django-navigation Documentation

Release 0.8.1

Andrey Mikhaylenko

January 02, 2013

CONTENTS

Extensible breadcrumbs navigation for Django.

Installation is simple:

```
$ pip install django-navigation
```

The only requirement is a more or less recent version of Django.

Contents:

CHAPTER

ONE

OVERVIEW

Let's assume we have this URL path:

```
/news/2010/oct/hello-world/
```

We need to convert it to *breadcrumbs* and display along the heading this way:

```
News → 2010 news → October 2010
```

So we just type this in our template:

```
{% get_breadcrumb_trail as trail %}
<ul>
{% for crumb in trail %}
    <li><a href="{{ crumb.url }}">{{ crumb }}</a></li>
{% endfor %}
</ul>
<h1>{% breadcrumb %}</h1>
```

...and this is the result:

```
<ul>
    <li><a href="/news/>News</li>
    <li><a href="/news/2010/">2010 news</li>
    <li><a href="/news/2010/oct/">October 2010</a></li>
</ul>
<h1>Hello world</h1>
```

1.1 How does this work?

Current URL path is split into hierarchical parts:

```
* /news/
* /news/2010/
* /news/2010/oct/
* /news/2010/oct/hello-world/
```

For each part a `navigation.helpers.Crumb` instance is created. It stores the URL and the corresponding title. But how do we know the title?

The URL title is resolved by a *crumb resolver*. By default two resolvers are available: *_resolve_flatpage* and *_resolve_by_callback*.

The first one looks for a *FlatPage* object with given URL path in the database (if *django.contrib.flatpages* is activated in settings). If this resolver failed (i.e. flatpages are not available or there's no *FlatPage* with such URL path), then next crumb resolver is called.

The crumb resolver *_resolve_by_callback* peeks into the URL maps and attempts to resolve the URL into a view function. If such function is found, the resolver looks whether the function has the "breadcrumb" attribute. This attribute can be set by wrapping the view in decorator `navigation.decorators.breadcrumb()`:

```
from navigation.decorators import breadcrumb

@breadcrumb('Hello')
def say_hello(request):
    ...

@breadcrumb(lambda request: u'%s settings' % request.user)
def user_settings(request):
    ...
```

If the attribute is not found, we can't guess the name and give up. A dummy breadcrumb is add to the trail.

However, we could also try "humanizing" the function's `__name__` attribute or use a custom path-to-name mapping. You can do that easily by creating your own crumb resolvers and registering them this way:

```
from navigation.resolvers import crumb_resolver

@crumb_resolver
def my_custom_resolver_function(request, url):
    return Crumb(url, 'Hello!')
```

TODO

I'll probably make this more explicit, e.g. add a settings variable like this:

```
NAVIGATION_RESOLVERS = [
    'navigation.resolvers.resolve_flatpage',
    'navigation.resolvers.resolve_by_callback',
    'utils.navigation.my_custom_resolver_function',
]
```

CHAPTER

TWO

API REFERENCE

2.1 Decorators

navigation.decorators.**breadcrumb**(*crumb*, *coerce_to=None*)

Usage:

```
from navigation.decorators import breadcrumb

@breadcrumb('greeting')
def some_view(request):
    return 'Hello world!'

@breadcrumb(lambda request: u'greeting for %s' % request.user.username)
def some_view(request):
    return 'Hello %s!' % request.user.username
```

Note: By default the value returned by a callable is required to be a `unicode` object. If the function returns a model instance, its `__unicode__` method is *not* called. Use `coerce_to=unicode`.

Parameters

- **crumb** – A `unicode` string or a callable that returns it.
- **coerce_to** – Coerces *crumb* to given type. The value can be `unicode` or any callable that returns a `unicode` object.

2.2 Template tags and filters

Loading:

```
{% load navigation_tags %}
```

navigation.templatetags.navigation_tags.**named_crumb**(*parser*, *tokens*)

Resolves given named URL and returns the relevant breadcrumb label (if available). Usage:

```
<a href="{% url project-detail project.slug %}">
    {% named_crumb project-detail project.slug %}
</a>
```

navigation.templatetags.navigation_tags.**crumb_link**(*parser*, *tokens*)
Acts like named_crumb() but also wraps the result into a link tag. Usage:

```
<ul>
    <li>{% crumb_link 'auth_login' %}</li>
    <li>{% crumb_link 'project-index' %}</li>
</ul>
```

The result:

```
<ul>
    <li><a href="/accounts/login/">Log in</a></li>
    <li><a href="/projects/">Projects</a></li>
</ul>
```

Please note that you have to use quotes, otherwise the arguments are considered variable names.

navigation.templatetags.navigation_tags.**get_breadcrumb_sections**(*parser*, *tokens*)
Returns a list of *sections*. Usage:

```
{% get_breadcrumb_sections as sections %}
{% for section in sections %}
    ...
{% endfor %}
```

navigation.templatetags.navigation_tags.**get_breadcrumb_trail**(*parser*, *tokens*)
Returns the trail of *breadcrumbs*. Each breadcrumb is represented by a navigation.helpers.Crumb instance.

navigation.templatetags.navigation_tags.**get_navigation**(*request*)
Returns the rendered navigation block. Requires that the *navigation.html* template exists. Two context variables are passed to it:

- sections (see get_breadcrumb_sections())
- trail (see get_breadcrumb_trail())

2.3 Helpers

class navigation.helpers.**Crumb**(*url*, *title*, *is_current=False*, *is_active=False*, *is_dummy=False*)
A navigation node.

url
this breadcrumb's URL.

title
this breadcrumb's title, as determined by the first successful *crumb resolver*.

is_current
True if this breadcrumb's URL corresponds to the current request path.

is_active
True if current request path begins with this breadcrumb's URL.

is_dummy
True if this breadcrumb is a stub, i.e. its URL could not be resolved by a *crumb resolver*.

CHAPTER

THREE

TESTING

Django-navigation is covered by tests itself and provides a specialized *TestCase* class that can be reused to test other applications. For instance:

```
from navigation.tests import NavigationTest


class GameTest(NavigationTest):
    fixtures = ['test_data.yaml']
    urls = 'games.urls'

    def test_breadcrumbs(self):
        self.assertTitle('/', 'Games')
        self.assertTitle('/pc-linux/', 'PC / Linux')
        self.assertTitle('/pc-linux/wesnoth/', 'Battle for Wesnoth')
```

This example makes sure that certain titles correspond to given URLs, whatever breadcrumb resolver(s) are involved.

CHAPTER

FOUR

HISTORY

This breadcrumbs application was originally inspired by two different snippets:

1. FlatPages trail by **jca**
2. Idea and implementation of universal breadcrumbs for custom views by **Thomas Guettler**

Then Andrey Mikhaylenko (that's me) wrote a unified extensible templated breadcrumbs application for both FlatPages and custom views and here it is.

CHAPTER

FIVE

GLOSSARY

breadcrumbs a trail of links to higher-level documents. Represents levels of given URL path. For example, URL `/news/2010/oct/` corresponds to a document with heading "October 2010" and this path of breadcrumbs:

- `/news/` "News"
- `/news/2010/` "2010 news"

crumb resolver a function that takes arguments *request* and *url* and returns either a `navigation.helpers.Crumb` instance or *None*. Can be registered by using the decorator `navigation.resolvers.crumb_resolver()`:

```
@crumb_resolver
def custom_resolver(request, url):
    if url == '/secret/url/':
        return Crumb(url, 'Hello')
    else:
        return None  # pass to another resolver, if any
```

If current URL is `/secret/url/`, then the resolver will be called for both `/secret/` and `/secret/url`. The resolver may not be called if another resolver did not return *None* for given URL (i.e. first resolver wins).

If all resolvers returned *None* for a URL, then a dummy crumb is created. It can be told from a regular crumb in templates this way:

```
{% if crumb.is_dummy %}
    <a href="{{ crumb.url }}">(???)</a>
{% else %}
    <a href="{{ crumb.url }}">{{ crumb }}</a>
{% endif %}
```

This will produce the path of breadcrumbs like "(???) -> Hello" if `/secret/url/` could be resolved but `/secret/` couldn't.

sections First-level URLs explicitly listed as *NAVIGATION_SECTIONS* setting (optional; only required by `get_sections()`).

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

n