django-prefetch

Release 1.1.0

January 03, 2017

CHAPTER 1

Overview

docs	
tests	
package	

Simple and generic model related data prefetch framework for Django solving the “1+N queries” problem that happens when you need related data for your objects.

In most of the cases you’ll have forward relations (foreign keys to something) and can use select_related to fetch that data on the same query. However, in some cases you cannot design your models that way and need data from reverse relations (models that have foreign keys to your objects).

Django 1.4 has prefetch_related for this, however, this framework provides greater flexibility than Django 1.4’s prefetch_related queryset method at the cost of writting the mapping and query functions for the data. This has the advantage that you can do things prefetch_related cannot (see the latest_book *example* bellow).

- Free software: BSD license

1.1 Installation guide

Install it:

```
pip install django-prefetch
```

Use it as your model’s default manager (or as a base class if you have custom manager).

1.2 Requirements

OS Any

Runtime Python 2.6, 2.7, 3.3+ or PyPy

Packages Django>=1.1

1.3 Example

Here's a simple example of models and prefetch setup:

```
from django.db import models
from prefetch import PrefetchManager, Prefetcher

class Author(models.Model):
    name = models.CharField(max_length=100)

    objects = PrefetchManager(
        books = Prefetcher(
            filter = lambda ids: Book.objects.filter(author__in=ids),
            reverse_mapper = lambda book: [book.author_id],
            decorator = lambda author, books=(): setattr(author, 'books', books)
        ),
        latest_book = Prefetcher(
            filter = lambda ids: Book.objects.filter(author__in=ids),
            reverse_mapper = lambda book: [book.author_id],
            decorator = lambda author, books=(): setattr(
                author,
                'latest_book',
                max(books, key=lambda book: book.created)
            )
        )
    )

class Book(models.Model):
    class Meta:
        get_latest_by = 'created'

    name = models.CharField(max_length=100)
    created = models.DateTimeField(auto_now_add=True)
    author = models.ForeignKey(Author)
```

Use it like this:

```
for a in Author.objects.prefetch('books', 'latest_book'):
    print a.books
    print a.latest_book
```

1.3.1 Prefetcher arguments

Example models:

```
class LatestNBooks(Prefetcher):
    def __init__(self, count=2):
        self.count = count

    def filter(self, ids):
        return Book.objects.filter(author__in=ids)

    def reverse_mapper(self, book):
        return [book.author_id]

    def decorator(self, author, books=()):
        books = sorted(books, key=lambda book: book.created, reverse=True)
        setattr(author,
                'latest_%s_books' % self.count,
                books[:self.count])

class Author(models.Model):
    name = models.CharField(max_length=100)

    objects = PrefetchManager(
        latest_n_books = LatestNBooks
    )
```

Use it like this:

```
from prefetch import P

for a in Author.objects.prefetch(P('latest_n_books', count=5)):
    print a.latest_5_book
```

Note: `P` is optional and you can only use for prefetch definitions that are Prefetcher subclasses. You can't use it with prefetcher-instance style definitions like in the first example. Don't worry, if you do, you will get an exception explaining what's wrong.

1.3.2 Other examples

Check out the tests for more examples.

1.4 TODO

- Document `collect` option of `Prefetcher`
- Create tests covering custom `collect` and `mapper`

CHAPTER 2

Installation

At the command line:

```
pip install django-prefetch
```

CHAPTER 3

Usage

To use django-prefetch in a project:

```
import prefetch
```

CHAPTER 4

Reference

4.1 prefetch

class `prefetch.Prefetcher`(*filter=None*, *reverse_mapper=None*, *decorator=None*, *mapper=None*, *collect=None*)

Prefetch definitition. For convenience you can either subclass this and define the methods on the subclass or just pass the functions to the contructor.

Eg, subclassing:

```
class GroupPrefetcher(Prefetcher):

    @staticmethod
    def filter(ids):
        return User.groups.through.objects.filter(user__in=ids).select_related('group')

    @staticmethod
    def reverse_mapper(user_group_association):
        return [user_group_association.user_id]

    @staticmethod
    def decorator(user, user_group_associations=()):
        setattr(user, 'prefetched_groups', [i.group for i in user_group_associations])
```

Or with contructor:

```
Prefetcher(
    filter = lambda ids: User.groups.through.objects.filter(user__in=ids).select_related('group'
    reverse_mapper = lambda user_group_association: [user_group_association.user_id],
    decorator = lambda user, user_group_associations=(): setattr(user, 'prefetched_groups', [
        i.group for i in user_group_associations
    ])
)
```

Glossary:

- filter(list_of_ids):

  A function that returns a queryset containing all the related data for a given list of keys. Takes a list of ids as argument.

- reverse_mapper(related_object):

  A function that takes the related object as argument and returns a list of keys that maps that related object to the objects in the queryset.

- mapper(object):

  Optional (defaults to `lambda obj:  obj.id`).

  A function that returns the key for a given object in your query set.

- decorator(object, list_of_related_objects):

  A function that will save the related data on each of your objects in your queryset. Takes the object and a list of related objects as arguments. Note that you should not override existing attributes on the model instance here.

CHAPTER 5

Contributing

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

5.1 Bug reports

When 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/ionelmc/django-prefetch/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 code contributions are welcome :)

5.4 Development

To set up *django-prefetch* for local development:

1. Fork django-prefetch (look for the "Fork" button).
2. Clone your fork locally:

```
    git clone git@github.com:your_name_here/django-prefetch.git
```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
    tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`) [1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Ionel Cristian Mărie - http://blog.ionelmc.ro
- Markus Kaiserswerth - https://github.com/mkai
- Marcin Szamotulski - https://github.com/coot
- Slava Bacherikov - https://github.com/bacher09
- George Ma - https://github.com/georgema1982

CHAPTER 7

Changelog

7.1 1.1.0 (2016-02-20)

- Fixed a test assertion. Contributed by George Ma in #12.
- Added support for Django 1.9. Contributed by Will Stott in #14.
- Fixed use of deprecated *field.rel.to* momdel API (Django 1.9+).

7.2 1.0.1 (2015-09-05)

- Fixed manager type check. Contributed by George Ma in #11.

7.3 1.0.0 (2014-12-05)

- Fixed issues with `select_related` being removed when prefetch is used (#9).

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index