

 bjakushka 15 октября 2014 в 15:50

Эффективный Django. Часть 1

Python *, Django *

Из песочницы

Tutorial



Представляю вам перевод 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 свое доброе дело сам. Этот цикл статей, как мне думается, будет полезен веб-разработчикам, которые делают только первые шаги в изучении Django.

Оглавление

- 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

Введение ↑

Django — это популярный, мощный фреймворк на языке Python. Он имеет множество "батареек", и позволяет сразу начать разработку. Однако вся эта мощь означает, что вы можете написать низкопробный код, который будет казаться рабочим. Так что же подразумевается под Эффективным Django? Под Эффективным Django будем понимать использование Django таким образом, чтобы написанный код был *связным*, *тестируемым* и *масштабируемым*. Что же каждое из этих слов значит?

"Связный" код — это код, который сосредоточен на выполнении одной вещи, только одной единственной вещи. Это значит, что когда вы пишете функцию или метод — написанный вами код должен делать что-то одно и делать это хорошо.

Это непосредственно относится к написанию *тестируемого кода*: код, который делает много вещей, достаточно часто является чересчур сложным для тестирования. Когда я ловлю себя на мысли: «Хорошо, этот кусок кода слишком сложен, чтобы писать для него тесты — это просто не стоит потраченных усилий» — вот сигнал к тому, чтобы вернуться назад и сосредоточиться на упрощении. *Тестируемый код* — такой код, который позволяет просто писать для него тесты; код,

в котором легко найти проблемы.

И наконец, мы хотим писать *масштабируемый код*. Это означает не просто масштабировать его в терминах исполнения, но так же увеличивать в терминах команды и командного понимания. Хорошо протестированные приложения проще для понимания другими (и проще для изменения ими), что подразумевает большую возможность улучшить ваше приложение, путем добавления новых инженеров.

Моя цель — убедить вас в важности этих принципов, и предоставить примеры того, как следуя им, построить более стойкое Django-приложение. Я собираюсь последовательно пройти через процесс построения приложения для управления контактами, рассказывая про решения и стратегию тестирования, которые я использую.

Эти документы являются сочетанием заметок и примеров подготовленных для PyCon 2012, PyOhio 2012, и PyCon 2013, а также для web-разработки [Eventbrite](#). Я все еще работаю над объединением их в один документ, но надеюсь вы найдете их полезными.

Примеры кода для этого руководства доступны на [github'e](#). Отзывы, предложения и вопросы можете присылать на nathan@yergler.net.

Этот документ доступен на [сайте](#), а также в форматах [PDF](#) и [EPub](#).

Видео этого руководства с PyCon можно посмотреть на [YouTube](#).

Глава 1. Приступая к работе ↑

1.1. Ваша среда разработки

Когда ведется разговор о вашей среде разработки, существует три важных факта, которые необходимо иметь в виду: изоляция, предопределенность и сходство. Каждый из них важен и все они взаимодействуют друг с другом согласованно.

Изоляция означает, что вы не сможете случайно воспользоваться инструментами или пакетами установленными вне вашего окружения. Это особенно важно, когда подобное происходит с чем-то, похожим на пакеты Python с расширениями написанными на C: если вы используете что-то установленное на системном уровне и не знаете об этом, то при развертывании или распространении своего кода вы можете обнаружить, что он работает не так как предполагалось. Инструменты наподобие [virtualenv](#) могут помочь создать нечто похожее на изолированную среду.

Ваша среда предопределена, если вы уверены в том, на какую версию ваших зависимостей вы полагаетесь и сможете ли вы наверняка воспроизвести системное окружение.

И на конец, сходство с производственной средой или средой сервера разработки означает, что везде установлена одна и та же операционная система (возможно даже одинаковый выпуск) и вы используете одинаковые инструменты как для конфигурирования вашей среды разработки, так и для конфигурирования вашей производственной среды. Это отнюдь не необходимость, но если вы строите большое и сложное программное обеспечение — сходство будет полезно для уверенности в том, что все проблемы, которые вы можете увидеть на «боевом» сервере, воспроизводимы в той среде, где вы ведете разработку. К тому же сходство ограничивает область исследования вашего кода.

1.1.1. Изоляция

- Мы хотим избежать использования неизвестных зависимостей или неизвестных версий.
- [virtualenv](#) предоставляет простой путь для работы над проектом, без использования системных site-packages.

1.1.2. Предопределенность

- Предопределенность означает управление зависимостями.

- Выберите один из инструментов и используйте как при разработке, так на «боевом» сервере:
 - `pip` и специальные файлы зависимостей;
 - `buildout`;
 - `install-requires` в `setup.py`.
- Определите точные версии зависимостей.

Вы можете точно определить версии используя либо версию пакета на PyPI, либо же определенную ревизию (SHA в `git`, номер ревизии в `Subversion` и т. д.). Это гарантирует вам возможность получить в точности те же версии пакетов, которые вы используете при тестировании.

1.1.3. Сходство

- Работа в среде, похожей на ту, где вы будете разворачивать ваше приложение и пытаться обнаружить проблемы.
- Если вы разрабатываете что-то, требующее дополнительных сервисов — сходство становится еще более важным.
- `Vagrant` — это инструмент для управления виртуальными машинами, позволяющий вам легко создавать окружение отделенное от вашего повседневного окружения.

1.2. Настройка вашего окружения

1.2.1. Создание чистого рабочего пространства

Примечание переводчика:

Для начала создадим каталог (`tutorial`), в котором будем работать:

```
~$ mkdir tutorial
~$ cd tutorial
~/tutorial$ mkdir venv project
```

В каталоге `venv` будет находится наше виртуальное окружение, а в каталоге `project` — Django-проект

```
~/tutorial$ virtualenv --prompt="(venv:tutorial)" ./venv/

New python executable in ./venv/bin/python
Installing setuptools.....done.
Installing pip.....done.

~/tutorial$ source ./venv/bin/activate
(venv:tutorial)~/tutorial$
```

1.2.2. Создание файла зависимостей

Создайте файл `requirements.txt` в директории `tutorial` с единственной строкой (зависимостью) в нем:

```
Django==1.6.7
```

Примечание переводчика:

В случае, если вы хотите использовать последнюю версию Django (1.7 — на момент написания перевода) — вместо строки `Django==1.6.7` оставьте просто `Django` — `pip` установит последнюю доступную версию.

1.2.3. Установка зависимостей

А теперь мы можем использовать `pip` для установки зависимостей:

```
(venv:tutorial)~/tutorial$ pip install -U -r requirements.txt

Downloading/unpacking Django==1.6.7 (from -r requirements.txt (line 1))
  Downloading Django-1.6.7.tar.gz (6.6MB): 6.6MB downloaded
  Running setup.py egg_info for package Django

    warning: no previously-included files matching '__pycache__' found under directory '*'
    warning: no previously-included files matching '*.py[co]?' found under directory '*'
Installing collected packages: Django
  Running setup.py install for Django
    changing mode of build/scripts-2.7/django-admin.py from 644 to 755

    warning: no previously-included files matching '__pycache__' found under directory '*'
    warning: no previously-included files matching '*.py[co]?' found under directory '*'
    changing mode of /home/nathan/p/edt/bin/django-admin.py to 755
Successfully installed Django
Cleaning up...
```

1.3. Начало проекта Django

Когда здание находится в процессе постройки, строительные леса часто используются для поддержания структуры до того как строительство будет завершено. Строительные леса могут быть временными или они могут служить частью фундамента здания, но несмотря на это, они представляют некоторую поддержку когда вы только начинаете работу.

Django, как и многие web-фреймворки, представляет скаффолдинг для вашей разработки. Это происходит при помощи принятия решений и предоставления отправной точки для вашего кода, что позволяет вам сосредоточиться на проблеме, которую вы пытаетесь решить, а не на том, как разобрать HTTP-запрос. Django предоставляет скаффолдинг как для работы с HTTP, так и для работы с файловой системой.

HTTP-скаффолдинг управляет, например, преобразованием HTTP-запроса в объект языка Python, а также предоставляет инструменты для более простого создания серверных ответов. Скаффолдинг файловой системы иной: это набор соглашений по организации вашего кода. Эти соглашения облегчают добавление новых инженеров в проект, так как инженеры (гипотетически) уже понимают как организован код. В терминах Django, проект — это конечный продукт, и он объединяет внутри себя одно или несколько приложений. В Django 1.4 было изменено то, как проекты и приложения размещаются на диске, что облегчило разъединение и повторное использование приложений в разных проектах.

1.3.1. Создание проекта

Django устанавливает в систему скрипт `django-admin.py` для обработки задач скаффолдинга. Для создания файлов проекта используется задача `startproject`. Мы определим имя проекта и имя директории, в которой хотим разместить проект. Так как мы уже находимся в

Все потоки Разработка Администрирование Дизайн Менеджмент Маркетинг Научпоп 🔍

Примечание переводчика:

Перейдем директорию `~/tutorial/project/` и в дальнейшем будем работать только из этой директории (под `$` далее будем подразумевать `~/tutorial/project/$`):

```
(venv:tutorial)~/tutorial/$ cd project
```

```
(venv:tutorial)$ django-admin.py startproject addressbook .
```

Созданный проект имеет следующую структуру

```
manage.py
./addressbook
  __init__.py
  settings.py
  urls.py
  wsgi.py
```

1.3.2. Скаффолдинг проекта

- `manage.py` — является ссылкой на скрипт `django-admin`, но с уже предустановленными переменными окружения, указывающими на ваш проект, как для чтения настроек оттуда, так и для управления им при необходимости;
- `settings.py` — здесь находятся настройки вашего проекта. Файл уже содержит несколько разумных настроек, но база данных не указана;
- `urls.py` — содержит URL'ы для [маппирования \(отображения\)](#) представлений: мы вскоре (в *дальнейших главах*) поговорим об этом подробнее;
- `wsgi.py` — это [WSGI обёртка](#) для вашего приложения. Этот файл используется сервером разработки Django и возможно другими контейнерами, такими как `mod_wsgi`, `uwsgi` и др. на «боевом» сервере.

1.3.3. Создание приложения

```
(venv:tutorial)$ python ./manage.py startapp contacts
```

Созданное приложение имеет следующую структуру:

```
./contacts
  __init__.py
  models.py
  tests.py
  views.py
```

- Начиная с Django 1.4, приложения размещаются внутри пакета с проектом. Это замечательное улучшение, особенно когда приходит время разворачивать проект на «боевом» сервере;
- `models.py` будет содержать Django ORM-модели для вашего приложения;
- `views.py` будет содержать код представлений;
- `tests.py` будет содержать написанные вами модульные и интеграционные тесты.
- *Django 1.7: `admin.py` будет содержать модель для административного интерфейса.*
- *Django 1.7: `migrations/` содержит файлы миграций*

Примечание переводчика:

На текущий момент наша директория `~/tutorial/` содержит файл зависимостей (`requirements.txt`), директорию с виртуальным окружением (`venv/`), один проект (`project/addressbook`), одно приложение (`project/contacts`) и имеет следующее содержание:

```
~/tutorial/
  requirements.txt
  venv/
  ...
  project/
    manage.py
    addressbook/
      __init__.py
      settings.py
      urls.py
      wsgi.py
    contacts/
      __init__.py
      models.py
      tests.py
      views.py
```

Глава 2. Используем модель

2.1. Конфигурирование базы данных

Django поддерживает «из коробки» MySQL, PostgreSQL, SQLite3 и Oracle. [SQLite3](#) входит в состав Python начиная с версии 2.5, так что мы будем использовать его в нашем проекте (для простоты). Если вы хотите, к примеру, использовать MySQL, то нужно добавить [mysql-python](#) в ваш `requirements.txt`.

Для того чтобы в качестве базы данных использовать SQLite, отредактируйте определение `DATABASES` в файле `addressbook/settings.py`. Файл `settings.py` содержит настройки Django для нашего проекта. В нем есть несколько настроек, которые вы обязаны указать — например `DATABASES` — а так же другие, необязательные, настройки. Django устанавливает некоторые настройки сам, когда генерирует проект. Документация содержит [полный список настроек](#). К тому же вы можете добавить свои собственные настройки, если это необходимо.

Для использования SQLite нам нужно указать движок (`ENGINE`) и имя базы (`NAME`). SQLite интерпретирует имя базы как имя файла для базы данных:

```
DATABASES = {
    'defaults': {
        'ENGINE': 'django.db.backends.sqlite3', # 'postgresql_psycopg2', 'mysql', 'sqlite3'
        'NAME': os.path.join(BASE_DIR, 'address.db'),
        'USER': '', # Not used with sqlite3.
        'PASSWORD': '', # Not used with sqlite3.
        'HOST': '', # Set to empty string for localhost. Not used with sqlite3.
        'PORT': '', # Set to empty string for default. Not used with sqlite3.
    }
}
```

Заметьте, что движок базы данных указан строкой, а не прямой ссылкой на объект Python. Это сделано по той причине, что файл настроек должен быть легко импортирован не вызывая сторонних эффектов. Вы должны избегать добавления вызовов `import` в этот файл.

Вам редко придется непосредственно импортировать файл настроек: Django импортирует его за вас, и делает настройки доступными как `django.conf.settings`. Вы, как правило,

импортируете настройки из `django.conf` :

```
from django.conf import settings
```

2.2. Создание модели

Модели Django отображают (грубо говоря) таблицы базы данных, и предоставляют место для инкапсулирования бизнес-логики. Все модели являются наследниками базового класса `Model` и содержат поля определений. Давайте создадим простую модель `Contacts` для нашего приложения в файле `contacts/models.py` :

```
from django.db import models

class Contact(models.Model):

    first_name = models.CharField(
        max_length=255,
    )
    last_name = models.CharField(
        max_length=255,
    )

    email = models.EmailField()

    def __str__(self):

        return ' '.join([
            self.first_name,
            self.last_name,
        ])
```

Django предоставляет **набор полей** для отображения типов данных и различных правил валидации. К примеру, `EmailField`, который мы использовали, является отображением на колонку с типом `CharField`, но добавляет валидацию данных.

После того, как вы создали модель, необходимо дополнить вашу базу данных новыми таблицами. Команда Django `syncdb` смотрит установленные модели и создает (если нужно) таблицы для них:

Примечание переводчика:

Django предложит создать суперпользователя для админки, которая включена в этой версии по умолчанию. Воспользуйтесь его предложением.

Примечание переводчика:

С версии Django 1.7 во фреймворк добавлена нативная поддержка **миграций** и команда `syncdb` объявлена устаревшей. Так что будьте так любезны, воспользуйтесь командой `migrate` вместо `syncdb`.

```
(venv:tutorial)$ python ./manage.py syncdb
```

```
Creating tables ...
Creating table django_admin_log
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_groups
Creating table auth_user_user_permissions
Creating table auth_user
Creating table django_content_type
```

Creating table django_session

```
You just installed Django's auth system, which means you don't have any superusers defined
Would you like to create one now? (yes/no): yes
Username (leave blank to use 'bjakushka'):
Email address:
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
installing indexes ...
Installed 0 object(s) from 0 fixture(s)
```

```
(venv:tutorial)$
```

Примечание переводчика:

Если вы используете Django версии 1.7 и выше — вывод будет следующий:

```
(venv:tutorial)$ python ./manage.py migrate

Operation to perform:
  Apply all migrations: admin, contenttypes, auth, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK

(venv:tutorial)$
```

Однако нашей таблицы с контактами нигде не видно. Причина этого состоит в том, что нам нужно еще указать проекту использовать приложение.

Настройка `INSTALLED_APPS` содержит список приложений, используемых в проекте. Этот список содержит в себе строки, которые отображают пакеты Python. Django будет импортировать каждый из указанных пакетов, а потом смотреть модуль `models`. Давайте добавим наше приложение `contacts` в настройки проекта (`addressbook/settings.py`):

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'contacts',
)
```

После этого запустите `syncdb` снова:

Примечание переводчика:

Для Django версии 1.7 и выше вам нужно будет запустить сначала команду `makemigrations` — для создания миграций на основе изменений в моделях, а после этого выполнить команду `migrate` — для того чтобы применить созданные миграции.

```
(venv:tutorial)$ python ./manage.py syncdb

Creating tables ...
Creating table contacts_contact
```



```
Installing custom SQL ...
Installing indexes ...
Installed 0 object(s) from 0 fixture(s)

(venv:tutorial)$
```

Примечание переводчика:

Вывод для Django 1.7 и выше:

```
(venv:tutorial)$ python ./manage.py makemigrations

Migrations for 'contacts':
  0001_initial.py:
    - Create model Contact

(venv:tutorial)$ python ./manage.py migrate

Operation to perform:
  Apply all migrations: admin, contenttypes, sessions, auth, contacts
Running migrations:
  Applying contacts.0001_initial... OK

(venv:tutorial)$
```

Заметьте, что Django создает таблицу с именем `contacts_contact` : по умолчанию Django дает таблицам имена используя комбинацию имени приложения и имени модели. Вы можете изменить это с помощью опций модели [Meta](#).

2.3. Взаимодействие с моделью

Теперь, когда модель синхронизирована с базой данных мы можем взаимодействовать с ней используя интерактивную оболочку:

```
(venv:tutorial)$ python ./manage.py shell
Python 2.7.3 (default, Mar 14 2014, 11:57:14)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from contacts.models import Contact
>>> Contact.objects.all()
[]
>>> Contact.objects.create(first_name='Nathan', last_name='Yergler')
<Contact: Nathan Yergler>
>>> Contact.objects.all()
[<Contact: Nathan Yergler>]
>>> nathan = Contact.objects.get(first_name='Nathan')
>>> nathan
<Contact: Nathan Yergler>
>>> print nathan
Nathan Yergler
>>> nathan.id
1
```

Здесь использовались несколько новых штук. Во-первых, команда `manage.py shell` запускает для нас интерактивную оболочку Python'a с правильно установленными путями для Django. Если вы попытаетесь запустить интерпретатор Python и просто импортировать ваше приложения, будет выброшено исключение, потому что Django не знает, какие настройки использовать, и не может отобразить экземпляры модели на базу данных.

Во-вторых, здесь использовалось свойство `objects` нашей модели. Это [менеджер модели](#). Так, если один экземпляр модели является аналогией для строки в базе, то менеджер модели —

аналогией для таблицы. По умолчанию менеджер модели предоставляет функциональность запросов и может быть настроен. Когда мы вызываем `all()`, `filter()` или сам менеджер, возвращается объект `QuerySet`. `QuerySet` является итерируемым объектом и загружает данные из базы по необходимости.

И последнее — выше использовалось поле с именем `id`, которое мы не определяли в нашей модели. Django добавляет это поле как первичный ключ для модели, но только в том случае если вы сами не определили какое поле будет **первичным ключом**.

2.4. Написание тестов

В нашей модели определен один метод, `__str__`, так что настало время писать тесты. Метод `__str__` будет использоваться всего лишь в нескольких местах, и, вполне возможно, полностью будет показан конечному пользователю. Для этого метода стоит написать тест, пока мы понимаем как он работает. Django создал файл `tests.py` когда создавал приложение, так что мы добавим первый тест в этот файл, приложения `contacts`.

```
from django.test import TestCase

from contacts.models import Contact

class ContactTests(TestCase):
    """Contact model tests."""

    def test_str(self):

        contact = Contact(first_name='John', last_name='Smith')
        self.assertEqual(
            str(contact),
            'John Smith',
        )
```

Вы можете запустить тесты для вашего приложения используя команду `manage.py test`:

```
(venv:tutorial)$ python ./manage.py test
```

Если вы запустите это, то увидите что выполнилось около 420 тестов. Это удивляет, так как мы написали только один. Произошло это потому, что по умолчанию Django запускает тесты для всех установленных приложений. Когда вы добавляли приложение `contacts` в наш проект, то могли увидеть, что там по умолчанию были добавлены несколько встроенных приложений Django. Дополнительные 419 тестов были взяты оттуда.

Примечание переводчика:

В нашем случае (при использовании версии Django 1.6.7) предыдущий абзац несколько устарел: запустится только один тест — тот который мы создали. Вывод команды будет такой как указано ниже.

Если же вы захотите запустить тесты для определенного приложения — укажите имя приложения в команде:

```
(venv:tutorial)$ python manage.py test contacts

Creating test database for alias 'default'...
.
-----
Ran 1 tests in 0.001s

OK
Destroying test database for alias 'default'...
```

```
(venv:tutorial)$
```

Еще одна интересная вещь на заметку, прежде чем двигаться дальше — первая и последняя строка вывода: `Creating test database` и `Destroying test database`. Некоторым тестам необходим доступ к базе данных, и поскольку мы не хотим мешать тестовые данные с «реальными» (по разным причинам, не последней из которых является предопределенность), Django услужливо создает тестовую базу для нас, прежде чем запускать тесты. По существу, создается новая база данных, и потом запускается `syncdb` для нее. Если тестовый класс является потомком класса `TestCase` (как у нас), Django так же сбросит данные в значения по умолчанию после запуска каждого теста, так что изменения в одном из тестов не затронут другие.

2.5. Резюме

- Модель определяет поля в таблице, и содержит бизнес-логику.
- Команда `syncdb` создает таблицы в вашей базе данных из моделей. В Django версии 1.7 и выше вместо команды `syncdb` необходимо использовать сначала команду `makemigrations` — для создания миграций, а после этого команду `migrate` — для внесения изменений в базу.
- Менеджер модели позволяет вам оперировать коллекциями экземпляров: запросы, создание и т. д..
- Пишите модульные тесты для методов, которые вы добавили в модель.
- Команда управления `test` запускает модульные тесты на выполнение.

Примечание переводчика:

Для того чтобы протестировать наше, пока еще пустое, приложение нужно выполнить следующую команду:

```
(venv:tutorial)$ python ./manage.py runserver 0.0.0.0:8080
```

Это запустит встроенный сервер, функционал которого любезно предоставляет нам Django. В параметрах после `runserver` указывается ip-адрес и порт, который будет слушаться работающим сервер. В нашем случае сервер будет принимать запросы от всех ip-адресов при обращении на 8080 порт.

Я использую для разработки домашний сервер с внутренним IP 192.168.1.51. Так что для того что-бы увидеть результат работы сервера разработки в браузере я захожу по адресу <http://192.168.1.51:8080/>. Вы же должны подставить адрес своего сервера.



Как думаете, нужно ли продолжать перевод остальных глав? Оказался ли перевод для Вас полезен?

Буду рад конструктивной критике в комментариях.

Об ошибках и неточностях перевода прошу сообщать в личном сообщении.

Использованное в начале поста изображение создано как вариация изображения пользователя [@MaGlc2laNTern](#)

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

Хабы: [Python](#), [Django](#)

Редакторский дайджест

Присылаем лучшие статьи раз в месяц



22

Карма

0

Рейтинг

Kyrylo Petrov @bjakushka

Software Engineer

Комментарии 15



horridum 15.10.2014 в 13:12

Нужно) Кому надо — те конечно и оригинал уже прочитали, но в целом — годно, продолжайте, пожалуйста.



+6

Ответить



voron3x 15.10.2014 в 13:50

Автор, присоединитесь пожалуйста к команде djbook.ru/. Зачем повторять пройденное? ИМХО лучше сосредоточиться на новых материалах.



+4

Ответить



radz 15.10.2014 в 22:10

Да, присоединяйтесь, мы добрые :)



+2

Ответить



lesha_firs 15.10.2014 в 13:51

Статья конечно полезная, но зачем? уже есть довольно хорошая русскоязычная статья, причем по django 1.7 а у вас 1,6. djbook.ru/rel1.7/



+1

Ответить



bjakushka 15.10.2014 в 15:52

Хороших статей много не бывает, разве нет?

Насчет версий Django — данная статья, по-моему, вполне актуальна (кроме некоторых моментов) и для версии 1.7. Но спасибо за комментарий — различия в версиях опишу в примечаниях переводчика.

P.S. Что бы не писать в отдельном комментарии: подскажите пожалуйста, как можно указать у статьи, что это перевод и добавить автора оригинала?



+4

Ответить



ShpuntiK 16.10.2014 в 14:50

При создании поста надо выбрать «Хочу разместить **перевод**», нажав на ссылку, подчеркнутую пунктиром.



0

Ответить



bjakushka 16.10.2014 в 23:07

Это верно в том случае если я создам новый пост (после получения инвайта)

А как мне сделать перевод из **этого** поста (он из песочницы)?

0 Ответить

NCNecros 17.10.2014 в 15:39

Никак, первый пост будет только таким. А Вы планируете делать перевод остальных тем с сайта? В оглавлении я увидел только первую тему.

0 Ответить

bjakushka 17.10.2014 в 16:28

Спасибо за ответ.

Да, планирую. Добавлю их в оглавление чуть позже.

0 Ответить

johniek_comp 15.10.2014 в 14:22

написанный вами код должен делать что-то одно и делать это хорошо

философия linux? автор хорош)

+1 Ответить

overmes 15.10.2014 в 17:09

философия здравого смысла

+3 Ответить

lpeacocks 15.10.2014 в 19:15

Отлично! Только вот начал изучать джангу.

0 Ответить

dunmaksim 16.10.2014 в 10:32

Изучайте по djbook.ru, как выше сказали — очень много статей уже переведено на русский, всегда можно и оригинал посмотреть на английском.

0 Ответить

wunderwaffel 16.10.2014 в 00:51

Пока не понятно, чем описанное эффективнее других подходов. Пошаговое следование учебнику с официального сайта...

+1 Ответить

EgorLyutov 16.10.2014 в 09:56

Спасибо за перевод, думаю стоит продолжать.

+2 Ответить

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

Реклама



Обучение Питону с нуля 18+

Курс «Python-разработчик» 18+

Станьте backend-разработчиком с нуля. Гарантия трудоустройства. Приступайте сейчас! 18+

practicum.yandex.ru

ОТКРЫТЬ

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

20 ноября 2014 в 13:29

Эффективный Django. Часть 2

👍 +19 👁 64K 📖 338 💬 9

МИНУТОЧКУ ВНИМАНИЯ

Разместить



Учимся продуктовой мудрости на мемах и идеях от разработов



Админ, Агент или Монстр? Облачный квест



Верни мне мой 1998-й: об истоках развития Wi-Fi

КУРСЫ



Углубленный курс по Python

12 января 2022 · 55 080 Р · GeekBrains



Программирование на Python для детей

12 января 2022 · 14 700 Р · GeekBrains



Python-разработчик с нуля

17 января 2022 · 64 800 Р · Нетология



Python для анализа данных

18 января 2022 · 27 000 Р · Нетология



Python-разработчик плюс

11 января 2022 · 168 000 Р · Яндекс.Практикум

[Больше курсов на Хабр Карьере](#)

ЛУЧШИЕ ПУБЛИКАЦИИ ЗА СУТКИ

вчера в 22:34

Солнечная электростанция в квартире: собственный опыт + варианты реализации

 +51  25K  79  65 +65


сегодня в 03:14

Технология Вялых Проектов

 +43  5.9K  26  25 +25

вчера в 21:53

Бесплатные ресурсы для инди-разработчиков

 +36  3.6K  123  4 +4

вчера в 20:52

РНР дайджест № 218 (13 – 27 декабря 2021)

 +34  5.1K  17  9 +9

вчера в 19:36

Fuchsia OS от Google: от никому не известного проекта до ОС для смартфонов Samsung и Google



 +33  13K  13  20 +20

ЧИТАЮТ СЕЙЧАС

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

 73K  84 +84

Расчетный листок, полученный римским солдатом X легиона. 2000 лет пролетело — ничего не изменилось...

 7.3K  36 +36

Гипотеза Коллатца — самый крутой математический фокус всех времён

 12K  16 +16

Солнечная электростанция в квартире: собственный опыт + варианты реализации

 25K  65 +65

Взлом Госуслуг: утечка исходного кода

 80K  106 +106

Wi-Fi в большом городе: телемедицина, футбол и умные светофоры

Мегапост

РАБОТА

Python разработчик

168 вакансий

Django разработчик

57 вакансий

Data Scientist

98 вакансий

[Все вакансии](#)

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»