django-babel Documentation

Release 0.5.1

Christopher Grebs

November 02, 2016

Contents

1 Tools for using Babel with Django **1**
- 1.1 Extracting Messages . . . 1
- 1.2 Using the Middleware . . . 3
- 1.3 Using the Template Tags . . . 3

2 Changelog **7**
- 2.1 0.5.1 - 2016-03-30 . . . 7
- 2.2 0.5.0 - 2016-02-29 . . . 7
- 2.3 0.4.0 - 2015-04-22 . . . 7
- 2.4 0.3.9 - 2014-12-24 . . . 7
- 2.5 0.3.8 - 2014-10-14 . . . 7
- 2.6 0.3.7 - 2014-10-14 . . . 7
- 2.7 0.3.6 - 2014-10-05 . . . 8
- 2.8 0.3.5 - 2014-09-10 . . . 8
- 2.9 0.3.4 - 2014-05-25 . . . 8
- 2.10 0.3.3 - 2014-04-22 . . . 8
- 2.11 0.3.2 - 2014-04-22 . . . 8
- 2.12 0.3.1 - 2013-12-11 . . . 8
- 2.13 0.3.0 - 2013-12-11 . . . 8
- 2.14 0.2.3 - 2013-12-11 . . . 8

3 Indices and tables **11**

Python Module Index **13**

CHAPTER 1

Tools for using Babel with Django

This package contains various utilities for integration of Babel into the Django web framework:

- A message extraction plugin for Django templates.
- A middleware class that adds the Babel Locale object to requests.
- A set of template tags for date and number formatting.

1.1 Extracting Messages

Babel provides a message extraction framework similar to GNU `xgettext`, but more extensible and geared towards Python applications. While Django does provide wrapper scripts for making the use of `xgettext` more convenient, the extraction functionality is rather limited. For example, you can't use template files with an extension other than `.html`, and everything needs to be in your project package directory.

1.1.1 Extraction Method Mapping

So django-babel comes with an extraction method plugin that can extract localizable messages from Django template files. Python is supported out of the box by Babel. To use this extraction functionality, create a file called `babel.cfg` in your project directory (the directory above your project package), with the content:

```
[django: templates/**.*]
[django: mypkg/*/templates/**.*]
[python: mypkg/**.py]
```

This instructs Babel to look for any files in the top-level `templates` directory, or any files in application `templates` directories, and use the extraction method named "django" to extract messages from those template files. You'll need to adjust those glob patterns to wherever you my be storing your templates.

Also, any files with the extension `.py` inside your package directory (replace "mypkg" with the actual name of your Django project package) are processed by the "python" extraction method.

If you don't use setuptools, or for some reason haven't installed django-babel using setuptools/pip, you'll need to define what function the extraction method "django" maps to. This is done in an extra section at the top of the configuration file:

```
[extractors]
django = django_babel.extract:extract_django
```

The encoding of the templates is assumed to be UTF-8. If you are using a different encoding, you will need to specify it in the configuration. For example:

```
[django: templates/**.*]
encoding = iso-8859-1
```

1.1.2 Running the Extraction Process

Once you've set up the configuration file, the actual extraction is performed by executing the command-line program `pybabel` which is installed alongside the Babel package:

```
$ cd projectdir
$ pybabel extract -F babel.cfg -o mypkg/locale/django.pot .
```

This creates the PO file template in `mypkg/locale/django.pot`.

1.1.3 Creating and Updating Translations Catalogs

If you don't already have translation catalogs, you need to create them. This is done using the `pybabel init` command:

```
$ pybabel init -D django -i mypkg/locale/django.pot -d mypkg/locale -l en_US
$ pybabel init -D django -i mypkg/locale/django.pot -d mypkg/locale -l de_DE
```

This should create two files: `mypkg/locale/en_US/django.po` and `mypkg/locale/de_DE/django.po`. These files are where you put the actual translations.

When you modify your Python source files or your templates, you genereally need to sync the translation catalogs. For that, you first perform a fresh extraction as described in the previous section, so that the `django.pot` file gets updated.

Then, you run the `pybabel update` command to merge the changes into the translation catalogs:

`'bash $ pybabel update -D django -i mypkg/locale/django.pot -d mypkg/locale '`

This will update all the `.po` files found in the `mypkg/locale` directory.

1.1.4 Compiling Translations Catalogs

Finally, you need to compile those `.po` files to binary `.mo` files. Use the *pybabel compile* command for that:

```
$ pybabel compile -D django -d mypkg/locale
```

Add the `--statistics` option to get information about the completeness of your translations:

```
$ pybabel compile -D django -d mypkg/locale --statistics
```

1.1.5 Using `setup.py`

Much of the above process can be automated if you add a `setup.py` script to your project and use the distutils/setuptools commands that come with Babel. This is described at Distutils/Setuptools Integration.

1.2 Using the Middleware

To use the Babel middleware, add it to the list of `MIDDLEWARE_CLASSES` in your settings module. If you're also using Django's own `LocaleMiddleware` to vary the locale based on user preference, the Babel middleware must be inserted after the Django one:

```
MIDDLEWARE_CLASSES = (
    ...
    'django.middleware.locale.LocaleMiddleware',
    'django_babel.middleware.LocaleMiddleware',
    ...
)
```

This adds a `locale` attribute to the request object, which is an instance of the Babel `Locale` class. You can access the locale via `request.locale` when the request object is available, or otherwise use the `django_babel.middleware.get_current_locale()` function to get the current locale from a thread-local cache.

1.3 Using the Template Tags

The template filters provided by django-babel allow formatting of date/time and number values in a locale-sensitive manner, providing much more powerful alternatives to the `date`, `time`, and `floatformat` filters that come with Django.

To make the template filters/tags available, you need to add django-babel to the list of `INSTALLED_APPS` in your settings module:

```
INSTALLED_APPS = (
    ...
    'django_babel',
    ...
)
```

And in every template you want to use the filters, you need to explicitly load the django-babel library:

```
{% load babel %}
```

General information on date/time and number formatting can be found at Date Formatting and Number Formatting.

The following filters are made available. The examples assume a locale of `en_US`.

1.3.1 datefmt

Renders a string representation of a date.

- **Input**: `datetime.date`, `datetime.datetime`, or a float/int timestamp
- **Parameters**: the format name or pattern (optional)

Assuming that `book.pubdate` returns a `datetime.date` or `datetime.datetime` object:

```
{{ book.pubdate|datefmt:"short" }}
```

would render: **4/1/07**, and

```
{{ book.pubdate|datefmt:"E, MMM dd yyyy GGG" }}
```

would render: **Sun, Apr 01 2007 AD**

1.3.2 datetimefmt

Renders a string representation of a date and time.

- **Input**: `datetime.datetime`, or a float/int timestamp
- **Parameters**: the format name or pattern (optional)

Examples:

```
{{ book.pubdate|datetimefmt:"short" }}
```

would render: **4/1/07 3:30 PM**, and

```
{{ book.pubdate|datetimefmt:"E, MMM dd yyyy GGG' - 'HH:mm:ss'" }}
```

would render: **Sun, Apr 01 2007 AD - 15:30:00**

1.3.3 timefmt

Renders a string representation of a time.

- **Input**: `datetime.datetime`, `datetime.time`, or a float/int timestamp
- **Parameters**: the format name or pattern (optional)

Examples:

```
{{ book.pubdate|timefmt:"short" }}
```

would render: **3:30 PM**, and

```
{{ book.pubdate|timefmt:"h 'o''clock' a'" }}
```

would render: **3 o'clock PM**

1.3.4 decimalfmt

Renders a string representation of a decimal number.

- **Input**: a *Decimal* object, or a float/int/long value
- **Parameters**: the format name or pattern (optional)

Examples:

```
{{ book.pagecount|decimalfmt }}
```

would render: **1,234**, and

```
{{ book.pagecount|decimalfmt:"#,##0.00" }}
```

would render: **1,234.00**

1.3.5 currencyfmt

Renders a number formatted as a currency value.

- **Input**: a `Decimal` object, or a float/int/long value
- **Parameters**: the currency code

Examples:

```
{{ book.price|currencyfmt:"USD" }}
```

would render: **$49.90**

1.3.6 percentfmt

Renders a string representation of a number as a percentage.

- **Input**: a `Decimal` object, or a float/int/long value
- **Parameters**: the format name or pattern (optional)

Examples:

Assuming `book.rebate` would return `0.15`,

```
{{ book.rebate|percentfmt }}
```

would render **15%**, and

```
{{ book.rebate|percentfmt:"#,##0.00%" }}
```

would render **15.00%**.

1.3.7 scientificfmt

Renders a string representation of a number using scientific notation.

- **Input**: a `Decimal` object, or a float/int/long value
- **Parameters**: none

Examples:

Assuming `book.numsold` would return 1.000.000,

```
{{ book.numsold|scientificfmt }}
```

would render **10E5**.

CHAPTER 2

Changelog

2.1 0.5.1 - 2016-03-30

- make imports absolute in babel templatetags
- strip quotes from translations via _()
- fix links in docs
- Add support for “trimmed” blocktrans content

2.2 0.5.0 - 2016-02-29

- Add compatibility for Django-1.9

2.3 0.4.0 - 2015-04-22

- Add compatibility for Django 1.8
- Add compatibility for latest django master
- Various python 3 fixes

2.4 0.3.9 - 2014-12-24

- Fix dependencies on Django/Babel to use lower-case egg names.

2.5 0.3.8 - 2014-10-14

- Fix old reference to *babeldjango* module in entry points.

2.6 0.3.7 - 2014-10-14

- Fix Python 3.x compatibility in *babel makemessages* command.

2.7 0.3.6 - 2014-10-05

- Django 1.7 compatibility

2.8 0.3.5 - 2014-09-10

- Create .po and .pot files if not existing, plus it's specific base directories.

2.9 0.3.4 - 2014-05-25

- Fixed django compatibility

2.10 0.3.3 - 2014-04-22

- Fixed release builds

2.11 0.3.2 - 2014-04-22

- Initial testing infrastructure
- Add management command *babel* with *makemessages* and *compilemessages* labels. Mimics django's *makemessages* and *compilemessages* commands.
- Various unicode fixes

2.12 0.3.1 - 2013-12-11

- fix relative import in template tags

2.13 0.3.0 - 2013-12-11

- Rename package to django_babel

2.14 0.2.3 - 2013-12-11

- Rename package on PyPI
- Use GitHub as source control

Contents:

2.14.1 Extract

django_babel.extract.**extract_django**(*fileobj*, *keywords*, *comment_tags*, *options*)
Extract messages from Django template files.

Parameters

- **fileobj** – the file-like object the messages should be extracted from
- **keywords** – a list of keywords (i.e. function names) that should be recognized as translation functions
- **comment_tags** – a list of translator tags to search for and include in the results
- **options** – a dictionary of additional options (optional)

Returns an iterator over `(lineno, funcname, message, comments)` tuples

Return type `iterator`

django_babel.extract.**trim_whitespace**(*string*)
Trim whitespace.

This is only supported in Django>=1.7. This method help in cases of older Django versions.

2.14.2 Management Commands

2.14.3 Middleware

django_babel.middleware.**get_current_locale**()
Get current locale data outside views.

See http://babel.pocoo.org/en/stable/api/core.html#babel.core.Locale for Locale objects documentation

class django_babel.middleware.**LocaleMiddleware**
Simple Django middleware that makes available a Babel *Locale* object via the *request.locale* attribute.

2.14.4 Template Tags

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

d

django_babel.extract, 9
django_babel.management.commands.babel, 9
django_babel.middleware, 9
django_babel.templatetags.babel, 9

D

django_babel.extract (module), 9
django_babel.management.commands.babel (module), 9
django_babel.middleware (module), 9
django_babel.templatetags.babel (module), 9

E

extract_django() (in module django_babel.extract), 9

G

get_current_locale() (in module django_babel.middleware), 9

L

LocaleMiddleware (class in django_babel.middleware), 9

T

trim_whitespace() (in module django_babel.extract), 9