The Django Web Framework – Part IV

Hamid Zarrabi-Zadeh

Web Programming – Fall 2013
Outline

• Introduction
• URL Patterns
• Views
• Generic Views
• Class-Based Views
Introduction

• In last session, we learned how to define the data model

• In this session, we focus on the controller logic which decides what data to render which template with, and URL dispatching to determine what logic is performed for a given URL
URL Patterns
URLs

- A clean, elegant URL scheme is very important in a high-quality Web application
- Django lets you design URLs however you want, with no framework limitations
- There is no .asp or .cgi required, and none of the nonsense stuff like ?q=12-329&post=09u
To design URLs for an app, you create a Python module informally called a **URLconf** (URL configuration)

This module is pure Python code and is a simple mapping between URL patterns (simple regular expressions) to Python functions (your views)

This mapping can reference other mappings, and, because it’s pure Python code, it can be constructed dynamically

This enables practically unlimited flexibility
Example

- Here's a sample URLconf

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

urlpatterns = patterns('blog.views',
    url(r'^$', 'index'),
    url(r'^archive/(\d{4})/$', 'archive'),
    url(r'^archive/(\d{4})/(\d{2})/(\d+)/$', 'details'),
)

# Example: /archive/2013/12/08/
```
Named Groups

• It is possible to use named regular-expression groups to capture URL parts and pass them as keyword arguments to a view

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

urlpatterns = patterns('blog.views',
    url(r'^$', 'index'),
    url(r'^archive/(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d+)/$', 'details'),
)

# example: /archive/2013/12/08/ would call
# blog.views.details(request, year='2013',
# month='12', day='08')
```
Multiple Patterns

• We can add multiple `patterns` objects together, possibly with different view prefixes, like this:

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

urlpatterns = patterns('blog.views',
    url(r'^$', 'index'),
    url(r'^archive/(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d+)/$', 'details'),
)

urlpatterns += patterns('mytags.views',
    url(r'^tag/(?P<tag>\w+)/$', 'tag'),
)
```
Including URL Patterns

- At any point, your URL patterns can include other URLconf modules or URL patterns

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

urlpatterns = patterns('',
    url(r'^$','mysite.main.views.homepage'),
    url(r'^help/','include('apps.help.urls'),),
    url(r'^credit/','include(extra_patterns)),
)

extra_patterns = patterns('',
    url(r'^reports/$','credit.views.report'),
    url(r'^charge/$','credit.views.charge'),
)
```
Passing Extra Arguments

- you pass extra arguments to your view functions, as a Python dictionary

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

urlpatterns = patterns('blog.views',
    url(r'^blog/(?P<year>[\d{4}]+)/$', 'archive', {'foo': 'bar'}),
)
```
A common need when working on a Django project is the possibility to obtain URLs in their final forms.

```python
from django.conf.urls import patterns, url
urlpatterns = patterns('blog.views',
    url(r'^blog/\d{4}/$', 'archive',
        name='archive'),
)

<a href="{% url 'archive' 2013 %}">Archive</a>
```
Views
A view function, or view for short, is simply a Python function that takes a Web request and returns a Web response.

The response can be an HTML 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.

There’s no other requirement—no “magic”
A Simple View

• Here’s a simple view that returns the current date and time

```python
from django.http import HttpResponse
import datetime

def current_datetime(request):
    now = datetime.datetime.now()
    html = 'It is now {}'.format(now)
    return HttpResponse(html)
```
Returning Errors

- Returning HTTP error codes in Django is easy
- There are subclasses of HttpResponse for a number of common HTTP status codes

```python
from django.http import HttpResponse, HttpResponseNotFound

def my_view(request):
    if foo:
        return HttpResponseNotFound('Page not found')
    else:
        return HttpResponse('Page was found')
```
Because 404 errors are the most common HTTP error, there’s an easier way to handle them.

You can raise Http404 at any point in a view.

```python
from django.shortcuts import render
from django.http import Http404
from blog.models import BlogPost

def detail(request):
    try:
        p = BlogPost.objects.get(pk=poll_id)
    except BlogPost.DoesNotExist:
        raise Http404
    context = { 'post': p }
    return render(request, 'post.html', context)
```
Generic Views
Class-Based Views

• A view is a callable which takes a request and returns a response
• This can be more than just a function; e.g., it can be a class

```python
from django.conf.urls import patterns
from django.views.generic import TemplateView

urlpatterns = patterns('',
    (r'^about/', TemplateView.as_view(
        template_name='about.html')),
)
```
Subclassing

- You can use inheritance as usual

```python
# blog/views.py
from django.views.generic import TemplateView

class AboutView(TemplateView):
    template_name = 'about.html'

# blog/url.py
from django.conf.urls import patterns
from blog.views import AboutView

urlpatterns = patterns('',
    (r'^about/$', AboutView.as_view),
)
```
Generic Views

• Django ships with generic views to do the following
  – Display list and detail pages for a single object
  – Present date-based objects in year/month/day archive pages, associated detail, and “latest” pages
  – Allow users to create, update, and delete objects
Generic List View

```python
# blog/views.py
from django.views.generic import ListView
from blog.models import BlogPost
class BlogList(TemplateView):
    model = BlogPost
    template_name = 'about.html'
```

```python
# blog/url.py
from django.conf.urls import patterns
from blog.views import BlogList
urlpatterns += patterns('',
    url(r'^list/$', BlogList.as_view()),
)```

# blog/views.py
from django.views.generic import ListView

class PostDetail(DetailView):
    model = BlogPost
    template_name = 'detail.html'
    context_object_name = 'post'

# blog/url.py
from blog.views import PostDetail

urlpatterns += patterns('',
    url(r'^post/(?P<pk>\d+)/$','
        PostDetail.as_view(), name='post-details'),
)
Generic Edit View

```python
# blog/views.py
from django.views.generic.edit import UpdateView

class PostUpdate(UpdateView):
    model = BlogPost
    fields = ['title', 'body']

# blog/url.py
from blog.views import PostUpdate

urlpatterns += patterns('',
    url(r'^update/(?P<pk>\d+)/$',
        PostUpdate.as_view(),)
)```
References

- Django Documentation

- Python Web Development with Django
  - By Jeff Forcier, Paul Bissex, Wesley Chun