Học Django

1. Tạo project django

Cmd: django-admin startproject tên project

1. Start server:

Command trong vs code: python manage.py runserver

1. Tạo app

* Python manage.py start app tên app

Khi tạo xong 1 app thì ta phải đăng ký app đó trong file setting.py



Đường dẫn trang web sẽ dc định nghĩa trong file urls.py trong thư mục chính

Với mỗi app ta có thể định nghĩa các file urls.py riêng biệt nhưng sau đó trong file urls.py ở thư mục gốc chúng ta phải include vào. **Ví dụ poll là tên app, urls là file path**



Sau đó vào trình duyệt gõ 127.0.0.1:8000/path

Chức năng Model để tương tác với CSDL

Trước hết tạo cơ sở dl ta phải tạo cái migration trước để nó lưu lại những thay đổi trong csdl. 

Để tạo migration ta gõ lệnh. Python manage.py makemigrations

Để tạo database gõ lệnh: Python manage.py migrate

* Tạo user admin: python manage.py createsuperuser
* Khi muốn quản lý data thì ta vào phần admin.py
* Trong file admin.py import class trong phần models vào
* Comand: admin.site.register(tên class)
* Mặc định thì django chỉ hiển thị một phần của database ví dụ như title, để hiển thị thêm ta có thể bổ sung các list list\_display, list\_filter, search\_fieldsm,…
* 

Khi tương tác với csdl, khi muốn view ra nội dung trong database ta có thể sử dụng một số hàm tương tác để biến nội dung trong database thành các mã html để có thể hiển thị chính xác nội dung cần thể hiện.

Ví dụ như thẻ: safe: Thực thi nội dung trong một trường database có cú pháp là một mã HTML

Hoặc linrbreak: Tạo nội dung xuống dòng, nếu như trong nội dung của database muốn xuống dòng, còn nếu ko thì nó sẽ viết liền



**Loại bỏ hardcode url**

Trong đường dẫn path có thể có những đường dẫn tương đối giống nhau, nên khi gọi view sẽ dễ bị nhầm lẫn, vậy ta có cách là trong phần path của file url ta đặt tên cho path đó, rồi sau đó gọi tên nó trong file template





Khi ta bị lỗi 404,505 thì django sẽ trả về lỗi mặc định có chi tiết lỗi thì như thế sẽ bị lỗi bảo mật, và ta có thể tự định nghĩa lỗi trả về. Ví dụ:





Tham số Debug là khi ta phát triển web thì giá trị là True, còn khi deploy thì ta cho là false

**Mô hình MVC có nghĩa là Model = models**

View = Template

Controler = views

Vậy trong django còn gọi là mô hình MTV

* Khi chạy server thì ta dùng lệnh python manage.py runserver <port> ví dụ 8888
* Để vào dc trang admin thì ta phải tạo 1 tài khoản admin bằng câu lệnh: python manage.py createsuperuser
* File url của project sẽ trỏ đến file url của các app, sau đó trong file url của app mới dẫn đến các view

url(r'^(?P<question\_id>[0-9]+)/results/$', views.results, name='results'),

Chuỗi Regex (?P<question\_id>[0-9]+) cho Django biết có một chuỗi con trong đoạn URL có dạng một con số (có 1 hoặc nhiều chữ số) và biến được truyền vào phương thức detail() có tên là question\_id.

* Sử dụng module “loader” để get template
* from django.template import loader
* template = loader.get\_template('polls/index.html')

|  |
| --- |
| * return HttpResponse(template.render(context, request)) |

Sau đó chúng ta gọi đến phương thức template.render() để tạo nội dung HTML có sử dụng template. Tham số đầu tiên là nội dung HTML trả về, tham số thứ 2 là đối tượng request được gửi đến.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | from django.shortcuts import render    def index(request):      latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]      context = {          'latest\_question\_list': latest\_question\_list,      }      return render(request, 'polls/index.html', context) |

Ngoài ra bạn cũng có thể dùng hàm render() để trả về một đối tượng HttpResponse một cách trực tiếp luôn, với tham số thứ nhất là đối tượng request, tham số thứ 2 là đường dẫn đến file template, tham số thứ 3 là nội dung HTML trả về.

**Đặt namespace cho URL**

 Chúng ta có thể tham chiếu đến các đối tượng url đã định nghĩa trong file urls.py của ứng dụng. Khi chúng ta tạo các đối tượng url trong file urls.py, tham số thứ 3 là name, tham số này do chúng ta tự đặt, và Django cho phép chúng ta tham chiếu đến chúng trong các file template.

Ví dụ: url(r'^details/(?P<question\_id>[0-9]+)/$', views.detail, name='detail'),

Index.html

|  |  |
| --- | --- |
| 1  2  3 | <a href="{% url 'detail' question.id %}">      {{ question.question\_text }}  </a> |

Bằng cách này chúng ta có thể tham chiếu đến đối tượng url trong file urls.py, và khi nào cần thay đổi URL mới thì chúng ta chỉ cần thay đổi trong file urls.py là được:

Để đặt tên namespace cho các đối tượng url thì chúng ta chỉ cần đặt giá trị cho biến app\_nametrong file urls.py là được.

from django.conf.urls import url

from . import views

app\_name = "polls"

### STATIC\_URL

URL để sử dụng khi tham chiếu đến các tệp tĩnh nằm trong STATIC\_ROOT.

**STATICFILES\_DIRS**

Cài đặt này xác định các vị trí bổ sung mà ứng dụng staticfiles sẽ đi qua nếu công cụ tìm FileSystem Downloader được bật

### STATIC\_ROOT

Đường dẫn tuyệt đối đến thư mục nơi “collectstatic” sẽ thu thập các “static file” để triển khai

Đặt cài đặt STATIC\_ROOT vào thư mục mà từ đó bạn muốn sử dụng các tệp này

Chạy lệnh  [**collectstatic**](https://docs.djangoproject.com/en/2.1/ref/contrib/staticfiles/#django-admin-collectstatic) :

$ python manage.py collectstatic

Điều này sẽ sao chép tất cả các tệp từ các thư mục tĩnh của bạn vào thư mục STATIC\_ROOT.

Tức là nó sẽ sao chép tất cả các file khai báo trong **STATICFILES\_DIRS đến thư mục** STATIC\_ROOT

### Ví dụ:

STATICFILES\_DIRS = [

"/home/special.polls.com/polls/static",

"/home/polls.com/polls/static",

"/opt/webfiles/common",

]

**URL:**

**Mỗi yêu cầu url phải kết thúc bằng một dấu gạch chéo**

**VD: url(‘articles/2003/’** ,view**)**

Sử dụng biểu thức chính quy (Regular expressions)

Nếu cú ​​pháp đường dẫn và trình chuyển đổi là không đủ để xác định các mẫu URL của bạn, bạn cũng có thể sử dụng các biểu thức thông thường. Để làm như vậy, hãy sử dụng re\_path () thay vì path ().

Trong các biểu thức chính quy của Python, cú pháp cho các nhóm biểu thức chính quy được đặt tên là (? P <name> pattern), trong đó name là name của group và pattern là một số pattern phù hợp.

VD:

**from** **django.urls** **import** path, re\_path

**from** **.** **import** views

urlpatterns = [

path('articles/2003/', views.special\_case\_2003),

re\_path(r'^articles/(?P<year>[0-9]**{4}**)/$', views.year\_archive),

re\_path(r'^articles/(?P<year>[0-9]**{4}**)/(?P<month>[0-9]**{2}**)/$', views.month\_archive),

re\_path(r'^articles/(?P<year>[0-9]**{4}**)/(?P<month>[0-9]**{2}**)/(?P<slug>[\w-]+)/$', views.article\_detail),

]

Tại bất kỳ thời điểm nào, các mô hình url của bạn có thể “include” các mô-đun URL khác của URL. Bất cứ khi nào Django gặp phải include (), nó sẽ loại bỏ bất kỳ phần nào của URL khớp với điểm đó và gửi chuỗi còn lại đến URLconf kèm theo để xử lý thêm.

## **Path converters**

・str - Khớp với bất kỳ chuỗi không trống nào, ngoại trừ dấu phân cách đường dẫn, '/'. Đây là mặc định nếu một trình chuyển đổi không kèm theo trong biểu thức.

・int - Khớp 0 hoặc bất kỳ số nguyên dương nào. Trả lại một giá trị int.

VD: path('blog/page<int:num>/', views.page),

・slug - Ghép nối bất kỳ chuỗi sên nào bao gồm các chữ cái hoặc số ASCII, cộng với dấu gạch nối và ký tự gạch dưới.

VD: **building-your-1st-django-site**.

・UUID: Ghép một UUID được định dạng. Để ngăn nhiều URL ánh xạ vào cùng một trang, các dấu gạch ngang phải được bao gồm và các chữ cái phải là chữ thường.

Ví dụ: 075194d3-6885-417e-a8a8-6c931e272f00.

・path - Khớp với bất kỳ chuỗi không trống nào, kể cả dấu phân cách đường dẫn, '/'. Điều này cho phép bạn khớp với một đường dẫn URL hoàn chỉnh thay vì chỉ một đoạn của đường dẫn URL như với str.

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('articles/2003/', views.special\_case\_2003),

path('articles/<int:year>/', views.year\_archive),

path('articles/<int:year>/<int:month>/', views.month\_archive),

path('articles/<int:year>/<int:month>/<slug:slug>/', views.article\_detail),

Ví dụ:

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('<page\_slug>-<page\_id>/history/', views.history),

path('<page\_slug>-<page\_id>/edit/', views.edit),

path('<page\_slug>-<page\_id>/discuss/', views.discuss),

path('<page\_slug>-<page\_id>/permissions/', views.permissions),

]

Chúng ta có thể cải thiện điều này bằng cách chỉ nêu tiền tố đường dẫn chung một lần và nhóm các hậu tố khác nhau:

**from** **django.urls** **import** include, path

**from** **.** **import** views

urlpatterns = [

path('<page\_slug>-<page\_id>/', include([

path('history/', views.history),

path('edit/', views.edit),

path('discuss/', views.discuss),

path('permissions/', views.permissions),

])),

]

URLconfs có một hook cho phép bạn chuyển các đối số bổ sung cho các hàm xem của bạn, như một Dictionary Python.

Hàm path () có thể lấy một đối số thứ ba tùy chọn phải là một dictionary của các đối số từ khóa bổ sung để chuyển đến hàm xem.

**from** **django.urls** **import** path

**from** **.** **import** views

urlpatterns = [

path('blog/<int:year>/', views.year\_archive, {'foo': 'bar'}),

]

Trong ví dụ này, đối với yêu cầu tới / blog / 2005 /, Django sẽ gọi view.year\_archive (request, year = 2005, foo = 'bar').

Kỹ thuật này được sử dụng trong khung cung cấp để chuyển siêu dữ liệu và các tùy chọn cho các view

-VIEWS

Phương thức trả về đối tượng HttpResponse từ thư viện

from django.http import HttpResponse

- Khi làm việc với Templates ta thường sử dụng một trong hai thư viện

from django.template import loader

from django.shortcuts import render

- Tham số được kèm trong lời gọi hàm HttpResponse bao gồm có tham số mặc định “request”,”Templates”, và content thường là một dictionnary.

Ví dụ:

Sử dụng thư viện “loader”

def index(request):

    latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

    template = loader.get\_template('polls/index.html')

    context = {

        'latest\_question\_list': latest\_question\_list,

    }

    return HttpResponse(template.render(context, request))

Sử dụng thư viện “render” trả về một đối tượng HttpResponse một cách trực tiếp

from django.shortcuts import render

def index(request):

    latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

    context = {

        'latest\_question\_list': latest\_question\_list,

    }

    return render(request, 'polls/index.html', context)

- [**HttpResponseRedirect**](https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpResponseRedirect): Trả về một URL thích hợp cho các đối số được truyền.

Có các tham số là:

- [get\_absolute\_url()](https://docs.djangoproject.com/en/2.2/ref/models/instances/#django.db.models.Model.get_absolute_url): hàm [get\_absolute\_url()](https://docs.djangoproject.com/en/2.2/ref/models/instances/#django.db.models.Model.get_absolute_url) của model sẽ được gọi

- một view có thể được gọi với tham số :reverse() sẽ được sử dụng để phan giải ngược tên

- Một URL tuyệt đối hoặc tương đối, sẽ được sử dụng như là dành cho vị trí chuyển hướng.

VD:

HttpResponseRedirect(reverse('polls:results', args=(question1.id,)))

HttpResponseRedirect nhận một đối tượng url thông qua hàm reverse(), hàm này sẽ trả về một đường dẫn có dạng như /polls/[ args ]/results/.

Hàm Reverse () có thể reverse rất nhiều mẫu biểu thức chính quy cho URL, nhưng không phải mọi mẫu có thể. Hạn chế chính tại thời điểm này là mẫu không thể chứa các lựa chọn thay thế bằng ký tự thanh dọc ("|"). khá vui khi sử dụng các mẫu như vậy để khớp với các URL đến và gửi chúng đến các chế độ xem, nhưng bạn không thể đảo ngược các mẫu đó.

- **Templates**

Sử dụng ngôn ngữ HTML xen thêm các câu lệnh Python với những ký hiệu đặc biệt gọi là jinja2

Ví dụ:

<h1>{{ question.question\_text }}</h1>

{% if error\_messsage %}

<strong>{{ error\_messsage }}</strong>

{% endif %}

<form action="{% url 'polls:vote' question.id %}" method="post">

    {% csrf\_token %}

    {% for choice in question.choice\_set.all %}

        <input type="radio" name="choice" id="choice{{ forloop.counter }}" value="{{ choice.id }}" />

        <label for="choice{{ forloop.counter }}">{{ choice.choice\_text }}</label>

    {% endfor %}

    <input type="submit" value="Vote" />

</form>

Phần bôi vàng là những thành phần “KEY” nằm trong dictionary, file html lấy ra sử dụng.

Bất cứ bảng nào có một bảng khác chứa khóa ngoại tham chiếu đến khóa chính của nó đều sẽ có một thuộc tính được tạo tự động là một tập hợp các đối tượng của bảng kia. Ở đây bảng Choice chứa khóa ngoại tham chiếu đến bảng Question, do đó bảng Question sẽ có một danh sách các đối tượng Choice, chúng ta có thể lấy danh sách này qua choice\_set.all() (tên danh sách được đặt theo <tên bảng>\_set).

- Tức là từ bảng mà có một bảng khác chứa khóa ngoại tham chiếu đến khóa chính của nó thì đều có thể từ bảng đó mà thao tác với bảng kia.

Ví dụ:

q=bangkhoachinh.objects.get(pk=1)

q.bangkhoangoai\_set.all(): Danh sách các đối tượng bangkhoangoai

q.bangkhoangoai\_set.create(tencot =”giatri”,…): Tạo các đối tượng cho bangkhoangoai

q.bangkhoangoai\_set.count():Lấy số lượng các bản ghi

q.bangkhoangoai\_set.filter():Lọc lấy bản ghi để hiển thị

q.bangkhoangoai\_set.filter():Lọc lấy bản ghi và lưu vào một biến nào đó.

q.save():Lưu vào cơ sở dữ liệu

c.delete():Xóa một bản ghi

Nhưng chỉ áp dụng với những trường mà có q.id = bangkhoangoai.bangkhoachinh\_id

|  |
| --- |
| {% csrf\_token %} |

câu lệnh trên sẽ giúp website của chúng ta chống lại kiểu tấn công *CSRF*

*Các lớp View có sẵn trong django*

|  |
| --- |
| from django.views import generic |

Đầu tiên chúng ta import module generic.

class IndexView(generic.ListView):

class DetailView(generic.DetailView):

Chúng ta sẽ sử dụng 2 loại view là ListView và DetailView, ListView lưu trữ danh sách các đối tượng, DetailView lưu thông tin về một đối tượng cụ thể.

model = Question

Mỗi lớp View cần biết về mô hình dữ liệu mà nó sẽ lưu trữ thông qua thuộc tính model. Khi đã biết loại model mà mình sẽ dùng, các view này tự động “nghĩ” rằng khóa chính trong CSDL có tên là pk, do đó trong các đối tượng url chúng ta gửi lên tham số có tên là pk.

-Chúng ta phải sửa lại các đường dẫn url :

*url(r'^(?P<pk>[0-9]+)/results/$', views.ResultsView.as\_view(), name='results'),*

Mặc định các lớp View này cũng “nghĩ” rằng template sẽ được dùng có dạng <app\_name>/<model\_name>\_detail.html (hoặc <app\_name>/<model\_name>\_list.html), nếu không có file nào như vậy tồn tại thì Django sẽ tự động tạo các file này và sử dụng, chúng ta có thể “bảo” Django sử dụng template do chúng ta tự viết bằng cách gán vào thuộc tính template\_name, như thế Django sẽ không tạo template cho nó nữa.

template\_name = 'polls/detail.html'

Trong hàm detail() cũ, chúng ta tự khai báo các biến question và latest\_question\_list để dùng trong template, mặc định các biến này đã có sẵn trong các lớp DetailView. Tuy nhiên đối với hàm ListView thì biến mặc định lại có tên là question\_list, nên để Django dùng tên do chúng ta tự đặt thì chúng ta phải gắn tên đó vào thuộc tính context\_object\_name hoặc bạn phải dùng tên do Django đặt trong các file template.

class IndexView(generic.ListView):

    ...

    context\_object\_name = 'latest\_question\_list'

*-* ***Static file***

Ngoài nội dung HTML được sinh ra bởi server thì một ứng dụng web còn cần đến các file bổ sung khác, chẳng hạn như các file hình ảnh, Javascript, CSS… Trong Django thì các file này được gọi là file tĩnh.

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />

Dòng {% load staticfiles %} sẽ tự động gán đường dẫn đến thư mục mysite/polls/static/ vào một biến có tên static do Django tự đặt, từ đó bạn chỉ cần lấy biến static là có thể lấy được đường dẫn tuyệt đối đến thư mục này.

Sau đó chúng ta chỉ cần lấy biến static trong cặp thẻ {% %} ra và gắn thêm đường dẫn đến các file css, js… của bạn.

<link rel="stylesheet" type="text/css" href="{% static 'polls/style.css' %}" />

{% extends 'base.html' %}

Kế thừa từ một file html khác. Ở đây ta sẽ khai báo các block giống như các block trong file base.html với mục đích là có thể overide nội dung trong các phần block của file base.html

{% load staticfiles %}

{% block title %} Home Page

{% endblock title %}

Ở đây trong file base.html và file thừa kế nó đều phải có cùng tên block.

Trong file base.html đôi khi chỉ có các dòng layout html còn lại là các khối block lấy từ các file html thừa kế nó.

Hoặc có thể lấy trực tiếp từ một file html bất kỳ bằng hàm “**include**”

{% block sidebar %}

            <div class="col-md-4 side-bar">

            {% include 'includes/sidebar.html' %}

        {% endblock sidebar %}

## Tùy chỉnh form

Khi chúng ta đăng ký các lớp model Question và Choice bằng hàm admin.site.register(), Django sẽ dựa vào các thuộc tính mà chúng ta khai báo trong các lớp đó để hiển thị form trên trang admin. Chúng ta có thể quy định Django chỉ chọn một số thuộc tính được phép hiện ra để chỉnh sửa, thay vì hiện ra hết, bởi vì khi làm một ứng dụng có sử dụng CSDL, có những thuộc tính mà bạn muốn máy tính tự động sinh ra chứ không phải do con người nhập vào, chẳng hạn như ID, ngày giờ…

Chúng ta sẽ thay đổi bằng cách sửa lại code trong file polls/admin.py.

from django.contrib import admin

# Register your models here.

from .models import Question, Choice

class QuestionAdmin(admin.ModelAdmin):

    fields = ['pub\_date', 'question\_text']

admin.site.register(Question, QuestionAdmin)

admin.site.register(Choice)

Việc này làm rất đơn giản, chúng ta chỉ cần viết một lớp kế thừa từ lớp admin.ModelAdmin, ở đây mình đặt tên là QuestionAdmin, sau đó khai báo list có tên là fields có các item là tên các thuộc tính trong model mà chúng ta muốn hiện ra trong trang Admin, sau đó truyền lớp này vào hàm admin.site.register().

*MODEL*

## Sử dụng model

Sau khi đã định nghĩa xong các model thì chúng ta phải thiết lập để Django sử dụng các model đó, bằng cách chỉnh sửa biến INSTALLED\_APPS trong file settings.py.

## Các trường

Thành phần quan trọng và cũng là thành phần bắt buộc phải có trong một model là danh sách các trường hay các thuộc tính của một lớp. Chú ý khi đặt tên bạn nhớ tránh đặt tên trùng với các từ khóa có sẵn trong Django.

## Tham số trong Field

Ngoài các tham số bắt buộc thì còn có một số tham số tùy chọn nữa, ở đây mình chỉ giới thiệu một số:

* null: có giá trị True hoặc False. Tham số này cho Django biết sẽ dùng giá trị NULL thay cho các bản ghi có giá trị rỗng, tức là trong CSDL mà kiểu chuỗi mà có giá trị "" thì sẽ thay bằng NULL.
* blank: có giá trị True hoặc False. Tham số này cho Django biết cột tương ứng trong bảng CSDL được phép NULL hay NOT NULL.
* default: thiết lập giá trị mặc định cho trường.
* help\_text: đây một giá trị chuỗi dùng để hiển thị một đoạn text ngắn mô tả về trường đó, vd như khi di chuột vào cột trên form HTML thì hiện ra đoạn text.
* primary\_key: True hoặc False, chỉ định trường đó có phải là khóa chính hay không. Như bạn đã biết, nếu bạn không chỉ định trường nào làm khóa chính thì Django sẽ tự động tạo một trường AutoField để làm khóa chính luôn. Có một điều nữa là khi bạn thay đổi giá trị khóa chính của một model trong Python thì Django không báo lỗi mà thay vào đó là tạo một đối tượng model mới với khóa chính mới.

## Khóa chính tự tạo

Khóa chính do Django tự tạo ra có dạng sau:

id = models.AutoField(primary\_key=True)

## Quan hệ

Sức mạnh của các cơ sở dữ liệu quan hệ nằm ở tính năng mối quan hệ giữa các bảng. Django hỗ trợ 3 loại quan hệ phổ biến của CSDL quan hệ là: **many-to-one** (**một-nhiều**), **many-to-many**(**nhiều-nhiều**), **one-to-one** (**một-một**).

## Quan hệ Many-to-one

Chúng ta đã làm việc với kiểu quan hệ này trong các bài trước bằng cách dùng lớp django.db.models.ForeignKey. Tham số bắt buộc phải có của ForeignKey là model cha của nó.

## Quan hệ Many-to-many

Để sử dụng mối quan hệ này thì chúng ta sử dụng lớp ManyToManyField với tham số bắt buộc là tên model mà nó sẽ trỏ đến.

Cũng giống như với ForeignKey, bạn cũng có thể dùng quan hệ đệ quy với ManyToManyField.

## Thuộc tính của một mối quan hệ many-to-many

Một mối quan hệ không đơn giản chỉ là quan hệ giữa bảng này với bảng kia, mà chúng còn có các thông tin riêng nữa.

Ví dụ chúng ta có model Person và Group có quan hệ many-to-many, ý nghĩa là một người có thể tham gia nhiều group và một group có thể có nhiều người, ý nghĩa của quan hệ này không chỉ có bao nhiêu đó mà còn có các thông tin khác như người đó tham gia vào nhóm vào ngày nào, lý do tham gia chẳng hạn…

Để thêm các trường vào một mối quan hệ thì chúng ta sẽ dùng đến một tham số tùy chọn trong hàm khởi tạo ManyToManyField là through.

from django.db import models

class Person(models.Model):

    name = models.CharField(max\_length=128)

class Group(models.Model):

    name = models.CharField(max\_length=128)

    members = models.ManyToManyField(Person, through='Membership')

class Membership(models.Model):

    person = models.ForeignKey(Person, on\_delete=models.CASCADE)

    group = models.ForeignKey(Group, on\_delete=models.CASCADE)

    date\_joined = models.DateField()

    invite\_reason = models.CharField(max\_length=64)

Chúng ta phải định nghĩa một model khác lưu trữ các thông tin của một mối quan hệ, tham số through sẽ nhận tên của model này.

Trong Django thì các model kiểu này (Membership) được gọi mà model trung gian (Intermediate model), trong đó bạn phải khai báo rõ ràng các model tham gia vào mối quan hệ này, ở đây là 2 đối tượng ForeignKey: person và group.

Chúng ta phải định nghĩa một model khác lưu trữ các thông tin của một mối quan hệ, tham số through sẽ nhận tên của model này.

Trong Django thì các model kiểu này (Membership) được gọi mà model trung gian (Intermediate model), trong đó bạn phải khai báo rõ ràng các model tham gia vào mối quan hệ này, ở đây là 2 đối tượng ForeignKey: person và group.

bạn phải khai báo rõ ràng các thông tin của mối quan hệ đó ra (bằng cách dùng hàm khởi tạo

ví dụ: m1 = Membership(person=ringo, group=beatles,

... date\_joined=date(1962, 8, 16),

... invite\_reason="Needed a new drummer.")

## Mối quan hệ one-to-one

Để định nghĩa một mối quan hệ one-to-one thì chúng ta sử dụng lớp OneToOneField, lớp này cũng nhận tham số là tên của model mà nó sẽ trỏ tới.

## Tham chiếu model thông qua file

Bạn có thể định nghĩa các lớp model trong các file khác nhau và khi cần tham chiếu đến chúng thì chỉ cần import vô là được.

## Metadata

Các lớp trong Django có thể chứa các metadata (siêu dữ liệu), đây là các thông tin lưu trữ về các thiết lập của model, các thông tin này không được dùng để tạo bảng.

Ví dụ:

from django.db import models

class Ox(models.Model):

    horn\_length = models.IntegerField()

    class Meta:

        ordering = ["horn\_length"]

        db\_table = "oxen"

## Thừa kế model

Model trong Django cũng có thể thừa kế lẫn nhau. Chúng ta có thể viết các lớp model cơ sở (hay model cha) rồi sau đó viết các model khác thừa kế từ lớp model cơ sở này. Các lớp model cơ sở phải được thừa kế từ lớp django.db.models.Model, rồi từ đó các lớp con sẽ kế thừa từ lớp cha này. Model cha có thể có bảng riêng hoặc không có.

Có 3 cách để một model được thừa kế trong Django:

* Thừa kế từ lớp trừu tượng.
* Thừa kế đa bảng.
* Dùng lớp Proxy.

### Thừa kế từ lớp trừu tượng

Chúng ta sử dụng lớp trừu tượng khi muốn model cha chỉ có nhiệm vụ là lưu trữ các thông tin dùng chung trong các model con chứ không phải nhằm mục đích lưu các thông tin thật trong CSDL. Để một model là “trừu tượng” thì chúng ta gán thuộc tính abstract = True của lớp nội Meta.

Ví du:

from django.db import models

class CommonInfo(models.Model):

    name = models.CharField(max\_length=100)

    age = models.PositiveIntegerField()

    class Meta:

        abstract = True

class Student(CommonInfo):

    home\_group = models.CharField(max\_length=5)

Model Student thừa kế từ model trừu tượng là CommonInfo, Student sẽ bao gồm 3 trường là name, age và home\_group.

Khi chúng ta dùng lệnh manage.py migrate, chỉ có model Student mới được tạo bảng cùng với 3 trường của nó. Khi một model con thừa kế model cha mà không khai báo lớp nội Meta thì sẽ tự động thừa kế lớp nội đó từ model cha. Nhưng nếu muốn các model vẫn có thể có lớp nội riêng hoặc kế thừa và thêm các thuộc tính mới từ model cha.

## Thừa kế đa bảng

Thừa kế đa bảng chỉ khác kiểu thừa kế lớp trừu tượng ở chỗ là lớp cha ở đây là lớp “thật” chứ không phải lớp trừu tượng. Tức là các model cha bây giờ sẽ có cả các bảng riêng của chúng trong CSDL.

Khác với kiểu kế thừa từ lớp trừu tượng, ở lớp trừu tượng thì các trường của lớp cha sẽ nằm cùng bảng với các lớp con vì lớp cha không được tạo bảng riêng, với thừa kế đa bảng thì các lớp con cũng sẽ kế thừa các trường của lớp cha nhưng các trường của lớp cha sẽ nằm trong bảng riêng của lớp cha.

Ví dụ:

>>> Place.objects.filter(name="Bob's Cafe")

>>> Restaurant.objects.filter(name="Bob's Cafe")

Chúng ta vẫn có thể truy xuất các thông tin từ cả 2 lớp con và lớp cha như thường.

## Lớp Proxy – lớp ủy quyền

Django cho phép chúng ta tạo ra các lớp Proxy (lớp ủy quyền), các lớp này kế thừa từ lớp cha nhưng không được tạo bảng mới trong CSDL, những lớp con này sẽ chỉ dùng để lưu các phương thức mới chứ không lưu các thuộc tính mới. Bạn có thể thực hiện thêm, sửa, xóa dữ liệu được kế thừa từ lớp cha, những dữ liệu này vẫn lưu ở bảng cha.

Ví dụ:

from django.db import models

class Person(models.Model):

    first\_name = models.CharField(max\_length=30)

    last\_name = models.CharField(max\_length=30)

class MyPerson(Person):

    class Meta:

        proxy = True

    def do\_something(self):

        # ...

        pass

Bất cứ thao tác cập nhật dữ liệu nào từ lớp MyPerson cũng sẽ là thao tác trực tiếp với dữ liệu từ lớp Person.

## Đa thừa kế

Vì Django được phát triển từ Python nên các model trong Django có thể thừa kế từ nhiều model cha. Lớp Meta của model con chỉ có thể thừa kế từ lớp Meta của lớp cha đầu tiên. Ngoài ra khi sử dụng đa thừa kế mà các model cha có cùng chung tên thuộc tính khóa chính thì Django sẽ báo lỗi. Do đó khi sử dụng đa thừa kế, chúng ta nên khai báo các trường làm khóa chính một cách rõ ràng.

Ví dụ:

class Article(models.Model):

    article\_id = models.AutoField(primary\_key=True)

    ...

class Book(models.Model):

    book\_id = models.AutoField(primary\_key=True)

    ...

class BookReview(Book, Article):

    pass

## Tạo đối tượng

Mỗi bản ghi hay mỗi dòng trong từng bảng CSDL sẽ tương ứng với một đối tượng cụ thể. Để tạo một bản ghi trong CSDL thì chúng ta chỉ cần tạo một đối tượng tương ứng với trong Python rồi gọi phương thức save() là xong. Vd:

>>> from blog.models import Blog

>>> b = Blog(name='J.R.R Blog', tagline='All the latest J.R.R news.')

>>> b.save()

Đoạn code trên sẽ thực hiện câu lệnh INSERT, bạn phải gọi phương thức save() thì dữ liệu mới được cập nhật lên CSDL.

Ngoài ra nếu bạn muốn vừa tạo đối tượng vừa cập nhật thẳng lên CSDL luôn thì dùng phương thức create() trong đối tượng objects, đối tượng này là một đối tượng tĩnh do Django tạo ra cho chúng ta để đơn giản hóa việc thao tác với CSDL.

>>> Blog.objects.create(name='J.R.R Blog', tagline='All the latest J.R.R news.')

## Cập nhật đối tượng

Bạn có thể thay đổi giá trị của các đối tượng trong Python rồi chỉ cần gọi phương thức save() là Django sẽ cập nhật mới trong CSDL.

>>> b.name = 'New name'

>>> b.save()

### Cập nhật các thuộc tính khóa ngoại

Việc gán giá trị cho các thuộc tính khóa ngoại **many-to-one** (ForeignKey) rất đơn giản, bạn chỉ cần gán thuộc tính khóa ngoại với đối tượng cần trỏ đến là xong.

Ví dụ:

>>> from django.utils import timezone

>>> e = Entry(blog=b,

... headline='The lord of the rings',

... body\_text='Chapter 0',

... pub\_date=timezone.now(),

... mod\_date=timezone.now(),

... n\_comments=0,

... n\_pingbacks=0,

... rating=0)

>>> e.save()

Đối với các thuộc tính khóa ngoại **many-to-many** thì chúng ta không được phép gán trực tiếp như đối với ForeignKey mà phải dùng phương thức riêng của thuộc tính khóa ngoại đó, bởi vì khóa ngoại ForeignKey chỉ lưu trữ các đối tượng đơn lẻ trong khi khóa ngoại ManyToManyField thì lại lưu trữ một danh sách các đối tượng khác nhau:

>>> e.authors.add(a)

>>> e.save()

## Truy xuất dữ liệu

Ngoài cách thao tác với dữ liệu thông qua từng bản ghi như trên thì Django còn cung cấp cho chúng ta một đối tượng thuộc lớp django.db.models.manager.Manager để chúng ta thao tác với chính bảng được tạo ra đó, đối tượng này mặc định được đặt tên là objects.

## Lấy toàn bộ dữ liệu

Khi chúng ta lấy dữ liệu từ CSDL về thì objects sẽ trả về một đối tượng QuerySet.

>>> Blog.objects.all()

>>> [<Blog: New name>]

## Lọc dữ liệu

Có 2 phương thức hỗ trợ lọc dữ liệu là filter() và exclude(). Tham số của 2 phương thức là một biểu thức tìm kiếm, trong đó phương thức filter() sẽ trả về dữ liệu khớp với biểu thức tìm kiếm, còn exclude() sẽ trả về dữ liệu không khớp.

>>> Entry.objects.filter(pub\_date\_\_year==2016)

## Lấy một dòng dữ liệu

Django cung cấp phương thức get() giúp bạn lấy một bản ghi duy nhất, tham số của phương thức này cũng là một biểu thức tìm kiếm như 2 phương thức filter() và exclude(). Bạn cũng chỉ dùng get() khi biết dữ liệu trả về chỉ có 1 bản ghi thôi, nếu có nhiều bản ghi khớp với biểu thức tìm kiếm thì get() sẽ báo lỗi exception MultipleObjectsReturned.

>>> one\_entry = Entry.objects.get(pk=1)

Một sự khác nhau nữa giữa get() và filter()/exclude() là nếu get() không tìm thấy bản ghi nào thì sẽ trả về một lỗi exception là DoesNotExist còn 2 phương thức kia sẽ trả về một đối tượng QuerySet rỗng.

Ngoài ra:

Entry.objects.filter(pub\_date\_\_year=2005).order\_by('-pub\_date', 'headline')

Author.objects.distinct()

Blog.objects.filter(name\_\_startswith='Beatles').**values**()

[{'id': 1, 'name': 'Beatles Blog', 'tagline': 'All the latest Beatles news.'}]

Author.objects.values\_list('name', 'entry\_\_headline')

[('Noam Chomsky', 'Impressions of Gaza'),

('George Orwell', 'Why Socialists Do Not Believe in Fun'),

('George Orwell', 'In Defence of English Cooking'),

('Don Quixote', None)]

**>>>** Entry.objects.values\_list('headline', flat=**True**).get(pk=1)

'First entry'

Entry.objects.dates('pub\_date', 'year')

[datetime.date(2005, 1, 1)]

**>>>** Entry.objects.dates('pub\_date', 'month')

[datetime.date(2005, 2, 1), datetime.date(2005, 3, 1)]

**>>>** Entry.objects.dates('pub\_date', 'day')

[datetime.date(2005, 2, 20), datetime.date(2005, 3, 20)]

## Lọc số lượng bản ghi

Bạn có thể lọc số lượng bản ghi cần lấy theo cú pháp của Python.

Ví dụ:

>>> Entry.objects.all()[:5]     # lấy 5 phần tử đầu tiên

>>> Entry.objects.all()[5:10]   # lấy các phần tử từ vị trí 5 đến 10

## Biểu thức tìm kiếm

Các biểu thức tìm kiếm sẽ thực hiện câu truy vấn SQL WHERE trong CSDL. Biểu thức tìm kiếm là tham số cho các phương thức lọc dữ liệu filter(), exclude() và get().

Cú pháp lọc có dạng **<tên thuộc tính>\_\_<kiểu tìm kiếm>=<giá trị>** (lưu ý ở đây có 2 dấu gạch dưới).

Ví dụ:

>>> Entry.objects.filter(pub\_date\_\_lte='2016-01-01')

Trong biểu thức tìm kiếm thì **<tên thuộc tính>** bao giờ cũng là tên do chúng ta đặt khi định nghĩa model, ngoại trừ thuộc tính khóa ngoại phải thêm vào \_id ở cuối tên. Ví dụ:

>>> Entry.objects.filter(blog\_id=4)

- Một số cú pháp tìm kiếm thông dụng:

* exact: dữ liệu tìm được phải giống chính xác với giá trị cần tìm. Vd Entry.objects.get(headline\_\_exact='Cat bites dog') chỉ trả về những bản ghi có headline là *“Cat bites dog*“.
* iexact: dữ liệu tìm được chỉ cần giống kí tự với giá trị cần tìm, không phân biệt chữ HOA-thường. Vd Blog.objects.get(name\_\_iexact='beatles blog'), những bản ghi có headline là *“Beatles Blog”*, *“beatles blog”* hoặc *“BeAtlES blOG”*đều đ*ược nhận*.
* contains: dữ liệu chỉ cần chứa chuỗi giá trị là được. Vd Entry.objects.get(headlines\_\_contains='Lennon'). Giống LIKE tròn SQL Ngoài ra còn có icontains cũng có chức năng như contains nhưng không phân biệt chữ HOA-thường (giống ILIKE trong SQL).
* **startswith, endswith**:  dữ liệu có chuỗi bắt đầu hoặc kết thúc giống với giá trị.
* **istartswith, iendswith**:  có chức năng tương tự như startswith nhưng không phân biệt chữ hoa thường.
* in:  dữ liệu tồn tại trong khoảng. Ví dụ: Entry.objects.filter(id\_\_in=[1, 3, 4])
* gt: lớn hơn
* gte: lớn hơn hoặc bằng
* lt: nhỏ hơn
* lte: nhỏ hơn hoặc bằng

Ví dụ:

Entry.objects.filter(id\_\_gt=4) ⬄ SELECT ... WHERE id > 4;

* range: kiểm tra phạm vi ví dụ:

Entry.objects.filter(pub\_date\_\_range=(start\_date, end\_date))

SQL: SELECT ... WHERE pub\_date BETWEEN '2005-01-01' **and** '2005-03-31'

* date: kiểm tra các giá trị dạng ngày tháng năm

Entry.objects.filter(pub\_date\_\_date=datetime.date(2005, 1, 1))

Entry.objects.filter(pub\_date\_\_date\_\_gt=datetime.date(2005, 1, 1))

* year: kiểm tra các giá trị năm

ví dụ:

Entry.objects.filter(pub\_date\_\_year=2005)

Entry.objects.filter(pub\_date\_\_year\_\_gte=2005)

SQL tương đương

SELECT ... WHERE pub\_date BETWEEN '2005-01-01' AND '2005-12-31';

SELECT ... WHERE pub\_date >= '2005-01-01';

* iso\_year: kiểm tra các giá trị năm (Giá trị là một năm nguyên)
* month: kiểm tra các giá trị tháng

ví dụ:

Entry.objects.filter(pub\_date\_\_month=12)

Entry.objects.filter(pub\_date\_\_month\_\_gte=6)

Sql tương đương:

SELECT ... WHERE EXTRACT('month' FROM pub\_date) = '12';

SELECT ... WHERE EXTRACT('month' FROM pub\_date) >= '6';

* day: kiểm tra các giá trị ngày
* week: Giá trị tuần

ví dụ:

Entry.objects.filter(pub\_date\_\_week=52)

Entry.objects.filter(pub\_date\_\_week\_\_gte=32, pub\_date\_\_week\_\_lte=38)

* week\_day: Giá trị ngày trong tuần từ 1 (Sunday) đến 7 (Saturday).

Ví du:

Entry.objects.filter(pub\_date\_\_week\_day=2)

Entry.objects.filter(pub\_date\_\_week\_day\_\_gte=2)

* quarter: Giá trị một phần tư của năm
* time: Giá trị thời gian

ví dụ:

Entry.objects.filter(pub\_date\_\_time=datetime.time(14, 30))

Entry.objects.filter(pub\_date\_\_time\_\_range=(datetime.time(8), datetime.time(17)))

* hour: Giá trị thời gian (giờ) có 24 giờ từ 0-23
* minute: Giá trị thời gian (phút) có 60 phút từ 0-59
* second: Giá trị thời gian (giây) có 60 giây từ 0-59
* isnull: Trả về giá trị True hoặc False
* regex: Biểu thức chính quy

ví dụ:

Entry.objects.get(title\_\_regex=r'^(An?|The) +')

SQL tương đương:

SELECT ... WHERE title REGEXP BINARY '^(An?|The) +'; -- MySQL

SELECT ... WHERE REGEXP\_LIKE(title, '^(An?|The) +', 'c'); -- Oracle

SELECT ... WHERE title ~ '^(An?|The) +'; -- PostgreSQL

SELECT ... WHERE title REGEXP '^(An?|The) +'; -- SQLite

* regex: Biểu thức chính quy không phân biệt chữ hoa thường

## Tìm kiếm đa bảng

Các bảng trong CSDL có quan hệ với nhau nhờ vào các khóa ngoại, khi truy vấn dữ liệu, SQL cho phép bạn truy vấn các bản ghi có liên quan với nhau bằng cách nối các bảng lại bằng câu lệnh JOIN, Django cũng cho phép bạn làm điều đó, bạn chỉ cần đưa vào biểu thức tìm kiếm cú pháp **<tên khóa ngoại>\_\_<tên thuộc tính của bảng khác>=<giá trị>**. Ví dụ:

|  |  |
| --- | --- |
| 1 | >>> Entry.objects.filter(blog\_\_name='Beatles Blog') |

Câu lệnh trên sẽ tìm các bản ghi Entry có quan hệ với bảng Blog với name là “Beatles Blog”.

# **Django – Form có sẵn của Django**

## Tạo Form

Trong thư mục user-auth chúng ta tạo một file mới với tên là forms.py để lưu các lớp form.

Ví dụ:

from django import forms

class RegisterForm(forms.Form):

    username = forms.CharField(label='Username', max\_length=100)

    password = forms.CharField(widget=forms.PasswordInput)

    email = forms.EmailField(label='Email')

Các lớp dùng để tạo form được kế thừa từ lớp django.forms.Form.

Bên trong lớp này chúng ta cũng khai báo các trường là các đối tượng Field, nhưng các đối tượng Field này không giống như Field khi tạo model, Field ở đây là để tạo form HTML còn field bên model là để tạo bảng CSDL

Vì HTML có nhiều thẻ element có công dụng chung nhưng lại hiển thị dữ liệu khác nhau, chẳng hạn như một thẻ <input> có thể dùng để nhập tên, số điện thoại, password… do đó Django cung cấp cho mỗi đối tượng Field một đối tượng Widget để chúng ta có thể chỉ định loại text nào hiển thị cái gì.  label của CharField sẽ được dùng cho thuộc tính name trong thẻ <label>...

## Tạo Template

<form action="" method="post">

    {% csrf\_token %}

    {{ form }}

    <input type="submit" value="Submit" />

</form>

Khi chúng ta viết các lớp form, Django sẽ nhìn các trường mà chúng ta khai báo rồi sinh ra các thẻ <input> tương ứng, ngoại trừ cặp thẻ <form>...</form> và thẻ <input type="submit"/>

<form action="" method="post">

Thuộc tính action sẽ chuyển dữ liệu tới đường dẫn tương ứng, ở đây mình để trống tức là gửi lại cho chính trang đó. Phương thức POST sẽ mã hóa dữ liệu được gửi đi.

{{ form }}

Chúng ta có thể tham chiếu đến phần tử form trong list context mà chúng ta sẽ khai báo trong các hàm view

## Tạo View

from django.shortcuts import render

# Create your views here.

from django.http import HttpResponse

from .forms import RegisterForm

def register(request):

    if request.method == 'POST':

        response = HttpResponse()

        response.write("<h1>Thanks for registering</h1></br>")

        response.write("Your username: " + request.POST['username'] + "</br>")

        response.write("Your email: " + request.POST['email'] + "</br>")

        return response

    registerForm = RegisterForm()

    return render(request, 'user\_auth/register.html', {'form':registerForm})

**ten form.cleaned\_data[‘ten fied’]: Phương thức chuẩn hóa các trường trong form rồi sau đó đưa vào trong một dictionarry**

# **Django – Upload file**

## Tạo form upload file

Chúng ta tạo một app mới với tên là file\_uploader:

C:\Project\mysite>python manage.py startapp file\_uploader

Chúng ta khai báo app cho project.

INSTALLED\_APPS = [

    #...

    'file\_uploader',

    #...

]

Kế tiếp chúng ta tạo file forms.py bên trong thư mục file\_uploader:

file\_uploader/forms.py

from django import forms

class UploadFileForm(forms.Form):

    title = forms.CharField(max\_length=50)

    file = forms.FileField()

Để hiển thị thẻ element chọn file thì chúng ta dùng lớp FileField().

Tiếp theo chúng ta tạo thư mục templates và file template:

file\_uploader/templates/fileUploaderTemplate.html

<form action="" method="POST" enctype="multipart/form-data">

    {% csrf\_token %}

    <table>

        {{ form.as\_table }}

    </table>

    <input type="submit" value="Submit" />

</form>

Để gửi file lên thì trong thẻ <form> chúng ta phải khai báo thuộc tính enctype="multipart/form-data".

Sau khi đã có template và form để hiển thị thì chúng ta tiến hành tạo view để kết nối chúng với nhau.

from django.db import models

# Create your models here.

from django.http import HttpResponse

from .forms import UploadFileForm

def fileUploaderView(request):

    if request.method == 'POST':

        form = UploadFileForm(request.POST, request.FILES)

        if form.is\_valid():

            upload(request.FILES['file'])

            return HttpResponse("<h2>File uploaded successful!</h2>")

        else:

            return HttpResponse("<h2>File uploaded not successful!</h2>")

    form = UploadFileForm()

    return render(request, 'fileUploaderTemplate.html', {'form':form})

def upload(f):

    file = open(f.name, 'wb+')

    for chunk in f.chunks():

        file.write(chunk)

Phương thức is\_valid() có trong lớp django.forms.Form sẽ kiểm tra tính hợp lệ của dữ liệu được gửi lên. Mục đích chính của lớp Form trong Django chính là hỗ trợ chúng ta kiểm tra sự đúng đắn của dữ liệu được gửi lên. Chẳng hạn như khi bạn xây dựng chức năng up ảnh avatar thì bạn chỉ cho phép user gửi các file ảnh lên thôi, nhưng kẻ tấn công có thể gửi các file ảnh “giả” lên mà trình duyệt không biết, do đó khi dữ liệu được gửi tới server, chúng ta nên kiểm tra trước các dữ liệu này trước khi lưu vào hệ thống.

Ở đây phương thức is\_valid() chỉ có chức năng đơn giản là kiểm tra xem ô text title có dữ liệu hay không vì mặc định các field của Form trong Django bắt buộc phải có dữ liệu mới được nhận.

upload(request.FILES['files'])

return HttpResponse("<h2>File uploaded successful!</h2>")

Nếu dữ liệu là hợp lệ thì chúng ta tiến hành lưu file vào thư mục gốc của server và trả lời thông báo gửi thành công về cho người dùng. Ở đây ta lấy thuộc tính “files” trong tham số FILES của đối tượng request

def upload(f):

    file = open(f.name, 'wb+')

    for chunk in f.chunks():

        file.write(chunk)

Hàm upload() sẽ thực hiện copy file vào thư mục gốc của server (thư mục có file manage.py).

Hàm sẽ tạo một file có tên và định dạng giống file trong request.FILES['file']) sau đó mở file rồi ghi dữ liệu chunk vào file vừa tạo theo kiểu dữ liệu của file đó.

Tóm tắt quy trình tạo file upload

- Tạo form của django trong đó có 1 trường thuộc kiểu

forms. fileField

- Tạo tempaltes để hiển thị với form action=”” method=”post”

- lấy đối tượng request(FILES[“files”]) ra để sử dụng

- Hàm upload với tham số là đối tượng request(FILES[“files”]) ban đầu sẽ tạo một file có kiểu và tên giống với thuộc tính name trong request(FILES[“files”].name)

files = open(f.name,'wb+')

Sau đó sẽ thực hiện vòng lặp với giá trị chunk và ghi vào file

for chunk in f.chunks():

        files.write(chunk)

# **Tạo form từ model**

## Kiểu dữ liệu

Mỗi lớp Field của Model sẽ có một lớp Field của Form tương ứng. Dưới đây là bảng tên các lớp Field tương ứng của Model và Form.

| **MODEL FIELD** | **FORM FIELD** |
| --- | --- |
| [AutoField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.AutoField) | Không có |
| [BigIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.BigIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) với min\_value là -9223372036854775808 và  max\_value là 9223372036854775807. |
| [BooleanField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.BooleanField) | [BooleanField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.BooleanField) |
| [CharField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.CharField) | [CharField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.CharField) với max\_length được thiết lập trong thuộc tính max\_length của model |
| [CommaSeparatedIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.CommaSeparatedIntegerField) | [CharField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.CharField) |
| [DateField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.DateField) | [DateField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.DateField) |
| [DateTimeField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.DateTimeField) | [DateTimeField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.DateTimeField) |
| [DecimalField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.DecimalField) | [DecimalField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.DecimalField) |
| [EmailField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.EmailField) | [EmailField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.EmailField) |
| [FileField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.FileField) | [FileField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.FileField) |
| [FilePathField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.FilePathField) | [FilePathField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.FilePathField) |
| [FloatField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.FloatField) | [FloatField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.FloatField) |
| [ForeignKey](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.ForeignKey) | [ModelChoiceField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.ModelChoiceField) |
| ImageField | [ImageField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.ImageField) |
| [IntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.IntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| IPAddressField | IPAddressField |
| [GenericIPAddressField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.GenericIPAddressField) | [GenericIPAddressField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.GenericIPAddressField) |
| [ManyToManyField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.ManyToManyField) | [ModelMultipleChoiceField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.ModelMultipleChoiceField) |
| [NullBooleanField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.NullBooleanField) | [NullBooleanField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.NullBooleanField) |
| [PositiveIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.PositiveIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| [PositiveSmallIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.PositiveSmallIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| [SlugField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.SlugField) | [SlugField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.SlugField) |
| [SmallIntegerField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.SmallIntegerField) | [IntegerField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.IntegerField) |
| [TextField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.TextField) | [CharField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.CharField) với widget=forms.Textarea |
| [TimeField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.TimeField) | [TimeField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.TimeField) |
| [URLField](https://docs.djangoproject.com/en/1.9/ref/models/fields/#django.db.models.URLField) | [URLField](https://docs.djangoproject.com/en/1.9/ref/forms/fields/#django.forms.URLField) |

Lưu ý là các lớp ModelField hỗ trợ 3 kiểu khóa ngoại là **1-n**, **n-n** và **1-1** nhưng bên FormField chỉ có 2 kiểu duy nhất được hỗ trợ là ModelChoiceField tương ứng với ForeignKey và ModelMultipleChoiceField tương ứng với ManyToManyField. Cả 2 lớp trên đều nhận một tham số khi khởi tạo là một đối tượng QuerySet.

Mỗi lớp *Form* *Field* có chung một số thuộc tính như sau:

* required: nếu thuộc tính blank của Model là True thì required = False và ngược lại.
* label: thuộc tính này được gán tự động bằng thuộc tính verbose\_name của Model và kí tự đầu được viết hoa.
* help\_text: thuộc tính này được gán tự động bằng thuộc tính help\_text của model.
* required: nếu thuộc tính blank của Model là True thì required = False và ngược lại.
* label: thuộc tính này được gán tự động bằng thuộc tính verbose\_name của Model và kí tự đầu được viết hoa.
* help\_text: thuộc tính này được gán tự động bằng thuộc tính help\_text của model.

## Ví dụ

from django.db import models

from django.forms import ModelForm

TITLE\_CHOICES = (

    ('MR', 'Mr.'),

    ('MRS', 'Mrs.'),

    ('MS', 'Ms.'),

)

class Author(models.Model):

    name = models.CharField(max\_length=100)

    title = models.CharField(max\_length=3, choices=TITLE\_CHOICES)

    birth\_date = models.DateField(blank=True, null=True)

class Book(models.Model):

    name = models.CharField(max\_length=100)

    authors = models.ManyToManyField(Author)

class AuthorForm(ModelForm):

    class Meta:

        model = Author

        fields = ['name', 'title', 'birth\_date']

class BookForm(ModelForm):

    class Meta:

        model = Book

        fields = ['name', 'authors']

Chúng ta định nghĩa 2 lớp Model là Author và Book, sau đó định nghĩa 2 lớp Form có các Fieldđược tạo tự động từ 2 lớp model là AuthorForm và BookForm. Các lớp này được kế thừa từ lớp django.forms.ModelForm thay thì lớp django.forms.Form như trước.

class Meta:

    model = Author

    fields = ['name', 'title', 'birth\_date']

Chúng ta định nghĩa model được dùng trong lớp nội Meta, thuộc tính model sẽ tạo các field tương ứng từ lớp Model tương ứng, thuộc tính fields sẽ chọn các trường nào được dùng, nếu chúng ta không khai báo thuộc tính fields thì mặc định Django sẽ dùng tất cả các thuộc tính có trong lớp Model, ngoại trừ thuộc tính id nếu bạn không khai báo thuộc tính khóa chính. Ngoài ra lớp Meta còn có thuộc tính exclude, thuộc tính này trái ngược với thuộc tính fields, tức là thuộc tính này sẽ quy định các trường nào không được phép sử dụng.

## Lưu dữ liệu

Lớp ModelForm có một phương thức tên là save(), phương thức này tạo mới hoặc lưu một đối tượng Model vào cơ sở dữ liệu giống như phương thức save() bên các lớp *Model* vậy.

Khi khởi tạo các đối tượng ModelForm thì chúng ta có thể truyền các đối tượng model có sẵn vào hàm khởi tạo, nếu chúng ta chỉ truyền vào không thôi thì Django sẽ tạo mới một đối tượng trên CSDL, nếu chúng ta truyền vào và ghi rõ truyền với tham số là instance thì Django sẽ cập nhật dữ liệu trong đối tượng đó. Ví dụ:

>>> from myapp.models import Article

>>> from myapp.forms import ArticleForm

>>> f = ArticleForm(request.POST)

>>> new\_article = f.save()

Đoạn code trên tạo đối tượng mới từ đối tượng POST được gửi lên và lưu vào CSDL.

>>> a = Article.objects.get(pk=1)

>>> f = ArticleForm(request.POST, instance=a)

>>> f.save()

Lưu ý là nếu chúng ta không kiểm tra sự đúng đắn của dữ liệu thì khi gọi phương thức save(),Django cũng sẽ tự động làm việc đó và sẽ giải phóng lỗi exception ValueError nếu dữ liệu có lỗi, trong [bài trước](http://phocode.com/django/django-upload-file/) chúng ta đã dùng một phương thức để kiểm tra dữ liệu là is\_valid(), chúng ta sẽ tìm hiểu thêm về cách kiểm tra dữ liệu trong các bài sau.

# **Kiểm tra dữ liệu gửi lên form**

## Validator

# **Phân trang**

Django cung cấp lớp django.core.paginator hỗ trợ phân trang rất tốt.

Views:

from django.shortcuts import render

# Create your views here.

from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger

from .models import Customer

def listing(request):

    customer\_list = Customer.objects.all()

    paginator = Paginator(customer\_list, 5)

    pageNumber = request.GET.get('page')

    try:

        customers = paginator.page(pageNumber)

    except PageNotAnInteger:

        customers = paginator.page(1)

    except EmptyPage:

        customers = paginator.page(paginator.num\_pages)

    return render(request, 'list.html', {'customers':customers})

Chúng ta sử dụng lớp Pagination để thực hiện phân trang.

paginator = Paginator(customer\_list, 5)

Hàm khởi tạo Paginator() nhận vào 2 tham số, tham số đầu tiên là một đối tượng QuerySet,tham số thứ 2 là số tượng item trên mỗi “trang”. Trong ví dụ trên chúng ta đưa đối tượng customer\_list vào với số lượng 5 item mỗi trang.

pageNumber = request.GET.get('page')

URL của chúng ta có thêm tham số page là số thứ tự của trang muốn xem.

try:

    customers = paginator.page(pageNumber)

except PageNotAnInteger:

    customers = paginator.page(1)

except EmptyPage:

    customers = paginator.page(paginator.num\_pages)

Nếu tham số page không hợp lệ, chẳng hạn như page=abc thì Paginator sẽ giải phóng lỗi PageNotAnInterger, trong trường hợp này chúng ta trả về trang đầu tiên với phương thức Paginator.page(), hoặc nếu page nằm ngoài phạm vi trang cho phép, chẳng hạn như chúng ta chỉ có 4 trang nhưng tham số page=1000 thì Paginator sẽ giải phóng lỗi EmptyPage, ở đây chúng ta xử lý bằng cách trả về trang cuối cùng bằng thuộc tính num\_pages.

## Template

## Biến:

Biến là những thứ giống như {{ variable }}. Khi trình dịch template đọc đến một biến thì biến sẽ được thay thế bằng một giá trị thật (mà chúng ta truyền vào từ hàm render() trong các view). Biến chỉ được đặt tên bằng các kí tự chữ cái và dấu gạch dưới (\_).

Chúng ta dùng dấu chấm (.) để truy xuất các thuộc tính của biến.

## Bộ lọc – Filter

Django cung cấp các bộ lọc để hỗ trợ chúng ta hiển thị dữ liệu theo nhiều cách khác nhau.

Ví dụ {{ name|lower }}, trong đó lower là một bộ lọc, có tác dụng chuyển toàn bộ chữ cái thành chữ thường. Để dùng các bộ lọc thì chúng ta kèm theo dấu | và tên bộ lọc vào sau tên biến.

Chúng ta cũng có thể dùng nhiều bộ lọc cùng một lúc, các bộ lọc được thực hiện tuần tự từ trái sang phải, ví dụ {{ text|escape|linebreaks }} có tác dụng xuống dòng sau khi in dữ liệu.

Một số bộ lọc cần có cả tham số nữa, ví dụ như {{ bio|truncatewords:30 }} có nghĩa là lấy 30 từ đầu tiên của biến bio.

Nếu tham số của bộ lọc có khoảng trống thì chúng ta phải kẹp chúng trong cặp dấu nháy kép “”. Ví dụ {{ list|join:", " }} sẽ nối các item trong biến list thành một string, ngăn cách nhau bởi dấu phẩy và dấu cách.

Một số bộ lọc thường dùng:

* default: nếu biến không có giá trị hoặc giá trị rỗng thì thay thế bằng giá trị default. Ví dụ {{value|default:"nothing"}}
* length: trả về độ dài của dữ liệu, có thể áp dụng cho string và list. Ví dụ {{value|length}}
* filesizeformat: đổi kiểu số thành định dạng file, ví dụ {{value|filesizeformat}} sẽ chuyển con số 123456789 thành 117.7 MB.

## Thẻ – Tag

Thẻ có cú pháp {% tag %}. Thẻ thì phức tạp hơn biến một tí, có thể dùng để tạo chuỗi, thực hiện các luồng điều khiển hoặc load các thông tin khác vào template.

Có một số thẻ đi kèm với cả thẻ kết thúc nữa, ví dụ {% tag %} thì sẽ có {% endtag %}.

 Các thẻ thường dùng:

* for: duyệt qua một đối tượng danh sách. Ví dụ:

{% for athlete in athlete\_list %}

<li>{{ athlete.name }}</li>

{% endfor %}

* if, elif và else: kiểm tra một biến, nếu biến đúng thì thực hiện đoạn code bên trong.

{% if athlete\_list %}

    Number of athletes: {{ athlete\_list|length }}

{% elif athlete\_in\_locker\_room\_list %}

    Athletes should be out of the locker room soon!

{% else %}

    No athletes.

{% endif %}

Ngoài kiểm tra các biến thì bạn cũng có thể áp dụng bộ lọc vào biến khi dùng thẻ if:

{% if athlete\_list|length > 1 %}

    Team: {% for athlete in athlete\_list %} ... {% endfor %}

{% else %}

    Athlete: {{ athlete\_list.0.name }}

{% endif %}

・block và extends: kế thừa template, tức là dùng các file template khác.

## Bình luận – Comment

Bình luận được đặt trong cặp dấu {# #}, các đoạn code bên trong cặp dấu này sẽ không được thực thi

Nếu muốn bình luận trên nhiều dòng thì dùng thẻ [comment](https://docs.djangoproject.com/en/1.9/ref/templates/builtins/#std:templatetag-comment).

## Thừa kế template

Tính năng mạnh mẽ nhất và cũng là phức tạp nhất của Template trong Django là tính năng thừa kế. Tính năng thừa kế cho phép bạn xây dựng một bộ template tổng quát và các template con, trong đó template tổng quát sẽ chứa các template con.

Ví dụ:

<!DOCTYPE html>

<html lang="en">

<head>

    <link rel="stylesheet" href="style.css" />

    <title>{% block title %}My amazing site{% endblock %}</title>

</head>

<body>

    <div id="sidebar">

        {% block sidebar %}

        <ul>

            <li><a href="/">Home</a></li>

            <li><a href="/blog/">Blog</a></li>

        </ul>

        {% endblock %}

    </div>

    <div id="content">

        {% block content %}{% endblock %}

    </div>

</body>

</html>

Đoạn code trên là template thiết kế bộ khung của một trang web, cấu trúc của template này bao gồm 2 cột, nằm giữa các thẻ block. Nhiệm vụ của các template con là lấp đầy các khoảng trống của 2 cột đó.

Trong đoạn code trên có 3 thẻ block là title, content và sidebar, nhiệm vụ của thẻ block là báo cho Django biết đây là nơi mà các template con có thể override lại và chèn dữ liệu cần hiển thị vào đó.

Ví dụ về một template con:

{% extends "base.html" %}

{% block title %}My amazing blog{% endblock %}

{% block content %}

    {% for entry in blog\_entries %}

        <h2>{{ entry.title }}</h2>

        {{ entry.body }}

    {% endfor %}

{% endblock %}

Để một template con có thể override lại các thẻ block của template khác thì ở đầu template chúng ta khai báo thẻ extends với tên file template. Trong ví dụ trên, trình biên dịch Django sẽ đọc trong template cha và thấy các thẻ block trong template cha cũng có trong template con nên phần block trong template con sẽ được chèn vào trong template cha.

Một số lưu ý:

* Thẻ {% extends %} luôn được đặt trước tất cả các thẻ còn lại.
* Nên override từng thẻ block trong từng file template chứ không nên “ôm” tất cả vào trong một file.
* Thẻ block trong template con cũng có thể dùng lại nội dung của template cha, chỉ cần gọi {{block.super}}
* Thẻ {% endblock %} không cần phải có tên block theo sau nhưng chúng ta cũng nên đưa tên block vào để code dể đọc và dễ quản lý hơn. Ví dụ:

{% block content %}

...

{% endblock content %}

* Không được có 2 thẻ block có tên giống nhau.

## Tự động thoát HTML

Thoát HTML tức là trang web tự động chuyển đổi các kí tự đặc biệt trong HTML sang một dạng mã, dùng để bảo vệ website.

Ví dụ chúng ta có đoạn code template như sau:

Hello, {{ name }}

Thoạt nhìn thì có vẻ đơn giản, chúng ta có thể yêu cầu người dùng nhập vào một textbox rồi lưu vào biến name, sau đó in nội dung trong biến name ra thôi.

Nhưng nếu người dùng không nhập vào biến name một đoạn chuỗi bình thường mà là đoạn chuỗi kì lạ như:

<script>alert('hello')</script>

Lúc này Django sẽ dịch đoạn template sang đoạn code HTML như sau:

Hello, <script>alert('hello')</script>

Trang web sẽ hiển thị một hộp thoại thông báo. Đó chỉ là trường hợp đơn giản, trong thực tế hacker có thể lợi dụng lỗ hổng này để khai thác nhiều thứ hơn nữa. Đây gọi là kỹ thuật tấn công Cross Site Scripting (XSS).

May mắn là mặc định trình dịch Template của Django tự động “thoát” (auto-escape) các kí tự đặc biệt, tức là chuyển đổi những kí tự sau đây thành những kí tự mã khác:

* Dấu < chuyển thành &lt;
* Dấu > chuyển thành &gt;
* Dấu nháy đơn ' chuyển thành &#39;
* Dấu nháy kép " chuyển thành &quot;
* Dấu & chuyển thành &amp;

Vì tính năng tự động thoát này mà bạn không cần phải lo đến vấn đề bảo mật XSS nữa.

Để tắt “thoát” trên từng biến: chúng ta dùng bộ lọc safe:

This will be escaped: {{ data }}

This will not be escaped: {{ data|safe }}

Tắt “thoát” trên template: chúng ta đặt nội dung file template hoặc một phần nào đó của template trong cặp thẻ autoescape:

{% autoescape off %}

    Hello {{ name }}

{% endautoescape %}

Thẻ autoescape nhận tham số on hoặc off tương ứng với bật và tắt. Bạn cũng có thể lồng các cặp thẻ autoescape vào nhau. khi kế thừa template thì nếu template cha tắt “thoát” thì các template con cũng sẽ tự tắt tính năng này, nếu muốn bật tính năng này thì template con phải override lại.

## Gọi phương thức

Bạn không chỉ có thể truy xuất dữ liệu từ các thuộc tính trong các biến mà còn có thể gọi phương thức của chúng nữa, tất nhiên là bạn chỉ có thể gọi các phương thức có trả về dữ liệu để hiển thị chứ không thể gọi các phương thức thực hiện tính toán mà không trả về thứ gì được.

Ví dụ, các đối tượng Queryset có phương thức count() trả về số lượng phần tử của nó, chúng ta có thể gọi phương thức này như sau:

{{ task.comment\_set.all.count }}

Bạn cũng có thể gọi các phương thức do bạn tự định nghĩa:

Đáng tiếc là bạn không thể truyền tham số vào các lời gọi hàm trong Template vì mục đích chính của template cũng chỉ là hiển thị dữ liệu chứ không phải tính toán, do đó bạn chỉ có thể gọi các phương thức không có tham số.

# **Gửi Email**

django.core.mail

## Hàm send\_mail()

Cú pháp:

**send\_mail(subject, message, from\_email, recipient\_list, fail\_silently=False, auth\_user=None, auth\_password=None, connection=None, html\_message=None)**

Trong đó subject, message, from\_email và recipient\_list là bắt buộc phải có.

* subject: tiêu đề mail.
* message: nội dung mail gửi đi.
* from\_email: địa chỉ mail dùng để gửi.
* recipient\_list: danh sách địa chỉ mail gửi tới.
* fail\_silently: nếu là True thì Django sẽ giải phóng lỗi smtplib.SMTPException nếu mail không gửi được, mặc định False.
* auth\_user: địa chỉ mail dùng để gửi, không cần đưa vào nếu đã thiết lập trong file settings.py.
* auth\_password: mật khẩu của địa chỉ mail dùng để gửi, không cần đưa vào nếu đã thiết lập trong file settings.py.
* connection: tên đối tượng mail backend xử lý việc gửi mail, nếu không truyền vào thì Django sẽ tự động tạo một đối tượng mặc định, thường chúng ta cũng không quan tâm đến tham số này.
* html\_message: nội dung mail gửi đi dưới dạng HTML.

Hàm này sẽ trả về 1 nếu mail gửi thành công và 0 nếu thất bại.

Ví dụ:

from django.core.mail import send\_mail

send\_mail('Subject', 'Message',

          'from@example.com',

          ['to@example.com'],

          fail\_silently=False)

Các thiết lập của mail được lưu trong file settings.py, trong file này chúng ta khai báo các biến sau:

* EMAIL\_HOST: tên máy chủ mail, ví dụ smtp.google.com
* EMAIL\_PORT: số port của máy chủ mail, ví dụ 587
* EMAIL\_HOST\_USER: địa chỉ email dùng để gởi đi, ví dụ from@example.com
* EMAIL\_HOST\_PASSWORD: mật khẩu đăng nhập email dùng để gởi đi.
* EMAIL\_USE\_TLS và EMAIL\_USE\_SSL: True nếu muốn dùng các giao thức bảo mật SSL/TLS

## Hàm send\_mass\_mail()

Hàm send\_mass\_mail() đơn giản là dùng để gửi một lúc nhiều mail.

Cú pháp:

**send\_mass\_mail(datatuple, fail\_silently=False, auth\_user=None, auth\_password=None, connection=None)**

Các tham số cũng giống như trong hàm send\_mail() ngoại trừ tham số datatuple, tham số này nhận một đối tượng tuple, mỗi phần tử lại là một tuple khác lưu những thông tin về mail được gửi đi có dạng như sau:

(subject, message, from\_email, recipient\_list)

Ví dụ:

message1 = ('Subject 1',

            'Message 1',

            'from@example.com',

            ['first@example.com', 'other@example.com'])

message2 = ('Subject 2',

            'Message 2',

            'from@example.com',

            ['second@test.com'])

send\_mass\_mail((message1, message2), fail\_silently=False)

Hàm send\_mass\_mail() cũng trả về 0 hoặc 1 tương ứng với thành công hoặc thất bại.

Sự khác nhau giữa send\_mail() và send\_mass\_mail() là send\_mail() khi gửi mail nào thì phải mở và đóng kết nối tới server SMTP, trong khi send\_mass\_mail() chỉ cần mở một kết nối rồi gửi tất cả luôn, do đó send\_mass\_mail() hiệu quả hơn send\_mail().

## Hàm mail\_admins()

Hàm này có tác dụng gửi mail cho ban quản trị website, mail của quản trị website (admin) được liệt kê trong biến ADMINS trong file settings.py dưới dạng:

settings.py

ADMINS = [('John', 'john@example.com'), ('Mary', 'mary@example.com')]

**mail\_admins(subject, message, fail\_silently=False, connection=None, html\_message=None)**

Lưu ý: đoạn chuỗi được lưu trong biến EMAIL\_SUBJECT\_PREFIX trong file settings.py sẽ được chèn vào trước tham số subject, mặc định biến này có giá trị *” [Django] “*.

Các thông tin khác như server, port, mail người gửi sẽ được dùng trong file settings.py.

## Hàm mail\_managers()

Hàm này có công dụng giống như hàm mail\_admins(), chỉ khác là gửi cho danh sách mail trong biến MANAGERS trong file settings.py.

**mail\_managers(subject, message, fail\_silently=False, connection=None, html\_message=None)**

## Lớp EmailMessage

Các hàm send\_mail() hay send\_mass\_mail() chỉ là các hàm cấp cao hỗ trợ chúng ta gửi mail cho dễ dàng, thực chất việc gửi mail được thực hiện qua lớp EmailMessage.

Cũng chính vì thế mà có nhiều tính năng có trong lớp EmailMessagemà chúng ta không sử dụng được với 2 hàm trên chẳng hạn như BCC, gửi file đính kèm, gửi nội dung đa phương tiện…

Bản chất thì lớp EmailMessage chỉ làm công việc là tạo nội dung email sẽ được gửi đi, còn phần gửi được thực hiện bởi email-backend phía dưới nữa. Lớp EmailMessage cũng chỉ hỗ trợ gửi từng mail đơn lẻ với phương thức send(). Để gửi nhiều mail cùng một lúc thì chúng ta phải can thiệp vào backend.

Lớp EmailMessage có các thuộc tính và cũng là tham số trong hàm khởi tạo như sau:

* subject: tiêu đề mail
* body: nội dung mail
* from\_email: địa chỉ mail gửi đi
* to: danh sách các mail gửi tới
* bcc: danh sách các địa chỉ được dùng trong Bcc
* connection: đối tượng backend, nếu chúng ta không khai báo thì Django sẽ tự tạo một đối tượng mặc định
* attachment: danh sách các file đính kèm, chúng ta có thể dùng lớp email.MIMEBase.MIMEBasehoặc khai báo theo dạng (filename, content, mimetype).
* headers: một đối tượng dictionary dùng cho header của mail.
* cc: list hoặc tuple các địa chỉ mail dùng cho Carbon Copy (Cc).
* reply\_to: list hoặc tuple địa chỉ mail dùng khi trả lời mail.

Ví dụ:

from django.core.mail import EmailMessage

email = EmailMessage('Hello',

                     'Body',

                     'from@example.com',

                     ['to1@example.com', 'to2@example.com'],

                     ['bcc@example.com'],

                     reply\_to=['another@example.com'],

                     headers={'Message-ID': 'foo'})

Lớp EmailMessage có các phương thức sau đây:

* send(fail\_silently=False) sẽ gửi mail đi. Tham số fail\_silently sẽ giải phóng lỗi exception nếu là True, ngược lại (và mặc định) là False.
* message() khởi tạo một đối tượng django.core.mail.SafeMIMEText hoặc django.core.mail.Safe.MIMEMultipart, đây là các lớp kế thừa từ lớp email.MIMEText.MIMETexttrong Python, có nhiệm vụ lưu thông tin về nội dung mail được gửi đi.
* recipients() trả về các danh sách địa chỉ mail có trong đối tượng EmailMessage, bất kể là to, bcc, hay cc...
* attach(): gửi file đính kèm, chúng ta có thể truyền vào một đối tượng email.MIMEBase.MIMEBase hoặc một tuple có dạng (filename, content, mimetype). Ví dụ:

message.attach('design.png', img\_data, 'image/png')

* attach\_file() cũng gửi file đính kèm với tham số là đường dẫn đến file trong máy. Ví dụ:

message.attach\_file('images/weather\_map.png')

# **Cache**

Website được tạo ra ngày nay là website động, tức là nội dung HTML sẽ được server sinh ra rồi trả về cho người dùng mỗi khi người dùng gửi request đến, khác với website tĩnh là các file HTML đã có sẵn, người dùng request thì chỉ cần trả về file HTML đó thôi. Điều này cũng có nghĩa là website động sẽ tốn nhiều thời gian trả lời hơn so với website tĩnh, và khi lượt truy cập website càng nhiều thì thời gian này càng tăng lên gấp nhiều lần.

Kỹ thuật cache ra đời là để cắt giảm quá trình tính toán của website để có thể cung cấp nội dung cho người dùng một cách nhanh chóng hơn, cache có nhiều loại nhưng nhìn chung thì đều tuân theo thuật toán sau đây:

**user gửi request một trang web, tìm xem trang đó đã có cache hay chưa**  
**nếu đã có cache:**  
**trả về cache**  
**ngược lại:**  
**tạo cache**  
**lưu lại trang cache vừa tạo**  
**trả về trang cache vừa tạo**

## Thiết lập Cache

Việc thiết lập cache trong Django rất đơn giản, chỉ là cho Django biết cache sẽ được lưu ở đâu thôi bởi vì cache lưu trên RAM sẽ có hiệu suất khác hẳn so với lưu trên file.

Tất cả các thông tin cài đặt cache đều được lưu trong biến CACHE trong file settings.py, mặc định khi tạo project thì thông tin này chưa có, chúng ta phải tự thêm vào.

#### Lưu cache trên RAM – Memcached

Memcached đúng với cái tên của nó, là lưu nội dung trên bộ nhớ RAM của máy chủ, do đó loại cache này có tốc độ tìm kiếm cũng như trả về nhanh nhất, thích hợp cho các website lớn có lượng truy cập cao, các website như Facebook hay Wikipedia đều dùng loại cache này.

Memcached là một chương trình dạng dịch vụ (tức là chạy ngầm bên dưới hệ điều hành) được cấp một lượng RAM nhất định, cung cấp các hàm cho phép cập nhật dữ liệu trên cache, không đụng chạm gì tới đĩa cứng hoặc cơ sở dữ liệu.

Để thiết lập memcached thì chúng ta cung cấp những thông tin sau:

* BACKEND: django.core.cache.backends.memcached.MemcachedCache hoặc django.core.cache.backends.memcached.PyLibMCCache.
* LOCATION: theo cú pháp IP:PORT, trong đó IP là địa chỉ máy lưu cache, port là cổng tương ứng của trình dịch vụ cache.

CACHES = {

    'default': {

    'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',

    'LOCATION': '127.0.0.1:11211',

    }

}

Đoạn code trên thiết lập memcache tại máy localhost, tức là lưu trên chính server đó trên port 11211.

Chúng ta cũng có thể cho chạy nhiều trình memcached trên nhiều máy:

CACHES = {

    'default': {

        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',

        'LOCATION': [

        '172.19.26.240:11211',

        '172.19.26.242:11211',

        ]

    }

}

Chỉ cần khai báo thêm IP và Port, ngăn cách nhau bởi dấu phẩy là được, port có thể khác nhau chứ không cần phải giống nhau.

#### Lưu cache trên cơ sở dữ liệu

Loại này sẽ lưu dữ liệu trong một bảng trên CSDL. Để sử dụng thì chúng ta thiết lập như sau:

* BACKEND: django.core.cache.backends.db.DatabaseCache
* LOCATION: tên bảng được dùng để lưu cache trong CSDL, tất nhiên phải là bảng trắng, chưa có gì trong đó

CACHES = {

    'default': {

    'BACKEND': 'django.core.cache.backends.db.DatabaseCache',

    'LOCATION': 'my\_cache\_table',

    }

}

Sau khi đã khai báo trong file settings.py thì chúng ta phải tạo bảng lưu cache bằng cách chạy lệnh:

python manage.py createcachetable

Django sẽ tự động tạo bảng với tên tương ứng trong biến LOCATION cùng các trường cần thiết để lưu cache

#### Lưu cache trong file

Loại cache này sẽ lưu dữ liệu trong file, khi nào cần thì sẽ đọc file. Chúng ta thiết lập như sau:

* BACKEND: django.core.cache.backends.filebased.FileBasedCache
* LOCATION: đường dẫn đến file

CACHES = {

    'default': {

    'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',

    'LOCATION': 'c:/cache.txt',

    }

}

Đường dẫn file phải là đường dẫn tuyệt đối – tức là phải có tên ổ đĩa cứng.

#### Các tham số khác

Ngoài 2 tham số bắt buộc là BACKEND và LOCATION thì khi thiết lập cache chúng ta còn có các tham số tùy chọn khác như sau:

* TIMEOUT: thời gian lưu trữ cache, mặc định là 300 giây (5 phút), bạn có thể đưa vào là Nonevà Django sẽ lưu cache vô thời hạn.
* OPTIONS: danh sách một số tùy chọn cache, bao gồm MAX\_ENTRIES là số lượng trang tối đa được phép cache, mặc định là 300; CULL\_FREQUENCY là số lượng trang cache bị xóa khi số lượng trang cache đã đạt mức tối đa, tính bằng 1 / CULL\_FREQUENCY, ví dụ CULL\_FREQUENCYlà 2 thì nếu số trang cache đã đạt đến 300 trang thì số lượng trang bị hủy là 150 trang. Nếu thiết lập CULL\_FREQUENCY=0 thì xóa toàn bộ cache.
* KEY\_PREFIX: đây là một chuỗi được thêm vào đầu các khóa được lưu trong cache. Chúng ta sẽ tìm hiểu thêm về khóa ở dưới.
* VERSION: số phiên bản cache sử dụng. Chúng ta cũng sẽ tìm hiểu ở dưới.
* KEY\_FUNCTION: tên hàm thực hiện việc tạo chuỗi key lưu trong cache.

Ví dụ:

CACHES = {

    'default': {

        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',

        'LOCATION': 'c:/cache.txt',

        'TIMEOUT': 60,

        'OPTIONS': {

            'MAX\_ENTRIES': 1000

        }

    }

}

Đoạn trên thiết lập cache được lưu trong file cache.txt, thời gian mỗi trang tồn tại là 60 giây, số lượng cache tối đa là 1000.

Nếu chúng ta có lỡ thiết lập sai biến nào thì Django sẽ không báo lỗi mà bỏ qua xem biến khác, do đó khi thiết lập cache chúng ta nên kiểm tra lại cho kỹ.

## Cache cả trang

Việc cache cũng có nhiều kiểu, chúng ta có thể cache từng phần hoặc cache nguyên cả trang web, nguyên trang tức là trong toàn bộ website của bạn có trang nào thì cũng đều được cache lại. Để cache cả trang web thì chúng ta khai báo các lớp sau trong biến MIDDLEWARE\_CLASSES trong file settings.py:

MIDDLEWARE\_CLASSES = [

    'django.middleware.cache.UpdateCacheMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.cache.FetchFromCacheMiddleware',

]

Lưu ý là bạn phải khai báo đúng thứ tự như trên thì mới sử dụng được.

Lớp FetchFromCacheMiddleware sẽ lưu lại các trang web mà có mã trả về là 200 và có request được gửi lên bởi phương thức GET và HEAD. Các yêu cầu đến trang web có tham số khác nhau thì được cache khác nhau, tức là giả sử chúng ta có một trang liệt kê danh sách sản phẩm được phân trang thì mỗi request với số trang khác nhau sẽ được phân trang khác nhau.

UpdateCacheMiddleware sẽ ghi một số thông tin vào cache như ngày/giờ tạo trang cache, thời gian tồn tại…

## Cache từng view

Nếu cache toàn bộ website có hơi thừa thì chúng ta cũng có thể cache từng trang tùy vào từng hàm view. Để cache từng view thì chúng ta dùng hàm cache\_page() trong lớp django.views.decorators.cache.

Chúng ta cũng không dùng hàm này như các hàm thông thường mà khai báo trước tên hàm view, ví dụ:

from django.views.decorators.cache import cache\_page

@cache\_page(60 \* 15)

def my\_view(request):

Trước hàm này chúng ta thêm dấu @. Hàm cache\_page() nhận 1 tham số là thời gian cache tồn tại tính theo giây, ở trên chúng ta cho thời gian này là 60 \* 15, tức là 15 phút, bạn có thể ghi ra số giây rõ ràng luôn chứ không nhất thiết phải dùng biểu thức nhân như vậy.

## Cache template

Ngoài việc cache các hàm view, bạn cũng có thể cache các phần của template. Thẻ {% load cache%} sẽ tải cache về nếu có, cặp thẻ {% cache %}...{% endcache %} sẽ cache lại nội dung bên trong nó trong một khoảng thời gian, thẻ này nhận vào 2 tham số bắt buộc là tên cache và thời gian cache. Ví dụ:

Cũng giống như cache cả trang, cache trên view cũng phân biệt tham số, tức là trang có URL như localhost:8000/cache-page/1 và localhost:8000/cache-page/2 sẽ được cache riêng.

{% load cache %}

{% cache 500 sidebar %}

    ...

{% endcache %}

## Cache dữ liệu

Nhiều khi việc cache nguyên cả một trang web là quá thừa thãi, đôi khi còn phản tác dụng. Đối với những trang web có nội dung thay đổi liên tục và mỗi lần tải trang là một lần truy vấn một lượng dữ liệu lớn, chẳng hạn như Facebook, thì việc cache nguyên trang là không tối ưu, thay vào đó chúng ta chỉ nên cache những thứ ít thay đổi như giao diện…

Django cũng cung cấp các hàm cache cấp thấp để hỗ trợ bạn làm việc này, bạn có thể cache bất cứ kiểu dữ liệu nào của Python như string, dictionary, list…

#### Truy vấn dữ liệu cache

Những dữ liệu mà bạn đã cache lại sẽ được lưu trong đối tượng django.core.cache.caches, đây là một đối tượng tĩnh toàn cục, tức là bạn có thể truy xuất ở bất kỳ đâu, đối tượng này lưu dữ liệu theo dạng dictionary, tức là mỗi phần tử là một cặp khóa-giá trị, chúng ta có thể truy xuất dữ liệu cache như sau:

>>> from django.core.cache import caches

>>> cache1 = caches['key1']

>>> cache2 = caches['key1']

>>> cache1 is cache2

Nếu chúng ta truy xuất sai tên khóa thì Django sẽ báo lỗi exception InvalidCacheBackendError.

#### Thiết lập cache

Lớp django.core.cache cung cấp 2 phương thức để thiết lập và lấy dữ liệu cache là **set(key,value,timeout)** và **get(key).** Ví dụ:

>>> cache.set('my\_key', 'hello, world!', 30)

>>> cache.get('my\_key')

'hello, world!'

Tham số timeout là tùy chọn, nếu chúng ta không đưa tham số này vào thì Django sẽ sử dụng thông số được thiết lập trong file settings.py. Nếu không tìm thấy dữ liệu thì phương thức get() sẽ trả về đối tượng None.

Nhưng nếu muốn chúng ta có thể thêm tham số default vào phương thức get() và Django sẽ trả về giá trị default nếu không tìm thấy dữ liệu cache:

>>> cache.get('my\_key', 'has expired')

'has expired'

Ngoài phương thức set() chúng ta còn có phương thức add() dùng để thiết lập thêm dữ liệu cache. Điểm khác nhau giữa 2 phương thức là set() sẽ cập nhật lại dữ liệu nếu cache đã tồn tại, còn add() thì không:

>>> cache.set('add\_key', 'Initial value')

>>> cache.add('add\_key', 'New value')

>>> cache.get('add\_key')

'Initial value'

Nếu bạn muốn lấy một giá trị mà không biết là đã có trong cache hay chưa, bạn có thể dùng phương thức get\_or\_set(), đúng như cái tên của nó, là sẽ trả về giá trị nếu đã có, còn nếu chưa có thì tạo mới:

>>> cache.get('my\_new\_key') # returns None

>>> cache.get\_or\_set('my\_new\_key', 'my new value', 100)

'my new value'

Ngoài các dữ liệu thông thường, bạn cũng có thể truyền giá trị là một phương thức nào đó, tất nhiên là phương thức này phải tra về một giá trị nào đó:

>>> import datetime

>>> cache.get\_or\_set('some-timestamp-key', datetime.datetime.now)

datetime.datetime(2014, 12, 11, 0, 15, 49, 457920)

Nếu bạn muốn lấy nhiều giá tri cùng một lúc thì dùng phương thức get\_many(), phương thức này rất hữu ích nếu chúng ta dùng phương pháp cache trên file hoặc trên cơ sở dữ liệu, vì chỉ cần đọc file cache một lần rồi lấy hết dữ liệu ra, không như phương thức get() là mỗi lần chỉ đọc một giá trị:

>>> cache.set('a', 1)

>>> cache.set('b', 2)

>>> cache.set('c', 3)

>>> cache.get\_many(['a', 'b', 'c'])

{'a': 1, 'b': 2, 'c': 3}

Tương tự, chúng ta cũng có phương thức set\_many() thiết lập nhiều giá trị cùng một lúc:

>>> cache.set\_many({'a': 1, 'b': 2, 'c': 3})

>>> cache.get\_many(['a', 'b', 'c'])

{'a': 1, 'b': 2, 'c': 3}

Để xóa một giá trị cache nào đó thì chúng ta dùng phương thức delete():

>>> cache.delete('a')

Tương tự với get\_many() và set\_many(), chúng ta có delete\_many():

>>> cache.delete\_many(['a', 'b', 'c'])

Nếu muốn xóa toàn bộ cache thì dùng phương thức clear():

>>> cache.clear()

#### Phiên bản cache

Phiên bản ở đây là chúng ta dùng các con số để đánh dấu dữ liệu cache nào thuộc về nhóm nào chứ không phải là phiên bản phần mềm cache của Django hay cái gì đó tương tự 🙂 Mặc định Django sẽ dùng số phiên bản là con số được khai báo trong file settings.py, nhưng chúng ta cũng có thể override lại trong khi viết code bằng cách đưa vào tham số version:

>>> cache.set('my\_key', 'hello world!', version=2)

>>> cache.get('my\_key')

None

>>> cache.get('my\_key', version=2)

'hello world!'

Trong đoạn code trên chúng ta thiết lập khóa my\_key với version là 2, khi chúng ta lấy giá trị của khóa này mà không chỉ định rõ version nào thì Django sẽ lấy khóa có version được lưu trong file settings.py, mặc định ở đây trong file settings.py lưu version là 1 nên đoạn code trên trả về đối tượng None.

Chúng ta cũng có thể tăng/giảm giá trị của version thông qua 2 phương thức incr\_version() và decr\_version():

# version = 3

>>> cache.incr\_version('my\_key')

>>> cache.get('my\_key')

None

>>> cache.get('my\_key', version=2)

None

>>> cache.get('my\_key', version=3)

'hello world!'

# **Sessions**

Đôi khi chúng ta muốn lưu lại một số thông tin trong quá trình duyệt web của user để sử dụng lại sau này, session cho phép chúng ta lưu trữ lại một số thông tin trên từng user, session có nhiều loại, có loại lưu dữ liệu trên server, có loại lưu trên client… Trong phần này chúng ta sẽ tìm hiểu về hệ thống session của Django.

## Kích hoạt session

Chúng ta kích hoạt session bằng cách khai báo django.contrib.middle.SessionMiddleware trong biến MIDDLEWARE\_CLASSES trong file settings.py:

mysite/settings.py

MIDDLEWARE\_CLASSES = [

    #...,

    'django.contrib.sessions.middleware.SessionMiddleware',

    #...

]

Mặc định thì session đã được kích hoạt sẵn khi tạo project nên chúng ta cũng không cần phải chỉnh sửa gì trong này.

## Cấu hình session

Dữ liệu trong session có thể được lưu trong cơ sở dữ liệu, file hoặc trong cache, mặc định thì Django lưu trong CSDL.

#### Lưu session trong cơ sở dữ liệu

Để cấu hình session lưu trong cơ sở dữ liệu thì chúng ta khai báo django.contrib.sessions trong biến INSTALLED\_APPS:

mysite/settings.py

INSTALLED\_APPS = [

    #...

    'django.contrib.sessions',

    #...

]

Sau khi thiết lập thì chúng ta phải chạy lệnh manage.py migrate để Django tạo bảng tương ứng trong CSDL.

#### Lưu session trong cache

Bạn chỉ nên dùng loại session này nếu bạn thiết lập kiểu cache của server là memcached. Để lưu session trong cache thì chúng ta thiết lập biến SESSION\_ENGINE trong file settings.py là:

* django.contrib.sessions.backends.cache: session lưu trong loại này không được đảm bảo vì nếu bộ nhớ cache đầy thì dữ liệu session sẽ bị xóa, nhưng loại này truy xuất dữ liệu cũng như ghi dữ liệu rất nhanh.
* django.contrib.sessions.backends.cached\_db: loại này thì vừa lưu dữ liệu session trong cache vừa lưu vào CSDL luôn, nếu session trong cache bị xóa thì Django sẽ tìm session trong CSDL nên dữ liệu được đảm bảo hơn loại trên nhưng cũng vì thế và tốc độ chậm hơn.

Trong thực tế thì chúng ta sẽ dùng loại thứ 2 vì dữ liệu session thường cũng không lớn nên việc đọc ghi sẽ không tốn thời gian mấy. Cũng chính vì loại thứ 2 lưu session trong cơ sở dữ liệu nên chúng ta cũng phải thiết lập luôn cả phần ***Lưu session trong cơ sở dữ liệu*** ở trên.

#### Lưu session trong file

Để lưu session trong file thì chúng ta khai báo SESSION\_ENGINE là django.contrib.sessions.backends.file và SESSION\_FILE\_PATH là đường dẫn đến tên file dùng để lưu session, đường dẫn phải là đường dẫn tuyệt đối và server phải có quyền đọc/ghi file trên đĩa cứng.

#### Lưu session trong cookie

Chúng ta khai báo SESSION\_ENGINE là django.contrib.sessions.backends.signed\_cookies. Bạn để ý là trong file settings.py có một biến tên là SECRET\_KEY có giá trị là một chuỗi được Django tạo ra ngẫu nhiên, biến này sẽ được dùng để mã hóa dữ liệu trong sessions, chúng ta sẽ tìm hiểu về biến này sau.

## Truy xuất session trong view

Tham số request trong hàm view sẽ chứa một thuộc tính có tên là session khi SessionMiddlewaređược kích thoạt, đây là một đối tượng dictionary. Chúng ta có thể đọc/ghi thuộc tính này ở bất kỳ đâu trong hàm view.

Dưới đây là một số thao tác với session:

Lấy giá trị theo khóa:

>>> color = request.session['color']

>>> color1 = request.session.get('color', 'red')

Tham số 'red' là tham số trả về mặc định, tức là nếu không tìm thấy khóa 'color' trong session thì trả về giá trị 'red'.

Thiết lập khóa:

>>> request.session['color'] = 'blue'

Xóa khóa ra khỏi session, nếu không tìm thấy khóa thì báo lỗi KeyError:

>>> del request.session['color']

Kiểm tra xem khóa có tồn tại trong session:

>>> 'color' in request.session

Xóa toàn bộ dữ liệu session:

>>> request.session.flush()

Thiết lập thời gian tồn tại cho session, tham số nhận vào là số giây, nếu để 0 thì tồn tại cho đến khi user tắt trình duyệt.

>>> request.session.set\_expiry(300)

Lấy thời gian còn lại của sessiontheo giây.

>>> request.session.get\_expiry\_age()

Với những lớp Django.Views là một lớp base view có hàm [**as\_view()**](https://docs.djangoproject.com/en/2.2/ref/class-based-views/base/#django.views.generic.base.View.as_view) gọi đến một hàm chứ không phải một lớp.

[**HttpResponseNotAllowed**](https://docs.djangoproject.com/en/2.2/ref/request-response/#django.http.HttpResponseNotAllowed) là một hàm thay thế khi không có phương thức nào phù hợp với request

[**dispatch()**](https://docs.djangoproject.com/en/2.2/ref/class-based-views/base/#django.views.generic.base.View.dispatch)  xác định đâu là phương thức **GET**, **POST**

Decorator là một mẫu thiết kế (Design pattern) thường được dùng để thay đổi hành vi, chức năng của hàm(function) hoặc lớp (class) mà không cần phải thay đổi code của hàm hoặc lớp.  
Tham khảo  
Python hỗ trợ cú pháp (syntax) cho Decorator từ version 2.4  
Về cơ bản Decorator giống như một lớp vỏ bọc (wrapper), nó thay đổi hành vi(behavior) của code trước và sau khi gọi hàm chính (hàm được decorate).

##2. Decorator được dùng làm gì?

Tại sao chúng ta cần Decorator?

Cho phép tái sử dụng code.  
Mở rộng các hàm, hoặc lớp mà không cần phải thay đổi code có sẵn —> không cần test lại.  
Ví dụ trong chương trình của bạn bạn cần kiểm tra quyền (permission) của người dùng trước khi thực hiện hàm. Bạn có thể phải thêm code vào tất cả các hàm đã có để kiểm tra. Thay vào đó với decorator, bạn chỉ cần định nghĩa một decorator và khai báo nó trước hàm.

##3. Làm sao để định nghĩa một decorator?

Decorator cũng là một hàm chỉ khác là hàm decorator nhận vào một hàm và kết quả trả về của nó là hàm sau khi được decorate.  
Như vậy để định nghĩa một decorator chỉ đơn giản là định nghĩa một hàm nhận vào một hàm khác và trả về một hàm mới có prototype tương đương với hàm nhận vào.

**Ví dụ 1:**

**def** **ten\_decorator**(f):

**def** **wrapper**(ten):

chuoi\_moi = "Ten tui la %s" % ten

**return** f(chuoi\_moi)

**return** wrapper

**def** **xuat\_ten**( ten ):

**print** ten

**CHÚ Ý QUAN TRỌNG**: hàm wrapper và hàm f phải có tham số phù hợp với nhau. Ví dụ như hàm f nhận vào chỉ 2 tham số thì hàm decorator không thể nhận vào 3 tham số hoặc 1 tham số.

##4. Sử dụng decorator như thế nào?

Sử dụng Decorator hết sức đơn giản. Sử dụng decorator trong ví dụ trên cho hàm xuat\_ten như sau:

@ten\_decorator

**def** **xuat\_ten**(ten):

**print** ten

Dùng dấu @ để thông báo đó là một decorator. Một hàm có thể dùng nhiều decorator cùng lúc:

@ten\_decorator1

@ten\_decorator2

@ten\_decorator3

**def** **xuat\_ten**(ten):

**print** ten

##5. Decorator hoạt động như thế nào?

Như trong ví dụ ở trên:

ten\_decorator nhận vào hàm f, sau đó bọc hàm f trong hàm wrapper của nó và trả về hàm wrapper. Hàm wrapper có nhiệm vụ gắn thêm thông tin vào tên rồi mới gọi thực hiện hàm f với chuỗi mới.  
Việc sử dụng:

@ten\_decorator

**def** **xuat\_ten**(ten):

**print** ten

xuat\_ten('coulson')

Tương đương với:

**def** **xuat\_ten**(ten):

**print** ten

ham\_xuat\_ten\_moi = ten\_decorator(xuat\_ten)

ham\_xuat\_ten\_moi('coulson')

Rõ ràng với việc sử dụng cú pháp decorator thì code sẽ ngắn gọn và đơn giản hơn. Developer không phải gọi decorator mỗi lần sử dụng mà trình thông dịch sẽ làm việc đó.  
Đối với hàm sử dụng nhiều decorator

Trong file company.py hàm

def doPostEvent(self, json):

dùng để nhận request từ client

Class

class BaseHtmlView(View):

Class cha các định các phương thức cơ bản để tạo khung view, sau khi các lớp con thừa kế từ nó thì có thể overwrite lại các hàm trong đó

class ModelManager():

Trong file modelmanage.py là lớp điều khiển mô hình. Ở đó nó tạo một đối tượng bao gồm các thuộc tính là các dict (đối tượng model hiện tại, chứa các tên của các file Json trong thư mục files – Typecode, modellist là các models có quan hệ với model cha,…) và bao gồm các hàm sử lý các model

Template

Base.html template gốc

Exs\_main.html template thừa kế từ template base.html. Nội dung được chủ yếu hiển thị ở đây, trong template này chứa các block để cho các template con sau khi thừa kế có thể overwrite lại.

Hàm

module = getattr(models, class\_name)

trong thư mục models hàm init.py là hàm liệt kê các đối tượng models trong một danh sách các file model.py (các file tạo nên cơ sở dữ liệu)

hàm getattr() để lấy một đối tượng class\_name trong danh sách đó

File

Model.py cung cấp các hàm để lấy ra đối tượng model

Đặt BUG ở hai hàm get và post của class BaseHTML

*Ban đầu code sẽ chạy theo url ở file url. Sau đó sẽ trỏ đến class mà nó trỏ đến. Class đó sẽ thừa kế từ lớp BaseHtmlView nên sẽ thừa kế hàm get(), post(). Sau đó nó sẽ dựa theo method là get hay là post để chạy vào hàm get() hoặc post() của class BaseHtmlView, nhưng biến self bây giờ sẽ chỉ đến class theo url ban đầu chứ không phải là đối tượng của BaseHtmlView*

*Thông thường hàm*

def doPageHtml(self):

*sẽ được overwrite ở lớp con (lớp kế thừa)*

*Tất cả các views đều được thực hiện theo quy trình của các hàm get và post trong class BaseHtmlView. Khi gọi hàm thì nếu class thừa kế có hàm đó thì nó sẽ thực hiện hàm đó còn nếu không thì nó sẽ thực hiện tại hàm của class BaseHtmlView.*