

Пользовательские теги шаблонов



Архив проекта: [lesson-11-coolsite.zip](#)

Давайте вернемся к проекту нашего сайта, откроем файл `women/views.py` и вспомним, что функции представления `index` и `show_category` имеют дублирование кода, в частности, они обе считывают список категорий и передают его шаблону. Пришло время это поправить. Существует несколько способов это сделать. Можно воспользоваться классами представлений и общий код вынести в отдельный базовый класс. Но пока у нас функции мы воспользуемся механизмом создания собственных шаблонных тегов. И с их помощью решим данную задачу.

Django позволяет использовать два вида пользовательских тегов:

- `simple tags` – простые теги;
- `inclusion tags` – включающие теги.

Подробную русскоязычную документацию по ним можно посмотреть по ссылке:

<https://djbbook.ru/re13.0/howto/custom-template-tags.html>

Simple Tags

Вначале мы создадим простой тег, который будет загружать категории из БД и использоваться непосредственно в шаблоне. Согласно документации теги должны располагаться в подкаталоге `templatetags` каталога приложения `women` и являться пакетом, то есть, содержать файл `__init__.py`. Сделаем это. Далее, нам нужно добавить в эту папку еще один `python`-файл, в котором и будем прописывать логику нового тега. Файл назовем каким-нибудь понятным именем, например, `women_tags`. Импортируем сюда модуль `template` для работы с шаблонами и наши модели:

```
from django import template
from women.models import *
```

Следующим шагом нам нужно создать экземпляр класса `Library`, через который происходит регистрация собственных шаблонных тегов:

```
register = template.Library()
```

И, далее, определим функцию, которая будет выполняться при вызове нашего тега из шаблона. Так как нам нужны будут списки категорий, то функция будет достаточно простой:

```
def get_categories():
    return Category.objects.all()
```

Название функции `get_categories` мы придумываем сами. Теперь, чтобы связать эту функцию с тегом, или, превратить эту функцию в тег, используется специальный декоратор, доступный через переменную `register`:

```
@register.simple_tag()
def get_categories():
    return Category.objects.all()
```

Все, мы только что создали свой простой пользовательский тег для использования его в шаблонах.

Давайте им воспользуемся. Перейдем в базовый шаблон `base.html` и вначале выполним загрузку тегов, определенных в файле `women_tags`:

```
{% load women_tags %}
```

После этого в шаблоне (и во всех его дочерних шаблонах) доступен тег по имени `get_categories`. Однако, если сейчас обновить главную страницу, то получим ошибку, что `women_tags` не зарегистрирован. Это связано с тем, что после добавления нового пакета `templatetags` и файла `women_tags.py` необходимо перезапустить тестовый веб-сервер, чтобы он подхватил изменения в проекте сайта. После перезагрузки никаких ошибок не появляется.

Далее, в месте вывода рубрик давайте просто запишем наш новый сформированный тег:

```
{% get_categories %}
```

и при обновлении главной страницы увидим отображение объекта `QuerySet`. Это говорит о том, что все работает, так и должно быть, т.к. именно этот объект и возвращает данный тег. Но как нам теперь перебрать элементы этого объекта? Подставить в цикл `get_categories` мы не можем, т.к. это не переменная, а тег шаблона. Для этого в Django в тегах шаблонов можно использовать специальное ключевое слово `as`, которое сформирует ссылку на данные тега. В нашем случае это можно записать так:

```
{% get_categories as categories %}
```

Сформируется переменная `categories`, которую уже можно использовать в теге цикла `for`. Кстати, при обновлении страницы, `get_categories` уже не будет отображаться на странице, т.к. изменилось его поведение – данные передаются в переменную (точнее ссылку). Подставим переменную `categories` в цикл и обновим страницу. У нас визуально ничего не изменилось, но прежний параметр `cats` теперь стал не нужен и его можно не передавать в шаблоны. Значит, в функциях представления `index` и `show_category` его можно убрать. Таким образом, мы с помощью простого пользовательского тега смогли исключить дублирующий параметр и строчки кода.

Всегда ли тег будет называться по имени функции? В действительности, мы можем указать любое другое. Для этого в декораторе `register.simple_tag` достаточно указать параметр `name` и через него определить желаемое имя тега, например, так:

```
@register.simple_tag(name='getcats')
def get_categories():
    return Category.objects.all()
```

И, далее, в шаблоне `base.html` следует использовать имя `'getcats'`:

```
{% getcats as categories %}
```

Как видите, все достаточно просто.

Inclusion Tags

Второй тип пользовательских тегов – включающий тег, позволяет дополнительно формировать свой собственный шаблон на основе некоторых данных и возвращать фрагмент HTML-страницы. Давайте посмотрим как он формируется и используется в шаблонах.

Сначала в файле `women_tags.py` добавим функцию для реализации этого второго тега и, так как она будет возвращать полноценный шаблон, то назовем ее `show_categories`:

```
@register.inclusion_tag('women/list_categories.html')
def show_categories():
    cats = Category.objects.all()
    return {"cats": cats}
```

Здесь у нас в функции происходит чтение всех рубрик из БД, а затем, возвращается словарь с этими данными. Словарь используется как набор параметров для шаблона, указанный в параметре декоратора `inclusion_tag`. То есть, в шаблоне `list_categories.html` будет доступна директива `cats` со списком всех рубрик. Именно этот сформированный шаблон и будет возвращаться данным тегом.

Осталось прописать сам шаблон. Разместим его среди всех остальных шаблонов (хотя, при необходимости, можно создать отдельный подкаталог и размещать шаблоны тегов в нем). И скопируем в него следующий фрагмент шаблона из `base.html`:

```
{% for c in categories %}
    {% if c.pk == cat_selected %}
        <li class="selected">{{c.name}}</li>
    {% else %}
        <li><a href="{ c.get_absolute_url }">{{c.name}}</a></li>
    {% endif %}
{% endfor %}
```

Здесь переменную `categories` заменим на `cats`, так как именно ее мы передаем как параметр этому шаблону, и наш тег готов.

Осталось вызвать его в базовом шаблоне `base.html` и вместо вывода рубрик, записать тег:

```
{% show_categories %}
```

Обновляем главную страницу сайта и видим, что все работает – рубрики выводятся с помощью нашего включающего тега. (При необходимости перезапустите тестовый веб-сервер). Правда, в шаблоне `list_categories.html` отсутствует директива `cat_selected`, но мы это сейчас поправим.

Передача параметров пользовательским тегам

Все наши теги могут принимать некоторые параметры, которые расширяют их функциональность. О чем здесь речь? Смотрите, для начала обратимся к простому тегу и изменим его функцию, следующим образом:

```
@register.simple_tag(name='getcats')
def get_categories(filter=None):
    if not filter:
        return Category.objects.all()
    else:
        return Category.objects.filter(pk=filter)
```

То есть, мы добавили один именованный параметр `filter`, который по умолчанию равен `None` и функция возвращает все категории. Если же указать какое-либо числовое значение, то будем получать категорию с указанным идентификатором. Как теперь все это можно использовать в шаблонах? Давайте перейдем в `base.html` и для начала просто вызовем этот тег:

```
{% getcats %}
```

При обновлении главной страницы, увидим список всех категорий. Это пример вызова тега без параметров. А теперь укажем параметр, например, так:

```
{% getcats filter=1 %}
```

Увидим первую рубрику. Или, можно указать его так:

```
{% getcats 2 %}
```

Увидим только вторую рубрику. Вот так определяются и используются параметры у пользовательских тегов.

Давайте проделаем похожую операцию, но с включающим тегом. Определим у него два именованных параметра:

```
@register.inclusion_tag('women/list_categories.html')
def show_categories(sort=None, cat_selected=0):
    if not sort:
        cats = Category.objects.all()
    else:
        cats = Category.objects.order_by(sort)

    return {"cats": cats, "cat_selected": cat_selected}
```

Первый будет выполнять сортировку по указанному полю, а второй – недостающий параметр `cat_selected`. Далее, в функции делаем выборку в соответствии с параметром `sort`, а затем, передаем шаблону дополнительный параметр `cat_selected`.

Все, теперь наш включающий тег может принимать два параметра. В шаблоне `base.html` пропишем его вызов, например, так:

```
{% show_categories '-name' cat_selected %}
```

Здесь первый параметр `-name` будет определять сортировку по названию рубрик, а второй – просто передает нужный для обработки данных параметр `cat_selected`. Также можно указать только какой-либо один параметр, например, так:

```
{% show_categories cat_selected=cat_selected %}
```

или так:

```
{% show_categories sort='name' %}
```

Но вернем вариант:

```
{% show_categories cat_selected=cat_selected %}
```

чтобы тег работал как положено. Как видите, в Django достаточно просто создавать и использовать пользовательские теги. Попробуйте самостоятельно сделать отображение главного меню через пользовательские теги, чтобы исключить и это дублирование в функциях.

Видео по теме



ие связей между и через класс `ForeignKey`



#10. Начинаем работу с админ-панелью



#11. Пользовательские теги шаблонов



#12. Добавляем слаг (slug) к URL-адресам



#13. Использование форм, не связанных с моделями



#14. Формы, связанные с моделями. Пользовательские



#15. Классы представлений `ListView`, `DetailView`, `CreateView`



















←

→

← Предыдущая

Следующая →

© 2021 Частичное или полное копирование информации с данного сайта для распространения на других ресурсах, в том числе и бумажных, строго запрещено. Все тексты и изображения являются собственностью сайта

Политика конфиденциальности | Пользовательское соглашение