dj-pagination Documentation

Release 2.4.0

Daniel Roy Greenfeld

Sep 11, 2018

Contents

See also:

To get started quickly see *Usage*

See also:

See what's new in version_2_3_3

CHAPTER 1

Features

- Quickly create nice-looking paginated lists without altering your views
- Support for multiple lists per page
- Support for using custom templates for each pagination

CHAPTER 2

Indices and tables

2.1 Installation

2.1.1 Prerequisites

This package requires django 1.8+. It is not tested on earlier versions and may not work properly there.

To build the documentation from source you will need sphinx.

2.1.2 Installation Options

There are several installation options available:

- using Python Package Index, this package is being actively maintained and published in the Python Package Index. You can install it if you have pip tool using just one line:

```
pip install dj-pagination
```

- or installing the development version:

```
git clone git@github.com:pydanny/dj-pagination.git
cd dj-pagination
python setup.py develop
```

2.2 Usage

2.2.1 How to use dj-pagination

`dj-pagination` allows for easy HTML-based pagination without modifying your views.

There are really 5 steps to setting it up with your projects (not including installation, which is covered in *Installation*.)

1. List this application in the INSTALLED_APPS portion of your settings file. Your settings file might look something like:

```
INSTALLED_APPS = (
    # ...
    'dj_pagination',
)
```

2. Install the pagination middleware. Your settings file might look something like:

```
MIDDLEWARE = (
    # ...
    'dj_pagination.middleware.PaginationMiddleware',
)
```

   or MIDDLEWARE_CLASSES for Django <1.10.

3. If it's not already added in your setup, add the request context processor. Note that context processors are set by default implicitly, so to set them explicitly, you need to copy and paste this code into your under the value TEMPLATE_CONTEXT_PROCESSORS:

```
("django.core.context_processors.auth",
"django.core.context_processors.debug",
"django.core.context_processors.i18n",
"django.core.context_processors.media",
"django.core.context_processors.request")
```

4. Add this line at the top of your template to load the pagination tags:

```
{% load pagination_tags %}
```

5. Decide on a variable that you would like to paginate, and use the autopaginate tag on that variable before iterating over it. This could take one of two forms (using the canonical object_list as an example variable):

```
{% autopaginate object_list %}
```

   This assumes that you would like to have the default 20 results per page. If you would like to specify your own amount of results per page, you can specify that like so:

```
{% autopaginate object_list 10 %}
```

   Note that this replaces object_list with the list for the current page, so you can iterate over the object_list like you normally would.

   If you are using template {% block %} tags, the autopaginate tag must exist in the same {% block %} where you access the paginated object_list.

   In general the full syntax is:

```
autopaginate QUERYSET [PAGINATE_BY] [ORPHANS] [as NAME]
```

6. Now you want to display the current page and the available pages, so somewhere after having used autopaginate, use the paginate inclusion tag:

```
{% paginate %}
```

   This does not require any arguments, but does assume that you have already called autopaginate, so make sure to do so first.

That's it! You have now paginated `object_list` and given users of the site a way to navigate between the different pages–all without touching your views.

2.2.2 Custom pagination templates

By default the objects will be paginated using a helper template "pagination/pagination.html". You can change this with an argument to `paginate`.

In general the full syntax is:

```
paginate [using "TEMPLATE"]
```

For example, to paginate posts on a hypothetical blog page you could use something like this:

```
{% autopaginate posts pagesize %}
{% paginate using "pagination/blog/post.html" %}
```

The default pagination template is contained in the `pagination/pagination.html` file inside the distribution. You could extend it and only customize the parts you care about. Please inspect the template to see the blocks it defines that you could customize.

2.2.3 Multiple paginations per page

You can use autopaginate/paginate multiple times in the same template. The only requirement is to call autopaginate before calling paginate. That is, paginate acts on the most recent call to autopaginate.

2.2.4 A Note About Uploads

It is important, when using dj-pagination in conjunction with file uploads, to be aware of when `request.page` is accessed. As soon as `request.page` is accessed, `request.upload_handlers` is frozen and cannot be altered in any way. It's a good idea to access the `page` attribute on the request object as late as possible in your views.

2.2.5 Optional Settings

In dj-pagination, there are no required settings. There are, however, a small set of optional settings useful for changing the default behavior of the pagination tags. Here's an overview:

`PAGINATION_DEFAULT_PAGINATION` The default amount of items to show on a page if no number is specified. Defaults to 20

`PAGINATION_DEFAULT_WINDOW` The number of items to the left and to the right of the current page to display (accounting for ellipses). Defaults to 4.

`PAGINATION_DEFAULT_MARGIN` FIXME: This needs to be documented.

`PAGINATION_DEFAULT_ORPHANS` The number of orphans allowed. According to the Django documentation, orphans are defined as "The minimum number of items allowed on the last page, defaults to zero."

`PAGINATION_INVALID_PAGE_RAISES_404` Determines whether an invalid page raises an `Http404` or just sets the `invalid_page` context variable. `True` does the former and `False` does the latter. Defaults to False

`PAGINATION_DISPLAY_PAGE_LINKS` If set to `False`, links for single pages will not be displayed. Defaults to True.

PAGINATION_PREVIOUS_LINK_DECORATOR An HTML prefix for the previous page link; the default value is `&lsaquo;&lsaquo;`.

PAGINATION_NEXT_LINK_DECORATOR An HTML postfix for the next page link; the default value is `&rsaquo;&rsaquo;`.

PAGINATION_DISPLAY_DISABLED_PREVIOUS_LINK If set to `False`, the previous page link will not be displayed if there's no previous page. Defaults to False.

PAGINATION_DISPLAY_DISABLED_NEXT_LINK If set to `False`, the next page link will not be displayed if there's no next page. Defaults to False.

PAGINATION_DISABLE_LINK_FOR_FIRST_PAGE If set to `False`, the first page will have `?page=1` link suffix in pagination displayed, otherwise is omitted. Defaults to True.

2.3 Version History

2.3.1 Version 2.4.0

- Markdown readme
- Formal support for Django 2.0 and 2.1
- Remove support for unsupported versions of Python and Django

2.3.2 Version 2.3.3

- Formal support for Django 2.0 and 2.1
- Remove support for unsupported versions of Python and Django

2.3.3 Version 2.3.2

Fixed extras_require for py2/3 differences

2.3.4 Version 2.3.1

use extras_require for py2/3 differences

2.3.5 Version 2.3.0

Add request object to context

2.3.6 Version 2.0.4

This is a micro release to push minor fixes to a PyPI release.

2.3.7 Version 2.0.2

This is an another micro release. There are no code changes (apart from setup.py). The only change is to make it pip-friendly by using new integration mode with versiontools.

2.3.8 Version 2.0.1

This is a micro release. There are no code changes so there is no need to upgrade. The only changes are to documentation and infrastructure files.

The following changes are included:

- Improve documentation for using custom pagination templates
- Document multiple paginations per page
- Use correct template name in do_paginate docstring
- Provide correct link to installation instructions
- Fix documentation referencing all project name
- Ignore vim swap files
- Add templates from the test project to MANIFEST.in

2.3.9 Version 2.0

- Revived the project as a fork of git://github.com/ericflo/django-pagination.git. The project now has a new maintainer (Zygmunt Krynicki) and a new home (on pypi and launchpad).
- Merged a lot of branches of the old project. In general this was made to show people "here is the new good stuff" and to get as much contributions, back into the trunk, as possible.
- Merge a lot of translations: de, es, fr, it, nn, no, pl, pt, pt_BR, ru and tr. Translations are still in a bad state (they are not built automatically, they are in incorrect place) but the first step is done.
- Add support for custom pagination templates. You can now use the optional argument on paginate to use different template:

```
{% autopaginate obj_list %}
...
{% paginate using "something/custom_template.html" %}
```

- Pagination template has support for specific blocks. Those blocks are 'previouslink', 'pagelinks' and 'nextlink'. Make sure to base your template on pagination/pagination.html end extend the blocks you care about.
- Add support for using multiple paginations on a single page. Simply use multiple autopaginate/paginate tags. The only limitation is that you must use paginate before using the next autopaginate tag. For an example see the test project and the example application inside.
- Simplify building documentation. To build the documentation simply run *setup.py build_sphinx*. You will need sphinx installed obviously.
- Simplify running tests. To run tests just invoke *setup.py test*. That's all! This is based on the goodness of django-testproject that simplifies setting up helper projects just for testing.

2.3.10 Version 1.0.7

- Last release from previous upstream developer.
- genindex
- modindex
- search