



Python *, Django *

Tutorial

Автор оригинала: Nathan Yergler

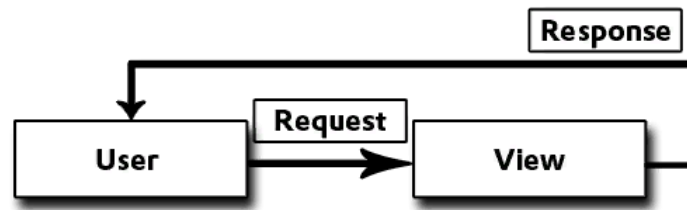
[illegible]

Продолжение перевода статей о Django с сайта effectivedjango.com. Наткнулся я на этот сайт во время изучения данного фреймворка. Информация размещенная на этом ресурсе показалась мне полезной, но так как нигде не нашел перевода на русский, решил сделать сие доброе дело сам. Этот цикл статей, как мне думается, будет полезен веб-разработчикам, которые делают только первые шаги в изучении Django.

- Введение
- Эффективный Django. Руководство
 - Глава 1. Приступая к работе
 - Глава 2. Используем модель
 - Глава 3. Пишем представление
 - Глава 4. Используем статические ресурсы
 - Глава 5. Дополнительные базовые представления
 - Глава 6. Основы форм
 - Глава 7. Связанные модели
 - Глава 8. Обработка аутентификации и авторизации
- Тестирование в Django
- Понимание Middleware'ов
- Базы данных и модели
- Представления-классы (CBV)
- Формы

3.1. Основы представлений

Представления Django получают [HTTP запрос](#) и возвращает пользователю [HTTP ответ](#):



Любой вызываемый объект языка Python может быть представлением. Единственное жесткое и необходимое требование заключается в том, что вызываемый объект Python должен принимать объект запроса в качестве первого аргумента (обычно этот параметр так и именуют — `request`). Это означает, что минимальное представление будет очень простым:

```

from django.http import HttpResponseRedirect

def hello_world(request):
    return HttpResponseRedirect("Hello, World")
  
```

Конечно, как и большинство других фреймворков, Django позволяет вам передавать аргументы в представление через URL. Мы поговорим об этом, когда будем строить наше приложение.

3.2. Общие представления и представления-классы

- [общие представления](#) всегда предоставляют какой-нибудь базовый функционал: [визуализировать шаблон](#), [перенаправить](#), [создать](#), [отредактировать модель](#) и т. д.
- начиная с версии 1.3, для общих представлений в Django появились [представления-классы](#) (CBV);
- общие представления и CBV предоставляют более высокий уровень абстракции и компонентности;
- кроме того, они скрывают немало сложности, которая иначе могла бы сбить с толку новичков;
- к счастью, в новых версиях Django документация всего этого стала намного лучше.

Django 1.3 вводит представления-классы на которых мы сосредоточимся в этой главе.

Представления-классы или CBV, могут устранить много шаблонного кода из ваших представлений, особенно из представлений для редактирования чего-либо, где вы хотите предпринимать различные действия для GET и POST запросов. Представления-классы дадут вам возможность собирать функционал по частям. Недостаток заключается в том, что вся эта мощь влечет за собой некоторое дополнительное усложнение.

3.3. Представления-классы (CBV)

Минимальное представление, реализованное как CBV, является наследником класса `View` ^{doc} и реализует поддерживаемые HTTP-методы. Ниже находится очень небольшое представление «Hello, World» написанное нами ранее, но выполненное в виде представления-класса:

```

from django.http import HttpResponseRedirect
from django.views.generic import View

class MyView(View):

    def get(self, request, *args, **kwargs):
        return HttpResponseRedirect("Hello, World")
  
```

В представлении-классе имена методов HTTP отображаются на методы класса. В нашем случае мы определили обработчик для GET-запроса используя метод класса `get`. Точно так же, как при реализации функций, этот метод принимает объект запроса в качестве первого параметра и возвращает HTTP ответ.

Примечание: Допустимые сигнатуры

Вы можете заметить несколько дополнительных аргументов в сигнатуре функции, по сравнению с представлением написанным нами ранее, в частности `*args` и `**kwargs`. CBV впервые были создавались для того, что бы сделать «общие» представления Django более гибкими. Подразумевалось, что они будут использоваться в множестве различных контекстов, с потенциально различными аргументами, извлеченными из URL. Занимаясь последний год написанием представлений-классов, я продолжаю писать их используя допустимые сигнатуры, и каждый раз это оказывается полезным в неожиданных ситуациях.

3.4. Вывод перечня контактов

Мы начнем с представления, которое выводит список контактов из базы данных.

Базовая реализация представления очень коротка. Мы можем написать его всего-лишь в несколько строк. Для этого в файле `views.py` нашего приложения `contacts` наберем следующий код:

```
from django.views.generic import ListView

from contacts.models import Contact

class ListContactView(ListView):

    model = Contact
```

Класс `ListView` ^{`doc`}, от которого мы наследовали представление, сам составлен из нескольких примесей, которые реализуют некоторое поведение. Это дает нам большую мощь при малом количестве кода. В нашем случае мы указали модель (`model = Contact`), что заставит это представление вывести список всех контактов модели `Contact` из нашей базы данных.

3.5. Определяем URL'ы

Конфигурация URL (URLconf) указывает Django как по адресу запроса найти ваш Python-код. Django смотрит конфигурацию URL, которая определена в переменной `urlpatterns` файла `urls.py`.

Давайте добавим в файл `addressbook/urls.py` URL-шаблон для нашего представления, которое отображает список контактов:

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

import contacts.views

urlpatterns = patterns('',
    url(r'^$', contacts.views.ListContactView.as_view(),
        name='contacts-list'),
)
```

- использование функции `url()` не является строго обязательным, но я предпочитаю использовать ее: когда вы начнете добавлять больше информации в URL-шаблоны, она позволит вам использовать именованные параметры, делая код более чистым;
- первый параметр принимаемый функцией `url()` — это регулярное выражение. Обратите

внимание на замыкающий символ `$` ; как думаете, почему он важен?

- вторым параметром указывается вызываемое представление. Это может быть как непосредственно вызываемый объект (импортированный вручную), так и строка описывающая его. Если это строка, то в случае соответствия URL-шаблона строке запроса, Django сам импортирует необходимый модуль (до последней точки), и вызовет последний сегмент строки (после последней точки);
- замете, что когда мы используем представление-класс, мы обязаны использовать реальный объект, а не строку описывающую этот объект. Мы должны так делать из-за того, что мы вызываем метод класса `as_view()` . Этот метод возвращает обертку над нашим классом, которую может вызвать диспетчер URL Django;
- имя, данное URL-шаблону, позволят вам делать обратный поиск;
- имя URL полезно, когда вы ссылаетесь из одного представления на другое, или выполняете перенаправление, так как вы можете управлять структурой URL'ов из одного места.

В то время, как переменная `urlpatterns` должна быть определена, Django также позволяет вам определить несколько других значений (*переменных*) для исключительных ситуаций. Эти значения включают в себя `handler403` , `handler404` , и `handler500` , которые указывают Django, какое представление использовать в случае ошибки HTTP. Для более подробной информации смотрите [документацию Django](#) по конфигурации URL.

Примечание: Ошибки импортирования конфигурации URL

Django загружает конфигурацию URL очень рано во время старта и пытается импортировать найденные внутри модули. Если хотя бы одна из операций импорта обернется неудачей, сообщение об ошибке может быть немного непонятным. Если ваш проект перестал работать по причине ошибки импорта, попробуйте импортировать конфигурацию URL в интерактивной оболочке. Это позволяет, в большинстве случаев, определить в каком месте возникли проблемы.

3.6. Создание Шаблона

Сейчас, определив URL для нашего представления списка контактов, мы можем испытать его. Django включает в себя сервер подходящий для целей разработки, который вы можете использовать для тестирования вашего проекта:

Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп 🔍

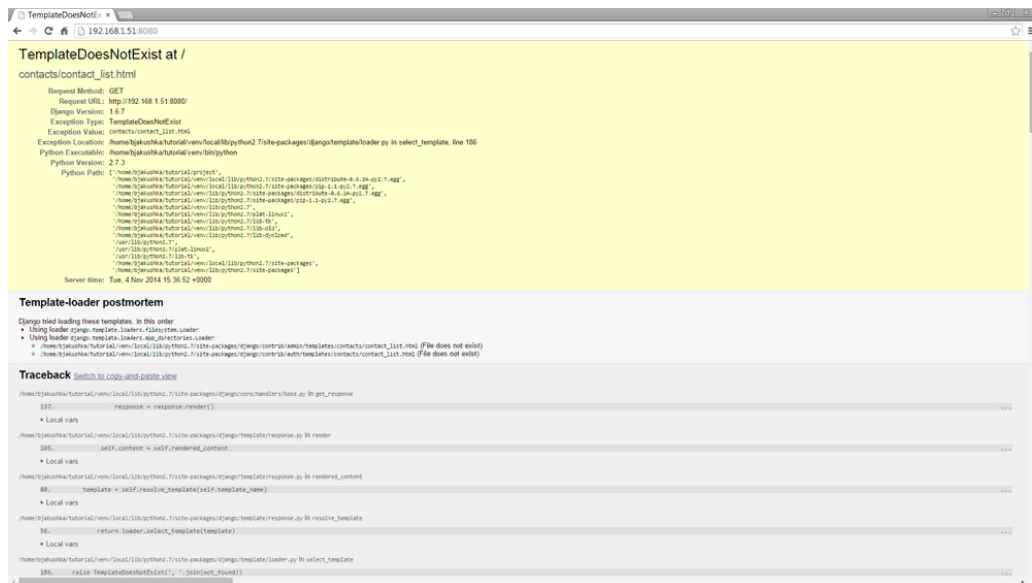
```
validating models...

0 errors found
November 04, 2014 - 15:25:05
Django version 1.6.7, using settings 'addressbook.settings'
Starting development server at http://0.0.0.0:8080/
Quit the server with CONTROL-C.
```

Если вы посетите <http://localhost:8080/> в вашем браузере, вы все же получите ошибку `TemplateDoesNotExist`

Примечание переводчика:

Localhost указывайте в том случае, если запускаете браузер с того же хоста, где запущен сервер. Если сервер у вас запущен на другом хосте (как у меня) — вместо *Localhost* укажите IP адрес этого хоста (в моем случае это `192.168.1.51`).



Большинство общих представлений Django (сюда относиться, `ListView` использованный нами) имеет предустановленное имя шаблона, который они ожидают найти. Мы можем увидеть в этом сообщении об ошибке, что представление ожидало найти файл шаблона с именем `contact_list.html`, которое было получено из имени модели. Давайте создадим такой шаблон.

По умолчанию, Django ищет шаблоны в приложениях, так же как и в директориях, указанных в `settings.TEMPLATE_DIRS`. Общие представления ожидают, что шаблоны найдутся в директории приложения (в данном случае в директории `contacts`), и имя файла будет содержать имя модели (в данном случае ожидаемое имя файла: `contact_list.html`). Это приходится кстати, когда вы разрабатываете приложения для повторного использования: пользователь вашего приложения может создать свои шаблоны, которые переключат шаблоны по умолчанию, и они будут храниться в директории, прямо связанной с приложением.

Примечание переводчика:

◆ +19 👁 64K 📖 338 ➡

Для наших целей, однако, нам не нужен дополнительный слой из структур директорий, так что мы определим шаблон явно, используя свойство `template_name` нашего представления-класса. Давайте добавим одну строчку в `views.py`:

```
from django.views.generic import ListView

from contacts.models import Contact

class ListContactView(ListView):

    model = Contact
    template_name = 'contact_list.html'
```

Создадим в директории `contacts` (это директория с приложением) поддиректорию `templates`, а в ней создадим файл шаблона `contact_list.html`:

```
<h1>Contacts</h1>

<ul>
    {% for contact in object_list %}
    <li class="contact">{{ contact }}</li>
    {% endfor %}
```


Открыв заново (или обновив) в браузере страницу <http://localhost:8080/>, вы должны будете увидеть как минимум один контакт, созданный нами ранее через интерактивную оболочку.



3.7. Создаем контакты

Добавление информации в базу данных через интерактивную оболочку занимает слишком много времени, так что давайте создадим представление для добавления новых контактов.

Так же как и в случае с выводом списка контактов, воспользуемся одним из общих представлений Django. В файле `views.py` мы добавим несколько строчек:

```
from django.core.urlresolvers import reverse
from django.views.generic import ListView
from django.views.generic import CreateView

from contacts.models import Contact

class ListContactView(ListView):

    model = Contact
    template_name = 'contact_list.html'

class CreateContactView(CreateView):

    model = Contact
    template_name = 'edit_contact.html'

    def get_success_url(self):
        return reverse('contacts-list')
```

Примечание переводчика:

Если вы используете Django версии ≥ 1.7 , то можете добавить к классу `CreateContactView` дополнительное поле:

```
fields = ['first_name', 'last_name', 'email']
```

Это не обязательно, но с версии Django 1.7 неиспользование этого поля в классах с автоматическим генерированием форм объявлено устаревшим (при выполнении тестов вам об этом сообщат). Если вы его не укажете — то в форме редактирования будут

использоваться все поля, но с версии Django 1.8 такое поведение будет удалено.

Большинство общих представлений, которые работают с формами имеют концепцию «удачного URL»: такого URL, на которой перенаправляется пользователь, после того как форма была удачно обработана. Все представления обрабатывающие формы твердо придерживаются практики POST-перенаправление-GET для принятия изменений, так что обновление конечной страницы не отправляет заново форму. Вы можете реализовать это концепцию как свойство класса, либо переопределить метод `get_success_url()`, как это сделали мы. В нашем случае мы используем функцию `reverse` для вычисления URL'a списка контактов.

Примечание: Контекстные переменные в представлениях-классах

Набор переменных доступных в шаблоне, когда он выводится, называется Контекстом.

Контекст — это комбинация данных из представления и информации из процессоров контекста.

Когда вы используете встроенные общие представления, не совсем очевидно какие переменные доступны в контексте. Со временем вы откроете для себя, что их имена (предоставляемые общими представлениями в контекст шаблона) достаточно последовательны — `form`, `object` и `object_list` часто используются — хотя это не поможет вам в начале пути. К счастью, документация по этому вопросу очень похорошела с версии Django 1.5.

В представлениях-классах метод `get_context_data()` используются для добавления информации в контекст. Если вы перегрузите этот метод, вам нужно будет разрешить `**kwargs` и вызвать суперкласс.

Шаблон добавления контакта будет немного более сложный, чем шаблон списка контактов, но не слишком. Наш шаблон `contacts/templates/edit_contact.html` будет выглядеть как то так:

```
<h1>Add Contact</h1>

<form action="{% url 'contacts-new' %}" method="POST">
    {% csrf_token %}
    <ul>
        {{ form.as_ul }}
    </ul>
    <input id="save_contact" type="submit" value="Save" />
</form>

<a href="{% url 'contacts-list' %}">back to list</a>
```

Несколько новых штук на заметку:

- `form` из контекста — это Django Form для нашей модели. Так как мы не указали ни одного своего, Django создал один для нас. Как заботливо;
- если бы мы просто написали `{{ form }}` мы бы получили табличные ряды; но мы добавляем `.as_ul`, что заставляет выводить поля ввода как элементы нумерованного списка. Попробуйте вместо этого использовать `.as_p` и посмотреть что из этого получится;
- когда мы выводим форму (*пр.: автоматически сгенерированную*) выводятся только наши поля, но не обрамляющий форму тег или кнопка отправки `</>`, так что мы добавим их в шаблон сами; шаблонный тег `{% csrf_token %}` вставляет скрытое поле, по которому Django определяет, что запрос пришел с вашего проекта и что это не межсайтовая подделка запроса (CSRF). Попробуйте не включать его в шаблон: вы все еще будете иметь доступ к странице, но как только вы попытаетесь отправить форму — вы получите ошибку; мы используем шаблонный тег `url` для генерирования ссылки назад к списку контактов. Замете, что `contacts-list` — это имя нашего представления, которое мы указали в настройках URL. Используя `url` вместо полного пути, нам не придется беспокоиться про битые ссылки. `url` в шаблоне является эквивалентом `reverse` в коде Python.

Теги: [python](#), [django](#), [effective django](#), [эффективный django](#), [Nathan Yergler](#)

Хабы: [Python](#), [Django](#)

Редакторский дайджест

Присылаем лучшие статьи раз в месяц

Электронпочта



22

Карма

0

Рейтинг

Kyrylo Petrov [@bjakushka](#)

Software Engineer

Комментарии 9



karech 20.11.2014 в 14:45

Все таки очень долго думал, что же такое «Представление». Неужли принято так их называть?



+1

[Ответить](#)



jafte 20.11.2014 в 14:57



В простонародье «вьюшки» (View), еще называют иногда «вид» ([Model-View-Controller](#), например) Во всех документациях, что читал, они именно «представления».



+5

[Ответить](#)



random1st 20.11.2014 в 18:32



Собственно говоря, представления в Django скорее не представления, поскольку так скорее можно назвать темплейты.



0

[Ответить](#)



jafte 20.11.2014 в 18:34



Кстати да, это отличный повод, чтобы запутаться в терминологии Django и классического MVC.



0

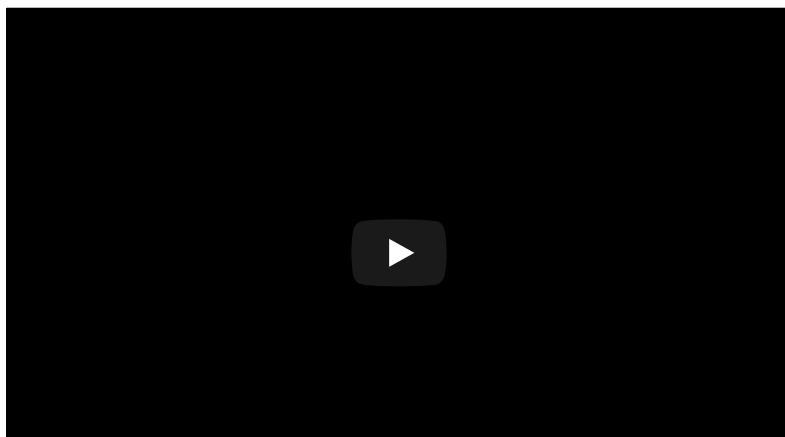
[Ответить](#)



aruseni 21.11.2014 в 19:11



Про то, чем отличается MTV, то есть используемый в Django подход, хорошо рассказал Jacob Kaplan-Moss во время выступления на конференции Google в 2006.



Конкретно об этом на 12:35. Но настоятельно рекомендую смотреть с самого начала — чувак действительно очень крутой и очень интересно говорит.

0 Ответить

 **Ruckus** 20.11.2014 в 15:04

Ну думаю это тем или иным образом относится к [этому](#).

А вообще термин «представление» по-моему общепринят и используется очень часто, когда речь заходит о пользовательском интерфейсе. Хотя возможно я не прав.

+2 Ответить

 **korkholeh** 20.11.2014 в 16:31

ну да

0 Ответить

 **Tarvitz** 20.11.2014 в 15:28

В качестве конструктива можно еще посоветовать в связке с pip (virtualenv) использовать менеджеры для фронтенд (js) решений, bower или еще что-нибудь — это на ваш собственный выбор.

+2 Ответить

 **Jazzina** 26.11.2014 в 09:51


Мне одному советы от effective django кажутся, мягко говоря, странноватыми?

Зачем использовать статический модуль, если в 99.9% случаев в приложении будет не только бутстрап и нужно будет компоновать (а то и сжимать) media через, например, mediagenerator?

0 Ответить

Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

Реклама




Единая витрина технологий Сбера

Создавайте цифровые продукты с продвинутыми IT-инструментами

[Присоединиться](#)

ПАО Сбербанк. Генеральная лицензия Банка России на осуществление банковских операций № 1481 от 11.08.2015.



ПОХОЖИЕ ПУБЛИКАЦИИ

3 июня в 21:35

Автоматизация тестирования на Python. Шесть способов тестировать эффективно

+8 17K 86 4 +4

3 июня 2020 в 18:57

Django: один пользователь для всего

+4 7.6K 51 11 +11

15 октября 2014 в 15:50

Эффективный Django. Часть 1

+35

227K

784

15

МИНУТОЧКУ ВНИМАНИЯ

Разместить



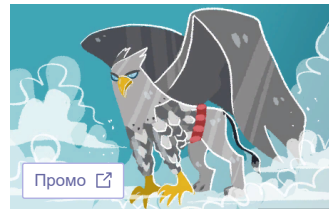
Мегапост

Инфобез от первого лица:
снимаем проклятие знания



Мегапост

Ленты, козлы и костыли:
выступления фронтов и бэков



Промо

Одной ногой в облаке: гибридные
подходы к разработке

ВАКАНСИИ

Python developer (Django)

от 180 000 до 220 000 Р · Кокос Group · Можно удаленно

Django / Python разработчик

от 100 000 Р · SkyDNS · Екатеринбург · Можно удаленно

Backend разработчик Python / Django / middle

от 2 000 \$ · Praktika · Можно удаленно

Python(Django) разработчик

от 100 000 до 250 000 Р · Новая Школа · Можно удаленно

Разработчик Python/Django

от 90 000 до 150 000 Р · JustWork · Можно удаленно

[Больше вакансий на Хабр Карьере](#)

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 14:05

Паразит, дарующий своим хозяевам «вечную молодость» и многократно удлиняющий срок их жизни

+144

57K

50

70 +70

вчера в 16:00

Сеульская агломерация: борьба с последствиями стремительного роста

+47

5.2K

29

19 +19

вчера в 22:34

Солнечная электростанция в квартире: собственный опыт + варианты реализации

+37

14K

50

26 +26

вчера в 16:04

Собираем DOS 2.11 из исходников 80-х годов

+32

4.3K

23

3 +3

вчера в 21:53

Бесплатные ресурсы для инди-разработчиков

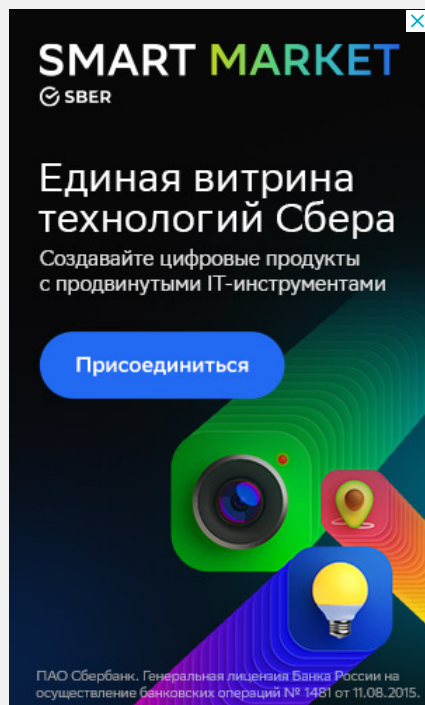
+29

2.1K

83

4 +4

Реклама



ЧИТАЮТ СЕЙЧАС

Паразит, дарующий своим хозяевам «вечную молодость» и многократно удлиняющий срок их жизни

57K

70 +70

Солнечная электростанция в квартире: собственный опыт + варианты реализации

14K

26 +26

Взлом Госуслуг: утечка исходного кода

69K

88 +88

Пользователь GitHub опубликовал код инструментов для скачивания фильмов с онлайн-кинотеатров

11K

8 +8

Чувствовал себя порнозвездой: выпускники Elbrus Bootcamp – о диких собеседованиях

19K

59 +59

«Мир» на китах: что мы узнали о кешбэке

Мегапост

РАБОТА

Python разработчик

168 вакансий

Data Scientist

97 вакансий

Django разработчик

57 вакансий

Ваш аккаунт

[Войти](#)
[Регистрация](#)

Разделы

[Публикации](#)
[Новости](#)
[Хабы](#)
[Компании](#)
[Авторы](#)
[Песочница](#)

Информация

[Устройство сайта](#)
[Для авторов](#)
[Для компаний](#)
[Документы](#)
[Соглашение](#)
[Конфиденциальность](#)

Услуги

[Реклама](#)
[Тарифы](#)
[Контент](#)
[Семинары](#)
[Мегапроекты](#)



[Настройка языка](#)

[О сайте](#)

[Техническая поддержка](#)

[Вернуться на старую версию](#)

© 2006–2021 «Habr»