ТЕХНОАТОМ

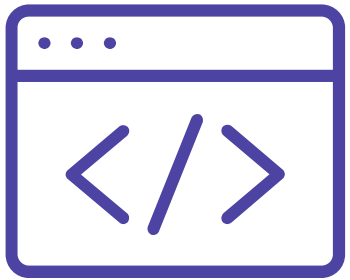
Веб-программирование Python

Лекция 4. Django, шаблоны и формы

Михалев Олег

Сегодня

- Представления
- Запросы и ответы
- Шаблоны
- Формы



Представления Django - вызываемый объект, который принимает запрос пользователя в качестве входного параметра и всегда возвращает ответ.

```
1. from django.http import HttpResponse


2. def hello_world(request):
3.     return HttpResponse('<h1>Hello, World!</h1>')
```

Представления (как вызываемый объект) в Django можно декорировать.

```
1. from django.views.decorators.http import (
2.     require_http_methods,
3.     require_GET,
4.     require_POST,
5.     ...
6. )
```

ТЕХНОАТОМ

Маршрутизатор Django позволяет передавать в представление позиционные параметры из URL.

```
1. urlpatterns = [
2.     url(r'^charges/(/d{4})-(/d{2})/$', views.charges_by_month),
3. ]
4. ...
5. def charges_by_month(request, year, month):
6.     ...
```

ТЕХНОАТОМ

Маршрутизатор Django позволяет передавать в представление именованные параметры из URL.

```
1. urlpatterns = [
2.     url(r'^charges/(?P<year>/d{4})-(?P<month>/d{2})/$',
   views.charges_by_dates),
3.     url(r'^charges/(?P<year>/d{4})/$', views.charges_by_dates),
4. ]
5. ...
6. def charges_by_dates(request, year, month=None):
7.     ...
```

Именованные группы переданы как именованные параметры, неименованные - как позиционные параметры.

Значения найденных параметров - значения групп совпадения (метода **match**) регулярного выражения URL, а значит параметры представления всегда являются строками.

Маршруты можно именовать, этим целям служат дополнительные параметры. Параметр **name** (имя представления) для **url**. Параметр **namespace** (пространство имен) для **include**.

```
1. from django.conf.urls import url, include
2. urlpatterns = [
3.     url(r'^account/$', account.views.current_state, name='account_view')
4.     url(r'^charges/', include(charges.urls, namespace='charges')),
5. ]
```

ТЕХНОАТОМ

Всегда можно получить URL представления по его имени.

```
1. from django.core.urlresolvers import reverse

2. reverse('account_view')
3. reverse('charges:list_by_dates', args, kwargs)
```

Когда пользователь запрашивает приложение - Django создает экземпляр класса **HttpRequest**, который передается в представление. Каждое представление должно возвращать экземпляр класса **HttpResponse**.

```
1. from django.http import HttpResponse


2. def hello_world(request):
3.     return HttpResponse('<h1>Hello, World!</h1>')
```

HttpRequest - запрос пользователя

scheme	схема (протокол)
body	тело сообщения
path	путь
method	http-метод

HttpRequest - запрос пользователя

GET	get-параметры (из query)
POST	post-параметры (из тела сообщения)
FILES	загружаемые файлы
META	словарь переменных окружения

Класс **QueryDict**, используемый как контейнер http-параметров, позволяет проводить синтаксический разбор параметров запроса, переданных строкой (не стоит забывать, что ключи и конечные значения параметров также будут являтся строкой).

HttpRequest совместно с параметрами, полученными на этапе маршрутизации, содержит всю необходимую для обработки запроса информацию.

```
1. @require_POST
2. def create_charge(request, account_id):
3.     value = request.POST.get('value')
4.     date = request.POST.get('date')
5.     ...
6.     return HttpResponse(...)
```

ТЕХНОАТОМ

HttpResponse - ответ приложения

```
1. response = HttpResponse(content, content_type='text/html', status=200)
2. ...
3. response = HttpResponse(content_type='text/html', status=200)
4. response.write(part_of_content)
```

Заголовки ответа устанавливаются напрямую в экземпляр возвращаемого представлением **HttpResponse**.

```
1. response = HttpResponse(content, content_type='text/xml', status=200)
2. response['Content-Disposition'] = 'attachment; filename="dump.xml"'
```

Для упрощения работы в **django.http** представлены и дочерние классы **HttpResponse**

HttpResponseRedirect
HttpResponsePermanentRedirect
HttpResponseNotFound
HttpResponseBadRequest
HttpResponseForbidden
HttpResponseServerError

StreamingHttpResponse позволяет возвращать итерируемые объекты (например, генераторы как в "голом" WSGI-приложении).

Совершенно очевидно, что формировать страницу в **HttpResponse** неудобно (и как минимум некрасиво смешивать логику и интерфейсы).

Для описания интерфейсной части и html-страниц в Django имеется понятие шаблонов - отдельных файлов, описывающих статическую структуру страниц в HTML и описанные специальным синтаксисом динамические данные.

При этом модуль **django.shortcuts** содержит вспомогательную функцию **render**, выполняющую загрузку шаблона и формирование ответа **HttpResponse** с переданным словарем контекста.

Функция **render** также принимает именованные параметры для формирования ответа - **content_type** и **status**.

```
1. from django.shortcuts import render

2. def example_view(request):
3.     return render(
4.         request, 'example.html',
5.         {'message': 'Hello, World!'}
6.     )
```

Загрузка шаблона будет произведена из директории шаблонов, согласно конфигурациям **TEMPLATES** из **settings.py** проекта (по умолчанию, это директории templates в корне проекта и его приложениях).

```
1. TEMPLATES = [
2.     {
3.         'BACKEND': '...', 'DIRS': [], 'APP_DIRS': True,
4.         'OPTIONS': {...},
5.     },
6. ]
```

Если указанный шаблон не найден, функция **render** сгенерирует исключительную ситуацию **TemplateDoesNotExist** из модуля **django.template**.

По умолчанию, Django использует собственный язык шаблонов - Django template language, но возможно использование и других шаблонизаторов.

Если указанный шаблон содержит синтаксические ошибки, функция **render** сгенерирует исключительную ситуацию **TemplateSyntaxError** из модуля **django.template**.

При включенной настройке **DEBUG = True** в **settings.py** проекта для исключительных ситуаций в представлениях Django сформирует человеко-читабельную страницу с полным описанием проблемы.

ТЕХНОАТОМ

Язык описания шаблонов Django

Переменные
Тэги
Фильтры

Переменные шаблонов Django выводят значения из контекста.

```
1. {{ simple }}
2. {{ object.attribute }}
3. {{ dict.key }}
4. {{ list.0 }}
```

Важно заметить, что при выводе в шаблон всегда будет вызван метод строкового представления объекта.

Если поведение процессора контекста не переопределено, то в каждом шаблоне доступна переменная **request**, содержащая экземпляр **HttpRequest** запроса пользователя.

Тэги шаблонов Django представляют логику формирования ответа.

```
1. {% if condition %}
2.     <b>It's alive!</b>
3. {% endif %}
```

Конструкции условий

```
1. {% if <condition> %}
2.     ...
3. {% elif <condition> %}
4.     ...
5. {% else %}
6.     ...
7. {% endif %}
```

В условиях возможно применение:

- условных операторов **or**, **and**, **not**
- операторов сравнения **==**, **!=**, **<**, **>**, **<=**, **=>**
- оператора **in**
- оператора **is**

Конструкции циклов

```
{% for <item> in <iterable> %}
    ...
{% empty %}
    ...
{% endfor %}
```

Внутри тела цикла доступны дополнительная переменная forloop, содержащая атрибуты (внутренние переменные контекста цикла):

- счетчик итераций (от единицы) **counter**
- счетчик итераций (от нуля) **counter0**
- флаг первого элемента **first**
- флаг последнего элемента **last**
- контекст родительского цикла (для вложенных) **parentloop**

Конструкции блоков

```
1. {% with <variable>=<value> %}
2.     ...
3. {% endwith %}
```

Конструкции наследования шаблонов

```
1. {% extends "<template>" %}
2. {% block content %}
3.     ...
4. {% endblock %}
```

Родительский шаблон определяет блоки, которые должен определять его потомок.

Язык шаблонов

ТЕХНОАТОМ

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4.     <title>
5.         {% block title %}{% endblock %}
6.     </title>
7. </head>
8. <body>
9.     ...
10.    {% block content %}{% endblock %}
11.    ...
12.</body>
13.</html>
```

Язык шаблонов

ТЕХНОАТОМ

```
1. {% extends "base.html" %}
2. {% block title %}
3.     My Page
4. {% endblock %}
5. {% block content %}
6.     Hello, World!
7. {% endblock %}
```

Конструкции включения шаблонов

```
1. {% include "<template>" with <variable>=<value> only %}
```

Конструкции подключения дополнительных модулей

```
1. {% load <library> %}
```

Функцию reverse

```
1. {% url "<name>" <parameter>=<value> %}
```

ТЕХНОАТОМ

Управляющие автоматическим экранированием блоки

```
1. {% autoescape <on|off> %}
2.     ...
3. {% endautoescape %}
```

И другие тэги шаблонов (в том числе и кастомные).

Фильтры шаблонов Django позволяют преобразование значений переменных.

```
1. {{ <variable>|<filter>:<arguments> }}
```

Встроенных фильтров довольно много, кроме того они также подключаются из сторонних модулей конструкцией **load**).

```
1. {{ charge.is_outcome|yesno:"Outcome,Income,Unknown" }}
```

Самой простым элементом синтаксиса языка описания шаблонов Django являются комментарии.

```
1. {# Comment #}

2. {% comment %}
3.     Multiline comments
4.     released by special tag
5. {% endcomment %}
```

Теперь мы умеем показывать пользователю информацию, но как получать данные от него?

HTML позволяет описывать формы.

```
1. <form action="{% url 'save_charge' %}" method="post">
2.     <input name="value" type="number" required>
3.     <button type="submit">Save</button>
4. </form>
```

Можно вручную обрабатывать параметры запроса:

```
1. @require_POST
2. def create_charge(request):
3.     value = request.POST.get('value')
4.     try:
5.         value = decimal.Decimal(value)
6.     except decimal.InvalidOperation:
7.         # Validation error
8.     ...
```

Модуль **django.forms** предлагает готовые инструменты для работы с формами.

```
1. class ChargeForm(forms.Form):
2.     value = forms.DecimalField(label= ‘Value', required=True)

3. @require_POST
4. def create_charge(request):
5.     form = ChargeForm(request.POST)
6.     if form.is_valid():
7.         ...
8.     return render(
9.         request, 'charge-form.html',
10.        {'form': form}
11.    )
```

Формы можно вставлять в шаблоны:

```
1. <form action="{% url 'save_charge' %}" method="post">
2.     {{ form }}
3.     <button type="submit">Save</button>
4. </form>
```

При этом базовые элементы форм можно разделить на виджеты (элементы, ответственные за то как именно поле будет представлено в HTML) и сами поля (атрибуты формы, включающие в себя механизмы валидации).

В виджетах нас в большей степени может интересовать переопределение метода **render**, возвращающий непосредстенно html-код. Но зачастую в этом нет необходимости.

```
1. class TextInput(widgets.Widget):

2.     def render(self, name, value, attrs=None):
3.         return '<input name="%(name)s" value="%(value)s">' % {
4.             'name': name, 'value': value if value else ''
5.         }
```

Поля формы представляют больший интерес, так как имеют методы для переопределения механизмов валидации (проверки корректности).

to_python(self, value)

Преобразование значения из строки в целевой тип

validate(self, value)

Проверка значения

clean(self, value)

Преобразование и проверка значения (комплексно)

Может вызывать несколько валидаторов

В случае возникновения ошибки на каком-то из этапов валидации поля Django сгенерирует исключительную ситуацию **ValidationError** из **django.core.exceptions**

```
1. class MoneyField(fields.DecimalField):

2.     accuracy = Decimal('0.01')

3.     def to_python(self, value):
4.         value = super().to_python(value)
5.         return value.quantize(self.accuracy)
```

```
1. class PositiveMoneyField(MoneyField):

2.     def validate(self, value):
3.         if value < 0:
4.             raise ValidationError('Must be positive')
```

Тем не менее, поля формы предполагают только проверку соотвествия базовому типу, и не следует определять класс каждого нового поля формы.

Для валидации поля формы по бизнес-логике можно использовать определение методов **clean_<имя поля>** в классе самой формы.

```
1. class ChargeForm(forms.Form):

2.     value = forms.DecimalField(label= ‘Value', required=True)

3.     def clean_value(self):
4.         value = self.cleaned_data.get('value')
5.         if value < 0:
6.             raise ValidationError('Must be positive')
7.         return value
```

При этом проверить можно и все поля сразу (это полезно, когда они зависят друг от друга) переопределив метод **clean** класса формы.

```
1. class ChargeForm(forms.Form):

2.     value = forms.DecimalField(label= ‘Value', required=True)

3.     def clean(self):
4.         cleaned_data = super().clean()
5.         if cleaned_data.get('value') < 0:
6.             self.add_error('value', 'Must be positive')
7.         return cleaned_data
```

Используя функцию **date.today** из модуля **datetime** и поле **DateField** из **django.forms.fields** давайте добавим в форму **ChargeForm** поле даты. При этом, заведение платежа "задним числом" невозможно (дата платежа не должна быть больше сегодняшней даты).

Спасибо за внимание!

Михалев Олег
mailto:mhalairt@gmail.com

Домашнее задание

ТЕХНОАТОМ

Задана сущность **Charge** (денежная транзакция) содержащая поля:

- value - вещественное число (**decimal.Decimal**), сумма транзакции (может быть отрицательным или положительным);
- date - дата (**datetime.date**), день проведения транзакции.

Домашнее задание

ТЕХНОАТОМ

Необходимо разработать представление и форму для сущности **Charge** и реализовать необходимые проверки, при этом транзакции-списания (с отрицательным значением) не могут быть заведены на будущее (дата **date** в таких транзакциях должна быть меньше или равна сегодняшнему дню из **date.today**). В случае успеха или неудачи (определяемым по корректности заполнения формы) нужно выводить соответствующие сообщения.

Задан генератор (источник данных), возвращающий случайное количество пар (**date**, **value**) для транзакций:

```
1.  from datetime import date
2.  from decimal import Decimal

3.  from random import randint

4.  def random_transactions( ):
5.      today = date.today( )
6.      start_date = today.replace(month=1, day=1).toordinal()
7.      end_date = today.toordinal()
8.      while True:
9.          start_date = randint(start_date, end_date)
10.         random_date = date.fromordinal(start_date)
11.         if random_date >= today:
12.             break
13.         random_value = randint(-10000, 10000), randint(0, 99)
14.         random_value = Decimal('%d.%d' % random_value)
15.         yield random_date, random_value
```

Домашнее задание

ТЕХНОАТОМ

Необходимо разработать представление, которое будет получать от генератора список транзакций и выводить их на страницу (для этого следует использовать шаблоны). При этом транзакции-списания (с отрицательным занчением **value**) и транзакции-зачисления (с положительным значением **value**) должны выводиться в разные списки на странице (можно использовать для верстки html-списки или таблицы).

Полезные ссылки

ТЕХНОАТОМ

Обобщенный пример из лекции доступен по ссылке
https://drive.google.com/open?id=0B3RxNTHeHOTKaGRSMENjY3FRMzg

Тэги и фильтры шаблонов Django
https://docs.djangoproject.com/es/1.10/ref/templates/builtins/
Формы Django
https://docs.djangoproject.com/en/1.10/ref/forms/api/
Виджеты Django
https://docs.djangoproject.com/en/1.10/ref/forms/widgets/
Поля формы Django
https://docs.djangoproject.com/en/1.10/ref/forms/fields/