pyramid_cron Documentation

Release 0.1.1.dev

Scott Torborg

November 22, 2016

Contents

Scott Torborg - Cart Logic

Provides the ability to register simple tasks (callback functions) for scheduled execution with a cron-like syntax.

Why it's better than a typical task queue like Celery, Resque, etc:

- No user permissions to worry about: everything is run inside a web request, so the task has all the same permissions as your web app.
- Very simple setup, no additional daemons required.
- The API follows Pyramid idioms.

Why it's worse:

- It's not well suited to long-running tasks: everything is run inside a web request.
- It does not distribute jobs across workers.
- It does not allow for prioritization of jobs, or have any support for non-synchronous tasks.

CHAPTER 1

Contents

1.1 Quick Start

1.1.1 Install

Install with pip:

```
$ pip install pyramid_cron
```

For each web app which uses pyramid_cron, you'll need to fire the cron handler from a whitelisted IP once per minute. An easy way to do this is by adding a cron job on the app server:

```
* * * * *   curl -o /dev/null http://localhost/cron
```

1.1.2 Integrate with a Pyramid App

Include pyramid_cron, by calling `config.include('pyramid_cron')` or adding pyramid_cron to `pyramid.includes`.

1.1.3 Register a Task

Register at least one task, using the `config.add_cron_task()` directive. You can also pass a dotted string (e.g. `myapp.tasks.some_task`) which will be resolved relative to the calling module.

Tasks are functions which accept a single `system` argument. `system` is a dict with two keys: `request` and `registry`, both of which refer to the Pyramid objects of that name.

```
def my_task(system):
    registry = system['registry']
    request = system['request']
    # do stuff

# Run every 3 hours.
config.add_cron_task(my_task, hour=(0, 24, 3))
```

See API Reference for more details.

1.1.4 Request Scope Caveats

All tasks that are run during a given minute will be run in the scope of the same request. This may impose constraints on your tasks, for example:

- Depending on your transaction management infrastructure, tasks will share the same SQL session.
- An accumulation of slow tasks may lead to an abnormally long HTTP request, tying up resources or exceeding your webserver's timeout threshold.
- Ideally, tasks which are on the slower side should be staggered so that they're unlikely to run at the same time.

1.1.5 Logging

Information about task execution (and timing) is logged to the `pyramid_cron` handler. If you wish to record it, you should configure logging explicitly for that handler in your app.

1.2 API Reference

`pyramid_cron.`**`add_cron_task`**(*config*, *f*, *min='*'*, *hour='*'*, *day='*'*, *month='*'*, *dow='*'*, *idle=False*)

Register a function for execution by the scheduler.

Task functions must have the following signature:

```
def mytask(system):
    request = system['request']
    registry = system['registry']
    time_left = system['time_left']
    # do stuff
```

Additional keys may be added in the future: the single-arg signature ensures that task functions will be forward-compatible.

The `time_left` member is a no-parameter function that returns how many seconds are remaining in the allotted 60 seconds of the current cron run. When the 60 seconds is exceeded, the returned value will be negative.

In addition to the callback function, you can specify a schedule, using a cron-like syntax. For the time periods of `min`, `hour`, `day`, `month`, and `dow` (day of week), you can specify an integer, a set of integers, or the '*' wildcard character. The default argument is '*'. Hours are specified in 24-hour time.

For example, this will run the task every day, at 2:00:

```
config.add_cron_task(..., hour=2)
```

This will run the task every day at 2:00, 10:00, and 18:00:

```
config.add_cron_task(..., hour=[2, 10, 18])
```

To run the task 'every 2 hours', you can use `range()`:

```
config.add_cron_task(..., hour=range(0, 24, 2))
```

Parameters

- **`f`** – The function to execute. Task functions must have accept a single argument, which will be a `system` dict containing keys for the Pyramid `request` and `registry`.

- **min** – Specify which minutes to run the task.
- **hour** – Specify which hours to run the task.
- **day** – Specify which days to run the task.
- **month** – Specify which months to run the task.
- **dow** – Specify which days of the week to run the task.
- **idle** – If true, executes the task after all non-idle tasks have completed, and only when there is time remaining in the 60 second window since the cron view was triggered.

1.3 Contributing

Patches and suggestions are strongly encouraged! GitHub pull requests are preferred, but other mechanisms of feedback are welcome.

pyramid_cron has a comprehensive test suite with 100% line and branch coverage, as reported by the excellent `coverage` module. To run the tests, simply run in the top level of the repo:

```
$ tox
```

This will also ensure that the Sphinx documentation builds correctly, and that there are no PEP8 or Pyflakes warnings in the codebase.

Any pull requests should preserve all of these things.

CHAPTER 2

Indices and Tables

- genindex
- modindex

A