Python

Jun 28, 2018

Contents

Integration of Celery 4 with Zope 3.

This package is compatible with Python version 2.7.

CHAPTER 1

Features

- integration into the Zope transaction (schedule tasks at `transaction.commit()`)
- runs jobs
  - in a Zope environment with loaded ZCML and ZODB connection
  - in a transaction with retry on *ConflictError*
  - as the user who scheduled the job
- test infrastructure to run tests in-line or in a worker
- support for py.test fixtures and zope.testrunner layers

1.1 Usage

1.1.1 Integration with Zope

To successfully configure Celery in Zope place the `celeryconfig.py` in the `PYTHONPATH`. The configuration will be taken from there.

Define your tasks as `shared_task()` so they can be used in the tests and when running the server.

z3c.celery provides its own celery app: `z3c.celery.CELERY`. It does the actual the integration work.

Jobs by default run as the same principal that was active when the job was enqueued. You can override this by passing a different principal id to `delay`:

```
my_task.delay(my, args, etc, _principal_id_='zope.otheruser')
```

1.1.2 Worker setup

Place the `celeryconfig.py` in your working directory. Now you can start the celery worker using the following command:

```
$ celery worker --app=z3c.celery.CELERY --config=celeryconfig
```

The celeryconfig can include all default celery config options. In addition the variable `ZOPE_CONF` pointing to your `zope.conf` has to be present. This `celeryconfig.py` and the referenced `zope.conf` should be identical to the ones, your Zope is started with.

Additionally you can specify a variable `LOGGING_INI` pointing to a logging config (an ini file in configuration file format, which might be your paste.ini). See *Logging* for details.

Example:

```
ZOPE_CONF = '/path/to/zope.conf'
LOGGING_INI = '/path/to/paste.ini'
broker_url = 'redis://localhost:6379/0'
result_backend = 'redis://localhost:6379/0'
imports = ['my.tasks']
```

1.1.3 Execute code after `transaction.abort()`

If running a task fails the transaction is aborted. In case you need to write something to the ZODB raise `z3c.celery.celery.HandleAfterAbort` in your task. This exception takes a callable and its arguments. It is run in a separate transaction after `transaction.abort()` for the task was called.

It is possible to pass a keyword argument `message` into `HandleAfterAbort`. This message will be serialized and returned to celery in the task result. It is not passed to the callback.

1.1.4 Accessing the `task_id` in the task

There seems currently no way to get the task_id from inside the task when it is a shared task. The task implementation in `z3c.celery` provides a solution. You have to bind the shared task. This allows you to access the task instance as first parameter of the task function. The `task_id` is stored there on the `task_id` attribute. Example:

```
@shared_task(bind=True)
def get_task_id(self):
    """Get the task id of the job."""
    return self.task_id
```

1.1.5 Logging

`z3c.celery` provides a special formatter for the python logging module, which can also be used as a generic formatter as it will omit task specific output if there is none. It allows to include task id and task name of the current task in the log message if they are available. Include it in your logging configuration:

```
[formatter_generic]
class = z3c.celery.logging.TaskFormatter
format = %(asctime)s %(task_name)s %(task_id)s %(message)s
```

1.1.6 Running end to end tests using layers

Motivation: Celery 4.x provides py.test fixtures. There is some infrastructure in this package to use these fixtures together with *plone.testing.Layer*. The following steps are required to set the layers up correctly:

In your package depend on `z3c.celery[layer]`.

Create a layer which provides the following resources:

- `celery_config`: dict of config options for the celery app. It has to include a key `ZOPE_CONF` which has to point to a *zope.conf* file. See the template in `z3c.celery.testing`.
- `celery_parameters`: dict of parameters used to instantiate Celery
- `celery_worker_parameters`: dict of parameters used to instantiate celery workers
- `celery_includes`: list of dotted names to load the tasks in the worker

Example:

```
class CelerySettingsLayer(plone.testing.Layer):
    """Settings for the Celery end to end tests."""

    def setUp(self):
        self['celery_config'] = {
            'ZOPE_CONF': '/path/to/my/test-zope.conf'}
        self['celery_parameters'] = (
            z3c.celery.conftest.celery_parameters())
        self['celery_worker_parameters'] = {'queues': ('celery',)}
        self['celery_includes'] = ['my.module.tasks']

    def tearDown(self):
        del self['celery_config']
        del self['celery_includes']
        del self['celery_parameters']
        del self['celery_worker_parameters']
```

Create a layer which brings the settings layer and the `EndToEndLayer` together, example:

```
CELERY_SETTINGS_LAYER = CelerySettingsLayer()
CONFIGURED_END_TO_END_LAYER = z3c.celery.layer.EndToEndLayer(
    bases=[CELERY_SETTINGS_LAYER], name="ConfiguredEndToEndLayer")
```

Create a layer which combines the configured EndToEndLayer with the ZCMLLayer of your application. (This should be the one created by `plone.testing.zca.ZCMLSandbox`.)

Example:

```
MY_PROJ_CELERY_END_TO_END_LAYER = plone.testing.Layer(
    bases=(CONFIGURED_END_TO_END_LAYER, ZCML_LAYER),
    name="MyProjectCeleryEndToEndLayer")
```

Note: The ZCMLLayer has to be the last one in the list of the bases because the EndToEndLayer forks the workers when it is set up. If the ZCML is already there running a task in the worker will break because as first step it has to load the *zope.conf*.

Caution: All tasks to be run in end to end tests have to shared tasks. This is necessary because the end to end tests have to use a different Celery instance than `z3c.celery.CELERY`. Example:

```
@celery.shared_task
def my_task():
    do_stuff()
```

1.1.7 Implementation notes

In case of a `ZODB.POSException.ConflictError` the worker process will wait and restart the operation again. This is done with active wait (`time.sleep()`) and not via the `self.retry()` mechanism of celery, as we were not able to figure out to get it flying.

1.2 Running the tests

To run the test suite of *z3c.celery* you need a `redis-server` listening on the default port. The tests use the redis database `/12`, which means that no other celery worker should be connected to that database and this database should not be used for other purposes.

To run the actual tests, simply run tox in the root directory of the package.

Together with the tests, this documentation will be build by tox.

1.3 API Reference

`z3c.celery.celery`
`z3c.celery.layer`

1.4 License

1.5 Change log for z3c.celery

1.5.1 1.3.0 (unreleased)

- Nothing changed yet.

1.5.2 1.2.3 (2018-06-28)

- Add logging for task retry.

1.5.3 1.2.2 (2018-03-23)

- Ensure ZODB connection can be closed, even if execution is aborted in the middle of a transaction

1.5.4 1.2.1 (2018-02-02)

- Add bw-compat for persisted tasks that still have a *_task_id_* parameter

1.5.5 1.2.0 (2018-01-23)

- Support task retry

1.5.6 1.1.0 (2017-10-11)

- Make worker process boot timeout configurable

1.5.7 1.0.2 (2017-10-05)

- Also apply "always endInteration" to HandleAfterAbort
- Also apply "retry on ConflictError" to HandleAfterAbort

1.5.8 1.0.1 (2017-10-04)

- Always call endInteraction, even on error during commit or abort, so we don't pollute the interaction state for the next task run

1.5.9 1.0 (2017-09-29)

- Introduce `Abort` control flow exception
- Allow overriding the principal id the job runs as
- Support reading configuration from a filesystem-based (non-importable) python file
- Don't use celery's deprecated default app mechanism
- Support running an actual "celery worker" with the single-process "solo" worker_pool

1.5.10 0.1 (2017-02-21)

- Initial release. Extract from zeit.cms.
- genindex