django$_{celery_{monitor}}$ Documentation

Release 1.1.2

Jannis Leidel

Nov 10, 2021

CONTENTS

Version 1.1.2

Web https://django-celery-monitor.readthedocs.io/

Download https://pypi.org/project/django_celery_monitor/

Source https://github.com/jazzband/django-celery-monitor

Keywords django, celery, events, monitoring

CHAPTER

ONE

ABOUT

This extension enables you to monitor Celery tasks and workers.

It defines two models (`django_celery_monitor.models.WorkerState` and `django_celery_monitor.models.TaskState`) used to store worker and task states and you can query this database table like any other Django model. It provides a Camera class (`django_celery_monitor.camera.Camera`) to be used with the Celery events command line tool to automatically populate the two models with the current state of the Celery workers and tasks.

CHAPTER

TWO

HISTORY

This package is a Celery 4 compatible port of the Django admin based monitoring feature that was included in the old django-celery package which is only compatible with Celery < 4.0. Other parts of django-celery were released as django-celery-beat (Database-backed Periodic Tasks) and django-celery-results (Celery result backends for Django).

CHAPTER

THREE

INSTALLATION

You can install django_celery_monitor either via the Python Package Index (PyPI) or from source.

To install using *pip*,:

```
$ pip install -U django_celery_monitor
```

CHAPTER

FOUR

USAGE

To use this with your project you need to follow these steps:

1. Install the django_celery_monitor library:

```
$ pip install django_celery_monitor
```

2. Add `django_celery_monitor` to `INSTALLED_APPS` in your Django project's `settings.py`:

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

   Note that there is no dash in the module name, only underscores.

3. Create the Celery database tables by performing a database migrations:

```
$ python manage.py migrate celery_monitor
```

4. Go to the Django admin of your site and look for the "Celery Monitor" section.

CHAPTER

FIVE

STARTING THE MONITORING PROCESS

To enable taking snapshots of the current state of tasks and workers you'll want to run the Celery events command with the appropriate camera class `django_celery_monitor.camera.Camera`:

```
$ celery -A proj events -l info --camera django_celery_monitor.camera.Camera --
↪frequency=2.0
```

For a complete listing of the command-line options available see:

```
$ celery events --help
```

CHAPTER

SIX

CONFIGURATION

There are a few settings that regulate how long the task monitor should keep state entries in the database. Either of the three should be a `datetime.timedelta` value or `None`.

- `monitor_task_success_expires` – Defaults to `timedelta(days=1)` (1 day)

  The period of time to retain monitoring information about tasks with a `SUCCESS` result.

- `monitor_task_error_expires` – Defaults to `timedelta(days=3)` (3 days)

  The period of time to retain monitoring information about tasks with an errornous result (one of the following event states: `RETRY`, `FAILURE`, `REVOKED`.

- `monitor_task_pending_expires` – Defaults to `timedelta(days=5)` (5 days)

  The period of time to retain monitoring information about tasks with a pending result (one of the following event states: `PENDING`, `RECEIVED`, `STARTED`, `REJECTED`, `RETRY`.

In your Celery configuration simply set them to override the defaults, e.g.:

```
from datetime import timedelta

monitor_task_success_expires = timedelta(days=7)
```

CHAPTER

SEVEN

CONTENTS

7.1 Copyright

django_celery_monitor User Manual

by Ask Solem and Jannis Leidel

Note: While the django_celery_monitor *documentation* is offered under the Creative Commons *Attribution-ShareAlike 4.0 International* license the django_celery_monitor *software* is offered under the BSD License (3 Clause)

7.2 API Reference

Release 1.1

Date Nov 10, 2021

7.2.1 django_celery_monitor.camera

The Celery events camera.

class django_celery_monitor.camera.**Camera**(**args*, ***kwargs*)

The Celery events Polaroid snapshot camera.

property TaskState

Return the data model to store task state in.

property WorkerState

Return the data model to store worker state in.

property expire_task_states

Return a twople of Celery task states and expiration timedeltas.

handle_task(*uuid_task*, *worker=None*)
Handle snapshotted event.

7.2.2 django_celery_monitor.humanize

Some helpers to humanize values.

django_celery_monitor.humanize.**naturaldate**(*date*, *include_seconds=False*)
Convert datetime into a human natural date string.

django_celery_monitor.humanize.**pluralize_day**(*n*)
Return a string with the number of days ago.

django_celery_monitor.humanize.**pluralize_month**(*n*)
Return a string with the number of months ago.

django_celery_monitor.humanize.**pluralize_week**(*n*)
Return a string with the number of weeks ago.

django_celery_monitor.humanize.**pluralize_year**(*n*)
Return a string with the number of yeargs ago.

7.2.3 django_celery_monitor.managers

The model managers.

class django_celery_monitor.managers.**ExtendedQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
A custom model queryset that implements a few helpful methods.

select_for_update_or_create(*defaults=None*, ***kwargs*)
Extend update_or_create with select_for_update.

Look up an object with the given kwargs, updating one with defaults if it exists, otherwise create a new one. Return a tuple (object, created), where created is a boolean specifying whether an object was created.

This is a backport from Django 1.11 (https://code.djangoproject.com/ticket/26804) to support select_for_update when getting the object.

class django_celery_monitor.managers.**TaskStateQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
A custom model queryset for the TaskState model with some helpers.

active()
Return all active task states.

expire_by_states(*states*, *expires*)
Expire task with one of the given states.

expired(*states*, *expires*)
Return all expired task states.

purge()
Purge all expired task states.

class django_celery_monitor.managers.**WorkerStateQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
A custom model queryset for the WorkerState model with some helpers.

7.2.4 django_celery_monitor.models

The data models for the task and worker states.

class django_celery_monitor.models.**TaskState**(**args*, ***kwargs*)

The data model to store the task state in.

exception DoesNotExist

exception MultipleObjectsReturned

args

The positional task arguments.

eta

An optional `datetime` describing the ETA for its processing.

expires

An optional `datetime` describing when the task expires.

hidden

Whether the task has been expired and will be purged by the event framework.

kwargs

The keyword task arguments.

name

The task name.

objects = <django.db.models.manager.ManagerFromTaskStateQuerySet object>

A `TaskStateManager` instance to query the *`TaskState`* model.

result

The result of the task.

retries

The number of retries.

runtime

The task runtime in seconds.

state

The `task state` as returned by Celery.

task_id

The task `UUID`.

traceback

The Python error traceback if raised.

tstamp

A `datetime` describing when the task was received.

worker

The worker responsible for the execution of the task.

class django_celery_monitor.models.**WorkerState**(**args*, ***kwargs*)

The data model to store the worker state in.

exception DoesNotExist

exception MultipleObjectsReturned

hostname

The hostname of the Celery worker.

is_alive()
Return whether the worker is currently alive or not.

last_heartbeat
A `datetime` describing when the worker was last seen.

objects = <django.db.models.manager.ManagerFromWorkerStateQuerySet object>
A `ExtendedManager` instance to query the *WorkerState* model.

7.2.5 django_celery_monitor.utils

Utilities.

django_celery_monitor.utils.**action**(*short_description*, ***kwargs*)
Set some admin action attributes.

django_celery_monitor.utils.**correct_awareness**(*value*)
Fix the given datetime timezone awareness.

django_celery_monitor.utils.**display_field**(*short_description*, *admin_order_field*, *allow_tags=True*, ***kwargs*)
Set some display_field attributes.

django_celery_monitor.utils.**fixedwidth**(*field*, *name=None*, *pt=6*, *width=16*, *maxlen=64*, *pretty=False*)
Render a field with a fixed width.

django_celery_monitor.utils.**fromtimestamp**(*value*)
Return an aware or naive datetime from the given timestamp.

django_celery_monitor.utils.**make_aware**(*value*)
Make the given datetime aware of a timezone.

7.3 Change history

release-date 2017-05-18 11:30 a.m. UTC+2

release-by Jannis Leidel

- More packaging fixes. Sigh.

  release-date 2017-05-18 10:30 a.m. UTC+2

  release-by Jannis Leidel

- Fix the folder that the extra stylesheet file was stored in.

  release-date 2017-05-11 10:25 p.m. UTC+2

  release-by Jannis Leidel

- Use `SELECT FOR UPDATE` SQL statements for updating the task and worker states to improve resilliance against race conditions by multiple simultaneously running cameras.

- Move worker state cache from in-process dictionary into database side timestamp to decide whether to do another worker update or not.

- Improved code structure by moving all utilities into same module.

  release-date 2017-05-08 16:05 a.m. UTC+2

  release-by Jannis Leidel

- Import Django models inline to prevent import time side effect.
- Run django.setup() when installing the Camera.

  release-date 2017-05-03 10:17 a.m. UTC+2

  release-by Jannis Leidel

- Fix the Python package manifest.
- Fix README rendering.

7.3.1 1.0.0

release-date 2017-05-03 08:35 a.m. UTC+2

release-by Jannis Leidel

- Initial release by extracting the monitor code from the old django-celery app.
- Add ability to override the expiry timedelta for the task monitor via the Celery configuration.
- Add Python 3.6 and Django 1.11 to text matrix. Supported versions of Django 1.8 LTS, 1.9, 1.10 and 1.11 LTS. Supported versions of Python are 2.7, 3.4, 3.5 and 3.6 (for Django 1.11).

PYTHON MODULE INDEX

d

INDEX

O

P

R

S

T

W