Django Simple Elasticsearch Documentation

Release 2.2.0

James Addison

Sep 27, 2017

Contents

This package provides a simple method of creating Elasticsearch indexes for Django models.

CHAPTER 1

Versions

Branch `master` targets both Elasticsearch 2.x and 5.x and will receive new features. Both *elasticsearch-py* 2.x and 5.x Python modules are currently supported. Documentation

Branch `1.x` is the maintenance branch for the legacy 0.9.x versioned releases, which targeted Elasticsearch versions less than 2.0. This branch is unlikely to receive new features, but will receive required fixes. Documentation

Using a version older than 0.9.0? Please be aware that as of v0.9.0, this package has changed in a backwards-incompatible manner. Version 0.5 is deprecated and no longer maintained.

CHAPTER 2

Documentation

Visit the django-simple-elasticsearch documentation on ReadTheDocs.

CHAPTER 3

Features

- class mixin with a set of `@classmethods` used to handle:
- type mapping definition
- individual object indexing and deletion
- bulk object indexing
- model signal handlers for pre/post_save and pre/post_delete (optional)
- management command to handle index/type mapping initialization and bulk indexing
- uses Elasticsearch aliases to ease the burden of re-indexing
- small set of Django classes and functions to help deal with Elasticsearch querying
- base search form class to handle input validation, query preparation and response handling
- multi-search processor class to batch multiple Elasticsearch queries via `_msearch`
- 'get' shortcut functions
- post index create/rebuild signals available to perform actions after certain stages (ie. add your own percolators)

CHAPTER 4

Installation

At the command line:

```
$ easy_install django-simple-elasticsearch
```

Or:

```
$ pip install django-simple-elasticsearch
```

CHAPTER 5

Configuring

Add the simple_elasticsearch application to your INSTALLED_APPS list:

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

Add any models to *ELASTICSEARCH_TYPE_CLASSES* setting for indexing using **es_manage** management command:

```
ELASTICSEARCH_TYPE_CLASSES = [
    'blog.models.BlogPost'
]
```

CHAPTER 6

License

django-simple-elasticsearch is licensed as free software under the BSD license.

CHAPTER 7

Todo

- Review search classes - simplify functionality where possible. This may cause breaking changes.
- Tests. Write them.
- Documentation. Write it.

CHAPTER 8

Jump to a section

Usage

For a minimal investment of time, Django Simple Elasticsearch offers a number of perks. Implementing a class with the `ElasticsearchTypeMixin` lets you:

- initialize your Elasticsearch indices and mappings via the included `es_manage` management command
- perform Elasticsearch bulk indexing via the same `es_manage` management command
- perform Elasticsearch bulk indexing as well as individual index/delete requests on demand in your code
- connect the available `ElasticsearchTypeMixin` save and delete handlers to Django's available model signals (ie `post_save`, `post_delete`)

Let's look at an example implementation of `ElasticsearchTypeMixin`. Here's a couple of blog-related Models in a `models.py` file:

```
class Blog(models.Model):
    name = models.CharField(max_length=50)
    description = models.TextField()

class BlogPost(models.Model):
    blog = models.ForeignKey(Blog)
    slug = models.SlugField()
    title = models.CharField(max_length=50)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

To start with `simple_elasticsearch`, you'll need to tell it that the `BlogPost` class implements the `ElasticsearchTypeMixin` mixin, so in your `settings.py` set the `ELASTICSEARCH_TYPE_CLASSES` setting:

```
ELASTICSEARCH_TYPE_CLASSES = [
    'blog.models.BlogPost'
]
```

If you do not add this setting, everything will still work except for the `es_manage` command - it won't know what indices to create, type mappings to set or what objects to index. As you add additional `ElasticsearchTypeMixin`-based index handlers, add them to this list.

All right, let's add in `ElasticsearchTypeMixin` to the `BlogPost` model. Only pertinent changes from the above `models.py` are shown:

```
from simple_elasticsearch.mixins import ElasticsearchTypeMixin

...

class BlogPost(models.Model, ElasticsearchTypeMixin):
    blog = models.ForeignKey(Blog)
    slug = models.SlugField()
    title = models.CharField(max_length=50)
    body = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    @classmethod
    def get_queryset(cls):
        return BlogPost.objects.all().select_related('blog')

    @classmethod
    def get_index_name(cls):
        return 'blog'

    @classmethod
    def get_type_name(cls):
        return 'posts'

    @classmethod
    def get_type_mapping(cls):
        return {
            "properties": {
                "created_at": {
                    "type": "date",
                    "format": "dateOptionalTime"
                },
                "title": {
                    "type": "string"
                },
                "body": {
                    "type": "string"
                },
                "slug": {
                    "type": "string"
                },
                "blog": {
                    "properties": {
                        "id": {
                            "type": "long"
                        },
                        "name": {
                            "type": "string"
                        },
                        "description": {
                            "type": "string"
                        }
                    }
```

```
                }
            }
        }

    @classmethod
    def get_document(cls, obj):
        return {
            'created_at': obj.created_at,
            'title': obj.title,
            'body': obj.body,
            'slug': obj.slug,
            'blog': {
                'id': obj.blog.id,
                'name': obj.blog.name,
                'description': obj.blog.description,
            }
        }
```

With this mixin implementation, you can now use the `es_manage` management command to bulk reindex all `BlogPost` items. Note that there are additional `@classmethods` you can override to customize functionality. Sane defaults have been provided for these - see the source for details.

Of course, our `BlogPost` implementation doesn't ensure that your Elasticsearch index is updated every time you save or delete - for this, you can use the `ElasticsearchTypeMixin` built-in save and delete handlers.

```
from django.db.models.signals import post_save, pre_delete

...

post_save.connect(BlogPost.save_handler, sender=BlogPost)
pre_delete.connect(BlogPost.delete_handler, sender=BlogPost)
```

Awesome - Django's magic is applied.

Notes

- Prior to version 2.2.0 of this package, only models with numerical primary keys could be indexed properly due to the way the `queryset_iterator()` utility function was implemented. This has been changed and the primary key no longer matters.

  Ordering the bulk queryset is important due to the fact that records may have been added during the indexing process (indexing data can take a long time); if the results are ordered properly, the indexing process will catch the most recent records. For most cases, the default bulk ordering of `pk` will suffice (Django's default primary key field is an auto-incrementing integer).

  If a model has PK using a `UUIDField` however, things change: UUIDs are randomly generated, so ordering by a `UUIDField` PK will most likely result in newly created items being missed in the indexin process. Overriding the `ElasticsearchTypeMixin` class method `get_bulk_ordering()` addresses this issue - set it to order by a `DateTimeField` on the model.

TODO:

- add examples for more complex data situations
- add examples of using `es_manage` management command options

- add examples/scenarios when to use `post_indices_create` and `post_indices_rebuild` signals (ie. adding percolators to new indices)

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/jaddison/django-simple-elasticsearch/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Django Simple Elasticsearch could always use more documentation, whether as part of the official Django Simple Elasticsearch docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/jaddison/django-simple-elasticsearch/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *django-simple-elasticsearch* for local development.

1. Fork the *django-simple-elasticsearch* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/django-simple-elasticsearch.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv django-simple-elasticsearch
$ cd django-simple-elasticsearch/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 simple_elasticsearch
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/jaddison/django-simple-elasticsearch/pull_requests and make sure that the tests pass for all supported Python versions.

Credits

Development Lead

- James Addison <addi00+github.com@gmail.com>

Contributors

None yet. Why not be the first?

History

2.2.0 (2017-07-17)

- Addressing inability to index models with a non-integer PK field (ie. *UUIDField*) - added ability to order bulk queryset on an arbitrary model field.

2.1.7 (2017-03-21)

- Allowing direct access (again) to underlying dict/list in *Result* and *Response* classes for serialization and other purposes.

2.1.5 (2017-03-20)

- Response class is now MutableSequence based, giving it the properties of a *list*. Its *results* attribute is deprecated, as you can now iterate over the results with the response instance itself.
- Result class *results_meta* is deprecated. Use *meta* instead.
- *get_from_es_or_None* now returns a *Result* object instead of the raw Elasticsearch result, for consistency.
- *get_from_es_or_None* now catches only the Elasticsearch *NotFoundError* exception; previously it caught the more expansive *ElasticsearchException*, which could hide unrelated errors/issues.

2.1.4 (2017-03-12)

- Result class is now MutableMapping based, giving it the properties of a *dict*. Its *data* attribute is deprecated.

2.1.3 (2017-03-11)

- Minor changes to enable support for elasticsearch-py 5.x.

2.1.0 (2017-03-10)

- Addressing a packaging problem which erroneously included pyc/__pycache__ files.

2.0.0 (2016-12-20)

- **ALERT: this is a backwards incompatible release**; the old *1.x* (formerly '0.9.x'+) code is maintained on a separate branch for now.
- Added support for Django 1.10.
- Ported delete/cleanup functionality from *1.x*.
- Removed support for Django versions older than 1.8. The goal going forward will be to only support Django versions that the Django core team lists as supported.
- Removed elasticsearch-dsl support: responses and results are now represented by simpler internal representations; queries can ONLY be done via a *dict* form.
- Removed *ElasticsearchForm* - it is easy enough to build a form to validate search input and then form a query *dict* manually.
- Renamed *ElasticsearchIndexMixin* to *ElasticsearchTypeMixin*, seeing as the mixin represented an ES type mapping, not an actual index.
- Renamed *ElasticsearchProcessor* to *SimpleSearch*.
- Overall, this module has been greatly simplified.

1.0.0 (2016-12-20)

- Updated 0.9.x codebase version to 1.0.0.
- Reversed decision on the deprecation of the 0.9.x codebase - it will be maintained in this new 1.x branch, although new functionality will mostly occur on newer releases.
- Adding cleanup command to remove unaliased indices.
- Added ELASTICSEARCH_DELETE_OLD_INDEXES setting to auto-remove after a rebuild.
- Thanks to Github user @jimjkelly for the index removal inspiration.

0.9.16 (2015-04-24)

- Addressing Django 1.8 warnings.

0.9.15 (2015-01-31)

- BUGFIX: Merging pull request from @key that addresses Python 3 support (management command now works).

0.9.14 (2015-01-31)

- BUGFIX: Adding in missing *signals.py* file.

0.9.13 (2015-01-30)

- Added in new *post_indices_create* and *post_indices_rebuild* signals to allow users to run actions at various points during the index creation and bulk indexing processes.

0.9.12 (2014-12-17)

- fixed an issue where per-item request parameters were being added to the bulk data request JSON incorrectly. Tests updated.

0.9.11 (2014-12-08)

- added warning if Django's DEBUG=True (causes out of memory errors on constrained systems due to Django query caching)
- added index setting modification on rebuilding indices to remove replicas, lucene segment merging and disabling the refresh interval - restoring the original settings afterwards.

0.9.10 (2014-12-04)

- added *page* and *page_size* validation in *add_search()*

0.9.9 (2014-11-24)

- Renamed search form related classes - more breaking changes. Added in support for Django's pagination classes (internal hack).

0.9.8 (2014-11-23)

- Revamped search form related classes - includes breaking changes.

0.9.7 (2014-11-16)

- Python3 supported mentioned in PyPi categorization; new testcases added. Minor interface change (added *@classmethod*).

0.9.6 (2014-11-16)

- Python 3.3+ support, modified (no new) testcases.

0.9.5 (2014-11-15)

- Added in tox support, initial set of test cases and verified travis-ci is working.

0.9.2 (2014-11-12)

- Fixed broken management command.

0.9.1 (2014-11-10)

- Added missing management command module.

0.9.0 (2014-11-10)

- In what will become version 1.0, this 0.9.x codebase is a revamp of the original codebase (v0.5.x). Completely breaking over previous versions.

0.5.0 (2014-03-05)

Final release in 0.x codebase - this old codebase is now unmaintained.

Indices and tables

- genindex
- modindex
- search