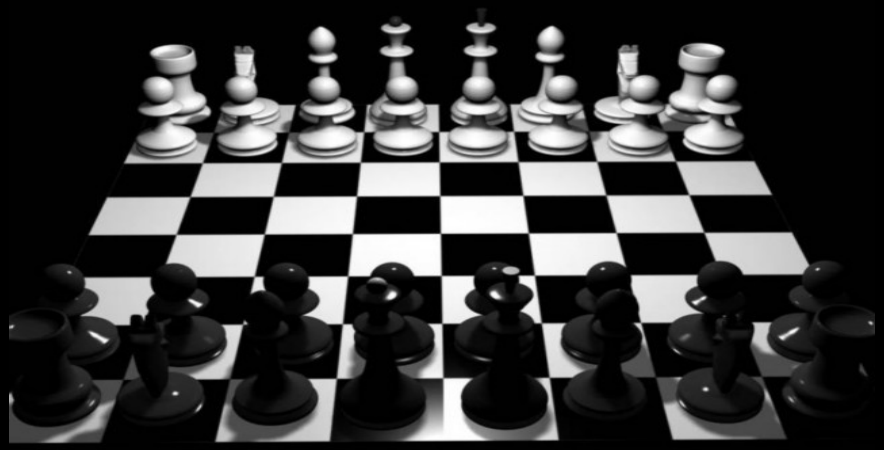
МГТУ им. Н.Э. Баумана
ТЕХНОПАРК
Mail.Ru Group

Обработка HTTP запросов. Django Views



Александр Бекбулатов
Дмитрий Смаль

План лекции

1. Конфигурация Django.
2. Маршрутизация URL.
3. Django Views.
4. Использование Django Forms для обработки входных данных.
5. Генерация HTML страницы с помощью шаблонизатора.

Конфигурация Django

Конфигурация Django

DEBUG = True

- Показываются отладочная информация
- Можно отдавать статику средствами сервера django
- Сохраняются запросы в django.db.connection

Конфигурация Django

Переход на DEBUG = False

- Создать шаблоны 404.html и 500.html
- Настроить почтовый сервер для отправки ошибок?
- Настроить LOGGING для логирования в stderr http://tp.mail.ru/blog/Web/571.html
- Указать разрешенные хосты ALLOWED_HOSTS

Конфигурация Django

Другие важные настройки

- DATABASES
- INSTALLED_APPS
- TEMPLATE_DIRS
- ROOT_URLCONF
- MEDIA_ROOT, MEDIA_URL
- STATIC_ROOT, STATIC_URL

Конфигурация Django

Относительные пути

```
import os.path

def rel(tail):
    p = os.path.abspath(os.path.dirname(__file__))
    return os.path.join(p, tail)

TEMPLATE_DIRS = (
    rel('../templates'),
)
MEDIA_ROOT = rel('../uploads')
STATIC_ROOT = rel('../static')
```

Конфигурация Django

Паттерн local_settings

global_settings.py – настройки по умолчанию

settings.py – публичные настройки

local_settings.py – настройки окружения

В settings.py:

```
try:
    from ask_pupkin.local_settings import *
except ImportError:
    pass
```

Маршрутизация URL

Маршрутизация URL

Маршрутизация проекта

```
urlpatterns = patterns('',
    url(r'^$', 'blog.views.home', name='home'),
    url(r'^', include('contacts.urls')),
    url(r'^admin/', include('admin.site.urls')),
)
```

Маршрутизация приложения

```
from blog.views import post_list
urlpatterns = patterns('blog.views',
    url(r'^$', post_list, name='post-list'),
    url(r'^category/(\d+)/$', 'category_view',
        name='post-list-by-category'),
    url(r'^(?P<pk>\d+)/$', ' post_detail',
        name='post-detail'),
)
```

Маршрутизация URL

Особенности

- Слеш в начале роутов не указывается
- Роуты описываются с помощью регулярных выражений
- Можно и нужно разносить роуты по приложениям
- Можно и нужно создавать именованые роуты
- **Одно действие – один роут – один контроллер**

Django Views

Django Views

Требования к контроллерам (вьюшкам)

- Должны быть callable объектом (функция или класс)
- Должны принимать HttpRequest первым параметром
- Должны возвращать HttpResponse

Django Views

```
def post_list(request):
    object_list = Post.objects.all()[:20]
    return render(request, 'blog/post_list.html', {
        'post_list': object_list
    })

def post_detail(request, pk):
    try:
        object = Post.objects.get(pk=pk)
    except Post.DoesNotExist:
        raise Http404
    return render(request, 'blog/post_detail.html',
        {'object': object})
```

Django Views

```
def category_view(request, pk):
    category = get_object_or_404(Category, pk=pk)
    post_list = Post.objects.filter(
        category=category)
    return render(request, 'blog/category.html', {
        'category': category,
        'post_list' : post_list
    })
```

Захват и передача параметров из URL во View

Захват параметров

```
url(r'^category/(\d+)/$', 'category_view’,
    name='post-list-by-category')

url(r'^(?P<pk>\d+)/$', ' post_detail',
    name='post-detail')
```

Захват и передача параметров из URL во View

МГТУ им. Н.Э. Баумана ТЕХНОПАРК Mail.Ru Group

Передача во View

```
def category_view(request, pk=None):
    if pk is None:
        # вывести все посты

def category_view(request, *args, **kwargs):
   pk = args[0]
   pk = kwargs['pk']
```

Построение URL по view

В python коде

```
from django.core.urlresolvers import reverse

reverse('home')
reverse('category-view', args=(10,))
reverse('post-detail', kwargs={'pk': 7})
```

В шаблоне

```
{% url 'question-view' question.id %}
```

HttpRequest

МГТУ им. Н.Э. Баумана ТЕХНОПАРК Mail.Ru Group

Свойства

- request.method
- request.GET
- request.POST
- request.COOKIES
- request.FILES
- request.META
- request.user

Методы

- request.get_full_path()
- request.build_absolute_uri()
- request.is_ajax()

HttpResponse

```
def get_time(request):
    now = datetime.datetime.now()
    return HttpResponse(
        "It is now %s" % now,
        content_type="text/plain")
```

Классы с другим кодом возврата

- HttpResponseRedirect
- HttpResponsePermanentRedirect
- HttpResponseNotFound
- HttpResponseForbidden

Получение GET и POST параметров

GET

```
page = request.GET.get('page') or 1
try:
    return int(page)
except ValueError:
    raise Http404



order = request.GET.get('sort')
if order == 'rating':
    queryset = queryset.order_by('rating')
```

Получение GET и POST параметров

POST

```
message = request.POST.get('message')
sender = request.POST.get('email')
if message and sender:
    send_mail('[My Blog]', message, sender,
              ['me@mail.ru'])
```

Получение и установка HTTP заголовков


```
user_agent = request.META.get('HTTP_USER_AGENT')

user_ip = request.META.get('REMOTE_ADDR', None)
if user_ip in INTERNAL_IPS:
    response.content += debug_output



response = HttpResponse(my_data,
  content_type='application/vnd.ms-excel')
response['Content-Disposition'] = \
    'attachment; filename="foo.xls"'
```

Получение и установка Cookie

Установка

```
response = render(request, 'blog/index.html')
response.set_cookie('visited', '1')
return response
```

Получение

```
if not request.COOKIES.get('visited'):
    show_banner = True
```

Декораторы в Python

Декоратор – функция, преобразует одну функцию в другую, может полностью изменить поведение.

```
def double_it(func):
    def tmp(*args):
        return func(*args) * 2
    return tmp

@double_it
def mult(a, b):
    return a*b

>>> mult(2,2)
8
```

Декораторы в Django

```
from django.views.decorators.http import …
@require_GET – только GET запросы
@require_POST – только POST запросы

from django.contrib.auth.decorators import …
@login_required(login_url='/login/')

from django.views.decorators.csrf import …
@csrf_protect – включить проверку CSRF
@csrf_exempt – отключить проверку CSRF

@require_POST
def like(request):
    pass
```

Использование Django Forms для обработки входных данных

Django Forms

```
from django import forms

class ContactForm(forms.Form):
    email = forms.EmailField(max_length=100)
    message = forms.CharField(
                widget=forms.Textarea)

    def clean_message(self):
        message = self.cleaned_data.get('message')
        if not check_antispam(message):
            raise form.ValidationError(
              'Spam detected!')
```

Django Forms

Особенности

- API аналогично моделям
- Валидация определяется типом полей
- Можно писать дополнительные правила валидации:
  - метод clean для общей валидации
  - метод clean_<поле> для валадиции поля
- По умолчанию поля обязательные (required)

Django Forms

Свойства полей формы

BooleanField, **CharField**, **ChoiceField**, TypedChoiceField,

DateField, **DateTimeField**, DecimalField, EmailField,

FileField, FilePathField, FloatField, ImageField, IntegerField,

MultipleChoiceField, TypedMultipleChoiceField,

NullBooleanField, RegexField, SlugField, TimeField, URLField,

ComboField, MultiValueField, SplitDateTimeField,

ModelChoiceField, ModelMultipleChoiceField

https://docs.djangoproject.com/en/dev/ref/models/fields/

Django Forms API

```
>>> f = ContactForm()
>>> data = {'email': 'foo@example.com', 'message':
'Hi there'}
>>> f = ContactForm(data)
>>> f.is_valid()
True
>>> f.cleaned_data
{'email': u'foo@example.com', 'message': u'Hi
there'}
>>> f = ContactForm({})
>>> f.is_valid()
False
>>> f.errors
{'email': [u'Enter a valid e-mail address.'],
'message': [u'This field is required.']}
```

Django Forms API

```
>>> f = ContactForm()

>>> print f.as_table()
<tr><th><label for="id_for_email">Ваш e-
mail:</label></th><td><input id="id_for_ email "
type="text" name="email" maxlength="100"
/></td></tr>…

>>> print f['email']
<input id="id_for_email" type="text" name="email"
maxlength="100" />
```

Django Forms во Views

```
def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            message = form.cleaned_data['message']
            sender = form.cleaned_data['email']
            recipients = send_mail('My Blog',
                  message, sender, ['me@mail.ru'])
            return HttpResponseRedirect('/')
    else:
        form = ContactForm()

    return render(request, 'blog/contact.html', {
        'form': form
    })
```

Django Forms

МГТУ им. Н.Э. Баумана ТЕХНОПАРК Mail.Ru Group

Визуализация

- Указывать в форме стилевые атрибуты
- Выводить поля по одному и оборачивать div
- Использовать сторонние приложения

Django Forms

Визуализация

```
<form class="form-horizontal" method="post">
    <fieldset>
        {% for field in form %}
            <div class="control-group">
                <label class="control-label">
                  {{ field.label }}</label>
                <div class="controls">{{ field }}</div>
            </div>
        {% endfor %}
    </fieldset>
    <div class="form-actions">
        <button type="submit" class="btn btn-primary" >
          Submit</button>
    </div>
</form>
```

Django Model Forms

```
class QuestionForm(forms.ModelForm):

    def __init__(self, *args, **kwargs):
        self.user = kwargs.pop('user', None)
        super(QuestionForm, self).__init__(*args, **kwargs)

    def save(self, commit=True):
        instance = super(QuestionForm, self).save(commit=False)
        instance.user = self.user
        if commit:
            instance.save()
        return instance

    class Meta:
        model = Question
        fields = ['title', 'content']
```

Django Model Forms

- Поля заполняются на основе модели
- Можно переопределять внешний вид полей
- Необходимо указывать, какие поля необходимо использовать в форме

Генерация HTML страницы с помощью шаблонизатора

Особенности шаблонизатора

- Шаблоном может быть любой текстовый файл (HTML, XML, CSV)

- В django все данные передаются в 1 шаблон

```
return render(request, 'blog/post_list.html', {
    'post_list' : object_list
})
```

- Шаблоны размещают по приложениям

- Выражения выделяются так: {% for %}

- Переменные выделяются так: **{{ email }}**

Особенности шаблонизатора

- Шаблоном может быть любой текстовый файл (HTML, XML, CSV)

- В django все данные передаются в 1 шаблон

```
return render(request, 'blog/post_list.html', {
    'post_list' : object_list
})
```

- Шаблоны размещают по приложениям

- Выражения выделяются так: {% for %}

- Переменные выделяются так: **{{ email }}**

Особенности шаблонизатора

- Через точку можно получить свойство, метод, ключ либо индекс объекта:

```
{{ object.content }}
{{ name.strip }}
{{ info.avatar }}
{{ post_list.0 }}
```

- Нельзя вызывать метод с параметрами, но можно что-нибудь удалить:

```
{{ post_list.order_by('id') }}
{{ post_list.delete }}
```

Особенности шаблонизатора

- Фильтры отделяются чертой | и могут принимать параметры:

```
{{ content|safe }}
{{ now|time:"H:i" }}
```

- Теги выделются скобками с %:

```
{% if post_list|length > 1%}  {% endif %}
```

- Однострочные комментарии выдяляются скобками с #:

```
{# user block #}
```

Особенности шаблонизатора

- Шаблоны наследуются.
-
- Шаблоны можно вставлять один в другой:

```
{% include "user.html" with object=user %}
```

- Шаблоны эскейпят HTML в сущности

- Можно расширять шаблонизатор своими тегами и фильтрами

Наследование шаблонов

Базовый шаблон

```
<!DOCTYPE HTML>
<html>
<head>
    <title>
        {% block title %}Мой блог{% endblock %}
    </title>
    {% block extrahead %}{% endblock %}
</head>
<body>
    <h1>Мой блог</h1>
    {% block content %}{% endblock %}
</body>
</html>
```

Наследование шаблонов

Шаблон страницы

```
{% extends "base.html" %}
{% block title %}
  {{ block.super }} – Список
{% endblock %}

{% block content %}
   <ul>
   {% for object in post_list %}
       <li><a href="{{ object.get_absolute_url }}">
           {{ object }}</a>
           {{ object.created_date|date:"d.m.Y" }}
       </li>
   {% endfor %}
   </ul>
{% endblock %}
```

Контекстные процессоры

TEMPLATE_CONTEXT_PROCESSORS:

django.core.context_processors.request (request)

-
- django.core.context_processors.csrf (csrf_token)
-
- django.core.context_processors.static (STATIC_URL)
-
- django.contrib.auth.context_processors.auth
- (user, perms)

Контекстные процессоры

RequestContext и Context

```
from django.shortcuts import render_to_response
return render_to_response(
    'my_template.html', {'foo': 'bar'}
)


from django.shortcuts import render
return render(request,
    'my_template.html', {'foo': 'bar'}
)
```

Контекстные процессоры


```
{% extends "base.html" %}

{% block content %}
   <form action="" method="post">
   {% csrf_token %}
   {{ form.as_p }}
   <input type="submit" value="Отправить" />
   </form>
{% endblock %}
```

Генерация ответа в JSON формате

Работа с JSON

```
class AjaxResponse(HttpResponse):

    def __init__(self, status, msg,
                 extra_context=None):

        response = {'status': status, 'msg': msg}

        if not extra_context is None:
            response.update(extra_context)

        super(AjaxResponse, self).__init__(
            content=json.dumps(response),
            content_type='application/json'
        )
```

Работа с JSON

```
@require_POST
def like(request):
    form = LikeForm(request.POST)
    if not form.is_valid():
        return AjaxResponse(False, 'Failure!',
           {'errors': dict(form.errors)})
    # do like logic
    return AjaxResponse(True, 'Success!')
```

Домашнее задание

“Отображение данных”
https://tech-mail.ru/blog/Web/2094.html

“Интерактивный сайт”
https://tech-mail.ru/blog/Web/2186.html

МГТУ им. Н.Э. Баумана
ТЕХНОПАРК
Mail.Ru Group

Спасибо за внимание

Александр Бекбулатов,
a.bekbulatov@corp.mail.ru

Дмитрий Смаль,
d.smal@corp.mail.ru