Jingo

Release 0.9.0

Sep 06, 2017

Contents

1 NB: Django 1.8 and django-jinja **3**

2 Usage **5**

3 Settings **7**

4 Template Helpers **9**
4.1 Default Helpers 9

5 Template Environment **11**

6 Localization **13**

7 Forms **15**

8 Testing **17**

Python Module Index **19**

Jingo is an adapter for using Jinja2 templates within Django.

CHAPTER 1

NB: Django 1.8 and django-jinja

In version 1.8, Django added support for multiple template engines, and the django-jinja project leverages that to support Jinja2, while Jingo does not.

django-jinja is recommended for new projects. Jingo supports Django 1.8, but it is not clear that its method will continue work beyond that. If you're already using Jingo, and not ready to make the switch, Jingo will continue to work for now, but is undecided about continuing to support new Django versions.

CHAPTER 2

Usage

When configured properly (see *Settings* below) you can render Jinja2 templates in your view the same way you'd render Django templates:

```
from django.shortcuts import render


def my_view(request):
    context = dict(user_ids=(1, 2, 3, 4))
    return render(request, 'users/search.html', context)
```

Note: Not only does `django.shortcuts.render` work, but so does any method that Django provides to render templates from files.

If you're using Django's low-level `Template` class with a literal string, e.g.:

```
from django.templates import Template

t = Template('template string')
```

then you'll need to change that code slightly, to:

```
from jingo import get_env

t = get_env().from_string('template_string')
```

and then the template will be rendered with all the same features that Jingo provides when rendering template files.

CHAPTER 3

Settings

You'll want to use Django to use jingo's template loader. In `settings.py`:

```
TEMPLATE_LOADERS = (
    'jingo.Loader',
    'django.template.loaders.filesystem.Loader',
    'django.template.loaders.app_directories.Loader',
)
```

This will let you use `django.shortcuts.render` or `django.shortcuts.render_to_response`.

You can optionally specify which filename patterns to consider Jinja2 templates:

```
JINGO_INCLUDE_PATTERN = r'\.jinja2'  # use any regular expression here
```

This will consider every template file that contains the substring *.jinja2* to be a Jinja2 file (unless it's in a module explicitly excluded, see below).

And finally you may have apps that do not use Jinja2, these must be excluded from the loader:

```
JINGO_EXCLUDE_APPS = ('debug_toolbar',)
```

If a template path begins with `debug_toolbar`, the Jinja loader will raise a `TemplateDoesNotExist` exception. This causes Django to move onto the next loader in `TEMPLATE_LOADERS` to find a template - in this case, `django.template.loaders.filesystem.Loader`.

Note: Technically, we're looking at the template path, not the app. Often these are the same, but in some cases, like 'registration' in the default setting–which is an admin template–they are not.

The default is in `jingo.EXCLUDE_APPS`:

```
EXCLUDE_APPS = (
    'admin',
    'admindocs',
    'registration',
```

```
    'context_processors',
)
```

Changed in version 0.6.2: Added `context_processors` application.

If you want to configure the Jinja environment, use `JINJA_CONFIG` in `settings.py`. It can be a dict or a function that returns a dict.

```
JINJA_CONFIG = {'autoescape': False}
```

or

```
def JINJA_CONFIG():
    return {'the_answer': 41 + 1}
```

CHAPTER 4

Template Helpers

Instead of template tags, Jinja encourages you to add functions and filters to the templating environment. In `jingo`, we call these helpers. When the Jinja environment is initialized, `jingo` will try to open a `helpers.py` file from every app in `INSTALLED_APPS`. Two decorators are provided to ease the environment extension:

`jingo.register.`**`filter`**`()`
: Adds the decorated function to Jinja's filter library.

`jingo.register.`**`function`**`()`
: Adds the decorated function to Jinja's global namespace.

Default Helpers

Helpers are available in all templates automatically, without any extra loading.

CHAPTER 5

Template Environment

A single Jinja `Environment` is created for use in all templates. This is available as `jingo.env` if you need to work with the `Environment`.

CHAPTER 6

Localization

Since we all love L10n, let's see what it looks like in Jinja templates:

```
<h2>{{ _('Reviews for {0}')|f(addon.name) }}</h2>
```

The simple way is to use the familiar underscore and string within a `{{ }}` moustache block. `f` is an interpolation filter documented below. Sphinx could create a link if I knew how to do that.

The other method uses Jinja's `trans` tag:

```
{% trans user=review.user|user_link, date=review.created|datetime %}
  by {{ user }} on {{ date }}
{% endtrans %}
```

`trans` is nice when you have a lot of text or want to inject some variables directly. Both methods are useful, pick the one that makes you happy.

CHAPTER 7

Forms

Django marks its form HTML "safe" according to its own rules, which Jinja2 does not recognize. Django marks its form HTML "safe" according to its own rules, which Jinja2 does not recognize.

This monkeypatches Django's Form classes to support __html__, which both Django and Jinja2 use to identify already-vetted markup.

Call the `patch()` function to execute the patch. It must be called before `django.forms` is imported for the conditional_escape patch to work properly. The root URLconf is the recommended location for calling `patch()`.

Usage:

```
import jingo.monkey
jingo.monkey.patch()
```

This patch was originally developed by Jeff Balogh.

CHAPTER 8

Testing

To run the test suite, you need to define `DJANGO_SETTINGS_MODULE` first:

```
$ export DJANGO_SETTINGS_MODULE="fake_settings"
$ nosetests
```

or simply run:

```
$ python run_tests.py
```

To test on all supported versions of Python and Django:

```
$ pip install tox
$ tox
```

Python Module Index

j

jingo, 1
jingo.monkey, 15

Index

J

jingo (module), 1
jingo.monkey (module), 15
jingo.register.filter() (in module jingo), 9
jingo.register.function() (in module jingo), 9