

Django Web Framework

Zhaojie Zhang

CSCI5828 Class Presentation

03/20/2012

Outline

- Web frameworks
- Why python? Why Django?
- Introduction to Django
- An example of Django project
- Summary of benefits and features of Django
- Comparison of Django and Ruby on Rails
- Additional tools to facilitate Django Development
- Applications made with Django
- To learn more about Django
- References

Web Frameworks

Languages

- Php
- Python
- Java
- Ruby
- Perl
- Javascript
- ...

Web frameworks

- Zend, Symfony, Phpdevshell...
- Django, web2py, CherryPy, ...
- Struts, Spring, Tapestry, GWT, ...
- Ruby on rails,...
- Catalyst, Mason, ...
- JavaScriptMVC, ...
- ...

Why Python?

- High-level language
- Concise syntax and easy to learn
- Large and growing developer community
- Portable on almost all platforms

Why Django?

- Python programming language
- Open-source project
- Large and growing community
- Well-documented
- Large collections of apps
- Good debugging feedbacks

History of the Django Project

- Django started as an internal project at the Lawrence Journal-world newspaper in 2003 and was created to meet the fast deadlines of journalism websites.
- Django was released to the public by the developers in 2005.
- The project was named after the jazz Guitarist Django Reinhardt.
- Now an open source web framework for building maintainable and reusable web applications.

Introduction to Django

- A high-level python web framework adhering to the DRY principle: Don't repeat yourself.
- MVC design patterns: code modules are divided into logic groups.
- Automatic Admin Interface.
- Elegant URL design.
- Powerful, extensible and designer-friendly template system.
- Cache system available for super performance.

Web Development without Web Frameworks

- A python CGI script example:

```
#!/usr/bin/python

import MySQLdb

print "Content-Type: text/html"
print
print "<html><head><title>Books</title></head>"
print "<body>"
print "<h1>Books</h1>"
print "<ul>"

connection = MySQLdb.connect(user='me', passwd='letmein', db='my_db')
cursor = connection.cursor()
cursor.execute("SELECT name FROM books ORDER BY pub_date DESC LIMIT 10")
for row in cursor.fetchall():
    print "<li>%s</li>" % row[0]

print "</ul>"
print "</body></html>"

connection.close()
```


The Django Counterpart with the MVC Design Pattern

```
# models.py (the database tables)                                # urls.py (the URL configuration)

from django.db import models                                    from django.conf.urls.defaults import *
                                                                import views

class Book(models.Model):                                       urlpatterns = patterns('',
    name = models.CharField(maxlength=50)                       (r'latest/$', views.latest_books),
    pub_date = models.DateField()                               )

# views.py (the business logic)

from django.shortcuts import render_to_response
from models import Book

def latest_books(request):
    book_list = Book.objects.order_by('-pub_date')[:10]
    return render_to_response('latest_books.html', {'book_list': book_list})

# latest_books.html (the template)

<html><head><title>Books</title></head>
<body>
<h1>Books</h1>
<ul>
{% for book in book_list %}
<li>{{ book.name }}</li>
{% endfor %}
</ul>
</body></html>
```

Steps for Using Django

- Install Django (python assumed to be installed)
- Create a project
- Start an application
- Create the database
- Define the models
- Write the templates
- Define the views
- Create URL mappings
- Test and deploy the application

Install and Start Django

- Download the tarball, which will be named something like Django-*.tar.gz.
- `tar xzvf Django-*.tar.gz.`
- `cd Django-*`.
- `sudo python setup.py install.`
- Start python and you should be able to import Django:

```
>>> import django
```

Start a Project

- This example project is done on a Mac pro notebook for the “polls” app from the Django project website.
- Create a new directory to start working in:
`mkdir /Users/zhaojie/DesktopDjango_project/`
- Run the following command to create a mysite directory in your current directory:
`django-admin.py startproject mysite`

Start a Project

Inside the “mysite” folder, four files will be generated:

- `__init__.py`: A file required for Python treat the directory as a package (i.e., a group of modules)
- `manage.py`: A command-line utility that lets you interact with this Django project in various ways
- `settings.py`: Settings/configuration for this Django project
- `urls.py`: The URL declarations for this Django project; a “table of contents” of your Django-powered site

Start a Project

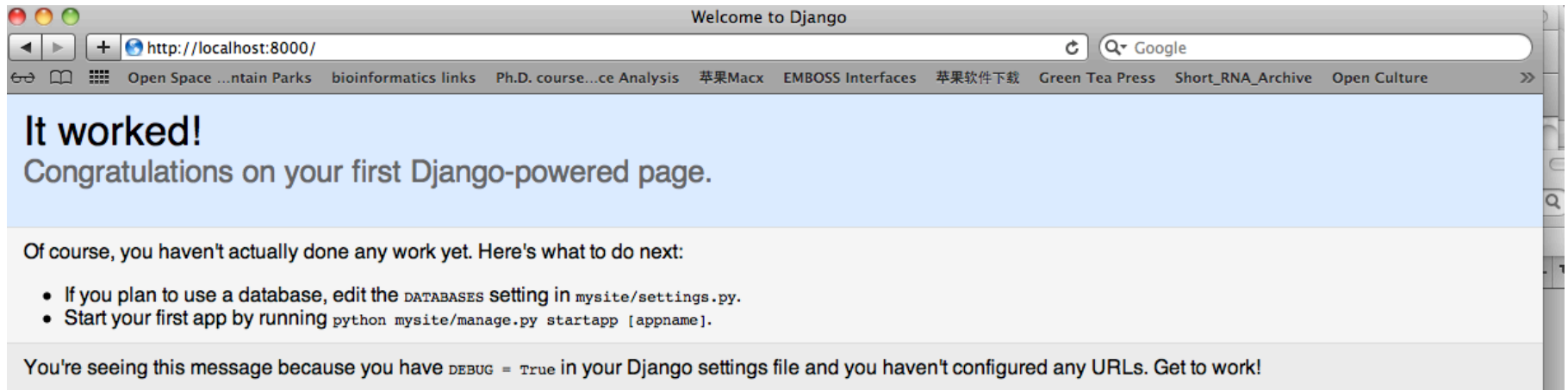
- Change into the “mysite” directory, and run the following command to start the built-in, lightweight development server:

```
python manage.py runserver
```

- You would see something like these in the terminal:
Validating models...
0 errors found
Django version 1.3.1, using settings 'mysite.settings'
Development server is running at http://localhost:8000/
Quit the server with CONTROL-C.

Start a Project

- If you go to <http://localhost:8000> in the browser, you should see the following webpage, suggesting the development web server is working:



Supported Databases

Django supports the following three databases:

- PostgreSQL (<http://www.postgresql.org/>)
 - SQLite 3 (<http://www.sqlite.org/>)
 - MySQL (<http://www.mysql.com/>)
-
- SQLite3 is the database used in this presentation.

Database Setup

- In “settings.py”, change the default to the following:

```
DATABASE_ENGINE = 'sqlite3'.
```

```
DATABASE_NAME = '/User/zhaojie/Desktop/Django_project/  
mysite/mydata.db
```

- “mydata.db” does not exist at this point, which will be created later. SQLite databases are just plain files and that is why the absolute path needs to be specified.
- Also in “settings.py”, add “polls” to the “INSTALLED_APPS” (the polls app will be created later).

Start an App

- Run the following command to create the polls app:

`Python manage.py startapp polls`

- A “polls” directory will be created in the mysite directory with the following files inside:

polls/ `__init__.py`

`models.py`

`tests.py`

`views.py`

Database Initialization

- Run “`python manage.py sql polls`”
-
- Then “`python manage.py syncdb`”
- These commands will create a database schema (CREATE TABLE statements) for the “polls” app and also create a Python database-access API for accessing Poll and Choice objects.

Define the Models

- Define the models in the “models.py” file under the “polls” directory:

```
from django.db import models
import datetime

class Poll(models.Model):
    question = models.CharField(max_length=200)
    pub_date = models.DateTimeField('date published')
    def was_published_today(self):
        return self.pub_date.date() == datetime.date.today()
    def __unicode__(self):
        return self.question
    def was_published_today(self):
        return self.pub_date.date() == datetime.date.today()

class Choice(models.Model):
    poll = models.ForeignKey(Poll)
    choice = models.CharField(max_length=200)
    votes = models.IntegerField()
    def __unicode__(self):
        return self.choice
```

URL Mapping

- In the “urls.py” file, make the following changes:

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

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

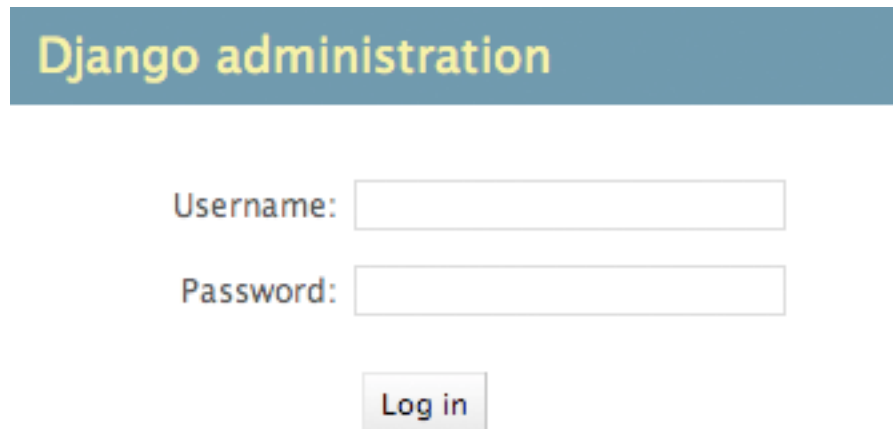
urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^mysite/', include('mysite.foo.urls')),

    # Uncomment the admin/doc line below to enable admin documentation:
    # url(r'^admin/doc/', include('django.contrib.admindocs.urls')),

    # Uncomment the next line to enable the admin:
    url(r'^admin/', include(admin.site.urls)),
)
```

Start the Server

- Use the “python manage.py runserver” command to start the server, and you should see the following admin interface:



The image shows a screenshot of the Django administration login page. At the top, there is a blue header bar with the text "Django administration" in yellow. Below the header, there are two input fields: "Username:" followed by a text box, and "Password:" followed by a text box. Below the password field is a "Log in" button.

Django Summary

- From the example above, we can see a few characteristics of Django:
 - 1) it provides a very user-friendly interface for dealing with databases;
 - 2) the template system enables the developers to focus on the app building instead of on the details of HTML;
 - 3) the built-in development server and admin interface helps the developers to test the applications easily.

Django Summary

- Besides the several characteristics mentioned in last slide, Django has more to offer which were not exemplified in the previous simple example. Here are a few of them:

4) Generating non-HTML content

5) Caching

6) Middleware

7) Internationalization

8) Security

9) Session framework

Django Summary

- 4) Generating non-HTML content (Built-in tools for producing non-HTML content):
 - RSS/Atom syndication feeds
 - Sitemaps

Django Summary

- 5) Caching:
- Django comes with a robust cache system that lets you save dynamic pages so that they do not need to be calculated each time a request is made.

Django Summary

- 6) Middleware:
- A middleware component is a python class that conforms to a certain API.
- All of the session and user tools are made possible by a few small pieces of middleware.

Django Summary

- 7) Internationalization:
- A minimal number of “translation strings” are needed in the python codes for translation.
- It allows the web applications to be translated for users with their own language preferences.

Django Summary

- 8) Security:
- Django is designed to automatically protect you from many of the common security mistakes that web developers make.

Django Summary

- Session framework:
- The session framework lets you store and retrieve arbitrary data on a per-site-visitor basis. It stores data on the server side and abstracts the sending and receiving of cookies.

Comparisons with Ruby on Rails

- A comparison made between Django and another popular web framework:

Factor	Rails	Django
Support for model and schema evolution	Integrated framework for schema evolution.	Minimal.
Internationalisation	Some support in Rails 1.2.	Some support.
Designer Friendly Templates?	Possible, with use of third-party library.	Yes.
Third Party Plugin Support	Mature plugin architecture, well used by the community.	Some support via the applications mechanism.
Javascript Support	Prototype and Scriptaculous bundled with Rails. RJS framework simplifies their use.	Possible, but no direct support for any particular library.
Flavour	Concise.	Explicit.

Additional Tools to Facilitate Django Development

- Django is a thriving framework, and lots of people are contributing their own modules/tools to facilitate Django development and make Django more powerful.
- According to the DjangoPackages website alone, to this date, it records at least 944 Apps, 51 Frameworks and 44 individual Projects written for the Django Web framework.

Additional Tools to Make Django More Powerful

Here are just a few I found interesting and useful:

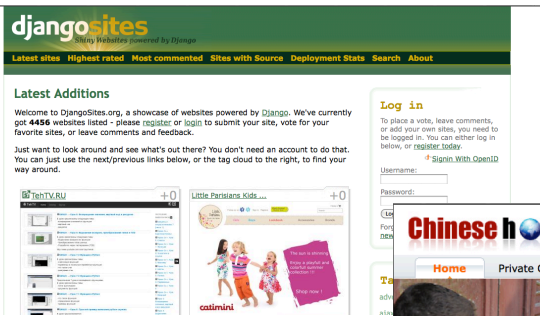
- **GeoDjango:** a customized GIS web framework
- **Django Debug Toolbar:** debugging tool
- **Django Easy Maps:** map displaying tool
- **Django Haystack:** modular search tool

Additional Tools to Facilitate Django Development

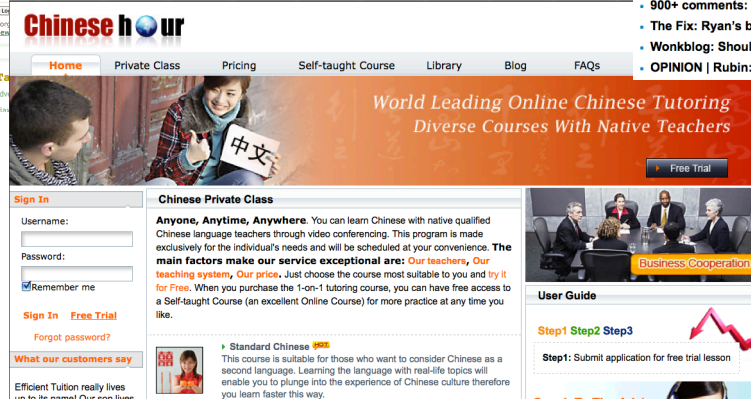
- **django-revision:** An extension to the Django web framework that provides comprehensive version control facilities.
- **South:** provides a simple, stable and database-independent migration layer to your Django applications.
- **Fabulous:** deploy django apps to Amazon EC2 with ONE command.

Websites Using Django

- An incomplete list of websites using Django is registered in the DjangoSites website. Here are a few examples:



DjangoSites website itself!



Wow, spreading across the world!



WashingtonPost! I knew it is a famous newspaper, and its website is made with Django!

To Learn More About Django

- There are lots of Django tutorials and examples on the internet. Here are a few:
- <https://docs.djangoproject.com/en/dev/intro/whatsnext/> (Official Django Documents)
- <https://code.djangoproject.com/wiki/Tutorials> (Django Tutorial)
- <http://invisibleroads.com/tutorials/geodjango-googlemaps-build.html> (Official GeoDjango Tutorial)

To Learn More About Django

Besides Django tutorials and examples, you can actually attend conferences with a focus on Django and here are a few:

- DjangoCon US
- DjangoCon Europe
- DjangoCon Asia Pacific
- DjangoCon Australia
- ...
- You can see Django is popular world-wide!

References

- <http://www.webdesignish.com/the-best-web-development-frameworks.html>
- <https://www.djangoproject.com/> (official Django website)
- <http://www.eecho.info/Echo/python/history-of-django/>
- Ben Askins & Alan Green. A Rails/Django Comparison. Open Source Developer's Conference, 2006
- Adrian Holovaty and Jacob Kaplan-Moss. The definitive guide to django, 2010
- <http://djangopackages.com/> (Django packages)

References

- <https://github.com/dcramer/django-debug-toolbar>
(Django Debugging Toolbar)
- <https://bitbucket.org/kmike/django-easy-maps>
(Django Easy Maps)
- <https://github.com/toastdriven/django-haystack>
(Django Haystack)
- <https://github.com/etianen/django-reversion>
(Django-revision)
- <http://south.aeracode.org> (South)
- <https://github.com/gcollazo/Fabulous> (Fabulous)

References

- <http://geodjango.org/> (GeoDjango)
- <http://www.djangosites.org/> (DjangoSites)
- <http://djangocon.us/blog/2011/11/08/get-ready-2012/> (DjangoCon US)
- <http://2012.djangocon.eu/> (DjangoCon EU)