django-meio-easytags Documentation

Release 0.7

Vinicius Mendes

May 10, 2013

CONTENTS

An easy way to create custom template tags for Django's templating system.

EasyNode will take care of the template tag's parsing, resolving variable and inspecting if the parameters are ok with the `render_context` signature.

It's almost like calling methods in Python.

CHAPTER

ONE

BASIC USAGE

Just subclass the EasyNode and create the method `render_context` and register the template tag:

```
from django.template import Library
from easytags.node import EasyNode

register = Library()

class BasicNode(EasyNode):

    def render_context(self, context):
        return 'basic template tag'

register.tag('basic', BasicNode.parse)
```

Then load your template tags in your template and use it:

```
{% load my_template_tags %}

{% basic %}
```

Ok. It's easy to do with built-in django's `Node`. In fact this case uses almost nothing of the django-meio-easytags power. Let's take a look at a more complex example.

CHAPTER

TWO

EASYLIBRARY USAGE

The `easytags.EasyLibrary` is a subclass of Django's default `django.template.Library`. It adds a method to create easy tags just defining a function very similar to the `render_context` above. Just do this:

```
from easytags import EasyLibrary

register = EasyLibrary()

def basic(context):
    return u'basic template tag'

register.easytag(basic)
```

That's it. We just created our template tag. Pay atention that our register is an instance of `easytags.EasyLibrary`, not of `django.template.Library`.

2.1 Custom template tag name

You may set a custom name for your template tag just like you do with ordinary `register.tag`:

```
register.easytag('my_fancy_name', basic)
```

2.2 register.easytag as decorator

You can register easy tags with decorators:

```
@register.easytag
def decorated(context):
    return u'decorated'
```

And define the tag name inside the decorator:

```
@register.easytag(name='fancy_decorated')
def decorated(context):
    return u'fancy decorated'
```

CHAPTER

THREE

ACCEPTS PARAMETERS

You can also create a template tag that receives one or more parameters and set default values to it like you do in Python methods:

```
from easytags import EasyLibrary

register = EasyLibrary()

def sum(context, arg1, arg2, arg3=0):
    return int(arg1) + int(arg2) + int(arg3)

register.easytag(sum)
```

In this case, your template tag have two mandatory parameters (`arg1` and `arg2`) and one optional parameter (`arg3`) that defaults to `0`. You can use this template tag in any of these ways:

```
{% sum "1" "2" %}
{% sum "1" arg2="2" %}
{% sum arg1="1" arg2="2" %}
{% sum "1" "2" "3" %}
{% sum arg2="2" arg1="1 %}
```

That is the way you are already used to do in Python. The django-meio-easytags parser will take care of verifying if your template tag accepts the parameters defined and if it doesn't, it will raise a `TemplateSyntaxError`. You don't need to define anywere else which parameters your template tag accepts except for the `render_context` signature definition.

3.1 Resolving context variable

Pay atention that in the example above, all parameters are defined as absolute values, that is, inside double quotes. This is needed because django-meio-easytags allows you to create template tags that accepts context variables as default:

```
{% sum variable1 "2" %}
```

The code above will resolve `variable1` in the context and use as first parameter for the `sum` template tag.

CHAPTER

FOUR

ACCEPTS *ARGS

There's a very nice feature in python that makes it possible to create methods that accepts infinite parameters. In django-meio-easytags it's possible too:

```
from easytags import EasyLibrary

register = EasyLibrary()

def join_lines(context, *args):
    return u'<br />'.join(args)

register.easytag(join_lines)
```

With this tag you can join as many lines as you want:

```
{% join_lines "line1" %}                 # Outputs "line1"
{% join_lines "line1" "line2" %}         # Outputs "line1<br />line2"
{% join_lines "line1" "line2" "lineN" %} # Outputs "line1<br />line2<br />lineN"
```

CHAPTER

FIVE

ACCEPTS **KWARGS

In python you may create methods that receives any named parameter and django-meio-easytags supports it too:

```
from easytags import EasyLibrary

register = EasyLibrary()

def querystring(context, **kwargs):
    return u'&'.join(u'%s=%s' % (k,v) for k, v in kwargs.items())

register.easytag(querystring)
```

With this tag you can build a querystring defining each key and value:

```
{% querystring key1="1" key2="2" %} # Outputs "key1=1&key2=2
```

CHAPTER

SIX

USING *AS* PARAMETER

Thats a common usage in template tags construction to use a parameter to define a variable to receive the content generated by the rendering of the template tag. In django-meio-easytags you can use *EasyAsNode* to achieve such a behavior. Just override the *EasyAsNode* instead of *EasyNode* and use it in your templates like this:

```
from django.template import Library
from easytags.node import EasyAsNode

register = Library()

class BasicAsNode(EasyAsNode):

    def render_context(self, context):
        return 'basic content to variable'

register.tag('basic', BasicAsNode.parse)
```

Then load your template tags in your template and use it:

```
{% load my_template_tags %}

{% basic as varname %}
```

And the Node will store the returning value from *render_context* in a context variable name *varname*. Nice, but what about all the library stuff? I can't use the *as* parameter and register it in EasyLibrary? Yes you can! Just create your method and register it as an *easyastag*:

```
from easytags import EasyLibrary

register = EasyLibrary()

def basic(context):
    return 'basic content to variable'

register.easyastag(basic)
```