dfpp

Release b

December 29, 2014

Contents

Chapter 1: Hello world

The objective of this chapter is to create a web server that listens on a TCP port and responds with "Hello, world!" when the server root (i.e. the path "/") is requested.

1.1 Pure Python

First, let's see how we can do it with pure *Python* before going on to do it with Django.

```
#!/usr/bin/env python

import textwrap

from six.moves.BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer


class HelloRequestHandler(BaseHTTPRequestHandler):

    def do_GET(self):
        if self.path != '/':
            self.send_error(404, "Object not found")
            return
        self.send_response(200)
        self.send_header('Content-type', 'text/html; charset=utf-8')
        self.end_headers()
        response_text = textwrap.dedent('''\
            <html>
            <head>
                <title>Greetings to the world</title>
            </head>
            <body>
                <h1>Greetings to the world</h1>
                <p>Hello, world!</p>
            </body>
            </html>
        ''')
        self.wfile.write(response_text.encode('utf-8'))


server_address = ('', 8000)
httpd = HTTPServer(server_address, HelloRequestHandler)
httpd.serve_forever()
```

If you execute this program and visit http://localhost:8000/ with your browser, you should see it in action.

1.2 Installing Django and creating a project and app

Doing the same thing with Django is considerably more complicated (but it scales; whereas if we added functionality to the pure python version it would become chaotic very soon).

First, make sure you have Django installed. Assuming you are using *virtualenv*, a simple `pip install django` should suffice.

Next, we need to create a Django project and a Django app:

```
django-admin startproject my_first_django_project
cd my_first_django_project
python manage.py startapp hello
```

We now have a project named `my_first_django_project` and an app named `hello`. The concept of a Django app is a big discussion which we will leave for later. For now we can make the simplifying assumption that a project consists of one or more apps.

When we executed `python manage.py startapp hello`, Django created a directory `hello` with several files. In this first chapter we will not be using some of these files, so it's better to get them out of the way. Delete the files `hello/admin.py` and `hello/models.py`, and the directory `hello/migrations`.

1.3 Writing the code

First, edit `hello/views.py` so that it contains this:

```
import textwrap

from django.http import HttpResponse
from django.views.generic.base import View


class HomePageView(View):

    def dispatch(request, *args, **kwargs):
        response_text = textwrap.dedent('''\
            <html>
            <head>
                <title>Greetings to the world</title>
            </head>
            <body>
                <h1>Greetings to the world</h1>
                <p>Hello, world!</p>
            </body>
            </html>
        ''')
        return HttpResponse(response_text)
```

If you've met the term "view" in the past, beware: Django's usage of the term might be completely different from what you've seen. Some texts use the term "controller" or "logic" or "functionality" instead, and they use "view" or "presentation" or "appearance" for what Django calls a template. In Django parlance, a view is a Python class (or a Python function, but this is legacy).

Next, create `hello/urls.py` with the following contents:

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

from hello.views import HomePageView

urlpatterns = patterns(
    '',

    url(r'^$', HomePageView.as_view(), name='home'),
)
```

Change `my_first_django_project/urls.py` so that it looks like this:

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

urlpatterns = patterns(
    '',

    url(r'', include('hello.urls')),
)
```

Finally, edit `my_first_django_project/settings.py`. Around line 55 the `DATABASES` variable is defined; remove the definition. Around line 30 a variable called `INSTALLED_APPS` is defined as a sequence. We need to remove the first four items of the sequence and add `'hello'` to the end. Variable `MIDDLEWARE_CLASSES` follows, and in there we need to remove the line containing `SessionAuthenticationMiddleware`. These two definitions thus become:

```
INSTALLED_APPS = (
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'hello',
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
)
```

Our Django site is ready. Start it like this:

```
python manage.py runserver
```

and then visit http://localhost:8000/ in your browser.

1.4 Anatomy of manage.py

As you have seen, Django execution begins at `manage.py`. Django created this file itself when we executed `django-admin startproject`. So far we have used this script twice: once in order to create a new app (`python manage.py startapp hello`), and once in order run the server (`python manage.py runserver`). `manage.py` can do many more things, of which you can get an idea by running `python manage.py help`. You can also try `python manage.py help runserver` to get help for the `runserver` subcommand. Here are some useful variations:

```
python manage.py runserver 8001  # Listen on specified port (8001 in this case)
python manage.py runserver 0.0.0.0:8001  # Listen on all interfaces
```

The last one is particularly important because by default the server listens on the local interface only. Usually this suffices, but sometimes it does not.

`python manage.py runserver` is only meant for development. When you deploy your application with nginx or apache or another web server, you won't be using `python manage.py runserver`; instead, you will be using `my_first_django_project/wsgi.py`. However, for the time we will stick to using `python manage.py runserver`.

These are the entire contents of `manage.py`:

```
#!/usr/bin/env python
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "my_first_django_project.settings")

    from django.core.management import execute_from_command_line

    execute_from_command_line(sys.argv)
```

What is important here is that the script sets an environment variable that specifies the project's configuration file, i.e. `my_first_django_project/settings.py`. So, to recap, when you type `python manage.py runserver`, django reads your project's configuration file, and then starts a server that is listening on port 8000, waiting for requests. Let's now see what happens when you give it such a request.

1.5 Anatomy of a web request

When you fire up your browser and ask it to give you http://localhost:8000/, your browser makes a request like this to Django:

```
GET / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Linux i686 on x86_64; rv:33.0) Gecko/20100101 Firefox/33.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The most important part is the first line: the browser asks to GET the resource named "/". If you had asked for http://localhost:8000/blog/18/, the browser would have asked to get the resource named "/blog/18/". In our case, Django needs to find out which view is responsible for the resource named "/". It does this by looking up a table that we call the URLconf.

If you look at the configuration file, `my_first_django_project/settings.py`, you will find the following line:

```
ROOT_URLCONF = 'my_first_django_project.urls'
```

This configuration parameter tells Django that our root URLconf is in file `my_first_django_project/urls.py`. This is one of the files we created above. Let's look at it again:

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

```
urlpatterns = patterns(
    '',

    url(r'', include('hello.urls')),
)
```

The table that maps resources to views is the contents of the `urlpatterns` variable. We won't go into details now; what our root URLconf says is that all requests should be handled by another URLconf: `hello.urls`:

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

from hello.views import HomePageView

urlpatterns = patterns(
    '',

    url(r'^$', HomePageView.as_view(), name='home'),
)
```

This says that the resource named "/" is to be served by `HomePageView`. (In the URLconf, resources are specified without the leading slash, so our resource is actually the empty string, which is matched by r'^$'.) There is no need to understand the details clearly at this stage; we will deal with it in more detail in the next chapter.

So now Django knows that the requested page must be served by `HomePageView`, which is this:

```
import textwrap

from django.http import HttpResponse
from django.views.generic.base import View


class HomePageView(View):

    def dispatch(request, *args, **kwargs):
        response_text = textwrap.dedent('''\
            <html>
            <head>
                <title>Greetings to the world</title>
            </head>
            <body>
                <h1>Greetings to the world</h1>
                <p>Hello, world!</p>
            </body>
            </html>
        ''')
        return HttpResponse(response_text)
```

The following things will now take place:

1. Django will construct a `HomePageView` object.
2. Django will call the object's `dispatch()` method.
3. `dispatch()` will return a `HTTPResponse` object, which Django will use in order to provide the response.

1.6 Unit testing

Let's see how to *unit test* our Django app. Remove file `hello/tests.py`. Instead, create directory `hello/tests`, an empty file `hello/tests/__init__.py`, and a `hello/tests/test_views.py`.

In fact, our Django app is so simple that if we just used the file `hello/tests.py` that Django created for us it would have been fine. However, your apps will rarely actually be that simple, and you will practically always create a `tests` directory that will be containing various test modules. This happens to be standard practice as well, and it's better to get into that habit right from the start. We are naming our first test module `test_views.py` because it tests the functionality of `views.py`; this is also a standard convention. In any case, it is very important that its name begins with `test`.

These should be the contents of `hello/tests/test_views.py`:

```
from django.test import SimpleTestCase


class HomePageViewTestCase(SimpleTestCase):

    def test_request_home_page(self):
        response = self.client.get('/')
        self.assertContains(response, 'Hello, world!', status_code=200)
```

`django.test.SimpleTestCase` inherits the Python standard library's `unittest.TestCase`. It adds some Django-specific functionality. One is the `client` attribute, which contains an HTTP client which can create HTTP requests to the Django server. It also adds some extra assertion methods, of which we have used one here. The rest should be self-explanatory.

You can run the unit tests simply like this:

```
python manage.py test
```

CHAPTER 2

What does "For Python Programmers" mean?

This tutorial assumes you already know some things. It does not attempt to teach you Python (I assume you are comfortable with Python 2, Python 3, and six), pip, virtualenv, unit testing, TCP/IP networking, HTTP, or relational databases. You can hardly be called a Python Programmer if you don't have good understanding of all these things. However, if you are missing something, I provide some pointers.

2.1 Python

Some people like the official Python Tutorial, but I don't. I learned Python with Python Essential Reference by D. M. Beazley. I think it was a great book, but it was suitable only for experienced programmers. If you are not an experienced programmer, maybe you need another book - there is a large number of such books on the market; you have to choose. If you are an experienced programmer, maybe the newer editions of Beazley are still great, but I don't really know since I haven't read them.

2.2 Python 2 vs. Python 3, and six

It is my opinion that everyone should learn Python 3 and that it should be used in all new projects. If you are an experienced Python 2 programmer, make sure you learn Python 3. If you don't know Python yet, make sure the book you choose teaches you Python 3. If it teaches you both 2 and 3, so much the better.

In this tutorial the examples run both with 2 and 3, and for that they use six.

2.3 pip and virtualenv

pip, practically speaking, is Python's package manager. For this tutorial, you don't need to know how to package your own code, but you need to be able to install and manage stuff with pip. virtualenv is a tool every Python programmer should be familiar with. After you become comfortable with it, virtualenvwrapper will make you life even easier.

2.4 Unit testing

There is so much urging on the net about the importance of unit testing that I have absolutely nothing to add. I assume you are familiar with the Python unittest standard library module.

2.5 TCP/IP networking and HTTP

Django is used to create web applications, and web applications are client-server applications where the client communicates with the server with HTTP, which is a protocol built on top of TCP, which is a protocol built on top IP. It's hard to develop web applications if you don't understand all that very clearly.

If you have studied computing, you probably took one or two courses on computer networks and you should know all that already. If you don't, it will be very useful to find a good book and learn.

2.6 Relational databases

The problem with relational databases is that everyone will tell you they know them, when in fact very few people do. Same thing with books. Anyway, you should be comfortable at least with the third normal form. I've found Practical Issues in Database Management, by Fabian Pascal, to be quite helpful. I think that the word "practical" in the title is meant to tell you that you can understand it, because it approaches the issues from a practical point of view, it doesn't use relational calculus. However, it is theoretical in the sense that it will tell you few solutions to your real problems; but it will help your mind think in the proper way. Don't be misled into thinking it's old - what the heck, it is more than 10 years younger than the second edition of K&R. It's much more likely for this tutorial to be outdated than for such books.

CHAPTER 3

Copyright

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*