

Django Hands on

Python WorkShop 2007

はじめに

今回はPython Workshop 2007 Django ハンズオンに参加ありがとうございます。

50分という短い枠において何をしたらいいのか迷いましたが、

広く浅く触れるよりもポイントを絞って何かをお伝えした方がよいと判断しました。

今回はURLのアクセスをどう受けて、どのように表示するかということをも50分の枠のなかで説明したいと思います。

Djangoには汎用ビューや便利なショートカットが数多く存在するためにMVCモデルでいうところのコントローラー、DjangoにおけるMVT(モデル、ビュー、テンプレート)におけるビューをあまり書く必要がないために学習が後回しになりがちです。しかしながら、非常に重要なポイントなので今回のセミナーで理解してください。

プロジェクトとアプリケーション

DjangoでWebアプリを作る時にはアプリケーションとアプリケーションを複数まとめたプロジェクトを作ることから始めます。

django-handsonという名前のプロジェクトを作りたい

`django-admin.py startproject`でプロジェクトを作ることができます。任意の場所で以下を実行してみてください。

例は~/dpにプロジェクトを作ってますが、どこでも結構です。

```
uemu:~% mkdir dp
uemu:~% cd dp
/Users/ue38mac/dp
uemu:~/dp% django-admin.py startproject django-handson
uemu:~/dp% ls (windowsの場合はdir)
django-handson
uemu:~/dp% cd django-handson
uemu:~/dp/django-handson% ls
__init__.py      manage.py      settings.py    urls.py
```

demoという名前のアプリを作りたい

プロジェクト`manage.py`ができたので、それを使ってアプリを作ります。

```
uemu:~/dp/django-handson% python manage.py startapp demo
uemu:~/dp/django-handson% ls demo
__init__.py      models.py      views.py
```

django-admin.pyとmanage.pyって何?

両方ともコマンドラインユーティリティで

プロジェクトやアプリを作ったり、データベースの操作、シェルモードを立ち上げなどいろいろなことをしてくれます。

`django-admin.py`も`manage.py`も同じ`django.core.management`の`execute_from_command_line`関数を実行しています。

`manage.py`の場合は`settings.py`に設定してある情報などをDjangoに教える役割があります。

`manage.py`がプロジェクト内でテストサーバーの操作や、コマンドラインやDBの操作などの役割があるのに対して`django-admin.py`を使うのはプロジェクトを作るときくらいです。

Django Hands on

Python WorkShop 2007

settings.pyにdemoアプリを登録する

djangohandson/settings.pyの最後の方にあるINSTALLED_APPSにアプリを追加します。

```
djangohandson/settings.py::  
INSTALLED_APPS = (  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.sites',  
    'djangohandson.demo', #←追加  
)
```

テストサーバーを動かしたり止めたりしたい

Djangoには開発用のテストサーバーが付属しています。

デフォルトの8000ポートで起動する場合

```
uemu:~/dp/djangohandson% python manage.py runserver
```

8000ポート以外で起動する場合 (例だと8080ポート)

```
uemu:~/dp/djangohandson% python manage.py runserver 8080
```

開発しているPC以外のPCからアクセスしたい場合

```
uemu:~/dp/djangohandson% python manage.py runserver 0.0.0.0:8080
```

<http://127.0.0.1:8000/>でブラウザからアクセスしてみましょう

テストサーバーを止める場合は

```
[ctrl] + c
```

HelloWorldを表示したい

<http://127.0.0.1:8080/helloworld/>でアクセスするとHello Worldを表示するように実装しましょう。

urls.pyでhelloworldでのアクセスの設定をしよう

urls.pyのurlpatterns関数の第二引数以降に

(URL正規表現, 呼び出すviews関数, views関数にオプションで渡す辞書(省略可))
のタプルを設定します。

djangohandson/urls.pyの編集

```
from django.conf.urls.defaults import *  
  
urlpatterns = patterns('',  
    # Example:  
    # (r'^djangohandson/', include('djangohandson.foo.urls')),  
    # Uncomment this for admin:  
    # (r'^admin/', include('django.contrib.admin.urls')),  
    (r'^helloworld/$', 'djangohandson.demo.views.hello', {'create_user': 'uemura'}),  
)
```

<http://127.0.0.1:8000/helloworld/>でアクセスがきたら、djangohandson/views.pyのhello関数に引数にcreate_userをつけて渡します。

Django Hands on

Python WorkShop 2007

djangohandson/demo/views.pyにhello関数を作ろう

```
djangohandson/demo/views.py::
```

```
from django.http import HttpResponseRedirect

def hello(request, create_user):
    return HttpResponseRedirect('<h1>Hello World</h1><p>%s</p>' % create_user)
```

view関数の役割

HttpRequestオブジェクトを第一引数に受けて、HttpResponseオブジェクトを返すのが決まりです。第一引数はrequestとすることが慣習になっています。

ではHttpRequestオブジェクトとHttpResponseオブジェクトとは何でしょう???

HttpRequest : リクエストのメタデータを持っています

HttpResponse : HTML (など) を作るデータを持っています

????? よく分からないのでPrintテストしてみましょう!

```
djangohandson/demo/views.py::
```

```
def printtest(dic):
    for key, value in dic.items():
        print ''

%s:
    type: %s
    dict: %s
    str: %s'' % (key, type(value), value.__dict__, value)

def hello(request, create_user):
    res = HttpResponseRedirect('<h1>Hello World</h1><p>%s</p>' % create_user)
    printtest(dict(request=request, response=res))
    return res
```

* 他にdir関数やhelp関数、requestやresponse以外にもglobals()やlocals()などもやってみましょう

サーバーを動かしたターミナル (コマンドプロンプト) を見てみましょう

```
request:
    type: <class 'django.core.handlers.wsgi.WSGIRequest'>
    dict: {~~長いので省略~~}
    str: <WSGIRequest
GET:<MultiValueDict: {}>,
POST:<MultiValueDict: {}>,
COOKIES:{},
META:{~~長いので省略~~}>

response:
    type: <class 'django.http.HttpResponse'>
    dict: {'cookies': <SimpleCookie: >, '_is_string': True, '_charset': 'utf-8',
'headers': {'Content-Type:text/html; charset=utf-8'}, 'status_code': 200, '_container':
['<h1>Hello World</h1><p>uemura</p>']}
    str: Content-Type: text/html; charset=utf-8

<h1>Hello World</h1><p>uemura</p>
```

Pythonらしくクラスや辞書や辞書拡張のデータ型でいろいろ持っているのが分かります。

django.core.handlers.wsgi.handlerや、django.http.HttpResponseクラスを使っています。興味があればこの辺のソースを読むとより深いrequest,HttpResponseオブジェクトの知識が得られるでしょう。

GETやPOST,COOKIESなども辞書や辞書拡張のデータ型で持っているのも分かります。

試しに

<http://127.0.0.1:8000/helloworld/?framework=django>

Django Hands on

Python WorkShop 2007

でアクセスしてみましょう。

するとGETの部分が以下ようになったのが分かります。

```
GET:<MultiValueDict: {'framework': ['django']}>,
```

ではGETにframeworkがあった場合にはその値、なければWorldになるようにしてみましょう。

```
djangohandson/demo/views.py::
```

```
def hello(request, create_user):
    if request.GET.has_key("framework"):
        w = request.GET["framework"]
    else:
        w = "World"
    #w = request.GET.get("framework", "World") #一行で書くなら
    res = HttpResponse('<h1>Hello %s</h1><p>%s</p>' % (w, create_user))
    return res
```

今回はGETでしたがPOSTも同じような書き方をします。

HTMLの直書きは微妙なのでTemplateを使おう

HTMLをロジックのなかに書き込むのはいい書き方とは言えません。Djangoには便利で高速なテンプレートシステムがあるのでそれを使いましょう。

Templateを使うためにはtemplateファイルがどこにあるのかDjangoに教えなくてははいけません。

方法には

- settings.pyのTEMPLATE_DIRSに絶対PATHを指定する方法
- アプリ直下にtemplatesディレクリを作る方法

があるのですが、今回は後者にしましょう。

```
uemu:~/dp/djangohandson% mkdir demo/templates
```

今回はbase.htmlというのを作ります

```
djangohandson/templates/base.html::
```

```
<html>
  <body>
    <h1>Hello {{ object }}</h1>
    <p>created by {{ create_user }}</p>
  </body>
</html>
```

views.pyは以下のように変更します。

```
djangohandson/demo/views.py::
from django.http import HttpResponse
from django.template import Context, loader

def hello(request, create_user):
    t = loader.get_template("base.html")
    if request.GET.has_key("framework"):
        w = request.GET["framework"]
    else:
        w = "World"
    #w = request.GET.get("framework", "World") #一行で書くなら
    c = Context({"object":w, "create_user":create_user})
    res = HttpResponse(t.render(c))
    return res
```

まずdjango.templateからContextとloaderをインポートします。

Django Hands on

Python WorkShop 2007

loaderのget_templateメソッドでbase.htmlを読み込み

テンプレートに渡す「変数名」と「変数の値」をContextで作ります。引数は辞書型になります。

この場合はobjectという変数名でworldかGETで取得したframework名を値に、create_userという変数名でcreate_userを値にしています。

テンプレートとContextをレンダリングすることでbase.htmlの{{ object }}と{{ create_user }}に値が入ります。

ちょっと分かりにくかったかもしれないのでコマンドラインでテンプレートとコンテキストの挙動を確認してみましょう。

Shellモードで確かめる

```
uemu:~/dp/djangohandson% python manage.py shell
```

でコマンドラインモードが起動します。iPythonをインストールしてる場合はiPythonが起動します。

iPythonはDjangoのようなフレームワークを扱うときには特に便利なので使ってない場合は是非インストールしておきましょう。

(今回はなくても大丈夫です)

```
>>> from django.template import Template, Context
>>> t=Template("{% hoge %} : {% huga %}")
>>> c = Context({"hoge":"HOGE", "huga":"HUGA"})
>>> t.render(c)
'HOGE : HUGA'
```

先ほどはloaderを使いましたが、今回は直接Templateオブジェクトを生成しています。

他にも.(ドット)を使うことで以下のように値の照合も行えます

```
>>> class Person:pass
...
>>> p = Person()
>>> p.name = "SUZUKI"
>>> t = Template("Hello! {% person.name %}")
>>> c = Context({"person":p})
>>> t.render(c)
'Hello! SUZUKI'
Out[10]: 'Hello! SUZUKI'
```

このようにテンプレートシステムが変数名中にドットを見つけた場合、以下の順で値の照合を試みます

- foo["bar"] のような辞書としての照合
- foo.bar のような属性値の照合.
- foo.bar() のようなメソッド呼び出し.
- foo[bar] のようなリストのインデックス指定.

(和訳ドキュメントより)

URLで正規表現にマッチさせた部分をviews.pyに送る

djangoの特徴の一つはURLの設計を正規表現のマッチで実現していることです。

今回は [http://127.0.0.1:8000/helloworld/\(任意の文字列\)/\(任意の整数\)/](http://127.0.0.1:8000/helloworld/(任意の文字列)/(任意の整数)/)

というURLにアクセスしたときに

Hello (任意の文字列) を(任意の整数) だけ繰り返すことを実現してみましょう。

例えば

<http://127.0.0.1:8000/helloworld/django/3/>の場合::

Django Hands on

Python WorkShop 2007

```
Hello django
Hello django
Hello django
```

と表示させます。

ではurls.pyはこのようにします。

```
djangohandson/urls.py::
from django.conf.urls.defaults import *

info_dict = {"create_user": "uemura"}
urlpatterns = patterns('',
    (r'^helloworld/$', 'djangohandson.demo.views.hello', info_dict),
    (r'^helloworld/(?P<name>[-\w]+)/(?P<repeat>\d+)/$',
'djangohandson.demo.views.hello_repeat', info_dict),
)
```

create_userを規定して辞書は重複になるのでinfo_dictという変数にして外にだしました。

(?P<name>[-\w]+)と(?P<repeat>\d+)ですが、これはPythonの正規表現の書き方でマッチした部分を<>で囲まれた変数として取得します。Djangoの場合はこれを続くviews関数の引数として渡します。

<http://127.0.0.1:8000/helloworld/django/3/>

というアクセスがあった場合。

(?P<name>[-\w]+)にマッチするdjangoがnameという変数、

(?P<repeat>\d+)にマッチする3がrepeatという変数、

3つ目の要素のinfo_dictのuemuraがcreate_userという変数として

djangohandson.demo.views.hello_repeat関数に渡されます。

djangohandson/demo/views.pyのhello_repeat関数を作る

3つの引数を渡されるので、必ず受け取るrequestを入れて4つの引数を持つ関数にします。

```
djangohandson/demo/views.py::
```

```
def hello_repeat(request, name, repeat, create_user):
    object_list = [name] * int(repeat)
    t = loader.get_template("hello_repeat.html")
    c = Context({"name": name, "object_list": object_list, "create_user": create_user})
    return HttpResponse(t.render(c))
```

注意点としては、repeat引数は整数でマッチしたもので整数型で渡されそうなものですが、引数はすべて文字列で渡されます。

よって、int型にキャストさせました。

あとはloaderで呼び出す新しいテンプレートhello_repeat.htmlを作ればOKです。

```
djangohandson/demo/templates/hello_repeat.html::
```

```
{% for object in object_list %}
    <h1>Hello {{ object }}</h1>
{% endfor %}
```

リストを渡すのでテンプレートのfor構文を使います。

Pythonに近い書き方なので何となく分かるのではないのでしょうか？

{% %}はロジックを制御するタグと呼ばれるものです。

{{}} はテンプレートを処理する際に実際の値に置き換えられる 変数 (variable) です。

Django Hands on

Python WorkShop 2007

テンプレートの継承

<html>や<body>タグやcreated byなど共通のところをテンプレートごとに重複して書くのは無駄だし間違いが生じやすいです。

djangoはテンプレートを継承することによって差分のみのテンプレートを書く事ができます。

```
djangohandson/demo/templates/base.html::
<html>
  <body>
    {% block content %}
      <h1>Hello {{ object }}</h1>
    {% endblock %}

    {% block footer %}
      <p>created by {{ create_user }}</p>
    {% endblock %}
  </body>
</html>
```

継承先のテンプレートで変更したいところをblockタグで名前をつけて囲みます。

今回の例ですとcontentとfooterという名前にしました。

```
django/handson/demo/hello_repeat.html::
{% extends "base.html" %}
{% block content %}
  {% for object in object_list %}
    <h1>Hello {{ object }}</h1>
  {% endfor %}
{% endblock %}
```

継承先のテンプレートではまず、継承元のテンプレートとextendsタグで指定します。

そして変更したいblockだけを書きます。

footerという名前をつけたblockは書き換えないので元のbase.htmlそのままを表示します。

毎回表示するかもしれない変数をurls.pyで毎回指定するのは面倒なのでなんとかしたい

今回の例ですとcreate_userをurls.pyにてviews関数すべてに指定しています。変化する可能性がないのならこれはちょっと面倒ですね。

こういった定数のようなものを扱うにはviews.pyの中でContextではなく、RequestContextを使えばOKです。

RequestContextを扱うにはsettings.pyにContextの辞書を返す関数を持ったファイルを指定する必要があります。

今回はdjangohandson/context_processors.pyというファイルとcontextという関数を作りましょう。(ファイル名関数名ともに何でも良い)

```
djangohandson/context_processors.py::
def context(request):
    return {'create_user': 'uemura'}
```

次にsettings.pyの設定です。

TEMPLATE_CONTEXT_PROCESSORSに関数をタプルで指定するのですがTEMPLATE_CONTEXT_PROCESSORSはdjango.conf.global_settingsで既に重要なcontextを指定しており、そのまま指定すると上書きしてしまうので以下のような書きかたをします。

djangohandson/settings.pyに追加::

Django Hands on

Python WorkShop 2007

```
from django.conf import global_settings

TEMPLATE_CONTEXT_PROCESSORS = global_settings.TEMPLATE_CONTEXT_PROCESSORS + (
    'djangohandson.context_processors.context',
)
```

djangohandson/demo/views.pyのContextをRequestContextに変更します。RequestContextは第一引数にrequest、第二引数にcontextの辞書を指定します

```
djangohandson/demo/views.py::
from django.http import HttpResponse
from django.template import RequestContext, loader #RequestContextをインポート

def hello(request): #create_user引数を削除
    t = loader.get_template("base.html")
    if request.GET.has_key("framework"):
        w = request.GET["framework"]
    else:
        w = "World"
    #c = Context({"object":w, "create_user":create_user})
    c = RequestContext(request, {"object":w}) #RequestContextの設定
    res = HttpResponse(t.render(c))
    return res

def hello_repeat(request, name, repeat): #create_user引数を削除
    object_list = [name] * int(repeat)
    print object_list
    t = loader.get_template("hello_repeat.html")
    #c = Context({"name":name, "object_list":object_list, "create_user":create_user})
    c = RequestContext(request, {"name":name, "object_list":object_list})
#RequestContextの設定
    return HttpResponse(t.render(c))
```

ではdjangohandson/urls.pyの不要になったinfo_dictを消します。

```
djangohandson/urls.py::
from django.conf.urls.defaults import *
import datetime

urlpatterns = patterns('',
    (r'^helloworld/$', 'djangohandson.demo.views.hello'),
    (r'^helloworld/(?P<name>[-\w]+)/(?P<repeat>\d+)/$',
'djangohandson.demo.views.hello_repeat'),
)
```

ブラウザでアクセスして確認してみましょう。

以上でハンズオンセミナーの講習を終えたいと思います。
最後まで参加していただき誠にありがとうございました。

株) 未来屋書店 上村 誠 ueblog@gmail.com <http://www.ueblog.org/blog/>