Django: шаблонизатор и HTML-шаблоны | Я.Шпора

Размещение шаблонов в структуре Django-проекта

Хранить шаблоны в проекте Django можно

- на уровне приложения: в директориях приложений создают папки `templates/имя_приложения/` и там хранят шаблоны, используемые в приложении;
- на уровне проекта: в корневой директории проекта создаётся папка `templates/`, в ней — директории, названные по именам приложений, и в них хранятся шаблоны, относящиеся к приложениям проекта.

Эти способы хранения шаблонов можно комбинировать.

Теги шаблонизатора

Шаблонизатор поддерживает все стандартные HTML-теги; помимо них в шаблонизаторе есть и несколько десятков собственных тегов.

Теги шаблонизатора обозначаются так: `{% tag %}`. Принято ставить пробел между именем тега и символами `%`.

Переменные шаблонизатора обозначаются так: `{{ var_name }}`.

Комментарии обозначаются так: `{% comment %}` и `{% endcomment %}`.

Теги шаблонизатора бывают одиночные и парные.
У парных есть начальные и конечные теги:
`{% tag_name %}` ... `{% endtag_name %}`.

Тег `include`

Вставляет содержимое другого шаблона в текущий шаблон.

```
<!-- base.html -->
<html>
<head><title>Верстаем лучше всех</title></head>
<body>
  <header>{% include "header.html" %}</header>
  ...
```

```
</body>
</html>
```

```
<!-- header.html -->
<h1>Include экономит время!</h1>
```

После рендера страницы получится такой код:

```
<html>
<head><title>Верстаем лучше всех</title></head>
<body>
  <header><h1>Include экономит время!</h1></header>
  ...
</body>
</html>
```

Теги `extends` и `block`

Разработчик создаёт **базовый шаблон** (обычно его называют *base.html*) с HTML-кодом, который повторяется на всех страницах сайта. В базовом шаблоне размечаются области, куда можно вставить уникальное для каждой страницы содержимое. Для разметки таких «посадочных мест» применяют парный тег `{% block имя_блока %}` — `{% endblock %}`.

```
<!-- base.html -->
<html>
<head><title>{% block title %}Заголовок страницы{% endblock %}</title></h
<body>
  ...
  <main>{% block content %}Контент страницы{% endblock %}</main>
  <footer>Общий подвал страницы</footer>
</body>
</html>
```

Разработчик создаёт **дочерние шаблоны**, в которых хранится лишь фрагмент HTML-кода — уникальное содержимое страницы. Это содержимое тоже

обрамлено тегами `{% block имя_блока %}` — `{% endblock %}`. Имена блоков должны совпадать с именами в *base.html*.

```
<!-- child.html -->
{% extends "base.html" %}
{% block title %}Заголовок страницы дочернего шаблона{% endblock %}
{% block content %}
  <p>Контент дочернего шаблона</p>
  <p>И ещё контент шаблона</p>
{% endblock %}
```

`{% extends %}` помещается всегда **в начало** дочернего шаблона.

Словарь контекста: отправка данных из view-функции в шаблон

Данные, которые необходимо вывести на веб-страницу, сохраняются в словарь, и этот словарь передаётся в функцию `render`. Словарь обычно называют `context`.

```
# homepage/views.py

def index(request):
    # Адрес шаблона сохранён в переменную, это не обязательно, но удобно.
    template_name = 'homepage/index.html'
    # Строки, которые надо вывести на страницу, тоже сохраненяются в пере
    title = 'Главная страница ACME'
    promo_product = 'Iron carrot'
    context = {
        # Ключ словаря и имя переменной не обязательно называть одинаково
        # но это удобнее, чем использовать два разных имени.
        'title': title,
        'promo_product': promo_product,
    }
    # Третьим аргументом в render() передаём словарь context:
    return render(request, template_name, context)
```

Ключи словаря указывают в двойных фигурных скобках в нужных местах шаблона.

```
<!-- templates/homepage/index.html -->
...
<h1>{{ title }}</h1>
...
<p>Товар дня: {{ promo_product }}</p>
...
```

После рендеринга страницы получится такой HTML-код:

```
...
<h1>Главная страница ACME</h1>
...
<p>Товар дня: Iron carrot</p>
...
```

Ветвления в Django-шаблонах

Тег ветвления `{% if %}` похож на инструкцию `if...elif...else` в Python.

У тега `{% if %}` обязательно должен быть закрывающий тег `{% endif %}`:

```
{% if promo_product == "Iron carrot" %}
  Железная морковка — хорошо для стальных зубов.
{% elif promo_product == "Giant mousetrap" %}
  Ребята, давайте жить дружно!
{% else %}
  Вы все меня ещё не знаете, но вы меня узнаете!
{% endif %}
```

Циклы в Django-шаблонах

Прямо в шаблоне можно выполнить цикл — например, перебрать в цикле список.

Список `products` передаётся в шаблон `catalog/product_list.html`:

```
# catalog/views.py
...
def product_list(request):
    template_name = 'catalog/product_list.html'
```

```
    products = [
      'Iron carrot',
      'Giant mousetrap',
      'Dehydrated boulders',
      'Invisible paint',
    ]
    context = {
        'products': products,
    }
    return render(request, template_name, context
```

В шаблоне итерируемся по списку `products`: каждый элемент помещаем в переменную `product`, выводим значение переменной на страницу, обрамляя его в тег `p`:

```
<!-- templates/catalog/product_list.html -->
...
{% for product in products %}
  <p>{{ product }}</p>
{% endfor %}
```

В итоге на страницу будет выведено четыре тега `<p>`, в каждый из которых будет заключено название продукта из списка `products`:

```
...
<p>Iron carrot</p>
<p>Giant mousetrap</p>
<p>Dehydrated boulders</p>
<p>Invisible paint</p>
```

Дополнительные атрибуты в циклах шаблонов

В цикле шаблонизатора доступен объект `forloop`, через его атрибуты можно получить доступ к определённым значениям:

`forloop.counter` — номер текущей итерации цикла (начинает отсчёт с 1).

`forloop.counter0` — номер текущей итерации цикла «по-программистски» (начинает отсчёт с 0).

`forloop.revcounter` — показывает количество итераций, прошедших с начала цикла (на последнем проходе цикла этот атрибут вернёт значение `1`).

`forloop.revcounter0` — показывает количество итераций, прошедших с начала цикла (на последнем проходе цикла этот атрибут вернёт значение `0`).

`forloop.first` — `True`, если это первая итерация.

`forloop.last` — `True`, если это последняя итерация.

Ссылки в Django-шаблонах

В Django-шаблонах вместо конкретных адресов в ссылках можно применять `name`, указанные для `path()` в *urls.py*:

```
# catalog/urls.py

urlpatterns = [
    # Для path() указаны name:
    path('catalog/', views.product_list, name='product_list'),
    path('catalog/<int:pk>/', views.product_detail, name='product_detail'
]
```

В Django-шаблоне в атрибуте `href` указываем специальный тег `{% url имя_ссылки %}` . , а в нём вместо адреса указываем `name`:

```
<!-- Вместо <a href="catalog/"> -->
<a href="{% url 'product_list' %}">
  Страница со списком товаров
</a>
<!-- Если в пути применяются аргументы, их тоже можно указать.
Вместо адреса <a href="catalog/1/"> указываем name и аргумент 1: -->
<a href="{% url 'product_detail' 1 %}">
  Страница с товаром номер 1
</a>
```

name и namespace в ссылках

У разных приложений в файлах `urls.py` могут быть одинаковые имена для `path()` — и это приведёт к конфликту адресов. Избежать такой ситуации поможет параметр **namespace** — пространство имён приложения.

`namespace` для всех адресов приложения указывается в *urls.py* **подключаемого приложения** в зарезервированной переменной `app_name`:

```
# catalog/urls.py

# Указываем namespace для ссылок приложения:
app_name = 'catalog'

urlpatterns = [
    path('', views.product_list, name='product_list'),
]
```

Устанавливайте `app_name` таким же, как имя приложения — тогда `namespace` этого приложения будет уникален в пределах проекта.

Имя ссылки в шаблоне следует указывать так: `namespace:name`.

```
<a href="{% url 'catalog:product_list' %}">
  Страница со списком товаров
</a>
```

Фильтры в Django-шаблонах

В коде шаблона название фильтра пишется после имени переменной, через символ `|`: `{{ variable|filter:параметры }}`.

Фильтр length: возвращает длину строки или другой последовательности: `{{ long_string|length }}`.

Фильтр safe: позволяет шаблонизатору Django «отрисовать» HTML-теги в строке, переданной через словарь контекста.

```
<!-- catalog/product.html -->
{{ product_name|safe }}
{{ product_description|safe }}
```

> **Фильтр safe потенциально опасен**. Например, если применить этот фильтр для вывода сообщений пользователей — то злоумышленники смогут опубликовать свой HTML-код для сбора данных посетителей или для каких-то иных пакостей.

Фильтр linebreaksbr: преобразует символы переноса строк `\n` в HTML-тег `<br>`.

```
{{ text|linebreaksbr }}
```

Фильтр date: позволяет изменять форматирование объектов `datetime` непосредственно в шаблоне.

```
{{ gogol_day|date:"d.m.y" }} {# Выведет 01.04.09 #}
{{ gogol_day|date:"j.n.Y" }} {# Выведет 1.4.1809 #}
{{ gogol_day|date:"d M Y" }} {# Выведет 01 Apr 1809 #}
{{ gogol_day|date:"d E Y" }} {# Выведет 01 April 1809 #}
```

В Django доступно множество других фильтров.

Фильтры можно объединять в цепочку:

```
{{ armstrong_said|title|truncatewords:4 }}
```

Подключение стилей, картинок и скриптов к Django-шаблонам

В начале того шаблона, в котором будет использоваться статика, нужно установить тег `{% load static %}`. Это следует делать даже в тех случаях, когда шаблоны наследуются через `extends`.

После этого в HTML-тегах с помощью тега `{% static 'адрес_файла' %}` нужно указать адреса файлов:

```
{% load static %}

<!-- Подключили CSS -->
<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">
```

```
<!-- Показали картинку -->
<img src="{% static 'img/logo.png' %}">
```

Имя папки со статикой нужно указать константе `STATICFILES_DIRS` в файле *settings.py*. Для папки с именем `static_dev`, лежащей в корне проекта, константа будет выглядеть так:

```
...
# Директории, где собраны статические файлы проекта (список):
STATICFILES_DIRS = [
    BASE_DIR / 'static_dev',
]
```

Практикум