views. A view is a “type” of Web page in your django application that generally serves a specific function and has a specific template.

Then we need enable our created application in settings.py：

```
import sys
sys.path.insert(0, os.path.join(PROJECT_ROOT))
sys.path.insert(0, os.path.join(PROJECT_ROOT, "apps"))

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    # 'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
    'bookmark',
)
```

Defining views for app

A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image . . . or anything, really. The view itself contains whatever arbitrary logic is necessary to return that response. This code can live anywhere you want, as long as it’s on your Python path. There’s no other requirement–no “magic,” so to speak. For the sake of putting the code somewhere, the convention is to put views in a file called views.py, placed in your project or application directory.

Imaging for the bookmark objects, the owner can create/update/delete/view a bookmark, and list bookmarks. So we need add some method in views, for every method can correspond an option of the owner.

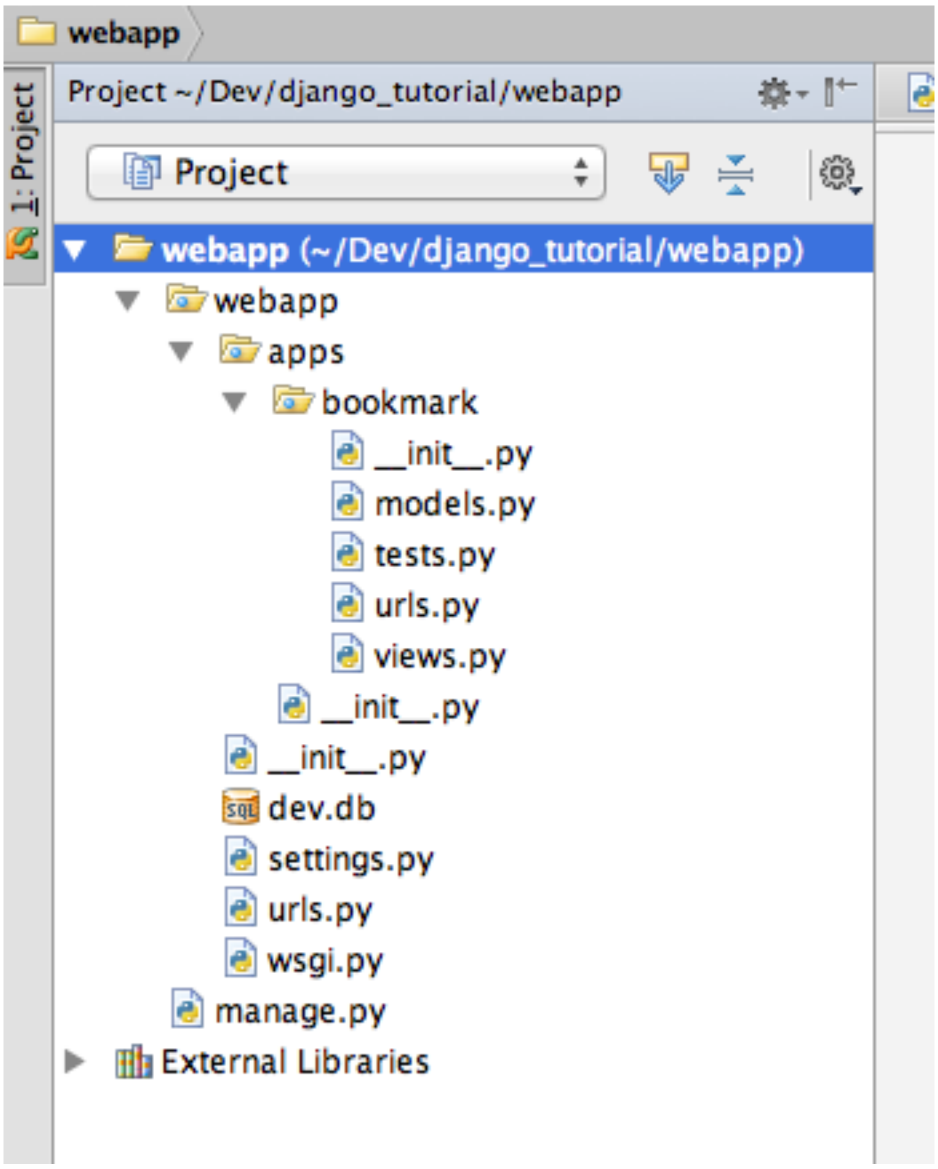
Now we add the following functions in the views.py:

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("index")
```

```
def create(request):
        return HttpResponse("create")

def update(request):
        return HttpResponse("update")

def delete(request):
        return HttpResponse("delete")

def show(request):
        return HttpResponse("show")
```

Define the urls

After defining the views.py, we need to define the urls.py. A urls.py is a url mapping to views method. We put the urls.py in the bookmark directory.



```
webapp (~/Dev/django_tutorial/webapp)
    webapp
        apps
            bookmark
                __init__.py
                models.py
                tests.py
                urls.py
                views.py
            __init__.py
        __init__.py
        dev.db
        settings.py
        urls.py
        wsgi.py
    manage.py
External Libraries
```

```
from django.conf.urls.defaults import patterns, url

urlpatterns = patterns('',
	url(r'^/index/$', 'bookmark.views.index', name='bookmark_index'),
	url(r'^/create/$', 'bookmark.views.create', name='bookmark_create'),
	url(r'^/update/$', 'bookmark.views.update', name='bookmark_update'),
	url(r'^/delete/$', 'bookmark.views.delete', name='bookmark_delete'),
	url(r'^/show/$', 'bookmark.views.show', name='bookmark_show'),
)
```

Then we need to connect our app url with project url mapping file. Find the urls.py in webapp/webapp directory, add the url mapping to bookmark.urls:

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

```
urlpatterns = patterns('',
    url(r'^bookmark', include('bookmark.urls')),
)
```

Now we run the app by using "python manage.py runserver". And input the url for "http://localhost:8000/bookmark/index", you will find the page show the text "index".

Why two URL configurations?
Loose coupling

Summary
In this part, we make a real app named "bookmark" and setup the Directory file structure. When users input the url, django will use the urls.py search corresponding view function to response. View function in views.py can get the user request and return response back to users.

Build the model

Now let's build our core data structure and business model for our app. A model is the single, definitive source of data about your data. It contains the essential fields and behaviors of the data you're storing. Generally, each model maps to a single database table.
Each model is a Python class that subclasses django.db.models.Model.
Each attribute of the model represents a database field.
With all of this, django gives you an automatically-generated database-access API;
For more information, please see also https://docs.djangoproject.com/en/dev/topics/db/models/.

```
from django.contrib.auth.models import User
from django.db import models

class Bookmark(models.Model):
    creator = models.ForeignKey(User,blank=True, null=True)
    url = models.URLField()
    title = models.CharField(blank=True, max_length=100, help_text='Limit of 100 characters')
    public = models.BooleanField(verbose_name='Public this bookmark', blank=True, default=True)
    create_time = models.DateTimeField(auto_now_add=True)
    last_modified_time = models.DateTimeField(auto_now=True)
    def __unicode__(self):
        return self.url
```

We have used different types of model fields. For more information, please check https://docs.djangoproject.com/en/dev/ref/models/fields/#field-types.

Notes:
DateField.auto_now

Automatically set the field to now every time the object is saved. Useful for "last-modified" timestamps. Note that the current date is always used; it's not just a default value that you can override.

DateField.auto_now_add

Automatically set the field to now when the object is first created. Useful for creation of timestamps. Note that the current date is always used; it's not just a default value that you can override.

We can delete the database file and syncdb the database again.

```
$python manage.py syncdb
```

django Admin

One of the most powerful parts of django is the automatic admin interface. It reads metadata in your model to provide a powerful and production-ready interface that content producers can immediately use to start adding content to the site. In this document, we discuss how to activate, use and customize django's admin interface.

1. Add django.contrib.admin into your installed apps list in settings.py

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # Uncomment the next line to enable the admin:
    'django.contrib.admin',
    # Uncomment the next line to enable admin documentation:
    # 'django.contrib.admindocs',
        'bookmark',
)
```

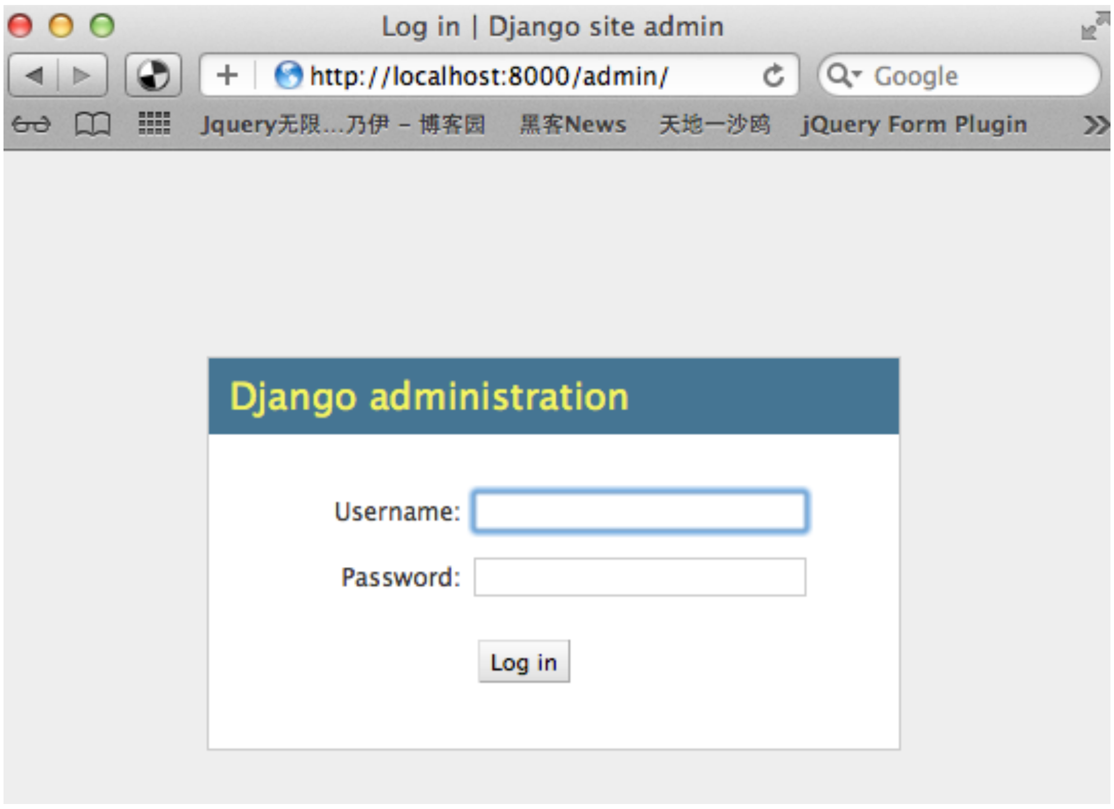
2. Add admin in urls.py

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

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
        url(r'^bookmark', include('bookmark.urls')),
        url(r'^admin/', include(admin.site.urls)),
)
```

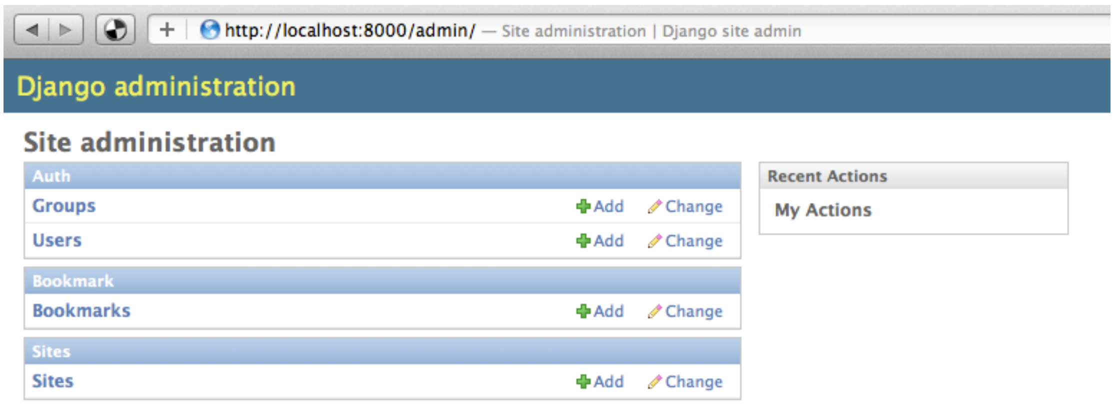
3. Add an admin.py (webapp/webapp/apps/bookmark/admin.py)

```
from django.contrib import admin
from models import Bookmark

class BookmarkAdmin(admin.ModelAdmin):
        pass

admin.site.register(Bookmark, BookmarkAdmin)
```

Now check the url http://localhost:8000/admin/ you will see



Input the admin user and password(remember we syncdb first time and let me input the user name and password?) After login, you can see django have build-in a power admin page let you CRUD data.



Define our own view functions

Define the index view

In the apps/bookmark/views.py, let's define a new method, which can show all bookmarks data from database to page.
The steps are:
1. Get a bookmarks list from database.
2. Render the bookmarks list to page (template)

That's all, now we code:

```
from django.http import HttpResponse
from django.shortcuts import render_to_response
from models import Bookmark

def index(request):
	bookmarks = Bookmark.objects.all()
	return render_to_response('index.html',
			{
			'bookmarks': bookmarks,
			})
```

bookmarks = Bookmark.objects.all() can get the all bookmarks list from database, there is no need to write any sql commands, but it actually work as you do in other language:
1. Create database connection
2. Write a sql and execute the sql then get a result set
3. Get the list from the result set

render_to_response('index.html', {...}) just tell us django will render the bookmarks list to the template "index.html", so we can use this "bookmarks" in the template page directly.

Let's look the "index.html" (in bookmarks/ templates directory)

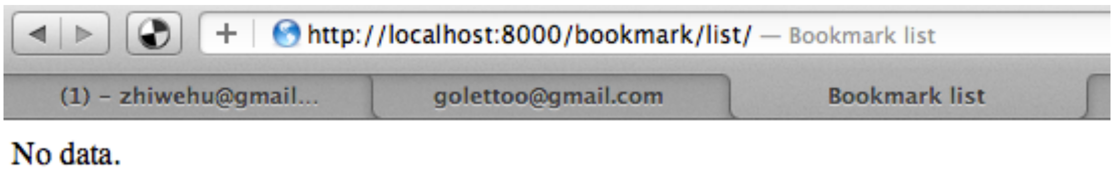
```
<html>
<head>
	<title>Bookmark list</title>
</head>
<body>
	{% for bookmark in bookmarks %}
{{ bookmark }}
{% empty %}
No data.
```

```
    {% endfor %}
</body>
</html>
```

In the “index.html” we can see the django template markup language:

- {% %} can execute some logic code, such as if, for,etc.
- {{   }} can display some variable to page.

In this example, we use {% for %} {% empty %} {% endfor %} markup language, you can get the meaning from the literal meaning. It’s means “loop the bookmarks list and print every bookmark, if the bookmarks is empty, just tell user no data.” Simple? Yes, but powerful!



No data.

Define the show view

In this section, we would like to define a show function to display the detail of a bookmark record.

Steps:

1. Get a bookmark which user want to show
2. Render the bookmark to page

Let’s look how to implement in code:

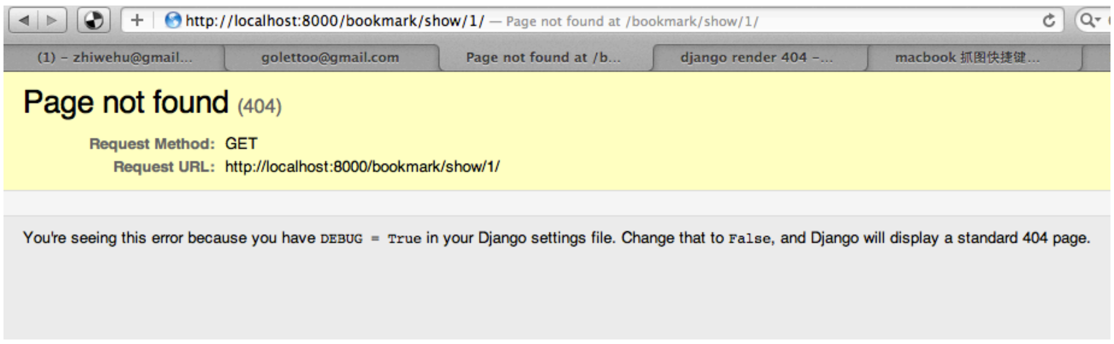
```
...
from django.http import HttpResponse, Http404
...
def show(request, bookmark_id):
    try:
        bookmark = Bookmark.objects.get(id=bookmark_id)
    except Bookmark.DoesNotExist:
        raise Http404
    return render_to_response('show.html',
                {
                'bookmark': bookmark,
                })
```

Please look the urls.py, and see how we define the “show” method url mapping:

```
url(r'^/show/(?P<bookmark_id>\d+)/$',                    'bookmark.views.show',
name='bookmark_show'),
```

You can see, we have defined a url which can get the bookmark_id from url input, for example, if the user input the url like this: http://localhost:8000/bookmark/show/1/, it tell the django urls.py to mapping this url to “bookmark.views.show” method, and pass the “bookmark_id” parameter to this method.

Using try…except… to protect if the user input a non-exists data, if the user input an id which not exists in db, it’ll raise a HTTP404 error.



Page not found (404)

Request Method: GET
Request URL: http://localhost:8000/bookmark/show/1/

You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

Define the delete view

Just like the “show” method, the delete method need a parameter so to know which data need to delete, and also, if the data does not exists, it’ll raise an HTTP404 error.

We can change the url of this method to:

```
    url(r'^/delete/(?P<bookmark_id>\d+)/$',           'bookmark.views.delete',
name='bookmark_delete'),
```

The view function code:

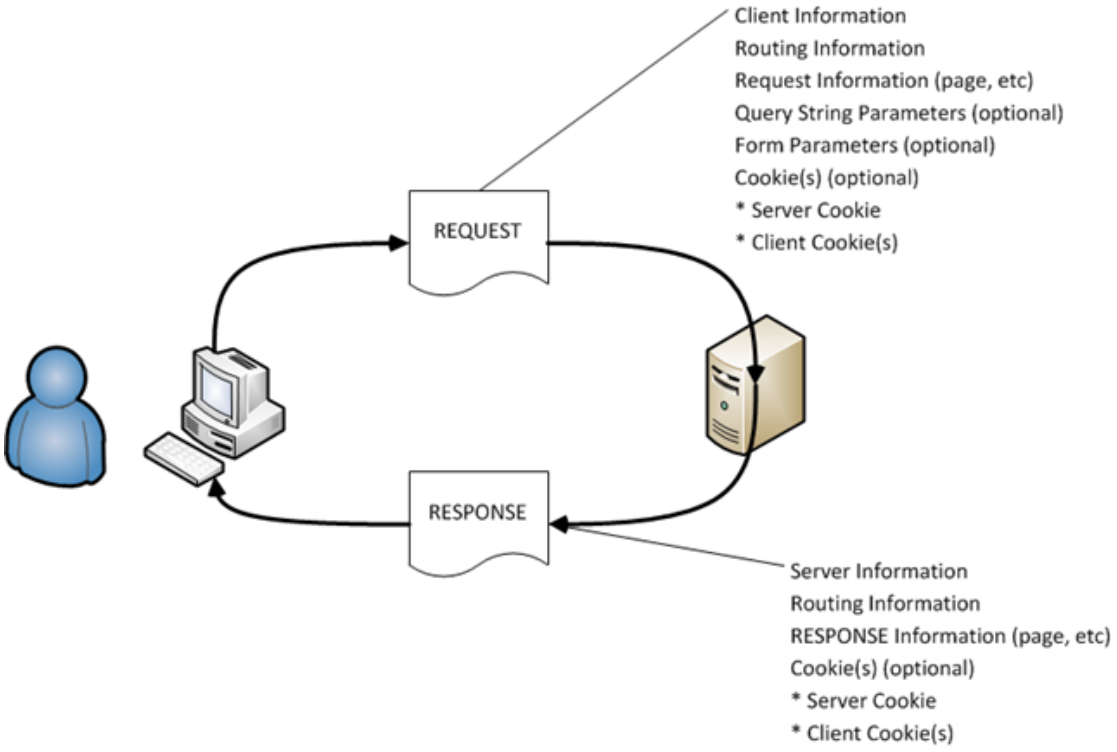
```
def delete(request, bookmark_id):
    try:
        bookmark = Bookmark.objects.get(id=bookmark_id)
        bookmark.delete()
    except Bookmark.DoesNotExist:
        raise Http404
return index(request)
```

bookmark.delete() method will delete this data in database. After delete, we just call return index(request) method, so we can go to the index page again. Another way to redirect is to use the following syntax:

```
Return HttpResponseRedirect(reverse('your_url_name',args=[arg1,]))
```

Models, Views and Templates

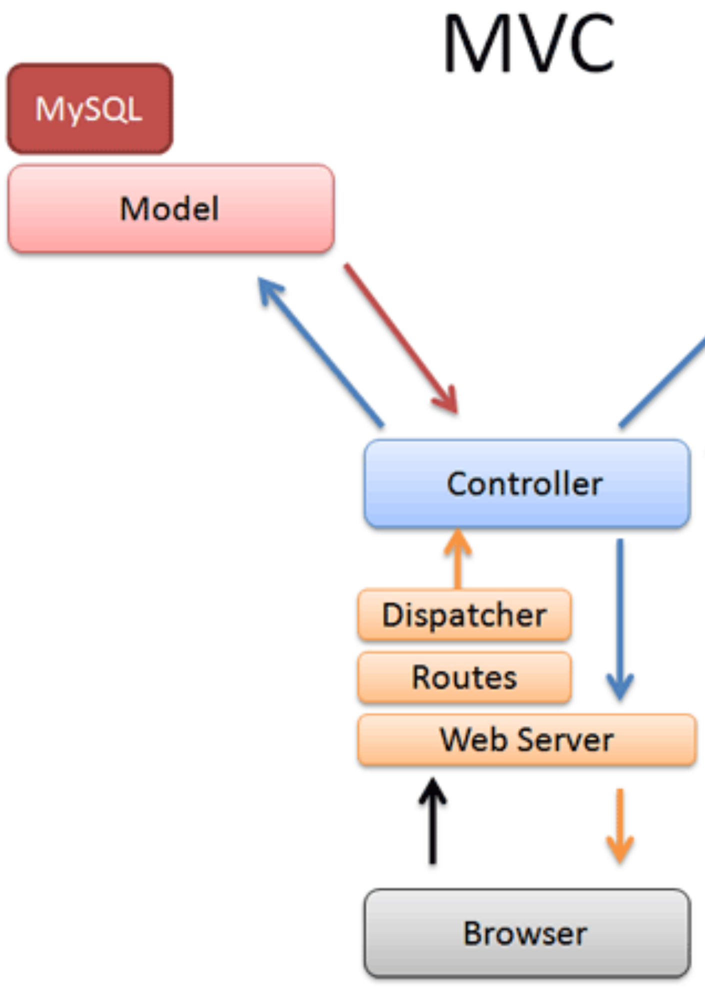
Web application is base on request-response mode.



The MVC design pattern:

Model–View–Controller (MVC) is a design pattern for software that divides an application into three areas of responsibility:

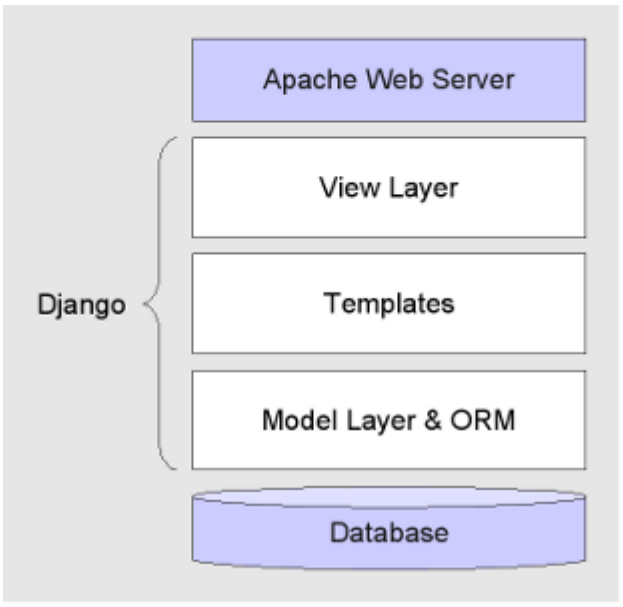
- Model: the domain objects or data structures that represent the application's state.
- View: which observes the state and generates output to the users.
- Controller, which translates user input into operations on the model.



In django,

- MVC=MVT
- Controller = django Views
- View = django templates

In django, the Python method called when a URL is requested is called a view, and the page loaded and rendered by the view is called a template. Because of this, the django team refers to django as an MVT (model-view-template) framework. TurboGears, on the other hand, calls its methods controllers and their rendered templates views so that they can fit squarely into the MVC acronym.



User input url address to request something

The urls.py will mapping the url to python method which defined in views.py.
views.py methods will filter users' request, get parameters from the request.GET or request.POST, then do some query from database, get the models from models.py.
views.py will use the templates to show the data to user, django will render the data (include models data) to template page(html).

Model

Data structure and mathematic model

View

Controller

Template

https://docs.djangoproject.com/en/dev/ref/templates/
django's template engine provides a powerful mini-language for defining the user-facing layer of your application, encouraging a clean separation of application and presentation logic. Templates can be maintained by anyone with an understanding of HTML; no knowledge of Python is required.

Inheritance

https://docs.djangoproject.com/en/dev/topics/templates/#template-inheritanc
e

Forms

Form is used to get input form the user. In django, Form framework is easy. We can ask django to generate form semi-automatically. We will use Form to create and modify data.

Define the “create” method

Let’s make a create method, first let user input a form, and submit the form, the method can handle the data which post in a form, then validate the data first
If it can pass validation, then django will save it to db.

First, define a forms.py in bookmark directory; we used to write all forms in this file:

```
from django import forms
from models import Bookmark

class BookmarkForm(forms.ModelForm):

    class Meta:
        model = Bookmark
        fields = ('creator','url','title','public')
```

The BookmarkForm is a subclass of django.forms.ModelForm, this from need can bind a model automatically. Simple? Powerful! But if you need customize the form, maybe you need to use the django.forms.Form, it’ll not bind the model, you should bind the data in the form to a model by yourself.

Model=Bookmark in “class Meta” tell us which fields in the Bookmark model need to show in the form.

For first time user input the “create” url, it’ll create an empty form and render it to the “create.html” template. Now we need to define the “create.html”, this html need render an empty form in the page.

```
<html>
<head>
```

```
    <title>Create a bookmark</title>
</head>
<body>
    <form action="." method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" />
        <input type="reset" />
    </form>
</body>
</html>
```

We defined a <form>, which action=".", it's mean the action is same with the current page, just like action="/bookmark/create/", the method is "post".

{% csrf_token %} is django build-in plugin which provide easy-to-use protection against "Cross site request forgeries". For the detail, please check here: https://docs.djangoproject.com/en/dev/ref/contrib/csrf/

{{ form.as_p }} is a quick way to display the form with <p></p>

Then, let's define the view function for "create":

```
from django.template.context import RequestContext
from forms import *
from django.http import *
from django.core.urlresolvers import reverse

def create(request):
    if request.method == 'POST': # If the form has been submitted...
        form = BookmarkForm(request.POST) # A form bound to the POST
data
        if form.is_valid(): # All validation rules pass
            bookmark = form.save()
            return HttpResponseRedirect(reverse('bookmark_index')) #
Redirect after POST
    else:
        form = BookmarkForm() # An unbound form

    return render_to_response('create.html',
        RequestContext(request, {
            'form': form,
            }))
```

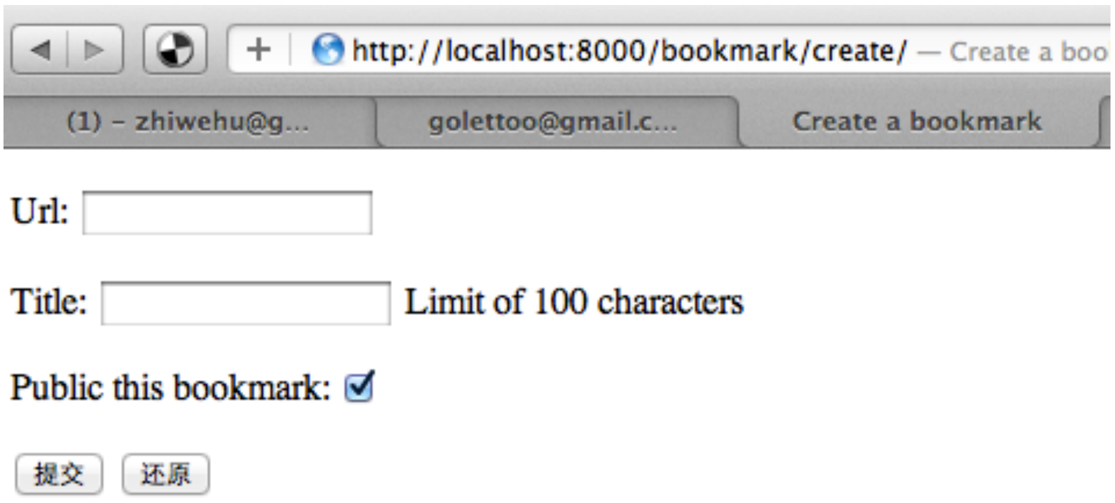
If a user submits the form with the "post" method, the method will check it and handle the data in the form.

form.is_valid() method will check if the data are all ok for this form. It can defined some valid method in the "BookmarkForm", we left it empty, but since the "BookmarkForm" is bind with the model "Bookmark", so for example, the Bookmark.url is not allow null or empty, so if the user didn't input the url field in the form, it'll get an error, we'll look later.

bookmark = form.save() can save the data just use the form.save() since the form is bound to the model, if the form doesn't bind the model, it needs you to write the method by self. For example:

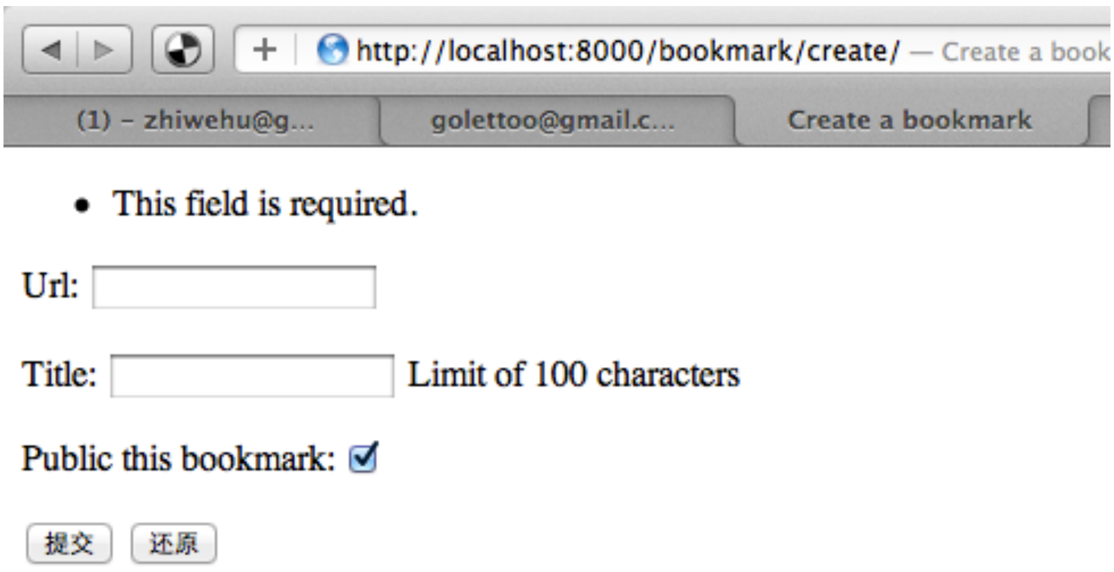
```
url=form.cleaned_data['url']
title=form.cleaned_data['title']
public= form.cleaned_data['public']
bookmark = Bookmark(url=url, title=title, public=public)
bookmark.save()
```

That's ok, let me check the result:



You can see the page display a form with labels, and input text, for the "public" field of Bookmark model, since this field is a Boolean type, it can use a checkbox to display it automatically! You just use the {{ form.as_p }}, and the django form template tag "as_p" will do it. Easy?

Then, let us chick the "input" button without input any data, and you will see:


http://localhost:8000/bookmark/create/ — Create a book

(1) – zhiwehu@g… | golettoo@gmail.c… | Create a bookmark

- This field is required.

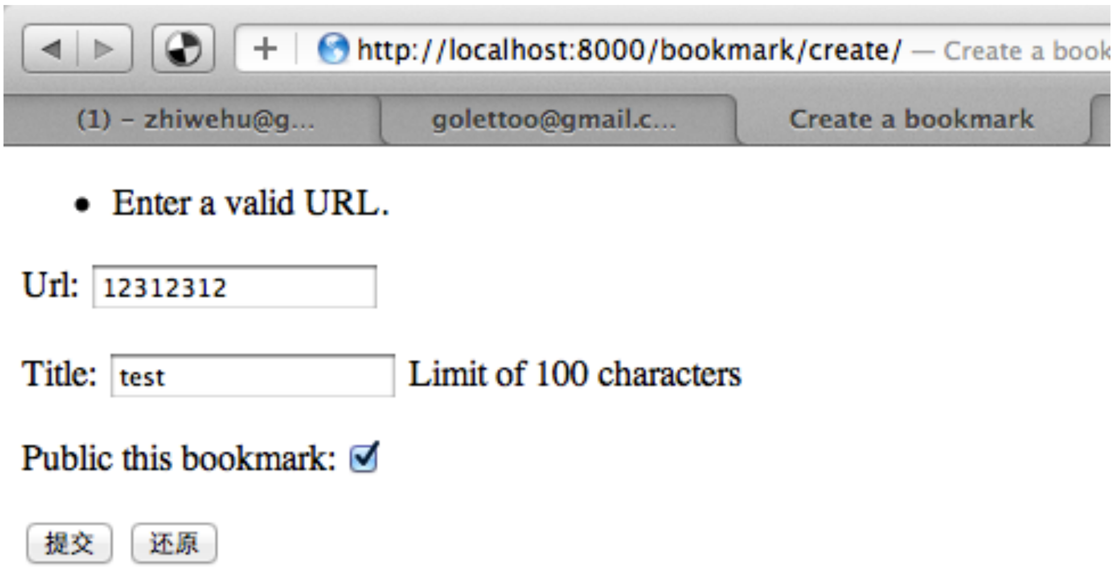
Url:

Title: Limit of 100 characters

Public this bookmark: ☑

Yeah! django checked the url field is not allow empty, so, if you forget to input this field, django will valid the data, and return errors.

Now, we input something, for the url input, we just input some text but not a valid url, and you can see:


http://localhost:8000/bookmark/create/ — Create a book

(1) – zhiwehu@g… | golettoo@gmail.c… | Create a bookmark

- Enter a valid URL.

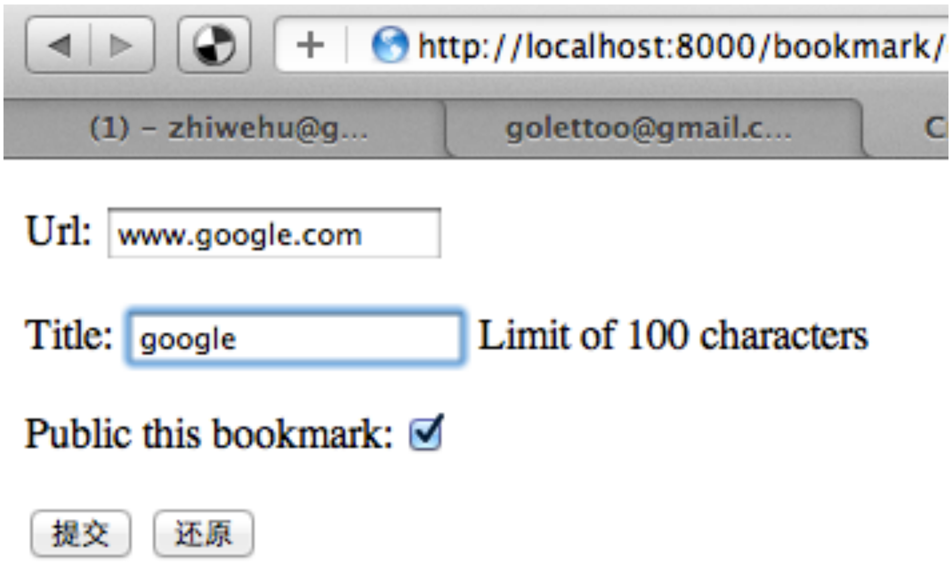
Url: 12312312

Title: test Limit of 100 characters

Public this bookmark: ☑

It cannot pass the form.is_valid() method again, and this time, the error is "Enter a valid URL", cool? Without write one line of code to validate the form data, but django help you do it!

At the end, let us input some valid data.


http://localhost:8000/bookmark/

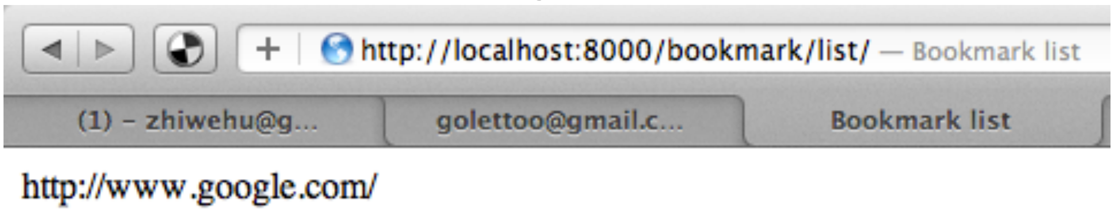
(1) – zhiwehu@g... golettoo@gmail.c... C

Url: www.google.com

Title: google Limit of 100 characters

Public this bookmark: ✓

After submit, it'll redirect to "index" page


http://localhost:8000/bookmark/list/ — Bookmark list

(1) – zhiwehu@g... golettoo@gmail.c... Bookmark list

http://www.google.com/

Tips:
We used HttpResponseRedirect(reverse('bookmark_index')) to redirect to another page. Reverse method can get the url by name, please check the urls.py, we defined the url like:

```
url(r'^/index/$', 'bookmark.views.index, name='bookmark_index'),
```

Define a url name can make django reuse it by name, even you changed the url.

Define "update" method

Ok, now the last method is how we update an existing bookmark?

Let's define the template for "update" (update.html):

```
<html>
<head>
	<title>Update a bookmark</title>
</head>
<body>
```

```
<form action="." method="post">
	{% csrf_token %}
	{{ form.as_p }}
	<input type="submit" />
	<input type="reset" />
</form>
</body>
</html>
```

Steps to update an existing record:

1. Get the bookmark by bookmark_id parameter
2. Render the bookmark form with the bookmark data
3. User can update the bookmark data in the form and then submit
4. If the data in form is valid, it'll update the bookmark and save it to db

Let's define the url for update method:

```
url(r'^/update/(?P<bookmark_id>\d+)/$',                    'bookmark.views.update',
name='bookmark_update'),
```

Let's see how the views.py looks:

```
def update(request, bookmark_id):
	try:
		bookmark = Bookmark.objects.get(id=bookmark_id)
	except Bookmark.DoesNotExist:
		raise Http404

	if request.method == 'POST': # If the form has been submitted...
		form = BookmarkForm(request.POST, instance=bookmark)
		if form.is_valid(): # All validation rules pass
			bookmark = form.save()
			return      HttpResponseRedirect(reverse('bookmark_index'))        #
Redirect after POST
	else:
		form = BookmarkForm(instance=bookmark)

	return render_to_response('update.html',
		RequestContext(request, {
			'form': form,
			}))
```

Unify “Create” and “Update”

Now you can see, the code of “update” is very similar with the “create” method, even the html template. A django philosophy is “do not repeat yourself”, so we’ll consider use a common method can handle “create” and “update”.

```
def foo(request, obj_id):
    form = ObjectForm(request.POST or None, request.FILES or None
        , instance= obj_id and SomeObject.objects.get(id=obj_id))

    if request.method == 'POST' and form.is_valid():
        form.save()
        return HttpResponseRedirect(“/”)

    variables = RequestContext(request, {'form': form})
    return render_to_response('create_update_template.html',variables)
```

Static and media files

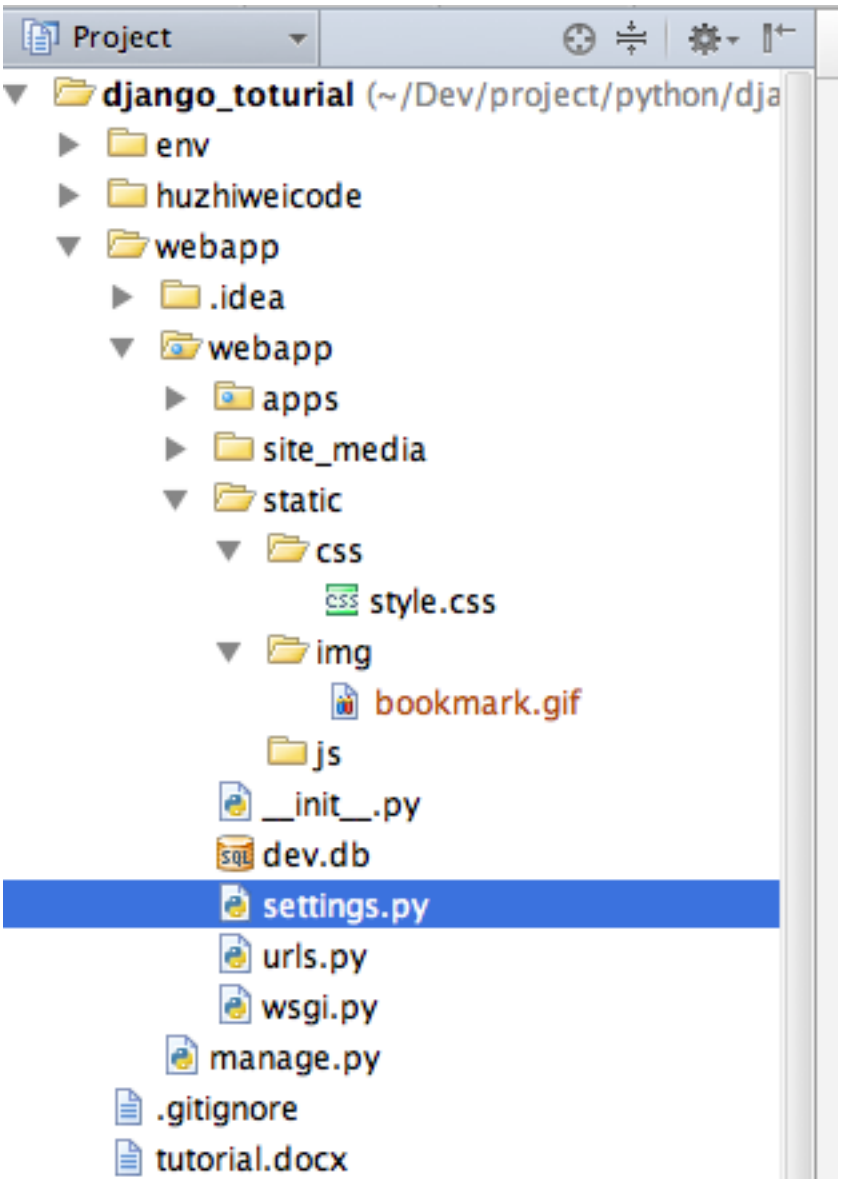
Image and other static files

To configure the static files, such as javascripts, images and css, we need to define some configuration in settings.py

```
# Absolute path to the directory static files should be collected to.
# Don't put anything in this directory yourself; store your static files
# in apps' "static/" subdirectories and in STATICFILES_DIRS.
# Example: "/home/media/media.lawrence.com/static/"
STATIC_ROOT = os.path.join(PROJECT_ROOT, 'site_media', 'static')

# URL prefix for static files.
# Example: "http://media.lawrence.com/static/"
STATIC_URL = '/site_media/static/'

# Additional locations of static files
STATICFILES_DIRS = (
    os.path.join(PROJECT_ROOT, 'static'),
)

# List of finder classes that know how to find static files in
# various locations.
```

```
STATICFILES_FINDERS = (
        'django.contrib.staticfiles.finders.FileSystemFinder',
        'django.contrib.staticfiles.finders.AppDirectoriesFinder',
)
```

Now we need to create the “static”directory in webapp directory.



You can see in static directory we defined 3 sub-directoies:

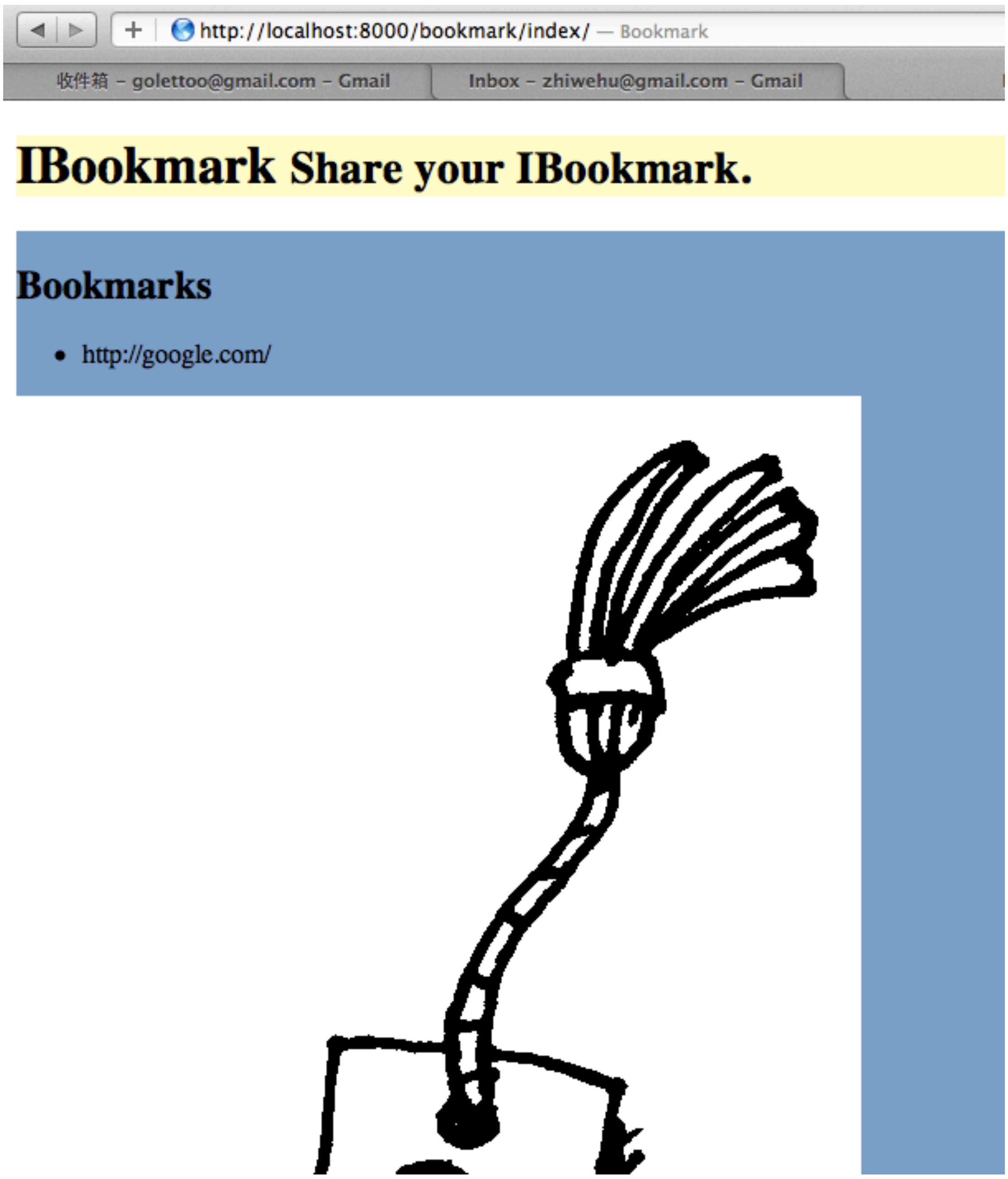
- img: store the images for the webapp
- css: store the css files
- js: store the javascript files

And we need to use these static files in our template html files (index.html). For example:

```
{% extends "base.html" %}
{% load static %}
{% block main_content %}
        <h1>Bookmarks</h1>
        <ul>
        {% for bookmark in bookmarks %}
                <li>{{ bookmark }}</li>
        {% endfor %}
        </ul>
        <img src="{% static 'img/bookmark.gif' %}">
{% endblock %}
```

You can see we should add {% load static %} tag to let template know you want to use static, and use {% static 'img/bookmark.gif' %} to load static files.

And let's see the result in browser.



http://localhost:8000/bookmark/index/ — Bookmark
收件箱 – golettoo@gmail.com – Gmail
Inbox – zhiwehu@gmail.com – Gmail

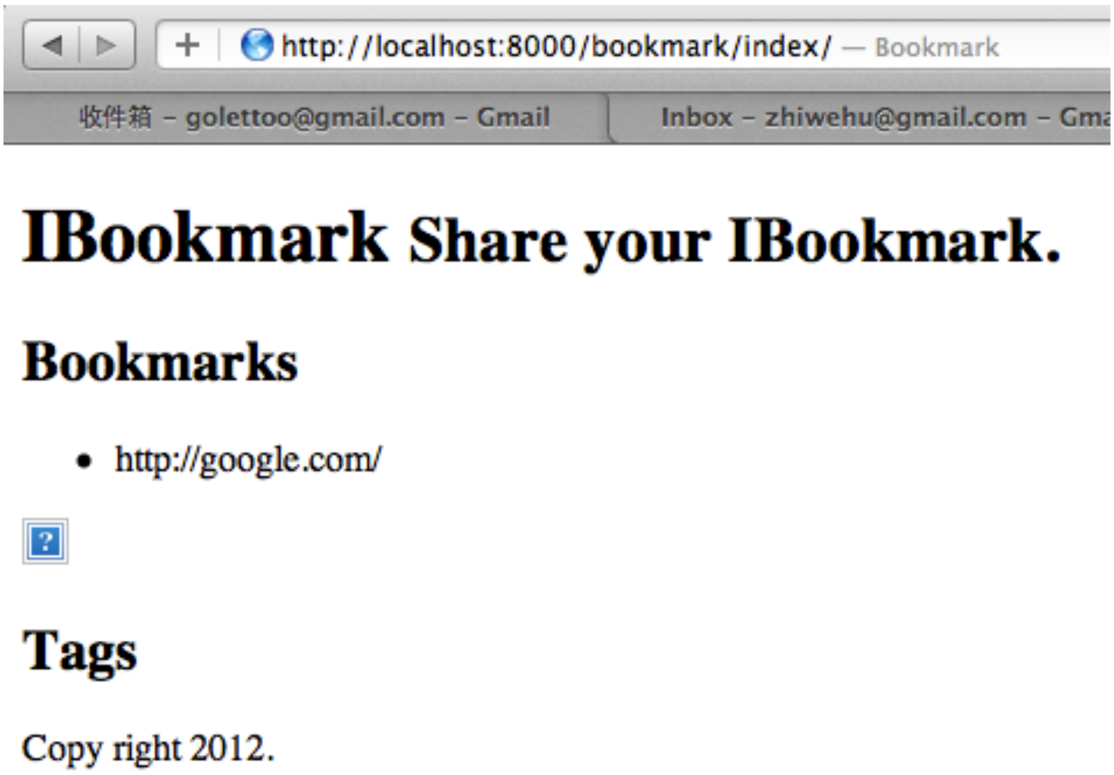
IBookmark Share your IBookmark.

Bookmarks

- http://google.com/

It works!

But please notice we just using “debug=True” mode in our webapp development now, if we set the “debug=False” in production mode, and let we check the result:


http://localhost:8000/bookmark/index/ — Bookmark
收件箱 – golettoo@gmail.com – Gmail
Inbox – zhiwehu@gmail.com – Gm

IBookmark Share your IBookmark.

Bookmarks

- http://google.com/

?

Tags

Copy right 2012.

Why?
Since the django will automatically call a view to serve static files in dev mode(debug=True).

So if we set the mode to production, it cannot work. In production, we’ll use such as httpd web server to handle the static files request. We need to use django admin tool to collect our static files to /site-media/static/

```
$ source env/bin/activate
(env)$ cd webapp/
(env)$ python manage.py collectstatic

You have requested to collect static files at the destination
location as specified in your settings.

This will overwrite existing files!
Are you sure you want to do this?

Type 'yes' to continue, or 'no' to cancel: yes
Copying
```

```
'/Users/jeffrey/Dev/project/python/django_toturial/webapp/webapp/static/cs
s/style.css'
……
Copying
'/Users/jeffrey/Dev/project/python/django_toturial/env/lib/python2.7/site-pa
ckages/django/contrib/admin/static/admin/js/admin/RelatedObjectLookups.js
'

74 static files copied.
```

CSS file

Let’s write a css file.

```
header {
	background-color: #fffccf;
}

section {
	background-color: #7CA0C7;
	width: 80%;
	float: left;
}

aside {
	background-color: #c9dbed;
	width: 20%;
	float: left;
}

footer {
	background-color: #f6f6f6;
	clear: both;
}
```

And load css in html files.

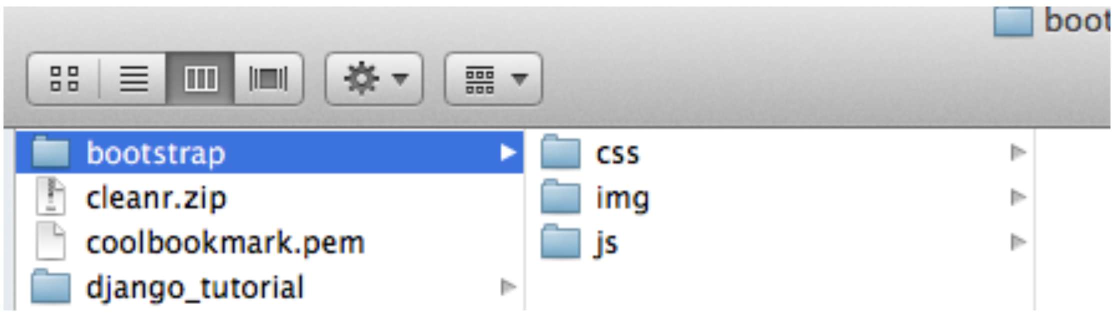
```
…
	<link href="{% static 'css/style.css' %}" rel="stylesheet">
…
```

Bootstrap template

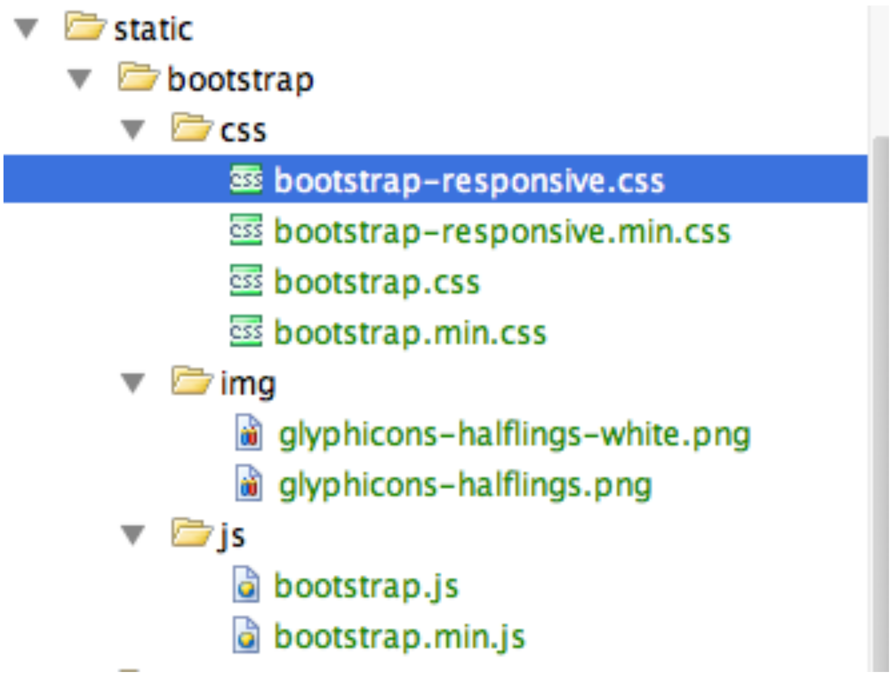
Twitter boostrap is a frontend framework which helps user build web app.
Simple and flexible HTML, CSS, and Javascript for popular user interface components and interactions.

In this chapter, I'll introduce how to integrate with twitter bootstrap in our webapp.

1. First, we need to download the bootstrap lib from http://twitter.github.com/bootstrap/ ,now the version is 2.0.4
2. Open the bootstrap.zip, we can found the architecture is same with our static directory.



3. Copy the bootstrap directory to our static directory.



4. Create a base.html using bootstrap.

```
<!DOCTYPE HTML>
{% load static %}
{% load i18n %}
```

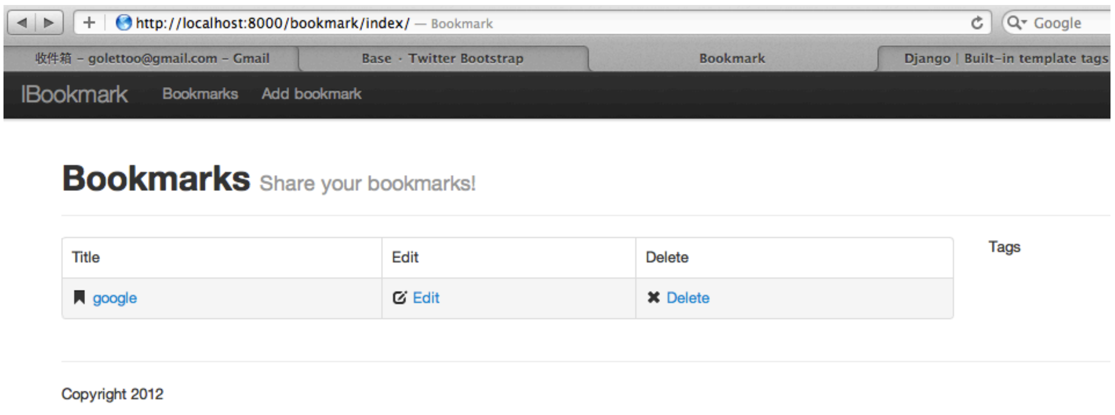
```
<html>
<head>
	<link href="{% static 'bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">
	<link href="{% static 'bootstrap/css/bootstrap-responsive.min.css' %}" rel="stylesheet">
	<title>Bookmark</title>
</head>
<body>
<header>
	<div class="navbar">
		<div class="navbar-inner">
			<div class="container">
				<a class="brand" href="{% url bookmark_index %}">{% trans "IBookmark" %}</a>
				<ul class="nav">
					<li><a href="{% url bookmark_index %}">{% trans "Bookmarks" %}</a></li>
					<li><a href="{% url bookmark_create %}">{% trans "Add bookmark" %}</a></li>
				</ul>
			</div>
		</div>
	</div>
</header>

<div class="container">
	<section>
		<div class="page-header">
			<h1>Bookmarks
				<small>Share your bookmarks!</small>
			</h1>
		</div>
		<div class="row">
			<div class="span8">{% block main_content %}{% endblock %}</div>
			<div class="span4">Tags</div>
		</div>
	</section>

	<hr />

	<footer class="footer">
		<p>Copyright 2012</p>
```

```
        </footer>
</div>

<!-- Le javascript
======================================================= -->
<!-- Placed at the end of the document so the pages load faster -->
<script src="{% static 'bootstrap/js/bootstrap.min.js' %}"></script>
</body>
</html>
```

5. And update our index.html

```
{% extends "base.html" %}
{% load static %}
{% load i18n %}

{% block main_content %}
<table class="table table-bordered table-striped">
        <thead>
        <td>{% trans "Title" %}</td>
        <td>{% trans "Edit" %}</td>
        <td>{% trans "Delete" %}</td>
        </thead>
        <tbody>
        {% for bookmark in bookmarks %}
        <tr>
                <td><i class="icon-bookmark"></i> <a
href="{{ bookmark.url }}">{{ bookmark.title }}</a></td>
                <td><i class="icon-edit"></i> <a href="{% url bookmark_update
bookmark.id %}">{% trans "Edit" %}</a></td>
                <td><i class="icon-remove"></i> <a href="{% url bookmark_delete
bookmark.id %}">{% trans "Delete" %}</a></td>
        </tr>
        {% empty %}
        <tr>
                <td colspan="3">{% trans "No data" %}</td>
        </tr>
        {% endfor %}
        </tbody>
</table>

{% endblock %}
```

And now let’s see the result.


http://localhost:8000/bookmark/index/ — Bookmark
收件箱 – golettoo@gmail.com – Gmail
Base · Twitter Bootstrap
Bookmark
Django | Built-in template tags
IBookmark
Bookmarks
Add bookmark

Bookmarks Share your bookmarks!

Title	Edit	Delete
google	Edit	✖ Delete

Tags

Copyright 2012

Cool now?
And update others html templates also to extend the base.html

User authentication

User registration and account management are universal features found in every web application. Users need to identify themselves to the application before they can post and share content with other users. User accounts are also required for activities such as online discussions and friend networks.

The django authentication system is available in the django.contrib.auth package. It is installed by default as part of django and projects created with the django-admin.py utility have it enabled by default.

https://docs.djangoproject.com/en/dev/topics/auth/

Creating the login page

```
from django.contrib.auth import authenticate, login

def my_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    if user is not None:
        if user.is_active:
            login(request, user)
            # Redirect to a success page.
        else:
```

```
                    # Return a 'disabled account' error message
          else:
                # Return an 'invalid login' error message.
```

Enabling logout functionality

```
from django.contrib.auth import logout

def logout_view(request):
          logout(request)
          # Redirect to a success page.
```

User registration

The same procedure with “Create” method of bookmark.

3rd party apps (django-allauth)

https://github.com/pennersr

EC2 (apache, MySQL)

In this chapter, we’ll deploy our webapp on amazon ec2 host, with apache httpd web server, and use the mysql database.

```
-- Login ec2
$ ssh -i your_ec2.pem ec2-user@your_ec2_ip_or_hostname

-- Update ec2
$ sudo yum update -y
$ sudo yum install gcc python-devel -y

-- Install git
$ sudo yum install git -y

-- Check out code from git server
$ cd /home/ec2-user
$ git clone git://github.com/zhiweihu/django_toturial.git

-- Install apache httpd and mod_wsgi, mod_ssl
$ sudo yum install httpd mod_wsgi mod_ssl -y

-- Config httpd server
$ sudo vi /etc/httpd/conf.d/wsgi.conf
[Add the below content to the end of the file]
#----------Begin----------#
LoadModule wsgi_module modules/mod_wsgi.so
WSGISocketPrefix run/wsgi
WSGIDaemonProcess django_toturial
python-path=/home/ec2-user/django_toturial
/env/lib64/python2.6/site-packages
WSGIProcessGroup django_toturial
#-----------End-----------#

$ sudo vi /etc/httpd/conf/httpd.conf
[Change "User apache" to "User ec2-user"]
[Add the below content to the end of the file]
#----------Begin----------#
WSGIScriptAlias / /home/ec2-user/django_toturial/webapp/webapp/wsgi.py
Alias /site_media/ /home/ec2-user/django_toturial/webapp/webapp
/site_media/
#-----------End-----------#

-- Install mysql and create user/password
$ sudo yum install mysql mysql-server mysql-devel -y
$ sudo service mysqld start
$ sudo mysqladmin -u root password Your_password
$ mysql -u root -p
[Input the password, then we use the mysql console]
mysql> CREATE DATABASE tutorial_db;
Query OK, 1 row affected (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON tutorial_db.* TO "youruser"@"localhost"
IDENTIFIED BY "youruser_password";
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> EXIT
Bye
```

```
-- Install virtualenv for python
$ sudo easy_install virtualenv

-- Use virtualenv to install django
$ cd /home/ec2-user/django_tutorial
$ virtualenv env
$ source env/bin/activate
[Now we use the virtualenv to do something, do not print the (env)$ on your console]
(env)$ pip install -r webapp/requirement.txt

-- Settings for production
[Edit "settings_local.py" to save the db settings for production]
DATABASES = {
        'default': {
                'ENGINE': 'django.db.backends.mysql',
                'NAME': 'tutorial_db',
                'USER': 'youruser',
                'PASSWORD': 'youruser_password',
                'HOST': '',
                'PORT': '',
        }
}

-- Sync db and collect static files
(env)$ python manage.py syncdb
(env)$ python manage.py collectstatic
s
-- Restart the apache http server
$ sudo /etc/init.d/httpd restart
```

Todo

第二次迭代：

1. 简单介绍如何用命令行和控制台调试 model，包括如何 get objects 等 queryset 的用法等等
2. 增加 South 介绍